Int. J. Microgravity Sci. Appl. Vol. 30 No. 3 2013 (152–158)

IIIII 技術解説講座 IIIII

航空機実験用加速度スイッチの作り方（第三回）
初心者のためのマイコン電子工作講座

夏井坂　誠

How to Make an Acceleration Switch for Parabolic Flight Experiments (No. 3)
A Lecture for a Beginner to Make a Microcomputer-Controlled, Electronic Device

Makoto NATSUISAKA

Abstract

The lecture introduces how to make a microcomputer-controlled, electronic device for a beginner. A series of lectures provides not only how to measure a physical property with an electronic sensor, convert it to a digit (analog digital conversion), switch on and off an electronic circuit with FET (Field Effect Transistor), and control those with a microcomputer but also practical know-how to design an actual electronic circuit, choose appropriate electronic parts, and mount those to a PCB (printed-circuit board), with explaining how to make "an acceleration switch". The switch can automatically turn on and off a connected device according to an acceleration level measured with an acceleration sensor and contribute to parabolic flight experiments through size reduction of an apparatus, less operation, and precise control of the experiments.

Keyword(s): acceleration, microcomputer, sensor, FET, parabolic flight

1. はじめに

前回はマイコン dsPIC30F2012 のプログラミング方法として，統合開発環境のインストール，LED 点滅プログラムのコーディング，ソースコードの HEX ファイルへの変換，HEX ファイルの dsPIC30F2012 へのダウンロード方法をご紹介しました．今回は，加速度センサの出力電圧を A/D（Analog/Digital）変換して，dsPIC30F2012 に取り込む方法，ならびに，その値に応じて出力制御を行う方法をご紹介します．

2. マイコンによる A/D 変換

2.1 はじめに

今回使用するサンハヤト（株）製 3 軸加速度センサモジュール MM-2860 の計測データはアナログ電圧で出力されます．この他のセンサでも，電子式であれば，その大半はアナログ電圧を出力します．（出力が電荷や電流で出力されるセンサもありますが，そのような場合も I/V 変換回路などを組み合わせて電圧に変換して取り扱います．）しかし，マイコン内部では，0，1 で表されるデジタル・データしか扱うことができませんので，これらセンサ出力をマイコンで処理する場合は，デジタル値に変換する作業が必要になります．この変換作業を，A/D 変換と呼びます．幸いなことに，dsPIC30F2012 には，10 ビットと 12 ビットの A/D 変換機能が内蔵されていて，センサを接続するだけで A/D 変換を行うことができます．（A/D 変換機能が内蔵されていないマイコンの場合は，A/D 変換専用 IC を外付けする必要があります．）

2.2 dsPIC30F2012 の A/D 変換機能

ここで，dsPIC30F2012 の A/D 変換機能を説明します．dsPIC30F2012 は，10 ビットまたは 12 ビットの A/D 変換が可能です（プログラム時にソフトウェア的に選択します）．10 ビットとか 12 ビットと言っているのは，入力されたアナログ電圧を，どれくらいの細かさ（分解能）でデジタル値に変換するかを表しています．**Fig. 1** の通り，10 ビットの場合，換算対象の電圧幅（VrefH-VrefL）を 2^{10}=1024 分割，12 ビットの場合 2^{12}=4096 分割した電圧

宇宙航空研究開発機構　宇宙科学研究所　ISS 科学プロジェクト室　〒305-8505　茨城県つくば市千現 2-1-1
Japan Aerospace Exploration Agency (JAXA), ISS Science Project Office, Institute of Space and Astronautical Science (ISAS), 2-1-1 Sengen, Tsukuba, Ibaraki 305-8505, Japan
(E-mail: natsuisaka.makoto@jaxa.jp)

Δ（LSB: Least Significant Bit）を単位としてデジタル値に換算します．12 ビットの方が 10 ビットに比べて 4 倍細かくデジタル化できるということがおわかりになるかと思います．また，当然のことながら，デジタル値はΔを単位として，とびとびの値しか取れないので，A/D 変換に伴い変換誤差（「量子化誤差」）を伴います．

dsPIC30F2012 の A/D 変換機能は，全てのアナログ入力ピン（AN0～9）で使用可能です．ただし，1 つの A/D 変換器を，9 つのアナログ入力ピンで共用する（マルチプレクサで切り替えて使用する）ので，処理はシーケンシャルなものとなります．

換算電圧範囲は，プログラミング時に，「_ADCON2」レジスタ（後述）の VrefH に上限の参照電圧を，VrefL に下限の参照電圧を設定して，指定します．VrefH=AV_{DD}（電源電圧：5V），VrefL=AV_{SS}（グランド：0V）とすれば，0~5V となります．また，電圧の変動幅がこれより小さい場合は，せっかくの分解能が損なわれますので，Vref+ピン，Vref-ピン双方もしくはいずれかに参照電圧を入力させ，これらを VrefH，VrefL 双方もしくはいずれかに設定することによって，換算電圧範囲を最適化します．

ところで，**Fig. 1** のように経時的に変化する信号を取り込みたい場合は，一定の時間間隔（サンプリング・インターバル）で A/D 変換を繰り返すことになります．アナログ信号を精度良く取り込むためには，サンプリング・インターバルを短くした方が良いのですが，A/D 変換処理は，入力電圧を一度内蔵のホールド・コンデンサに読み取り，これをデジタル変換するといった多段の処理となっているために，ある時間を必要とし，限りなく小さくすることはできません．このため，10 ビット A/D 変換では 500ksps（sampling per second）が，12 ビット A/D 変換では 100ksps が処理速度の上限となります．（ホールド・コンデンサの充電は，入力電圧，入力インピーダンス，温度などに依存するので，上記の速度を実現するには，入力電圧を高くする，入力インピーダンスを小さくする，使用温度を低く抑えるなど，入力回路の構成や使用環境を最適化する必要があります．）

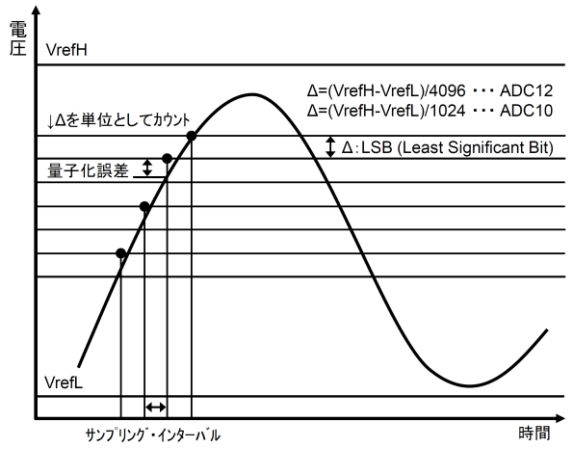

Fig. 1　A/D Conversion

先に述べた通り，12 ビット A/D 変換は 10 ビット A/D 変換に比べて，読み取り電圧の分解能を 4 倍高くできます．他方，処理速度の方は 10 ビットの方が 5 倍速いことがわかります．つまり，どちらのモードを使用するかは分解能を取るか，処理速度を取るかで決めることになります．

2.3　dsPIC30F2012 における A/D 変換機能の使い方

以下，今回使用する 12 ビット A/D 変換の手順を説明します．（本項は，トランジスタ技術 2007 年 9 月号 [1]の特集「新生 PIC マイコン・トレーニング」第二章の小野寺康幸さんの記事を参考にしています．記事中には，より詳しい解説が書かれています．また，サンプル・プログラムも提供されていますので，そちらも参照してください．なお，サンプル・プログラムを使用する際は，Readme テキストにも目を通すようにして下さい．）

【手順 1】　オブジェクト・ファイルの追加

12 ビット A/D 変換機能を使用する場合には，「libp30F2012-*omf*.a」（*omf* は coff または elf）というオブジェクト・ファイルをプロジェクトの Libraries に追加する必要があります．（具体的な手順は 3.項で説明します．）このオブジェクト・ファイルには，12 ビット A/D 変換に必要となる様々な処理（関数）が，バイナリ・コードとして書き込まれています．上記の追加作業を行うだけで，複雑な A/D 変換処理をいちいち記述することなく，A/D 変換が行えるようになっています．

【手順 2】　ヘッダ・ファイルの追加

次に，12 ビット A/D 変換に必要な関数を使えるように，ソース・コードにヘッダ・ファイル「adc12.h」の追加指示を書き込みます．ソース・ファイルの「#include <p30f2012.h>」の下に，

```
#include <p30f2012.h>
#include <adc12.h>
```

のように，一行追記して下さい．

【手順 3】　制御変数の宣言

さらに，以下の通り，A/D 変換用制御変数の宣言を書き加えます．

```
// A/D control register

unsigned int _ADCON1;
unsigned int _ADCON2;
unsigned int _ADCON3;
unsigned int _ADCHS;
unsigned int _ADPCFG;
unsigned int _ADCSSL ;
```

dsPIC30F2012 の A/D 変換機能には，分解能（10 ビット，12 ビット）以外にも，いくつか設定可能な項目（基準電圧，アナログ入力ピン，入出力モード，発振器選択等）があり，A/D 変換動作をきめ細かく設定可能となっ

ています．実際の設定作業は次の手順で行いますが，ここではその作業に先だって，必要となる変数の宣言を行っています．

【手順 4】　A/D 変換モードの設定

ここで，A/D 変換の動作を設定します．A/D 変換に限らず dsPIC30F2012 の動作設定は，レジスタと呼ばれる特定の内部メモリの，特定のビットまたはビット列に，特定の値を書き込むことによって行われます．この設定作業では，レジスタまたはレジスタ・ビットに二進数や十六進数を代入していくことになるのですが，幸いなことに，C 言語でプログラムする場合は，レジスタやレジスタ・ビット，さらには設定値に名前が付けられているので，以下の通り，わかりやすい記述が可能となっています．

```
// A/D 変換モジュールをオフにします・・・(1)
ADCON1bits.ADON=0;
// A/D 変換モジュールを初期化します・・・(2)
_ADCHS=ADC_CH0_POS_SAMPLEA_AN0          &
        ADC_CH0_NEG_SAMPLEA_NVREF;
SetChanADC12(_ADCHS);
ConfigIntADC12(ADC_INT_DISABLE);
_ADCON1=ADC_MODULE_ON                   &
        ADC_IDLE_CONTINUE               &
        ADC_FORMAT_INTG & ADC_CLK_AUTO &
        ADC_AUTO_SAMPLING_OFF           &
        ADC_SAMP_OFF;
_ADCON2=ADC_VREF_AVDD_AVSS              &
        ADC_SCAN_OFF                    &
        ADC_SAMPLES_PER_INT_1           &
        ADC_ALT_BUF_OFF                 &
        ADC_ALT_INPUT_OFF;
/*  Tad={Tcy(ADCS+1)}/2>334ns,  Then  ADCS>18.7,
Tad=10*Tcy */
_ADCON3=ADC_SAMPLE_TIME_1               &
        ADC_CONV_CLK_SYSTEM             &
        ADC_CONV_CLK_10Tcy;
_ADPCFG=ENABLE_AN0_ANA;
_ADCSSL=SCAN_NONE;
OpenADC12(_ADCON1,  _ADCON2,  _ADCON3,
_ADPCFG, _ADCSSL);
```

(1)では，A/D 変換モジュールをオフにしています．これは A/D 変換中に，次の A/D 変換を始めてしまわないようにするためです．

(2)では，制御レジスタ（_ADCON1~3，_ADCHS，_ADPCFG，_ADCSSL）に値をセットし，ADC12 関数（SetChanADC12()，ConfigIntADC12()，OpenADC12()）で，A/D 変換の動作設定を行っています．（制御レジスタの詳細を知りたい方は文献 3)を，ADC12 関数の詳細を知りたい方は，文献 4)を参照して下さい．）ここで，ポイントは，赤字（AN0）で示した部分で A/D 変換を行うポートの指定を行っている点です．AN0 以外のポートを使いたい場合は，ここを書き換えて下さい．

【手順 5】　A/D 変換の実行

A/D 変換の実行はソースファイルの該当箇所に以下を書き込みます．

```
// 自動サンプリングを開始します・・・(3)
ADCON1bits.SAMP=1;
// A/D 変換の完了を待ちます・・・(4)
while(BusyADC12());
// 12 ビットデータを読み出します・・・(5)
ResultDataGz=ReadADC12(0);
```

(3)，(4)で A/D 変換の自動実行を行っています．A/D 変換の処理状況は BusyADC12()が教えてくれます．処理中は 1 が，終了時は 0 が戻り値として返されますので，処理中は待機ということになります．

(5)では，変数 ResultDataGz に変換結果を読み込んでいます．なお，ここでは結果を読み込む変数として ResultDataGz を使っていますが，好きな変数名を使って構いません．ただし，ソースコードの冒頭で，変数宣言する必要があります．

3.　試してみよう

3.1　デモ回路の動作

今回は，3 軸加速度センサモジュール MM-2860（サンハヤト（株）製）の Z 軸方向の加速度出力電圧を AN0 ポートに取り込み，12 ビット A/D 変換を実施し，Z 軸方向の加速度が 0.5G を切る場合は，RB6 につないだ赤色 LED を点灯，0.5G を超える場合は，RB7 につないだ緑色 LED を点灯させ，一定時間待機の後，再度加速度を計測・・・という手順を繰り返すことにします（**Fig. 2**）．

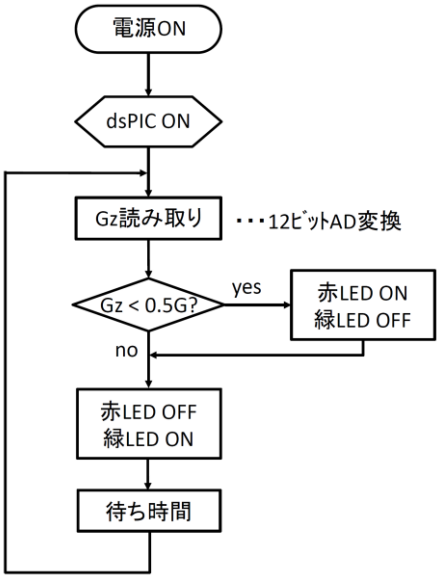

Fig. 2　Flow Chart of the Demonstration Program

3.2　デモプログラム

第二回の 3.2 項を参考に MPLAB X IDE を起動してプロジェクトを作成して下さい．過去にプロジェクトを作成している場合，右上の Projects ペインに古いプロジェクトが残ったままになっていますので，プロジェクト名を右クリックして Close を選択した上で，新しいプロジェクトを作成して下さい．なお，プロジェクト名は好きに付けていただいて結構です．リンカ・スクリプト・ファイル「p30F2012.gld」を追加して，新規ソース・ファイルを追加したら，そこに以下を記述して下さい．

```
---------------------------------------------------------------------------
#include "xc.h"

// Part1（ヘッダ・ファイルの追加指示）
#include <p30f2012.h>
#include <adc12.h>

// Part2（マイコンの動作設定）
// configuration
// _FWDT(WDT_OFF);
// _FGS(CODE_PROT_OFF);
// _FOSC(CSW_FSCM_OFF & FRC_PLL16);
// _FBORPOR(PBOR_OFF & PWRT_64 & MCLR_EN);

// Part3（A/D 変換の制御変数の型宣言）
// A/D control register
unsigned int _ADCON1;
unsigned int _ADCON2;
unsigned int _ADCON3;
unsigned int _ADCHS;
unsigned int _ADPCFG;
unsigned int _ADCSSL ;

// Part4（プログラムで使用する変数の型宣言）
// data
unsigned int ResultDataGz;  // 加速度 A/D 変換結果
unsigned int i;  // 待ち時間カウンタ
unsigned int j;  // 待ち時間カウンタ

// Part5（メイン・プログラム）
int main(void){

// Part6（無限ループ）
while (1){

// Part7（A/D 変換の準備）
// ADC
// A/D 変換モジュールのオフを確認します
ADCON1bits.ADON=0;
// A/D 変換モジュールを初期化します
_ADCHS=ADC_CH0_POS_SAMPLEA_AN0 &
        ADC_CH0_NEG_SAMPLEA_NVREF;
SetChanADC12(_ADCHS);
ConfigIntADC12(ADC_INT_DISABLE);
_ADCON1=ADC_MODULE_ON &
        ADC_IDLE_CONTINUE &
        ADC_FORMAT_INTG &
        ADC_CLK_AUTO &
        ADC_AUTO_SAMPLING_OFF &
        ADC_SAMP_OFF;
_ADCON2=ADC_VREF_AVDD_AVSS &
        ADC_SCAN_OFF &
        ADC_SAMPLES_PER_INT_1 &
        ADC_ALT_BUF_OFF &
        ADC_ALT_INPUT_OFF;
//  Tad={Tcy(ADCS+1)}/2>334ns,  Then  ADCS>18.7,
Tad=10*Tcy
_ADCON3=ADC_SAMPLE_TIME_1 &
        ADC_CONV_CLK_SYSTEM &
        ADC_CONV_CLK_10Tcy;
_ADPCFG=ENABLE_AN0_ANA;
_ADCSSL=SCAN_NONE;
OpenADC12(_ADCON1,  _ADCON2,  _ADCON3,
_ADPCFG, _ADCSSL);

// Part8（A/D 変換の実行）
//  自動サンプリングを開始します
ADCON1bits.SAMP=1;
// A/D 変換の完了を待ちます
while(BusyADC12());
// 12 ビットデータを読み出します
ResultDataGz=ReadADC12(0);

/* Part9（LED の点灯制御）
ResultDataGz（加速度センサー出力電圧）が 1678
（0.5G）より小ならば，RB6（赤色 LED）オン，RB7
（緑色 LED）オフでなければ，RB6（赤色 LED）オフ，
RB7（緑色 LED）オン */

TRISBbits.TRISB6=0;  // RB6 を出力ポートに設定
TRISBbits.TRISB7=0;  // RB7 を出力ポートに設定

if(ResultDataGz < 1678){
  PORTBbits.RB6=1;  // RB6 オン
  PORTBbits.RB7=0;}  // RB7 オフ
else{
  PORTBbits.RB6=0;  // RB6 オフ
  PORTBbits.RB7=1; }  // RB7 オン

// Part10
//  待ち時間
for (i=0; i<6; i++){
  for (j=0; j<65000; j++){
        ;}  // 何もしない
}
}
}
---------------------------------------------------------------------------
```

以下に説明を加えます．

Part1：

2.3 項で説明した通りです．

Part2：

PIC マイコンに必須のデバイス・コンフィギュレーション [3]です．ウォッチドッグ・タイマ（異常検出機能），コード・プロテクション，発振器設定，リセット動作など，マイコンの基本動作を設定します．この設定は通常必須なのですが，ブートローダを使用した場合は，以下の設定がブートローダによって書き込まれ，ユーザの記述は飛ばされるようになっているので，今回はコメント・アウトしてあります．

・ブートローダによる設定

```
_FWDT(WDT_OFF);
_FGS(CODE_PROT_ON);
_FOSC(CSW_FSCM_OFF & FRC_PLL16);
_FBORPOR(PBOR_OFF & PWRT_64 & MCLR_EN);
```

ただし，PICkit や ICD のような書き込み器を用いてプログラミングする場合は必要となりますので，そのような場合は，コメント記号「//」をはずして記述を有効化して下さい．なお，コード・プロテクションに関する記述が，ブートローダと Part2 で異なりますが，これはユーザ・コード（ユーザが書き込んだプログラム）を保護するかどうかの指定で，プログラムの動作には影響を及ぼしません．

Part3：

2.3 項で説明した通りです．

Part4：

プログラムで使用する変数の型宣言になります．加速度の A/D 変換結果を受け取る変数，待ち時間を生成させるためのカウンタ変数を宣言しています．

Part5：

ここからが main 関数となり，実際の処理となります．

Part6：

3.1 項で説明した通り，A/D 変換を一定時間毎に，延々と繰り返すので，while を使った無限ループを使用します．

Part7～8：

既述の通り，A/D 変換の準備と実行です．

Part9：

LED の点灯制御になります．Gz=0.5G という閾値は，加速度センサの出力電圧に置きかえると（加速度センサの感度を 800mV/G とする），「1.65V（0G）＋0.4V（0.5G）=2.05（V）」となるので，デジタル値に換算すると「2.05（V）×（4096／5（V））-1」（デジタル値は 0 からカウントするため）=1678 となります．そこで，A/D 変換の結果 ResultDataGz を 1678 と比較して，これより小さい場合は 0.5G より小ということなので赤色 LED を点灯（RB6 オン），それ以外は 0.5G 以上ということなので緑色 LED を点灯（RB7 オン）させます．なお，「もし，・・・ならば」という判断には，if 文を使います．

```
if(条件文)
{ 命令 1; }
else
{ 命令 2; }
```

上記で，条件を満足するときは命令 1 が，満足しないときは命令 2 が実行されます．else 以下は省略可能です．

Part10：

第二回で説明した通り，待ち時間（サンプリング・インターバル）を作っています．i, j の値を調整して，適当な時間となるようにして下さい．なお，正確なサンプリング・インターバルを作りたい場合は，水晶発振子のような正確なクロックを使うとともに，「割り込み」という機能を使うことになります．（今回は使用しません．興味のある方はトランジスタ技術 2007 年 9 月号を参考にして下さい．）

3.3 ビルド

前回は，そのままビルドしましたが，2.3 項の【手順 1】にも書いた通り，今回はライブラリ・ファイルをもう一つ追加する必要があります．ライブラリ・ファイルは，汎用性の高い関数（**Table 1**）をオブジェクト・ファイルとしてまとめたもので，ビルドする前に，使用するライブラリを指定しておくと，ビルド中に行われるリンク作業で，指定したライブラリが，自分の書いたコードに組み込まれることになります．（**Fig. 3**）これにより，複雑な機能（関数）を，その都度一から書き込む必要がなくなり，作業性が向上するとともに，信頼性も向上します．

Table 1 MPLAB XC16 C Compiler's Libraries[4]

ライブラリ名	含まれる関数
DSP ライブラリ	**libdsp-*omf*.a** ベクタ関数，ウィンドウ関数，行列関数，フィルタ関数，変換関数，制御関数，その他
16 ビット・ペリフェラル・ライブラリ	**libpXXX-*omf*.a**（XXX は PIC 型番） 外部 LCD 関数，CAN 関数，ADC12 関数，ADC10 関数，タイマ関数，リセット／制御関数，I/O ポート関数，入力キャプチャ関数，出力コンペア関数，UART 関数，DCI 関数，SPI 関数，QEI 関数，PWM 関数，I2C 関数
標準 C ライブラリ（算術関数付き）	**libc-*omf*.a** <assert.h>診断，<ctype.h>文字処理，<errno.h>エラー，<float.h>浮動小数の特性，<limits.h>実装による制約，<locale.h>ローカライゼーション，<setjmp.h>非ロケイル・ジャンプ，<signal.h>シグナル処理，<stdarg.h>変数引数リスト，<stddef.h>共通定義，<stdio.h>入力と出力，<stdlib.h>ユーティリティ関数，<string.h>文字列関数，<time.h>日付関数と時刻関数 **libm-*omf*.a** <math.h>算術関数 **libc-*omf*.a** pic30-libs

※ ***omf*** は coff または elf

今回は，12 ビット A/D 変換機能（ADC12 関数）を使いたいので，これを含む 16 ビット・ペリフェラル・ライブラリ「libp30F2012-coff.a」をリンクします．「libp30F2012-coff.a」のリンクは，Projects ペインの Libraries を右クリックして，「Add Library / Object File」で選択して下さい．当該ファイルはデフォルトのままであれば，以下のディレクトリにあります．

C:¥Program Files¥Microchip¥xc16¥v1.11¥lib¥dsPIC30F

「Store path as: Auto」を確認して，「Add」して下さい．

以上の作業が完了したら，いよいよビルドということになるのですが，実は，このままでは，

（C¥ ... libp30F2012-coff.a までのパス）：could not read symbols: File format not recognized

とエラーが表示されることとなります．これは，今回使用している MPLAB XC16 C コンパイラのデフォルト設定で，コンパイラの出力ファイル形式が COFF 形式ではなくて ELF/DWARF 形式となっているためです．（COFF 形式と ELF/DWARF 形式は，オブジェクト・ファイル形式の種類．）ソース・コードをコンパイルして得られるオブジェクト・ファイルの形式とライブラリ・ファイルであるオブジェクト・ファイルの形式が一致しないためなので（**Fig. 3**），以下のいずれかの対処方法を取って下さい．

【対処法 1】　COFF 形式で統一

Projects ペイン（左上のウィンドウ）で，プロジェクト名を右クリック，一番下の「Properties」を選択．「XC16 (Global Options)」をクリック．「Output file format」を ELF/DWARF から COFF に変更して，「Apply」．

【対処法 2】　ELF 形式で統一

上記の「Output file format」は ELF/DWARF のまま（要するにデフォルトのまま），Libraries ファイルに，「libp30F2012-coff.a」ではなくて，同じフォルダ内の「libp30F2012-elf.a」を追加．

ライブラリの追加が終わったら，ビルド（「Build Main Project」）を実行して下さい．

どうでしょうか？筆者の場合，さらに以下のエラーが出ました．やってしまいがちなエラーかと思いますので，あえて記載しておきます．

aaa.c:xx:y y：error: stray '¥33' in program

（aaa.c はソース・ファイル名）

これは，「xx 行目の yy 文字目に全角空白が入っている」ということです．ソース・コードを良く読み返して，全角空白を取り除いて下さい．（コメント部分の全角空白は問題ありません．）

3.4　動作確認

Figure 4 の通り，ブレッド・ボードに dsPIC30F2012

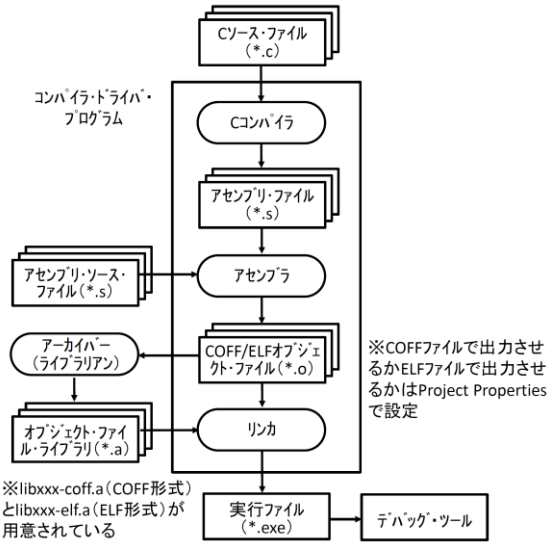

Fig. 3　Build Process of MPLAB C Compiler[5)]

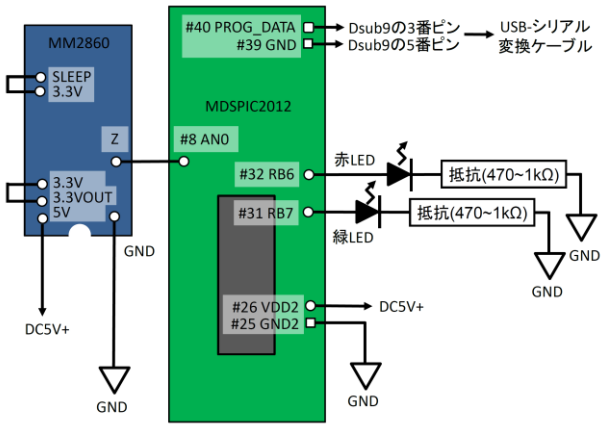

Fig. 4　Demonstration Circuit 1

と加速度センサを組み込んで下さい．電源電圧 5V を供給するとともに，上記プログラムをダウンロードして下さい．ダウンロードが完了したらプログラムを RUN させて下さい．加速度センサを組みつけたブレッド・ボードが水平に置かれているなら緑色 LED が点灯するはずです．うまくいっているようでしたら，ブレッド・ボードを傾けたり，ひっくり返してみて下さい．今度は赤色 LED が点灯するはずです．

3.5　スイッチング回路とのドッキング

最後に第一回の内容を思い出して下さい．スイッチングさせたい回路の電源を FET のドレインとソースにつないだ状態で，ゲートに電圧を印加すると，対象回路がオンされました．そこで，今回のデモ回路の RB6 出力を FET のゲートに入力させ，FET のドレインとソースに LED の点灯回路を接続します（**Figs. 5, 6**）．図中 LED 回路側電源 V は FET のドレイン－ソース間電圧より高い必

要があるので，5V 以上にして下さい．R1 は LED に流れる電流が 10mA 前後となる抵抗値のものを選んで下さい．R2，R3 は，470Ω～1kΩ程度のものを選んで下さい．（プログラムはそのままで OK です．）

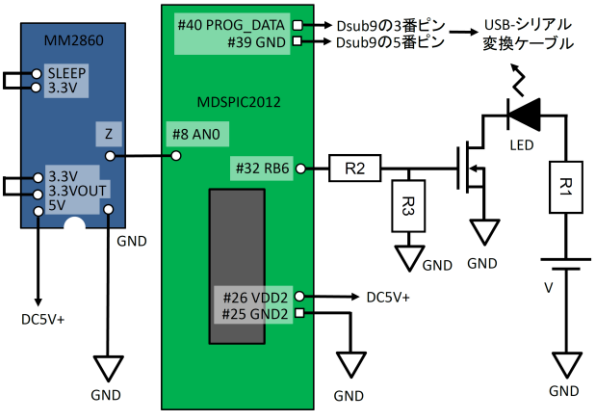

Fig. 5 Demonstration Circuit 2 Combined with a FET

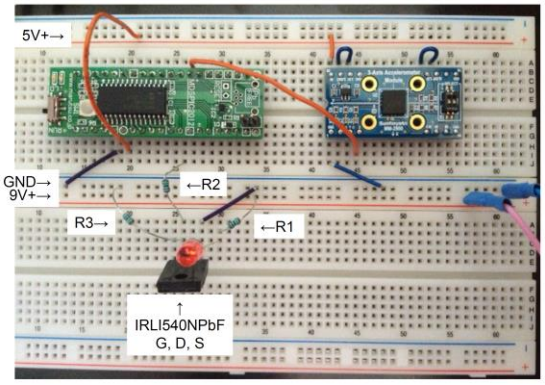

Fig. 6 Demonstration Circuit 2 Installed on a Bread Board

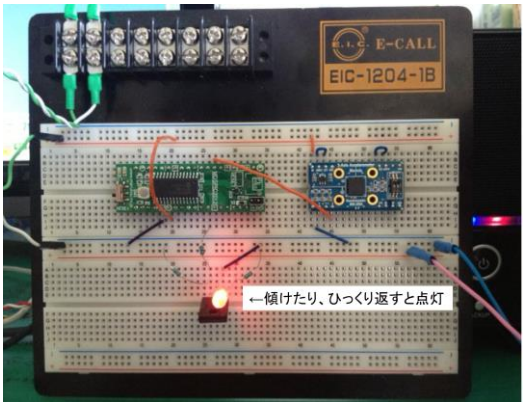

Fig. 7 Nominal Action of the Demonstration Circuit 2

さて，いかがでしょうか？ブレッド・ボードを傾けたり，ひっくり返した時に，赤色 LED が点灯すれば成功です（**Fig. 7**）．ここで，「なぜ，わざわざ FET を挟むの？」と思われる方がいるかもしれません．LED を点灯させるには 10mA 前後の電流があれば十分なので，FET を使う必要はなかったのですが，dsPIC30F2012 の最大定格電流（1 ピンあたり 25mA，全ピン合計で 200mA）を超える電流を必要とする場合は， FET（またはバイポーラ・トランジスタやリレー）を介在させる必要があるので，あえて使ってみた次第です．対象回路を変えたり，動作の内容を変えたりしてみて，理解を深めて下さい．

4. 結び

4.1 まとめ

今回は A/D 変換について学習しました．dsPIC に加速度センサの出力を入力させ，12 ビット A/D 変換を実施，加速度の値に応じた出力制御を行うことを学びました．さらに，dsPIC に FET を接続して，dsPIC の制御出力で FET を駆動，接続回路のスイッチングを行うことを学びました．これで加速度スイッチの基本構成，基本動作を全て理解したことになります．さて，次回はいよいよ，実際の装置で必要とされるシグナル・コンディショニング，回路保護などを説明したうえで，実際の装置に組み上げます．

4.2 参考情報

・マルツパーツ館　http://www.marutsu.co.jp/index.php
マルツエレック社の販売店．本講座で使用する dsPIC30F2012 モジュール基板 MDSPIC2012 や 3 軸加速度センサモジュール MM-2860（サンハヤト㈱製）の販売．店舗販売＆通販．
・秋月電子通商　http://akizukidenshi.com/
バルク品等お値打ち品が見つかる可能性大．店舗販売&通販.
・千石電商　https://www.sengoku.co.jp/
バルク品等お値打ち品が見つかる可能性大．店舗販売＆通販.
・RS コンポーネンツ　http://jp.rs-online.com/web/
法人契約が必要となりますが，在庫品に関しては翌日送が可能なので重宝しています．通販のみ.

参考文献

1) トランジスタ技術 2007 年 8，9 月号　CQ 出版
2) MPLAB X IDE ユーザガイド　マイクロチップ・テクノロジー社 DS52027A_JP
3) dsPIC30F ファミリーリファレンスマニュアル　マイクロチップ・テクノロジー社 DS70046B_JP
4) 16 ビット言語ツールライブラリ　マイクロチップ・テクノロジー社 DS51456C_JP
5) MPLAB C 30 ユーザズガイド　マイクロチップ・テクノロジー社 DS51284E_JP

(2013 年 7 月 3 日受理)