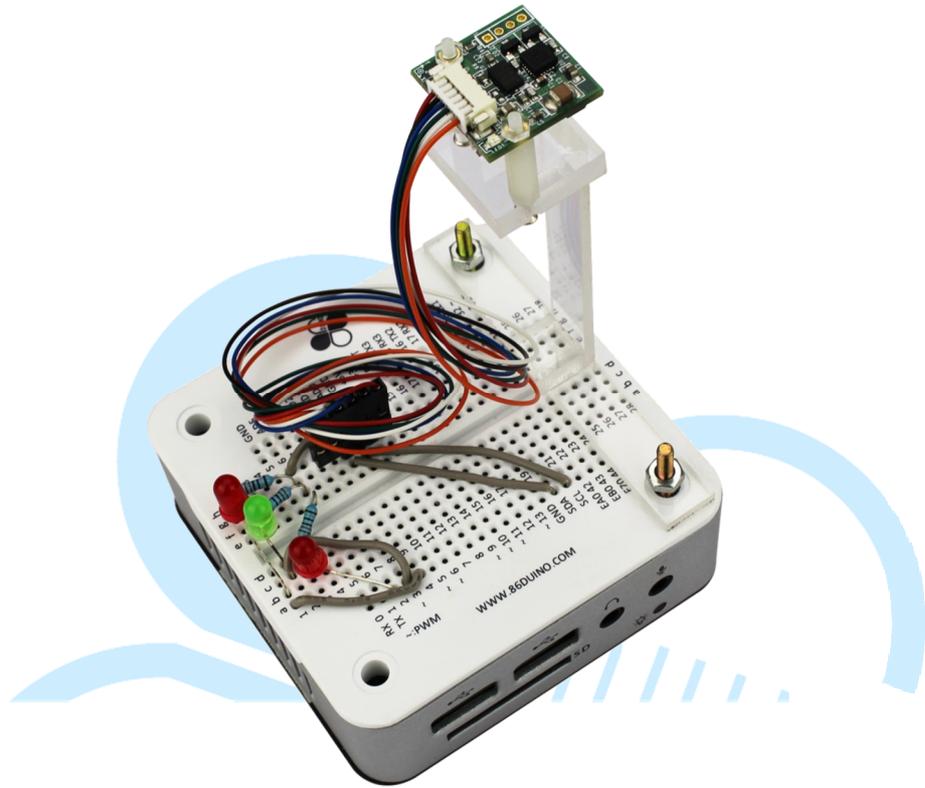


EduCake 的 I2C 通讯



一、 I2C 通讯原理简介

前面几个章节曾经介绍过 86duino EduCake 的串行通讯功能，使用的是「Serial」的程序对象，硬件上则是使用 TX/RX 的脚位元做数据的传输与接收。本章节要讲的则是另一种形式的串行通讯功能，称为「I²C (Inter-Integrated Circuit)」，中文虽然可翻译为「内部整合电路」，但一般还是习惯使用「I²C」称呼。

I²C 通讯接口与 Serial 通讯接口不同的地方，硬件部分在于 Serial 使用不同的脚位做数据传输 (TX 脚位) 与接收 (RX 脚位)，传输时格式则有每秒位数 (鲍率)、同位检查、数据位、结束位等等参数。I²C 虽然也是使用两条硬件线路，但其中一条为资料线，另一条则为频率线，因此只能做「半双工」的通讯，通讯

格式也与 Serial 不同；硬件线路上数据线脚位元通常标示为「SDA」，频率线脚位元则标示为「SCL」，用作 SDA 数据的同步用途。I²C 通讯装置分为 Master(主端)与 Slave(从端)，双方透过 SCL 讯号以得知何时可以读取有效数据。I²C 通讯的相关参数主要叙述如下：

- **速度模式：**

SCL 的频率会由 Master 装置发起与决定通讯速度，当然 Master 装置也得看 Slave 端的规格去决定可用的速度值；视装置的不同，有几种常用的频率速度：标准模式（100 Kb/s）、低速模式（10 Kb/s）、快速模式（400 Kbit/s）等等。

- **Slave 装置地址：**

代表 Slave 装置的特定识别号码，Master 端可藉由此号码与特定的 Slave 装置通讯。I²C 设计原始保留了 16 bits 的地址空间，但较常使用 7 bits 或 8 bits 地址，实际使用需注意规格表上的定义，否则可能会发现通讯时没有反应喔。

- **Slave 装置的缓存器地址：**

Slave 装置有许多的缓存器地址，有的用在装置的设定值，有的则是此装置测量到的数据等等。写入与读取特定的装置缓存器内容，便是 I²C 通讯方式的例行公事所在。

I²C 的 Slave 端装置通常会有各自定义的地址（有的装置是直接固定地址，有的可以透过硬件配置地址），另外也有许多制造厂商自行定义的缓存器地址；读者使用此类装置时须查阅装置的规格表、使用说明等等，才能知道如何与特定装置做通讯，以及了解如何解读这些数值；但通讯方式主要有两种共通的动作程序：

- **Master 端将某段数据写入 Slave 端特定的缓存器地址内：**

用在进行设定 Slave 装置的参数、模式等等。

- 主端指定由从端特定缓存器地址读取一段数据：

用在做 Slave 装置测量数值的读取，或读取装置设定值等用途。

I²C 的通讯格式与程序如下图 1 所示：

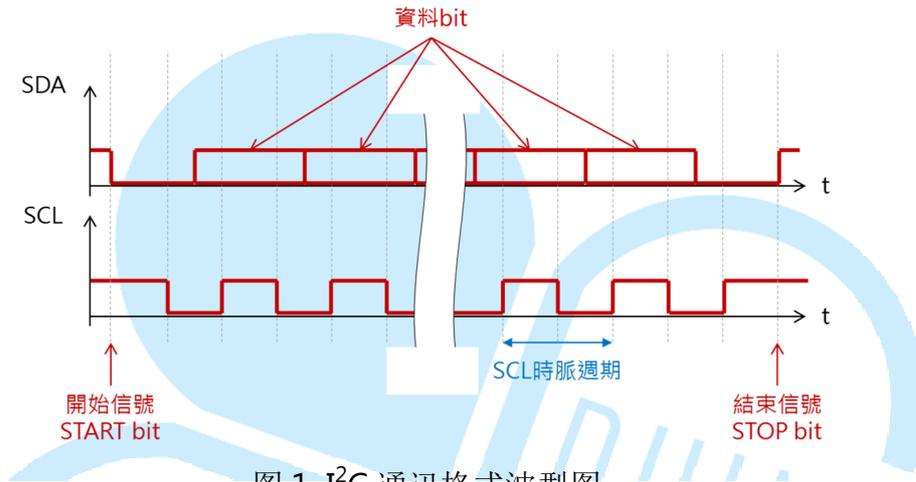


图 1. I²C 通讯格式波型图

由于 I²C 硬件线路与装置参数特性，多个 I²C 装置通讯可以使用称为「Daisy Chain」的联机方式，如图 2 所示；也就是同一条 I²C 总线 (SDA、SCL) 上面可并联多个地址不同的装置，Master 端不需与每个 Slave 装置都使用一条专属的总线。通讯时由于格式指定了特定的装置地址与缓存器地址，因此只有符合地址的 Slave 装置才会有响应动作，是相当省线路的一种通讯方式；但由于通讯格式较简易与缺乏容错性，且讯号也会受硬件线路衰减影响，缺点是仅适用于近距离的通讯应用，一般是在几公尺的距离内使用。

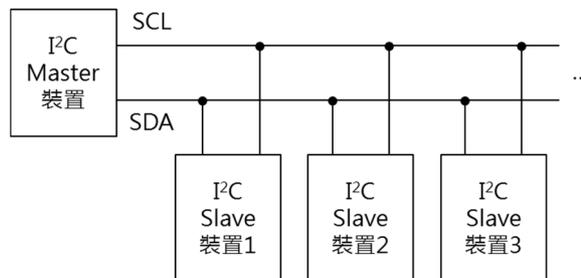


图 2. 多个 I2C 装置通讯联机方式

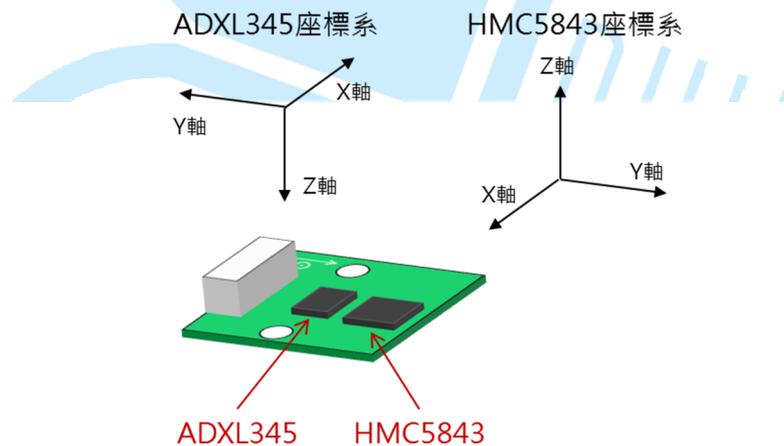
在 86Duino EduCake 上进行 I²C 通讯时，当然不必劳烦使用者去手动控制那些波型，只要使用到「Wire.h」这个函式库，以及内建的几个函式：Wire.begin()、Wire.beginTransmission(装置地址)、Wire.write(byte 资料)、Wire.endTransmission()、Wire.requestFrom(装置地址, 数据 byte 数)、Wire.Read()即可与装置做通讯，以下几个范例就实际使用一个 I²C 的传感器装置，来做 86Duino EduCake 通讯的程序练习吧。



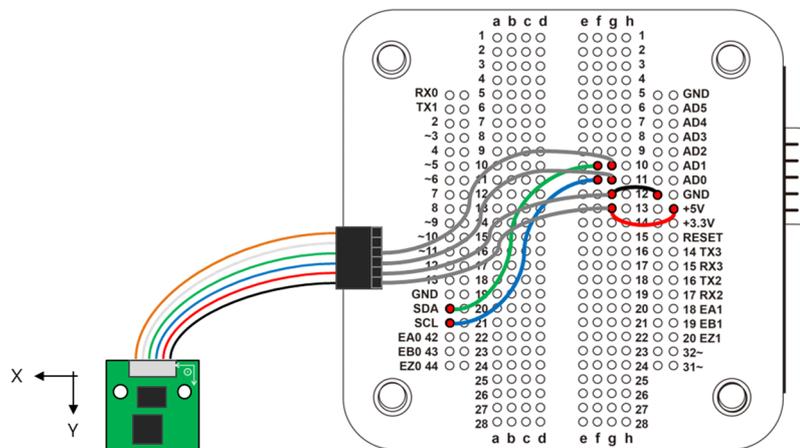
二、 第一个程序 – 使用 RM-G144 的加速度计

第一个范例程式，先来练习如何使用 86duino EduCake 的 I²C 通讯流程，这里我们选用的 Slave 端装置为 RM-G144 传感器，此装置含有三轴电子罗盘（compass）、三轴数字加速度计（accelerometer）两种 IC，电子罗盘 IC 型号为 HMC5843，数字加速度计 IC 的型号则是 ADXL345。由于这两个 IC 都是 I²C 的通讯接口，并联在同一个 I²C 总线上，所以使用时只要对不同的装置地址进行存取，就可以与两种传感器做通讯啰。

使用传感器前，先来认识一下传感器本身的坐标定义；RM-G144 上头两个 IC 的坐标正负号刚好相反，如下图所示；因此之后使用数值的时候得注意方向的问题。



我们首先来练习加速度计 IC 的通讯方式，读者请先依下图接线：



读者从上图可以观察到，I²C 通常会有几个一定会使用到的接线：Vcc（红线）、GND（黑线）、SCL（蓝线）、SDA（绿线），白线跟橘线在这边没有使用到。SCL、SDA 线分别接到 EduCake 上标示有 SCL、SDA 的地方即可。接着请打开 86Duino Coding IDE，输入以下程序代码：

```
// 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

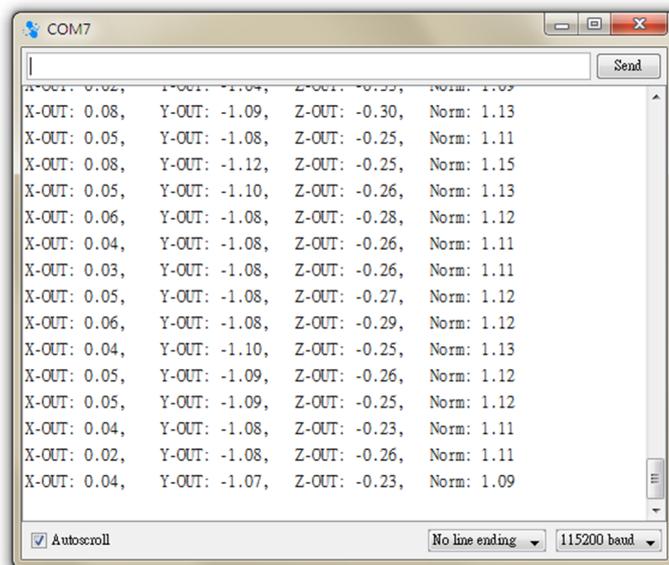
    int count = 0;
    while(Wire.available() && count < 6){
        temp[count] = Wire.read();// 接收傳回來的資料
        count++;
    }
    /*
    temp[0] : X LSB   temp[1] : X MSB
    temp[2] : Y LSB   temp[3] : Y MSB
    temp[4] : Z LSB   temp[5] : Z MSB
    */
    // 處理資料
    // ADXL345 回傳，每軸向分為兩個 byte
    // 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
}
}
```

此范例程式功能为每 100ms 读取一次 RM-G144 上的数位加速度计，读取并换算好三轴的数值，再经由 Serial 传输至 Serial Monitor 观看。编译并上传程序至 86Duino EduCake 之后，打开 Serial Monitor，执行结果如下图：



The screenshot shows a Serial Monitor window titled 'COM7' with a 'Send' button. The window displays a list of acceleration data points. Each line contains four values: X-OUT, Y-OUT, Z-OUT, and Norm. The X-OUT values range from 0.02 to 0.08, Y-OUT values range from -1.07 to -1.12, Z-OUT values range from -0.23 to -0.30, and Norm values range from 1.09 to 1.15. The window also has a 'Autoscroll' checkbox checked and a baud rate of 115200.

X-OUT	Y-OUT	Z-OUT	Norm
0.02	-1.09	-0.30	1.09
0.08	-1.09	-0.30	1.13
0.05	-1.08	-0.25	1.11
0.08	-1.12	-0.25	1.15
0.05	-1.10	-0.26	1.13
0.06	-1.08	-0.28	1.12
0.04	-1.08	-0.26	1.11
0.03	-1.08	-0.26	1.11
0.05	-1.08	-0.27	1.12
0.06	-1.08	-0.29	1.12
0.04	-1.10	-0.25	1.13
0.05	-1.09	-0.26	1.12
0.05	-1.09	-0.25	1.12
0.04	-1.08	-0.23	1.11
0.02	-1.08	-0.26	1.11
0.04	-1.07	-0.23	1.09

程序一开始须先写「`#include <Wire.h>`」，才能使用 I²C 相关的功能函数；`int acc_address` 变量则用来存 ADXL345 的地址，由于程序许多地方需使用地址数据，使用变量当作地址可以较容易修改程序，也是读者可以记起来的一个小技巧。整体流程可以分为两阶段：初始化装置→重复读取内容，由于这两个动作经常会使用，也可能会用到其他程序项目中，所以此处把所有 ADXL345 相关的动作包装成 `G144_Acc_Init()`、`G144_Acc_Read()` 两个函式，读者之后就可以方便使用这两个动作函数做作其他用途啰。

初始化阶段在 `setup()` 内完成，先呼叫 `G144_Acc_Init()` 做初始化设定；这边许使用前面提到的「**Master 端将某段数据写入 Slave 端特定的缓存器地址内**」动作，需做的流程语法为：

```
Wire.beginTransmission(装置位址); → 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 装置才会对这个指令有反应，并执行动作；ADXL345 的初始化须执行三次这个动作：

1. 对 0x2d 地址写入数据 0x28:

设定 Power Control 缓存器为 0010 1000，Link Bit=1, Measure Bit=1。

2. 对 0x31 地址写入数据 0x08:

设定 Data Format 缓存器为 0000 1000，指定感测范围为 Full_Resolution，

FULL_RES Bit=1，表示 scale factor（换算参数）为 4 mg/LSB，Range Bits（有效测量范围）为 00 (±2g)，此时数据为 10 bits 分辨率，

Range Bits 的 Bit 0 与 Bit 1 可设定 4 种量测范围 (±2g ~ ±16g)

分辨率随范围改变（10 bits ~ 13 bits），视实际使用状况去设定；

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

3. 对 0x38 地址写入数据 0x00:

设定 FIFO Control 缓存器为 0000 1000，为 bypass mode。

这边须注意，由于 IC 规格表内提到每次写入数据后都需间隔至少 100us 才能进行下一个动作，所以每次写入动作后使用 delayMicroseconds(100) 做延迟时间。

loop() 内持续呼叫 G144_Acc_Read() 做读取；这边许使用前面提到的「主端指定由从端特定缓存器地址读取一段数据」动作，需做的流程语法为：

一开始先指定要从特定的装置地址读取某个缓存器地址，然后使用

```
Wire.beginTransmission(装置位址);  
Wire.write(指定要讀取的暫存器位址);  
Wire.endTransmission();  
  
Wire.requestFrom(装置位址, 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++;  
}
```

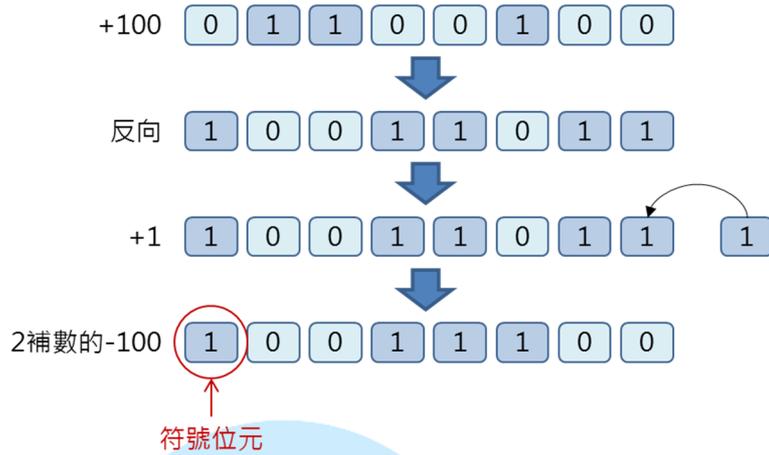
Wire.requestFrom(装置地址, 数据 byte 数)语法，指定 Slave 装置回传数个 byte 的数据。之后使用 Wire.available() 检查是否有资料回传，搭配 Wire.read() 方式读取数值到某个变量中。对于这个范例，加速度计的数值

被储存在 0x32 地址以下连续 6 个字段，因此数据 byte 数设定为 6，使用 while() 循环语法连续读取 6 个数据，并存到 temp[] 矩阵当中。

除了上述两个动作之外，数据换算也是重点之一，程序一开始宣告的 unsigned int temp[6] 是用来暂存传感器数值的变量，double acc_value[3] 则用来存放处理过后的三轴数值。由于 0x32 地址以下 6 个字段依序储存的是：X LSB、X MSB、Y LSB、Y MSB、Z LSB、Z MSB 六个 byte 数据，因此 G144_Acc_Read() 最后一阶段动作是进行数值的换算。ADXL345 每轴数据分为 MSB、LSB 两个 byte，须先进行组合再换算；而传感器数值使用 2 补码表示，数值有正负号，所以须分正负状况讨论，语法为：

```
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[j] = (double)temp_value * 4 / 1000;
```

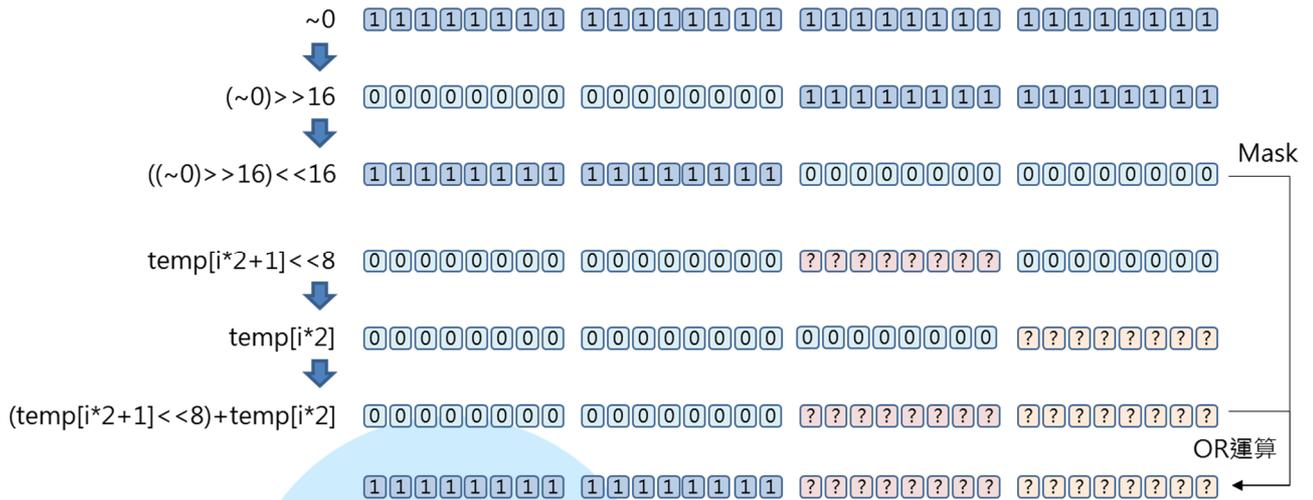
这边需要先解释一下「2 补码法」，如下图：



以正数「100」为例，2 补码表示的「-100」是将 100 的二进制值反向再加 1 的结果，读者可以简单验证图中 100 与 -100 的 2 进制数值相加是否刚好为 0 呢？这就是一般计算机内常用的二进制负数表示法则。

回到程序，第一行 `if()` 语法内的「`temp[i*2+1] & 0x80) != 0`」用在检查 MSB 的最左边位是否为 0，若是 0 表示这个资料是正数，反之为负数。由于 `Wire.read()` 每次只读一个 byte 的数据，重组数据须将 MSB 左移 8 个位，再与 LSB 相加。当数据重组为 int 型态（86Duino EduCake 的 int 型态是 32 bits）时，左方空间须做「符号延伸」，也就是把负数左边全部空位都补 1。

因此当数值以 2 补码法表示时，若为负数变量须做符号延伸，怎么改变内存中的「位」数值呢？这边就需要使用「位元运算」了，作法如下：



首先制造一个屏蔽（Mask）为 b'11111111 11111111 00000000 00000000，然后再与重组后的「(MSB<<8)+LSB」做 OR 运算，便可将左方全部补 1 啰。

最后一个步骤是将数值换算成实际测量单位，前面在做装置初始化时选择了 Full Resolution 模式，此模式下的换算参数（scale factor）为 4mg/LSB，因此 $acc_value[i] = (double)temp_value * 4 / 1000$ 便可以将数值换算为 g 了（1 单位的重力加速度 = $9.8m/s^2$ ）；这边运算时读者需要注意变量型态的转换，若没有先转成浮点数就做乘/除法，小数点是不会被保留的。这样便完成了 RM-G144 上头 ADXL345 加速度计的设定与读取啰。

（ RM-G144 详细规格与参数可参考 <http://www.roboard.com/G144.html>，读者若有兴趣的话也可以查询这些资料进行阅读。）

三、第二个程序 – 使用 RM-G144 的电子罗盘

第二个范例程式我们接着做 RM-G144 上头的 HMC5843 电子罗盘读取与设定，接线不用更动；读者请打开 86Duino Coding IDE，输入以下程序代码：

```
// 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();

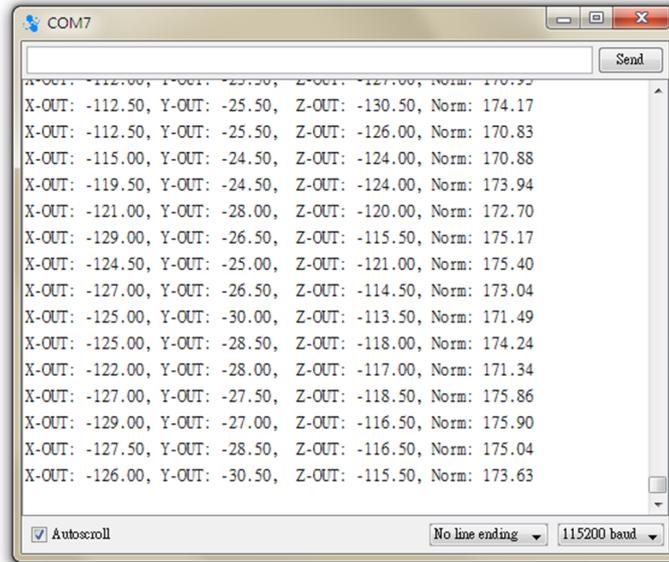
    Wire.requestFrom(mag_address, 6); // Read data form
Conversion Result Register, Data : 12bits
    int count = 0;
    while(Wire.available() && count < 6){
```

```

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
}
}
}

```

此范例程式功能与第一个类似，每 100ms 读取一次 RM-G144 上的电子罗盘，读取并换算好三轴的数值，再经由 Serial 传输至 Serial Monitor 观看。编译并上传程序至 86Duino EduCae 之后，打开 Serial Monitor，执行结果如下图：

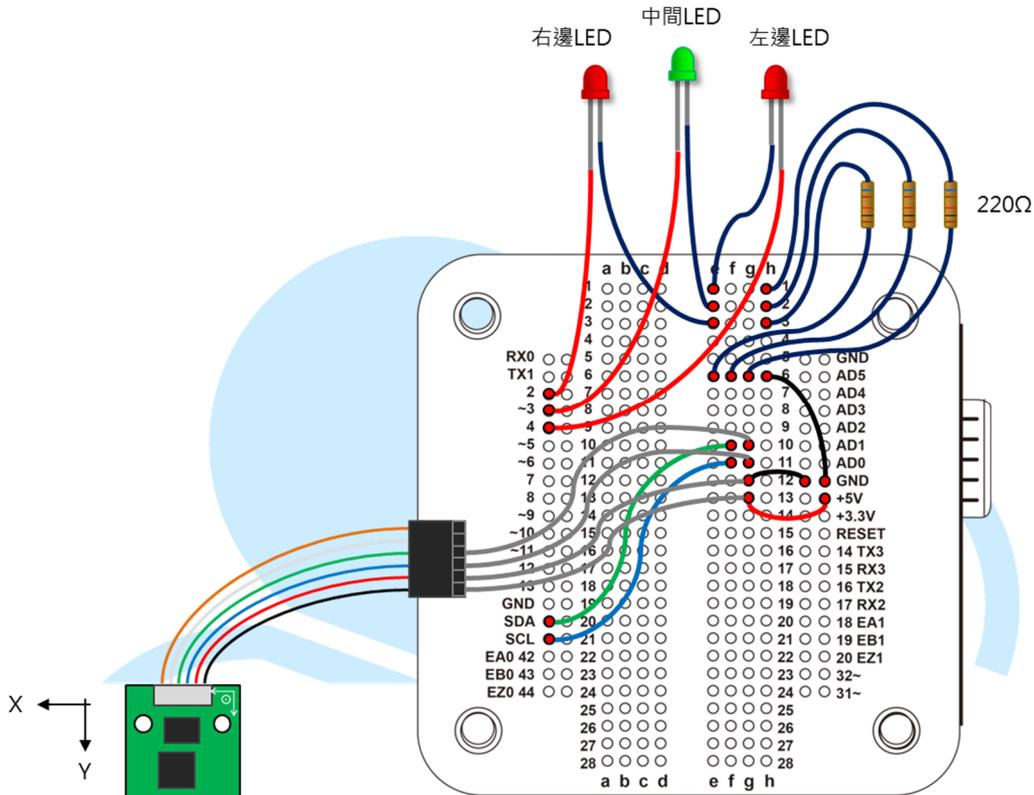


```
COM7
X-OUT: -112.50, Y-OUT: -25.50, Z-OUT: -130.50, Norm: 174.17
X-OUT: -112.50, Y-OUT: -25.50, Z-OUT: -126.00, Norm: 170.83
X-OUT: -115.00, Y-OUT: -24.50, Z-OUT: -124.00, Norm: 170.88
X-OUT: -119.50, Y-OUT: -24.50, Z-OUT: -124.00, Norm: 173.94
X-OUT: -121.00, Y-OUT: -28.00, Z-OUT: -120.00, Norm: 172.70
X-OUT: -129.00, Y-OUT: -26.50, Z-OUT: -115.50, Norm: 175.17
X-OUT: -124.50, Y-OUT: -25.00, Z-OUT: -121.00, Norm: 175.40
X-OUT: -127.00, Y-OUT: -26.50, Z-OUT: -114.50, Norm: 173.04
X-OUT: -125.00, Y-OUT: -30.00, Z-OUT: -113.50, Norm: 171.49
X-OUT: -125.00, Y-OUT: -28.50, Z-OUT: -118.00, Norm: 174.24
X-OUT: -122.00, Y-OUT: -28.00, Z-OUT: -117.00, Norm: 171.34
X-OUT: -127.00, Y-OUT: -27.50, Z-OUT: -118.50, Norm: 175.86
X-OUT: -129.00, Y-OUT: -27.00, Z-OUT: -116.50, Norm: 175.90
X-OUT: -127.50, Y-OUT: -28.50, Z-OUT: -116.50, Norm: 175.04
X-OUT: -126.00, Y-OUT: -30.50, Z-OUT: -115.50, Norm: 173.63
Autoscroll No line ending 115200 baud
```

程序流程与第一个几乎相同，主要差别在 HMC5843 的装置地址在 0x1e，一样是 7bit 地址；而初始化设定时写入的参数与缓存器地址也不同，只需在 0x02 缓存器写入 0x00，将模式设定成 continue-measure mode 即可。读取时也稍有不同，测量数值的缓存器地址由 0x03 开始，而且三轴数据的 LSB 与 MSB 存放顺序刚好跟 ADXL345 相反，所以数据重组时须注意这边的差异；换算参数则是 0.5 milli-gauss/LSB，换算完后是 milli-gauss 的单位。读者请仔细看传感器的 PCB 板上有标示一个小小的坐标，这就是 HMC5843 的参考坐标系；这个电子罗盘测量的是正北方向的磁场，当 Y 轴面向磁场北极时，Y 轴向会有最大读值，而 X 轴会接近 0；读者可以自行尝试多种不同的摆放方式看看数值的变化状况。

四、 第三个程序 – 利用传感器判断方位

成功读取电子罗盘数值之后，就可以拿它来做些应用了，请读者先增加接线如下图所示：



接着在 86Duino Coding IDE，输入以下程序代码：

```
// 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
    }

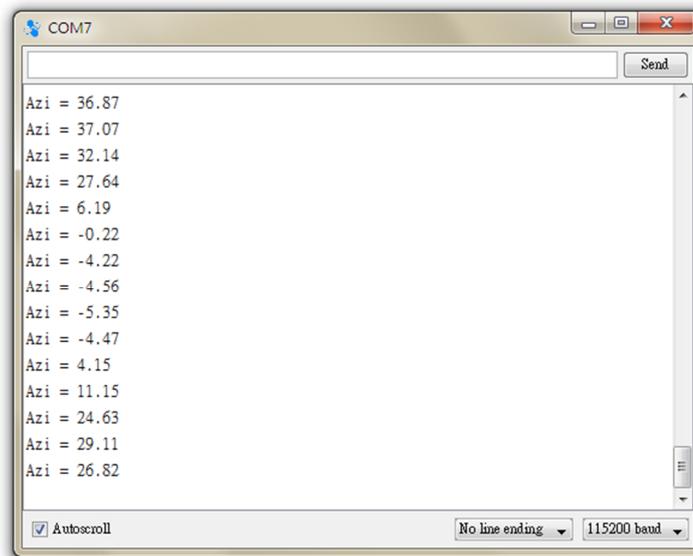
    delay(100);
}

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;  
}
```

其余四个函式: G144_Acc_Init()、G144_Mag_Init()、G144_Acc_Read()、G144_Mag_Read()与前两个范例程式相同, 就不再重复解释啰。

此范例程式功能为读取 HMC5843 的数值, 并且计算目前传感器面向的方位, 当传感器偏向西半侧(角度 < -10)时, 则右边 LED 亮起提醒使用者要朝顺时针方向走, 反之亮起左边 LED; 如果目前角度在地磁正北方的 ±10 内, 则亮起中间 LED, 表示目前方向朝向地磁北方。编译并上传程序至 86Duino EduCae 之后, 打开 Serial Monitor, 执行结果如下图:

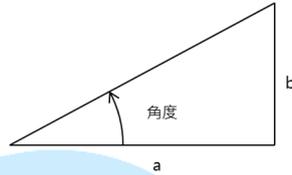


The screenshot shows a Serial Monitor window titled 'COM7'. The window contains a list of azimuth values printed on separate lines. The values are: 36.87, 37.07, 32.14, 27.64, 6.19, -0.22, -4.22, -4.56, -5.35, -4.47, 4.15, 11.15, 24.63, 29.11, and 26.82. At the bottom of the window, there are settings for 'Autoscroll' (checked), 'No line ending', and '115200 baud'.

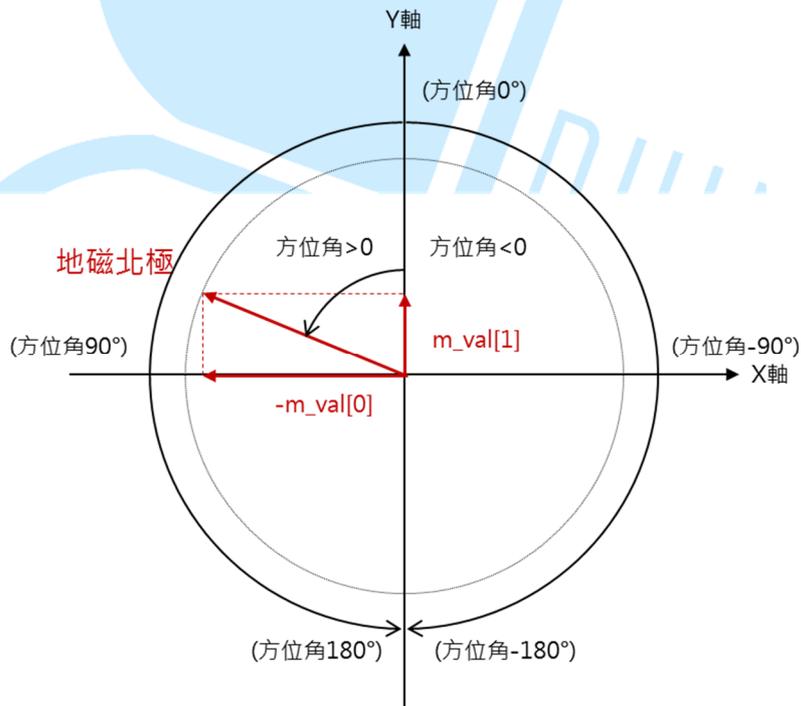
判断方向的程序在 double GetAzimuth(double *m_val)这个函式内, 传入函式的参数「*m_val」是矩阵的指标, 函式内利用语法:

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

做角度计算， $\text{atan2}(b, a)$ 为反正切三角函数，如下图定义，可从 a、b 长度计算出夹角，单位为弧度（radius）。此函数功能相当于 $\tan^{-1}(b/a)$ ，但可以得到 $-\pi \sim \pi$ 的角度值。



电子罗盘 X、Y 轴的读值刚好可以应用这个数学式，我们将电子罗盘的方位定义如下图：



由于逆时针方向定义为正的角度，所以在应用 atan2 时，须将 X 轴的数值乘上-1，使用者也可以依自己喜好去做方向定义。由于 atan2 回传值为弧度，这边希望以角度值观察，因此使用：

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

将数值转换为角度，这样就可以用来做方位的判断了。读者需要注意，地球上的「地磁北」不一定等于方向正北喔，地球上各地会有不同的磁偏角，所以如果要判断精确的方位，还需考虑磁偏角的因素；磁偏角的信息可以在此处查询：

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

loop()内最后使用方位角度的数值，做 LED 亮度的控制便可以达到简易指北针的功能；读者也可以试试使用其他输出装置，如 LCD、蜂鸣器等等，也可以有不同的呈现效果喔。

读者可能也已经发现，这个范例内没有用到加速度计的值？从上述的数学式来看，能成功计算出较佳的角度只有在电子罗盘接近水平放置时才行，那如果传感器有倾斜，甚至换成 Z 轴在水平时怎么办呢？这就需要更复杂的计算方式了，例如静态的应用可加入三轴加速度计的倾斜角做换算，动态应用甚至需要加入三轴陀螺仪做辅助等等。有兴趣的读者可以搜寻「IMU 算法」、「传感器融合算法」等等的关键词去做更深入的阅读喔。