dsPIC33AKのADC性能を向上させる方法

AN5971

はじめに

本書は、dsPIC33AKファミリのデバイスが備えるADCの性能を、校正および補償テクニックを使って向上させるためのガイダンスを提供します。本書にはADC誤差のタイプと補正方法に関する説明と、内蔵ADC校正機能の概要を記載しています。さらに、精度向上のためにゲイン誤差を計測して補償するための手順とその詳細を提供します。

1. ADC誤差のタイプ

ADCの性能と精度は、各デバイスのデータシートに記載されている「電気的特性」内で定義されています。標準的な精度特性にはオフセット誤差、ゲイン誤差、微分非直線性(DNL)、積分非直線性(INL)が含まれます。これら4つの特性から絶対誤差を求める事ができます。

1.1. 試験および計測方法

試験方法と電気的特性は、リニアランプ ヒストグラム法に基づきます。12ビットADCの理想的な伝達関数は下式により定義されます。

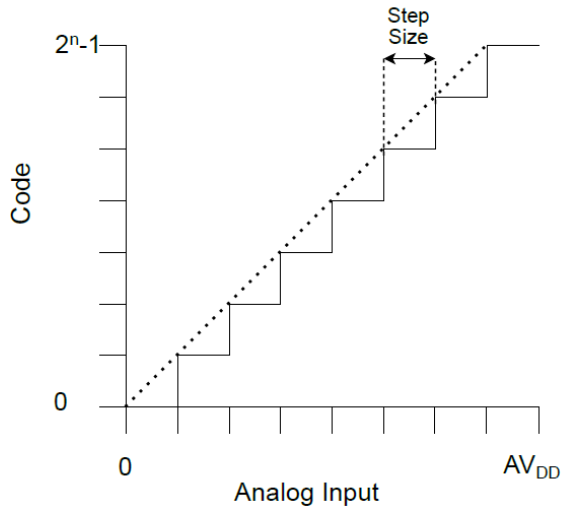
$$floor\left(\frac{VIN}{VDD} \times 4096\right)$$

各ADCカウントはAV_{DD}入力電圧の1/4096の電圧に対応します。AV_{DD}が3.3V (公称値)である場合、入力レンジは0～3.3Vです。

理想的には、1つのADCカウントは $\frac{3.3V}{4096} = 0.805\,mV$ のステップサイズに対応します。

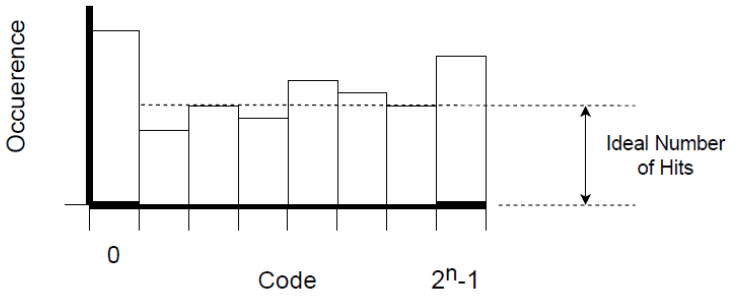
図1-1に、理想伝達関数を示します。上式内の floor() により、0V ～ [0V + ステップサイズ]のアナログ入力電圧から出力コード「0」が得られます。最大出力コードである「4095」は、[AV_{DD} - ステップサイズ] ～ AV_{DD}のアナログ入力電圧に対応します。

図1-1. 理想伝達関数



リニアランプ ヒストグラム法では、各出力コードに対応するビン(量子化区間)が使われます。この手法上、変換誤差に対応するため、上限の上と下限の下にもビンが用意されます。しかしこの2つのビンは特性評価ヒストグラムには使われません。また、下端のビンはオフセット誤差として表れる場合があります。ヒストグラムの例を図1-2に示します。各ビンのヒット数(変換結果がこのビン区間に入った数)がカウントされ、理想的なヒット数と比較されます。各ビンのヒット数の比率を使って実際の伝達関数がプロットされます。全てのADC精度仕様値の単位はLSBです。

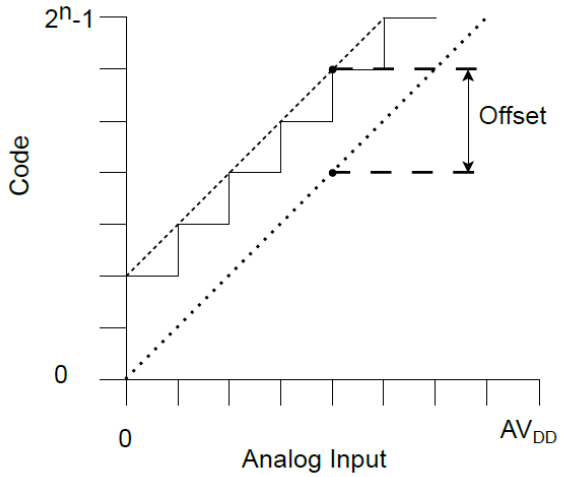
図1-2. ヒストグラム



1.2. オフセット誤差

オフセット誤差は、伝達関数の中央点の偏差(理想値に対する実際の計測値の偏差)として定義されます。オフセットを計算するため、ヒストグラムのヒット数を使って、計測された伝達関数の中央点が特定されます。計測された伝達関数の上端側と下端側の256個のビンは、オフセット誤差によるデータロスが含まれるため除外されます。計測された中央コードとn/2の間の偏差がオフセットです。計測された中央ビンのヒット数から1 LSBより小さな分解能でオフセット誤差が求まります。オフセット誤差は正または負の値を取ります。負のオフセットを計測するには、0V近傍で微調整可能な電圧が必要です。オフセット誤差は、出力に対してオフセット値を加算または減算する事により補正されます。図1-3に正のオフセット誤差の例を示します。

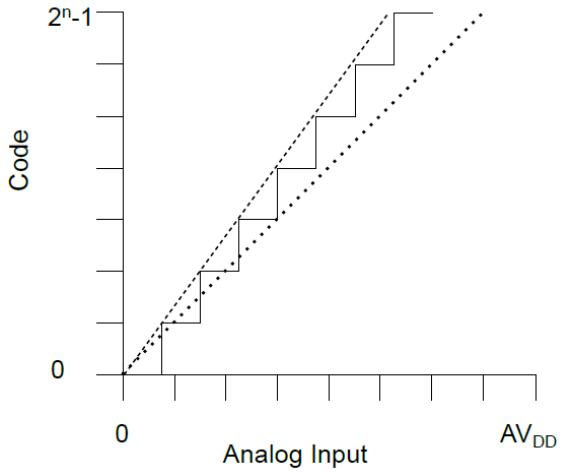
図1-3. 正のオフセット誤差の例



1.3. ゲイン誤差

ゲイン誤差は、オフセット誤差を補正した後の計測伝達関数と理想伝達関数の傾きの偏差として定義されます。これに使われる手法は、ヒストグラムのビン256とビンn-256を使った2点線形切片法です。
ゲイン誤差は出力値のスケーリングにより補正されます。図1-4に、正のゲイン誤差の例を示します。

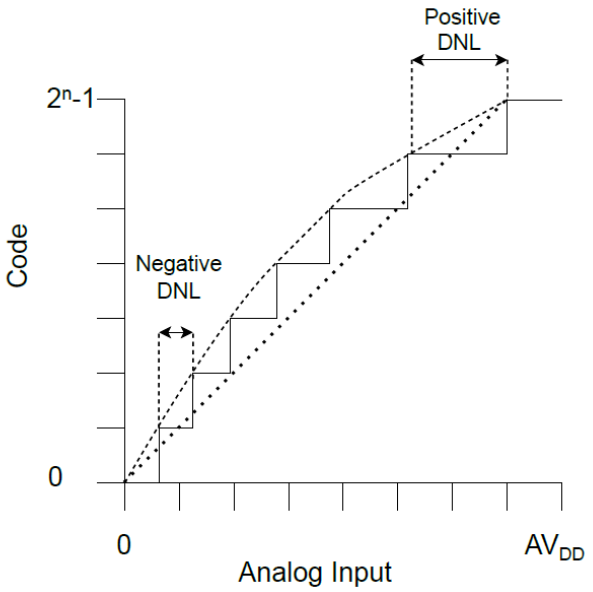
図1-4. 正のゲイン誤差の例



1.4. 微分非直線性(DNL)

DNLは、計測伝達関数と理想伝達関数の間の1 LSBステップ幅の偏差として定義されます。DNLは、ゲインとオフセットの補正後に計算されます。非直線性によって量子化ステップサイズは不均一になります。デバイス データシート内の「電気的特性」に記載されているDNL値はDNLの上限を示します。図 1-5に、DNLステップ誤差の例を示します。

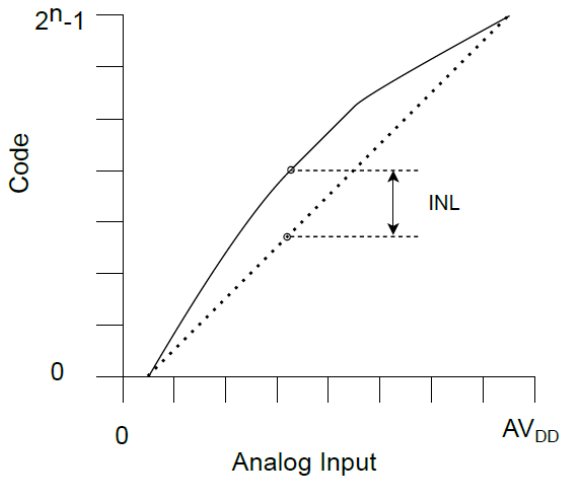
図1-5. 伝達関数のDNLの例



1.5. 積分非直線性(INL)

INLは、計測伝達関数と理想伝達関数の間の縦軸方向の偏差として定義されます。INLは、ゲインとオフセットが補正済みのDNLの累積から計算されます。デバイス データシートの「電気的特性」に記載されているINL仕様値は、INL曲線偏差の上限を示します。図1-6に、INL誤差曲線の例を示します。

図1-6. 伝達関数のINLの例



2. dsPIC33AK ADCの内蔵校正機能

dsPIC33AK ADCは校正機能(設定可能なオフセット校正を含む)を内蔵しています。校正機能を有効にすると、ADCは自動的に起動時校正処理(ゲインおよびオフセット校正を含む)を開始します。自動的に適用されるゲイン誤差補正ではアプリケーションの精度要件を満たせない場合があります。ADC性能は、次章の「ゲイン誤差の補償」に記載した方法で向上させる事ができます。下記の2通りの方法により、実行中に細かなオフセット校正を実行できます。

- 必要時にソフトウェアから実行を要求する
- スケジューリング機能を使って周期的に実行する

オフセット校正の詳細は、デバイス データシート内の「ADC」の項目を参照してください。

3. ゲイン誤差の補償

ADC起動時の自動校正処理を実行した後、温度変化等によりゲイン誤差が起動時から変化している場合があります。精度を向上させるため、dsPIC33AKは内部参照電圧源を入力として提供する事で、実行中のゲイン誤差補正をサポートします。起動時の自動校正処理が完了した後にソフトウェアによるゲイン校正 ルーチン(例3-3参照)を一度実行し、その補償係数を以後のADC結果に適用する事ができます。dsPIC33AKファミリのデバイスは複数のADCインスタンスを備えており、インスタンスごとに個別に校正する必要があります。

3.1. ゲイン補償スキーム

ゲイン誤差の補償は2点間直線補正スキームに基づきます。ADCが自動的オフセット校正を実行済みである場合、ゼロ点とフルスケール参照電圧より少し低い電圧(15/16倍の電圧)が使われます。内部のAV_{DD}を15/16倍した校正参照電圧(CalrefH)がADC入力の1つに提供されます。精度を上げるため、オーバーサンプリングを使ってCalrefHを計測する事を推奨します。CalrefHには一度に1つのADCインスタンスしか接続できません。CalrefH電圧のサンプリング時間要件は、デバイス データシートの「電気的特性」内で定義されています。CalrefHの変換後に、補償係数を計算してRAMに保存できます。この補償係数がアプリケーションのADC結果に適用されます。この手順を以下にまとめます。

これらの手順はADCインスタンスごとに実行する必要があります。

1. ADRDYビットを読み出す事により、自動校正が完了している事を確認します。
2. 任意のADCチャンネルを選択し、その入力をCalrefH電圧源向けに設定します。
3. オーバーサンプリングを使ってCalrefH電圧を計測します。
4. 補償係数を計算し、その値をRAMに保存します。
5. このADC補償係数を結果データに適用します。

この後にオフセット校正が実行されてもゲイン補償係数は有効なままであり、再実行する必要はありません。しかし、デバイスまたはADCがリセットされた場合、ゲイン誤差校正処理を再度実行する必要があります。

3.2. 補償係数の計算

補償係数の計算には式3-1を使います。

式3-1. ゲイン誤差補償係数の計算

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

3.3. 補償係数の適用

ゲイン誤差を補償するには、計算した係数を各ADC結果に乗算する必要があります。この計算は、固定小数点または浮動小数点の乗算を使って実行できます。浮動小数点計算のコードを例3-1に示します。この実行には21命令サイクルが必要です。CPUのクロックレートが200 MHzである場合、実行には最大105 nsを要します。

例3-1. 浮動小数点計算によるADC変換結果の補正

```
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));
```

固定小数点計算(例3-2)を使うと、乗算時間は6サイクル(30 ns)に削減できます。18ビット左シフトにより、係数を整数値へスケールアップする事で丸め誤差を最小化します。補正した結果は以下の2通りの方法で処理できます。
- 右シフトして12ビット値に戻す
- 例のようなint32_t型のままでLSBも切り詰めずに保持する

シフト(切り詰め)して12ビットに戻すと丸め誤差が生じ、結果にDNL誤差として表れます。補償スキームによる精度向上効果を最大限に活かすため、追加した数値精度を維持する事を推奨しますが、以後のデータの扱いに注意する必要があります。単純化と浮動小数点計算との一貫性のため、以下の例では切り詰め方式を使っています。

例3-2. 固定小数点計算による12ビット符号なしADC変換結果の補正

```
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. サンプルコード

例3-3に示すコードは、1つのADCインスタンス向けにゲイン誤差補償を実行します。これには必要な全ての手順と情報が含まれています。この例ではADC 1のチャンネル0を使いますが、任意のチャンネルが使えます。

例3-3. dsPIC33AK向けのゲイン誤差補償サンプルコード

```
#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. ゲインおよびオフセット両方の補償

さらに性能を向上させるため、第2の参照電圧(非ゼロ電圧)を使ってゲインとオフセットの両方の補償を実行できます。ゼロ点には計測できない負のオフセットが含まれる可能性があるため、ゼロ点は実データとして使わないようにします。このため、外付けの抵抗分圧回路を使って第2の参照電圧を提供します。この参照電圧(CalrefL)の推奨値はAV_{DD}/16以下です。AV_{DD}/16以下の電圧にする事で、部品許容差に対する堅牢性が向上します。常に1カウント以上のADC結果が得られるようにするため、この参照電圧は最大絶対誤差を下回らないよう注意する必要があります。補償係数は、CalrefL = 1/16 x AV_{DD}を使って式4-1により求まります。

式4-1. 2つの参照電圧を使ったゲイン/オフセット誤差の計算

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

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

5. 結果とまとめ

本書に記載した補償テクニックを使うと、dsPIC33AK ADCの精度は自動校正機能によって得られる精度からさらに向上させる事ができます。実行中の補償により、外部参照電圧を必要とせずに一貫したゲイン誤差性能が得られます。本書が推奨する手法、計算式、サンプルコードは、ADC性能の向上によって恩恵を受けるアプリケーション向けに、実証済みソリューションを提供します。

6. 改訂履歴

本書に適用された変更の履歴は以下の通りです。最新版から順にリビジョンごとに記載します。

リビジョン	日付	変更内容
A	2025年5月	本書は初版です。

Microchip社の情報

商標

ISBN: 979-8-3371-2522-0

法律上の注意点

Microchip社のデバイスコード保護機能

Microchip社製品のコード保護機能について以下の点にご注意ください。

- Microchip社製品は、該当するMicrochip社データシートに記載の仕様を満たしています。
- Microchip社では、通常の条件ならびに仕様に従って使った場合、Microchip社製品のセキュリティ レベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- Microchip社はその知的財産権を重視し、積極的に保護しています。Microchip社製品のコード保護機能の侵害は固く禁じられており、デジタル ミレニアム著作権法に違反します。
- Microchip社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip社が製品を「解読不能」として保証するものではありません。コード保護機能は常に進歩しています。Microchip社では、常に製品のコード保護機能の改善に取り組んでいます。