EduCake 的中断事件与脉波侦测使用

一、 中断机制与脉波侦测介绍

前面几个章节介绍过几种 86Duino EduCake 与外界互动的方法，例如数字输入/输出、模拟输入/输出等，再搭配各种传感器、马达等做各种应用。但读者可能也已经发现到，程序流程中使用 digitalRead/Write( )、analogRead/Write( )时多是搭配 delay( )之类函式作定时的功能触发，例如定时改变 LED 亮度、定时读取可变电阻电压数值等等。这种定时触发的方法（也称为 polling）对于变动缓慢的外界状况来说还算适用，但如果有种外界环境变化发生的速度很快，快到触发侦测当下刚好错过呢？ 读者可能又会想到，把 delay( )数值设定小一点不就好了？ 这的确也是一种方法，但是利用缩短 delay( )时间增加 loop( )执行次数，也等于增加了处理器的运算量，且 delay( )函式执行时处理器便无法做其他事情了。

如果有一种方式是：「当外界环境发生变化时再去触发对应的处理动作，其他时间处理器则做其他的事情」，是不是更有效率，也更能对外界快速变动做出反应呢？ 中央处理器的发展历史中早已经想出了这样的解决方法，称为中断（interrupt），处理器会有特定脚位可以侦测外界的变化，一旦变化发生，则马上触发对应的处理程序（interrupt service routine, ISR）做出特定的回应。这样的机制在你我的个人计算机处理器、许多嵌入式装置的微处理器、以及 86Duino EduCake 的处理器上都已经存在，连 Serial Port 通讯都需要这个机制的辅助才能正常运作呢。

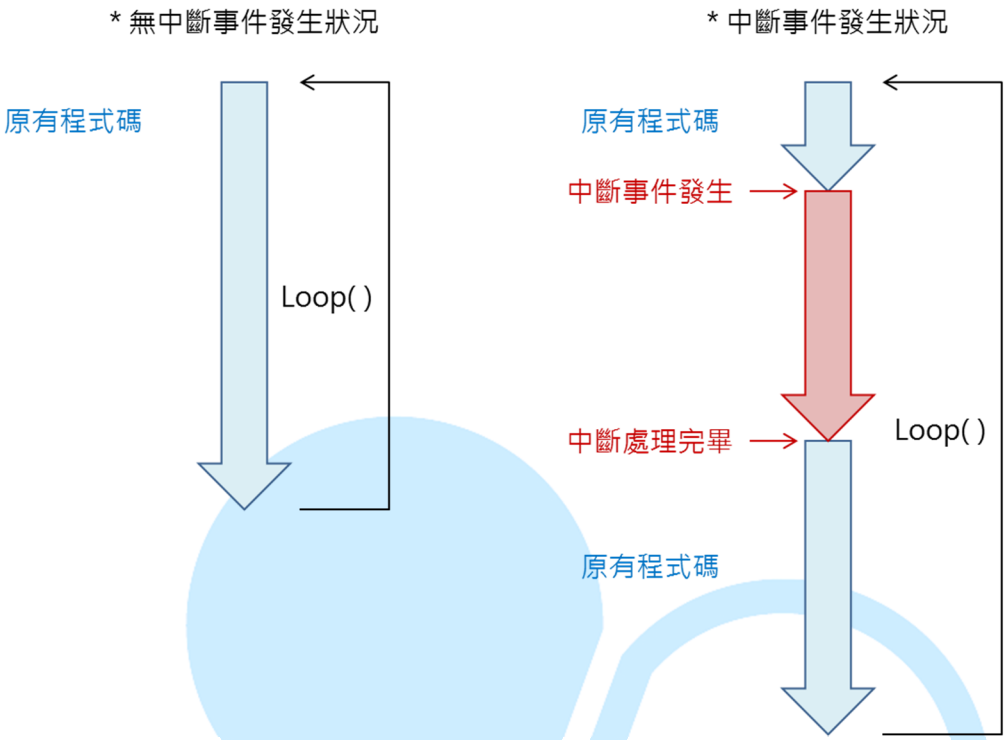


图 1. 处理器中断处理运作概念图

中断机制的示意图如图一所示，当没有中断触发时，处理器一次又一次专心处理手边工作，当中断发生时，处理器会将刚刚在执行的程序状态暂存起来，跳往中断处理程序（interrupt handler）进行对应动作，等中断处理结束再回头从刚刚暂存的程序状态继续往下执行，因此处理器处理中断事件时，其实会稍微影响原本程序代码的时序，所以中断事件处理程序中不宜作太耗时的操作。

中断机制又分为内部中断（internal interrupt）与外部中断（external interrupt）两种。「内部中断」是指处理器内部的中断机制，像是内建的定时器（Timer）定时触发，用在需要定时执行的功能，例如：每分钟侦测一次环境温度、每 30 秒改变一次伺服马达角度等等。注意这跟使用 delay( ) 进行定时的方式不一样，使用「内部中断方式定时」在没有中断处理的间隔时期之内，处理器依然可以进行其他工作；使用 delay( )时处理器一样会对中断有反应，但其他程序代码就不会执行了；另外这两种定时方法的时间精

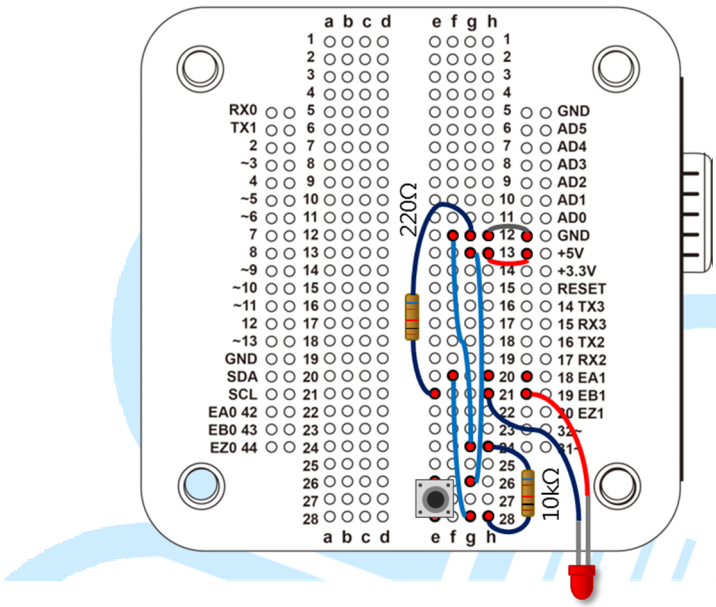
确度也不同，以定时中断方式触发的精准度较佳。「外部中断」顾名思义则是用在侦测「处理器外」的变化，例如某支针脚电压从高变低、从低变高之类的状况；当搭配合适的装置时，便能够与处理器的中断功能搭配运作，本章节后面的程序实作便会使用这种「外部中断」方式做练习。

除了利用中断机制侦测外界环境变化之外，一定要提到 86Duino EduCake 还有另一个侦测外界变化的功能，程序内语法为 pulseIn( )，此函式可用来侦测某脚位的电压变化，特别是一个脚位处于 HIGH 或 LOW 状态的时间长度。后续范例也会使用 pulesIn( )做实际练习，并与中断比较实际效果。

下面便让我们利用程序实际练习上面所讲解的两种机制，并利用这些机制做实用的功能来练习吧。

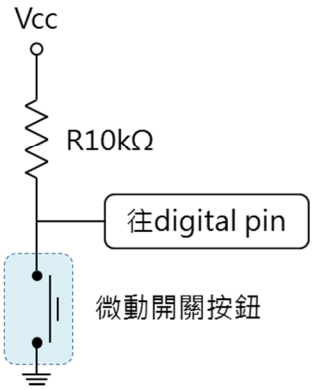
二、 第一个程序 – 练习 attachInterrupt()、deattachInterrupt()

第一个范例程序先来练习如何使用 86Duino EduCake 的中断功能，读者请先依下图接线：

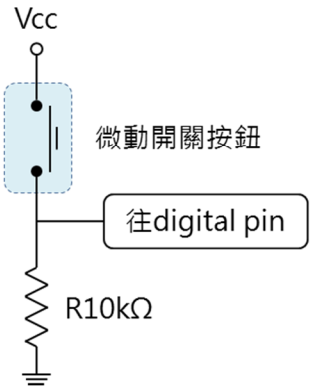


按钮部分有两种接法，原理如下图，这里由于需侦测 HIGH 的变化，所以选择「Normal LOW」的接线，读者可视实际需要选择接线方式。

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)

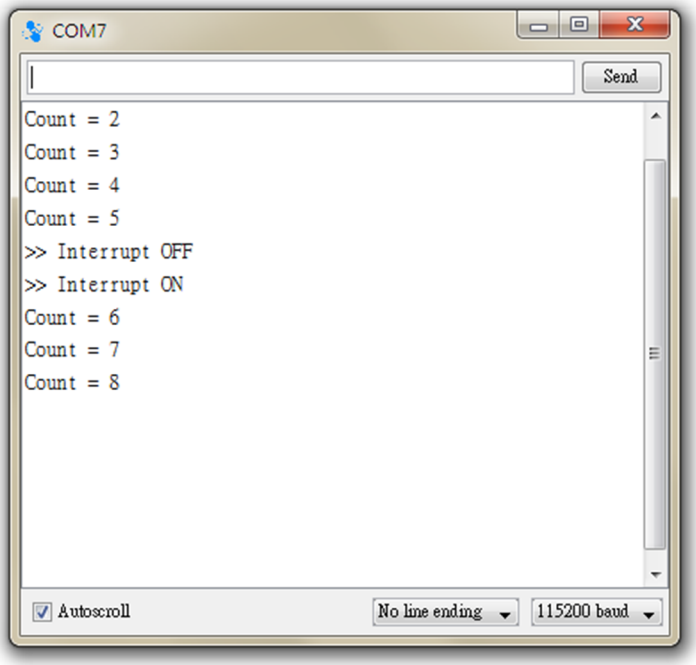


接着请打开 86Duino Coding IDE，输入以下程序代码：

```
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);
}
```

此范例程序功能为，当用户每按一次按钮时，便会触发中断处理程序，变换 LED 的亮灭状态，并将计数器加一，经由 Serial Monitor 传出计数讯息；当用户送了字符'B'给板子，则取消中断设定，送字符'A'则重新设定好中断处理。执行结果如下图：



程序一开始宣告几个会使用到的变量，定义针脚编号、纪录 LED 状态、计数器数值等，然后 setup( )里使用 attachInterrupt( )函式设定中断编号，以及对应的中断处理程序，中断侦测模式等参数。读者可能会有疑问，为何按钮的侦测是接在数字脚 18，但程序一开始却写 BTN_pin = 3 呢？这是因

为 86Duino 拥有好几组可设定中断侦测的脚位，如下表所示：（或参考 http://www.86duino.com/?p=1756）

中断编号	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)两个函式的 pin 参数栏须设定为此脚位对应的中断编号，而不是实际的脚位编号。中断侦测模式可设定为：CHANGE（电位由高变低或由低变高）、RISING（电位由低变高）、FALLING（电位由高变低）三种，用户可视实际电路设计与程序功能需求选择；此范例由于使用「Normal LOW」的按钮，且须侦测「按键每次按下」的动作，所以使用 RISING 的模式。

中断处理程序则在每次触发时将 LED 状态反向，将 count 变量加一，并将 count 数值藉由 Serial.print( )印出到 Serial Monitor 观察。

loop( )循环内负责检查 Serial port 的接收状态，当收到字符'A'时，执行 attachInterrupt(pin, ISR, MODE)，若收到字符'B'则执行 detachInterrupt(pin)，以取消某脚位的中断设定。另外 loop( )也使用 digitalWrite( )控制 LED 的亮度。

86Duino 除了执行使用者输入的程序代码外，其实背景还会处理其他的中断事件，例如收 Serial Port 的数据，上面范例程序使用的 attachInterrupt( )是在这些预设会侦测的中断事件外，额外针对某些 I/O 脚位加上的设定，而 detachInterrupt( )取消的也是对应的中断设定，那有没

有办法取消控制板所有的中断处理呢？86Duino EduCake 还有提供另两个中断相关的函式：interrupts( )及 noInterrupts( )。interrupts( )用来允许背景执行的中断处理（程序默认值便是允许的），而 noInterrupts( )则会让他们暂时停摆，直到再次呼叫 interrupts( )为止，读者也可以在程序里使用这两个函式试试效果喔。

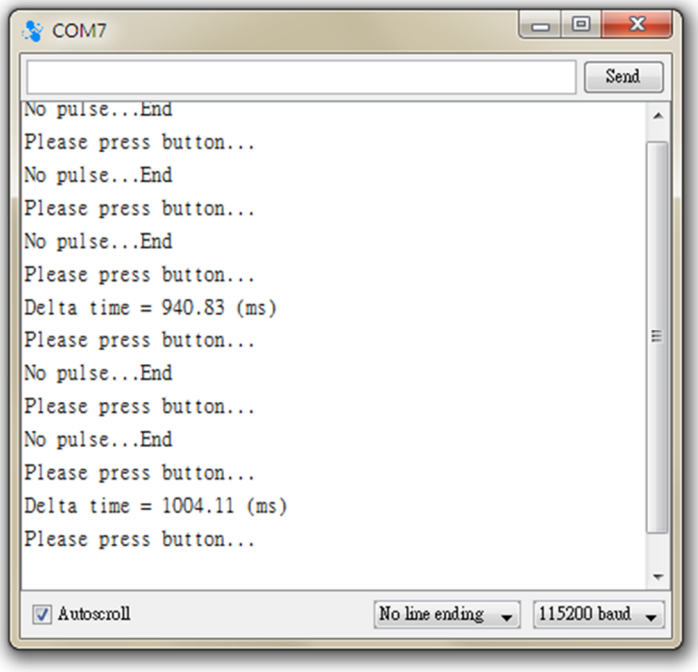
另外读者在执行上述范例程序时，可能会发现单击按钮时，怎么中断处理函式可能会执行好多次呢？这种一般称为「弹跳（bounce）」的现象偶尔会发生，因为微动开关按钮本身是机械装置，接点在稳定接触前可能在短时间内会有断续的接触现象，造成控制板的多次中断触发；「去弹跳（Debounce）」是一个微控制器应用常碰到的课题，有硬件也有软件的解决方法，读者若有兴趣的话也可以查询这方面的资料进行阅读。

三、 第三个程序 – 练习 pulseIn( )

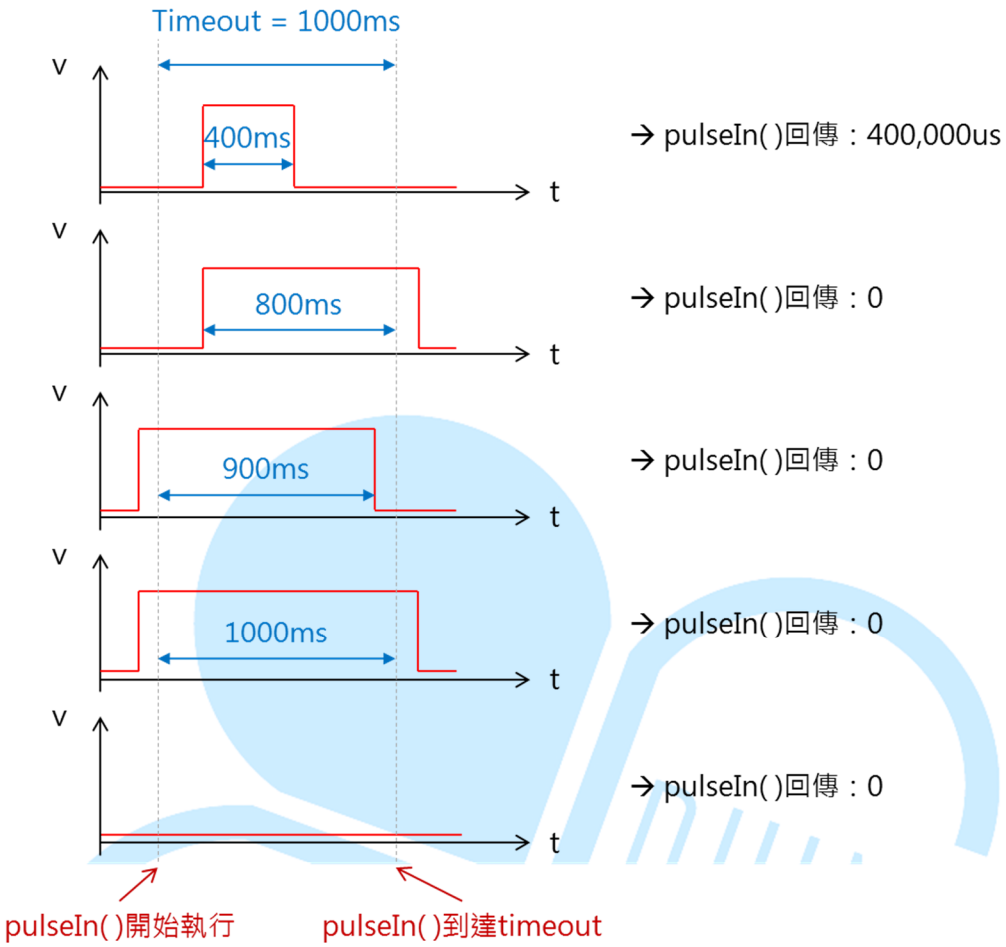
这个范例程序跟上一个是类似的功能，但改使用 pulseIn( )函式来实作，读者一样不用更改接线，请在 86Duino Coding IDE，输入以下程序代码：

```
// 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);
}
```

此范例程序跟上个一样是侦测用户按按钮的时间长度，使用前须开启 Serial Monitor，当窗口显示「Please press button...」讯息时，可选择按或不按下按钮，程序会在 2 秒的时间内侦测是否有「按下并放开按钮」的动作，如果有按下按钮则显示压按的时间长度，否则等到 2 秒的侦测时间结束后送出「No pulse...End」讯息；流程延迟 2 秒后便重新来一次；执行结果如下图：



虽然与上个范例都是侦测压按时间长度，但由于 pulseIn( )函式并不是利用中断机制侦测，只有当程序执行到此处才会触发侦测动作，因此程序代码须集中在 loop( )内依序执行。pulseIn( )拥有两种形式：pulseIn(pin, value)与 pulseIn(pin, value, timeout)两种，函式运作机制如下图所示：



pulseIn( )函式，測量波型 HIGH 运作概念图

以 value 设定为 HIGH 为例。此函式执行后，便会在时间限制（timeout）内侦测指定脚位的电压波型变化状况（value），只有当侦测期间内有完整波型（波型先上升后下降，或是先下降或上升）才会回传测量数值，其余状况皆回传 0。value 可设定为 HIGH 或 LOW，当侦测到 pin 为 HIGH 或 LOW 则开始计时，直到波型变为相反则停止计时，回传中间经过的时间。若时间限制内波型没有变成相反或是波形根本没有改变，则回传 0。

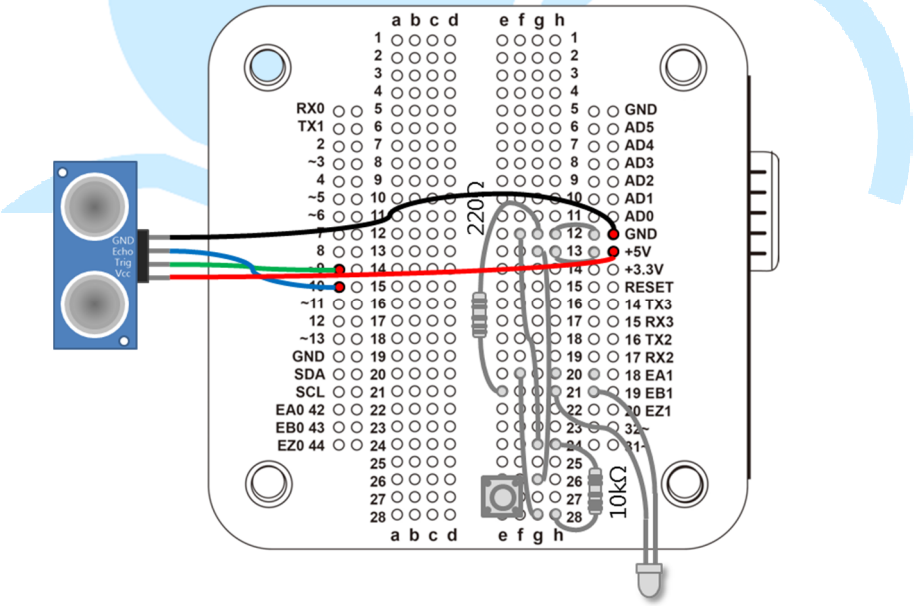
程序一开始先宣告 unsigned long max_duration = 2000000（单位为μs）用来当作 timeout 的时间长度，表示之后 pulseIn( )函式最多会等待 2 秒的时间才结束继续往下执行，若使用 pulseIn(pin, value)，则预设 timeout

时间为 1 秒。setup( )里须使用 pinMode(btn_pin, INPUT)将欲侦测的脚位设定为 INPUT；之后 loop( )里先以 Serial.print( )提示用户按下按钮，然后使用 unsigned long duration = pulseIn(btn_pin, HIGH, max_duration)做按钮动作变化的侦测，由于按钮前述已提过采用 Normal LOW 的设定，因此 pulseIn( )养当然是设定为 HIGH 的侦测模式，若是按钮采用 Normal HIGH 的接法，那就得把 pulseIn( )设定为侦测 LOW 啰。另外 pulseIn( )回传的数值单位是μs，所以使用上要小心，此处为了跟前一范例程序统一单位，因此印出前先除以 1000 变为 ms 单位。读者也可以把 timeout 参数设定为 5 秒之类试试，当出现提示按钮讯息时没有按下按钮，那程序真的会等待 5 秒才继续往下执行喔。

藉由此范例，读者应可看出 pulseIn( )与中断侦测的不同之处，一种可马上对外界变化做出反应，一种则需执行到函式才有侦测的作用，专题使用时当然也得好好做出最佳选择啰。读者可以思考看看，此范例如何修改为测量反应速度用的功能？ 当 LED 亮起，用户须马上按下按钮，然后程序显示从 LED 亮到按下按钮之间的间隔时间，这该如何修改呢？

四、 第四个程序

pulseIn( )函式虽然反应较慢，但还是有它的用武之处，接着便使用这个函式来做点有趣的应用。此范例以一个相当实用的超音波传感器来做练习，这里我们选用 HC-SR04 这颗容易购买到的型号。市面上当然也有其他型号可选择，通常是测量距离之类的规格不太一样，但基本使用方法都差不多，购买时注意使用说明即可。通常超音波传感器会有 3~4 条接线脚位，其中 2 条为 5V 跟 GND，其他则是讯号脚；读者请参考下图接好线路，旧有的线路可以不用拆除：



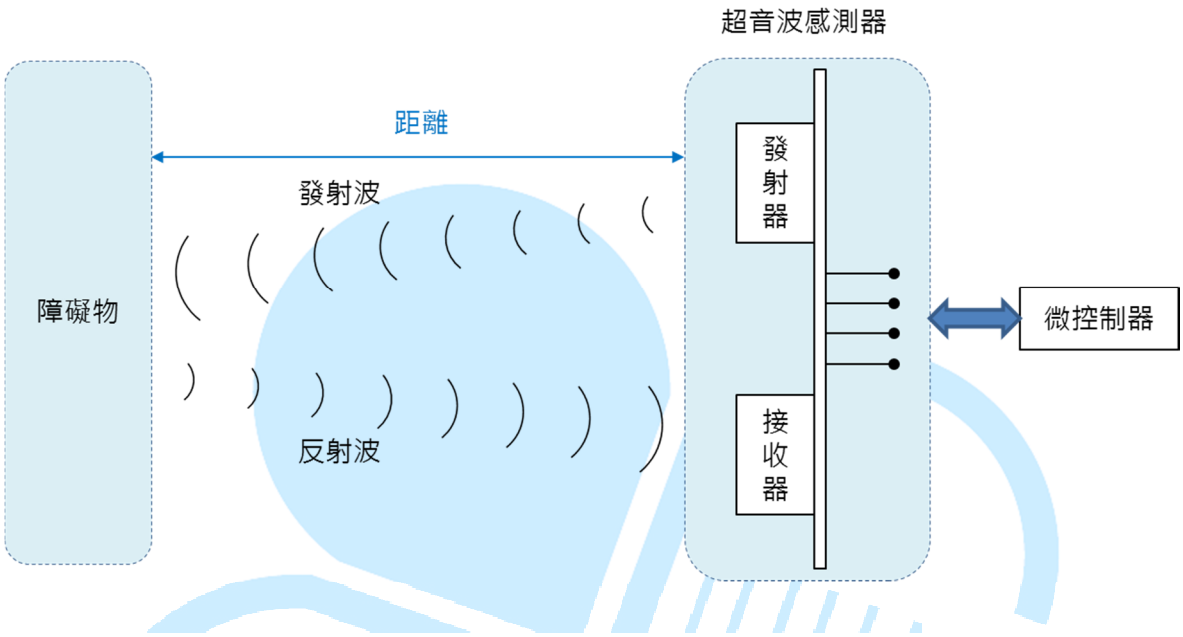
接着打开 86Duino Coding IDE，输入以下程序：

```
#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
```

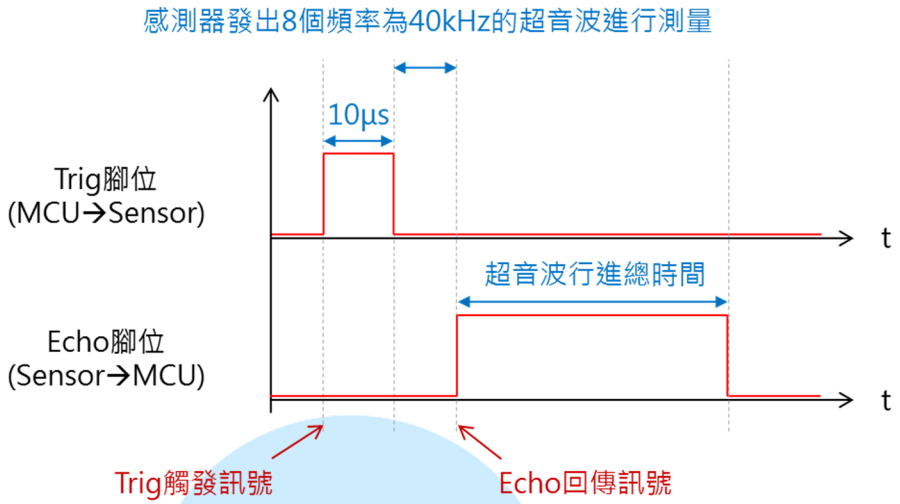
```
    unsigned int LED_ON_count_max = 1;// LED 點亮長度變數  最大值
    unsigned int LED_OFF_count = 0;// LED 熄滅長度變數
    int timeout = 12000; //  設定 pulseIn 的持續時間
    float  Sound_speed  =  343.0f  *  100  /  1000000;  //  音速：340.29,
343.2   m/s  換算為  cm/us

    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;
      }
```

此程序功能为类似一般汽车倒车侦测器，当障碍物距离传感器越近，则 LED 闪烁速率越快，同时 Serial Port 也会印出实际换算距离。解释程序做法前，我们先来看看超音波传感器的测量原理，示意图如下：



测量流程为，当控制器送出触发讯号给传感器，传感器发射源便发出超音波，开始计算接收器收到反射波的时间，接着将测量到的数值传回控制器。以 HC-SR04 而言，讯号脚为 Trig 与 Echo 两根，Trig 是控制器送给传感器表示开始进行测量的讯号；Echo 则是传感器测量完毕，回传到控制器的讯号（也有其他型号的超音波传感器是把这两只针脚做成同一脚位，但原理基本相同）。使用方法为，平常 Trig 与 Echo 脚位都是 LOW 的状态，当控制器想触发测量时，便由 Trig 脚位送出一个约 10μs 的 HIGH 讯号，接着变成 LOW，传感器便会开始测量。若传感器有收到反射波，那么 Echo 脚位便会送出一个长度为「从发射超音波到接收到回波经过时间」的 HIGH 讯号，单位为μs，这便是 pulseIn()可以派上用途的地方了，超音波传感器控制讯号示意图如下：



程序一开始宣告几个运作相关设定参数，如测量间隔时间、pulseIn( ) 的 timeout 时间、LED 闪烁参数、音速换算参数等等；接着 setup( )里设定超音波传感器讯号脚位的 I/O 模式，并初始化控制脚位的状态。读者可以看到 loop( )里面没有多少程序代码，因为一般应用传感器时，可能安装不只一颗传感器，同样的程序代码便要重复使用很多次，所以利用函式（Function）的方式便可以将程序代码适度的「模块化」，需要用时重复呼叫即可，省去重复撰写的时间，程序维护方便性与可读性也会增加。此范例程序处理超音波传感器的程序代码被移至 Get_US( )这个函式，当呼叫 Get_US( )时，函式便处理超音波传感器所有相关流程，再将结果回传给 loop( )主要流程，做 Serial Port 显示数值与 LED 亮暗的调控。

Get_US( )函式里程序一开始，为避免其他地方不小心改到 Trig 脚位状态，以便产生一个「干净」的讯号，因此先使用 digitalWrite(trigPin_1, LOW) 设定脚位状态，delayMicroseconds(2)是为了让脚位状态稳定，接着再用 digitalWrite(trigPin_1, HIGH)送出触发波型，持续 10μs 再变为 LOW，让超音波传感器开始工作。触发传感器后，接着就是使用 pulseIn(echoPin_1, HIGH, timeout)函式，等着接收传感器 Echo 脚的回传讯号了。pulseIn( )

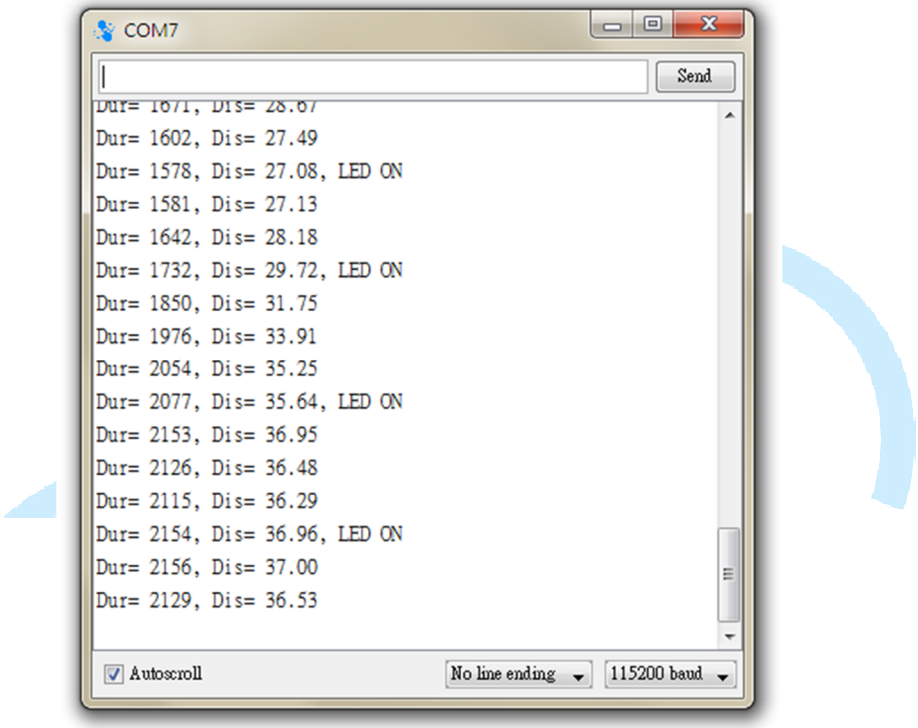
这个函式的回传数值形式是 long，因此宣告一个 long 变量 duration 储存传感器回传值，这就是超音波在传感器与障碍物之间行进的总时间长度，因此接着换算成实际距离时须将此数除以 2 才是单趟时间。

计算距离的算式使用「距离 = 时间 * 速度」，而音速一般经验公式是：音速 = 331 + 0.6 * 摄氏温度，此范例是假设温度摄氏 20 度，用固定数值 343（m/s），读者也可以使用随温度变化的公式做计算啰。另外因为音速单位是公尺/秒（m/s），而传感器回传值是微秒（μs）单位，想要换算成公分（cm）显示，所以要先做换算：343（m/s） = 343 * 100 / 1000000（cm/μs）。实际程序代码 float distance = float(duration) / 2 * Sound_speed 中，将 duration 先转换为浮点数，以便在数学计算中保留较多小数点。最后 Get_US( )回传计算过后的距离值，可做其他程序代码任何用途。例如自走车便可以用此数值判断障碍物距离自己多远等等。

此范例程序的 loop( )流程中，取得传感器数值后便将距离藉由 Serial.print( )印出在 Serial Monitor 上面观察，另外用来调整 LED 亮暗的频率。此处需要注意的是，LED 亮暗间隔虽然可以用 delay( )实作，但由于在 loop( )里面使用变动的延迟时间也会一并让超音波测量间隔变得忽快忽慢，所以此处改用计数器的作法来改变 LED 亮暗频率。开头宣告几个全局变量 LED_ON_count、LED_OFF_count、LED_ON，LED_ON 为 LED 亮度状态，当 LED 亮，每次 loop( )都会让 LED_ON_count 变量+1，上限可自由设定，此范例使用 1 当作上限，当变量达到上限，便让 LED 熄灭，LED_ON 变成 false，LED_ON_count 归零，进入 LED 熄灭阶段。LED_ON 为 false 时，则换成 LED_OFF_count 每次+1，上限为 int(distance/10)，由于 loop( )

间隔设定为 100ms，以测量距离为 200cm 而言，可以让 LED 有约 2 秒的熄灭时间，读者也可以修改成其他数字看看效果。

程序执行后，就可以观察 Serial Monitor 以及 LED 的亮暗，了解目前测量到的距离，程序执行结果如下图：



由超音波传感器的使用原理，读者也可以想想，如何用改用中断或是 digitalRead( )的方式，去测量 Echo 脚的讯号长度？ 另一方面，如果把 LED 换成简单的蜂鸣器与配套电路，是不是就变成简单的障碍物侦测器或倒车警示器了呢?