EduCake 的 Analog I/O 脚位介绍与功能应用

一、 Analog I/O 介绍

- **模拟 vs 数字：**

前篇已经介绍过 86Duino EduCake 的基本规格、IDE 程序撰写接口使用方式，与 Digital 脚位的基本使用方式，本篇就来介绍模拟(analog)输入/输出脚位的功能以及应用方式。

介绍「模拟」信号前，须先与所谓「数字」信号做个比较，前篇所提到的「数字」信号，一般而言是具有两种状态的电压讯号，高于某个程度的电压为 HIGH(一般以 1 表示)，低于某程度的电压则为 LOW(一般以 0 表示)，因此以 86Duino 为例，digitalWrite( )可以使某个脚位输出 HIGH 或 LOW 的电压，而 digitalRead( )则可以读取某个脚位电压 HIGH 或 LOW 的状态，对于开关、按钮这类仅有两种状态表示方式的信号而言已足够使用。

而本篇文章所提的模拟信号，指的是具有连续状态的数值，例如：亮度、温度、湿度、响度、长度、角度、重量……等自然界常见的物理量，各类型的传感器则会负责将物理量转变为电压或电流；但程序执行时所需的数值是 0 与 1 组成的数字信号，因此当某个应用须取得上述「连续状态」物理量给程序使用时，便需要称为「模拟数字转换器」(Analog to Digital Converter，简称 ADC)的电路来做转换，下面以一 ADC 电路作为范例解释转换原理：

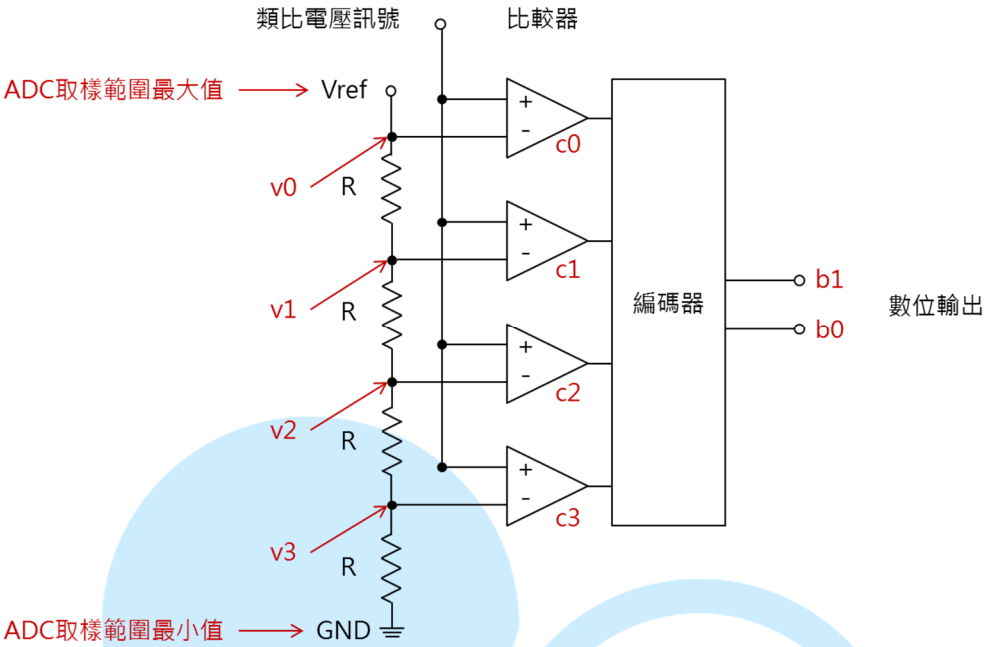


图 1

图 1 电路称为「Flash ADC」，是概念较易理解的一种。这里举拥有四个比较器的当作范例，输入处为模拟电压讯号与参考电压讯号 Vref；每个比较器「-」端接到 v0~v3，分别为 Vref、Vref(3/4)、Vref(1/2)、Vref(1/4)，当「+」端的电压讯号比「-」端高，则比较器输出为 1，反之为 0。FLASH 编码器则负责把 c0~c3 的输出转变成对应的二进制编码，此范例由于比较器只有 4 阶，因此实际二进制编码只有 2 位。若须 4 位输出，则须 16 个比较器，读者可依此类推。

以 Vref 等于 5V 为例，v0~v3 分别为 5V、3.75V、2.5V、1.25V；当模拟信号为 1.5V，则比较器 c0~c3 输出 0/0/0/1，二进制值为 01(十进制值为 1)；当模拟信号为 3.9V，则比较器 c0~c3 输出 0/1/1/1，二进制值为 11(十进制值为 3)，更多比较器的状况依此类推。所以此例的 ADC 输入范围为 0~5V，转换位数 2，仅有 4 阶的分辨率，每阶电压为 5/4 = 1.25V。

ADC 的转换电路与设计原理多样化，也是一门学问，本文仅简介其原理与应用，读者若有兴趣钻研可搜寻相关数据阅读。

- **ADC 规格与基本换算方式：**

提到 ADC 规格时，一般会描述 ADC 能接受的输入范围、转换位数等；由前段的 ADC 原理解释便能了解，任何低于或超出 ADC 输入范围的数值只能用数字最大或最小值表示，而转换位数则影响取样的分辨率，位数越高，越能细致地表示取样的信号。

举例 1：某脚位的 ADC 输入范围为电压 0~5V，而转换位数为 10bits，则 0V 时取样的数字值为 0(二进制值为 0000000000)，而 5V 时取样的数字值为 1024(二进制值为 1111111111)，0~5V 中间共有 2 的 10 次方=1024 个分辨率，表示此 ADC 取样分辨率为 5/1024，约等于每刻度有 0.004883V，可以想象一下一根温度计在 0~5 度之间细分为 1024 个等分；当读取到的数值为 1000 时，代表取样的原始信号为 1000*5/1024，约等于 4.883V。

举例 2：某脚位的 ADC 输入范围为电压 1~7V，而转换位数为 12bits，则每刻度有(7-1)/4096，约为 0.00146V 的电压分辨率；当读取到的数值为 1000 时，代表取样的原始信号为 1000*6/4096+1，约等于 2.465V。

因此使用 ADC 时，须了解它的基本规格才能进行数值的换算；以 86Duino EduCake 为例，输入范围为电压 0~3.3V，分辨率默认为 10 位。

- **传感器应用前须知：**

前面提到各类型的传感器会将物理量转变为电压或电流讯号，但 ADC 通常是读取电压讯号，况且各种传感器讯号范围也不一定符合 ADC 的输入规格，因此传感器使用前，另一个注意事项是须将输出讯号对应到 ADC 的输入范围，以下举例来做说明。

举例 1：电阻式的传感器，如一般旋转式可变电阻、压力传感器、光敏电阻、电阻式温度计等等，针对环境一定范围的变化改变本身电阻值。使用时可如下面图 2 所示，称为「电阻分压法」，其中 Vcc 通常使用接近 ADC 输入最大电压值，例如 EduCake 为 3.3V。传感器至 ADC 输入电压为 Vcc*R1/(R1+R2)，图 2(b)可变电阻的 R1+R2 为定值，而图 2(a)中的 R2 可自定义，通常选择 1KΩ 左右当作限制电流用，也不宜与 R1 相差太多。如果 Vcc 使用其他固定电压值，也可依照上述原理将 R1 端电压调整到适合 ADC 的输入范围。

图 2(a)　　图 2(b)

举例 2：以测量太阳照度常用的日射计(Pyranometer)来说，常见的是输出 4~20mA 的规格，这种传感器的特性是它输出电流信号。这种情况下，可用图 3 的方式，将传感器串接一电阻 R1，例如 250Ω，则依 V=I*R，可将 4~20mA 转变为 1~5V，便可让 ADC 进行取样，使用时须注意电流的输出方向。

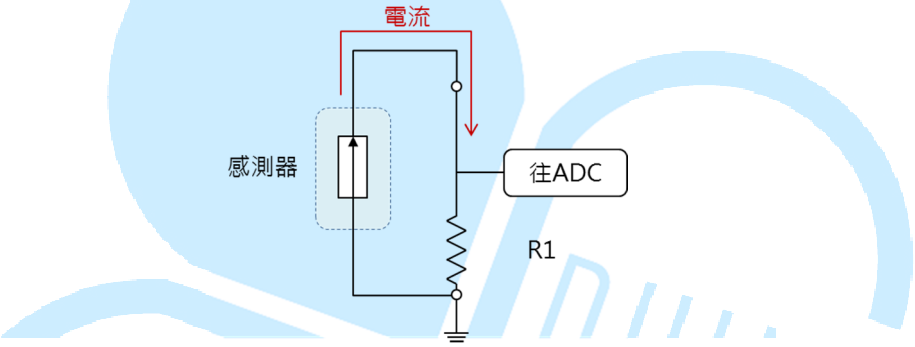


图 3

- **模拟输出**

前面提到了 ADC 将模拟讯号转换为数字数据进行处理，而模拟输出则刚好相反，是将程序内的数字数据转换成模拟的讯号输出，一般来说要将数字数据转换为「纯正」的模拟讯号，会使用到「数字模拟转换器」(Digital to Analog Converter，简称 DAC)的电路，但并非所有输出装置都需要使用纯正的模拟讯号。例如灯泡、直流有刷马达之类的装置，只要调整特定时间内电流或电压通过的时间比例，便可达到调整亮度，或是控制转速的效果。以 86Duino EduCake 而言，使用的模拟输出也是采用这种称为「PWM」(Pulse Width Modulation，脉冲宽度调变)的方法，PWM 并不局限用在达到模拟输出的效果，之后会有专门文章来做介绍，本文则着重在 PWM 模拟模拟输出的应用。

如图 4，由上到下分别为 25%、50%、75%、100%的 PWM 脉冲宽度比例(也称为 duty cycle)，由于人眼的视觉暂留现象，当频率够高，PWM 电压用在 LED 或灯泡上的时候，就会相当于亮度的变化，EduCake 的模拟输出频率约是 1000Hz。也可以用前篇文章的 digitalWrite( )配合 delay( )、delayMicroseconds( )函式来达到相似的效果，但频率太低则会出现明显亮灭情形。

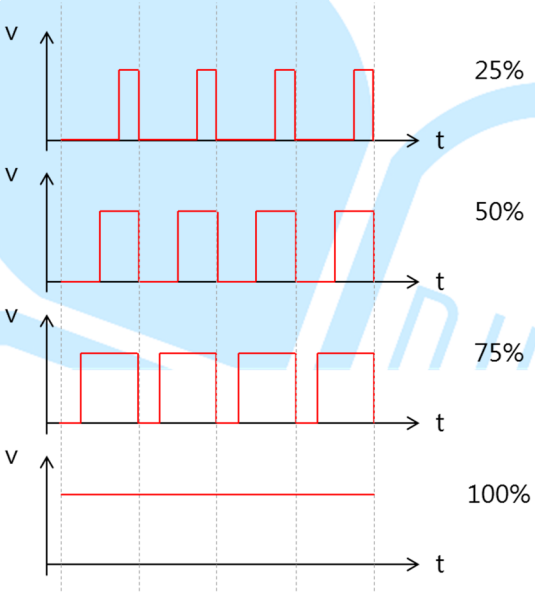
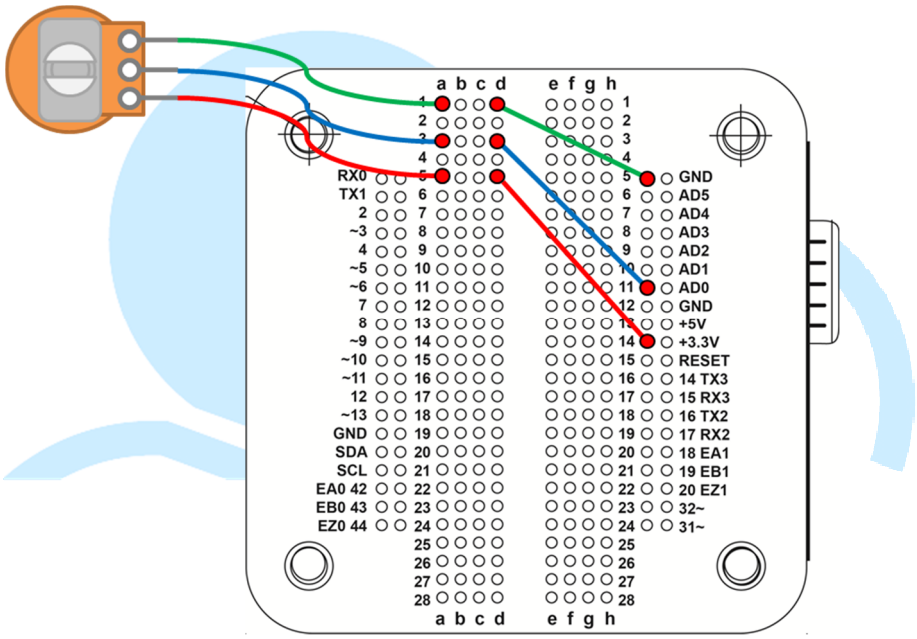


图 4

讲完原理部分，接下来便开始以实际程序操作进行练习。

二、 第一个程序 – 练习 analogRead()

一开始先使用 EduCake 来练习简单的模拟输入功能，这里使用的函式是 analogRead()，需准备的零件为一般电子材料行就很容易购买到的可变电阻，挑个 10kΩ即可，接线方式如下图：



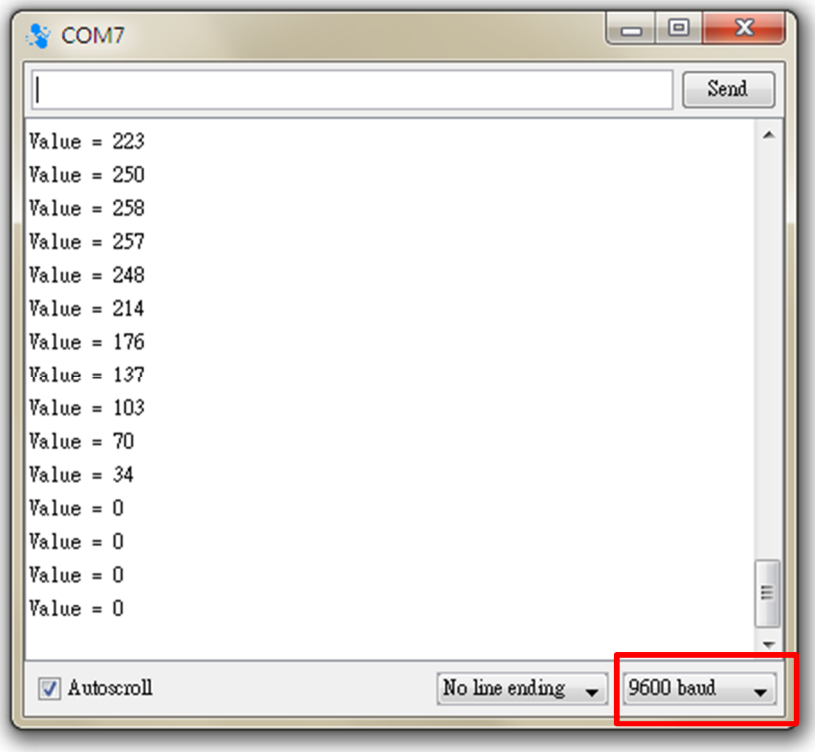
此接线方式就是前面介绍原理的图 2(b)，请注意通常可变电阻有 3 只脚，中间那只脚就是改变电阻比例的碳刷部位，其他两脚没有正负极分别，分别接上 EduCake 的 3.3V、GND 即可；但如果反过来接，会影响转动圆棒时的电阻值大小变化顺序，如果配合转动机构时须注意这点。此范例我们会将 ADC 读取到的数值，利用与计算机连接的 USB 传输线将数值印出来观察。接着请开启 86Duino Coding 100 的 IDE 接口，输入以下程序：

```
const int analogInPin = A0;// 宣告模拟输入脚位编号

int sensorValue = 0;// 宣告储存 ADC 读值

void setup() {
```

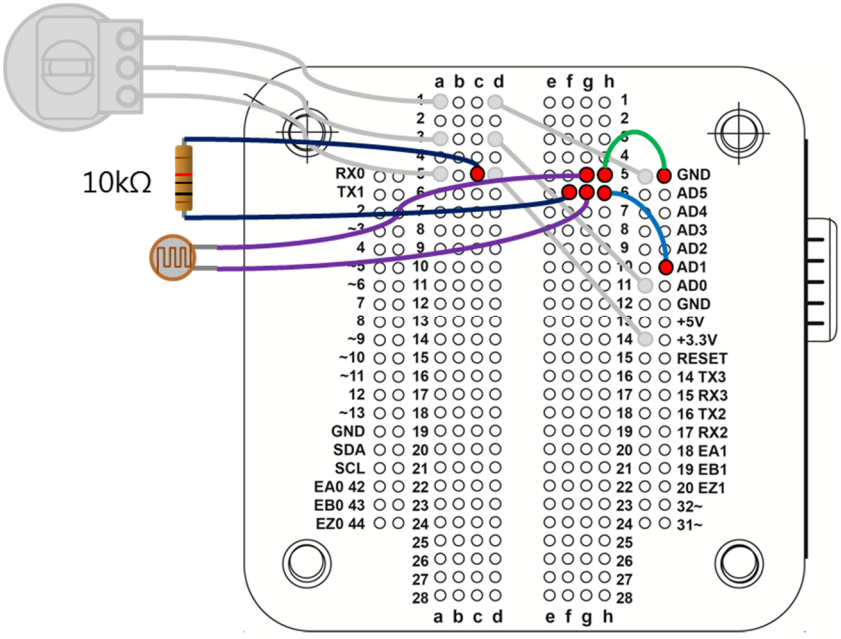
```
  Serial.begin(9600);// 设定与计算机通讯的 Serial Port 速度
}

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

  Serial.print("Value = " );// 印出字符串
  Serial.println(sensorValue, DEC);// 印出数值

  delay(100);// 加上延迟时间
}
```

程序一开始在 setup( )设定与计算机端传输的 baudrate(通常称为鲍率)数值，然后 loop( )里单纯执行 analogRead( )，并将数值以 Serial.print( )、Serial.println( )函式回传到 Serial Monitor 做显示。此两函式差别在于 println 会在印完字符串后换下一行，也可以方便阅读。注意 Serial.println(sensorValue, DEC)这行，第二个参数「DEC」表示将第一个参数以「十进制」印出，读者也可以尝试其他例如 BIN、HEX 等，将数值以二进制或 16 进位印出。delay(100)是让程序循环每次暂停 100ms(0.1 秒)，间隔时间依读者实际程序需要而定。

按下 Upload，上传程序后，请打开 Tools > Serial Monitor(或用 Ctrl+Shift+M 快捷键)，就可以看到一连串的 ADC 读值啰，如下图。接着若转动可变电阻的圆棒，便可发现数值在 0~1023 之间变化。若打开 Serial Monitor 发现没有反应，可能要检查一下右下方的「baud」设定有没有跟程序 setup( )里的 Serial.begin(9600)一样喔。之后任何程序都可以用这个方法来做变量监看、除错，是满好用的工具。



接着来改变一下传感器种类，利用图 2(a)的电路来做练习，为了后续几个程序方便，之前的接线先不拆，新增接线如下图：



电路使用光敏电阻来测试，也是电子材料行常见的组件，这里挑的是一般室内光照下约为数十 kΩ 的型号，搭配另一固定电阻值为 10kΩ 的电阻。

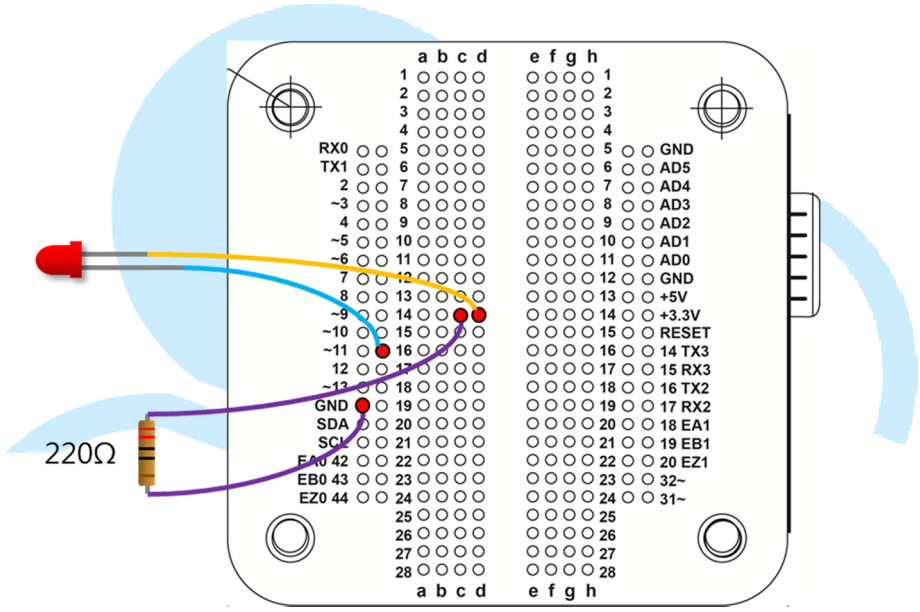
光敏电阻一端接地，一端接到 AD1 脚位，10kΩ电阻一端则接到 3.3V；由于光敏电阻照光时电阻值减小，不照光时电阻值极大(甚至到 MΩ等级)，所以依据电路接法，可预期亮度高时 ADC 读值减小，而遮蔽时 ADC 读值变大。程序修改如下：

```
const int analogInPin = A1;// 宣告模拟输入脚位编号
```

基本上只是把模拟输入脚位改成 AD1 而已，测试方法同上例，读者可以测试看看结果，是不是就如同描述的一样呢？读者到这边应该可以感觉到，简短几行程序便可以简单地让电路感知外界的变化现象，当然也就可以做更进阶的功能啰。

三、 第二个程序 – 练习 analogWrite( )

接下来我们实作一个 LED 呼吸灯效果来练习模拟输出功能。这个范例请读者准备一颗常见的 LED，颜色可挑红黄绿之类即可，另外需一个 220Ω 电阻，电阻主用用来保护 LED 用，以免电流过大烧毁 LED。接线图如下，一样不拆原有线路，另外接线。



LED「+」端(长脚部位)接至 EduCake 11 脚位，注意板子脚位前方标有「~」的皆为可输出 PWM 的脚位；接着输入下方程式：

```
const int analogOutPin = 11;// 宣告模拟输出脚位编号
const float Freq = 0.5;// LED 闪烁速率
unsigned long time;// 时间变量
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));// 计算 LED 亮度
  //Serial.print("LED_light = " );// 印出字符串
  //Serial.println(LED_light, DEC);// 印出亮度数值
  analogWrite(analogOutPin, LED_light);// 输出 PWM 波形至 LED

  delay(50);// 加上延迟时间
}
```

从第一行开始宣告一连串呼吸灯会用到的设定值，例如脚位编号、LED 闪烁速率、时间变量、LED 亮度等，主循环里每次会先取得时间变量，使用 millis( )函式可取得程序开始执行至目前为止的「毫秒数」，然后利用余弦(或正弦 sin 也行)公式 cos( )，利算每次需要的 LED 亮度，此处的数学计算式为：

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

模拟输出使用 analogWrite( )函式，参数范围为 0~255，0 代表 PWM 脉波周期没有 HIGH 的部分，而 255 代表周期中 HIGH 的比例为 100%，依此类推 128 代表 50%。由于 cos( )函数输出-1~1 的小数值，因此乘上震幅 127，再加上 128 便可将数值映像到 1~255；2πft 则用在调整 cos( )函数的频率，f 单位为 Hz，由于 time = millis()取得时间值为毫秒，因此 *0.001 将单位变成秒；最后加上 delay( )延迟时间。

此范例设定 LED 闪烁速率为 0.5Hz，执行此程序后，便可看到 LED 灯以 2 秒一次的频率进行渐变的亮暗，读者可以试着加大 delay( )，会发现 LED 灯的亮暗频率没变，但变化过程明显有阶段性的感觉，不够顺畅，因此此范例使用 50ms 的延迟时间，程序约以 20Hz 计算并更新 LED 亮度，视觉效

果较佳，这样就简单完成一个呼吸灯效果了。另外也示范另一种 Serial Monitor 的用法，读者可以将上面程序中的

```
//Serial.print("LED_light = " );// 印出字符串
//Serial.println(LED_light, DEC);// 印出亮度数值
```

改成：

```
for(int i=0;i<LED_light/10;i++){
   Serial.print(" " );// 印出空白字符串
}
Serial.println("|");// 印出符号字符串
```

这几行程序利用 LED 亮度值，让 Serial.print( )印出数个空白符号，最后再加上一个其他符号，便可以形成长短不一的字符串；Upload 程序完成并开启 Serial Monitor，便会看到一连串符号组成弦波如下图：