

マイコン講習マニュアル

京都大学機械研究会
工学部情報学科
松本直樹

1 はじめに

マイコンとは、いわゆるマイクロコンピューターの略称であり（諸説あり）、冷蔵庫や洗濯機などありとあらゆる電気製品に組み込まれているコンピュータである。

京大機械研究会では Arduino を利用してロボットの制御を行っている。今回の講習では Arduino の基本的な利用方法とプログラミングの基本を会得することを目的としている。

2 Arduino とは？

Arduino とは AVR マイコンを利用した、組み込みマイコンのシステムの一種である。Arduino では Arduino IDE なる開発環境を無償で利用することができ、C/C++ ライクな開発言語で開発することが出来る。従来の PIC や AVR とは大きく異なり、Arduino というプラットフォーム形態を確立しているため、抽象度の高いプログラミングが可能であり、初心者にとっては非常に扱いやすいシステムである。

Arduino で利用可能な機能としては、A/D 変換、PWM、I2C、SPI など、機械研において制作するロボットに必要な機能はほぼ網羅している。

3 機械研マイコンボード

今回の講習で利用するボードは、回路講習で作った機械研マイコンボードである。このマイコンボードは部内ロボコンだけでなく、合同ロボコンや NF での展示物にも利用できるため、是非使い方を習得し、製作の役に立ててもらいたい。

3.1 回路図

機械研マイコンボードの回路図は以下のとおりである。

3.2 実体配線図

機械研マイコンボードの実体配線図は以下のとおりである。

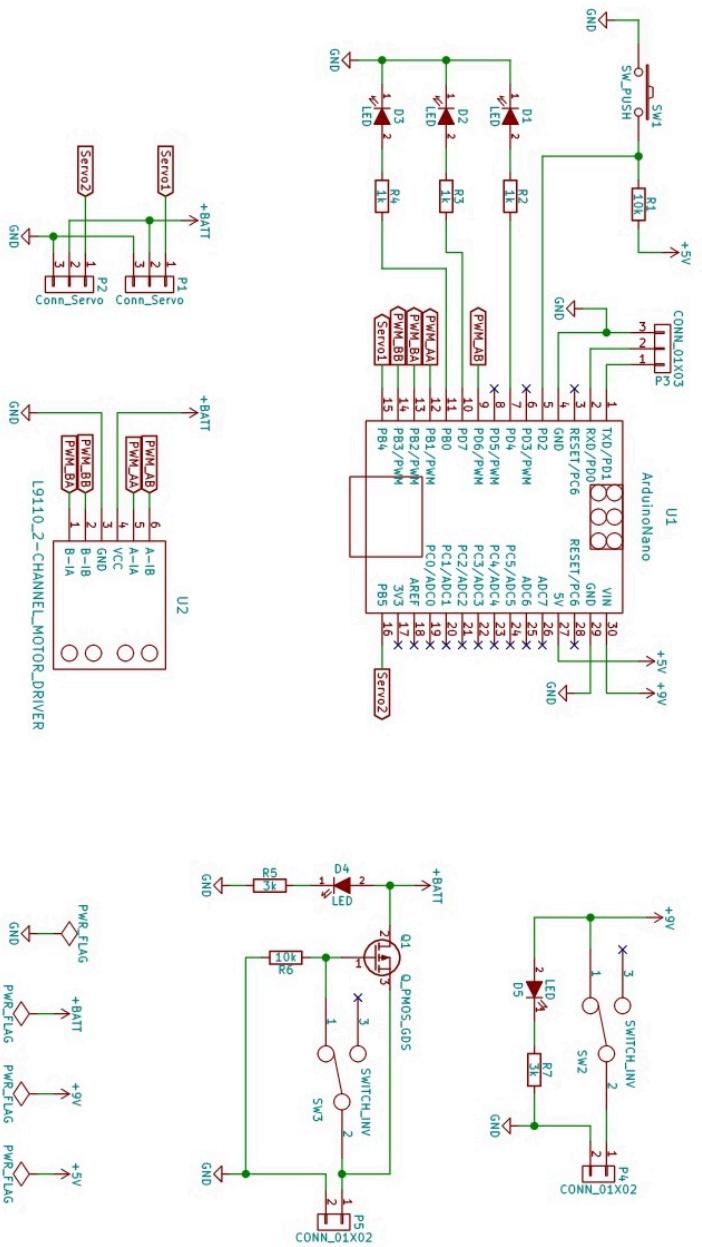


図1 機械研マイコンボード回路図

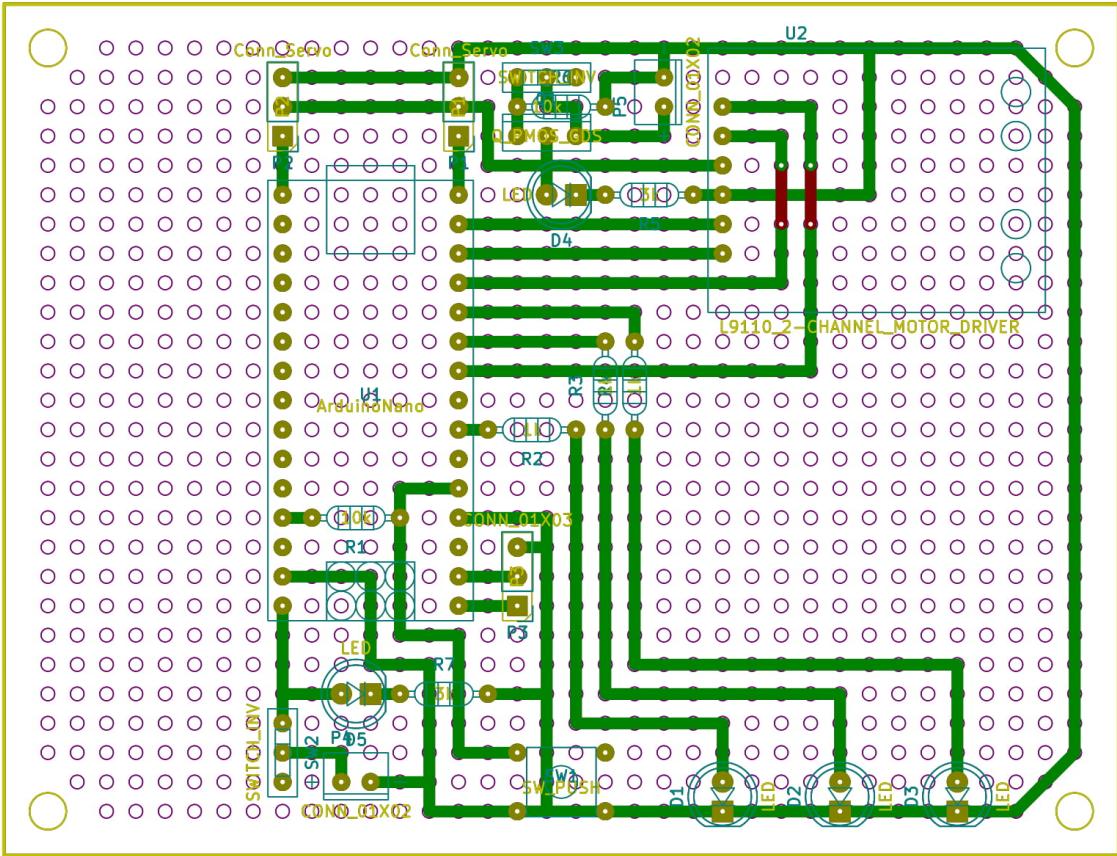


図 2 機械研マイコンボード実体配線図

3.3 利用可能な機能

Arduino nanoにおいて利用可能な機能は以下のとおりである。

- デジタル I/O ピン 14 本
- PWM 6 本
- アナログ入力ピン 8 本 (AD 変換精度 10bit)
- タイマー 3 つ (8bit 2 つ・16bit 1 つ)
- I2C 1 つ
- SPI 1 つ
- UART 1 つ

タイマーは PWM と競合するため、同時に利用する際は注意が必要である。また、I2C や SPI、UART もデジタル I/O ピン、アナログ入力ピンと重複しているため、同時に使えるピンに気をつける必要がある。

以下に Arduino nano におけるピンの機能割当の図を示す。

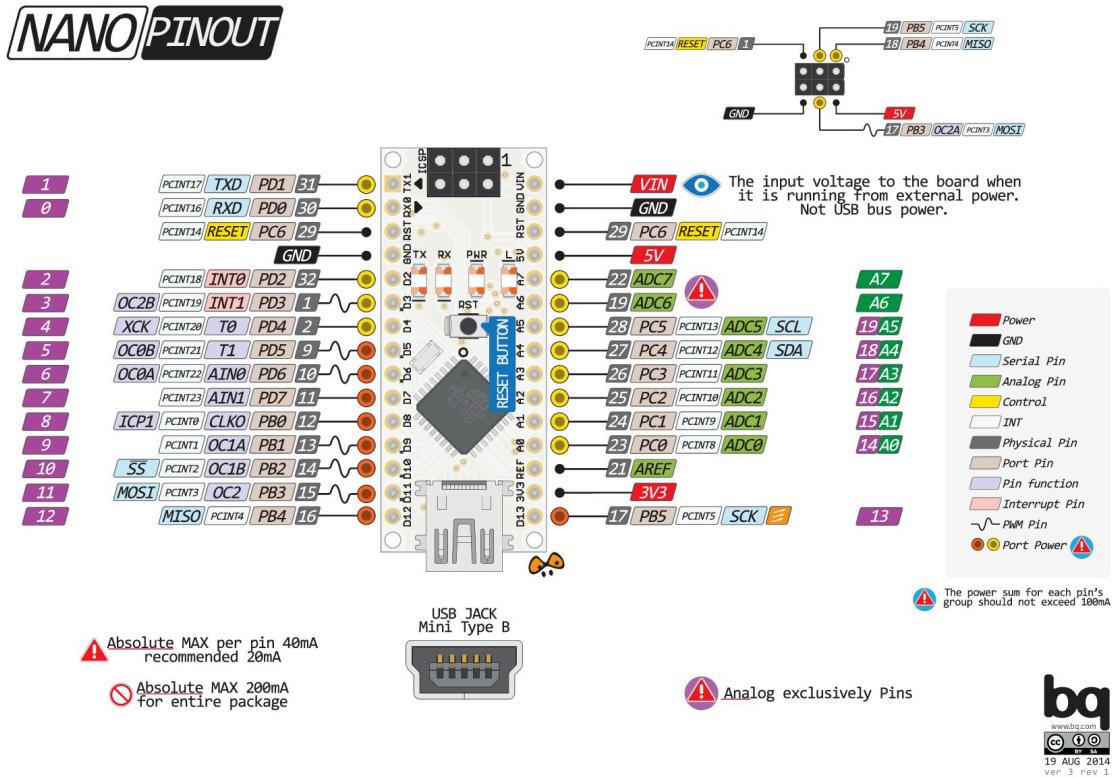


図3 Arduino nano におけるピンの機能割当

4 開発環境の構築

手元のPC上にArduinoの開発環境を構築する。

4.1 ドライバのインストール

機械研マイコンボードでは純正Arduinoとは異なり、ドライバを別途インストールする必要がある。ドライバを以下の手順でインストールする。

1. 「CH340 ドライバ」と検索する。検索結果から「CH340 Drivers for Windows, Mac and Linux」を選択する。
2. Windowsの項目から「Windows CH340 Driver」を選択する。すると、ドライバファイルがダウンロードされる。
3. ダウンロードしたファイルはZIPで圧縮されているため、解凍後、「CH430 Install Windows v3.4.EXE」を実行する。
4. 管理者権限を許可すると図のようなウィンドウが現れる。ここで「INSTALL」ボタンを選択する。
5. 成功したような雰囲気のするダイアログが出ればインストールは完了である。



図4 「CH340 ドライバ」と検索する

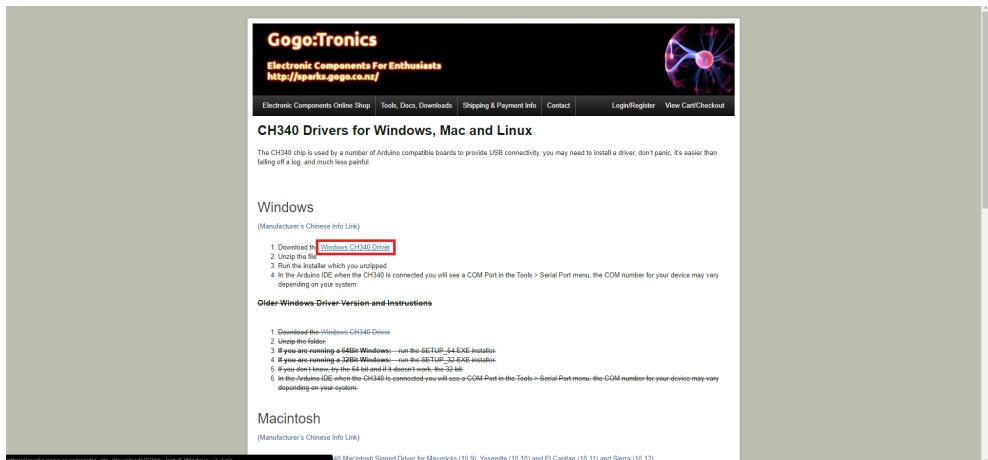


図5 「Windows CH340 Driver」を選択し、ドライバをダウンロードする

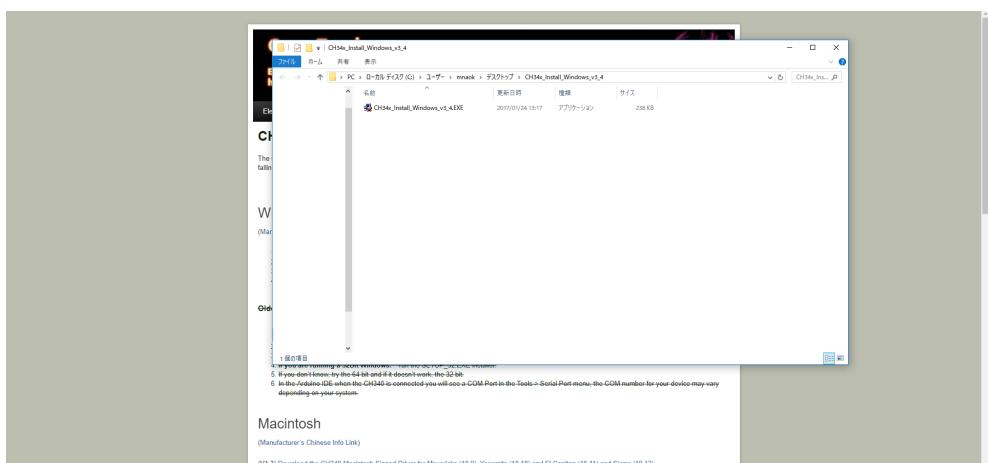


図6 解凍後、「CH340 Install Windows v3.4.EXE」を実行する

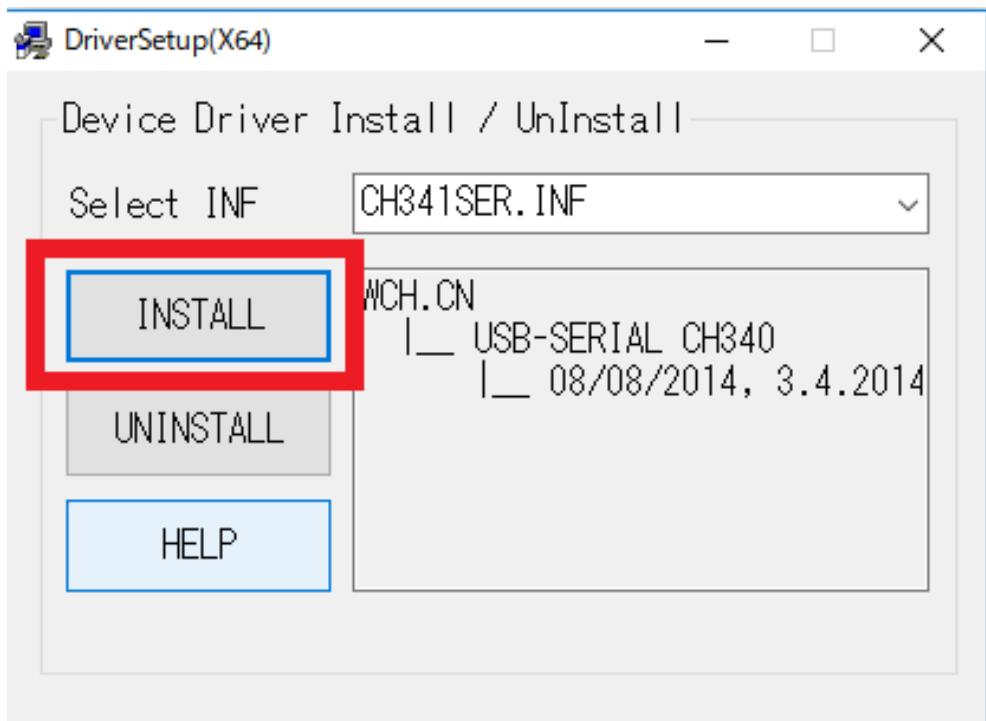


図7 「INSTALL」ボタンを選択する

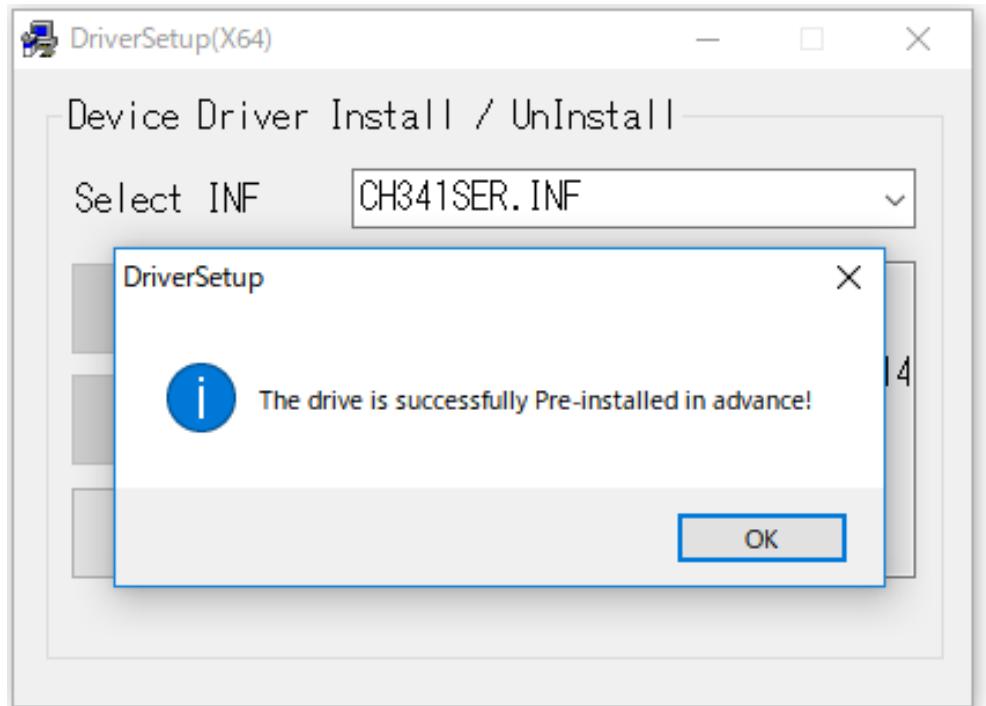


図8 いい感じのダイアログ



図9 「Arduino IDE」と検索し、「Arduino - Software」を選択する

4.2 Arduino IDE(統合開発環境)のインストール

次に、Arduino IDE という Arduino 専用の開発環境をインストールする。

Vimなどのエディタでも開発することは出来るが難易度の問題上、今回は割愛する。

1. 「Arduino IDE」と検索し、「Arduino - Software」を選択する。
2. 「Windows Installer, for Windows XP and up」を選択する。
3. 「JUST DOWNLOAD」を選択する。寄付をしたい人はしてもよいが、今回は割愛する。
4. ダウンロードした実行形式ファイルを実行する。利用規約をしっかり読み、承諾できるならば「I Agree」を選択する。
5. デフォルトのままで「Next」を選択する。
6. パスの設定をする。デフォルトのままで良いが、変えたい人は変えてても良い。「Install」を選択する。
7. インストールが始まる。
8. 完了すれば「Completed」と表示される。「Close」を選択し、インストーラーを終了する。

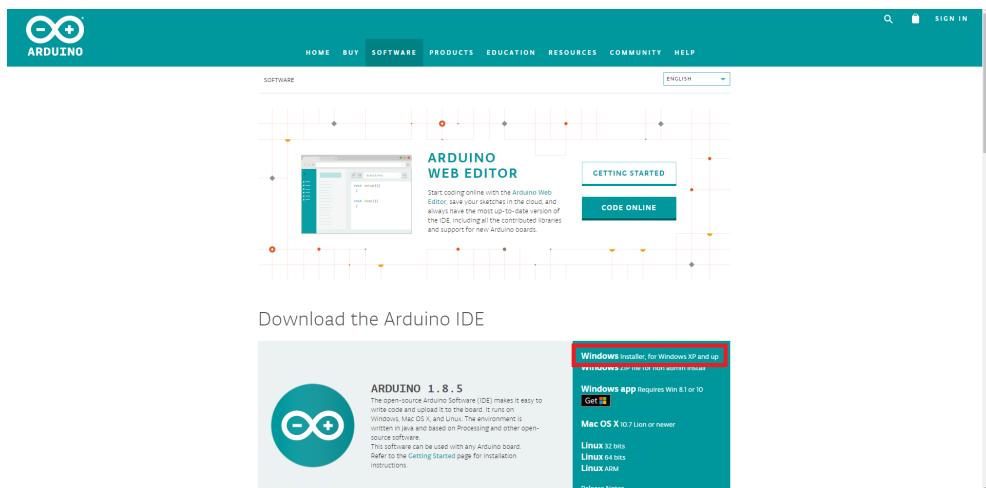


図 10 「Windows Installer, for Windows XP and up」を選択する

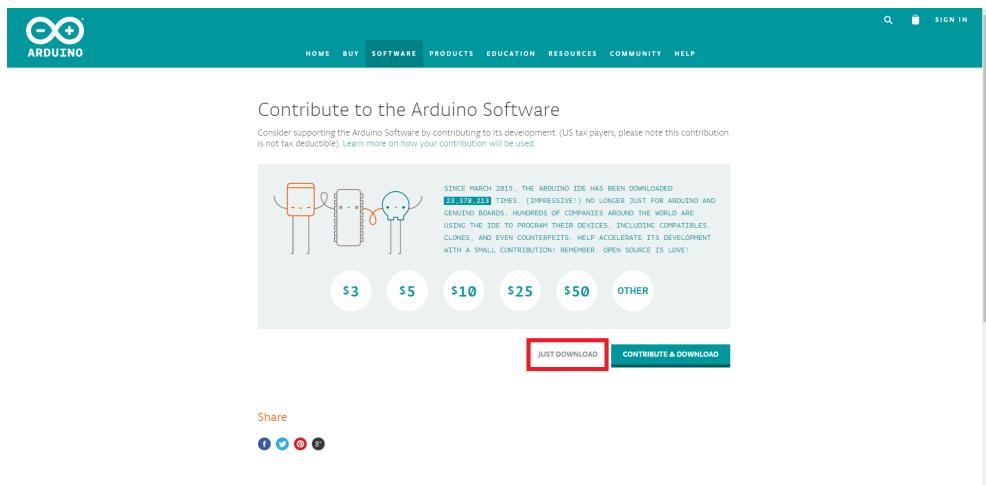


図 11 「JUST DOWNLOAD」を選択する

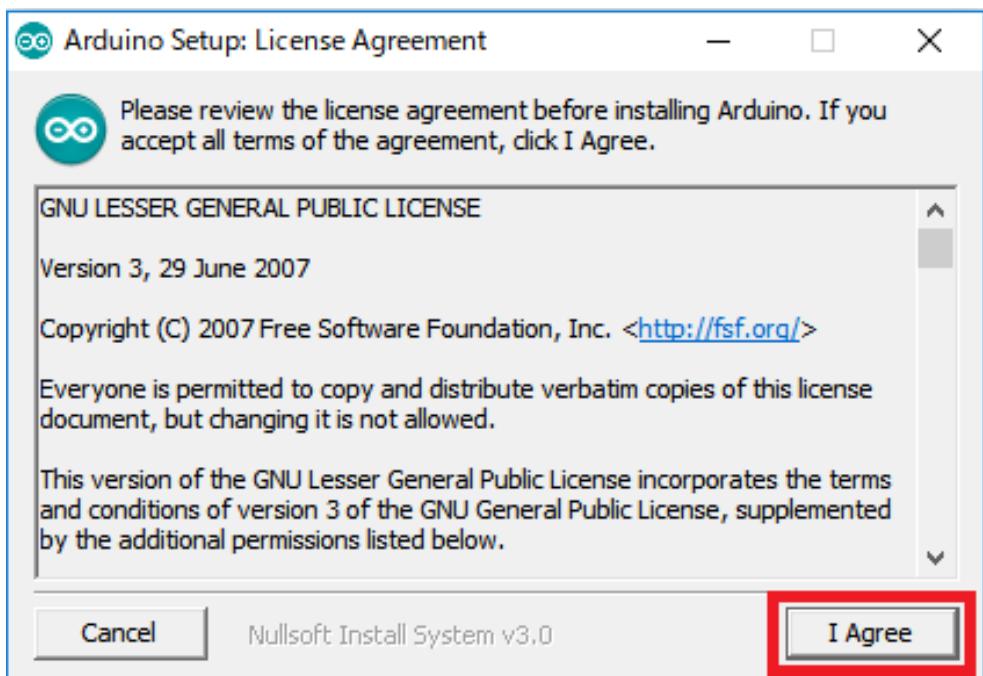


図 12 利用規約をしっかり読み、承諾できるならば「I Agree」を選択する

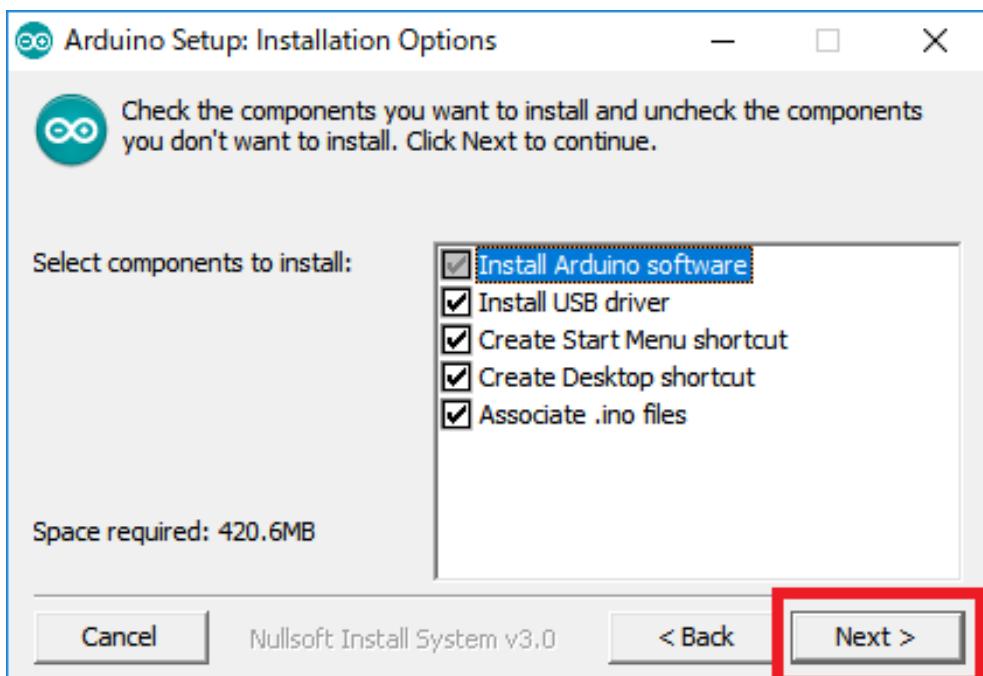


図 13 デフォルトのままで「Next」を選択する

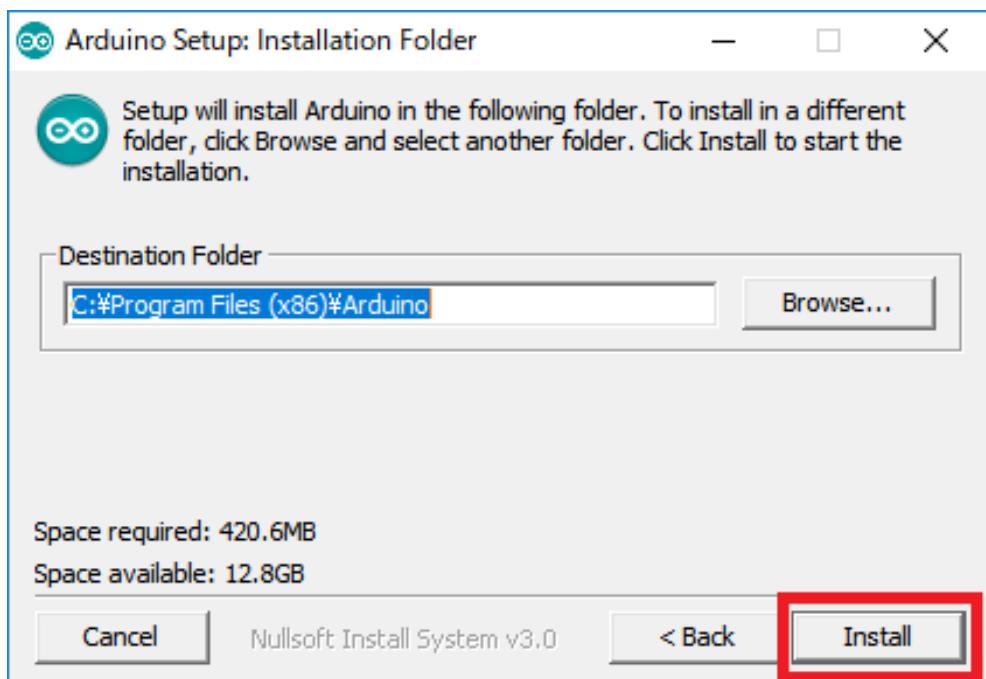


図 14 パスの設定をし、「Install」を選択する。

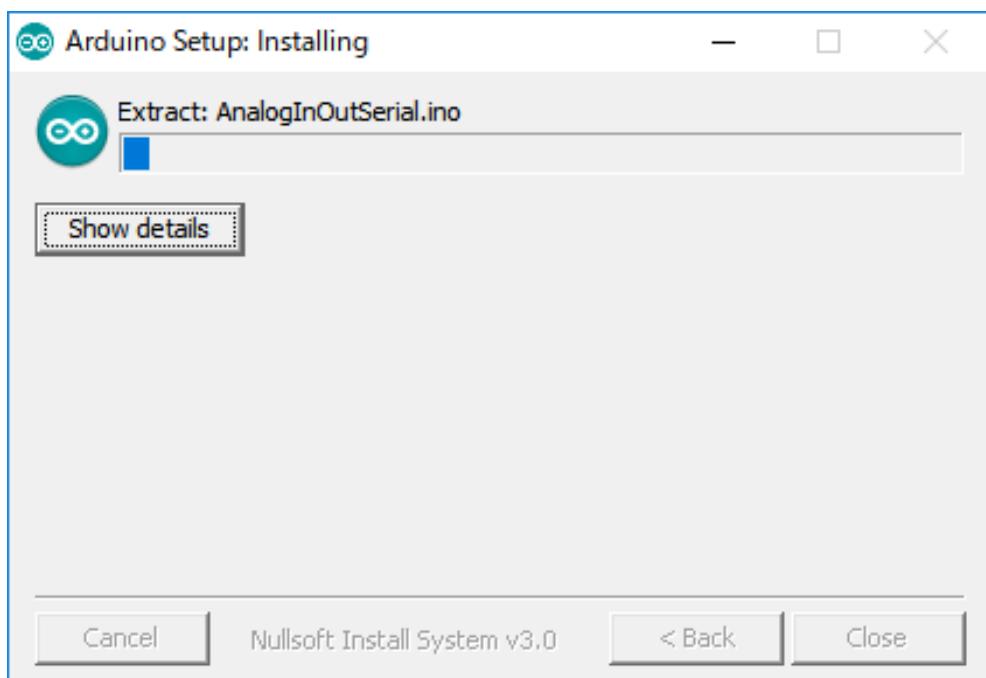


図 15 インストールが始まる。

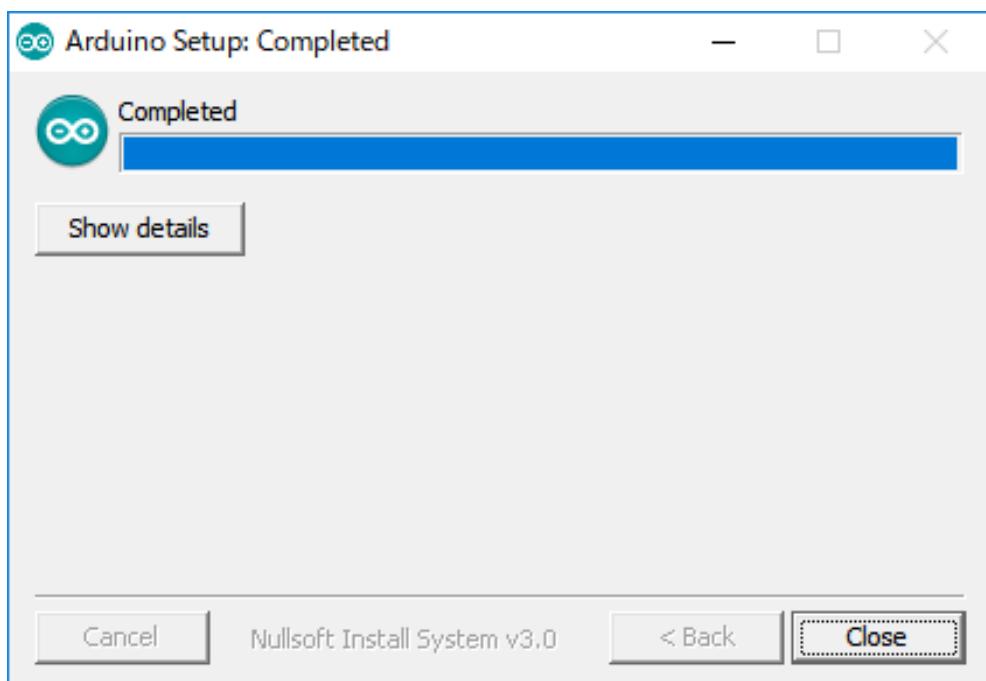


図 16 完了すれば「Completed」と表示される。「Close」を選択し、インストーラーを終了する

5 Arduino IDEについて

5.1 使い方

では、実際にパソコンにマイコンボードを接続し、Arduino IDEを起動してみよう。起動すると以下のようないい画面が表示される。

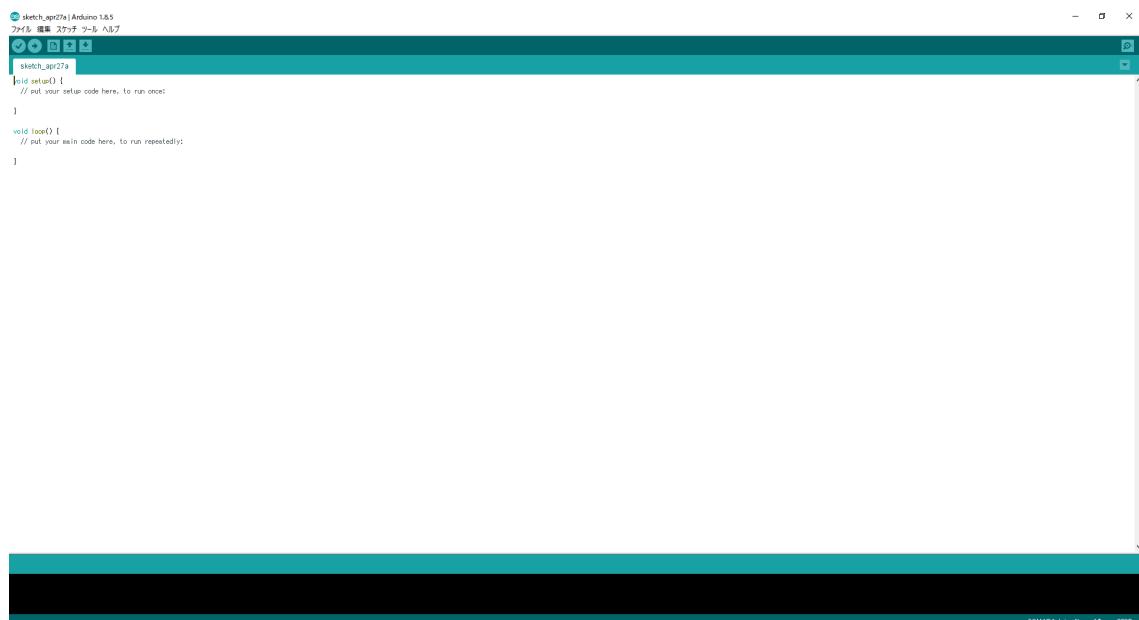


図 17 Arduino IDE

画面の大部分を占める白い部分はエディタ領域である。ここにプログラムのソースコードを書いていく。下部の黒い部分はログビューである。Arduino IDEが実行した作業に関するログ（結果）が表示される。



図 18 ボタン類

左上のボタンについて説明する。

①は「コンパイル実行」ボタンである。コンパイルとはプログラムをマイコンで実行可能な形式に変換することである。プログラムの文法に誤りがある場合は、コンパイルが失敗し、誤り箇所をログビューに表示してくれる。

②は「マイコンへの書き込み」ボタンである。プログラムをコンパイルし、マイコンボードへの書き込みまで一括で行ってくれる。

③は「新規作成」ボタンである。新たにウィンドウが表示され、別のプログラムを書くことが出来る。

④は「開く」ボタンである。既存のソースコード（プログラムを記したファイルのことである）を開き、編集することが出来る。

⑤は「保存」ボタンである。現在編集しているソースコードを保存する。なお、コンパイル時、もしくは、マイコンへの書き込み時にソースコードは自動で保存される。

5.2 マイコンボードの設定

Arduino IDE 起動時にはマイコンボードの書き込み設定を行う必要がある。設定は以下の手順で行う。

1. シリアルポートの設定を行う。上部メニューbaruから「ツール」→「シリアルポート」とたどり、「COM X」（X は数字）を選択する。ここに現れていない場合は、マイコンが正常に認識されていない。マイコンボードの電源がついているか、ドライバをインストールしたか確認する必要がある。よくわからない人は暇そうな先輩に聞くこと。

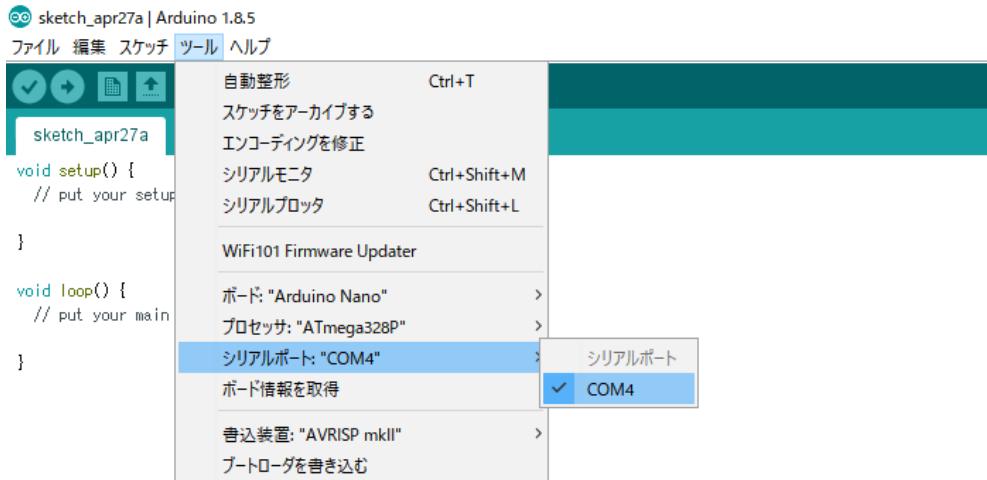


図 19 シリアルポートの設定

2. マイコンボードの種類を設定する。上部メニューbaruから「ツール」→「ボード」とたどり、「Arduino Nano」を選択する。

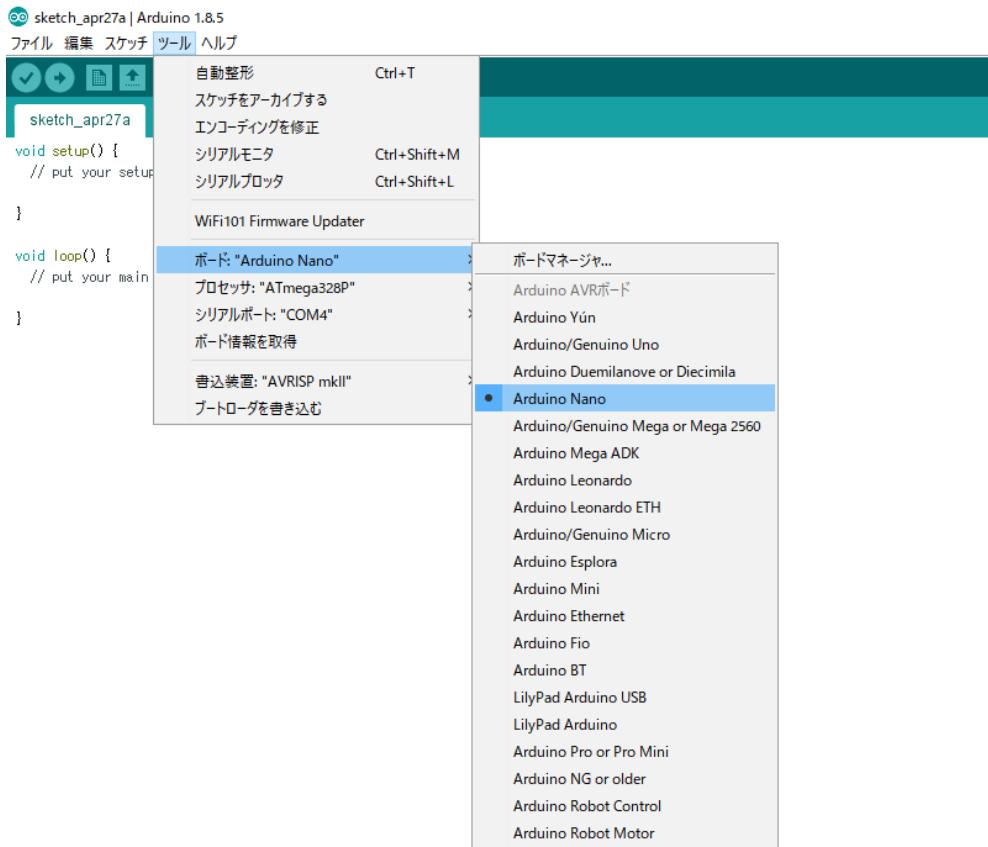


図 20 ボードの種類の設定

6 C 言語について

本講習はプログラミングを全くしたことがない人を対象にしているが、C 言語について教えるとなると時間も労力もかなり必要になるため Arduino C 特有の部分のみを重点的に説明している。C 言語について学びたい人は「苦しんで覚える C 言語」(<https://9cguide.appspot.com/>) や「やさしい C」(高橋麻奈 著) を読むと良いだろう。

7 Arduino C プログラミング

習うより慣れろということで、実際にプログラミングをしよう。なお、「Arduino 日本語リファレンス」(<http://www.musashinodenpa.com/arduino/ref/>) を読みながら進めることを強く推奨する。

7.1 L チカ (LED チカチカ)

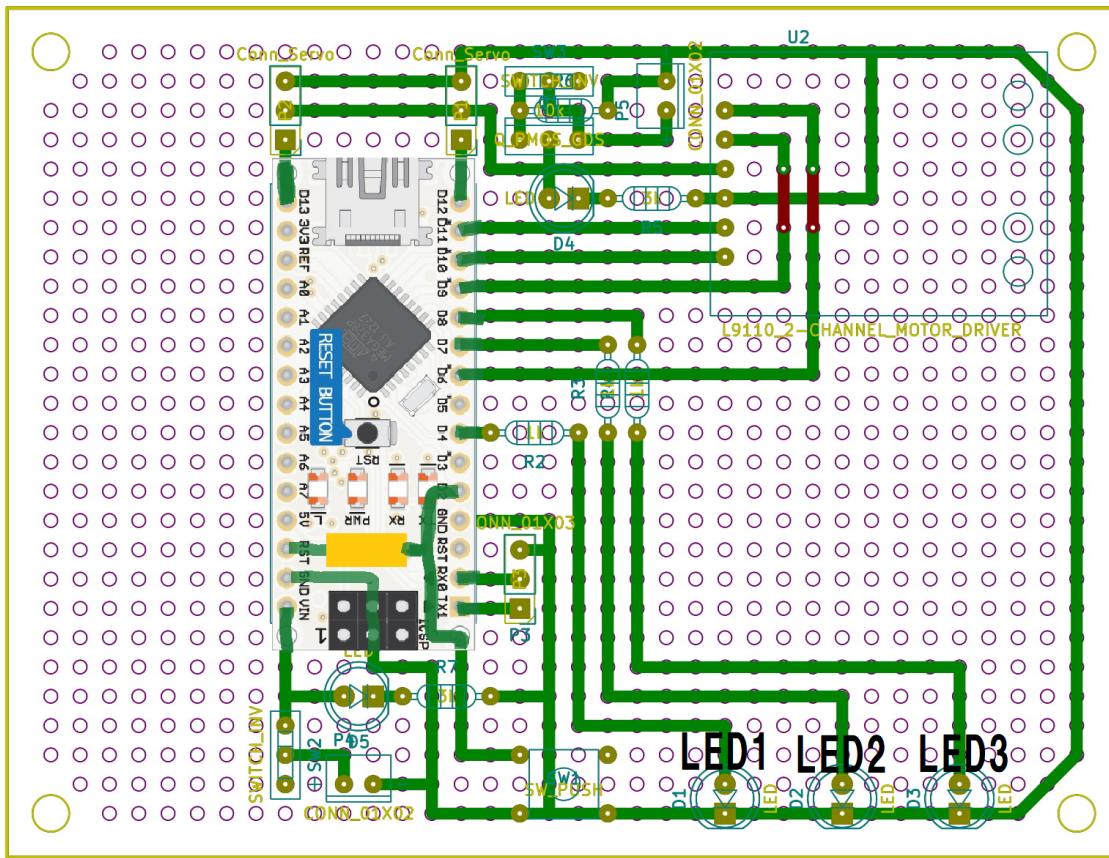


図 21 Arduino Nano と LED の接続状態

LED と Arduino Nano の接続状況は上図のとおりである。LED1 は D4 ピン、LED2 は D7 ピン、LED3 は D8 ピンに接続されている。

次に回路図の LED 部分を見てみよう。LED のカソードは GND に接続されている。このとき LED を点灯させるにはどうすれば良いだろうか？はんだ付け講習で LED はダイオードであり一方向にしか電流を流さないことを学んだはずである。アノード側に+を、カソード側に- (GND) をつなげれば、点灯したはずだ。つまり、(アノード側の電位) - (カソード側の電位) \geq (LED の順方向電圧) となれば LED は点灯する。詳細な内容は別途、松本以外の誰かが説明してくれるはずである。

要するに、マイコンボード側から LED に十分な電位をかけてやればいいわけである。そのために利用するのがデジタル I/O ピンである。

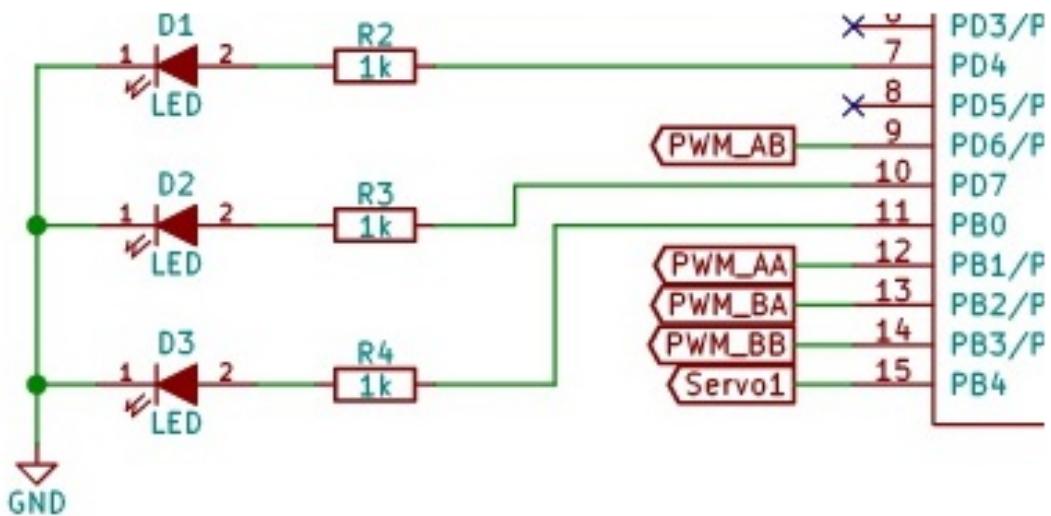


図 22 回路図上の Arduino Nano と LED 接続部分

7.1.1 DigitalPinについて

DigitalPin は主に、pin を 5V の電源や GND へ接続もしくはpin に対して 5V、0V のどちらの電圧がかかるか知ることが出来る I/O ピンである。今回は 5V を出力するために利用する。

7.1.2 プログラム

それでは、以下のプログラムを書き写してみよう。

Listing 1 L チカプログラム

```
#define LED1 4
#define LED2 7
#define LED3 8

void setup() {
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(LED3, OUTPUT);
}

void loop() {
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    delay(1000);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    delay(1000);
}
```

書き写せたら、マイコンボードに書き込んでみよう。エラーなく書き込むことができ、3つのLEDが全て点

灯しているだろうか？問題ないならば、実際にプログラムがどのようにになっているのか見ていく。

7.1.3 ;(セミコロン)について

;(セミコロン)はC言語における、重要な記号である。実行する関数や式の末尾につけることで、「この関数（式）はここで終わり」という表現になる。関数とは実行する操作の書かれた1つのまとまりのことである。付け忘れると、複数行が1つの関数もしくは式として解釈され、コンパイルエラーになる。コンパイルエラーが出る場合はまず、セミコロンを確認しよう。

7.1.4 #defineについて

#defineとはマクロと呼ばれ、端的に言うと、「定義」である。つまり、マクロ名と定義した値や式は同一のものとして扱われるわけである。コンパイル時にマクロ名の部分は定義したものに全て置き換えられる。つまり、

```
#define N 10  
p = N + N
```

は

```
p = 10 + 10
```

と等しいものである。使うときは

```
#define マクロ名 定数や式、文字など
```

のようにして使う。

7.1.5 void setup(){}について

void setup(){}とは、マイコン起動時に1度だけ実行される関数である。使うときは

```
void setup(){  
    (Write code here you want to execute)  
}
```

のようにする。

7.1.6 void loop(){}について

void loop(){}とは、setup()関数が実行された後に、恒久的に実行され続ける関数である。今回のプログラムでは、LEDの点灯と消灯が無限ループしている。使うときは

```
void loop(){  
    (Write code here you want to execute)  
}
```

のようにする。

7.1.7 pinMode()について

pinMode()関数は、マイコン起動時に1度だけ実行されている。この関数は、digitalPinのモードを決定するものであり、digitalPinのモードとしては

- INPUT
- OUTPUT

がある。

INPUT は `digitalRead()` 関数と組み合わせて使われることが多い。これに関しては後述する。

OUTPUT は出力するモードであり、`digitalWrite()` 関数と組み合わせて使う。使うときは `pinNumber`(ピン番号) と `Mode` を指定し、

```
void setup(){
    pinMode(pinNumber, Mode);
}
```

のようにする。具体的には、

```
void setup(){
    pinMode(1, OUTPUT);
}
```

とすると、D1 ピンを出力モードに設定したことになる。なお、ピン番号は DX の X である。アナログ入力用の AX(X は数字) はピン番号として AX をそのまま指定する。

7.1.8 `digitalWrite()` について

前述の `pinMode()` 関数で出力モードに設定したピンで使うことの出来る関数である。デジタル出力ピンの出力を決定できる。なお適切に `pinMode` を設定すればアナログ入力ピンでも利用可能。出力できる内容としては、

- HIGH (主に 5V 出力)
- LOW (0V、GND に接続された状態)

となっている。使うときは `pinNumber`(ピン番号) と `Mode` を指定し、

```
void loop(){
    digitalWrite(pinNumber, Mode);
}
```

のようにする。具体的には、

```
void loop(){
    pinMode(1, HIGH);
}
```

とすることで、D1 ピンに 5V が出力された状態となる。

7.1.9 `delay()` について

`delay()` 関数は「何もせずに待機」する関数である。使うときは時間 `wait_time` (ms 単位) を指定し、

```
delay(wait_time);
```

のようにする。具体的には、

```
delay(100);
```

とすることで、100ms 待機、つまり 0.1 秒待機することになる。

7.1.10 練習問題

LED1 から LED3 へと順番に点灯し、LED3 から LED1 へと順番に消灯するプログラムを書こう！

7.2 スイッチ入力

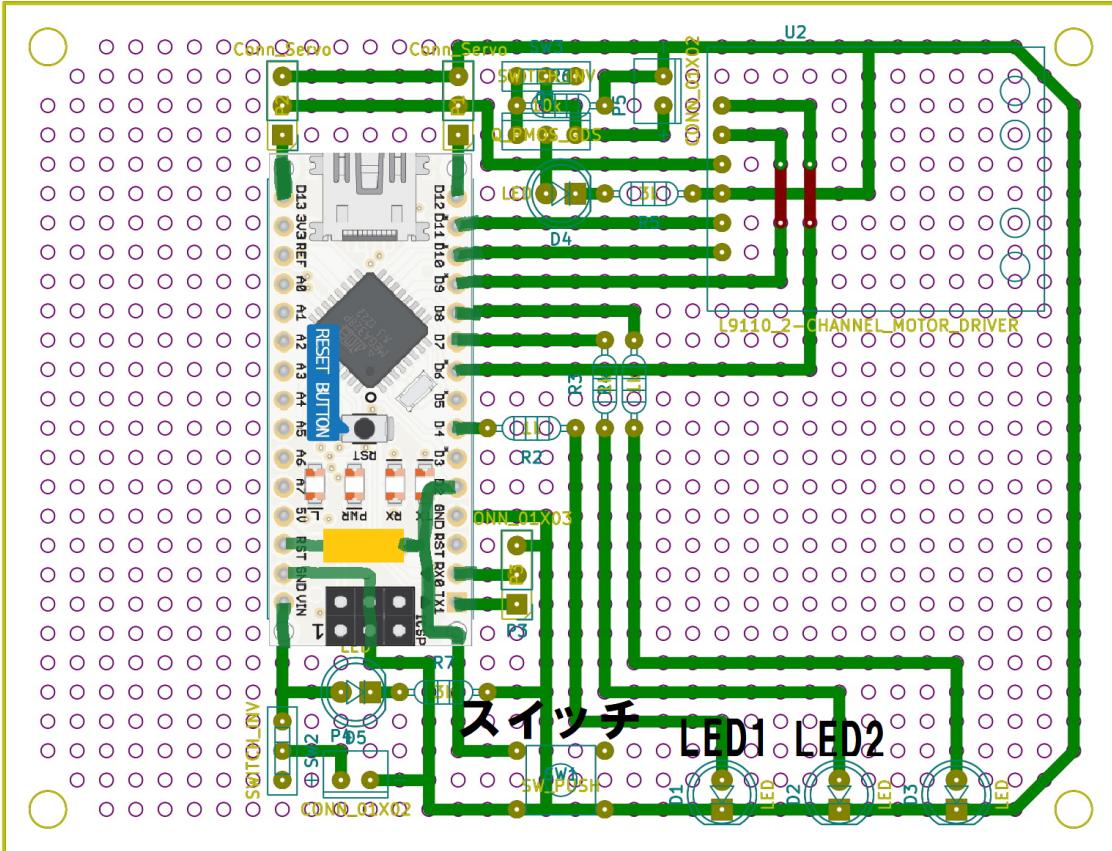


図 23 Arduino nano とスイッチの接続状態

プッシュスイッチは D2 ピン、LED1 は D4 ピン、LED2 は D7 ピンに接続されている。

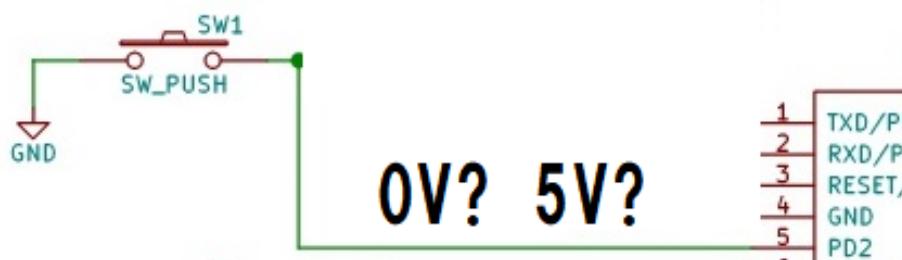
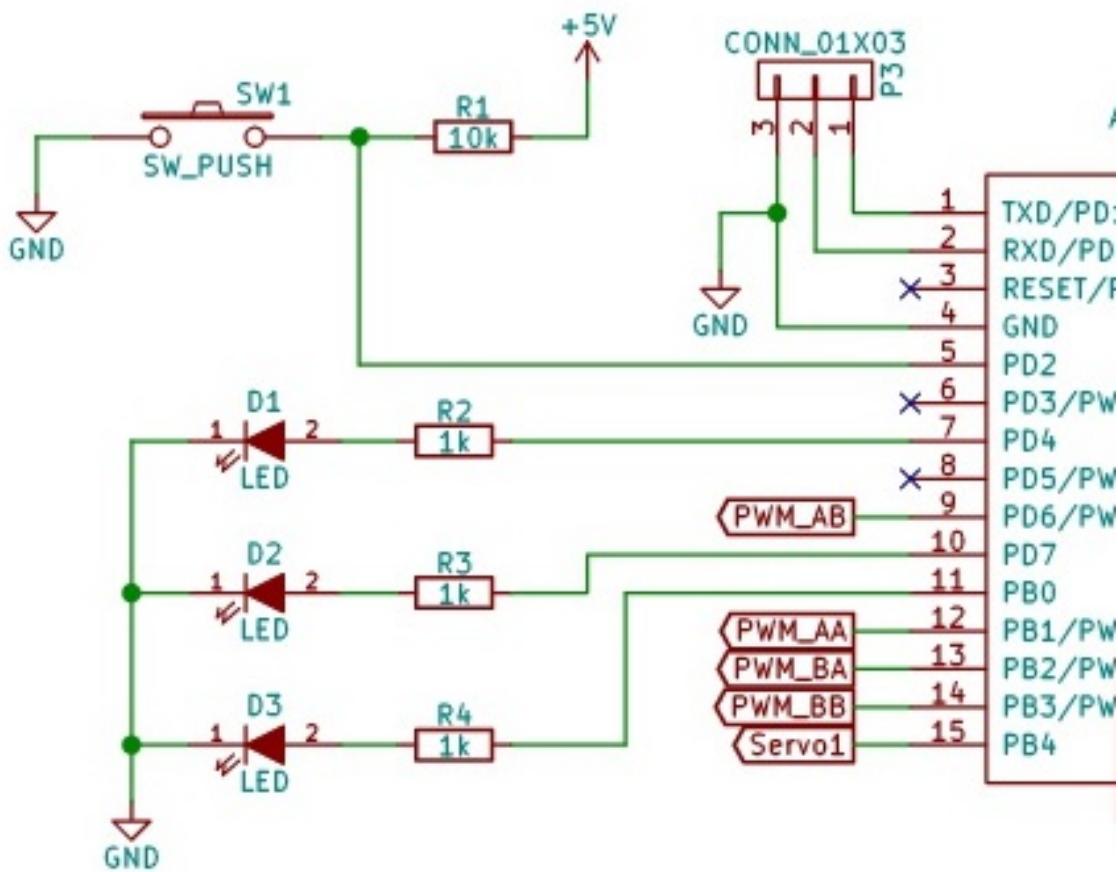
次に回路図のスイッチと LED 部分を見てみよう。ここで重要なのが、スイッチに直列に入った抵抗器 R1 である。この状況で使われる抵抗器はプルアップ抵抗と呼ばれ、マイコンの誤作動を抑制するためのものである。

その原理を見ていこう。

まず、プルアップ抵抗がない場合でスイッチが OFF の状態を考える。図のように D1 ピンには何も接続されていない状態となり、0V でも 5V でもない状態となる。これはマイコン的には非常にまづい状態であり、ON とも OFF とも認識されてしまう状況である。つまり、誤作動が起きやすい状態なのである。(厳密な原理としてはマイコンのインピーダンスの高さが原因であるが、今回は割愛する。)

そこで、プルアップ抵抗を挿入してみる。図を見れば分かるように、スイッチが OFF の状態では D1 ピンに抵抗 R1 からの 5V(厳密には 5V より少し低い) がかけられた状態になり、マイコンが正常に状態を認識することが出来る。

スイッチが ON になった場合は図のようになる。スイッチが短絡し、D1 ピンは GND に繋がった状態になるため、マイコンが D1 ピンは 0V であると認識することが出来る。



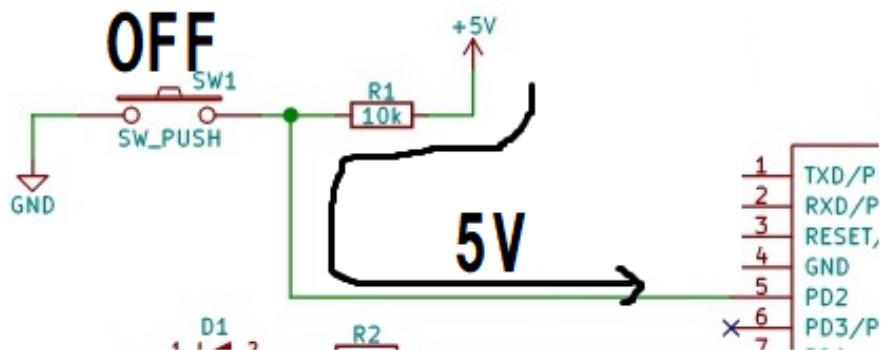


図 26 プルアップ抵抗がある状態でのスイッチ OFF

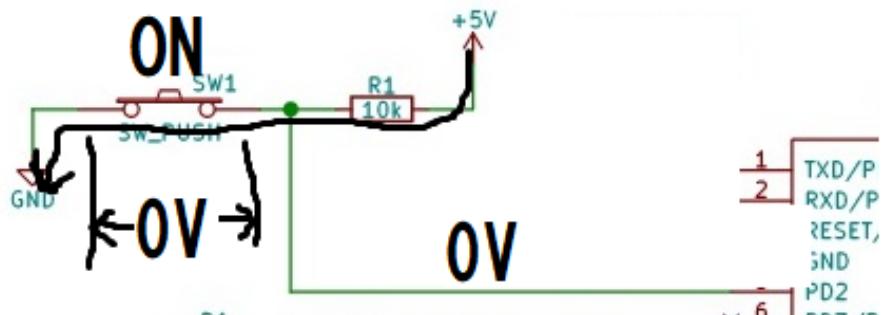


図 27 プルアップ抵抗がある状態でのスイッチ ON

7.2.1 プログラム

それでは、また以下のプログラムを書き写してマイコンに書き込んでみよう。

```
#define LED1 4
#define LED2 7
#define SW 2

void setup() {
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(SW, INPUT);
}

void loop() {
    if(digitalRead(SW) == 1){
        digitalWrite(LED1, HIGH);
        digitalWrite(LED2, LOW);
    }else{
        digitalWrite(LED1, LOW);
        digitalWrite(LED2, HIGH);
    }
}
```

正常に書き込みができたら動作を確認してみよう。プッシュスイッチを押していない状態では LED1 だけが

点灯し、押した状態では LED2 だけが点灯するはずだ。

7.2.2 digitalRead()について

前述したとおり、digitalRead() 関数は pinMode() 関数で INPUT モードに設定したデジタルピンで利用可能である。なお適切に pinMode を設定すればアナログ入力ピンでも利用可能。digitalRead() 関数は読み取ったピンの状態に応じて 0 もしくは 1 を戻り値とする。戻り値とは関数実行後に outputされる値のことである。具体的には、戻り値が 1 な関数 foo() があるとすると、

```
int p = foo();
```

は

```
int p = 1;
```

と等価である。digitalRead() 関数はピン番号を pinNumber とすると、

```
void loop(){
    int p = digitalRead(pinNumber);
}
```

のようにする。具体的には、

```
void loop(){
    int p = digitalRead(1);
}
```

となる。D1 ピンに 5V がかかっていれば 1 が戻り値となり、0V の場合は 0 が戻り値となる。

7.2.3 if(...){...}else{...}について

digitalRead() 関数で取得した値を利用する方法の 1 つとして if 文がある。if 文は次のような構造になっている。

```
if (conditional expression){
    (conditional expression is true, code here is executed)
} else {
    (conditional expression is false, code here is executed)
}
```

if 文は条件式 (conditional expression) が正 (true) であれば、すぐ下の範囲のプログラムが実行され、偽 (false) であれば、else 以下のプログラムが実行される。条件式に使えるものとしては代表的には以下のものが挙げられる。

- == a==b(a=b ならば true)
- != a!=b(a と b が等しくなければ true)
- > a>b(a より b が大きければ true)
- < a<b(a より b が小さければ true)
- <= a<=b(b が a 以上であれば true)
- >= a>=b(a が b 以上であれば true)
- && (AND) A&&B(A と B が両者ともに true であれば true)

- $\|(\text{OR}) A||B$ (A と B の少なくとも一方が true であれば true)

具体的には以下の通りである。

```
1 == 2 => false
4 == 4 => true
1 > 2 => false
1 < 2 => true
1 <= 1 => true
4 >= 5 => false
(1!=2)&&(5>2) => true
(1==3)&&(4<7) => false
(1>=2)|| (5>2) => true
(2>2)|| (7<2) => true
```

7.2.4 練習問題

1. スイッチを押して離す度に点灯する LED が右にずれていくプログラムを書こう！

7.3 PWM(サーボモータ編)

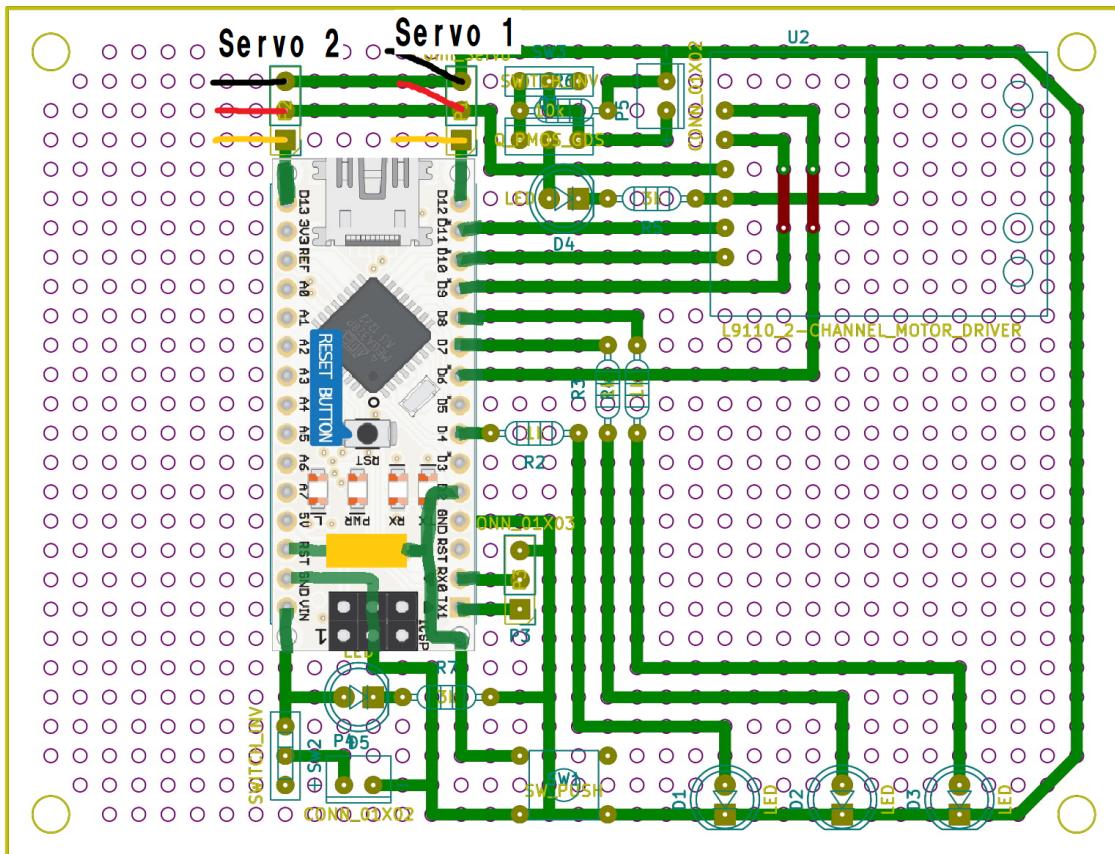


図 28 サーボモータと Arduino の接続状態

サーボモータと Arduino の接続状態は上図のとおりである。

7.3.1 サーボモータとは

サーボモータとは、マイコンから与えられた角度に軸を回転させ、保持する部品である。

サーボモータにはいくつかの種類があるが、機械研内で最も利用されているアナログサーボについて説明する。

アナログサーボの制御には PWM というものが用いられる。PWM とは ON と OFF の比率 (Duty 比) を変化させながら高速にスイッチングする機能である。アナログサーボモーターではこの Duty 比を変化させることで角度を指定している。Arduino では Servo オブジェクトを用いることで簡単に制御することが出来る。

732 プログラム

それでは、例のごとく以下のプログラムを書き写そう。

```
#include <Servo.h>
Servo servo1;
```

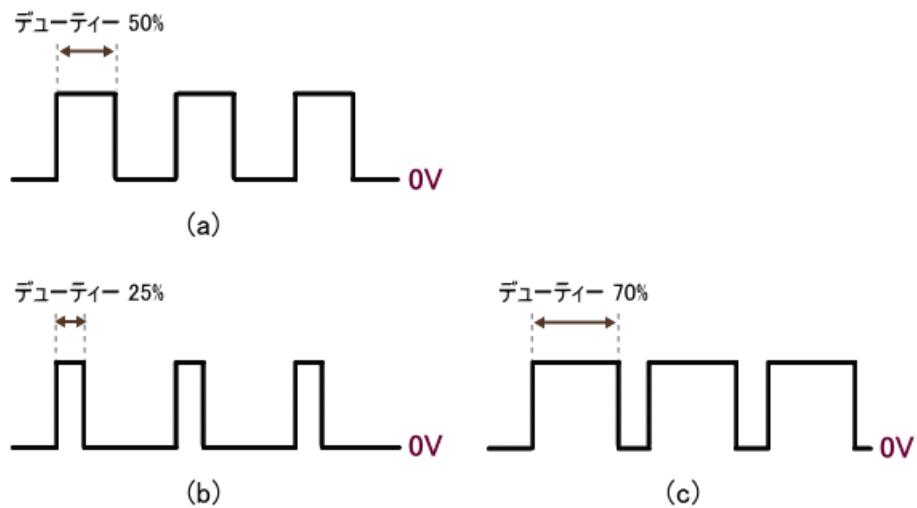


図 29 PWM の波形 (http://www.kairo-nyumon.com/pwm_signal.html より)

```

Servo servo2;
void setup() {
    servo1.attach(12);
    servo2.attach(13);
}

void loop() {
    servo1.write(30);
    servo2.write(30);
    delay(500);
    servo1.write(60);
    servo2.write(60);
    delay(500);
    servo1.write(90);
    servo2.write(90);
    delay(500);
    servo1.write(120);
    servo2.write(120);
    delay(500);
    servo1.write(150);
    servo2.write(150);
    delay(500);
}

```

書き写せたならばマイコンボードに書き込もう。書き込みが完了するとサーボモーターが少しづつ回転しているはずである。

7.3.3 #includeについて

#include とは主にライブラリの読み込み指定に用いられる。ライブラリとは関数やクラスなどをまとめたものである。サーボモーターを制御する Servo クラスでは別途ライブラリを読み込む必要があるため、以下のコードをプログラムの先頭に挿入する必要がある。

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

7.3.4 Servo クラスについて

クラスについて説明するとなると莫大な労力を要するため詳しく知りたい人は別途自分で調べること。ここでは具体的な利用方法についてのみ説明する。

まず、使いたいサーボの個数分だけ Servo クラスの変数 (インスタンス) を宣言する。

```
Servo servo1  
Servo servo2
```

ここでは、サーボを 2 つ用いるため、servo1 と servo2 の 2 つのインスタンスを宣言した。次に、各インスタンスに制御するピンを割り当てる。ピンを割り当てるときに使うのが attach() 関数である。以下のようにしてピンを割り当てる。

```
void setup() {  
    servo1.attach(12);  
    servo2.attach(13);  
}
```

ここでは servo1 に D12 ピンを、servo2 に D13 ピンを割り当てる。ピンの割当ができれば、次はサーボモータの角度を指定する。このときに使うのが write() 関数である。

```
void loop(){  
    servo1.write(30);  
    servo2.write(45);  
}
```

ここでは、servo1 に 30 度、servo2 に 45 度を指定している。指定可能な角度としては普通のサーボであれば 0~180 度である。

7.3.5 練習問題

1. スイッチを押す度に角度が 10 度ずつ増加し、180 度まで達した後はスイッチを押す度に 10 度ずつ減少するプログラムを書こう

7.4 PWM(モータードライバ編)

マイコンボード上にあるコネクタがついた基板はモータードライバというもので、大電流を流さなければならぬモーターを駆動するものである。

7.4.1 モーターの制御における PWM について

なぜモーターの制御で PWM を使うのか説明しておく。よく使う DC モーターは電圧に応じて回転速度・トルクが増減する。しかし、レギュレーターなどで電圧を下げるとき、電圧差の分だけが熱となり失われてしまう。その対策として、ON,OFF の時間比率 (Duty 比) を制御することで、時間平均すれば電圧を変化させていることと同等のものとしてみなせる PWM がよく用いられる。(PWM はモーターのインダクタンスのおかげでうまく動作しているらしいが詳しいことは割愛する) 具体的には図のように Duty 比を変化させることで DC モーターの速度を制御している。

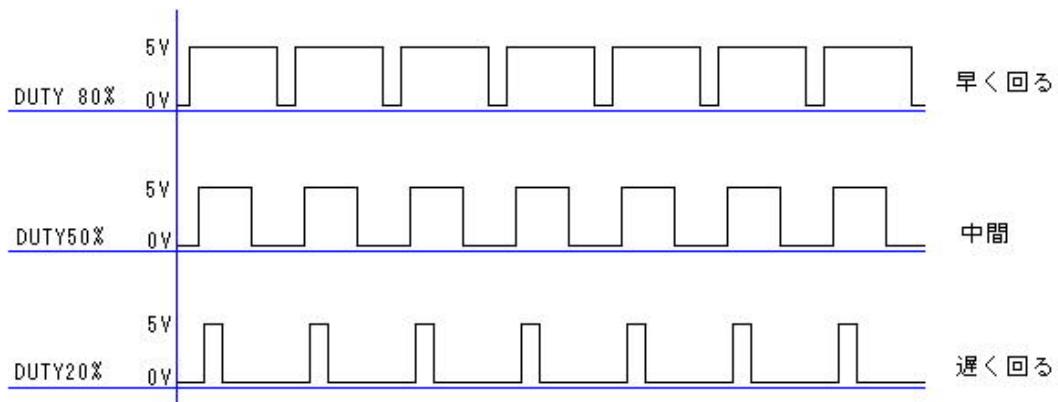


図 30 PWM の Duty 比 (<http://www.hokutodenshi.co.jp/PUPPYSupportPage/ensyu/pwm/pwm1.html> より)

7.4.2 モータードライバ L9110 の制御方法

モータードライバを用いて PWM 制御する方法を説明する。今回用いるモータードライバは L9110 というものであり、Google などで検索すればデータシート (部品の仕様書) が見つかる。下に要点のみ抜き出したデータシートの一部を示す。今回はモジュール化された基板を用いるため、IC パッケージに関しては直接気にする必要はない。しかし、入力が 2 つあり、出力が 2 つあることは把握しておく必要がある。赤い四角で囲まれた表が最も重要である。ここから IA と IB にかける電圧の組み合わせによって出力端子の様子が変化することが分かる。ちなみに、H は電源電圧、L は 0V である。

それでは組み合わせを詳しく見ていく。

- IA と IB の両者を同じ状態にすると出力は必ず L になっている。
- IA を H、IB を L にすると OA が H、OB が L となる。このときのモーターの回転を正転とする。
- IA を L、IB を H にすると OA が L、OB が H となる。上の場合を考えるとモーターにかかる電位が逆になり、モーターが逆転する。

このようになっている。つまり、正転させたい場合は IB を常に L にしておき、IA に PWM を入力すれば良

Pin definitions:

No.	Symbol	Function
1	OA	A road output pin
2	VCC	Supply Voltage
3	VCC	Supply Voltage
4	OB	B output pin
5	GND	Ground
6	IA	A road input pin
7	IB	B input pin
8	GND	Ground

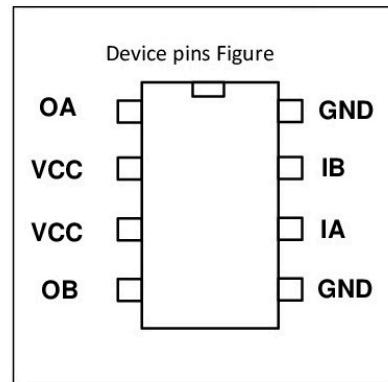


図 31 IC パッケージに関する部分

Symbol	Parameters	Range			Units
		Min	Typical	Max	
VCC	Supply Voltage	2.5	6	12	V
Idd	Quiescent Current	—	0	2	uA
Iin	Operating current	200	350	500	uA
IC	Continuous	750	800	850	mA
IMax	Current peak	—	1500	2000	mA

IA	IB	OA	OB
H	L	H	L
L	H	L	H
L	L	L	L
H	H	L	L

Application Circuit :

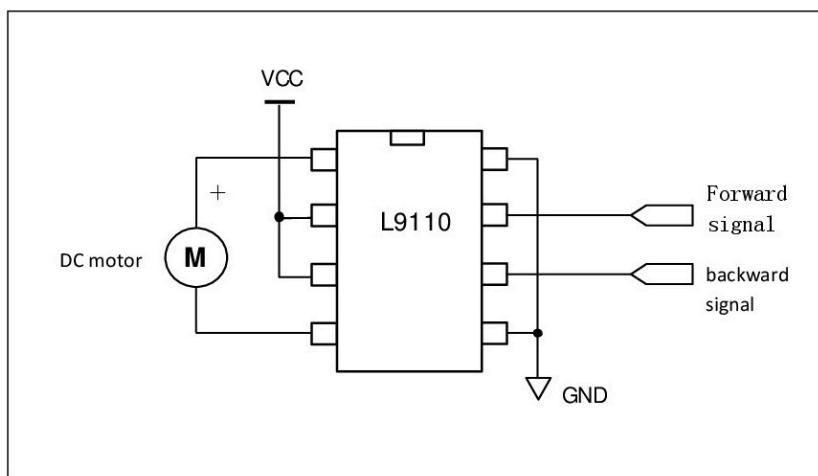


図 32 PWM の入力と出力に関する部分

い。逆転させたいときは、逆に IA を常に L にしておき、IB に PWM を入力すれば良い。

7.4.3 プログラム

それでは、例のごとく以下のプログラムを書き写そう。

```
#define MOTOR_A1 6
#define MOTOR_A2 9
#define MOTOR_B1 10
#define MOTOR_B2 11
```

```

void setup() {
}

void loop() {
    analogWrite(MOTOR_A1, 100);
    analogWrite(MOTOR_B1, 100);
    analogWrite(MOTOR_A2, 0);
    analogWrite(MOTOR_B2, 0);
    delay(1000);
    analogWrite(MOTOR_A1, 255);
    analogWrite(MOTOR_B1, 255);
    delay(2000);
    analogWrite(MOTOR_A1, 0);
    analogWrite(MOTOR_B1, 0);
    analogWrite(MOTOR_A2, 100);
    analogWrite(MOTOR_B2, 100);
    delay(1000);
    analogWrite(MOTOR_A2, 255);
    analogWrite(MOTOR_B2, 255);
    delay(2000);
}

```

うまく書き込めたならば、モーターが動き出し速度が速くなった後に逆転で動き出し以降同じような動きをする。

7.4.4 analogWrite()について

analogWrite() 関数は PWM 出力の Duty 比を指定する関数である。Duty 比は 0(0%)～255(100%) で指定することが出来る。なお、analogWrite() 関数は PWM 出力が利用可能な端子でのみ利用可能である。ピン番号を pinNumber、Duty 比を d とすると次のようになる。

```

void loop() {
    analogWrite(pinNumber, d);
}

```

具体的には以下のとおりである。

```

void loop() {
    analogWrite(6, 255);
}

```

D6 ピンに対して Duty 比 100% の PWM を出力することになる。

7.4.5 練習問題

- スイッチを押す度に回転速度が少しづつ早くなり、最高速度に達した後は少しづつ減少し、正転から逆転に切り替わり同じように正転時と同じ動作をするプログラムを書こう。

7.5 シリアル通信

シリアル通信はマイコンとの通信手段の1つである。おもにマイコン同士やパソコンとの通信で利用される。

7.5.1 Arduino IDE のシリアルモニタ

Arduino IDEにはマイコンとのシリアル通信をするためのシリアルモニタが標準搭載されている。ポート設定とボード設定を完了した状態で、「ツール」→「シリアルモニタ」と選択すると、シリアルモニタが起動する。



図33 シリアルモニタ起動

7.5.2 プログラム

それでは、例のごとく以下のプログラムを書き写そう。

```
void setup() {
  Serial.begin(9600);
  Serial.println("HelloWorld!");
}

void loop() {
  if (Serial.available() > 0){
    Serial.write(Serial.read());
  }
}
```

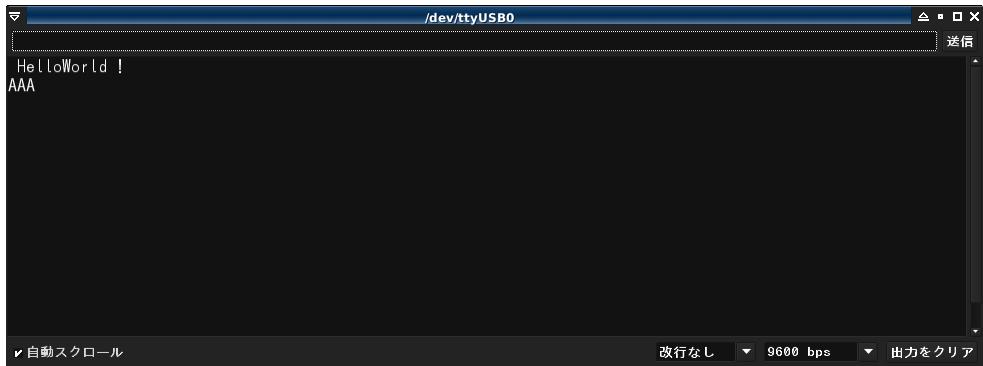


図 34 正常に動作している図

7.5.3 Serial.begin() について

begin() 関数はシリアル通信を行う場合必ず実行する必要がある。引数は通信速度であり、単位は bps(bit per second) である。

ただし、通信可能な速度は限られているため、特別な場合を除いては 9600bps を指定すること。

```
void setup() {
    Serial.begin(9600);
}
```

7.5.4 Serial.println() について

println() 関数は文字列をシリアル通信で送信する関数である。引数は整数型か String 型である必要がある。print() 関数との違いは、print() 関数は改行しないが、println() 関数は改行する点である。

```
void loop() {
    Serial.println("HOGE");
}
```

とすると文字列「HOGE」が送信される。

7.5.5 Serial.write() について

write() 関数は整数や文字列をバイナリで送信する関数である。

```
void loop() {
    Serial.write("HOGE");
}
```

とすると文字列「HOGE」がバイナリで送信される。

7.5.6 Serial.available() と Serial.read() について

available() 関数と read() 関数は図の通りの関係である。

シリアル通信で受信したデータは 1byte ずつバッファー（データの一時保管場所）に蓄積される。available() 関数はバッファーにデータが存在しているか確認し、存在する場合はその合計 byte 数を戻り値として返す。

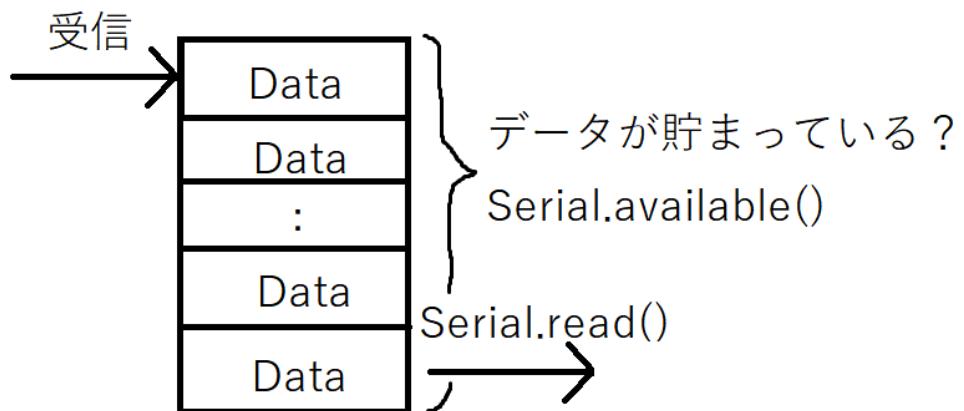


図 35 Serial.available() と Serial.read()

存在しない場合は 0 を返す。read() 関数はバッファーの一番古いデータから 1byte だけ取り出す関数である。取り出したデータはバッファーから消える。具体的には if 文や while 文と組み合わせて使うことが多い。

```
void loop() {
    String str;
    while(Serial.available() > 0){
        str += Serial.read();
    }
}
```

上のようにすると、受信したデータを String 型変数 str にどんどん代入していくことになる。

7.5.7 練習問題

以下の練習問題は「Arduino 日本語リファレンス」(<http://www.musashinodenpa.com/arduino/ref/>) の「String クラス」や Google 検索を利用すること。

(ヒント:String クラスには文字列 (String 型) を数値 (Int 型) に変換するメソッドが存在する。なお、Arduino 日本語リファレンスには記載されていないため、検索すること。)

1. シリアルモニタで送信した値にサーボモーターの角度を変化させるプログラムを書こう
2. シリアルモニタで送信した速さでモーターが回転するプログラムを書こう

7.6 Advanced な課題

これまでのプログラミング演習に物足りない人は以下の項目を実践してみるのも良いだろう。

- 通信形式が I2C なセンサの値を読み取りシリアルで出力する。
- 通信形式が SPI なセンサの値を読み取りシリアルで出力する。
- 一度のシリアル通信で複数の値を送受信する。
- Arduino のレジスタの設定を変更し、ADC の速度を高速化する。
- 圧電スピーカーを利用して音楽を演奏する。
- センサの値を Arduino 内で記録し、一定時間ごとにシリアル通信経由でデータを送り、パソコン側は自動で CSV ファイルに保存する。

7.7 できれば学習してほしいこと

以下に自主的な学習を強く推奨する事柄を示す。

- Git および GitHub によるソースコード管理の習得

8 最後に

マイコン講習を受け、回路講習で制作した機械研マイコンボードの基本的な機能は使いこなせるようになったはずだ。部内ロボコンだけでなく、ほかの製作物でもマイコンは必ず使うものであり、使いこなすことができれば製作可能な範囲はかなり広がる。機械研マイコンボードを性能不足に感じるようになったならば、STM32 などの高性能なマイコンを利用すれば良いだろう。出来る限り多くのプログラムを書き、試行錯誤してもらいたい。