Enhancing ADC Performance on dsPIC33AK Devices

AN5971

Introduction

This application note provides guidance on enhancing the performance of the ADC in the dsPIC33AK family of devices using calibration and compensation techniques. The types of ADC errors and correction methods are discussed, and an overview of the built-in ADC calibration features is provided. This document provides the procedure and details for measuring and compensating gain error for improved accuracy.

1. Types of ADC Errors

The performance and accuracy of an ADC are defined in the Electrical Characteristics section of the device-specific data sheet. Typical accuracy characteristics include offset error, gain error, differential nonlinearity (DNL) and integral nonlinearity (INL). From these four characteristics, the absolute error can be calculated.

1.1. Test and Measurement Methodology

The test methodology and electrical characteristics are based on a linear ramp histogram. For the 12-bit ADC, the ideal transfer function is defined as:

$$floor\left(\frac{VIN}{VDD} \times 4096\right)$$

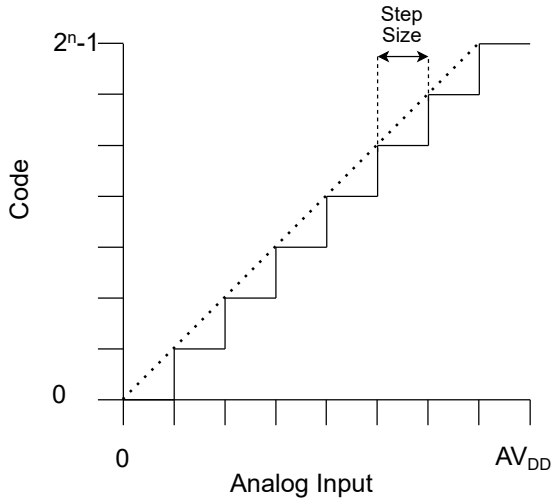
Each ADC count corresponds to 1/4096 of the input range.

When AV_{DD} is 3.3V nominal, the input range is 0V to 3.3V.

One ideal ADC count corresponds to a step size of $\frac{3.3V}{4096} = 0.805\,mV$.

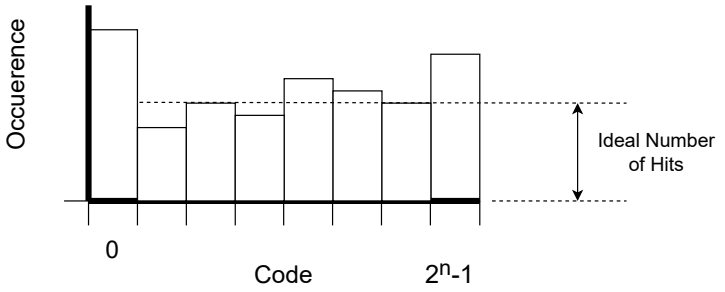
Figure 1-1illustrates the ideal transfer function. The floor() in the ideal transfer function yields an output code of 0 from 0V to one step size of the analog input. The maximum output code of 4095 corresponds to an analog input from AV_{DD} minus the step size to AV_{DD}.

Figure 1-1. Ideal Transfer Function



The linear ramp histogram method uses bins corresponding to each output code. Additional bins above and below the limits are included to account for errors. The two bins representing the ends of the actual transfer function are not included in the calculations. The lower-end bin can represent an offset error. An example of a histogram is shown in Figure 1-2. The number of hits in each bin is measured and compared to the ideal number of hits. The ratio of hits in each bin is then used to map the transfer function. Units for all specifications are in LSB.

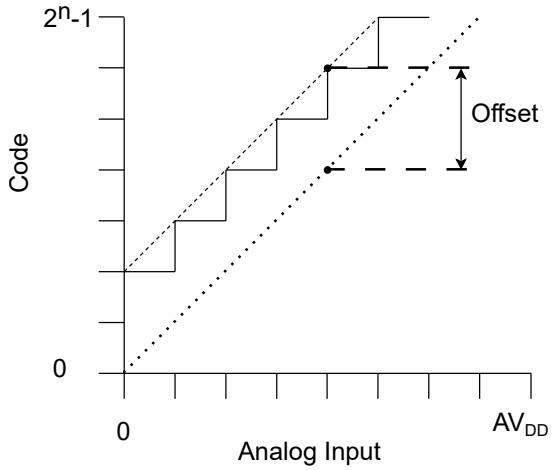
Figure 1-2. Histogram



1.2. Offset Error

Offset error is defined as the deviation of the transfer function's midpoint from the actual measurement compared to the ideal. Offset is calculated using histogram hits to determine the center point of the measured transfer function. The upper and lower 256 bins of the measured transfer function are excluded. The difference between the measured middle code and n/2 is the offset. The number of hits in the measured center bin provides resolution beyond 1 LSB. Offset error can be either positive or negative. Measuring negative offset requires a stimulus voltage below 0V. The offset error is corrected by adding or subtracting the offset error from the output. Figure 1-3 shows an example of a positive offset error.

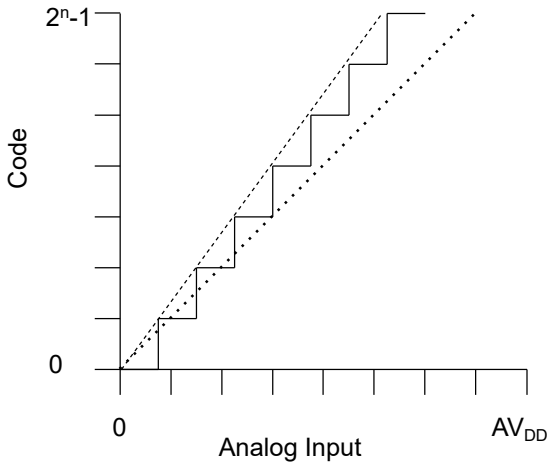
Figure 1-3. Positive Offset Error Example



1.3. Gain Error

Gain error is defined as the deviation of the measured transfer function slope from that of the ideal, after compensating for the offset error. The method used is a two-point linear intercept using histogram bins 256 and n-256. Gain error is corrected by scaling the output values. Figure 1-4 shows an example of positive gain error.

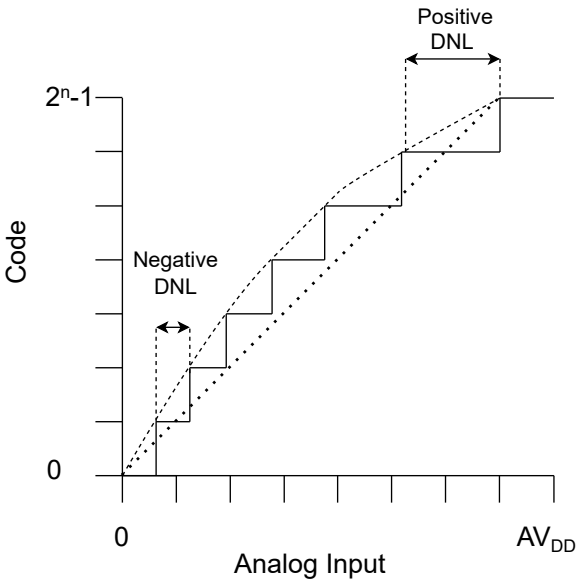
Figure 1-4. Positive Gain Error Example



1.4. Differential Nonlinearity (DNL)

DNL is defined as the difference in the step width between the measured transfer function and the ideal step width of 1 LSB. DNL is calculated after gain and offset have been corrected. Nonlinearity produces quantization step sizes with varying widths. DNL values defined in the Electrical Characteristics section of the device-specific data sheet represent the limits of DNL. Figure 1-5 shows an example of DNL step errors.

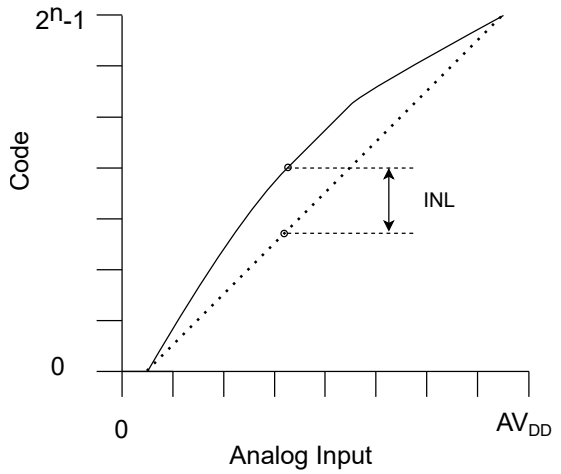
Figure 1-5. DNL Transfer Function Example



1.5. Integral Nonlinearity (INL)

INL is defined as the vertical difference between the measured and the linear transfer function. INL is calculated from the sum of DNLs that have already been corrected for gain and offset. INL specifications defined in the Electrical Characteristics section of the device-specific data sheet represent the limits of the INL curve deviation. Figure 1-6 shows an example of an INL error curve.

Figure 1-6. INL Transfer Function Example



2. dsPIC33AK ADC Built-In Calibration Features

The dsPIC33AK ADC has built-in calibration features, including configurable offset calibration. When enabled, the ADC will automatically perform a start-up calibration procedure, including gain and offset calibration. The automatically applied gain error correction may not meet the application's accuracy requirements. ADC performance can be enhanced using the gain error compensation described in the following section. During run time, fine offset calibration can be performed using two methods: either by software request or periodically using scheduling features.

For detailed information on offset calibration, refer to the ADC section of the device-specific data sheet.

3. Gain Error Compensation

After the ADC has run its automatic calibration procedure on start-up, the gain error may have shifted from that of the previous start-up calibration cycle. To enhance accuracy, the dsPIC33AK devices provide an internal voltage reference source to support run-time gain error compensation. After the automatic start-up calibration procedure is complete, the provided software gain calibration routine can be run once, and the compensation coefficient can be applied to the ADC results. The dsPIC33AK family of devices has more than one ADC instance, and each ADC instance needs to be calibrated independently.

3.1. Gain Compensation Scheme

The gain error compensation is based on a two-point linear correction scheme. Given that the ADC has already automatically performed offset calibration, the zero point is used in conjunction with a near full-scale reference voltage. An internal 15/16 * AV_{DD} calibration reference (CalrefH) is provided on one of the ADC inputs. It is recommended to use oversampling when measuring CalrefH to improve accuracy. Only one ADC instance can be connected to CalrefH at a time. The sampling time requirement for the CalrefH source is defined in the Electrical Characteristics section of the device-specific data sheet. Once converted, the compensation coefficient can be computed and stored in RAM. The compensation coefficient is then applied to the ADC results for the application. In summary, the procedure is as follows:

For each ADC instance:

1. Verify that the automatic calibration is complete by reading the ADRDY bit.
2. Select any ADC channel and configure the input for the CalrefH source.
3. Measure the CalrefH voltage using oversampling.
4. Compute the compensation coefficient and store it in RAM.
5. Apply the associated ADC compensation coefficient to the result data.

If an offset calibration is subsequently performed, the gain error calibration remains valid and does not need to be redone. However, if the device resets or the ADC is reset, the gain error procedure needs to be performed again.

3.2. Compensation Coefficient Calculation

The compensation coefficient equation is provided in Equation 3-1.

Equation 3-1. Gain Error Compensation Coefficient Calculation

$$Gain\ Error\ Compensation\ \ Coefficient = \frac{\frac{15}{16} \times 4096}{\frac{15}{16} \times \ CalrefH\_result} = \frac{3840}{CalrefH\_result}$$

3.3. Application of Compensation Coefficient

To compensate for gain error, each ADC result is multiplied by the calculated coefficient. The compensation can be performed using either fixed-point or floating-point multiplication. The floating-point calculation is shown in Example 3-1 and takes 21 instruction cycles. This results in up to 105 ns when the CPU is clocked at 200 MHz.

Example 3-1. ADC Conversion Result Correction Using Floating-Point Calculations

```
float coefficient = 3840.0/CalrefH_result; // needed to be done once for each
ADC instance
……
// takes 21 instruction cycles or 105 nS @ 200 MHz CPU clock
float corrected_result = (coefficient*((float)AD1CH0DATA));
```

The multiplication time can be reduced to 6 cycles (30 ns) when fixed-point calculation is used, as shown in Example 3-2. Note that the shift left by 18 is to scale up the coefficient to an integer value and minimize rounding errors. The corrected result can be handled in two ways: normalize (right-shift) back to a 12-bit value, or retain the additional LSBs of precision from the larger numerical format. Shifting (truncating) back to 12 bits incurs rounding errors that manifest as a DNL error in the result. To best utilize the improved accuracy of the compensation scheme, it is recommended to retain the additional precision; however, it must be accounted for in its subsequent usage. For simplicity and consistency with floating point, the truncation method is shown in the examples.

Example 3-2. 12-bit Unsigned ADC Conversion Result Correction Using Fixed-Point Calculations

```
int32_t coefficient = (uint32_t) ((1<<18)*3840.0/CalrefH_result; // needed to
be done once
// takes 6 instruction cycles or 30 nS @ 200 MHz CPU clock
uint32_t corrected_result = (coefficient*AD1CH0DATA)>>18;
```

3.4. Code Example

Example 3-3 contains all the necessary steps and information to perform gain error compensation for one ADC instance. In this example, channel 0 of ADC 1 is shown, but any channel can be used.

Example 3-3. Gain Error Compensation Code Example for dsPIC33AK Device

```
#include <xc.h>

int32_t result = 0;  // ADC conversion result output.
int32_t coefficient; // Gain compensation coefficient.

void OscillatorInitialization(); // Oscillator initialization procedure.

int main(){

   OscillatorInitialization();  // Initialize the oscillator.
   AD1CONbits.ON = 1; // Enable ADC.
   while(AD1CONbits.ADRDY == 0); // Wait when ADC will be ready/calibrated

   ////////////////////////////////////////////////////////////////////
   // GET A COEFFICIENT FOR THE GAIN ERROR COMPENSATION
   ////////////////////////////////////////////////////////////////////
   AD1CH0CONbits.MODE = 3; // Select oversampling mode
   AD1CH0CONbits.ACCNUM = 3; // 256 conversions
   AD1CH0CONbits.TRG1SRC = 1; // Software trigger will start a conversion
   AD1CH0CONbits.TRG2SRC = 2; // Back-to-back conversions
   AD1CH0CONbits.PINSEL = 14; // Select the AN14 input which is connected to 15/16 of AVDD
   AD1CH0CONbits.SAMC = 3; // Sampling time (6.5 TADs = 81nS @ 40MHZ ADC clock)
   AD1SWTRGbits.CH0TRG = 1; // Average 256 results of the reference voltage
   while(AD1STATbits.CH0RDY == 0); // Wait when the result is ready

   // Oversampling result is 16 Bit (has additional 4 bits)
   // Calculate the gain compensation coefficient
   // The coefficient is in fixed-point format (18 bits before point)
   coefficient = (int32_t)(3840.0*16.0*(1<<18)/AD1CH0DATA);

   ////////////////////////////////////////////////////////////////////
   // CONVERT AND COMPENSATE THE GAIN ERROR
   ////////////////////////////////////////////////////////////////////
   AD1CH0CON = 0; // Clean channel register for new settings
   AD1CH0CONbits.MODE = 0; // Select single conversion mode
   AD1CH0CONbits.TRG1SRC = 1; // Software trigger will start a conversion
   AD1CH0CONbits.PINSEL = 7; // Select the AN7 input for conversions
   AD1CH0CONbits.SAMC = 3; // Sampling time (6.5 TADs = 81nS @ 40MHZ ADC clock)

   // Trigger channel #1 in software and wait for the result
   while(1){
      AD1SWTRGbits.CH0TRG = 1; // Trigger channel # 1
      while(AD1STATbits.CH1RDY == 0); // Wait for a conversion ready flag
      // Read result. It will clear the conversion ready flag
      // The correction coefficient is in fixed-point format (18 bits before point)
      result = (coefficient*AD1CH0DATA)>>18;
      }
   return 1;
}

void OscillatorInitialization(){
   // Clock generator 6 should provide 320 MHZ to the ADCs
   PLL1CONbits.ON = 1;
   OSCCTRLbits.PLL1EN = 1;
```

```
    while(OSCCTRLbits.PLL1RDY == 0);
    PLL1CONbits.FSCMEN = 0; // disable clock fail monitor
    VCO1DIVbits.INTDIV = 1; // 1:2 = 320MHz
    PLL1DIVbits.PLLFBDIV = 80; // VCO = 640MHz
    PLL1DIVbits.PLLPRE = 1;
    PLL1DIVbits.POSTDIV1 = 4;
    PLL1DIVbits.POSTDIV2 = 1;
    PLL1CONbits.DIVSWEN = 1;
    while(PLL1CONbits.DIVSWEN == 1);
    PLL1CONbits.NOSC = 1; // FRC
    PLL1CONbits.OSWEN = 1;
    while(PLL1CONbits.OSWEN == 1);
    PLL1CONbits.FOUTSWEN = 1;
    while(PLL1CONbits.FOUTSWEN == 1);
    PLL1CONbits.PLLSWEN = 1;
    while(PLL1CONbits.PLLSWEN == 1);
    while(PLL1CONbits.CLKRDY == 0);
    CLK1CONbits.NOSC = 5; // PLL1
    CLK1CONbits.OSWEN = 1;
    while(CLK1CONbits.OSWEN == 1);
    while(CLK1CONbits.CLKRDY == 0);
    // ADC high speed clock (Generator 6), should be 320 MHz for 80MHz operation
    CLK6CONbits.ON = 1;
    CLK6CONbits.NOSC = 7; // PLL1 VCO divider
    CLK6CONbits.OSWEN = 1;
    while(CLK6CONbits.OSWEN == 1);
    while(CLK6CONbits.CLKRDY == 0);
}
```

4. Gain and Offset Compensation

For further improved performance, both gain and offset compensation can be done using a second non-zero voltage reference measurement. The zero point cannot be used due to the possibility of a negative offset that cannot be measured. An external resistor voltage divider can be used in conjunction with another ADC input for this purpose. The suggested reference is $AV_{DD}/16$ or below (CalrefL). Values below $AV_{DD}/16$ can provide additional robustness to component tolerances. However, care must be taken to ensure the reference voltage is not below the maximum absolute error so that the resulting ADC result is always at least one count. The compensation coefficients are calculated with Equation 4-1 using $1/16*AV_{DD}$ as CalrefL.

Equation 4-1. Two Reference Voltage Gain and Offset Error Calculation

$$Gain\ Error\ Compensation\ \ Coefficient\ = \frac{\frac{14}{16} \times 4096}{(CalrefH\_result - CalrefL\_result)}$$

$$Offset\ Error\ = \ CalrefL\_result\ \frac{\frac{1}{16} \times 4096}{Gain\ Error\ Compensation\ Coefficient}$$

5. Results and Conclusion

When the compensation techniques presented in this document are used, the accuracy of the dsPIC33AK ADC is enhanced beyond the capabilities provided by the automatic calibration features. Run-time compensation provides consistent gain error performance without the need for an external voltage reference. The recommended methods, equations and code examples provide a proven solution that can be integrated into applications that can benefit from enhanced performance.

6. Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
A	05/2025	Initial revision

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-1190-2

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.