EduCake 的中斷事件與脈波偵測使用

一、 中斷機制與脈波偵測介紹

前面幾個章節介紹過幾種 86Duino EduCake 與外界互動的方法，例如數位輸入/輸出、類比輸入/輸出等，再搭配各種感測器、馬達等做各種應用。但讀者可能也已經發現到，程式流程中使用 digitalRead/Write( )、analogRead/Write( )時多是搭配 delay( )之類函式作定時的功能觸發，例如定時改變 LED 亮度、定時讀取可變電阻電壓數值等等。這種定時觸發的方法（也稱為 polling）對於變動緩慢的外界狀況來說還算適用，但如果有種外界環境變化發生的速度很快，快到觸發偵測當下剛好錯過呢？ 讀者可能又會想到，把 delay( )數值設定小一點不就好了？ 這的確也是一種方法，但是利用縮短 delay( )時間增加 loop( )執行次數，也等於增加了處理器的運算量，且 delay( )函式執行時處理器便無法做其他事情了。

如果有一種方式是:「當外界環境發生變化時再去觸發對應的處理動作，其他時間處理器則做其他的事情」，是不是更有效率，也更能對外界快速變動做出反應呢？ 中央處理器的發展歷史中早已經想出了這樣的解決方法，稱為中斷（interrupt），處理器會有特定腳位可以偵測外界的變化，一旦變化發生，則馬上觸發對應的處理程式（interrupt service routine, ISR）做出特定的回應。這樣的機制在你我的個人電腦處理器、許多嵌入式裝置的微處理器、以及 86Duino EduCake 的處理器上都已經存在，連 Serial Port 通訊都需要這個機制的輔助才能正常運作呢。

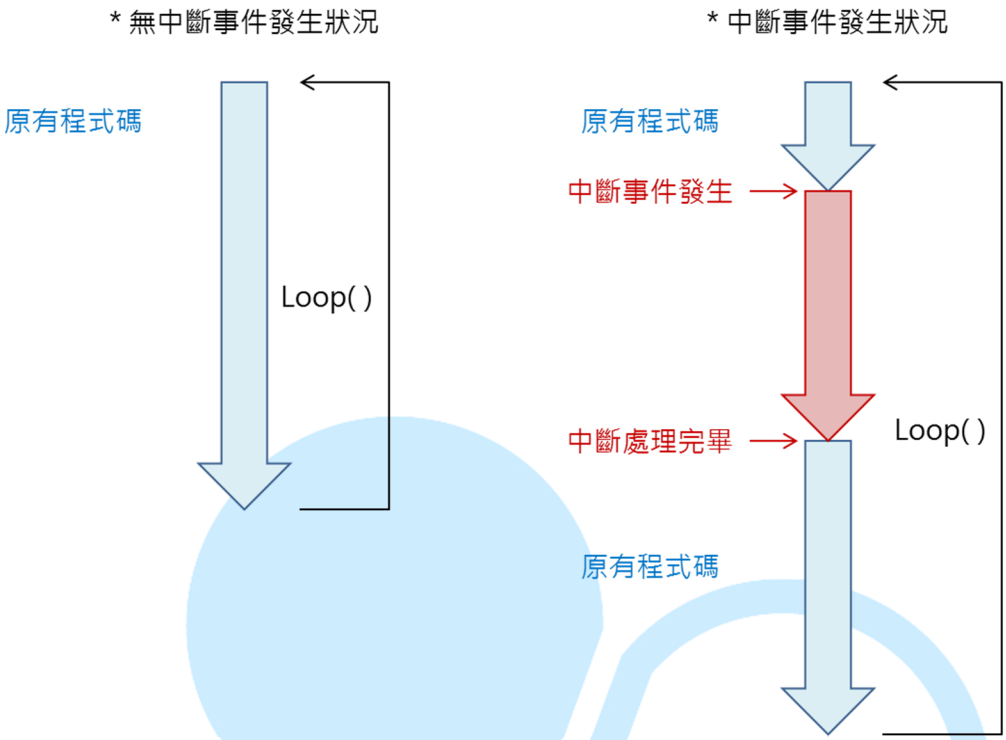


圖 1. 處理器中斷處理運作概念圖

中斷機制的示意圖如圖一所示，當沒有中斷觸發時，處理器一次又一次專心處理手邊工作，當中斷發生時，處理器會將剛剛在執行的程式狀態暫存起來，跳往中斷處理程式（interrupt handler）進行對應動作，等中斷處理結束再回頭從剛剛暫存的程式狀態繼續往下執行，因此處理器處理中斷事件時，其實會稍微影響原本程式碼的時序，所以中斷事件處理程式中不宜作太耗時的操作。

中斷機制又分為內部中斷（internal interrupt）與外部中斷（external interrupt）兩種。「內部中斷」是指處理器內部的中斷機制，像是內建的定時器（Timer）定時觸發，用在需要定時執行的功能，例如：每分鐘偵測一次環境溫度、每 30 秒改變一次伺服馬達角度等等。注意這跟使用 delay( ) 進行定時的方式不一樣，使用「內部中斷方式定時」在沒有中斷處理的間隔

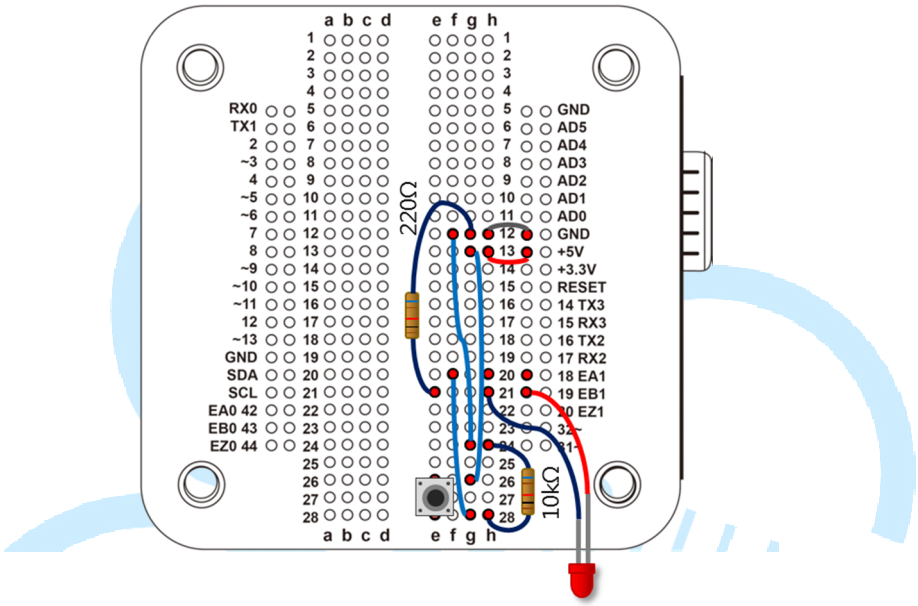
時期之內，處理器依然可以進行其他工作；使用 delay( )時處理器一樣會對中斷有反應，但其他程式碼就不會執行了；另外這兩種定時方法的時間精確度也不同，以定時中斷方式觸發的精準度較佳。「外部中斷」顧名思義則是用在偵測「處理器外」的變化，例如某支針腳電壓從高變低、從低變高之類的狀況；當搭配合適的裝置時，便能夠與處理器的中斷功能搭配運作，本章節後面的程式實作便會使用這種「外部中斷」方式做練習。

除了利用中斷機制偵測外界環境變化之外，一定要提到 86Duino EduCake 還有另一個偵測外界變化的功能，程式內語法為 pulseIn( )，此函式可用來偵測某腳位的電壓變化，特別是一個腳位處於 HIGH 或 LOW 狀態的時間長度。後續範例也會使用 pulesIn( )做實際練習，並與中斷比較實際效果。

下面便讓我們利用程式實際練習上面所講解的兩種機制，並利用這些機制做實用的功能來練習吧。

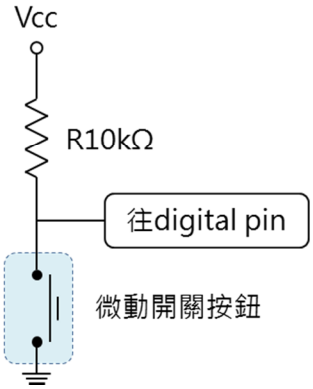
二、 第一個程式 – 練習 attachInterrupt( )、deattachInterrupt( )

第一個範例程式先來練習如何使用 86Duino EduCake 的中斷功能，讀者請先依下圖接線：

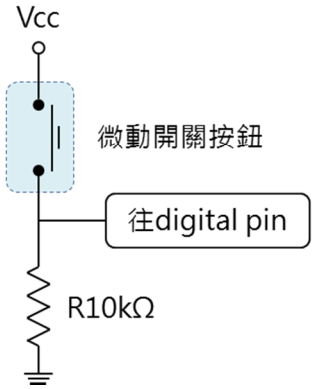


按鈕部分有兩種接法，原理如下圖，這裡由於需偵測 HIGH 的變化，所以選擇「Normal LOW」的接線，讀者可視實際需要選擇接線方式。

按鈕一般未按下時，控制板偵測到HIGH
(Normal HIGH)



按鈕一般未按下時，控制板偵測到LOW
(Normal LOW)

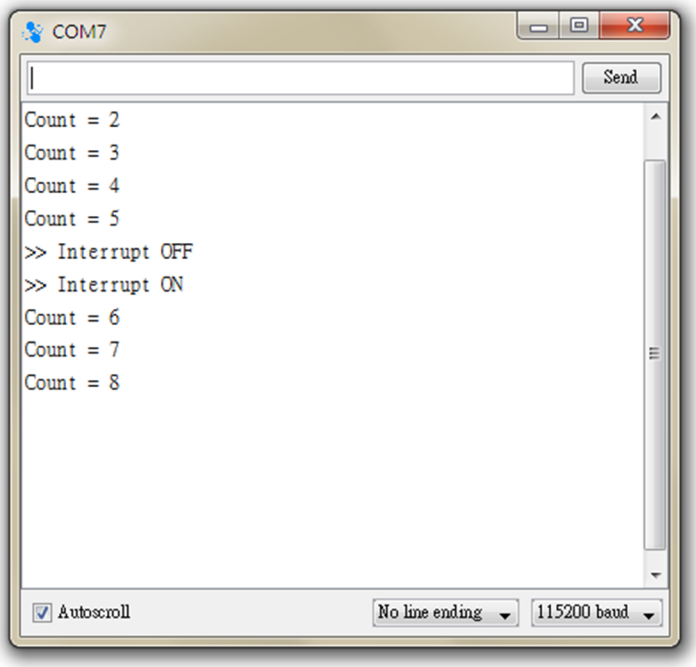


接著請打開 86Duino Coding IDE，輸入以下程式碼：

```
nt BTN_pin = 3;// Normal LOW, pin 18 = interrupt 3
int LED_pin = 19;
volatile int state = LOW;
int count = 0;

void setup()
{
  Serial.begin(115200);// 設定 Serial port
  pinMode(LED_pin, OUTPUT);// 設定 針腳模式
  digitalWrite(LED_pin, LOW);// 先讓 LED 熄滅
  attachInterrupt(BTN_pin, InterruptHandler, RISING);// 設定
針腳的 Interrupt
}

void loop()
{
  if(Serial.available()){// 檢查 Serial Port 是否有資料
    char data = Serial.read();
    if(data == 'A'){// 如果收到字元 A 則設定中斷
      attachInterrupt(BTN_pin, InterruptHandler, RISING);//
設定 針腳的 Interrupt
      Serial.println(">> Interrupt ON");
    }
    else if(data == 'B'){// 如果收到字元 B 則取消中斷
      detachInterrupt(BTN_pin);// 取消 針腳的 Interrupt
      Serial.println(">> Interrupt OFF");
    }
  }

  digitalWrite(LED_pin, state);// 點亮 LED
}

void InterruptHandler()
```

```
I{
state = !state;// 切換 LED 狀態
  count++;// 計數器+1
  Serial.print("Count = ");
  Serial.println(count);
}
```

此範例程式功能為，當使用者每按一次按鈕時，便會觸發中斷處理程式，變換 LED 的亮滅狀態，並將計數器加一，經由 Serial Monitor 傳出計數訊息；當使用者送了字元'B'給板子，則取消中斷設定，送字元'A'則重新設定好中斷處理。執行結果如下圖：



程式一開始宣告幾個會使用到的變數，定義針腳編號、紀錄 LED 狀態、計數器數值等，然後 setup( )裡使用 attachInterrupt( )函式設定中斷編號，以及對應的中斷處理程式，中斷偵測模式等參數。讀者可能會有疑問，為何按鈕的偵測是接在數位腳 18，但程式一開始卻寫 BTN_pin = 3 呢？這是因

為 86Duino 擁有好幾組可設定中斷偵測的腳位，如下表所示：（或參考 http://www.86duino.com/?p=1756）

中斷編號	int.0	int.1	int.2	int.3	int.4	int.5
EduCake	42	43	44	18	19	20

attachInterrupt(pin, ISR, MODE)、detachInterrupt(pin)兩個函式的 pin 參數欄須設定為此腳位對應的中斷編號，而不是實際的腳位編號。中斷偵測模式可設定為：CHANGE（電位由高變低或由低變高）、RISING（電位由低變高）、FALLING（電位由高變低）三種，使用者可視實際電路設計與程式功能需求選擇；此範例由於使用「Normal LOW」的按鈕，且須偵測「按鍵每次按下」的動作，所以使用 RISING 的模式。

中斷處理程式則在每次觸發時將 LED 狀態反向，將 count 變數加一，並將 count 數值藉由 Serial.print( )印出到 Serial Monitor 觀察。

loop( )迴圈內負責檢查 Serial port 的接收狀態，當收到字元'A'時，執行 attachInterrupt(pin, ISR, MODE)，若收到字元'B'則執行 detachInterrupt(pin)，以取消某腳位的中斷設定。另外 loop( )也使用 digitalWrite( )控制 LED 的亮度。

86Duino 除了執行使用者輸入的程式碼外，其實背景還會處理其他的中斷事件，例如收 Serial Port 的資料，上面範例程式使用的 attachInterrupt( )是在這些預設會偵測的中斷事件外，額外針對某些 I/O 腳位加上的設定，而 detachInterrupt( )取消的也是對應的中斷設定，那有沒

有辦法取消控制板所有的中斷處理呢？86Duino EduCake 還有提供另兩個中斷相關的函式：interrupts( )及 noInterrupts( )。interrupts( )用來允許背景執行的中斷處理（程式預設值便是允許的），而 noInterrupts( )則會讓他們暫時停擺，直到再次呼叫 interrupts( )為止，讀者也可以在程式裡使用這兩個函式試試效果喔。

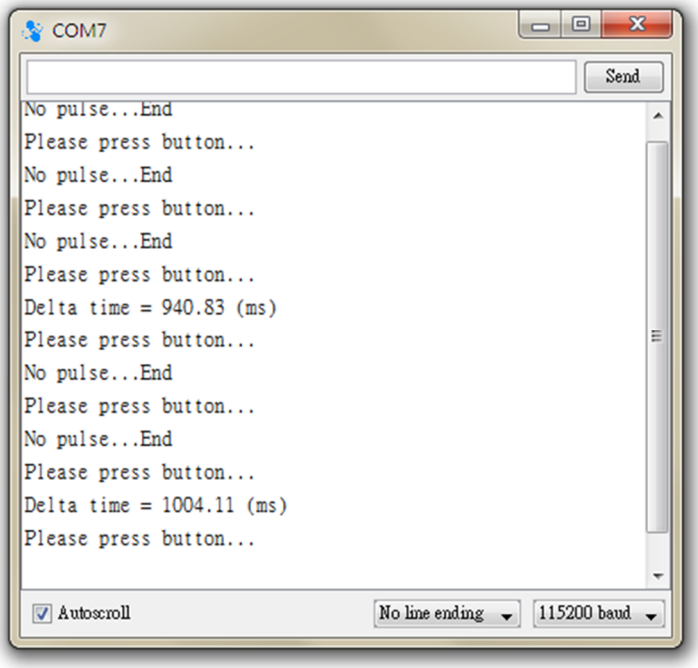
另外讀者在執行上述範例程式時，可能會發現按一下按鈕時，怎麼中斷處理函式可能會執行好多次呢？這種一般稱為「彈跳（bounce）」的現象偶爾會發生，因為微動開關按鈕本身是機械裝置，接點在穩定接觸前可能在短時間內會有斷續的接觸現象，造成控制板的多次中斷觸發；「去彈跳（Debounce）」是一個微控制器應用常碰到的課題，有硬體也有軟體的解決方法，讀者若有興趣的話也可以查詢這方面的資料進行閱讀。

三、 第三個程式 – 練習 pulseIn( )

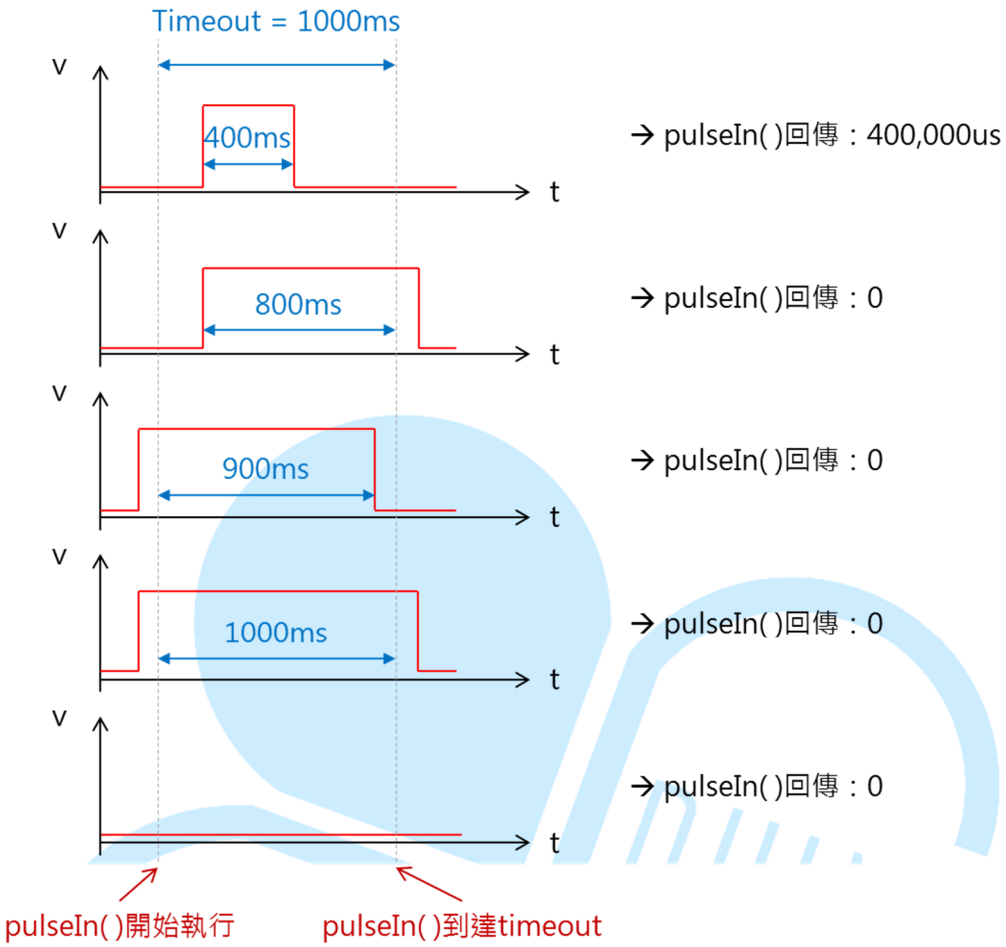
這個範例程式跟上一個是類似的功能，但改使用 pulseIn( )函式來實作，讀者一樣不用更改接線，請在 86Duino Coding IDE，輸入以下程式碼：

```
// pulseIn Timer
int btn_pin = 18;// 按鈕針腳編號 // Normal LOW, pin 18 = interrupt 3
unsigned long max_duration = 2000000;// pulseIn timeout 時間, 單位：us

void setup() {
  Serial.begin(115200);// 設定 Serial port
  pinMode(btn_pin, INPUT);// 設定 針腳模式
}

void loop() {
  Serial.println("Please press button...");// 印出開始測量訊息

  unsigned long duration = pulseIn(btn_pin, HIGH, max_duration);

  if(duration>0){// 如果有量到數值
    Serial.print("Delta time = ");
    Serial.print(((float)duration)/1000);// 印出間隔時間，單位：ms
    Serial.println(" (ms)");
  }
  else{
    Serial.println("No pulse...End");
  }

  delay(2000);
}
```

此範例程式跟上個一樣是偵測使用者按按鈕的時間長度，使用前須開啟 Serial Monitor，當視窗顯示「Please press button...」訊息時，可選擇按或不按下按鈕，程式會在 2 秒的時間內偵測是否有「按下並放開按鈕」的動作，如果有按下按鈕則顯示壓按的時間長度，否則等到 2 秒的偵測時間結束後送出「No pulse...End」訊息；流程延遲 2 秒後便重新來一次；執行結果如下圖：



雖然與上個範例都是偵測壓按時間長度，但由於 pulseIn( )函式並不是利用中斷機制偵測，只有當程式執行到此處才會觸發偵測動作，因此程式碼須集中在 loop( )內依序執行。pulseIn( )擁有兩種形式 : pulseIn(pin, value) 與 pulseIn(pin, value, timeout)兩種，函式運作機制如下圖所示：



pulseIn( )函式，測量波型 HIGH 運作概念圖

以 value 設定為 HIGH 為例。此函式執行後，便會在時間限制(timeout)內偵測指定腳位的電壓波型變化狀況（value），只有當偵測期間內有完整波型（波型先上升後下降，或是先下降或上升）才會回傳測量數值，其餘狀況皆回傳 0。value 可設定為 HIGH 或 LOW，當偵測到 pin 為 HIGH 或 LOW 則開始計時，直到波型變為相反則停止計時，回傳中間經過的時間。若時間限制內波型沒有變成相反或是波形根本沒有改變，則回傳 0。

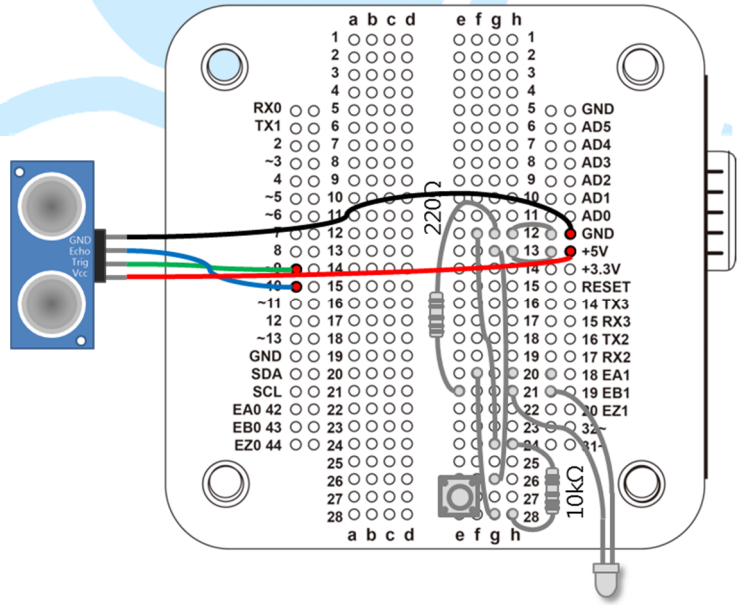
程式一開始先宣告 unsigned long max_duration = 2000000(單位為 µs) 用來當作 timeout 的時間長度，表示之後 pulseIn( )函式最多會等待 2

秒的時間才結束繼續往下執行，若使用 pulseIn(pin, value)，則預設 timeout 時間為 1 秒。setup( )裡須使用 pinMode(btn_pin, INPUT)將欲偵測的腳位設定為 INPUT；之後 loop( )裡先以 Serial.print( )提示使用者按下按鈕，然後使用 unsigned long duration = pulseIn(btn_pin, HIGH, max_duration)做按鈕動作變化的偵測，由於按鈕前述已提過採用 Normal LOW 的設定，因此 pulseIn( )養當然是設定為 HIGH 的偵測模式，若是按鈕採用 Normal HIGH 的接法，那就得把 pulseIn( )設定為偵測 LOW 囉。另外 pulseIn( )回傳的數值單位是μs，所以使用上要小心，此處為了跟前一範例程式統一單位，因此印出前先除以 1000 變為 ms 單位。讀者也可以把 timeout 參數設定為 5 秒之類試試，當出現提示按鈕訊息時沒有按下按鈕，那程式真的會等待 5 秒才繼續往下執行喔。

藉由此範例，讀者應可看出 pulseIn( )與中斷偵測的不同之處，一種可馬上對外界變化做出反應，一種則需執行到函式才有偵測的作用，專題使用時當然也得好好做出最佳選擇囉。讀者可以思考看看，此範例如何修改為測量反應速度用的功能？ 當 LED 亮起，使用者須馬上按下按鈕，然後程式顯示從 LED 亮到按下按鈕之間的間隔時間，這該如何修改呢？

四、 第四個程式

pulseIn( )函式雖然反應較慢，但還是有它的用武之處，接著便使用這個函式來做點有趣的應用。此範例以一個相當實用的超音波感測器來做練習，這裡我們選用 HC-SR04 這顆容易購買到的型號。市面上當然也有其他型號可選擇，通常是測量距離之類的規格不太一樣，但基本使用方法都差不多，購買時注意使用說明即可。通常超音波感測器會有 3~4 條接線腳位，其中 2 條為 5V 跟 GND，其他則是訊號腳；讀者請參考下圖接好線路，舊有的線路可以不用拆除：



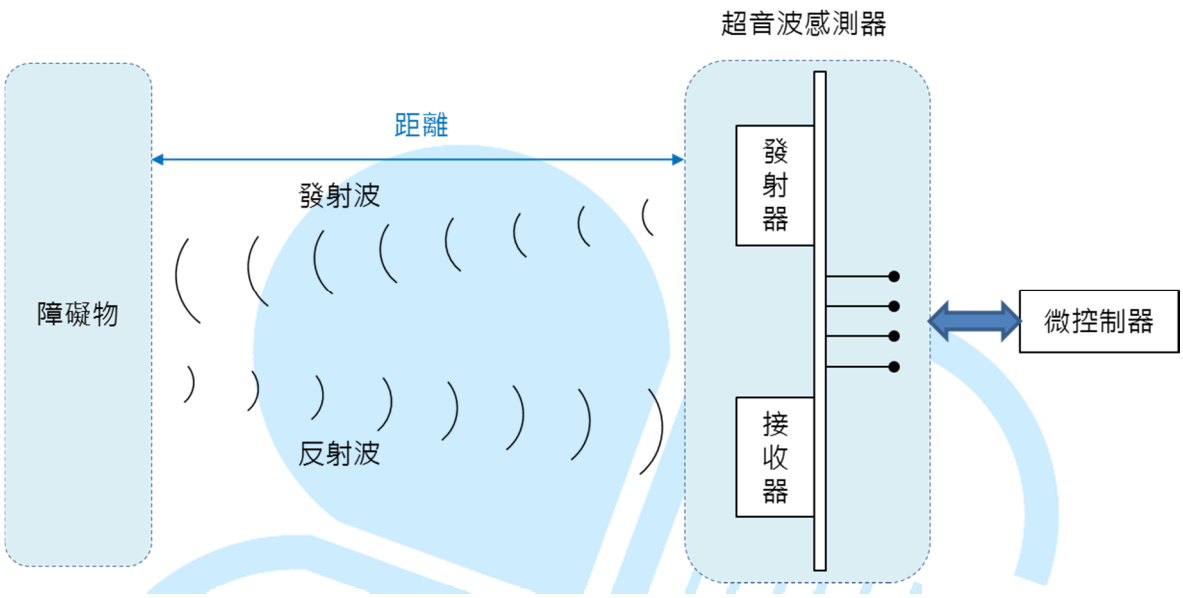
接著打開 86Duino Coding IDE，輸入以下程式：

```
#define trigPin_1 9 // Trig  腳位編號
#define echoPin_1 10 // Echo  腳位編號
#define intervaltime 100 //  設定測量間隔時間，單位：毫秒
#define LED_Pin 19//  宣告類比輸出腳位編號
boolean LED_ON = false;// LED  狀態
unsigned int LED_ON_count = 0;// LED 點亮長度變數
```

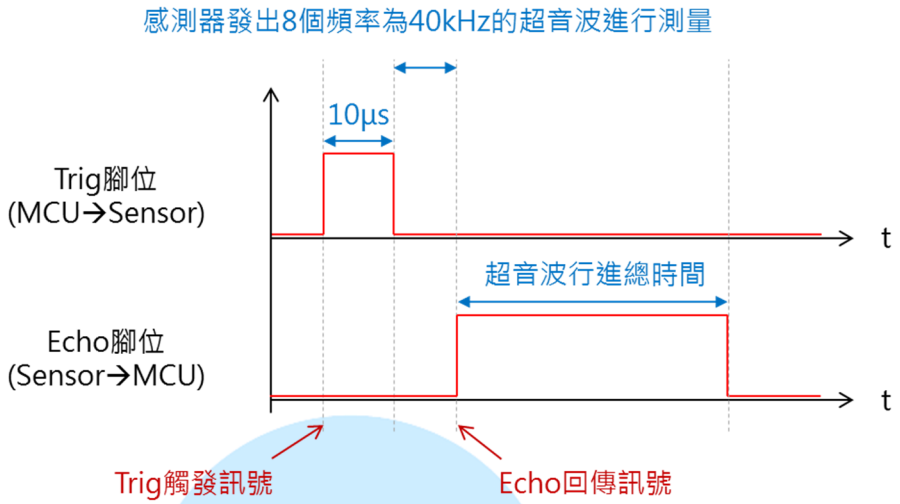
```
unsigned int LED_ON_count_max = 1;// LED 點亮長度變數 最大值
unsigned int LED_OFF_count = 0;// LED 熄滅長度變數
int timeout = 12000; // 設定 pulseIn 的持續時間
float Sound_speed = 343.0f * 100 / 1000000; // 音速：340.29, 343.2 m/s 換算為 cm/us

void setup()
{
  Serial.begin(115200);
  pinMode(trigPin_1, OUTPUT);// 設定腳位模式
  pinMode(echoPin_1, INPUT);// 設定腳位模式
  pinMode(LED_Pin, OUTPUT);// 設定腳位模式
  digitalWrite(trigPin_1, LOW);// 先讓觸發腳位為 LOW
  delay(1);
}

void loop() {
  float distance = Get_US();// 讀取超音波感測器，並將 long 數值轉換成 float
  if(distance>0){
    Serial.print(", Dis= ");
    Serial.print(distance);

    if(LED_ON) {// LED 亮
      LED_ON_count++;
      if(LED_ON_count >= LED_ON_count_max) {
        LED_ON_count = 0;
        LED_ON = false;
      }
      digitalWrite(LED_Pin,HIGH);
      Serial.print(", LED ON");
    }
    else {// LED 暗
```

```
        LED_OFF_count++;
            if(LED_OFF_count >= int(distance/10)) {
              LED_OFF_count = 0;
              LED_ON = true;
            }
            digitalWrite(LED_Pin,LOW);
          }
          Serial.println();
        }
        else {
          Serial.println("Out of range !");
        }
        delay(intervaltime);// 加上延遲時間
      }

      float Get_US() {// 處理超音波感測器的函式
        // Trigger
        digitalWrite(trigPin_1, LOW);
        delayMicroseconds(2);
        digitalWrite(trigPin_1, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin_1, LOW);

        // Read
        long duration = pulseIn(echoPin_1, HIGH, timeout);//
timeout in us
        Serial.print("Dur= ");
        Serial.print(duration);

        // 換算數值
        float distance = (float)(duration) / 2 * Sound_speed;
        return distance;
      }
```

此程式功能為類似一般汽車倒車偵測器，當障礙物距離感測器越近，則 LED 閃爍頻率越快，同時 Serial Port 也會印出實際換算距離。解釋程式做法前，我們先來看看超音波感測器的測量原理，示意圖如下：



測量流程為，當控制器送出觸發訊號給感測器，感測器發射源便發出超音波，開始計算接收器收到反射波的時間，接著將測量到的數值傳回控制器。以 HC-SR04 而言，訊號腳為 Trig 與 Echo 兩根，Trig 是控制器送給感測器表示開始進行測量的訊號；Echo 則是感測器測量完畢，回傳到控制器的訊號（也有其他型號的超音波感測器是把這兩隻針腳做成同一腳位，但原理基本相同）。使用方法為，平常 Trig 與 Echo 腳位都是 LOW 的狀態，當控制器想觸發測量時，便由 Trig 腳位送出一個約 10μs 的 HIGH 訊號，接著變成 LOW，感測器便會開始測量。若感測器有收到反射波，那麼 Echo 腳位便會送出一個長度為「從發射超音波到接收到回波經過時間」的 HIGH 訊號，單位為μs，這便是 pulseIn()可以派上用途的地方了，超音波感測器控制訊號示意圖如下：



程式一開始宣告幾個運作相關設定參數，如測量間隔時間、pulseIn( )的 timeout 時間、LED 閃爍參數、音速換算參數等等；接著 setup( )裡設定超音波感測器訊號腳位的 I/O 模式，並初始化控制腳位的狀態。讀者可以看到 loop( )裡面沒有多少程式碼，因為一般應用感測器時，可能安裝不只一顆感測器，同樣的程式碼便要重複使用很多次，所以利用函式（Function）的方式便可以將程式碼適度的「模組化」，需要用時重複呼叫即可，省去重複撰寫的時間，程式維護方便性與可讀性也會增加。此範例程式處理超音波感測器的程式碼被移至 Get_US( )這個函式，當呼叫 Get_US( )時，函式便處理超音波感測器所有相關流程，再將結果回傳給 loop( )主要流程，做 Serial Port 顯示數值與 LED 亮暗的調控。

Get_US( )函式裡程序一開始，為避免其他地方不小心改到 Trig 腳位狀態，以便產生一個「乾淨」的訊號，因此先使用 digitalWrite(trigPin_1, LOW)設定腳位狀態，delayMicroseconds(2)是為了讓腳位狀態穩定，接著再用 digitalWrite(trigPin_1, HIGH)送出觸發波型，持續 10μs 再變為 LOW，讓超音波感測器開始工作。觸發感測器後，接著就是使用 pulseIn(echoPin_1,

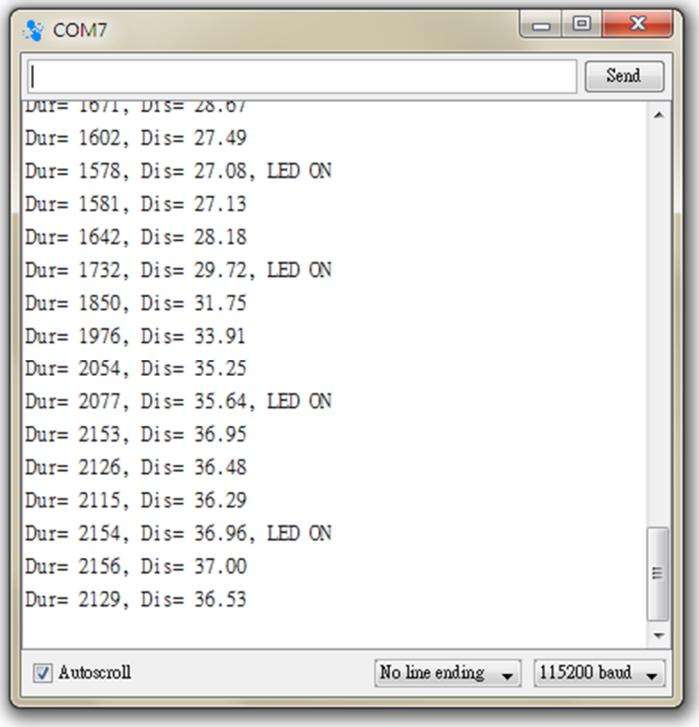
HIGH, timeout)函式，等著接收感測器 Echo 腳的回傳訊號了。pulseIn( )這個函式的回傳數值形式是 long，因此宣告一個 long 變數 duration 儲存感測器回傳值，這就是超音波在感測器與障礙物之間行進的總時間長度，因此接著換算成實際距離時須將此數除以 2 才是單趟時間。

計算距離的算式使用「距離 = 時間 * 速度」，而音速一般經驗公式是：音速 = 331 + 0.6 * 攝氏溫度，此範例是假設溫度攝氏 20 度，用固定數值 343（m/s），讀者也可以使用隨溫度變化的公式做計算囉。另外因為音速單位是公尺/秒（m/s），而感測器回傳值是微秒（μs）單位，想要換算成公分(cm)顯示，所以要先做換算：343(m/s) = 343 * 100 / 1000000（cm/μs）。實際程式碼 float distance = float(duration) / 2 * Sound_speed 中，將 duration 先轉換為浮點數，以便在數學計算中保留較多小數點。最後 Get_US( )回傳計算過後的距離值，可做其他程式碼任何用途。例如自走車便可以用此數值判斷障礙物距離自己多遠等等。

此範例程式的 loop( )流程中，取得感測器數值後便將距離藉由 Serial.print( )印出在 Serial Monitor 上面觀察，另外用來調整 LED 亮暗的頻率。此處需要注意的是，LED 亮暗間隔雖然可以用 delay( )實作，但由於在 loop( )裡面使用變動的延遲時間也會一並讓超音波測量間隔變得忽快忽慢，所以此處改用計數器的作法來改變 LED 亮暗頻率。開頭宣告幾個全域變數 LED_ON_count、LED_OFF_count、LED_ON，LED_ON 為 LED 亮度狀態，當 LED 亮，每次 loop( )都會讓 LED_ON_count 變數+1，上限可自由設定，此範例使用 1 當作上限，當變數達到上限，便讓 LED 熄滅，LED_ON 變成 false，LED_ON_count 歸零，進入 LED 熄滅階段。LED_ON 為 false

時，則換成 LED_OFF_count 每次+1，上限為 int(distance/10)，由於 loop( ) 間隔設定為 100ms，以測量距離為 200cm 而言，可以讓 LED 有約 2 秒的熄滅時間，讀者也可以修改成其他數字看看效果。

程式執行後，就可以觀察 Serial Monitor 以及 LED 的亮暗，了解目前測量到的距離，程式執行結果如下圖：



由超音波感測器的使用原理，讀者也可以想想，如何用改用中斷或是 digitalRead( )的方式，去測量 Echo 腳的訊號長度？ 另一方面，如果把 LED 換成簡單的蜂鳴器與配套電路，是不是就變成簡單的障礙物偵測器或倒車警示器了呢?