LV8548MCSLDGEVK

モジュールソリューションキット ＜ステッパモータ編＞

ON Semiconductor®

www.onsemi.jp

APPLICATION NOTE

キット概要

オンセミコンダクター社が提供するモータドライバ製品を使用してモータ制御アプリケーションシステムを開発する場合、まず最初にモータドライバ製品の仕様を理解した上でハードウエアの設計を行い、次にドライバ製品に入力する動作制御信号(回転方向、回転速度、回転角度など)を生成する必要があります。一般的にこの様な動作制御信号はマイコンなどを使用して生成するため、上述ハードウエア設計に加えてマイコン用ソフトウェアの開発も必要となります。

本キットは、オンセミコンダクター社のモータドライバ製品(LV8548MC)をArduino MICRO マイコンから制御するためのモータドライバ制御用API 関数ライブラリを提供します。

また本キットは、本API関数ライブラリを組込んだArduino MICROをパソコンからUSB通信を介してターゲットモータを制御するための専用GUIも提供しておりますので、Arduino MICRO用モータ制御ソフトを開発することなくモータの制御シーケンスや動作パラメータをチューニングするなどのデバッグ作業を容易に行うことが可能となっています。

更には、本GUIはデバッグした制御シーケンスや動作パラメータをArduino MICROマイコンで実現するための制御ソフト（ソースコード）をArduino IDEでコンパイル可能な形式(スケッチ)で出力する、自動コード生成機能も有しております。

従って、本キットを使用することにより、モータドライバの仕様や、制御用API関数を使用したソフト開発など、特別な知識を有していなくてもモータ制御アプリケーションシステムの雛形(プロトタイプ)を容易に開発することを可能としているため、開発期間の短縮と大幅なコスト低減を実現します。

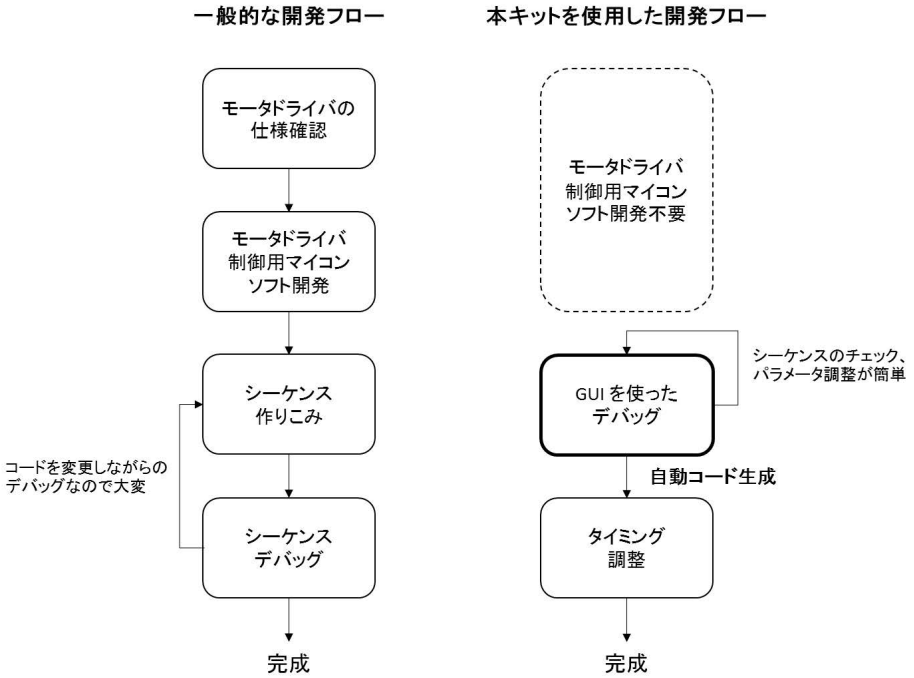


図 1. 一般的な開発フローと本キットを使用した開発フロー

Publication Order Number:
AND9854/D
Publication Order Number:
AND9854JP/D

GUIを使ったデバッグモードとスタンドアローン開発モード

本キットは、GUIを使ったデバッグモードと自動コード生成機能を用いたスタンドアローン開発モード、オリジナルスケッチを用いたスタンドアローン開発モードの計3種類のデバッグおよび開発モードの使用を想定しています。

GUIを使ったデバッグモード

GUIを使ったデバッグモードは、ユーザがPCにインストールされた専用GUIを操作することで、LV8548 に対してモータの制御シーケンスや動作パラメータを変更でき、実際にモータを動かしながらパラメータのチューニングが行えます。

また、ユーザがGUIを操作して設定したモータの制御シーケンスや動作パラメータを反映し、Arduino MICROにコンパイル・書き込みできるスケッチ(.inoファイル)として出力できる自動コード生成機能が備わっています。

このモードを使用するためには、Arduino IDEを使用し、キットに同梱されるGUI専用スケッチ(LV8548_STEP_Program.ino)とArduino MICRO専用モータ駆動API関数ライブラリ(LV8548_STEP_Lib.cpp/h)、および別途インターネットから取得するTimerOneライブラリをコンパイルすることで生成されるファームウェアをArduino MICROに書き込む必要があります。

GUIを使ったデバッグモードの概要を図2に示します。

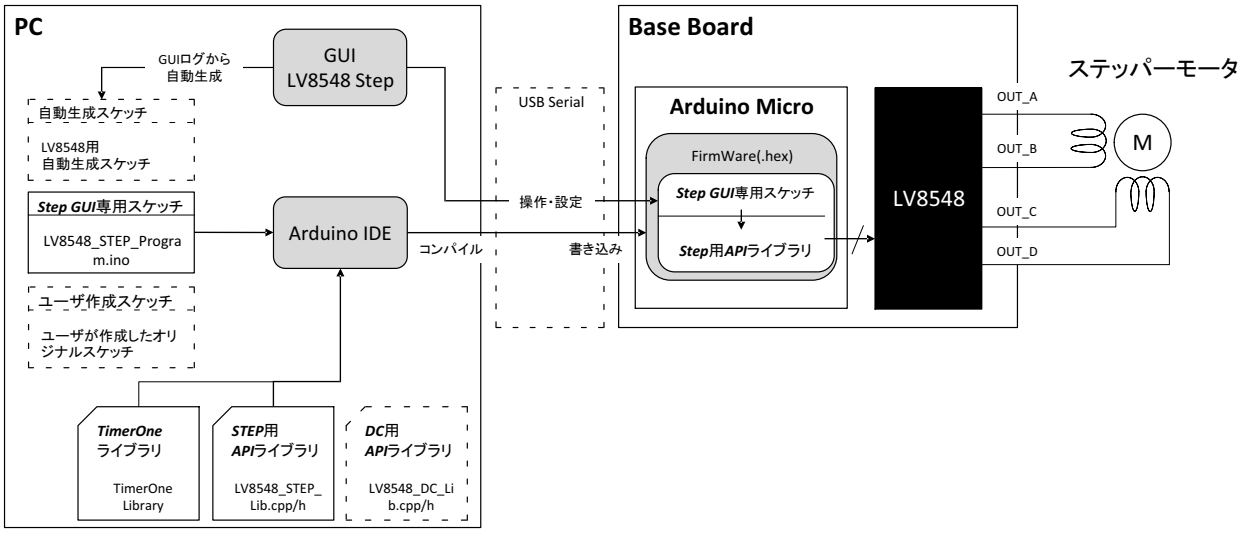


図 2. GUIを使ったデバッグモードの概要

スタンドアローン開発モード

スタンドアローン開発モードは、GUIを使ったデバッグモードで作成した自動生成スケッチを元に、モータ駆動タイミングの調整やユーザオリジナルのソースコードの追加などを行い、

GUI専用スケッチの代わりにArduino MICROに書き込むことで、本キットを用いたスタンドアローンでのステッパモータ駆動を容易に行える開発モードです。

自動生成機能を用いたスタンドアローン開発モードの概要を図3に示します。

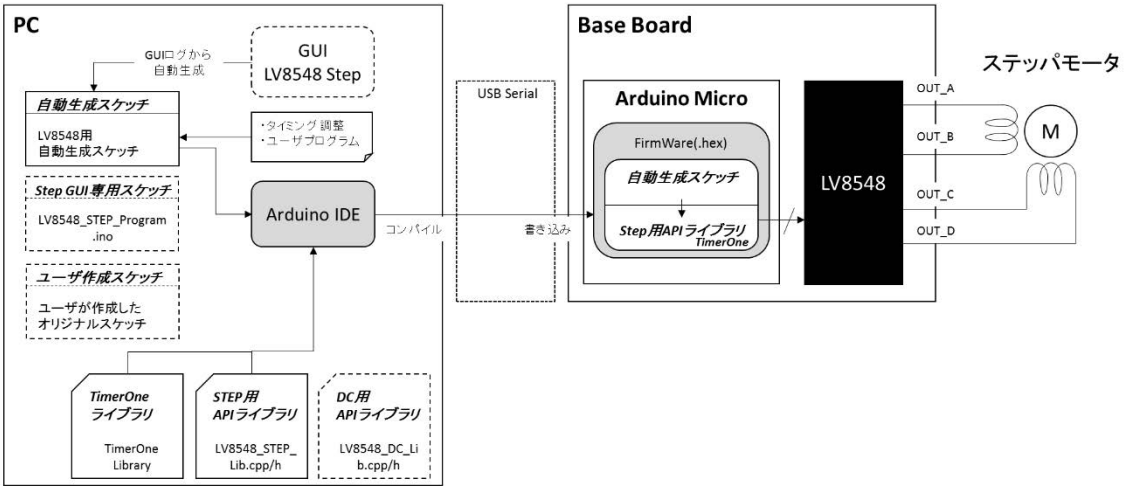


図 3. 自動生成機能を用いたスタンドアローン開発モード

自動コード生成機能を用いずに、API関数ライブラリを利用したオリジナルスケッチを利用することもできます。高度なプログラミング知識があれば、より複雑で高度なモータ制御アプリケーションを開発することも可能です。

ユーザオリジナルスケッチを用いたスタンドアローン開発モードの概要を図4に示します。

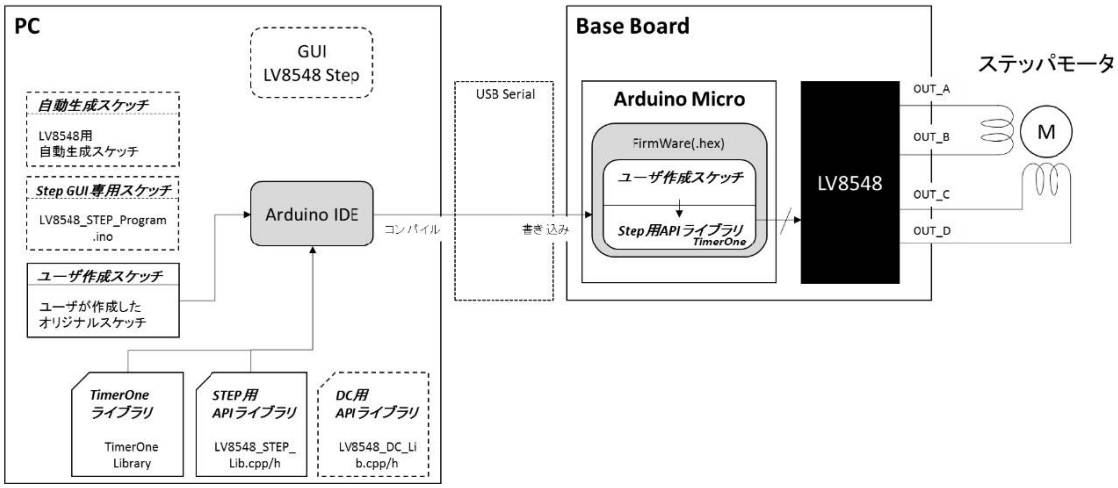


図 4. オリジナルスケッチを用いたスタンドアローン開発モード

プログラミングガイド

Arduinoのスケッチについて

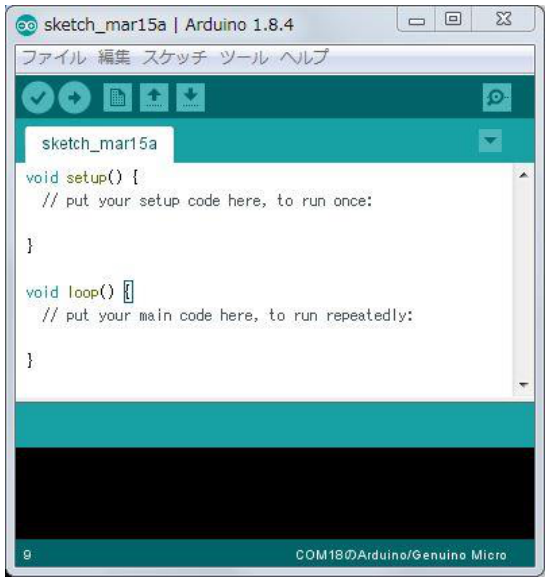
スケッチの概要

スケッチは、Arduino言語で記述されたArduinoのためのプログラムの名称であり、Arduino Boardにアップロードされて動くコードのまとまりを指します。Arduino言語はC/C++をベースに構築されており、C言語の全構造とC++の一部機能をサポートしています。

スケッチの詳細は以下を参照してください。
https://www.arduino.cc/en/Tutorial/Sketch

setup()とloop()関数

Arduino IDEから新規スケッチの作成を行うと、下図のように自動的にsetup()とloop()が挿入されます。



setup()は、Arduino Boardの電源オンの後、またはリセット後に一度だけ呼び出される関数で、変数やピンモードの初期化ライブラリの使用準備などを記述します。

setup()の詳細は以下を参照してください。
https://www.arduino.cc/reference/en/language/structure/sketch/setup/

loop()は名前のとおり、setup() の実行後に繰り返し実行される関数で、この中に実際に繰り返し動作させたいプログラムを記述します。

loop()の詳細は以下を参照してください。
https://www.arduino.cc/reference/en/language/structure/sketch/loop/

ステッパモータ制御APIライブラリの概要

ステッパモータ制御APIライブラリ(LV8548_STEP_APILibrary)は、Arduino Microからオンセミコンダクター社製 モータドライバLV8548を使用して、ステッパモータを制御するためのライブラリを提供します。ユーザーは、本APIライブラリをArduino IDEからインクルードし、目的に沿ったAPI関数をスケッチに記述することで容易にLV8548を使用したステッパモータ制御が可能となります。

表 1. ステッパモータ制御 APIライブラリファイル一覧

#	ファイル名	内容
1	keywords.txt	キーワードファイル(スケッチ内の表示色変更対象ワードの設定)
2	LV8548_STEP_Lib.cpp	ソースファイル
3	LV8548_STEP_Lib.h	ヘッダファイル

ステッパモータ制御APIライブラリの使い方

ライブラリのインクルードについては、第2章クイックスタートガイドを参照してください。

ステッパモータ制御APIライブラリをArduinoで利用する場合、以下のようにスケッチの冒頭でステッパモータ制御APIライブラリのヘッダファイルをインクルードし、使用するクラスのインスタンス化を行います。

GUIツールを使用する場合は、別途シリアル通信APIを呼び出す必要があります。詳細は、API関数仕様を参照してください。

ステッパモータ制御APIライブラリをインクルードしたスケッチを以下に示します。

ステッパモータ制御APIライブラリのインクルード

```
#include <LV8548_STEP_Lib.h>
//LV8548STEP用API関数を使用するためのヘッダファイルの取り込み
#include <TimerOne.h>
//TimerOneライブラリを使用するためのヘッダファイルの取り込み
Lib_LV8548STEP  Lib;        //LV8548STEPクラスのインスタンス化(※1)
void setup(){               //スタート時に呼び出される関数(setup()とloop()関数参照)
}
void loop(){                //繰り返し実行される関数(setup()とloop()関数参照)
}
```

1. 今回の例では「Lib」という名前でインスタンス化しているため、接頭に「Lib.」をつけることでLib_LV8548StepクラスのAPI関数を呼び出し可能になります。例：Lib.initLib();

スケッチのコンパイル、書き込み

スケッチ(Arduinoプログラム)のコンパイル、書き込み手順については、第2章 クイックスタートガイド 「Arduinoプログラムのコンパイル・Arduinoへの書き込み」を参照してください。

プログラム（スケッチ）のコーディング

自動コード生成機能詳細

自動コード生成機能で出力したスケッチの一例を用いて、スケッチを出力する際に自動で記述する関数等の役割を説明します。

GUIデバッグ操作によって記述されるステッパモータ制御APIライブラリの各API関数詳細は、API関数仕様 を参照してください。

自動生成コード例

```
#include <LV8548_STEP_Lib.h> //ステッパモータ制御APIライブラリの使い方参照
#include <TimerOne.h>        //ステッパモータ制御APIライブラリの使い方参照
Lib_LV8548Step Lib;          //ステッパモータ制御APIライブラリの使い方参照
void setup() {               //setup()とloop()関数参照
  Serial.begin(19200);       //Baud rateを19200に設定してポートを開く(注意11)
  Lib.initLib();             //Arduinoパラメータとレジスタの初期化
  Timer1.initialize(65);     //Timer1の初期化とタイマ割り込み間隔を65[μsec]に設定
  Timer1.attachInterrupt(interrupt); //タイマ割り込みで呼び出す関数の指定
  delay(5000);               //Arduino起動後のインターバル時間[ms](注意12)
  Lib.setStepAngle(1.8);
  delay(0);                  //API関数実行後のインターバル時間[ms]
  Lib.motorRotationStep(100, 100.0, 0, 0);
  delay(0);                  //モータ駆動時間[ms] (注意13)
  Lib.motorRotationStop();
  delay(0);                  //API関数実行後のインターバル時間[ms]
  Lib.motorRotationFree();
  delay(0);                  //API関数実行後のインターバル時間[ms]
}
void interrupt() {           //タイマ割り込みで呼び出される関数
  Lib.timerFire(65);
}
void loop() {                //setup()とloop()関数参照
}
```

2. シリアル通信を使用する場合に必要な関数です。
   使用しない場合は削除しても動作に影響はありません。
3. デフォルト5000として設定されます。
4. モータスタート後のdelay()はモータの駆動時間になります。
   デフォルト0として設定されるため、必要に応じて変更してください。

自動生成されたスケッチの利用

自動生成されたスケッチは、プログラミング初心者でも使いやすいようシンプルな構成になっています。これをカスタマイズすることで、より実践的なプログラムになります。

今回は、アプリケーション適用例 で生成したスケッチのカスタマイズ例として、Arduinoセットアップ部分とモータ制御部を関数化し、それぞれsetup()とloop()で呼び出すスケッチを紹介します。

```
#include <LV8548_STEP_Lib.h>
#include <TimerOne.h>
Lib_LV8548Step Lib;
void setup() {
  motorSetup(); //関数化したArduinoの初期設定をsetup()で呼び出します。
}
void interrupt() {
  Lib.timerFire(65);
}
void loop() {
  motorControl(); //関数化したモータ制御部をloop()で呼び出します。
}
//Arduinoの初期設定を関数化します。//
void motorSetup() {
  Serial.begin(19200);
  Lib.initLib();
  Timer1.initialize(65);
  Timer1.attachInterrupt(interrupt);
  delay(5000);
}
//モータ制御部を関数化します。//
void motorControl() {
  Lib.setStepAngle(1.8);
  delay(0);
  Lib.motorRotationStep(100, 100.0, 0, 0);
  delay(5000);          //delayを5000に変更し、モータ1駆動時間を5000[ms]に設定します。
  Lib.motorRotationStop();
  delay(0);
  Lib.motorRotationFree();
  delay(0);
}
```

アプリケーション適用例

概要

本キットを使用して、キットに付属するステッパモータ(MDP–35A)を時計にするアプリケーションを開発します。

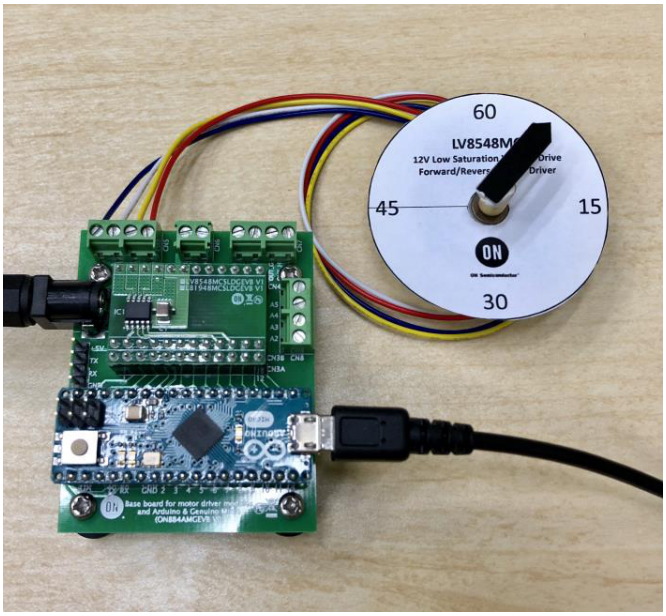


図 5. 時計アプリ

必要なもの

- 時計の型紙（次のステップ にて後述）

仕様

（基礎編）

- 1分で針が1周する
- フルステップで、モータが脱調(*)しない程度の速度で針を動かす
- 時間は delay 関数で作り出す（1ステップの時間は 60/48 = 1.25秒）（応用編）
- PC からキーボードで時計の（r: 実行、s: 停止）操作ができる

ブロック図

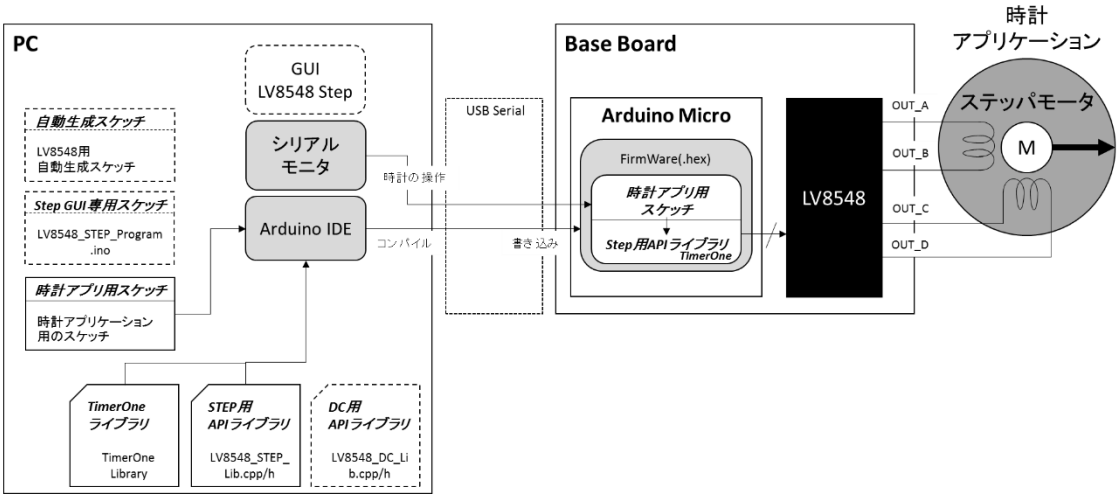


図 6. ブロック図

接続について

- キットに付属するステッパモータをそのまま使います。
  クイックスタートガイドと同じ接続です。時計の動きが分かりやすいように付属の時計の型紙（*次のステップ*）を使用してください。

GUI を使った基本作成フロー（基礎編）

手順1:
　クイックスタートガイドの手順に沿って進めます。
　GUI を操作するスケッチ LV8548_STEP_Program.ino をコンパイルし、マイコンボードに書き込み、GUI を立ち上げ、シリアルポートを接続し、LV8548 Step のタブを立ち上げてください。
　GUIのパラメータを以下のように設定します。

- Excitation: Full step
- Direction: CW
- Step Angle: 7.5 (Set ボタンを押す)
- Transfer Unit: Steps
- Transfer Step: 1.0 steps

　設定後、Motor Speed を適当な値 (200 steps/s) に変えて、Start を押します。
　最初に試す簡単な時計アルゴリズムは Motor Speed が速い方が時間の誤差が少なくなるので、色々な値を試して安定動作する上限を探します。Motor Speed が決まったら Generate Program ボタンを押して適当な名前で保存します。

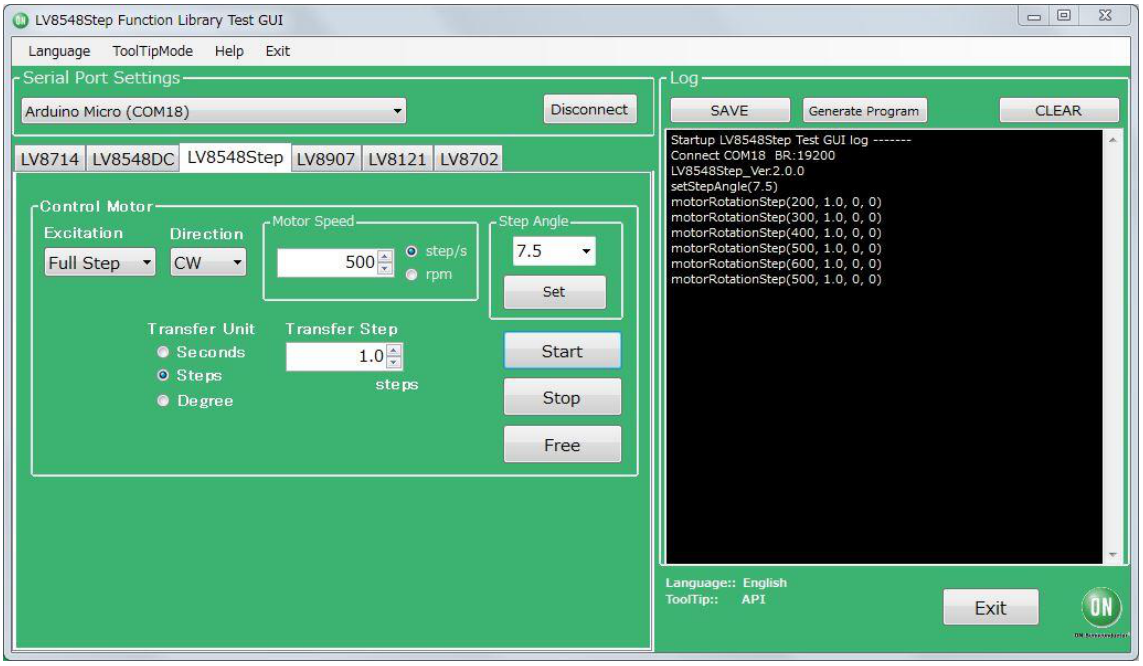


図 7. GUI を使ったデバッグ

手順2:
　手順1で自動生成したスケッチを開き、カスタマイズを行います。setup() から不要な部分を削除して、1ステップ動作させるコードとdelay() の部分を loop() に移動させます。
　付属のステッパモータ(MDP−35A)はフルステップの場合、一周48ステップで、1ステップの時間は60/48 = 1.25秒なので、delay に 1250 ms を設定します（例では速度を500step/s にしているので、1250 − 2 = 1248 msがより正確です。他にもオーバーヘッドがありますが、ここでは精度については追求しません。）。完成したスケッチをスケッチ1 に示します。

スケッチ1

```
#include <LV8548_STEP_Lib.h>
#include <TimerOne.h>
Lib_LV8548Step Lib;
void setup()
{
  Serial.begin(19200);
  Lib.initLib();
  Timer1.initialize(10);
  Timer1.attachInterrupt(interrupt);
  delay(5000);
  Lib.setStepAngle(7.5);
  delay(0);//0msec
}
void interrupt()
{
  Timer1.initialize(Lib.timerFire());
}
void loop()
{
  Lib.motorRotationStep(500, 1, 0, 0);
  delay(1250);//1.25sec
}
```

出来上がったスケッチをマイコンボードに書き込んでください。約5秒後に時計が動作し始めます。

PCから時計の操作ができるようにする（応用編）

GUIを用いてモータ制御を行う場合、ステッパモータ制御APIライブラリの guiSerialRead() 関数を利用して PCとArduino Micro間の通信を成立させています。応用編ではこの関数を利用し、先程作成した時計をPC のキーボードから操作できるようにします。

手順:

Lib_LV8548Step クラスを継承した my_Lib_LV8548Step クラスを作ります。

新たに作成したmy_Lib_LV8548Step クラスでは、ステッパモータ制御APIライブラリのguiSerialParse() 関数を独自の関数に置き換えることができますので、（詳しくは C++ のクラスの継承や仮想関数について調べてください）この中に時計を操作するためのコマンドを実装します。

キーボードからコマンドを受け取った時に、新たに作った関数 guiSerialParse() で新しく作った変数 clock_run の値をコマンドが‘r’ なら true、‘s’ なら falseに変更します。

ステッパは loop() 関数の中で delay() 時間ごとに繰り返し実行されるので、clock_run が true の時だけステッパを動かすようにします。

loop() の中に guiSerialRead() を入れて、delay() 時間ごとに、シリアルポートを監視するようにします。完成したスケッチをスケッチ2 に示します。

スケッチ2

```
#include <LV8548_STEP_Lib.h>
#include <TimerOne.h>
// 時計の状態を示す変数
// volatile を忘れないで下さい。組み込み特有です。
volatile bool clock_run = true;
// クラスの継承をして、新しいシリアルコマンド処理関数を書き換えます
class my_Lib_LV8548Step : public Lib_LV8548Step
{
public:
  virtual ~my_Lib_LV8548Step() {} // Don't remove!
  int guiSerialParse(char *type) override;
};
// 新しいシリアルコマンドを処理する関数です
int my_Lib_LV8548Step::guiSerialParse(char *serialRecvStr)
{
  switch (serialRecvStr[0])
  {
    case 'r':
      clock_run = true;
      return SUCCESS;
      break;
    case 's':
      clock_run = false;
      return SUCCESS;
      break;
    default:
      return FAILURE;
  }
}
// カスタムクラスのインスタンシエーション
my_Lib_LV8548Step Lib;
void setup()
{
  Serial.begin(19200);
  Lib.initLib();
  Timer1.initialize(10);
  Timer1.attachInterrupt(interrupt);
  delay(5000);
  Lib.setStepAngle(7.5);
  delay(0);//0msec
}
void interrupt()
{
  Timer1.initialize(Lib.timerFire());
}
void loop()
{
  Lib.guiSerialRead();      // シリアルコマンドの処理をします
  if (clock_run == true) {  // true なら針を動かします
    Lib.motorRotationStep(500, 1, 0, 0);
  }
  delay(1250);//1.25sec
}
```

Arduino IDE の [ツール] → [シリアルモニタ] を開いて下さい。

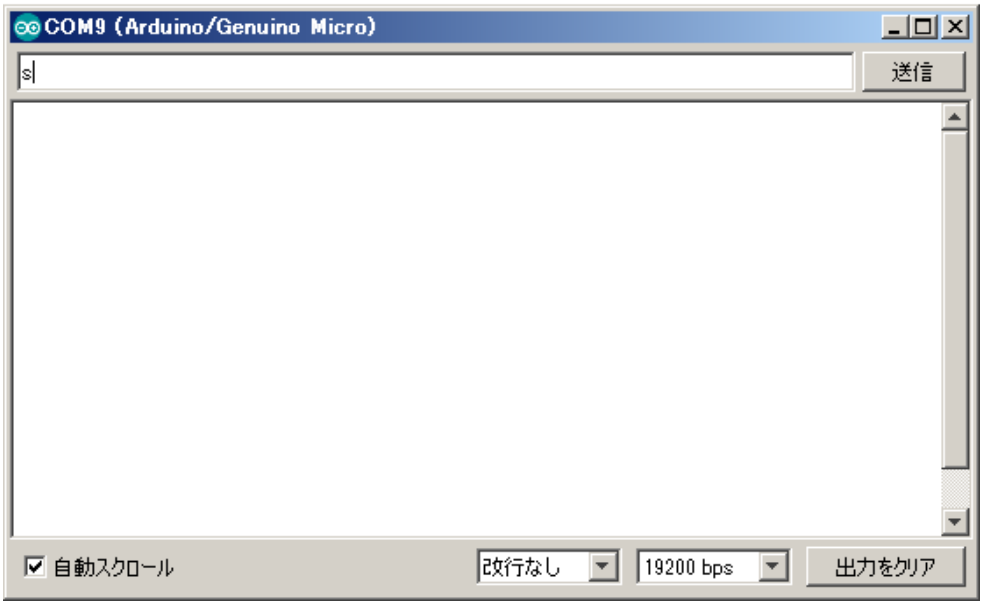


図 8. PC からの操作（シリアルモニタ）

上部の入力欄にs または r を入力後、送信ボタンを押してください。

正しく動作しない場合は、シリアルモニタの通信速度が正しいことや、GUI が接続されていないことなどを確認して下さい。

次のステップ

本アプリケーション適用例を参考に、以下のような、さらに高度なアプリケーション開発にチャレンジしてみてください。

時計の型紙

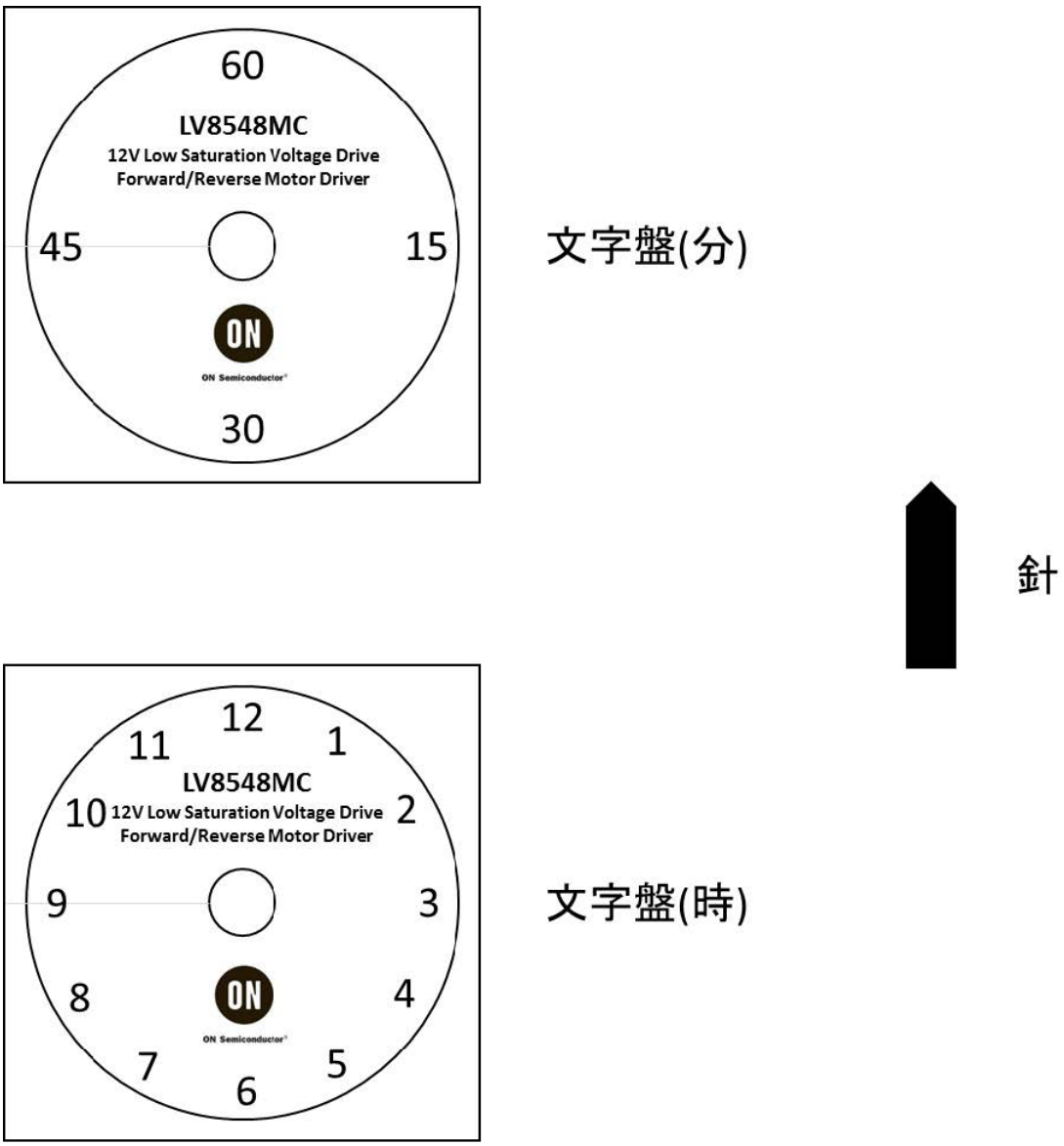

60
LV8548MC
12V Low Saturation Voltage Drive
Forward/Reverse Motor Driver
45
15
30
文字盤(分)
針
12
11
1
10
2
9
3
8
4
7
5
6
LV8548MC
12V Low Saturation Voltage Drive
Forward/Reverse Motor Driver
文字盤(時)

図 9. 型紙

API関数仕様

ステッパモータ制御API 概要

本章では、LV8548モータドライバ向けArduino Micro ステッパモータ制御APIの概要について記述します。

概要

本APIは、Arduino Microからオン・セミコンダクター社製 モータドライバLV8548を使用して、ステッパモータを制御するためのライブラリを提供します。ユーザーは、本APIライブラリをArduino IDEからインクルードし、目的に沿ったAPI関数をスケッチに記述することで容易にLV8548を使用したステッパモータ制御が可能となります。

ライブラリファイル構成

表 2. ライブラリファイル一覧

#	ファイル名	内容
1	keywords.txt	キーワードファイル(スケッチ内の表示色変更対象ワードの設定)
2	LV8548_STEP_Lib.cpp	ソースファイル
3	LV8548_STEP_Lib.h	ヘッダファイル

Arduino Micro/LV8548 Base Board ピンアサイン

Arduino Micro/LV8548 Base Board ピンアサインを以下に示します。

Arduino Micro入出力ピンの白背景色は、ユーザーが利用可能なリソースを表します。

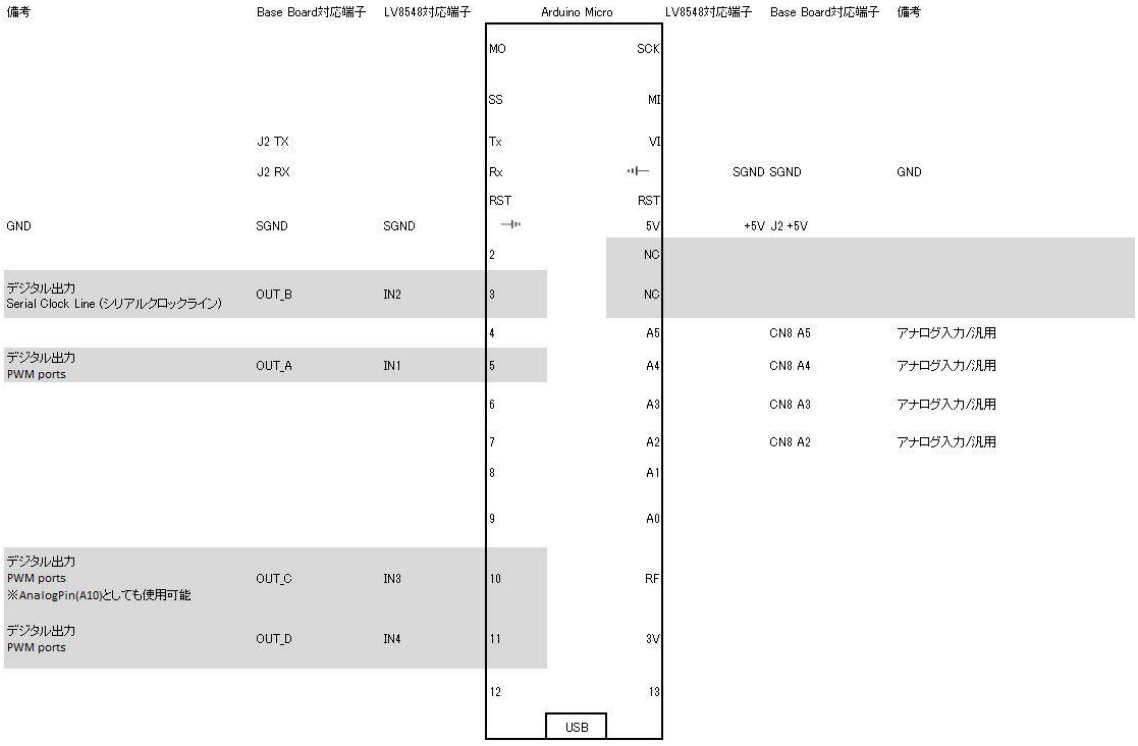


図 10. Arduino Micro/LV8548 Base Board ピンアサイン

ステッパモータ制御API 初期設定

本章では、ステッパモータ制御APIで行う初期設定について記述します。

初期設定

Arduino Microの各出力ピンの初期化は、initLib関数によって行われます。本APIライブラリを使用する際は、必ずinitLib関数をスケッチのsetup()内で呼び出し、各出力ピンの初期化を行ってください。

initLib関数の使用方法は、initLibを参照してください。

表 3. 初期化ピン一覧

#	Arduino Micro出力ピン	初期設定値	関連
1	D3	OUTPUT	GUIを使ったデバッグモードとスタンドアローン開発モード
2	D5	OUTPUT	GUIを使ったデバッグモードとスタンドアローン開発モード
3	IO10	OUTPUT	GUIを使ったデバッグモードとスタンドアローン開発モード
4	IO11	OUTPUT	GUIを使ったデバッグモードとスタンドアローン開発モード

API関数仕様

API関数概要

LV8548モータドライバ ステッパモータ制御API関数を以下に示します。

表 4.

#	関数名	概要	章番号
1	initLib	– 入出力ピンの設定	initLib
2	setStepAngle	– ステップ角の設定	setStepAngle
3	motorRotationDeg	– 設定された度数分回転する	motorRotationDeg
4	motorRotationTime	– 設定された時間分回転する	motorRotationTime
5	motorRotationStep	– 設定されたステップ分回転する	motorRotationStep
6	motorRotationStop	– モータを停止する(励磁状態を保持)	motorRotationStop
7	motorRotationFree	– モータを停止する(全出力OFF)	motorRotationFree
8	readAdc	– アナログ電圧値の測定	readAdc
9	readModule	– A0ピンアナログ電圧値の取得(機種判別用)	readModule
10	setDelay	– Timer0分周比に合わせた専用delay関数	setDelay
11	checkRotating	– モータの駆動状態を判定	checkRotating
12	guiSerialRead	– シリアル受信データのバッファ管理	guiSerialRead
13	guiSerialParse	– GUIからのメッセージの解析と実行	guiSerialParse

API関数詳細

initLib

表 5. initLib

API関数名	initLib()		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	int		常に0: “成功”を返す
処理概要	デジタル出力ピンの設定 – IN1をD5に設定 – IN2をD3に設定 – IN3をIO10に設定 – IN4をIO11に設定		
使用例 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { **Lib.initLib();** // 初期設定 }		

setStepAngle

表 6. setStepAngle

API関数名	setStepAngle(*float* deg)		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	float	deg	ステップ角[degree] 0.01−360.00
戻り値	型		内容
	int		0: “成功” 1: “失敗”
処理概要	– ステップ角をセットします。(初期値0.01) – 引数degが値域外の場合、失敗(1)を返します。		
使用例 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.setStepAngle(7.5);** // ステップ角 7.5度設定 }		

motorRotationDeg

表 7. **motorRotationDeg**

API関数名	motorRotationDeg(*float* freq, *float* deg, *byte* cwccw, *byte* exc)		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	float	freq	周波数[Hz] 1−4800
	float	deg	角度[degree] 1−65535 0: infinity(永続駆動)
	byte	cwccw	回転方向(0:正転、1:逆転)
	byte	exc	励磁方式(0:フルステップ、1:ハーフステップ)
戻り値	型		内容
	int		0: “成功” 1: “失敗”(設定値が範囲外)
処理概要	– 引数freq, cwccw, excで指定される周波数、回転方向、 励磁方式による回転制御を引数degで指定した角度分行います。 (各パラメータの初期値は 内部パラメーター一覧を参照) – 指定角度回転後にモータを停止します。(励磁状態を保持) – 引数freq, deg, cwccw, excが値域外の場合、失敗(1)を返します。		
使用例 **(sketch)**	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.motorRotationDeg(100,180,0,0);** // フルステップモードにて周波数100 Hzで正転で180度回転する		

motorRotationTime

表 8. **motorRotationTime**

API関数名	motorRotationTime(*float* freq, *uint16_t* time, *byte* cwccw, *byte* exc)		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	float	freq	周波数[Hz] 1−4800
	uint16	time	駆動時間[sec] 1−65535 0: infinity(永続駆動)
	byte	cwccw	回転方向(0:正転、1:逆転)
	byte	exc	励磁方式(0:フルステップ、1:ハーフステップ)
戻り値	型		内容
	int		0: “成功” 1: “失敗”(設定値が範囲外)
処理概要	– 引数freq, cwccw, excで指定される周波数、 回転方向、励磁方式による回転制御を引数timeで指定した時間行います。 (各パラメータの初期値は 内部パラメーター覧を参照) – 指定時間回転後にモータを停止します。(励磁状態を保持) – 引数freq, time, cwccw, excが値域外の場合、失敗(1)を返します。		
使用例 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.motorRotationTime(100,10,0,0);** // フルステップモードにて周波数100 Hzで正転で10秒回転する		

motorRotationStep

表 9. **motorRotationStep**

API関数名	motorRotationStep(*float* freq, *uint16_t* step, *byte* cwccw, *byte* exc)		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	float	freq	周波数[Hz] 1−4800
	uint16	step	ステップ数 1−65535 0: infinity(永続駆動)
	float	cwccw	回転方向(0:正転、1:逆転)
	float	exc	励磁方式(0:フルステップ、1:ハーフステップ)
戻り値	型		内容
	int		0: “成功” 1: “失敗”(設定値が範囲外)
処理概要	– 引数freq, cwccw, excで指定される周波数、 回転方向、励磁方式による回転制御を引数stepで指定したステップ数行います。 (各パラメータの初期値は 内部パラメーター一覧を参照) – 指定ステップ数回転後にモータを停止します。(励磁状態を保持) – 引数freq, step, cwccw, excが値域外の場合、失敗(1)を返します。		
使用例 **(sketch)**	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.motorRotationStep(100,100,0,0);** // フルステップモードにて周波数100 Hzで正転で100ステップ回転する		

motorRotationStop

表 10. **motorRotationStop**

API関数名	motorRotationStop()		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	void		なし
処理概要	モータを停止します。(励磁状態でトルクを保持)		
使用例 **(Sketch)**	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.motorRotationStop();** // モータを停止する }		

motorRotationFree

表 11. **motorRotationFree**

API関数名	motorRotationFree()		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	void		なし
処理概要	モータを停止します。(全出力OFFでトルクを失う)		
使用例 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.motorRotationFree();** // モータ全出力OFF }		

readAdc

表 12. **readAdc**

API関数名	readAdc(*byte pin*)		
Class	Lib_LV8548Step		
属性	public		
引数	型	変数	内容
	byte	pin	ピン番号 1: A1 2: A2 3: A3 4: A4 5: A5
戻り値	型		内容
	int		成功…指定したピンのアナログ電圧値 失敗…地域外の入力をした場合
処理概要	1　引数値域チェック 2　引数のピンに対してanalogRead()を実行し、取得した値を返す		
使用例 (sketch)	#include <LV8548_Lib.h> // LV8548Step API Library読込み Lib_LV8548Step Lib;　// Lib_LV8548Step インスタンス void setup() { Lib.initLib(); // 各パラメータ初期化 } void loop(){ int value; value = Lib.readAdc (1); // 1ピンを読み込む }		

readModule

表 13. **readModule**

API関数名	readModule()		
Class	Lib_LV8548Step		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	int		指定したピンのアナログ電圧値
処理概要	1　A0ピンにanalogRead()を実行し、取得した値を返す		
使用例 (sketch)	#include <LV8548_Lib.h> // LV8548Step API Library読込み Lib_LV8548Step Lib;　// Lib_LV8548Step インスタンス void setup() { 　Lib.initLib(); // 各パラメータ初期化 } void loop(){ 　int value; 　value = Lib.readModule ();　// A0ピンのアナログ電圧値を読み込む }		

setDelay

表 14. **setDelay**

API関数名	setDelay(*uint32_t* msec)		
Class	Lib_LV8548Step		
属性	public		
引数	型	変数	内容
	uint32_t	msec	待機時間
戻り値	型		内容
	int		0: 成功
処理概要	1　引数(msec)の時間、待機する。		
使用例 (sketch)	#include <LV8548_Lib.h> // LV8548Step API Library読込み Lib_LV8548Step Lib; // Lib_LV8548Step インスタンス void setup() { 　Lib.initLib(); // 各パラメータ初期化 　Lib.setDelay(1000); // 1秒待機 } void loop() { }		

checkRotating

表 15. **checkRotating**

API関数名	checkRotating ()		
Class	Lib_LV8548Step		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	bool		0: 駆動中 1: 待機中
処理概要	1　モータが駆動しているかを判別する		
使用例 (sketch)	#include <LV8548_Lib.h> // LV8548Step API Library読込み Lib_LV8548Step Lib;　// Lib_LV8548Step インスタンス void setup() { 　Lib.initLib(); // 各パラメータ初期化 　　Lib.motorRotationStep(200, 400, 0, 0); 　While(Lib.cehckRotating()){Lib.setDelay(1);} // 200ステップ回り終えるまで待つ } void loop() { }		

checkStepCount

表 16. **checkStepCount**

API関数名	checkStepCount()		
Class	Lib_LV8548Step		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	int32		現在のステップカウント数
処理概要	1　現在のステップカウント数を取得する		
使用例 (sketch)	#include <LV8548_Lib.h> // LV8548Step API Library読込み Lib_LV8548Step Lib;　// Lib_LV8548Step インスタンス void setup() { 　Lib.initLib(); // 各パラメータ初期化 　Lib.motorRotationStep(200, 400, 0, 0); 　Lib.checkstepcount(); //　ステップカウント数の取得 } void loop() { }		

guiSerialRead

表 17. **guiSerialRead**

API関数名	guiSerialRead()		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	void	なし	なし
戻り値	型		内容
	void		なし
処理概要	– シリアル受信データのバッファリング管理を行います。 – guiSerialParse を呼び出し、受信データの解析と実行を行います。 – 3秒間シリアル受信がない場合はモータを停止します。(全出力OFFでトルクを失う)		
使用例 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Serial.begin(19200); // Baud rate 19200でポートを開く Lib.initLib(); // 初期設定 } void loop() { **Lib. guiSerialRead** (); //シリアルメッセージの受信 }		

guiSerialParse

表 18. guiSerialParse

API関数名	guiSerialParse(*char* *type)		
Class	Lib_LV8548_STEP		
属性	public		
引数	型	変数	内容
	char*	*type	[値域 0x03: getId識別子 0x04: timeoutPol識別子 0x69: setStepAngle識別子 [0x0000−0x8CA0](Step角×100 [0〜36000]) 0x6A: motorRotationDeg識別子 [0x0001−0xBB80](周波数(Hz)×10 [1−48000]) [0x0000−0xFFFF](回転角×10 [1−65535], 0=infinity) [0x00−0x01](回転方向) [0x00−0x01](フルステップ、ハーフステップ) 0x6B: motorRotationTime識別子 [0x0001−0xBB80](周波数(Hz)×10 [1−48000]) [0x0000−0xFFFF](回転時間 [1−65535], 0 = infinity) [0x00−0x01](回転方向) [0x00−0x01](フルステップ、ハーフステップ) 0x6C: motorRotationStep識別子 [0x0001−0xBB80](周波数(Hz)×10 [1−48000]) [0x0000−0xFFFF](ステップ数 [1−65535], 0 = infinity) [0x00−0x01](回転方向) [0x00−0x01](フルステップ、ハーフステップ) 0x6E: motorRotationStop識別子 0x6F: motorRotationFree識別子
戻り値	型		内容
	int		0: “成功” 1: “失敗”
処理概要	– GUI からのメッセージの解析、実行を行います。 – 引数が値域外の場合は失敗(1)を返します。		
使用例1 (Sketch)	Lib_LV8548_STEP Lib; // Lib_LV8548_STEP class定義 void setup() { Lib.initLib(); // 初期設定 } void loop() { **Lib.guiSerialRead();** //guiSerialRead関数処理でguiSerialParse処理を呼ぶ。 }		

表 18. **guiSerialParse**

使用例2 (Sketch)	// ユーザーは、guiSerialParse() をオーバーライドすることによって、シリアルインターフェースをカスタマイズして利用できます。// //Lib_LV8548_STEPクラスを継承して派生クラスの作成// class Lib_LV8548_STEP_custom : public Lib_LV8548_STEP{ public: virtual ~Lib_LV8548_STEP_custom() {} int guiSerialParse(char *type) override; }; Lib_LV8548_STEP_custom Ex; //継承クラスのインスタンス化 //guiSerialParse関数をオーバーライド// int Lib_LV8548_STEP_custom::guiSerialParse(char *serialRecvStr) { // Implement for serial code execution. switch (serialRecvStr[0]) { case ' a' : { Serial.println("Command"); return SUCCESS; //成功(0)を返します。 } default: { Serial.println("unknown command"); } } return FAILURE; //失敗(1)を返します。 } void setup() { Serial.begin(19200); // Baud rate 19200でポートを開く Ex.initLib(); // 初期設定 } void loop() { Ex.guiSerialRead(); //guiSerialRead関数処理でguiSerialParse処理を呼ぶ。 }

シリアルインターフェース仕様

本章では、PCとArduino MicroをUSB接続した際のシリアルインターフェースについて記述します。

Arduino側プログラム（スケッチ）に、シリアル通信設定、並びにguiSerialRead関数を実装し、PCからシリアル通信を介して各APIに対応したメッセージ送信することで、LV8548を使用したステッパモータ制御が可能です。

（このシリアルインターフェース仕様は予告なく変更される可能性があります。）

guiSerialRead関数実装方法は、guiSerialRead を参照してください。

表 19. メッセージ一覧

#	コマンド	対応API	コマンド概要	Length	章番号
1	0x03	None	APIライブラリ識別用IDを取得します。	1 byte	getId
2	0x04	timeoutPole	一定間隔で送信されるシリアル接続監視用のコマンドです。	1 byte	timeoutPole
3	0x69	setStepAngle	SetStepAngleをCallし、 ステップ角を設定します。	3 byte	setStepAngle
4	0x05	initLib	対応APIをCallし、APIの各設定を初期化します。	1 byte	initLib
5	0x6A	motorRotationDeg	MotorRotationDegをCallし、 角度指定でモータの回転制御を行います。	7 byte	motorRotationDeg
6	0x6B	motorRotationTime	MotorRotationTimeをCallし、 時間指定でモータの回転制御を行います。	7 byte	motorRotationTime
7	0x6C	motorRotationStep	MotorRotationStepをCallし、 ステップ指定でモータの回転制御を行います。	7 byte	motorRotationStep
8	0x6E	motorRotationStop	MotorRotationStopをCallし、 モータを停止します。(励磁状態を保持)	1 byte	motorRotationStop
9	0x6F	motorRotationFree	MotorRotationFreeをCallし、 モータを停止します。(全出力OFF)	1 byte	motorRotationFree
10	0x64	readAdc	対応APIをCall し、指定したアナログ入力ピンの電圧を読み出します。	3 byte	readAdc
11	0x48	readModule	対応APIをCallし、Moduleを識別します。	3 byte	readModule
12	0x70	checkStepCount	対応APIをCallし、ステップカウントを読み出します。	5 byte	checkStepCount

メッセージ構成詳細

GUIからArduino Microに送信されるメッセージ構成の詳細を記述します。

getId

- コマンド概要
  APIライブラリ識別用IDを取得するためのコマンドです。
  guiSerialParse関数は、このコマンドを受信すると対応モータドライバ名、APIバージョンが識別できるIDを返します。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x03

timeoutPole

- コマンド概要
  Arduino Microのシリアル接続状態を監視するためのコマンドです。
  GUIはこのコマンドを1秒ごとにArduino Microに送信します。APIはこのコマンドを含めて3秒間のシリアル受信データがない場合、フェイルセーフのためtimeoutPole関数がCallされ、自動的にモータを停止します。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x04

initLib

- コマンド概要
  APIの各設定を初期化するためのコマンドです。
  GUIはGUIからシリアル切断を行う際にこのコマンドをArduino Microに送信し、initLib関数をCallします。
  initLib関数の詳細はinitLibを参照してください。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x05

setStepAngle

- コマンド概要
  setStepAngle関数を呼び出し、ステップ角を設定するためのコマンドです。
  guiSerialParse関数は、受信した各パラメータを引数の形式に変換してsetStepAngle関数をCallします。
  setStepAngle関数の詳細はsetStepAngleを参照してください。

Command from GUI to Motor Driver Kit

Byte	0	1 (下位)	2 (上位)
Field	CMD	STEP ANGLE	
Value	0x41	0x0000−0x8CA0	

Field STEP ANGLE: ステップ角[degree × 100] (0−36000)
GUIは入力されたステップ角(最小単位0.01)を100倍し、整数値としてArduino Microに送信します。guiSerialParse関数は、受信したSTEP ANGLEをsetStepAngle関数の引数として扱えるステップ角(0.00−360.00[degree])に再変換します。

motorRotationDeg

- コマンド概要
  motorRotationDeg関数を呼び出し、角度指定でのモータ駆動を実行するためのコマンドです。
  guiSerialParse関数は、受信した各パラメータを引数の形式に変換してmotorRotationDeg関数をCallします。
  motorRotationDeg関数の詳細はmotorRotationDegを参照してください。

Command from GUI to Motor Driver Kit

Byte	0	1	2	3	4
Field	CMD	FREQ			
Value	0x6A	0x41000000−0x420C0000			

Byte	5	6	7	8	9	10
Field	ANGLE				ROT	EXC
Value	0x00000000−0xFFFFFFFF				0x00−0x01	0x00−0x01

Field FREQ: 励磁信号周波数[Hz × 10](1−48000)
GUIは入力された励磁信号周波数(最小単位0.1)をArduino Microに送信します。guiSerialParse関数は、受信したFREQをmotorRotationDeg関数の引数として扱える励磁信号周波数 (1−4800[Hz])に変換します。
Field ANGLE: 駆動角[degree]
Field ROT: 回転方向
Field EXC: 励磁方式

motorRotationTime

- コマンド概要
  motorRotationTime関数を呼び出し、時間指定でモータ駆動を実行するためのコマンドです。
  guiSerialParse関数は、受信した各パラメータを引数の形式に変換してmotorRotationTime関数をCallします。
  motorRotationTime関数の詳細はmotorRotationTimeを参照してください。

Command from GUI to Motor Driver Kit

Byte	0	1	2	3	4
Field	CMD	FREQ			
Value	0x6A	0x41000000−0x420C0000			

Byte	5	6	7	8	9	10
Field	TIME				ROT	EXC
Value	0x00000000−0xFFFFFFFF				0x00−0x01	0x00−0x01

Field FREQ: 励磁信号周波数[Hz × 10](1−48000)
GUIは入力された励磁信号周波数(最小単位0.1)をArduino Microに送信します。 guiSerialParse関数は、受信したFREQをmotorRotationTime関数の引数として扱える励磁信号周波数 (1−4800[Hz])に変換します。

Field TIME: 駆動時間[sec]
Field ROT: 回転方向
Field EXC: 励磁方式

motorRotationStep

- コマンド概要
  motorRotationStep関数を呼び出し、ステップ数指定でモータ駆動を実行するためのコマンドです。
  guiSerialParse関数は、受信した各パラメータを引数の形式に変換してmotorRotationStep関数をCallします。
  motorRotationStep関数の詳細はmotorRotationStepを参照してください。

Command from GUI to Motor Driver Kit

Byte	0	1	2	3	4
Field	CMD	FREQ			
Value	0x6A	0x41000000−0x420C0000			

Byte	5（下位）	6	7	8（上位）	9	10
Field	STEP				ROT	EXC
Value	0x00000000−0xFFFFFFFF				0x00−0x01	0x00−0x01

Field FREQ: 励磁信号周波数[Hz × 10](1−48000)
GUIは入力された励磁信号周波数(最小単位0.1)をArduino Microに送信します。guiSerialParse関数は、受信したFREQをmotorRotationStep関数の引数として扱える励磁信号周波数(1−4800[Hz])に変換します。
Field STEP: 駆動ステップ数[step]
Field ROT: 回転方向
Field EXC: 励磁方式

motorRotationStop

- コマンド概要
  motorRotationStop関数を呼び出し、モータを停止(励磁状態でトルクを保持)するためのコマンドです。
  motorRotationStop関数の詳細はmotorRotationStopを参照してください。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x6E

motorRotationFree

- コマンド概要
  motorRotationFree関数を呼び出し、モータを停止(全出力OFF)するためのコマンドです。
  motorRotationFree関数の詳細はmotorRotationFreeを参照してください。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x6F

readAdc

- コマンド概要
  readAdc関数を呼び出し、指定したアナログピンの電圧測定を行うためのコマンドです。
  readAdc関数をCallし、読み出したアナログ電圧値をCMD 0x62と共に送信側に返します。
  readAdc関数の詳細はreadAdcを参照してください。

Command from GUI to Motor Driver Kit

Byte	0	1
Field	CMD	CH
Value	0x64	0x01−0x05

Field CH: アナログピン(A1~A5)

Command from Motor Driver Kit to GUI

Byte	0	1（下位）	2（上位）
Field	CMD	RECVADC	
Value	0x62	0x0000−0x03FF	

Field RECVADC: アナログ電圧値(0~1023)

readModule

- コマンド概要
  readModule関数を呼び出し、Arduinoと接続されているモジュールを識別するためのコマンドです。
  guiSerialParse関数は、readModule関数をCallし、読み出したアナログ電圧値をCMD 0x49と共に送信側に返します。
  readModule関数の詳細は readModuleを参照してください。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x48

Field RECVMODULE: 機種別アナログ電圧値 (0~1023)

checkStepCount

- コマンド概要
  checkStepCount関数を呼び出し、Arduinoと接続されているモジュールを識別するためのコマンドです。
  guiSerialParse関数は、checkStepCount関数をCallし、目標までの到達ステップ数をCMD 0x71と共に送信側に返します。
  checkStepCount関数の詳細は checkStepCountを参照してください。

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x70

Command from Motor Driver Kit to GUI

Byte	0	1 (最下位)	2	3	4 (最上位)
Field	CMD	STEP			
Value	0x71	0x00−0xFFFFFFFF			

Field STEP: 到達ステップ数(0 ~ 4294967295)

内部パラメータ/テーブル/関数 概要

内部パラメーター覧

内部パラメーター一覧を下記に示します。

表 20. 内部パラメーター一覧

#	パラメータ名	初期値	内容	更新契機
1	StepDeg	ANGLE_MIN	モータステップ角	setStepAngleで変更
2	StepFrequency	FREQ_MIN	ステップ周波数	freqChangeで変更
3	Excitation	FULLSTEP	励磁方式	motorRotationDeg motorRotationTime motorRotationStepで変更
4	CwCcw	ROTATION_CW	回転方向	motorRotationDeg motorRotationTime motorRotationStepで変更
5	Target_Deg	0	目標操作角	motorRotationDeg motorRotationTime motorRotationStepで変更
6	TimerCounter	0	タイマカウンタ	timerFireで変更
7	PhaseCounter	0	位相カウンタ	timerFireで変更
8	Now_Deg	0	現在角	timerFireで変更
9	isRotation	FALSE	モータ回転フラグ	motorRotationDeg motorRotationTime motorRotationStep motorRotationStop motorRotationFreeで変更
10	inPin[4]	なし	PWM出力ピン	initLibで変更
11	StartCount	0	駆動開始時のステップカウント	motorRotationDeg motorRotationTime motorRotationStepで変更
12	NowCount	0	ステップカウント	timerFireで変更
13	ROTATION_CW	0	正転	固定値
14	ROTATION_CCW	1	逆転	固定値
15	FULLSTEP	0	Full Step	固定値
16	HALFSTEP	1	Half Step	固定値
17	FREQ_MIN	1	PWM周波数最小値	固定値
18	FREQ_MAX	4800	PWM周波数最大値	固定値
19	DEG_MIN	0	回転角最小値	固定値
20	DEG_MAX	65535	回転角最大値	固定値
21	TIME_MIN	0	回転時間最小値	固定値
22	TIME_MAX	65535	回転時間最大値	固定値
23	STEP_MIN	0	ステップ数最小値	固定値
24	STEP_MAX	65535	ステップ数最大値	固定値
25	ANGLE_MIN	0.01	ステップ角最小値	固定値
26	ANGLE_MAX	360	ステップ角最大値	固定値
27	SRMES_GET_ID	0x03	getId API対応 シリアルメッセージ 識別子	固定値
28	SRMES_SET_INITIAL	0x05	initLib API対応シリアルメッセージ識別子	固定値
29	SRMES_POLLING_ID	0x04	timeoutPol API対応 シリアルメッセージ 識別子	固定値
30	PIN_NUMBER_MIN	1	ピン番号最小値(A0)	固定値
31	PIN_NUMBER_MAX	5	ピン番号最大値(A5)	固定値
32	FAILURE_ADC	0xFFFF	readAdc失敗時の戻り値	固定値

表 20. 内部パラメーター一覧 (continued)

#	パラメータ名	初期値	内容	更新契機
33	SRMES_STEP_ANGLE	0x69	setStepAngle API対応 シリアルメッセージ 識別子	固定値
34	SRMES_ROTATION_ANGLE	0x6A	motorRotationDeg API対応 シリアルメッセージ 識別子	固定値
35	SRMES_ROTATION_TIME	0x6B	motorRotationTime API対応 シリアルメッセージ 識別子	固定値
36	SRMES_ROTATION_STEP	0x6C	motorRotationStep API対応 シリアルメッセージ 識別子	固定値
37	SRMES_ROTATION_STOP	0x6E	motorRotationStop API対応 シリアルメッセージ 識別子	固定値
38	SRMES_ROTATION_FREE	0x6F	motorRotationFree API対応 シリアルメッセージ 識別子	固定値
39	SRMES_READ_ADC	0x64	readAdc API対応シリアルメッセージ識別子	固定値
40	SRMES_READ_MODULE	0x48	readModule API対応シリアルメッセージ識別子	固定値
41	SRMES_CHK_STEP	0x70	checkStepCount API対応シリアルメッセージ識別子	固定値
42	SRMES_RES_READ_MODULE	0x49	readModule API 応答用対応シリアルメッセージ識別子	固定値
43	SRMES_RES_READ_ADC	0x62	readAdc API応答用対応シリアルメッセージ識別子	固定値
44	SRMES_RES_CHK_STEP	0x71	checkStepCount API応答用対応シリアルメッセージ識別子	固定値

内部構造体一覧

内部構造体一覧を下記に示します。

表 21. 内部構造体一覧

#	構造体名	内容
1	SrMesDivSetStepAngle	setStepAngleシリアル通信パラメータ格納構造体
2	SrMesDivRotationDeg	motorRotationDegシリアル通信パラメータ格納構造体
3	SrMesDivRotationTime	motorRotationTimeシリアル通信パラメータ格納構造体
4	SrMesDivRotationStep	motorRotationTimeシリアル通信パラメータ格納構造体
5	SrMesDivReadAdc	readAdcシリアル通信パラメータ格納用構造体
6	SrMesDivRecvReadAdc	readAdc応答用シリアル通信パラメータ格納用構造体
7	SrMesDivRecvReadModule	readModuleシリアル通信パラメータ格納用構造体
8	SrMesDivRecvStepCount	checkStepCountシリアル通信パラメータ格納用構造体
9	SrMesDivRotationStop	motorRotationStopシリアル通信パラメータ格納構造体
10	SrMesDivRotationFree	motorRotationFreeシリアル通信パラメータ格納構造体

内部関数一覧

内部関数一覧を下記に示します。

表 22. 内部構造体一覧

#	関数名	内容	Call元関数
1	freqChange	励磁周波数の変更	motorRotationDeg motorRotationTime motorRotationStep
2	timerFire	タイマー割り込み関数	なし
3	timeoutPol	ポーリングタイムアウト時にモータを停止	guiSerialRead
4	setPinMode	ピン設定関数	initLib
5	checkStepCount	目標ステップ到達判定関数	なし

ハードウエア仕様

動作条件

電源電圧について

LV8548のデータシートには推奨動作条件として電源電圧が4.0～16Vと規定されています。

この範囲の電圧がICに印加されているとき、ICは安定して動作することを示しています。

ただし、その電圧でモータを駆動することになるため、この範囲の電圧であってもモータにとっては低すぎて駆動できなかったり、高すぎてモータを故障させる場合があります。

また、巻き線抵抗が低いモータを使用した場合、過電流が生じ、ICが破壊する可能性があります。使用するモータのスペックを確認のうえ、電圧を決定してください。

電源に電池を使用する場合、電圧が低下して4V以下になることが想定されます。このとき、IC内部の回路動作が不安定となる場合があるため、ICが自動的に動作を停止します。（減電圧保護機能）

また、モータを駆動すると電源電圧が持ち上がる現象が起こることがあります。LV8548は最大定格として20Vを規定していますので、瞬時であってもこの電圧を超えることはIC破壊の原因となります。

（次頁「回生電流とは」参照）

最大電流について

モータドライバICが内蔵する出力トランジスタや、ICチップとICピンを接続する金属線が流せる電流は限られています。これ以上の電流を流すとIC破壊の原因となります。これにより、モータドライバICは機種ごとに出力最大電流（以下Iomax）が規定されています。

LV8548のIomaxは1Aと規定されており、これを超えないように注意が必要です。

また、Iomaxは常に流せる電流を意味するものではありません。Iomax以下の電流値であっても、温度上昇により動作が停止することがあります。（次項参照）

動作温度について

半導体製品は最大定格ジャンクション温度（150℃前後）を超えるとIC破壊の原因となります。

逆の言い方をすると、最大定格ジャンクション温度までは正常動作が期待されます。

IC、特にモータドライバICは電力を消費し、それ自体が発熱します。使用環境の周囲温度とICの自己発熱により、IC内部の温度が最大定格ジャンクション温度を超えるとICが自動的に動作を停止します。（サーマルシャットダウン機能）

また、ICの最大定格、推奨動作条件、電気的特性は周囲温度（Ta）が25℃で規定されているということをご理解ください。（詳しくはLV8548 ICのデータシートをご参照ください。）

関連用語集

逆起電圧とは

モータは電気エネルギーを機械エネルギーに変換すると同時に、その逆となる発電もしています。このときに発生する電圧を逆起電圧、逆起電力、誘起電圧とも呼びます。

モータの起動時は逆起電圧が発生しておらず、モータへの印加電圧とモータの巻き線抵抗による電流が流れます（突入電流）。

突入電流はモータ起動時の短時間ではありますが、モータ駆動電流のピークとなります。

逆起電圧はモータの回転スピードがあがるにつれて大きくなり、印加電圧を打ち消す働きをするため、モータ駆動電流は減少します。

Hブリッジとは

図11のようにBaseboardのモータ接続端子（OUT_A, OUT_B, OUT_C, OUT_D）につながるICの端子内部にはトランジスタがつながっています。このトランジスタとモータコイルによって『H』の文字を構成することから、この回路をHブリッジと呼びます。

LV8548は2組のHブリッジを内蔵しています。

これにより、DCモータを2個ないしステッパモータ（バイポーラタイプ）を1個駆動することができます。

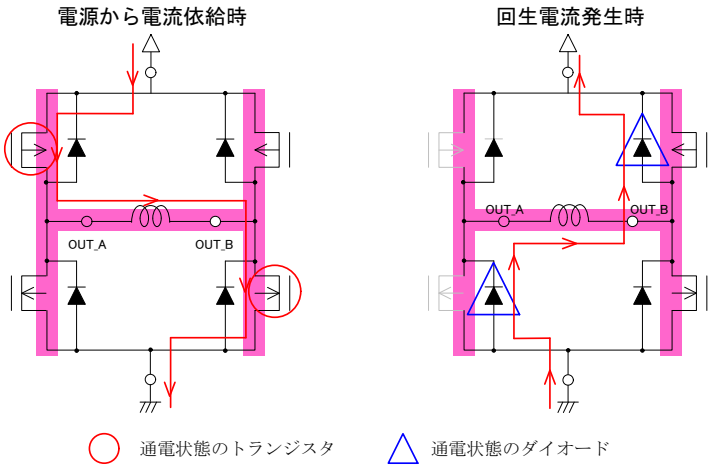


図 11. LV8548 の H ブリッジ状態遷移による電流経路

回生電流とは

モータ（コイル）の特徴として、モータが回転することで発電し、電流を流すエネルギーをもつことが挙げられます。

図11の場合、丸印の2つのトランジスタがONしてOUT_AからOUT_Bの方向に電流が流れます。このとき、モータが回転すると同時に、コイルがエネルギーを蓄えます。

モータを停止するときなど、トンランジスタがOFFすると、コイルは蓄えたエネルギーによって、電源から電流が供給された時と同じ方向に電流を流そうとするため、IC内蔵の三角印のダイオードを経由して、矢印の経路で電流が流れます。このときの電流を回生電流といいます。回生電流は電源に戻っていきます。

安定化電源の場合、この電流を吸収して、印加電圧を保つことができますが、ACアダプタや電池の場合、電流を吸収することができず、大きな電圧上昇を引き起こします。

これにより、ICに印加される電圧が急上昇し、ICの破壊に至る場合があります。

LV8548の場合、電源電圧の最大定格が20 Vと規定されており、これを超えないように注意が必要です。Baseboardに設置されている100 μFの電解コンデンサはこのときの電圧上昇を抑える働きをします。（図12参照）

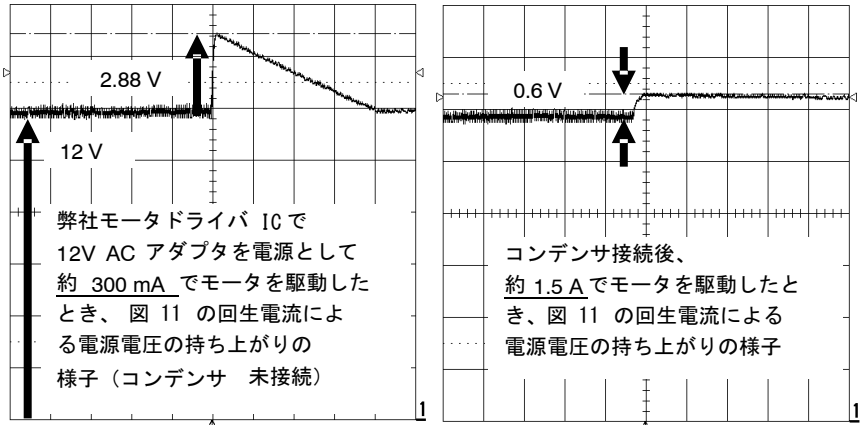


図 12. 回生電流による電源電圧の持ち上がりと電解コンデンサの効果

この電圧上昇を抑える方法として、図13に示すような電圧クランプ回路を導入する方法もあります。Tr：FQP20N06，Di：1N5240B（ともにオンセミコンダクター），R1：150 Ωとした場合、R2を100～330 Ωの範囲で調整することでVCC電圧を20 V以下に制限することができます。

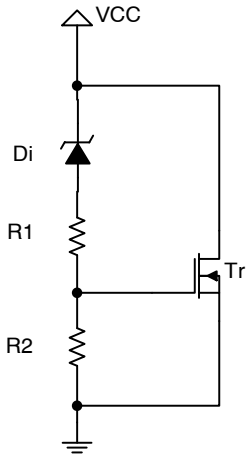


図 13. 電源電圧クランプ回路例

ユニポーラタイプとバイポーラタイプ

ステッパモータは図14, 15に示すように、内部構造とそれに伴う駆動方式の違いから、2種類に分けられます。

	通電方向	ワイヤ本数
ユニポーラ	片方向	6本ないし5本
バイポーラ	双方向	4本

弊社モノリシックLSIのステッパモータドライバICはLV8548も含めすべて、バイポーラタイプ（2相）のモータ用に設計されています。

ユニポーラモータを駆動することも可能ですが、専用ドライバICと比べるとモータのパフォーマンスに違いが生じる場合がありますので、確認、評価が必要です。（図16）

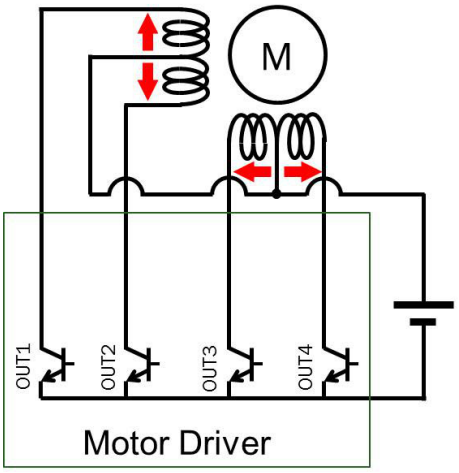


図 14. ユニポーラタイプの構造と駆動方式

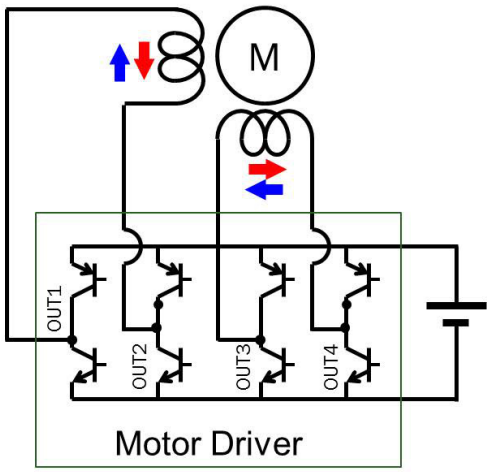
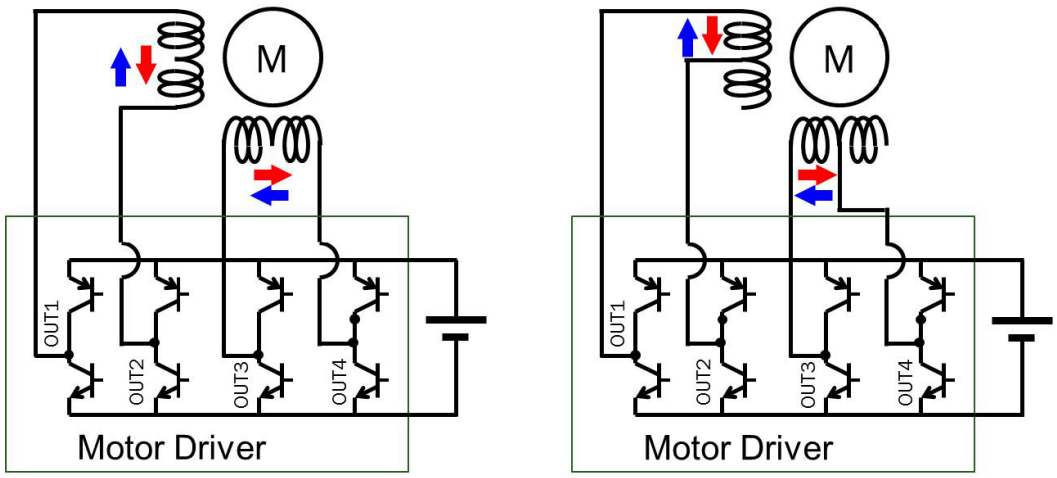


図 15. バイポーラタイプの構造と駆動方式



接続例 1　　接続例 2

図 16. ユニポーラタイプのモータをバイポーラ駆動する場合の接続方法

ステップ角とは

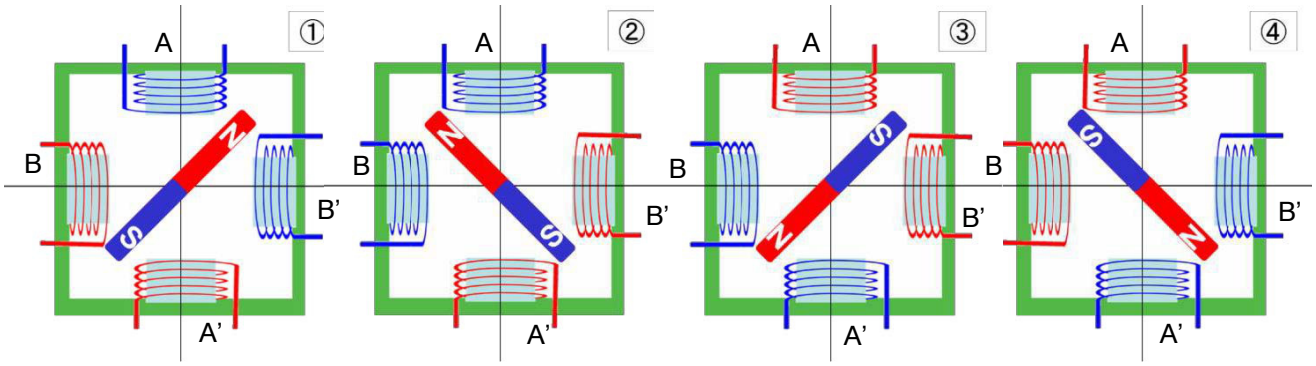
ステッパモータはブラシDCモータとは異なり、センサーがなくとも位置制御が可能です。これはモータ固有のステップ角が決まっているためです。

ステップ角とは『Full stepで駆動したときに1ステップでモータ（ロータ）が回る角度』

と理解できます。そのため、モータドライバICに入力した信号が何ステップ分かを管理することで、モータの回転角を把握できます。

ステッパモータの構造は直交した2組（2相）の向かい合うコイルとロータを表す棒磁石で簡易的に説明されます。（図17, 18）

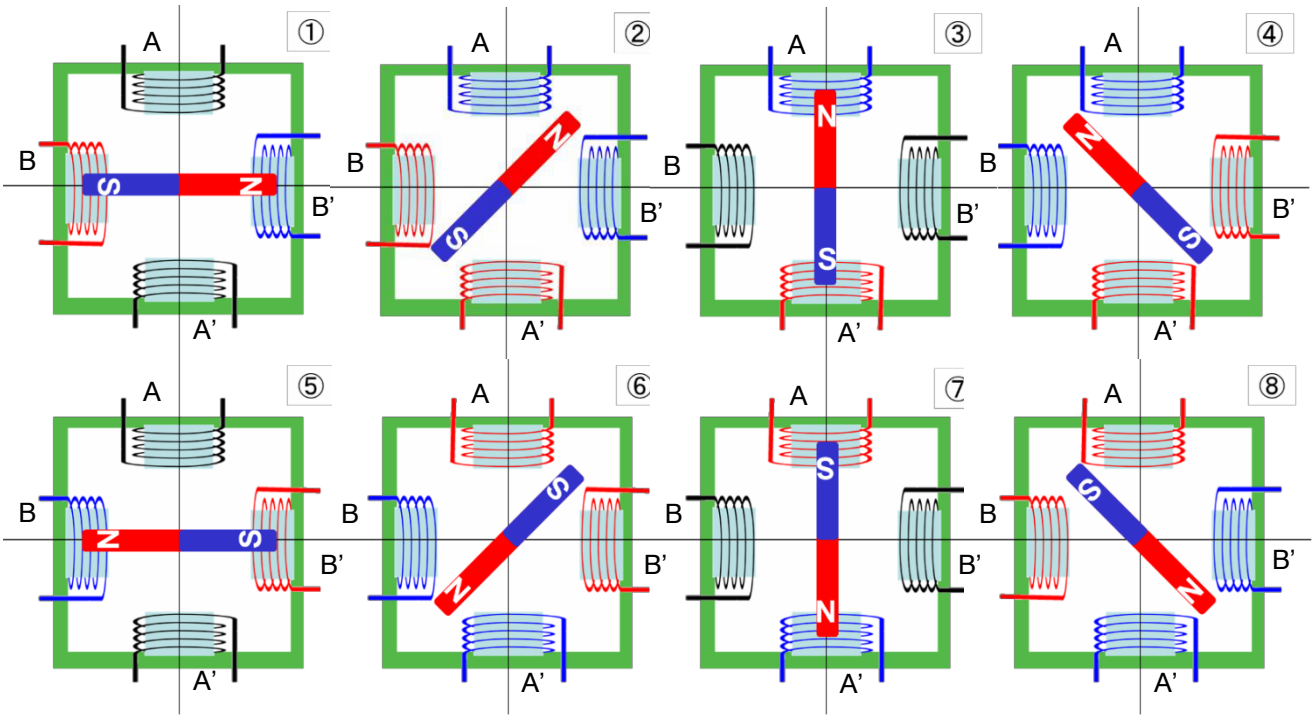
Full stepの場合、2組のコイルはどちらも常に通電（励磁）されています。このため、棒磁石は隣り合うコイルが引き合う位置で安定します。棒磁石が回転するように2組のコイルの通電方向を連続して切り替えます。そのためFull stepは2相励磁方式とも呼ばれます。（図17参照）



赤：N 極に励磁、青：S 極に励磁

図 17. Full Step 電気角遷移図

Half stepで駆動したときは1ステップの移動量がFull stepの半分となります。
Half stepの場合、1組のコイルのみが通電されるタイミング（1相励磁）と2相励磁を交互に切り替えるためです。このことからHalf stepは1−2相励磁とも呼ばれます。（図18参照）



赤：N 極に励磁、青：S 極に励磁、黒：非励磁状態

図 18. Half step 電気角遷移図

Full stepが4つのポジションを取るのに対し、Half stepは8つのポジションを取ります。
ただし、この4つないし8つのポジションで表されるのは電気角であり、ステップ角とは異なります。
前述のとおり、ステップ角は『Full stepで駆動したときに1ステップでモータ（ロータ）が回る角度』ですが、Full stepで駆動したときの1ステップは電気角90°に相当します。
ステップ角＝電気角90°と理解してください。
Full step設定時のステップ角、Motor Speed、Transfer Unitによる回転に要する時間の算出式を以下に示します。
delay関数の引数への利用が可能です。
代数による演算式は回転時間が10000[mS]すなわち10秒となる一例です。

条件1）

- Motor Speed単位：[step/s]
- Transfer Unit：[Degree]

回転時間[mS]
＝Transfer Angle[Degree]÷(ステップ角[Degree]×Motor Speed[step/s])×1000
*Half step設定時は2倍の回転時間となります

750[Degree]/(7.5[Degree]×10[step/s])×1000

条件2）

- Motor Speed単位：[rpm]
- Transfer Unit：[Degree]

回転時間[mS]
＝Transfer Angle[Degree]÷(Motor Speed[rpm]×360[°])×60[秒]×1000
*Half step 設定時は 2 倍の回転時間となります

1200[Degree]/(20[rpm]×360[°])×60[秒]×1000

条件 3）

- Motor Speed単位：[step/s]
- Transfer Unit：[Step]

回転時間[mS]
＝Transfer Step[Step]÷Motor Speed[step/s]×1000
*Half step 設定時の回転時間も同様です

500[Step]/50[step/s]×1000

条件 4）

- Motor Speed単位：[rpm]
- Transfer Unit：[Step]

回転時間[mS]
＝Transfer Step[Step]×ステップ角[Degree]÷(Motor Speed[rpm]×360[°])×60[秒]×1000
*Half step 設定時は 1/2 の回転時間となります

800[Step]×7.5[Degree] /(100[rpm]×360[°])×60[秒]×1000

脱調とは

ステッパモータはコイルの励磁状態に同期してロータの位置が遷移することでモータが回ることは前述のとおりです。制御信号による励磁状態の遷移が早すぎたり、モータにかかる負荷が大きすぎることでロータが追従できず、モータが振動したり停止することがあります。この現象を脱調（だっちょう）といいます。

モータによって、回転できる最高速度や、負荷の最大値は異なります。

またモータの駆動電流にも依存しますので、LV8548の場合は電源電圧を調整することでこれらの特性が変化する場合があります。

基板レイアウトの留意点

モータドライバICを実装する基板は、電流、電力消費による発熱、モータ駆動によるノイズなどを考慮したレイアウトを施さないと、期待されるパフォーマンスが得られない場合があります。

LV8548は単純な機能の製品であるため、比較的設計の余裕度はありますが、ポイントを紹介します。

1. VCC（電源）－GND間には必ずコンデンサを設置する
   モータを回すために電源からモータに電流が供給されます。その瞬間、電源電圧が降下します。PWM制御の場合、この現象が頻発するため、ICの内部回路の動作が不安定になることがあります。この電圧降下を抑えるため、コンデンサを設置します。
   このコンデンサはICのできるだけ近くに設置することが重要です。
2. VCC、GND、OUTのラインは太く、短く
   この3種類のラインは大きな電流が流れます。細く、長いラインは抵抗を持つことになり、発熱や電圧降下の原因になります。
   多層基板で設計する場合は、層間の同じライン同士をスルーホールで接続し、抵抗分をさげる方法もあります。
3. 制御回路系のGNDとパワー系のGNDは分けて配線する
   ICによっては2種類のGNDピンを持っているものがあります。IC内部の制御回路系のGNDピンをSGND、モータ駆動電流が流れるパワー系のGNDをPGNDなどとしています。1で説明した電源電圧の降下とは逆に、モータ駆動電流がGNDに流れることで、GNDの配線抵抗と電流の関係で、GNDのレベルが持ち上がります。この影響を制御回路が受けると、制御の基準となるGNDレベルが変動するため、誤動作の原因となります。これを避けるため、SGNDとPGNDは極力分けて配線し、VCC−GND間コンデンサが設置されているGNDの位置で1点あわせするのが理想です。
   LV8548の場合はSGND、PGNDの区別はありませんが、ArduinoのGNDとの接続ラインは基板裏面の半分に大きな面積をとり、コンデンサの真裏でLV8548のGNDに接続しています。

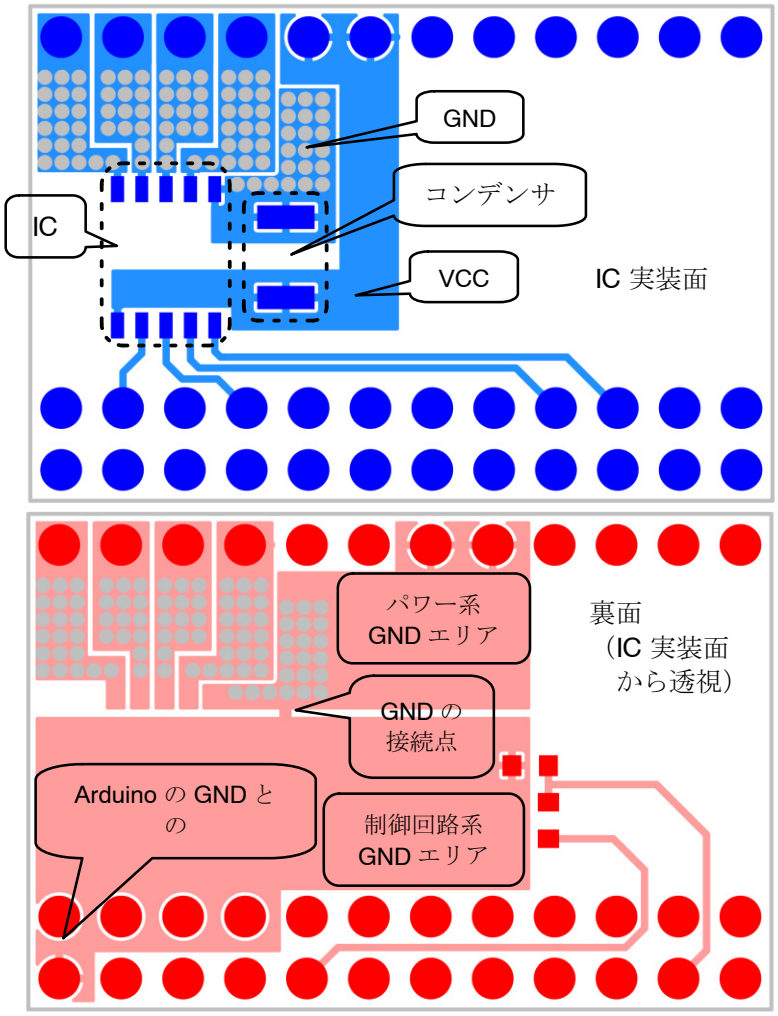


図 19. 基板レイアウトの留意点

ボード回路図

モータ ドライバモジュール (LV8548MCSLDGEVB)

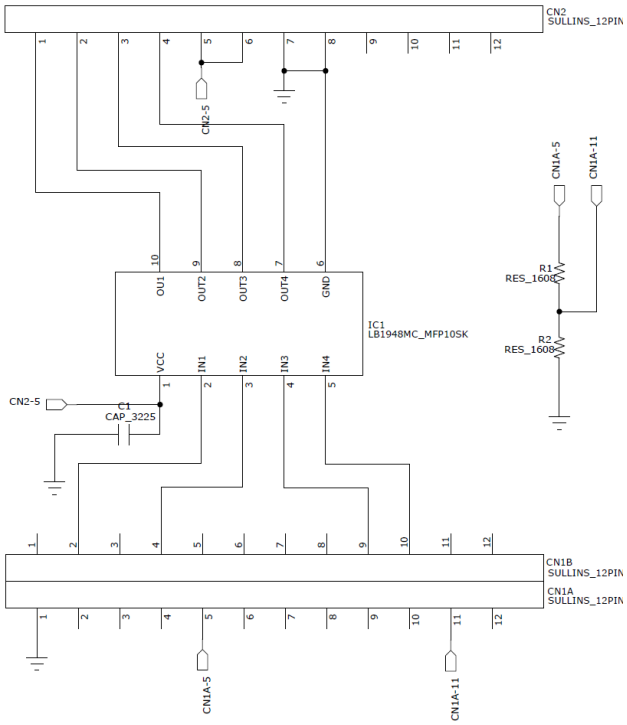


図 20. LV8548MCSLDGEVB モジュール回路図

表 23. LV8548MCモータ ドライバモジュール 部品表

部品記号	数量	部品名	値	許容差	サイズ	メーカ	製品名
IC1	1	モータドライバ	−	−	MFP10SK	オン セミコンダクター	LV8548MC
R1	1	チップ抵抗	未定	±5%	1608(0603Inch)		
R2	1	チップ抵抗	未定	±5%	1608(0603Inch)		
C1	1	チップコンデンサ	10μF，50V	±20%	3225(1210Inch)	村田製作所	GRM32ER71H106KA12L
CN1A, 1B	1	ピンヘッダ	12 pins x 2	−	30.48 x 5.08	Wurth Electronik	6, 13E+10
CN2	1	ピンヘッダ	12 pins	−	30.48 x 2.54	Wurth Electronik	6, 13E+10
PCB	1	基板	−		30.48 x 20.32		

R1，R2は製品出荷時には実装していません。

ベースボード (ONBB4AMGEVB)

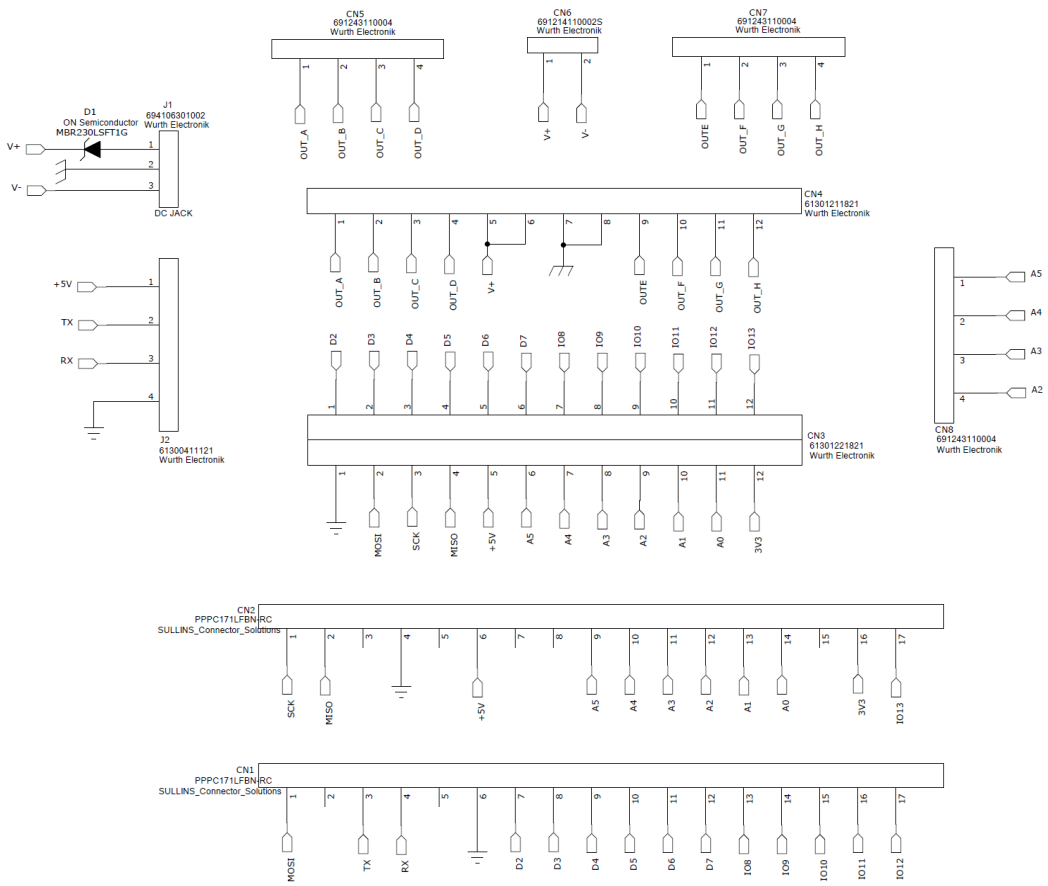


図 21. ONBB4AMGEVB ベースボード回路図

- 回路図内の同じピン名同士は配線で接続されています。
- DCジャックにコネクタを挿入すると、CN6のV−はオープンとなります。
- Arduinoの電源はUSBコネクタから供給されます。スタンドアローン動作時はスマートフォン充電器からのUSB給電も可能です。
- Arduino，J2のGNDとDCジャックのGNDまたはCN6のV−はモータドライバモジュールが挿入されることによって接続します。

表 24. ベースボード 部品表

部品記号	数量	部品名	値	許容差	サイズ	メーカ	製品名
D1	1	ダイオード	−	−	SOD123	ON Semiconductor	MBR230LSFT1G
CN1,2	2	Arduino Micro用コネクタ	−	−	Φ1.02 x17 −2.54 pitch	SULLINS connector solutions	PPPC171LFBN−RC
CN3	1	モジュール用コネクタ	−	−	Φ1.02 x12 x2lines −2.54 pitch	Wurth Electronik	61302421821
CN4	1	モジュール用コネクタ	−	−	Φ1.02 x12 −2.54 pitch	Wurth Electronik	61301211821
CN5,7,8	3	モータ接続用コネクタ	−	−	Φ1.1 x4 −3.5 pitch	Wurth Electronik	691243110004
CN6	1	電源接続用コネクタ	−	−	Φ1.1 x2 −3.5 pitch	Wurth Electronik	691214110002S
J1	1	DCジャック	−	−	9.0 x 14.5	Wurth Electronik	694106301002
J2	1	UART用ピンヘッダ	−	−	Φ1.1 x4 −2.54 pitch	Wurth Electronik	61300411121
C1	1	電解コンデンサ	100 μF, 50 V	±10%	−	Wurth Electronik	860020674015
PCB	1	PCB	−	−	80 x 60		

ベースボードの代わりに自作基板などを使用する場合はVCC−GND端子間に必ずC1相当の電解コンデンサを設置してください。未設置はモジュールの破壊、故障の原因となります。

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative