EduCake 赤外線トランシーバ機能実作

一、　赤外線発射/接收原理と応用紹介

章で 86Duino EduCake を用いることの可能な、UART Serial Port、I^2C 等の通信方式について説明したが、これらは「実態電線接続」による方式通信であり、本篇ではその他に生活の中でよく見られる「無線」通信方式、つまり「赤外線通信」並びに 86Duino EduCake を用いた赤外線発射、接収等機能、について説明したいと思う。実際に手を動かして遊んでみよう。

赤外線は電磁波の中の一つに類別されているので、まず、先に簡単に電磁波のスペクトルについて説明する必要がある。通常人が見ることのできる光の波長範囲は 400nm（紫色）~700nm（赤色）の間にある。波長が短いと紫色

である。範囲は約 10nm~400nm（周波が大きいと紫色である）のものが「赤外線」（或いは赤外光）であり、英語では「Infrared」と称され通常、省略形の「IR」で表示される；紫外線（Ultraviolet）はすなわち「UV」と主に表示される。赤外線が属する電磁波スペクタル範囲中は下記の図 1 の様に示される：

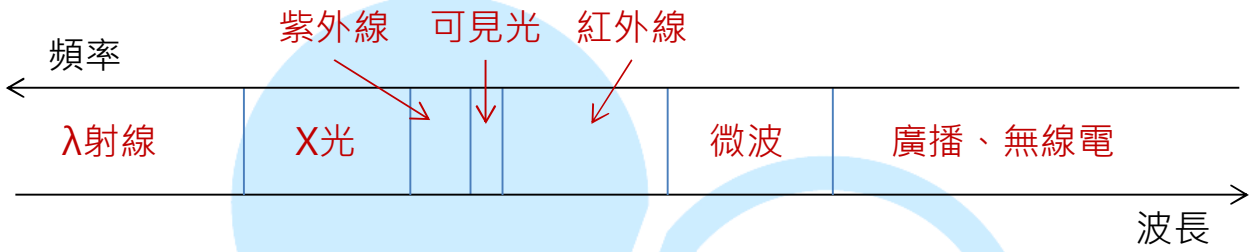


図 1. 電磁波スペクタル簡易表示図

外線のスペクタル内には、測定可能範囲が異なるものをそれぞれ近赤外線、短波赤外線、中波赤外線、長波赤外線、遠赤外線等に分けることが可能であり、もし、読者が更に多くの電磁波通信について知りたいともうならば、参考にしてほしい：

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

赤外線は見ることが出来ないが、生活の中のいたるところで赤外線の存在を感じることが可能である、例えば、ミリタリー映画の中の「赤外線サーモグラフィ」は赤外線センサーが異なる物体が発する赤外線を受信し、異なるカラーを表示することで、人類がモニター上に合わされる温度分布状況を知ることが出来る。

外観温度の他、赤外線のその他の応用は例の如くである：衛星或いは大型望遠鏡が宇宙の星を観察したり、屋内冷気/テレビ/カメラ/CD プレーヤー等電器によくみられる赤外線コントローラー、一部スマートフォンに配備されている赤外線データ送信機能、ポータブルゲーム機、人体動作を感知する等、その用途は大変広い。

本篇文章で紹介するのは、赤外線無線通信という機能を如何に実作するかである。赤外線通信は先に紹介した通信方式同様、発信端と受信端という装置を持つ。実作時、発信端とは主制御回路が赤外線発信端に搭載されており、また受信端主制御電気回路は赤外線受信機が搭載されている。赤外線発信機の外観は LED の様であり、特定の波長の関井外線電磁波を発する。外観は下図 2 の通りである（発信機の中にはダークブルーをしている）

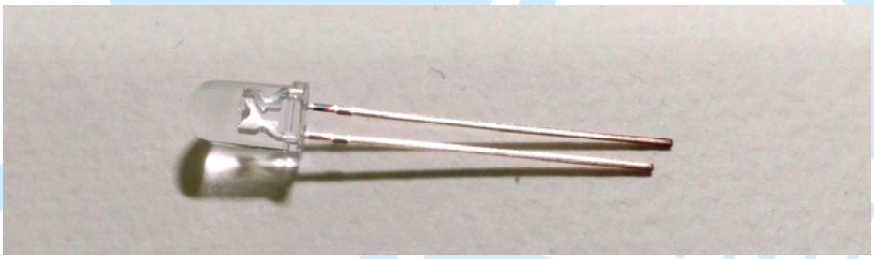


図 2. 赤外線発信機外観

赤外線受信機内部構造は比較的複雑であり、IR 受信機及びシグナル処理電気回路を含む。受信機一つはで 3 ピンから構成され、その中の 2 ピンは電源、アース、その他 1 ピンはシグナル出力用で、外観は下図 3 の通りである。

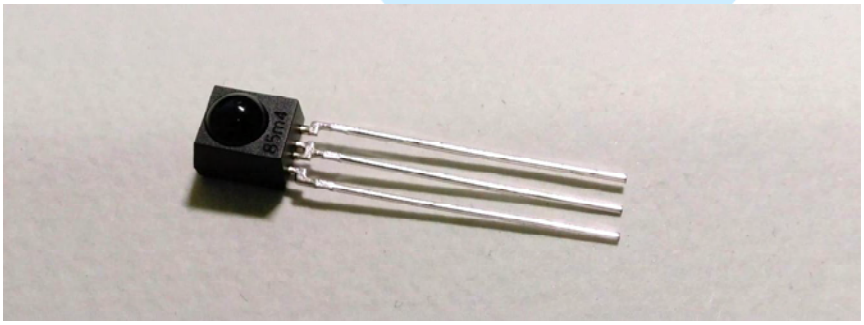
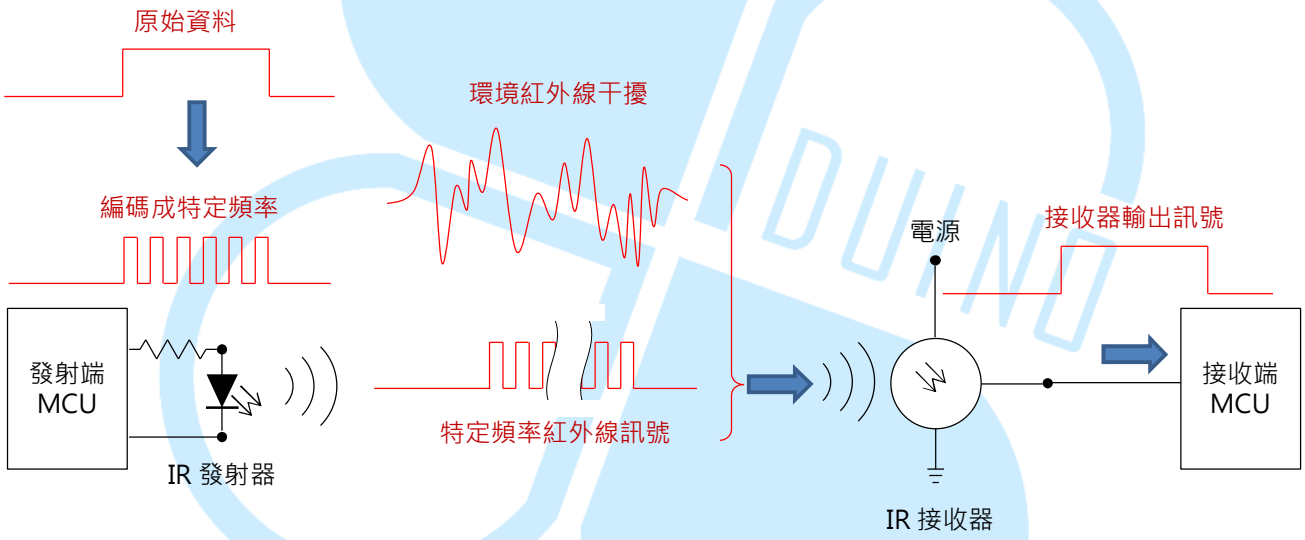


図 3. 赤外線受信機外観図

環境の中に溢れている各種様々なものからくる赤外線電磁波は、受信端が正確なデータを受信することを確保する為に、一般に送受端はデータ「モデュレーション（modulation）」を特定周波数の赤外線シグナルに変換する。そして受信端が赤外線シグナル主神後、シグナルは電気回路がフィルタリングをする処理を行い（わずかに特定周波数範囲のシグナルを留める）、フィルタリング後のシグナルが「デモデュレーション（demodulation）」を行い、HIGH 或いは LOW シグナルとして出力し、HIGH/LOW 定義は受信機により決定されるが、これは受信端に搭載されている装置或いは処理機が送信端が送信する資料が何かを知る為である。赤外線出力フローチャートの簡易図は図 4 の通り



である。

図 4. 赤外線フローチャート簡易図

上述した特定の赤外線を送信/受信する方式は、二つの同じ周波帯が必要で、一般的に 36 kHz、38 kHz、40 kHz、56 kHz 等がよく見られる。

注意しなくてはならないのは、ここで上げた「特定周波数」は必ずしも電磁波スペクトルを指すわけではなく、赤外線発射器が特定間隔時間において発する赤外線周波数を「搬送波周波数」と呼ぶ。モデュレーション（modulation）とデモデュレーション（demodulation）はデータと搬送波周波数を結合し、

搬送波周波数の中から鳥です過程において、無線出力方式をよく用いる。その他、赤外線発射器は一般の LED よりも大きな電流を必要とする為、搬送波周波数は赤外線シグナルを発する際、通電が有利となっている。

搬送波周波数はデータをコーディングする以外に、異なる電気用品、ひいては異なるメーカーの間で、自身の赤外線「通信協定（protocol）」を持っている。

上述の赤外線シグナルは転送前、受信後はみな数字シグナルとなり、その為通信シグナル協定は一連の HIGH、LOW 電圧から構成され、各メーカーのシグナル定義が異なり、そして異なる長さ等に依拠する。その為、メーカーA のコントローラーとメーカーB の電器の間には、同じ波長の赤外線と搬送波周波数を選ぶ必要があるが、必ずしもその制御が上手くいくとは限らない。

SONY コントローラーを例にとると、使用されている搬送波周波数は 40 kHz であり、コントローラーを押した後、赤外線発射器は異なるビット列を出力し、対応の IR 制御例を表示し、受信した電器は異なる指令に依拠して対応する動作を行う、例えば、テレビは電源のオン・オフ。音量調整、番組選択等である。下図 5 が示した SONY コントローラが発する赤外線指令により、読者は指令に含まれるシグナルとデータシグナルを見ることが出来その中の 0、1 は異なる長さの HIGH/LOW コードに分けられる、その他、ボタンを押したままにする時、通信協定により、一定間隔で重複して特定数量の同様の指令を発するが、あるものは持続的に重複した指令を出すものもある。その為、あるメーカーの電気装置を制御したいと思うのなら、まず先に赤外線転送規格が何なのかを知る必要がある。

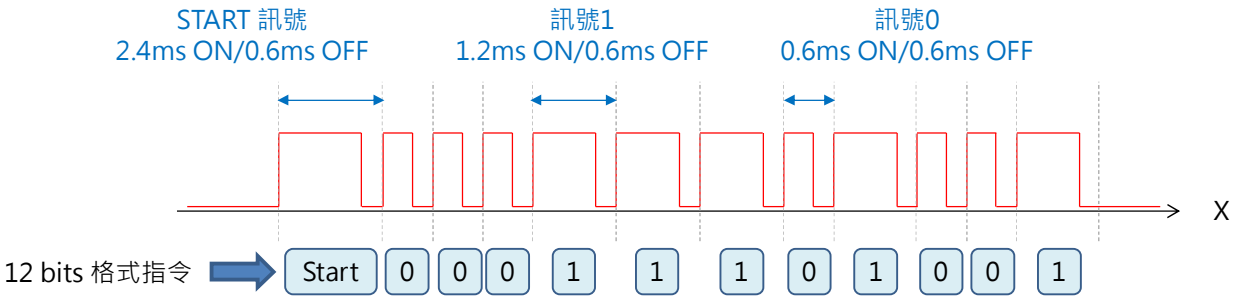


図 5. 赤外線通信シグナル協定例

SONY 赤外線通信協定、参考までに：

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

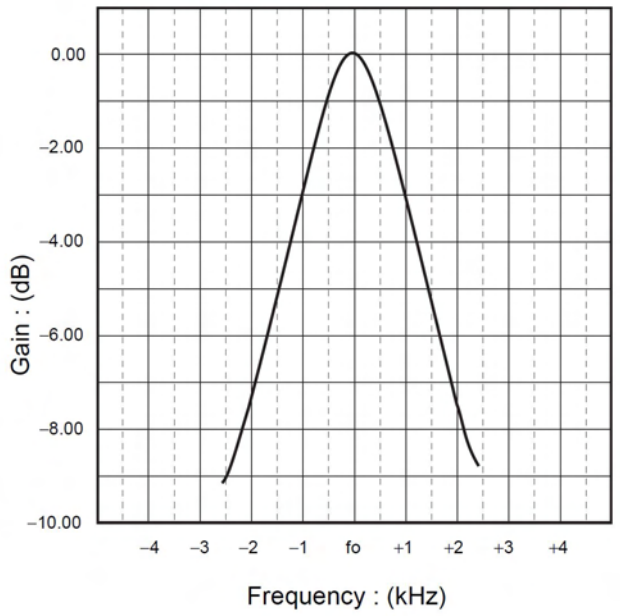
http://www.sbprojects.com/knowledge/ir/sirc.php

続いて 86Duino EduCake 実作上において、私たちは一般に入手可能な 920nm 赤外線発射器を選び、また、筆者は材料店で RPM6938 赤外線受信機を見つけたので、これを用いて実作を行おうと思う。この受信機のピン定義は下図 6 の通りである：

針腳定義，依序為：
[訊號　GND　電源]

図 6.赤外線受信機ピン定義

RPM6938 の企画表に依拠し、電源ピンは 5V を使用し、受信したシグナルの搬送波周波数は 37.8kHz であり、レベルは受信角度約 70 度、垂直約 60 度程である：シグナル未受信時のピンは HIGH、そうでない時は LOW である。注意してほしいのは、規格表情に表示された搬送波周波数が 37.8kHz であっても、実際はこの周波数は「裁量の受信効果」であり、例えば規格表上の図に示されたように、周波数がやや高かったり低かったりした場合、その受信効果は減少してしまうため、周波数に若干の誤差が有っても、問題はない。

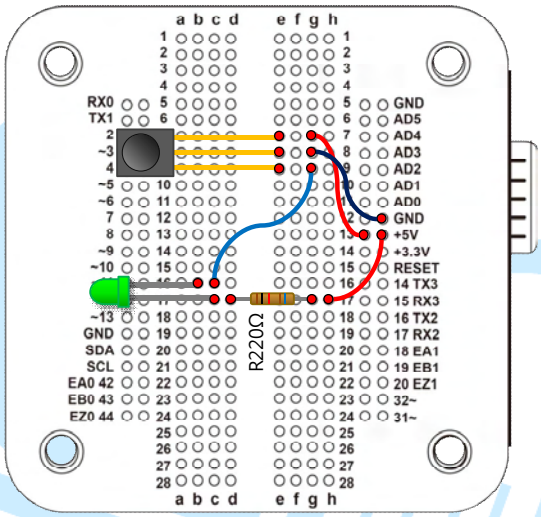


受信機の種類は大変多いため、もし、読者が購入される時に別の型式の赤外線受信機を購入したとしたら、搬送波周波数以外に、受信機の方をよく見る必要がある。その為、一部の受信機の電源は異なる可能性が有る為、規格をよく見て配線する必要がある。その他、2 本のピンは、見た目は青色 LED の赤外線受信機に似ているが、この種の赤外線受信機には搬送波周波数を処理する能力がないので、間違えて購入しないよう、注意する必要がある。

赤外線発射器の配線方式は通常の LED と同様であり、長いピンがプラス、短い方がマイナスである。続いて、私たちは赤外線転送の機能を実作して練習してみよう。

二、　第一の練習 – 赤外線受信機テスト

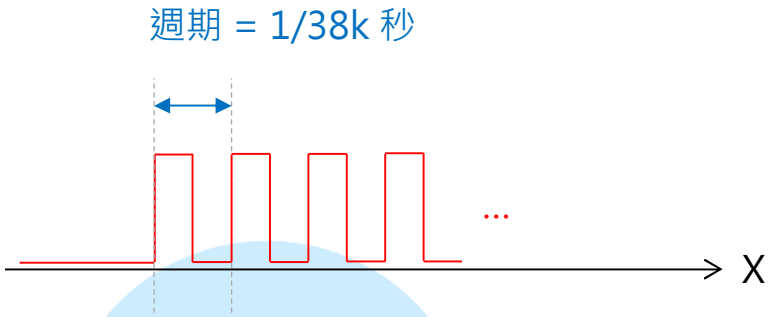
第一の練習において、私たちは暫時 EduCake のプログラムを使わず、まず、簡単な方式で赤外線受信機と発射器が正常かどうか試してみる必要がある。読者はまず、下図の配線をご覧いただきたい：



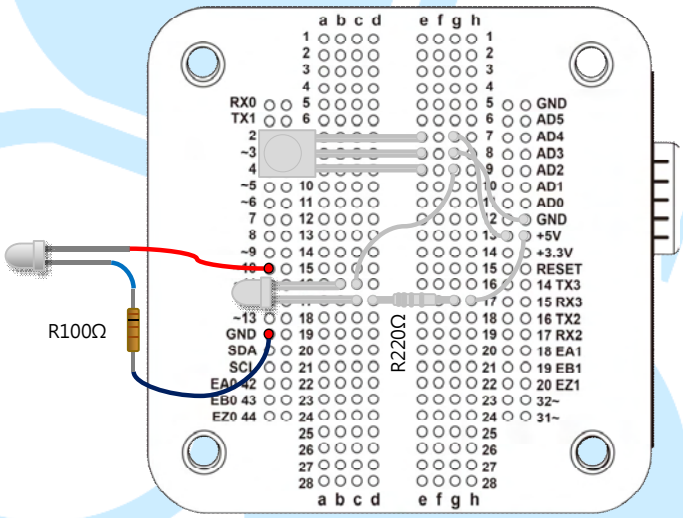
赤外線受信機の電源は 5V、ピンは GND、出力シグナルピンは LED のマイナス、LED プラスは 220 オームの電圧となっている。この赤外線受信機がシグナルを受信していない時、出力ピンは HIGH であり、その為、LED は平常、光ることはない。続いて、読者は傍らの赤外線コントローラーを使用して、受信機に向かって任意のボタンを押し、もし、コントローラーが使用する搬送波周波数は大体 38kHz 程であり、LED ランプは断続的に光り、手の上の赤外線受信機が正常に作動することを示す。

もし傍らに赤外線コントローラがなく、ただ先に挙げた赤外線発射器（見た目が LED に似ているもの）の部品しかないとしたら、テストはどうしたらよいのだろう?このような時は、プログラミングを使い、赤外線発射器の点滅周波数を制御すればよい。

ここで選択された受信機の約 38kHz の搬送波周波数は反応を示す。その為、使用するコントローラーは下図の様な波長である必要がある：



この様な波形を生み出すには、第一の方法では digitalWrite( )を使用し発生することが出来るので、読者は下図に則って配線してほしい：



赤外線発射器プラス端を pin 10 に接続すると、発射器から流れ出す電流は LED よりも大きいものとなる。その為、マイナス端の 100Ω レジスタンスを GND に接続し（LED に通常搭載されているのは 220Ω レジスタンスである）、赤外線受信機の電気回路図を不動のものとする。

続いて、86Duino Coding IDE を開いて、以下のプログラミングを入力する：

```
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 );
}
```

このプログラミングは「void SendIR( )」関数において、「digitalWrite( )」が「delayMicroseconds( )」を搭載して使用され、PWM 波形を生産し、38kHz 波形の周期時間は約 26us 程である。その為、LOW/HIGH は各 13us を使用する。コードは 86Duino EduCake アップローダー後であり、赤外線発射機が毎

秒発する周波数は 38.4kHz の波長であり、赤外線受信機が出力するピンが受信する LED のきらめきを観察することが可能である。

第一種は簡単なテスト方式であり、もし、ハードを使用し実作した PWM 波形を使用するのであれば、周波数制御は更に正確なものとなる。86Duino EduCake が提供する優れた関数ライブラリは「TimerOne」と言い、このライブラリを使用し、86Duino EduCake 内の CPU の 32 ビット関数機能を設定することが可能であり、並びに、特定のピンが出力する PWM シグナルを制御することが可能である。ハードウェアは規則正しく電圧 HIGH/LOW に接触する為、時間制御上プログラミングがピンビッド電圧切り替えを制御するよりもより正確である。機能と先のテストプログラミング同様のプログラミングは下記の通り：

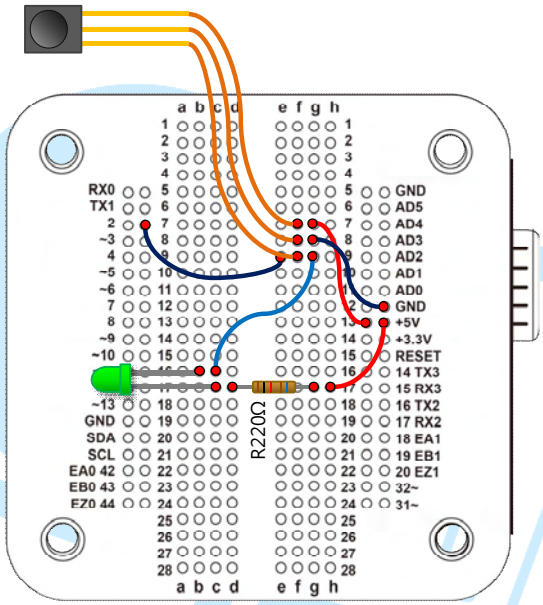
```
#include "TimerOne.h"
int IR_pin = 10;
void SendIR( )
{
  Timer1.pwm( IR_pin, 512, 26 );// pin, duty (512=50%), period(us)
  delay( 20 );
  Timer1.disablePwm( IR_pin );
  delay( 20 );
  Serial.println( "IR send." );
}
void setup( ) {
  Serial.begin( 115200 );
  pinMode( IR_pin, OUTPUT );
  Timer1.initialize(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

TimerOne ライブラリを使用したいと思うなら、まず使用言語「#include "TimerOne.h" 」をプログラミングし、続いて setup( )において「Timer1.initialize(period);」を使用し TimerOne 対照を初期化し、その中のパラメータ period をタイマー周期とし、単位は microsecond とする。

「void SendIR( )」ライブラリにおいて、「Timer1.pwm(IR_pin, 512, 26);」言語を使用して IR ピンが PWM 機能を起動する設定を行い、512 は 50%の duty 、 period は 26 μs 、持続時間は 20ms である。「Timer1.disablePwm(IR_pin);」言語は赤外線発射器ピンの PWM 出力切断に用いられる。赤外線発射器、受信機テストの目的に用いることも可能である。

三、　第二の練習 – 赤外線通信フォーマット認識

赤外線受信/発射器の基本原理をよく理解したら、続いて実用段階へと入ろう。この練習で私たちは実際の電気用品転送の「通信協定」がどのようなものか知ることとなる。読者は、下図に基づき配線してほしい：



この電気回路図は先の配線を保存するが、別の赤外線受信機が出力するピンは元来 86Duino EduCake デジタルピンビット 2 に接続され、プログラミングは電圧状態を読み取る。配線完成後、86Duino Coding IDE を開き、下記のプログラミングを入力しよう：

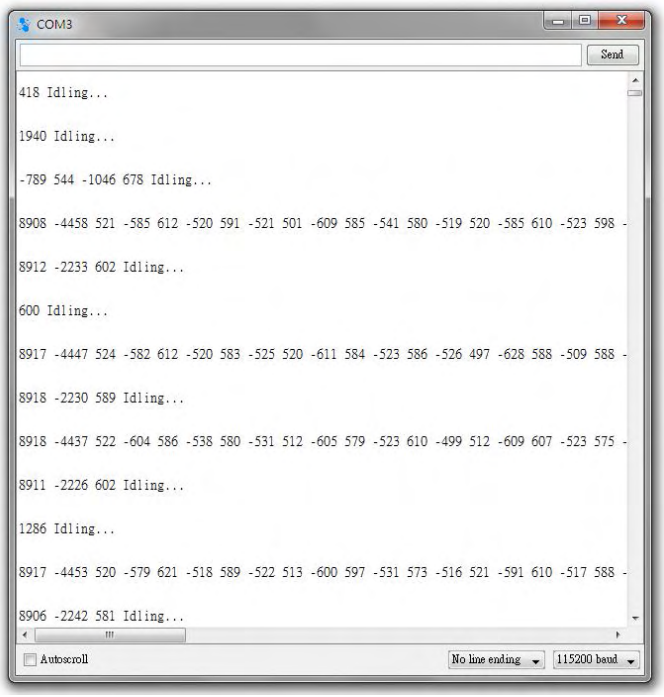
```
int IR_rec_pin = 2;// IR 受信機出力ピン
int IRstate = LOW;// IR 受信機出力ピンビット状態
int IRstate_last = LOW;// IR 受信機出力ピンビット状態
long int time_last = 0;// 先の IRstate 変化時間記録

boolean isIdle = true;// IR 通信モジュール機状態か否か
const long int durationMax = 10000;//一定時間変化が無ければ IR シグナル状態に入り、単位は us とする
```

```
    const long int durationMin = 400;// 電圧単位不変の最小持続時間，単位は us とする
    void IR_rec_Check( )
    {
        IRstate = digitalRead( IR_rec_pin );// ピンビット状態読み取り

        if( IRstate != IRstate_last ){// 今回と先のピンビット状態は異なる

          long int timeNow = micros( );// 目下時間読み取り
          long int dT = timeNow - time_last;// 先のピンビット変化経過の時間

          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( " " );
          }

          // この時間記録
          time_last = timeNow;
        }
        // この状態記録
        IRstate_last = IRstate;
    }
```

: }

```
void setup( ) {
  Serial.begin( 115200 );
  pinMode( IR_rec_pin, INPUT );//  ピン I/O モジュール記録
  IRstate = digitalRead( IR_rec_pin );//  ピンビット状態初期値記
録
  IRstate_last = IRstate;

void loop( ) {

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

コード並びにアップロードをしたら、Serial Monitor を開き、傍らのこの赤外線受信機のコントローラーを使用することが出来（まず、先に練習用に提供した方法を使用し、搬送波周波数が符合しているかテストする）、受信機のボタンを押してみて、Serial Monitor 上に一続きのデジタルが表示され、正の数が出力ビット電圧 HIGH の持続時間を、負の数が電圧 LOW の持続時間を、下図の様に表しているのを確認する：



ここで注意するのは、赤外線発射器が「発射」状態にあるとき、受信機が受信する赤外線は出力 LOW であり、その反対の場合は HIGH である。その為、ここで読み取った正の値は実際の所、受信されて「いない」のである。

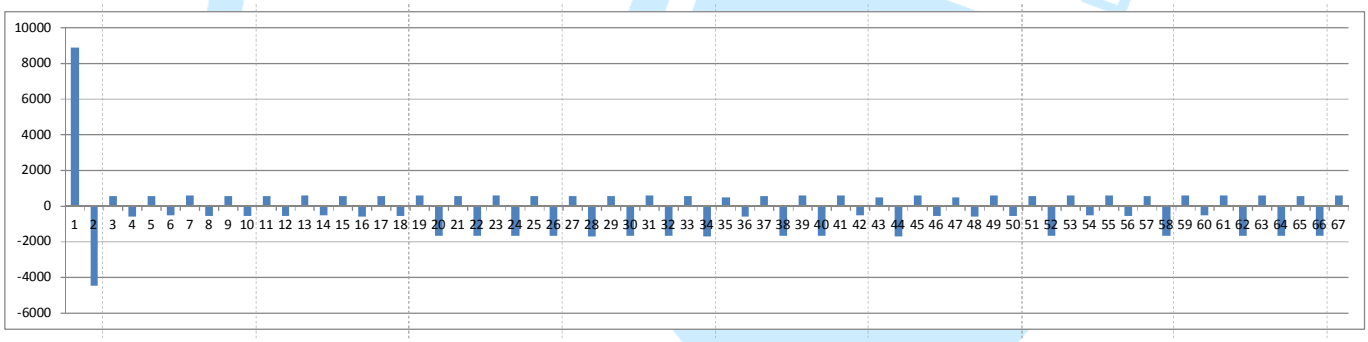
このプログラミングは毎 20μs ごとに、「IR_rec_Check( )」関数を呼び出し、「digitalRead( )」を使用して出力ピンビットの電圧状態を読み取り、電圧に変化があった際に、「micros( )」言語を用いてその時の秒を記録し、次の状態変化時間と持続時間を測る。

このプログラミング内には durationMax、durationMin が設定されており、フィルター条件となっている。大は durationMax で一定の時間内に受信が得られないと、idle 上位対へと突入する；反対に小は durationMin でノイズを考慮しない。Ms 作為間隔があるのは、赤外線受信機は数百μs（規格による）間持続する赤外線シグナルを判断し HIGH 或いは LOW とするからである。低いものはノイズを無視し、その為周波数の間隔はやや長く。持続時間は正確ではない。

読者に注意してほしのは、コントローラーを押していない時、受信機は環境赤外線の影響を受け、持続時間の長さが不均等な HIGH/LOW シグナルを出力するため、実際の伝送を正確なものとする為、「通信協定」を明確に説明する必要がある。「通信協定」は伝送/受信両端があらかじめ定義した一連の信号順序定義のことであり、シグナル開始、終了、データ長さ等を含み、先に挙げた Serial 伝送概念と類似している。

筆者はここでテスト使用の車用 MP3 コントローラーを使用するが、その外観は下記の如くである；

このコントローラを例にとり、毎度ボタン 0 を押す度のその持続時間は約「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」のデータ系列とし，時間単位をμs とするが，これでは観察しづらいため、，下記の様にエクセルを使って表すこととする：



初めから最初までちょうど 64 個のデータが残っているが、もしその他のボタンを押すとその他のデータ系列を得ることが出来る。図上で見ることが出来る、このコントローラーが使用する HIGH 時間は約 600μs、LOW 時間は約 600μs 或いは 1600μs を組合せたデータビットに当る。外部からの干渉を容易にうけず、ちょうどこのコーディングに符合しているため、その他のメーカーの赤外線コード方式とは異なる可能性がある。その為、伝送/受信両端は特

定のデータ系列の時、外界の影響を受けその能力が低下する可能性が有る。読者は、手元の各種コントローラーで様々な組み合わせを楽しんでみてほしい。

四、　第三の練習-1 – IRremote ライブラリ使用紹介(受信)

五、 先の章で、赤外線通信協定の多様さを練習したが、実際の応用はいかがだっただろうか？

六、 このライブラリは通信協定を用いて赤外線データの転送と受信を行い、並びに NEC、Sony SIRC、Philips RC5、Philips RC6、Sharp、Panasonic、JVC、Sanyo、Mitsubishi の協議コーディングをサポートし、元データを出力することが可能である。この練習は私たちに IRremote ライブラリを使用し赤外線シグナルを受信し、実用的な重要機能を紹介している。

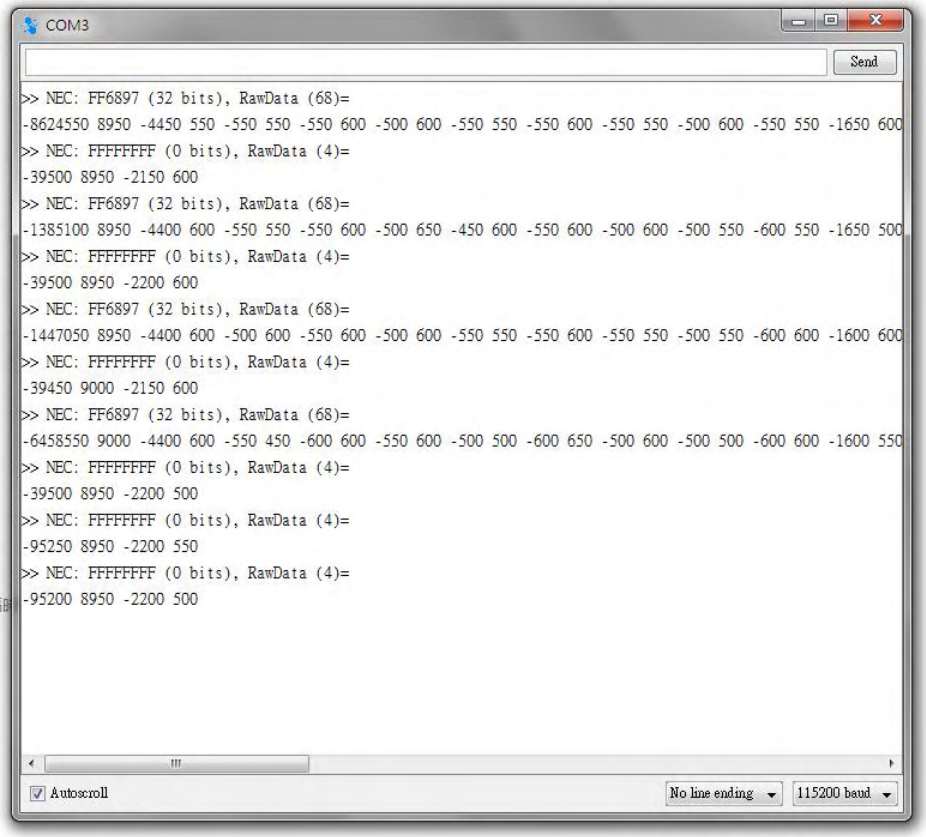
七、 86Duino EduCake 電気回路維持と練習 2 の様な配線方法に続いて、読者は 86Duino Coding IDE を開き（104 バージョン以上でないと IRremote ライブラリは存在しない）、以下のプログラミングを入力する：

```
#include <IRremote.h>
int IR_rec_pin = 2;// IR 出力ピン受信

IRrecv IRrecver( IR_rec_pin );// IRremote ライブラリ対象受信

decode_results results;// プログラミング解読結果をデータ用入れる


void Print_IRdecodeResult( decode_results &decodeResults )// データ解読成功結果を観察
{
  int dataLength = decodeResults.rawlen;

  switch( decodeResults.decode_type )
  {
```

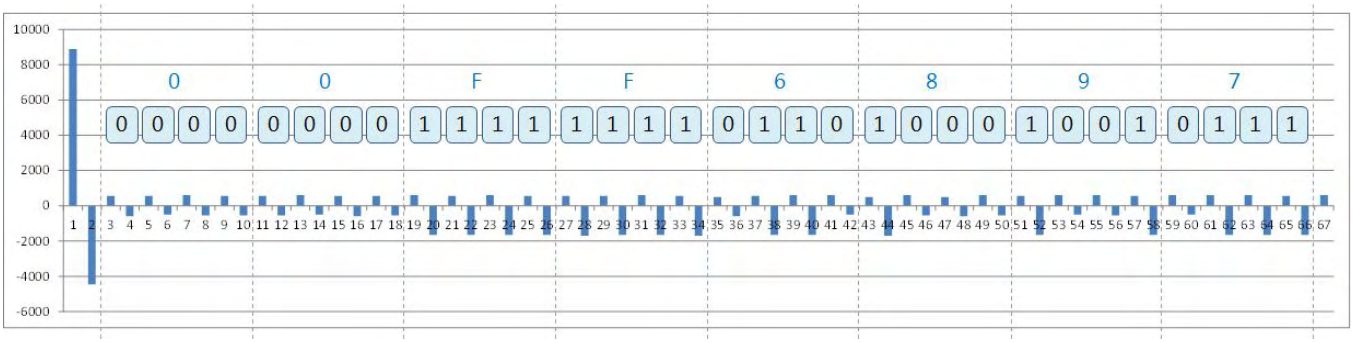
```
            Serial.print( ">> NEC:\t" );
                break;

            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 );//通信協定使用欄、，
16 ビット表示
        Serial.print( " (" );
        Serial.print( decodeResults.bits, DEC );//
    元資料総受信    Serial.print( " bits), " );

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

        // 元データ系列出力
        for ( int i = 0; i < dataLength; i++) {
            int data = decodeResults.rawbuf[i] * USECPERTICK;
            // buf 読み取り個数を入れ、毎読み取り間隔を USECPERTICK
    とする
            if ( (i % 2) == 1 ) {// 偶数を HIGH とする
                Serial.print( data, DEC );// HIGH
            }
```

コーディング並びにアップデート後、Serial Monitor を開き、続いてコントローラーを赤外線受信機に向け、筆者が先に練習した車用 MP3 を使用するのは言うまでもなく、下図のデータが出現した：



このコントローラーが 0 を押したとき、IRremote ライブラリは復号を開始し、データを受信する。筆者は使用したコントローラーを例にとり、IRremote 関数復号が可能な通信協定を NEC のコーディングを使用し、基本的な原データ時間系列の長さを練習 2 同様、下図の様な Excel を利用し行おう



と思う：

NEC 通信協定の受信機出力ピンビット電圧とはつまり：

λ　開始シグナル：約「9ms の HIGH 及び 4.5ms の LOW」組み合。わせ

λ　データ論理 0：約「560μs の HIGH 及び 560μs の LOW」の組み合わせ。

λ　データ論理 1：約「560μの HIGH 及び 1690μs の LOW」の組み合わせ。

λ　重複指令：約「9ms の HIGH 及び 2.25ms の LOW と 560μs の HIGH」の組み合わせ。

このコントローラーボタン 0 が発するデータは 0X00FF6897 のコードに相当し、64 個の HIGH/LOW がちょうど 32 個の理論ビットであり、合計 4 個の byte の中の 16 個のビットでコーディングは終了し、後に 16 個のビットが指令を行う。

その他、もしあるボタンを持続したいと思うなら、第一のデータコード受信後、続いて「重複指令」FFFFFFFF を、約毎 110ms ごとに重複する必要が出てくる。例えば、音量+ボタンを押し続けたいと思うのなら、重複指令に依拠し増加音量を持続することが可能である。その為、NEC 装置の様なものも、ビットを解析することで、通信協定の面白さを味わうことが出来る。

IRremote ライブラリを使用し、このプログラミングは「IRremote.h」の引用を開始し、その後「IRrecv IRrecver(IR_rec_pin);」言語を使用し赤外線受信機に対し宣告をし、伝送されたパラメーターを受信出力ピンの番号とする。「decode_results results」は class データであり、内部に入れられている：

λ　decode_type：標示通訊協定編碼類型。

λ　panasonicAddress：Panasonic 協議専用的位址欄位。

λ　value：通信協定資料欄数値。

λ bits：デジタル系列総ビット数。

λ unsigned int *rawbuf：元 HIGH/LOW 読み取り資料。

λ rawlen：読み取りデータ個数。

λ setup( )段階では「IRrecver.enableIRIn( )」を使用し赤外線受信機の初期化を執行する必要が有り、続いて loop( ) 内で「IRrecver.decode(&results)」を呼び出し、この関数は目下受信する赤外線データをスキャンし、並びにコードを検査し、もし解読に成功すれば、解読後の結果を results 変数に入れ、true へポストバックする；反対に不成功或いはデータがまだ未受信なら false をポストバックする。True ポストバック時、「Print_IRdecodeResult( )」関数を使用し。成功した各種情報を印刷し観察することができる。

λ 練習 2 と比べると、これは読者が更に手元の各種コントローラーがどのような通信協定やボタン入力後どのような指令データが出るのか見ることが可能である。

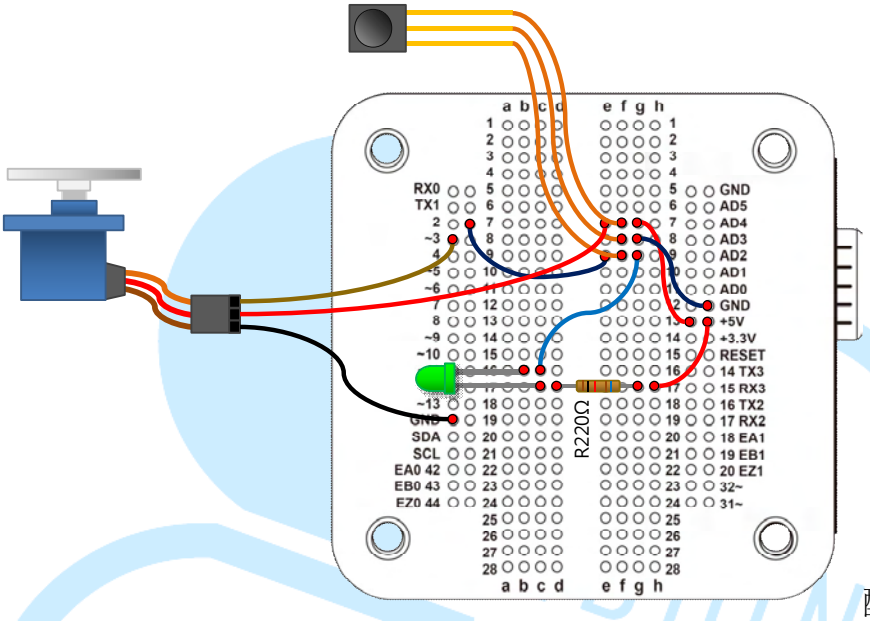
λ NEC 赤外線通信協定については、参考してほしい：

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

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

八、　第三の練習-2 – IRremote ライブラリ使用紹介(受信)

この練習は先のものわずかに複雑にし、手元のコントローラを利用して EduCake を「起動」する。読者は下図に基づき配線してほしい：



配線後、

86Duino Coding IDE に以下のプログラムを入力する：

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

int IR_rec_pin = 2;// IR 受信機ピン位置出力
int servo_pin = 3;// Servo  ピン出力

IRrecv IRrecver(IR_rec_pin);// IRremote  ライブラリ対象受信
decode_results results;//解読データ保存

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

```
    DIR_RIGHT
  } ServoDir;//  方向定義交換
  int servoDir = DIR_NONE;// Servo  方向転換
  unsigned int ServoPosition = 1500;// Serco  角度

  void Print_IRdecodeResult( decode_results &decodeResults )//
解読成功データを出力し観察
  {
    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; }
```

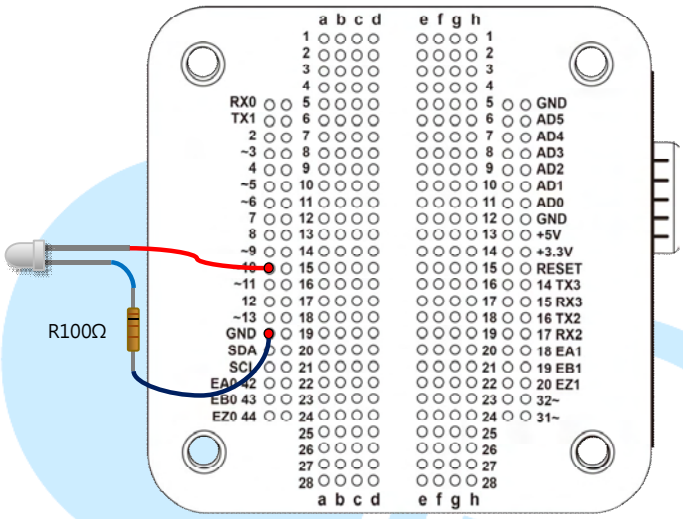
```
        ServoPosition = constrain( ServoPosition, 1100, 1900 );// 安全
範囲制限
                servo_0.writeMicroseconds( ServoPosition );// 角度制
御

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

            default:
                break;
        }
    }
}


void setup( )
{
    Serial.begin( 115200 );
    IRrecver.enableIRIn( );// 初期化対象受信
    servo_0.attach( servo_pin );// Servo 設定 ピン連結
}

void loop( ) {
    if ( IRrecver.decode( &results ) )
    {
        Print_IRdecodeResult( results );
        IRrecver.resume( );//解読後この行を呼びだし、再継続受信後
データ持続
    }
}
```

コード並びにアップロード後、コントロールボタンを用いて EduCake 連接の RC Servo を連接する。全てのプログラムフローと先の練習は同様であり、

主に「Print_IRdecodeResult( )」関数内で改変行う。筆者が使用するコントローラーにより、「CH-」ボタンが体操する赤外線協定の数値は 0x00FFA25D であり、「CH」ボタンは 0x00FF629D、「CH+」は 0x00FFE21D であり、押した状態持続コードは 0xFFFFFFFF なので、switch-case 言語はここではこの数値を使用し、必ず NEC コードは対応する Servo 制御動作を行い、NEC コードが対応する Servo 制御動作を行い、誤受信の影響を低下させる。もし読者がその他のコントローラを使用いたのなら、通信協定が異なるので、ボタン対応数値が異なり、これらの場所を修正し正常動作を行う必要がある。

第三の練習-3 – IRremote ライブラリ使用紹介(発射)

Irremote ライブラリの受信機能実作後、続いてこの実作発射赤外線指令機能の実作を行う。下図の電気回路図に基づいて配線してほしい

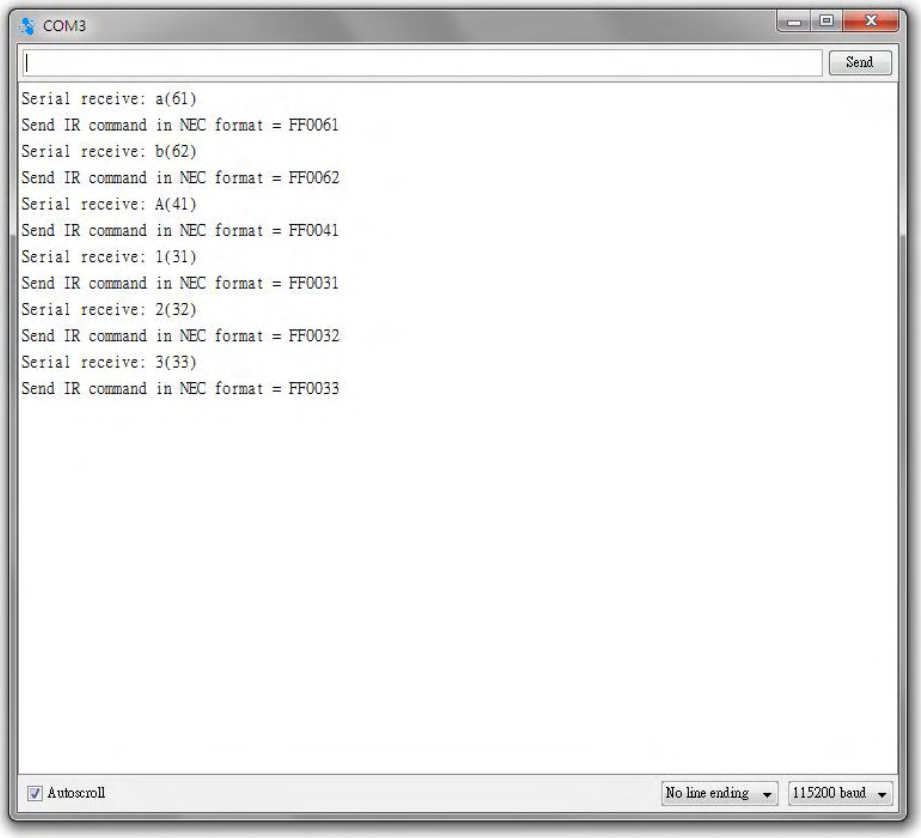



```
#include <IRremote.h>

int ID_send_pin = 10;// IR 受信機ピン出力

IRsend IR_send;// IRremote ライブラリ転送用物件

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

IR_send.outPin( ID_send_pin );// 86Duino EduCake があらかじめ設定
した IR 発射器 PWM ピンを pin 10 とすることに注意する
}

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;
  // bitwise or を使用し、Serial port を文字エンコード指令に入れる
            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 );
        }
```

続いて 86Duino Coding IDE を開き、以下のプログラムを入力する：

コード及びにアップデート後、文字を入力し、このプログラムが文字を ASCII コードを赤外線通信協定の指令ビット欄に加えると、続いて Serial Monitor に下図の様なデータが表示される：



このプログラミングは開始時やはり「IRremote.h」を引用し、その後「IRsend IR_send」を持って転送用の物件を宣告し、続いて setup( )段階で「IR_send.outPin(ID_send_pin)」を使用して赤外線発射器のピン位置を指定する。このピンが必ず EduCake 上に表示される「~」の位置に注意すると、ピンが PWM 出力を使用することが可能となる。もし「outPin( )」を使用し出力ピンを使用しないと、86Duino の IRremote ライブラリ内が黙認するデジタルピンは 10 番である。

（86Duino Coding IDE において \hardware\86duino\x86\libraries\IRremote\IRremote.h 設定後、定義は #define TIMER_PWM_PIN 10 である。）

loop( )ループ内において、「Serial.available( )」を使用して使用者が Serial Monitor か、転送した字記号をテストし、並びに「char data」変数を保存する。転送した通信協定の長さは、ここで使用したものと先の先の練習で使用した車用 MP3 コントローラーと一致し、32 位とし、「cmd = (DeviceAddr<<16) | (unsigned long)data;」言語を用いてデータを引っ張りだし、左側の 16 個の位置に置き、続いてデータを右側 16 個の位置に置く。読者は練習時、データ自身が望んだ数値を試してみるのも良いが、データの字数の長さに注意する必要がある。

得到欲送出的指令後，使用 IR_send 物件的函式「IR_send.sendNEC(unsign long command, int bits)」から NEC コーディング指令を送り、この練習指令データを 32 ビットとする。その為、sendNEC の長さ欄に 32 と記入する。もし読者が自らその他の指令を定義する場合は、ここで相対する変化を行う必要がある。

等函式，使用方法則跟 sendNEC 相同。

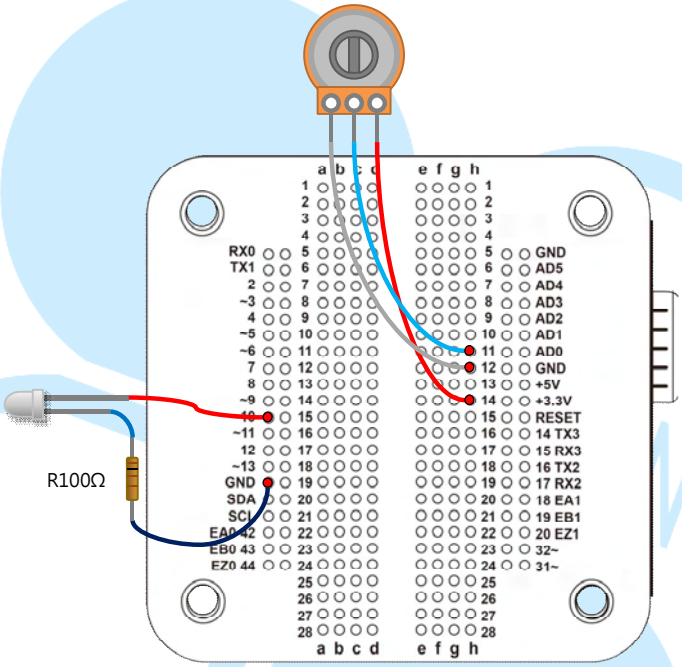
各メーカーが定義したロジックビットの組み合わせは異なり、フォーマットやキャリア周波数も異なるため、sendNEC を除いて、IRremote ライブラリもその他の例えば sendSony、sendRC5、sendRC6、sendDISH、sendSharp、sendPanasonic、sendJVC の様な関数を提供し、使用方法は sendNEC と同様である。

```
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 );
```

「sendRaw(unsigned int buf[ ], int len, int khz)」のパラメータ差異はとても重要である：注意して入力すること。

九、　第四の練習 –2 つの 86Duino Cake 赤外線通信を行う

十、 Irremote の基本用法を理解した後、変化を作るのを開始するが、この練習で私たちは 2 台の 86Duino EduCake を用いて その中の一台を発射端、もう一台を受信端とする。受信端は練習 3-2 の図に基づき配線する；発射端は下図に基づき配線する：



続いて 86Duino Coding IDE を開き、以下の転送端のプログラミングを入力する：

```
#include <IRremote.h>

int ID_send_pin = 10;// IR 受信機出力ピン

int VR_pin = A0;

IRsend IR_send;// IRremote  ライブラリ転送用物件

void setup( )
```

```
        IR_send.outPin( ID_send_pin );// 86Duino EduCake があらかじめ設けた IR 発射器 PWM ピンを pin 10 とすることに注意}

    void loop( ) {

        unsigned int VRvalue = analogRead( VR_pin );// AD 数値 0~1023 読み取り
        unsigned long DeviceAddr = 0x00AA;// 受信端、転送端結合
// bitwise or を使用しデータを IR 指令に入れる

        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 );
    }
```

その後受信端は以下のプログラミングを使用する：

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

int IR_rec_pin = 2;// IR 受信機出力ピン
```

```
int servo_pin = 3;// Servo 輸出ピン


IRrecv IRrecver( IR_rec_pin );// IRremote  ライブラリ対象受信

decode_results results;//  コード解読結果データ用保存


        Servo servo_0;// Servo  物件


void Print_IRdecodeResult( decode_results &decodeResults )//  コード解読結果成功のデータチェック

        {
  // bitwise AND 取得装置位置
        unsigned int DeviceAddr = ( unsigned int )( ( decodeResults.value & 0xFFFF0000 )>>16 );
  // bitwise AND  データ欄取得
        0~1023
        unsigned int VRvalue = ( unsigned int )( decodeResults.value & 0x0000FFFF );


  // NEC をコーディングし、装置符合した後動作を対応する
        if( decodeResults.decode_type == NEC && DeviceAddr == 0x00AA )
        {
  // 0~1023 から 1000~2000 の数値範囲
          int ServoPosition = map( VRvalue, 0, 1023, 1000, 2000 );

        }

{
```

```
  ServoPosition = constrain( ServoPosition, 1100, 1900 );// 安全範囲制
限

                servo_0.
                Serial.print( "IR receive OK, raw data = " );
                Serial.print( VRvalue, DEC );
writeMicroseconds( ServoPosition );// 角度制御
            Serial.print( "ServoPosition = " );
              Serial.println( ServoPosition, DEC );
            }
          }
          void setup( )


          Serial.begin( 115200 );
  IRrecver.enableIRIn( );// 受信対象初期化

  servo_0.attach( servo_pin );// Servo 連接ピン設定


          }

          void loop( ) {
            if ( IRrecver.decode( &results ) )
            {
              Print_IRdecodeResult( results );
    IRrecver.resume( );// コーディング後、この行を呼び出すことで、そ
の後のデータ受信再継続
            }
          }
```

転送端、受信端を別々にアップロード後、転送端を編電圧に動かし、受

信端の Servo を制御する。転送端のプログラムと、0x00AA を組合せ、アップロードを完成させる。

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

この二つの語法は、データ解析に用いられるものである。

続いてその他の制限条件を増加する：「通信協定 NEC が解析した装置を受信端に接続して設定することで、初めて Servo の制御が継続可能である」。この様にして同類の通信協定

を用いるとしたら、自身で行う設定は簡略化することが可能である。

このやり方を認識したら、読者は先の章で述べたロボットアームを搭載し、赤外線無線制御のバージョンを実作しよう。そしてまた再度頭を使い、もし、86Duino EduCake を用いてマルチタイプのコントローラーwを実作するとしたら、どうだろうか？