

Inductive Touch Software Design

*Author: Keith Curtis
Microchip Technology Inc.*

Note: Microchip inductive mTouch™ sensing solution is proprietary technology. It is available to customers free-of-charge under a license agreement permitting use and implementation of the technology on any PIC® microcontroller or dsPIC® digital signal controller.

INTRODUCTION

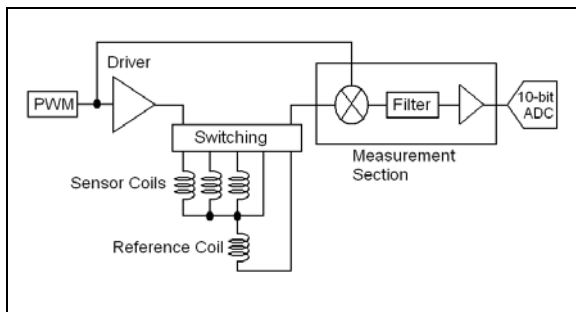
This application note is intended as a companion to application notes AN1237, “Inductive Touch Hardware Design” and AN1239, “Inductive Touch Sensor Design”. Its purpose is to acquaint the designer with the software functions used in Microchip’s proprietary inductive touch user interface system.

All information in this application note is applicable to any Microchip microcontroller with a 10-bit ADC, or an 8-bit ADC if the analog multiplexer switching topology is used.

THEORY OF OPERATION

The inductive touch user interface system is composed of three elements: the sensors, the hardware to perform the impedance measurement of the sensors, and the software to scan and interpret the impedance shift reported by the hardware.

FIGURE 1: INDUCTIVE TOUCH BLOCK DIAGRAM



A two part ratio-metric reading is used in order to measure the sensor. One reading is the amplitude of the pulsed DC voltage across the reference coil. The second reading is the amplitude of the pulsed DC voltage, generated across the sensor coil. By dividing the sensor coil reading by the reference coil reading, a normalized value corresponding to the impedance of the reference coil is generated, which is both drive voltage and temperature compensated.

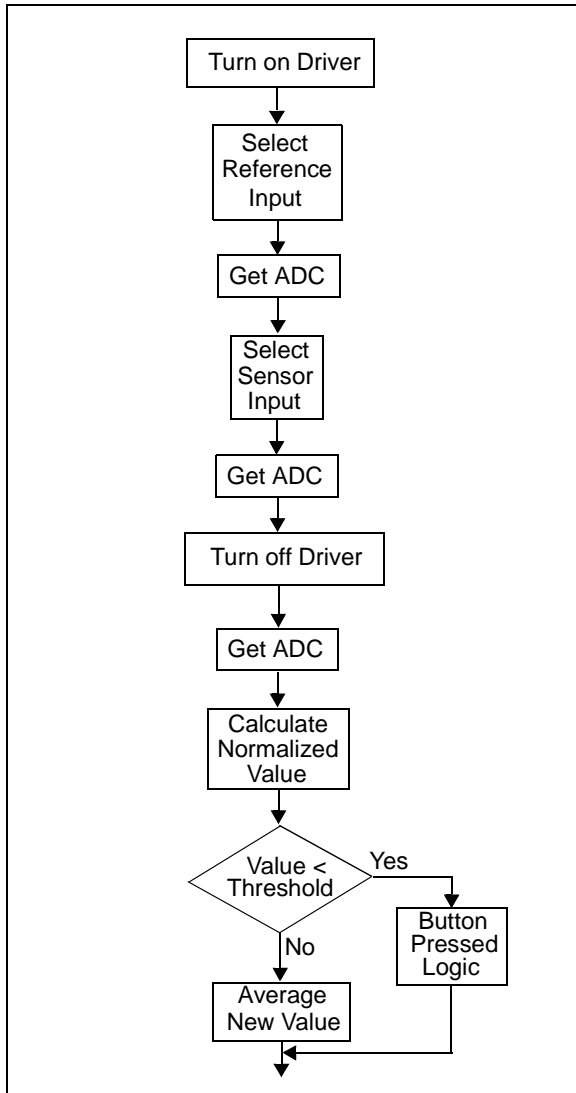
The advantage of a ratio-metric measurement is the elimination of supply voltage and temperature drift from the impedance measurement. Any change in the AC drive current, derived from the supply voltage, will affect the voltage on both coils equally. In the same way, any physical changes due to thermal expansion in the coils will affect the voltage. The result is that both effects cancel out in the division of one value, by the other.

To measure the two values, three ADC conversions must take place. The first measures the pulsed DC voltage across both coils. The second measures the pulsed DC voltage across the sensor coil (or reference coil in an analog multiplexer topology). And, the optional third measurement is the virtual ground reference of the detector.

Using these three values, the impedances of both the reference and the sensor coils, can be calculated. The sensor value is then divided by the reference value to produce the normalized ratio-metric impedance value for the sensor. Typically, the sensor coil value is found by subtracting the reference coil reading from the reading for both coils. The ground reference reading is then subtracted from the reference coil to remove the DC offset due to the virtual ground of the detector.

Determining a touch is accomplished by comparing the resulting normalized impedance value with an average value for the sensor. If the new value is lower, by a preprogrammed threshold, then the button is considered pressed, if not, it is released. See Figure 2 for a flowchart of the sensor measurement and decoding buttons.

FIGURE 2: FLOWCHART FOR INDUCTIVE TOUCH



CONTROLLING PULSED DC DRIVER

The frequency source, used to create the AC source for the driver, can be generated by a Capture/Compare/PWM (CCP) peripheral in its PWM mode, or by the CLKOUT function of the PIC microcontroller. This drive must be as high in frequency as possible, and have a 50% duty cycle. Given a 4 MHz oscillator clock, that means a 1 MHz clock source is possible if the CCP peripheral, in PWM mode, is used to generate the drive (due to the resolution of the PR2 register when it is used to set the period of the CCP PWM). Other clock sources can also be used in place of the PWM output, provided they have the ability to start and stop the clock under software control. For this application note, it will be assumed that the clock source is the CCP in PWM mode.

To turn on the driver, assuming a 1 MHz output and a 4 MHz oscillator, all that is needed is to change the duty cycle of the PWM, from 0% to 50%. This is accomplished by setting the PWM LSB (DC1B<0>, bit 4) in the CCP1CON register. To turn off the Pulsed DC Driver, simply clear the LSBs in the CCP1CON register.

SELECTING THE SENSOR OR REFERENCE INPUTS

The routing of the sensor and reference input voltages to the detector are handled by a Single Pole, Double Throw (SPDT) multiplexer at the input of the detector circuit. Selecting between the inputs is just a matter of setting or clearing the appropriate GPIO tied to the select line of the multiplexer.

Note: Before performing the actual ADC conversion, it is necessary for the output of the detector to stabilize. For this reason, a delay should be inserted between any changes to the configuration of the detector (driver on/off, or sensor/reference input select changes) and the conversion of the detectors output by the ADC. Equations for determining the appropriate delay are given in AN1237, "Inductive Touch Hardware Design", which covers the hardware design of the inductive touch interface.

GET ADC

The function that reads the ADC actually performs four ADC conversions. The results are added together and divided by 2. The result is an 11-bit number, using a 10-bit ADC.

The increase in resolution is due to random noise at the ADC input. If the conversion voltage is exactly centered on an ADC value, then the 4 values will average out to the ADC value multiplied by two. However, if the conversion voltage is centered between the ADC value, then the 4 conversion values will average out to a value half way between the two values, hence the 11th bit of the conversion. While two values would be sufficient most of the time, four values help to insure a more accurate conversion.

NORMALIZING THE IMPEDANCE MEASUREMENT VALUE

Once the ADC has converted the three values, they can be converted into a single normalized value. However, the exact form of the equation will depend upon the switching topology of the sensor and the reference coils. (see AN1237).

If the system uses a GPIO selection system, the ADC will return values `BothCoils_val` for the reference coil plus the sensor coil, `Sensor_val` just for the sensor coil, and a `Vref_val` for the detector circuit's virtual ground. The values `BothCoils_val` and `Sensor_val` incorporate an additional offset, `Offset_val`. This offset is estimated for each key at power-up, based on the assumption that all keys are initial un-pressed and the values for reference coil and sensor coils are the same. Under these conditions, the following equations are used to describe a GPIO selection system:

EQUATION 1: GPIO SELECTION SYSTEM EQUATIONS

$$\begin{aligned} \text{BothCoil_val} &= \text{Sensor_Coil} + \text{Ref_coil} + \text{Offset_val} + \text{Vref_val} \\ \text{Ref_coil} &= \text{Sensor_coil} \\ \text{Ref_coil} &= \text{BothCoil_val} - \text{Sensor_val} \\ \text{Offset_val} &= 2 * \text{Sensor_val} - \text{BothCoil_val} - \text{Vref_val} \\ \text{Sensor_coil} &= \text{Sensor_val} - \text{Offset_val} - \text{Vref_val} \\ \text{Normal_coil} &= (\text{Sensor_coil} * 1024) / \text{Ref_coil} \end{aligned}$$

If the system uses an analog multiplexer-based system, then the ADC will return a `BothCoils_val` value, a `Ref_val` value, and a `Vref_val` for the detector circuit's virtual ground. The equations for calculating a normalized value for this system are shown below:

EQUATION 2: ANALOG MULTIPLEXER SYSTEM EQUATIONS

$$\begin{aligned} \text{Sensor_coil} &= \text{BothCoil_val} - \text{Ref_val} \\ \text{Ref_coil} &= \text{Ref_val} - \text{Vref_val} \\ \text{Normal_coil} &= \frac{(\text{Sensor_coil} * 1024)}{\text{Ref_coil}} \end{aligned}$$

Once a normalized impedance value is obtained, it is compared against a coil average value, offset by a pre-programmed threshold. The threshold offset prevents low-level noise from triggering the sensor incorrectly, so the value of the threshold should be chosen such that a reasonable user press will trigger a press condition, but the worst-case noise in the normalized value will not cause a trigger.

Typically, the individual threshold for each button is calculated as a percentage of the current sensor average. This keeps the overall sensitivity of the various sensors equal.

The difference between the normalized impedance value and the coil average value is also a measure of the force exerted by the user on the sensor. By monitoring this value, the application software can modify the rate of change generated by the user's press, or even change the function of the button based on the force of the press. The result is a more intuitive feel for the user. For example, the system can increment a parameter faster in response to a hard press, or even change the function from a momentary button, into a latched function in response to the strong press. All that is required in the software is a variable to hold the peak difference during the last press, and logic to determine the appropriate response.

SYSTEM POWER-UP ROUTINES

In order to avoid false key-pressed events and fast system response, the software module will initialize the average and offset value with the value resulted from the first key scan:

EXAMPLE 1: SYSTEM INIT() ROUTINE

```
for(index=0; index < NUM_BTNS; index++)
{
    measure_pad(index);
    AVERAGE[index] = presspercent;
    AVERAGE[index] <<= 4;
    TRIP[index] = AVERAGE[index]*SENSITIVITY;
    OFFSET[index] = 2*sensor_coil-both_coil-Vref;
}
```

Note: At power-up, all keys should be un-pressed. If a key is pressed, additional detection logic will be required in software.

SYSTEM AVERAGING ROUTINES

The final segment of the conversion software is the sensor average for each input. This routine adds a small portion of each new value to an ongoing average, retained for each button. This average tracks the minor changes in the systems un-pressed button values for each sensor. An example code snippet is shown below:

EXAMPLE 2: SENSOR AVERAGING ROUTINE

```
small_avg = AVERAGE[channel] » 4;
AVERAGE[channel]- = small_avg;
AVERAGE[channel]+=presspercent;
```

For a GPIO selection system, the calculated offset should also track the minor system changes, as shown in Example 3:

EXAMPLE 3: SENSOR OFFSET ROUTINE

```
small_Offset_val = Offset_val[channel] >> 4
Offset_val[channel]- = small_Offset_val
Offset_val[channel]+ =
(2*Sensor_val-BothCoil_val-Vref-val)>> 4
```

Note: The average routine should only be called if the button is not currently being pressed. Averaging in a pressed value would result in the depression of the average and, ultimately, in the loss of the press condition even though the user was still holding down the Target.

Note: Other forms of averaging routines are discussed in the capacitive mTouch™ Software Developer's Kit, available on Microchip's web site at:
www.microchip.com/mtouch

The final function to perform at the end of a sensor conversion is to reconfigure the system for the next sensor to be scanned. This will leave the peripherals ready for the next conversion.

Appendix A contains a complete inductive touch code listing for a multiplexer-based system of 4 buttons, and a GPIO-based system. Other configurations can be found on Microchip's web page, attached to the inductive touch developer's page.

CONCLUSION

The software required to implement an inductive touch solution is relatively simple, when compared to the complexity of a typical capacitive touch solution. This simplicity is due to the ratio-metric measurement system employed by the inductive touch system. Because the reference coil provides a stable reference, against which the buttons sensors can be measured, there is no need to compensate for variations in either the temperature of the system, or the drive voltage.

MEMORY USAGE

A typical implementation will require approximately 2K words of program memory, and between 120 and 150 bytes of data.

APPENDIX A: SOFTWARE EXAMPLE FOR ANALOG MULTIPLEXER SYSTEM
Software License Agreement

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

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.

```
void main(void)
{
    OSCCON = 0x71;           // Internal oscillator 8mhz
    OSCTUNE= 0;
    while (HTS == 0);       // wait for clock to stabilise
    InitPORT();
    InitPeripheral();
    keypressed = 0;         // Initialise a few variables
    padcount = 0;
    general_delay(WAIT_100ms); // Settle down delay
    for (index = 0; index < NUM_BTNS; index++)
    {
        measure_pad(index);           // get a value for the button
        AVERAGE[index] = presspercent; // load into array
        AVERAGE[index] <= 4;         // multiply by 16
        TRIP[index] = AVERAGE[index] * SENSITIVITY; // calculate a trip point
    }
// Main program loop
while (1)                       /* INFINITE LOOP */
{
    measure_pad(padcount);       // Do actual pad read
    if (padcount == 0)
    {
        anykeypressed = 0;       // reset flag at the start of cycle
    }
    if (padpressed == 1)
    {
        if ((( keypressed >> padcount)& 0x01) == 0)
        {
            Beep(padcount);
        }
        keypressed |= (1<<padcount);
    }
    else keypressed &= (1<<padcount)^0xFFFF;
    padcount++;
    if (padcount >= NUM_BTNS) padcount = 0;
}
}
```

AN1241

```
// ***** Subroutines *****

void measure_pad(unsigned char channel)    // Performs actual read of key
{
    signed int    threshold;              // value for calculated threshold
    unsigned int  small_avg;              // current button value
    select_keypad(channel);               // first select the required keypad
    CCP1CON = 0b00101100;                 // Turn on pulse DC driver
    T2CON = 0b00000100;                   // select button + reference coils
    SelectRef = Button;                    // stability
                                           general_delay(RX_DELAY); // stability
    general_delay (RX_DELAY);              // stability
    both_coils = adc();                    // Read ADC
    SelectRef = Reference;                 // select reference coil only
    general_delay(REF_DELAY);              // stability delay
    ref_coil = adc();                      // Read ADC RS232 third field
    select_keypad(NUM_BTTNS);              // turn off buttons
    CCP1CON = 0b00001100;                 // turn off Pulsed DC driver
    T2CON = 0b00000000;
    general_delay(REF_DELAY);              // stability delay
    Vref = adc();                          // Read ADC - virtual ground reference
    presspercent = (ref_coil - both_coils ); // Subtract reading from background
    presspercent = presspercent << 10;      // Multiply to get sensible scale
    presspercent = presspercent / (Vref - ref_coil ); // Divide by (reference - Vref)
    small_avg = AVERAGE[channel]>>4;
    threshold = small_avg - (TRIP[channel]>>4); // effective threshold number
    if (presspercent < threshold)          // Compare reading to threshold
    {
        anykeypressed = 1;                // set flags if key pressed
        padpressed = 1;
    }
    else
    {
        padpressed = 0;
        if ((presspercent >= 400) & (presspercent <2000)) // remove bad values
        {
            AVERAGE[channel] -= small_avg;
            AVERAGE[channel] += presspercent;           // Averaging
        }
    }

    if (presspercent > small_avg) AVERAGE[channel] = (presspercent << 4);
    // slow press bleeds average, this is quick avg reset
    RAW[channel] = presspercent<<4;          // save raw value
}

unsigned int adc(void)                      // Get ADC reading
```

```
{
    unsigned int adc_avg = 0;
    unsigned char cntr;
    for (cntr = 0; cntr < 4; cntr++)
    {
        GODONE = SET;           // start conversion
        while (GODONE == SET); // wait while busy
        adc_avg += (ADRESH<<8) | ADRESL; // add value to average
    }
    adc_avg >>=1;              // multiply total average by 2 (shift by 1)
    return (adc_avg);
}

void general_delay(int delay) {
    while(--delay!=0);
    return;
}
```

APPENDIX B: SOFTWARE EXAMPLE FOR GPIO SELECTION SYSTEM

Software License Agreement

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

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.

```
void main(void)
{
    OSCCON = 0x71;           // Internal oscillator 8mhz
    OSCTUNE= 0;
    while ((OSCCON & 0x04) == 0); // wait for clock to stabilise
    InitPORT();
    InitPeripheral();
    keypressed = 0;         // Initialise a few variables
    padcount = 0;
    general_delay(WAIT_100ms); // Settle down delay
    for (index = 0; index < NUM_BTNS; index++)
    {
        measure_pad(index); // get a value for the button
        AVERAGE[index] = presspercent; // load into array
        AVERAGE[index] <<= 4; // multiply by 16
        TRIP[index] = AVERAGE[index] * SENSITIVITY; // calculate a trip point
        OFFSET[index]=2*sensor_coil-both_coil-Vref; //estimate the offset
    }
// Main program loop
    while (1)               /* INFINITE LOOP */
    {
        measure_pad(padcount); // Do actual pad read
        if (padcount == 0)
        {
            anykeypressed = 0; // reset flag at the start of cycle
        }
        if (padpressed == 1)
        {
            if ((( keypressed >> padcount)& 0x01) == 0)
            {
                Beep(padcount);
            }
            keypressed |= (1<<padcount);
        }
        else keypressed &= (1<<padcount)^0xFFFF;
        padcount++;
        if (padcount >= NUM_BTNS) padcount = 0;
    }
}
```



```

// ***** Subroutines *****

void measure_pad(unsigned char channel)      // Performs actual read of key
{
    signed int    threshold;                // value for calculated threshold
    unsigned int  small_avg;                // current button value
    select_keypad(channel);                 // first select the required keypad
    CCP1CON = 0b00101100;                   // Turn on oscillator
    T2CON = 0b00000100;
    SelectRef = Button;                     // select button + reference coils
    general_delay(RX_DELAY);                 // stability delay
    both_coils = adc();                      // Read ADC
    SelectRef = Reference;                   // select reference coil only
    general_delay(REF_DELAY);                // stability delay
    sensor_coil = adc();                     // Read ADC RS232 third field
    select_keypad(NUM_BTTNS);                // turn off buttons
    CCP1CON = 0b00001100;                   // turn off the drive
    T2CON = 0b00000000;
    general_delay(REF_DELAY);                // stability delay
    Vref = adc();                            // Read ADC - virtual ground reference
    presspercent = (sensor_coil - Vref-offset[channel]); // Subtract reading from
    background
    presspercent = presspercent << 11;       // Multiply to get sensible scale
    presspercent = presspercent / (both_coils-sensor_coil); // Divide by (reference
    - Vref)
    small_avg = AVERAGE[channel]>>4;
    threshold = small_avg - (TRIP[channel]>>4); // effective threshold number
    if (presspercent < threshold)           // Compare reading to threshold
    {
        anykeypressed = 1;                  // set flags if key pressed
        padpressed = 1;
    }
    else
    {
        padpressed = 0;
        if ((presspercent >= 1500) & (presspercent <2500)) // kill bad values
        {
            AVERAGE[channel] -= small_avg;
            AVERAGE[channel] += presspercent;           // Averaging
            small_offset=OFFSET[CHANNEL]>>4;
            OFFSET[channel]-=small_offset;
            OFFSET[channel]+=2*sensor_coil-both_coil-Vref;
        }
    }
    if (presspercent > small_avg) AVERAGE[channel] = (presspercent << 4);
    // slow press bleeds average, this is quick avg reset
    RAW[channel] = presspercent<<4;           // save raw value
}

```

AN1241

```
unsigned int adc(void) // Get ADC reading
{
    unsigned int adc_avg = 0;
    unsigned char cntr;
    for (cntr = 0; cntr < 4; cntr++)
    {
        GODONE = SET; // start conversion
        while (GODONE == SET); // wait while busy
        adc_avg += (ADRESH<<8) | ADRESL; // add value to average
    }
    adc_avg >>=1; // multiply total average by 2 (divide by 1)
    return (adc_avg);
}

void general_delay(int delay) {
    while(--delay!=0);
    return;
}
```

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.

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.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC, SmartShunt and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, WiperLock 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.

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

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

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

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



WORLDWIDE SALES AND SERVICE

AMERICAS

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

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

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

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

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

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

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

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

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

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

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EUROPE

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

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

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

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

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

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

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

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