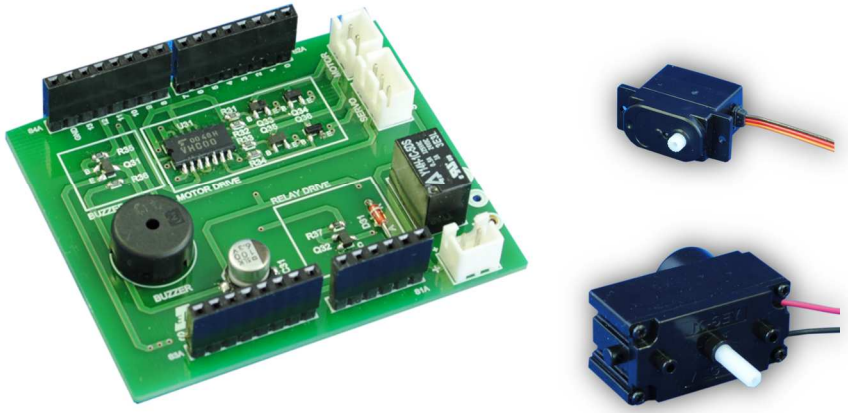
Arduino ビギナーのための

モーター,リレー,ブザー制御入門



株式会社　イーケイジャパン

はじめに

電子回路を使って何かを作ろうと思ったとき、マイコンと電子部品を使うととても便利になります。今までは多くの電子部品を並べてつないで、それぞれの動作を理解して回路を設計し、希望する機能を作る必要がありました。さらに機能を変更したいときには、大変な手間をかけて回路を修正しなければなりませんでした。しかし、マイコンの登場で高度な動きをわずか数点の電子部品で実現できるようになり、動作の修正も簡単になりました。だだし、マイコンを使いこなすには専門的な知識や言語の理解、専用の機器を用意する必要があり、ハードルが高かったのですが、最近では例えば Arduino のようなマイコンボードが登場し、比較的簡単な言語で、希望の動作を得られるようになりました。

マイコンで電子部品を制御できるようになると、思いついたことをどんどん試すことができます。しかし、「電子部品」を使うためには、どうしてもその特徴や使い方を知っておく必要があります。電子部品を知るための近道は、やはり、実際に触って動かしてみることです。さらに、電子部品の特徴やポイントを知っておくと、余計なことに悩まされることなく、自分の作りたいモノに集中できるようになります。

本製品は、実際の部品の動作を確認しながら、電子部品の特徴や解説を読むことで効率的に学習できるセットです。マイコンには Arduino ボードを利用します。説明を読みながら、実際に動作させ、結果を確認していくことで、プログラムで電子部品を動かすときのポイントがわかるようになっています。本製品があなたのアイデアを創造する手助けになればと思います。

重要

本機(SU-1204：ドライバーシールド)を動作させるためには、Arduino 基板が必要です。
Arduino 基板には多くの種類が発売されていますが、本書の説明には Arduino-UNO(R3)を使っています。
また、Arduino 基板の接続にはご使用の環境に合わせた USB ケーブルを準備してください。

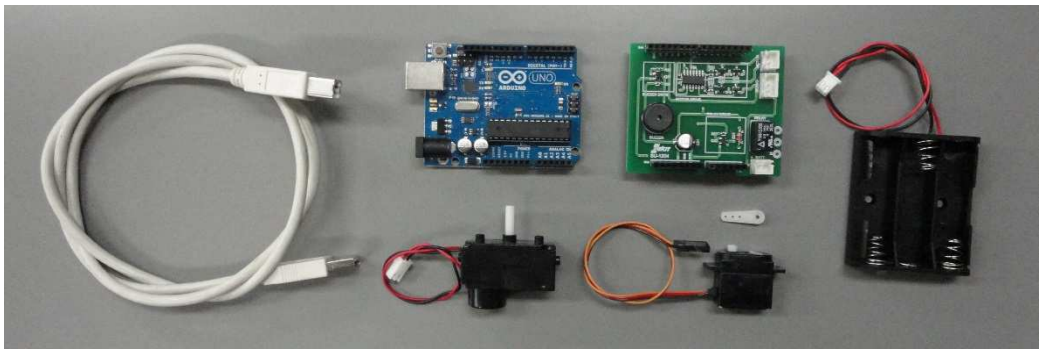
●おことわり

本書では Arduino 基板の説明や、プログラム開発環境の入手～セットアップ方法、プログラム用命令の説明については詳しく記載していません。本格的に Arduino のプログラムを学びたい場合は、Arduino のサイト http://www.arduino.cc/や、インターネット、書籍などの説明や解説を参考にして学習してください。

もくじ

0．準備

●用意するもの



（1） Arduino-UNO (R3) ドライバーシールド(SU-1204)を動作させるために必要です。

（2） ドライバーシールド SU-1204

（3） DC モーター

（4） サーボモーター

（5） 電池ボックス 単 3×3 本の電池をセットしてください。

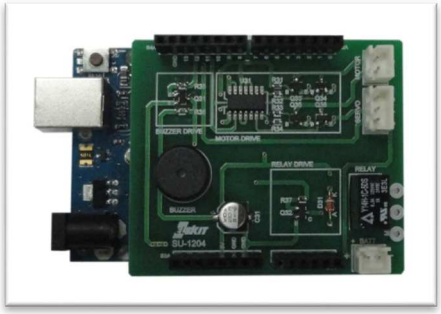
（6） USB ケーブル Arduino 基板とパソコンを接続します。

（7） パソコン Arduino プログラム開発環境がインストールされているパソコン

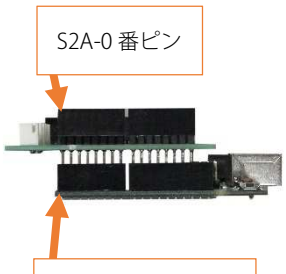
（※プログラム開発環境の入手～インストールがまだの場合は、巻末をご参照の上インストールしてください。）

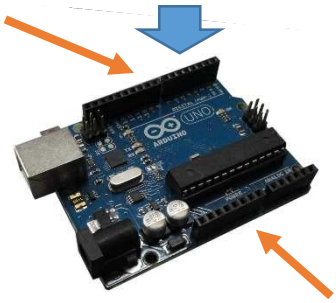
●Arduino とドライバーシールドの接続

Arduino にドライバーシールドのピンを差し込みます。
向きを確認し、ピンの位置を合わせて、ピンが曲がらないように注意して差し込んでください。

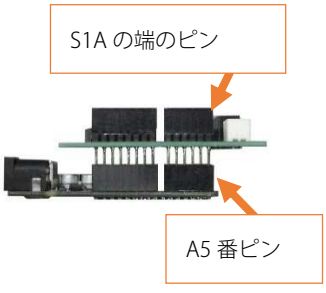


Arduino 基板の DIGITAL-0 番ピンとシールド基板の S2A-0 番ピンを合わせます。

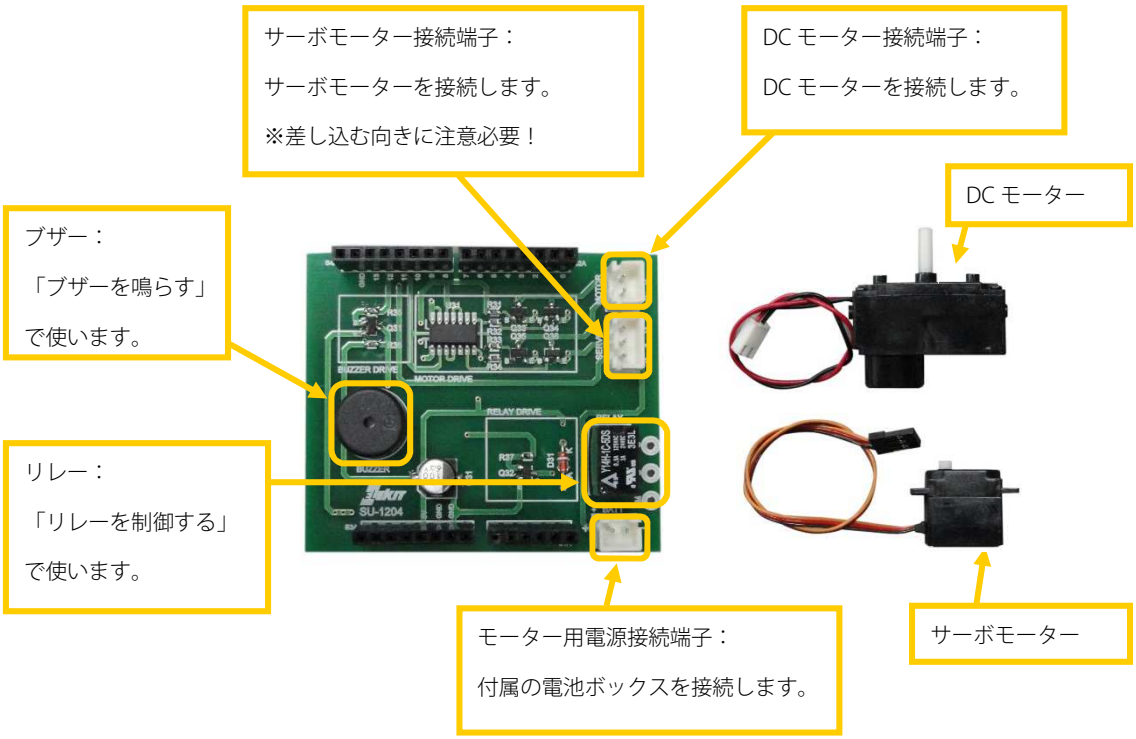




Arduino 基板の ANALOG-A5 番ピンとシールド基板の S1A の端のピンを合わせます。



●ドライバーシールドについて

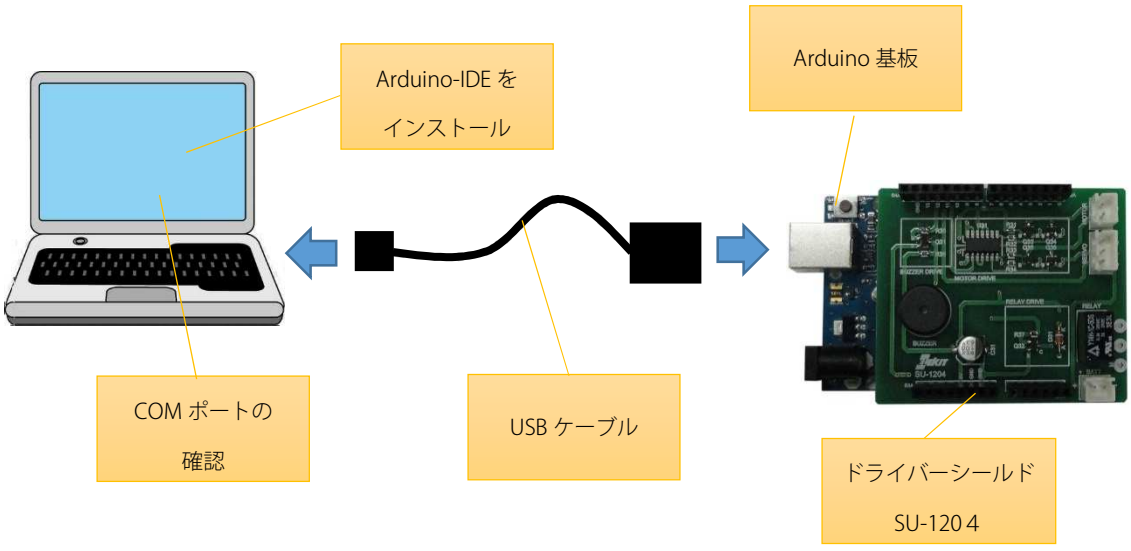


※ドライバーシールドの DC モーター、サーボモーターの電源には電池ボックスから電源を供給しています。それ以外の制御回路の電源には Arduino 基板の 5V を利用しています。

●パソコンとの接続

本書の「Arduino 基板と Arduino-IDE」を参考に、パソコンに Arduino の開発環境である Arduino-IDE をインストールし、USB ケーブルでパソコンと Arduino 基板を接続します。

COM ポートの設定・確認も忘れないようにします。



1．リレーを制御する

小さな電流で大きな電流を制御するときによく使われる「リレー」を使ってみます。リレーはスイッチとして使われることが多く、普通のスイッチは指で操作するのに対して、電気信号で ON/OFF できるので、多くの場所で使われています。ここではリレーを制御する方法について説明します。

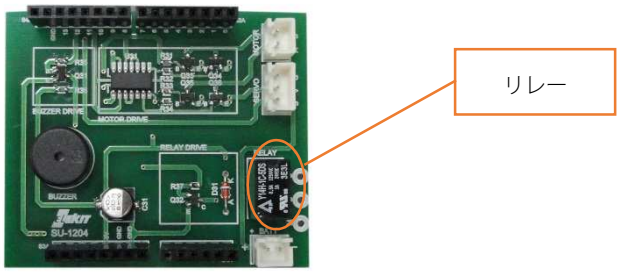
●プログラム

次のプログラムを作成して、マイコンにアップロードします。

0.5 秒ごとにリレーが ON/OFF するプログラム

```
void setup(){
   pinMode(13, OUTPUT);            //13 番ピンを出力にする
}
void loop(){
   digitalWrite(13, HIGH);         //13 番ピンから HIGH を出力
   delay(500);
   digitalWrite(13, LOW);          //13 番ピンから LOW を出力
   delay(500);
}
```

プログラムを実行すると、カチッという動作音とともにリレーが 0.5 秒ごとに動作します。



テスターを用意しましょう。

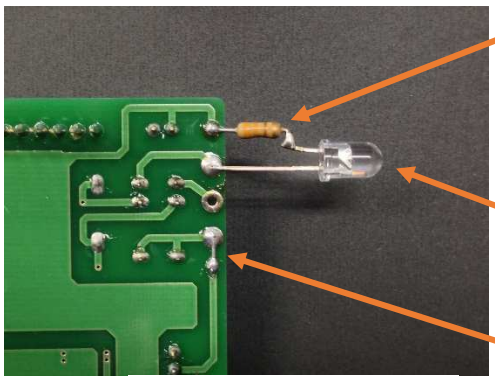
リレーに何もつないでいない状態では、リレー動作が動作しているかどうかは音でしか確認できず、ON なのか OFF なのかが分かりにくくなっています。

ぜひテスターを用意して、端子間の導通を測定することで、リレーの動作確認をしてみてください。

リレーに LED を接続してみましょう。

はんだ付けが必要ですが、比較的簡単に作業できますのでぜひチャレンジしてみてください。
リレーの出力に応じて LED が点灯、消灯するので、リレー動作の確認がやりやすくなります。

実際に配線した写真



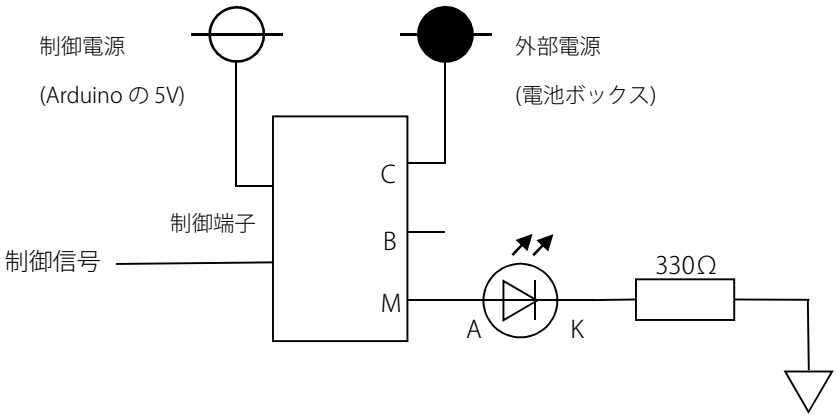
ドライバーシールド基板を
裏から見たところ

抵抗　330Ω 程度：
片側を LED の K 側へ、もう片方を写真の位置のランドに配線する。

LED：
A 側をリレーの M に配線する。

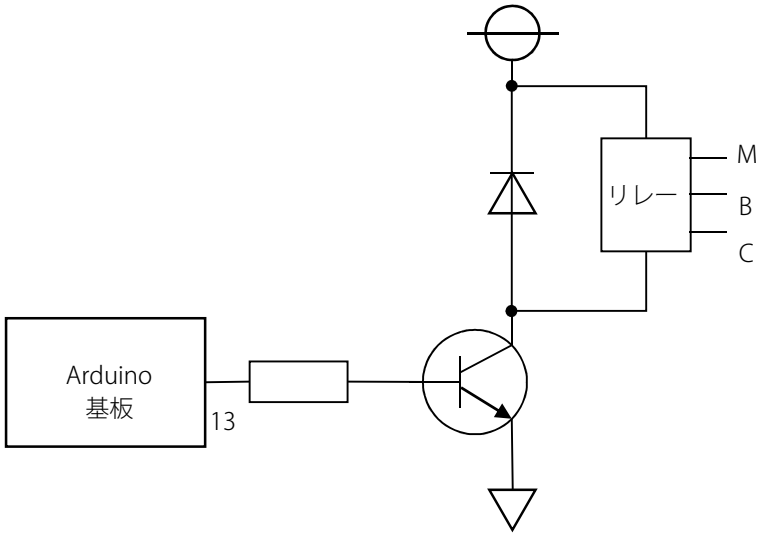
ジャンパー線：
抵抗の足の切れ端などで、リレーの C と、写真の位置のランドに配線する。

配線した状態を回路図で示すと以下のようになります。



正しく配線できると、リレーが ON したときに、LED が点灯し、リレーが OFF しているときは消灯します。

●回路



Arduino の 13 番ピンに抵抗とトランジスタを介してリレーがつながっています。

●解説

まず、`setup` の中で `pinMode` 命令を使って 13 番ピンを出力ポートに宣言します。これは Arduino 基板の電源を入れたときに全てのポートが入力ポートになっているためです。

`loop` の中では、まず 13 番ピンを H にして、`delay` 命令で 0.5 秒待ったあとで 13 番ピンを L にし、再び 0.5 秒待つことを繰り返します。

13 番ピンから H が出力されるとトランジスタが ON になり、リレーの制御端子に電流が流れます。リレーの制御端子に電流が流れるとリレーが ON し、図中の C と M が導通(つながる)します。

●使い方、ポイント

・リレーとは

リレーとは簡単に言うと電磁石で ON・OFF できるスイッチのことです。継電器と言われることもあります。大きく 2 つの使い方があり、1 つは、小さな信号で大きな電流を制御したいとき。もうひとつは、信号を遠く離れた場所に伝えるときに、距離により段々減衰して弱ってしまう信号を元気にする使い方です。遠く離れた場所まで次々と信号を伝えていくのでリレーといわれます。

リレーは動作の方法の違いにより、以下のように分けることができます。

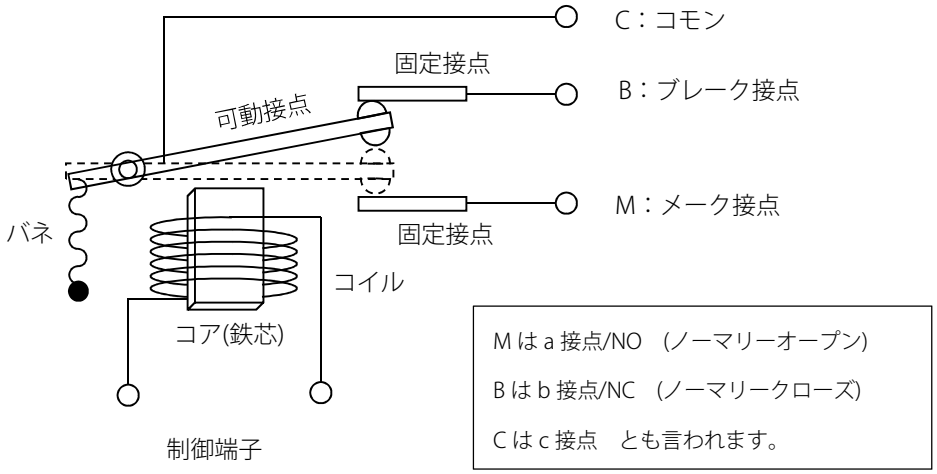
有接点リレー	コイル(電磁石)やバネなどの機械的な部品を動かして接点を ON・OFF する。
無接点リレー	電磁石でなく、半導体の性質を利用して、電流を ON・OFF します。 ソリッドステートリレー(SSR)と言うほうが一般的です。

無接点リレーは、動作音がせず、機械的に動く部品もないので接触不良が発生しないなどのメリットがあ

りますが、取り扱う電流が大きな無接点リレーは価格がとても高くなるので、大きな電流を取り扱う場合は有接点リレーが、小さな電流を制御する場合には、無接点リレーを使うことが多いようです。

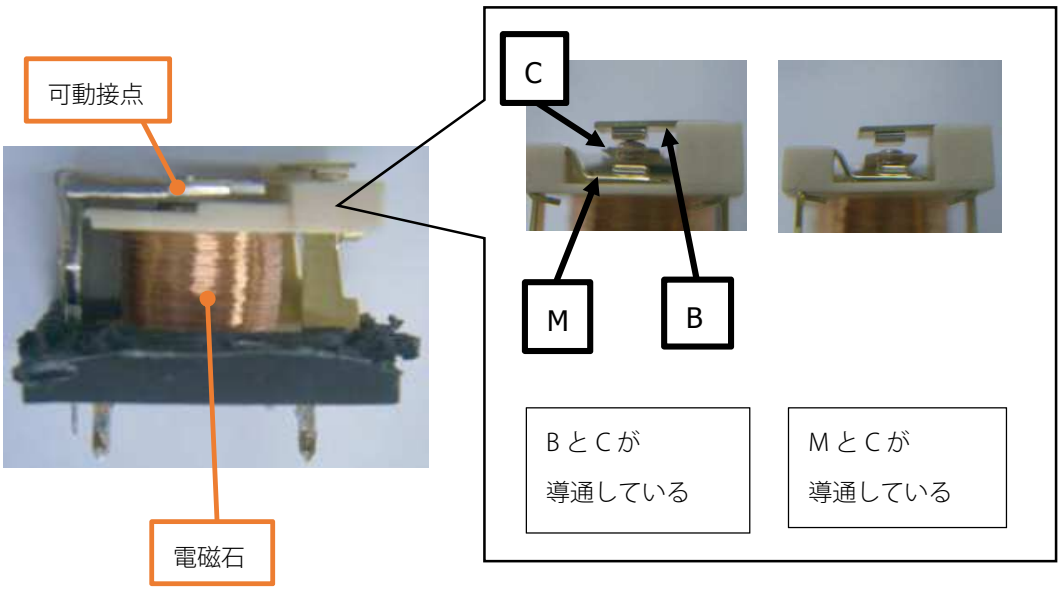
・リレーのしくみ（有接点リレー）

リレーは図のように、接点となる金属の板と、その板を動かすための電磁石で構成されています。



制御端子に電流が流れていないときは、バネの力で、可動接点は固定接点の B 側にくっついています。制御端子に電流が流れると、コイルと鉄芯の働きで電磁石になり、可動接点が電磁石に引き寄せられて、可動接点が固定接点の M 側にくっつきます。

リレーを分解したところ



・リレーの選び方

リレーを選ぶときは、接点側、コイル側のそれぞれで決められている定格、仕様を確認します。

■接点側のポイント

まず接点側で最も注意しなければいけない定格は、定格電流です。定格電流は、交流(AC)と直流(DC)のそれぞれの場合で決められています。リレーに接続する回路に合った定格を選びます。

リレーに定格電流以上の電流が流れると、接点が焼きついたり、逆に接点が破損して全く電流が流れなくなります。ON する瞬間(動き始める瞬間)に通常動作時よりも遥かに多くの電流が流れてしまうモーターなどのような誘導負荷を接続する場合は、特に定格をオーバーしないように注意して選びます。

あるリレーのデータシートの例（１）

項目			特性		備考
			基準形	高感度形	
接点	接点構成		1a（1メーク）		
	接点材質		銀合金		
	接点形状		単子接点		
	接点接触抵抗（初期値）		70mΩ以下		1A　6VDCにて
	接点定格		5A　30VDC 5A　250VAC		抵抗負荷にて
	最大通電電流		5A		
	最大開閉電力		1,250VA / 150W		
	最大開閉電圧		250VAC　150VDC		
	最大開閉電流		5A		
	最小適用負荷		10mA　5VDC		参考値*
コイル	定格消費電力		300mW	200mW	周囲温度20℃にて
	感動消費電力		130mW	113mW	周囲温度20℃にて
	使用周囲温度		-40℃～+70℃		結露・氷結しないこと
時間	動作時間		8ms以下（バウンス含まず）		コイル定格電圧印加にて
	復帰時間		4ms以下（バウンス含まず）		
寿命	機械的		500万回以上		
	電気的		10万回以上		接点定格負荷（抵抗負荷）
その他	耐振動性	誤動作	10～55～10 Hz 片振幅0.825mm		直交する3軸方向励磁無励磁にて計6サイクル
		耐久	10～55～10 Hz 片振幅2.5mm		直交する3軸方向無励磁にて計6時間
	耐衝撃性	誤動作	100m/s^2（11±1ms）		直交する3軸方向励磁無励磁にて計36回
		耐久	1,000m/s^2（6±1ms）		直交する3軸方向無励磁にて計18回
	外形寸法（縦×横×高）/質量		10.0×17.5×12.5mm / 約4.3g		

また接点側で気をつけたいのは接点の最小適用負荷です。これは、接点が正常に ON、OFF するために必要な電流がどのくらいであるかの値です。

リレーに接続された機器の負荷が軽い（＝機器が使用する電流が少ない）場合は、接点を流れる電流が少なく、この最小適用負荷を満たさずに、リレーが ON しても電流が流れないことがあります。

リレーの接点は金属でできていますので、その表面にはさびのような膜ができています。接点が ON つまり接点どうしが接触しても、その膜が邪魔をして電流が流れないことがあり、その膜を確実に破壊するのに必要な電流が最小適用負荷として記載されています。

一般的に大きな電流を ON/OFF できるリレー、つまり定格電流が大きなリレーほど最小適用負荷が大きくなります。定格が大きいからといて何にでも使うとトラブルの元になります。

■コイル側のポイント

コイル側で注意しなければいけない定格は、コイル定格電圧です。コイル定格電圧は、コイルを安定して動作させるために必要な電圧です。例えばコイル定格電圧 12V の仕様のリレーを 5V で動作させることはできません。逆に 5V 用のリレーを 12V の電源で使用することもできません。使用する回路にあったコイル定格電圧であることを確認します。

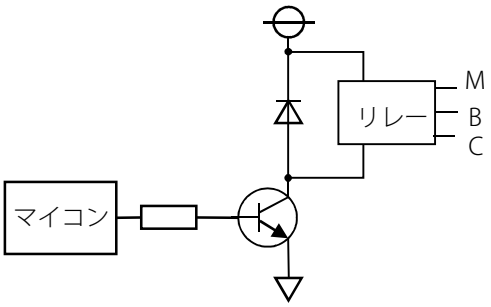
あるリレーのデータシートの例（2）

タイプ	コイル定格電圧 (VDC)	コイル定格電圧記号	コイル抵抗 (Ω) ±10%	感動電圧* (VDC)	開放電圧* (VDC)	定格消費電力 (mW)
基準形	3	3	30	1.98	0.15	300
	5	5	83.3	3.3	0.25	300
	6	6	120	3.96	0.3	300
	9	9	270	5.94	0.45	300
	12	12	480	7.9	0.6	300
	18	18	1,080	11.9	0.9	300
	24	24	1,920	15.8	1.2	300
	48	48	7,680	31.7	2.4	300

次に確認しておきたい定格は、コイル抵抗です。コイル抵抗はその名前の通りコイルの抵抗値で、コイル側にどのくらいの電流が流れるのかを知るために必要で、コイルに流れる電流に対し、十分な制御ができる回路を作る必要があります。一般的にコイル抵抗は小さく、マイコンで直接制御すると、マイコンに電流が流れすぎてマイコンを破損してしまう恐れがありますので、殆どの場合リレードライブ回路を使います。

・リレードライブ回路

以下に基本的なリレードライブ回路を示しています。



トランジスタを使ってコイルに電流を流します。
マイコンはトランジスタの ON/OFF を行います。

リレーを制御する場合は、逆起電力から回路を保護するため、逆起電力を低減するためのダイオードを使うのが一般的です。ダンパーダイオードといわれることもあります。

逆起電力とは、コイルに流れる電流を OFF した瞬間に、コイルが今まで加わっていた電流を維持しようとして発生する電圧のことです。この逆起電力が回路側に伝わり、誤動作の原因となります。

ダンパーダイオードを入れることで、逆起電力によって発生した電圧を電源に流すことができ、トランジスタ側に掛かる負担が少なくなります。

・リレーの出力

電気店などで入手できるリレーのほとんどは、接点は単なるスイッチで、接点から電圧が出てくることはありません。これを無電圧接点出力といいます。リレーを初めて使う人には、リレーの接点から電圧が出てくると思っている人が多いようです。繰り返すようですが、リレーは指で操作する必要がないだけで、単なるスイッチです。

コラム 1

「誘導負荷」　「抵抗負荷」とは

抵抗負荷とは電流を流した瞬間から表示されている消費電流（例えば 100W の機器であれば 100V をかけたときに 1A の電流）が流れる理想的な負荷のことです。また、誘導負荷というのはソレノイドやモーターなどのコイル成分を持った負荷のことで、これらは電源を投入した瞬間には定常的に流れる電流の 10 倍程度の電流が瞬間的に流れます。

2．ブザーを制御する

電気信号を「音」に変換して人間の耳で聞こえるようにする機器には、イヤホンやブザー、スピーカーがあります。これらの機器には多くの種類があり、その分類や使う場所によって呼び方もいろいろあります。電子工作では特にブザーの一種である圧電ブザーが多く使われます。ここでは本機にも使用している圧電ブザーについて説明します。

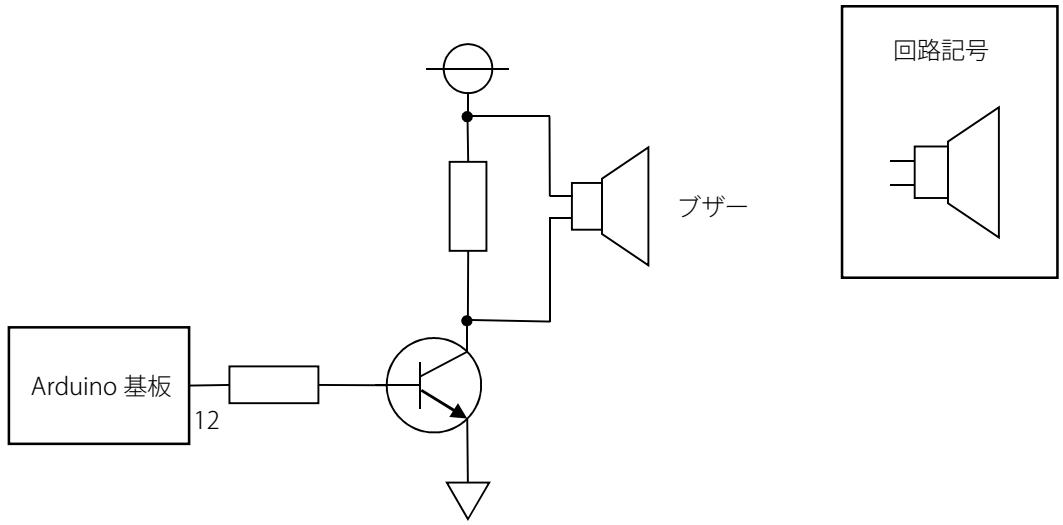
●プログラム 1

次のプログラムを作成して、マイコンにアップロードします。

ブザーから音を出すプログラム。

```
void setup(){
   pinMode(12,OUTPUT);          //ブザーの接続されたピンを出力にする
}
void loop(){
   digitalWrite(12, HIGH);      //ブザーのピンを H にする
   delay(1);
   digitalWrite(12, LOW);       //ブザーのピンを L にする
   delay(1);
}
```

●回路

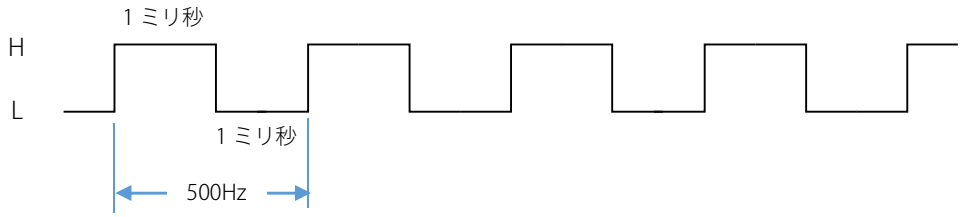


ブザーはトランジスタ、抵抗を介して、Arduino 基板の 12 番ピンに接続されています。

●解説

圧電ブザーは Arduino 基板の 12 番ピンに接続されていますので、**pinMode** で 12 番ピンを出力にします。

digitalWrite 命令を使って、12 番ピンの出力を 1 ミリ秒ごとに H→L と切り替えます。



H が 1 ミリ秒、L が 1 ミリ秒を繰り返すので、周期は 2 ミリ秒です。

周期が 2 ミリ秒ですので、周波数は 500Hz ということになります。

※周波数と周期の関係は以下の計算式で求めます。

周波数 f (Hz)　＝　1　÷　周期 T(秒)

●使い方、ポイント

・特徴

圧電ブザーはそのしくみから、人間の声や音楽などを再生するのには向いていません。そのかわりに小型であることを生かして、電子機器の動作確認音や操作音の発生に多く使われます。

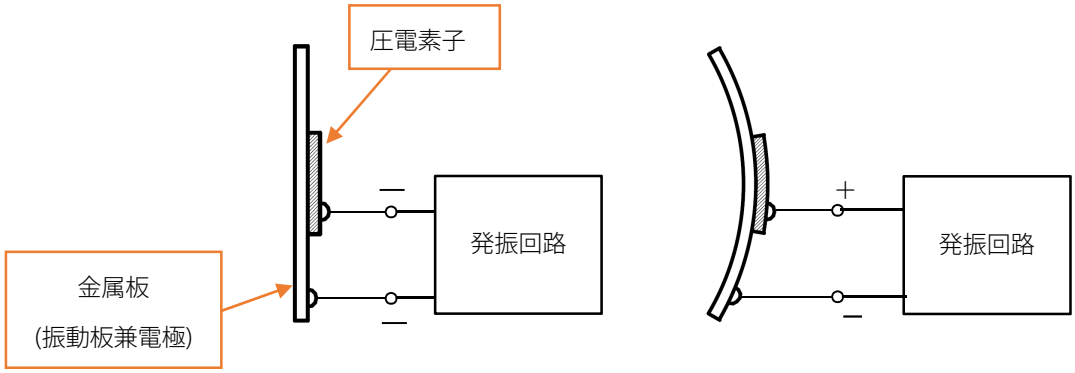
圧電ブザーは、圧電サウンダーや圧電スピーカーとも言われます。音を発する部品は、同じ内容の部品であっても、部品メーカーによって名称が違っていたりしますので、部品を探したり、調べたりするときは色々な呼び方がされていることを知っておきましょう。

・圧電ブザーのしくみ

圧電素子という、電圧を加えると変形する素子を金属板に貼り付けた構造になっています。

圧電素子が変形することで金属板も一緒に変形し、その変形により空気を振動させて音を鳴らします。

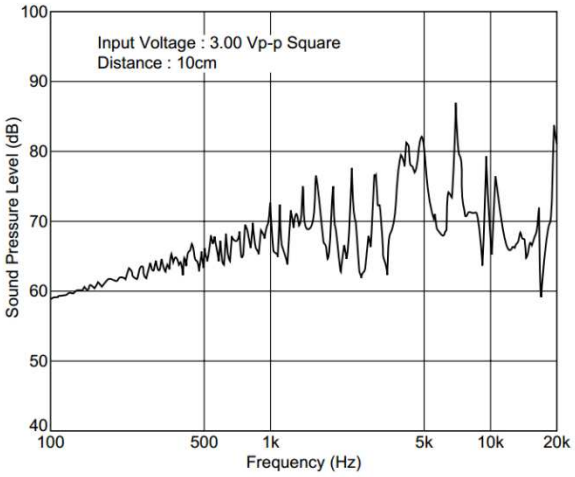
人間が聞くことができる音の周波数は 20Hz～20kHz と言われていますので、金属板をその範囲で振動させなければなりません。そのため、圧電素子をつかったブザーには発振回路が必要となります。



電圧を加えると素子が変形する。

・周波数応答

圧電ブザーには発振回路が必要ですが、その発振の周波数によって音の大きさが変わります。
例えば次の図は、ある圧電ブザーの周波数レスポンス(応答)のグラフです。



5kHz の周波数で動作させたときの音圧レベルと、1kHz で動作させたときの音圧レベルを比べると約 10db 違っています。10db の違いは人間の感覚では 3 倍程度音の大きさが違うことになります。
できるだけ大きな音で鳴らしたいというときは、データシートを確認して最適な周波数で動作させるようにします。

・音階

圧電ブザーは前述した周波数応答により音圧レベルの違いはありますが、ある程度の範囲内であればいろんな周波数の音、つまり音階を鳴らすことができます。ちょっとしたメロディならば圧電ブザーで奏でることも可能です。
「音階」つまりドレミファソ・・・を圧電ブザーで鳴らすためには、鳴らしたい音階の周波数で圧電ブザーに電圧を加えます。

音階と周波数の例

	Hz		Hz		Hz		Hz
ド	264	ド	528	ド	1056	ド	2112
レ	294	レ	588	レ	1176	レ	2352
ミ	330	ミ	660	ミ	1320	ミ	2640
ファ	352	ファ	704	ファ	1408	ファ	2816
ソ	396	ソ	792	ソ	1584	ソ	3168
ラ	440	ラ	880	ラ	1760	ラ	3520
シ	495	シ	990	シ	1980	シ	3960

コラム 2

ブザーの種類

本書では圧電素子を使った発音体のことをひとくくりに電子ブザーと言っています。しかし、
一般的には以下のように種類を呼び分けていることが多いようですので参考にしてください。

ブザー
発振回路が内蔵されており、電池(電源)をつなぐだけで音が鳴るもの。
回路が入っているので圧電スピーカーに比べて少し大きめであることが多い。

圧電スピーカー、圧電サウンダー
発振回路が内蔵されていないので、外部に発振回路が必要なもの。
薄型のもの、小型のもの、大口径のもの、ケースに入っていないものなど多種多様です。
本機で使用しているブザーはこのタイプです。

圧電ブザーで音階を出せることが分かりましたが、具体的にはどのように制御すればよいのでしょうか。
次のプログラムでは、実際に圧電ブザーに音階の周波数を与えて音階を鳴らしてみます。

●プログラム 2

圧電ブザーで、音の長さが 0.5 秒のドレミを繰り返すプログラム。
ドは 1056Hz、レは 1176Hz、ミは 1320Hz を使います。

```
void setup(){
  pinMode(12,OUTPUT);                    //ブザーの端子を出力にする
}
void loop(){
  for(int i=0; i<528; i++){              //ドの音(1056Hz の周波数)を 528 回繰り返す
    digitalWrite(12,HIGH);
    delayMicroseconds(473);
    digitalWrite(12,LOW);
    delayMicroseconds(473);
  }
  for(int i=0; i<588; i++){              //レの音(1176Hz の周波数)を 588 回繰り返す
    digitalWrite(12,HIGH);
    delayMicroseconds(425);
    digitalWrite(12,LOW);
    delayMicroseconds(425);
  }
  for(int i=0; i<660; i++){              //ミの音(1320Hz の周波数)を 660 回繰り返す
    digitalWrite(12,HIGH);
    delayMicroseconds(379);
    digitalWrite(12,LOW);
    delayMicroseconds(379);
  }
 }
```

H 期間が 473u 秒、L 期間が 473u 秒とすることで、1056Hz の周波数をつくっています。

●解説

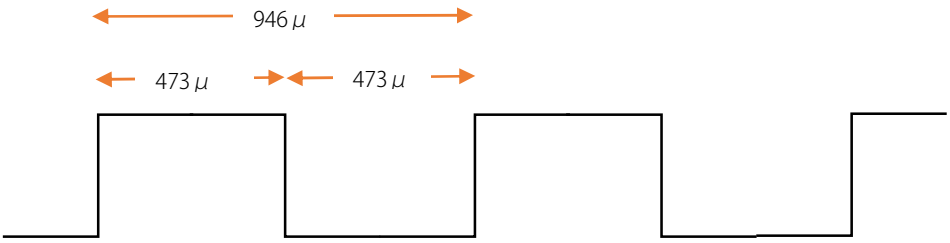
ブザーから音を出す基本的な方法はプログラム 1 と同じですが、「音階」を「決められた時間鳴らす」には計算が必要になります。「1056Hz のド」を鳴らす方法を例に説明します。
まず希望する周波数をつくる考え方です。

1056Hz の周波数の周期は、

周期 T(秒)＝ 1 ÷周波数(Hz)　の計算式から、約 0.000946 秒(946μ秒)

となります。

考えやすくするために H 期間と L 期間は同じ時間とすると、H 期間を 473μ秒、L 期間を 473μ秒出力することを繰り返せば、1056Hz の周波数をつくれることになります。



マイクロ秒単位でポートの H/L を制御するために、Arduino の `delayMicroseconds(時間)` 命令を使って次のように書きます。

```
digitalWrite(12,HIGH);
delayMicroseconds(473);
digitalWrite(12,LOW);
delayMicroseconds(473);
```

ドの音の周波数の
1 回分を出力する部分

次に「決められた時間鳴らす」ことを考えます。

先ほどの計算より、1056Hz のドの音の周期は 946μ秒です。この 946μ秒を何回繰り返せば 0.5 秒になるかと考えればよいので、0.5(秒)÷0.000946(秒)＝528.5 となり、約 528 回繰り返せば 0.5 秒になることが分かります。繰り返し処理したいので、`for` 文を使って次のように書きます。

```
for(int i=0; i<528; i++){
    ドの音（周波数）
}
```

このように、目的の音階の周波数から周期(時間)を計算、鳴らす時間から繰り返し回数を計算しプログラムすることで、1 つの音を出すことができます。

プログラム 2 では、ド・レ・ミと鳴らしています。音階に応じて順番に、周期と、繰り返し回数の値が変わっていることを確認してみてください。

●使い方、ポイント

・tone 関数

マイコンから音階を出す原理はこれまでの説明で分かったと思いますが、実際にメロディを作るときにこんな計算作業を行うと思うとかなり大変でメロディ作りが面倒です。Arduino にはそんな音階作成作業を劇的に軽減してくれる「tone 関数」が用意されています。

tone 関数は、

tone(ピン番号, 周波数, 鳴らす時間) と書きます。

※鳴らす時間は省略することもできます。省略した場合は、次の tone 命令まで音が鳴り続けます。

鳴り続ける音を止めるときは、 noTone(ピン番号) 命令を使います。

実際の使い方を見てみましょう。

まず、鳴らす時間を省略した場合で、ドレミと音を出します。

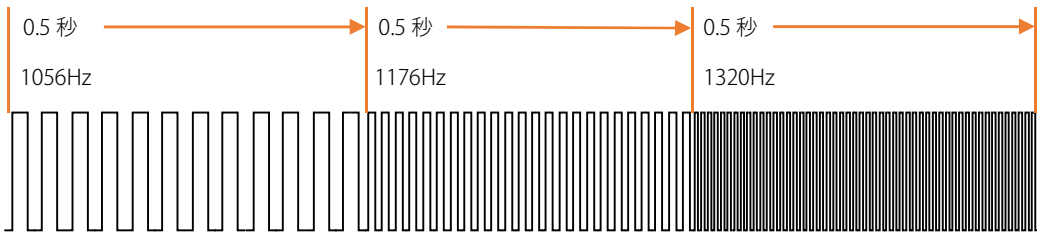
tone 関数を使う場合は、そのピンを pinMode で出力にする必要はありません。

```
void setup(){
  tone(12, 1056);
  delay(500);
  tone(12, 1176);
  delay(500);
  tone(12, 1320);
  delay(500);
  noTone(12);
}
void loop(){
}
```

このプログラムは setup の中に書いたので、1 回だけ実行します。

圧電ブザーが接続された 12 番ピンから、まず tone 関数で 1056Hz を出力、delay で 0.5 秒待ったあとに、次の tone 命令で 1176Hz を出力・・・という具合にド～ミを鳴らします。

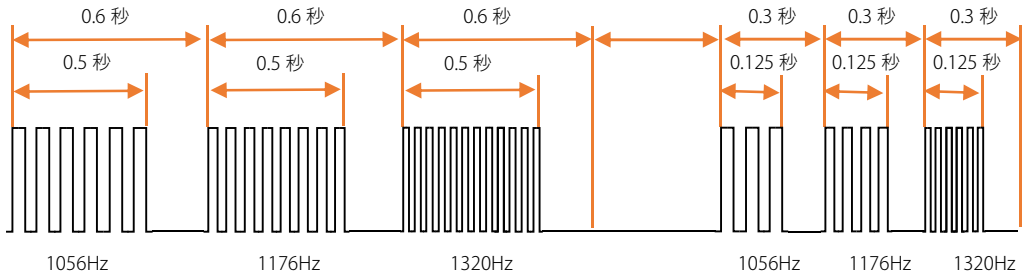
12 番ピンの出力の状態を図にすると以下のようになります。



実際にメロディを作成するときは、4 分音符や 2 分音符なども簡単に設定できる方が便利です。そんなときに、「鳴らす時間」を設定します。鳴らす時間はミリ秒で設定します。
全分音符を 1 秒とすると、2 分音符は 500 ミリ秒、4 分音符は 250 ミリ秒、8 分音符は 125 ミリ秒と設定すれば希望の音符を得られます。
次のプログラムが実際の使い方です。ドレミを 2 分音符で鳴らし、1 秒待ったあとに、8 分音符でドレミを鳴らします。

```
void setup(){
  tone(12,1056,500); delay(600);          //2 分音符のドを設定鳴り終わるまで待つ
  tone(12,1176,500); delay(600);          //2 分音符のレを設定鳴り終わるまで待つ
  tone(12,1320,500); delay(600);          //2 分音符のミを設定鳴り終わるまで待つ
  delay(1000);
  tone(12,1056,125); delay(300);          //8 分音符のドを設定鳴り終わるまで待つ
  tone(12,1176,125); delay(300);          //8 分音符のレを設定鳴り終わるまで待つ
  tone(12,1320,125); delay(300);          //8 分音符のミを設定鳴り終わるまで待つ
  noTone(12);
}
void loop(){
}
```

12 番ピンの出力の状態を図にすると以下のようになります。



コラム３

tone 関数で鳴らす時間を設定しているのに、音が鳴らない。

まず、「ド」を 1 秒間鳴らすだけのプログラムは、

```
void setup(){
    tone(12,528,1000);
}
```

と書けば OK です。しかし、ドレミと続けて音を鳴らそうとしたときに、よくやってしまう間違いに、delay を入れないことがあります。

```
void setup(){
    tone(12,528,1000);
    tone(12,588,1000);
    tone(12,660,1000);
    noTone(12);
}
```

と書けば、1 秒間ずつド、レ、ミと鳴りそうですが、このプログラムはうまく音が鳴りません。tone 関数の中で「鳴らす時間」を設定しているので 1 秒間鳴りそうな気がしますが、tone 関数は正しくは、「鳴らすための値を設定して出力を始める」です。つまり、上のプログラムでは、ドを出力し始めてもすぐに次の tone 命令が実行されるので、レの出力が始まります。しかしここでもすぐに次の tone 命令があるのでミの出力が始まってしまいます。ミの出力が始まるとすぐに次の命令を実行し、次の命令は noTone なので出力を停止してしまいます。結果としてほとんど音が出る暇もなくプログラムが終わってしまうのです。

音をしっかり出すためには、プログラム 2 にあるように、delay を使って音を鳴らす時間以上に設定しないと入れないと意図したとおりの音が出ないことを覚えておきましょう。

コラム４

tone 関数を使うときの注意点。

tone 関数は Arduino のマイコン内部の PWM 機能で実現しています。つまり tone 関数は PWM 機能の一部を占有してしまいます。

Arduino 基板で PWM 機能を使えるピンは、3,5,6,9,10,11 の 6 ピンなのですが、tone 関数を使うと noTone 命令を実行するまで 3 番ピンと 11 番ピンの PWM 機能が使えなくなります。

また、tone 関数に設定する周波数には 31Hz 以下の値は設定できません。これも Arduino の PWM 機能が 31Hz 以下の数値を設定できないことが理由です。

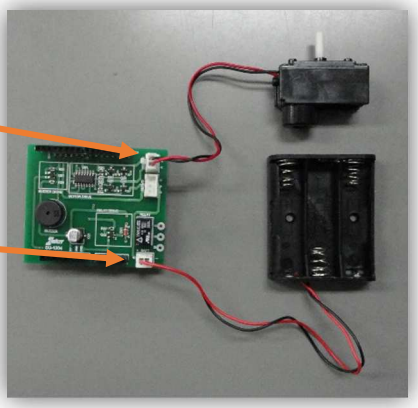
3．DC モーターを制御する

身の回りの動くものの多くにモーターが使われています。車の模型、展示台や表示の回転、電話の着信を知らせるバイブレーション機能にもモーターが使われています。これらモーターは単にフルパワーで回転させるだけでなく、ゆっくり回したり、バイブレーションのように緩急をつけて回したりすることで、その用途が広がります。ここでは乾電池などの直流電源をつなぐと回りだす DC モーターをマイコンで制御する方法を説明しています。

●DC モーターを使う準備

ドライバーシールドの MOTOR の端子に、
DC モーターを接続します。

ドライバーシールドのモーター用電源端子に、
電池ボックスを接続し、電池をセットします。

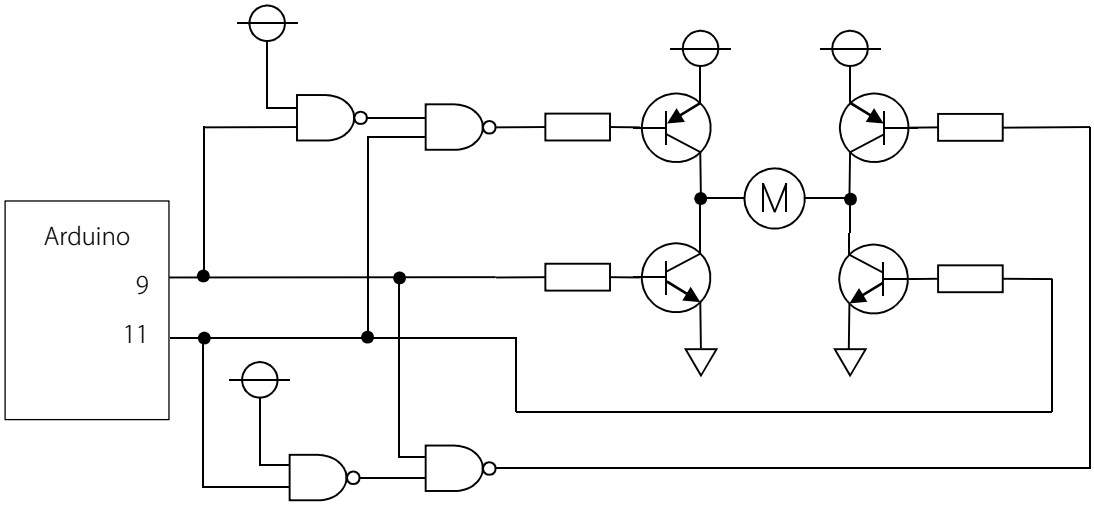


●プログラム 1

次のプログラムは、MOTOR に接続したモーターを 1 秒ごとに正回転→逆回転→停止させることを繰り返します。

```
void setup(){
  pinMode(9, OUTPUT);                    //モーター制御ピンを出力にする
  pinMode(11, OUTPUT);
}
void loop(){
  digitalWrite(9, HIGH);  digitalWrite(11, LOW);    //モーターを反転
  delay(1000);                                      //1 秒待つ
  digitalWrite(9, LOW);  digitalWrite(11, HIGH);    //モーターを正転
  delay(1000);                                      //1 秒待つ
  digitalWrite(9, LOW);  digitalWrite(11, LOW);     //停止
  delay(1000);                                      //1 秒待つ
}
```

●回路図



●解説

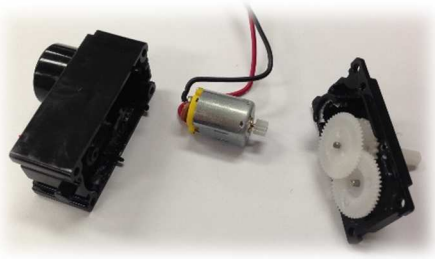
`setup` の中で、モーターの ON/OFF を制御する端子、9 番ピンと 11 番ピンを出力に設定します。

`loop` の中で、9,11 番ピンの出力を変えて、モーターの正転、停止、反転を行います。

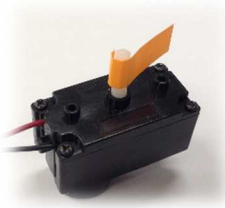
本機の回路では、9 番ピン、11 番ピンの出力の状態とモーター動作の関係は次の表のようになっています。

9 番ピン	11 番ピン	モーター動作
`LOW`	`LOW`	停止
`LOW`	`HIGH`	正転（反転）
`HIGH`	`LOW`	反転（正転）
`HIGH`	`HIGH`	ブレーキ

DC モーター単体では回転が速すぎて実験の確認がやりづらいため、本機に使用している DC モーターにはギアが一体となったギアボックスタイプを使用しています。ギアボックスの中には一般的な DC モーターが入っています。



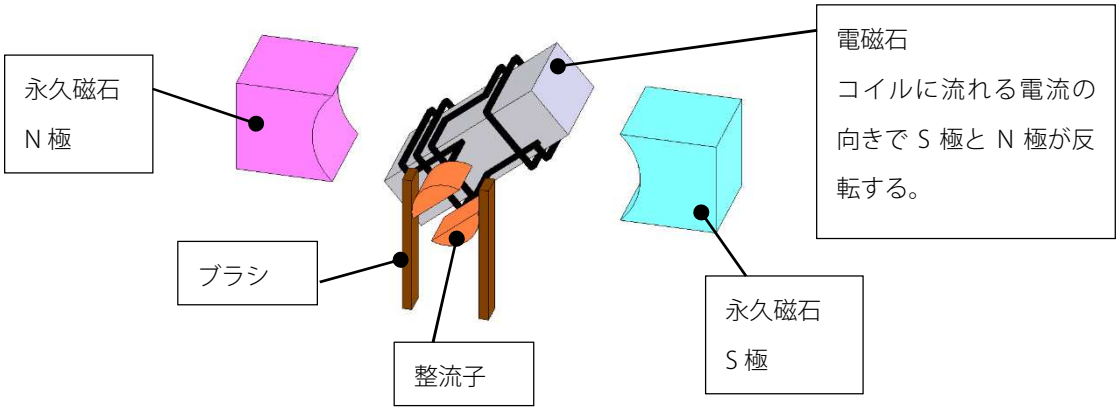
モーターの回転が分かりづらいときは、ビニールテープなどを軸に貼り付けましょう。



●使い方、ポイント

・しくみ、特徴

モーターとは、電気のエネルギーを回転エネルギーに変換する機械のことで、日本語では電動機といわれます。モーターのしくみを簡単に説明すると、永久磁石と、極性（S 極/N 極）を自動的に切り替える構造を持った電磁石を使い、その 2 種類の磁石が引き付けあう、反発しあう性質を利用して、回転の力に変えるものということになります。電磁石の極性を自動的に切り替えるために、整流子(コミュテーター)、ブラシといった部品が使われています。

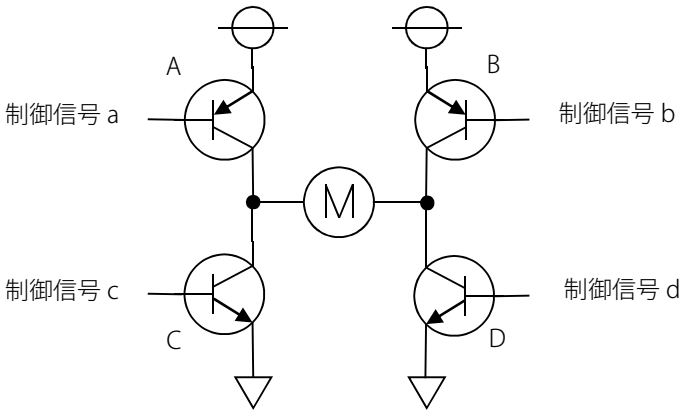


この「ブラシ」は、モーターの寿命に大きく影響しています。ブラシは回転する軸に常に接触した状態であるため、使っている間に磨耗してしまい、ある程度進むとモーターの動きが悪くなって電気的なノイズを撒き散らしたり、悲鳴のようなかん高い回転音を出すようになったりして、最終的には動かなくなってしまいます。

・モータードライバー回路

一般的にモーターは非常に大きな電流が流れます。マイコンでは取り出せる電流が決められていますので、マイコンで直接モーターを動かそうとすると、マイコンが壊れてしまいます。モーターをマイコンで制御するには、大きな電流を流せる回路を用意して、その回路をマイコンで制御します。モーターをドライブする回路なので、モータードライバー回路といわれます。

次の図にはよく使われるモータードライバー回路を示しています。

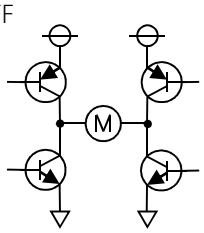
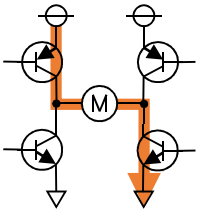
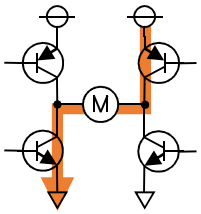
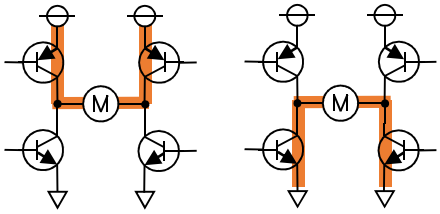
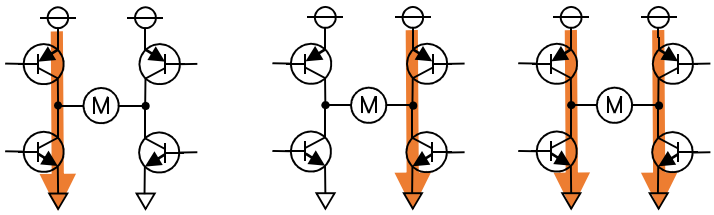


形がアルファベットの H に似ていることから、H ブリッジ回路といわれています。

H ブリッジ回路の具体的な動きを見てみましょう。

H ブリッジ回路は A、B、C、D の４つのトランジスタを制御するのが基本です。

それぞれのトランジスタの状態とモーターに流れる電流の向きを図示します。

① A、B、C、D が全て OFF 	電流はどこにも流れませんので、モーターは停止しています。
② A と D が ON、B と C が OFF 	電流は A→モーター→D と流れますので、モーターが正回転します。
③ A と D が OFF、B と C が ON 	電流は B→モーター→C と流れます。電流の向きが正回転のときと変わりますのでモーターは逆回転します。
④ A と B が ON、C と D が OFF　または A と B が OFF、C と D が ON 	モーターの両端子に同じ電圧が加わっている場合は、**電気ブレーキ**（発電ブレーキとも言います）になります。
⑤ A と C が ON、B と D が OFF　または A と C が OFF、B と D が ON　または A、B、C、D が全て ON 	**禁止状態** 部品が破損する危険な状態です。

・電気ブレーキ

回転しているモーターをピタッっと止めたいときなどに使います。

もちろん回転するスピードや、モーターに取り付けた負荷によっては、急停止にならない場合もありますが、①のようにトランジスタを OFF するだけの状態よりも早く停止します。

・禁止状態

絶対に、これら⑤の状態にならないように注意しなければなりません。回路図で一直線上にあるトランジスタ A と C（または C と D）が同時に ON するということは、電源と GND が直接つながっている状態と同じになります。つまりショート（短絡）です。トランジスタに過大な電流が流れ、ほとんどの場合トランジスタが破損します。ひどい場合は、煙が上がり、トランジスタのパッケージが溶けたり、割れて飛び散ったりします。

前述の図の①から⑤の状態を、制御信号とトランジスタの状態を表にすると以下のようになります。

	a	b	c	d	A	B	C	D	モーター動作
①	H	H	L	L	OFF	OFF	OFF	OFF	停止
②	L	H	L	H	ON	OFF	OFF	ON	正回転（逆回転）
③	H	L	H	L	OFF	ON	ON	OFF	逆回転（正回転）
④	L	L	L	L	ON	ON	OFF	OFF	電子ブレーキ
	H	H	H	H	OFF	OFF	ON	ON	電子ブレーキ
⑤	L	H	H	L	ON	OFF	ON	OFF	禁止状態
	H	L	L	H	OFF	ON	OFF	ON	禁止状態
	L	L	H	H	ON	ON	ON	ON	禁止状態

・トランジスタの選び方

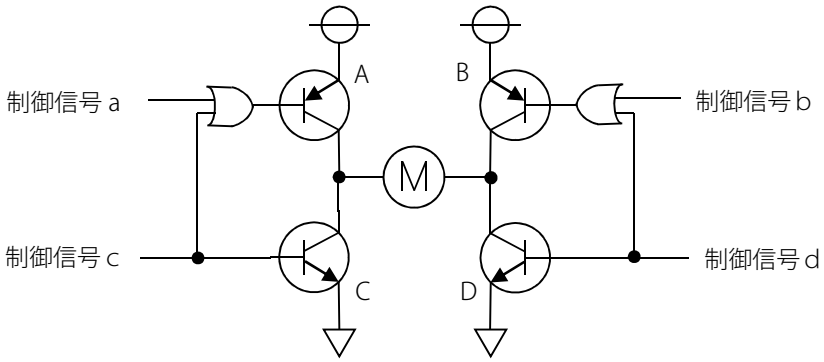
モータードライバーに使うトランジスタは、モーターに流れる電流に対応したものを選びます。また、モーターは一般的に回転を始める瞬間に、通常動作しているときに流れる電流の 4 倍～10 倍程度の始動電流（突入電流ともいわれることがあります。）が流れますので、この始動電流にも対応できるものを選ぶ必要があります。

電流が大きくなるとトランジスタではなく FET が使われることが多くなります。

・安全回路

H ブリッジを使ったモータードライバーは、前述した「禁止」されたトランジスタの状態があります。注意してプログラムを作るのは当然ですが、人間にはミスがつきものです、そこで、回路的に工夫して、禁止状態にならないように安全回路を加えると安心です。

次の図は安全回路を付け加えた例です。



ABCD のトランジスタを制御する信号 a,b,c,d の H/L の組み合わせがどんな状態であっても、ショートになりません。

以下の表は上記の安全回路の入った場合の動作状態の表です。
前述した安全回路の入っていない表と比べると分かるように、⑤のときの状態が変わっていて、禁止状態にならなくなります。

	a	b	c	d	A	B	C	D	モーター動作
①	H	H	L	L	OFF	OFF	OFF	OFF	停止
②	L	H	L	H	ON	OFF	OFF	ON	正回転（逆回転）
③	H	L	H	L	OFF	ON	ON	OFF	逆回転（正回転）
④	L	L	L	L	ON	ON	OFF	OFF	電子ブレーキ
	H	H	H	H	OFF	OFF	ON	ON	電子ブレーキ
⑤	L	H	H	L	OFF	OFF	ON	OFF	停止
	H	L	L	H	OFF	OFF	OFF	ON	停止
	L	L	H	H	OFF	OFF	ON	ON	停止

なお本機にも安全回路が入っていますが、上図の回路とは違ったものになっています。これは制御信号を 4 つではなく、２つ使っている場合の安全回路の一例になります。

●プログラム２

プログラム１では、モーターの正転、反転を行えましたが、この方法ではいつも全力でモーターが回転してしまいます。少しゆっくり回したいときや、電車が動き出すときのように少しずつ回転数を上げたい場合があります。このようなモーターの回転数を調整するときには、PWM 制御という方法が使われます。プログラム２では PWM 制御の方法を説明します。

モーターをゆっくり回転させます。

```
void setup(){
  pinMode(11,OUTPUT);              //モーター制御用 11 番ピンを出力にする
  digitalWrite(11,LOW);            //11 番ピンから L を出力

  analogWrite(9,100);              //9 番ピンからデューティ 100 の PWM 信号を出力開始
  delay(2000);                     //2 秒間待つ

  analogWrite(9,0);                //デューティ 0 つまり PWM 信号停止
}
void loop(){
}
```

●解説

このプログラムは、setup の中に書いたので一回だけ動きます。

モータードライバーの制御端子である 11 番ピンを出力にします。なお 9 番ピンは後で analogWrite 命令を使ったときに自動で出力になるため、pinMode で設定する必要はありません。

モーターの片側の端子を０V にするために、11 番ピンを LOW にします。

モーターのもう片方の端子 9 番ピンは、analogWrite 命令で PWM 機能を使って電圧を加えます。

電圧を加えるとモーターが回りだしますので、delay 命令でその状態を 2 秒間保持します。

2 秒後に、9 番ピンに電圧を加えるのを止めて、モーターを停止させています。

●使い方、ポイント

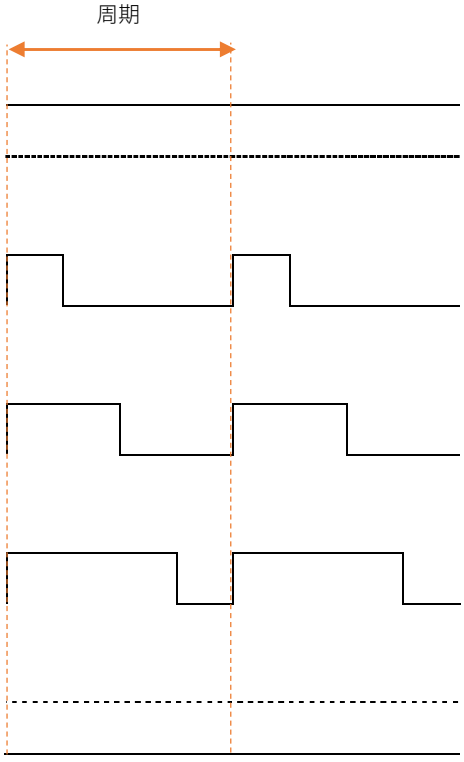
・モーターの PWM 制御

モーターの回転スピードを調整するにはどうしたらよいでしょうか。1 つの方法として、モーターに加える電圧を変える方法があります。しかしマイコンは H、L の 2 つの電圧の状態を取り扱うのは得意ですが、電圧を少しずつ変化させることには不向きで、マイコンの外部に大掛かりな回路が必要になります。プログラムでモーターのスピードを調節する方法としては、モーターを素早く ON/OFF させる方法があります。

ある一定の期間で OFF する時間が長ければそれだけモーターに流れる電流が減るので、モーターの回転は遅くなります。

このように、ある一定の期間の中で H と L の割合を変えることを PWM と言います。Arduino にはこの PWM を簡単に設定することができる **analogWrite** 命令が用意してあります。
analogWrite(9,100)と書くと、自動的に 9 番ピンから H と L を切り替えながら出力を続けます。
この H を出力する期間（ここでは 100 の部分）を変更することで、モーターの回転数を調節することができます。
Arduino では、この H を出力する期間を 0 から 255 の範囲で設定します。
0 を設定すると H は出力されません。また、255 を設定すると、常に H を出力することになります。
一定の期間の中で H を出力する時間の割合のことをデューティ比といっています。

図に示すと以下のようになります。



(1) 常に H を出力（OFF する時間なし）
analogWrite(9,255); デューティ比=100

(2) H を少しの間出力
analogWrite(9,64); デューティ比=25

(3) H を半分の期間出力
analogWrite(9,128); デューティ比=50

(4) H を多めに出力
analogWrite(9,192); デューティ比=75

(5) H を出力しない（OFF だけ）
analogWrite(9,0); デューティ比=0

・ON-OFF 式、ON-SHORT 式

PWM 制御でモーターを動かすときのテクニックに、ON-OFF 式と ON-SHORT 式の 2 つがあります。
PWM 制御は前述したように ON と OFF を繰り返す方法ですが、その OFF のときに、ブレーキを掛けるか掛けないかの違いです。
ブレーキを掛けない制御方法のことを ON-OFF 式、ブレーキを掛けることを ON-SHORT 式といいます。
例えばモーターで動く車が坂道を登るときを考えます。モーターが ON して動いている間は当然坂を登り

ますが、モーターが OFF のときにブレーキを掛けなかったらバックしてしまいます。つまり少し進んで少しバックしてという具合に、全体としては弱々しい感じで登ることになります。しかしモーターが OFF のときにブレーキを掛ければバックしませんので、少し登って停止、さらに少し登って停止を繰り返します。つまり、ブレーキを掛けない場合に比べると力強く登っていくことになります。
この２つの方式はどちらが良いということはありませんので、使用状況にあった制御を選択します。

●プログラム３

PWM 制御で、モーターの回転数を段々下げて停止、を繰り返します。

```
void setup(){
  pinMode(9,OUTPUT);
  digitalWrite(9,LOW);
}
void loop(){
  for(int rev = 140; rev >80; rev--){
    analogWrite(11,rev);
    delay(200);
  }
  analogWrite(11,0);
  delay(1000);
}
```

●解説

プログラム２で説明した PWM 制御を使って回転数をだんだん下げます。**analogWrite** 命令のデューティの値部分を変数にしておき、**for** 文を使って変数の値を減らす処理を行っています。
本機に使用しているモーターで実際に実験すると、デューティの値が 80 以下になると回転しなくなるので、遅い回転のデューティの値は 80 までとしています。
また速い回転のデューティの値は 140 までとしていますが、その理由は、モーターに何もつないでいない状態ではデューティの値が 140 でも 255 でも回転数にあまり差を感じなかったため、実験の分かりやすさを優先して 140 にしました。ただし、実際にモーターに負荷をつないで動かすとこれらデューティの値と回転数の関係は変わってきますので、デューティの値を決める場合は使用する機器でしっかり実験して決める必要があります。

コラム 5

電気的なノイズ対策

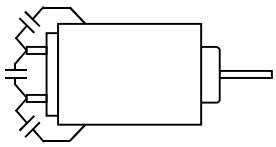
モーターを動かすときに電気的なノイズが回路に影響を与えて誤動作する問題があります。
ノイズの原因には大きく次の 2 つがあります。

・モーターが回転することで発生する電気ノイズ（モーターノイズ）。
・PWM 制御のように、電圧を素早く ON/OFF と切り替えることで発生するスイッチングノイズ。

ノイズの対策には、コレだ！という決定版はないのですが、モーターノイズ、スイッチングノイズの対策としてよく行われる方法を紹介します。

（1） モーターの端子にコンデンサーをとりつける。
コンデンサーを 1 個使うとき。

コンデンサーを 3 個使うとき。

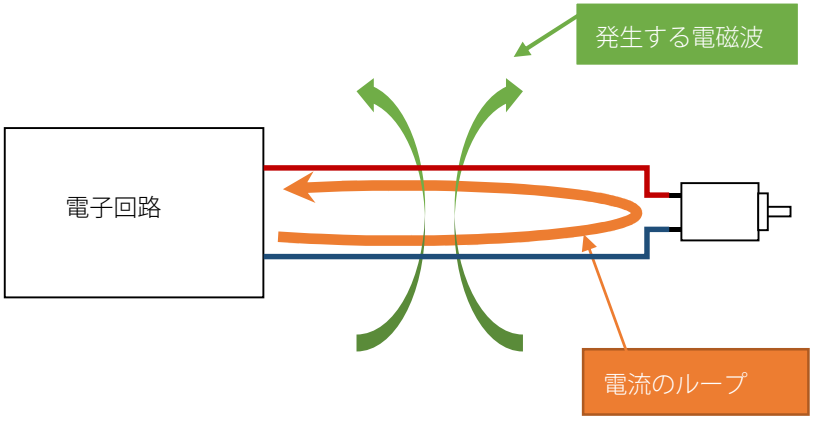


取り付けるコンデンサーはできるだけノイズの発生源つまりモーターに近い場所に取り付けるのがポイントです。

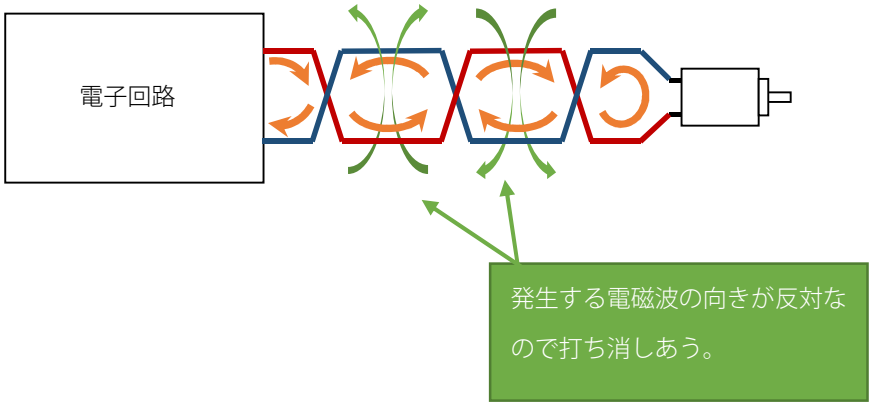
本機のモーターにも端子に 0.1μF のセラミックコンデンサーが付けられています。

（2）　モーターまでの配線コードを寄り合わせる。

モーターを使うときは基板に直接取り付けられることはなく、配線でつないでいます。すると図のように配線のループができてしまいます。そしてループに電流が流れると、必ず電磁波が発生します。このループの中を PWM で ON-OFF された電流が激しく流れ、それがスイッチングノイズになります。



配線を寄り合わせることで、図のように、ループで発生した電磁波を、隣のループで発生した電磁波が打ち消す効果があり、ノイズを低く抑えることができます。



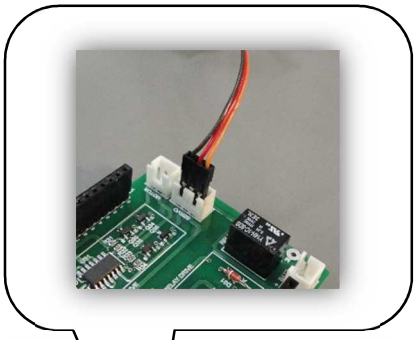
コードを寄り合わせた例

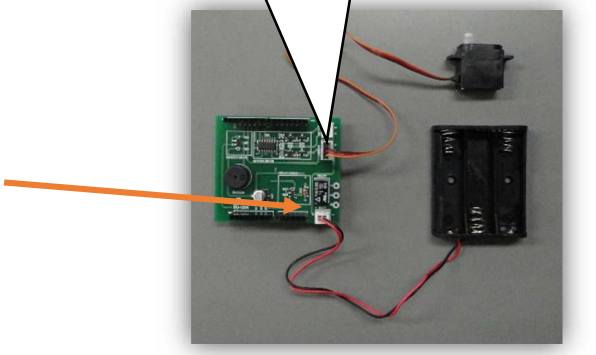
4. サーボモーターを制御する

マイコンとモーターを使って動くものを制御するときに、回転の制御ではなく、角度の制御が必要な場合があります。例えば車のステアリングやロボットの関節などです。このようなときに便利に使える部品としてサーボモーターがあります。サーボモーターは電源をつなぐだけでは動かず、制御信号を入力して動かす必要があります。ここではサーボモーターについて説明します。

●サーボモーターを使う準備

サーボモーターは、本機の「SERVO」端子に、
リレーに近いほうから、橙、赤、茶になるように
向きを注意して接続します。
間違わないようにしっかり確認します。





ドライバーシールドのモーター用電源端子に、
電池ボックスを接続し、電池をセットします。

注意してください

サーボモーターの軸をロックさせないこと。

本機に付属のサーボモーターは消費電流が少ないものを採用していますが、このサーボモーターであっても、動いているときに、無理矢理動きを止めたりすると、回路に過大な電流が流れて、最悪の場合回路が故障することがあります。または、ギアの歯が折れてしまうことがあります。

●プログラム１

サーボモーターの最も基本的な使い方でプログラムしてみます。

プログラムを書き込む（アップロード）と、サーボモーターが開始位置に移動し、3 秒後に約 120 度回転し、3 秒後に開始位置に戻ることを繰り返します。

```
void setup(){
  pinMode(10, OUTPUT);
}
void loop(){
  for(int i=0; i<40; i++){
  digitalWrite(10,HIGH);
  delayMicroseconds(900);
  digitalWrite(10,LOW);
  delay(19);
  }
  delay(3000);

  for(int i=0; i<40; i++){
  digitalWrite(10,HIGH);
  delayMicroseconds(2100);
  digitalWrite(10,LOW);
  delay(18);
  }
  delay(3000);
}
```

（A）

（B）

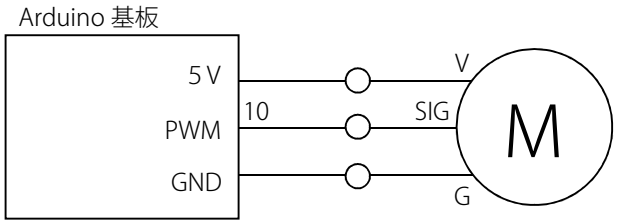
●解説

`setup` の中で、サーボモーターの角度制御信号端子を接続した 10 番ピンを出力にします。

`loop` の中では、まず、（A）で開始位置に移動する制御信号を発生します。3 秒後に、（B）で 120 度移動した位置に移動する制御信号を発生します。次に delay 命令を実行した後に `loop` の先頭に戻ります。

※制御信号については、次の使い方、ポイントの中で説明します。

●回路



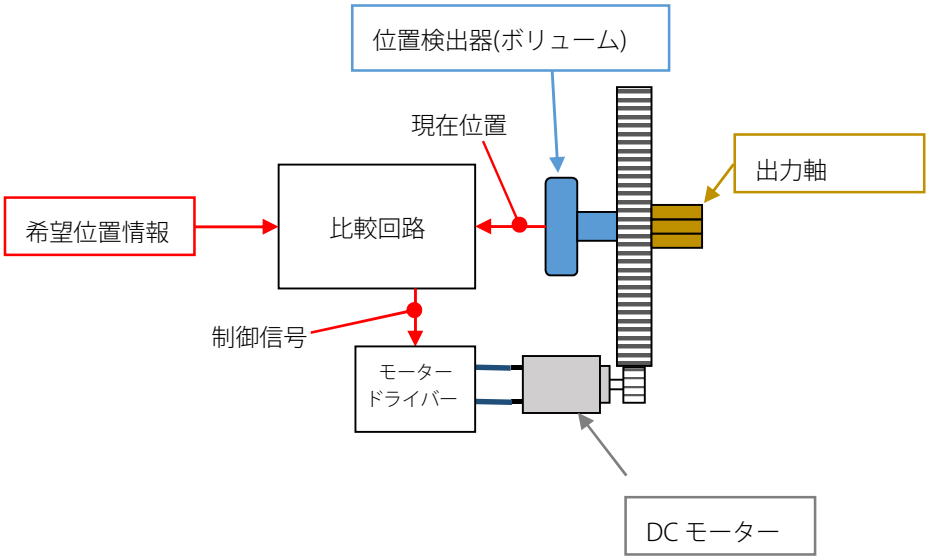
●使い方、ポイント

・しくみ

サーボモーターは内部に特別なしくみを持たせることで、希望の角度まで回して停止させることができます。そのかわりにほとんどのサーボモーターは回転し続けることができないようになっています。

サーボモーターには色々な種類がありますが、ここでは本機で使用しているサーボモーターについて説明します。

本機のサーボモーターの原理を以下の図に示します。



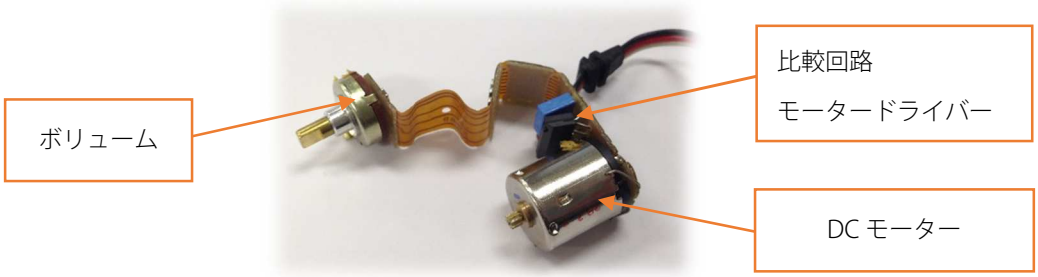
サーボモーターといっても中身に特別なモーターを使っているわけではなく、乾電池のような直流電源を接続すると回転し続ける一般的なＤＣモーターを使っています。違いは、『今、出力軸がどの位置にあるか。』を検出する「位置検出器」と、その位置情報と、希望の位置情報を比較する「比較回路」。比較の結果でモーターをもっと回すのか、止めるのかを制御する「モータードライバー」がセットになっていることです。

出力軸の回転にあわせて同時にボリュームのツマミが回転するように取り付けられており、その回転角度に応じてボリュームの抵抗の値が変わることを利用して位置検出を行っています。

モータードライバーは、位置検出器(ボリューム)の情報と、プログラムで書いた希望の位置情報を常に比較して、その差が０になるまでモーターを動かします。
位置の差が０になるか、または、希望の位置情報が送られてこなくなると、モーターを停止させます。

注意しておきたい点として、サーボモーターは、位置情報が送られてきている間は、位置を合わせようとモーターを制御し続けるけれども、位置情報が送られてきていないときは、制御しないということです。つまり、サーボモーターに重たいものを取り付けているような場合は、希望の角度まで移動させて、位置情報（制御信号）を送るのを停止すると、その重さで、位置がずれてしまうことがあります。

あるサーボモーターを分解したところ

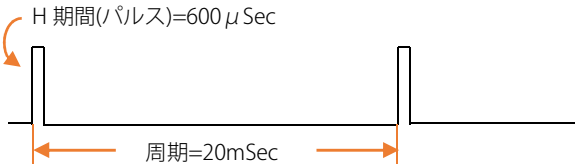


・角度制御信号

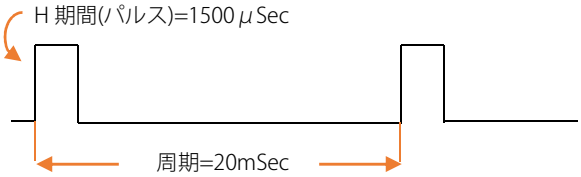
サーボモーターの角度制御は、PWM 方式（パルス変調方式）によって行われます。本機のサーボの場合、具体的には 15～20mSec（※Sec＝秒です）の周期で、H の期間を 600μSec から 2300μSec の間で変更して、希望の角度を指定します。

※なお、本機のサーボモーターの移動角度は 170 度以下で、ニュートラル（中間位置）にするには 1500μSec を入力します。一般的なサーボモーターも大体この辺りの値になりますが、メーカーや種類によっても変わり、さらに個体差もありますので実際に使用するサーボモーターにあわせて確認調整が必要です。

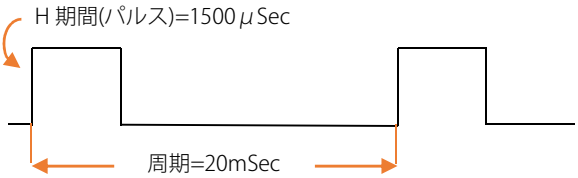
（ａ） 600μSec の制御信号



（ｂ） 1500μSec の制御信号



（ｃ） 2300μSec の制御信号



勘違いしやすいのは、「制御信号を 1 回だけ入力すれば希望の位置まで動く。」と考えてしまうことです。実際にはしくみで説明したように、

「制御信号(希望位置)と現在位置に差があればモーターを動かす。」

「制御信号がないときはモーターを停止。」

となっていますので、制御信号が 1 回しか入力されなければ、モーターが少し動くだけとなります。つまり、希望位置まで移動するまでは制御信号を入力する必要があります。この移動するまでの時間は、データシートに「動作スピード」として記載されています。

・動作スピード

本機のサーボモーターの軸の回転スピードは、無負荷時(出力軸に何も付けていない時)に、
「60 度回転するのに 0.2 秒」となっています。別の言い方をすれば、
「60 度回転させるためには、0.2 秒間制御信号を与えなければいけない」ということです。
例えば、現在の位置から 60 度動かしたいときに、制御信号を 0.1 秒しか与えなければ、希望の位置まで移動せずに動きが終わってしまいます。
制御信号は、出力軸が希望の位置まで移動するのに十分な時間与え続ける必要があります。

・モーターのロック

サーボモーターは、内部で DC モーターを動かしていますので、動いている途中で、出力軸を手で無理矢理止めたり、サーボモーターで動かすことができないような重量物を取り付けると、モーターがロック状態となりモーターに大量の電流が流れ続けます。すると、モータードライバーにも負担がかかり、最終的にはドライバーが破損してしまいます。
また、決められた範囲以外まで動かそうとする信号をサーボモーターに与えた場合も同様に、出力軸がこれ以上動けない位置にも関わらず、モータードライバーはいつまでも与えられた位置まで移動させようとして電流を流し続け、結果としてドライバーが破損してしまいます。
モーターがロックする使い方をしていないか、制御信号は適切かどうかなどをしっかり確認しましょう。

●プログラム２

プログラム１で行った方法でサーボモーターは一応動きますが、サーボモーターの最大の特徴である、角度を自由に設定して次々と変化させるといった動かし方をするにはプログラムが大変になり、この方法では不向きです。

サーボモーターを少しでも簡単に動作させるための「ライブラリ」が Arduino には用意されています。

次のプログラムは、そのライブラリを使った例です。

ライブラリを使ってプログラム１と同じようにサーボモーターを動かすプログラム

```
#include <Servo.h>
Servo myservo;
void setup(){
  myservo.attach(10,600,2300);
}
void loop(){
  myservo.write(0);
  delay(3000);
  myservo.write(180);
  delay(3000);
}
```

●解説

プログラムの最初に、「このプログラムでは Servo 用のライブラリを使います。」という「宣言」を書かなければいけません。その宣言が、 `#include <Servo.h>` です。

続いて、Servo ライブラリの機能を、このプログラムの中では `myservo` という名前をつけて使うことを宣言します。この名前は自由につけることができます。

次に、サーボモーターを Arduino 基板の何番のピンにつないだのかを `attach` 命令で設定します。

この `attach` 命令は初期設定なので、`setup` の中に書くようにします。

`loop` の中で、`write` 命令で角度を設定してサーボモーターを動かします。

最初に 0 度の位置に動かし、3 秒後に 180 度の位置に動かしています。

（※本機のサーボの可動範囲は 170 度以下なので、プログラムで指定した角度と、実際に動く角度が少し違ってきますが、実験のやりやすさと分かりやすさを優先しています。）

●使い方、ポイント

・ライブラリを使った場合に使用する命令

myservo.attach(pin, min, max)

この命令は、pin で Arduino 基板の何番ピンにサーボモーターをつないだのかを設定し、min でそのサーボモーターの角度が 0 度のときのパルス幅(マイクロ秒)、max で 180 度のときのパルス幅(マイクロ秒)を設定します。

min,max は省略することもでき、myservo.attach(pin)と書くこともできます。min,max を省略した場合はデフォルトで、min に 544μ秒が、max に 2400μ秒が設定されます。

myservo.write(angle)

この命令で、サーボモーターの角度を設定します。angle に出力軸を向けたい角度を書きます。0 度～180 度の間で設定します。

・Servo ライブラリが使用できるピン

このライブラリは、Arduino 基板の 9 番ピン、10 番ピンを使用します。サーボモーターは 9 番または 10 番に接続する必要があります。

このライブラリは 9 番、10 番の両方のピンを使用しますので、例えばサーボモーターを 10 番ピンにつないで、9 番ピンを別の機能のために使うということはできません。

どうしても途中で違う機能に切り替える必要がある場合は、detach()　命令を使います。

今回の場合では、myservo.detach();　と書くと、9 番、10 番ピンがサーボ機能から切り離され、通常のポートとして使用できるようになります。

コラム６

大きな電流が流れると・・・

マイコンでサーボモーターや DC モーターを制御するときには、モーターロック時の電流や、始動電流といった、大きな電流に注意する必要があるといわれますが、具体的な問題点は何なのでしょうか。

１つはもちろん、とても大きな電流が流れた場合は、電流で部品そのものが壊れてしまう、または、その部品は大丈夫でもその部品の制御に関係したマイコンなどの部品が壊れてしまう問題です。
これについては、特性やデータシートを確認して、誤った使い方をしないようにしなければなりません。

もう１つの問題点は、破損までには到らない電流の場合です。例えばモーター側に大きな電流が流れてしまうと、それ以外の回路へ十分な電流が流れず、制御回路側の電圧が下がってしまいます。この問題のやっかいなところは、一瞬または途中までは正常に動くので、原因が分かりづらいことです。

Arduino の場合では、電圧が下がることにより、途中でマイコンがおかしな動作や信号を出力したりします。それを「マイコンが暴走した。」と表現したりします。
マイコンが暴走すると、どの回路にどう影響が出るか分かりません。PC とつないで動かしている場合は PC との接続状態が勝手に切断されたりすることもあります。

モーターなどの大きな電流を取り扱う回路で、Arduino 基板の挙動がおかしくなった場合は、一度電源を切る(USB ケーブルを抜く)、接続している部品を取り外す、Arduino-IDE を立ち上げなおす、などの作業をやってみると良い場合があります。

対策として最も効果があるのは、制御回路用の電源と、モーター用の電源を別にすることです。本機も、制御系には USB から供給される電源を利用し、モーターには外部に接続した電池ボックスを電源として動かしています。

Arduino 基板と Arduino-IDE

（1） Arduino 基板

●いろいろな Arduino

市販されている Arduino には多くの種類があります。Arduino の最も標準的な Arduino-UNO に加えて、ブレッドボード上でのテストに便利な Nano、衣服に縫い付けることを考えられた LilyPad、PC との通信部分を取り外して安価にした Pro、入出力が強化された Mega などがあります。

これら全ての Arduino は、プログラムを作る仕組みが同じで、同じ Arduino-IDE といわれるプログラム開発ソフト（総合開発環境といいます）で作成することができます。

プログラムの書き換えに特別な機器は必要としないので、プログラムの書き込みが手軽ですぐに使い始められます。

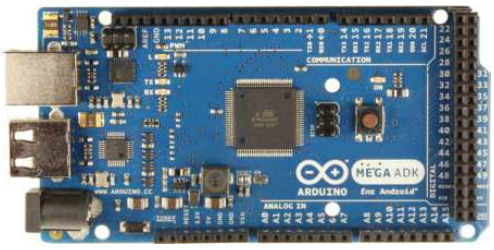
Arduino の例

Arduino-UNO

Arduino-Nano

LilyPad

Arduino-Pro



Arduino-Mega

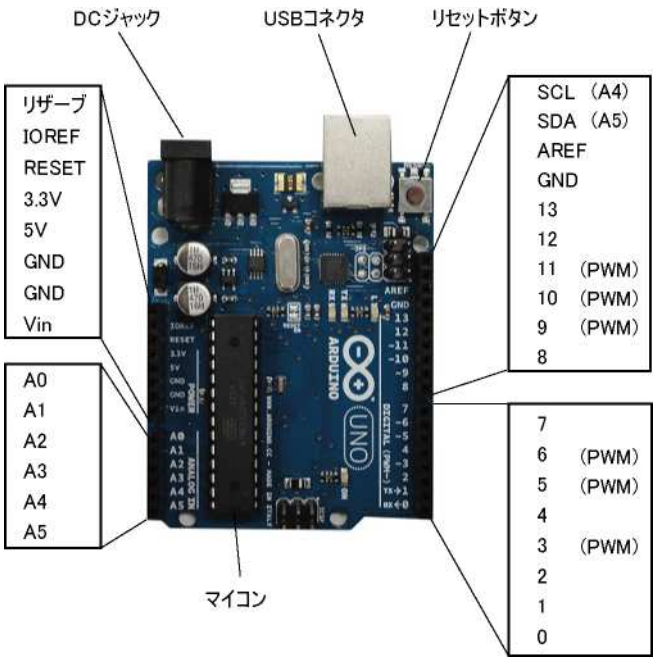
本書で説明に使用しているシールドは、**Arduino-UNO(R3)**と接続します。

●機能説明

【ピン配置図】

Arduino-UNO(R3)
の場合

※ Arduino-UNO(R3) 以外ではピン数、機能が異なる場合があります。



【ピンの機能】

機能	説明
リザーブ	将来の拡張用に予約されているピンで使うことはできません。
IOREF	ArdunoUNOではVccに接続されています。
RESET	リセットボタンに接続されています。リセットボタンの追加に使用できます。
3.3V	3.3Vの電源になります。
5V	5Vの電源になります。シールド基板の電源はここから供給しています。
GND	グラウンドです。シールド基板の電源のマイナスはここにつながっています。
GND	
Vin	DCジャックから入力された電源につながっています。 （内部でダイオードを経由しますので、電圧が少し低くなります。）

機能1	機能2	説明
A0		アナログ入力ポートです。 センサーなどの接続に向いています。 デジタル入出力ピンとして使うこともできます。
A1		
A2		
A3		
A4	SCL	I2C(Wire)通信を使うときには、SCL/SDAとしてここを使います。
A5	SDA	

機能1	機能2	説明
SCL	(A4)	A4,A5はここにもつながっています。
SDA	(A5)	
AREF		AD変換器の基準電圧のためのピンです。使用するためには特別な命令や配線が必要です。
GND		グラウンドです。
13		13～0はデジタル入出力ピンとして使用します。 11,10,9,6,5,3はPWM制御できるピンとして使用することもできます。
12		
11	PWM	
10	PWM	
9	PWM	
8		
7		
6	PWM	
5	PWM	
4		
3	PWM	
2		
1	TX	1,0はUSB経由でPCと通信するときに使用します。
0	RX	

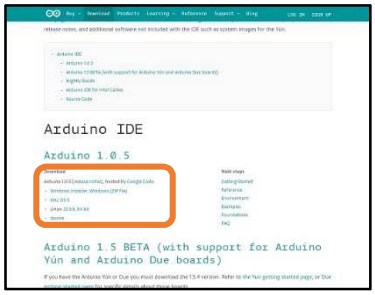
（2） Arduino-IDE の準備

Arduino を制御するためには、プログラムを作成し、Arduino 基板へ書き込みを行う必要があります。このプログラムを作成するエディタとプログラム書き込み機能を持った開発環境のことを、Arduino-IDE といいます。

Arduino-IDE の入手と起動

1. まず Arduino のホームページにアクセスします。
   http://www.arduino.cc/ (英語のサイトです)

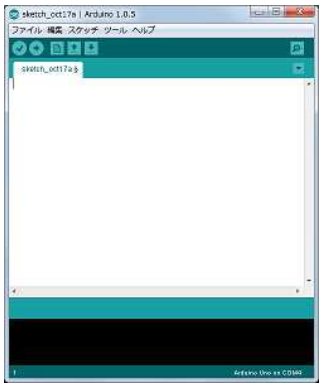
2. ダウンロード画面から、お使いの OS に対応したソフトを選択し、適当なフォルダーに保存します。



画面は変わる可能性があります
画面は 2013 年 10 月現在のものです。

3. ダウンロード後、保存したフォルダー内に、arduino-X.X.X-XXX が作成されます。このファイルは圧縮されているので解凍(展開)ソフトなどで解凍してください。
   ※XXX は使用する OS やソフトのバージョンによって異なります。
4. ファイルを解凍すると、Arduino-1.0.1 のように、バージョン番号が付いたフォルダーが作成されます。
5. 先ほど解凍したフォルダー内にある、arduino のアイコンをダブルクリックすると、arduino-IDE が起動し画面が表示されます。

↓フォルダーの中の arduino をクリック　　IDE 起動画面



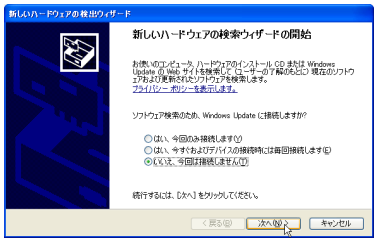
ドライバーのインストールとシリアルポート(COM ポート)の確認

Windows の場合

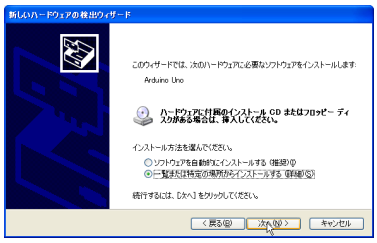
次に、Arduino 基板とパソコンが通信するために必要なドライバーをインストールします。

1. Arduino 基板とパソコンを USB ケーブルで接続します。
2. WindowsXP の場合：

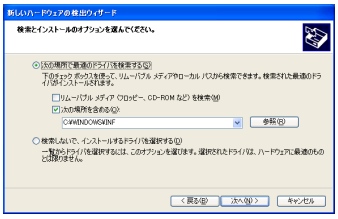
   接続後に「新しいハードウェアの検出」画面が表示されますので、「いいえ。今回は接続しません。」を選択して次へ進みます。

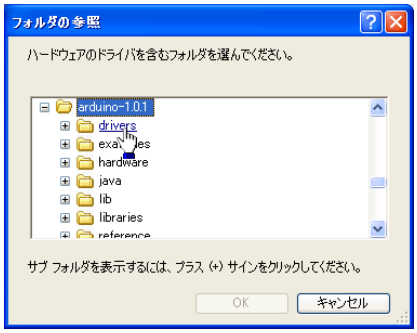


「一覧または特定の場所からインストールする。」を選択して次へ進みます。



「次の場所を含める。」で、参照のボタンをクリックして、先ほど解凍したフォルダーの中にある「Drivers」を選択して、次へ進みます。





ドライバーのインストールが始まり、自動的に完了します。

完了後に、もう一度ドライバーのインストールを求められる場合は、1 回目と同じように進めて完了させます。

WindowsVista/7 の場合：

「デバイスドライバーソフトウェアをインストールしています。」と表示され、しばらく待つと、パソコンが自動でドライバーを検索しインストールが完了します。もしも自動でインストールができない場合は手動でインストールする必要があります。

＊トラブルシューティング「ドライバーを手動でインストールする。」をお読みください。

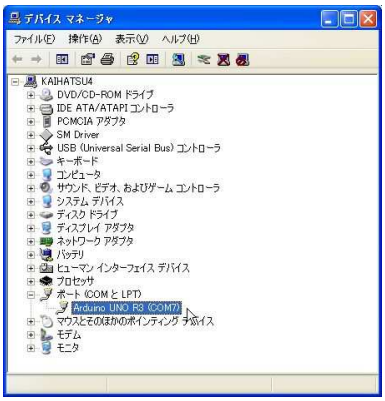
3. 次に Arduino 基板が、どのシリアルポート（COM ポート）に接続されたか確認します。

   WindowsXP では、マイコンピュータのアイコンを右クリックして「プロパティ」を選択し、「ハードウェア」のタブの中にある「デバイスマネージャー」をクリックします。

   WindowsVista/7 では、「コントロールパネル」→「システムとセキュリティ」→「デバイスマネージ

ャー」と進みます。

デバイスの一覧が表示されますので、「ポート（COM と LPT)」という項目の下を確認します。



Arduino 基板が接続されたポートが表示されます。

図では Arduino 基板が「COM7」に接続されていることが分かります。この COM の番号を覚えておきます。

4. Arduino-IDE 画面の「ツール」メニューから、「マイコンボード」を選択し、パソコンに接続した Arduino 基板を選択します。
5. 次に Arduino-IDE 画面の「ツール」メニューから、「シリアルポート」を選択し、先ほど確認した COM ポートの番号と同じものを選択します。

これで Arduino を使用する準備が整います。

ドライバーのインストールとシリアルポート（COM ポート）の確認

Mac OS X の場合

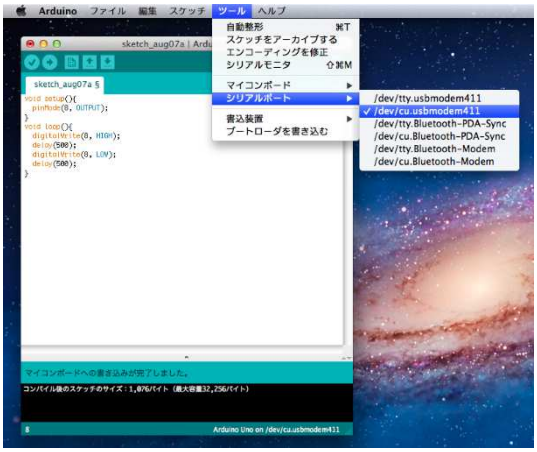
Mac OS X の場合、ダウンロードが完了すると自動的にドライバーがインストールされますので、表示されるメッセージに従い必要に応じて管理者パスワードの入力や再起動を行ってください。

Mac OS X でシリアルポートを選択する。

1.Arduino 基板をパソコンと接続します。

2.Arduino-IDE を起動し、画面の「ツール」メニューから、「マイコンボード」を選択し、パソコンに接続した Arduino 基板を選択します。

3.次に Arduino-IDE 画面の「ツール」メニューから、「シリアルポート」を選択し、「/dev/cu.usbmodem-」または「/dev/tty.usbmodem-」ではじまる項目を選んでください。



これで Arduino を使用する準備が整います。

起動画面

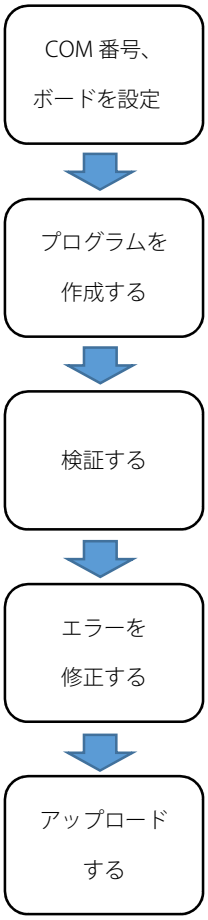
新規ファイル　：　新しいプログラム作成エリアを開きます。
開く　：　保存しているプログラムを開きます。
保存　：　作成したプログラムを保存します。
検証(コンパイル)　：　正しい文法でプログラムが作成されているかチェックします。
シリアルモニタ　：　シリアルデータを表示します。
（データに何を表示させるかあらかじめプログラムに書く必要があります。）
マイコンボード　：　プログラムを PC に接続している Arduino 基板に書き込みます。
に書き込む　アップロードと表示されている場合もあります。
プログラム作成エリア　：　プログラムを編集するエリアです。
メッセージエリア　：　操作に応じてメッセージやエラーが表示されます。
コンソール　：　メッセージやエラーの詳細が表示されます。(※)
ボード・COM 番号　：　Arduino-IDE で設定している、Arduino 基板の種類と COM ポート番号が表示されます。

※　ほとんどの場合、非常に分かりづらい意味不明の文字が表示されます。そんなときは、表示された文字や単語を使って検索サイトで調べることになります。

（3） Arduino-IDE の使い方

●プログラムの作成からアップロードまでの流れ

プログラムを作成し、Arduino 基板に書き込むまでのおおよその流れは以下のようなイメージになります。



Arduino-IDE 起動後に一度だけ設定すれば OK です。
特に COM ポートの設定は忘れることが多いので注意しましょう。
Arduino 基板と PC は USB ケーブルでつないでおく必要があります。

プログラム作成エリアに、プログラムを作成します。

Arduino 基板に書き込む前に検証します。
エラーが表示された場合はエラーを修正します。
エラー表示は、どこが間違っているか的確に表示してくれることもありますが、ほとんどの場合、読んでもよく分からない表示になっています。それでも、エラーの発生している近くを指摘していることが多いので、エラー表示された部分の周辺をチェックすることから始めましょう。

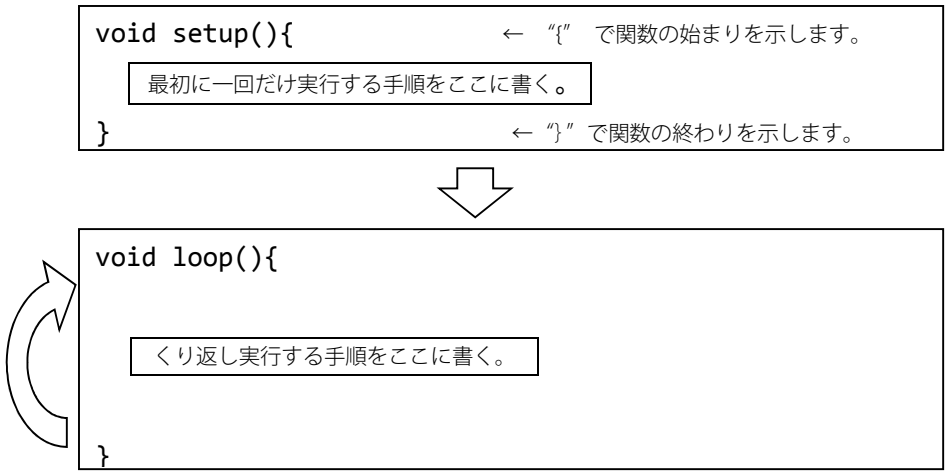
作成したプログラムを Arduino 基板に書き込みます。書き込み完了後はメッセージエリアに「書き込みが完了しました」と表示されます。

●Arduino のプログラムの基本

Arduino-IDE で作成したプログラムをアップロードすると、Arduino 基板のマイコンの中に書き込まれます。書き込まれたプログラムは、基板の電源を ON/OFF しても消えません。書き込まれたプログラムは、基板の電源を ON 後、またはリセット後に、自動で実行されます。

Arduino のプログラムには、「setup」と「loop」という 2 つの関数が必要です。
関数とは、Arduino 基板上のマイコンに、何をどんな順番で行わせるかを記述した手順書です。
電源 ON 後に、「setup」の中の手順を一回だけ実行し、続けて、「loop」の中の手順を、電源を切るまでくり返し実行するということを覚えてください。setup と loop は省略できません。

プログラムを実行するイメージ



void setup()、void loop() って何？と思うかもしれませんが、これは Arduino-IDE で、setup 関数と loop 関数を使うときの決められた形ですので、「この形で使うもの。」と覚えてください。

●よくやってしまうミス

しっかり注意しながらプログラムしたつもりでも、書き忘れやタイプミスをやってしまいます。とくに以下のようなミスはエラー表示でどこを示しているのか分かりづらいので十分注意しましょう。

； ←セミコロンを忘れる。またはセミコロンではなく、： ←（コロン）を使ってしまっている。
, ←(コロン)ではなく、 . ←（ピリオド）を使ってしまっている。
｛ と ｝ ←（波カッコ）が対になっていない。 ｛ ｝が多重になるときは要注意です。
全角スペースを使っている。プログラムの中では全角文字は使用できません。

●Arduino の言語

ここに書いたものが全てではありませんが、Arduino のプログラムでよく使う文法や関数をまとめています。Arduino-IDE のその他の関数や詳細については書籍などで学習を行ってください。

基本となる文法

形式　　;　（セミコロン）

説明　　文の終わりに使います。基本的に Arduino では文の終わりにセミコロンが必要です。
　　セミコロンによって文がどこで区切られているかを示します。日本語の句点に似ています。

使用例
```
int redLED 9 ;
pinMode(9, OUTPUT);
delay(100);
```

形式　　{　}　　（波カッコ）

説明　　複数の文をまとめるために使用します。

使用例
```
if(x==1){
  文 1;
  文 2;
}
```

デジタル入出力

形式　　`pinMode(pin, mode)`

説明　　ピンの動作を入力または出力または、プルアップ機能を利用した入力に設定します。

パラメータ　`pin`　　設定したいピンの番号

　　`mode`　　`INPUT`　または　`OUTPUT` または　`INPUT_PULLUP`

使用例
```
pinMode(9, OUTPUT);          9 番ピンが出力になります。
pinMode(13,INPUT);           13 番ピンが入力になります。
pinMode(A0, INPUT_PULLUP);   A0 番ピンがプルアップされた入力になります。
```

形式　`digitalWrite(pin, value)`

説明　指定したピンを、`HIGH` または `LOW` にします。`HIGH` は 5V、`LOW` は 0V(GND)になります。

パラメータ　`pin`　設定したいピンの番号

`value`　`HIGH` か `LOW`　（または、`1`　か　`0`）

使用例　`digitalWrite(9, HIGH);`　9 番ピンが `HIGH` になります。

`digitalWrite(2, 0);`　2 番ピンが `LOW` になります。

形式　`digitalRead(pin)`

説明　指定したピンの状態を調べます。結果は `HIGH` か `LOW` になります。

パラメータ　`pin`　調べたいピンの番号

使用例　`x = digitalRead(9);`　x に 9 番ピンの状態が `HIGH` か `LOW` で記憶されます。

例えば 9 番ピンが `GND` に接続されている場合は `LOW` になります。

アナログ入出力

形式　`analogRead(pin)`

説明　指定したアナログピンの状態を調べます。結果は 0 から 5V の電圧範囲を 0 から 1023 の範囲に変換した値になります。Arduino-UNO では A0～A5 がアナログピンとして利用できます。

パラメータ　`pin`　調べたいアナログピンの番号

使用例　`x = analogRead(A0);`　x に A0 番ピンの状態が 0 から 1023 の整数値で記憶されます。

形式　`analogWrite(pin, value)`

説明　指定したピンから PWM 波を出力（アナログ出力）します。

`analogWrite` の前に `pinMode` で出力にする必要はありません。

Arduino-UNO では３，５，６，９，１０，１１番ピンがアナログ出力ピンとして利用できます。

`analogWrite` を実行すると、次に同じピンで `analogWrite` を実行するまで PWM 波が出力されます。

パラメータ　`pin`　出力に設定するピン番号

`value`　PWM 出力のデューティ比（0 から 255 の範囲で設定します。）

使用例　`analogWrite(3, 64);`　3 番ピンからデューティ比が 64 の波形を出力します。

`analogWrite(3, 0);`　3 番ピンからはデューティ比が 0(=0V)が出力されます。

`analogWrite(3, 255);`　3 番ピンからはデューティ比が 255(=5V)が出力されます。

制御文

形式　`if (条件) { 条件に一致したときに実行する文 }`

説明　カッコ内の条件が true つまり条件が満たされている場合は、次に続く波カッコ内の文を実行します。
false つまり条件が満たされていない場合は、波カッコ内の文を実行せずに次に進みます。
波カッコの中の文が 1 つだけの場合は波カッコを省略できます。

使用例
```
if ( x == 1 ) {
  digitalWrite(9, HIGH);
  delay(100);
}

if ( y < 500 ) digitalWrite(4, LOW);
```

注意点　比較に利用する　＝（等号）は　＝＝のように 2 つ書かなければいけません。
＝が 1 つの場合は代入となり、全く違った意味になり、結果としていつも true と判断されます。

形式　`if(条件){条件に一致したときに実行する} else {条件に一致しないときに実行する}`

説明　カッコ内の条件に一致したときに実行する文と、一致しないときに実行する文をそれぞれ分けて書くことができ、if よりも細かく制御できます。

使用例
```
if (x == 1) {
  digitalWrite(9, HIGH);
}else{
  digitalWrite(9, LOW);
}
```

形式　`for (初期化 ; 条件式 ; 加算) {実行する文}`

説明　条件を満たしている間、波カッコで囲まれた文を繰り返し実行します。
カッコの中は 3 つの部分から成り立っています。
動作は、まず初期化が一度だけ実行されます。次に、条件式がテストされ条件を満たしていれば、加算と波カッコ内の文が繰り返し実行されます。次に条件がテストされたときに条件を満たしていなければ、そこで繰り返しが終わります。

使用例
```
for ( i = 0; i <= 255; i++){
  analogWrite(9, i);
  delay(100);
}
```

形式　`while(条件) {条件に一致したときに実行する文}`

説明　カッコ内の条件を満たさなくなるまで（`false` になるまで）波カッコ内を永遠に繰り返します。

使用例
```
x = 1;
While( x < 50 ){
  x++;
}
```

時間

形式　`delay(ms)`

説明　カッコ内で指定された時間待って、次の文の処理へ進みます。単位はミリ秒です。

パラメータ　`ms`　次の処理までの待ち時間。単位はミリ秒。（1 ミリ秒は 1000 分の 1 秒です。）

使用例
```
digitalWrite(9,HIGH);
delay(500);
digitalWrite(9,LOW);
```

便利な機能

形式　`#define　定数名　値`

説明　検証時（コンパイル時）に自動的に定数名を値に変換します。プログラムを見やすく、間違いを少なくするために有効です。`#`を付けることを忘れないようにします。

`#define` 文の最後には；（セミコロン）は必要ありません。

使用例
```
#define ledPin 4
digitalWirte(ledPin, HIGH);
```
`ledPin` は検証時に自動的に `4` に置き換えられます。

形式　`#include　<ライブラリ名>`

説明　外部に用意されているライブラリをプログラムに取り入れたいときに使います。

ライブラリは Arduino-IDE にあらかじめ用意されているものや、インターネットから入手できるものなどがあります。

`#include` 文の最後には；（セミコロン）は必要ありません。

使用例
```
#include  <LiquidCrystal.h>
```
LCD 表示器のライブラリを使うことができるようになります。

形式　　// または /* ～ */

説明　　プログラムの中にメモしておきたい内容、つまりコメントを書くときに使います。
コメントは検証（コンパイル）から無視され、プログラムに影響しません。
コメントは、プログラムの動作内容を理解したり、思い出したりするために使われます。

使用例　x = 1;　//ここがコメントになる。　ダブルスラッシュの後ろはすべてコメントです。

```
/*      ここから複数行のコメントになります。
  digitalWrite(3,LOW);
  x = analogRead(A0);
*/      ここまでがコメントです。
```

シリアル通信

形式　　Serial.begin(speed)

説明　　シリアル通信の通信速度を設定します。

パラメータ　speed　通信スピードの値。一般的にパソコンと通信する場合は、
300,1200,2400,4800,9600,14400,28800,38400,57600,11520
から 1 つを選びます。

使用例　Serial.bigin(9600);

形式　　Serial.print(data, format)

説明　　シリアルポートに指定されたデータを出力します。

パラメータ　data　すべての整数型と String 型(*)
(※アルファベット、数字、区切り記号などの文字が連続したもの。
ダブルクオテーションで囲むことで String 型になります。)

format　data を変換する方法を指定します。省略可能です。
省略すると、data は 10 進数の数値として取り扱われます。

使用例　Serial.print(58);　58 を出力します。
Serial.print(58, HEX);　3A を出力します。
Serial.print(“Hello”);　Hello を出力します。

形式　　Serial.println(data, format)

説明　　データの最後に改行コードを付けて出力します。それ以外の動作は Serial.print と同じです。

使用例　Serial.println(58);

データ型

変数そのものや、変数に代入される値の範囲に合わせて、データ型を選ぶ必要があります。
取り扱える範囲を超えた計算を行うと、変数は期待している値と違ったものになるので注意が必要です。
データ型と、値の範囲は以下の表のようになります。

データ型	値の範囲
boolean	true か false のどちらかの値を取り扱います。
byte	0 から 255 までの値を取り扱います。負の値は扱えません。
int	-32,768 から 32.767 までの値を取り扱います。
long	-2,147,483,648 から 2,147,483,647 までの値を取り扱います。

※これ以外にもデータ型はあります。代表的なものだけ紹介しています。

使用例　`int x ;`　　x は int 型の範囲の数値を取り扱う変数になります。
　　`byte y = 128 ;`　　byte 型の変数 y に 128 を代入します。

演算子

形式　＝
説明　定数や計算結果を変数に記憶させるには、＝(等号) を使います。＝は代入演算子といわれます。
使用例　`a = 10;`　　a は 10 になります。
　　`b = a;`　　b は a と同じ値になります。

形式　`+ , - , * , /`
説明　加算、減算、乗算、除算を行います。これらは算術演算子といわれます。
　　データ型によって計算結果が期待しているものと違ってくることがありますので、
　　計算結果を格納するのに十分な大きさの型になっていることに注意が必要です。
　　※プログラムをスマートにする演算子は他にもあります。ここでは基本的なものだけです。
使用例　`a = 10 + 1;`　　a は 11 になります。
　　`a = 9 - 1;`　　a は 8 になります。
　　`a = 3 * 33;`　　a は 99 になります。
　　`a = 9 / 3;`　　a は 3 になります。
　　`a = 9 / 4;`　　a のデータ型によって変わります。a が int 型であれば、
　　a は 2 になります。
　　※少数が取り扱いたい場合は別のデータ型を選ぶ必要があります。

形式　　++　--

説明　　++をインクリメントといい、1 を足します。--をデクリメントといい、1 を引きます。

使用例　++a;　　a に 1 を足して a に記憶します。

　　--a;　　a から 1 を引いて a に記憶します。

比較演算子

形式　　== , != , < , > , <= , >=

説明　　2 つの変数や定数の関係を調べるには、==,!=,<,>,<=,>=を使います。

関係が正しい場合を真(true)といい、正しくない場合を偽(false)といいます。

演算子	使用例	結果
==	a==b	a と b が等しければ真
!=	a!=b	a と b が等しくなければ真
<	a<b	a が b より小さければ真
>	a>b	a が b より大きければ真
<=	a<=b	a が b 以下なら真
>=	a>=b	a が b 以上なら真

使用例　a = (1>2);　　a は false(=0)になる。

a = (1!=2);　　a は true(=1)になる。

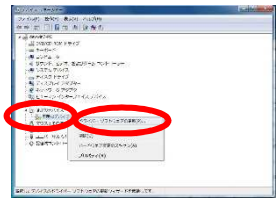
a = (1==2);　　a は false(=0)になる。

トラブルシューティング

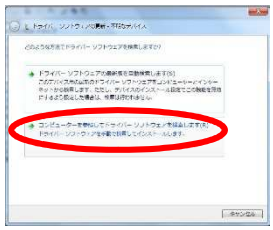
WindowsVista/7 に手動でドライバーをインストールする

（1） コントロールパネル → システムのセキュリティ → デバイスマネージャー と選んで、デバイスマネージャーの画面を開きます。

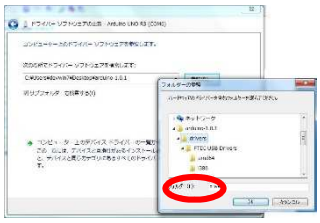
（2） デバイスマネージャー画面の中に、「不明なデバイス」という表示がされていますので、その文字の上で右クリックして、「ドライバーソフトウェアの更新・・・」を選びます。



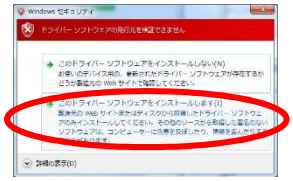
（3） 「ドライバーソフトウェアの更新」画面から、「コンピューターを参照してドライバーソフトウェアを検索します。」を選びます。



（4） 「次の場所でドライバーソフトウェアを検索します。」→「参照」を選び、最初に保存した Arduino のフォルダーの中から Drivers を選びます。



(5) Windows セキュリティの画面が表れますので、「このドライバーソフトウェアをインストールする。」を選びます。



(6) しばらく待つと、ドライバーのインストール完了画面が表示されます。

(7) もう一度デバイスマネージャーを表示して、Arduino 基板が、ポート（COM と LPT）に登録されていることを確認してください。