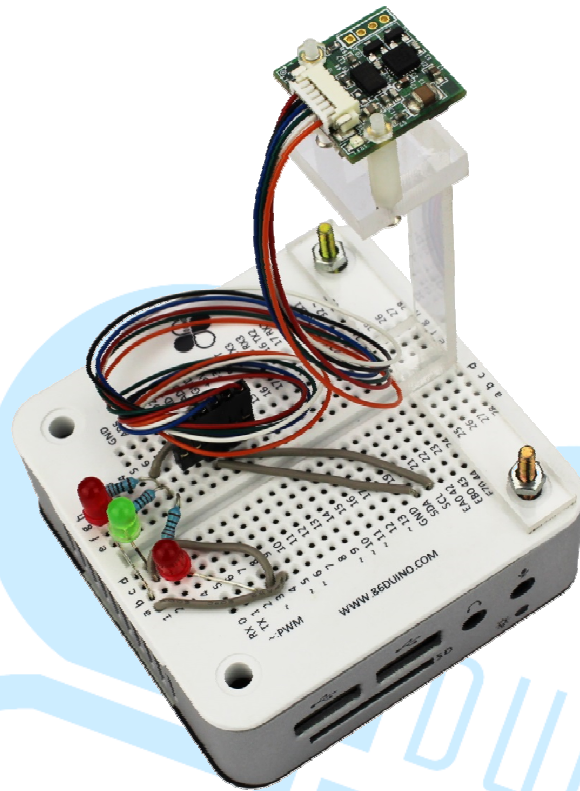


EduCake の I²C 通信



1. I²C 通信とは

前章にて紹介した 86Duino EduCake シリーズ通信機能では、「シリアル」という機能を紹介している。シリアルはハードウェア上で TX/RX ピンを通じデータの送受信を行っている。本章では別形式の通信機能、すなわち、「I²C (Inter-Integrated Circuit)」と称されているものについて述べたいと思う。

I²C とシリアル通信の違いは、ハードウェアがシリアルと異なるピンを用いて、データの転送 (TX ピン) と受信 (RX ピン) を行うことにある。データ転送時の構成にはボーレート、データビット、パリティ、ストップビット等のパラメータが存在する。I²C はハードウェア上の二本の回線において使用されており、そのうちの一本をデータ回線、残りの一本をクロック回線と呼ぶ。「半二重」の通信としてのみ機能し、通信形式もシリアルとは異なっている。ハードウェア回線におけるデータピンは通常「SDA」、クロック線のピンは「SCL」と表され、SDA データの同期として用いられる。I²C 通信デバイスは Master 及び Slave に分かれ、共に SCL シグナルによりタイミングを知ることでデータの読み取りを行うことができる。

I²C 通信の関連パラメータは、およそ以下の様になる。

スピードモード :

SCL のクロックは Master 装置が作動することによりデータ速度を決定する。当然ながら、Master 装置は Slave の規格に基づき、利用可能な速度値の決定を行う。装置の違いにより、幾つかのクロック速度が存在する。すなわち、標準モード (100 Kb/s)、低速モード (10 Kb/s)、快速モード (400 Kbit/s) 等である。

Slave 装置アドレス

Master 装置が Slave 装置の認識ナンバーにより特定の Slave 装置と通信を行う。基本的に I²C の設計が 16 bits のアドレスを保ったとしても、よく使用する 7 bits や 8 bits のアドレスに気が付かない場合、通信不能となる場合がある。

Slave 装置レジスタアドレス

Slave 装置には多くのレジスタアドレスが有り、それぞれ機能が異なる。設定値の設定、数値の測定、レジスタの内容の読み取りと書き込み等である。

I²C の Slave 装置にはそれぞれ定義されたアドレスがある。(ある装置のアドレスは固定であり、また、ある装置アドレスはハードウェアのアドレスにより決定する。)なお、メーカーによりアドレス設定に違いがあるので、使用する前にスペックやマニュアル等を確認し、使用方法を理解しておくこと。通信方法は二つ存在する。

Master にて特定な Slave のレジスタへの書き込み

Slave 装置の関数設定、モード設定等に使用される。

Master にて特定のレジスタアドレスのデータ読み込み

Slave 装置の測定数値の読み込み、或いはデバイス設定値の読み込み等に使用される。

I²C 通信形式とプログラミングについては、下記の図を参考されたい。

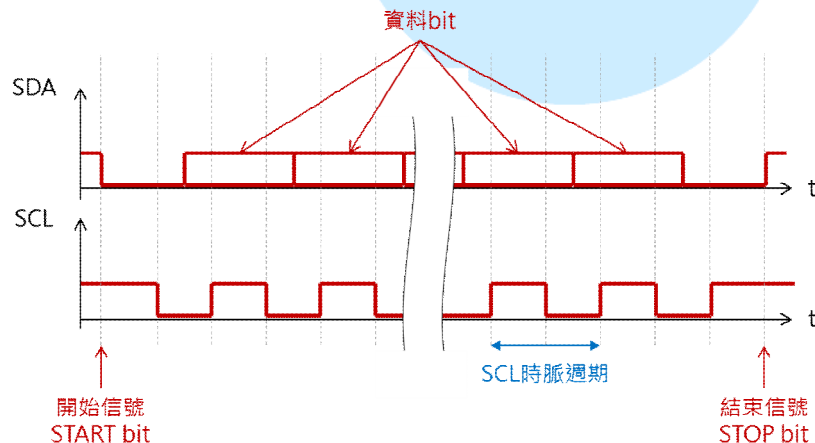


図 1. I²C 通信図面

I²C の設計では、装置はデジチェーン（SDA と SCL を結ぶ、図 2）構成が可能で、同じ制御ラインにいくつかの装置をつなぐことができる。I²C モードの場合は、MASTER 装置を使って同一バス上にある SLAVE 装置を使うことができる。しかし、この通信方法はフォールトトレランス性が低いため、近距離においてのみ使用されている。

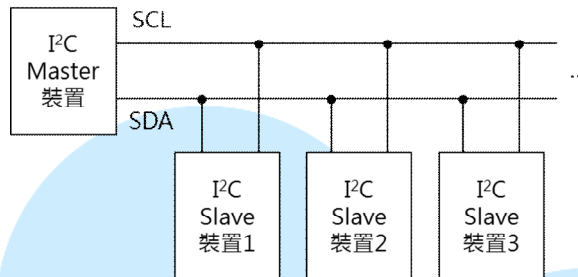


図 2. 複数の I²C デバイスの通信方式

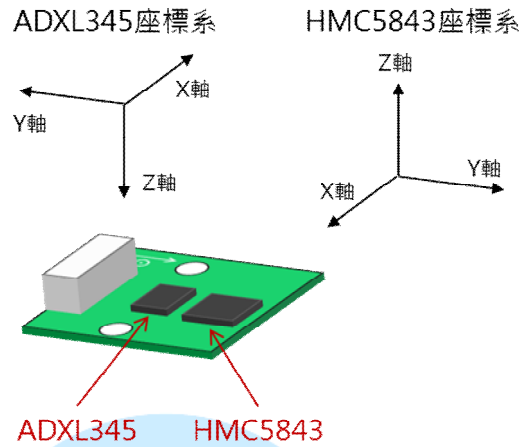
86Duino EduCake にて I²C 通信を行う際に、「Wire.h」、Wire.begin、Wire.beginTransmission、Wire.write、Wire.endTransmission、Wire.requestFrom、Wire.Read というプログラムを使い通信が可能である。

以下にいくつか例を挙げ、86Duino EduCake の通信について練習する。

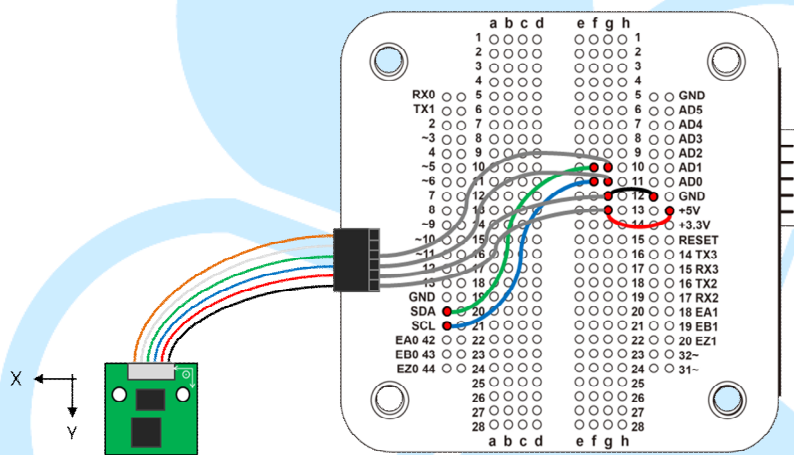
2. 例 1 - > RM-G144 の加速度計を使用

86Duino EduCake の I²C 通信の流れについて練習する。まず SLAVE 装置として RM-G144 センサーを使用する。この装置にはコンパス(型番 HMC5843)、加速度センサー(型番 ADXL345)がある。この二種類の IC チップは I²C の通信インタフェースとして、同一の I²C バス上で連結されるので、使用時に異なるデバイスアドレスの読み取りが行われれば、二種類のセンサーと通信することが可能である。センサー使用に当たり、センサー座標の定義について再確認しよう。

RM-G144 の IC 座標は通常の XYZ 軸の方向と異なる(以下の図を参照)



加速度センサーIC の通信方法について練習しよう。まず下図に基づき配線を行う。



I²Cには必ず VCC(赤コード)、GND(黒コード)、SCL(青コード)、SDA(緑コード)を用いる。白コードとオレンジ色のコードは用いない。SCL、SDAのコードを EduCake 上の SCL、SDA 部分にそれぞれ接続する。続いて 86Duino Coding IDE を開き、以下のプログラムコードの入力を実行する。

```
// for RM-G144
// IC : ADXL345
// Digital Accelerometer
#include <Wire.h>

int acc_address = 0x53;// I2C 装置アドレス (7bit)
unsigned int temp[6];

char *out_title[] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double acc_value[3] = {0,0,0};// 三軸数値のマトリクスを保存する
// 0:X-out, 1:Y-out, 2:Z-out

void setup() {
  Serial.begin(115200); // Initial Serial port

  Wire.begin(); // Initial I2C device

  G144_Acc_Init(); // 初期化された ADXL345 のレジスタ
}

void loop() {
  G144_Acc_Read();// センサーの数値を読み込み

  // 数値を print out する
  for(int i=0 ; i<3 ; i++){
    Serial.print(out_title[i]);
    Serial.print(acc_value[i]);
  }
}
```

```
        Serial.print(",\t");
    }
    Serial.print("Norm: ");
    Serial.print(
        sqrt(acc_value[0]*acc_value[0]          +
acc_value[1]*acc_value[1] + acc_value[2]*acc_value[2]));
    Serial.println( );//

    delay(100);
}

void G144_Acc_Init( ){// 初期化の ADXL345 のレジスタ
-----
    Wire.beginTransmission(acc_address);
        Wire.write(0x2d); //Power_Control register
        Wire.write(0x28); //link and measure mode = 0010 1000,
Link Bit=1, Measure Bit=1
    Wire.endTransmission( );

    delayMicroseconds(100);

    Wire.beginTransmission(acc_address);
        Wire.write(0x31); //Data_Format register
        Wire.write(0x08);
        // 設定為 Full_Resolution = 0000 1000
        // FULL_RES Bit=1 (scale factor = 4 mg/LSB), Range
Bits=00 (±2g) · 10 bits の解像度
        // Range Bits の Bit 0 と Bit 1 の 4 つ測定範囲 ( ±2g ~ ±
16g ) ·
        // ( 10 bits ~ 13 bits ) ·
        // Full_Resolution モードで · scale factor は 4mg/LSB
    Wire.endTransmission( );
```

```
    delayMicroseconds(100);

    Wire.beginTransaction(acc_address);
        Wire.write(0x38); //FIFO_Control register
        Wire.write(0x00); //bypass mode
    Wire.endTransmission();

    delayMicroseconds(100);
}

void G144_Acc_Read( ){// センサーの数値の読み取りと処理
-----
    Wire.beginTransaction(acc_address);
        Wire.write(0x32); //Read from X register (Address : 0x32)
    Wire.endTransmission();

    Wire.requestFrom(acc_address, 6); // 指定アドレスにて 6 個
byte データを読み取り
    int count = 0;
    while(Wire.available() && count < 6){
        temp[count] = Wire.read( );// フィードバックのデータを
受信する
        count++;
    }
    /*
    temp[0] : X LSB   temp[1] : X MSB
    temp[2] : Y LSB   temp[3] : Y MSB
    temp[4] : Z LSB   temp[5] : Z MSB
    */
    // データを処理する
    //
    //
    // mask = 1111 1111 0000 0000
```

上記プログラミングは 100ms 毎に一度 RM-G144 上の加速度センサーの読み込みを行う。読み込みと同時に、三方向の軸の数値を換算してから、シリアル経由でシリアルモニターに転送を行うので、画面で確認をとることが可能である。シリアルモニターを開き、実行結果を下図の通りか確認する。

The screenshot shows a serial monitor window titled 'COM7'. The window contains a list of data points, each representing a set of acceleration measurements. Each line follows the format: X-OUT: [value], Y-OUT: [value], Z-OUT: [value], Norm: [value]. The values for X-OUT range from 0.02 to 0.08, Y-OUT from -1.07 to -1.12, Z-OUT from -0.23 to -0.30, and Norm from 1.09 to 1.15. The window also features a 'Send' button at the top right, an 'Autoscroll' checkbox checked at the bottom left, and dropdown menus for 'No line ending' and '115200 baud' at the bottom right.

プログラムコード「`#include <Wire.h>`」を入力することにより、I2Cの関連関数が使用可能となる。`int acc_address` という変数は ADXL345 のアドレスにアクセスするため用いられる。プログラムには様々なアドレスが存在するが、変数をアドレスとして使用するの、最も良い方法である。

デバイスを初期化->内容を繰り返し読み込む

この二つの動作を常に用いることにより、他のプログラムにおいても使用することが可能となるので、ADXL345 が関連する動作を `G144_Acc_Init()`、`G144_Acc_Read()` という関数に変更を行う。こうすることにより、別の用途へ応用することが可能となる。

初期化段階では `setup()` において、まず `G144_Acc_Init()` を用いて初期設定を行う。

ここで上述の「Master にて特定な Slave のレジスタへの書き込み」について述べたいと思う。


```
Wire.beginTransmission(装置アドレス);→図 1 START 波型に相当  
Wire.write(指定されたレジスタアドレスへの書き込み);  
Wire.write(データ);  
Wire.endTransmission(); → 図 1 の STOP 波型に相当
```

以上のプログラムを入力することで Slave 装置はこの命令に反応し動作を行う。
ADXL345 の初期化には以下の動作が必要である。

1. 0x2d アドレスに 0x28 データを書き込む。

Power Control レジスタを 0010 1000, Link Bit=1, Measure Bit=1 にする。

2. 0x31 アドレスに 0x08 データを書き込む。

Data Format レジスタを 00001000 にして、指定する感知範囲 Full_Resolution、FULL_RES Bit=1、scale factor は 4 mg/LSB, Range Bits(有効範囲)は 00 (±2g)、このときのデータは 10 bits 解像度である。

Range Bits の Bit 0 と Bit1 にて 4 種類の測定範囲 (±2g ~ ±16g)

を設定することができる。解像度は範囲によって変わる (10 bits ~ 13 bits) 、

Full_Resolution モードでは scale factor は 4mg/LSB である。

0x38 アドレスに 0x00 データを書き込んでいく。

FIFO Control レジスタを 00001000 にしてから bypass mode とする。

ここで注意しなくてはならないのは、IC スペックによるデータを次々に書き込んでゆくには、100us 時間がかかるということである。その為、プログラムコードの後ろに必ず delayMicroseconds(100)を加えなくてはならない。

最初から特定の装置アドレスを指定して、`Wire.requestFrom` というプログラムコードを使って、Slave デバイスのデータ転送量(数 byte)を指定する。

`loop()`に `G144_Acc_Read()`を呼びだし、データを読み込む方法は下記の通りである。

```
Wire.beginTransmission(装置アドレス);
Wire.write(指定されたレジスタアドレスを読み込む);
Wire.endTransmission();

Wire.requestFrom(装置アドレス, データ byte);

int count = 0;
while(Wire.available() && count < データ byte){
    temp[count] = Wire.read(); // フィードバックデータを受信する
    count++;
}
```

`Wire.available()` と `Wire.read()` というプログラムコードは正しくデータの転送が行われているかどうかの監査官の役目を果たす。上記例図では、加速度センサーの数値が `0x32` アドレスの連続する 6 つの欄に保存されているので、データ byte 数を 6 に設定すること。

`While()` というプログラムコードを用い、連続して 6 つのデータを読みこんだ後、`temp[]` の配列に保存する。

`unsigned int temp[6]` はセンサーの数値を保存するため使用される。

`double acc_value[3]` は処理後の三軸数値を保管するために使用される。

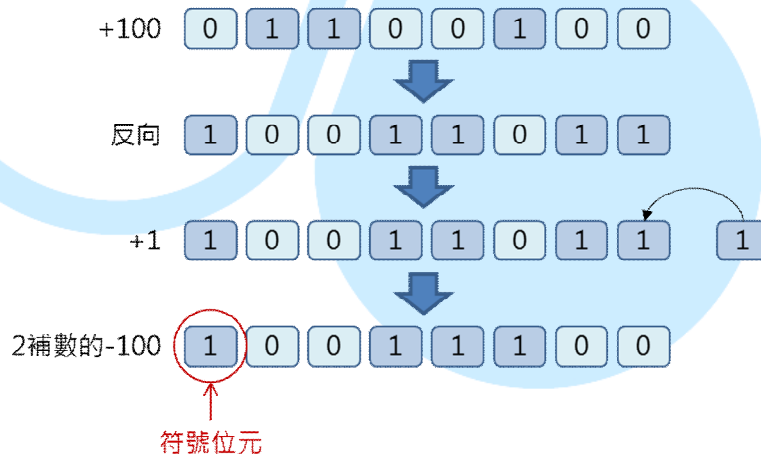
0x32 アドレスに保存されているデータには X LSB, X MSB, Y LSB, Y MSB, Z LSB, Z MSB の 6 つの byte データが存在する。その為 G144_Acc_Read() は最後の数値換算として用いられている。ADXL345 の各軸データは MSB、LSB の二つの byte に分けられるが、その前に数値化(組み合わせ)する必要がある。センサーの数値は 2 の補数で表される(+及び-がある)。下記にそのことについて詳しく述べたい。

「二の補数」とは、即ち下図の様な事である。

```

if((temp[i*2+1] & 0x80) != 0){// MSB AND b'10000000・もし
MSB の sign bit は 0 ではないと・負数となる
    temp_value      =      (((~0)>>16)<<16)      |
((temp[i*2+1]<<8)+temp[i*2]);
}
else{
    temp_value = (temp[i*2+1]<<8) + temp[i*2];
}
acc_value[i] = (double)temp_value * 4 / 1000;

```



100 を例にとると 2 の補数は-100 であり、これは上記 100 の二進表記をビット反転し 1 を加えた値である。実際に上記の 100 と-100 を加えた場合 8 ビットの範囲内では 0 になっている事を確認して下さい。これが 2 進数で使われる負数の取り扱いとなります。

第一行目の if() の「temp[i*2+1] & 0x80 != 0」は MSB の一番左側の Bit の数値のチェック役で、0 ならばこのデータは正数、逆は負数。Wire.read() は毎回 1byte のデータを読んで、MSB にて 8 個 Bit を左に移動して、LSB とたす。

2 の補数法は以下の図を参考にして、Full Resolution のモードにて換算係数 (scale factor) 4mg/LSB と acc_value[i] = (double)temp_value * 4 / 1000 を使って、換算数値が g になる。(重力加速度 = 9.8m/s²)。



RM-G144 の詳しい内容については、<http://www.roboard.com/G144.html> をご参考頂きたい。

3. 例2 -> RM-G144 の電子コンパスを使用。

以下の例は RM-G144 の電子コンパス(HMC5843)の設定だが、配線は例1と同じく、86Duino Coding IDE を開いて、上記のプログラムコードを入力する。

```
// for RM-G144
// IC : HMC5843
// Digital Compass
#include <Wire.h>

int mag_address = 0x1e;// I2C の装置アドレス(7bit)
unsigned int temp[6];

char *out_title[] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double mag_value[3] = {0,0,0};//三軸数値のマトリクスを保存する
// 0:X-out, 1:Y-out, 2:Z-out

void setup() {
  Serial.begin(115200); // Initial Serial port

  Wire.begin();// Initial I2C device

  G144_Mag_Init();//初期化された HMC5843 のレジスタ
}

void loop() {

  G144_Mag_Read();// センサーのアドレスを読み込む

  // 数値を PRINT OUT する
  for(int i=0 ; i<3 ; i++){
    Serial.print(out_title[i]);
    Serial.print(mag_value[i]);
  }
}
```

```
        Serial.print(",\t");
    }
    Serial.print("Norm: ");
    Serial.print(
        sqrt(mag_value[0]*mag_value[0]           +
mag_value[1]*mag_value[1] + mag_value[2]*mag_value[2]) );
    Serial.println();

    delay(100); // continue-measure mode > 100us
}

void G144_Mag_Init( ) { // 初期化された HMC5843 のレジスタ
-----
    Wire.beginTransmission(mag_address);
    Wire.write(0x02); // mode register
    Wire.write(0x00); // continue-measure mode
    Wire.endTransmission();

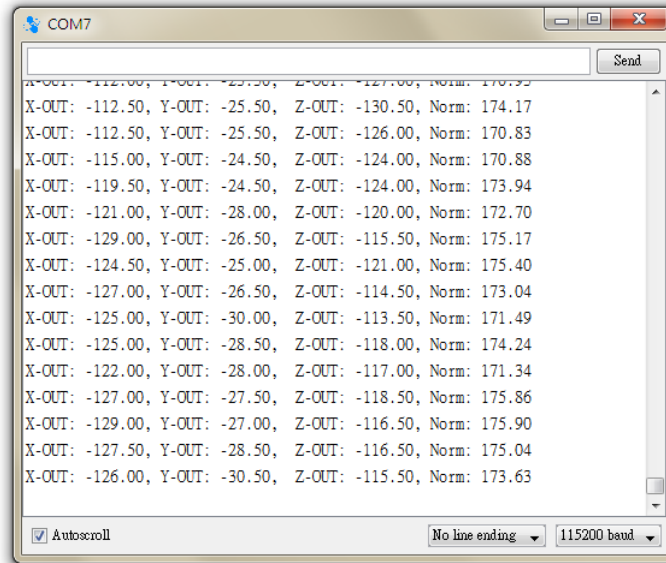
    delayMicroseconds(100);
}

void G144_Mag_Read( ) { // センサーのアドレスの読み込みと処理
-----
    Wire.beginTransmission(mag_address);
    Wire.write(0x03); // Read from data register (Address :
0x03)
    Wire.endTransmission();

    Wire.requestFrom(mag_address, 6); // Read data form
Conversion Result Register, Data : 12bits
    int count = 0;
    while(Wire.available() && count < 6){
```

```
temp[count] = Wire.read( );// フィードバックデータを受信す  
る  
    count++;  
}  
/*  
temp[0] = X MSB  temp[1] = X LSB  
temp[2] = Y MSB  temp[3] = Y LSB  
temp[4] = Z MSB  temp[5] = Z LSB  
*/  
// データ処理  
//  
//  
//  
int temp_value = 0;  
for(int i=0; i<3; i++){  
    if((temp[i*2] & 0x80) != 0){// MSB AND b'10000000、もし  
MSB の sign bit は 0 ではないと、負数となる  
        temp_value = (((~0)>>16)<<16) | ((temp[i*2]<<8) +  
temp[i*2+1]);  
    }  
    else{  
        temp_value = (temp[i*2]<<8) + temp[i*2+1];  
    }  
    mag_value[i] = (double)temp_value * 0.5;// 0.5  
milli-gauss/LSB、換算単位：milli-gauss  
}  
}
```

上記プログラムは 100ms 毎事に RM-G144 上の加速度センサーIC を読み込む。読み込みながら三方向の軸の数値を換算するので、我々はシリアル経由によりシリアルモニターで画像を確認することが可能である。結果は以下の図面のよう
に示される。



The screenshot shows a serial terminal window titled 'COM7'. The window contains a list of sensor data lines, each with four values: X-OUT, Y-OUT, Z-OUT, and Norm. The data is as follows:

X-OUT	Y-OUT	Z-OUT	Norm
-112.50	-25.50	-130.50	174.17
-112.50	-25.50	-126.00	170.83
-115.00	-24.50	-124.00	170.88
-119.50	-24.50	-124.00	173.94
-121.00	-28.00	-120.00	172.70
-129.00	-26.50	-115.50	175.17
-124.50	-25.00	-121.00	175.40
-127.00	-26.50	-114.50	173.04
-125.00	-30.00	-113.50	171.49
-125.00	-28.50	-118.00	174.24
-122.00	-28.00	-117.00	171.34
-127.00	-27.50	-118.50	175.86
-129.00	-27.00	-116.50	175.90
-127.50	-28.50	-116.50	175.04
-126.00	-30.50	-115.50	173.63

At the bottom of the window, there are three controls: a checked 'Autoscroll' checkbox, a 'No line ending' dropdown menu, and a '115200 baud' dropdown menu.

このプログラムは例 1 とほぼ同じであるが、主な違いは HMC5843 のデバイスアドレスが 0x1e 上で一様に 7bit である、ということである。

初期化設定時に入力するパラメータとレジスタアドレスが異なる。

詳しくは以下の説明をご参考願いたい。

データ書込み : 0x02 レジスタに 0x00 を書き込んでから、モードを continue-measure mode に設定する

データ読み込み : 0x03 から、測定数値のレジスタアドレスの読み込みを開始すると、三軸のデータである LSB と MSB の保存順序はちょうど ADXL345 の正反対となる。

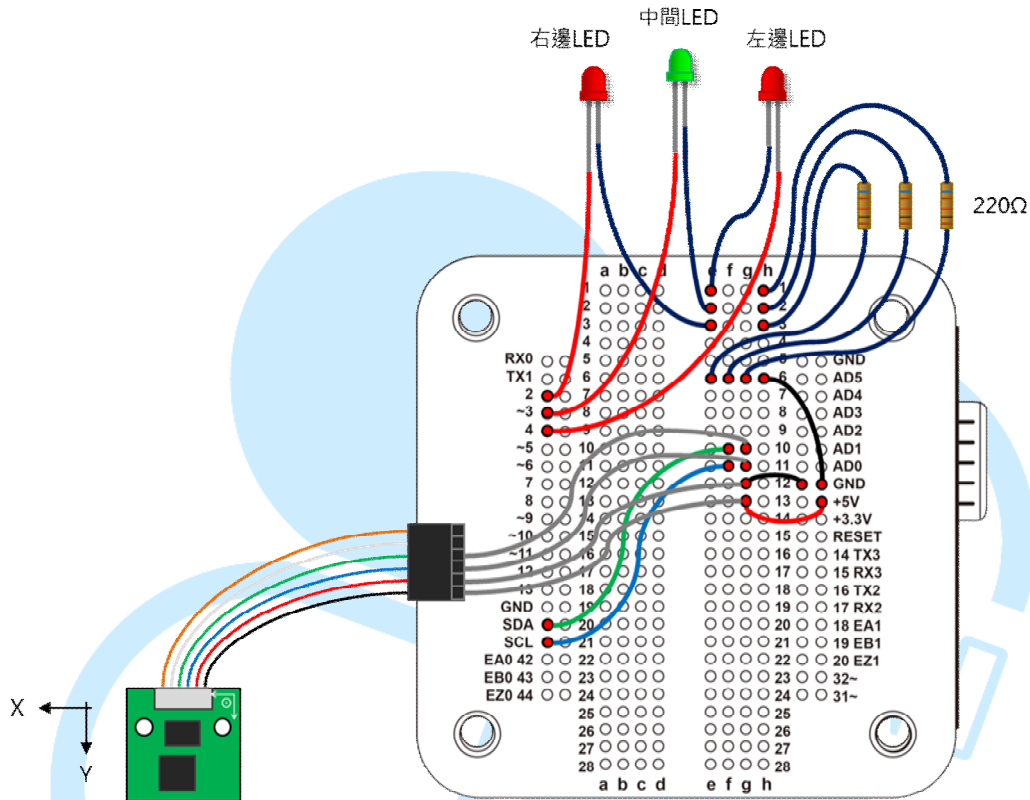
データ換算 : 元の換算単位である 0.5 milli-gauss/LSB は milli-gauss と変化する。

恐らくお気づきだと思うが、PCB ボード上に小さな座標がある。これがつまり HMC5843 の参考用座標である。

4. 例 3-センサーで方向を判断する

加速度センサーIC の読み込みが成功したら、ぜひ応用を行おう。

配線は以下の図を参考されたい。



86duino Coding IDE には、以下のプログラムコードの入力を行う。

```
// for RM-G144
#include <Wire.h>

int acc_address = 0x53;// ADXL345 装置アドレス(7bit)
int mag_address = 0x1e;// HMC5843 装置アドレス(7bit)
unsigned int temp[6];

char *out_title[] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double acc_value[3] = {0,0,0};//三軸数値のマトリクスを保存する
0:X-out, 1:Y-out, 2:Z-out
```

```
double mag_value[3] = {0,0,0}; // 三軸数値のマトリクスを保存する
0:X-out, 1:Y-out, 2:Z-out

int LED_L_pin = 4; // 左側 LED
int LED_M_pin = 3; // 真ん中 LED
int LED_R_pin = 2; // 右側 LED

void setup() {
  Serial.begin(115200); // Initial Serial port

  Wire.begin(); // Initial I2C device

  G144_Acc_Init(); // 初期化された ADXL345 のレジスタ
  G144_Mag_Init(); // 初期化された HMC5843 のレジスタ

  // 初期化 I/O
  pinMode(LED_L_pin, OUTPUT);
  pinMode(LED_M_pin, OUTPUT);
  pinMode(LED_R_pin, OUTPUT);
}

void loop() {
  G144_Acc_Read(); // センサーの数値の読み込みと処理
  G144_Mag_Read(); // センサーの数値の読み込みと処理

  float azimuth = GetAzimuth(mag_value);

  Serial.print("Azi = ");
  Serial.println(azimuth);

  // 右側の LED を制御する
  if(azimuth < -10) { // コンパスの Y 軸は西北方向に指す
```

```
digitalWrite(LED_R_pin, HIGH);// 右側の LED が点灯ならば、回転
方向をクロックワイズにする }
else{
    digitalWrite(LED_R_pin, LOW);// LED の電気を消す
}
// 真ん中の LED を制御する
if(abs(azimuth)<=10){// コンパスの Y 軸方向が北+/-10 度以内に
指す
    digitalWrite(LED_M_pin, HIGH);// 真ん中の LED が点灯なら
ば、正しく方向に示されている
}
else{
    digitalWrite(LED_M_pin, LOW);// LED の電気を消す
}
// 左側の LED を制御する
if(azimuth>10){// コンパスの Y 軸は東北方向に指す
    digitalWrite(LED_L_pin, HIGH);// 左側の LED が点灯なら
ば、回転方向をクロックワイズにする
}
else{
    digitalWrite(LED_L_pin, LOW);// LED の電気を消す
}

delay(100);
}

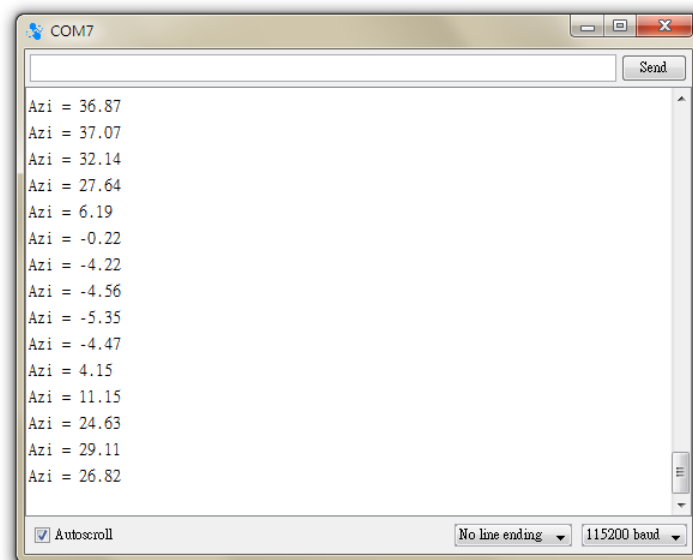
double GetAzimuth(double *m_val) {//
    // atan2(y,x) = arccos(y/x);
    double azimuth = atan2(-m_val[0], m_val[1]);// フィードバッ
クの弧度の範囲は-PI ~ PI
    /*
    if(azimuth < 0){//
        azimuth = 2 * PI + azimuth; // PI = 3.1415967
```

```
    }  
    */  
    azimuth = azimuth * 180 / PI;// 弧度を角度に換算する  
    return azimuth;  
}
```

G144_Acc_Init()、G144_Mag_Init()、G144_Acc_Read()、G144_Mag_Read()の4つは前述の二つのプログラミングと同様なので、重複を避け、ここではあらためて説明は行わないこととする。

このプログラムはHMC5843の数値を読み込んで、センサーの方向を算出する。

センサーが西側に傾いた時（角度<-10）は、右側のLEDが点灯しユーザーに知らせる。もし角度が北±10以内に傾いているならば、真ん中のLEDが点灯し、現在の方向が地磁気センサーから向かって北側であることを示す。コンパイルが86Duino Educateにアップロードされた後、Serial Monitorを開くと、実行結果が下図のように示される。



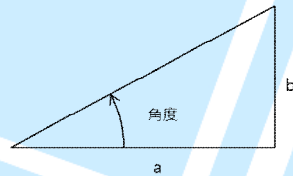
The screenshot shows a Serial Monitor window titled 'COM7'. The window contains a list of azimuth values: 36.87, 37.07, 32.14, 27.64, 6.19, -0.22, -4.22, -4.56, -5.35, -4.47, 4.15, 11.15, 24.63, 29.11, and 26.82. The window has a 'Send' button at the top right and 'Autoscroll', 'No line ending', and '115200 baud' options at the bottom.

方向を判断するプログラムは `double GetAzimuth(double *m_val)` という関数の中に在るが、関数のパラメータである「*m_val」を入力することは即ちそ

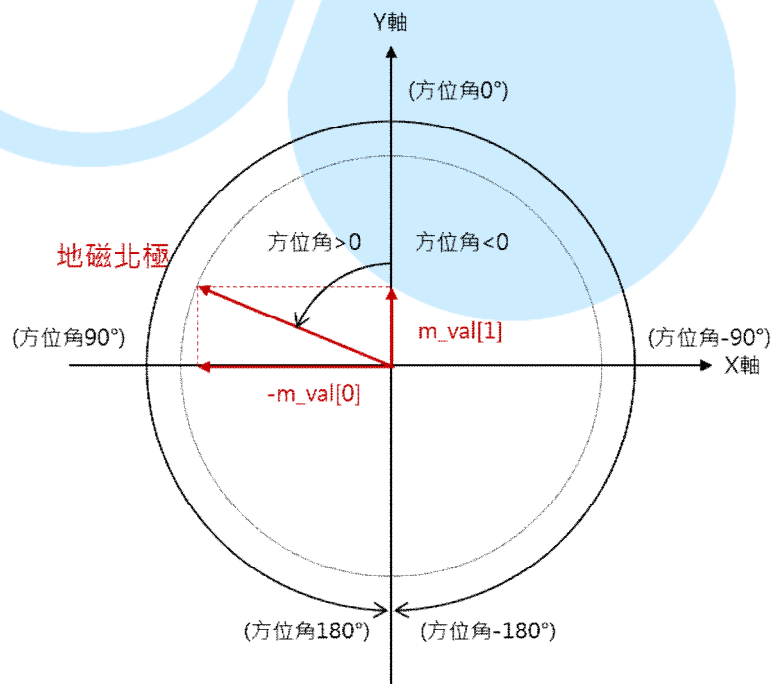
これはマトリックスのインジゲーターであると言う事が可能である。これは関数を利用する際に用いられる用語の一つである。

```
azimuth = atan2(-m_val[0], m_val[1]);
```

角度の計算をする際に、 $\text{atan2}(b, a)$ は正接三角関数として用いられる a, b の長さにより半径や、三角関数 $\tan^{-1}(b/a)$ による $-\pi \sim \pi$ の間の角の値を導き出すことが可能である。



電子コンパスの X、Y 軸でも上述の公式を応用することができる。
電子コンパスの方位は下図の様に定義したい。



反時計回りの定義から角度を求めれば、atan2 を応用する時、X 軸上に

-1 を加えなくてはならない。こうすることで、ユーザーは自身の好みに基づき、方向を定義することが可能となる。

atan2 がポストバックする時の値を経度とする。ここでもし角度値を確認したいのであれば、下記を使用すること。

```
azimuth = azimuth * 180 / PI;
```

azimuth = azimuth * 180 / PI;というコードを使用することで、様々な角度を計算ができ、また、方位も計算することが可能である。もし詳細な方位を確認したいのであれば、下記リンク先を参考願いたい。

<http://www.ngdc.noaa.gov/geomag-web/#declination>