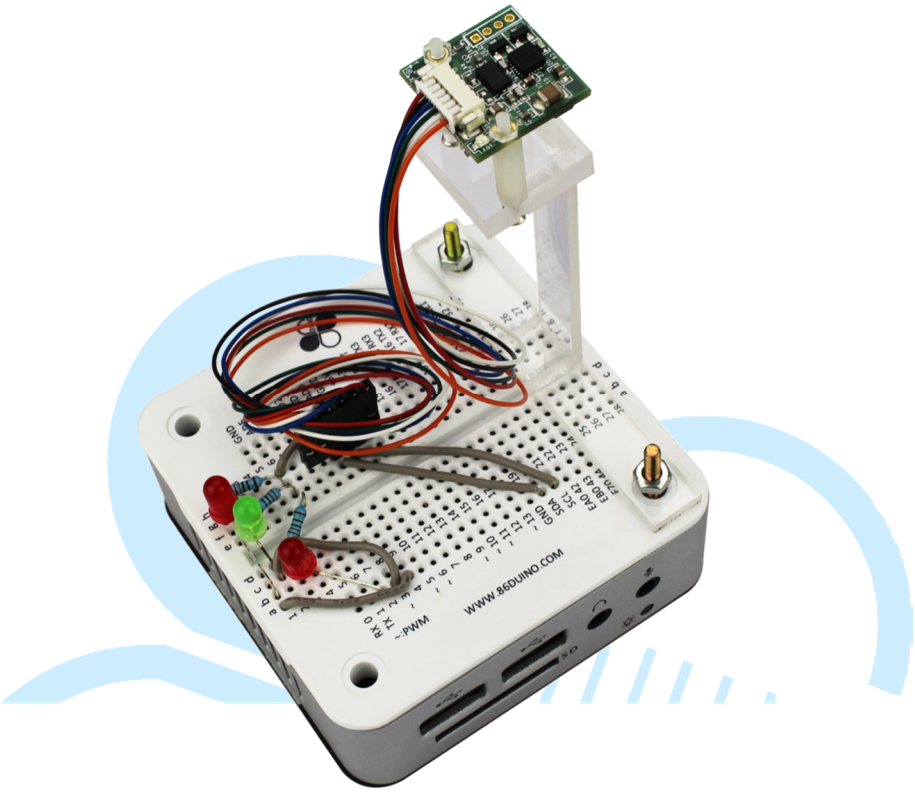
EduCake 的 I2C 通訊



一、 I2C 通訊原理簡介

前面幾個章節曾經介紹過 86Duino EduCake 的串列通訊功能，使用的是「Serial」的程式物件，硬體上則是使用 TX/RX 的腳位元做資料的傳輸與接收。本章節要講的則是另一種形式的串列通訊功能，稱為「I^2C（Inter-Integrated Circuit）」，中文雖然可翻譯為「內部整合電路」，但一般還是習慣使用「I^2C」稱呼。

I^2C 通訊介面與 Serial 通訊介面不同的地方，硬體部分在於 Serial 使用不同的腳位做資料傳輸（TX 腳位）與接收（RX 腳位），傳輸時格式則有每秒位數（鮑率）、同位元檢查、資料位元、結束位元等等參數。I^2C 雖然也是使用兩條硬體線路，但其中一條為資料線，另一條則為頻率線，因此只能做「半雙工」的通訊，

通訊格式也與 Serial 不同；硬體線路上資料線腳位元通常標示為「SDA」，頻率線腳位元則標示為「SCL」，用作 SDA 資料的同步用途。I^2C 通訊裝置分為 Master(主端)與 Slave(從端)，雙方透過 SCL 訊號以得知何時可以讀取有效資料。I^2C 通訊的相關參數主要敘述如下：

- **速度模式：**

  SCL 的頻率會由 Master 裝置發起與決定通訊速度，當然 Master 裝置也得看 Slave 端的規格去決定可用的速度值；視裝置的不同，有幾種常用的頻率速度：標準模式（100 Kb/s）、低速模式（10 Kb/s）、快速模式（400 Kbit/s）等等。

- **Slave 裝置位址：**

  代表 Slave 裝置的特定識別號碼，Master 端可藉由此號碼與特定的 Slave 裝置通訊。I^2C 設計原始保留了 16 bits 的位址空間，但較常使用 7 bits 或 8 bits 位址，實際使用需注意規格表上的定義，否則可能會發現通訊時沒有反應喔。

- **Slave 裝置的暫存器位址：**

  Slave 裝置有許多的暫存器位址，有的用在裝置的設定值，有的則是此裝置測量到的資料等等。寫入與讀取特定的裝置暫存器內容，便是 I^2C 通訊方式的例行公事所在。

I^2C 的 Slave 端裝置通常會有各自訂的位址（有的裝置是直接固定位址，有的可以透過硬體設定位址），另外也有許多製造廠商自行定義的暫存器位址；讀者使用此類裝置時須查閱裝置的規格表、使用說明等等，才能知道如何與特定裝

置做通訊，以及瞭解如何解讀這些數值；但通訊方式主要有兩種共通的動作程式：

- **Master 端將某段資料寫入 Slave 端特定的暫存器位址內：**

  用在進行設定 Slave 裝置的參數、模式等等。

- **主端指定由從端特定暫存器位址讀取一段資料：**

  用在做 Slave 裝置測量數值的讀取，或讀取裝置設定值等用途。

I^2C 的通訊格式與程式如下圖 1 所示：

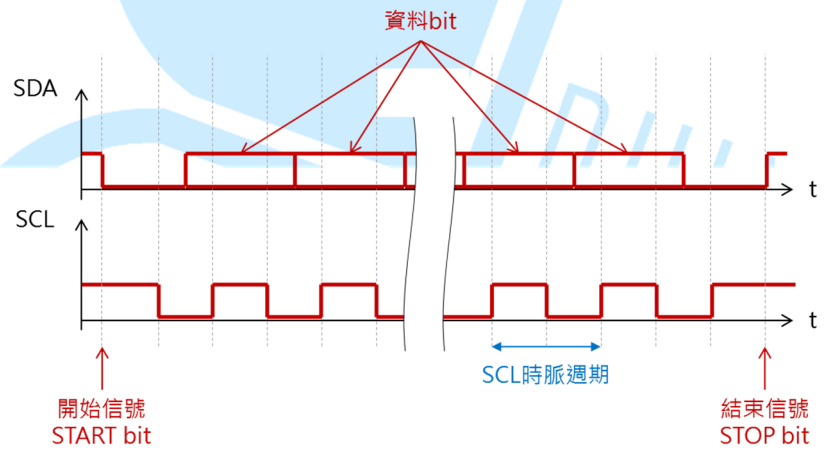


圖 1. I^2C 通訊格式波型圖

由於 I^2C 硬體線路與裝置的參數特性，多個 I^2C 裝置通訊可以使用稱為「Daisy Chain」的連線方式，如圖 2 所示；也就是同一條 I^2C 匯流排（SDA、SCL）上面可並聯多個位址不同的裝置，Master 端不需與每個 Slave 裝置都使用一條專屬的匯流排。通訊時由於格式指定了特定的裝置位址與暫存器位址，因此只有符合位址的 Slave 裝置才會有回應動作，是相當省線路的一種通訊方式；但由於通訊格式較簡易與缺乏容錯性，且訊號也會受硬體線路衰減影響，缺點是僅適用於近距離的通訊應用，一般是在幾公尺的距離內使用。

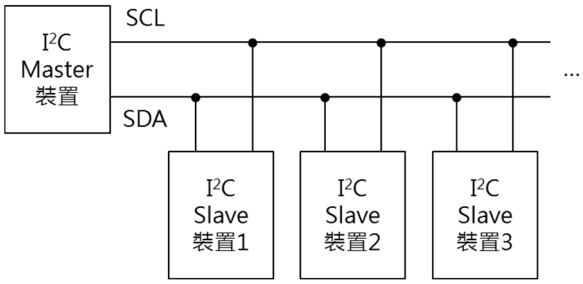


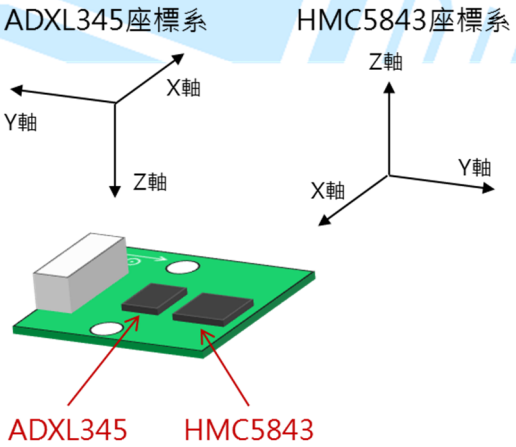
圖 2. 多個 I2C 裝置通訊連線方式

在 86Duino EduCake 上進行 I²C 通訊時，當然不必勞煩使用者去手動控制那些波型，只要使用到「Wire.h」這個函式庫，以及內建的幾個函式：Wire.begin()、Wire.beginTransmission(裝置位址)、Wire.write(byte 資料)、Wire.endTransmission()、Wire.requestFrom(裝置位址, 資料 byte 數)、Wire.Read()即可與裝置做通訊，以下幾個範例就實際使用一個 I²C 的感測器裝置，來做 86Duino EduCake 通訊的程式練習吧。

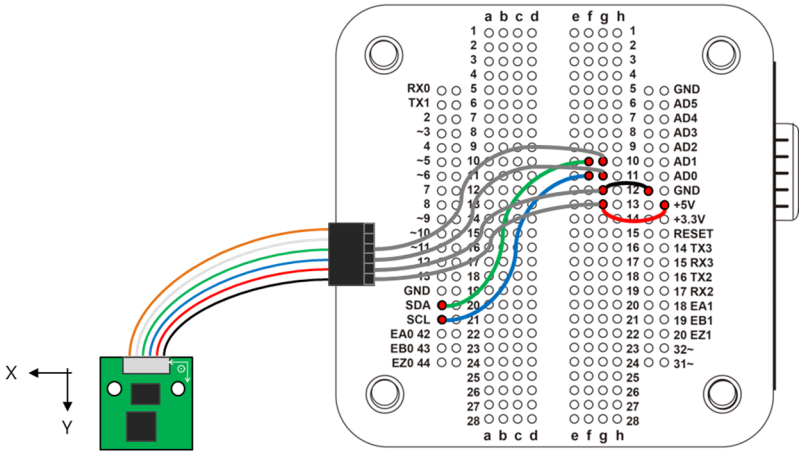
二、 第一個程式 – 使用 RM-G144 的加速度計

第一個範常式式，先來練習如何使用 86Duino EduCake 的 I²C 通訊流程，這裡我們選用的 Slave 端裝置為 RM-G144 感測器，此裝置含有三軸電子羅盤（compass）、三軸數位加速度計（accelerometer）兩種 IC，電子羅盤 IC 型號為 HMC5843，數位加速度計 IC 的型號則是 ADXL345。由於這兩個 IC 都是 I²C 的通訊介面，並聯在同一個 I²C 匯流排上，所以使用時只要對不同的裝置位址進行存取，就可以與兩種感測器做通訊囉。

使用感測器前，先來認識一下感測器本身的座標定義；RM-G144 上頭兩個 IC 的座標正負號剛好相反，如下圖所示；因此之後使用數值的時候得注意方向的問題。



我們首先來練習加速度計 IC 的通訊方式，讀者請先依下圖接線：



讀者從上圖可以觀察到，I^2C 通常會有幾個一定會使用到的接線：Vcc（紅線）、GND（黑線）、SCL（藍線）、SDA（綠線），白線跟橘線在這邊沒有使用到。SCL、SDA 線分別接到 EduCake 上標示有 SCL、SDA 的地方即可。接著請打開 86Duino Coding IDE，輸入以下程式碼：

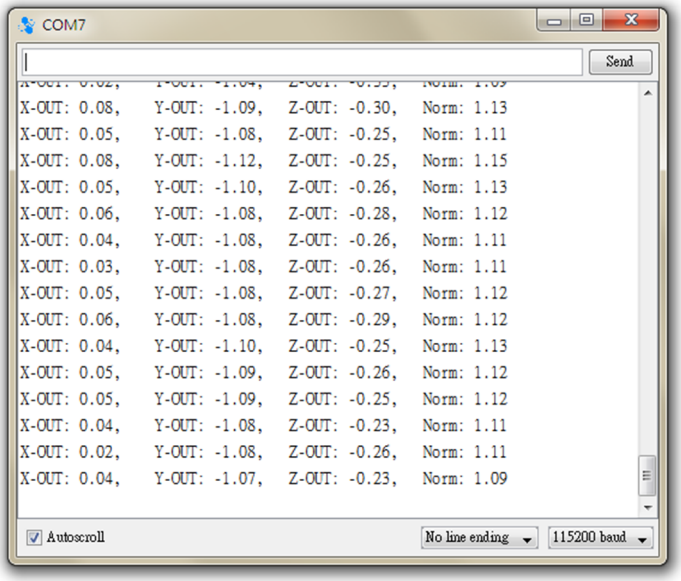
```
// for RM-G144
// IC : ADXL345
// Digital Accelerometer
#include <Wire.h>

int acc_address = 0x53;// I2C 裝置位址，此為 7bit 地址
unsigned int temp[6];

char *out_title[ ] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double  acc_value[3]  =  {0,0,0};//  儲存感測器三軸數值的矩陣
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( ); //  初始化 ADXL345 的暫存器
}

void loop( ) {
  G144_Acc_Read( );//  讀取感測器並處理數值

  //  印出數值
  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( );// 換行

        delay(100);
    }

    void  G144_Acc_Init(  ){//  初 始 化  ADXL345  的 暫 存 器
-------------------------------------------
        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 解析度
            // Range Bits 的 Bit 0 與 Bit 1 設定4種量測範圍( ±2g ~ ±
16g），
            // 解析度隨範圍改變（10 bits ~ 13 bits），
            // 在 Full_Resolution 模式下，scale factor 為 4mg/LSB
        Wire.endTransmission( );
```

```
        delayMicroseconds(100);

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

        delayMicroseconds(100);
    }

    void  G144_Acc_Read(  ){//  讀 取 感 測 器 並 處 理 數 值
-------------------------------------------
        Wire.beginTransmission(acc_address);
            Wire.write(0x32); //Read from X register (Address : 0x32)
        Wire.endTransmission( );

        Wire.requestFrom(acc_address, 6); // 從指定位址依序讀取 6
個 byte 的資料

        int count = 0;
        while(Wire.available( ) && count < 6){
            temp[count] = Wire.read( );// 接收傳回來的資料
            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
        */
        // 處理資料
        // ADXL345 回傳，每軸向分為兩個 byte
        // 數值以 2 補數法表示，int 變數須做符號延伸
        // mask = 1111 1111 0000 0000
```

```
        // 負值符號延伸 = mask | 0000 0000 MSB LSB = 1111 1111 MSB LSB
        // 回傳值範圍：-512 ~ 511（± 2 g），若加速度超過表示範圍，將會回傳最大值
        int temp_value = 0;
        for(int i=0 ; i<3 ; i++){
          if((temp[i*2+1] & 0x80) != 0){// MSB AND b'10000000，如果 MSB 的 sign bit 不為 0，表示為負值
            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; // 4mg/LSB，換算為單位：g
        }
      }
```

此範常式式功能為每 100ms 讀取一次 RM-G144 上的數位加速度計，讀取並換算好三軸的數值，再經由 Serial 傳輸至 Serial Monitor 觀看。編譯並上傳程式至 86Duino EduCake 之後，打開 Serial Monitor，執行結果如下圖：



程式一開始須先寫「#include <Wire.h>」，才能使用 I²C 相關的功能函數；int acc_address 變數則用來存 ADXL345 的位址，由於程式許多地方需使用位址資料，使用變數當作位址可以較容易修改程式，也是讀者可以記起來的一個小技巧。整體流程可以分為兩階段：初始化裝置→重複讀取內容，由於這兩個動作經常會使用，也可能會用到其他程式專案中，所以此處把所有 ADXL345 相關的動作包裝成 G144_Acc_Init()、G144_Acc_Read()兩個函式，讀者之後就可以方便使用這兩個動作函數做作其他用途囉。

初始化階段在 setup()內完成，先呼叫 G144_Acc_Init()做初始化設定；這邊許使用前面提到的「**Master 端將某段資料寫入 Slave 端特定的暫存器位址內**」動作，需做的流程語法為：

```
Wire.beginTransmission(裝置位址); → 相當於產生圖 1 的 START 波型
Wire.write(指定要寫入的暫存器位址);
Wire.write(資料);
Wire.endTransmission(); → 相當於產生圖 1 的 STOP 波型
```

Slave 裝置才會對這個指令有反應，並執行動作；ADXL345 的初始化須執行三次這個動作：

1. 對 0x2d 位址寫入資料 0x28：

   設定 Power Control 暫存器為 0010 1000，Link Bit=1, Measure Bit=1。

2. 對 0x31 位址寫入資料 0x08：

   設定 Data Format 暫存器為 0000 1000，指定感測範圍為 Full_Resolution，FULL_RES Bit=1，表示 scale factor（換算參數）為 4 mg/LSB，Range Bits（有效測量範圍）為 00 (±2g)，此時數據為 10 bits 解析度，

Range Bits 的 Bit 0 與 Bit 1 可設定 4 種量測範圍（±2g ~ ± 16g）

解析度隨範圍改變（10 bits ~ 13 bits），視實際使用狀況去設定；

在 Full_Resolution 模式下，scale factor 為 4mg/LSB。

3. 對 0x38 位址寫入資料 0x00：

設定 FIFO Control 暫存器為 0000 1000，為 bypass mode。

這邊須注意，由於IC規格表內提到每次寫入資料後都需間隔至少100us才能進行下一個動作，所以每次寫入動作後使用 delayMicroseconds(100)做延遲時間。

loop( )內持續呼叫 G144_Acc_Read( )做讀取；這邊許使用前面提到的「**主端指定由從端特定暫存器位址讀取一段資料**」動作，需做的流程語法為：

```
Wire.beginTransmission(裝置位址);
Wire.write(指定要讀取的暫存器位址);
Wire.endTransmission( );

Wire.requestFrom(裝置位址, 資料 byte 數);

int count = 0;
while(Wire.available( ) && count <資料 byte 數){
  temp[count] = Wire.read( );// 接收傳回來的資料
  count++;
}
```

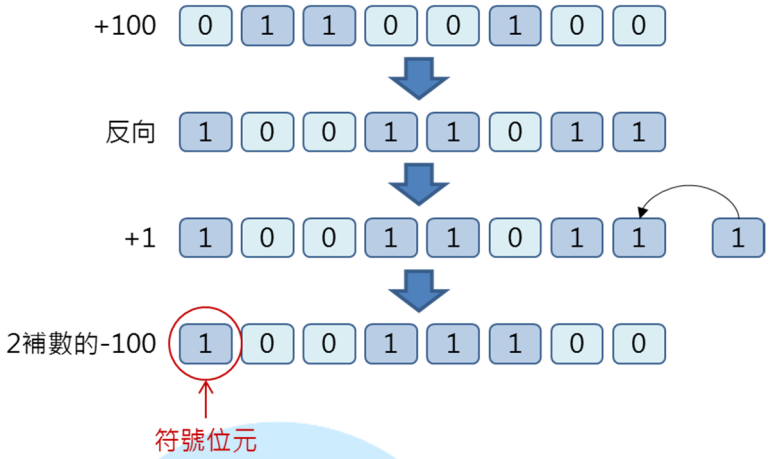
一開始先指定要從特定的裝置位址讀取某個暫存器位址，然後使用 Wire.requestFrom(裝置位址, 資料 byte 數)語法，指定 Slave 裝置回傳數

個 byte 的資料。之後使用 Wire.available( )檢查是否有資料回傳，搭配 Wire.read( )方式讀取數值到某個變數中。對於這個範例，加速度計的數值被儲存在 0x32 位址以下連續 6 個欄位，因此資料 byte 數設定為 6，使用 while( )迴圈語法連續讀取 6 個資料，並存到 temp[ ]矩陣當中。

除了上述兩個動作之外，資料換算也是重點之一，程式一開始宣告的 unsigned int temp[6]是用來暫存感測器數值的變數，double acc_value[3]則用來存放處理過後的三軸數值。由於 0x32 位址以下 6 個欄位依序儲存的是：X LSB、X MSB、Y LSB、Y MSB、Z LSB、Z MSB 六個 byte 資料，因此 G144_Acc_Read( )最後一階段動作是進行數值的換算。ADXL345 每軸資料分為 MSB、LSB 兩個 byte，須先進行組合再換算；而感測器數值使用 2 補數表示，數值有正負號，所以須分正負狀況討論，語法為：

```
    if((temp[i*2+1] & 0x80) != 0){// MSB AND b'10000000，如果
MSB 的 sign bit 不為 0，表示為負值
        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;
```

這邊需要先解釋一下「2 補數法」，如下圖：



以正數「100」為例，2 補數表示的「-100」是將 100 的二進位值反向再加 1 的結果，讀者可以簡單驗證圖中 100 與-100 的 2 進制數值相加是否剛好為 0 呢？這就是一般電腦內常用的二進位負數標記法則。

回到程式，第一行 if( )語法內的「temp[i*2+1] & 0x80) != 0」用在檢查 MSB 的最左邊位是否為 0，若是 0 表示這個資料是正數，反之為負數。由於 Wire.read( )每次唯讀一個 byte 的資料，重組資料須將 MSB 左移 8 個位，再與 LSB 相加。當數據重組為 int 型態（86Duino EduCake 的 int 型態是 32 bits）時，左方空間須做「符號延伸」，也就是把負數左邊全部空位都補 1。

因此當數值以 2 補數法表示時，若為負數變數須做符號延伸，怎麼改變記憶體中的「位元」數值呢？這邊就需要使用「位元元運算」了，作法如下：

首先製造一個遮罩（Mask）為 b'11111111 11111111 00000000 00000000，然後再與重組後的「(MSB<<8)+LSB」做 OR 運算，便可將左方全部補 1 囉。

最後一個步驟是將數值換算成實際測量單位，前面在做裝置初始化時選擇了 Full Resolution 模式，此模式下的換算參數（scale factor）為 4mg/LSB，因此 acc_value[i] = (double)temp_value * 4 / 1000 便可以將數值換算為 g 了（1 單位的重力加速度 = 9.8m/s^2）；這邊運算時讀者需要注意變數型態的轉換，若沒有先轉成浮點數就做乘/除法，小數點是不會被保留的。這樣便完成了 RM-G144 上頭 ADXL345 加速度計的設定與讀取囉。

（RM-G144 詳細規格與參數可參考 http://www.roboard.com/G144.html，讀者若有興趣的話也可以查詢這些資料進行閱讀。）

三、第二個程式 – 使用 RM-G144 的電子羅盤

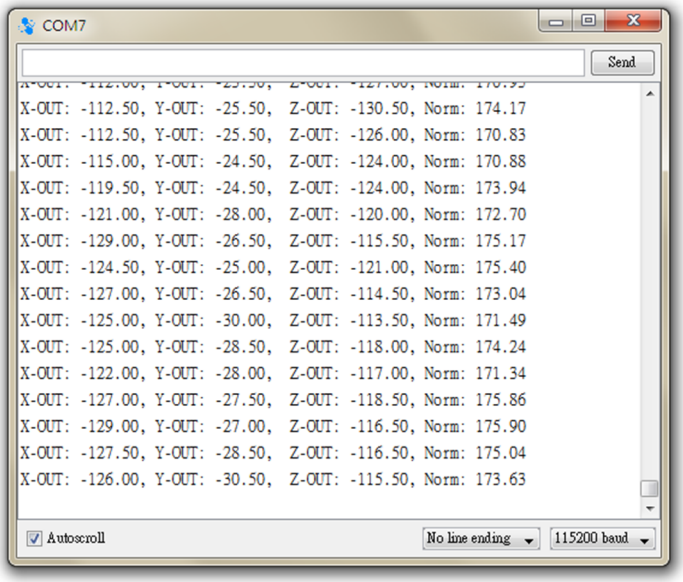
第二個範常式式我們接著做 RM-G144 上頭的 HMC5843 電子羅盤讀取與設定，接線不用更動；讀者請打開 86Duino Coding IDE，輸入以下程式碼：

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

int mag_address = 0x1e;// I2C 裝置位址，此為 7bit 地址
unsigned int temp[6];

char *out_title[ ] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double  mag_value[3]  =  {0,0,0};//  儲存感測器三軸數值的矩陣
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( ); //  初始化 HMC5843 的暫存器
}

void loop( ) {

  G144_Mag_Read( );//  讀取感測器並處理數值

  //  印出數值
  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( );// 換行

        delay(100);// continue-measureture mode 須有>100us 的間
隔時間
    }

    void  G144_Mag_Init(  ){//  初始化  HMC5843  的暫存器
----------------------------------------------
        Wire.beginTransmission(mag_address);
            Wire.write(0x02); //mode register
            Wire.write(0x00); //continue-measureture mode
        Wire.endTransmission( );

        delayMicroseconds(100);
    }

    void  G144_Mag_Read(  ){//  讀取感測器並處理數值
----------------------------------------------
        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( );// 接收傳回來的資料
            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
        */
        // 處理資料
        // HMC5843 回傳數值，每軸向分為兩個 byte
        // 數值以 2 補數法表示，int 變數須做符號延伸
        // 回傳值範圍：-2048~2047，若外在磁力超過表示範圍，將會回傳-4096
        int temp_value = 0;
        for(int i=0 ; i<3 ; i++){
          if((temp[i*2] & 0x80) != 0){// MSB AND b'10000000，如果MSB 的 sign bit 不為 0，表示為負值
            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，換算為單位：milli-gauss
        }
      }
```

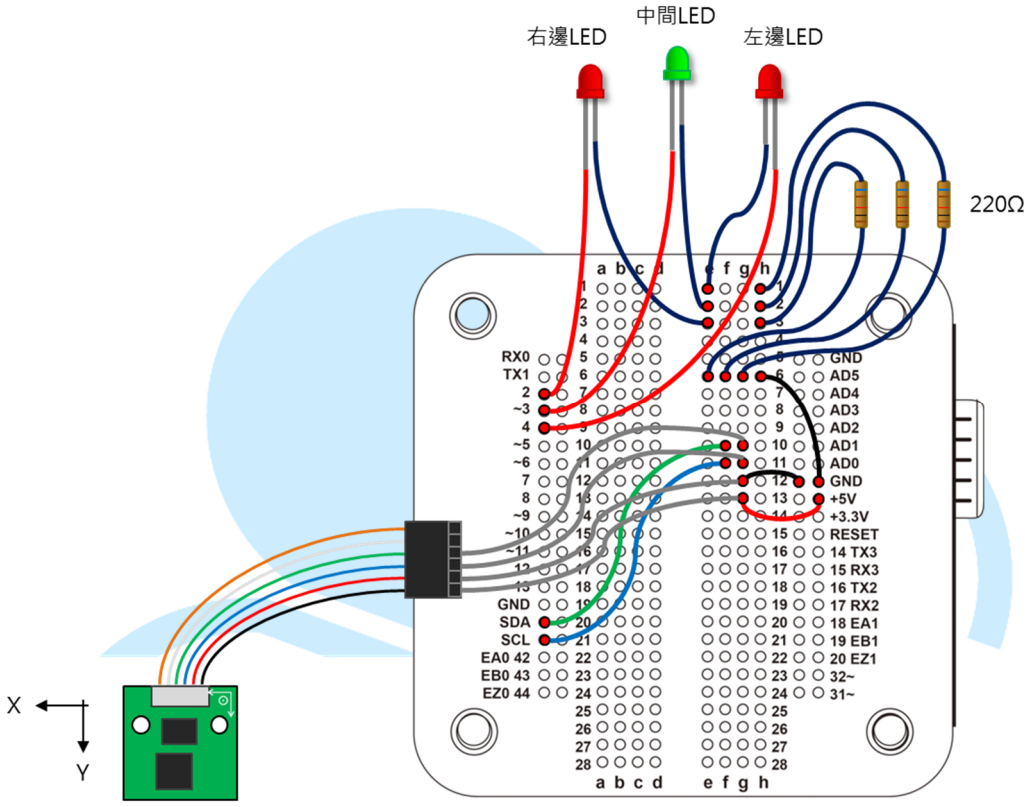
此範常式式功能與第一個類似，每 100ms 讀取一次 RM-G144 上的電子羅盤，讀取並換算好三軸的數值，再經由 Serial 傳輸至 Serial Monitor 觀看。編譯並上傳程式至 86Duino EduCae 之後，打開 Serial Monitor，執行結果如下圖：



程式流程與第一個幾乎相同，主要差別在 HMC5843 的裝置位址在 0x1e，一樣是 7bit 地址;而初始化設定時寫入的參數與暫存器位址也不同,只需在 0x02 暫存器寫入 0x00，將模式設定成 continue-measureture mode 即可。讀取時也稍有不同,測量數值的暫存器位址由 0x03 開始,而且三軸資料的 LSB 與 MSB 存放順序剛好跟 ADXL345 相反，所以資料重組時須注意這邊的差異；換算參數則是 0.5 milli-gauss/LSB，換算完後是 milli-gauss 的單位。讀者請仔細看感測器的 PCB 板上有標示一個小小的座標，這就是 HMC5843 的參考坐標系；這個電子羅盤測量的是正北方向的磁場，當 Y 軸面向磁場北極時，Y 軸向會有最大讀值，而 X 軸會接近 0；讀者可以自行嘗試多種不同的擺放方式看看數值的變化狀況。

四、 第三個程式 – 利用感測器判斷方位

成功讀取電子羅盤數值之後，就可以拿它來做些應用了，請讀者先增加接線如下圖：



接著在 86Duino Coding IDE，輸入以下程式碼：

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

int acc_address = 0x53;// ADXL345 裝置位址，此為 7bit 地址
int mag_address = 0x1e;// HMC5843 裝置位址，此為 7bit 地址
unsigned int temp[6];

char *out_title[] = {"X-OUT: ","Y-OUT: ","Z-OUT: "};
double  acc_value[3]  =  {0,0,0};//  儲存感測器三軸數值的矩陣 0:X-out, 1:Y-out, 2:Z-out
```

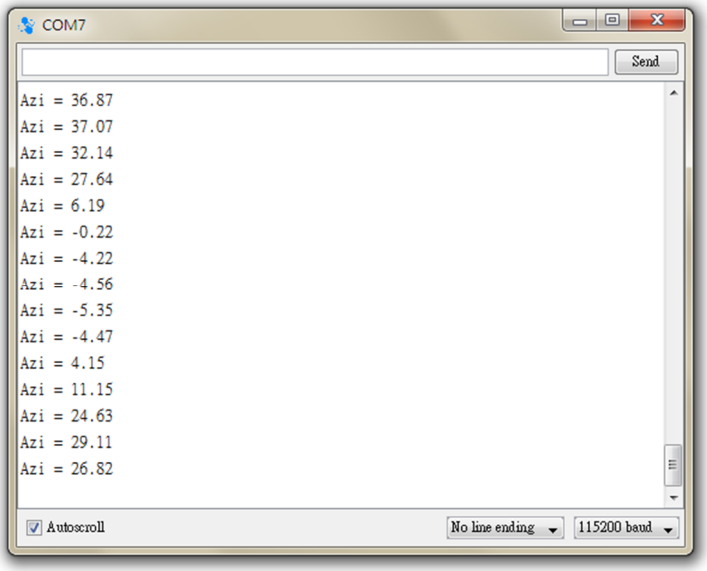
```
    double  mag_value[3]  =  {0,0,0};//  儲存感測器三軸數值的矩陣
0:X-out, 1:Y-out, 2:Z-out

    int LED_L_pin = 4;//  左邊 LED
    int LED_M_pin = 3;//  中間 LED
    int LED_R_pin = 2;//  右邊 LED

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

      Wire.begin( );// Initial I2C device

      G144_Acc_Init( );//  初始化 ADXL345 的暫存器
      G144_Mag_Init( ); //  初始化 HMC5843 的暫存器

      //  初始化 I/O
      pinMode(LED_L_pin,OUTPUT);
      pinMode(LED_M_pin,OUTPUT);
      pinMode(LED_R_pin,OUTPUT);
    }

    void loop() {
      G144_Acc_Read( );//  讀取感測器並處理數值
      G144_Mag_Read( );//  讀取感測器並處理數值

      float azimuth = GetAzimuth(mag_value);

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

      //  控制右邊 LED
      if(azimuth<-10){//  羅盤 Y 軸方向偏西北
```

```
      digitalWrite(LED_R_pin, HIGH);// 亮起右邊 LED，表示要向順時針方向轉
        }
        else{
          digitalWrite(LED_R_pin, LOW);// LED 熄滅
        }
        // 控制中間 LED
        if(abs(azimuth)<=10){// 羅盤 Y 軸方向在正北+/-10 度內
          digitalWrite(LED_M_pin, HIGH);// 亮起中間 LED，表示目前方向正確
        }
        else{
          digitalWrite(LED_M_pin, LOW);// LED 熄滅
        }
        // 控制左邊 LED
        if(azimuth>10){// 羅盤 Y 軸方向偏東北
          digitalWrite(LED_L_pin, HIGH);// 亮起左邊 LED，表示要向時針方向轉
        }
        else{
          digitalWrite(LED_L_pin, LOW);// LED 熄滅
        }

        delay(100);
      }

      double GetAzimuth(double *m_val) {//)
        // atan2(y,x) = arccos(y/x);
        double azimuth = atan2(-m_val[0], m_val[1]);// 回傳值為徑度，範圍為 -PI ~ PI
        /*
        if(azimuth < 0){// 修正 azi 的範圍至 0 到 2*PI 之間
          azimuth = 2 * PI + azimuth; // PI = 3.1415967
```

```
    }
      */
      azimuth = azimuth * 180 / PI;// 徑度換算成角度
      return azimuth;
    }
```

其餘四個函式：G144_Acc_Init()、G144_Mag_Init()、G144_Acc_Read()、G144_Mag_Read()與前兩個範常式式相同，就不再重複解釋囉。

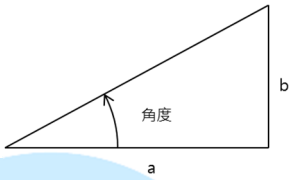
此範常式式功能為讀取 HMC5843 的數值，並且計算目前感測器面向的方位，當感測器偏向西半側（角度<-10）時，則右邊 LED 亮起提醒使用者要朝順時針方向走，反之亮起左邊 LED；如果目前角度在地磁正北方的±10 內，則亮起中間 LED，表示目前方向朝向地磁北方。編譯並上傳程式至 86Duino EduCae 之後，打開 Serial Monitor，執行結果如下圖：



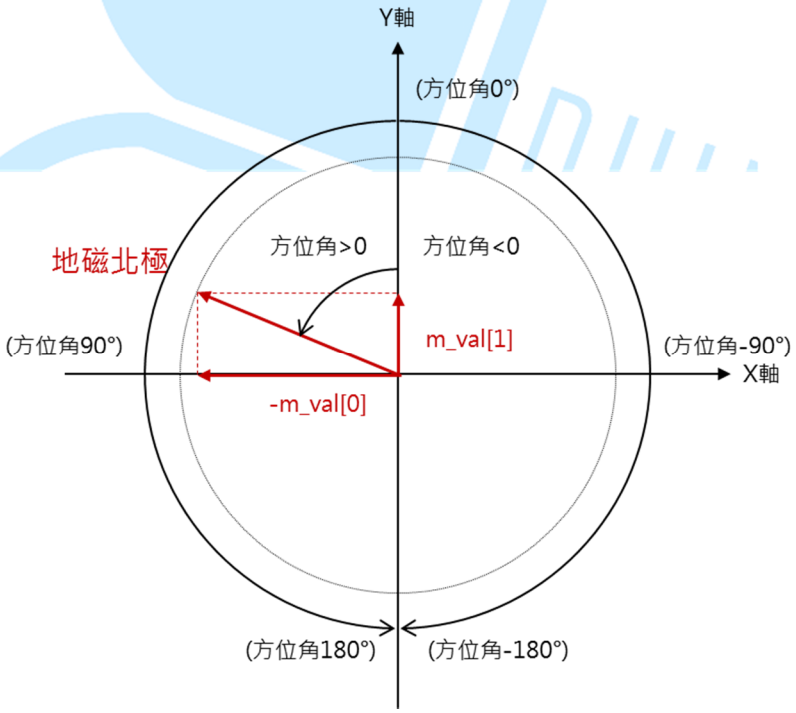
判斷方向的程式在 double GetAzimuth(double *m_val)這個函式內，傳入函式的參數「*m_val」是矩陣的指標，函式內利用語法：

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

做角度計算，atan2(b, a)為反正切三角函數，如下圖定義，可從 a、b 長度計算出夾角，單位為徑度（radius）。此函數功能相當於 $\tan^{-1}(b/a)$，但可以得到-π~π的角度值。



電子羅盤 X、Y 軸的讀值剛好可以應用這個數學式，我們將電子羅盤的方位定義如下圖：



由於逆時鐘方向定義為正的角度，所以在應用 atan2 時，須將 X 軸的數值乘上-1，使用者也可以依自己喜好去做方向定義。由於 atan2 回傳值為徑度，這邊希望以角度值觀察，因此使用：

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

將數值轉換為角度，這樣就可以用來做方位的判斷了。讀者需要注意，地球上的「地磁北」不一定等於方向正北喔，地球上各地會有不同的磁偏角，所以如果要判斷精確的方位，還需考慮磁偏角的因素；磁偏角的資訊可以在此處查詢：

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

loop( )內最後使用方位角度的數值，做 LED 亮度的控制便可以達到簡易指北針的功能；讀者也可以試試使用其他輸出裝置，如 LCD、蜂鳴器等等，也可以有不同的呈現效果喔。

讀者可能也已經發現，這個範例內沒有用到加速度計的值？從上述的數學式來看，能成功計算出較佳的角度只有在電子羅盤接近水準放置時才行，那如果感測器有傾斜，甚至換成 Z 軸在水準時怎麼辦呢？這就需要更複雜的計算方式了，例如靜態的應用可加入三軸加速度計的傾斜角做換算，動態應用甚至需要加入三軸陀螺儀做輔助等等。有興趣的讀者可以搜尋「IMU 演算法」、「感測器融合演算法」等等的關鍵字去做更深入的閱讀喔。