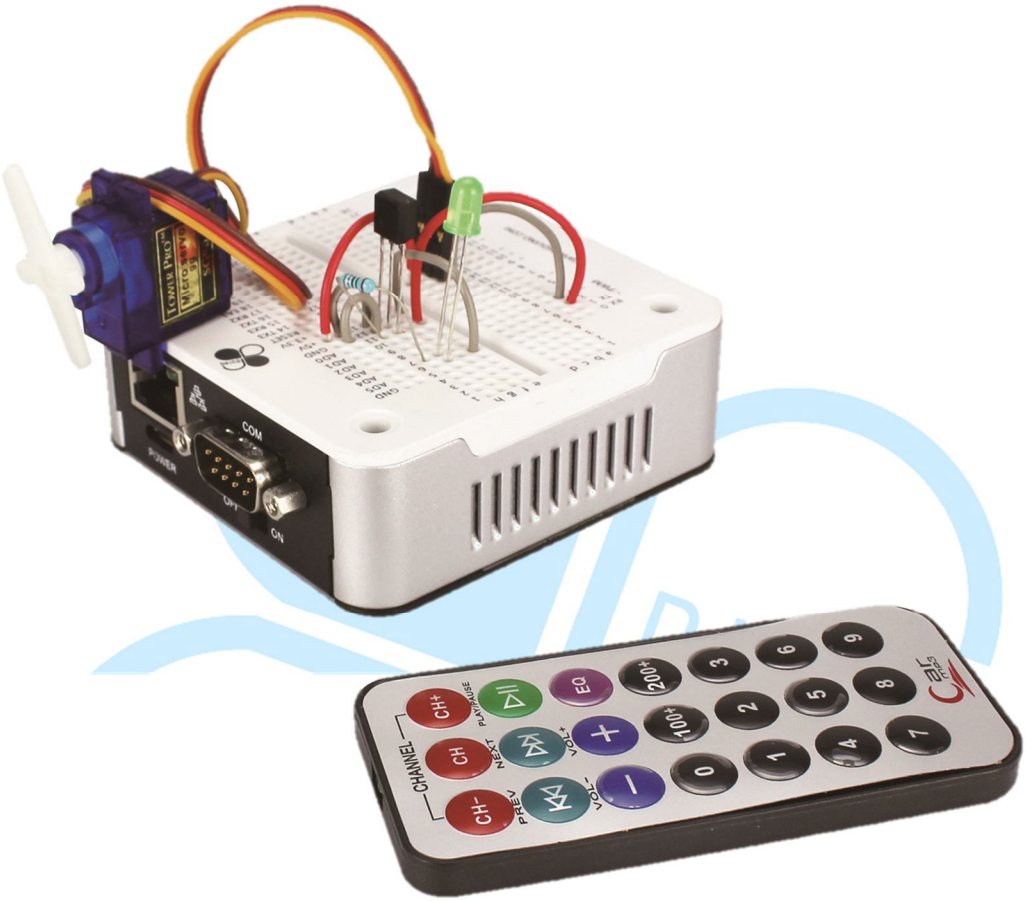
EduCake 红外线收发功能实作



一、 红外线发射/接收原理与应用介绍

前面章节讲解过几种 86Duino EduCake 可用的通讯方式，像是 UART Serial Port、I^2C 等等，不过这些都是以「实体电线连接」的方式通讯的，本篇则另外来介绍一种生活中常见的「无线」通讯方式，也就是「红外线通讯」，并且使用 86Duino EduCake 来实作红外线发射、接收等功能，实际动手玩玩看。

红外线属于电磁波的其中一个类别，因此得先简单介绍电磁波的频谱。一般人眼可见光的波长范围约在 400nm（紫色）~700nm（红色）之间。

波长小于紫色，范围约为 10nm~400nm（频率大于紫色）的就是常听到的「紫外线」；而波长大于红色，范围约为 700nm~1mm（频率小于红色）的就是「红外线」（或红外光），也因为英文称为「Infrared」而常以缩写「IR」来表示；紫外线（Ultraviolet）则常使用「UV」来表示。红外线位于电磁波频谱范围中如下图 1 所示：

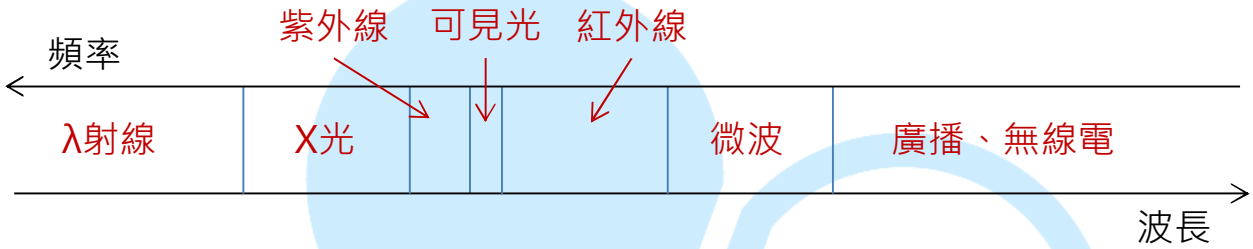


图 1. 电磁波频谱简易示意图

红外线频谱内，还依照可侦测范围的不同分为近红外线、短波红外线、中波红外线、长波红外线、远红外线等等，如果读者想知道更多关于电磁波的讯息，可以参考：

http://en.wikipedia.org/wiki/Electromagnetic_radiation
http://en.wikipedia.org/wiki/Visible_spectrum
http://en.wikipedia.org/wiki/Infrared
http://en.wikipedia.org/wiki/Ultraviolet

虽然红外线看不到，但生活中其实到处都有着红外线的存在，例如军事电影中常看到的「红外线热像仪」便是以红外线传感器接收各种不同温度物体发出的红外线，并以不同颜色标示，以便人类从屏幕上观察温度的分布状况。除了观察温度外，红外线的其他应用例如：卫星或大型望远镜观察外层空间的星球、家中冷气/电视/相机/光盘播放器等电器常见的红外线遥控器、部分智能型手机配备的红外线数据传输功能、掌上型游戏机近距离联机对打功能、人体动作感应灯等等，用途相当广泛。

本篇文章要介绍的，便是如何实作红外线无线通信这项功能。红外线通讯跟之前介绍过的通讯方式一样有发射端与接收端装置。实作时，发射端是由主控电路搭配红外线发射源，以及接收端主控电路搭配一个红外线接收器所组成的。红外线发射器外型就像是一个 LED，只是他会发出特定波长的红外线电磁波，外型如下图 2（有些发射器是深蓝色的）。

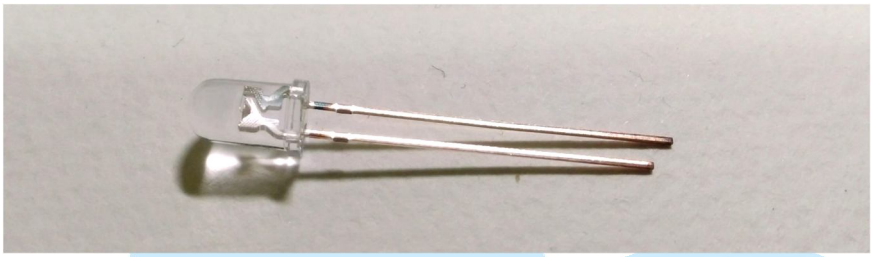


图 2. 红外线发射器外观图

而红外线接收器内部结构较为复杂，包含 IR 接收器以及讯号处理电路。接收器整体是一个三脚位的封装，其中两脚位是电源、接地，另一脚位是讯号输出用途，外观如下图 3。

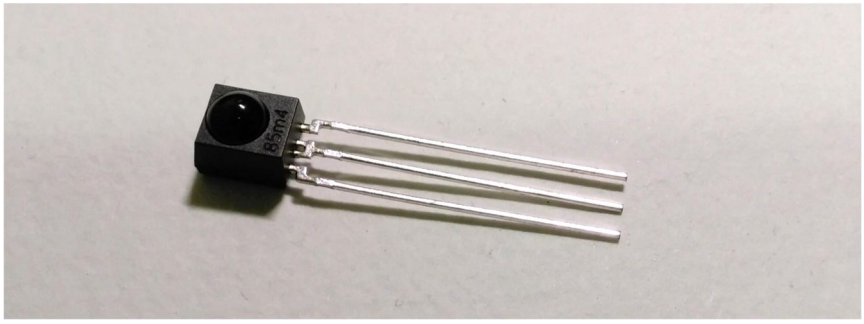


图 3. 红外线接收器外观图

由于环境中充斥着各种不同来源的红外线电磁波，为了确保接收端能够正确收到数据，一般会在传送端将数据「调变（modulation）」成特定频率的红外线讯号。而接收端收到红外线讯号后，讯号处理电路会进行滤波（仅留下特定频率范围的讯号），并针对滤波后的讯号进行「解调（demodulation）」，输出成为 HIGH 或 LOW 讯号，HIGH/LOW 定义依接收器而定，因此接收端搭配的装置或处理器便能够知道传送端送出的数据是什么了。红外线传输流程简易示意图如下图 4。

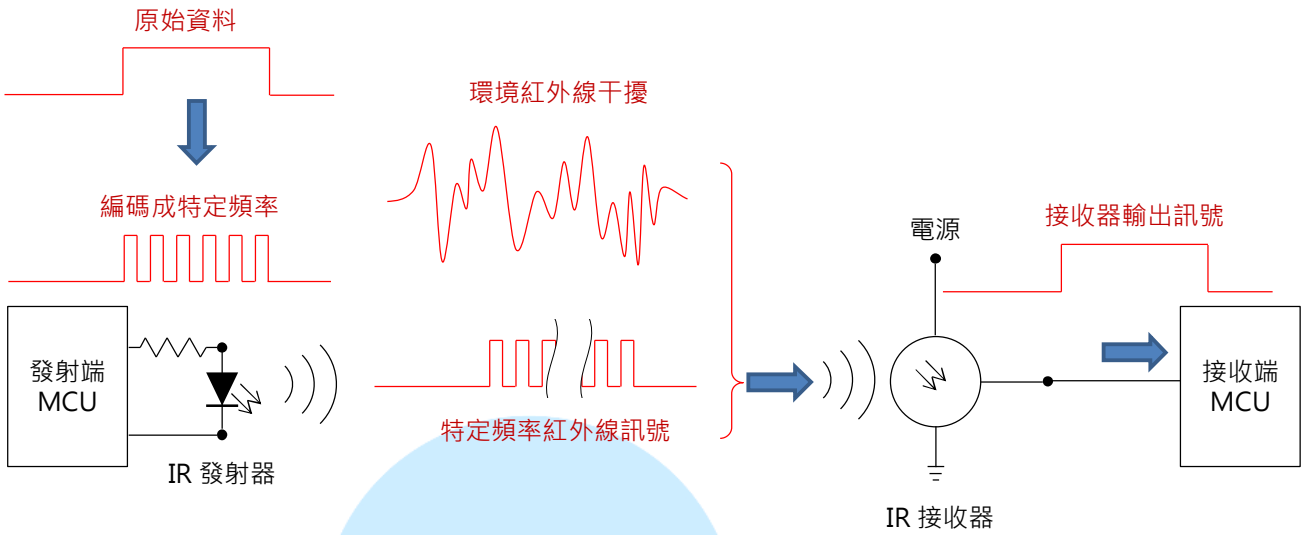


图 4. 红外线通讯流程示意图

上述这种传送/接收特定红外线的方式，需要两端在相同的频段内，一般常见有 36 kHz、38 kHz、40 kHz、56 kHz 等。须注意这里提到的「特定频率」并不是指电磁波光谱的频率，而是红外线发射器以特定间隔时间，间歇地发射红外线的频率，也称为「载波频率」。调变（modulation）与解调（demodulation）便是将数据与载波频率结合，以及从载波频率中取出的过程，为无线传输常用的方法。另外，由于红外线发射器比一般 LED 需要略大的电流，因此使用载波频率间歇式发射红外线讯号也有避免发射器持续通电过热的额外优点。

除了以载波频率将数据编码以外，不同电器用品，甚至不同厂牌之间，也会有自己的红外线「通讯协议（protocol）」。由于上述的红外线讯号在传送前、接收后都是数字讯号，因此通讯协议为一连串的 HIGH、LOW 电压组成，依据各厂商的讯号定义不同而有不同的位长度、字段定义等等。因此，A 厂商的遥控器与 B 厂商的电器之间，即使选用相同波长的红外线，相同的载波频率，也不一定能够兼容进行遥控喔。

以一个 SONY 遥控器为例，使用的载波频率是 40 kHz，遥控器上按键被按下后，红外线发射器会送出不同的位序列，表示对应的 IR 遥控指令，

接收的电器便依据不同的指令执行对应的动作，例如电视机开关电源、调整音量、选取频道等等。下图 5 所示为一 SONY 遥控器发射的红外线指令，读者可以看到指令共包含了起始讯号与数据讯号，其中 0、1 分别以不同长度的 HIGH/LOW 宽度编码。另外，当按键被持续按着时，依据通讯协议的规定，有的会间隔一段时间重复送出特定数量的相同指令，也有的是一直重复送出指令。因此如果想要遥控某厂牌的电器装置，就得先了解所用的红外线传输规格为何了。

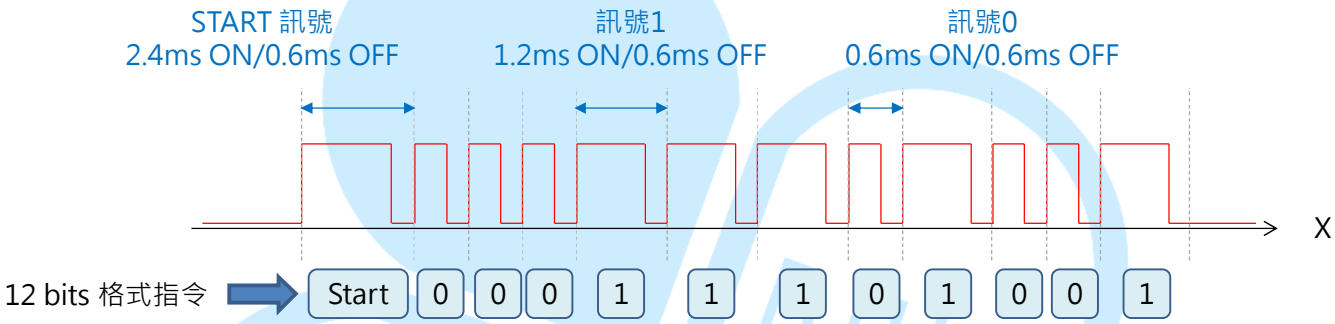


图 5. 红外线传送通讯协议范例

关于 SONY 红外线通讯协议，可参考：

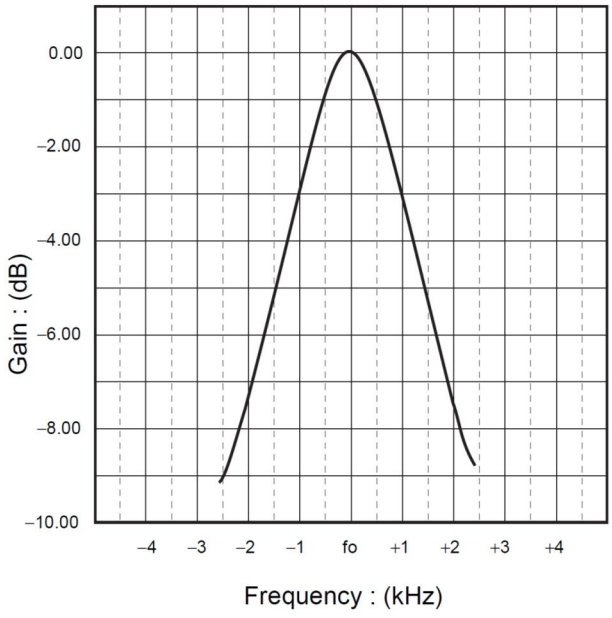
http://www.righto.com/2010/03/understanding-sony-ir-remote-codes-lirc.html
http://users.telenet.be/davshomepage/sony.htm
http://www.cypress.com/？docID=46755
http://www.sbprojects.com/knowledge/ir/sirc.php

接下来的 86Duino EduCake 实作上，我们选用一般容易买到的 920nm 红外线发射器，以及笔者在电料行找到的 RPM6938 红外线接收器来进行功能实作，此接收器的脚位定义如下图 6：

針腳定義，依序為：
[訊號　GND　電源]

图 6. 红外线接收器脚位定义

依据 RPM6938 的规格表，电源针脚使用 5V，接收讯号的载波频率是 37.8kHz，水平有效接收角度约 70 度，垂直约 60 度左右；无接收到讯号时输出脚为 HIGH，反之为 LOW。须注意虽然规格表上面标示的载波频率是 37.8kHz，但实际上是在这个频率「有最佳的接收效果」，如规格表上的图标，频率略高/略低时接收的效果会减少，因此频率稍有误差也是没问题的。



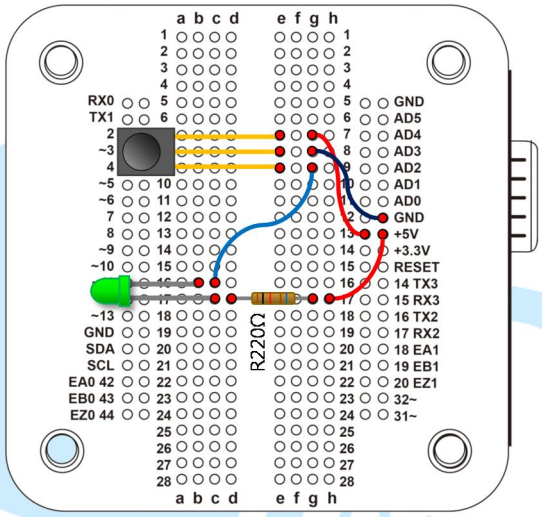
接收器的型号众多，若读者实作时买的是其他款式红外线接收器，除了载波频率外，还得看清楚接收器型号，因为有些接收器电源针脚跟接地针脚是相反的，得查清楚规格才能接线喔。另外也有两支针脚，长得像深蓝色

LED 的红外线接收器，但此类接收器没有载波频率的处理功能，须注意别买错。

红外线发射器接线方式则跟一般 LED 相同，长脚为正端，短脚为负端。接下来，就让我们来练习实作红外线传输的功能吧。

二、 第一个练习 – 测试红外线接收器

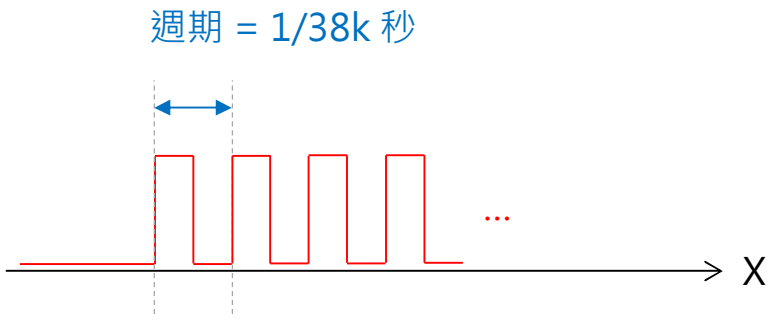
第一个练习，我们暂时不用 EduCake 的程序，先以简单的方式测试一下红外线接收器与发射器是否正常，读者请先依下图接线：



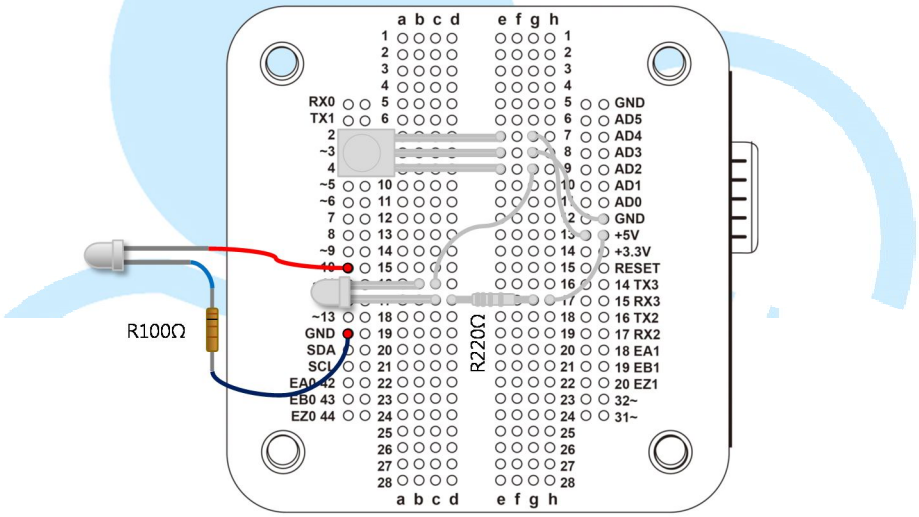
红外线接收器电源接 5V，接地脚接 GND，输出讯号脚接 LED 的负端，LED 正端串联一个 220 奥姆电阻到 5V。由于此红外线接收器没接收到讯号时，输出脚为 HIGH，因此 LED 平常不会亮。接着读者可以使用手边的红外线遥控器，朝着接收器按下任意按钮，如果遥控器使用的载波频率大约是 38kHz 左右，就会看到 LED 灯断断续续地闪烁啰，也表示手上的红外线接收器是可以正常运作的。

如果手边没有红外线遥控器，只有前面提到的红外线发射器（长得像 LED 的）零件怎么测试呢？这就需要使用程序来控制红外线发射器的闪烁速率了。

这里选用的接收器对约 38kHz 的载波频率有反应，因此需要使用控制器发射如下图的波形：



要产生这样的波形，第一种方法可以使用 digitalWrite( )来产生，读者请先依下图接线：



将红外线发射器正端接 pin 10，由于发射器所需电流比 LED 大些，因此负端串接一个 100Ω 电阻后接到 GND（而不是 LED 常搭配的 220Ω 电阻），红外线接收器的电路不变动。

接着请打开 86Duino Coding IDE，输入以下程序代码：

```
int IR_pin = 10;

void SendIR( )
{
  for( int i = 0; I < 800; i++ )
  {
    // ON + OFF 26us ~= 38.4kHz
    digitalWrite( IR_pin, HIGH );
    delayMicroseconds( 13 );

    digitalWrite( IR_pin, LOW );
    delayMicroseconds( 13 );
  }
  Serial.println( "IR send." );
}

void setup( ) {
  Serial.begin( 115200 );

  pinMode( IR_pin, OUTPUT );
}

void loop( ) {
  SendIR( );
  delay( 1000 );
}
```

此程序在「void SendIR( )」函式里，使用「digitalWrite( )」搭配「delayMicroseconds( )」来产生 PWM 波形，38kHz 波形的周期时间约是 26us 左右，因此使用 LOW/HIGH 各 13us。编译并上传程序至 86Duino

EduCake 后，红外线发射器每秒都会送出一次频率约 38.4kHz 的波形，就可以观察到红外线接收器输出脚位所接的 LED 闪烁啰。

第一种这是简易的测试方式，若使用硬件来实作所需的 PWM 波形，频率控制也会更精确。86Duino EduCake 有提供一个好用的函式库称为「TimerOne」，此函式库可用来设定 86Duino EduCake 内 CPU 的 32 位计数器功能，并控制特定脚位输出 PWM 讯号。由于是硬件在定时触发电压 HIGH/LOW，因此时序控制上会比程控脚位电压切换要更精确。功能与前面测试程序相同的程序代码如下：

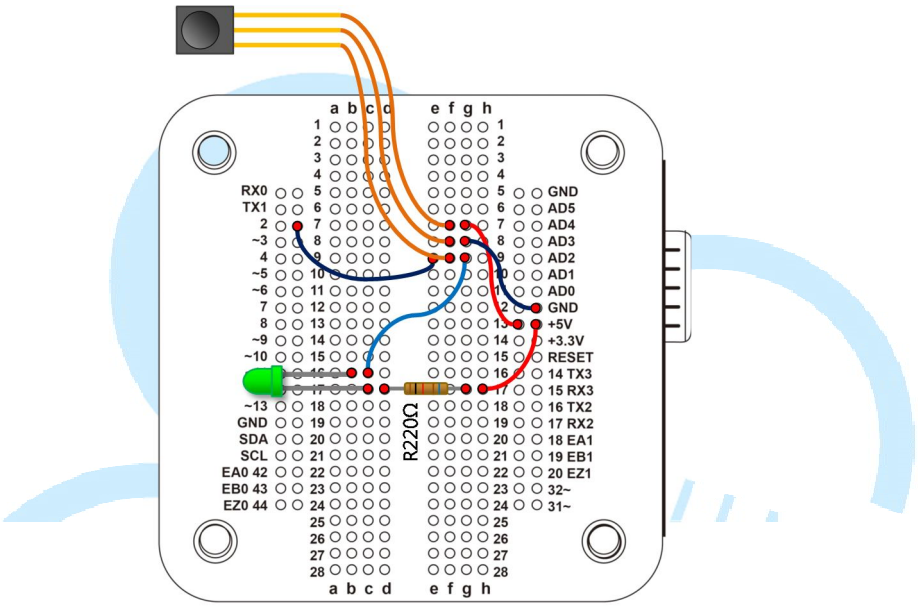
```
#include "TimerOne.h"
int IR_pin = 10;
void SendIR( )
{
  Timer1.pwm( IR_pin, 512, 26 );// pin, duty (512=50%), period(us)
  delay( 20 );
  Timer1.disablePwm( IR_pin );
  delay( 20 );
  Serial.println( "IR send." );
}
void setup( ) {
  Serial.begin( 115200 );
  pinMode( IR_pin, OUTPUT );
  Timer1.initialize( 26 );// TimerOne initialize, period(us)
}
void loop( ) {
  SendIR( );
  delay( 1000 );
}
```

想要使用 TimerOne 函式库，程序一开始要先使用语法「#include "TimerOne.h"」，接着在 setup( )阶段使用「Timer1.initialize(period);」做 TimerOne 对象初始化，其中参数字段 period 为定时器的周期，单位为 microsecond。

「void SendIR( )」函式里，则使用「Timer1.pwm(IR_pin, 512, 26);」语法来设定 IR 脚位启动 PWM 功能，512 表示 50%的 duty，period 为 26μs，持续时间为 20ms。「Timer1.disablePwm(IR_pin);」语法则用来关掉红外线发射器脚位的 PWM 输出。如此也可以达到测试红外线发射器、接收器的目的喔。

三、 第二个练习 – 认识红外线通讯格式

了解红外线接收/发射的基本原理后，接着就要进展到较实用的阶段了。这个练习题我们来认识一下实际电器用品传输的「通讯协议」长甚么样子。读者请依下图接线：



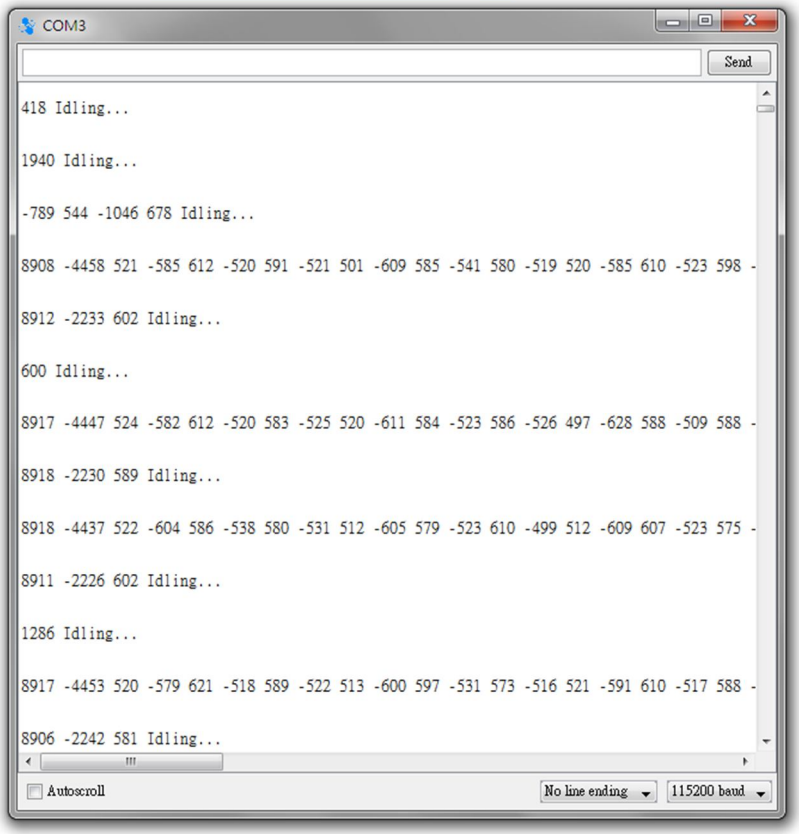
此电路保留之前的接线，但额外将红外线接收器输出脚位接到 86Duino EduCake 的数字脚位 2，以便程序读取电压状态。完成接线后，请打开 86Duino Coding IDE，输入以下程序代码：

```
int IR_rec_pin = 2;// IR 接收器输出脚位
int IRstate = LOW;// IR 接收器输出脚位状态
int IRstate_last = LOW;// IR 接收器输出脚位状态
long int time_last = 0;// 纪录上一次 IRstate 变化的时间

boolean isIdle = true;// 是否在等待 IR 讯号模式
const long int durationMax = 10000;// 一段时间没变化就进入等待 IR 讯号状态，单位 us
```

```
    const long int durationMin = 400;// 电压状态不变的最小持续时
间，单位 us
    void IR_rec_Check( )
    {
        IRstate = digitalRead( IR_rec_pin );// 读取脚位状态

        if( IRstate != IRstate_last ){// 这次跟上次脚位状态不同

          long int timeNow = micros( );// 取得目前时间
          long int dT = timeNow - time_last;// 上一次脚位状态变化
经过的时间

          if( dT >= durationMax && !isIdle ){
            isIdle = true;
            Serial.println( "Idling...\n" );
          }
          else if( dT < durationMax && dT > 400 ){
            isIdle = false;
            Serial.print( IRstate == HIGH?  dT : dT ); Serial.print( "
" );
          }

          // 记录此次时间
          time_last = timeNow;
        }
        // 记录此次状态
        IRstate_last = IRstate;
    }

    void setup( ) {
      Serial.begin( 115200 );
      pinMode( IR_rec_pin, INPUT );// 设定针脚 I/O 模式
      IRstate = digitalRead( IR_rec_pin );// 取得脚位状态初始值
      IRstate_last = IRstate;
    }
```

```
void loop( ) {

  IR_rec_Check( );
  delayMicroseconds( 20 );
}
```

编译并上传程序后，请打开 Serial Monitor，使用手边可以兼容所这个红外线接收器的遥控器（可以先使用前面练习提供的方法测试载波频率是不是符合），朝着接收器按几下按钮看看，Serial Monitor 上便会显示出一系列的数字，正值代表输出脚位电压 HIGH 的持续时间，负值则是电压 LOW 的持续时间，如下图：



这里须注意，当红外线发射器「有发射」时，接收器收到红外线是输出 LOW，反之为 HIGH，因此这里读到的正值实际上是「没有」收到红外线的时候喔。

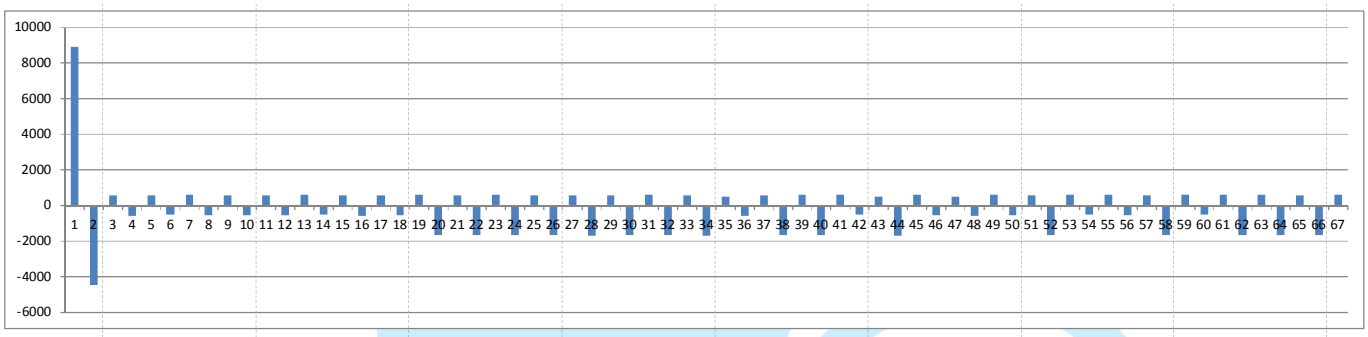
这个程序代码会每隔 20μs，呼叫「IR_rec_Check( )」函式，使用「digitalRead( )」读取接收器输出脚位的电压状态，当电压有变化时，使用「micros( )」语法记录下当时的毫秒时间，以便跟下一次状态变化时间点相减取得持续时间长度。程序代码里面设定了 durationMax、durationMin 做为过滤条件，大于 durationMax 则当作一段时间没接收到，进入 idle 状态；反之小于 durationMin 则当作噪声不予理会。使用μs 作为取样间隔，是因为红外线接收器会将持续数百μs（依规格）的红外线讯号判断为 HIGH 或 LOW，低于的当作噪声滤除，因此设定数十μs 当作取样频率，取样间隔若太长，侦测出的持续时间会较不精准。

读者会注意到，即使没有按下遥控器时，接收器也可能会受环境红外线影响，输出持续时间长度不等的 HIGH/LOW 讯号，因此为了实际传输的正确，就需要「通讯协议」的说明了。「通讯协议」是传输/接收两端预先定义好的一连串信号顺序定义，包含讯号开头、结尾、数据长度等等，跟之前提过的 Serial 传输概念很类似。

笔者在这里测试使用的是车用 MP3 遥控器，外观如下：

以这个遥控器当例子，每次按下按钮 0，都会收到持续时间约为「8883 -4487 524 -599 591 -525 594 -516 522 -610 580 -527 596 -513 514 -600 595 -524 593 -1632 596 -1630 516 -1697 606 -1631 602 -1625 519 -1707 609 -1630 585 -1630 525 -595 630 -1594 523 -1704 610

-509 509 -1717 606 -510 517 -609 615 -503 582 -1630 594 -534 591 -506 520 -1707 607 -509 514 -1715 602 -814 1405 -1632 518」的数据序列，时间长度单位为μs，这不太方便观察，我们使用 Excel 来画成图表如下：



去头去尾刚好剩下 64 个数据，如果按下其他按钮则会得到其他数据序列。从图上可以看到，这个遥控器使用 HIGH 时间约 600μs、LOW 时间约 600μs 或 1600μs 的组合当作数据位。由于外界干扰不容易整串刚好符合这个格式，其他厂商的红外线编码方式也会不同，因此当传送/接收两端以特定的数据序列沟通时，就能将外界环境的干扰影响尽量降低。读者也可以使用手边各种遥控器试试看会收到怎样的数据组合喔。

四、 第三个练习-1 – IRremote 函式库使用介绍(接收)

经过前一章节的练习，红外线的通讯协议这么多种，该如何实际应用呢？在 86Duino EduCake 里面有提供移植自 Ken Shirriff 的「IRremote」函式库。这个函式库可以用通讯协议来传送与接收红外线资料，并且支持了 NEC、Sony SIRC、Philips RC5、Philips RC6、Sharp、Panasonic、JVC、Sanyo、Mitsubishi 等协议格式，并可输出原始数据供观察。这个练习就让我们使用 IRremote 函式库来接收红外线讯号，并且介绍一些实用的重要功能吧。

86Duino EduCake 电路维持与练习 2 一样的接线方式，接着请读者打开 86Duino Coding IDE(注意需使用 104 版本以上才有 IRremote 函式库)，输入以下程序代码：

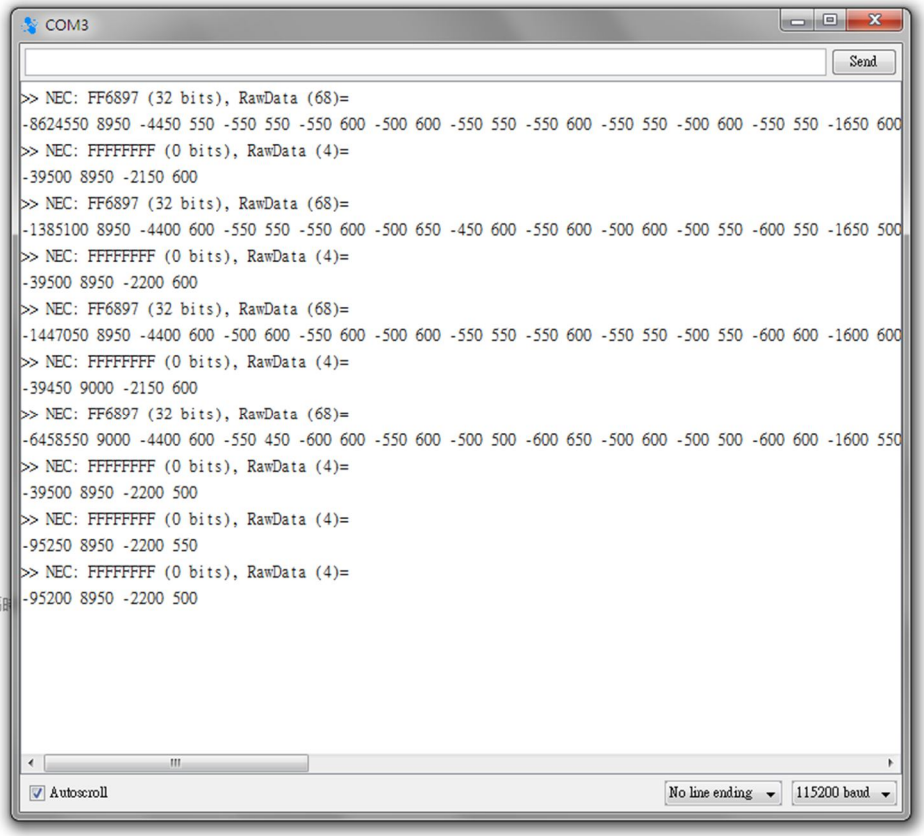
```
#include <IRremote.h>
int IR_rec_pin = 2;// IR 接收器输出脚位

IRrecv IRrecver( IR_rec_pin );// IRremote 函式库接收用对象

decode_results results;// 译码结果存放数据用


void Print_IRdecodeResult( decode_results &decodeResults )// 印出译码成功的讯息以便观察
{
  int dataLength = decodeResults.rawlen;

  switch( decodeResults.decode_type )
  {
    case NEC:
```

```
            Serial.print( ">> NEC:\t" );
                break;

            case SONY:
                Serial.print( ">> SONY:\t");
                break;

            case RC5:
                Serial.print( ">> RC5:\t");
                break;

            case RC6:
                Serial.print( ">> RC6:\t");
                break;

            case DISH:
                Serial.print( ">> DISH:\t");
                break;

            case SHARP:
                Serial.print( ">> SHARP:\t");
                break;

            case SANYO:
                Serial.print( ">> SANYO:\t" );
                break;

            case MITSUBISHI:
                Serial.print( ">> MITSUBISHI:\t" );
                break;

            case PANASONIC:
                Serial.print( ">> PANASONIC(addr=\t" );
```

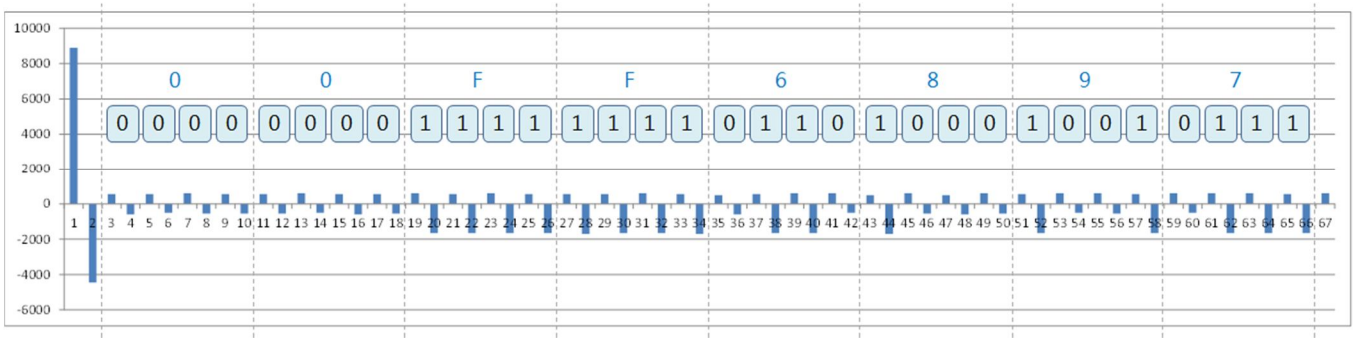
```
            Serial.print( results.panasonicAddress );
                Serial.print( "):\t" );
                break;

            case JVC:
                Serial.print( ">> JVC:\t" );
                break;

            case UNKNOWN:
                Serial.print( ">> Unknown:\t" );
                break;

            default:
                break;
        }

        Serial.print( decodeResults.value, HEX );// 通讯协议数据字段，
以 16 进位显示
        Serial.print( " (" );
        Serial.print( decodeResults.bits, DEC );// 总共收到几位数据
        Serial.print( " bits), " );

        Serial.print( "RawData (" );
        Serial.print( dataLength, DEC );
        Serial.println( ")= " );

        // 印出原始资料序列
        for ( int i = 0; i < dataLength; i++) {
            int data = decodeResults.rawbuf[i] * USECPERTICK;
            // buf 存放取样个数，以及每次取样间隔时间 USECPERTICK
            if ( (i % 2) == 1 ) {// 偶数存放的是 HIGH
                Serial.print( data, DEC );// HIGH
            }
```

```
        else {
              Serial.print( -data, DEC );// LOW
            }
            Serial.print( " " );
          }
          Serial.println( );
        }


        void setup( )
        {
          Serial.begin( 115200 );
          IRrecver.enableIRIn( );//  初始化接收对象
        }

        void loop() {
          if ( IRrecver.decode( &results ) )
          {
            Print_IRdecodeResult( results );
            IRrecver.resume(  );//  译码完需要呼叫这一行，才能再继续接
收后续资料
          }
        }
```

编译并上传程序后，请打开 Serial Monitor，接着拿遥控器朝红外线接收器按几下，以笔者前面练习所用的车用 MP3 遥控器而言，会出现下图画面：



当这个遥控器按下按钮 0 时，IRremote 函式库会进行接收并译码，此函式库方便之处在于可以辨识收到的资料序列属于何种编码方式，以及共收到了多长的数据。以笔者使用的遥控器为例，IRremote 函式库译码出来的通讯协议是NEC格式，基本上原始数据的时间序列长度跟练习 2 是一样的，使用 Excel 画波形图如下：



以 NEC 通讯协议的接收器输出脚位电压来说：

λ 开始讯号：为约「9ms 的 HIGH 与 4.5ms 的 LOW」组合。

λ 数据逻辑 0：为约「560μs 的 HIGH 与 560μs 的 LOW」组合。

λ 数据逻辑 1：为约「560μs 的 HIGH 与 1690μs 的 LOW」组合。

λ 重复指令：为约「9ms 的 HIGH 与 2.25ms 的 LOW 和 560μs 的 HIGH」组合。

此遥控器按键 0 发出的数据，是对应到 0X00FF6897 的编码，64 个 HIGH/LOW 刚好是 32 个逻辑位，共 4 个 byte，其中前 16 个位为装置地址编码，后 16 个位为指令编码。

另外，若持续按着某个按钮，在收到第一次的数据编码后，后面收到的则是「重复指令」FFFFFFFF，约每 110ms 会重复一次，例如持续按着音量+按钮，装置就可以依此重复指令持续增加音量了。因此即使房间里一样都是 NEC 的装置，也可以根据辨识不同的地址、指令而做对应动作，不会有冲突发生，这就是通讯协议的妙用。

要使用 IRremote 函式库，此程序一开始须引用「IRremote.h」，然后使用「IRrecv IRrecver(IR_rec_pin);」语法宣告一个红外线接收器对象，传入的参数为接收器输出脚位号码。「decode_results results」为一个 class 数据结构，里面存放了：

λ decode_type：标示通讯协议编码类型。

λ panasonicAddress：Panasonic 协议专用的地址字段。

- λ value：通讯协议数据字段数值。
- λ bits：数据序列总位个数。
- λ unsigned int *rawbuf：原始 HIGH/LOW 取样资料。
- λ rawlen：取样资料个数。

setup( )阶段则需使用「IRrecver.enableIRIn( )」执行红外线接收器对象的初始化，接着在 loop( )里面定期呼叫「IRrecver.decode(&results)」，此函式会扫描目前接收到的红外线数据，并检查编码，如果译码成功，就将译码后的结果放入 results 变量中，并回传 true；反之译码不成功或还未收完数据则回传 false。当回传 true 时，便可以使用「Print_IRdecodeResult( )」函式，印出译码成功的各种讯息来观察啰。

与练习 2 相较之下，此范例程序可以让读者进一步看看手边各种遥控器是甚么通讯协议，以及按钮按下后会传出甚么指令数据啰。

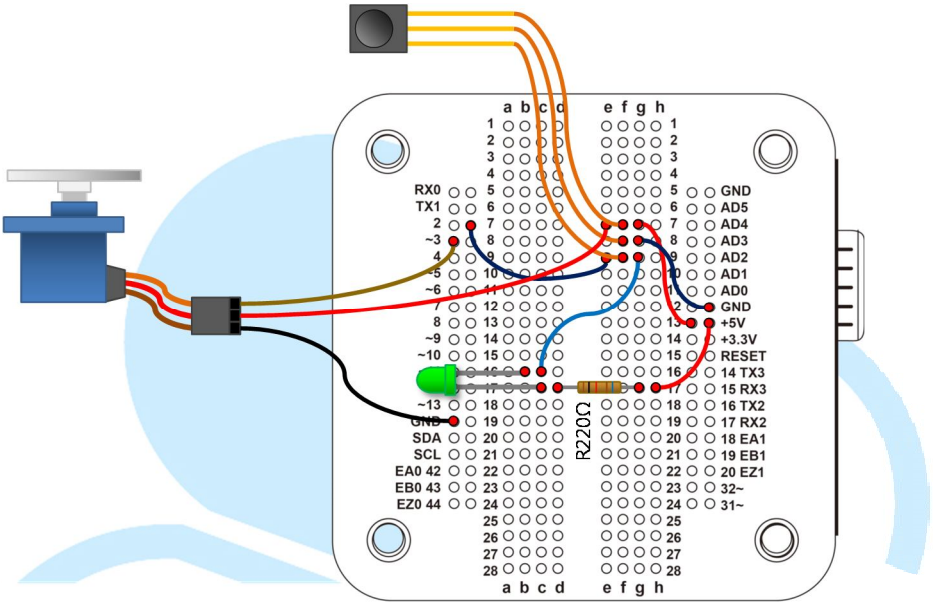
关于 NEC 红外线通讯协议，也可以参考：

http://mcudiy.blogspot.tw/2010/11/22-irinfrared-nec-protocol.html

http://www.sbprojects.com/knowledge/ir/nec.php

五、 第三个练习-2 – IRremote 函式库使用介绍(接收)

这个练习将前一个稍微变复杂些，利用手边的遥控器让 EduCake 可以「动起来」。读者请依下图接线：



接好线后，在 86Duino Coding IDE 输入以下程序代码：

```
#include <IRremote.h>
#include <Servo.h>

int IR_rec_pin = 2;// IR 接收器输出脚位
int servo_pin = 3;// Servo 输出脚位

IRrecv IRrecver(IR_rec_pin);// IRremote 函式库接收用对象
decode_results results;// 译码结果存放数据用

Servo servo_0;// Servo 物件
typedef enum
{
  DIR_NONE = 0,
  DIR_LEFT,
```

```
    DIR_RIGHT
  } ServoDir;// 转动方向定义
  int servoDir = DIR_NONE;// Servo 转动方向
  unsigned int ServoPosition = 1500;// Serco 角度

  void Print_IRdecodeResult( decode_results &decodeResults )//
印出译码成功的讯息以便观察
  {
    if( decodeResults.decode_type == NEC )
    {
      switch( decodeResults.value )
      {
        case 0x00FFA25D:// CH- buttom
          servoDir = DIR_LEFT;
          ServoPosition += 50;
          break;

        case 0x00FF629D:// CH buttom
          servoDir = DIR_NONE;
          ServoPosition = 1500;
          servo_0.writeMicroseconds(ServoPosition);// 置中
          break;

        case 0x00FFE21D:// CH+ buttom
          servoDir = DIR_RIGHT;
          ServoPosition -= 50;
          break;

        case 0xFFFFFFFF:// Repeat
          if( servoDir == DIR_RIGHT )
          { ServoPosition -= 50; }
          else if( servoDir == DIR_LEFT )
          { ServoPosition += 50; }
```

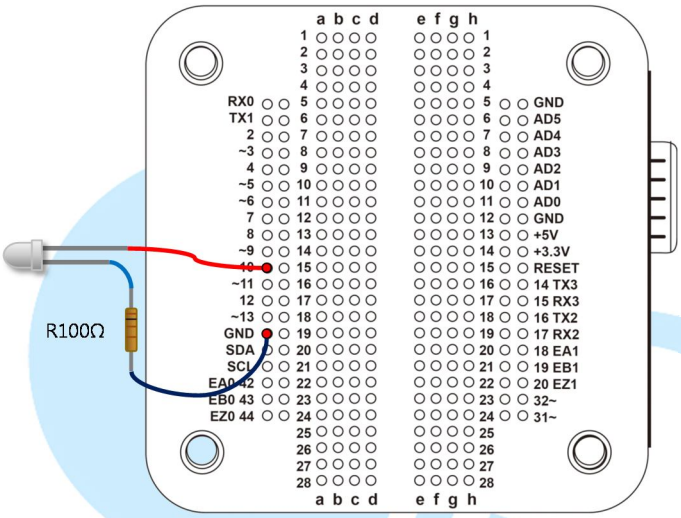
```
        ServoPosition = constrain( ServoPosition, 1100, 1900 );// 限制在安全范围
                servo_0.writeMicroseconds( ServoPosition );// 控制角度

                Serial.print( "ServoPosition = " );
                Serial.println( ServoPosition );
                break;

            default:
                break;
        }
    }
}

void setup( )
{
    Serial.begin( 115200 );
    IRrecver.enableIRIn( );// 初始化接收对象
    servo_0.attach( servo_pin );// 设定 Servo 连接脚
}

void loop( ) {
    if ( IRrecver.decode( &results ) )
    {
        Print_IRdecodeResult( results );
        IRrecver.resume( );// 译码完需要呼叫这一行，才能再继续接收后续资料
    }
}
```

编译并上传成功后，便可以用遥控器按钮控制 EduCake 连接的 RC Servo 左右转啰。整体程序流程跟前一个练习是一样的，主要在

「Print_IRdecodeResult( )」函式内做了改变。由于笔者使用的遥控器，「CH-」按钮对应的红外线协定数值为 0x00FFA25D，「CH」按钮为 0x00FF629D，「CH+」按钮为 0x00FFE21D，持续压着的重复码为 0xFFFFFFFF，因此 switch-case 语法这边使用了这几个数值，且必须为 NEC 编码才做对应的 Servo 控制动作，以减低接收错误的影响。如果读者使用了其他遥控器，由于通讯协议不同、按钮对应数值不同，就需要修改这些地方才能正常运作喔。

六、 第三个练习-3 – IRremote 函式库使用介绍(发射)

实作过 IRremote 函式库的接收功能后，接下来这个练习来实作发射红外线指令的功能。请依下图电路接线

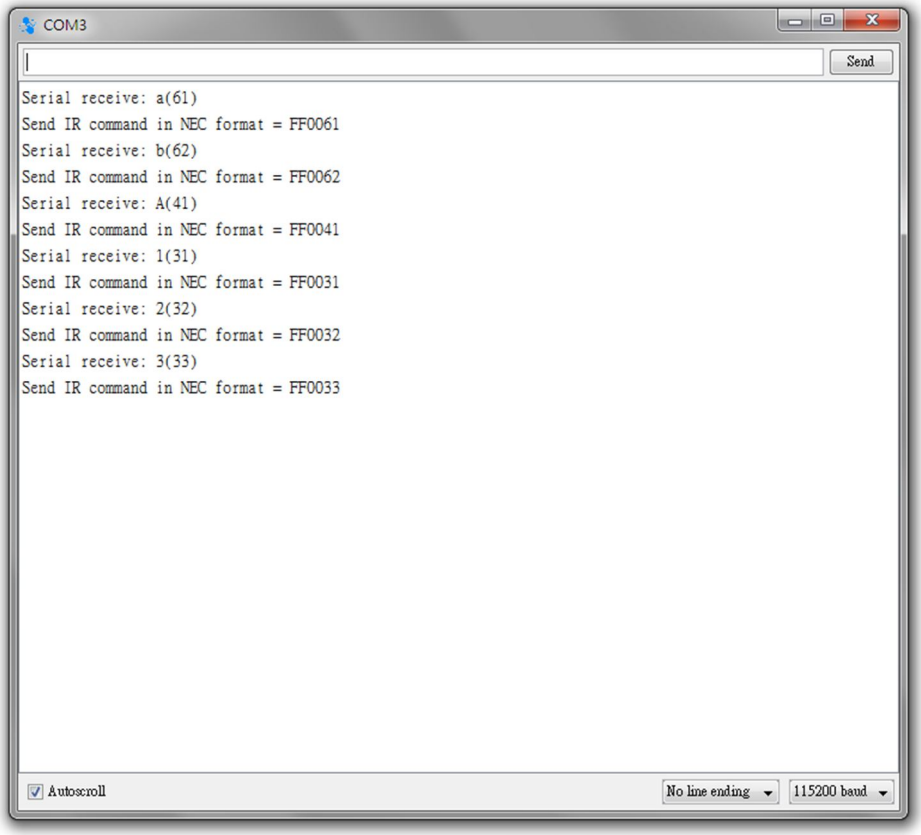


接着请打开 86Duino Coding IDE，输入以下程序代码：

```
#include <IRremote.h>

int ID_send_pin = 10;// IR 接收器输出脚位

IRsend IR_send;// IRremote 函式库传送用对象

void setup( )
{
  Serial.begin( 115200 );

  IR_send.outPin( ID_send_pin );// 注意 86Duino EduCake 预设的 IR 发射器 PWM 脚位为 pin 10
}

void loop( ) {
```

```
if ( Serial.available( ) ) {
        char data = Serial.read( );
        //unsigned long cmd = 0x00FF1234;// Addr = 00FF, Data = 1234
        unsigned long cmd = 0x0;// Addr = 00FF, Data = ?
        unsigned long DeviceAddr = 0x00FF;
        // 使用 bitwise or，将 Serial port 收到的字符编码合进 IR 指令中
        cmd = ( DeviceAddr<<16 ) | (unsigned long)data;
        IR_send.sendNEC( cmd, 32 );// sned NEC code format (command, data bits)

        Serial.print( "Serial receive: " );
        Serial.print( data );// char
        Serial.print( "(" );
        Serial.print( data, HEX );
        Serial.println( ")" );

        Serial.print( "Send IR command in NEC format = " );
        Serial.println( cmd, HEX );
    }
    delay( 100 );
}
```

编译并上传程序后，请打开 Serial Monitor，然后输入一些字符，此程序便会将字符的 ASCII 编码加入红外线通讯协议的指令字段元，并且在 Serial Monitor 显示信息如下图：



此程序在一开始也需引用「IRremote.h」，然后以「IRsend IR_send」宣告一个传送用的对象，接着在 setup( )阶段使用「IR_send.outPin(ID_send_pin)」指定红外线发射器街的脚位。注意这个脚位必须为 EduCake 上面有标示「~」的位置，表示此脚位可使用 PWM 输出。如果没有使用「outPin( )」指定输出脚的话，86Duino 的 IRremote 函式库内是默认为数位脚 10 号。

（在 86Duino Coding IDE 安装路径内的\hardware\86duino\x86\libraries\IRremote\IRremote.h 档案中，有定义 #define TIMER_PWM_PIN 10）。

在 loop( )循环内，使用了「Serial.available( )」侦测用户由 Serial Monitor 传送的字符符号，并存在「char data」变量中。传送的通讯协议

长度，这里使用与前面练习使用的车用 MP3 遥控器一致，为 32 位，「cmd = (DeviceAddr<<16) | (unsigned long)data;」语法用在将地址数据放在左侧 16 个位位置，并将数据放于右侧 16 个位位置，读者练习时也可以将地址、数据换成自己想要的数值试试看，不过须注意数据的位长度喔。

得到欲送出的指令后，使用 IR_send 对象的函式「IR_send.sendNEC(unsign long command, int bits)」来送出 NEC 编码的指令，由于这个练习的指令数据为 32 位，因此 sendNEC 的位长度字段填入 32。若读者自行定义了其他长度的指令，这里也须作相对应的变化。

由于各厂商定义的逻辑字节合不同，指令格式或载波频率也不同，因此除了 sendNEC 以外，IRremote 函式库也提供了其他像是：sendSony、sendRC5、sendRC6、sendDISH、sendSharp、sendPanasonic、sendJVC 等函式，使用方法则跟 sendNEC 相同。

另外，若是读者想定义自己的红外线通讯协议逻辑组合、载波频率，而不想使用任何现成厂商的定义，IRremote 也提供了「sendRaw(unsigned int buf[ ], int len, int khz)」这个函式。

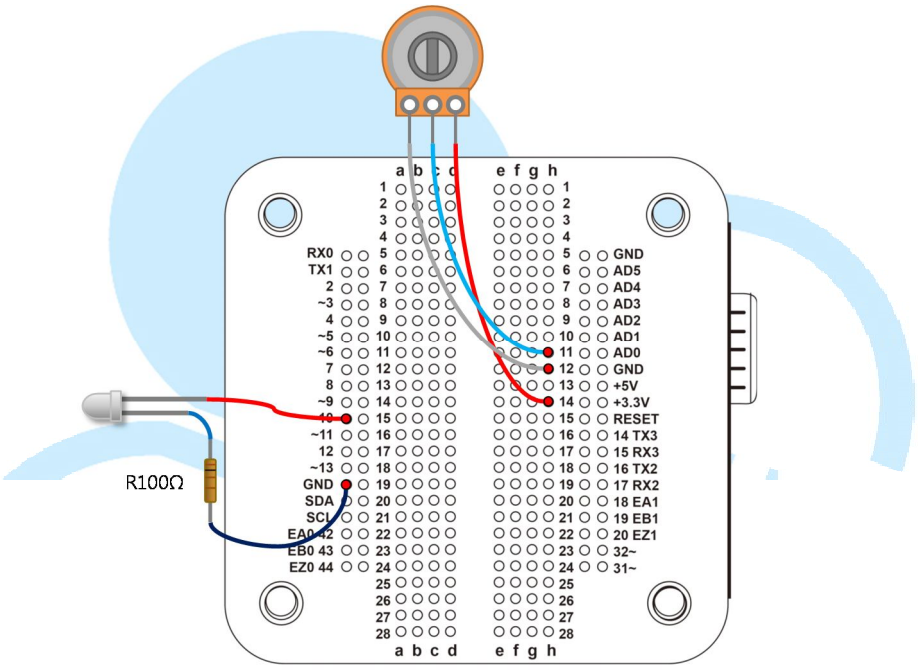
使用时可搭配以下语法：

```
unsigned int cmdBuf[ ] = {
        8900, 4450,// H L
        550, 600, 550, 500,// H L H L
        600, 550, 550, 550,
        550, 550, 600, 500,
        550, 600, 550, 550,
        600, 1650, 550, 1650,
        600, 1650, 550, 1650,
        550, 1700, 550, 1650,
        600, 1650, 550, 1700,
        500, 600, 550, 1650,
        600, 1650, 600, 500,
        500, 1700, 600, 550,
        500, 600, 600, 550,
        550, 1650, 600, 500,
        600, 550, 550, 1650,
        600, 500, 600, 1650,
        600, 1650, 550, 1650,
        600
    };// 67
    int cmdLength = 67;
    IR_send.sendRaw( cmdBuf, cmdLength, 38 );
```

此段程序代码中「cmdBuf[ ]」储存了一系列的 HIGH/LOW 序列时间长度，由 HIGH 开始。「sendRaw(unsigned int buf[ ], int len, int khz)」的参数字段元需要：数据矩阵（单位为μs）、送出的序列长度个数、载波频率（单位为 kHz）等等，这样就可以送出特定长度的序列组合啰。读者也可以尝试这几个 IRsend 内部函数，若有第二台 86Duino EduCake 的话也可以与练习三-1 的程序搭配，看看各种指令格式的差异。

七、 第四个练习 – 两台 86Duino Cake 以红外线通讯

了解 IRremote 的基本用法之后，开始来做点变化，这个练习我们需要两台 86Duino EduCake，其中一台为发射端，一台为接收端。接收端依练习三-2 的图接线；发射端请依下图接线：



接着请打开 86Duino Coding IDE，输入以下传送端的程序代码：

```
#include <IRremote.h>

int ID_send_pin = 10;// IR 接收器输出脚位
int VR_pin = A0;

IRsend IR_send;// IRremote 函式库传送用对象

void setup( )
{
  Serial.begin( 115200 );
```

```
    IR_send.outPin( ID_send_pin );// 注意 86Duino EduCake 预设的 IR 发射器 PWM 脚位为 pin 10
  }

  void loop( ) {

    unsigned int VRvalue = analogRead( VR_pin );// 读取 AD 数值 0~1023
    unsigned long DeviceAddr = 0x00AA;// 接收端、传送端 须配合
    // 使用 bitwise or，将数据合进 IR 指令中
    unsigned long cmd = ( DeviceAddr<<16 ) | ( unsigned long )VRvalue;

    IR_send.sendNEC( cmd, 32 );// sned NEC code format (command, data bits)

    Serial.print( "VR value: " );
    Serial.println( VRvalue, DEC );
    Serial.print( "Send IR command in NEC format = " );
    Serial.println( cmd, HEX );

    delay( 200 );
  }
```

然后接收端使用以下程序代码：

```
#include <IRremote.h>
#include <Servo.h>

int IR_rec_pin = 2;// IR 接收器输出脚位
int servo_pin = 3;// Servo 输出脚位

IRrecv IRrecver( IR_rec_pin );// IRremote 函式库接收用对象
```

```
    decode_results results;// 译码结果存放数据用

    Servo servo_0;// Servo 物件

    void Print_IRdecodeResult( decode_results &decodeResults )// 印出译码成功的讯息以便观察
    {
      // bitwise AND 取得装置地址
      unsigned int DeviceAddr = ( unsigned int )( ( decodeResults.value & 0xFFFF0000 )>>16 );
      // bitwise AND 取得数据字段 0~1023
      unsigned int VRvalue = ( unsigned int )( decodeResults.value & 0x0000FFFF );

      // 要 NEC 编码，且装置地址符合才做对应动作
      if( decodeResults.decode_type == NEC && DeviceAddr == 0x00AA )
      {
        // 从 0~1023 映像到 1000~2000 的数值范围
        int ServoPosition = map( VRvalue, 0, 1023, 1000, 2000 );
        ServoPosition = constrain( ServoPosition, 1100, 1900 );// 限制在安全范围

        servo_0.writeMicroseconds( ServoPosition );// 控制角度

        Serial.print( "IR receive OK, raw data = " );
        Serial.print( VRvalue, DEC );
        Serial.print( "ServoPosition = " );
        Serial.println( ServoPosition, DEC );
      }
    }

    void setup( )
  {
```

```
    Serial.begin( 115200 );
      IRrecver.enableIRIn( );// 初始化接收对象
      servo_0.attach( servo_pin );// 设定 Servo 连接脚
    }

    void loop( ) {
      if ( IRrecver.decode( &results ) )
      {
        Print_IRdecodeResult( results );
        IRrecver.resume( );// 译码完需要呼叫这一行，才能再继续接收后续资料
      }
    }
```

将传送端、接收端分别上传程序后，就可以从传送端转动可变电阻，来控制接收端的 Servo 转动了。传送端程序会定时将模拟数字转换器读到的数据，与装置地址 0x00AA 组合，并传送出去。

接收端流程与练习三-2 类似，但做解碼成功后，利用：

```
    unsigned int DeviceAddr = (unsigned int)((decodeResults.value & 0xFFFF0000 )>>16 )
    unsigned int VRvalue = (unsigned int)(decodeResults.value & 0x0000FFFF)
```

这两个语法，来解出装置地址字段以及数据字段（数字模拟转换器的数据）。接下来增加了额外限制条件：「通讯协议需为 NEC 且解出的装置地址等于接收端设定的地址，才能继续做 Servo 的控制」。这样即使是同类型的通讯协议，也可避免干扰，确定是自己要接收的指令了，其余的数据传送也可以如法炮制。

认识了这个做法，读者也可以搭配前面章节提过的机械手臂，实作红外线无线遥控的版本喔。也可以再动动脑，如果要用 86Duino EduCake 实作一个万用遥控器的话，该怎么做呢？