

Ratiometric Sensing Using the PIC16C774

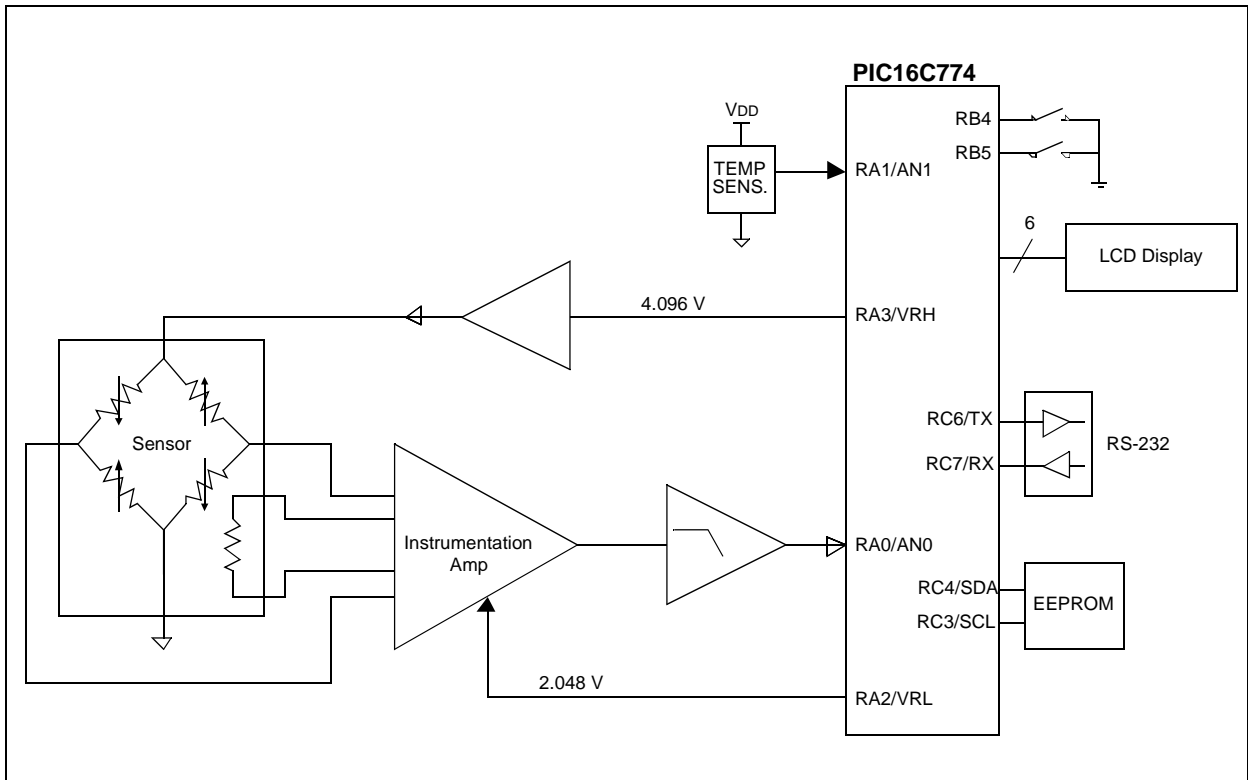
Authors: Steve Bowling
Microchip Technology Inc.

Other useful features of the microcontroller include a 9-bit addressable USART for serial communications and Master Synchronous Serial Port (MSSP) that supports the I²C™ and SPI™ protocols.

INTRODUCTION

This application note shows how to use the PIC16C774 microcontroller (MCU) in a ratiometric sensing application. A block diagram of the application is shown in Figure 1. The design takes advantage of the advanced analog peripherals of the PIC16C774, including a 12-bit A/D converter and two on-chip voltage references.

FIGURE 1: BLOCK DIAGRAM FOR APPLICATION CIRCUIT



THEORY

Many types of sensors may be used in a ratiometric sensing application, including those for measuring force, acceleration, temperature, or position. A pressure sensor has been used here due to its wide availability and low cost.

Pressure sensors are classified by how they measure pressure. In general, there are three different types of pressure measurements; absolute, gauge, and differential. An absolute pressure sensor has the rear of the sensor diaphragm connected to a sealed cavity and is referenced to a near perfect vacuum (0 psi). Because of this, all measurements made with the sensor will include the effects of the current atmospheric pressure. In contrast, the rear cavity of the gauge pressure sensor is vented to the atmosphere. Measurements made with a gauge sensor are referenced to the current ambient pressure conditions and the sensor will give a reading of 0 psi when at rest. The differential pressure sensor is a special variation of the gauge sensor. The rear cavity of the differential pressure sensor is connected to an inlet port so the pressure difference between two points can be measured.

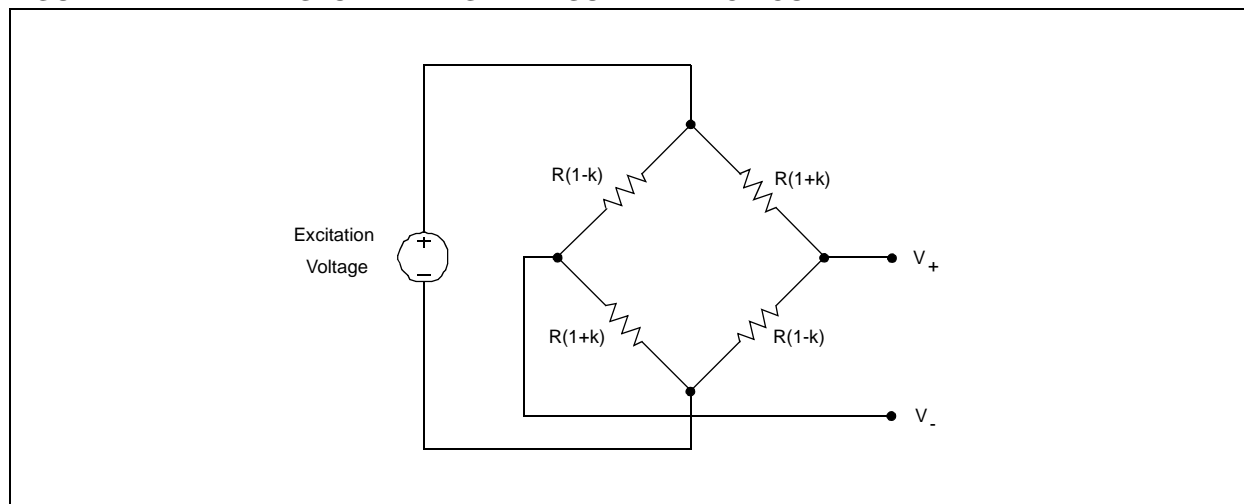
The pressure sensor chosen for this application is a Lucas Novasensor type (NPC-1210-50G). This sensor may be used for gauge pressure measurements up to 50 psi. The sensor is constructed using silicon micro-machining techniques to implant piezoresistive strain gauge elements in a Wheatstone bridge configuration on a mechanical diaphragm. The resistance of the piezoresistive elements changes when mechanical stress is applied to the diaphragm. Pressure sensors manufactured using silicon piezoresistive elements are

available from many manufacturers and are often referred to as 'solid-state' or IC pressure sensors because of the process used to manufacture them.

Piezoresistive elements are used in the pressure sensor because of their high sensitivity to applied stress. However, the elements are also very sensitive to variations in manufacturing process and temperature. An uncompensated or 'raw' pressure sensor will have large variations in its output offset and/or sensitivity. The sensor may also exhibit offset and sensitivity variations that are a function of temperature. The offset and sensitivity errors must be compensated using hardware or software techniques. To simplify the design process, internally compensated devices are available that have a specified offset and span over a given temperature range. The compensated sensor will typically have requirements for the excitation source. For example, many internally compensated sensors must be driven with a constant current source to achieve the offset, sensitivity and thermal accuracy given in the specifications. It is always best to check the sensor manufacturer's literature for the specific sensor requirements.

The piezoresistive elements of the pressure sensor are connected to form a Wheatstone bridge measurement circuit as shown in Figure 2. The four piezoresistive elements are arranged on the diaphragm of the sensor so two of the resistances will increase and the other two will decrease for a given pressure input. An electrical excitation (V_{EXC}) must be applied to the bridge as shown to produce an output voltage. The bridge produces an output voltage that is a function of the excitation source and the variation in resistance of the elements.

FIGURE 2: WHEATSTONE BRIDGE MEASUREMENT CIRCUIT



In general, a voltage source or current source may be used to excite the bridge.

The variable k in Figure 2 is the change in resistance normalized to a value of 1. Assuming the bridge excitation source is a voltage, and applying the rules for voltage division, the differential output of the bridge is given by:

EQUATION 1: DIFFERENTIAL OUTPUT

$$V_O = V_+ - V_- =$$

$$V_{EXC} \cdot \left[\left(\frac{R(1+k)}{R(1+k) + R(1-k)} \right) - \left(\frac{R(1-k)}{R(1+k) + R(1-k)} \right) \right]$$

This formula reduces to:

$$V_O = V_{EXC} \cdot k$$

The factor, k , becomes the output sensitivity of the bridge normalized to an excitation of 1 volt. Since the output sensitivity of a Wheatstone bridge circuit is a function of its excitation source, the source must be stable over time and temperature.

When an A/D converter is used to measure a bridge sensor output, errors due to drift of the excitation source can be eliminated by using the A/D converter reference as the source of excitation for the sensor bridge. This type of measurement is called ratiometric. Figure 3 shows the basic schematic diagram for a ratiometric measurement.

The measurement result obtained with an A/D converter is a comparison of input voltage to the A/D reference voltage. Specifically, the input voltage is divided by the reference voltage to obtain the conversion result and is given by:

EQUATION 2: CONVERSION RESULT

$$A/D \text{ RESULT} = \left(\frac{V_{IN}}{V_{REF}} \right) \cdot FULL-SCALE$$

If the expression for the sensor output, V_O , is substituted for V_{IN} , the expression for the A/D result becomes:

EQUATION 3: RATIOMETRIC A/D RESULT

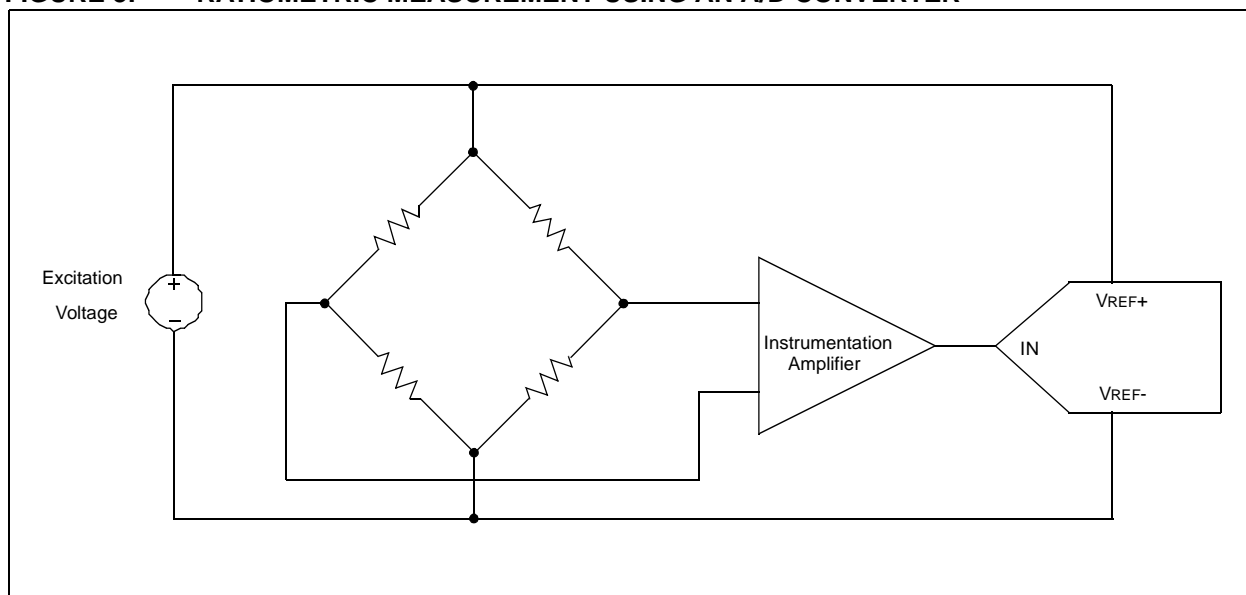
$$A/D \text{ RESULT} = \left(\frac{k \cdot V_{EXC}}{V_{REF}} \right) \cdot FULL-SCALE$$

For a ratiometric measurement, $V_{EXC} = V_{REF}$; therefore, the terms cancel and the expression for the A/D result reduces to:

$$A/D \text{ RESULT} = k \cdot FULL-SCALE$$

This formula shows that the ratiometric measurement result is only a function of the sensor gain and the full-scale result of the A/D converter. The effects due to drift of the excitation source have been eliminated.

FIGURE 3: RATIOMETRIC MEASUREMENT USING AN A/D CONVERTER



AN694

The output of the pressure sensor is a small differential voltage superimposed on a large common mode voltage. To provide a usable signal, the amplifier should provide high differential gain with a high common mode rejection ratio (CMRR). The amplifier should also have a high input impedance to avoid loading the sensor.

The classic three op-amp instrumentation amplifier topology shown in Figure 4 has these properties and is a good choice to amplify the output of the pressure sensor.

Assuming the third op-amp is configured for unity gain as shown in Figure 4, the gain of the instrumentation amplifier is determined by resistors R_F and R_G and is given by:

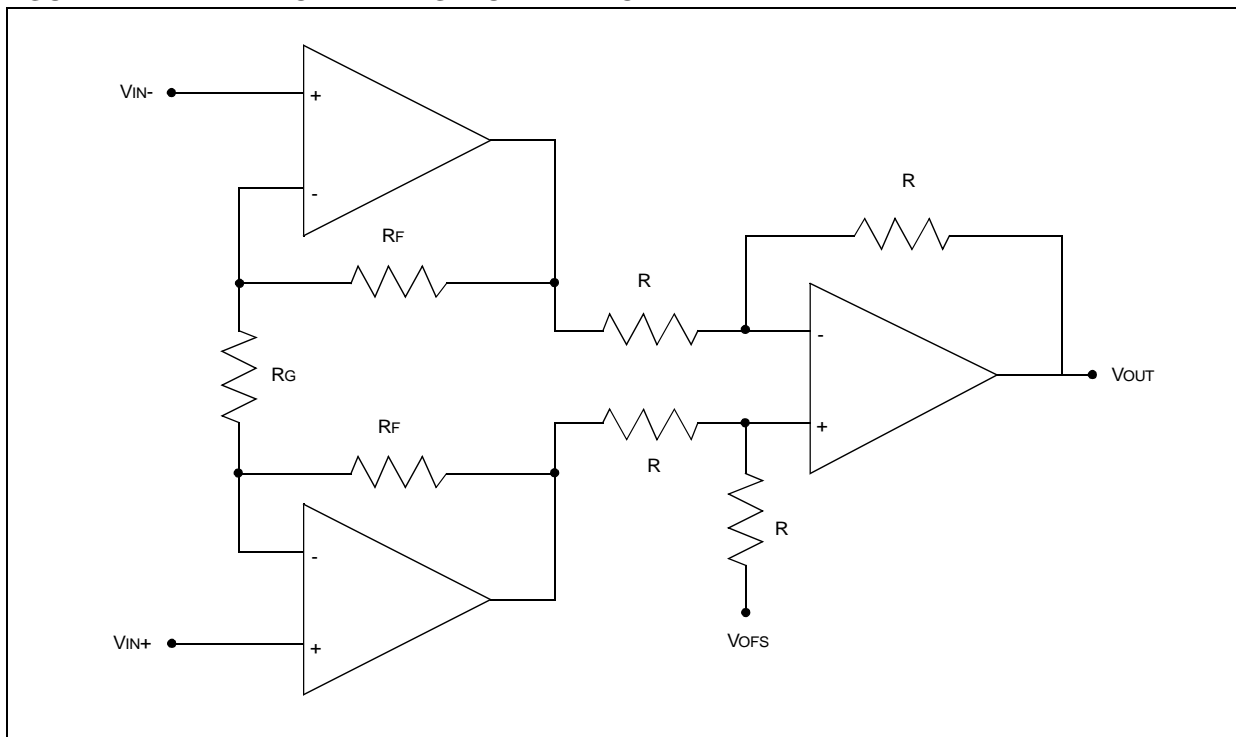
EQUATION 4: AMPLIFIER GAIN

$$A = 1 + 2 \cdot \frac{R_F}{R_G}$$

To allow bipolar measurements, an offset voltage can be connected at the non-inverting input of the third op-amp. This is especially useful in single-supply designs.

Many semiconductor manufacturers offer complete instrumentation amplifiers in a single IC package with the topology shown in Figure 4. These devices offer the advantages of reduced parts count and higher performance due to precise component matching. For these devices, the user typically only needs to provide the external gain resistor to complete the circuit. Depending on the application, an instrumentation amplifier constructed of individual op-amps may still be desirable because of reduced parts cost.

FIGURE 4: THREE OP-AMP INSTRUMENTATION AMPLIFIER



PCB LAYOUT

The hardware for the sensor application must be implemented so it is possible to get 12 noise-free bits of measurement resolution. Since the application PCB must carry both digital and analog signals, special considerations must be made to reduce the effects of noise on the A/D conversion results. High-frequency switching noise generated by digital circuits will easily find its way into the analog signal conditioning circuitry, corrupting the measurement results. A well designed PCB should minimize the effects of conducted noise and radiated noise.

Conducted paths allow noise to propagate into sensitive areas of the circuit through PCB traces and circuit elements. Conducted noise paths can be controlled by using proper decoupling and bypassing techniques. To control conducted noise, the designer should ensure that noise currents are given the lowest possible impedance along the desired route back to the power supply.

In contrast, a radiated noise path is produced when noise is coupled into unwanted circuit areas by some airborne means. These airborne paths are produced by stray capacitances and resistances formed by the physical orientation of circuit elements and PCB traces.

A good power supply is essential to minimize noise in the analog circuits. The power for the application should be provided by a linear supply. Although a switching power supply has obvious benefits, the switching noise present on the output negates the advantages. Central ground and power nodes should be established near the power supply on the PCB.

A ground plane is essential for noise reduction in the analog signal conditioning circuit, because signals are referenced to this ground. The ground plane has two purposes. First, the ground plane gives the lowest impedance possible back to the central ground point for return currents. Without the ground plane, it is easy for common mode noise voltages to be developed due to the series resistance and inductance in the ground circuit traces. Secondly, the ground plane provides shielding for sensitive circuits and PCB traces.

The analog ground plane should be separated from the digital ground plane, if one is present, and the two ground planes should only connect at the power supply. If a two-layer PCB construction is used for cost savings, one side of the PCB can be dedicated to a ground plane. The ground plane should encompass the PCB areas that contain the analog signal conditioning circuits and should have minimal interruptions due to signal traces.

If a digital ground plane is not implemented, a 'star' topology should be used to connect individual IC's to the central ground. Care should be taken not to connect the grounds between individual IC's, which could

form a ground loop. The digital ground traces should be two to three times the width of signal traces to minimize series resistance and inductance.

A power plane is not essential, particularly in applications that require 12 bits of accuracy or less. However, special precautions do need to be taken. First, power traces should be two to three times the width of signal traces and a 'star' connection topology should be implemented. Second, proper power supply decoupling techniques should be used. Separate analog and digital supply busses should be established on the PCB. These two busses should only connect at the power supply. The analog power supply bus is decoupled from the main supply using a series 10 Ω resistor and two shunt capacitors. This decoupling circuit ensures that noise currents induced on the digital supply bus will not be conducted into the analog supply.

Decoupling capacitors should be installed near the power pin of all IC's on the PCB. Two capacitors should be used at each location — a larger electrolytic capacitor and a smaller ceramic capacitor. Typical application values for these capacitors are 10 μ F and 0.1 μ F, respectively. The smaller capacitor is installed closest to the power supply pin and provides effective bypassing at higher frequencies. The larger electrolytic capacitor is used for local energy storage.

Physical distance is one of the best methods for reducing the effects of radiated noise in a circuit. Consequently, the analog circuits should be located away from the MCU and other digital circuits on the PCB for this application. The designer should also check the layout to verify the orientation of sensitive analog signal traces. In general, these traces should be kept as short as possible. Long runs of analog signal traces parallel to digital signal traces should be avoided. Stray capacitance that is a function of trace width and physical separation of the traces will couple digital signals into the analog signal path.

HARDWARE

A schematic of the complete pressure measurement circuit has been included in Appendix B. Separate analog and digital power supply busses have been established in the circuit. The PIC16C774 has separate analog and digital supply pins that have been connected to the appropriate supply bus. The PIC16C774 is operated at 4 MHz using a crystal. A 16 x 2 character LCD module is connected to PORTD of the MCU. I/O pin RE0 is used to control the LED backlight on the LCD module. Two pushbuttons are connected to pins RB4 and RB5 for data entry. A serial EEPROM is connected to the MSSP module for storage of the calibration values.

The pressure sensor includes an internal resistor, R_G, used as the gain setting resistor of the instrumentation amplifier. The purpose of the resistor is to normalize the

full-scale output of the sensor/instrumentation amp combination, so the same sensitivity may be maintained across a range of sensors.

Op-amp U3A (MCP602) is configured as a unity-gain buffer for the 4.096 voltage reference output used as the excitation source for the pressure sensor. The voltage reference output is decoupled from the input of the op-amp using a resistor and two capacitors.

A constant voltage source is used to excite the sensor in order to simplify the design. Because a voltage source is used instead of a current source as recommended by the manufacturer, the internal gain compensation provided by the sensor is lost. However, the benefits of internal offset compensation are still achieved. The internal offset calibration is important because the output offset of an uncompensated sensor can easily be equal in magnitude to the total output span. Sensor output offset can easily be corrected in software, but without external compensation resistors large sensor output offsets will reduce the total measurement range by lowering the available headroom in the amplifier stages.

Op-amps U2A, U2B and U3B (MCP602) form the instrumentation amplifier. The internal gain resistor, R_G , is used in the feedback circuit to set the gain. Feedback resistors R_2 and R_3 are set to 100 k Ω . Resistor R_4 is not used because of the internal sensor resistor, but may be used if another type of sensor is installed. Based on the specifications for the pressure sensor, the output of U3 is approximately 2 volts for a +50 psi input.

Jumper J1 allows a 2.048 volt offset to be applied to the output of the instrumentation amplifier, if desired. The offset voltage is generated by the RA2/VRL output of the PIC16C774. The offset voltage biases the quiescent output of the instrumentation amplifier to the center of the A/D scale, which permits negative pressure (vacuum) measurements.

Resistor R_8 and capacitor C_5 form a single order low-pass filter. The purpose of this filter is to remove high-frequency noise generated in the sensor amplifier circuit. Without the filter, this noise will be aliased into the measurement results.

Temperature sensor U4 is included in the circuit for the purpose of offset and gain compensation if an uncompensated sensor is used in the design. The temperature sensor produces a voltage output of 10 mV/ $^{\circ}$ C. Since the voltage reference for the A/D converter is 4.096 volts, a resolution of 1 mV/bit is obtained. Therefore, each LSB represents 0.1 $^{\circ}$ C in the conversion result.

SOFTWARE

The software for this application was written in C for the Hi-Tech PICC compiler. The compiled code uses approximately 1800 words of program memory. The routines for reading and writing the EEPROM and writing the LCD display are included in separate files and linked to the final project. A complete listing of the source code is provided in Appendix A.

Button entry is handled in the main program loop. The design makes use of the PORTB interrupt-on-change feature to detect when a button has been pressed. When a keypress is detected, the value of PORTB is stored in a temporary variable, `RBTemp`. A short delay is invoked for button debouncing and then PORTB is read again. If the debounce check is OK, the PORTB value is checked to see what button has been pressed. The action taken depends on the calibration mode of the software.

Timer1 is set up to overflow at 10 millisecond intervals and is used to time the A/D conversions and display updates. An interrupt service routine (ISR) is used to handle the Timer1 overflows. The ISR reloads Timer1, clears the interrupt flag, and sets `DispFlag = 1`, which tells the main program loop to do an A/D conversion and update the display.

The software turns on both on-chip voltage references and enables their output by writing to the REFCON register. When the reference outputs are enabled, the function of the RA2(VRL) or RA3(VRH) pins is overridden and the pin becomes a voltage reference output.

A/D conversions are performed with the MCU in SLEEP mode to minimize the effects of noise on the conversion. The A/D converter must be configured to use its own internal RC oscillator to perform conversions in SLEEP. When using the RC oscillator, the A/D converter waits one instruction cycle before the conversion begins. This allows the time needed to execute the SLEEP instruction. Global interrupts are disabled before starting the conversion. When the ADIE bit is set and global interrupts are disabled, the MCU will wake up when the conversion is complete and continue execution at the next instruction. The ADIF flag is cleared before global interrupts are reenabled, so an unexpected interrupt will not be generated.

A circular buffer is maintained in RAM and is used to calculate a running average of the last 32 conversion results. After each conversion, the contents of the buffer are summed and shifted to the right by one bit, producing a 16-bit integer result. The 16-bit offset calibration value is added to this result and multiplied by the 16-bit gain calibration value. The calibrated pressure is contained in the upper 16 bits of the multiplication result. This value is converted to a formatted ASCII string using the `prestoa()` function and sent to the LCD display.

The software has two calibration modes for performing gain and offset corrections. If one of the calibration modes is active (`CalMode = 1` or `CalMode = 2`), an indicator is written to the LCD module to inform the user.

When the MCU is RESET, the calibration values stored in the EEPROM are retrieved. After power-up, different calibration modes may be invoked using the MCLR button. If the RB4 button is depressed and a MCLR Reset is performed, the offset calibration mode is entered. If the RB5 button is depressed and a MCLR Reset is performed, the gain calibration mode is entered. An "OF" or "GN" indicator is placed at the right side of the LCD display to indicate that one of the calibration modes is active. In both modes, the user can raise or lower the calibration value using the RB4 and RB5 buttons. The calibration values can be lowered or raised in small increments by repeatedly pressing the RB4 or RB5 buttons, respectively. If either button is held continuously for a period of time, the calibration value will begin to change rapidly. Depending on the calibration mode, the adjusted gain or offset value is stored in the EEPROM by pressing the RB4 and RB5 buttons simultaneously. The calibration indicator at the right side of the LCD display is turned off to indicate that the calibration value has been stored and the program has returned to normal operating mode. Resetting the MCU without pressing RB4 or RB5 will exit any active calibration mode and return to normal operation without saving the calibration value. When the MCU is not in either of the calibration modes, pressing the RB4 button will toggle the LCD backlight on or off.

CALIBRATION

The pressure sensor is calibrated by adjusting the gain and offset values. The offset calibration is adjusted with no input to the sensor and should be adjusted so the display indicates 0.00 psi. The gain calibration is adjusted with a known maximum input applied to the sensor and should be adjusted using a stable pressure reference source. The gain calibration is adjusted so the display indicates the known value of the reference.

REFERENCES FOR FURTHER READING

Lucas Novasensor Website:

www.novasensor.com

Microchip Technology Inc.

- AN682 – "Using Single Supply Operational Amplifiers in Embedded Systems"
- AN688 – "Layout Tips for 12-bit A/D Converter Applications"
- AN699 – "Anti-aliasing, Analog Filters for Data Acquisition Systems"

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") for its PICmicro® Microcontroller is intended and supplied to you, the Company's customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: SOURCE CODE

```

//*****
/** CPRES.C *
//*****
/**
/** Written by: Stephen Bowling *
/** Sr. Applications Engr. *
/** Microchip Technology Inc. *
/** Date: 6 October 1999 *
/** Revision: 1.03 *
//*****
/**
/** This program demonstrates a ratiometric pressure measurement *
/** using the PIC16C774. Offset and gain calibration values are *
/** stored in EEPROM memory. *
//*****

#include <pic.h>
#include<stdio.h>
#include "16lcd.h" // Contains LCD functions.
#include "16i2c.h" // Contains I2C functions.

char i,
      CalMode,
      RBTemp,
      DispFlag,
      ButtonDly;

char data[8];

unsigned int TIMER1 @ &TMR1L;

unionINTVAL // Union to handle 16-bit values
{ // as 1 integer or two bytes.
unsigned int ui;
int i;
char b[2];
};

unionLNGVAL // Union to handle 32-bit values
{ // as 1 long, 2 integers, or
// 4 bytes.
long l;
int i[2];
char b[4];
};

union INTVAL Gain, Offset;
union LNGVAL Pressure, TmpPressure;

bank1 unsigned char ADPtr;

```



```

bank1 int ADTable[32];

void interrupt isr_handler(void);           // Does measurement timing.
int ConvADC(void);
void DisplayBanner(void);
void prestoa(int value, char *string);

void main(void)
{
  InitLCD();                               // Initialize LCD display.
  SSPADD = 9;                               // Setup MSSP for master I2C
  SSPSTAT = 0;                              // "
  SSPCON2 = 0;                              // "
  SSPCON = 0x28;                            // "
  ADCON1 = 0xCD;                            // Setup A/D converter
  ADCON0 = 0xc1;                            //
  REFCON = 0xF0;                            // Setup VREFs

  RBPU = 0;                                 // Setup PORTB I/O
  PORTB = PORTB;

  PORTC = 0;                                // Setup PORTC I/O
  TRISC = 0xdf;

  PORTE = 0;                                // Setup PORTE I/O
  TRISE = 0x06;                            // RE0 controls LED backlight.
  // 1 = off, 0 = on

  CalMode = 0;                              // Variable indicates offset or gain calibration
  DispFlag = 0;                             // Flag tells main loop to do A/D conversion
  ButtonDly = 0;                            // Stores time button has been pressed
  ADPtr = 0;                                // Pointer to A/D conversion result buffer

  Offset.b[0] = EERandomRead(0xA0, 0);
  Offset.b[1] = EERandomRead(0xA0, 1);
  Gain.b[0] = EERandomRead(0xA0, 2);         // Get calibration values from
  Gain.b[1] = EERandomRead(0xA0, 3);         // EEPROM

  TMR1H = 0xd8;                             // Load Timer1 overflow value
  TMR1L = 0xf0;
  TMR1IF = 0;                                // Clear Timer1 interrupt flag
  TMR1IE = 1;                                // Enable Timer1 interrupts
  T1CON = 1;                                  // Turn on Timer1
  ADIF = 0;                                  // Clear A/D interrupt flag
  ADIE = 1;                                  // Enable A/D interrupts
  PEIE = 1;                                  // Enable peripheral interrupts
  GIE = 1;                                    // Enable all interrupts

  if(!POR)                                   // If this was a Power-on Reset:
  {
    POR = 1;                                 // Reset bit
    DisplayBanner();                         // Display intro message
  }
  else
  {
    // If this wasn't a Power-on Reset:
    if(!RB4 && !RB5)                         // Both buttons pressed: restore default
    {                                         // gain and offset values
      Offset.i = -32371;
      Gain.i = 8323;
      EEAckPolling(0xA0);
      EEByteWrite(0xA0, 0, Offset.b[0]);
      EEAckPolling(0xA0);
      EEByteWrite(0xA0, 1, Offset.b[1]);
      EEAckPolling(0xA0);
      EEAckPolling(0xA0);
      EEByteWrite(0xA0, 2, Gain.b[0]);
    }
  }
}

```

AN694

```
EEAckPolling(0xA0);
EEByteWrite(0xA0, 3, Gain.b[1]);
EEAckPolling(0xA0);

}
else // RB4: enter offset calibration mode
if(!RB4) CalMode = 1;
else
if(!RB5) CalMode = 2; // RB5: enter gain calibration mode
else;
}

while(1)
{
if(RBIF)
{
RBTemp = PORTB & 0xf0; // Get PORTB values.
for(i=0;i<20;i++); // Delay a little for debounce.

if(!(RBTemp^(PORTB & 0xf0))) // Process input if debounce is OK.
{
RBTemp ^= 0xf0;
if(!RBTemp) ButtonDly = 0; // Clear button delay if release
else // is detected.
if(RBTemp == 0x10) // If RB4 button was pressed.
{
if(CalMode == 0) // If not in calibration mode:
{
if(RE0) RE0 = 0; // Turn on LCD backlight.
else RE0 = 1; // Turn off LCD backlight.
}
if(CalMode == 1) // If in offset calibration mode.
Offset.i += 4; // Increment offset value.

else
if(CalMode == 2) // Increment gain value.
Gain.i += 2; // If in gain calibration mode
}
else
if(RBTemp == 0x20) // If RB5 button was pressed.
{
if(CalMode == 1) // If in offset calibration mode.
Offset.i -= 4; // Decrement offset value.

else
if(CalMode == 2) // If in gain calibration mode.
Gain.i -= 2; // Decrement gain value.
}
else
if(RBTemp == 0x30) // If both buttons pressed:
{
if(CalMode == 1) // Write new offset to EEPROM.
{
EEAckPolling(0xA0);
EEByteWrite(0xA0, 0, Offset.b[0]);
EEAckPolling(0xA0);
EEByteWrite(0xA0, 1, Offset.b[1]);
EEAckPolling(0xA0);
// Read back values for error checking.
Offset.b[0] = EERandomRead(0xA0, 0);
Offset.b[1] = EERandomRead(0xA0, 1);
CalMode = 0;
}
}
}
}
}
```

```

else
    if(CalMode == 2)                // Write new gain to EEPROM
    {
        EEAckPolling(0xA0);
        EEByteWrite(0xA0, 2, Gain.b[0]);
        EEAckPolling(0xA0);
        EEByteWrite(0xA0, 3, Gain.b[1]);
        EEAckPolling(0xA0);
        // Read back values for error checking.
        Gain.b[0] = EERandomRead(0xA0, 2);
        Gain.b[1] = EERandomRead(0xA0, 3);
        CalMode = 0;
    }
}

RBIF = 0;                          // Clear interrupt flag.
}

if(DispFlag)
{
    if(RBTemp)
    {
        if(ButtonDly < 255) ButtonDly++; // Delay value is cleared whenever
        else // a button is released. This makes
        if(RBTemp == 0x10) // data value increment rapidly
        { // after button is held for a while.
            if(CalMode == 1)
                Offset.i += 4;

            else
            if(CalMode == 2) // Increment gain value.
                Gain.i += 2; // Set flag for EEPROM write in main
            }
        else
        if(RBTemp == 0x20) // If RB5 button still pressed.
        {
            if(CalMode == 1)
                Offset.i -= 4; // Decrement offset value.

            else
            if(CalMode == 2)
                Gain.i -= 2; // Decrement gain value.
            }
    }

    ADPtr++; // Increment pointer to result buffer
    if(ADPtr == 32) ADPtr = 0; // If at the end of the buffer, set to 0
    ADTable[ADPtr] = ConvADC(); // Do a conversion and store in buffer

    TmpPressure.l = 0;
    for(i = 0; i < 32; i++) // Average last 32 A/D conversions stored
        TmpPressure.l += (long)ADTable[i]; // in the buffer
    TmpPressure.l >>= 1; // Shift by one to get 16 bit result
    TmpPressure.i[0] += Offset.i; // Add offset to result.
    Pressure.l =
    (long)TmpPressure.i[0] * (long)Gain.i; // Compute gain.

    prestoa(Pressure.i[1], data); // Convert pressure
    SendCmd(0x80); // reading to ASCII string.
    putsLCD(data);
    putsLCD(" PSI");
}

```

AN694

```
SendCmd(0x8e);
if(CalMode == 0)
{
    putsLCD(" ");           // Put calibration indicator
                           // on LCD if depending on mode.
}
else
if(CalMode == 1)
{
    putsLCD("OF");
}

else
if(CalMode == 2)
{
    putsLCD("GN");
}
else;

DispFlag = 0;           // Clear display flag.
}
}

//-----
// Timer1 times the A/D conversions and display updates.
// A flag is set to signal the main program loop.
//-----

void interrupt isr_handler(void)    // Interrupt service routine
{
if(TMR1IF)           // Timer1 set to overflow
{
    DispFlag = 1;           // every 10 msec
    TIMER1 += 55536;       // Signal main loop to display
    TMR1IF = 0;
}
}

//-----
// This routine does the following steps for an A/D conversion:
//
// -global interrupts are disabled
// -conversion started
// -MCU is put to SLEEP--will wake when conversion is complete
// -Operation continues at next instruction when MCU wakes
//   since global interrupts are disabled.
// -clear A/D interrupt flag and reenables global interrupts
//-----

int ConvADC(void)
{
union INTVAL ADCRes;

GIE = 0;           // Disable global interrupts.
ADGO = 1;         // Start conversion.

#asm
sleep
nop
#endasm
```

```

ADIF = 0; // Clear A/D interrupt flag.
GIE = 1; // Enable global interrupts.

ADCRes.b[0] = ADRESL;
ADCRes.b[1] = ADRESH;

return ADCRes.i; // Return 12-bit result.
}

//-----
// This routine displays intro message and also displays calibration
// values stored in the EEPROM
//-----

void DisplayBanner(void)
{
putsLCD("Microchip"); // Display intro message.
SendCmd(0xa8);
putsLCD("Technology");
i = 0;
while(i < 255) // Pause for a little while
{
if(DispFlag)
{
i++;
DispFlag = 0;
}
}
SendCmd(0x80);
putsLCD("PIC16C774"); // Display next message.
SendCmd(0xa8);
putsLCD("PSI Monitor 1.03");
i = 0;
while(i < 255) // Pause for a little while
{
if(DispFlag)
{
i++;
DispFlag = 0;
}
}
SendCmd(0x80); // Display offset and gain values
presto(Offset.i, data);
putsLCD(data);
putsLCD(" ");
presto(Gain.i, data);
SendCmd(0xa8);
putsLCD(data);
putsLCD(" ");
i = 0;
while(i < 255) // Pause for a little while
{
if(DispFlag)
{
i++;
DispFlag = 0;
}
}
clrLCD();
i = 0;
}

```

AN694

```
//-----  
// Converts pressure to an ASCII string with the format '+xxx.xx'  
//-----  
  
void preston(int value, char *string)  
{  
    unsigned char flag;                //  
                                        // Leading '0's are removed.  
  
    flag = 0;  
    if( value < 0 )                    // If negative  
    {                                   // 2's complement the number  
        value = ~value;  
        value++;  
        *string = '-';                 // Store a minus sign in 1st character  
    }  
    else *string = '+';  
    string++;                           // Increment string pointer  
  
    *string = 0x30;                     // Start with ascii '0'  
    while(value > 9999)                 // Check to see how many 10000s in number  
    {  
        value -= 10000;                // Subtract 10000 from number  
        *string += 1;                  // Increment the 10000s character in  
                                        // the string  
        flag++;  
    }  
    if(!flag)  
        *string = ' ';                 // Remove leading '0'  
        string++;                       // Increment string pointer  
  
    *string = 0x30;                     // Start with ascii '0'  
    while(value > 999)                 // Check to see how many 1000s in number  
    {  
        value -= 1000;                 // Subtract 1000 from number  
        *string += 1;                  // Increment the 1000s character in  
                                        // the string  
        flag++;  
    }  
    if(!flag)  
        *string = ' ';                 // Remove leading '0'  
        string++;                       // Increment the string pointer  
  
    *string = 0x30;                     // Start with ascii '0'  
    while(value > 99)                 // Check to see how many 100s in number  
    {  
        value -= 100;                  // Subtract 100 from number  
        *string += 1;                  // Increment the 100s character in  
                                        // the string  
        flag++;  
    }  
  
    string++;                           // Increment the string pointer  
    *string = '.';                       // Add in the decimal place  
    string++;                           // Increment the string pointer  
  
    *string = 0x30;                     // Start with ascii '0'  
    while(value > 0x09)               // Check to see how many 10s in number  
    {  
        value -= 10;                   // Subtract 10 from number  
        *string += 1;                  // Increment the 10s character in  
                                        // the string  
        flag++;  
    }  
    string++;                           // Increment the string pointer  
}
```

```
*string = 0x30;           // Start with ascii '0'  
*string +=  
(char)(value&0x00ff);   // Add the remainder to the number  
string++;                // Increment the string pointer  
*string = 0;             // Add the null character  
}
```

AN694

```
//*****
//* 16lcd.c *
//*****
//
// This file contains the functions necessary to communicate with *
// a Hitachi compatible LCD display. *
// The functions are written for the HiTech PICC compiler. *
// The display is used in 4-bit mode and 6 I/O lines are used *
// for communication. *
// To save I/O lines, these functions do not check the display's *
// busy flag. The R/W line on the display is tied low. *
//*****

#include <pic.h>
#include "16lcd.h"

// Defines for I/O ports that provide LCD data & control
// The lower 4 bits of the port are used for data lines

#defineLCD_DATAPORTD
#defineLCD_CNTLTRISD

// Defines for I/O pins that provide LCD control

volatile bit LCD_RS @ (unsigned)&PORTD*8 +5;
volatile bit LCD_E @ (unsigned)&PORTD*8 +4;

void InitLCD(void)
{
LCD_DATA = 0;
LCD_CNTL = 0xC0;

SendCmd(0x2c);
LongDelay();
SendCmd(0x2c);
LongDelay();
SendCmd(0x2c);
LongDelay();
SendCmd(0x0C);
SendCmd(0x06);
SendCmd(0x80);

clrLCD();
}

//*****
//*putcLCD() - Sends character to LCD *
//*This routine splits the character into the upper and lower *
//*nibbles and sends them to the LCD, upper nibble first. *
//*****

void putcLCD(char lcdbyte)
{
LCD_DATA = lcdbyte >> 4;
LCD_RS = 1;
LCD_E= 1;
LCD_E = 0;
LCD_DATA = lcdbyte &= 0x0f;
LCD_RS = 1;
LCD_E= 1;
LCD_E = 0;
Delay();
}
}
```



```

/*****
/** SendCmd() - Sends command to LCD
/** This routine splits the command into the upper and lower
/** nibbles and sends them to the LCD, upper nibble first.
*****/

void SendCmd(char lcdbyte)
{
LCD_DATA = lcdbyte >> 4;
LCD_E= 1;
LCD_E = 0;
LCD_DATA = lcdbyte &= 0x0f;
LCD_E= 1;
LCD_E = 0;
Delay();
}

/*****
/** clrLCD - Clear the contents of the LCD
*****/

void clrLCD(void)
{
SendCmd(0x01);
}

/*****
* Function Name: putsLCD
* Return Value: void
* Parameters: buffer: pointer to string
* Description: This routine writes a string of bytes to the
* Hitachi HD44780 LCD controller.
*****/

void putsLCD(const char *buffer)
{
do
{
putcLCD(*buffer); // Write character to LCD
buffer++;
}while(*buffer);
return;
}

/*****
/** Delay(), LongDelay() - Generic LCD delays
/** Since the microcontroller can not read the busy flag of the
/** LCD, a specific delay needs to be executed between writes to
/** the LCD.
*****/

void Delay(void)
{
char i;
for(i=0;i < 5;i++);
}

void LongDelay(void)
{
int i;

for(i=0;i < 0x400;i++);
}

```

AN694

```
//*****
//* 16i2c.c *
//*****
//
// This file contains the functions necessary to communicate with *
// a 24C01 serial EEPROM connected to the MSSP module of a 16CXXX *
// device. The functions are written for the HiTech PICC compiler.*
//*****

#include <pic.h>
#include "16i2c.h" // contains the prototypes for the
                  // function calls.

void Nop(void)
{
return;
}

unsigned char EEByteWrite(unsigned char control,
                          unsigned char address, unsigned char data)
{
IdleI2C(); // ensure module is idle
SEN = 1; // initiate START condition
Nop(); // 1Tcy required before test can be made
if (BCLIF) // test for bus collision
{
return (-1); // return with Bus Collision error
}
else // start condition successful
{
IdleI2C(); // ensure module is idle
WriteI2C(control); // write byte
IdleI2C(); // ensure module is idle

if (!AKSTAT) // test for ACK condition received
{
WriteI2C(address); // write byte - word address location
IdleI2C(); // ensure module is idle

if (!AKSTAT) // test for ACK condition received
{
WriteI2C(data); // data byte to be written
IdleI2C(); // ensure module is idle
}
}
else
{
return (-2); // return with Not Ack error condition
}
}

IdleI2C(); // ensure module is idle
PEN = 1; // send STOP condition
Nop(); // 1Tcy required before test can be made
if (BCLIF) // test for bus collision
{
return (-1); // return with BUS Collision error
}
return (0); // return with no error
}
}
```

```

char EERandomRead(unsigned char control, unsigned char address)
{
    IdleI2C(); // ensure module is idle
    SEN = 1; // initiate START condition
    Nop(); // 1Tcy required before test can be made
    if (BCLIF) // test for bus collision
    {
        return (-1); // return with Bus Collision error
    }
    else
    {
        IdleI2C(); // ensure module is idle
        WriteI2C(control); // WRITE control byte - R/W bit should be 0
        IdleI2C(); // ensure module is idle

        if (!AKSTAT) // test for ACK condition received
        {
            WriteI2C(address); // WRITE word address for EEPROM
            IdleI2C(); // ensure module is idle

            if (!AKSTAT) // test for ACK condition received
            {
                RSEN = 1;
                Nop(); // 1Tcy required before test can be made
                if (BCLIF) // test for bus collision
                {
                    return (-1); // return with Bus Collision error
                }

                IdleI2C(); // ensure module is idle
                WriteI2C(control+1); // control byte - R/W bit should be 1 for read
                IdleI2C(); // ensure module is idle

                if (!AKSTAT) // test for ACK condition received
                {
                    ReadI2C(); // initiate read of 1 byte

                    IdleI2C(); // ensure module is idle
                    AKDT = 1; // send ACK condition
                    AKEN = 1;
                    IdleI2C(); // ensure module is idle
                    PEN = 1; // send STOP condition
                    Nop(); // 1Tcy required before test can be made
                    if (BCLIF) // test for bus collision
                    {
                        return (-1); // return with Bus Collision error
                    }
                }
            }
            else
            {
                return (-2); // return with Not Ack error
            }
        }
        else
        {
            return (-2); // return with Not Ack error
        }
    }
    else
    {
        return (-2); // return with Not Ack error
    }
}
return (SSPBUF); // return with data
}

```

AN694

```
char WriteI2C(unsigned char data_out)
{
    SSPBUF = data_out;                // write single byte to SSPBUF
    if (WCOL)                          // test if write collision occurred
        return (-1);                  // if WCOL is set return negative #
    else
    {
        while(STAT_BF);               //
        return (0);                   // if WCOL is not set return positive #
    }
}

char EEAckPolling(unsigned char control)
{
    IdleI2C();                          // ensure module is idle
    SEN = 1;                            // initiate START condition
    Nop();                               // 1Tcy required before test can be made
    if (BCLIF)                           // test for bus collision
    {
        return (-1);                  // return with Bus Collision error
    }
    else
    {
        IdleI2C();                     // ensure module is idle
        WriteI2C(control);              // write byte - R/W bit should be 0
        IdleI2C();                     // ensure module is idle

        while (AKSTAT)                 // test for ACK condition received
        {
            RSEN = 1;                  // initiate Restart condition
            Nop();                       // 1Tcy required before test can be made
            if (BCLIF)                  // test for bus collision
            {
                return (-1);           // return with Bus Collision error
            }
            IdleI2C();                  // ensure module is idle
            WriteI2C(control);          // write byte - R/W bit should be 0
            IdleI2C();                  // ensure module is idle
        };
    }

    PEN = 1;                            // send STOP condition
    Nop();                               // 1Tcy required before test can be made
    if (BCLIF)                           // test for bus collision
    {
        return (-1);                  // return with Bus Collision error
    }
    return (0);                          // return with no error
}

char ReadI2C(void)
{
    RCEN = 1;                            // enable master for 1 byte reception
    while (STAT_BF);                     // wait until byte received
    return(SSPBUF);                       // return with read byte
}

void IdleI2C(void)
{
    while ((SSPCON2 & 0x1F) | STAT_RW)
        continue;
}
```

```
// LCD and EEPROM Function Prototypes -----  
  
void InitLCD(void);  
void putcLCD(char lcdbyte);  
void SendCmd(char lcdbyte);  
void clrLCD(void);  
void putsLCD(const char *buffer);  
void Delay(void);  
void LongDelay(void);  
  
unsigned char EEByteWrite(unsigned char control,  
                          unsigned char address, unsigned char data);  
char EERandomRead(unsigned char control, unsigned char address);  
char EEAckPolling(unsigned char control);  
char WriteI2C(unsigned char data_out);  
char ReadI2C(void);  
void IdleI2C(void);  
void Nop(void);
```

APPENDIX B: SCHEMATICS

FIGURE B-1: PRESSURE MONITOR SCHEMATIC DIAGRAM (PAGE 1 OF 3)

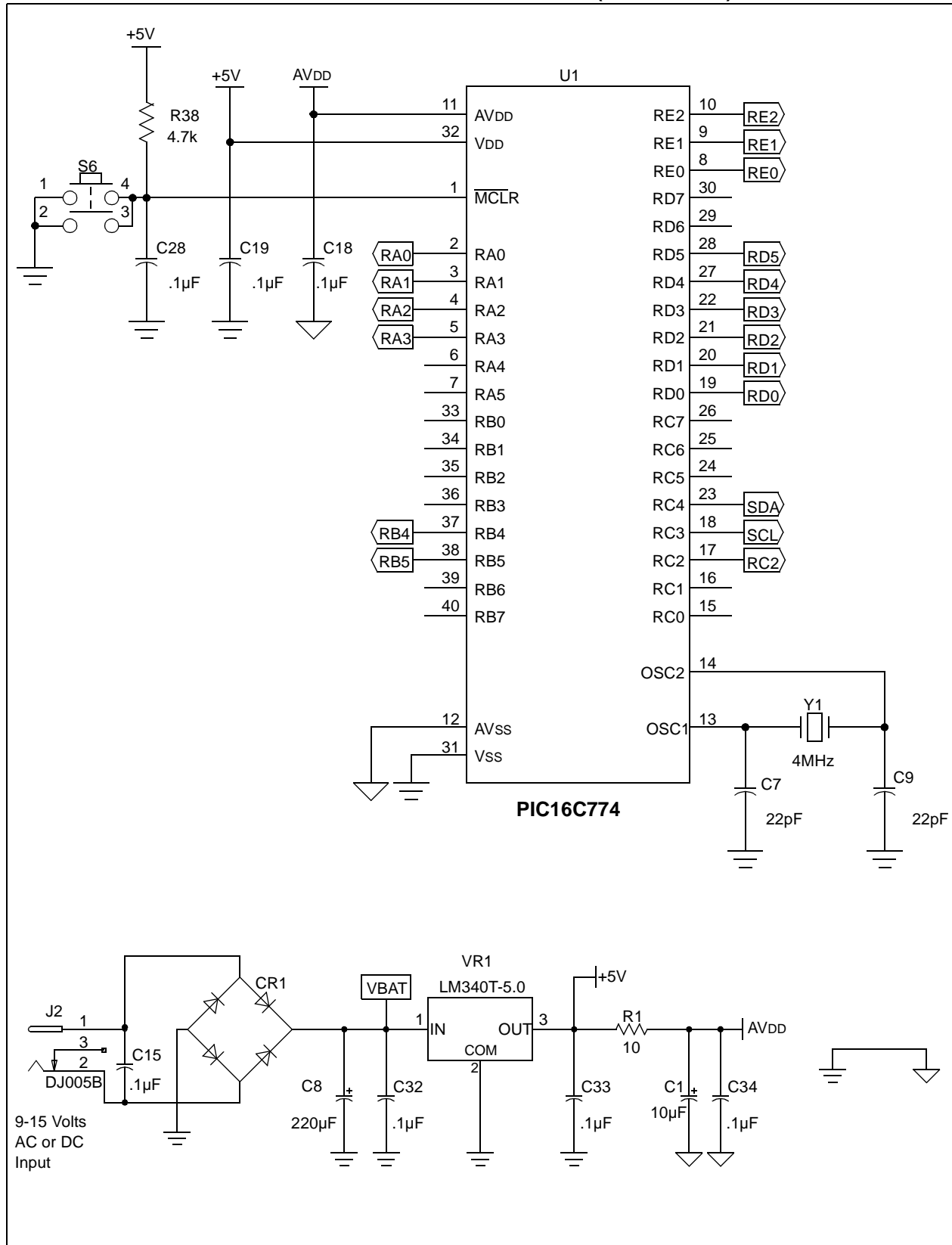


FIGURE B-2: PRESSURE MONITOR SCHEMATIC DIAGRAM (PAGE 2 OF 3)

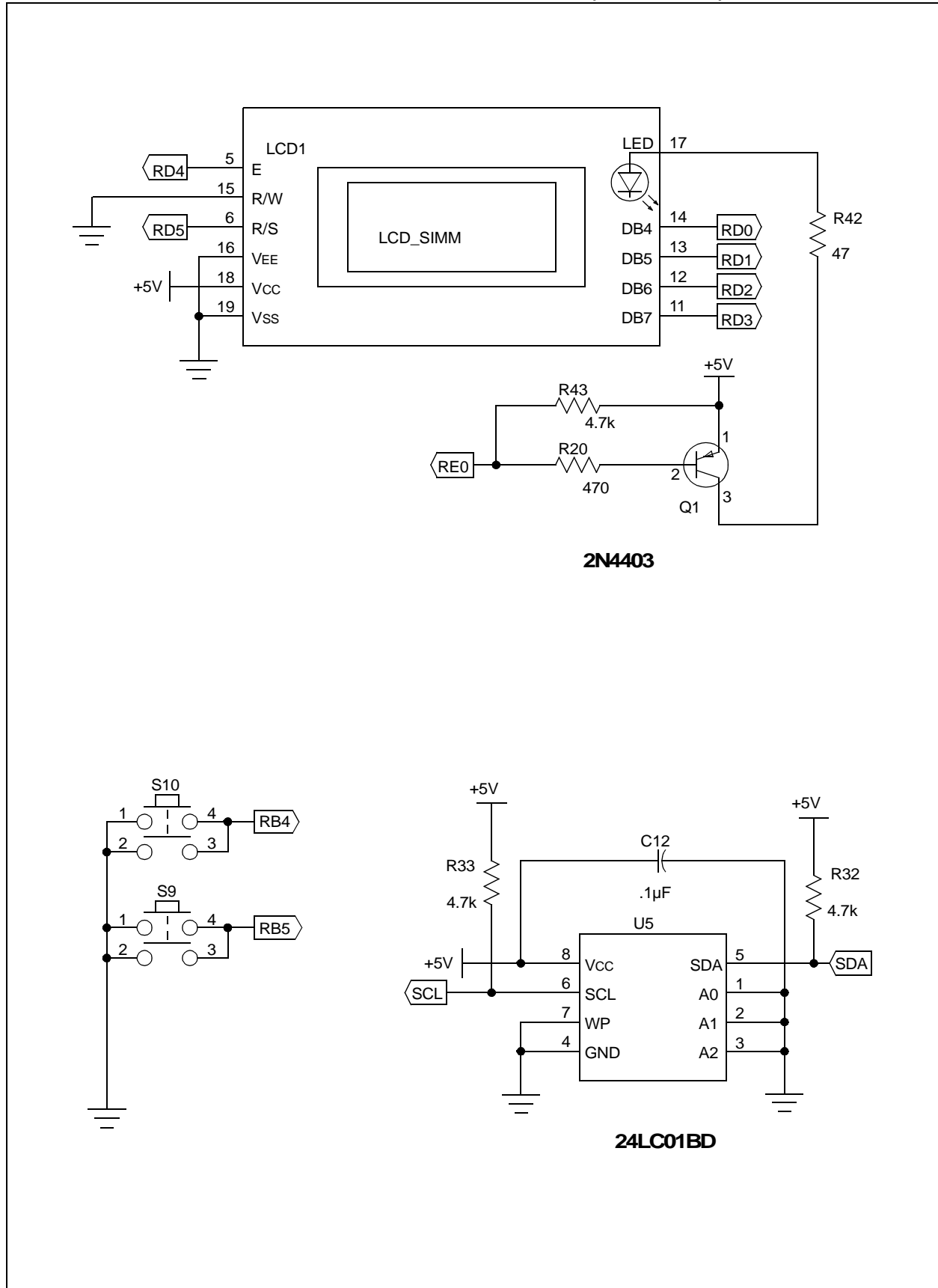
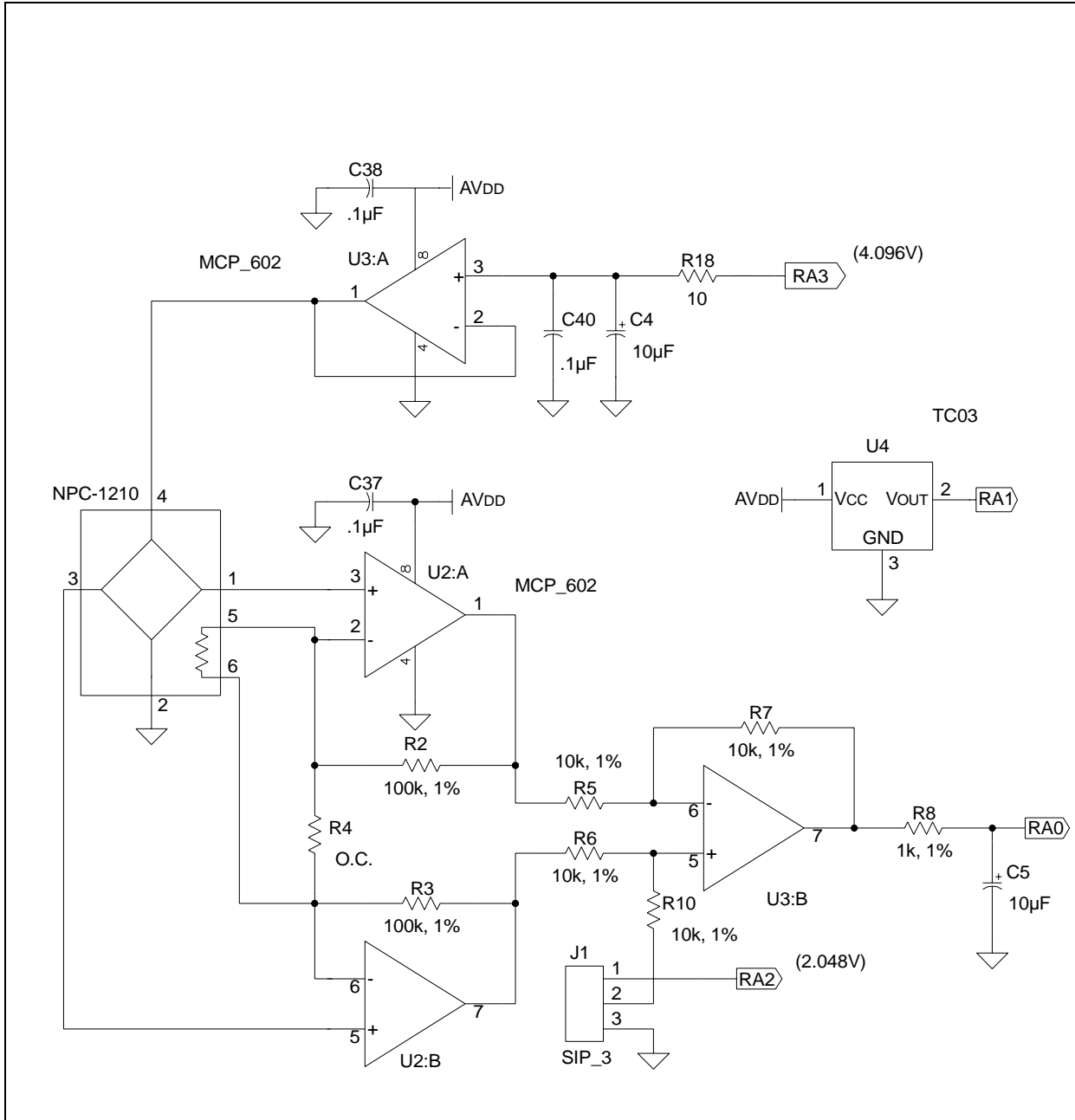


FIGURE B-3: PRESSURE MONITOR SCHEMATIC DIAGRAM (PAGE 3 OF 3)



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.

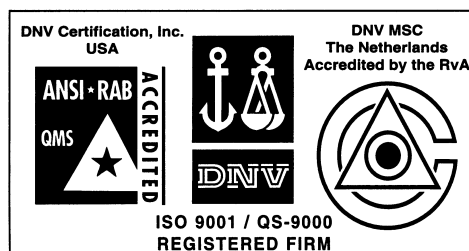
dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

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 - ler 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