AN1007

Designing with the MCP3551 Delta-Sigma ADC

Author: Craig L. King
Microchip Technology Inc.

INTRODUCTION

The MCP3551 delta-sigma ADC is a high-resolution converter. This application note discusses various design techniques to follow when using this device. Typical application circuits are discussed first, followed by a section on noise analysis. This device has a LSB size that is smaller than the noise voltage, typical of any high-resolution ADC. Due to this, the performance of the device (and system) cannot be analyzed by simply looking at the binary output stream. Collecting data and visually analyzing the result is required; when designing circuits it is important to provide a way to get the data points to a PC. This application note shows how to use the MCP3551 22-Bit Delta-Sigma ADC PICtail™ Demo Board circuitry and DataView® software to quickly evaluate sensor or system performance, as well as how to interface the device to PICmicro® microcontrollers.

The DataView software allows real-time visual evaluation of system noise performance using histogram and scope plot graphs pertaining to many of the issues discussed herein.

Sections on anti-aliasing filter design and input settling time issues are also included. The serial communication firmware supplied is written in both software and hardware SPI™, C and Assembly for the PICmicro microcontroller. The software SPI™ code written in C is working code supplied with the MCP3551 22-Bit Delta-Sigma ADC PICtail™ Demo Board.

TYPICAL CONNECTION

A typical application for the MCP3551 device is shown in Figure 1, with the sensor connected to the MCP3551 22-Bit Delta-Sigma ADC PICtail™ Demo Board for system noise analysis and debugging.

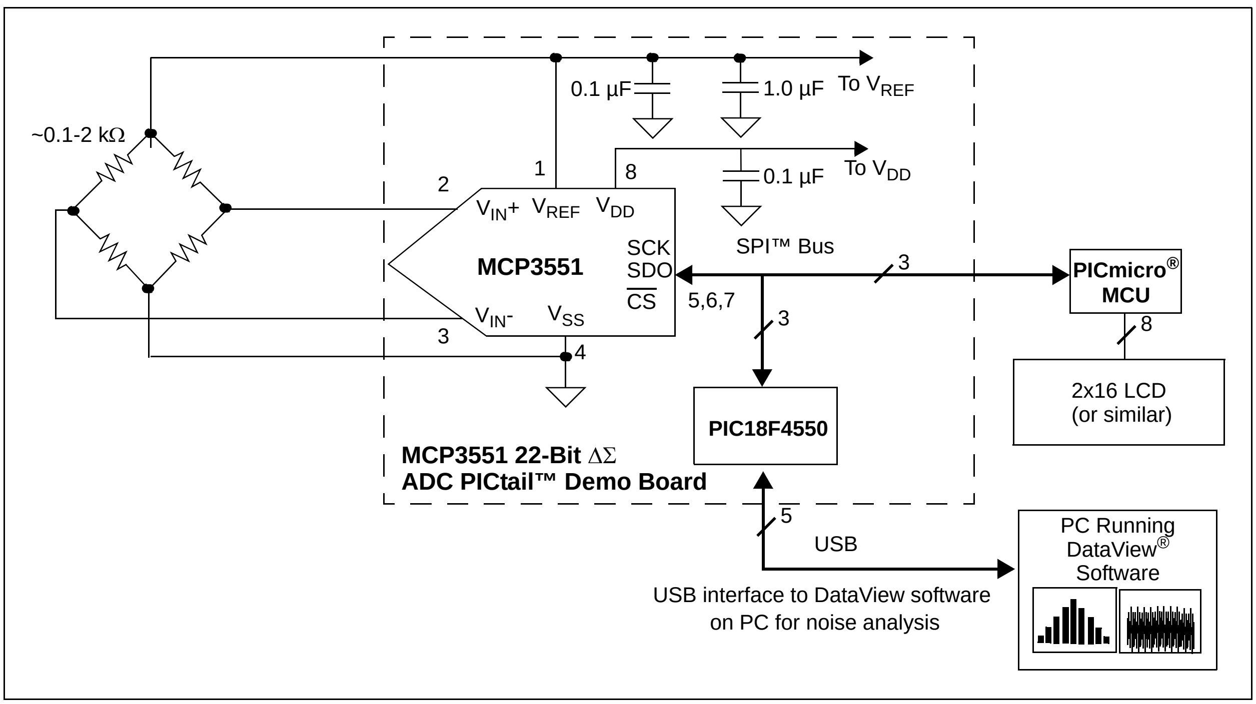

FIGURE 1: Typical Bridge Sensor Application Showing Connection for System Noise and Debug.

Sensors for temperature, pressure, load or other physical excitation are most often configured in a Wheatstone bridge configuration, as shown in Figure 1. The bridge can have anywhere from one to all four elements reacting to the physical excitation and should be used in a radiometric configuration when possible, with the system reference driving both the sensor and the ADC voltage reference. One example is General Electric's NovaSensor® absolute pressure sensor (NPP-301), shown in Figure 2 in a four-element varying bridge.

When designing with the MCP3551 ADC, the initial step should be to first evaluate the sensor performance and then determine what steps (if any) should be used to increase the overall system resolution. In many situations, the MCP3551 device can be used to directly digitize the sensor output, eliminating any need for external signal-conditioning circuitry.

The NPP-301 device has a typical full-scale output of 60 mV when excited with a 3V battery. The pressure range for this device is 100 kPa. The MCP3551 has an output noise specification of 2.5 μV_{RMS}.

The following equation is a first-order approximation of the relationship between pressure in Pascals (P) and altitude (h) in meters.

$$log(P) \approx 5 - \frac{h}{15500}$$

Using 60 mV as the full-scale range and 2.5 µV as the resolution, the resulting resolution from direct digitization (in meters) is 0.64 meters, or approximately 2 feet.

It should be noted that this is only used as an example for discussion; temperature effects and the error from a first-order approximation must be included in final system design.

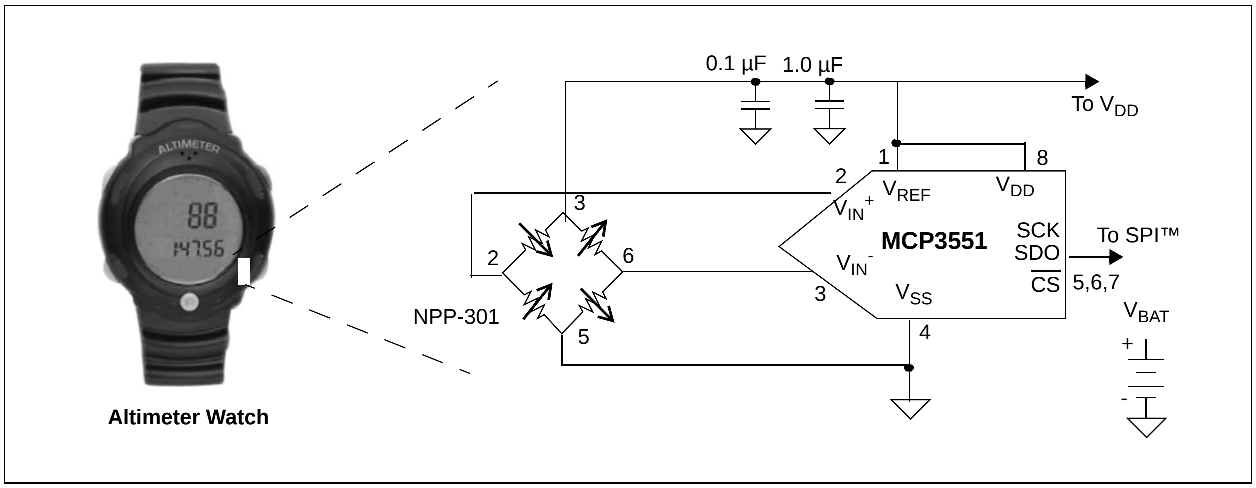

FIGURE 2: *Example of a direct digitization application. This is a low-power, absolute pressure-sensing module using the GE NovaSensor (NPP-301) series low-cost, surface-mount pressure sensor*

High-resolution ADCs, such as the MCP3551, can also be used to replace a solution that uses a lower-resolution ADC and a gain stage. The system block diagram shown in Figure 3 represents a typical signal-conditioning circuit. In this example, the required accuracy is 12 bits. A 12-bit ADC was selected and a gain stage was required to gain the signal prior to conversion. To achieve 12-bit accuracy, the entire input range of the ADC must be used. In this example, the signal also has a varying Common mode, which requires some offset adjustment calibration, along with perhaps a summing amplifier (i.e., the signal must be centered prior to the gain).

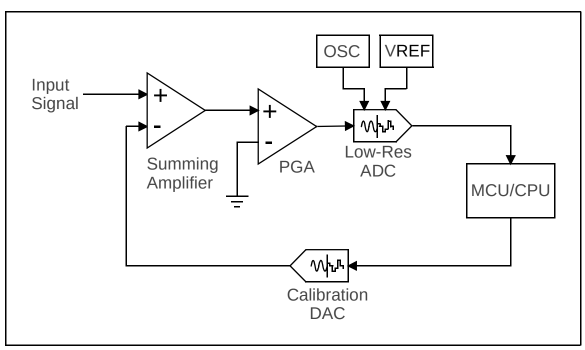

FIGURE 3: *Example Application Using Low-Resolution ADC and Signal-Conditioning Circuitry.*

The entire signal-conditioning circuitry can be eliminated in this situation by using the higher-resolution MCP3551 device.

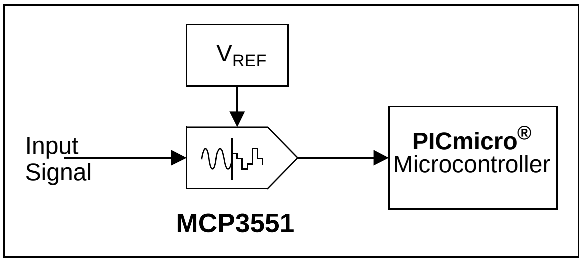

FIGURE 4: *Use of High-Resolution ADC, Eliminating Signal-Conditioning Circuitry.*

The large dynamic range of a high-resolution ADC (e.g., 22 bits, in the case of the MCP3551, eliminates the need for any system gain). In the above example, 12-bit accuracy was required. With 22-bit dynamic range, 12-bit accuracy exists anywhere within the input range of the ADC. Figure 4 shows this comparison with V_{REF} = 2.5V (**Note:** Not to scale).

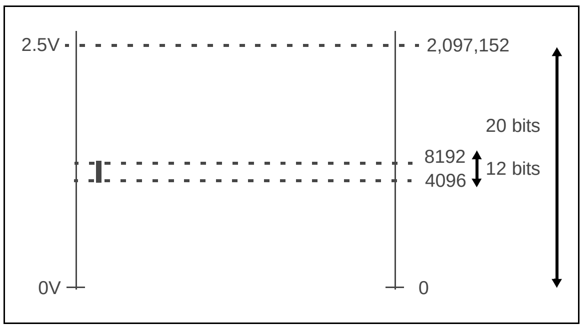

FIGURE 5: *The Large Dynamic Range of the MCP3551/3 Compared to that of a 12-bit ADC.*

Bits and Noise Analysis

With higher-resolution converters, the LSB size of the device is smaller than the device noise (i.e., there will always be a distribution of codes returned from the device). This output noise specification is measured by performing calculations on the output code distribution. The output code distribution defines what the effective resolution is, or Effective Number of Bits (ENOB) of the device. The output code distribution will have some standard deviation associated with it. This standard deviation is the RMS noise of the device (σ). The ratio of RMS noise (smallest signal that can be measured), to the full-scale input range of the device (largest signal that can be measured) is the effective resolution of the ADC. Converting to base 2 yields ENOB, as defined by Equation 1:

EQUATION 1:

$$ENOB = \frac{ln(FSR \diagup RMS\ Noise)}{ln(2)}$$

It should be noted that the formula for ENOB (or effective resolution) used in Equation 1 assumes a purely DC signal. A sinewave signal has 1.76 dB more AC power than a random signal uniformly distributed between the same peak levels.

If your application deals more with AC signals, the ADC performance can be viewed in the frequency domain using AC FFTs. These plots show Signal-to-Noise Ratio (SNR) or Signal-to-Noise And Distortion (SINAD). However, these are not typically found in low-bandwidth, delta-sigma data sheets.

The ENOB is naturally superior for large DC inputs compared to large AC inputs since, for AC inputs, the value comes close to 0 when the phase is close to 90°, which adds more uncertainty to the signal.

To calculate the ENOB using the standard SNR (dB)_=_6.02n+1.76 (which is derived using V_{RMS}_=_V_{PEAK}/2√(2), or a pure sine wave as the signal), Equation 2 should be used. The resulting ENOB has a difference of 1.76 dB in the calculation, or a difference of 0.292 bits less ENOB.

EQUATION 2:

$$ER\ in\ bits\ rms = \frac{20 \bullet \log\left(\frac{FSR}{RMS\ Noise}\right)}{6.02}$$

For a sensor with a 100 mV full-scale range output, the ENOB based on the MCP3551 resolution can be calculated as:

EQUATION 3:

$$ENOB = \frac{ln((100mV) \diagup (2.5\mu V))}{ln(2)}$$

Where:

ENOB = 15.3 bits

The MCP3551 output noise or effective resolution is specified with V_{REF} = 5V at 21.9 bits RMS. Predicting peak noise (or flicker-free) bits relies on statistical analysis and is discussed in a later section.

It should be noted that lowering the V_{REF} voltage of the ADC will not improve the output noise or effective resolution of the device, as this is dominated by the input thermal noise of the input structure.

In some applications, signal amplification will still be required to achieve the required system resolution. Analysis of the signal-conditioning circuitry required in these applications will not be covered in this application note.

When determining the sensor and, ultimately, the system resolution, all errors must be considered. Most errors can be calibrated out depending on the application. For example, consider a load cell with a specified error of 0.01%. With no calibration, the sensor limits the overall system resolution to 13.2 bits, still below the MCP3551 resolution with a full-scale sensor output of 100 mV.

Noise, by definition, is an aperiodic signal not having any wave or shape. This randomness is best dealt with in statistical properties, hence, the RMS measurement of the Gaussian (or normal) distribution. When designing a system and attempting to measure the performance, the RMS noise is much more repeatable than the peak-to-peak noise. Figure 6 shows two different distributions with different RMS and PEAK values, representing two different ADC output distributions.

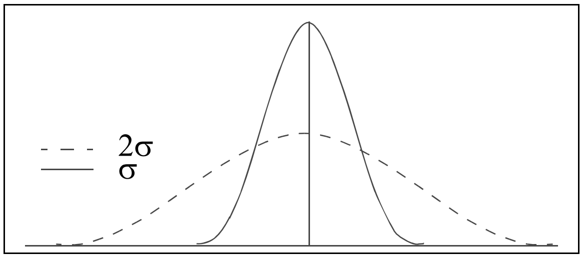

FIGURE 6: *Two Normal (Gaussian) Output Distributions.*

The DataView® software tool is a visualization tool showing real-time histograms using the MCP3551. The software also calculates the RMS noise of the current distribution. Additionally, the number of samples in the distribution is scalable, allowing post-averaging experiments.

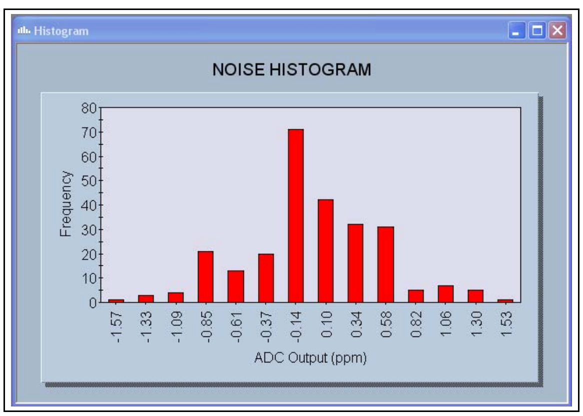

FIGURE 7: *DataView® software showing system performance in a histogram format.*

In the above example, the RMS noise was 0.8 ppm and the voltage reference was 2.5V. In this system, our ENOB was 21.6 using Equation 1.

The software can also be used for time-based system analysis using the scope plot window. Any system drift or other time-based errors can be analyzed using this visual analysis tool.

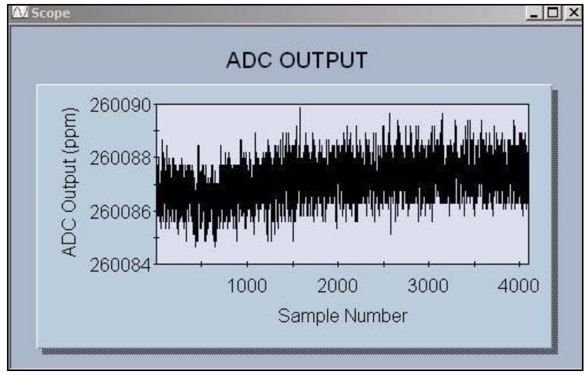

FIGURE 8: *DataView® Software Scope Plot View.*

DEBUG POLLING AND DATA LOGGING

The DataView software tool also allows the flexibility of changing the USB polling interval to a wide range of time periods, from milliseconds to hours. For applications requiring long-term data analysis, the system cache can be configured to show performance over long periods of time. Changing the DataView software's USB polling interval allows the designer to easily investigate long-term drift system issues, typical of high-resolution systems (shown in Figure 9). See the MCP3551 22-Bit Delta-Sigma ADC PICtail™ Demo Board User's Guide (DS51579) for more information on this feature.

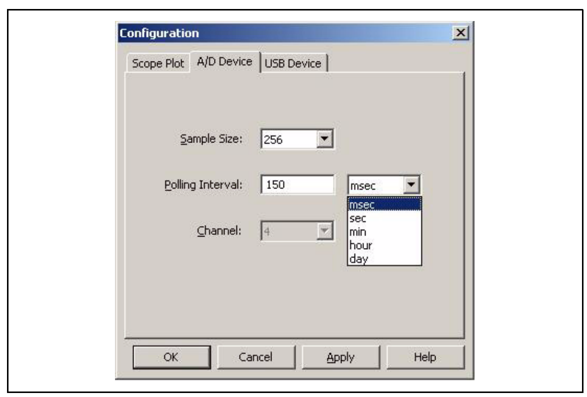

FIGURE 9: USB Polling Interval Control for System Drift Analysis

PREDICTING PEAK NOISE AND "NOISE-FREE BITS"

Peak-to-peak noise is much more difficult to measure, or predict, than measuring RMS noise. This peak-to-peak noise is also referred to as "noise-free" or "flicker-free" bits. Here we are attempting to predict the possibility of an output code occurring at the tips of the distribution. Based on the fact that the distribution is normal, or Gaussian (assuming the noise is entirely random), Table 1 is generated using standard statistical tables.The multiplier in the first column is the ratio of peak-to-RMS. This multiple (or ratio) is also known as a signal's "crest factor" when analyzing the power content of a signal. When analyzing noise, however, the multiplier should be chosen based on your application requirements.

The "empirical rule" of statistics can also be used as a general rule of thumb when approaching a good peak-to-peak window for your system during debug. The empirical rule states that 68% of normally distributed data falls within 1 standard deviation of the mean, 95% falls within 2 standard deviations of the mean and 99.7% falls within 3 standard deviations of the mean. For digital system designs, the most popular choice is 3.3 standard deviations from the mean, or 99.9% probability. For more or less rigid system designs, see Table 1 for other RMS-to-peak ratios or crest factors.

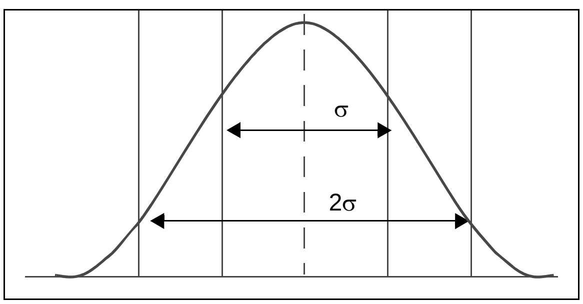

FIGURE 10: *n sigma (standard deviations) from the mean, basis for Table 1.*

As an example, let us choose an application that requires slightly more confidence in noise-ree bits (e.g., the feedback loop of a electronic defibrillator for heart failure).

Using the DataView software too, the characterized system noise is 0.5 ppm, RMS.

$$e_N(RMS) = 0.5ppm$$

$$e_N(p-p) = e_N(RMS) \bullet K$$

$$= (0.5ppm) \bullet (10)$$

$$= 5.0ppm$$

$$1ppm = \frac{2^N codes}{1000000} = \frac{2^{22}}{1000000}$$

$$= 4.194\ codes$$

$$e_N(p-p) = 5ppm = 20.97\ codes$$

$$= 21\ codes$$

Here, the multiplier of 5 was chosen to be more conservative, with the resulting window having a width of 21 output codes.

TABLE 1: CONFIDENCE TABLE TO PREDICT "NOISE FREE" BITS

Distribution Window Around Mean (Peak-to-Peak Window)	Probability of Output Codes Within Window	Probability of Output Codes NOT Within Window
2.0 x RMS or 1 sigma	68%	32%
3.0 x RMS or 1.5 sigma	87%	13%
4.0 x RMS or 2 sigma	95.4%	4.6%
5.0 x RMS or 2.5 sigma	98.8%	1.2%
6.0 x RMS or 3 sigma	99.73%	0.27%
6.6 x RMS or 3.3 sigma	**99.9%**	**0.10%**
8.0 x RMS or 4 sigma	99.954%	0.046%
10.0 x RMS or 5 sigma	99.994%	0.006%

ANALOG FRONT-END (AFE) DESIGN

Before looking at anti-aliasing and settling time issues, the input structure and operation of the device must first be evaluated. The input pins of the MCP3551 device are switched-capacitor-type inputs. The input pins go directly into the delta-sigma modulator, which oversamples the input at a frequency equivalent to the internal oscillator divided by four ($f_{INT}/4$). The result is a four-phase sampling scheme between the reference and input. During the sample time t_{CONV}, the ADC is constantly comparing the differential input voltage to the voltage reference and transferring this charge to the input capacitors. Figure 11 illustrates this timing using the MCP3551 device.

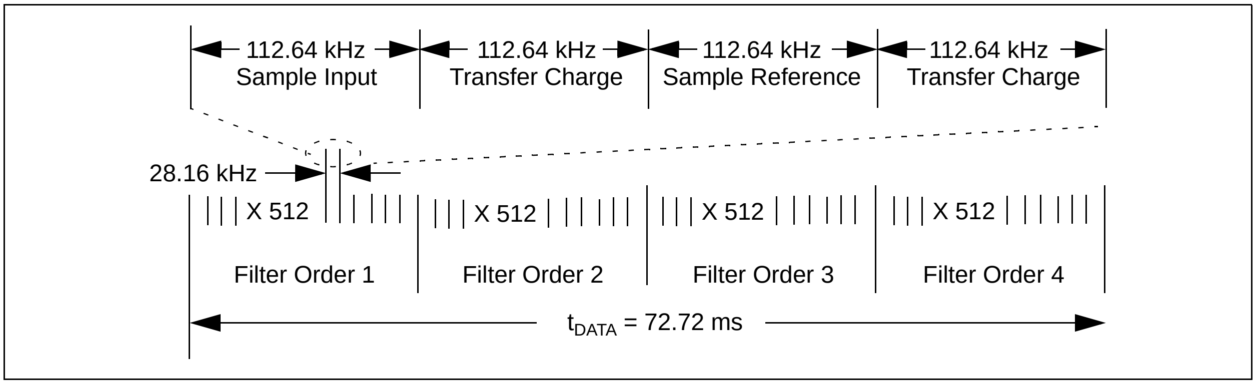

FIGURE 11: *Internal timings of the MCP3551 device. For settling time issues, charge transfer frequency must be observed. For aliasing issues, the oversampling frequency of 28.16 kHz is the focus.*

ANTI-ALIASING FILTER DESIGN

Regardless of the ADC architecture, an anti-aliasing filter is sometimes required. The delta-sigma ADC is no exception. Based on the SINC filter response in Figure 12, a simple, low-cost RC filter is all that is required to eliminate unwanted signals around the oversampling frequency.

The MCP3551 device has an oversampling frequency of 28.16 kHz. The MCP3553 device has an oversampling frequency of 30.72 kHz, with a lower Oversampling Ratio (OSR) for higher data rate or Nyquist frequency. The Nyquist or output data rate of the MCP3551 and MCP3553 devices are 13.75 Hz and 60 Hz, respectively.

The SINC filter response of the MCP3551 has lobes that give increasing attenuation with frequency, as shown in Figure 12. The anti-aliasing filter requirements should be selected with the attenuation of the SINC filter in mind.

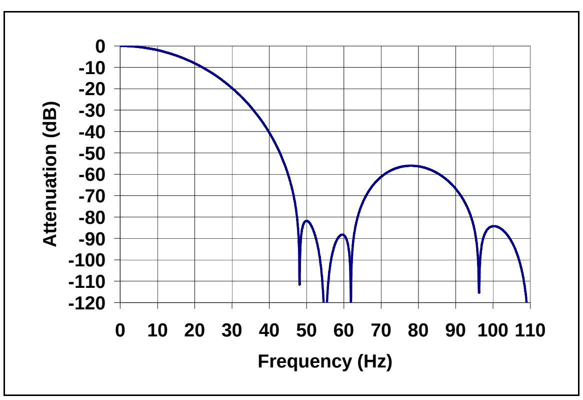

FIGURE 12: *MCP3551's modified SINC filter.*

Keep in mind that the ill-used components will not be at full-scale, and will typically be at a smaller amplitude. From Figure 12, the largest SINC lobe is down approximately 60 dB (the aliasing components are at -20 dB), so an additional 20 dB is required from the anti-lasing filter to get to 100 dB.

Microchip's free FilterLab® filter design tool can be used to easily estimate the single-pole RC attenuation for specific filter cut-off frequencies and aliased signal frequency components. Figure 13 shows a RC designed with a 1 kHz cut-off frequency, giving greater than 30 dB at the sampling frequency of 30.72 kHz.

It should be noted that at integer multiples of the sampling frequency, the SINC filter response will repeat, in which the SINC filter response will be zero.

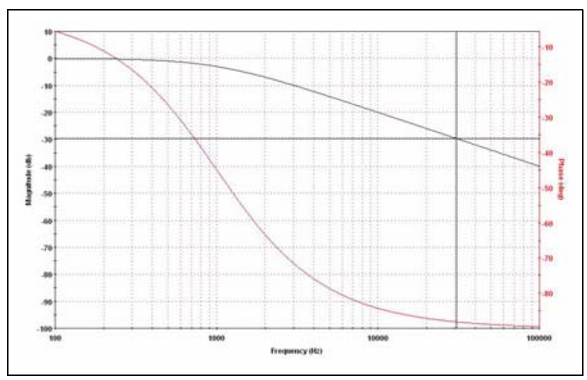

FIGURE 13: *FilterLab® filter design tool showing RC response.*

INPUT IMPEDANCE

In Figure 11, the switching frequency at the inputs of the devices is equivalent to the internal oscillator frequency in every phase. The input pin resistance is calculated to be the switching frequency multiplied by the capacitance and the equivalent capacitance (C_{EQ}). The resulting RC defines the settling time required at the input to the device.

Any additional RC added to the input will cause the input signal to not be completely settled during the oversampling internal to the device. It is important to note that, due to the oversampling and averaging performed by the delta-sigma architecture, the additional RC added here will be consistent across each oversampled charge. The resulting effect on the device output will be an error in conversion offset and gain.

The linearity of the device will not be compromised. The output noise performance will also not be compromised, assuming the thermal noise added by the input resistance does not exceed the output noise specification

If analysis of the offset/gain effect is desired, analysis of the settling time curve of the internal RC, compared to the desired system accuracy, should be performed. Figure 14 shows the RC charging curve for the internal resistance and capacitance only (R_{SW} and C_{EQ}).

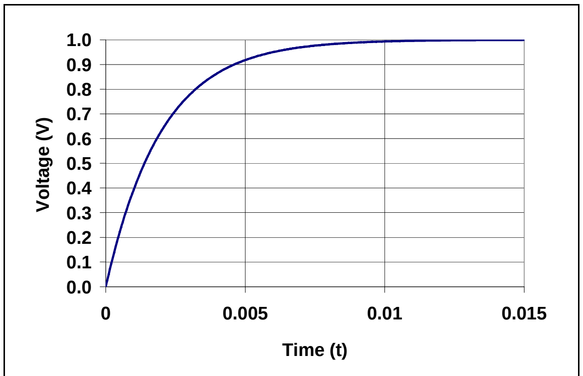

FIGURE 14: *Standard RC curve. The time required for the input signal to settle to within* ***x*** *ppm must be considered.*

The amount to which the internal charge must settle for absolute measurement accuracy (i.e., a system with no offset or gain adjustment can be defined as a percentage of the final charge). For example, if the target absolute accuracy percentage is 1 ppm, the following settling time must exist, represented by a multiple of the RC time constant (T).

EQUATION 4:

$$
\begin{aligned}
V_C &= V_F\left(1 - e^{\frac{-t}{\tau}}\right) \\
t &= n\tau \\
V_e &= 1 - \frac{V_C}{V_F} \\
V_C &= V_F(1 - e^{-n}) \\
e^{-n} &= 1 - \frac{V_C}{V_F} = V_e \\
n &= -ln(V_e) \\
&= -ln(1ppm) \\
&= 13.8
\end{aligned}
$$

In this example, for 1 ppm absolute accuracy, 14 time constants are required to complete the settling. Using Equation 4, the same calculation can be used for other accuracy requirements. Again, in calibrated systems where offset and gain errors are removed, the settling time analysis is not necessary due to the delta-sigma oversampling and averaging of each sample.

Communication Firmware

The MCP3551 ADCs are serial SPI™ devices. This application note includes code written in both C and assembly languages. The MCP3551 22-Bit Delta-Sigma ADC PICtail™ Demo Board connects with the DataView software through the PIC18F4550 via USB and is supplied with code written using Microchip's C18 compiler. An overview of the SPI communication protocol used is shown here:

```
void Read3551(char *data)
{
  unsigned char n;
    data[2] = ReadSPI();
    data[1] = ReadSPI();
    data[0] = ReadSPI();
}

 //MCU checks every 10 ms if conversion is
finished
  if(AquireData & gSampleFlag)
  {
    CS_PTBoard_LOW();           //
    for(n=0;n<5;n++);
    if (SDIpin == 0)
      Read3551(outbuffer);
    CS_PTBoard_HIGH();          //
    gSampleFlag = 0;            //clear
timeout indicator

    if(!mHIDTxIsBusy())
      HIDTxReport(outbuffer,3);
  }
```

As long as the system has been put into the Acquire Data mode from the DataView software (by sending an ASCII "S" via USB to the PIC18F4550), the `Acquire-Data` flag will be set. During this time, the MCP3551 is constantly converting, with the read data being sent back up to the PC via USB.

The code used on the PIC18F4550 sends $\overline{\text{CS}}$ low pulses to the MCP3551 every 10 ms. The `Timer1` flag and the `TimerCounter` variable are used to set this time. This low pulse effectively puts the device into Single Conversion mode, as the rising edge is less than the conversion time. During the $\overline{\text{CS}}$ pulse low time, the state of the SDO is tested to determine if the conversion is complete. If the pin is low, the firmware will retrieve the data using the `Read3551` subroutine. The `Read3551()` routine calls three separate `ReadSPI` routines and retrieves the three bytes of data containing the 22-bit word and the 2 overflow bits. Once the 3 bytes of data are returned, the `Sample` flag is reset and the process starts over.

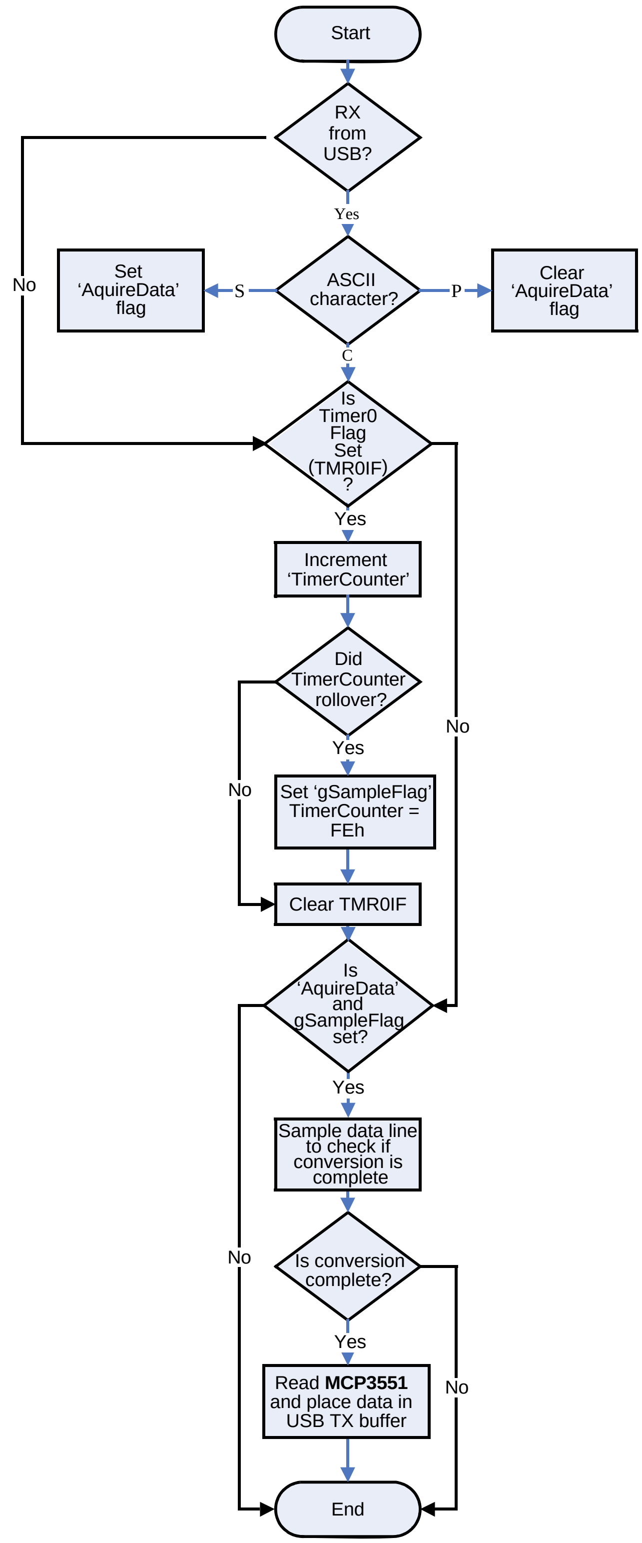

FIGURE 15: *PIC18F4550 flowchart. USB TX buffer is sent to the DataView® software for visual analysis.*

REFERENCES

[1] "Delta-Sigma Data Converters Theory, Design and Simulation", Steven R. Norsworthy, Richard Schreier, Gabor C. Temes, IEEE Press, 1997, pp. 4-9.

[2] AN9504, "A Brief Introduction to Sigma Delta Conversion", David Jarman, Intersil®, 1995.

[3] "Modern Business Statistics", Ronald L. Iman, W.J. Conover, Second Edition, John Wiley & Sons, Inc., 1989.

APPENDIX A: OVERSAMPLING ANALYSIS

The delta-sigma ADC is an oversampling device with many up-sides. High resolution, excellent line frequency rejection, limited external component requirements and low power are a few examples of its benefits. The high-resolution benefit is not a product of simple oversampling, which is sometimes confused.

WHY NOT JUST OVERSAMPLE WITH A PICmicro® MCU SAR ADC?

The answer is easy: noise shaping. Simply oversampling with a fast Successive Approximation Routine (SAR) ADC and averaging the results will not achieve the resolution performance of a delta-sigma ADC. Oversampling and averaging will only increase accuracy by 1/2 bit for each doubling of the sample frequency. The theory behind this comparison is presented here by comparing the noise power of both approaches.

SIMPLE OVERSAMPLING

For a generic quanitized unit (or LSB), the noise within this quanta is assumed to be entirely random, or assumed to be white-noise. Therefore, the quantization noise power and RMS quantization voltage for an digital or quantized output (ADC) can be given by the following equations:

EQUATION 5:

$$e^2 rms = \frac{1}{q}\int_{-\frac{q}{2}}^{\frac{q}{2}} e^2 de = \frac{q^2}{12} \quad (V^2)$$

$$^{e}rms = \frac{q}{\sqrt{12}} \quad (V)$$

For example, for a 16-bit converter with a V_{REF} of 5V, the RMS quantization noise would be 22 µV.

Taking this noise and folding it into the frequency band from 0 to fs/2, due to Nyquist, we can determine what the spectral density of the noise is in V/√(Hz):

EQUATION 6:

$$E(f) = (^{e}rms)\sqrt{\frac{2}{f_s}} \quad \left(\frac{V}{\sqrt{Hz}}\right)$$

To determine the noise power within a bandwidth of interest (f_0), we must now square, and then integrate, the noise over that bandwidth of interest.

EQUATION 7:

$$n_o^2 = (e(f)) \qquad (V^2)$$

$$n_o = (e_{rms})\left(\frac{2f_o}{f_s}\right)^{1/2} \qquad (V)$$

Where:

$f_o < f_s/2$

Recalling that $f_s/2f_o$ is the OSR, we now have the well established result that increasing the OSR reduced the noise by the square root of the OSR [1]. Therefore, each doubling of the sampling frequency only yields 3 dB better performance, or only 0.5 bits of resolution.

The delta-sigma modulator will increase the performance of oversampling by pushing the low-frequency noise towards the higher frequencies, see Figure 17. This benefit of delta-sigma modulation is referred to as noise shaping. A first-order delta-sigma modulator will increase accuracy by 9 dB, or 1.5 bits of resolution, for every doubling of the OSR.

The output of the accumulator is the input signal plus the error introduced by the quanitzation error, as well as the quantized signal, represented by the following figure and equation:

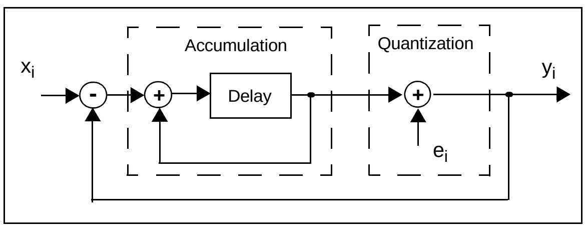

FIGURE 16: Representation of a first-order delta-sigma modulator in its sampled-data equivalent form [1].

EQUATION 8:

$$y_i = x_{i-1} + (e_i - e_{i-1})$$

Taking spectral density of the noise (e_i-e_{i-1}) and then again converting this to noise power by squaring it and integrating it over the bandwidth of interest eventually yields:

EQUATION 9:

$$n_o = e_{RMS}\frac{\pi}{\sqrt{3}}\left(\frac{2f_o}{f_s}\right)^{\frac{3}{2}}$$

The delta-sigma modulator decreases in-band noise by 9 dB (or 1.5 bits) for every doubling of the OSR. Three times better than simple oversampling.

Improving noise-shaping performance can be achieved using a higher-order delta-sigma modulator design. The noise power for higher-order modulators is summarized with the following equation:

EQUATION 10:

$$n_o = e_{RMS}\frac{\pi^M}{\sqrt{2M+1}}\left(\frac{2f_o}{f_s}\right)^{M+\frac{1}{2}}$$

In this case, the noise falls 3 (2M - 1) dB for every doubling of the sampling frequency for an Mth-order modulator. As an example, for M = 3 (MCP3551/3 devices are third-order modulators), for each doubling of the sampling frequency we have an increase in 2.5 bits of resolution. It is this architecture that allows < 3 µV of noise performance using devices such as the MCP3551/3. Figure 17 presents the noise-shaping in the frequency domain. The noise has been pushed to the higher frequencies, around the oversampling frequency (f_s). It should also be noted that thermal noise follows the standard averaging rule of 3 dB (1/2 bit) improvement with every doubling of the OSR, as it is taken and processed as part of the signal.

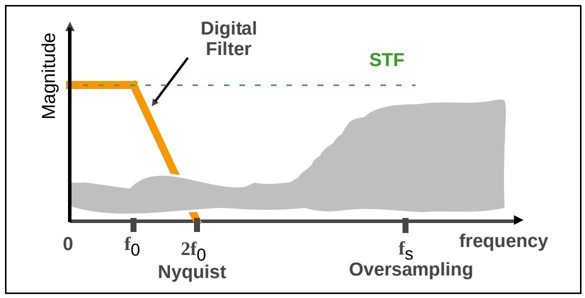

FIGURE 17: Noise-shaping from a delta-sigma modulator achieving lower noise floor in the bandwidth of interest. This is not possible by simply oversampling and averaging with a faster SAR ADC.

APPENDIX B: SOFTWARE - SPI™ COMMUNICATION IN C

TABLE B-1: SPI _ PIC18F252.ASM

```
;=============================================================================
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro(r) microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
; Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=============================================================================
;   Filename:          840DSM.asm
;=============================================================================
;   Author:            Craig L. King
;   Company:           Microchip Technology Inc.
;   Revision:          1.00
;   Date:              July 21, 2004
;   Assembled using MPASM(tm) WIN compiler
;=============================================================================
;   Include Files:     p18f252.inc    V1.3
;=============================================================================
;
;=============================================================================

        list p=18f252;list directive to define processor
        #include <p18f252.inc>;processor specific definitions

; Change the following lines to suit your application.

    __CONFIG           _CONFIG1H, _OSCS_OFF_1H & _HS_OSC_1H
    __CONFIG           _CONFIG2L, _BOR_ON_2L & _PWRT_OFF_2L
    __CONFIG           _CONFIG2H, _WDT_OFF_2H
    __CONFIG           _CONFIG3H, _CCP2MX_OFF_3H
    __CONFIG           _CONFIG4L, _STVR_OFF_4L & _LVP_OFF_4L & _DEBUG_OFF_4L
    __CONFIG           _CONFIG5L, _CP0_OFF_5L & _CP1_OFF_5L & _CP2_OFF_5L & _CP3_OFF_5L
    __CONFIG           _CONFIG5H, _CPB_OFF_5H & _CPD_OFF_5H
    __CONFIG           _CONFIG6L, _WRT0_OFF_6L & _WRT1_OFF_6L & _WRT2_OFF_6L & _WRT3_OFF_6L
    __CONFIG           _CONFIG6H, _WRTC_OFF_6H & _WRTB_OFF_6H & _WRTD_OFF_6H
    __CONFIG           _CONFIG7L, _EBTR0_OFF_7L & _EBTR1_OFF_7L & _EBTR2_OFF_7L & _EBTR3_OFF_7L
    __CONFIG           _CONFIG7H, _EBTRB_OFF_7H




;----------------------------------------------------------------------------
;Constants

SPBRG_VAL   EQU     .64            ;set baud rate 19.2 for 20Mhz clock

;----------------------------------------------------------------------------
```


TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
;Bit Definitions

GotNewData      EQU     0       ;bit indicates new data received

#define do      0               ;transmit bit
#define di      1               ;transmit bit
#define CLK     PORTC,2         ;clock
#define ADCS    PORTC,3         ;chip select
#define DIN     PORTC, 4        ;data in / READY
#define BITCOUNT 0x08



;-----------------------------------------------------------------------------
;Variables

                CBLOCK     0x000
                Flags           ;byte to store indicator flags
                RxData          ;data received
                TxData          ;data to transmit
                ParityByte      ;byte used for parity calculation
                ParityBit       ;byte to store received parity bit


                COUNT
                COUNT2
                COUNT3
                DLYCNT
                DLYCNT1
                DLYCNT2
                byte2
                byte1
                byte0

                ENDC

;-----------------------------------------------------------------------------
;This code executes when a reset occurs.

                ORG     0x0000  ;place code at reset vector

ResetCode:      bra     Main    ;go to beginning of program

;-----------------------------------------------------------------------------
;This code executes when a high priority interrupt occurs.

                ORG 0x0008      ;place code at interrupt vector

HighIntCode:    ;do interrupts here

                reset           ;error if no valid interrupt so reset

;-----------------------------------------------------------------------------
;This code executes when a low priority interrupt occurs.

                ORG     0x0018  ;place code at interrupt vector

LowIntCode:     ;do interrupts here

                reset           ;error if no valid interrupt so reset
;
```

TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
;               MAIN ROUTINE
;-----------------------------------------------------------------------------
;Main routine calls the receive polling routines and checks for a byte
;received. It then calls a routine to transmit the data back.
;
;
;This routine sets up the USART and then samples the MCP3551, and sends the 3 bytes out
;on the USART, THEN REPEAT
;
;-------------------------------------------------------------------------------



Main:           rcall   SetupSerial    ;set up serial port

                ;do other initialization here
                movlw   b'11010000'
                movwf   TRISC



MainLoop:


                ;go get the 3551 data



                rcall   Sample3551




                movff   byte2,TxData

                bsf     TXSTA,TX9D
                rcall   TransmitSerial ;go transmit the data

                movff   byte1,TxData


                bcf     TXSTA,TX9D
                rcall   TransmitSerial ;go transmit the data

                movff   byte0,TxData


                bcf     TXSTA,TX9D
                rcall   TransmitSerial ;go transmit the data


DoOtherStuff:;do other stuff here

                bra     MainLoop       ;go do main loop again

;-----------------------------------------------------------------------------
```

TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
;Check if data received and if so, place in a register and check parity.

ReceiveSerial: btfss   PIR1,RCIF      ;check if data received
               return                 ;return if no data

               btfsc   RCSTA,OERR     ;if overrun error occurred
               bra     ErrSerialOverr ;then go handle error
               btfsc   RCSTA,FERR     ;if framing error occurred
               bra     ErrSerialFrame ;then go handle error

               movf    RCSTA,W        ;get received parity bit
               movwf   ParityBit      ;and save
               movf    RCREG,W        ;get received data
               movwf   RxData         ;and save

               rcall   CalcParity     ;calculate parity
               movf    ParityBit,W    ;get received parity bit
               xorwf   ParityByte,F   ;compare with calculated parity bit
               btfsc   ParityByte,0   ;check result of comparison
               bra     ErrSerlParity  ;if parity is different, then error
               bsf     Flags,GotNewData ;else indicate new data received
               return

;error because OERR overrun error bit is set
;can do special error handling here - this code simply clears and continues

ErrSerialOverr:bcf     RCSTA,CREN     ;reset the receiver logic
               bsf     RCSTA,CREN     ;enable reception again
               return

;error because FERR framing error bit is set
;can do special error handling here - this code simply clears and continues

ErrSerialFrame:movf    RCREG,W        ;discard received data that has error
               return

;error because parity bit is not correct
;can do special error handling here - this code simply clears and continues

ErrSerlParity: return                 ;return without indicating new data

;-----------------------------------------------------------------------------
;Transmit data in WREG with parity when the transmit register is empty.

TransmitSerial:btfss   PIR1,TXIF      ;check if transmitter busy
               bra     $-2            ;wait until transmitter is not busy

               movf    TxData,W       ;get data to be transmitted


               movf    TxData,W       ;get data to transmit
               movwf   TXREG          ;transmit the data
               return

;-----------------------------------------------------------------------------
```

TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
;Calculate even parity bit.
;Data starts in working register, result is in LSB of ParityByte

CalcParity:    movwf   ParityByte         ;get data for parity calculation
               rrncf   ParityByte,W       ;rotate
               xorwf   ParityByte,W       ;compare all bits against neighbor
               movwf   ParityByte         ;save
               rrncf   ParityByte,F       ;rotate
               rrncf   ParityByte,F       ;rotate
               xorwf   ParityByte,F       ;compare every 2nd bit and save
               swapf   ParityByte,W       ;rotate 4
               xorwf   ParityByte,F       ;compare every 4th bit and save
               return

;-----------------------------------------------------------------------------
;Set up serial port.

SetupSerial:   movlw   0xc0               ;set tris bits for TX and RX
               iorwf   TRISC,F
               movlw   SPBRG_VAL          ;set baud rate
               movwf   SPBRG
               movlw   0x64               ;enable nine bit tx and high baud rate
               movwf   TXSTA
               movlw   0xd0               ;enable serial port and nine bit rx
               movwf   RCSTA
               clrf    Flags              ;clear all flags
               return

;**********************************************************
;**********************************************************
;--------------------------------------------------------------------------------------
;
;Sample3551
;
;This is where you sample the MCP3551 and put your 22-bit
;answer into the following bytes:

;       byte 2        --        byte 1      --      byte 0
;       MSB                                                 LSB
;
; This routine returns the data
;--------------------------------------------------------------------------------------
```


TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
Sample3551

    clrf    byte2                   ; reset input buffer
    clrf    byte1                   ; reset input buffer
    clrf    byte0                   ; reset input buffer


    bsf     CLK                     ; clock idle high
    bcf     ADCS                    ;INITIATE THE CONVERSION


    movlw   .6
    call    VAR1000TcyDELAY         ; delay 1ms

    bsf     ADCS                    ;CS HIGH (Single Conversion mode)

    movlw   .160                    ; total delay 110ms (GREATER THAN TCONV, CAN BE REDUCED)
    call    VAR1000TcyDELAY         ; delay 160k Tcy
    call    VAR1000TcyDELAY         ; delay 250k Tcy
    call    VAR1000TcyDELAY         ; delay 250k Tcy



    bcf     ADCS                    ; GET THE CONVERSION DATA

    movlw   BITCOUNT
    movwf   COUNT                   ; FIRST BYTE
FIRST_BYTE
    bcf     CLK                     ; drop clock for next bit
    bsf     CLK                     ; set clock to latch bit
    bcf     STATUS,C                ; pre-clear carry
    btfsc   DIN                     ; check for high or low bit
    bsf     STATUS,C                ; set carry bit
    rlcf    byte2, f                ; roll the carry bit left into position
    decfsz  COUNT, f                ; decrement bit counter
    goto    FIRST_BYTE              ; get next bit


    movlw   BITCOUNT
    movwf   COUNT                   ; SECOND BYTE
SECOND_BYTE
    bcf     CLK                     ; drop clock for next bit
    bsf     CLK                     ; set clock to latch bit
    bcf     STATUS,C                ; pre-clear carry
    btfsc   DIN                     ; check for high or low bit
    bsf     STATUS,C                ; set carry bit
    rlcf    byte1, f                ; roll the carry bit left into place
    decfsz  COUNT, f                ; decrement bit counter
    goto    SECOND_BYTE             ; get next bit
    movlw   BITCOUNT
    movwf   COUNT                   ; THIRD BYTE
THIRD_BYTE
    bcf     CLK                     ; drop clock for next bit
    bsf     CLK                     ; set clock to latch bit
    bcf     STATUS,C                ; pre-clear carry
    btfsc   DIN                     ; check for high or low bit
    bsf     STATUS,C                ; set carry bit
    rlcf    byte0, f                ; roll the carry bit left into place
    decfsz  COUNT, f                ; decrement bit counter
    goto    THIRD_BYTE              ; get next bit
    bsf     CLK                     ;clock idles high
```

TABLE B-1: SPI _ PIC18F252.ASM (CONTINUED)

```
    bsf     ADCS                       ;deselect A/D converter
    retlw   0                          ; We're finished - Return!

;****************** VARIABLE DELAY SUBROUTINES *******************
;           DLYCNT1 = F9h = 249d    DLYCNT2 = W
;           DELAY = T((4 DLYCNT1 + 4) DLYCNT2 + 4)
;
;   ex. To create a 300ms delay when using a 4Mhz osc, 300-250 = 50
;           movlw   .50                ;load .50 into WREG
;           call    VAR1000TcyDELAY    ;call VAR1000TcyDELAY = 50ms delay w/4MHz Osc
;           call    VAR1000TcyDELAY    ;call VAR1000TcyDELAY = 250ms delay w/4MHz Osc
;                                      ;total = 300ms delay
;************
; The value in W at the time of the CALL = x.  Delay = 1000Tcy*x
VAR1000TcyDELAY
            movwf   DLYCNT2            ;LOADS CONTROLLING DLY # INTO PRIMARY COUNTER
DLOOP2      movlw   .249               ;MAXIMIZES THE SECONDARY DLY COUNTER
            movwf   DLYCNT1            ;
DLOOP1      clrwdt                     ;or NOP
            decfsz  DLYCNT1,f          ; DECREMENT AND TEST SECONDARY LOOP FOR ZERO
            goto    DLOOP1             ; CONTINUE SECONDARY LOOP
            decfsz  DLYCNT2,f          ; DECREMENT AND TEST PRIMARY DLY COUNTER
            goto    DLOOP2             ; CONTINUE PRIMARY LOOP
            retlw   .250               ; preload W for the next CALL VAR1000TcyDELAY
;************
;  VARIABLE 5 Tcy DELAY UP TO 256*5Tcy+5Tcy
;           DLYCNT1 = W
;           DELAY = T(1 + 5 DLYCNT1 - 1) + CALL + RETLW
;
;   ex. To create a 250us delay, (250/5)-1 = 49
;           movlw   .49                ;load .49 into WREG
;           call    VAR5TcyDELAY       ; call VAR5TcyDELAY
;************
; The value in W at the time of the CALL = x.  Delay = 5*Tcy + 5Tcy
VAR5TcyDELAY
            movwf   DLYCNT1                ; LOADS CONTROLLING DLY # INTO PRIMARY COUNTER
DLOOP3      clrwdt                         ;or NOP
            nop
            decfsz  DLYCNT1,f              ; DECREMENT AND TEST ZERO
            goto    DLOOP3                 ; CONTINUE LOOP
            retlw   .250                   ; preload W for the next CALL VAR5TcyDELAY
;******************************************************************************




H
       END
```


APPENDIX C: HARDWARE - SPI™ COMMUNICATION IN ASSEMBLY

TABLE C-1: MCP3551.C

```
/*********************************************************************
 *
 *                Microchip USB C18 Firmware - MCP3551 PICtail(tm) Demo
 *
 *********************************************************************
 * FileName:        MCP3551.c
 * Dependencies:    See INCLUDES section below
 * Processor:       PIC18
 * Compiler:        C18 2.30.01+
 * Company:         Microchip Technology Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the Company) for its PICmicro(r) microcontroller is intended and
 * supplied to you, the Company’s customer, for use solely and
 * exclusively on Microchip PICmicro(r) microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN ”AS IS” CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author              Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Pat Richards       xx/xx/xx    Original.
 ********************************************************************/

/** I N C L U D E S **********************************************************/
#include <p18cxxx.h>
#include <usart.h>
#include <spi.h>
#include "system\typedefs.h"
#include "system\usb\usb.h"

#include "io_cfg.h"             // I/O pin mapping
#include "user\MCP3551.h"

/** V A R I A B L E S ********************************************************/
#pragma udata
//byte old_sw2,old_sw3;

BOOL emulate_mode;
rom signed char dir_table[]={-4,-4,-4, 0, 4, 4, 4, 0};
byte movement_length;
byte vector = 0;

byte AquireData = 1;  //0 = STOP; 1 = Aquire
char buffer[3];

char inbuffer[BUF_SIZE];            // 8 byte input to USB device buffer
char outbuffer[BUF_SIZE];           // 8 byte output to USB device buffer
byte TimerCounter = 0xF0;
static unsigned char gSampleFlag;
```

TABLE C-1: MCP3551.C (CONTINUED)

```
/** P R I V A T E  P R O T O T Y P E S ***************************************/
void Read3551(char *data);
unsigned char ReadSPI( void );
void CheckBoardConnect(void);

/** D E C L A R A T I O N S **************************************************/
#pragma code
void UserInit(void)
{
  byte i;

  CS_PTBoard_HIGH();            //Drive high
  tris_CS = 0; //Output
    OpenSPI(SPI_FOSC_16, MODE_11, SMPMID);

  TRISBbits.TRISB0 = 1;        //SDI
  TRISBbits.TRISB1 = 0;        //SCK

  //------------------------
  // initialize variables
  //------------------------
  for (i=0; i<BUF_SIZE; i++)   // initialize input and output buffer to 0
  {
    inbuffer[i]=0;
    outbuffer[i]=0;
  }

    //Timer 0
  TMR0H = 0;                   //clear timer
  TMR0L = 0;                   //clear timer
    T0CONbits.PSA = 0;        //Assign prescaler to Timer 0
    T0CONbits.T0PS2 = 1;      //Setup prescaler
    T0CONbits.T0PS1 = 1;      //Will time out every 51 us based on
    T0CONbits.T0PS0 = 1;      //20 MHz Fosc
    T0CONbits.T0CS = 0;       //Increment on instuction cycle
}//end UserInit

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

 * Function:        void ProcessIO(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side-Effects:    None
 *
 * Overview:        This function is a place holder for other user routines.
 *                  It is a mixture of both USB and non-USB tasks.
 *
 * Note:            None
 *****************************************************************************/
```

TABLE C-1: MCP3551.C (CONTINUED)

```
void ProcessIO(void)
{
  char n;
  // User Application USB tasks
  if((usb_device_state < CONFIGURED_STATE)||(UCONbits.SUSPND==1)) return;

  if (HIDRxReport(inbuffer, 1))   // USB receive buffer has data
  {
    switch(inbuffer[0])           // interpret command
    {
      case START_ACQUISITION:     // 'S' START aquisition of data
        AquireData = 1;
        CS_PTBoard_LOW();         //Start conversion
      TimerCounter = 0xFF;
      break;

      case STOP_ACQUISITION:      // 'T' STOP aquisition of data
        AquireData = 0;
      break;

      case CHANNEL_SELECTION:     // 'C' A/D Channel Selection
      break;

      default:                    // unrecognized or null command
        ;
    }// END switch(inbuffer[0])
  }//END if (HIDRxReport(inbuffer, 1)

  //Inst. cycle = 200 ns; TMR0IF sets every 51 us
  if(INTCONbits.TMR0IF)
  {
   TimerCounter++;
   if (!TimerCounter)      //if rolled over, set flag. User code will handle the rest.
   {
     TimerCounter = 0xFE;
     gSampleFlag = 1;
   }
   INTCONbits.TMR0IF = 0;
  }

  //MCU checks every 10 ms if conversion is finished
  if(AquireData & gSampleFlag)
  {
    CS_PTBoard_LOW();           //
    for(n=0;n<5;n++);
    if (SDIpin == 0)
      Read3551(outbuffer);
    CS_PTBoard_HIGH();          //
    gSampleFlag = 0;            //clear timeout indicator

    if(!mHIDTxIsBusy())
      HIDTxReport(outbuffer,3);
  }
}//end ProcessIO
```

TABLE C-1: MCP3551.C (CONTINUED)

```
/****************************************************************************
 * Function:        void Read3551(char *data)
 *
 * PreCondition:    None
 *
 * Input:           Pointer to a string; must be three bytes min
 *
 * Output:          None
 *
 * Side-Effects:    None
 *
 * Overview:
 *
 * Note:
 ***************************************************************************/
void Read3551(char *data)
{
  unsigned char n;
    data[2] = ReadSPI();
    data[1] = ReadSPI();
    data[0] = ReadSPI();
}

/****************************************************************************
 * Function:        CheckBoardConnect(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Overview:        Checks if the Stimulus Board is attached. Future expansion.
 *
 * Side-Effects:    None
/****************************************************************************/
void CheckBoardConnect(void)
{
}

/** EOF MCP3551.c********************************************************/
```

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

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, 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
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

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

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

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

ASIA/PACIFIC

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: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

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

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 - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

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

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

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 - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

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

Malaysia - Penang
Tel: 604-646-8870
Fax: 604-646-5086

Philippines - Manila
Tel: 632-634-9065
Fax: 632-634-9069

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 - Weis
Tel: 43-7242-2244-399
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: 34-91-352-30-52
Fax: 34-91-352-11-47

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

08/24/05