INTRODUCTION
The PIC14C000 comes with several peripherals specifically aimed at the battery market. The programmable reference and onboard comparators are useful for creating charge control circuits, while the analog-to-digital (A/D) converter can monitor the charge state to prevent overcharge. The control software is written in "C" for maintainability and transportability. Where necessary, in line assembly is used.

This application note is intended to demonstrate the use of the PIC14C000 in an intelligent battery charger. The charger is designed to charge a sealed lead-acid battery (YUASA NP7-12 12V, 7AH); however, the charge parameters are easily modified to work with different lead-acid batteries.

The typical method of charging lead-acid batteries is with a constant voltage, current-limited source. That method allows a high initial charge current that tapers off until the battery reaches full charge.

This design uses a constant current, allowing the voltage to rise until the battery voltage reaches a full charge. The charge current is then turned off to prevent overcharging. This allows a high initial charge to quickly bring the battery to a full charge and a low maintenance charge current as needed to maintain the full charge. The constant current design is also easily adaptable to NiCd batteries.

As voltage rises during the charge cycle of the lead-acid battery, it quickly passes 2.1 V/cell. As charging progresses, oxygen begins to be liberated at the positive plates at 2.2 V/cell. At 2.3 V/cell, hydrogen is liberated at the negative plates. This is considered a full charge, as any further current passed into the cell simply releases gasses rather than charging the battery. Hence, the upper voltage limit is set at 13.8V (2.3 V/cell), and the lower voltage is set at 12.6V (2.1 V/cell). As a practical consideration, the lower voltage limit is set slightly lower (12.5V) to lengthen the charge cycles. The battery voltage takes just minutes to decay from over 13.8V to 12.6V. It then takes several hours to decay from 12.6V to 12.5V.

THEORY OF OPERATION
Charge current is controlled by a comparator and programmable reference onboard the PIC14C000. The other side of the comparator is fed by the voltage across a sense resistor. The output of the comparator controls a FET (Q1), which switches the charge voltage on and off. The charge is interrupted once-per-second to read the battery voltage. When it reaches a maximum voltage, the charge current is shut off to prevent overcharge.

The PIC14C000 continues to monitor battery voltage. Over time, the battery voltage decays. When the voltage drops below the lower threshold, the trickle charge is activated to bring the battery back up to full charge.

The computing power of the PIC14C000 allows the charger to accurately control the charge cycles to quickly recharge the battery while preventing overcharge.
CHARGING STRATEGY

First, charge at a high rate (1A for this battery) until the battery voltage is above the high limit (13.8V). The charge current is then cut off, allowing the battery voltage to decay until it descends past the low limit (12.5V). A low current charge (150 mA) is then applied to again bring the battery voltage up past the high limit. The drift-down/trickle charge cycle repeats (Figure 1).

FIGURE 1: CHARGING STRATEGY
ALGORITHM

The controller starts by measuring the voltage on the battery to determine the initial charge rate (high, low or off). Next, it sets up the comparator to control the constant current charge and then goes to sleep.

Note: The comparator continuously controls the current even while the controller sleeps.

After 1 second, the watchdog timer (WDT) wakes the controller, and the battery voltage is measured again. If any of the trip points are reached, the charge rate is adjusted. After the comparator is reset, the controller goes back to sleep, and the cycle repeats (Figure 2).

Once the measurement/decision cycle is complete, the controller goes to sleep for about 1.15 seconds (subject to the drift of the WDT's internal RC oscillator). Time-out of the WDT wakes up the controller and continues the cycle. The sleep cycle is used to save power and let the hardware do the work of counting the time rather than using timing loops.

Two LEDs are included to provide feedback on what the charger is doing. The red LED signifies a high current charge, and the green LED signifies a low current charge. While the charge is in progress, the active LED blinks at 1 Hz. This is the momentary suspension of charging while the battery voltage is measured.

As the battery ages, it may no longer be able to charge up past the high charge threshold. Time limits have been implemented to account for this. The high charge and low charge cycles have maximum time limits associated with them. The discharge cycle has a minimum time limit.

FIGURE 2: CONTROLLER FLOW CHART
Algorithm Parameters:
These parameters are #defines at the top of the code, meaning the code must be recompiled to change the parameters. The hardware could be modified to include dip or rotary switches to change some or all of these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Units</th>
<th>Range</th>
<th>Resolution</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V limit low</td>
<td>Volts</td>
<td>0-255.996</td>
<td>.00390625</td>
<td>Fixed point (16:8)</td>
<td>Minimum battery voltage</td>
</tr>
<tr>
<td>V limit high</td>
<td>Volts</td>
<td>0-255.996</td>
<td>.00390625</td>
<td>Fixed point (16:8)</td>
<td>Maximum battery voltage</td>
</tr>
<tr>
<td>High current</td>
<td>Amperes</td>
<td>0-255.996</td>
<td>.00390625</td>
<td>Fixed point (16:8)</td>
<td>High charge current</td>
</tr>
<tr>
<td>Low current</td>
<td>Amperes</td>
<td>0-255.996</td>
<td>.00390625</td>
<td>Fixed point (16:8)</td>
<td>Low charge current</td>
</tr>
<tr>
<td>High charge time limit</td>
<td>Minutes</td>
<td>0-65536</td>
<td>1</td>
<td>Unsigned long</td>
<td>Maximum time for high charge</td>
</tr>
<tr>
<td>Low charge time limit</td>
<td>Minutes</td>
<td>0-65536</td>
<td>1</td>
<td>Unsigned long</td>
<td>Maximum time for low charge</td>
</tr>
<tr>
<td>Charge rest time</td>
<td>Minutes</td>
<td>0-65536</td>
<td>1</td>
<td>Unsigned long</td>
<td>Minimum no charge time</td>
</tr>
</tbody>
</table>

† Range and Resolution are the mathematical precision that can be expressed with the parameter, not necessarily what the circuit is capable of.
HARDWARE

The PIC14C000 provides two comparators and programmable references (Figure 3). One set is used to maintain the charge current on the battery. Once the comparator is set up, it controls the current without processor intervention. The other comparator is not used in this application; however, it could be used to control current on a second battery to implement parallel charging. Battery voltage is measured using the 16-bit A/D converter.

The board used assumes the existence of an external power supply. This supply needs to provide some headroom above the expected maximum battery voltage and supply enough current for the selected high charge current. In this example, a 16.7V, 2.6A power supply was used.

The charger current to the battery is controlled by the comparator/buck converter. When the comparator senses that the charge current is too high, it pulls the gate of the Q1 low, turning off the current from the power supply and allows current to flow through D2. The buck converter (L1, C2, and D2) takes over and modulates the current to the battery at a controlled rate. When the comparator senses the charge current is too low, it turns on, allowing current from the power supply to flow through Q1. The buck converter now increases current at a controlled rate.

The component values for L1 and C2 are chosen based on the operating parameters of the system. For this system, the buck converter frequency is 15 KHz. The inductor (L1) is calculated from the equation:

\[ L = \frac{(V_I - V_{SAT} - V_O)}{I_{PK}} \cdot T_{ON} \]

For this design, \( V_I = 16.7 \text{ V} \), \( V_{SAT} = 0.25 \text{ V} \), \( V_O = 6.0 \text{ V} \) (minimum to support 6V battery) \( I_{PK} = 2A \), \( T_{ON} = 54 \mu s \) (80% duty cycle for high current charge):

\[ L = \frac{(16.7 - 0.25 - 6.0)/2 \cdot 5 \mu s}{282 \mu H} \]

The output capacitance is chosen such that:

\[ C_0 \geq \frac{I_{PK}T}{8V_{RIPPLE}} \]

For this design:

\[ I_{PK} = 2A, T = 66 \mu s, V_{RIPPLE} = 400mV \]

\[ C_0 \geq \frac{(2 \cdot 66)/8 \cdot 400mV}{41\mu F} \]

The diode (D2) needs to be sized large enough to handle I_{PK}.

FIGURE 3: SIMPLIFIED SCHEMATIC

See Figure 4 for actual circuit design.
DETAILS OF THE SOFTWARE IMPLEMENTATION

Constant Current Control

The comparator is used in conjunction with a programmable voltage reference to control the current into the battery. The voltage reference feeds one side of the comparator, while a sense resistor feeds the other. The output switches a FET to control the current. The voltage reference (PIC14C000 Data Sheet, Section 9 DS40122) is programmed as follows:

\[ V = \text{Current} \cdot \text{SenseResistor} + \text{LevelShift} \]

where:
- **Current** = The value we want the comparator to control to.
- **Sense Resistor** = The value of the current resistor (0.2 Ohms).
- **Level Shift** = The 0.5V shift performed on the voltage at the sense resistor (PIC14C000 Data Sheet, Section 9.2 – DS40122B).

This voltage is used in a lookup table which returns the coarse bits \( \text{PREFx<7:3>} \) for the programmable reference. The fine bits \( \text{PREFx<2:0>} \) are calculated as the difference between the voltage and coarse range, divided by the resolution of the table.

Analog-to-Digital Conversion

The battery voltage is measured via the A/D converter. The main program turns off the charger, then runs a conversion on the battery channel. Function AD_Counts performs 16 conversions on the channel, subtracts off the comparator/capacitor offset, and returns the average. The averaging is necessary to remove the noise from the system. The same A/D conversion is also performed on the internal bandgap reference. These values are then used in the following equation from AN624 to obtain the A/D converter voltage:

\[ V_{\text{IN}} = ((\text{Vin} - \text{OFFSET})/(\text{Nbg} - \text{OFFSET})) \cdot \text{Kbg} \]

The A/D converter operates most accurately with voltages near the bandgap reference. Therefore, the hardware runs the battery voltage through a voltage divider (R8 and R9) to drop the battery voltage (approximately 13V) down to 1.2V. The battery measurement calculation then multiplies the result from the A/D converter by the resistor ratio to get the original battery voltage:

\[ \text{BatteryVoltage} = V_{\text{IN}} \cdot (\text{R8 + R9})/(\text{R8}) \]

Recalibrating the A/D converter (Measuring OFFSET and Nbg) to compensate for component drift is done every cycle. Since the components do not drift very quickly, it’s not necessary to recalibrate this frequently, however, it is more accurate and takes advantage of otherwise idle processor time. If processor time is a concern, recalibrating once per minute is sufficient.

Time Keeping

The program counts the seconds to limit the charge cycles on the battery. The WDT times out about every 1.15 seconds, and the rest of the measurement cycle takes about 0.1 seconds, giving a total cycle time of 1.25 seconds. Each bump of the timer counts for 1 second, and every fourth second another is added to keep the count accurate. This method of timing is only accurate to a few percent. While not good enough for a clock, it’s accurate enough to limit the charge cycles on the battery.
Math

Fixed point math is used where resolutions of less than one are needed. This code can be updated to use floating point or 32-bit integers, which will allow a cleaner implementation of the calculations required. For this example, limited versions of basic Add/Subtract/Multiply/Divide functions operating on positive 32-bit integers are used. File "MATH32.C" implements 32-bit add, subtract, multiply, divide, and shift. The add, subtract, divide, and shift functions start with 32-bit values and give 32-bit results. The multiply starts with 32-bit multiplicands and gives a 64-bit result.

Charging Circuit Bench Results

When the programmable voltage reference was set to supply the fast charge current of 1A, the actual charge current was measured within 50 mA. However, when the programmable voltage reference was set to supply 150 mA trickle current, the actual output was measured to be as high as 275 mA. This higher trickle current is acceptable for this application since its purpose is to keep the battery topped off at full charge.

This delta is due to design limitations encountered when integrating analog components onto a digital substrate.

Converting the Charger to Work With Other Lead-Acid Batteries

This application note was specifically written to charge a YUASA 12V, 7AH battery, however other batteries may be charged with few modifications. Most of the charge algorithm parameters are in software. However, if the voltage is other than 12V, different resistors (R8 and R9) may be needed in the voltage divider leading to the A/D converter. For example, to charge a 6V, 2AH sealed lead-acid battery requires the necessary changes:

1. Swap R8 for a 137Ω, which will keep the voltage at the A/D converter around the 1.2V optimum.
2. Change the multiplier at the bottom of AN624, Equation 5 to:

\[
\frac{(R9 + R8)}{R8} = \frac{(1M + 137K)}{137K} = 8.29927
\]

3. Lower the power supply voltage (9V, 1A).
4. Change the charging parameters (#defines at the top of the code):
   a) V_LIMIT_HIGH 7.2V (0x0733 = 7.2 • 256)
   b) V_LIMIT_LOW 6.3V (0x064C = 6.3 • 256)
   c) HIGH_CURRENT (No change)
   d) LOW_CURRENT (No change)
   e) HIGH_CHARGE_TIME_LIMIT 120
   f) LOW_CHARGE_TIME_LIMIT 180
   g) CHARGE_REST_TIME 0

Conclusions

The PIC14C000 has several onboard peripherals that are specifically designed to simplify battery management applications. Further enhancements could be made such as:

1. Use an onboard temperature sensor to monitor battery temperature, which is another sign of overcharging. Once an over temperature condition is detected, the charge cycle would be terminated regardless of the battery voltage. This, however, would require the PIC14C000 to be physically attached to the battery. Alternatively, a remote temperature sensor could be mounted to the battery and read via the A/D converter.
2. The internal RC oscillator frequency drifts with temperature. This drift rate is known and stored in the calibration data (KTC), which would allow the timer to compensate with more accurate timing.
3. If better charge current accuracy is required, the charging circuitry should be implemented using external components.

REFERENCES

1. PIC14C000 data sheet (DS40122), Microchip Technology, Inc.
FIGURE 4:  PIC14C000 LEAD-ACID BATTERY CHARGER
# PARTS LIST

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>100 µF</td>
</tr>
<tr>
<td>C2</td>
<td>47 µF</td>
</tr>
<tr>
<td>Q1</td>
<td>MTP2955E</td>
</tr>
<tr>
<td>Q2</td>
<td>2N3904</td>
</tr>
<tr>
<td>C4</td>
<td>0.047 µF</td>
</tr>
<tr>
<td>C5</td>
<td>10 µF</td>
</tr>
<tr>
<td>C6</td>
<td>0.1 µF</td>
</tr>
<tr>
<td>D1</td>
<td>B220</td>
</tr>
<tr>
<td>D2</td>
<td>1N5817</td>
</tr>
<tr>
<td>D3</td>
<td>Green LED</td>
</tr>
<tr>
<td>D4</td>
<td>Red LED</td>
</tr>
<tr>
<td>J1</td>
<td>Connector appropriate to power supply.</td>
</tr>
<tr>
<td>J2</td>
<td>Connector appropriate to battery.</td>
</tr>
<tr>
<td>L1</td>
<td>270 µH</td>
</tr>
<tr>
<td>R1, R5, R6</td>
<td>1K</td>
</tr>
<tr>
<td>R2</td>
<td>150</td>
</tr>
<tr>
<td>R4</td>
<td>82</td>
</tr>
<tr>
<td>R8</td>
<td>68K</td>
</tr>
<tr>
<td>R9</td>
<td>1M</td>
</tr>
<tr>
<td>R11</td>
<td>0.2 Ohm, 1W, wire wound</td>
</tr>
<tr>
<td>R12, R13</td>
<td>470</td>
</tr>
<tr>
<td>R14</td>
<td>4.7K</td>
</tr>
<tr>
<td>U1</td>
<td>PIC14C000</td>
</tr>
<tr>
<td>VR1</td>
<td>7805 Voltage Regulator</td>
</tr>
</tbody>
</table>
APPENDIX A: LEADACID.C

/***************************************************************************
* Filename: LEADACID.C
***************************************************************************/

* Author:        Dan Butler
* Company:       Microchip Technology
* Revision:      Rev 1.0
* Date:          29 January 1997
* Compiler:      MPLAB-C rev 1.10

***************************************************************************
* Include files:
*               14000.h       Version 1.01
*               delay14.h     Version 1.00
*               Math.h        Version 1.00
*               math32.c      32 bit integer math functions
*               timer.c       simple timing functions
*
***************************************************************************
* Implements a simple battery charging algorithm. Uses comparator B to
* set up a constant current charge. Takes a reading on the battery once per
* second to see if the charge is complete.
*
* Clock Frequency     4 MHz Internal RC
* Configuration Bit Settings   WDT on, Power up timer off
* Program and Data Memory Usage
*   Program:  0x4E3
*   Data:     0x7f
*
***************************************************************************/

#include <14000.h>
#include <delay14.h>
#include <math.h>

/* ********************************************************************* */
/* Charging algorithm parameters. Change these as appropriate for your */
/* application.                                                          */
/* Fixed point values are used below. To calculate new values, multiply */
/* your floating point value by 2^(number of bits behind the decimal)    */
/* ********************************************************************* */
#define  V_LIMIT_LOW    0x0C80       /* Lower voltage limit 12.50V(2.08V/c)*/
#define  V_LIMIT_HIGH   0x0DCD       /* Upper voltage limit 13.8V(2.3V/c)*/
#define  HIGH_CURRENT   0x0100       /* 0x0100 = 1.000A, Fixed pt (16:8) */
#define  LOW_CURRENT    0x0026       /* 0x0026 = 0.150A, Fixed pt (16:8) */
#define  NO_CURRENT     0             /* off */
#define  HIGH_CHARGE_TIME_LIMIT  60  /* Max time for high current charge */
#define  LOW_CHARGE_TIME_LIMIT   120 /* max time for low current charge */
#define  CHARGE_REST_TIME        0   /* min time for rest between charge */
#define  SENSE_RESISTOR 0x3333       /* .2 ohms, Fixed point 16:16 */
#define  LEVEL_SHIFT    0x8000       /* .5 volt level shift, fp 16:16 */

/* End Algorithm Parameters ********************************************

#include <14000.h>
#include <delay14.h>
#include <math.h>

.FILL

/* what's changed */

/* Date          Description of Change */
/*            */
/*            */
***************************************************************************

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).
#define TEMP_SENSOR_CHANNEL 0x70       /* internal temperature sensor */

/* Variables global to eliminate parameter passing and visibility in emulator */
unsigned int Kref [3];                  /* 24 bit unsigned integer */
unsigned int KrefLo @ Kref;
unsigned int KrefMid @ Kref+1;
unsigned int KrefHi @ Kref+2;
unsigned int Krefexp;
unsigned int Kbg [3];                   /* 24 bit unsigned integer */
unsigned int KbgLo @ Kbg;
unsigned int KbgMid @ Kbg+1;
unsigned int KbgHi @ Kbg+2;
unsigned int Kbgexp;
unsigned long Nbg;
unsigned int NbgLo @ Nbg;
unsigned int NbgHi @ Nbg+1;
unsigned long Noffset;
unsigned int NoffsetLo @ Noffset;
unsigned int NoffsetHi @ Noffset + 1;
unsigned long Nbattery;
unsigned int NbatteryLo @ Nbattery;
unsigned int NbatteryHi @ Nbattery + 1;
unsigned long Vbattery;
unsigned int VbatteryLo @ Vbattery;
unsigned int VbatteryHi @ Vbattery + 1;
unsigned long ChargeState;

#include "math32.c"
#include "cmp-ref.c"
#include "timer.c"

/****************************************************************************
*       RunA2DConv
*              runs a conversion on the currently selected AD channel.
*              Input Variables:
*                     None
*              Output Variables:
*                     Returns ADCOUNT value.
****************************************************************************/
unsigned long RunA2DConv ()
{
    unsigned long adcounts @ ADCAPL;
    ADCON1 &= 0x0F;                     /* select current constant (27uA). */
    PIR1.ADCIF = 0;
    SLPCON.REFOFF = 0;
    SLPCON.ADOFF = 0;                   /* enable the AD module */
    ADCON0.ADRST = 1;                   /* Stop timer and fully dischage ramp capacitor */
    Delay_10xUs_4MHz (50);             /* delay 500 us */
    ADTMRH = 0;
    ADTMRL = 0;                        /* clear the conversion clock */
    ADCON0.ADRST = 0;                   /* start conversion */
    while (! (PIR1.ADCIF));            /* wait for conversion to complete */

    return (adcounts);
}

/****************************************************************************
*       Calibrate_AD
*              Runs an AD conversion on the High and Low references and
*              calculates the offset value. This is necessary to account
*              for capacitor dielectric absorption and comparator offset.
*              For more information, see AN624.
*
**Input Variables:**
- Kref (global) slope reference value from calibration table.

**Output Variables:**
- Noffset (global) AD counts to subtract to account for capacitor dielectric absorption and comparator offset

**Return Value:**
- None

******************************************************************************/

```c
void Calibrate_AD ()
{
    unsigned long difference;
    unsigned int difflo @ difference;
    unsigned int diffhi @ difference+1;
    unsigned long Nrefhi;
    unsigned long Nreflo;
    unsigned int i;
    unsigned int round;

    reg1 [0] = 0;
    reg1 [1] = 0;
    reg1 [2] = 0;
    reg1 [3] = 0;
    for (i = 0; i < 16; i++)
    {
        ADCON0 &= 0x0F;
        ADCON0 |= 0x50;  /* Select SREFHI */
        Nrefhi = RunA2DConv ();

        ADCON0 &= 0x0F;
        ADCON0 |= 0x60;  /* Select SREFLO */
        Nreflo = RunA2DConv ();

        difference = (Nrefhi - Nreflo);
        reg2 [0] = 0;
        reg2 [1] = 0;
        reg2 [2] = diffhi;
        reg2 [3] = difflo;   /* binary point all the way to the right */
        add32 ();
        reg1 [0] = reg3 [0];
        reg1 [1] = reg3 [1];
        reg1 [2] = reg3 [2];
        reg1 [3] = reg3 [3];
    }

    round = reg3 [3]; /* save off the highest order bit that will be lost*/
    for (i = 0; i < 4; i++)
    {
        asm
        bcf STATUS,RP0       /* assembly is necessary here. Using the */
        bcf STATUS,C         /* C shift operator (>>) doesn't work */
        rrf reg1,1           /* here because it clears the carry bit */
        rrf reg1+1,1         /* between shifts. */
        rrf reg1+2,1
        rrf reg1+3,1
    #endasm
    }
    diffhi = reg3 [2];
    difflo = reg3 [3];
    if (round & 0x08)             /* complete rounding operation rather than truncation */
        ++ difference;
    //    difference *= Kref;
    reg2 [0] = 0;
    reg2 [1] = KrefHi;
    //
}
```
reg2[2] = KrefMid;
reg2[3] = KrefLo;
Shift_R2_Left(); /* align the binary point just ahead of reg2[1] */
mult32(); /* binary point now between req3[4] and reg3[5] */
difflo = reg3[4];
diffhi = reg3[3];

if (difference > Nreflo) /* check to see Noffset would be negative */
    Noffset = 0;
else
    Noffset = Nreflo - difference;
}

/****************************************************************************
*       ADC_Counts
*              Do a conversion on the specified AD channel. Channel is
*              measured 16 times and averaged.
*              
*              Input Variables:
*                     channel (Parameter) AD MUX channel - ADCON0(7:4) per table 8-1.
*                     Noffset (Global)    16 bit unsigned int
*              Output Variables:
*                     None
*              Return Value
*                     Vbattery 16 bit fixed point
****************************************************************************/

unsigned long ADC_Counts(unsigned int channel)
{
    unsigned long Ncounts;
    unsigned int NcountsHi = Ncounts+1;
    unsigned int NcountsLo = Ncounts;
    unsigned int i,
    unsigned int round;
    for (i = 0; i < 4; i++)
    {      
        reg1[i] = 0;
    }
    ADCONO &= 0x0F;
    ADCONO |= (channel & 0xF0);
    for (i = 0; i < 16; i++)
    {
        Ncounts = RunA2DConv();
        Ncounts -= (long) Noffset;
        reg2[0] = 0;
        reg2[1] = 0;
        reg2[2] = NcountsHi;
        reg2[3] = NcountsLo;
        add32();
        reg1[0] = reg3[0];
        reg1[1] = reg3[1];
        reg1[2] = reg3[2];
        reg1[3] = reg3[3];
    }
    round = reg1[3];
    round &= 0x08; /* save off the highest order bit that will be lost */
    for (i = 0; i < 4; i++)
    {
        #asm
        bcf STATUS, RP0
        bcf STATUS, C
        rrf reg1, 1
        rrf reg1+1, 1
        rrf reg1+2, 1
    }
```c
unsigned long AN624Eq5 ()
{
   // Vin = Kbg * Nbattery;
   reg1 [0] = 0;
   reg1 [1] = 0;
   reg1 [2] = NbatteryHi;
   reg1 [3] = NbatteryLo;

   reg2 [0] = 0;
   reg2 [1] = KbgHi;
   Shift_R2_Left (); // align the decimal point
   mult32 (); // answer in Reg3

   // Vin /= Nbg; /* AN624 Equation 5. (Subtractions previously done) */
   reg2 [0] = 0;
   reg2 [1] = 0;
   reg2 [2] = NbgHi;
   reg2 [3] = NbgLo;

   reg3 [0] = reg3 [3];
   reg3 [1] = reg3 [4];
   reg3 [2] = reg3 [5];
```

---

**AN624Eq5**

- Takes the conversions previously done on the battery and the Bandgap, along with the calibration data in memory, and performs AN 624 Equation 5 to convert the battery counts back to the original voltage.

- An 624 Equation 5:
  \[
  V = \frac{(Nin - Noffset) \times Kbg}{(Nbg - Noffset)}
  \]

- Actual Calculation performed here:
  \[
  V = \frac{AvgNin \times Kbg}{AvgNbg}
  \]

- This answer gives the voltage at the ADC, however the circuit uses a resistive divider to take the ~12V at the battery down to the ~1.2V for the ADC. We need to multiply by the ratio of the resistances to get actual battery voltage.

- \[ V_{battery} = V \times \frac{1Mohm + 68Kohm}{68Kohm} \]

- For best accuracy, use measured values on the resistors

- Input Variables:
  - Nbattery unsigned int (16 bits)
  - Nbg unsigned int (16 bits)
  - Kbg fixed point (24 bits, 16 behind decimal)

- Output Variables:
  - None

- Return Value
  - Vbattery fixed point (16 bits, 8 behind decimal)
reg3 [3] = reg3 [6];

div32 ();

//  Vbattery = Vin * 1062000/62000;
reg2 [0] = 0;  //  15.70588 = 1068000/68000  (Nominal values)
reg2 [1] = 15; //  15.9598  = 1072180/67180  (measured values)
reg2 [2] = 180; // decimal portion (.9598 * 256)
reg2 [3] = 181;
mult32 ();

VbatteryHi = reg3 [3];
VbatteryLo = reg3 [4];

return (Vbattery);
}

/*******************************************************************************
*       GetCalData
*              retrieves the calibration parameters, and converts them to
*              fixed point format.
*              
*              Input Variables:
*              
*              Output Variables:
*              
*              fixed point (24 bits, 23 behind decimal point)
*              
*              fixed point (24 bits, 23 behind decimal point)
******************************************************************************/

void  GetCalData ()
{

#asm
    bsf      PCLATH,3  ; select page 1
    bcf      STATUS,RP0
    call     0x07c0
    movwf    Krefexp
    call     0x07c1
    IORLW    0x80      ; ignore sign bit, force implied bit
    movwf    KrefHi
    call     0x07c2
    movwf    KrefMid
    call     0x07c3
    movwf    KrefLo
    call     0x07c4
    movwf    Kbgexp
    call     0x07c5
    IORLW    0x80      ; ignore sign bit, force implied bit
    movwf    KbgHi
    call     0x07c6
    movwf    KbgMid
    call     0x07c7
    movwf    KbgLo

    bcf      PCLATH, 3
#endasm

    for (; Krefexp < 0x7f; Krefexp ++)
    {
        //  Kref >>= 1;

#asm
    bcf      STATUS,RP0
    bcf      STATUS,C
    rrf      KrefHi,F
    rrf      KrefMid,F
    rrf      KrefLo,F
#endasm

for (; Kbgexp < 0x7f; Kbgexp++)
{
    // Kbg >>= 1;
    bcf STATUS,RP0
    bcf STATUS,C
    rrf KbgHi,F
    rrf KbgMid,F
    rrf KbgLo,F
}

return;

/**************************************************************************
* Select New Charge
* Takes the current charge state, battery voltage and time in
* the current state and comes up with the new state. If a new
* state is entered, the timer is reset.
* 
* Input Variables:
* BatteryVoltage       (Parameter)       Current battery voltage
* ChargeState          (Global)          Current Charge State
*                                                            (Hi, Low or No current)
* Output Variables:
* ChargeState          (Global)          New Charge State
**************************************************************************/
void select_new_charge (unsigned long BatteryVoltage)
{
    unsigned long ChargeTime;

    ChargeTime = ChargeMinutes ();  /* how long have we been in the current charge state */
    if (ChargeState == HIGH_CURRENT)
    {
        if ((BatteryVoltage >= V_LIMIT_HIGH)
            || (ChargeTime >= HIGH_CHARGE_TIME_LIMIT))
        {
            ChargeState = NO_CURRENT;
            ResetTimer ();
        }
    }
    else if (ChargeState == LOW_CURRENT)
    {
        if ((BatteryVoltage >= V_LIMIT_HIGH)
            || (ChargeTime >= LOW_CHARGE_TIME_LIMIT))
        {
            ChargeState = NO_CURRENT;
            ResetTimer ();
        }
    }
    else if ((BatteryVoltage <= V_LIMIT_LOW)
        && (ChargeTime >= CHARGE_REST_TIME))
    {
        ChargeState = LOW_CURRENT;
        ResetTimer ();
    }
}

/**************************************************************************
* Setup WDT
* Sets up the WatchDog Timer for a 64:1 prescale. This will
* wake the part up 1.15 seconds after it goes to sleep.
**************************************************************************/
with interrupts disabled, processing resumes immediately
after the sleep instruction. Ref PIC14C000 Data sheet,
section 10.7 (1996/1997 databook).

Input Variables:
None
Output Variables:
None

void Setup_WDT ()
{
    OPTION = 0xCE; /* Prescaler on WDT, 64:1 prescale */
    INTCON = 0x00; /* all interrupts disabled */
    CLRWDT ();
}

void main ()
{
    unsigned long   BVoltage; /* battery voltage Fixed point (16:8) */
    unsigned int i;

    TRISD = 0x30; /* AN 4 and 5 inputs, rest of port D all outputs */
    StopCharge (); /* just in case it was running previously */
    GetCalData (); /* get the Kref & Kbg values from cal data memory */
    Calibrate_AD ();
    Nbattery = ADC_Counts (BATTERY_CHANNEL);
    Nbg = ADC_Counts (BAND_GAP_CHANNEL);
    BVoltage = AN624Eq5 ();

    if (BVoltage < V_LIMIT_LOW)
        ChargeState = HIGH_CURRENT;
    else if (BVoltage > V_LIMIT_HIGH)
        ChargeState = NO_CURRENT;
    else
        ChargeState = LOW_CURRENT;
    ResetTimer ();
    Setup_WDT ();
    while (1) /* loop forever */
    {
        ChargeCurrent (ChargeState);
        SLEEP ();
    }

    Input Variables:
    None
    Output Variables:
    None
    Returned Value
    None

****************************************************************************/
BumpTimer ();
Calibrate_AD ();  /* Calculates Noffset */
StopCharge ();
Nbattery = ADC_Counts (BATTERY_CHANNEL);
Nb = ADC_Counts (BAND_GAP_CHANNEL);
ChargeCurrent (ChargeState);  /* turn charger back on while we */
BVoltage = AN624Eq5 ();       /* crunch the numbers */
select_new_charge (BVoltage);
}
APPENDIX B: CMP-REF.C

/* ***************************************************************** */
/* Comparator - Reference utilities */
/* */
/* Functions: */
/* */
/* StopCharge - turn off the comparator and force RD2 low */
/* */
/* ChargeCurrent - Calculates the appropriate Vref value and */
/* Sets up Voltage Reference and Comparator B. */
/* ****************************************************************** */
/* these two arrays form a look up table for the Programmable voltage */
/* reference. The top voltage of the range is in the first table, */
/* and the corresponding coarse bits for the programmable reference */
/* are in the second table. The lookup then subtracts off the bottom */
/* of the range and divides by the set size to get the fine bits. */
/* ****************************************************************** */
const unsigned long  TopVoltage[32] = {
13107, /* .1500 - .2000 */
16384, /* .2000 - .2500 */
19660, /* .2500 - .3000 */
22937, /* .3000 - .3500 */
26214, /* .3500 - .4000 */
29491, /* .4000 - .4500 */
29497, /* .4500 - .4550 */
29818, /* .4550 - .4600 */
30146, /* .4600 - .4650 */
30474, /* .4650 - .4700 */
30801, /* .4700 - .4750 */
31129, /* .4750 - .4800 */
31457, /* .4800 - .4850 */
32112, /* .4850 - .4900 */
32440, /* .4900 - .4950 */
32768, /* .4950 - .5000 */
33095, /* .5000 - .5050 */
33423, /* .5050 - .5100 */
33751, /* .5100 - .5150 */
34078, /* .5150 - .5200 */
34406, /* .5200 - .5250 */
34734, /* .5250 - .5300 */
35061, /* .5300 - .5350 */
35389, /* .5350 - .5400 */
35717, /* .5400 - .5450 */
36044, /* .5450 - .5500 */
39321, /* .5500 - .6000 */
42598, /* .6000 - .6500 */
45875, /* .6500 - .7000 */
49152, /* .7000 - .7500 */
52428, /* .7500 - .8000 */
55705}; /* .8000 - .8500 */

const unsigned int   Coarse[32] = {
0xf8, /* .1500 - .2000 */
0xfe, /* .2000 - .2500 */
0xe8, /* .2500 - .3000 */
0xe0, /* .3000 - .3500 */
0xd8, /* .3500 - .4000 */
0xd0, /* .4000 - .4500 */
0xc8, /* .4500 - .4550 */
0xc0, /* .4550 - .4600 */
0xb8, /* .4600 - .4650 */
0xb0, /* .4650 - .4700 */
0xa8, /* .4700 - .4750 */
0xa0, /* .4750 - .4800 */  
0x98, /* .4800 - .4850 */  
0x90, /* .4850 - .4900 */  
0x88, /* .4900 - .4950 */  
0x80, /* .4950 - .5000 */  
0x00, /* .5000 - .5050 */  
0x10, /* .5100 - .5150 */  
0x18, /* .5150 - .5200 */  
0x20, /* .5200 - .5250 */  
0x28, /* .5250 - .5300 */  
0x30, /* .5300 - .5350 */  
0x38, /* .5350 - .5400 */  
0x40, /* .5400 - .5450 */  
0x48, /* .5450 - .5500 */  
0x50, /* .5500 - .6000 */  
0x58, /* .6000 - .6500 */  
0x60, /* .6500 - .7000 */  
0x68, /* .7000 - .7500 */  
0x70, /* .7500 - .8000 */  
0x78); /* .8000 - .8500 */

#define GREEN_ON PORTC.2 = 0 /* Macro to turn on Green LED */  
#define GREEN_OFF PORTC.2 = 1 /* Macro to turn off Green LED */  
#define RED_ON PORTC.3 = 0 /* Macro to turn on Red LED */  
#define RED_OFF PORTC.3 = 1 /* Macro to turn off Red LED */

/***************************************************************************/
/* StopCharge */  
/*              Disables the charge comparator and turns off the indicators */  
/* */  
/* Input Variables: */  
/* None */  
/* Output Variables: */  
/* None */  
/* Return Value */  
/* None */
/***************************************************************************/

void StopCharge ()
{
    CHGCON.CMBOE = 0; /* Disconnect RD2 from Comparator */  
    PORTD.2 = 0; /* Set RD2 LOW (turn off FET) */  
    RED_OFF;  
    GREEN_OFF;  
}

/***************************************************************************/
/* StartCharge */  
/*              Sets up the comparator and programmable voltage reference */  
/* */  
/* Input Variables: */  
/* PrefValue - Value for Programmable Voltage Ref */  
/* See PIC14C000 DataSheet tables 9-1 and 9-2. */  
/* Output Variables: */  
/* None */  
/* Return Value */  
/* None */
/***************************************************************************/

void StartCharge (PrefValue)
{
    unsigned int PrefValue;
    
    SLPCON.CMOFF = 0; /* enable comparators */  
    SLPCON.REFOFF = 0; /* enable power control references. */  
    SLPCON.LSOFF = 0; /* enable level shift network */  
    CHGCON.CMBOE = 1; /* comparator B output on RD2 */
CHGCON.CPOLB = 1;  /* comparator B output inverted. */
PREFB = PrefValue;
}
*/

void ChargeCurrent (ChargeRate) 
unsigned long ChargeRate; 
{
    unsigned long ControlV;
    unsigned int ControlVHi @ ControlV+1;
    unsigned int ControlVLo @ ControlV;
    unsigned long step;
    unsigned long templong;
    unsigned int fine; /* fine adjust bits of PREFB */
    unsigned int coarse;
    unsigned int i,j;
    TRISD = 0x30;           /* Set AN4 and AN5 for input */
    RED_OFF;
    GREEN_OFF;
    TRISC = 0x00;          /* RC for output */

    reg1 [0] = 0;
    reg1 [1] = 0;
    reg1 [2] = (ChargeRate & 0xFF00) >> 8;
    reg1 [3] = ChargeRate & 0x00FF;       /* decimal point before here */
    reg2 [0] = 0;
    reg2 [1] = 0;
    reg2 [2] = (SENSE_RESISTOR & 0xFF00) >> 8; /* decimal point before here */
    reg2 [3] = SENSE_RESISTOR & 0x00FF;
    mult32 ()
    ControlVLo = reg3 [6];  /* keep most significant bits */
    ControlVHi = reg3 [5];
    ControlV += LEVEL_SHIFT;

    for (i = 0; i < 32; i++)       /* now need to convert ControlV to PREFB value */
    {
        templong = TopVoltage [i];
        if (ControlV < templong)
        {
            if ((i < 6) || (i > 25))
                step = 409;
            else
                step = 41;
            j = i-1;
            templong = TopVoltage [j];
            ControlV -= templong;
            fine = ControlV / step;
        }
        CONTROLVLo = fine;
        CONTROLVHi = (ControlVHi + ControlVLo) & 0xFF;

        if (i < 6)"
        RED_ON;
        else"
        RED_OFF;
    }
    CONTROLVLo = ControlVLo;
    CONTROLVHi = ControlVHi;
    CONTROLVL = ControlVL;
    CONTROLVH = ControlVH;
    RETURN;
}
coarse = Coarse [i];
break;
}

if (ChargeRate == HIGH_CURRENT)
{
    RED_ON;    /* turn on red LED */
    StartCharge (coarse | fine);
}
else if (ChargeRate == LOW_CURRENT)
{
    GREEN_ON;    /* Turn on green LED */
    StartCharge (coarse | fine);
} else
    StopCharge ();
APPENDIX C: TIMER.C

/***************************************************************************
* Filename: timer.c
***************************************************************************/

* Author: Dan Butler
* Company: Microchip Technology
* Revision: Rev 1.0
* Date: 29 January 1997
* Compiler: MPLAB-C rev 1.10

***************************************************************************/

* Include files:
* none
*
***************************************************************************/

* Implements a timer operation:
*   ResetTimer
*   BumpTimer
*   ChargeMinutes
*   ChargeSeconds
*   
*   Clock Frequency   4 MHz Internal RC
*   Configuration Bit Settings   WDT on
*   Program and Data Memory Usage
*    
***************************************************************************/

* What’s Changed
*
* Date      Description of Change
*
*
***************************************************************************/

unsigned int   seconds;
unsigned int   correction;
unsigned long   minutes;

/***************************************************************************/
void  ResetTimer ()
{
    seconds = 0;
    minutes = 0;
    correction = 0;
}

/***************************************************************************/

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).
void BumpTimer ()
{
    if (++seconds == 60)
    {
        ++minutes;
        seconds = 0;
    }
    if (++correction == 4)
    {
        if (++seconds == 60)
        {
            ++minutes;
            seconds = 0;
        }
        correction = 0;
    }
}

unsigned long ChargeMinutes ()
{
    return (minutes);
}

unsigned int ChargeSeconds ()
{
    return (seconds);
}
APPENDIX D: MATH32.C

#include <14000.h>
#include <math.h>

unsigned int reg1 [4];       //math routine registers - 32 bits
unsigned int reg2 [4];       //32 bits normally, but need 64 for the div.
unsigned int reg3 [8];       //64 bits - used for multiply routine
unsigned int Quotient [4];
unsigned int carry;          //flag register for math routine
unsigned int sign;           // 1 - positive or zero, 0 - negative
unsigned long longtemp;
unsigned long X, Y;
unsigned int i,j;

/** add32
 * 32 bit unsigned addition reg3 = reg1 + reg2
 */

void add32 ()
{
    carry = 0;
    for (i = 3; i != 0xFF; i--)
    {
        X = reg1 [i];
        Y = reg2 [i];
        longtemp = X + Y + carry;
        reg3 [i] = (unsigned int) longtemp;
    }
}
carry = longtemp >> 8;
}
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
* sub32
*
* 32bit unsigned subtraction: reg1 = reg3 - reg2
* *
* Input Variables:
* reg3 - 32 bit unsigned integer
* reg2 - 32 bit unsigned integer
* *
* Output Variables:
* reg1 - 32 bit unsigned integer
* sign - 1: positive or zero. 0: negative

***************************************************************************/
void sub32 ()
{
    #asm
        movf reg3,w ; copy Reg3 to Reg1
        movwf reg1
        movf reg3+1,w
        movwf reg1+1
        movf reg3+2,w
        movwf reg1+2
        movf reg3+3,w
        movwf reg1+3

    ; Reg + 3
        movf reg2+3,w ; subtract low byte
        subwf reg1+3,1

    ; Reg + 2
        movf reg2+2,w ; move borrow bit to W reg
        btfss STATUS,
        incfsz reg2+2,w
        subwf reg1+2,1

    ; Reg + 1
        movf reg2+1,w
        btfss STATUS,
        incfsz reg2+1,w
        subwf reg1+1,1

    ; Reg + 0
        movf reg2,w
        btfss STATUS,
        incfsz reg2,w
        subwf reg1,1
        movf STATUS,w ; move borrow bit to W reg
        andlw 0x01 ; get rid of the rest
        movwf sign,1
    #endasm
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
* mult32
*
* 32 bit unsigned multiplication: reg3 = reg1 * reg2
* *
* Input Variables:
* reg1 - 32 bit unsigned integer
* reg2 - 32 bit unsigned integer
* *
* Output Variables:
* reg3 - 64 bit unsigned integer

***************************************************************************/
void mult32 ()
{
  for (i = 0; i < 8; i++)
    reg3 [i] = 0;

  for (i = 3; i != 0xFF; i--)
  {
    carry = 0;
    for (j = 3; j != 0xFF; j--)
    {
      X = reg1 [i];
      Y = reg2 [j];
      longtemp = X * Y;
      longtemp += reg3 [i + j + 1];
      longtemp += carry;
      reg3 [i + j + 1] = (unsigned int) longtemp;
      carry = longtemp >> 8;
    }
    reg3 [i] = carry;
  }
}

/***********************************************************/
*       Shift_R2_Left                                    *
*              Shifts all 32 bits of reg2 left one position:* *
*              reg2 <<= 1;                               *
*              Input Variables:                          *
*              reg2 - 32 bit unsigned integer             *
*              Output Variables:                         *
*              reg2 - 32 bit unsigned integer              *
/***********************************************************/
void Shift_R2_Left ()
{
  #asm
  bcf STATUS,C
  rlf reg2+3,1
  rlf reg2+2,1
  rlf reg2+1,1
  rlf reg2,1
  #endasm
}

/***********************************************************/
*       Shift_Q_Left                                    *
*              Shifts all 32 bits of Quotient left one position:* *
*              Quotient <<= 1;                           *
*              Input Variables:                          *
*              Quotient - 32 bit unsigned integer          *
*              Output Variables:                         *
*              Quotient - 32 bit unsigned integer          *
/***********************************************************/
void Shift_Q_Left ()
{
  #asm
  bcf STATUS,C
  rlf Quotient+3,1
  rlf Quotient+2,1
  rlf Quotient+1,1
  rlf Quotient,1
  #endasm
/****************************************************************************
*       Shift_R2_Right
*              Shifts all 32 bits of reg2 right one position:
*                     reg2 >>= 1;
*              Input Variables:
*                     reg2 - 32 bit unsigned integer
*              Output Variables:
*                     reg2 - 32 bit unsigned integer
****************************************************************************/
void Shift_R2_Right ()
{
#asm
   bcf       STATUS,C
   rrf       reg2,1
   rrf       reg2+1,1
   rrf       reg2+2,1
   rrf       reg2+3,1
#endasm
}

****************************************************************************
*       div32
*              32 bit unsigned division
*                     reg1 = reg3 / reg2
*              Input Variables:
*                     reg3 - 32 bit unsigned integer
*                     reg2 - 32 bit unsigned integer
*              Output Variables:
*                     reg1 - 32 bit unsigned integer
****************************************************************************/
void div32 ()
{
   i = 0;
   while (!(reg2[0] & 0x80))
   {
      Shift_R2_Left ();
      ++i;
   }

   Quotient [0] = 0;
   Quotient [1] = 0;
   Quotient [2] = 0;
   Quotient [3] = 0;

   for (j = 0; j <= i; j++)
   {
      Shift_Q_Left ();
      sub32 ();
      if (sign) // was the result positive?
      {
         reg3 [0] = reg1 [0];
         reg3 [1] = reg1 [1];
         reg3 [2] = reg1 [2];
         reg3 [3] = reg1 [3];
         Quotient [3] |= 0x01;
      }
      Shift_R2_Right ();
   }
reg1 [0] = Quotient [0];
reg1 [1] = Quotient [1];
reg1 [2] = Quotient [2];
reg1 [3] = Quotient [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.

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.

© 2002 Microchip Technology Inc.
# WORLDWIDE SALES AND SERVICE

## AMERICAS

**Corporate Office**
2335 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](http://www.microchip.com)

**Rocky Mountain**
2335 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: 770-640-0075

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

**Chicago**
500 Sugar Mill Road, Suite 200B
Atlanta, GA  30350
Tel: 770-640-0034 Fax: 770-640-0075

**Dallas**
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7943 Fax: 972-818-7204

**Detroit**
Tri-Aria 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: 765-864-8378

**Los Angeles**
18201 Von Karman, Suite 1090
Irvine, CA 92825
Tel: 949-263-1887 Fax: 949-263-1338

**New York**
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-758-5305 Fax: 631-758-5305

**San Jose**
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-9750 Fax: 408-436-9755

**Toronto**
6285 Northview 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
Tel: 61-2-9868-6750 Fax: 61-2-9868-6755

**China - Beijing**
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 101
Beihai Building 105, No. 4, 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 7F, 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 Xing Road
Shanghai, 200005
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

**China - Shenzhen**
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1715, 17/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

**Hong Kong**
Microchip Technology Hong Kong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-1341

**India**
Microchip Technology India
India Liaison Office
Divyasee Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessy 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, Shin Yokohama
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, 189890
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
Ballrup DK-2750 Denmark
Tel: 45 4420 9685 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 30-39-20 Fax: 33-1-69-30-90-79

**Germany**
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-6275-5700 Fax: 49-89-6275-5060

**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: 39-039-6899983

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