

Yet Another Clock Featuring the PIC16C924

*Author: Rodger Richey
Microchip Technology Inc.*

INTRODUCTION

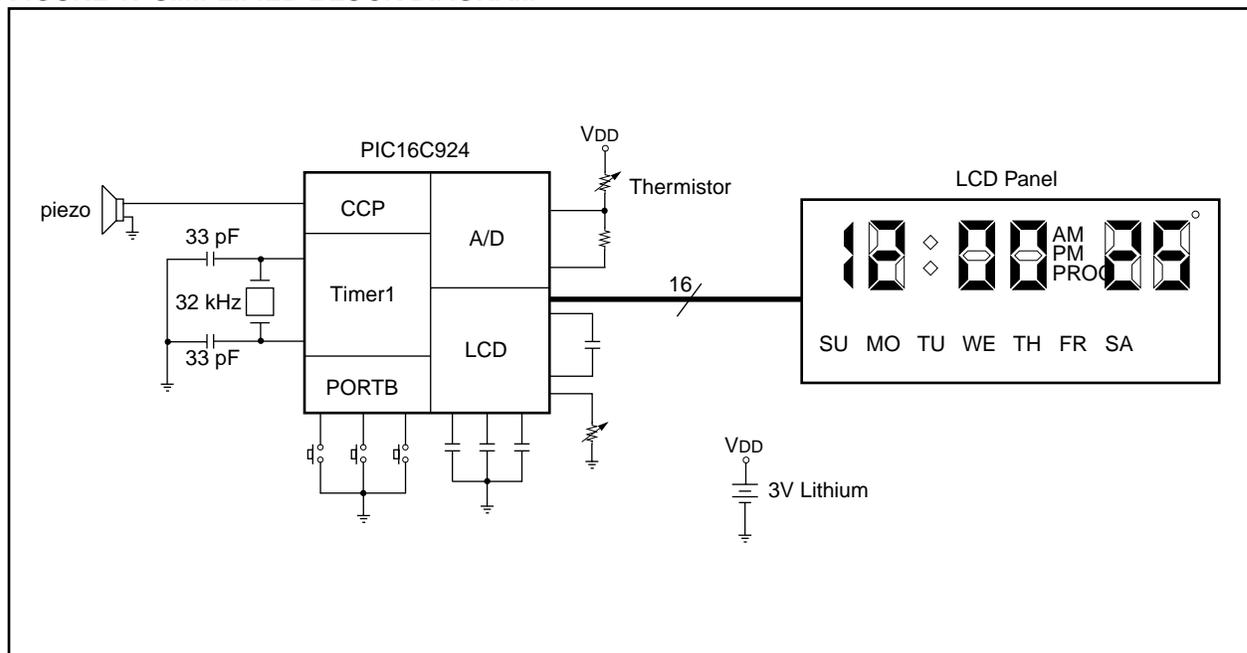
Once again, because of its universal familiarity and range of functionality, the clock is used to convey the use of the PIC16C92X microcontrollers. In this case we have added a twist to the clock with the addition of a thermometer. The LCD panel has a two digit temperature readout.

This application note will discuss the use of the following peripherals used to implement the clock: Timer1, PORTB, CCP, A/D converter, and the LCD Module. All source code and examples are written in C and compiled using Microchip's MPLAB-C compiler.

The features of the PIC16C924 are:

- 4K x 14 EPROM program memory
- 176 x 8 SRAM data memory
- DC - 8 MHz operating speed
- Timer0, Timer1, and Timer2
- One CCP pin
- SSP Module with SPI and I²C capability
- 8-bit, 5 channel A/D converter
- LCD Module
 - Multiple LCD timing sources
 - LCD can be driven while in SLEEP
 - Static, 1/2, 1/3 and 1/4 multiplex modes
 - Static and 1/3 bias capability
 - Up to 32 segments, up to 4 commons
 - 1 COM x 32 SEGs = 32 pixels
 - 2 COMs x 31 SEGs = 62 pixels
 - 3 COMs x 30 SEGs = 90 pixels
 - 4 COMs x 29 SEGs = 116 pixels
- Available in DIE form, 68-pin PLCC, and 64-pin TQFP packages

FIGURE 1: SIMPLIFIED BLOCK DIAGRAM



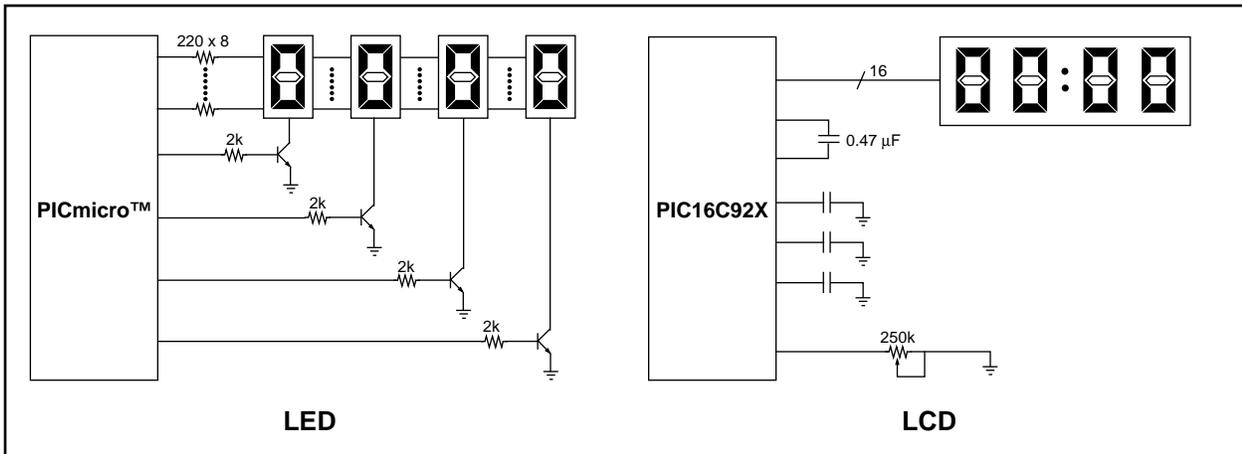
AN649

LCD panels offer many advantages over LED type displays such as; lower cost, lower power consumption, and better display quality. Figure 2 shows typical examples of an LED and an LCD application. Table 1 further describes each application according to components, cost, power consumption, etc.

TABLE 1: LED vs. LCD

	LED	LCD
Cost (1000 units)	\$7.05	\$5.42
# Components	20	6
Power Consumption	~ 10 mA	~ 50 μ A
Hardware	Timer, 12 I/O pins	LCD Module
Signal Generation	Firmware	LCD Module

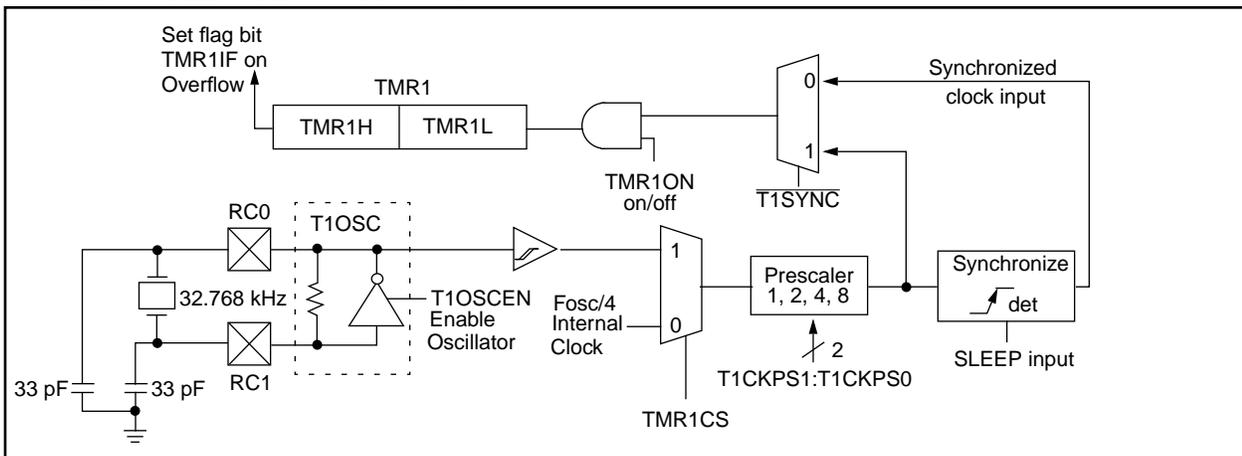
FIGURE 2: EXAMPLE LED AND LCD APPLICATIONS



TIMER1

Currently, the Timer1 module exists in all the PIC16CXXX devices with 28 or more pins. This module can be used to easily implement a real-time clock. Instead of an external real-time clock device, an inexpensive 32.768 kHz watch crystal and two 33 pF capacitors are used to complete the circuit. Figure 3 shows the block diagram for Timer1.

FIGURE 3: TIMER1 BLOCK DIAGRAM



In this application, Timer1 is clocked by an external crystal connected across RC0 and RC1. The following is some sample code to initialize Timer1.

```
TMR1H = 0x80;
TMR1L = 0x00;
T1CON = 0b00001111;
PIR1.TMR1IF = 0;
PIE1.TMR1IE = 1;
```

The first step in initializing Timer1 is to preset TMR1H:TMR1L to 0x8000. Since Timer1 is a 16-bit timer, the 32.768 kHz crystal will cause Timer1 to overflow every two seconds or 65536 counts. For this reason, Timer1 is initialized to 0x8000 so that every overflow relates to one second. The second step is to configure Timer1. For this mode, the Timer1 oscillator must be enabled via T1CON<T1OSCEN>. This bit enables the internal oscillator, which is functionally equivalent to the LP oscillator of the microcontroller. The clock source for Timer1 is selected to be the external input using the T1CON<TMR1CS> bit. The prescaler is set to 1:1 using the T1CKPS1:T1CKPS0 bits of the T1CON register. Since the timer is to operate during SLEEP, it is not synchronized to the internal Fosc clock. Synchronization is controlled by the T1SYNC bit of the T1CON register. Finally, the clock source is enabled to clock TMR1H:TMR1L using the TMR1ON bit of T1CON.

In this mode of operation, Timer1 can operate during SLEEP while consuming a minimal amount of current ($\approx 20 \mu\text{A}$). The last two lines of code clear the Timer1 Overflow flag and enable the Timer1 Overflow interrupt. This interrupt will wake up the processor from SLEEP at a predetermined rate for updating the clock, in this case once each second. The following is a sample interrupt service routine for Timer1 Overflow:

```
if(PIR1.TMR1IF)
{
    Seconds++;          //Increment seconds
    if(Seconds > 59)    //60 seconds?
    {
        Seconds = 0;
        Minutes++;
    }
    if(Minutes > 59)    //60 minutes?
    {
        Minutes = 0;
        Hours++;
    }
    if(Hours > 12)      //Do not use 24hr
        Hours = 1;     //military time

    TMR1H |= 0x80;     //Reset timer1
    PIR1.TMR1IF = 0;   //Clear flag
}
```

First, register PIR1 is checked to verify that the Timer1 Overflow interrupt occurred. Next the variable Seconds is incremented every time a Timer1 overflow occurs. Once Seconds reaches 60, the Minutes variable is incremented and Seconds is cleared thus implementing 60 seconds per minute. If Minutes reaches 60, then the Hours variable is incremented

and Minutes is cleared which implements 60 minutes per hour. For civilian time, if Hours is greater than 12, Hours is reset to 1. Military time uses 24 hours a day, so that when Hours reaches 24 it is reset to 1.

Since it takes a finite amount of time to enter the interrupt service routine and execute it, the program adds 0x80 to TMR1H so that the next Timer1 Overflow interrupt occurs exactly in one second. If the program simply cleared TMR1L and set TMR1H to 0x80, the real-time clock would now be off by the finite amount of time previously described. Finally, the Timer1 Overflow interrupt flag is reset.

PORTB

The wake-up on change feature of PORTB was specifically designed to interface keys or a keypad directly to the microcontroller. Internal weak pull-up resistors are provided to reduce external parts count, while adding only a small amount of current. This feature allows the microcontroller to remain in the low-power SLEEP mode until a key is pressed. The device will wake-up from SLEEP when the key is pressed. The interrupt service routine will process the key input. The following example routine shows how to initialize the PORTB wake-up on change interrupt.

```
OPTION.RBPU = 0;
Temp = PORTB;
INTCON.RBIF = 0;
INTCON.RBIE = 1;
```

Whenever the state of the RB7:RB4 pins change, a mismatch condition occurs inside the microcontroller. The only way to clear the mismatch condition is to read PORTB, which also allows the RBIF interrupt flag to be cleared. In the previous code, the first line enables the internal weak pull-up resistors. The second line of the code resets the mismatch condition. Then the following lines clear the interrupt flag and enable the interrupt. A sample interrupt service routine is given below.

```
if(INTCON.RBIF)
{
    Temp = PORTB;
    Delay_Ms_4MHz(16); //key debounce
    if(Temp!=0xf0 && Temp==PORTB)
    {
        StartBEEP(); //key press starts
        beep

        if(!Temp.SET)      //SET key
            Flags.SET = 1;
        if(!Temp.UP)       //UP key
            Flags.UP = 1;
        if(!Temp.SOUND)    //SOUND key
        {
            if(Flags.SOUND_STATE)
                Flags.SOUND_STATE = 0;
            else
                Flags.SOUND_STATE = 1;
        }
    }
    INTCON.RBIF = 0;      //clear flag
}
```

The first line of the above example is used to verify that a change on PORTB interrupt has occurred. The next line is a 5 ms delay followed by a read of PORTB. This resets the mismatch condition. The following 20 ms delay is used in conjunction with the previous 5 ms delay for switch debouncing. The value of `Temp` and PORTB are compared and if equal a key press has been detected. The if statement also checks to see if the mismatch condition is from a key press or when the key is released. The key inputs are processed only when a key is pressed. When a key is pressed, it grounds the respective input pin. The nested if statements check each of the individual keys to see which have been pressed. If the SOUND key has been pressed, it merely toggles whether the hourly beep is enabled. Finally, the RBIF interrupt flag is cleared.

This application takes advantage of the wake-up on change and internal pull-up resistors to implement the SET, ▲, and SOUND keys. The SET key puts the clock in program mode and the PROG icon appears on the LCD. Program mode allows the user to set the time, AM or PM, and day of the week (see LCD Module for details on the LCD). Once in program mode, the hours digits flash. Pressing the ▲ key increments `Hours` from 12 to 11 AM and then from 12 to 11 PM. At any point the SET key can be pressed to advance to the minutes digits. The minutes digits are incremented using the ▲ key. `Minutes` can be incremented from 0 to 59. Pressing the SET key again flashes the day of the week. Using the ▲ key increments the day of the week from SU to SA. Finally, pressing the SET key takes the user out of program mode. If no key presses are detected for five seconds the program exits program mode.

CCP

The CCP module is used in Pulse Width Modulation (PWM) mode. The PWM signal is used to drive a piezo alarm circuit. The operation of PWM mode will not be discussed since it has been explained in great depth in application notes AN531, AN538, AN539, AN564, and AN594.

An example of configuring the CCP as a PWM output is shown below.

```
CCP1CON = 0x0f;    //Set CCP to PWM
PR2 = 122;        //period of 2048 Hz
CCPR1L = 5;      //Duty Cycle very low
TRISC.RC2 = 0;   //PWM pin output
T2CON = 0b01111101; //Enable timer2
PIR1.TMR2IF = 0; //Clear flag
PIE1.TMR2IE = 1; //Enable interrupt
```

The CCP1CON register is set to 0x0F which puts the CCP into PWM mode. The period or frequency of the PWM output is set using the PR2 register. The following formula is used to calculate the value in PR2.

$$\text{PWM period} = [(\text{PR2}) + 1] \cdot 4 \cdot \text{Tosc} \cdot (\text{TMR2 prescale value})]$$

Another step in configuring the CCP is to set the value of the T2CON register. In the case above, a value of 0x7D configures the Timer2 output postscaler to 1:16, enables Timer2, and sets the Timer2 clock prescaler to 4. Using PR2 = 122, TOSC = 250 ns, and TMR2 prescale = 4, the resultant PWM period is approximately 500 μs or 2 kHz. This is the resonant frequency of the piezo alarm. The duty cycle, or more specifically the time the PWM output stays high, of the PWM output is set by the value of CCPR1L and bits CCP1CON<5:4>. The following formula uses the 10-bit value of CCPR1L:CCP1CON<5:4> to calculate the duty cycle.

$$\text{PWM duty cycle} = [(\text{CCPR1L:CCP1CON<5:4>}) \cdot \text{Tosc} \cdot (\text{TMR2 prescale value})]$$

Using 0x014 as the value of CCPR1L:CCP1CON<5:4>, TOSC = 250 ns, and a TMR2 prescale = 4, the PWM duty cycle is calculated to be 20 μs or 4% duty cycle. The TRISC register must also be configured such that the PWM pin is setup as an output.

The Timer2 Overflow interrupt is used to turn off the PWM output. This produces a “beep” when the keys are pressed or an hourly alarm. The Timer2 postscaler waits for 16 PWM periods before the interrupt occurs. The following interrupt service routine is an example of how to generate a “beep”.

```
if(PIR1.TMR2IF)
{
    Count--;
    if(!Count)
    {
        CCP1CON = 0;    //Disable CCP, and
        T2CON = 0;      //Timer2 and TMR2
        PIE1.TMR2IE = 0; //interrupts
        CCPR1L = 0;
    }
    PIR1.TMR2IF = 0;   //Clear flag
}
```

The first line of code detects if a Timer2 Overflow interrupt has occurred. The variable `Count` is used to vary the length of the beep. `Count` is set previous to enabling the PWM. If `Count` reaches zero the following occurs:

- The PWM is disabled by clearing the CCP1CON register
- Timer2 is disabled by clearing the T2CON register
- The Timer2 Overflow interrupt is disabled by clearing the TMR2IE bit of the register PIE1
- The duty cycle register is cleared

Before exiting the service routine, the TMR2IF interrupt flag is cleared.

A/D CONVERTER

Since the PIC16C924 has a five channel, 8-bit A/D converter and the LCD has a ° (degree) symbol and two digits, the application circuit has a thermistor for measuring temperature. Thermistors typically take hundreds of milliseconds to stabilize at a particular temperature and therefore the A/D converter is ideal for temperature measurements. Another feature of the A/D converter is the on-chip RC oscillator that can be used as the conversion clock. This feature allows the A/D to operate in SLEEP. The following code segment is used to initialize the A/D converter.

```
ADCON0 = 0b11000001; //Enable A/D
ADCON1 = 0b00000100; //Configure D/A I/O
PIR1.ADIF = 0; //Clear flag
PIE1.ADIE = 1; //Enable interrupt
```

The first line enables the internal RC for conversion clock, channel 0, and enables the A/D converter. The second line makes PORTA<1:0> analog inputs and PORTA<5,3:2> as digital I/O. The following lines clear the A/D conversion complete interrupt flag and enable the interrupt. The application uses the Timer1 Overflow interrupt to start a conversion every second using the GO bit of the ADCON0 register. The A/D conversion complete interrupt then processes the result of the conversion. The following is an example of the service routine.

```
if(PIR1.ADIF)
{
    TempC = ThermTable[ADRES];
    PIR1.ADIF = 0;
}
```

The first line checks for the ADIF interrupt flag. A lookup table, `ThermTable`, is used to convert the A/D result into a temperature reading. This table was created using calibration data from the thermistor. Finally, the ADIF interrupt flag is cleared.

The following code was added to the Timer1 interrupt service routine to start the A/D conversion.

```
TRISA.THERM_GND = 0;
DELAY_10_us_4MHz(4);
ADCON0.GO = 1;
NOP();
NOP();
TRISA.THERM_GND = 1;
```

The first line of code grounds the I/O pin connected to resistor R1. The 40 µs delay is provided so the A/D converter can sample the input signal. The A/D converter is then instructed to convert by setting the GO bit. A two cycle delay is added so that the sampling capacitor is disconnected from the input. Finally the I/O pin connected to R1 is made an input. This scheme only powers the thermistor when sampling, to reduce power consumption.

AN649

LCD MODULE

The LCD Module has a wealth of features typically found on more expensive dedicated LCD driver devices. The following is a detailed list of features:

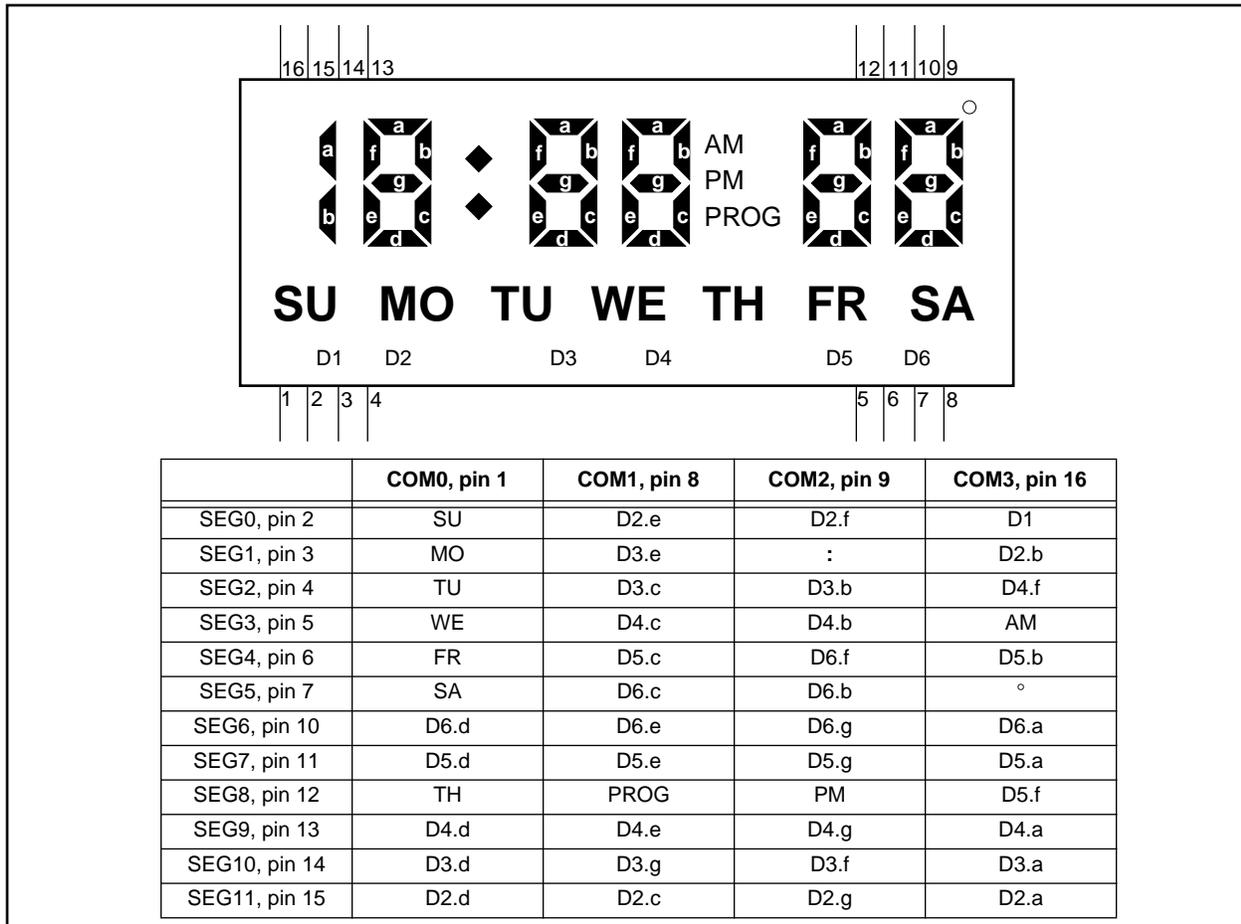
- LCD Timing Sources
 - Fosc/256 (internal system clock)
 - Timer1 oscillator
 - Internal RC oscillator
- LCD Voltage Generation
 - Internal charge pump
 - External resistor ladder
- Bias
 - Static
 - 1/3
- Multiplex
 - Static
 - 1/2
 - 1/3
 - 1/4
- Operation during SLEEP
 - Only with internal RC or Timer1 clock sources

- Capable of up to 4 commons and 32 segments
 - 1 COM x 32 SEGs, total of 32 pixels
 - 2 COMs x 31 SEGs, total of 62 pixels
 - 3 COMs x 30 SEGs, total of 90 pixels
 - 4 COMs x 29 SEGs, total of 116 pixels
- 16 x 8 LCD data registers

The LCD Module is ideal for systems that use one controller board that has several applications with different displays. Microchip's OTP technology means that the controller board can be assembled with blank micro-controllers, and at final test the device can be programmed depending on the display type. This helps to reduce overhead and creates one board to track in inventory.

The particular LCD panel that was selected for this application is shown in Figure 2. It has four common electrodes and 12 segment electrodes. The panel provides 3 1/2 digits, colon, and AM/PM icons for time display. The panel also has day of the week icons (SU,MO,TU,WE,TH,FR,SA). This was the basis of another clock design using the PIC16C924.

FIGURE 4: LCD PINOUT



The LCD Module is initialized by the following code:

```

STATUS.RP1 = 1;           //Change to Bank 2
LCDPS = 6;               //Frame freq to 37Hz
LCDSE = 0xff;           //All LCD I/O as LCD
LCDCON = 0b00010111;    //1/4 MUX charge pump
LCDD00 = 0;             //Clear all LCD
LCDD01 = 0;             //data RAM
LCDD02 = 0;
LCDD03 = 0;
LCDD04 = 0;
LCDD05 = 0;
LCDD06 = 0;
LCDD07 = 0;
LCDD08 = 0;
LCDD09 = 0;
LCDD10 = 0;
LCDD11 = 0;
LCDD12 = 0;
LCDD13 = 0;
LCDD14 = 0;
LCDD15 = 0;
LCDCON.LCDEN = 1;       //Enable LCD module
STATUS.RP1 = 0;         //Back to Bank 0
PIR1.LCDIF = 0;        //Clear flag
PIE1.LCDIE = 1;        //Enable interrupt

```

Note: At the time when this application note was generated, four banks of data memory were not supported by the MPC or MPLAB-C compilers.

The PIC16C924 starts a new page in Microchip history with four banks of data memory for the PIC16CXXX mid-range products. The first line of code switches to the second set of banks. The second line of code sets a frame frequency of approximately 37 Hz using the Timer1 oscillator. This frequency can be calculated by using the following formula:

$$\text{Clock source} / (128 \cdot (\text{LP3:LP0} + 1))$$

Using 32.768 kHz as the clock source and LP3:LP0 = 6 the resultant frame frequency is 36.57 Hz. Setting LCDSE to 0xFF configures ports D,E,F, and G as LCD drivers. Setting LCDCON to 0x17 configures the LCD Module for 1/4 MUX, 1/3 Bias, Timer1 clock source, charge pump enabled, and the LCD module will continue to drive during SLEEP. The LCD data registers LCDD00 - LCDD15 are all cleared, which turns off all pixels on the LCD panel. Any unused bits in the LCD data registers can be used as general purpose RAM. The PIC16C924 Clock can use the upper 4-bits of registers LCDD01, LCDD05, LCDD09, LCDD13 and all of LCDD02, LCDD03, LCDD06, LCDD07, LCDD10, LCDD11, LCDD14, LCDD15. This is a result of not using segments 13 through 28. The LCD Module is then enabled by setting the LCDEN bit in the LCDCON register. Finally the LCD interrupt flag is cleared and the interrupt is enabled.

The following is an example of the LCD interrupt service routine.

```

if(PIR1.LCDIF)
{
    Flags.FRAME = 1;
    PIR1.LCDIF = 0;
}

```

In this application, the LCD interrupt is used to signal the main routine that the LCD data registers can be updated without causing any flicker on the panel. Finally, as in any interrupt service routine, the interrupt flag is cleared.

CONCLUSION

The PIC16C924 is ideally suited towards battery applications such as thermostats or meters. It can extend battery life while maintaining a rich feature set such as A/D converter, SPI/I²C module, and CCP module. All of the peripherals that have been previously described can operate during SLEEP, thus lowering the average current consumption. If the 8-bit A/D converter is not sufficient or not used in your design, Microchip also offers the PIC16C923 which has all the features of the PIC16C924 without the A/D converter. This device can use an external higher resolution A/D converter that can be interfaced to the SSP module, specifically the SPI port.

APPENDIX B: PIC16C924 CLOCK FIRMWARE LISTING

```

/*****
*   Filename: CLK.C
*****
*   Author:      Rodger Richey
*   Company:    Microchip Technology Incorporated
*   Revision:   A3
*   Date:       12-17-96
*   Compiled using MPLAB-C Version 00.00.14
*****
*   Include files:
*   16C924.h    Rev 1.00
*   MUSIC.C     Rev A2
*   LCD.C       Rev A1
*   TIME.C      Rev A1
*****
*   Peripheral Modules Used:
*   Timer0      : Used by the music generation program for timing notes
*                 and durations
*   Timer1      : Used as a real time clock using an external 32.768KHz
*                 crystal and (2) 33pF capacitors
*   Timer2      : Used in conjunction with the PWM
*   CCP         : Used in PWM mode, for driving the piezo alarm
*   PORTB       : Used to decode key presses
*   A/D         : Used to measure temperature via a thermistor on RA0
*   LCD         : Used to display time, temperature, day of week
*****
*   External Clock Frequency      : 4MHz
*   Timer1 OSC Frequency          : 32.768KHz
*   Configuration Bit Settings   : XT Oscillator
*                                 : Watchdog Timer OFF
*                                 : Code Protect OFF
*                                 : Power-Up Timer ON
*   Program Memory Usage         : 1095 words
*   Data Memory Usage            : 19 bytes
*****
*   Revision History
*   A1 - First Release
*   A2 - Added code to clear CCPRL1 after PWM has finished
*       - Set No sleep flag in StartMusic
*   A3 - Changed __INT interrupt service routine.
*       Added INTCON.T0IE to check for Timer0 interrupt
*****
*   Note: Make sure that the temporary variables in the 16c924.h file
*         have been changed to locations 0x7a to 0x7f
*****/
#include <16c924.h>
#include <delay14.h>

// PORTA pin defines
0002 #define THERM_GND 2

// PORTB key defines
0004 #define EXTRA 4
0005 #define SOUND 5
0006 #define UP 6
0007 #define SET 7

// Variable declarations
0026 bits Flags; // Contains flag bits for various events
0027 bits Temp; // Temporary storage
0028 bits Count; // Number of Timer2 Interrupts to count
0029 bits Ticks; // Counts seconds will in program mode
0078 bits Mode @ 0x78; // Contains flag bits to turn on LCD pixels
002A unsigned char FrameCnt; // Count LCD frames

```

0015	3400	RETLW	00h
0016	3400	RETLW	00h
0017	3400	RETLW	00h
0018	3400	RETLW	00h
0019	3400	RETLW	00h
001A	3400	RETLW	00h
001B	3400	RETLW	00h
001C	3400	RETLW	00h
001D	3400	RETLW	00h
001E	3400	RETLW	00h
001F	3400	RETLW	00h
0020	3400	RETLW	00h
0021	3400	RETLW	00h
0022	3400	RETLW	00h
0023	3400	RETLW	00h
0024	3400	RETLW	00h
0025	3400	RETLW	00h
0026	3400	RETLW	00h
0027	3400	RETLW	00h
0028	3400	RETLW	00h
0029	3400	RETLW	00h
002A	3400	RETLW	00h
002B	3400	RETLW	00h
002C	3400	RETLW	00h
002D	3400	RETLW	00h
002E	3400	RETLW	00h
002F	3400	RETLW	00h
0030	3400	RETLW	00h
0031	3400	RETLW	00h
0032	3401	RETLW	01h
0033	3402	RETLW	02h
0034	3403	RETLW	03h
0035	3404	RETLW	04h
0036	3405	RETLW	05h
0037	3406	RETLW	06h
0038	3406	RETLW	06h
0039	3407	RETLW	07h
003A	3408	RETLW	08h
003B	3409	RETLW	09h
003C	3409	RETLW	09h
003D	3410	RETLW	10h
003E	3411	RETLW	11h
003F	3411	RETLW	11h
0040	3412	RETLW	12h
0041	3412	RETLW	12h
0042	3413	RETLW	13h
0043	3413	RETLW	13h
0044	3414	RETLW	14h
0045	3414	RETLW	14h
0046	3415	RETLW	15h
0047	3415	RETLW	15h
0048	3416	RETLW	16h
0049	3416	RETLW	16h
004A	3417	RETLW	17h
004B	3417	RETLW	17h
004C	3418	RETLW	18h
004D	3418	RETLW	18h
004E	3419	RETLW	19h
004F	3419	RETLW	19h
0050	3420	RETLW	20h
0051	3420	RETLW	20h
0052	3421	RETLW	21h
0053	3421	RETLW	21h
0054	3422	RETLW	22h
0055	3422	RETLW	22h
0056	3423	RETLW	23h

AN649

0057	3423	RETLW	23h
0058	3424	RETLW	24h
0059	3424	RETLW	24h
005A	3425	RETLW	25h
005B	3425	RETLW	25h
005C	3425	RETLW	25h
005D	3426	RETLW	26h
005E	3426	RETLW	26h
005F	3427	RETLW	27h
0060	3427	RETLW	27h
0061	3428	RETLW	28h
0062	3428	RETLW	28h
0063	3428	RETLW	28h
0064	3429	RETLW	29h
0065	3429	RETLW	29h
0066	3430	RETLW	30h
0067	3430	RETLW	30h
0068	3430	RETLW	30h
0069	3431	RETLW	31h
006A	3431	RETLW	31h
006B	3432	RETLW	32h
006C	3432	RETLW	32h
006D	3432	RETLW	32h
006E	3433	RETLW	33h
006F	3433	RETLW	33h
0070	3433	RETLW	33h
0071	3434	RETLW	34h
0072	3434	RETLW	34h
0073	3435	RETLW	35h
0074	3435	RETLW	35h
0075	3435	RETLW	35h
0076	3436	RETLW	36h
0077	3436	RETLW	36h
0078	3437	RETLW	37h
0079	3437	RETLW	37h
007A	3438	RETLW	38h
007B	3438	RETLW	38h
007C	3438	RETLW	38h
007D	3439	RETLW	39h
007E	3439	RETLW	39h
007F	3440	RETLW	40h
0080	3440	RETLW	40h
0081	3440	RETLW	40h
0082	3441	RETLW	41h
0083	3441	RETLW	41h
0084	3442	RETLW	42h
0085	3442	RETLW	42h
0086	3443	RETLW	43h
0087	3443	RETLW	43h
0088	3443	RETLW	43h
0089	3444	RETLW	44h
008A	3444	RETLW	44h
008B	3445	RETLW	45h
008C	3445	RETLW	45h
008D	3445	RETLW	45h
008E	3446	RETLW	46h
008F	3446	RETLW	46h
0090	3447	RETLW	47h
0091	3447	RETLW	47h
0092	3448	RETLW	48h
0093	3448	RETLW	48h
0094	3448	RETLW	48h
0095	3449	RETLW	49h
0096	3449	RETLW	49h
0097	3450	RETLW	50h
0098	3450	RETLW	50h

0099	3450	RETLW	50h
009A	3451	RETLW	51h
009B	3451	RETLW	51h
009C	3452	RETLW	52h
009D	3452	RETLW	52h
009E	3453	RETLW	53h
009F	3453	RETLW	53h
00A0	3454	RETLW	54h
00A1	3454	RETLW	54h
00A2	3455	RETLW	55h
00A3	3455	RETLW	55h
00A4	3456	RETLW	56h
00A5	3456	RETLW	56h
00A6	3457	RETLW	57h
00A7	3457	RETLW	57h
00A8	3458	RETLW	58h
00A9	3458	RETLW	58h
00AA	3459	RETLW	59h
00AB	3459	RETLW	59h
00AC	3460	RETLW	60h
00AD	3460	RETLW	60h
00AE	3461	RETLW	61h
00AF	3461	RETLW	61h
00B0	3462	RETLW	62h
00B1	3462	RETLW	62h
00B2	3463	RETLW	63h
00B3	3463	RETLW	63h
00B4	3464	RETLW	64h
00B5	3464	RETLW	64h
00B6	3465	RETLW	65h
00B7	3465	RETLW	65h
00B8	3466	RETLW	66h
00B9	3466	RETLW	66h
00BA	3467	RETLW	67h
00BB	3467	RETLW	67h
00BC	3468	RETLW	68h
00BD	3468	RETLW	68h
00BE	3469	RETLW	69h
00BF	3469	RETLW	69h
00C0	3470	RETLW	70h
00C1	3471	RETLW	71h
00C2	3471	RETLW	71h
00C3	3472	RETLW	72h
00C4	3473	RETLW	73h
00C5	3473	RETLW	73h
00C6	3474	RETLW	74h
00C7	3474	RETLW	74h
00C8	3475	RETLW	75h
00C9	3476	RETLW	76h
00CA	3476	RETLW	76h
00CB	3477	RETLW	77h
00CC	3478	RETLW	78h
00CD	3478	RETLW	78h
00CE	3479	RETLW	79h
00CF	3479	RETLW	79h
00D0	3480	RETLW	80h
00D1	3481	RETLW	81h
00D2	3481	RETLW	81h
00D3	3482	RETLW	82h
00D4	3483	RETLW	83h
00D5	3484	RETLW	84h
00D6	3484	RETLW	84h
00D7	3485	RETLW	85h
00D8	3486	RETLW	86h
00D9	3487	RETLW	87h
00DA	3488	RETLW	88h

AN649

```
00DB 3489   RETLW  89h
00DC 3490   RETLW  90h
00DD 3491   RETLW  91h
00DE 3492   RETLW  92h
00DF 3493   RETLW  93h
00E0 3494   RETLW  94h
00E1 3495   RETLW  95h
00E2 3496   RETLW  96h
00E3 3497   RETLW  97h
00E4 3498   RETLW  98h
00E5 3499   RETLW  99h
00E6 3499   RETLW  99h
00E7 3499   RETLW  99h
00E8 3499   RETLW  99h
00E9 3499   RETLW  99h
00EA 3499   RETLW  99h
00EB 3499   RETLW  99h
00EC 3499   RETLW  99h
00ED 3499   RETLW  99h
00EE 3499   RETLW  99h
00EF 3499   RETLW  99h
00F0 3499   RETLW  99h
00F1 3499   RETLW  99h
00F2 3499   RETLW  99h
00F3 3499   RETLW  99h
00F4 3499   RETLW  99h
00F5 3499   RETLW  99h
00F6 3499   RETLW  99h
00F7 3499   RETLW  99h
00F8 3499   RETLW  99h
00F9 3499   RETLW  99h
00FA 3499   RETLW  99h
00FB 3499   RETLW  99h
00FC 3499   RETLW  99h
00FD 3499   RETLW  99h
00FE 3499   RETLW  99h
00FF 3499   RETLW  99h
```

```
/*
 * StartBEEP
 * Function: This routine configures the necessary hardware to emit a
 *          beep from the piezo
 */
void StartBEEP(void)
{
0100 1283   BCF    03,5   Flags.SLEEP_STATE = 1; // Don't let the 924 go to sleep
0101 16A6   BSF    26,5
0102 3004   MOVLW  04h   Count = BEEP_COUNT; // Set Count for length of beep
0103 00A8   MOVWF  28
0104 300F   MOVLW  0Fh   CCP1CON = 0x0f; // Set the CCP module to PWM
0105 0097   MOVWF  17
0106 307A   MOVLW  7Ah   PR2 = 122; // Set the period to 2048Hz
0107 1683   BSF    03,5
0108 0092   MOVWF  12
0109 3003   MOVLW  03h   CCPR1L = 3; // Set the duty cycle very low
010A 1283   BCF    03,5
010B 0095   MOVWF  15
010C 307D   MOVLW  7Dh   T2CON = 0b01111101; // Enable Timer2
010D 0092   MOVWF  12
010E 108C   BCF    0C,1   PIR1.TMR2IF = 0; // Clear the Timer2 Interrupt flag
010F 1683   BSF    03,5   PIE1.TMR2IE = 1; // Enable the Timer2 Interrupt
0110 148C   BSF    0C,1
0111 0008   RETURN

}

// Include source files
```

```

#include "lcd.c"           // Contains programs that control the LCD
/*****
*   Filename: LCD.C
*****
*   Author:      Rodger Richey
*   Company:    Microchip Technology Incorporated
*   Revision:   A0
*   Date:       6-14-96
*   Compiled using MPLAB-C Version 00.00.14
*****
*   This file contains routines to output time, day of week, AM/PM, and
*   temperature.  It also contains routines to blink the different groupings
*   of numbers, i.e. hours,seconds,day of the week.  Finally, the last
*   routine displays the state of the hourly beep.
*****/

// Bit defines for the Mode variable, tells the UpdateLCD routine which
// groups of numbers to display
0000 #define HOURS 0
0001 #define MINUTES 1
0002 #define DAYOFWEEK 2
0005 #define PROG 5           // PROG icon
0006 #define COLON 6         // colon, :, for the hours/seconds time display
0007 #define DEGREES 7       // degrees symbol for temperature

/*****
*   UpdateLCD
*   Function: This function updates the LCD display based on the Mode
*             variable.
*****/
void UpdateLCD(void)
{
    // Array of 7-segment numbers           // gfedcba
    const unsigned char SevenSegTable[16]={ 0b00111111, // Zero
                                             0b00000110, // One
                                             0b01011011, // Two
                                             0b01001111, // Three
                                             0b01100110, // Four
                                             0b01101101, // Five
                                             0b01111101, // Six
                                             0b00000111, // Seven
                                             0b01111111, // Eight
                                             0b01101111, // Nine
                                             0b01110111, // Ten
                                             0b01111100, // Eleven
                                             0b01011000, // Twelve
                                             0b01011110, // Thirteen
                                             0b01111001, // Fourteen
                                             0b01110001}; // Fifteen

0112 2924    GOTO    0124h
0113 0782    ADDWF   02
0114 343F    RETLW   3Fh
0115 3406    RETLW   06h
0116 345B    RETLW   5Bh
0117 344F    RETLW   4Fh
0118 3466    RETLW   66h
0119 346D    RETLW   6Dh
011A 347D    RETLW   7Dh
011B 3407    RETLW   07h
011C 347F    RETLW   7Fh
011D 346F    RETLW   6Fh
011E 3477    RETLW   77h
011F 347C    RETLW   7Ch
0120 3458    RETLW   58h
0121 345E    RETLW   5Eh
0122 3479    RETLW   79h

```

AN649

```
0123 3471    RETLW    71h

// Temporary variables in common RAM locations
0076          bits segment @ 0x76;
0077          unsigned char index @ 0x77;

// Change to Bank 2, and clear all LCD data RAM
0124 1703    BSF      03,6    STATUS.RP1 = 1;
0125 1283    BCF      03,5    LCDD00 = 0;
0126 0190    CLRF     10
0127 0191    CLRF     11      LCDD01 = 0;
0128 0194    CLRF     14      LCDD04 = 0;
0129 0195    CLRF     15      LCDD05 = 0;
012A 0198    CLRF     18      LCDD08 = 0;
012B 0199    CLRF     19      LCDD09 = 0;
012C 019C    CLRF     1C      LCDD12 = 0;
012D 019D    CLRF     1D      LCDD13 = 0;

// Update day of the week if enabled
012E 1D78    BTFSS    78,2    if(Mode.DAYOFWEEK)
012F 2965    GOTO     0165h
0130

{
// Update Day of the Week
0130 08F4    MOVF      74          if(DayOfWeek == 0)          // Sunday
0131 1D03    BTFSS    03,2
0132 2936    GOTO     0136h          LCDD00.0 = 1;
0133 1283    BCF      03,5
0134 1410    BSF      10,0
0135 2965    GOTO     0165h          else if(DayOfWeek == 1)    // Monday
0136 3001    MOVLW    01h          LCDD00.1 = 1;
0137 1283    BCF      03,5
0138 0274    SUBWF    74,W
0139 1D03    BTFSS    03,2
013A 293E    GOTO     013Eh
013B 1283    BCF      03,5
013C 1490    BSF      10,1
013D 2965    GOTO     0165h          else if(DayOfWeek == 2)    // Tuesday
013E 3002    MOVLW    02h          LCDD00.2 = 1;
013F 1283    BCF      03,5
0140 0274    SUBWF    74,W
0141 1D03    BTFSS    03,2
0142 2946    GOTO     0146h
0143 1283    BCF      03,5
0144 1510    BSF      10,2
0145 2965    GOTO     0165h          else if(DayOfWeek == 3)    // Wednesday
0146 3003    MOVLW    03h          LCDD00.3 = 1;
0147 1283    BCF      03,5
0148 0274    SUBWF    74,W
0149 1D03    BTFSS    03,2
014A 294E    GOTO     014Eh
014B 1283    BCF      03,5
014C 1590    BSF      10,3
014D 2965    GOTO     0165h          else if(DayOfWeek == 4)    // Thursday
014E 3004    MOVLW    04h          LCDD01.0 = 1;
014F 1283    BCF      03,5
0150 0274    SUBWF    74,W
0151 1D03    BTFSS    03,2
0152 2956    GOTO     0156h
0153 1283    BCF      03,5
0154 1411    BSF      11,0
0155 2965    GOTO     0165h          else if(DayOfWeek == 5)    // Friday
0156 3005    MOVLW    05h          LCDD00.4 = 1;
0157 1283    BCF      03,5
0158 0274    SUBWF    74,W
0159 1D03    BTFSS    03,2
015A 295E    GOTO     015Eh
```

```

015B 1283   BCF    03,5
015C 1610   BSF    10,4
015D 2965   GOTO   0165h           else if(DayOfWeek == 6)           // Saturday
015E 3006   MOVLW 06h             LCDD00.5 = 1;
015F 1283   BCF    03,5
0160 0274   SUBWF  74,W
0161 1D03   BTFSS  03,2
0162 2965   GOTO   0165h
0163 1283   BCF    03,5
0164 1690   BSF    10,5
}

// Update Time if enabled
if(Mode.HOURS)

0165 1283   BCF    03,5
0166 1C78   BTFSS  78,0
0167 2988   GOTO   0188h
0168
{
// Update AM/PM icons
if(LStatus.AMPM)

0168 1FF3   BTFSS  73,7
0169 296C   GOTO   016Ch
016A 1419   BSF    19,0
016B 296D   GOTO   016Dh           else
016C 159C   BSF    1C,3             LCDD12.3 = 1;

// Digit 1
if(Hours&0x10)
016D 1A72   BTFSC  72,4
016E 141C   BSF    1C,0             LCDD12.0 = 1;

// Digit 2
index = Hours & 0x0f;

segment = SevenSegTable[index];

016F 300F   MOVLW 0Fh
0170 0572   ANDWF  72,W
0171 00F7   MOVWF  77
0172 00FB   MOVWF  7B
0173 3001   MOVLW 01h
0174 008A   MOVWF  0A
0175 08FB   MOVF   7B
0176 0877   MOVF   77,W
0177 2113   CALL  0113h
0178 1283   BCF    03,5
0179 00F6   MOVWF  76
017A 1876   BTFSC  76,0           if(segment.0)           // D2.a
017B 159D   BSF    1D,3             LCDD13.3 = 1;
017C 18F6   BTFSC  76,1           if(segment.1)           // D2.b
017D 149C   BSF    1C,1             LCDD12.1 = 1;
017E 1976   BTFSC  76,2           if(segment.2)           // D2.c
017F 1595   BSF    15,3             LCDD05.3 = 1;
0180 19F6   BTFSC  76,3           if(segment.3)           // D2.d
0181 1591   BSF    11,3             LCDD01.3 = 1;
0182 1A76   BTFSC  76,4           if(segment.4)           // D2.e
0183 1414   BSF    14,0             LCDD04.0 = 1;
0184 1AF6   BTFSC  76,5           if(segment.5)           // D2.f
0185 1418   BSF    18,0             LCDD08.0 = 1;
0186 1B76   BTFSC  76,6           if(segment.6)           // D2.g
0187 1599   BSF    19,3             LCDD09.3 = 1;
}

// Update Minutes if enabled
if(Mode.MINUTES)

0188 1CF8   BTFSS  78,1
0189 29C6   GOTO   01C6h
018A
{
// Digit 3
index = Minutes & 0xf0;

018A 30F0   MOVLW F0h
018B 0571   ANDWF  71,W
018C 00F7   MOVWF  77
018D 1003   BCF    03,0           index >>= 4;
018E 0CF7   RRF    77

```

AN649

```
018F 1003    BCF    03,0
0190 0CF7    RRF    77
0191 1003    BCF    03,0
0192 0CF7    RRF    77
0193 1003    BCF    03,0
0194 0CF7    RRF    77
0195 00FB    MOVWF  7B                segment = SevenSegTable[index];
0196 3001    MOVLW 01h
0197 008A    MOVWF  0A
0198 08FB    MOVF   7B
0199 0877    MOVF   77,W
019A 2113    CALL  0113h
019B 1283    BCF    03,5
019C 00F6    MOVWF  76
019D 1876    BTFSC  76,0                if(segment.0)                // D3.a
019E 151D    BSF    1D,2                LCDD13.2 = 1;
019F 18F6    BTFSC  76,1                if(segment.1)                // D3.b
01A0 1518    BSF    18,2                LCDD08.2 = 1;
01A1 1976    BTFSC  76,2                if(segment.2)                // D3.c
01A2 1514    BSF    14,2                LCDD04.2 = 1;
01A3 19F6    BTFSC  76,3                if(segment.3)                // D3.d
01A4 1511    BSF    11,2                LCDD01.2 = 1;
01A5 1A76    BTFSC  76,4                if(segment.4)                // D3.e
01A6 1494    BSF    14,1                LCDD04.1 = 1;
01A7 1AF6    BTFSC  76,5                if(segment.5)                // D3.f
01A8 1519    BSF    19,2                LCDD09.2 = 1;
01A9 1B76    BTFSC  76,6                if(segment.6)                // D3.g
01AA 1515    BSF    15,2                LCDD05.2 = 1;

                                // Digit 4
01AB 300F    MOVLW 0Fh                index = Minutes & 0x0f;
01AC 0571    ANDWF  71,W
01AD 00F7    MOVWF  77
01AE 00FB    MOVWF  7B                segment = SevenSegTable[index];
01AF 3001    MOVLW 01h
01B0 008A    MOVWF  0A
01B1 08FB    MOVF   7B
01B2 0877    MOVF   77,W
01B3 118A    BCF    0A,3
01B4 2113    CALL  0113h
01B5 118A    BCF    0A,3
01B6 1283    BCF    03,5
01B7 00F6    MOVWF  76
01B8 1876    BTFSC  76,0                if(segment.0)                // D4.a
01B9 149D    BSF    1D,1                LCDD13.1 = 1;
01BA 18F6    BTFSC  76,1                if(segment.1)                // D4.b
01BB 1598    BSF    18,3                LCDD08.3 = 1;
01BC 1976    BTFSC  76,2                if(segment.2)                // D4.c
01BD 1594    BSF    14,3                LCDD04.3 = 1;
01BE 19F6    BTFSC  76,3                if(segment.3)                // D4.d
01BF 1491    BSF    11,1                LCDD01.1 = 1;
01C0 1A76    BTFSC  76,4                if(segment.4)                // D4.e
01C1 1495    BSF    15,1                LCDD05.1 = 1;
01C2 1AF6    BTFSC  76,5                if(segment.5)                // D4.f
01C3 151C    BSF    1C,2                LCDD12.2 = 1;
01C4 1B76    BTFSC  76,6                if(segment.6)                // D4.g
01C5 1499    BSF    19,1                LCDD09.1 = 1;
                                }

                                // Update Temperature
                                // Digit 5
01C6 30F0    MOVLW F0h                index = TempC&0xf0;
01C7 0575    ANDWF  75,W
01C8 00F7    MOVWF  77
01C9 1003    BCF    03,0                index >>= 4;
01CA 0CF7    RRF    77
```

```

01CB 1003   BCF    03,0
01CC 0CF7   RRF    77
01CD 1003   BCF    03,0
01CE 0CF7   RRF    77
01CF 1003   BCF    03,0
01D0 0CF7   RRF    77
01D1 00FB   MOVWF  7B                segment = SevenSegTable[index];
01D2 3001   MOVLW  01h
01D3 008A   MOVWF  0A
01D4 08FB   MOVF   7B
01D5 0877   MOVF   77,W
01D6 2113   CALL   0113h
01D7 1283   BCF    03,5
01D8 00F6   MOVWF  76
01D9 1876   BTFSC  76,0             if(segment.0)                // D5.a
01DA 179C   BSF    1C,7             LCDD12.7 = 1;
01DB 18F6   BTFSC  76,1             if(segment.1)                // D5.b
01DC 161C   BSF    1C,4             LCDD12.4 = 1;
01DD 1976   BTFSC  76,2             if(segment.2)                // D5.c
01DE 1614   BSF    14,4             LCDD04.4 = 1;
01DF 19F6   BTFSC  76,3             if(segment.3)                // D5.d
01E0 1790   BSF    10,7             LCDD00.7 = 1;
01E1 1A76   BTFSC  76,4             if(segment.4)                // D5.e
01E2 1794   BSF    14,7             LCDD04.7 = 1;
01E3 1AF6   BTFSC  76,5             if(segment.5)                // D5.f
01E4 141D   BSF    1D,0             LCDD13.0 = 1;
01E5 1B76   BTFSC  76,6             if(segment.6)                // D5.g
01E6 1798   BSF    18,7             LCDD08.7 = 1;

                                // Digit 6
01E7 300F   MOVLW  0Fh             index = TempC&0x0f;
01E8 0575   ANDWF  75,W
01E9 00F7   MOVWF  77
01EA 00FB   MOVWF  7B                segment = SevenSegTable[index];
01EB 3001   MOVLW  01h
01EC 008A   MOVWF  0A
01ED 08FB   MOVF   7B
01EE 0877   MOVF   77,W
01EF 118A   BCF    0A,3
01F0 2113   CALL   0113h
01F1 118A   BCF    0A,3
01F2 1283   BCF    03,5
01F3 00F6   MOVWF  76
01F4 1876   BTFSC  76,0             if(segment.0)                // D6.a
01F5 171C   BSF    1C,6             LCDD12.6 = 1;
01F6 18F6   BTFSC  76,1             if(segment.1)                // D6.b
01F7 1698   BSF    18,5             LCDD08.5 = 1;
01F8 1976   BTFSC  76,2             if(segment.2)                // D6.c
01F9 1694   BSF    14,5             LCDD04.5 = 1;
01FA 19F6   BTFSC  76,3             if(segment.3)                // D6.d
01FB 1710   BSF    10,6             LCDD00.6 = 1;
01FC 1A76   BTFSC  76,4             if(segment.4)                // D6.e
01FD 1714   BSF    14,6             LCDD04.6 = 1;
01FE 1AF6   BTFSC  76,5             if(segment.5)                // D6.f
01FF 1618   BSF    18,4             LCDD08.4 = 1;
0200 1B76   BTFSC  76,6             if(segment.6)                // D6.g
0201 1718   BSF    18,6             LCDD08.6 = 1;

                                // Turn on : if enabled
0202 1B78   BTFSC  78,6             if(Mode.COLON)
0203 1498   BSF    18,1             LCDD08.1 = 1;

                                // Turn on degrees symbol if enabled
0204 1BF8   BTFSC  78,7             if(Mode.DEGREES)
0205 169C   BSF    1C,5             LCDD12.5 = 1;

```

AN649

```
// Turn on PROG symbol if enabled
0206 1AF8    BTFSC  78,5    if(Mode.PROG)
0207 1415    BSF     15,0    LCDD05.0 = 1;

// Make copies of the LCD data registers
0208 0810    MOVF    10,W    LCDD02 = LCDD00;
0209 0092    MOVWF   12
020A 0811    MOVF    11,W    LCDD03 = LCDD01;
020B 0093    MOVWF   13
020C 0814    MOVF    14,W    LCDD06 = LCDD04;
020D 0096    MOVWF   16
020E 0815    MOVF    15,W    LCDD07 = LCDD05;
020F 0097    MOVWF   17
0210 0818    MOVF    18,W    LCDD10 = LCDD08;
0211 009A    MOVWF   1A
0212 0819    MOVF    19,W    LCDD11 = LCDD09;
0213 009B    MOVWF   1B
0214 081C    MOVF    1C,W    LCDD14 = LCDD12;
0215 009E    MOVWF   1E
0216 081D    MOVF    1D,W    LCDD15 = LCDD13;
0217 009F    MOVWF   1F

0218 1303    BCF     03,6    STATUS.RP1 = 0;    // Return to Bank 0
0219 0008    RETURN   return;

}

/*****
 * BlinkLCD
 * Function: This function is used in program mode to blink hours, minutes
 *           or day of the week depending on the current status.
 *****/
void BlinkLCD(bits which)
{
002B
021A 1283    BCF     03,5
021B 00AB    MOVWF   2B
021C 1C26    BTFSS   26,0    if(Flags.UPDATE)    // If UPDATE flag is set, blank the
021D 2A3C    GOTO    023Ch
021E
{
021E 1026    BCF     26,0    Flags.UPDATE = 0;    // Clear UPDATE flag
021F 3001    MOVLW   01h    if(which==0x01)    // Blank Hours
0220 022B    SUBWF   2B,W
0221 1D03    BTFSS   03,2
0222 2A28    GOTO    0228h
0223
{
0223 30E6    MOVLW   E6h    Mode = 0b11100110;
0224 1283    BCF     03,5
0225 00F8    MOVWF   78
0226 2112    CALL    0112h    UpdateLCD();
}
0227 2A3B    GOTO    023Bh
0228 3002    MOVLW   02h    else if(which == 0x02)    // Blank Minutes
0229 1283    BCF     03,5
022A 022B    SUBWF   2B,W
022B 1D03    BTFSS   03,2
022C 2A32    GOTO    0232h
022D 30E5    MOVLW   E5h    Mode = 0b11100101;
022E 1283    BCF     03,5
022F 00F8    MOVWF   78
0230 2112    CALL    0112h    UpdateLCD();
}
0231 2A3B    GOTO    023Bh
0232 3004    MOVLW   04h    else if(which == 0x04)    // Blank day of the week
0233 1283    BCF     03,5
0234 022B    SUBWF   2B,W
0235 1D03    BTFSS   03,2
0236 2A3B    GOTO    023Bh
```

```

0237 30E3    MOVLW  E3h                Mode = 0b11100011;
0238 1283    BCF      03,5
0239 00F8    MOVWF   78
023A 2112    CALL    0112h                UpdateLCD();
                                }
                                }
023B 2A40    GOTO    0240h                else // Turn on all groups if UPDATE
flag is
                                {
                                // cleared
023C 1426    BSF      26,0                Flags.UPDATE = 1; // Set the UPDATE flag
023D 30E7    MOVLW  E7h                Mode = 0b11100111; // Set bits in Mode to turn all on
023E 00F8    MOVWF   78
023F 2112    CALL    0112h                UpdateLCD();
                                }
0240 0008    RETURN
                                return;
                                }

/*****
*   DisplaySoundState
*   Function: When the SOUND button is pressed the state of the hourly beep
*             is toggle between on and off. This routine displays to the
*             *           user the state, Snd OF or Snd On.
*****/
void DisplaySoundState(void)
{
002C                unsigned char frames; // Frame count variable

0241 300C    MOVLW  0Ch                frames = FRAME_COUNT; // Initialize the frame counter
0242 1283    BCF      03,5
0243 00AC    MOVWF   2C

0244 1283    BCF      03,5                while(!Flags.FRAME); // Wait for next frame to occur
0245 1CA6    BTFSS  26,1
0246 2A44    GOTO    0244h
0247 1283    BCF      03,5                Flags.FRAME = 0; // Clear FRAME flag
0248 10A6    BCF      26,1

0249 1703    BSF      03,6                STATUS.RP1 = 1; // Change to Bank 2
024A 3080    MOVLW  80h                LCDD00 = 0b10000000; // Write data common to both
024B 0090    MOVWF   10
024C 300A    MOVLW  0Ah                LCDD01 = 0b00001010; // Snd On and Snd OF
024D 0091    MOVWF   11
024E 300E    MOVLW  0Eh                LCDD05 = 0b00001110;
024F 0095    MOVWF   15
0250 300A    MOVLW  0Ah                LCDD09 = 0b00001010;
0251 0099    MOVWF   19
0252 3009    MOVLW  09h                LCDD13 = 0b00001001;
0253 009D    MOVWF   1D
0254 1303    BCF      03,6                STATUS.RP1 = 0; // Change to Bank 0

0255 1E26    BTFSS  26,4                if(Flags.SOUND_STATE) // Sound on
0256 2A5F    GOTO    025Fh
0257
                                {
0257 1703    BSF      03,6                STATUS.RP1 = 1; // Change to Bank 2
0258 30FE    MOVLW  FEh                LCDD04 = 0b11111110; // Write data specific to Snd On
0259 0094    MOVWF   14
                                LCDD08 = 0b01001001;
025A 3049    MOVLW  49h
025B 0098    MOVWF   18
                                LCDD12 = 0b10010000;
025C 3090    MOVLW  90h
025D 009C    MOVWF   1C
                                }
025E 2A66    GOTO    0266h                else // Sound off
                                {
025F 1703    BSF      03,6                STATUS.RP1 = 1; // Change to Bank 2
0260 30DE    MOVLW  DEh                LCDD04 = 0b11011110; // Write data specific to Snd OFF
0261 0094    MOVWF   14

```

AN649

```
0262 3059    MOVLW  59h                LCDD08 = 0b01011001;
0263 0098    MOVWF   18
0264 30D0    MOVLW  D0h                LCDD12 = 0b11010000;
0265 009C    MOVWF   1C
                                }
0266 1303    BCF     03,6                STATUS.RP1 = 0;          // Change to Bank 0

                                while(frames)    // Delay a specific number of frames so
                                {                          // the user can see the message

0267 1283    BCF     03,5
0268 08AC    MOVF   2C
0269 1903    BTFSC  03,2
026A 2A71    GOTO   0271h
026B                                if(Flags.FRAME)    // Decrement frames until 0
                                {
026B 1283    BCF     03,5
026C 1CA6    BTFSS  26,1
026D 2A70    GOTO   0270h
026E 10A6    BCF     26,1                Flags.FRAME = 0;
026F 03AC    DECF   2C                frames--;
                                }
0270 2A67    GOTO   0267h
0271 0008    RETURN

                                }

                                #include "time.c"          // Contains programs for timekeeping
/*****
* Filename: TIME.C
*****/
* Author:      Rodger Richey
* Company:    Microchip Technology Incorporated
* Revision:    A0
* Date:       6-14-96
* Compiled using MPLAB-C Version 00.00.14
*****/
* This file contains the routines to increment time and set the time in
* program mode.
*****/

/*****
* IncMinutes
* Function: Increments minutes and performs checks. Minutes is in BCD.
*****/
void IncMinutes(void)
{
0272 1283    BCF     03,5                Seconds = 0;          // Clear Seconds
0273 01F0    CLRF   70
0274 0AF1    INCF   71                Minutes++;          // Increment Minutes
0275 300F    MOVLW  0Fh                if( (Minutes&0x0f) > 0x09) // Check for BCD overflow
0276 0571    ANDWF  71,W
0277 00FB    MOVWF  7B
0278 3009    MOVLW  09h
0279 027B    SUBWF  7B,W
027A 1D03    BTFSS  03,2
027B 1C03    BTFSS  03,0
027C 2A82    GOTO   0282h
027D                                {
027D 30F0    MOVLW  F0h                Minutes &= 0xf0;
027E 1283    BCF     03,5
027F 05F1    ANDWF  71
0280 3010    MOVLW  10h                Minutes += 0x10;
0281 07F1    ADDWF  71
                                }
0282 0008    RETURN                return;
                                }

/*****
```

```

*   IncHours
*   Function: Increments hours and performs checks.  Hours is in BCD.
*****
void IncHours(void)
{
0283 1283   BCF    03,5           if(!Flags.PROGRAM) // If program mode, do not clear Minutes
0284 1DA6   BTFSS  26,3
0285 01F1   CLRF   71           Minutes = 0;
0286 0AF2   INCF   72           Hours++;           // Increment Hours
0287 300F   MOVLW  0Fh         if( (Hours&0x0f) > 0x09) // Check for BCD overflow
0288 0572   ANDWF  72,W
0289 00FB   MOVWF  7B
028A 3009   MOVLW  09h
028B 027B   SUBWF  7B,W
028C 1D03   BTFSS  03,2
028D 1C03   BTFSS  03,0
028E 2A94   GOTO   0294h
028F                               {
028F 30F0   MOVLW  F0h           Hours &= 0xf0;
0290 1283   BCF    03,5
0291 05F2   ANDWF  72
0292 3010   MOVLW  10h         Hours += 0x10;
0293 07F2   ADDWF  72
                                }
0294 3012   MOVLW  12h         if(Hours == 0x12) // If hours = 12 then change AM/PM
0295 1283   BCF    03,5
0296 0272   SUBWF  72,W
0297 1D03   BTFSS  03,2
0298 2AA8   GOTO   02A8h
0299                               { // and day of the week accordingly
0299 1283   BCF    03,5           if(LStatus.AMPM) // If PM
029A 1FF3   BTFSS  73,7
029B 2AA7   GOTO   02A7h
029C                               {
029C 13F3   BCF    73,7           LStatus.AMPM = 0; // change to AM
029D 19A6   BTFSC  26,3           if(!Flags.PROGRAM) // If not prog mode, increment day
029E 2AA6   GOTO   02A6h
029F                               { // of the week and check for overflow
029F 0AF4   INCF   74           DayOfWeek++;
02A0 3007   MOVLW  07h         if(DayOfWeek == 0x07)
02A1 0274   SUBWF  74,W
02A2 1D03   BTFSS  03,2
02A3 2AA6   GOTO   02A6h
02A4 1283   BCF    03,5           DayOfWeek = 0;
02A5 01F4   CLRF   74
                                }
                                }
02A6 2AA8   GOTO   02A8h           else // Otherwise change to PM
02A7 17F3   BSF    73,7           LStatus.AMPM = 1;
                                }
02A8 1283   BCF    03,5           if(Flags.SOUND_STATE) // Start hourly beep if enabled
02A9 1E26   BTFSS  26,4
02AA 2AAC   GOTO   02ACh
02AB 2100   CALL   0100h           StartBEEP();

02AC 3013   MOVLW  13h         if(Hours == 0x13) // If hours has overflowed, set to 1
02AD 1283   BCF    03,5
02AE 0272   SUBWF  72,W
02AF 1D03   BTFSS  03,2
02B0 2AB4   GOTO   02B4h
02B1 3001   MOVLW  01h         Hours = 0x01;
02B2 1283   BCF    03,5
02B3 00F2   MOVWF  72

02B4 0008   RETURN

return;
}

```

AN649

```

/*****
 *   SetTime
 *   Function: This routine sets the time in program mode.  Allows hours,
 *             minutes and day of the week to be configured.
 *****/
void SetTime(void)
{
    02B5 300C    MOVLW  0Ch           FrameCnt = FRAME_COUNT; // Initialize the frame counter
    02B6 1283    BCF     03,5
    02B7 00AA    MOVWF  2A

    while(!Flags.SET) // Wait for the SET button to be
    {                  // hit before advancing to minutes

        02B8 1283    BCF     03,5
        02B9 1BA6    BTFSC  26,7
        02BA 2AD2    GOTO   02D2h

        02BB                if(Flags.UP) // If UP button pressed, inc Minutes
        {
            02BB 1F26    BTFSS  26,6
            02BC 2ABF    GOTO   02BFh
            02BD 1326    BCF     26,6           Flags.UP = 0;
            02BE 2283    CALL   0283h           IncHours();
        }

        02BF 1283    BCF     03,5           if(Flags.FRAME) // Toggle display state (blink) every
        02C0 1CA6    BTFSS  26,1
        02C1 2ACC    GOTO   02CCh
        02C2                {
            02C2 10A6    BCF     26,1           // FRAME_COUNT frames
            02C3 03AA    DECF   2A           Flags.FRAME = 0;
            02C4 08AA    MOVF   2A           FrameCnt--;
            02C5 1D03    BTFSS  03,2           if(!FrameCnt) // If frame count = zero, toggle state
            02C6 2ACC    GOTO   02CCh
            02C7                {
            02C7 300C    MOVLW  0Ch           FrameCnt = FRAME_COUNT;
            02C8 1283    BCF     03,5
            02C9 00AA    MOVWF  2A
            02CA 3001    MOVLW  01h           BlinkLCD(0x01);
            02CB 221A    CALL   021Ah
            }
        }

        02CC 3005    MOVLW  05h           if(Ticks == 5) // If no button pressed in 5 secs
        02CD 1283    BCF     03,5
        02CE 0229    SUBWF  29,W
        02CF 1903    BTFSC  03,2
        02D0 0008    RETURN
        02D1 2AB8    GOTO   02B8h           return; // exit program mode
        02D2 1283    BCF     03,5
        02D3 13A6    BCF     26,7
        02D4 300C    MOVLW  0Ch           FrameCnt = FRAME_COUNT; // Initialize the frame counter
        02D5 00AA    MOVWF  2A

        while(!Flags.SET) // Wait for the SET button to be hit
        {                  // before advancing to day of the week

            02D6 1283    BCF     03,5
            02D7 1BA6    BTFSC  26,7
            02D8 2AF7    GOTO   02F7h

            02D9                if(Flags.UP) // If the UP button is hit, inc Minutes
            {
                02D9 1F26    BTFSS  26,6
                02DA 2AE4    GOTO   02E4h
                02DB 1326    BCF     26,6           Flags.UP = 0;
                02DC 2272    CALL   0272h           IncMinutes();
                02DD 3060    MOVLW  60h           if(Minutes >= 0x60) // Check for upper limit
                02DE 1283    BCF     03,5
                02DF 0271    SUBWF  71,W
                02E0 1C03    BTFSS  03,0
                02E1 2AE4    GOTO   02E4h
                02E2 1283    BCF     03,5           Minutes = 0;
                02E3 01F1    CLRWF  71
            }
        }
    }
}

```

```

02E4 1283   BCF    03,5           if(Flags.FRAME)      // Toggle the display state (blink)
02E5 1CA6   BTFSS  26,1
02E6 2AF1   GOTO   02F1h
02E7                                {                          // every FRAME_COUNT frames
02E7 10A6   BCF    26,1           Flags.FRAME = 0;
02E8 03AA   DECF   2A             FrameCnt--;
02E9 08AA   MOVF   2A             if(!FrameCnt)       // If FRAME_COUNT=zero, toggle state
02EA 1D03   BTFSS  03,2
02EB 2AF1   GOTO   02F1h
02EC                                {
02EC 300C   MOVLW  0Ch           FrameCnt = FRAME_COUNT;
02ED 1283   BCF    03,5
02EE 00AA   MOVWF  2A
02EF 3002   MOVLW  02h           BlinkLCD(0x02);
02F0 221A   CALL   021Ah
                                }
                                }
02F1 3005   MOVLW  05h           if(Ticks == 5)      // If no button pressed in 5 secs
02F2 1283   BCF    03,5
02F3 0229   SUBWF  29,W
02F4 1903   BTFSC  03,2
02F5 0008   RETURN
                                return;                          // exit program mode
02F6 2AD6   GOTO   02D6h
02F7 1283   BCF    03,5           Flags.SET = 0;
02F8 13A6   BCF    26,7

02F9 300C   MOVLW  0Ch           FrameCnt = FRAME_COUNT; // Initialize the frame counter
02FA 00AA   MOVWF  2A

                                while(!Flags.SET)       // Wait for SET button to be hit
                                {                          // before exiting program mode
02FB 1283   BCF    03,5
02FC 1BA6   BTFSC  26,7
02FD 2B1B   GOTO   031Bh
02FE                                if(Flags.UP)           // If the UP key is pressed, inc
02FE 1F26   BTFSS  26,6           {                          // inc. day of week
02FF 2B08   GOTO   0308h
0300 1326   BCF    26,6           Flags.UP = 0;
0301 0AF4   INCF   74             DayOfWeek++;
0302 3007   MOVLW  07h           if(DayOfWeek == 0x07) // Check for overflow
0303 0274   SUBWF  74,W
0304 1D03   BTFSS  03,2
0305 2B08   GOTO   0308h
0306 1283   BCF    03,5           DayOfWeek = 0;
0307 01F4   CLRFB  74
                                }
0308 1283   BCF    03,5           if(Flags.FRAME)     // Toggle display state every
0309 1CA6   BTFSS  26,1
030A 2B15   GOTO   0315h
030B                                {                          // every FRAME_COUNT frames
030B 10A6   BCF    26,1           Flags.FRAME = 0;
030C 03AA   DECF   2A             FrameCnt--;
030D 08AA   MOVF   2A             if(!FrameCnt)
030E 1D03   BTFSS  03,2
030F 2B15   GOTO   0315h
0310                                {
0310 300C   MOVLW  0Ch           FrameCnt = FRAME_COUNT;
0311 1283   BCF    03,5
0312 00AA   MOVWF  2A
0313 3004   MOVLW  04h           BlinkLCD(0x04);
0314 221A   CALL   021Ah
                                }
                                }
0315 3005   MOVLW  05h           if(Ticks == 5)      // If no button pressed in 5 secs
0316 1283   BCF    03,5
0317 0229   SUBWF  29,W
0318 1903   BTFSC  03,2
0319 0008   RETURN
                                return;                          // exit program mode

```

AN649

```
031A 2AFB    GOTO    02FBh        }
031B 1283    BCF     03,5        Flags.SET = 0;
031C 13A6    BCF     26,7
031D 0008    RETURN                   return;
                                }

/*****
 *   Init924
 *   Function: This routine initializes the peripherals and CPU of the
 *             PIC16C924.
 *****/
void Init924(void)
{
031E 1283    BCF     03,5        STATUS.RP0 = 0;        // Change to Bank 0
031F 1303    BCF     03,6        STATUS.RP1 = 0;
0320 3053    MOVLW  53h        OPTION = 0b01010011; // Pull-ups on,T0 int clk source,
0321 1683    BSF     03,5
0322 0081    MOVWF  01
                                // Prescaler assigned to T0, 1:16
0323 30C1    MOVLW  C1h        ADCON0 = 0b11000001; // Internal RC clk src, Ch0, A/D on
0324 1283    BCF     03,5
0325 009F    MOVWF  1F
0326 0185    CLRF   05        PORTA = 0;        // Clear ports A,B,C
0327 0186    CLRF   06        PORTB = 0;
0328 0187    CLRF   07        PORTC = 0;
0329 3007    MOVLW  07h        TRISA = 0b00000111; // RAZ:RA1> digi outputs
032A 1683    BSF     03,5
032B 0085    MOVWF  05
032C 30F0    MOVLW  F0h        TRISB = 0xf0;        // Upper 4 pins are inputs for keys
032D 0086    MOVWF  06
032E 3003    MOVLW  03h        TRISC = 0x03;        // RC<0:1> used for Timer1
032F 0087    MOVWF  07        // external crystal
0330 3004    MOVLW  04h        ADCON1 = 0b00000100; // RA<0:1,3> are analog
0331 009F    MOVWF  1F
0332 1283    BCF     03,5        PIR1.ADIF = 0;        // Clear A/D interrupt flag
0333 130C    BCF     0C,6
0334 1683    BSF     03,5        PIE1.ADIE = 1;        // Enable A/D interrupt
0335 170C    BSF     0C,6

0336 1283    BCF     03,5        Temp = PORTB;        // Clear mismatch condition
0337 0806    MOVF   06,W        // on PORTB
0338 00A7    MOVWF  27
0339 100B    BCF     0B,0        INTCON.RBIF = 0;        // Clear PORTB interrupt flag
033A 158B    BSF     0B,3        INTCON.RBIE = 1;        // Enable PORTB interrupt

033B 3080    MOVLW  80h        TMR1H = 0x80;        // Initialize Timer1 to 0x8000
033C 008F    MOVWF  0F
033D 018E    CLRF   0E        TMR1L = 0x00;
033E 300F    MOVLW  0Fh        T1CON = 0b00001111; // Timer1 1:1 prescale, Osc
033F 0090    MOVWF  10        // enabled, no sync, external
                                // clock source, Timer1 on
0340 100C    BCF     0C,0        PIR1.TMR1IF = 0;        // Clear Timer1 Overflow int flag
0341 1683    BSF     03,5        PIE1.TMR1IE = 1;        // Enable T1 Overflow interrupt
0342 140C    BSF     0C,0

0343 1703    BSF     03,6        STATUS.RP1 = 1;        // Go to Bank 2
0344 3006    MOVLW  06h        LCDPS = 6;        // Set LCD frame freq to 37 Hz,
0345 1283    BCF     03,5        // Timer1 clk source
0346 008E    MOVWF  0E
0347 30FF    MOVLW  FFh        LCDSE = 0xff;        // Ports D,E,F,G are all LCD pins
0348 008D    MOVWF  0D
0349 3017    MOVLW  17h        LCDCON = 0b00010111; // Drive in SLEEP,charge pump on,Timer1 clk src
034A 008F    MOVWF  0F        // Timer1 clk src 1/4 mux, 1/3 bias
034B 0190    CLRF   10        LCDD00 = 0;        // Clear all LCD data registers
034C 0191    CLRF   11        LCDD01 = 0;
```

```

034D 0192    CLRF    12          LCDD02 = 0;
034E 0193    CLRF    13          LCDD03 = 0;
034F 0194    CLRF    14          LCDD04 = 0;
0350 0195    CLRF    15          LCDD05 = 0;
0351 0196    CLRF    16          LCDD06 = 0;
0352 0197    CLRF    17          LCDD07 = 0;
0353 0198    CLRF    18          LCDD08 = 0;
0354 0199    CLRF    19          LCDD09 = 0;
0355 019A    CLRF    1A          LCDD10 = 0;
0356 019B    CLRF    1B          LCDD11 = 0;
0357 019C    CLRF    1C          LCDD12 = 0;
0358 019D    CLRF    1D          LCDD13 = 0;
0359 019E    CLRF    1E          LCDD14 = 0;
035A 019F    CLRF    1F          LCDD15 = 0;
035B 178F    BSF     0F,7        LCDCON.LCDEN = 1;    // Enable the LCD Module
035C 1303    BCF     03,6        STATUS.RP1 = 0;     // Go to Bank 0
035D 138C    BCF     0C,7        PIR1.LCDIF = 0;    // Clear LCD interrupt flag
035E 1683    BSF     03,5        PIE1.LCDIE = 1;   // Enable LCD interrupt
035F 178C    BSF     0C,7

0360 1283    BCF     03,5        Seconds = 0;        // Initialize data variables
0361 01F0    CLRF    70
0362 01F1    CLRF    71          Minutes = 0;
0363 3012    MOVLW  12h         Hours = 0x12;      // Set time to 12:00AM Sunday
0364 00F2    MOVWF  72
0365 01F3    CLRF    73          LStatus = 0;
0366 01F4    CLRF    74          DayOfWeek = 0;
0367 3011    MOVLW  11h         Flags = 0b00010001;
0368 00A6    MOVWF  26
0369 01F5    CLRF    75          TempC = 0;
036A 01A9    CLRF    29          Ticks = 0;
036B 3004    MOVLW  04h         Count = BEEP_COUNT;
036C 00A8    MOVWF  28
036D 30C7    MOVLW  C7h         Mode = 0b11000111; // Turn on :,degrees,hours,
036E 00F8    MOVWF  78          // minutes, day of week

036F 170B    BSF     0B,6        INTCON.PEIE = 1;   // Enable peripheral interrupts
0370 178B    BSF     0B,7        INTCON.GIE = 1;   // Enable global interrupts
0371 0008    RETURN
                                return;
                                }

/*****
*   main
*   Function: Controls the clock. Calls routines to update the LCD panel,
*             play music, program mode, and display state of sound.
*****/
void main(void)
{
0372 231E    CALL    031Eh      Init924();        // Initialize the PIC16C924

                                while(1)
                                {
0373 1283    BCF     03,5        if(Flags.UPDATE&&Flags.FRAME) // Refresh the LCD
0374 1C26    BTFSS  26,0        // data registers
0375 2B7D    GOTO   037Dh
0376 1CA6    BTFSS  26,1
0377 2B7D    GOTO   037Dh
0378                                {
                                // based on new data
0378 1283    BCF     03,5        Flags.UPDATE = 0;    // Clear the UPDATE flag
0379 1026    BCF     26,0
037A 10A6    BCF     26,1        Flags.FRAME = 0;    // Clear the FRAME flag
037B 2112    CALL    0112h    UpdateLCD();    // Update LCD data regs
                                }
                                else if(!Flags.UPDATE&&Flags.FRAME) // Clear FRAME
037C 2B84    GOTO   0384h    Flags.FRAME = 0;    // flag if no UPDATE
037D 1283    BCF     03,5
037E 1826    BTFSC  26,0

```

AN649

```
037F 2B84 GOTO 0384h
0380 1CA6 BTFSS 26,1
0381 2B84 GOTO 0384h
0382 1283 BCF 03,5
0383 10A6 BCF 26,1

0384 1283 BCF 03,5 if(Flags.SET) // Enter program mode
0385 1FA6 BTFSS 26,7
0386 2B9A GOTO 039Ah
0387 {
0387 13A6 BCF 26,7 Flags.SET = 0; // Clear the SET, UP flags
0388 1326 BCF 26,6 Flags.UP = 0;
0389 01A9 CLRF 29 Ticks = 0; // Clear the Ticks
038A 15A6 BSF 26,3 Flags.PROGRAM = 1; // Change to program mode
038B 1283 BCF 03,5 while(!Flags.FRAME); // Wait for next frame to occur
038C 1CA6 BTFSS 26,1
038D 2B8B GOTO 038Bh
038E 1283 BCF 03,5 Flags.FRAME = 0; // Clear FRAME flag
038F 10A6 BCF 26,1
0390 30E7 MOVLW E7h Mode = 0b11100111; // Enable PROG icon on LCD
0391 00F8 MOVWF 78
0392 2112 CALL 0112h UpdateLCD(); // Refresh the LCD

0393 22B5 CALL 02B5h SetTime(); // Call program to set time

0394 1283 BCF 03,5 Flags.PROGRAM = 0; // Exit program mode
0395 11A6 BCF 26,3
0396 01F0 CLRF 70 Seconds = 0; // Clear seconds
0397 1426 BSF 26,0 Flags.UPDATE = 1; // Set UPDATE flag
0398 30C7 MOVLW C7h Mode = 0b11000111; // Reset display mode,
0399 00F8 MOVWF 78 // PROG icon off
}

039A 1EF3 BTFSS 73,5 if(LStatus.SOUND) // Enable/disable hourly beep
039B 2B9E GOTO 039Eh
039C {
039C 12F3 BCF 73,5 LStatus.SOUND = 0; // Reset SOUND flag
039D 2241 CALL 0241h DisplaySoundState(); // Display state of hourly beep
}

039E 1283 BCF 03,5 if(!Flags.SLEEP_STATE) // If 924 can go to sleep,
039F 1EA6 BTFSS 26,5 // go ahead
03A0 0063 SLEEP SLEEP();
03A1 2B73 GOTO 0373h }
03A2 0008 RETURN }

/*****
* __INT
* Function: Interrupt service routine for LCD, PORTB, Timer2, Timer1,
* Timer0, and A/D
*****/
0004 2BA3 GOTO 03A3h void __INT(void)
03A3 {
asm // "push" W and STATUS
03A3 00FA movwf temp_WREG
03A4 0E03 swapf STATUS,W
03A5 1283 bcf STATUS,RP0
03A6 1303 bcf STATUS,RP1
03A7 00FE movwf temp_STATUS
03A8 0804 movf FSR,W
03A9 00FF movwf temp_FSR
endasm

03AA 1283 BCF 03,5 if(PIR1.LCDIF) // Ok to write to LCD data regs
03AB 1F8C BTFSS 0C,7
03AC 2BAF GOTO 03AFh
```

```

03AD                                     {
03AD 14A6   BSF    26,1                   Flags.FRAME = 1;    // Set FRAME flag
03AE 138C   BCF    0C,7                   PIR1.LCDIF = 0;    // Clear LCD interrupt flag
                                        }

03AF 1C0B   BTFSS 0B,0                   if(INTCON.RBIF)    // Key press/release detected
03B0 2BD7   GOTO  03D7h
03B1                                     {
03B1 3005   MOVLW 05h                     Delay_Ms_4MHz(5);  // Debounce for 5msec
03B2 2432   CALL  0432h
03B3 1283   BCF    03,5                   Temp = PORTB;      // Read PORTB
03B4 0806   MOVF  06,W
03B5 00A7   MOVWF 27
03B6 3014   MOVLW 14h                     Delay_Ms_4MHz(20); // Debounce for 20msec more
03B7 2432   CALL  0432h
03B8 30F0   MOVLW F0h                     if(Temp!=0xf0 && Temp==PORTB) // If same state as
03B9 1283   BCF    03,5                                     // previous read
03BA 0227   SUBWF 27,W
03BB 1903   BTFSC 03,2
03BC 2BD4   GOTO  03D4h
03BD 1283   BCF    03,5
03BE 0827   MOVF  27,W
03BF 0206   SUBWF 06,W
03C0 1D03   BTFSS 03,2
03C1 2BD4   GOTO  03D4h
03C2                                     {
03C2 2100   CALL  0100h                   StartBEEP();      // Beep when key is pressed

03C3 1283   BCF    03,5                   if(!Temp.SET)     // Set the SET flag
03C4 1FA7   BTFSS 27,7
03C5 17A6   BSF    26,7                   Flags.SET = 1;
03C6 1F27   BTFSS 27,6                   if(!Temp.UP)      // Set the UP flag
03C7 1726   BSF    26,6                   Flags.UP = 1;
03C8 118A   BCF    0A,3                   if(!Temp.SOUND&&!Flags.PROGRAM)
03C9 1AA7   BTFSC 27,5
03CA 2BD4   GOTO  03D4h
03CB 19A6   BTFSC 26,3
03CC 2BD4   GOTO  03D4h
03CD                                     {
03CD 1283   BCF    03,5                   // Toggle the SOUND state
03CE 1E26   BTFSS 26,4                   if(Flags.SOUND_STATE)
03CF 2BD2   GOTO  03D2h
03D0 1226   BCF    26,4                   Flags.SOUND_STATE = 0;
03D1 2BD3   GOTO  03D3h                   else
03D2 1626   BSF    26,4                   Flags.SOUND_STATE = 1;
03D3 16F3   BSF    73,5                   LStatus.SOUND = 1; // Set the SOUND flag
                                        }
03D4 1283   BCF    03,5                   Ticks = 0;        // Reset Ticks, because key
03D5 01A9   CLRf   29                          // was pressed
03D6 100B   BCF    0B,0                   INTCON.RBIF = 0;  // Clear PORTB interrupt flag
                                        }

03D7 1C8C   BTFSS 0C,1                   if(PIR1.TMR2IF&&PIE1.TMR2IE)// T2 Overflow used for beep
03D8 2BEE   GOTO  03EEh
03D9 1683   BSF    03,5
03DA 1C8C   BTFSS 0C,1
03DB 2BEE   GOTO  03EEh
03DC                                     {
03DC 1683   BSF    03,5                   if(PIE1.TMR2IE)   // If Timer2 int is enabled
03DD 1C8C   BTFSS 0C,1
03DE 2BEC   GOTO  03ECh
03DF                                     {
03DF 1283   BCF    03,5                   Count--;           // Decrement count
03E0 03A8   DECF  28
03E1 08A8   MOVF  28                   if(!Count)        // If count has reached zero

```

AN649

```
03E2 1D03    BTFSS  03,2
03E3 2BEC    GOTO   03ECh
03E4
03E4 1283    BCF     03,5
03E5 0197    CLRF   17
03E6 0192    CLRF   12
03E7 0195    CLRF   15
03E8 1683    BSF    03,5
03E9 108C    BCF    0C,1
03EA 1283    BCF    03,5
03EB 12A6    BCF    26,5

                                {
                                CCP1CON = 0;           // Disable CCP module
                                T2CON = 0;           // Disable Timer2
                                CCPR1L = 0;         // Clear the Duty Cycle
                                PIE1.TMR2IE = 0;    // Disable Timer2 Interrupt
                                Flags.SLEEP_STATE = 0; // Enable 924 to SLEEP
                                }
                                }
03EC 1283    BCF    03,5
03ED 108C    BCF    0C,1

                                }

03EE 1283    BCF    03,5
03EF 1C0C    BTFSS  0C,0
03F0 2C23    GOTO   0423h
03F1
03F1 19A6    BTFSS  26,3
03F2 2C16    GOTO   0416h
03F3
03F3 0AF0    INCF   70
03F4 300F    MOVLW  0Fh
03F5 0570    ANDWF  70,W
03F6 00FB    MOVWF  7B
03F7 3009    MOVLW  09h
03F8 027B    SUBWF  7B,W
03F9 1D03    BTFSS  03,2
03FA 1C03    BTFSS  03,0
03FB 2C01    GOTO   0401h
03FC
03FC 30F0    MOVLW  F0h
03FD 1283    BCF    03,5
03FE 05F0    ANDWF  70
03FF 3010    MOVLW  10h
0400 07F0    ADDWF  70

                                {
                                Seconds += 0x10;
                                }

0401 3060    MOVLW  60h
0402 1283    BCF    03,5
0403 0270    SUBWF  70,W
0404 1C03    BTFSS  03,0
0405 2C07    GOTO   0407h
0406 2272    CALL   0272h
0407 3060    MOVLW  60h
0408 1283    BCF    03,5
0409 0271    SUBWF  71,W
040A 1C03    BTFSS  03,0
040B 2C0D    GOTO   040Dh
040C 2283    CALL   0283h

                                IncMinutes(); // increment minutes routine
                                if(Minutes >= 0x60) // check for hours overflow

                                IncHours(); // increment hours routine

040D 1283    BCF    03,5
040E 178F    BSF    0F,7
040F 1426    BSF    26,0

                                TMR1H |= 0x80; // Set Timer1 to 0x8000
                                // + current time
                                Flags.UPDATE = 1; // Set UPDATE flag

0410 1F78    BTFSS  78,6
0411 2C14    GOTO   0414h
0412 1378    BCF    78,6
0413 2C15    GOTO   0415h
0414 1778    BSF    78,6

                                if(Mode.COLON) // Toggle whether the colon is
                                Mode.COLON = 0; // on or off every second
                                else
                                Mode.COLON = 1;
                                }
0415 2C17    GOTO   0417h
0416 0AA9    INCF   29

                                else // If in program mode
                                Ticks++; // inc Ticks, used for timeout
```

```

0417 1683   BSF    03,5           TRISA.THERM_GND = 0; // Apply power to thermistor
0418 1105   BCF    05,2
0419 3002   MOVLW 02h           Delay_10xUs_4MHz(2); // Allow 20us for sampling
041A 243C   CALL   043Ch
041B 1283   BCF    03,5           ADCON0.GO = 1;      // Start a temp A/D conversion
041C 151F   BSF    1F,2
041D 0000   NOP
041E 0000   NOP           NOP();           // Wait for charging cap to
041F 1683   BSF    03,5           TRISA.THERM_GND = 1; // Remove power from thermistor
0420 1505   BCF    05,2
0421 1283   BCF    03,5           PIR1.TMR1IF = 0;   // Clear Timer1 interrupt flag
0422 100C   BCF    0C,0
    }

0423 1F0C   BTFSS 0C,6           if(PIR1.ADIF)      // A/D conversion complete
0424 2C2B   GOTO   042Bh
0425
    {
0425 018A   CLRf   0A           TempC = ThermTable[ADRES]; // Use converted value
0426 081E   MOVF   1E,W           // for table
0427 2005   CALL   0005h
0428 1283   BCF    03,5
0429 00F5   MOVWF 75
042A 130C   BCF    0C,6           PIR1.ADIF = 0;     // lookup of temperature
    }

    #asm                // "pop" W and STATUS
042B 087F   movf   temp_FSR,W
042C 0084   movwf  FSR
042D 0E7E   swapf  temp_STATUS,W
042E 0083   movwf  STATUS
042F 0EFA   swapf  temp_WREG,F
0430 0E7A   swapf  temp_WREG,W
    #endasm

0431 0009   RETFIE           return;
    }

/*****/

void Delay_Ms_4MHz(registerw delay)
/*
  Clock Speed = 4MHz
  Inst. Clock = 1MHz
  Inst. dur. = lus */
0000
    {
    #asm
0432 1283   BCF    STATUS, RP0
0433 00FB   MOVWF  __WImage
    DLMS4M1
    RADIX  DEC
0434 30F9   MOVLW  249
0435 0084   MOVWF  FSR
    DLMS4M2
0436 0000   NOP
0437 0B84   DECFSZ  FSR
0438 2C36   GOTO   DLMS4M2

0439 0BFB   DECFSZ  __WImage
043A 2C34   GOTO   DLMS4M1
    #endasm
043B 0008   RETURN
    }

/*****/

void Delay_10xUs_4MHz(registerw delay)
/*

```

AN649

```

                                Clock Freq. = 4MHz
                                Inst. Clock = 1MHz
                                Inst. dur. = 1000ns */
0000                                {
                                #asm
043C 1283                            BCF     STATUS, RPO
043D 00FB                            MOVWF  __WImage
                                DL10XMS4M
                                DL10XMS4M__ REPT 7
                                NOP
                                ENDM

043E 0000
043F 00 00
0440 00 00
0441 00 00
0442 00 00
0443 00 00
0444 00 00
0445 FB 0B                            DECFSZ __WImage
0446 3E 2C                            GOTO  DL10XMS4M
                                #endasm
0447 0008    RETURN                    }
                                /*****/

0000 3003    MOVLW  03h
0001 008A    MOVWF  0A
0002 2B72    GOTO   0372h

ROM USAGE MAP

    0000 to 0002    0004 to 0447
    Total ROM used 0447

Errors      :    0
Warnings   :    0
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products 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 PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be 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 product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

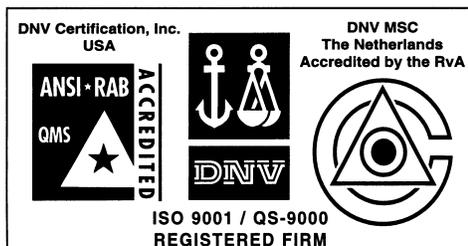
dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



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



MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

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

Rocky Mountain

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

Atlanta

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

Boston

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

Chicago

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

Dallas

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

Detroit

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

Kokomo

2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles

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

New York

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

San Jose

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

Toronto

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

ASIA/PACIFIC

Australia

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

China - Beijing

Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu

Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou

Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai

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

China - Shenzhen

Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong

Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India

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

Japan

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

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore

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

Taiwan

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

EUROPE

Denmark

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

France

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

Germany

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

Italy

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

United Kingdom

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

01/18/02