M5Stack GRBL 13.2

2022.08.27 内井

公式ドキュメント：https://docs.m5stack.com/en/module/grbl13.2
サンプルコード：https://github.com/m5stack/MODULE_GRBL13.2/tree/master

概要

- M5Stackで使用できるGRBLモジュール
- Motor Driver: DRV8825PWPR
- Controller Chip: ATmega328P-AU
- 1台で最大3つのステップモータを制御できる
- I2Cアドレスを変更すれば2台スタック可能
- 外部リミットスイッチを接続できる
- コマンドはGRBL0.8系で通用する

GRBL0.8系：https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.8

はじめに

チップマウンター（基板部品を基板上に配置してくれる機構）を作ることに挑戦し、その時にGRBLを用いてステップモーターを制御するということをやった.
そこでほぼ初めてGRBLを使ってモノづくりをしてみたが、以下の壁にぶつかった.

- GRBLについての情報がネットにあまり載っていない
  （M5Stack GRBL 13.2に関する情報なんてほとんどない）
- いろんなバージョンが混ざってよく分からない
  （Arduinoとか、CNCオープンソースとか. GRBLにも0.8系と0.9系と1.1系がある）
- そもそもGRBLについての基礎知識すらなくてよく分からない

とにかく情報がなく、公式ドキュメントを読めるだけの知識もなく、自力でしらみつぶしで検証することが多かった.
結局分からなかったものもあるが、多くは正解に辿り着いたと思う.
結果的に、公式ドキュメントが間違っていたり、公式のサンプルコードが間違っていたりと、いろいろと穴だらけだった. またそもそも初心者つぶしのトラップがいくつも潜んでいることが分かった.

今後の自分のために、そして誰かのためにもドキュメントを残しておく.

GCODE

初心者のために. そもそもGcodeとは何か、ステップモーターをどうやって動かすのか.

ステップモータを動かす

動かし方１:
M5StackとArduinoIDEをUSBで接続する（シリアル通信ができる状態にする）
M5StackにM5_serial.inoを書き込む
シリアルモニタの入力窓から、コマンドを入力する.
例えば、G91G0X1Y1Z1、と入力すれば現在位置から相対座標で（1,1,1）に動いてくれる.

動かし方２:
M5_Serial.inoのLoop関数内に、以下のコードを入れる.
「if(M5.BtnA.wasPressed()){ _GRBL0.Gcode(“G91G0X1Y1Z1”);」
M5StackとArduinoIDEをUSBで接続する（シリアル通信ができる状態にする）
M5StackにM5_serial.inoを書き込む.
M5StackのAボタンを押す.

ステップモータが動かない

いくつかよくあるミスを書いておく.

・供給されている電圧が低い / 電源が供給されていない
　パソコン給電ではなく、ACアダプタ等の外部電源が必要（適切な供給電圧で）

・配線がうまくされていない
　ステップモーターには大きくユニポーラ式とバイポーラ式がある.
　またコードが途中で（親切心から）クロスしているものもある.
　ステップモータの種類によって配線が違うので、ドキュメントを見ながら配線する.
　特に４線のうち真ん中２本がどう繋がっているかが問題になりやすい. 違った場合は真ん中２本だけをハンダで繋ぎかえるなどの対応をする

・GRBLライブラリが入っていない
　ファームウェアはGRBLモジュール側に事前に書き込まれているため、不要.
　しかしM5Stack本体にはGRBLライブラリ（例えば、GrblControl.hとGrblControl.cppなど）を書き込む必要がある. 制御コードにインポートさせておくと良い.

・内部変数が適切でない
　後述の内部変数の設定が適切でない場合がある.
（パルスが細かすぎてモータが動かない、トルクが足りなくて動かない等）
　内部変数を見直してみると良い.

・ロックがかかっている
　$17を1に設定していると、初めは必ずロック状態になる. ロック状態ではどの小アンドも機能しない. $Xか$Hで解除してからコマンドを入力するか、初めから$17を0にしておく
　シリアル通信でコマンド”@”を送信すると、現在のステータスを返してくれる.「ALART」と返ってきた場合は、ロック状態である可能性が高い.（問題なければ「IDLE」と返ってくる）

内部変数 / Internal variable

GRBLではモーターの速度やパルスの周期などのパラメータを事前に書きこんでおく.
各変数は書き込んだ時点で、自動でEEPROMに保存される.
ただ何かのエラーの影響で、変数が知らない間に変更されることがあるので、挙動が急におかしくなった時は、変数が正常に設定されているか確認すると良い.

内部変数の設定方法

ArduinoIDEとGRBLモジュールをUSBでつなぐ（シリアル通信できるようにする）
シリアルモニタを開いて、入力窓から直接入力するのが、一番簡単.
以下のようにArduino上に含めておけば、そこからモジュールに書き込むこともできる.
（注意：一度に5つ以上の変数は設定できない）

M5grbl_serial.ino | GrblControl.cpp | GrblControl.h

```
27
28    // GRBL0 Setting
29    //_GRBL0.Init();
30    _GRBL0.Gcode("$0=160");   //X[step/mm]
31    _GRBL0.Gcode("$1=160");   //Y[step/mm]
32    _GRBL0.Gcode("$2=160");   //Z[step/mm]
33    //_GRBL0.Gcode("$3=30");    //step pulse
34    //_GRBL0.Gcode("$4=160");   // default feed rate[mm/min] (for G1)
35    //_GRBL0.Gcode("$5=160");   // default seek rate[mm/min] (for G0)
36    //_GRBL0.Gcode("$6=32");     // step port invert mask (invert X&Y axis, 32(X))
37    //_GRBL0.Gcode("$8=50");   // acc[mm/sec^2]
38    //_GRBL0.Gcode("$15=0");    // invert ST
39    //_GRBL0.Gcode("$16=0");    // enable hard limit[bool](1=enable)
40    //_GRBL0.Gcode("$17=1");    // enable homing[bool](1=have to homing first)
41    //_GRBL0.Gcode("$18=32");    // homing direction(32(X))
42    //_GRBL0.Gcode("$19=160");  // homing feed rate[mm/min]
43    //_GRBL0.Gcode("$20=160");  // homing seek rate[mm/min]
44    //_GRBL0.Gcode("$21=1");
45    //_GRBL0.Gcode("$22=0");    // disable homing pulloff[mm]
46
```

内部変数の設定

公式ドキュメントに記載されているものは誤りなので、気をつけること.
Arduino式と設定内容が違うので注意すること（ネットのものはだいたいArduino式）
以下、実践の下、正しいと思われるものを載せる.

$0 = x [step/mm]　脱調（振動、騒音）している場合はここを変更すると良い
$1 = y [step/mm]
$2 = z [step/mm]
$3 = step pluse
$4 = feed rate [mm/min] (for G1)
$5 = seek rate [mm/min] (for G0)
$6 = step port invert mask　設定の仕方は後述
$8 = acc [mm/sec^2]
$10 = settings.mm_per_arc_segment
$11 = settings.n_arc_correction
$12 = settings.decimal_places

$13 = REPORT INCHES
$14 = BITFRAG_AUTO_START (=1)
$15 = invert ST (0=default, 1=モータの動作/停止が反対)
$16 = enable hard limit (0=disable, 1=enable)
$17 = enable homing (0=disable, 1=enable)　1だとhoming($H)されるまでlockされる
$18 = settings.homing_dir_mask　設定の仕方は後述
$19 = settings.homing_feed_rate
$20 = settings.homing_seek_rate
$21 = settings.homing_debounce_delay　homingを確かめる時間スパン[msec]
$22 = settings.homing_pulloff　homingした後に少し戻る

各変数について詳しい説明
（少し数字と内容が違うかもしれないが、参考程度に．）
https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.8

$6, $18の設定方法

$6 = default時の軸の正方向を指定する
$18= homing時の軸の正方向を指定する

各軸にビットが指定されている (X=5, Y=6, Z=7).
1をこれだけビットシフトしたものの論理和として指定する.

分かりやすく言うと、
軸それぞれに以下の数字が与えられている
X = 00100000 (2進数表記) = 32 (10進数表記)
Y = 01000000 (2進数表記) = 64 (10進数表記)
Z = 10000000 (2進数表記) = 128 (10進数表記)

変更したい軸の数字を全て足したものを10進数で指定する.
（例）
X軸だけを変更したい (32(X)なので)　$6=32
X軸とY軸を変更したい (32(X)+64(Y)=96)　$6=96
3軸ともに変更したい (32(X)+64(Y)+128(Z))　$6=224

内部変数を設定する時の注意点

モジュールのバッファサイズが小さいので注意

一般的なGRBLでは、コマンド “$$” を入力すると、内部変数の一覧を書き出してくれるようになっている（自分が過去に何を設定したかが分かるようになっている）.

M5Stack GRBL13.2でも同様のコマンドがあるが、バッファサイズの都合で10byteしか出力できない. なので、ほとんど見れない（$0～$4くらいまでは見れる）. 設定した数字はどこかにメモっておくか、しっかり設定できているか不安ならばその都度書き込むしかない.

また内部変数の設定は1つずつ行うのが最善．バッファサイズが10byteしかないので、一度に大量の変数（5つ以上）を書き込むと全部書き込めない．オーバーした分は書きこまれないので注意
.

GRBLコマンド

GRBLは特定のコマンドを用いて操作する.
よく使うものをいくつか載せておく.

- $$ = 内部変数の書き出し
- $X = ロックの解除（$17=1でロックされている状態を解除する）
- $H = homing（原点復帰）を実行する
- F = feed rateを指定して移動（G0X1Y1Z1F200など）
- G0 = 単純移動（G0X1Y1Z1など）移動用で加工には向かない
- G1 = 直線移動（G1X1Y1Z1など）
- G90 = 絶対位置系で移動（G90G0X1Y1Z1など）
- G91 = 相対位置系で移動（G91G0X1Y1Z1など）
- G92X0Y0Z0 = 現在位置を原点に設定

他にも知りたい場合は以下の情報を参考にすると良い
（少し変数の内容が違うが、GRBL13.2モジュールにおいては、このドキュメントに書いてあるのが正しい）

- https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.8
- https://bbs.avalontech.jp/t/grbl-g/750/2

リミットスイッチ

homingをするときなどにリミットスイッチを使う.
（リミットスイッチにぶつかるまで動き続けるみたいな処理をする．）

一般的にリミットスイッチは、物理的な接触によって、内部スイッチが閉じる（または開く）ことによって、信号を変化させ、ぶつかったことを知らせる.

モジュールに接続するには、リミットスイッチ3つ（X, Y, Z）を1つのコネクタにまとめなくてはいけない．以下のように配線する.

COM = 3本それぞれのGNDと接続する
X = X軸スイッチの信号線（右側）
Y = Y軸スイッチの信号線（右側）
Z = Z軸スイッチの信号線（右側）

DIPスイッチ

GRBLモジュールの中にDIPスイッチ（4つ並んでいるもの）が見える.
そのスイッチを使って、マイクロステップ幅とI2Cアドレスを設定する.

マイクロステップは左側3つ（1,2,3番）の組み合わせで設定する
組み合わせは公式ドキュメントに載っている.

I2Cアドレスは一番右（4番）で設定する.
OFF側がDefaultアドレス（0x70）、ON側が代わりのアドレス（0x71）

GRBL Tips

GRBLの状態を知りたい

GRBLはGcodeを送信することで、ステップモータを制御することができる.
しかし、Gcodeを送信するだけで終わりなので、処理が終了した合図は返ってこない.
（前の処理が終わったら次のコマンドを送信するというデザインにしたいが、前の処理が終わったのかどうか誰も教えてくれない）

モーターの状態が今どうなっているのかを知ることはできない.
（知るにはセンサーを入れてマイコンプログラムを書くことになる）

しかしGRBLモジュールがどうなっているのかは知ることができる.
基本的に、パソコン→M5Stackの向きでシリアル通信をして、Gcodeを送信している.
GRBLの状態を知りたい時は、パソコン→M5Stackの方向にコマンド”@”を送信すれば、
M5Stack→パソコンの方向に現在のステータスを返してくれる.

ステータスはいくつかある.
・IDLE ＝ 待ち
・RUN ＝ 何かの処理を実行中
・BUSY = 何かの処理を実行中
・HOME = homing中
・ALARM = 警告またはロック状態をさす（ロック解除でIDLEに変わる）

参考

参考になった文献やサイトたち（閲覧：2022.08.27）

前述のものもあるが、最後にまとめてリンクを並べておいた.

M5Stack GRBL 13.2 公式ドキュメント
https://docs.m5stack.com/en/module/grbl13.2

M5Stack GRBL 13.2 公式サンプルコード
https://github.com/m5stack/MODULE_GRBL13.2/tree/master

GRBL0.8系について
https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.8

秋田純一先生のGitHubデータ
https://github.com/akita11/ChipMounter

- M5Stack GRBL 13.2 を使うためのライブラリ＆サンプルコード（動作確認済み）
  https://github.com/akita11/ChipMounter/tree/main/M5StackFW/M5grbl_serial

- M5Stack GRBL 13.2 に関する補足ドキュメント
  https://github.com/akita11/ChipMounter/blob/main/M5StackFW/M5GRBL_memo.md

秋田純一先生のチップマウンタ開発記録
https://note.com/akita11/n/ne17326245943

GCODEについて資料
https://bbs.avalontech.jp/t/grbl-g/750/2

ステッピングモーターの動作原理（動画）
https://www.youtube.com/watch?v=g18dOXJ1Lgc&t=4s