PWM 使用方法

1. PWM とは?

基本的な信号の制御を前章で紹介しました。前章でデジタル信号及びアナログ信号を紹介し、最後にデジタル信号を使ってアナログ信号に擬似化する PWM と呼ばれるパルス幅変調(Pulse Width Modulation)に触れましたが、本章では PWM を使った電子部品の電圧を調整する応用を紹介します。実際にはモータの回転、灯りの明暗、ディスプレイの輝度、スピーカのボリューム、音声の高低等に利用されます。デジタル信号のパルスを調整してプログラムで High/Low の状態を高速で切り替えます。通常はシステムタイマと合わせて一連のパルスを以下の図の様に出力します。

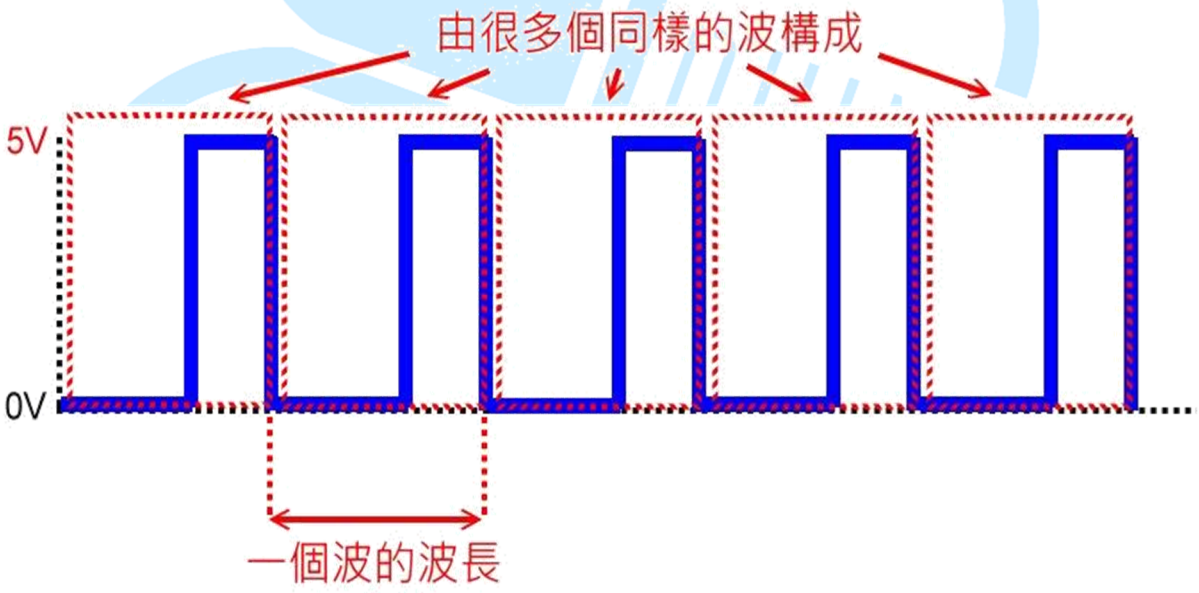

図 1. PWM パルス

パルスが示す間隔を周期と言い、周波数(Hz)は一定時間内に幾つの波形が発生したかを意味します。例：1 秒間に 60 個のパルスが発生したら 60Hz/sec と表現します。大量のパルスを発生させ、且つ周期、周波数、波長を調整する事で図 2 の様にアナログ出力に似た信号として使い様々な応用が出来る。

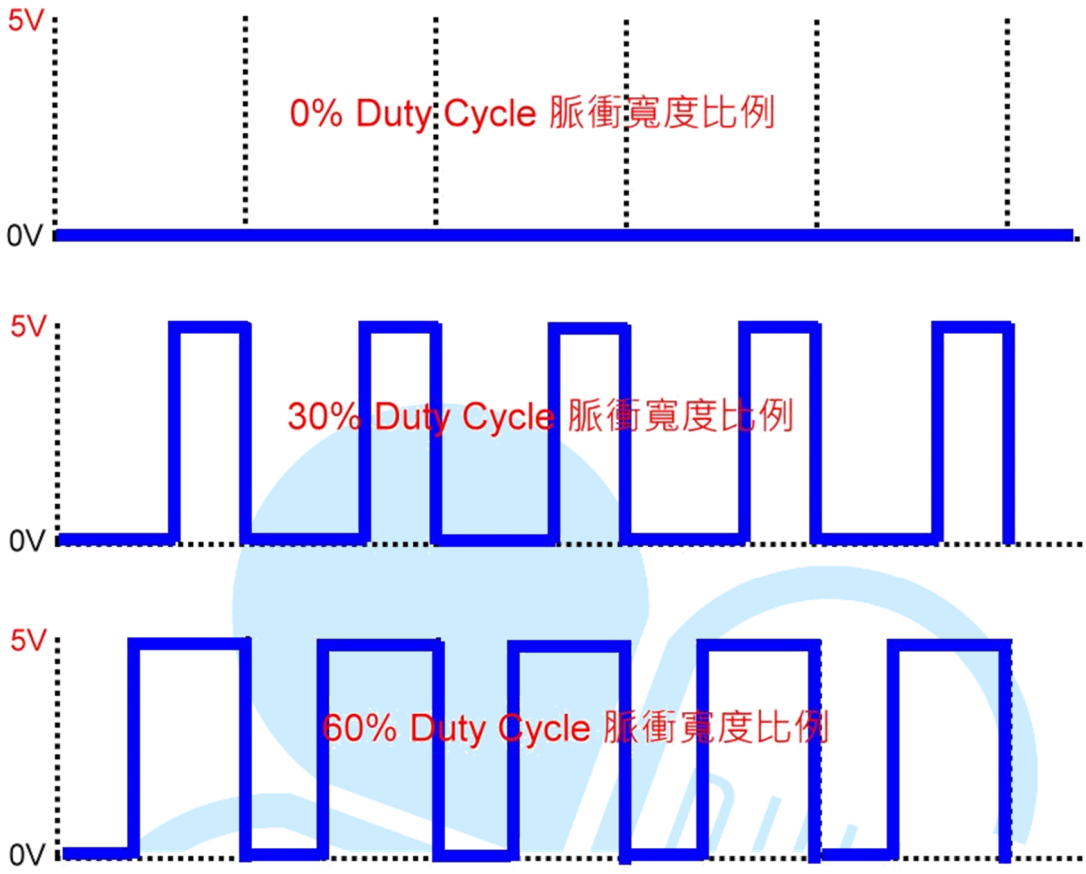

図 2. 異なる Duty 比の PWM

通常の PWM 波形を作る場合、複雑なプログラミングは必要ありません。(後に複雑な波形或いは高周波波形の作り方を紹介します)　以下のコマンドで図 2 の波形が作成出来ます。

図 2 上段：　analogWrite(ピン番号,0);

図 2 中段：　analogWrite(ピン番号,78);

図 3 下段：　analogWrite(ピン番号,153);コマンド中の数値は概算である。

PWM 制御コマンドのパラメータは 0～255 の範囲となり、従って 30%High とすると、255*30%=約 78 となる。analogWrite コマンドについては、指定した特定のデジタルピンに使われ、デジタル信号を利用して PWM 信号を出力している。コマンドは以下の通り。

```
analogWrite(ピン番号, 数値)
```

コマンドの中の数値は必ず 0～255 となり、0～5V に対応させる場合 5V/255=0.0196V の分解能となる。上記の様に analogWrite(ピン番号,160) とした場合、160*0.0196V=3.13V の擬似アナログ電圧として出力される。周波数は 1KHz である。図 3 の EduCake 表面の”～”表示があるピンで本 PWM 機能が設定可能です。

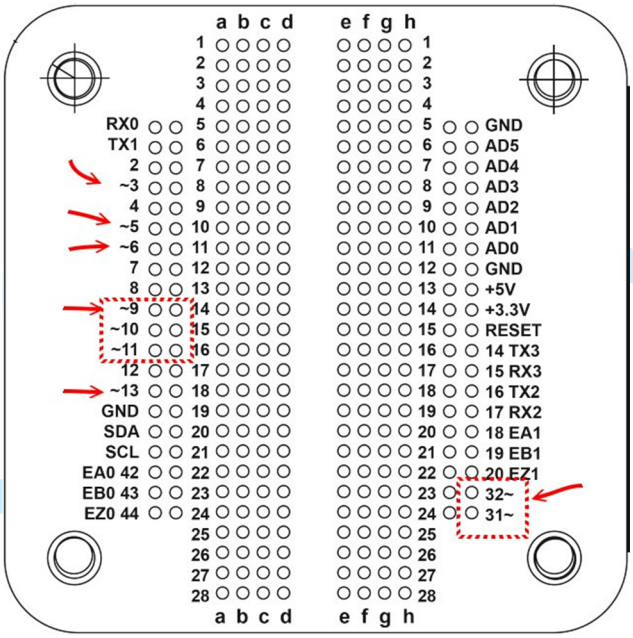

図 3. 86Duino EduCake PWM 対応ピン配置

重要な事は、今までに説明した信号は High/Low のデジタル信号を周期的に組み合わせたデータであり、信号の劣化が無く保存、転送、ノイズに強いデータとして使える事である。LED 点滅制御は良く見かける例です。以下は制御プログラムの一部です。

```
digitalWrite(3,HIGH); // HIGH 設定:ピン番号 3LED 点灯
delay(1000);
digitalWrite(3,LOW); // LOW 設定:ピン番号 3LED 消灯
```

上記はピン番号 3 の LED を点灯した後、1 秒後に消灯する単純で直接的な点灯／消灯のデジタルプログラミングである。明暗の切り替えが ON/OFF で単調な切り替えとなるが、PWM を使ってアナログ的に変化させれば様々な面白味のある表現が可能です。プログラムは以下となります。

```
for(int a=0;a<=255;a++) // ループ  ピン番号 3 の LED をゆっくりと点灯させる
    {analogWrite(3,a); delay(10);}
delay(1000);
for(int a=255;a>=0;a--)// ループ  ピン番号 3 の LED をゆっくり消灯させる
    {analogWrite(3,a); delay(10);}
```

上記のプログラムで LED の点灯がゆっくり点灯し最も明るくなった後、ゆっくり消灯する演出となる。この動作を繰り返し行い、ちょっとした調整で LED 点灯があたかも呼吸しているように表現する事が出来る。試にこの装置を寝室に置いてテストしましたが、スイッチを入れてゆっくりと明るくなり、また暗くなりその周期が次第に長くなって行き 30 分後に完全に消灯し変化しなくなります。この実験に合わせて聞き心地の良い軽めの音楽との相乗効果であかちゃんを寝かせるにも最適な環境を作る事が出来た。勿論プログラミングはここで紹介したよりも、もう少し工夫している。同様な処理として Loop の中で a++を使わずに、乱数用いて明暗を変化させれば蝋燭の揺らぎの様な明かりを作り出す事も可能である。呼吸の様な明かりの変化をする LED ライトを蝋燭の様に変化する LED ライトに変える事も面白い実験と言えるでしょう。また、音センサと組み合わせるれば息で明かりを吹き消す事が出来る LED ライトも作り出せる。拍手の音でも同じように LED ライトの消灯に使う事が出来る。レストランのテーブルに応用すれば直ぐにでも実用出来る一例となるでしょう。

いろいろな用途で使える応用事例を考える事も実験をより面白くしていくでしょう。

当然、上記で説明した簡単なプログラミングにはちょっとした問題が含まれています。LED ライトを点灯させるには一定の電圧が必要です例:2.5V の場合 PWM で擬似的に表現する場合 125 以上に設定が必要で、0～124 の場合は LED ライトは消灯状態となり、変化があまり見られない。同様に LED ライトが 4V の時には非常に明るい状態で,4～5V の間では明るさの違いは殆ど見られない。従って、この実験を行う時は使用する LED の最も暗い状態と最も明るい状態の範囲をテストする必要がある。その値に合わせて Loop の値を修正すると、より良い効果が得られる。

以上紹介しましたのが基本的な PWM 機能で、86Duino EduCake ではこの基本 PWM 信号を周波数 1,000Hz/sec で提供している。つまり波長は 1sec/1,000=0.001 秒であり、このスピードは日常的な使い方では十分に利用可能な仕様である。もっと精密な PWM 出力を得るには他に幾つもの方法があります。注意としてピンの出力電流が非常に小さい事で、大型の電力が必要なモータ、扇風機、更にエアコン(可変周波数 DC エアコン)等に使う事は可能であるが、この場合出力電流を増幅して使う事になる。後に PWM に関する幾つかの応用を見てみましょう。

2. PWM 応用 1　呼吸ライト

以下の応用回路図はとても簡単で、図 4 の様に直接 LED と抵抗を半田付けして 2 本の端子をピンに差し込んでいますが、以前の様に LED と抵抗をそれぞれピンに差し込んで使う事も出来ます。

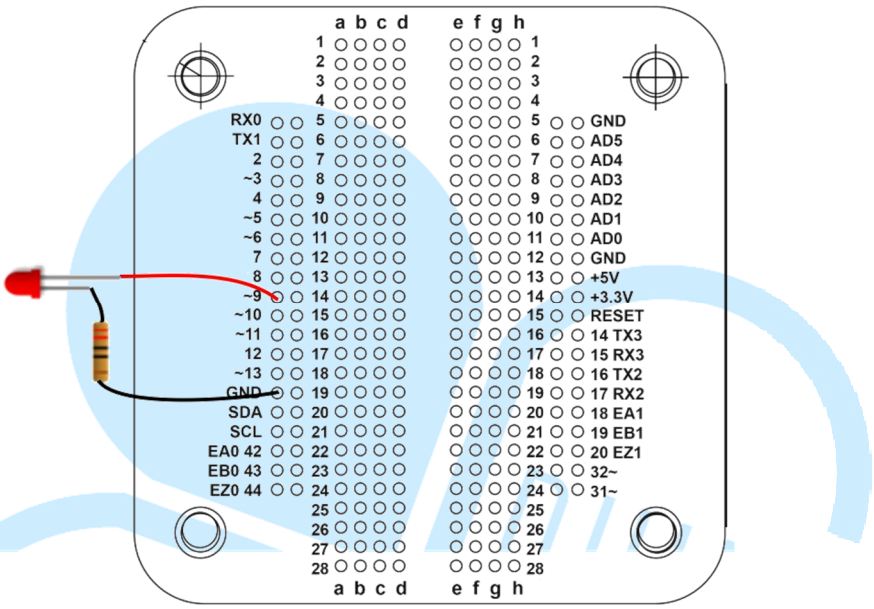

図 4. 接続図

プログラム内容

```
int brightness = 0; // 輝度変数

int fade= 5; // 輝度変化変数

int delaytime= 50; // 輝度変化速度制御
```

```
void setup()  {
  pinMode(9, OUTPUT);
}

void loop()  {
  analogWrite(9, brightness);
  brightness = brightness + fade ;
  if (brightness <= 50 || brightness >= 205) {
    fade = -fade;
  }
  delay(delaytime);
}
```

その中の **if (brightness <= 50 || brightness >= 205)** 2 つの数値は以前に話した完全に消灯した状態から、僅かに明るくなる数値(この実験では 50)と最も明るい位置(この実験では 205)を実際に使う LED で確かめて下さい。この値が不正確だと最も暗くなってからと、最も明るくなってからの待ち時間が長くなる可能性があります。LED の変化が滑らかな呼吸で表現されるかどうかは、これらの値の調整に掛かってきます。

それでは先程説明した蝋燭の明かりを表現する方法を説明します。まず、蝋燭の明かりを想像してみて下さい。

1. 蝋燭の明かりは必ず発光する。　　＜＜当然ですね！！
2. 蝋燭の明かりは揺らぐ　　＜＜空気の揺れに依る効果を実現する為にはモータと機構部品を合わせて使う必要がある。後に説

明するモータ制御と組み合わせて実現する事が出来るが、ここではLEDしかない為に省く。もっと簡単な類似効果としては透明なセロハンと風の組み合わせで揺らぎを表現する、例えばセブン・イレブンにあるおでん売り場等ではアツアツの商品を表現するのにこのような装置が使われているところもある。

3. 蝋燭の明かりが風に吹かれて、輝度が不規則に変化する。

実際、第1条件をリストアップした時に自分自身失笑してしまった。ばかげた事の様に思われるかもしれないが、事象をプログラミング化する場合、事象を細分化して一つ一つの動きをIDEが提供するコマンドに擬似化し積み木の様に積み上げる事で実現している。大型プロジェクトでも同様な手順で解決するものであり、この手順は非常に重要な解析方法となる。以下のプログラムを見てみましょう(回路の修正は不要)

```
int brightness = 160;
int fade;
int delaytime= 20;

void setup()  {
  pinMode(9, OUTPUT);
  analogWrite(9, brightness); //  第1条件蝋燭の明かりは発光
```

```
  randomSeed(analogRead(0)); //第 3 条件の為の乱数初期化
}


void loop()  {
 fade= random(0, 61); // 0～60 の乱数発生
 fade-= 30 //  乱数-30 : -30～+30
   brightness = brightness + fade ; //  鍵となる第 3 条件
 // fade を brightness に加える  輝度が乱数により不規則変化する
  if (brightness <= 60 )   //  蝋燭の明かりが消えるのを防ぐ、微かな
点灯状態よりも暗くならない値に設定
    brightness = 60;
 if (brightness >= 200)   //  蝋燭の明かりが上限の明るさを超えない
値に設定
    brightness = 200;
 analogWrite(9, brightness);
  delay(delaytime);
}
```

それらの数値とパラメータは需要に合わせて調整して下さい。この調整次第で明かりの効果が満足できるものになるか決まります。RGB3 色の LED を使えば、予想外の変化をもたらす事も可能である。今回の実験を生活のどこかに応用できるかもしれません。是非、効果的に使える場所を探してみて下さい。

3. PWM 応用 2　SERVO モータ制御

各種リモコン用の SERVO または PWM 制御を用いるモータ等は全て、EduCake の PWM 信号を使ってコントロールする事が出来る。ここでは最も入手しやすい SERVO モータを使って実験します。図 5 の様に SG90 と呼ばれ入門用に用いられる一般的な SERVO モータです。

図 5.　一般的な SERVO で使われるコネクタ

このリモコン用の小さい SERVO モータは手頃の値段で入手可能で、オークション等でも安価で購入出来ます。主にはラジコン用飛行機、ミニカー等の制御に用いられる。ここで注意して欲しいのは、橙 : 信号ケーブル(メーカによっては白)、赤 : +電源、茶 : GND(メーカによっては黒)の 3 種類のケーブルが使われます。この中で橙は EduCake の PWM ピンに直接接続する事で制御が可能である。但し、このモータは入門レベルのおもちゃである為、品質は期待出来ない。従って、実験用には最適で壊したとしても心は傷まない。規格は以下の通り。

1. 重量　9g
2. 寸法　23*12.2*29mm
3. トルク　1.8kg/cm(4.8V、最大 6V)
4. 速度　0.1sec/60degree(4.8v) SERVO が 4.8V で 60 度回転する際に早くても 0.1 秒要する

5. Dead band width 10us　個人的には“無応答幅”と翻訳したい。これは前回 SERVO に送り出した PWM 信号と今回送り出した信号との差が 10μs より小さい場合、SERVO が完全に無反応となる可能性が大きい。或いは SERVO 精度制御と言っても良い、これは主に SERVO の中の VR 解析度に関わるからです。

6. 回転角度約 150 度　＜＜これはとても重要で、使用する SERVO モータが持つ最大回転角度を超えた PWM 信号を送るとモータが焼損する可能性がある。

以下の SERVO でも実験可能です。

（http://www.roboard.com/Files/RS-0263/RoBoard_RS-0263.pdf）　このモータは精度が明らかに良くて、無応答幅が小さく、パワーも大きい。より精度を求める方にはこちらの SERVO をお勧めする。この SERVO を利用する事でより精度の高い動作が期待できる、金額に見あった品質が結果に表れます。接続は図 6 の通りです。本来 SERVO モータを接続する場合は、このような単純な接続をしてはいけませんが、今回使用する SERVO モータは低電力ですので単純で簡単に接続したテストを行っています。制御盤にとってはモータの電流が比較的に大きく負荷が掛かった場合、焼損する恐れがある。(EduCake 内部は過電流保護回路があり問題無いが、その他のマイクロコントローラでは注意が必要)。一般的には一番良いのはモータの+側を独立した電源に接続して、独立電源の GND と制御盤の GND を共通にし、更に一つのダイオードを加えて逆電流を防止する接続方法である。この接続でモータの過電流による制御盤への影響と、小電流によるモータの非稼働を防ぐことが出来る。

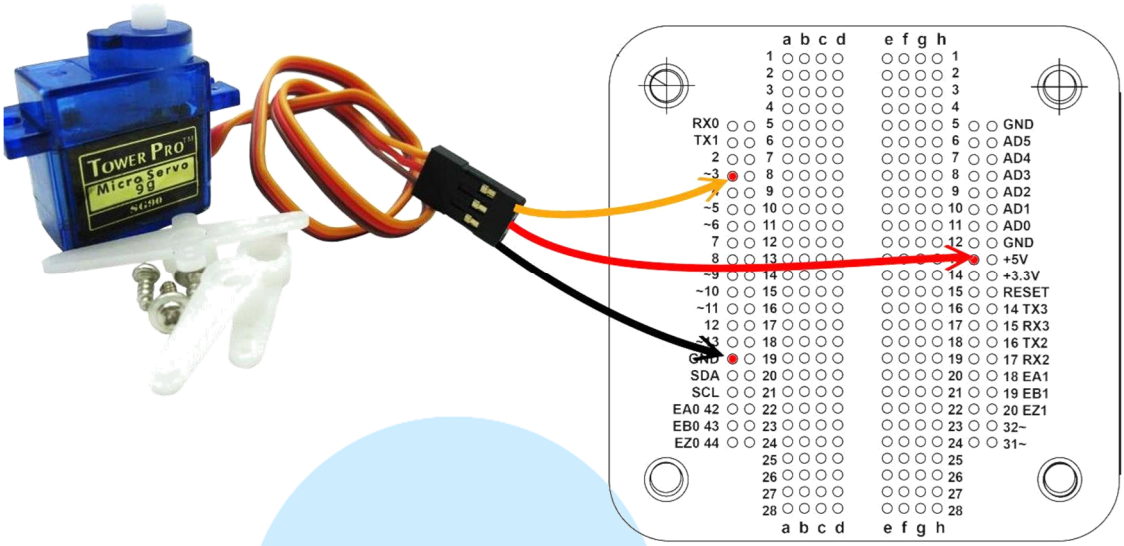

図 6. SERVO 接続図

以下のプログラムと合わせる事で、モータを左右に動かす事が出来、モータの耐久度を試すのに良く利用される。SERVO のアームに負荷を付けて、実際の出力、耐久性、消費電流、温度変化等のデータを取りモータが壊れるまで徹底的に酷使します。ロボット用途に使用する為には、このようなテストを行います。SERVO を購入する際にはこれらの注意事項の他に、SERVO の無応答幅(より小さいほうが良い)、速度、寸法、形状(機構部品の設計に関わる)、歯車の隙間等を考慮しなければならない。これらは経験値から適度な SERVO モータを選ぶ必要がある。

```
#include <Servo.h>

Servo myservo; // Servo オブジェクト作成、サーボモータ制御

void setup()
```

```
{
  myservo.attach(3);   // デジタルピン 3 に接続
}
void loop()
{
  for(int a = 30; a <= 150; a++){
    myservo.write(a); // モータ角度 30～150 度に制御
    delay(20);
  }
  for(int a = 150; a >= 30; a--){
    myservo.write(a); // 150～30 度
   delay(20);
  }
}
```

プログラム中のコメント以外に幾つかの注意事項がある。先ず一つ目のループが 30～150 度となっているが、何故 0～180 度または 0～360 度ではないかとの疑問が出るかもしれない。これは、SERVO の中の VR(可変抵抗)で値を読み取り SERVO の位置を特定しています。電子部品販売店等で可変抵抗を購入した事があればお分かりでしょうが、一般的な可変抵抗の可動部は約 120 度となっています。各 SERVO メーカの角度設定に関しては、ばらつきがあり 80 度(中心から±40 度)の SERVO もあれば 360 度の SERVO もある。もし許容されている回転角度を超えた場合、サーボがぎこちない動きをしたり過負荷で焼損したり、制御盤が焼損する場合がある(SERVO が同時にぎこちない動きをすると制御盤に流れる電流が瞬

間的に高くなる)。これは特に注意が必要な点ですので、販売店に確認するか購入後に少しずつ調整して 30～150 で確認したり、25～155 に設定を変えたり、最適な値を探してください。但し、調整するときは少しずつ値を変更してテストして下さい、例えば次の設定値を 20～160 で行ない、その後の増減値を 5 から 2 に変更して 18～162 でテストする。この様な方法で限界値を探ります。一般的には 120～150 度を超える SERVO は少なく(たった 80 度のものもある)上限値をテストする為には少しずつ値を変化させ、もし SERVO がぎこちない動きや異音がするようであれば、直ぐに電源を落としてください。

2 つ目の注意としては **delay(20);** の行です。これは SERVO の特性に関連しており、一般的に安価な SERVO は通常 50Hz/sec 仕様の PWM を用いておりパルス波長周期は 1 秒/50=20ms となる。従って位置変更のコマンドを送り出した後 20ms 待って、もう一度コマンドを送り出す必要がある。そもそも SERVO には運動スピードがり、確実に移動が行われない状態で次のコマンドが送られた場合 SERVO の運転に支障が出る可能性がある。

高価な SERVO では 65,120,200,333HZ 更にもっと高い周波数をもつ SERVO を使えば高速に応答が必要とされるジャイロスコープ等の用途でも使用出来る。このような高周波の特殊な SERVO を使う際は、高速な応答速度に対応するようプログラム中のパラメータを修正する必要がある。

KONDO というメーカが販売している SERVO では位置をフィードバックする機能がある。300μs の PWM を送り出した後、SERVO から動作完了の HIGH 信号が返信されるようです。pulseIn を使えば SERVO の現在位置を測定する事も可能です。この章では触れず、後の応用実験で触れる事にします。

4. PWM 応用 3　DC モータ制御

DC 直流モータは、制御も構造も簡単で、安価且つ入手も容易である。生活のあらゆる場所に存在し、シャッター、各種玩具、小型扇風機、電動自転車等で見る事が出来ます。EduCake は直接直流モータを制御する事は出来ません。何故なら、小さい直流モータでも少なくとも数十 mA の電流が必要となりますが、EduCake は全てのマイクロコントローラと同じく出力する電流が小さく、DC 直流モータを駆動させるほどの出力電流を持たない装置です。最も費用の掛からない対応としては、トランジスタを使った電流増幅がある。良く使われるのは 9012/9013、2N2222/2N2907、或いは TIP120 といったダーリントントランジスタである。仕様を確認して構成する抵抗を選び、逆電流防止用のダイオード等を購入する。回路をレイアウトしてブレッドボードで実験、次に半田付けでようやくモータが駆動出来る。EduCake で直流モータを制御するには以上の様な努力が必要となる。万が一、半田付け若しくは部品に問題があると、テストしてトラブルシュートの手順が必要になる。この手順は、回路を学んだことのある方には簡単であるが、電子回路の背景を持たない一般の方には接続も複雑で難しい。

ここでも簡単に入手可能な部品を使って、テストを行います。EduCake ブレッドボードを直接利用し、L293(秋葉原等で安価で購入可能)、DC モータ(同じく秋葉原などで入手可能、または壊れた玩具から抜き出しても使える)だけで 2 つの玩具モータを駆動させる図 7 の実験を行います。この図はモータを接続する際のイメージ図で、ピンの説明を行っている表と組み合わせる事で簡単に接続できるはずです。2 つのモータの正転、反転、停止、速度制御を一度に行う事が出来る。L293 の詳細な規格は以下を参照。

http://pdf.datasheetcatalog.com/datasheets/90/69628_DS.pdf

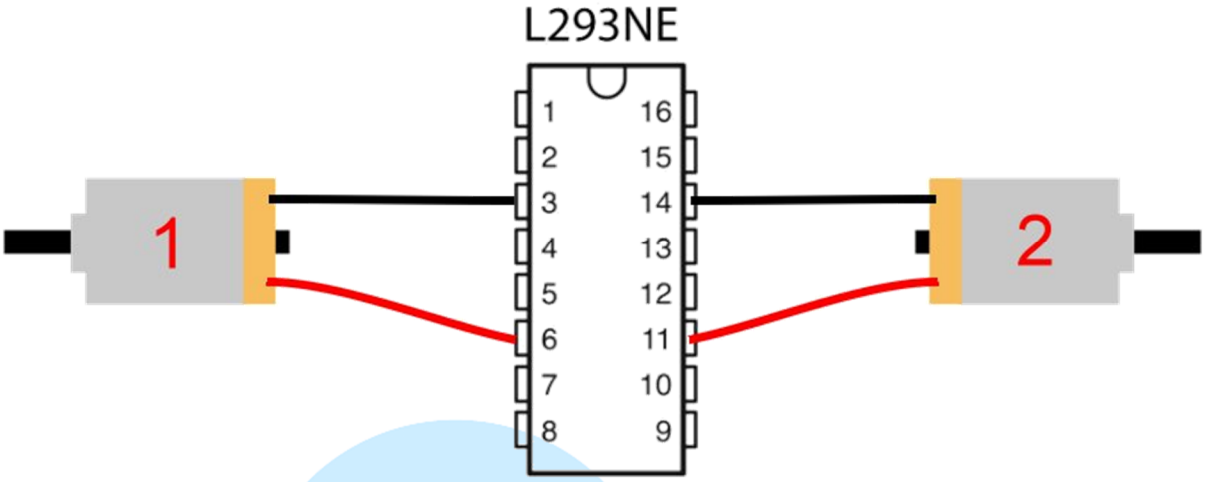

図 7. L293 接続図

L293 ピン仕様

ピン番号	規格・仕様
4、5、12、13	GND に接続
1、9	夫々はモータ１，２の速度制御 PWM 制御可能速度を入力、High 入力で最高速度回転
8	モータの電源、3V 規格であれば 5V 電源入力可。主に 4.5V～36V の電源に対応している。4.5V 以下の場合は PWM による制御を使うしかない、さもなければモータは焼損してしまう。注意が必要な点は L293 が持つ最大電流は 0.6A しかないが、玩具用のモータ等では問題は無い。
16	VCC、L293 電源、5V 入力
2、7	第 1 モータの正転、反転、停止制御
10、15	第 2 モータの正転、反転、停止制御

ピン番号を定義した後、簡単な制御プログラムを記述します。ここでは説明を簡単にする為、1 つのモータだけを制御しますが 2 つのモータを同時に制御する事も同様な操作で簡単に行えます。先ず、図 8 の様に接続します。接続ケーブルは少ししか無いので、簡単に接続できますがモータの電源の接続に注意が必要です。L293 の 8 ピンがモータの電源となりますが、EduCake の 5V 出力に直接接続しても構いません。玩具で使用す

るモータは負荷の無い時は数十 mA であり、USB 接続時 1A の供給が可能な EduCake にとっては全く問題ありません。もし大きいモータによる制御を行う場合、ケーブル接続時独立電源或いは電池に接続しパソコンなどの損壊から保護する必要がある。(EduCake は保護回路を持っているが、USB に過電流が流れるとパソコンの場合危険である)

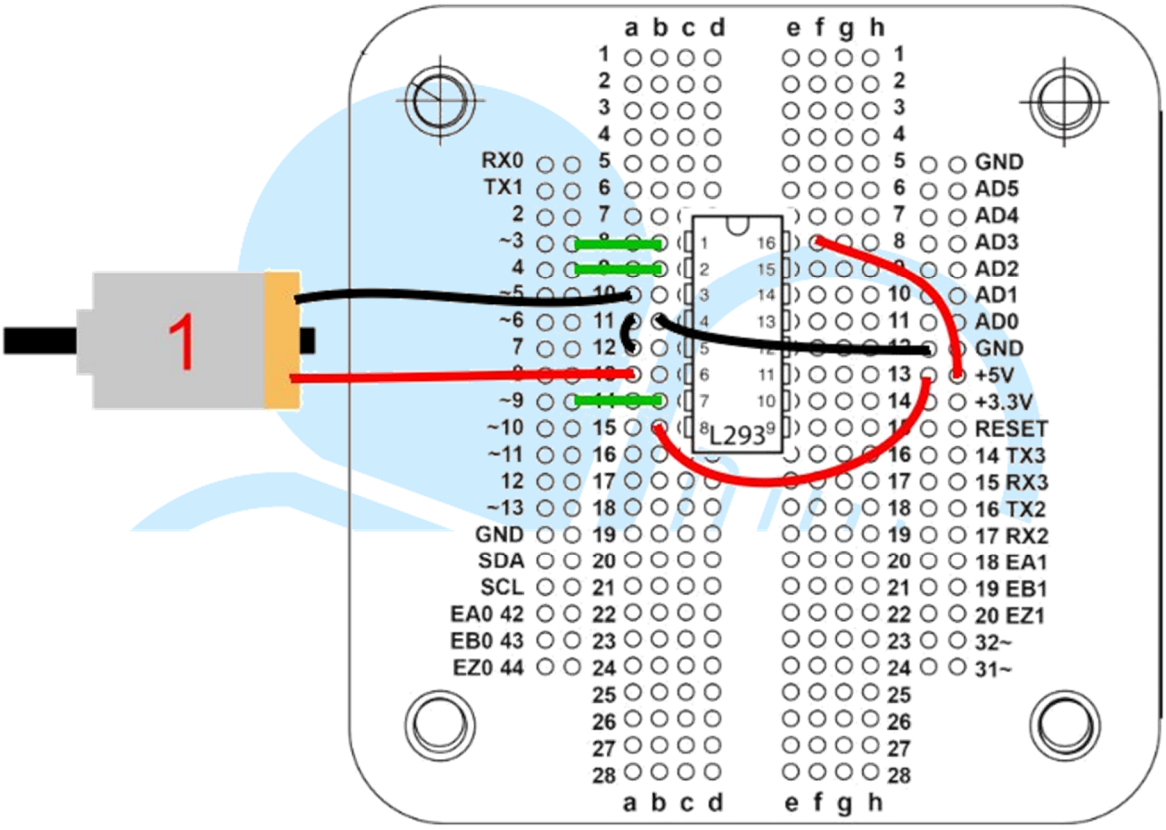

図 8. 1 つのモータ接続図

以下は、モータが最高速で左回転を 1 秒、停止、右回転 1 秒、停止、その後ゆっくりと最高速度まで加速し停止する。本動作を繰り返す

```
int M1=4,M2=9; // 2 つのピンは適当なデジタルピンを使用
int EN=3; //  必ず PWM 信号に割り当てる


void setup()
{
    pinMode(M1,OUTPUT);
   pinMode(M2,OUTPUT);
   pinMode(EN,OUTPUT);
}

void loop()
{
    int a;
   digitalWrite(M1,HIGH); //  モータ左回転
   digitalWrite(M2,LOW);
   digitalWrite(EN, HIGH);
   delay(1000);
```

```
digitalWrite(EN, LOW); // 停止
  delay(1000);
  digitalWrite(M1,LOW); // モータ右回転
  digitalWrite(M2, HIGH);
  digitalWrite(EN, HIGH);

  delay(1000);

  digitalWrite(EN, LOW); // 停止
  delay(1000);

  for(a = 0; a <= 255; a++)
  {
      digitalWrite(M1,HIGH);
    digitalWrite(M2, LOW);
    analogWrite(EN, a); // PWM を使って EN ピン番号を制御：速度制御が可能
    delay(20);
   }
      digitalWrite(EN, LOW); // 停止

      delay(1000);

    }
```

PWM 制御コマンドの説明

```
analogWrite(EN, a); // PWM を使って EN ピン番号を制御 : 速度制御が可能
```

以前 LED について説明した時と同じように、モータにも無応答幅がある。電圧が 0 から一定の数字を超えて初めて回転する。同じモータの型式でも個体差により多少の違いが発生する。従って、実際に使用するモータで 0 から電圧を変化した際にどこで回転が始まるかを調べる必要がある。もし 0.5V だとしたら PWM に対応する値は : (0.5V/5V)*255=約 25　従ってループの中の a=0 を a=25 に変更、そうするとモータは直ぐにゆっくりと加速する。玩具のモータは 3V 仕様であるが、今回 5V に接続して運用している。安全の為には PWM の値を 255 に到達させない方が良い。(3V/5V)*255=153 前後になったら停止させるように調整する事が望ましい。当然モータは使用電圧範囲を持ちますが、少しであればこの値を超えても問題ありません。負荷も掛かっていないので、少しの時間であれば 5V で運転を継続しても問題は無いでしょう。しかしながら電子部品は仕様に合わせて使う事で、安全かつ長期の利用が出来る。許容周波数、電圧を超えても動作はするが寿命に影響するため、許容範囲内で使う事をお勧めします。

将来的にもっと大きなモータを使う、インバータエアコンや車(実際に積載重量 2t トラックにも同様の回路とプログラムを応用した経験があります)。　基本的に電子回路が一緒でプログラムの原理もさほど変わらない(勿論商用には複雑となり考慮すべき点も多い)、但しその際は制御 IC を仕事効率の大きいものに変更するか、MOSFET を使う必要がある。勿論、配線部材もより太い大電流に対応する線材に変更しなければならない。

5.　PWM 応用 4　ブラシレスモータ制御

今まで説明した実験以外にもラジコン飛行機、船等でもモータを使って動かした事がありますが、これらには DC モータは使えません。何故

なら、これらの機器には瞬間出力パワーが大きく超高速回転(1 分間に数万回転)が要求され、普通の DC モータでは使えない。また、通常のブラシ付き DC モータでは、連続運転により摩耗速度が早くなる。さらに、ブラシから電磁石に流れる電流が大きくなり火花が発生し、ノイズによりリモコン操作が出来ない。リモコン操作が行われる機器では必ずブラシレスモータが使われる。シールドで影響を防ぐ手もあるが、余分な作業となりブラシレスモータを使った方が簡単である。ブラシレスモータは図 9 の通り。このモータは DC 玩具モータに比べて質感が断然と高く、表面の光沢はステンレスまたは光沢クロムメッキで加工され値段も勿論高い。3 本の信号線が出ている事に注意して下さい。

図 9. ブラシレスモータ

EduCake を用いて制御する前に、関連の規格について理解しなければいけません。通常ブラシレスモータが入手出来たら消費電流、KV 値を理解しましょう。KV とは 1 V あたりの回転数(k)。ラジコン飛行機、船、車等、対象となる装置により型式、積載重量、速度が異なり求められる KV 値が変わります。先ずは必要な KV 値にあったモータを選びます。飛行機を例に取ると、取り付けるプロペラのサイズが違うと必要とされる KV 値も異なります。同じサイズの飛行機でも小さいプロペラならば高い KV 値(相対的にトルクが小さい)のモータが必要で、大きいプロペラなら

ば低い KV 値で大電流(トルクが大きい)のモータを使い飛行に十分な推力重量比を得る必要がある。この KV 値はあくまでも参考値に過ぎず算出された出力効率(効率=電圧*電流)と飛行機の型式により左右される。同じ飛行機でも大きいプロペラか小さいプロペラを組み合わせるかで、構成が変わります。もう一つ、通常ブラシレスモータは流れる電流が大きく放熱に注意が必要です。この様な実験を行なっていると好奇心から、効率を高くなる様に改造して運転し焼損、発火等を引き起こす事もあります。

各種ラジコン用のブラシレスモータは通常 Electric Speed Controller(ESC)と呼ばれる回路を用いてコントロールされている。これは安定化電源回路とモータ回路によって構成された装置である。ここではこの回路を作る事はしない(作ろうと思えば作れなくはないが、この章からは外れるので興味ある方は資料を探してください)。市販されているものを購入しても高いものではない。図 10 上方の写真参照

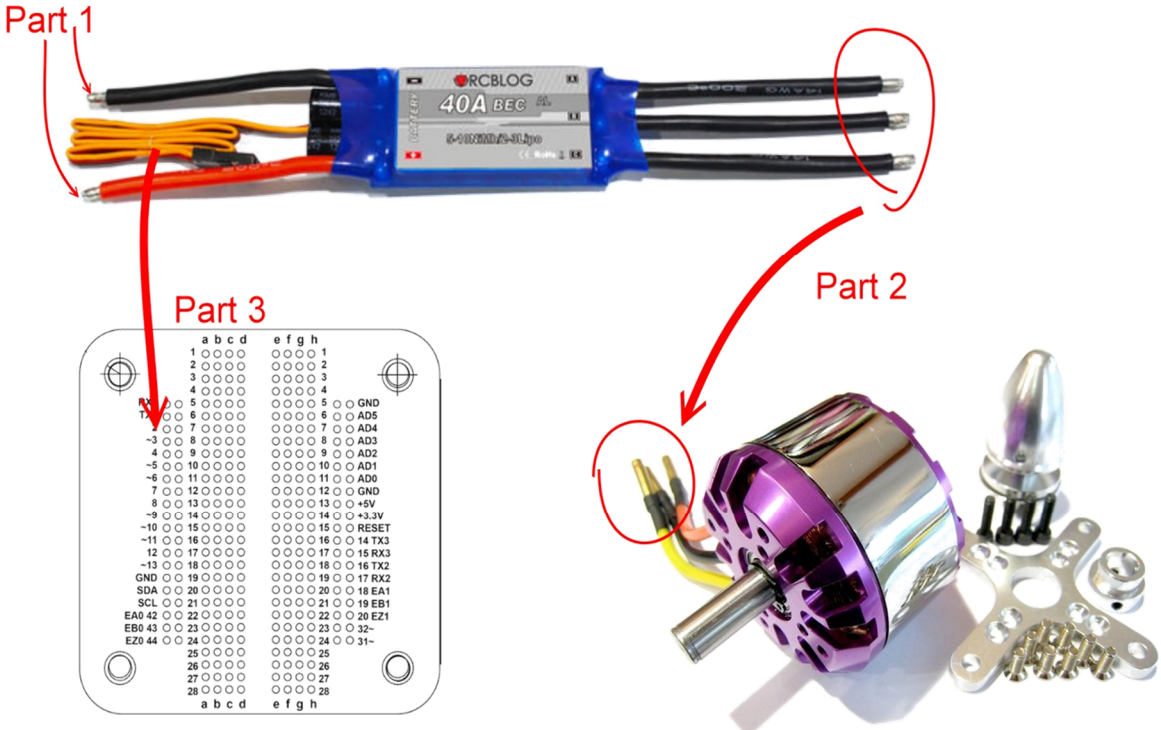

図 10.ブラシレスモータ接続

この回路は 3 つに分ける事が出来る。上方のスピードコントローラ・右のブラシレスモータ・左の EduCake に依って構成され Part1～3 で標記しています。

Part1 はモータとスピードコントローラへの電力供給です。±の極性は絶対に間違ってはいけません。(赤はプラスに、黒はマイナスに接続するのが一般的な常識である)。通常は表示されている電圧に従い供給しなければならない。(7.4V/11.2V 等の表示がある筈)　例えばこのモータが 120A の電流が必要なら十分な出力に対応するバッテリーが必要です。どの程度の大きさになるかは、模型屋さん等で確認してください。先に制御に関して説明します。大電流を必要とするモータには電気抵抗の少ない太いケーブルを使わないと焼損の可能性があります。従ってケーブルとバッテリー間のコネクタにも大電流の場合は金メッキ加工等、高価なものを検討する必要があるが、小電流であれば適当なコネクタを選べば良い。

Part2　スピードコントローラから出てきた 3 本のケーブルをブラシレスモータに接続する。接続の方法は右でも左でもどちらでも良い。A,B,C と表記されていても接続方法が変わる事で極性が変わり、モータ内部のコイルの回転方向が変わるだけである。テストしながら確認する場合、ケーブルをピン等で固定して思い通りの回転方向が得られない場合左右のケーブルを繋ぎかえれば、必要な回転方向が得られる筈である。これは 4 軸あるいは多軸の飛行機を取り扱う場合に特に重要となります。4 軸飛行機のプロペラのように慣性モーメントを相殺させるには、回転方向の違う 2 つのプロペラを組み合わせる必要がある。飛行機を飛ばすにはモータの回転方向の調整が絶対条件となる。

Part3　スピードコントローラ左側の真ん中のケーブルはサーボコントロールと全く同様に使う信号であり、接続もサーボの時と同じ接続方法となる。詳細はサーボ接続方法を参照してください。その線を EduCake と接しすれば全ての回路接続は終了です。ブラシレスモータを使い大容

量の電流で制御する場合、焼損や発火等のトラブルが発生しやすい為、回路の接続ミスが無いように何度かチェックして下さい。回路接続終了後にプログラミングを記述して制御を行います。この部分のプログラミングはサーバ制御と似ているが、スピードコントローラとサーボの動作には違いがあるのでプログラミングには注意が必要です。まず、スピードコントローラは設定して初めてつかえる。飛行機のプロペラで考えると主に 2 つの状況があり、停止、通常の速度から最速への変化となる。設定順序は以下の通り：

1. リモコンの電源を入れてスロットルを最大にし、受信機の電源を入れる。
2. この時スピードコントローラはピッという音が鳴る筈。直ちにリモコンのスロットルを最小に戻す。この操作はリモコンのスロットルの最大幅をスピードコントローラに感知させる為です。
3. この時もう一度ピッと音が鳴れば設定は OK です
4. スピードコントローラは設定を保存できるものと出来ないものがあり、保存できないものは電源を入れる都度この設定を行う必要がある。

ブラシレスモータのスピードコントローラの設定は上記の様な方法がよく見られる。但し各メーカの機能とプロセスは異なる為、取扱い説明書を読んでから行ってください。とにかく安全措置としてモータは先に接続しない事です。接続完了確認後に実際の動作を見てみましょう。プログラムは以下の様になります。

```
#include <Servo.h>

Servo myservo; // Servo オブジェクト作成

void setup()
```

```
{
  myservo.attach(3);  // デジタルピン 3 接続
}
void loop()
{
 int a = analogRead(0); // アナログピン 0 に VR を使ってモータのスピードを制御
 a=map(a,0,1024,1000,2100); // 読み取った値を先に PWM 回転角度に変換
    myservo. writeMicroseconds (a);  // 今回使用 writeMicroseconds より精密制御を行う
    delay(20);
}
```

その中の **a=map(a,0,1024,1000,2100);** 可変抵抗値が対応する PWM のパルス幅 1000~2100us を読み取れる。そして **writeMicroseconds()** を使って書き込む。このコマンドは比較的に精密且つ正確な PWM 制御を行う事が可能。正確な動作にはこのコマンドが有効で **write ()** コマンドは正確さが足りず、サーボの種類が多い、夫々の角度が対応する PWM も違い()内のパラメータの意味が無い等であまり用いられない。むしろ直接 **writeMicroseconds()**を使った方が良い。プログラムの実行後に VR を回転させる事でブラシレスモータが停止状態から速くなり確実な加速音が発生しドキドキするほど最高速度に至るまでを見る事が出来る。この制御の機能が出来たらリモコンは捨てても良い。近い将来より多くの応用に向かってラジコンカー、ラジコン飛行機をバラしパソコンと EduCake を直接接続してこのようなラジコン玩具を制御する事が出来る。