AN3538

A Simple Water Monitoring System with I²C Communication

Introduction

Author: Christopher Best, Microchip Technology Inc.

Water monitoring systems are used in a wide variety of applications, from monitoring pressure and flow to checking water quality. As the climate and environment changes, managing the water supply is critical for life on this planet. Water monitoring systems can help prevent waste by watching for leaks and contaminants, or by preventing overuse in farming by tracking water levels in the soil.

A water monitoring system can also be used to maintain the proper water quality, level and temperature of an aquarium. This is vital in aquaponic applications, where the aquarium acts as the water and nutrient supply for growing plants.

This application note will discuss the creation of a simple water monitoring system for a small aquarium. This water monitoring system uses the PIC® microcontrollers' Master Synchronous Serial Port (MSSP) module in I²C mode to create a bus that allows a master device to collect and display the data from the following three sensors:

- pH Sensor
- Temperature Sensor
- Water Level Sensor

Table of Contents

Introduction....1

1. Application Overview....3
   - 1.1. pH Sensor....3
   - 1.2. Temperature Sensor....4
   - 1.3. Water Level Sensor....5

2. Building the System....7
   - 2.1. Part One: The pH Sensor....7
   - 2.2. Part Two: The Temperature and Water Level Sensors....13
   - 2.3. Part Three: The I^2C Master....15

3. Conclusion....17

The Microchip Website....18

Product Change Notification Service....18

Customer Support....18

Microchip Devices Code Protection Feature....18

Legal Notice....18

Trademarks....19

Quality Management System....19

Worldwide Sales and Service....20

1. Application Overview

This application highlights the use of the MSSP in I^2C mode to create a communications bus that will monitor three sensors in an aquarium. The communications bus consists of a single master and three slave devices. Each slave device reads a sensor output, performs any necessary computations and prepares the sensor data for transmission to the master. The master requests the sensor data from each slave and displays the information on a terminal application, such as Tera Term.

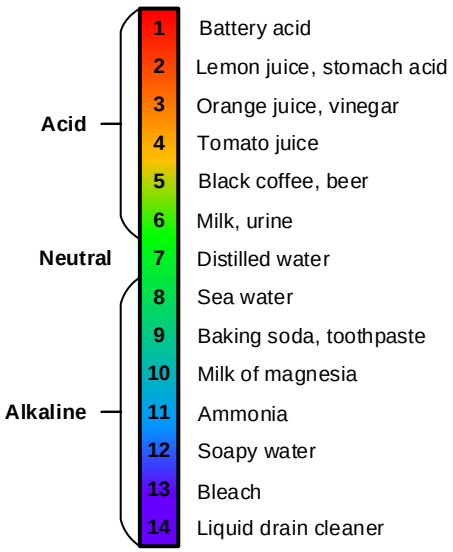
The following components were used in this application:

- Microchip MPLAB® X IDE v5.40
- Microchip XC8 Compiler v2.20
- Curiosity Low Pin Count Development Board (DM164137)
- PIC16F15245 Microcontrollers (3)
- Tera Term (PC terminal application)
- Atlas Scientific Consumer Grade pH Probe (ENV-30-pH))
- Atlas Scientific Gravity™ Analog pH Sensor/Meter (GRV-pH)
- DROK 10k-Ω B3950 NTC Temperature Sensor Probe (B01MR37GOQ)
- 10k-Ω through-hole resister 1%
- Gikfun M8 32mm Liquid Level Sensor (EK1373x3)

1.1 pH Sensor

Since the aquarium will house live fish, the quality of the water is extremely important. The Atlas Scientific Consumer Grade pH (potential of hydrogen) probe is a sensor that measures the hydrogen ion activity in a liquid. The pH scale (ranging from 1-14) is used to determine the acidity of a substance. When a substance has a pH value of “1”, the substance is considered highly acidic and has the highest amount of hydrogen ions. A pH value of “7” is considered neutral, while a value of “14” is considered highly alkaline and has fewer hydrogen ions. Figure 1-1 shows common substances and their associated pH values.

Figure 1-1. pH Scale

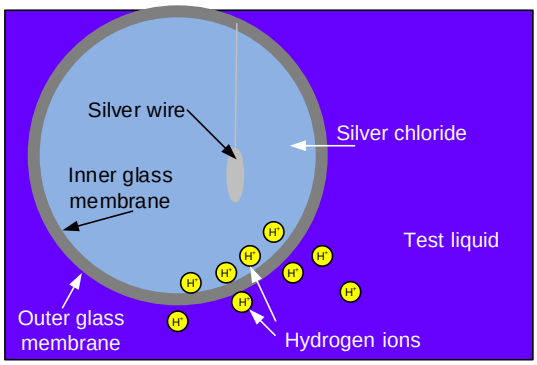


When water has a pH value of “1”, there are one million times more hydrogen ions than if the water had a neutral value of “7”, which is considered “pure” water or water that does not contain dissolved substances. This is due to the

fact that a pH value of “1” represents 10^1 hydrogen ions, while the neutral value of “7” represents 10^7 hydrogen ions. The difference between a pH of “1” and a pH of “7” is 10^6, or one million. A pH value of “1” will have ten trillion (10^{13}) times more hydrogen ions than a substance with a pH value of “14”.

The Atlas Scientific probe uses a silver chloride electrode and a reference wire to detect changes in ion activity. The electrode consists of a small, glass membrane constructed of silica glass containing metal salts. Inside of the glass membrane is a silver chloride solution and a silver wire. The reference wire is suspended in a neutral solution. Figure 1-2 shows the view of the glass electrode suspended in a test liquid.

Figure 1-2. pH Electrode



When the probe is placed into a test liquid, an ion exchange occurs between the metal salts on the inside of the glass membrane and the chloride solution. This creates a very small current in the chloride solution. Simultaneously, the test liquid also exchanges ions with the outer surface of the electrode, creating a current in the test liquid. The output of the probe represents the potential difference as an analog voltage signal.

The output of the probe is connected to the Atlas Scientific Gravity pH meter, which has hardware built in to provide any necessary analog signal conditioning, such as signal amplification. This allows the probe’s output voltage to be large enough to be read by an ADC.

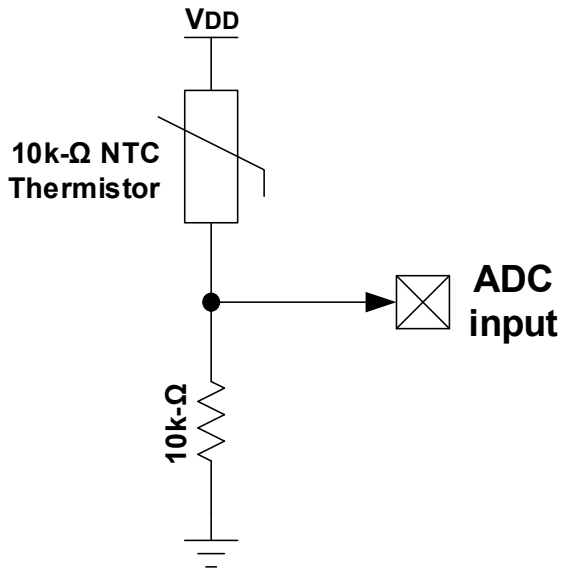
Important: The analog output voltage of the probe is too small of a signal to detect with a voltmeter or through an ADC. Proper signal conditioning is required to obtain accurate results.

1.2 Temperature Sensor

A temperature sensor is needed to monitor the aquarium's water temperature. The DROK temperature sensor is ideal for aquarium application because the sensor is sealed in a stainless steel sleeve, allowing it to be safely submersed into the water.

The temperature sensor is configured as part of a voltage divider circuit (see Figure 1-3), and the output of the divider circuit is fed into an ADC input. In terms of ADC voltage, as the temperature increases, the temperature sensor experiences less resistance and allows a higher voltage to reach the pin.

Figure 1-3. Thermistor Circuit

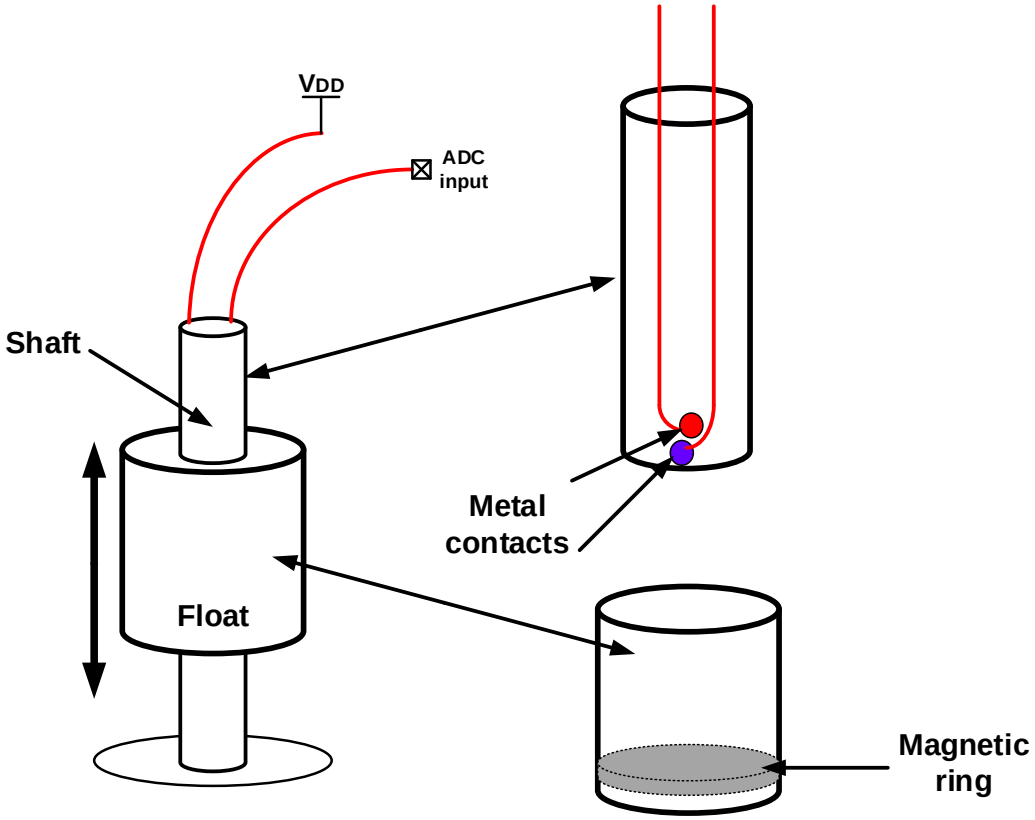


1.3 Water Level Sensor

Evaporation or leaks will cause a drop in the water level of an aquarium. If the water level gets too low, the water's pH level will change due to increased concentration of dissolved substances contained in the water. A liquid level sensor can be used to determine the water level in the aquarium.

The Gikfun Liquid Level Sensor is a simple reed switch. The main body of the switch is a thin shaft that contains two metal contacts that are normally "open", meaning the two do not make physical contact (see Figure 1-4). Surrounding the thin shaft is a float, which houses a thin magnetic ring at one end. When the magnetic ring is very close to the two metal contacts, the magnetic force pushes the two contacts together and creates a closed circuit. When placed in a liquid, the position of the float relative to the shaft prevents the magnetic coupling from occurring and, as the liquid level drops, the float slides down the shaft. When the water level forces the float to the bottom of the shaft, the float's magnetic ring forces the two contacts together. The completed circuit can then be used to detect low water levels using the microcontroller.

Figure 1-4. Water Level Sensor



2. Building the System

This system should monitor the pH, water temperature and water level of a small aquarium, and display each of the three sensor's data in a PC terminal window. Although the system is relatively simple, it does require three independent subsystems to work together. This application note will break down each part of the system in the order they were created. This will allow a reader who intends to use this application note as a guide to build and test each part of the system independently. Once the reader is familiar with the process, they can expand the system or make modifications easily.

2.1 Part One: The pH Sensor

The Atlas Scientific pH probe is connected to the Gravity pH Meter, which outputs an analog voltage range from approximately 3.0V to 0.265V (see Table 2-1) and can be read using the ADC of the PIC16F15245. Equation 2-1 is used to convert the ADC input voltage into a pH value.

Table 2-1. pH Meter Output Voltage

pH	Volts
0	2.745
1	2.570
2	2.390
3	2.210
4	2.030
5	1.855
6	1.680
7	1.500
8	1.330
9	1.155
10	0.975
11	0.800
12	0.620
13	0.445
14	0.265

Equation 2-1. Voltage to pH Conversion

$$pH = \left(-5.6548 \times \left(\frac{ADC\_RESULT}{MAX\_ADC} \times 5\right)\right) + 15.509$$

where:

ADC_RESULT = the ADC input conversion results, found in the ADRESH:ADRESL register pair

MAX_ADC = maximum resolution of the ADC in terms of bits

The pH calculation requires the use of floating-point arithmetic, which will require more memory. To minimize the memory requirements needed by the compiler when performing floating-point arithmetic, a small change in the MPLAB X's IDE will be required. When creating software in MPLAB X IDE version 5.40, the XC8 compiler by default uses the C99 compiler standard that utilizes 32-bit floating-point variables. For this application, 32-bit floats will not only consume more memory than is required, but also make transmitting the results over I^2C much more difficult. Instead, this project will make use of the C90 standard that uses 24-bit floats.

To change from the C99 to C90 standard, click on the Project Properties button (Figure 2-1) or **File > Project Properties**. In the **Categories:** window under **XC8 Global Options**, select **C90** from the **C standard** drop down menu (see Figure 2-2) and click **Apply**. Under **XC8 Linker**, select **C90** from the **Link in C library** drop down menu (see Figure 2-3). Click **Apply**.

Figure 2-1. Project Properties Button

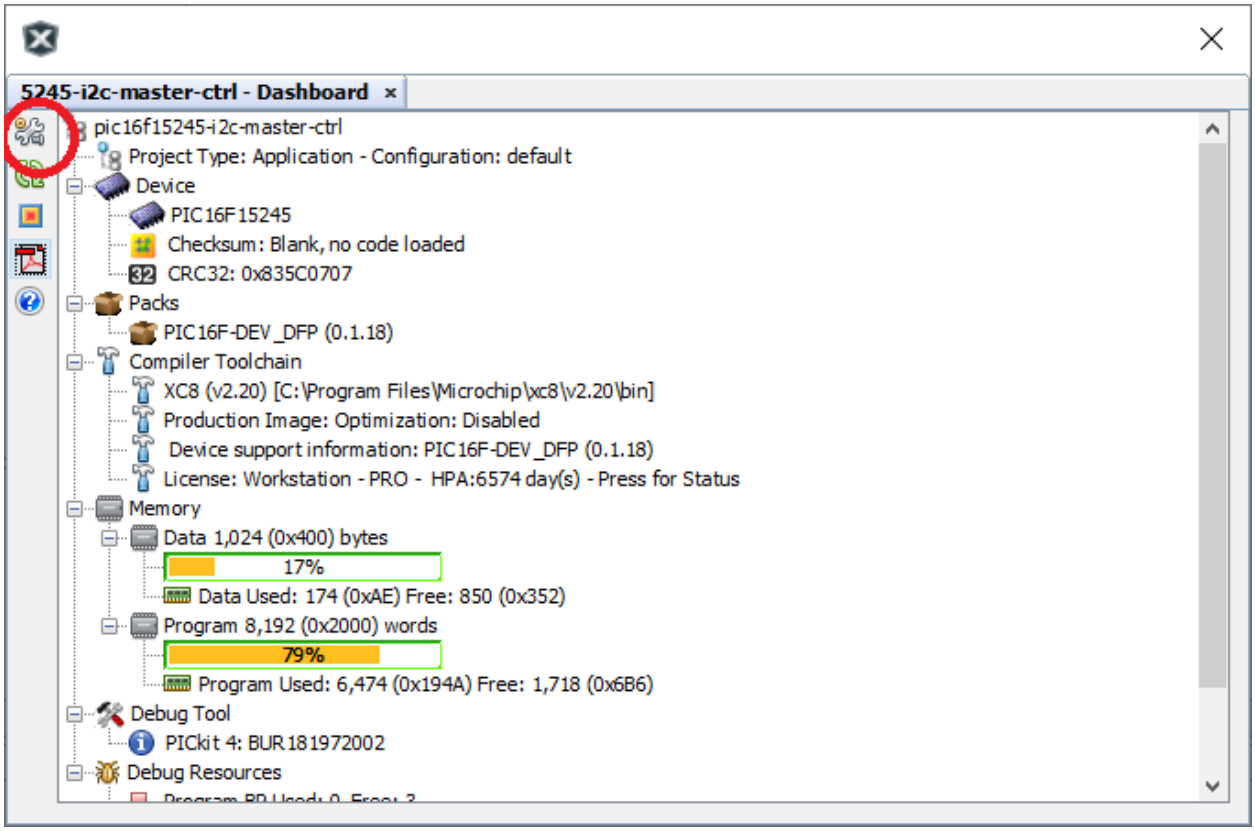


Figure 2-2. C99 to C90 in XC8 Global Options

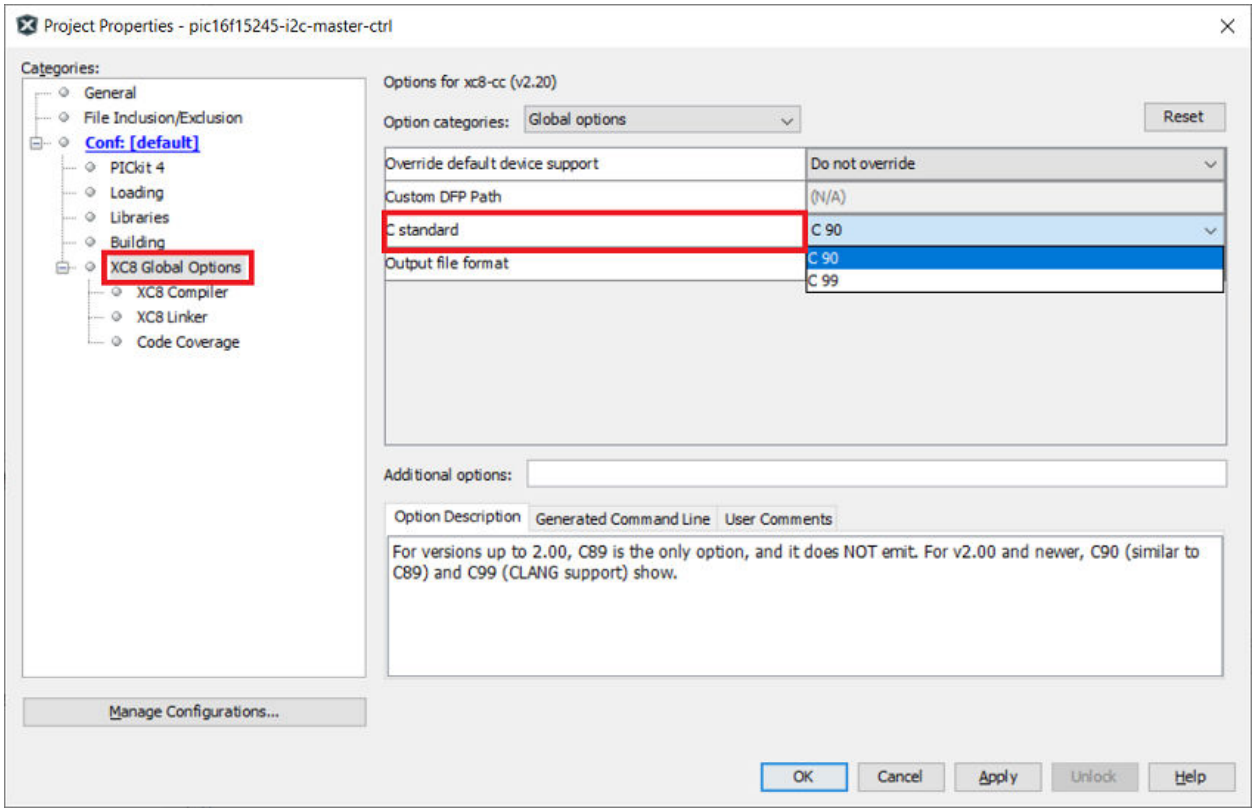
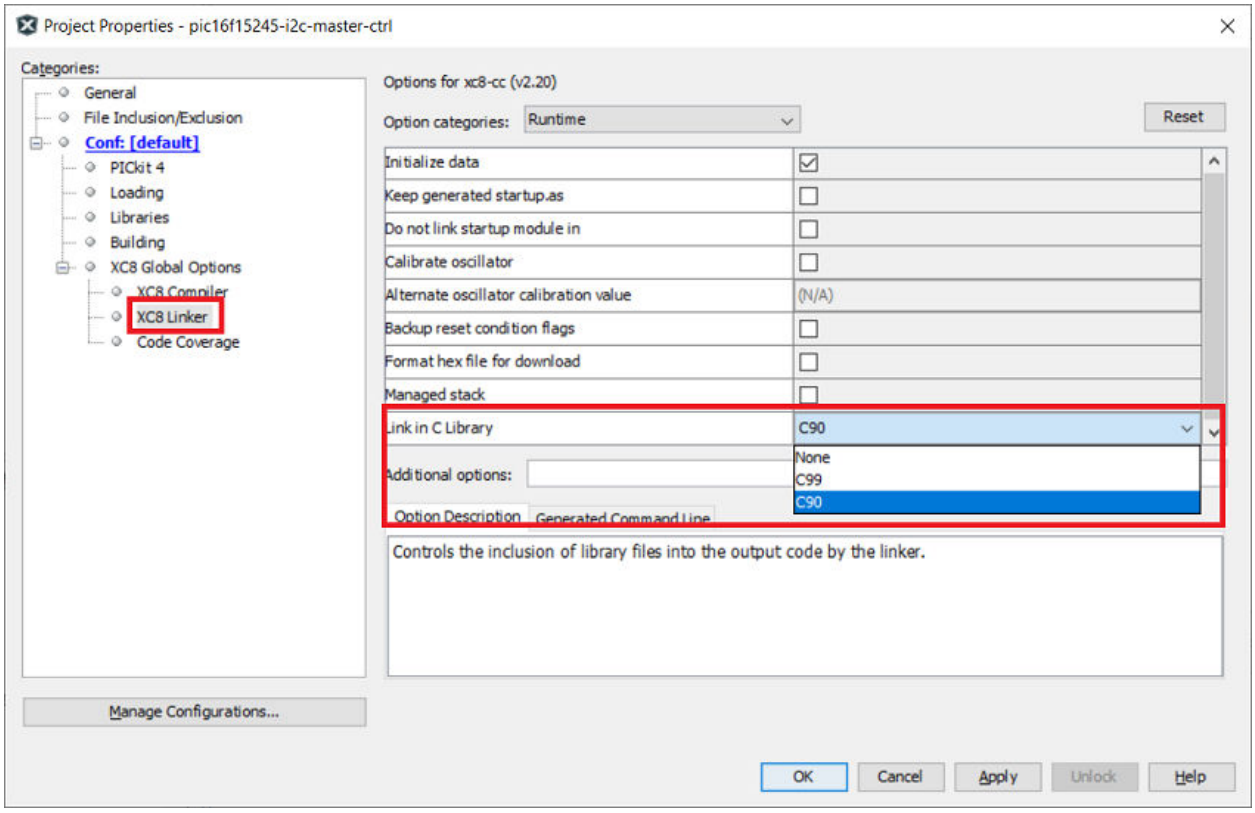


Figure 2-3. C99 to C90 in XC8 Linker



Example 2-1 shows the lines of code used in the project's `main while()` loop. The program reads the pH voltage provided to the analog input and converts the input into a floating-point pH value. At this point, a `printf` statement outputs the pH value via the EUSART. This is a great point to check to see if the output is what is expected.

Example 2-1. ADC Input to pH Calculations in `main()`

```
while (1)
{
    pHSensor = ADC_GetConversion(channel_ANA2);
    pHVoltage = (pHSensor/MAX_ADC);
    pHVoltage = pHVoltage * 5;
    pHValue = (PH_SLOPE * pHVoltage);
    pHValue = pHValue + PH_OFFSET;
    printf("pH Value = %1.1f \r\n", pHValue);
}
```

If the system only required the pH sensor, this code snippet can read and convert the data from the probe and output the results via the EUSART. Since the system requires I^2C communication, the 24-bit floating-point pH value must be converted into individual bytes that can be transmitted over the I^2C bus.

Example 2-2 shows the code used to convert the 24-bit float into three individual bytes that can be easily transmitted over the I^2C bus. The code breaks the float into three bytes and loads them into an array. When the I^2C master requests data from this slave, the slave will transmit the values contained in the array.

Example 2-2. Converting the 24-bit Float into Three Bytes

```
while (1)
{
    pHSensor = ADC_GetConversion(channel_ANA2);
    pHVoltage = (pHSensor/MAX_ADC);
    pHVoltage = pHVoltage * 5;
    pHValue = (PH_SLOPE * pHVoltage);
    pHValue = pHValue + PH_OFFSET;
    (uint24_t)pHCopy = (float)pHValue * 100;

// Break down the float variable into 3 bytes
    pHLowByte = pHCopy & 0xFF;                              // Get low byte
    pHHighByte = (pHCopy >> 8) & 0xFF;                      // Get high byte
    pHUpperByte = (pHCopy >> 16) & 0xFF;                    // Get upper byte
    (uint8_t)i2cArray[0] = (uint8_t)pHLowByte;
    (uint8_t)i2cArray[1] = (uint16_t)pHHighByte;
    (uint8_t)i2cArray[2] = (uint24_t)pHUpperByte;
}
```

Before the data is transmitted, the math used to convert the float into three integer bytes should be checked. This is accomplished by using the same method the master would use to convert the three integers back into the 24-bit float. Example 2-3 adds the extra code that the master will use to convert the three integers back into a floating-point variable. Again, the data is sent over the EUSART so that the results can be easily verified.

Example 2-3. Converting Three Bytes into a 24-bit Float

```
while (1)
{
    pHSensor = ADC_GetConversion(channel_ANA2);
    pHVoltage = (pHSensor/MAX_ADC);
    pHVoltage = pHVoltage * 5;
    pHValue = (PH_SLOPE * pHVoltage);
    pHValue = pHValue + PH_OFFSET;
    (uint24_t)pHCopy = (float)pHValue * 100;

// Break down the float variable into 3 bytes to transmit
    pHLowByte = pHCopy & 0xFF;                                  // Get low byte
    pHHighByte = (pHCopy >> 8) & 0xFF;                          // Get high byte
    pHUpperByte = (pHCopy >> 16) & 0xFF;                        // Get upper byte
    (uint8_t)i2cArray[0] = (uint8_t)pHLowByte;
    (uint8_t)i2cArray[1] = (uint16_t)pHHighByte;
    (uint8_t)i2cArray[2] = (uint24_t)pHUpperByte;
// Test convert back into 24-bit float
    (uint24_t)newpH = ((uint24_t)i2cArray[2] << 16);
    (uint24_t)newpH = (uint24_t)newpH + ((uint16_t)i2cArray[1] << 8);
    (uint24_t)newpH = (uint24_t)newpH + i2cArray[0];

    (float)pHValue = (uint24_t)newpH;
    pHValue /= 100.00;
    printf("pH Value = %1.1f \r\n", pHValue);
}
```

Once the results are verified, the last step is to add the I²C slave drivers. Example 2-4 shows the MSSP configuration and Interrupt Service Routine. The MSSP is configured in I²C Slave mode and uses the Interrupt Service Routine to transmit data to the master upon request.

Example 2-4. I2C Configuration and Interrupt Service Routine

```
void I2C_Initialize(void)
{
    SSP1STATbits.SMP = 1;                // Disable slew control
    SSP1CON1bits.SSPM = 0b0110;          // 7-bit slave mode
    SSP1CON2bits.SEN = 1;                // Enable clock stretching
    SSP1CON3bits.SBCDE = 1;              // Enable bus collision interrupts
    SSP1ADD = slaveAddress;              // Load slave address
    SSP1CON1bits.SSPEN = 1;              // Enable the module

    PIR1bits.BCL1IF = 0;                 // Clear Bus Collision interrupt flag
    PIR1bits.SSP1IF = 0;                 // Clear the SSP interrupt flag
    PIE1bits.BCL1IE = 1;                 // Enable BCLIF
    PIE1bits.SSP1IE = 1;                 // Enable SSPIF
    INTCONbits.PEIE = 1;                 // Enable peripheral interrupts
    INTCONbits.GIE = 1;                  // Enable global interrupts
}

void __interrupt() ISR(void)
{
    if(PIR1bits.SSP1IF)                  // Check for SSPIF
    {
        if(SSP1STATbits.R_nW == 1)       // Master to read (slave transmit)
        {
            SSP1BUF = i2cArray[index++]; // Load array value
            SSP1CON1bits.CKP = 1;        // Release clock stretch
        }
        if(SSP1STATbits.R_nW == 0)       // Master to write (slave receive)
        {
            if(SSP1STATbits.D_nA == 0)   // Last byte was an address
            {
                regAdd = 1;              // Next byte will be register address
                temp = SSP1BUF;          // Clear BF
                SSP1CON1bits.CKP = 1;    // Release clock stretch
            }
            if(SSP1STATbits.D_nA == 1)   // Last byte was data
            {
                if(regAdd == 1)          // Last byte was the register address
                {
                    index = SSP1BUF;     // Load register address into index
                    regAdd = 0;          // Next byte will be true data
                }
                else
                {
                    if(index < ARRAY_CNT)     // Within array boundaries?
                    {
                        i2cArray[index++] = SSP1BUF;  // Yes, load data
                    }
                    else
                    {
                        temp = SSP1BUF;      // No, loc invalid, discard data
                    }
                }
                SSP1CON1bits.CKP = 1;        // Release clock stretch
            }
        }
    }
    if(PIR1bits.BCL1IF == 1)
    {
        temp = SSP1BUF;                      // Clear BF
        PIR1bits.BCL1IF = 0;                 // Clear BCLIF
        SSP1CON1bits.CKP = 1;                // Release clock stretching
    }
    PIR1bits.SSP1IF = 0;                     // Clear SSP1IF
}
```

Important: This MSSP configuration is used in both slave devices in this project.

2.2 Part Two: The Temperature and Water Level Sensors

The next part of the project is to add the thermistor and water level sensor. First, start with the thermistor since it is more difficult to implement than the water level sensor.

As previously mentioned, the thermistor is part of a voltage divider system as shown in Figure 1-3. The output of the divider circuit is connected to an ADC input. Once the ADC reads and performs the ADC conversion, the conversion result is then used as part of the Simplified β (beta) Parameter equation (see Equation 2-2), which is a derivative of the Steinhart-Hart equation. The Simplified β Parameter equation uses terms that are easily attainable, such as the thermistor resistance value at 25°C, and reduces the complexity of the calculations that software must perform. To further simplify the software calculations, several of the more math-intensive variables have been pre-calculated by hand and listed as software constants.

Equation 2-2. Simplified β Parameter Equation

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln\left(\frac{R}{R_0}\right)$$

where:

T = Temperature in Kelvin (K)

T_0 *= 298.15 K (room temperature (25°C))*

B = Thermistor coefficient (in this case 3950)

R = measured thermistor resistance (ADC value (in voltage) must be converted to resistance)

R_0 *= Thermistor's rated resistance at 25°C* (in this case 10k Ω)

These calculations require floating-point math, similar to the pH sensor. When the project is created for this part of the system, the same steps to change from the C99 standard to the C90 standard must be followed. This will allow the use of 24-bit floats, which can then be broken down into three individual bytes for I^2C transmission (see Part One: The pH Sensor for details).

Example 2-5 shows the code used to calculate the temperature value and convert the 24-bit floating-point variable into three individual bytes. The snippet includes variables that have been precalculated in an effort to save additional memory space.

Example 2-5. Converting the Thermistor Input into Temperature

```
uint16_t adc_result = 0;
float temperature = 0.00000;
float T0 = 0.00335;                                 // 1 / 298.15
float B = 0.00025;                                  // 1 / 3950 (beta value)
#define MAXADC          1023.000                    // 10-Bit ADCC
#define K               273.1500                    // Kelvin constant
#define LN              0.4343                      // Log(e)

while (1)
{
    adc_result = ADC_GetConversion(channel_ANA2);

    temperature = (MAXADC / (float)adc_result) - 1.00000;// R equiv of voltage
    temperature = log10(temperature);  // Take Log of the temperature
    temperature = temperature / LN;    // Divide log(temp) by Log(e) value
    temperature = B * temperature;     // Multiply by B constant
    temperature += T0;                 // Add T0 constant
    temperature = 1 / temperature;     // Invert to get degrees in Kelvin (K)
    temperature -= K;                  // Subtract K constant
    //printf("%1.2fC \r\n", temperature);       // Display temp

    (uint24_t)thermCopy = (float)temperature * 100;

    // Break down the float variable into 3 bytes to transmit
    thermLowByte = thermCopy & 0xFF;                  // Get low byte
    thermHighByte = (thermCopy >> 8) & 0xFF;          // Get high byte
    thermUpperByte = (thermCopy >> 16) & 0xFF;        // Get upper byte
    (uint8_t)i2cArray[0] = (uint8_t)thermLowByte;
    (uint8_t)i2cArray[1] = (uint16_t)thermHighByte;
    (uint8_t)i2cArray[2] = (uint24_t)thermUpperByte;
}
```

Once the temperature has been calculated, the water level sensor can be added. The water level sensor is very easy to implement. Since the ADC is already in use, another ADC input can be enabled to read the sensor output. The water level sensor is a “normally open” circuit, meaning that no current can flow. When the water level is low, magnetic forces from the floating part of the sensor are close enough to force the two metals contacts located in the shaft of the sensor to make contact. This allows current to flow through the sensor, which results in a voltage that can be read by the ADC.

Two wires protrude from the shaft of the water level sensor. One wire is connected to a power source; in this case, it is connected to V_{DD}. The other end is connected to an ADC input pin. To prevent any transients from damaging the ADC input, a 100 Ω resistor is added in series to the ADC input. When the sensor is connected and the water level is good, the ADC pin will be floating and the ADC readings will be random. However, when the circuit closes, V_{DD} appears on the ADC pin. Software can read the input, and if the ADC value is anything below V_{DD}, the water level can be determined as good. If the ADC value is equal to V_{DD}, the water level is too low.

Example 2-6 shows the routine used to determine the water level. Since the sensor output can be converted into a 10-bit result, it is easy to convert the 10-bit reading into two bytes for I^2C transmission.

Example 2-6. Water Level Sensor Calculation

```
lvlSensor = ADC_GetConversion(channel_ANA4);
lvlSnsLowByte = lvlSensor & 0xFF;
lvlSnsHighByte = (lvlSensor >> 8) & 0xFF;
(uint8_t)i2cArray[3] = (uint8_t)lvlSnsLowByte;
(uint8_t)i2cArray[4] = (uint8_t)lvlSnsHighByte;
//printf("%d ADC Value \r\n", lvlSensor);
```

The last step is to add the I^2C slave drivers. The same driver set as used in Part One can be used here as well.

2.3 Part Three: The I²C Master

The final part of this project is the I²C master. The job of the master is to read the data from each of the two slaves, convert the data back into the proper variable type, and display the results on a PC terminal program.

The MSSP is configured in Master mode as shown in Example 2-7. The master begins by requesting the data from the slave device that controls the pH sensor. Once the master receives the pH data, it reconstructs the three bytes back into a floating-point number and writes the pH value to the PC terminal program. Next, the master requests data from the slave device that controls the temperature and water level sensors. Once the data is received, the master rebuilds the temperature data back into a floating-point number and writes the value to the terminal program. Then, the master converts the water level sensor data back into a 16-bit unsigned integer and compares the value to the expected values that represent either a “pass” or “fail” condition. Once the status has been selected, it is displayed on the PC terminal program. Example 2-8 shows the routines in `main()` that handle the calculations and display the data.

Important: When creating the master project, the C99 standard must also be changed to the C90 standard following the same steps as in Part One.

Example 2-7. MSSP Initialization

```
void I2C_Init(void)
{
    SSP1STAT = 0x80;         // Sample end of data output time
    SSP1CON1 = 0x28;         // SSPEN enabled; I2C master clk =FOSC/4(SSPxADD+1)
    SSP1CON3 = 0x00;
    SSP1ADD = 0x09;          // 0x4F = 100 kHz Clock @ 32 MHz

    PIR1bits.SSP1IF = 0;     // Clear the master interrupt flag
    PIE1bits.SSP1IE = 0;     // Disable the master interrupt
}
```

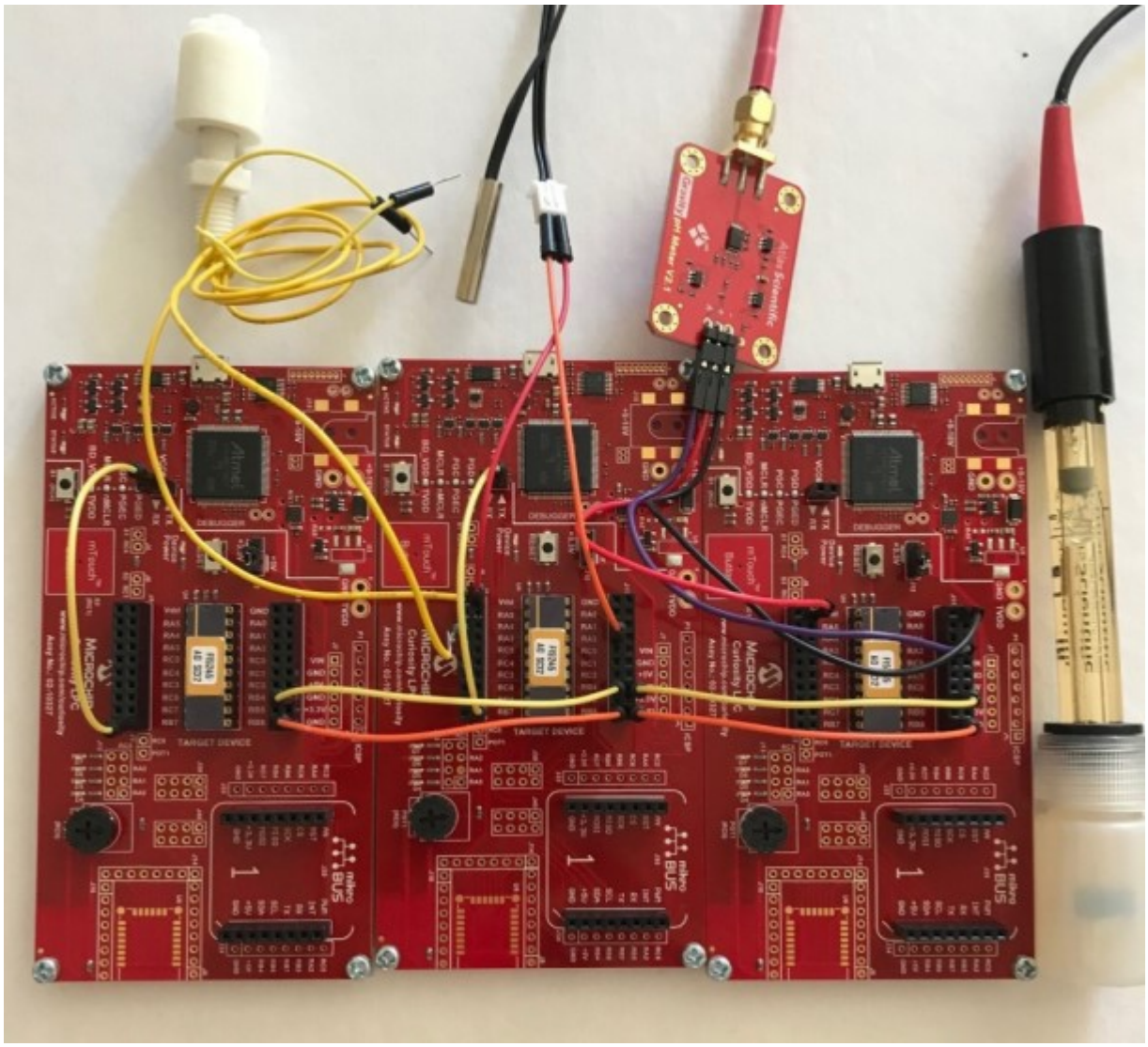
Example 2-8. MSSP Master Routines

```
// MSSP defines and macros
#define Idle        !(SSP1STATbits.R_nW | (0x1F & SSP1CON2)) // I2C Idle
#define I2C_Start   (SSP1CON2bits.SEN)                       // I2C Start
#define I2C_Restart (SSP1CON2bits.RSEN)                      // I2C Restart
#define I2C_Stop    (SSP1CON2bits.PEN)                       // I2C Stop


#define PH_ADDRESS      0x30
#define THERM_ADDRESS   0x32
#define PH_READS         0x3
#define THERM_READS      0x5

void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        I2C_Read(PH_ADDRESS, 0x00, PH_READS);  // Read pH sensor
        (uint24_t)newpH = ((uint24_t)I2CData[2] << 16);
        (uint24_t)newpH = (uint24_t)newpH + ((uint16_t)I2CData[1] << 8);
        (uint24_t)newpH = (uint24_t)newpH + I2CData[0];
        (float)pHValue = (uint24_t)newpH;
        pHValue /= 100.00;
        printf("pH Value = %1.1f \r\n", pHValue);

        I2C_Read(THERM_ADDRESS, 0x00, THERM_READS);    // Read temp sensor
        (uint24_t)newTherm = ((uint24_t)I2CData[2] << 16);
        (uint24_t)newTherm = (uint24_t)newTherm + ((uint16_t)I2CData[1] << 8);
        (uint24_t)newTherm = (uint24_t)newTherm + I2CData[0];
        (float)thermValue = (uint24_t)newTherm;
        thermValue /= 100.00;
        printf("Therm Value = %1.1f \r\n", thermValue);

        (uint16_t)lvlSensor = ((uint16_t)I2CData[4] << 8);
        (uint16_t)lvlSensor = (uint16_t)lvlSensor + I2CData[3];

        if(lvlSensor > 1010)
        {
            printf("WARNING: WATER LEVEL LOW!! ADD WATER! \r\n");
        }
        else
        {
            printf("Water level OK! \r\n");
        }
        __delay_ms(3500);
    }
}
```

3. Conclusion

The MSSP peripheral on PIC® microcontrollers can be configured in I²C mode to create a water monitoring system. Although this system consists of one master and two slaves, additional sensors can be easily added to the project. A new slave project can be added for each sensor and the master project files can be updated to add the new slave address, as well as any necessary calculations. This project also provides the technique required to convert a 24-bit floating-point variable into three integer bytes that can be transmitted over the I²C bus, and then back into a floating-point variable for display.

Figure 3-1. Complete Application



The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

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

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

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

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

ISBN: 978-1-5224-6329-0

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820