EduCake　Analog I/O ピン機能応用編

1.　Analog I/O 紹介

- **アナログ vs デジタル：**

前篇では86Duino EduCakeの基本仕様とIDEの簡単な使い方やデジタルIOを説明しました。この章では、アナログIO機能を紹介します。

アナログ信号の説明の前に、デジタルとは何かをお話してデジタルとの比較を紹介します。前篇ではデジタル信号についてお話ししましたが一般的に2つの状態を表す電圧信号、ある程度よりも高い電圧：Highとある程度よりも低い電圧：Lowを示します。86Duinoを例にとると、digitalWrite()はあるピンを通じてHighまたはLowをOutputします。一方deitalRead()はあるピンを通じてHighまたはLowの状態を読み取る事が出来ます。スイッチやボタン等わずか2つの状態しか表さない信号にとっては十分です。

この章でいうアナログ信号とは連続状態の値をもつ、輝度、温度、湿度、音量、長さ、角度、重量等々　自然界に於いてよく見る事の出来る物理量で、各タイプの測定器は物理量を各電圧や電流値に変換する機能を持ちます。しかしプログラムが実行する際に必要な0と1を組み合わせたデジタル信号です。従ってある応用が上記の連続状態の物理量を取得してプログラムに使用した時にアナログtoデジタルコンバータ(ADC)と言われる回路にて変換される。以下はADC回路を持つ転換原理の一例である：

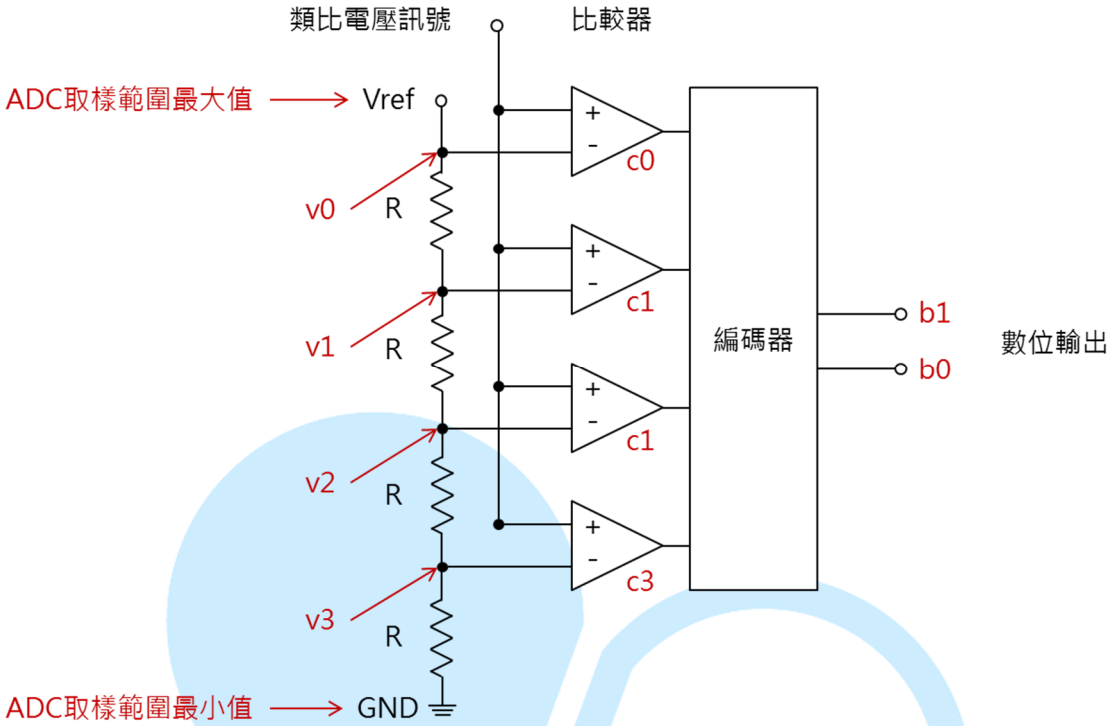


図 1. 「Flash ADC」

図 1 回路は「Flash ADC」と呼ばれるもので、比較的に概念が理解し易い一例である。ここで、4 つの比較器を持つ例を挙げる。入力部はアナログ電圧信号と比較電圧信号 Vref ; それぞれの比較器「-」側を v0~v3 に接続しそれぞれが Vref、Vref(3/4)、Vref(1/2)、Vref(1/4)となり、「+」側の電圧信号が「-」側よりも高い時に比較器は 1 と出力する。反対は 0 となる。フラッシュのエンコーダは c0~c3 の出力に対応する 2 ビットの数値に変換する機能を持つ。この例では比較器が 4 層のみで 2 ビットの出力となり、もし 4 ビットの出力が必要な場合 16 個の比較器が必要となる。それ以上のビット変換が必要な場合も想像できるでしょう。

Vref が 5V の例を紹介する。v0~v3 それぞれ 5V、3.75V、2.5V、1.25V の場合 ; アナログ信号が 1.5V で比較器は c0~c3 より 0/0/0/1 と出力され二進数の値は 01(十進数で 1)となる。3.9V の場合比較器は c0~c3 より、0/1/1/1 と出力され、二進数の値は 11(十進数 3)となる。比較器がもっとたくさんある場合以上を類推可能。従ってこの例における ADC 入力回路が 0~5V、出力ビット数:2 ビット、4 段階の解析により各電圧幅は 5/4 = 1.25V となる。

ADC の変換回路と設計の原理が多様化されており一つの学門となる。ここでは原理と応用例だけを紹介するので、興味のある方は関連資料を探してください。

ADC 仕様と変換方法：

通常 ADC 仕様に関して説明する際、一般的な入力可能な範囲と変換ビット数に触れる。前段で ADC の原理を説明しましたが ADC 入力範囲よりも低いまたは入力範囲を超えた時は Max あるいは Min でしか表現できない。一方変換ビット数はサンプリングの解析に影響する。ビット数が多ければより緻密なサンプリングが可能となる。

例 1：あるピンの ADC 入力電圧範囲が 0~5V、変換ビット数を 10bit とすると、0V 時はサンプリングの値が 0 となる。(二進数:0000000000)、5V の場合は 1023(二進数 : 1111111111)、0~5V 間を 2^{10}=1024 分割で解析可能。つまりこの ADC サンプリング解析度は 5/1024：約 0.004883V の分解能となる。想像してみて下さい、一本の温度計の 0~5 度が 1024 等分され、読み取った値が 1000 となった時、サンプリング信号は 1000*5/1024、つまり 4.883V を意味する。

例 2：あるピンの ADC 入力範囲が 1~7V、変換ビット数が 12bit の場合、各目盛は(7-1)/4096、電圧解析度は約 0.00146V となる。読み取った値が 1000 となった時、サンプリング原始信号は 1000*6/4096+1、つまり 2.465V を意味する。

従って ADC を使用するとき、基本仕様を理解して初めて値の変換が出来る。86Duino EduCake の場合入力電圧範囲 0~5V、解析度 10 ビットの仕様となる。

- **測定器に応用する際の心得：**

前項で各タイプの測定器は物理量を電圧あるいは電流信号に変換する事に触れましたが、通常 ADC は電圧信号を読み取るものであり、まして各種測定器の信号範囲が ADC の入力規格値に当て嵌まらないかもしれ

ないとの事から、測定器を使う前に出力信号を ADC の入力範囲に調整する事に注意が必要である。以下は例を挙げて説明します。

例 1 : 一般的な回転式の可変抵抗、圧力、光電、温度抵抗等の電気抵抗測定器は環境における一定範囲の変化に従って、自分自身の抵抗値を変化させる。使用時に以下の図に示されるように「抵抗分圧法」と呼ばれており、その中で Vcc が ADC 最大入力電圧値に近いものを使用する。例えば EduCake は 5V となる。測定器から ADC までの入力電圧は Vcc*R1/(R1+R2)、図 2(b)可変抵抗の R1+R2 は固定値とし、図 2(a)の中の R2 は自由に設定出来る。通常、1KΩ前後を選択し電流制限用にする。と言っても R1 との差異が大きくてはいけない。もし VCC がその他の固定電圧値を使用するときは上記の原理通り、R1 側の電圧を ADC 入力範囲に当て嵌まる様に調整する。

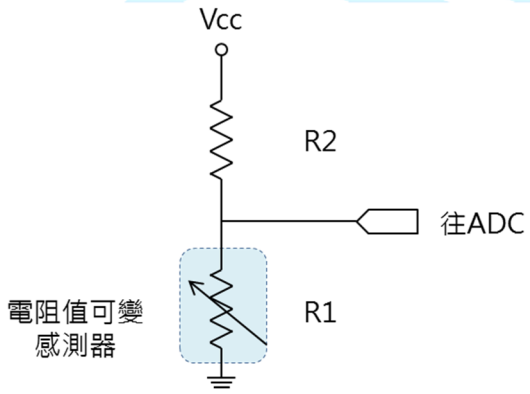


図 2(a)

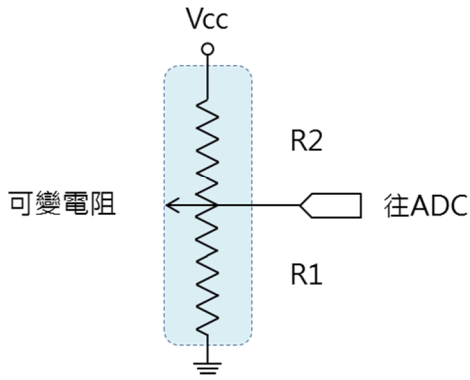


図 2(b)

例 2 : 太陽からの放射照度を測定する際に使用される日射計 (Pyranometer) を例に取ると、一般的な出力仕様は 4~20mA、この測定器の特性は電流に変換して出力する事である。この場合は図 3 の通り測定器を抵抗 R1 に直列接続する事で 250Ω では V=I*R の計算で 4～20mA を 1～5V に変換する ADC 仕様となる。但し使用する際には電流流れの方向に注意が必要。

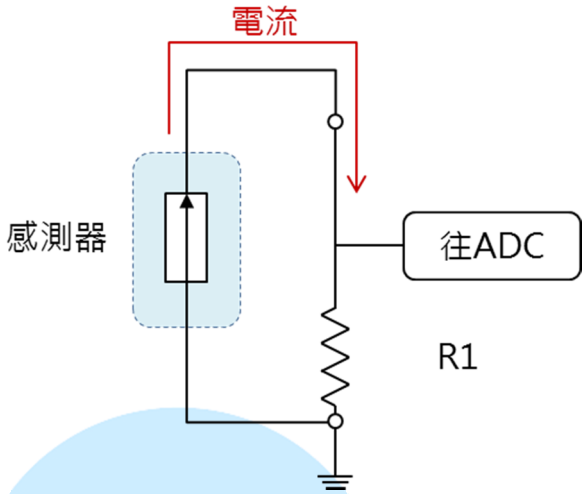


図 3

- アナログ出力

前の章では ADC を用いてアナログ信号をデジタルデータに変換する処理を説明した。アナログ出力は丁度反対の仕様で、プログラム内のデジタルデータをアナログ信号に変換し出力する。一般的にはデジタルデータを「純正」なアナログ信号に変換する際にデジタル-アナログ変換回路(Digital to Analog Converter: DAC)を使用する。但し、全ての出力装置は純正なアナログ信号を使うとは限らない。例えば電球または直流整流子モータ等の装置に関し、特定時間内における電流または電圧の通過時間の比例を調整すれば輝度、または回転速度を調整する事が可能。86Duino EduCake では使用するアナログ出力も「PWM」(Pulse Width Modulation　パルス幅変調)方法を使用している。PWM はアナログ出力として使用するとは限らない。後に専門文書を用いて紹介する。本文では PWM を用いたアナログ出力応用に注目して説明する。

図 4 の通り、上から下にそれぞれ 25%、50%、75%、100%の PWM 比(duty cycle とも呼ばれている)とする。目の残像現象によっては周波数が高ければ PWM 電圧が LED または電球に使われる際の輝度の変化に相当する。EduCake のアナログ出力周波数は約 1000Hz である。前の章で説明した digitalWrite( )を delay( )と組み合わせて delayMicroseconds( )となる関数と同等の効果となる。但し周波数が低すぎると点滅現象が発生する。

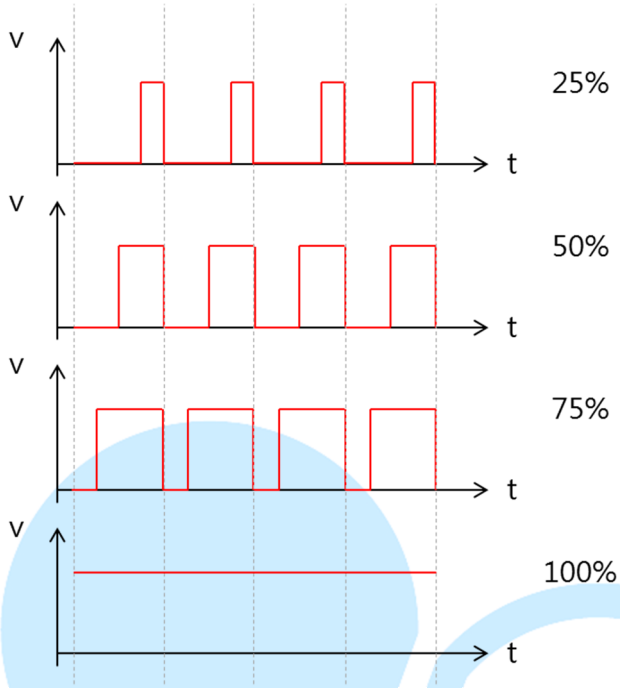
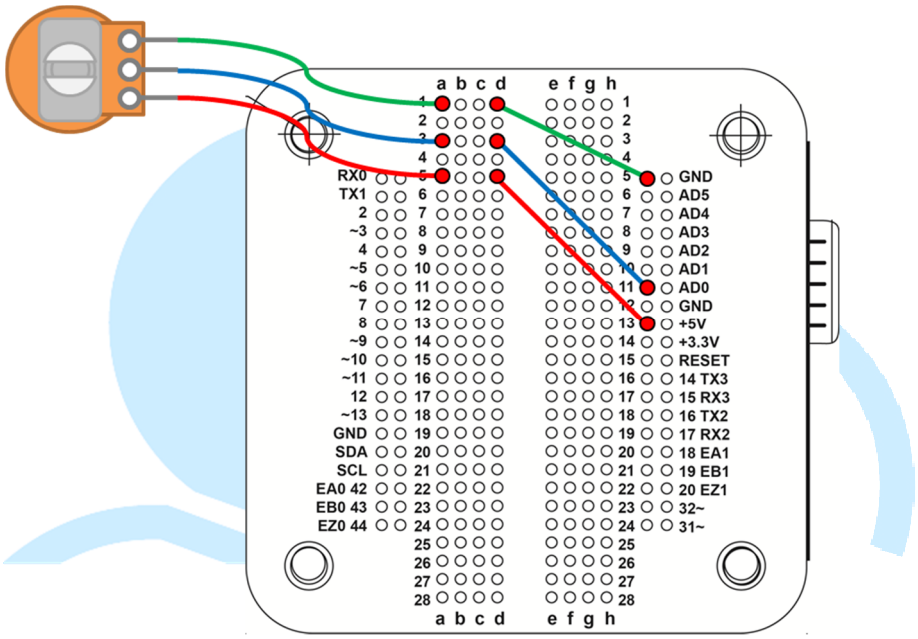


図 4

原理の説明はここまでとして、ここからはプログラムを使って練習する。

2. プログラミング 1 – analogRead( )

先ず EduCake の簡単なアナログ入力機能を使ってみます。ここで使う関数は analogRead( )です。必要な部品は秋葉原等で簡単に入手できる 10kΩの可変抵抗で十分です。接続は以下の通り



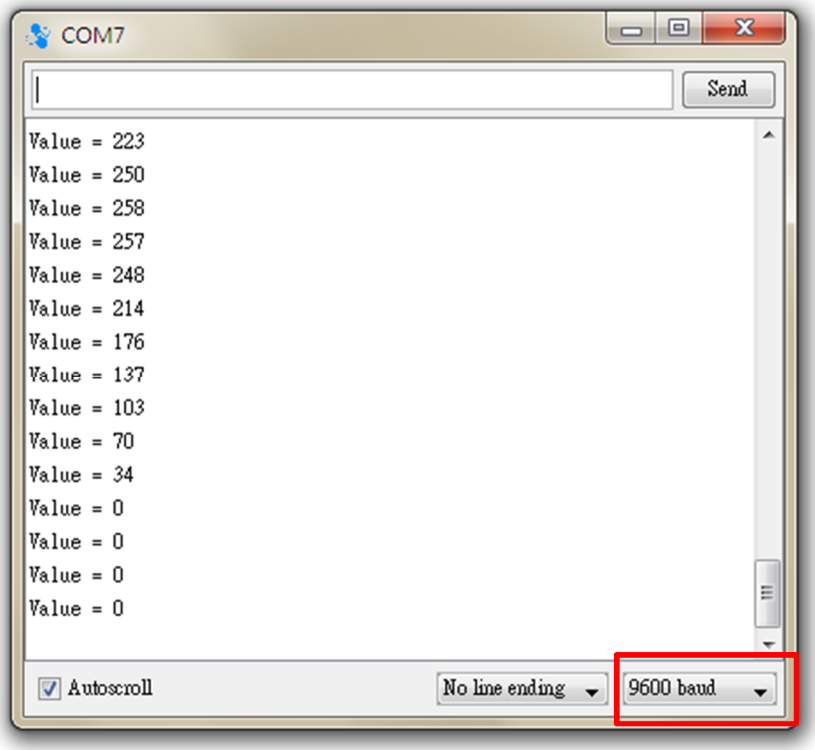
この接続方法は先程説明した原理図 2(b)に相当する。ここで注意して欲しいのは、可変抵抗は 3 つのピンを持っているが真ん中のピンが変化抵抗の端子として使用される。その他 2 つのピンは±の指定が無く、それぞれを EduCAKE の GND と 5V に接続すれば良い。もし、逆に接続した場合可変抵抗を操作した際の変化量が逆になる。表示内容と合わせる場合、接続に注意が必要。この例では ADC が読み取った値を、パソコンと接続している USB ケーブルを通じてデータを渡し印刷する。次に 86Duino Coding 100 の IDE 画面から以下のプログラミングを入力する。

```
const int analogInPin = A0;// アナログ入力ピン番号定義

int sensorValue = 0;// ADC 読取数値定義

void setup() {
  Serial.begin(9600);// パソコンとの通信ボーレート設定
}

void loop() {
```

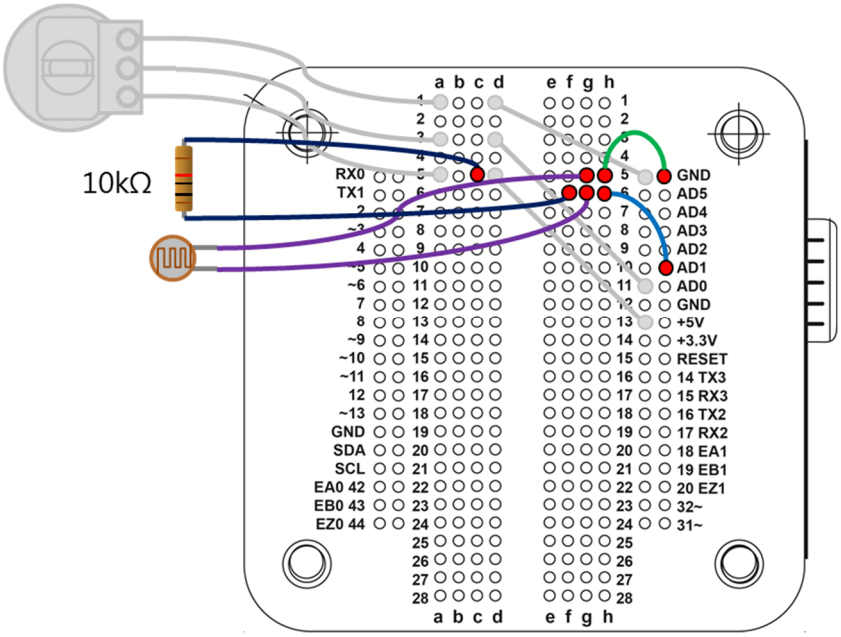
```
    sensorValue = analogRead(analogInPin);// ADC データ読取

    Serial.print("Value = " );//  印字
    Serial.println(sensorValue, DEC);// ADC データ印刷

    delay(100);// 0.1 秒タイマ
}
```

プログラムは最初に setup( )にてパソコンとの通信スピード(baudrate)を設定し、loop( )にて analogRead( )で ADC の読取を連続して行う。また値を Serial.print( )、Serial.println( )使い Serial Monitor に出力させる。この二つの関数の違いは println が文字列を読み易くするために印刷後に改行する。Serial.println(sensorValue, DEC)の 2 つ目のパラメータ DEC は読み取った値を 10 進数で表示することである。読者も BIN,HEX 等の 2 進数や 16 進数で印字してお試しください。delay(100)は印字後に 100ms(0.1 秒)のタイマを設けているが、時間に関しては調整して下さい。

Upload を選択しプログラミングをアップロード後に Tools > Serial Monitor(または Ctrl+Shift+M)を開けば、下図の通り直ぐに一連の ADC 読取データを見ることができます。ここで、可変抵抗を操作すると表示される値が 0～1023 に変化します。もし、Serial Monitor を開いて反応が無ければ右下の baud の設定が Serial.begin(9600)と一致しているかのチェックが必要。今後、全てのプログラムに於いて変数のチェック、トラブルシューティングに使い勝手の良いツールとなる。



次に計測器の種類を変えて図 2(a)の回路を使って練習しましょう。後に紹介する幾つかのプログラムの為、今までの部品は取り外さずに、下記の様に新たな部品を追加してみましょう。



フォトレジスタを使った回路ですが、この部品も秋葉原等で購入する事が出来ます。ここで選択したフォトレジスタは一般室内で数十 kΩ の値

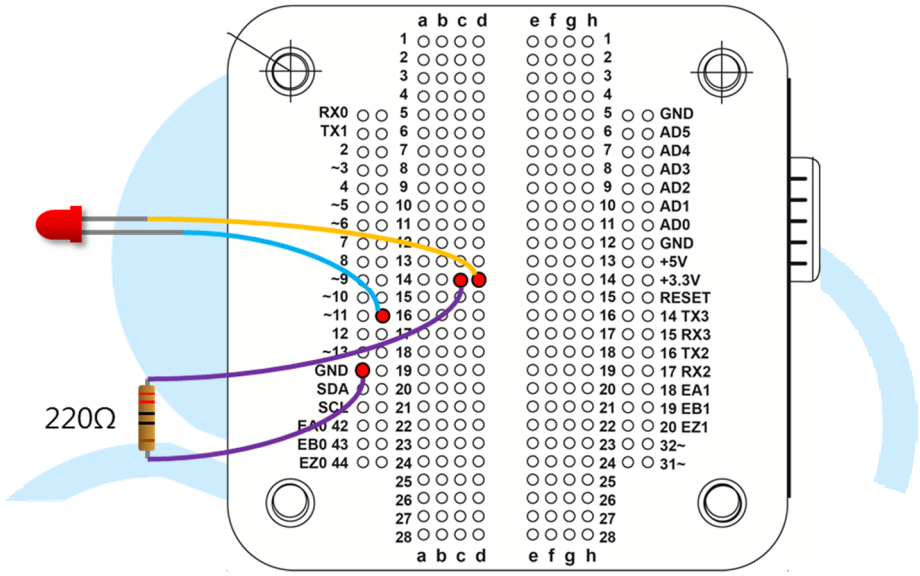
が得られるものです。更に、10kΩの抵抗を組み合わせて使用する。フォトレジスタの片方はGNDに接続し、もう片方はAD1に接続する。10kΩの抵抗の片方は5Vに接続する。フォトレジスタは光の照射によって抵抗値が低くなり、光の照射が無い時に抵抗値が最大となる。(最大ではMΩクラスもある)。従って、回路の接続方法によっては輝度が高くなればADCの読取数値が小さくなり、輝度が低くなれば読取数値が大きくなる。プログラムは下記の通り修正

```
const int analogInPin = A1;//  アナログ入力ピン番号定義
```

基本的にアナログ入力ピンをAD1に変更しただけであるので、前のテスト同様に光の照射によって出力が変化する事を確認してください。ここまでの説明でわずか数行のプログラミングにより、様々な変化量を簡単な回路で読み取る事が出来る事をお判り頂けたでしょう。

3. プログラミング２ – analogWrite( )

ここでは LED の輝度を連続的に変化させて呼吸の様にゆっくり変化させて点灯／消灯させます。この例では一般的な LED を用意して下さい。もう一つは過電流で LED を壊さない様に 220Ω の抵抗を用意して下さい。接続は下記の通りですが、今までの回路はそのままで追加してください。



LED「+」側(長いピン側)を EduCake11 ピンに接続。「~」が付いているのは PWM での出力可能なピンである。以下のプログラムを入力

```
const int analogOutPin = 11;// アナログピン番号定義
const float Freq = 0.5;// LED 点滅サイクル
unsigned long time;// 時間変数
byte LED_light = 0;// LED 輝度

void setup() {
}

void loop() {
   time = millis();
   LED_light = byte(128 + 127 * cos((float)time * 0.001 * 2 * PI * Freq));//
LED 輝度計算
   //Serial.print("LED_light = " );// 印字
   //Serial.println(LED_light, DEC);// 輝度値を印字
   analogWrite(analogOutPin, LED_light);// PWM 波形を LED に出力

   delay(50);// 0.05 秒タイマ
}
```

本プログラミングで使用される一連の設定値、ピン番号、LED 点滅サイクル、時間変数、LED 輝度などを定義する。メインループにて時間変数を取得し millis( )関数で、プログラム起動後の時間をミリ秒単位で取得する。そして、余弦計算式(正弦 sin も可能)を使って毎回必要とする LED 輝度を計算する。ここでの数学計算式は

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

アナログ出力は analogWrite( )関数を使用する。パラメータ範囲として 0～255、0 とは PWM 周期の High 部分が無い状態となり、255 とは PWM 周期の High が 100%となる。128 では High が 50%となる。cos( )関数出力結果は-1~1 の範囲での値となり、127 の振動幅を掛けて更に 128 を加える事で 1～255 の値に反映させる。2πft 演算は cos( )関数の周波数調整に用いられる。f : 周波数 Hz、time ： millis()で取得される為、掛ける 0.001 で実際の秒数となる。最後にタイマを設ける。

この例では、LED の点滅比率を 0.5Hz としているので LED の明るさが 2 秒毎にゆっくりと変わっていく。Delay()の数値を大きくすることで、LED 明暗が段階的に変化する事が確認出来る。従って本プログラミングでは例として 50ms のタイマと、20Hz の計算式を用いて LED 輝度を表現する事でより滑らかな視覚効果が現れている。もう一つシリアルモニタの使い方を紹介します。以下のプログラムで上段を下段の様に変更します。

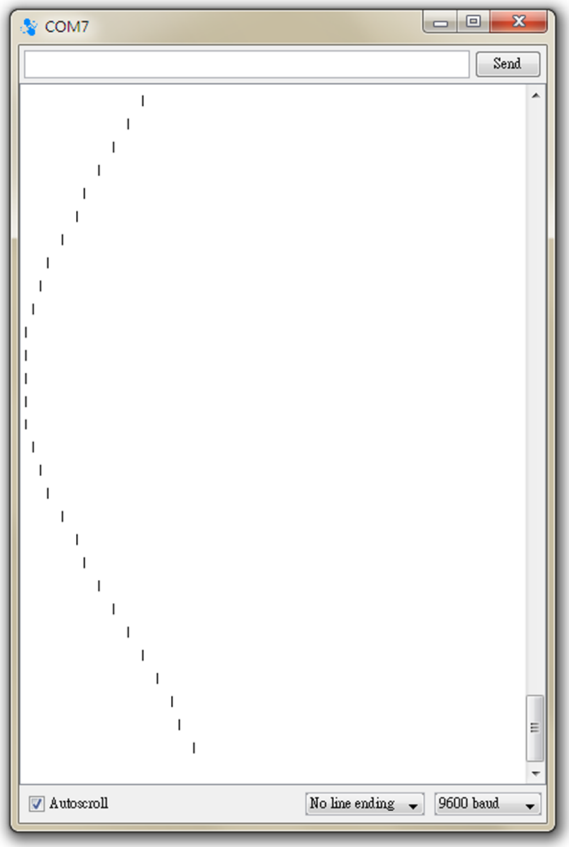
```
//Serial.print("LED_light = " );// タイトル印字
//Serial.println(LED_light, DEC);// 輝度データ印字
```

変更後

```
for(int i=0;i<LED_light/10;i++){
   Serial.print(" " );// 空白印字
}
Serial.println("|");//   変化量を記号で表示
```

このプログラミングは輝度データを利用し Serial.print( )にて空白文字と記号を組み合わせて表示する事で、変化量を視覚的に表示させている。

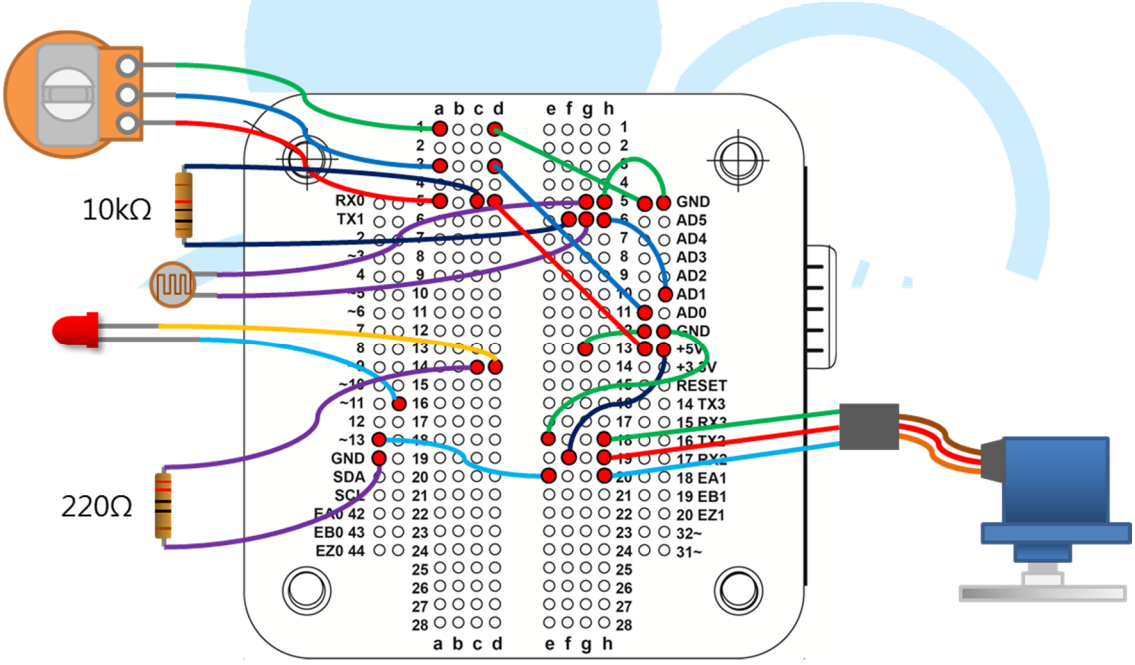
プログラムをアップロード後 SerialMonitor を開く事で以下の様な表示を確認出来ます。



この波型はほぼ LED とリアルタイムです。輝度計算式を変える事で、もっと沢山の変化を作り出す事が出来ます。今回のプログラミングを使う事で LED があたかも呼吸をしているように点灯させたり、心臓の鼓動の様に表示さることも可能です。

4. プログラミング 3

本文最後のプログラミングは集大成として、前述の内容にもう一つの部品を加えてより多くの変化を作り出します。ここでは小型のサーボモータを用意して下さい。(説明では Tower Pro SG90 を使っています)これはラジコン等で良く見かけられるモータで RC Servo とも呼ばれています。通常、比較的大型の RC Servo は高電流と高電圧が必要となります。ここでは SG90 を選んでおり EduCake の 5V に直接接続して給電可能なため、その他の電源供給は不要となる。前のプログラミングで使用した接続にプラスして以下の様に小型サーボモータを追加する。



SG90 本体から出ているケーブルの茶色：GND、赤：5V、橙：信号

RC Servo の信号ケーブルは通常 PWM 信号を入力する。この例では橙ケーブルを EduCake の 13 ピンに接続し、以下の通り入力する。

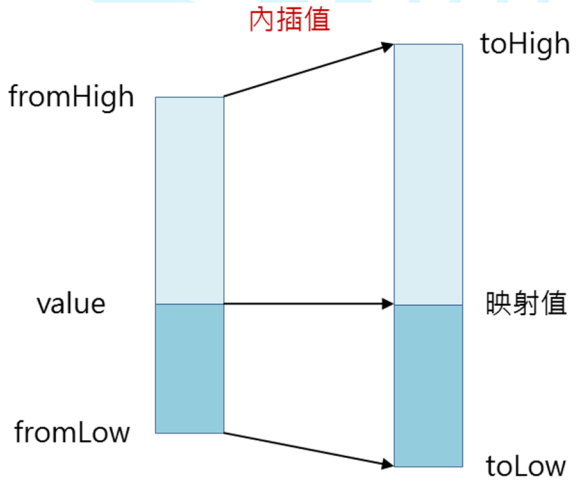
```
#include <Servo.h>

const int Vr_analogInPin = A0;// 可変抵抗　アナログ入力ピン番号定義
const int Ls_analogInPin = A1;// フォトレジスタ　アナログ入力ピン番号定義
const int LED_analogOutPin = 11;// アナログ出力ピン番号定義
const int Servo_Pin = 13;// Servo 出力ピン番号定義

int Light_threshold = 0;
```

```
Servo Servo_1;

void setup() {
  Serial.begin(9600);
  Servo_1.attach(Servo_Pin);//, 500, 2400
  Servo_1.write(90);// Servo 中央設定
  Light_threshold = analogRead(Ls_analogInPin);// フォトレジスタの
ADC データを読取、初回のデータを下限とする
}

void loop() {
  int Vr_rsensorValue = analogRead(Vr_analogInPin);// 可変抵抗の ADC
データ読取
  int Ls_rsensorValue = analogRead(Ls_analogInPin);// フォトレジスタの
ADC データを読取

  Ls_rsensorValue = constrain(Ls_rsensorValue, Light_threshold, 1023);//
読取データ範囲の制限
  byte LED_light = map(Ls_rsensorValue, 512, 1023, 0, 255);// フォトレ
ジスタの ADC データを読取、LED の輝度とする
  analogWrite(LED_analogOutPin, LED_light);// PWM の波形を LED に出
力

  int Servo_1_angle = map(Vr_rsensorValue, 0, 1023, 0, 180);// 可変抵抗
の ADC データを読取 Servo 角度とする。
Servo_1.write(Servo_1_angle);// PWM 波形を Servo に出力する
  /*
  Serial.print("Vr Sensor value = " );
  Serial.print(Vr_rsensorValue, DEC);
  Serial.print(", Servo angle = " );
  Serial.print(Servo_1_angle, DEC);
  Serial.print(". Light Sensor value = " );
  Serial.print(Ls_rsensorValue, DEC);
  Serial.print(", LED light = " );
  Serial.println(LED_light, DEC);
*/

  delay(20);
}
```

SG90 Servo も PWM 信号を受信するが、周波数と duty cycle 共に analogWrite()とは異なるものである。従って、プログラムの最初に include<Servo.h>を記述し Servo オブジェクトを使う事によってより多くの関数を使う事が出来る。この例の機能として可変抵抗を使って周りの明

るさを測定し LED の明るさを調整する。また、可変抵抗を回す事で RC サーボの回転を調整する。注意が必要なのは Servo.h がオブジェクト仕様言語である為、最初に Servo_1 の定義を行う。引き続き setup()にて Servo_1.attach(Servo_Pin)を使い Servo_1 を Servo_Pin に渡す。複数の Servo を接続する際は、夫々を Servo.attach()で記述する必要がある。Servro_1Write()は Servo_1 を指定した角度に回転させ、通常は中央が 90 度、両端が夫々0,180 度である。Servo の使い勝手に関しては後程、詳しく説明します。ここでは Servo.Writeh()関数を使い角度(回転)の制御方法を説明する。

loop( )関数にて可変抵抗とフォトレジスタの ADC データを読取、次に map( )関数を用いてデータに変換する。map(value, fromHigh, fromLow, toHigh, toLow)関数は比例に基づいて計算された値が変換される。以下が概念図となる。



Value：変換したい値

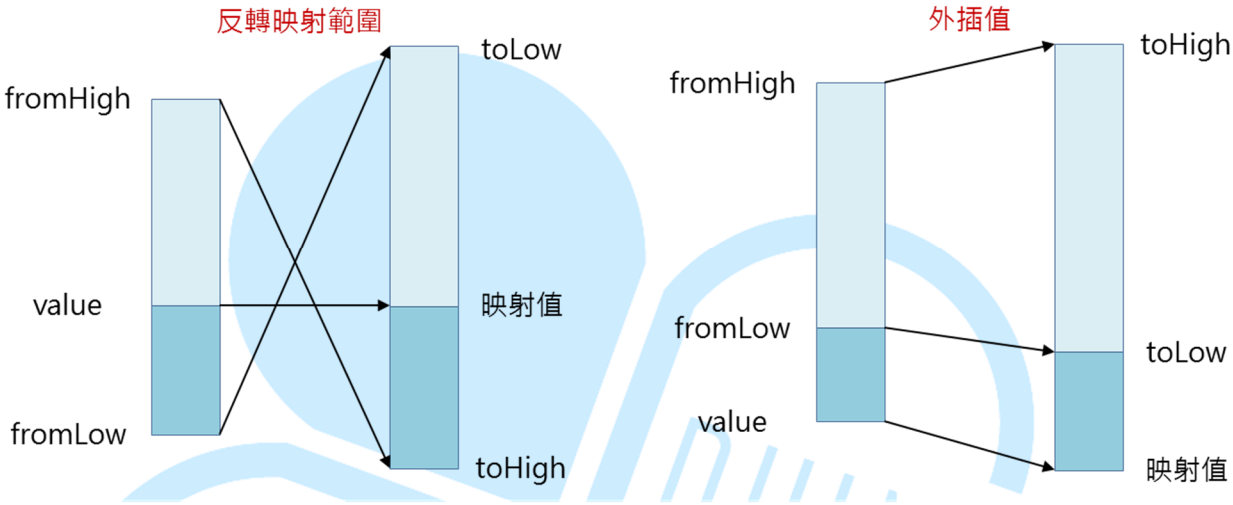
fromHigh：変換したい値の上限　　fromLow：変換したい値の下限

toHigh：変換後の上限　　toLow：変換後の下限

例えば、ADC が読み取った値:127、変換したい値の下限-上限:0~1023、, 変換後の下限 - 上限:0~255 従って y = map(127, 0, 1023, 0, 255)　結果として y=31 となる。範囲の下限を上限より大きな値に設定した場合、値の反

転として使えます。また、上限／下限には負の数を使う事も可能、小数が生じた場合整数に変換後計算される。以下 2 つの図の通りです。以上の点を注意しないと思い通りの結果が得られません。

プログラミング 2 で行った実験を map()関数で置き換えるとどうなるか、試してみて下さい。



loop( )関数にて 2 回 map( )関数を使用し、ADC データを LED 輝度 0～255、及び Servo 角度 0～180 度に変換させる。これは、今後多くの状況で使える機能である。このプログラムの特色としては変数 Light_threshold を定義し、起動時にフォトレジスタの値を下限として保存するときに使用する。このプログラミングにより暗くなった時に LED を点灯させる事が可能となる。constrain(Ls_rsensorValue, Light_threshold, 1023)は後でフォトレジスタが読み取った値が Light_threshold から 1023 までの値に入っているべきで、変換データが 0~255 の範囲に収めるものとなる。これも、計測器を使う際の技の一つである。そもそも回路は同じ環境条件で運転されるわけでは無く、電源が投入された時点でデータを読み取る事で、校正データとして利用する事が出来る。

このプログラムを Upload すると、照明が暗い状態でテストすると LED は消灯している状態に近い。フォトレジスタへの光を徐々に遮る事で、LED が段々明るくなり完全に遮った際、LED が最も明るい状態になる。同時にシリアルモニタを使えば、ADC データの変化状況を見ること

が出来る。これは通常の明かり(電源は別途)と組み合わせる事で、夜間照明等の応用として使う事も出来る。可変抵抗を操作する事で Servo 角度の変化をみる事も出来る。delay( )を長くする事で可変抵抗の変化に対する反応の遅れを見ることが出来る。これは、ADC データを読み取りサーボへの書き込み周期が減る事を意味しています。

86Duino EduCake を使い、今までとは違った日常生活にも関わるような自動制御のアプリを試してみましょう。もう少し上級の使い方を別途ご案内します。