ADC 搭載 M5Stack / M5StickC で IoT センサ開発入門
補足資料

M5Stack の A/D 変換誤差を改善して高精度な電圧測定に挑戦してみた

国野 亘

本稿の内容

[1] プログラム example04_test.ino；A/D 変換誤差を測定しグラフ表示してみよう
[2] プログラム example05_ads1100；外付け A-D コンバータで高精度な電圧測定

本稿の最新版

最新版は下記に保存しています．予告なく修正する場合があります．

最新版：
https://bokunimo.net/git/m5adc/blob/master/READMEforEx4toEx5.pdf

関連ドキュメント

本稿は下記の IoT センサ開発入門の補足資料です．

ADC 搭載 M5Stack / M5StickC で IoT センサ開発入門：
https://git.bokunimo.com/m5adc/

ご注意

本稿で紹介するハードウェアやプログラムで製作した機器は，測定機並みの高精度なものも含まれていますが，計測機器ではありません．このため，本機で測定した電圧は，正式には目安値です．性能表示などに使用する場合は，トレーサビリティのとれた計測器で測定するか，本機の特性を正規の測定器で十分に補正したうえで使用してください．

[1] プログラム example04_test.ino；A/D 変換誤差を測定しグラフ表示してみよう

example04_test.ino は，M5Stack の D-A コンバータの出力を A-D コンバータに入力し，A-D コンバータの誤差をグラフ表示するプログラムです．D-A コンバータの出力 GPIO26 と A-D コンバータの入力 GPIO36 を，ジャンパ・ワイヤで接続し，サンプル・プログラム m5stack/example04_test.ino を書き込むと，各コンバータの入出力特性とその差(誤差)を液晶ディスプレイに表示します(写真 1)．M5StickC または M5StickC Plus の場合は，G26 と G36 をジャンパ・ワイヤで接続し，モデル名に合わせて m5stickc/example04_test.ino または m5stickcPlus/example04_test.ino を書き込んでください．

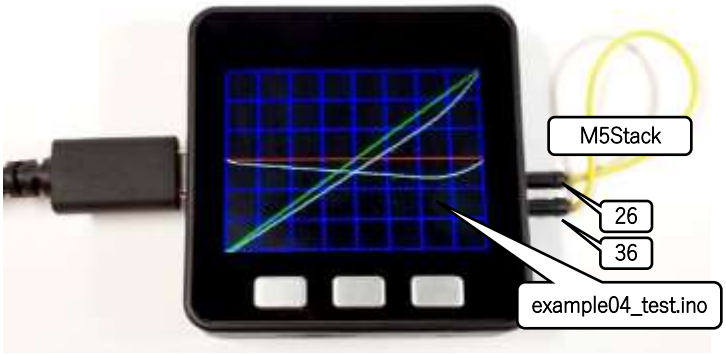

写真 1　M5Stack の液晶ディスプレイに A-D コンバータの誤差をグラフ表示する
GPIO26（G26）と GPIO36（G36）を，ジャンパ・ワイヤで接続し，D-A コンバータと A-D コンバータとの誤差をグラフ表示した

D-A コンバータの出力を使って A-D コンバータの特性を測る

液晶ディスプレイに表示されるグラフの横軸は D-A コンバータの出力値，縦軸は A-D コンバータの取得値と誤差です．但し，測定の基準となる横軸の D-A コンバータの誤差は考慮していません．正確な電圧源を制御して，横軸に用いれば，より正確に測定することも出来るでしょう．

図 1 に筆者が保有する M5Stack の A-D コンバータの特性例を示します．全体的に低い値を示し，グラフ左下 0mV と右上 3300mV 付近のみ誤差が少なくなる特性を示していました．また，D-A コンバータの出力値との最大差は約 350mV と大きいことも分かりました．なお，個体差もあるので，お持ちの M5Stack の特性とは異なる場合があります．

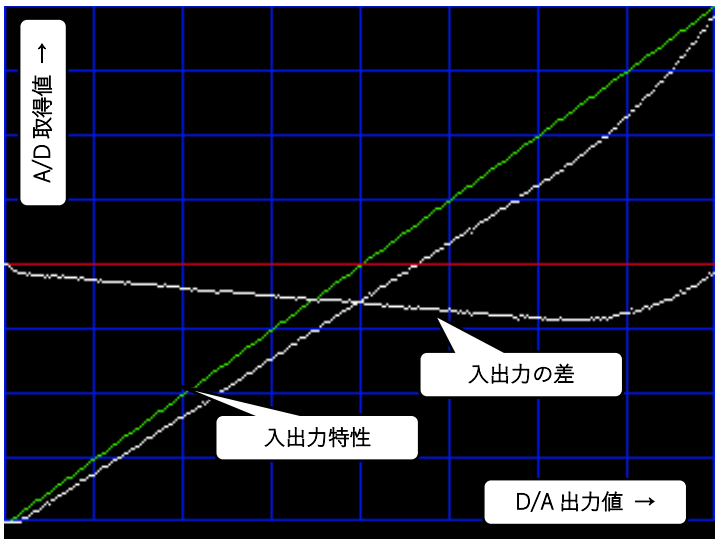

図 1　A-D コンバータの誤差のグラフ表示例
D-A コンバータの出力値を横軸に，A-D コンバータの入力値を縦軸にプロットした．誤差の最大値は約 350mV と大きいことが分かった

プログラム example04_test.ino の解説

以下に，M5Stack 用のサンプル・プログラム m5stack/example04_test.ino の D-A コンバータ部と，グラフへのプロット部の処理内容を説明します．グラフ座標以外は，M5StickC 用も同じです．

① D-A コンバータ（DAC2）の GPIO ピン番号 26 を，定数名 DAC_PIN として定義します．

② プロット座標の横軸の値となる変数 x について，x=0 から x=319 になるまで繰り返します．

③ D-A コンバータ出力用の変数 dac に，変数 x に応じた 8 ビットの範囲 0～255 の値を代入します．

④ D-A コンバータ出力コマンド dacWrite を使って，変数 dac の値を D-A コンバータに設定します．

⑤ A-D コンバータ入力値を取得し，変数 adc へ代入します．

⑥ 横軸に変数 x の値を，縦軸に A-D コンバータから取得した変数 adc の値を M5.Lcd.drawPixel に渡してグラフにプロットします．縦軸は 0～232 の範囲に制限し，また（液晶の下が座標値のプラスの為）最大値 232 から adc 値を減算することで，正の値を上向きにします．

⑦ D-A コンバータの出力値と A-D コンバータの入力値との差を求めます．分解能の違いにより，変数 adc は約 16 倍（4095/255 倍）になるので，16 で除算しました．また，液晶上側をプラス値に，センターラインの縦軸座標 116 を 0 にするために，符号を反転し，最大値 116 から減算しました．

⑧ 横軸に変数 x の値を，縦軸に処理⑦で求めた入出力の差の値をとる点にプロットします．

リスト 1 サンプル・プログラム m5stack/example04_test.ino （M5Stack 用）
ESP32 マイコン内蔵 A-D コンバータの誤差を液晶ディスプレイにグラフ表示する

```
#include <M5Stack.h>                          // M5Stack 用ライブラリ
#define DAC_PIN 26                            // GPIO 26 ピン(DAC2)
#define ADC_PIN 36                            // GPIO 36 ピン(ADC1_0)

void setup() {                                // 起動時に一度だけ実行する関数
    M5.begin();                               // M5Stack 用ライブラリの起動
    pinMode(DAC_PIN, OUTPUT);                 // GPIO26 を出力に
    pinMode(ADC_PIN, ANALOG);                 // GPIO36 をアナログ入力に
}

void loop() {                                 // 繰り返し実行する関数
    int dac, adc, x, y;                       // 変数 dac と adc,x,y を定義

    /* 罫線描画部 */
    M5.Lcd.fillScreen(BLACK);                 // LCD を消去
    M5.Lcd.drawRect(0, 0, 320, 232, BLUE);    // 座標 0,0 から 320x234 の箱を描画
    for(y = 0; y < 231; y += 29) M5.Lcd.drawLine(0,y,319,y, BLUE);  // 罫線 Y 描画
    for(x = 0; x < 319; x += 40) M5.Lcd.drawLine(x,0,x,231, BLUE);  // 罫線 X 描画

    /* グラフ作成部 */
    for(x = 0; x < 320; x++) {                // 変数 x=0～319 まで繰り返し
        dac = 255 * x / 319;                  // DAC 出力値(0～255)を設定
        dacWrite(DAC_PIN, dac);               // 変数 dac の値を DAC 出力
        adc = analogRead(ADC_PIN);            // ADC 値を adc へ代入
        M5.Lcd.drawPixel(x, 232 - 232 * dac / 255, GREEN);  // DAC 値をプロット
        M5.Lcd.drawPixel(x, 232 - 232 * adc /4095, WHITE);  // ADC 値をプロット
        y = 116 - adc / 16 + dac;             // DAC 出力と ADC 入力の誤差を計算
        M5.Lcd.drawPixel(x, 116, RED);        // 誤差 0 の値(80)をプロット
        M5.Lcd.drawPixel(x, y, WHITE);        // 誤差値をプロット
    }
    delay(1000);
}
```

A-D コンバータの誤差を補正する試作プログラム（M5Stack 用）

D-A コンバータの出力を基準値に A-D コンバータの補正を行うことも可能です. M5Stack のボタン操作で，補正無し（Mode=Raw），補正有（Mode=Cor），補正＋A-D コンバータのアッテネータ制御（Mode=Cor+Att）の 3 種類のグラフを表示する試作プログラム m5stack/example04_test_sd.ino を本サンプル・プログラム集に収録しました. 描画したグラフはスクリーンショットとしてマイクロ SD カードに保存することも出来ます．なお，この試作プログラムは M5Stack 版のみです.

GPIO26 と GPIO36 を，ジャンパ・ワイヤで接続した状態でプログラムを起動すると，自動で A-D コンバータの補正が行われ，ボタン操作で図 2 のように画面を切り替えることが出来ます.

表 1 の比較結果の一例では，補正有（Mode=Cor）により誤差を最大 20 分の 1 まで改善しました．さらに，A-D コンバータのアッテネータ制御（Mode=Cor+Att）を追加することにより，0mV 付近の低い入力レベルの変換精度を改善しました．照度センサなど，測定値と体感値が指数的に変化する測定で効果があると思います．なお，効果は一例であり，個体差により異なります.

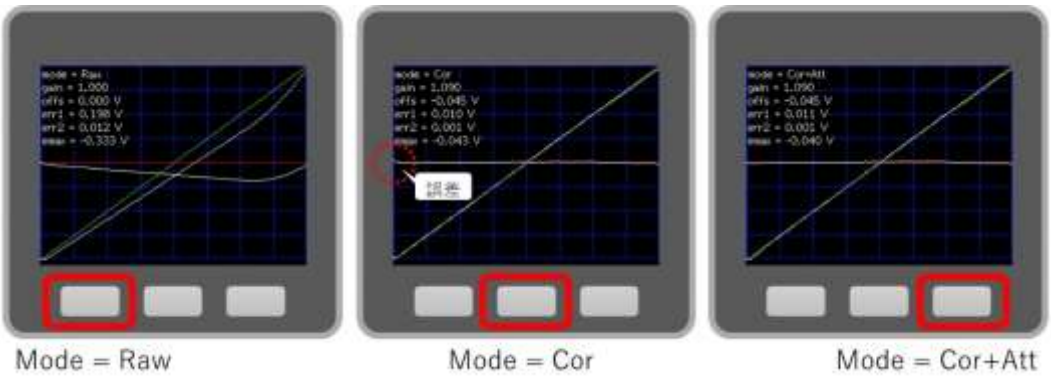

図 2　A-D コンバータの誤差を補正するプログラムの効果
M5Stack のボタン操作で，補正無し（Mode=Raw），補正有（Mode=Cor），補正＋A-D コンバータのアッテネータ制御（Mode=Cor+Att）の 3 種類のグラフを表示する

表 1　誤差を補正するプログラムの効果の一例（個体差あり）

実行モード	内容	平均誤差	平均二乗誤差	最大値
Mode=Raw	補正無し	198 mV	12 mV	333 mV
Mode=Cor	補正有	10 mV	1 mV	43 mV
Mode=Cor+Att	補正有＋ATT 制御	11 mV	1 mV	40 mV

コラム：A-D コンバータの誤差改善アルゴリズム

実行モード Mode=Cor では ESP32 内蔵 A-D コンバータの入力歪を 2 次多項式で補正しました．係数は筆者が所持している ESP32 マイコンに最適化してあります．個体差については，補正式をさらに 1 次式で変換する方法を用い，その係数（gain と offs）は起動時に自動的に実測して求めます．十分に補正しきれない場合は，Mode=Raw で得られた値（マイクロ SD カード内に adc0.csv として保存される）から，係数を求める方法（lib_mvAnalogIn.ino ファイル内の mvAnalogIn_correction 関数の係数で調整）や，D-A コンバータの出力値と A-D コンバータの読み値のリストをプログラム内で保持して，参照して補正する方法などがあります.

さらに，実行モード Mode=Cor+Att では，入力レベルが低い時の内蔵 A-D コンバータの減衰量を小さくして，A-D コンバータの分解能を疑似的に高めることで，誤差を改善しました.

[2] プログラム example05_ads1100；外付け A-D コンバータで高精度な電圧測定

精度の高い外付け A-D コンバータを使用する方法もあります. ここでは, M5Stack 社の純正 ADC Unit または Voltmeter Unit ， M5StickC 用 ADC HAT のいずれかを用い，高い精度での測定をしてみます.

写真 1　M5StickC と専用 A / D コンバータ ADC HAT
外付け A-D コンバータ ADC HAT（M5Stack の場合は ADC Unit）を接続し，測定精度を改善する
使用するプログラム：
example05_ads1100.ino: ADC Unit / HAT
example05_ads1115.ino: Voltmeter Unit

Texas Instruments 製 A-D コンバータ ADS1100

M5Stack 製 ADC Unit や ADC HAT には, 米 Texas Instruments 製 16 ビット A-D コンバータ ADS1100 が搭載されており，0V～約 12V までの直流電圧を高精度に測定することが出来ます（Voltmeter Unit は ADS1115 を搭載）. 16 ビット時のサンプリング・レートは約 8（SPS）なので，オシロスコープのような使い方は出来ませんが，精度を優先したい環境センサの A/D 変換に適したデバイスです.

ADS1100 の入力電圧の範囲は±3.3V（電源電圧）です. ADC Unit や ADC HAT では，図 1 のように，負入力を GND に接続し，正入力を分圧（1/4）してから ADS1100 に入力しているので，入力電圧の範囲は 0～13.2 V の正電圧です.

M5Stack / M5StickC との接続には，データ SDA とクロック SCL の 2 本の信号で通信が行える I²C インタフェースを使用します. Arduino 言語であれば，比較的簡単に ADS1100 から A/D 変換値を取得できるので，本稿では ADS1100 や ADS1115 用の専用ライブラリを使わずに取得します.

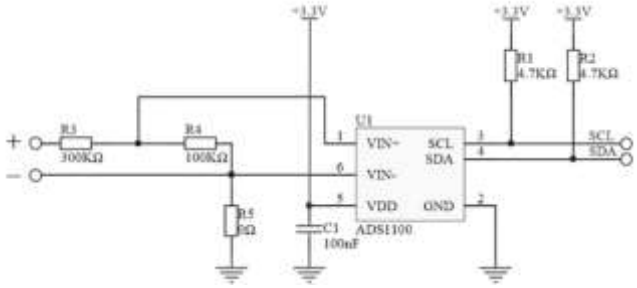

図 1 M5Stack 製 ADC HAT の回路図
A-D コンバータ ADS1100 を搭載し，0V～約 12V までの直流電圧を高精度に測定することが出来る. ADC Unit には他にインピーダンス変換器と電源 IC を搭載する
参考文献：
https://docs.m5stack.com/en/hat/hat-adc

サンプル・プログラム 2 example05_ads1100.ino の解説

以下に，M5StickC 用のサンプル・プログラム m5stickc/example05_ads1100.ino の I²C インタフェース通信部の処理について説明します. m5stack フォルダ内の M5Stack 用も同じように動作します.

① I²C インタフェース通信用ライブラリ Wire を本プログラムに組み込みます.

② Wire.begin は，I²C インタフェースの初期化設定を行う関数です. 括弧内の左側の第 1 引数はデータ SDA 信号，右側の第 2 引数はクロック SCL 信号の GPIO ポート番号です. M5Stack に ADC Unit を接続する場合は，引数不要で，SDA が GPIO21，SCL が GPIO22 に設定されます.

③ I²C インタフェースを使って，I²C アドレス 0x48 の ADC1100 に設定データ 0x0C を送信する処理部です. Wire.write の括弧内の設定データ 0x0C を 2 進数に変換すると 00001100 になります. 下 2 ビ

ット（00）は内蔵アンプの利得を，次の 2 ビット（11）は分解能 16 ビット・8SPS を，次の 1 ビット（0）は連続モードを示します．

④ ADC1100 から受信した A/D 変換値を保持するための 16 ビット整数型の変数 adc を定義します．ESP32 マイコンの int 型は 32 ビットですが，ADC1100 は 16 ビット値を応答するので，負の値を正しく取得するには，int16_t のように 16 ビットの変数に代入する必要があります．

⑤ I²C インタフェースを使ってデバイスからデータ受信を要求する Wire.requestFrom 命令です．ここでは，2 バイトのデータの受信要求を I²C アドレス 0x48 の ADC1100 に送信します．

⑥ 処理⑤の受信要求に応じて，ADC1100 から 2 バイト（16 ビット）のデータが得られたときに，Wire.read 命令を使って，受信データを 1 バイトずつ変数 adc に代入します．

なお，他の I²C デバイスでは，処理⑤の受信要求前に，処理③と同じ手順を使って，要求するデータのレジスタ番号（デバイス内のメモリのアドレス）を Wire.write 関数で送信する方法が一般的です．その一例として，ADS1115 を搭載した M5Stack 製 Voltmeter Unit 用の m5stack/example05_ads1115.ino も収録しました．Voltmeter Unit には I²C インタフェースの EEPROM が内蔵されており，工場出荷時の補正データが書き込まれています．この補正データを使うことで，より精度の高い測定が行えます．

リスト 2 サンプル・プログラム m5stickc/example05_adc1100.ino （M5StickC 用）

I²C 接続の外付け A-D コンバータから取得した値を液晶ディスプレイにメータ表示する

```
#include <M5StickC.h>                          // M5StickC 用ライブラリ
#include <Wire.h>                              // I2C 通信用ライブラリ

int range = 0;                                 // 自動レンジ用

void setup(){                                  // 起動時に一度だけ実行する関数
    M5.begin();                                // M5StickC 用ライブラリの起動
    Wire.begin(0,26);                          // I2C を初期化(GPIO 0, GPIO 26)
    Wire.beginTransmission(0x48);              // ADS1100(0x48)との通信を開始
    Wire.write(0x0C);                          // 連続,16bit-8SPS,Gain=1 を設定
    Wire.endTransmission();                    // ADS1100(0x48)との通信を終了
    M5.Axp.ScreenBreath(7 + 3);                // LCD の輝度を 3 に設定
}

void loop(){                                   // 繰り返し実行する関数
    int16_t adc = 0x0000;                      // 変数 adc を定義
    float mv;                                  // 浮動小数点数型変数 mv を定義
    Wire.requestFrom(0x48, 2);                 // 2 バイトのデータを要求
    if(Wire.available() >= 2){                 // 2 バイト以上を受信
        adc = ((int16_t)Wire.read()) << 8;     // 1 バイト目を変数 adc の上位へ
        adc |= (int16_t)Wire.read();           // 2 バイト目を変数 adc の下位へ
    }
    mv = 4. * (float)adc * 3300. / 32768.;     // ADC 値を電圧に変換して mv へ代入
    Serial.printf("adc=%d, mv=%f¥n", adc, mv); // ADC 値と電圧値 mv をシリアル出力

    if((mv <= 3500. && range != 4) || range == 0){  // 電圧値が 3500mV 以下時
        range = 4;                                  // レンジを 4(V)に
        analogMeterInit("mV", "ADS 4V", 0, 4000);   // メータのレンジ設定
    }
                    ～～～ 一部省略（メータのレンジ切替）～～～

    analogMeterNeedle(mv);                     // ADC の電圧値をメータ表示
    delay(100);                                // 0.1 秒（100ms）の待ち時間処理
}
```