EduCake 的 Analog I/O 腳位介紹與功能應用

一、 Analog I/O 介紹

- **類比 vs 數位：**

前篇已經介紹過 86Duino EduCake 的基本規格、IDE 程式撰寫介面使用方式，與 Digital 腳位的基本使用方式，本篇就來介紹類比(analog)輸入/輸出腳位的功能以及應用方式。

介紹「類比」信號前，須先與所謂「數位」信號做個比較，前篇所提到的「數位」信號，一般而言是具有兩種狀態的電壓訊號，高於某個程度的電壓為 HIGH(一般以 1 表示)，低於某程度的電壓則為 LOW(一般以 0 表示)，因此以 86Duino 為例，digitalWrite( )可以使某個腳位輸出 HIGH 或 LOW 的電壓，而 digitalRead( )則可以讀取某個腳位電壓 HIGH 或 LOW 的狀態，對於開關、按鈕這類僅有兩種狀態表示方式的信號而言已足夠使用。

而本篇文章所提的類比信號，指的是具有連續狀態的數值，例如：亮度、溫度、濕度、響度、長度、角度、重量……等自然界常見的物理量，各類型的感測器則會負責將物理量轉變為電壓或電流；但程式執行時所需的數值是 0 與 1 組成的數位信號，因此當某個應用須取得上述「連續狀態」物理量給程式使用時，便需要稱為「類比數位轉換器」(Analog to Digital Converter，簡稱 ADC)的電路來做轉換，下面以一 ADC 電路作為範例解釋轉換原理：

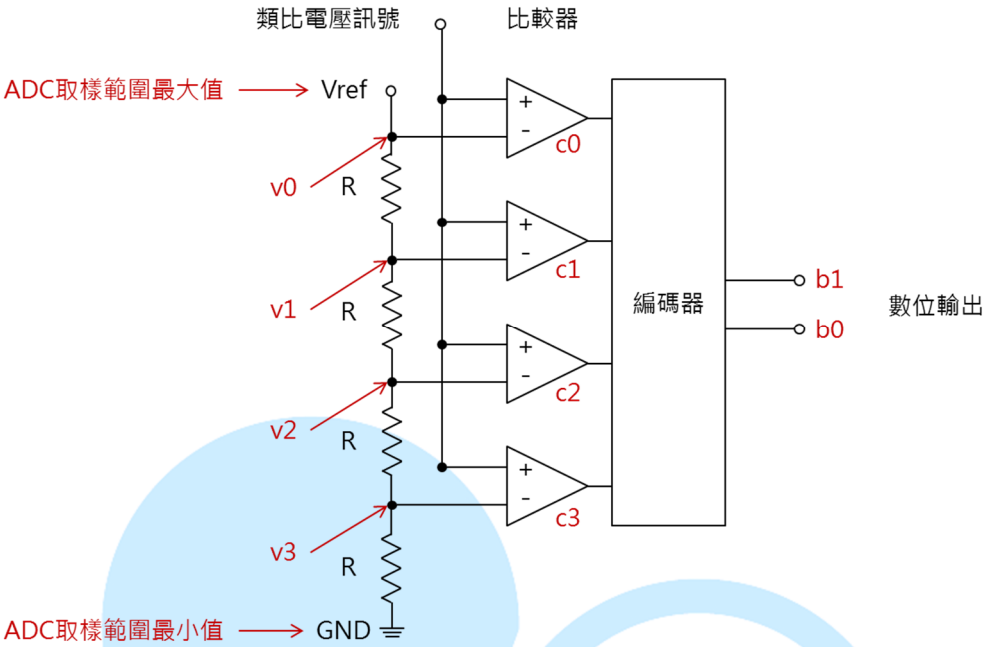


圖 1

圖 1 電路稱為「Flash ADC」，是概念較易理解的一種。這裡舉擁有四個比較器的當作範例，輸入處為類比電壓訊號與參考電壓訊號 Vref；每個比較器「-」端接到 v0~v3，分別為 Vref、Vref(3/4)、Vref(1/2)、Vref(1/4)，當「+」端的電壓訊號比「-」端高，則比較器輸出為 1，反之為 0。FLASH 編碼器則負責把 c0~c3 的輸出轉變成對應的二進位編碼，此範例由於比較器只有 4 階，因此實際二進位編碼只有 2 位元。若須 4 位元輸出，則須 16 個比較器，讀者可依此類推。

以 Vref 等於 5V 為例，v0~v3 分別為 5V、3.75V、2.5V、1.25V；當類比信號為 1.5V，則比較器 c0~c3 輸出 0/0/0/1，二進位值為 01(十進位值為 1)；當類比信號為 3.9V，則比較器 c0~c3 輸出 0/1/1/1，二進位值為 11(十進位值為 3)，更多比較器的狀況依此類推。所以此例的 ADC 輸入範圍為 0~5V，轉換位元數 2，僅有 4 階的解析度，每階電壓為 5/4 = 1.25V。

ADC 的轉換電路與設計原理多樣化，也是一門學問，本文僅簡介其原理與應用，讀者若有興趣鑽研可搜尋相關資料閱讀。

- **ADC 規格與基本換算方式：**

提到 ADC 規格時，一般會描述 ADC 能接受的輸入範圍、轉換位元數等；由前段的 ADC 原理解釋便能了解，任何低於或超出 ADC 輸入範圍的數值只能用數位最大或最小值表示，而轉換位元數則影響取樣的解析度，位元數越高，越能細緻地表示取樣的信號。

舉例 1：某腳位的 ADC 輸入範圍為電壓 0~5V，而轉換位元數為 10bits，則 0V 時取樣的數位值為 0(二進位值為 0000000000)，而 5V 時取樣的數位值為 1024(二進位值為 1111111111)，0~5V 中間共有 2 的 10 次方=1024 個解析度，表示此 ADC 取樣解析度為 5/1024，約等於每刻度有 0.004883V，可以想像一下一根溫度計在 0~5 度之間細分為 1024 個等分；當讀取到的數值為 1000 時，代表取樣的原始信號為 1000*5/1024，約等於 4.883V。

舉例 2：某腳位的 ADC 輸入範圍為電壓 1~7V，而轉換位元數為 12bits，則每刻度有(7-1)/4096，約為 0.00146V 的電壓解析度；當讀取到的數值為 1000 時，代表取樣的原始信號為 1000*6/4096+1，約等於 2.465V。

因此使用 ADC 時，須了解它的基本規格才能進行數值的換算；以 86Duino EduCake 為例，輸入範圍為電壓 0~3.3V，解析度預設為 10 位元。

- **感測器應用前須知：**

前面提到各類型的感測器會將物理量轉變為電壓或電流訊號，但 ADC 通常是讀取電壓訊號，況且各種感測器訊號範圍也不一定符合 ADC 的輸入規格，因此感測器使用前，另一個注意事項是須將輸出訊號對應到 ADC 的輸入範圍，以下舉例來做說明。

舉例 1：電阻式的感測器，如一般旋轉式可變電阻、壓力感測器、光敏電阻、電阻式溫度計等等，針對環境一定範圍的變化改變本身電阻值。使用時可如下面圖 2 所示，稱為「電阻分壓法」，其中 Vcc 通常使用接近 ADC 輸入最大電壓值，例如 EduCake 為 3.3V。感測器至 ADC 輸入電壓為 Vcc*R1/(R1+R2)，圖 2(b)可變電阻的 R1+R2 為定值，而圖 2(a)中的 R2 可自訂，通常選擇 1KΩ左右當作限制電流用，也不宜與 R1 相差太多。如果 Vcc 使用其他固定電壓值，也可依照上述原理將 R1 端電壓調整到適合 ADC 的輸入範圍。

圖 2(a)　　圖 2(b)

舉例 2：以測量太陽照度常用的日射計(Pyranometer)來說，常見的是輸出 4~20mA 的規格，這種感測器的特性是它輸出電流信號。這種情況下，可用圖 3 的方式，將感測器串接一電阻 R1，例如 250Ω，則依 V=I*R，可將 4~20mA 轉變為 1~5V，便可讓 ADC 進行取樣，使用時須注意電流的輸出方向。

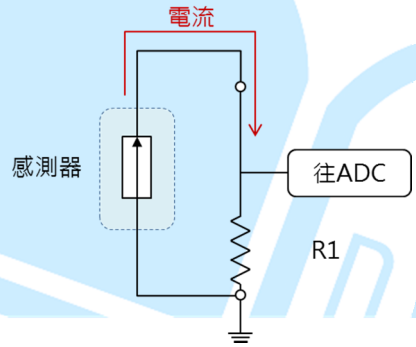


圖 3

- **類比輸出**

前面提到了 ADC 將類比訊號轉換為數位資料進行處理，而類比輸出則剛好相反，是將程式內的數位資料轉換成類比的訊號輸出，一般來說要將數位資料轉換為「純正」的類比訊號，會使用到「數位類比轉換器」(Digital to Analog Converter，簡稱 DAC)的電路，但並非所有輸出裝置都需要使用純正的類比訊號。例如燈泡、直流有刷馬達之類的裝置，只要調整特定時間內電流或電壓通過的時間比例，便可達到調整亮度，或是控制轉速的效果。以 86Duino EduCake 而言，使用的類比輸出也是採用這種稱為「PWM」(Pulse Width Modulation，脈衝寬度調變)的方法，PWM 並不侷限用在達到類比

輸出的效果，之後會有專門文章來做介紹，本文則著重在 PWM 模擬類比輸出的應用。

如圖 4，由上到下分別為 25%、50%、75%、100%的 PWM 脈衝寬度比例(也稱為 duty cycle)，由於人眼的視覺暫留現象，當頻率夠高，PWM 電壓用在 LED 或燈泡上的時候，就會相當於亮度的變化，EduCake 的類比輸出頻率約是 1000Hz。也可以用前篇文章的 digitalWrite( )配合 delay( )、delayMicroseconds( )函式來達到相似的效果，但頻率太低則會出現明顯亮滅情形。

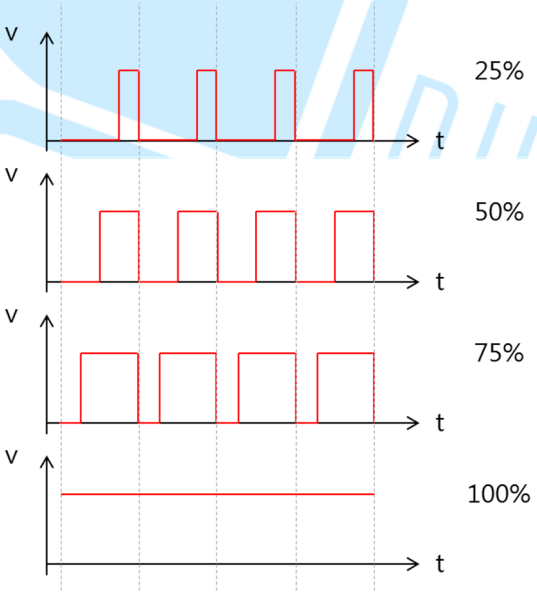
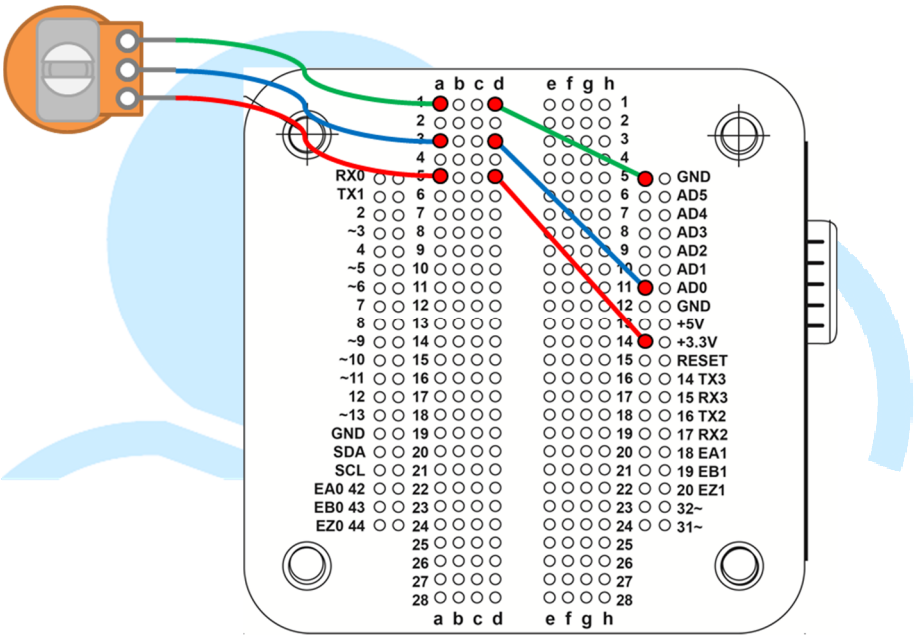


圖 4

講完原理部分，接下來便開始以實際程式操作進行練習。

二、 第一個程式 – 練習 analogRead( )

一開始先使用 EduCake 來練習簡單的類比輸入功能，這裡使用的函式是 analogRead( )，需準備的零件為一般電子材料行就很容易購買到的可變電阻，挑個 10kΩ即可，接線方式如下圖：



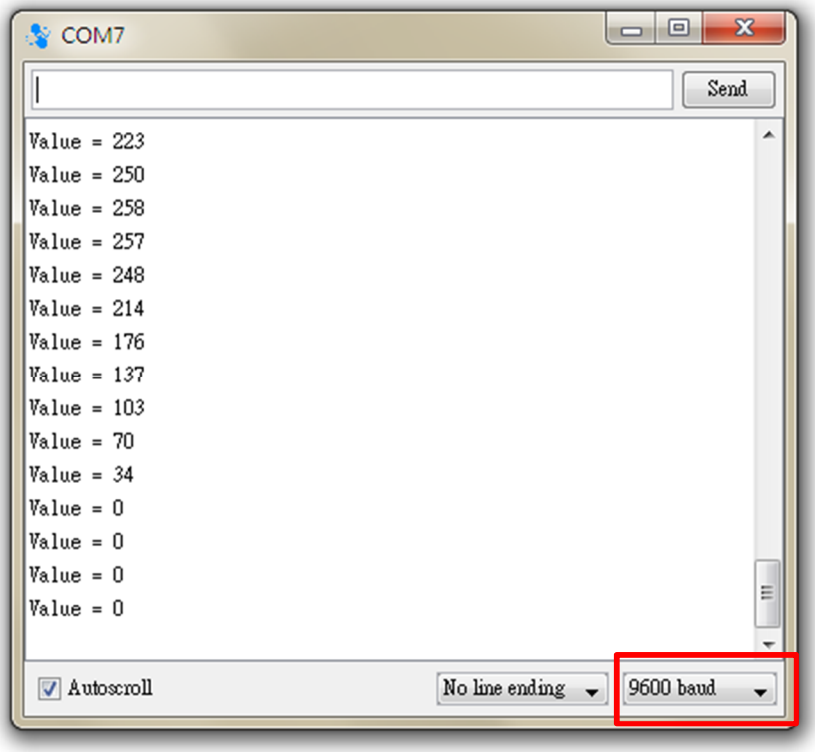
此接線方式就是前面介紹原理的圖 2(b)，請注意通常可變電阻有 3 隻腳，中間那隻腳就是改變電阻比例的碳刷部位，其他兩腳沒有正負極分別，分別接上 EduCake 的 3.3V、GND 即可；但如果反過來接，會影響轉動圓棒時的電阻值大小變化順序，如果配合轉動機構時須注意這點。此範例我們會將 ADC 讀取到的數值，利用與電腦連接的 USB 傳輸線將數值印出來觀察。接著請開啟 86Duino Coding 100 的 IDE 介面，輸入以下程式：

```
const int analogInPin = A0;//  宣告類比輸入腳位編號

int sensorValue = 0;//  宣告儲存 ADC 讀值

void setup() {
```

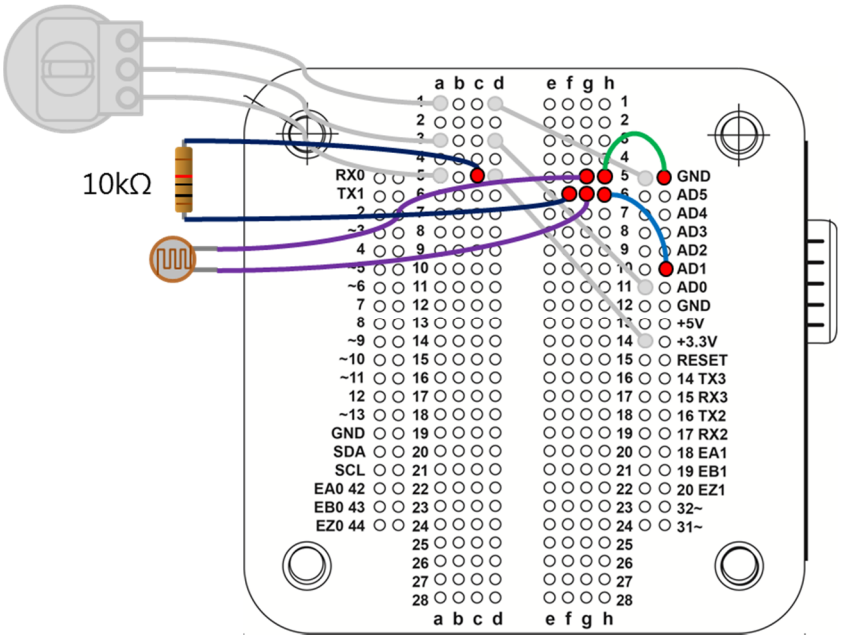
```
  Serial.begin(9600);// 設定與電腦通訊的 Serial Port 速度
}

void loop() {
  sensorValue = analogRead(analogInPin);// 讀取 ADC 值

  Serial.print("Value = " );// 印出字串
  Serial.println(sensorValue, DEC);// 印出數值

  delay(100);// 加上延遲時間
}
```

程式一開始在 setup( )設定與電腦端傳輸的 baudrate(通常稱為鮑率)數值，然後 loop( )裡單純執行 analogRead( )，並將數值以 Serial.print( )、Serial.println( )函式回傳到 Serial Monitor 做顯示。此兩函式差別在於 println 會在印完字串後換下一行，也可以方便閱讀。注意 Serial.println(sensorValue, DEC)這行，第二個參數「DEC」表示將第一個參數以「十進位」印出，讀者也可以嘗試其他例如 BIN、HEX 等，將數值以二進位或 16 進位印出。delay(100)是讓程式迴圈每次暫停 100ms(0.1 秒)，間隔時間依讀者實際程式需要而定。

按下 Upload，上傳程式後，請打開 Tools > Serial Monitor(或用 Ctrl+Shift+M 快捷鍵)，就可以看到一連串的 ADC 讀值囉，如下圖。接著若轉動可變電阻的圓棒，便可發現數值在 0~1023 之間變化。若打開 Serial Monitor 發現沒有反應，可能要檢查一下右下方的「baud」設定有沒有跟程式 setup( )裡的 Serial.begin(9600)一樣喔。之後任何程式都可以用這個方法來做變數監看、除錯，是滿好用的工具。



接著來改變一下感測器種類，利用圖 2(a)的電路來做練習，為了後續幾個程式方便，之前的接線先不拆，新增接線如下圖：



電路使用光敏電阻來測試，也是電子材料行常見的元件，這裡挑的是一般室內光照下約為數十 kΩ的型號，搭配另一固定電阻值為 10kΩ的電阻。

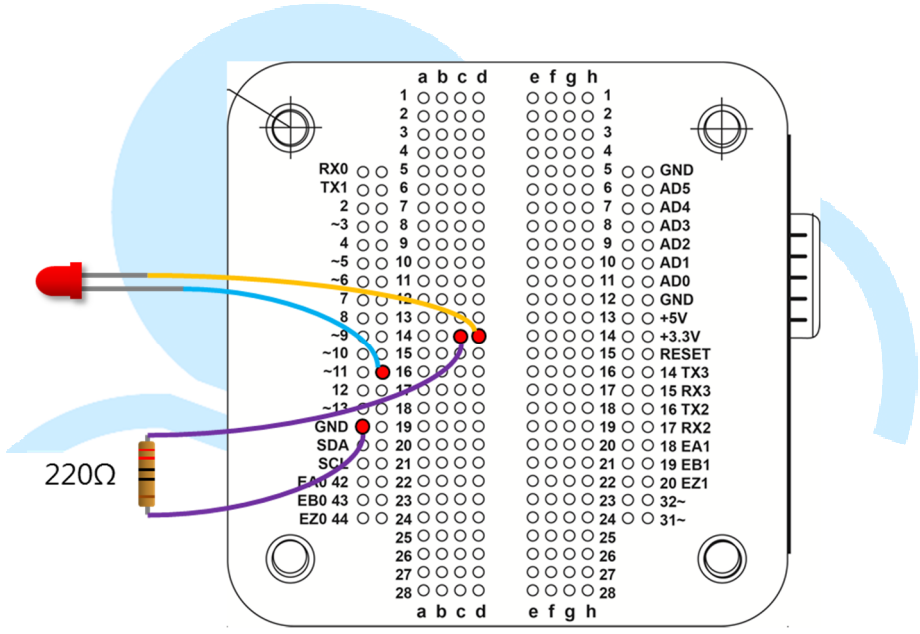
光敏電阻一端接地，一端接到 AD1 腳位，10kΩ電阻一端則接到 3.3V；由於光敏電阻照光時電阻值減小，不照光時電阻值極大(甚至到 MΩ等級)，所以依據電路接法，可預期亮度高時 ADC 讀值減小，而遮蔽時 ADC 讀值變大。程式修改如下：

```
const int analogInPin = A1;//  宣告類比輸入腳位編號
```

基本上只是把類比輸入腳位改成 AD1 而已，測試方法同上例，讀者可以測試看看結果，是不是就如同描述的一樣呢？讀者到這邊應該可以感覺到，簡短幾行程式便可以簡單地讓電路感知外界的變化現象，當然也就可以做更進階的功能囉。

三、 第二個程式 – 練習 analogWrite( )

接下來我們實作一個 LED 呼吸燈效果來練習類比輸出功能。這個範例請讀者準備一顆常見的 LED，顏色可挑紅黃綠之類即可，另外需一個 220Ω 電阻，電阻主用用來保護 LED 用，以免電流過大燒毀 LED。接線圖如下，一樣不拆原有線路，另外接線。



LED「+」端(長腳部位)接至 EduCake 11 腳位，注意板子腳位前方標有「~」的皆為可輸出 PWM 的腳位；接著輸入下方程式：

```
const int analogOutPin = 11;// 宣告類比輸出腳位編號
const float Freq = 0.5;// LED 閃爍頻率
unsigned long time;// 時間變數
byte LED_light = 0;// LED 亮度

void setup() {
}

void loop() {
   time = millis();
```

```
  LED_light = byte(128 + 127 * cos((float)time * 0.001 * 2 * PI *
Freq));// 計算 LED 亮度
  //Serial.print("LED_light = " );// 印出字串
  //Serial.println(LED_light, DEC);// 印出亮度數值
  analogWrite(analogOutPin, LED_light);// 輸出 PWM 波形至 LED

  delay(50);// 加上延遲時間
}
```

從第一行開始宣告一連串呼吸燈會用到的設定值，例如腳位編號、LED 閃爍頻率、時間變數、LED 亮度等，主迴圈裡每次會先取得時間變數，使用 millis( )函式可取得程式開始執行至目前為止的「毫秒數」，然後利用餘弦(或正弦 sin 也行)公式 cos( )，利算每次需要的 LED 亮度，此處的數學計算式為：

LED 亮度 = 128 + 127 * cos(2 * π * freq * time)

類比輸出使用 analogWrite( )函式，參數範圍為 0~255，0 代表 PWM 脈波週期沒有 HIGH 的部分，而 255 代表週期中 HIGH 的比例為 100%，依此類推 128 代表 50%。由於 cos( )函數輸出-1~1 的小數值，因此乘上震幅 127，再加上 128 便可將數值映射到 1~255；2πft 則用在調整 cos( )函數的頻率，f 單位為 Hz，由於 time = millis()取得時間值為毫秒，因此*0.001 將單位變成秒；最後加上 delay( )延遲時間。

此範例設定 LED 閃爍頻率為 0.5Hz，執行此程式後，便可看到 LED 燈以 2 秒一次的頻率進行漸變的亮暗，讀者可以試著加大 delay( )，會發現 LED 燈的亮暗頻率沒變，但變化過程明顯有階段性的感覺，不夠順暢，因此此範例使用 50ms 的延遲時間，程式約以 20Hz 計算並更新 LED 亮度，視覺效

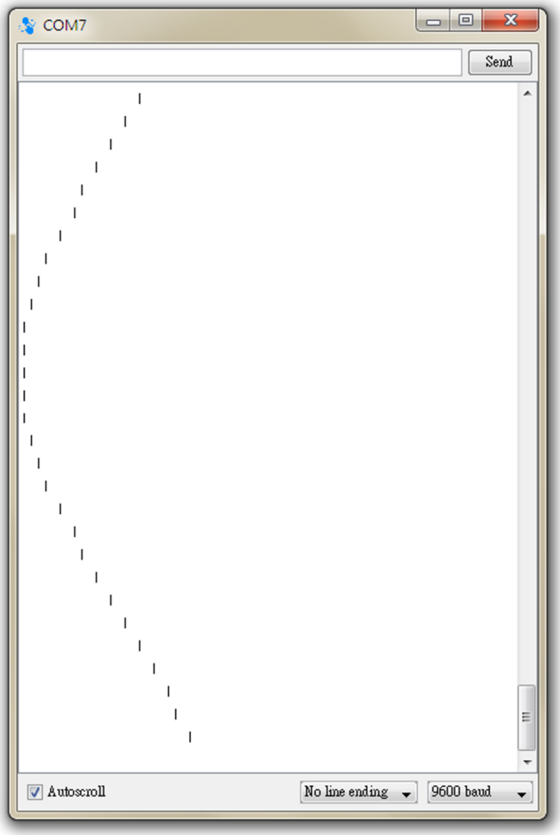
果較佳，這樣就簡單完成一個呼吸燈效果了。另外也示範另一種 Serial Monitor 的用法，讀者可以將上面程式中的

```
//Serial.print("LED_light = " );// 印出字串
//Serial.println(LED_light, DEC);// 印出亮度數值
```

改成：

```
for(int i=0;i<LED_light/10;i++){
   Serial.print(" " );// 印出空白字串
}
Serial.println("|");// 印出符號字串
```

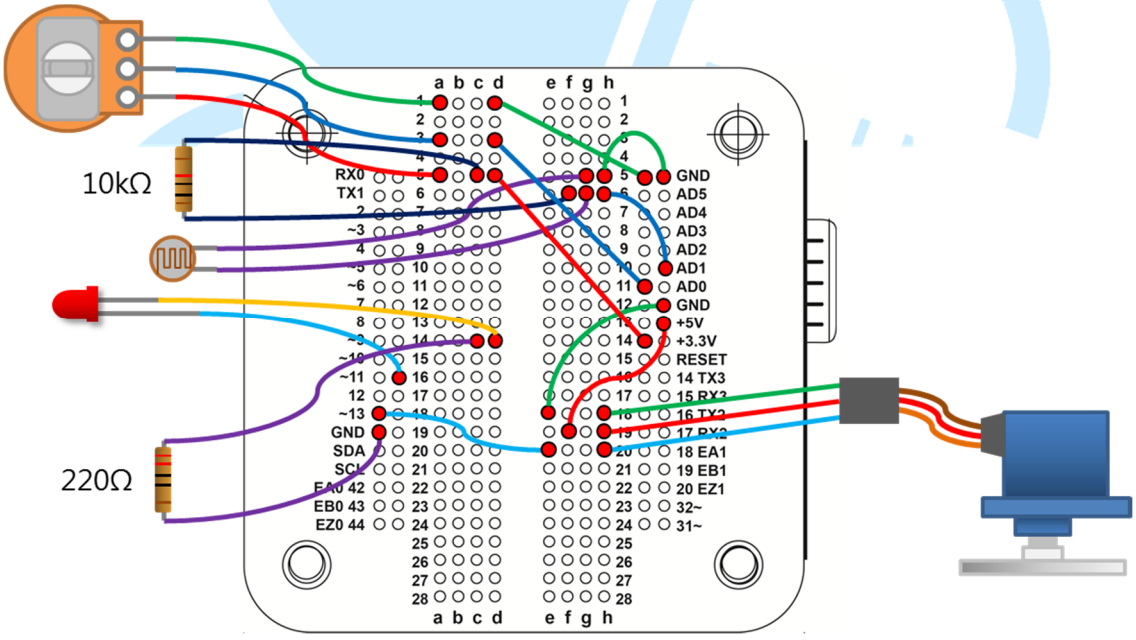
這幾行程式利用 LED 亮度值，讓 Serial.print( )印出數個空白符號，最後再加上一個其他符號，便可以形成長短不一的字串；Upload 程式完成並開啟 Serial Monitor，便會看到一連串符號組成弦波如下圖：



這個波型跟 LED 幾乎是同步的，讀者也可以自行改變計算亮度的公式做出更多變化；之後將專題加上呼吸燈，就像有了心跳一般，也會有作品活起來的畫龍點睛之效呢。

四、 第三個程式

本文最後一個範例程式，將前面幾個用法集大成，並加上另一個零件來做出更多變化。此處讀者可準備一個小型的伺服馬達(Servo Motor)，此處採用 Tower Pro SG90，這是一個常見的小型伺服馬達，因為常見於遙控模型，因此也稱為 RC Servo；通常較大型的 RC Servo 需要較高電壓與高電流才能運作，這裡選用 SG90，直接使用 EduCake 上的 5V 供電即可運作，不用額外電源供應。我們直接採用前面幾個程式的接線，並加上這個小型伺服馬達，完整接線如下：



注意 SG90 本體延伸出的的褐色線為 GND，紅色線為 5V，橙色線為訊號線。由於 RC Servo 的訊號線通常是接收 PWM 訊號，此範例將橙色線接至 EduCake 的 13 號腳。接著輸入程式如下：

```
#include <Servo.h>

const int Vr_analogInPin = A0;// 宣告 可變電阻 類比輸入腳位編號
```

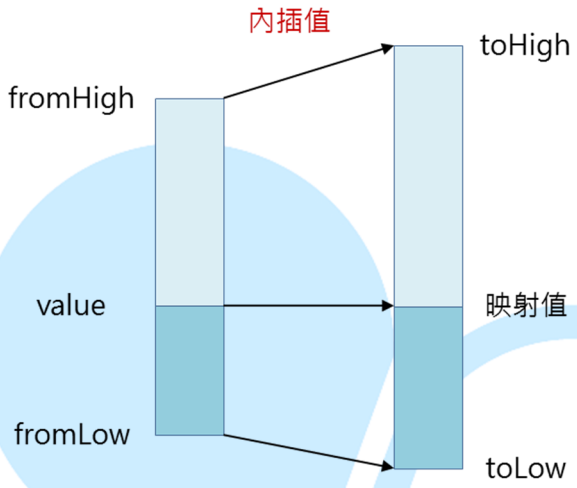
```
const int Ls_analogInPin = A1;// 宣告 光敏電阻 類比輸入腳位編號
const int LED_analogOutPin = 11;// 宣告類比輸出腳位編號
const int Servo_Pin = 13;// 宣告 Servo 輸出腳位編號

int Light_threshold = 0;

Servo Servo_1;

void setup() {
  Serial.begin(9600);
  Servo_1.attach(Servo_Pin);//, 500, 2400
  Servo_1.write(90);// Servo 置中
  Light_threshold = analogRead(Ls_analogInPin);// 讀取 光敏電阻 ADC 值，使用第一次讀取的初始值成為下限
}

void loop() {
  int Vr_rsensorValue = analogRead(Vr_analogInPin);// 讀取 可變電阻 ADC 值
  int Ls_rsensorValue = analogRead(Ls_analogInPin);// 讀取 光敏電阻 ADC 值

  Ls_rsensorValue = constrain(Ls_rsensorValue, Light_threshold, 1023);// 限制讀值範圍
  byte LED_light = map(Ls_rsensorValue, 512, 1023, 0, 255);// 重新映射 光敏電阻 ADC 值 範圍，成為 LED 亮度
  analogWrite(LED_analogOutPin, LED_light);// 輸出 PWM 波形至 LED

  int Servo_1_angle = map(Vr_rsensorValue, 0, 1023, 0, 180);// 重新映射 可變電阻 ADC 值 範圍，成為 Servo 角度
  Servo_1.write(Servo_1_angle);// 輸出 PWM 波形至 Servo
  /*
```

```
    Serial.print("Vr Sensor value = " );
    Serial.print(Vr_rsensorValue, DEC);
    Serial.print(", Servo angle = " );
    Serial.print(Servo_1_angle, DEC);
    Serial.print(". Light Sensor value = " );
    Serial.print(Ls_rsensorValue, DEC);
    Serial.print(", LED light = " );
    Serial.println(LED_light, DEC);
*/

    delay(20);
}
```

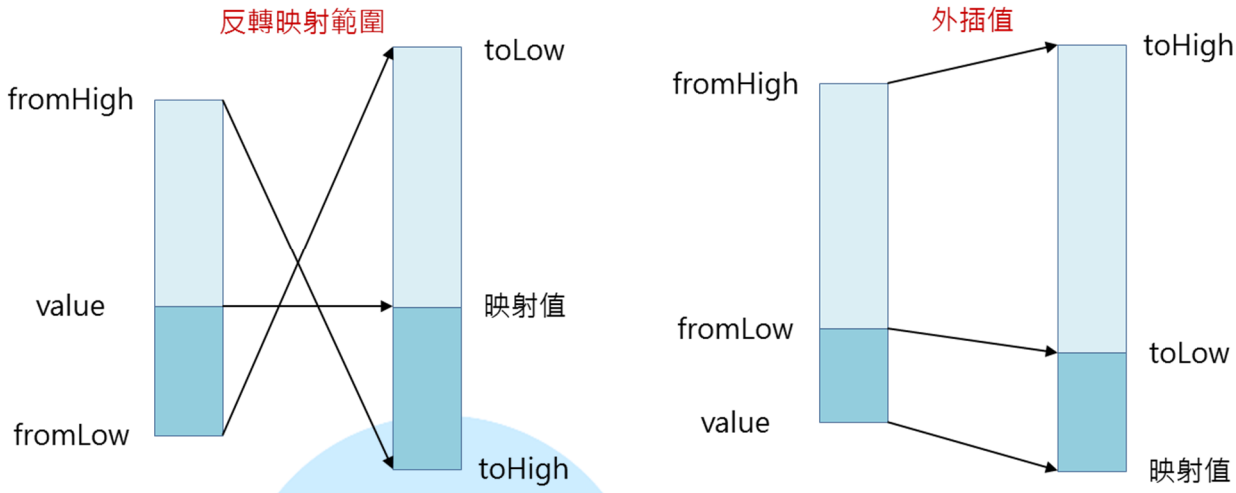
雖然 Servo 也是接收 PWM 訊號，但是頻率跟 duty cycle 都與 analogWrite( )不同，因此程式一開始便寫了#include <Servo.h>，使用 Servo 物件也有更多的函式方便使用。本範例功能為利用可變電阻偵測環境亮度，並調整 LED 的亮度，另外也利用轉動可變電阻，同步控制 RC Servo 的轉角。

需注意的地方是，由於 Servo.h 是物件導向語法，因此使用前需要宣告一個 Servo 型別的物件 Servo_1，接著在 setup( )裡使用 Servo_1.attach(Servo_Pin)，指定 Servo_1 物件接在 Servo_Pin 這個腳位，接很多顆時就得一顆一顆寫 Servo.attach( )語法了；至於 Servo_1.Write( )則可以指定 Servo_1 轉到某個特定的角度，一般置中為 90 度，兩邊界限分別為 0 跟 180 度。關於 Servo 的更多詳細用法介紹之後也會有專文介紹，此處先使用 Servo.Write( )函式進行轉動角度控制即可。

loop( )函式裡先讀取可變電阻及光敏電阻的 ADC 數值，接著使用 map( )函式進行數值的映射。map(value, fromHigh, fromLow, toHigh, toLow)會回傳依比例映射後的數值，概念如下圖所示：



value 為原始值，fromHigh、fromLow 為原始值上、下限，toHigh、toLow 為映射範圍上、下限；例如 ADC 讀值為 127，原始上下限 0~1023，需映射至 0~255，則 y = map(127, 0, 1023, 0, 255)，結果為 y=31。實際上 map( )函式也不限定正數內插值，或上限值一定要比較大，使用者也可以外插、使用負數上下限、反轉映射範圍大小順序等等，如下面二圖所示。這裡需要特別注意 map( )函式使用整數值運算，因此若 value 傳入小數值，將會先被轉換成整數再進行計算，使用時須注意這一點，否則會發現輸出值並不如預期；讀者可思考一下呼吸燈的案例，該如何以 map( )函式進行改寫呢?



loop( )中使用兩次的 map( )函式，將 ADC 讀值映射至 LED 亮度 0~255，以及 Servo 角度 0~180 之間，這是之後很多狀況都可以用到的功能。此程式較特別的一點，在於定義一變數「Light_threshold」，用在儲存電路剛開機時光敏電阻的數值；由於房間內可能已有些微燈光，為了讓開機後 LED 先維持不亮，環境全暗時 LED 最亮，因此使用此變數作為光敏電阻原始 ADC 值的下限。constrain(Ls_rsensorValue, Light_threshold, 1023)這行則是用在將之後的光敏電阻讀值都限制在下限與 1023 之間，避免數值被外插至 0~255 範圍以外的狀況。這也是一個使用感測器的小技巧，畢竟電路不是每次都在一樣的環境條件下運作，這個做法可以簡易達到開機校準的效果。

Upload 這個程式之後，讀者便可以看到在當時燈光不變的情況下，LED 是接近熄滅的；拿物品遮住光敏電阻過程中，LED 亮度開始上升，全遮住時 LED 亮度達到最大，讀者也可以嘗試用 Serial Monitor 看看實際的 ADC 數值變化狀況；若接上較大功率的燈泡(也需額外電源)，就可以當作輔助照明之類的功能使用囉。轉動可變電阻的圓棒，則可以發現 Servo 角度跟著旋

轉；讀者可以嘗試把 delay( )加長，會發現 Servo 對可變電阻旋轉的反應變得斷斷續續的，這是因為控制指令與讀取 ADC 頻率下降的原因。

此外，前面段落曾提到 EduCake 的類比輸入預設為 10 位元，其實也可以在 analogRead( )語法之前加上：

analogReadResolution(bits)，可以改變類比輸入的解析度，EduCake 的硬體最高可以支援 bits = 11 位元，若輸入更高數值，則多於位元會被補 0，bits 小於 11 則讀值解析度將被「裁切」掉。例如原始讀值為 b'111,1111,0000，則 analogReadResolution(15)會讀到 b'000,0111,1111,0000；analogReadResolution(9) 會讀到 b'1,1111,0000。

EduCake 的 PWM 類比輸出也有特殊之處，預設的 analogWrite(value)，為 8bit 解析度，也就是說 0~100%的 duty cycle 可以被切成 256 等分，所以前面程式練習使用 value = 0~255。實際上 EduCake 可支援到 13 位元的 PWM 解析度，讀者可使用 analogWriteResolution(bits)進行更改，例如 bits = 13 時，analogWrite(value)的 value 可使用 0~8191，依此類推；讀者使用這些功能的時候都需注意函式輸入、輸出數值換算的問題。

用 86Duino EduCake 作環境互動與控制，就是這麼簡單，往後我們再來嘗試整合更多複雜的應用。