ブラシレスモータスタータキット(RA6T3/RA4T1)
[ソフトウェア　サンプルプログラム編]
取扱説明書

ルネサス エレクトロニクス社　RA6T3/RA4T1(QFP-64 ピン)搭載
ブラシレスモータスタータキット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.0.0.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

⚠警告	取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される
⚠注意	取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

⚠警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプ点灯中に電源の切断を行わないでください。

製品の故障や、データの消失の原因となります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

CD 内容

添付「ソフトウェア CD」には、以下の内容が収録されています。

・チュートリアル
　TUTORIAL フォルダ以下
・サンプルプログラム
　SAMPLE フォルダ以下
・プログラムのバイナリ（srec ファイル）
　BIN フォルダ以下
・マニュアル
　manual フォルダ以下

注意事項

本マニュアルに記載されている情報は、ブラシレスモータスタータキットの動作例・応用例を説明したものです。

お客様の装置・システムの設計を行う際に、本マニュアルに記載されている情報を使用する場合は、お客様の責任において行ってください。本マニュアルに記載されている情報に起因して、お客様または第三者に損害が生じた場合でも、当社は一切その責任を負いません。

本マニュアルに、記載されている事項に誤りがあり、その誤りに起因してお客様に損害が生じた場合でも、当社は一切その責任を負いません。

本マニュアルの情報を使用した事に起因して発生した、第三者の著作権、特許権、知的財産権侵害に関して、当社は一切その責任を負いません。当社は、本マニュアルに基づき、当社または第三者の著作権、特許権、知的財産権の使用を許諾する事はありません。

1. サンプルプログラム

1.1. プログラム仕様・動作

サンプルプログラムは、TUTORIAL_B2（相補 PWM 駆動）, TUTORIAL_C2（相補 PWM 駆動+センサレス）チュートリアルの内容を踏まえ、回転数や回転方向を制御できるものとなっています。

参照プロジェクト：RA6T3_BLMKIT_SAMPLE, RA4T1_BLMKIT_SAMPLE

（アーカイブファイル：RA6T3_BLMKIT_SAMPLE.zip, RA4T1_BLMKIT_SAMPLE.zip）

本プログラム(RA6T3_BLMKIT_SAMPLE, RA4T1_BLMKIT_SAMPLE)は、以下の仕様を実装しています。

・ホールセンサを使用（ソースファイルの変更でセンサレス化も可能）
・VR にて回転数を変更
・SW1 回転・停止を制御
・SW2 回転方向を制御
・SW3 ブレーキ
・過電流保護停止機能(*1)
・過熱停止機能（デフォルトでは、50℃で停止）
・進角調整機能（シリアル端末を接続し、キーボードの qwe で変更）

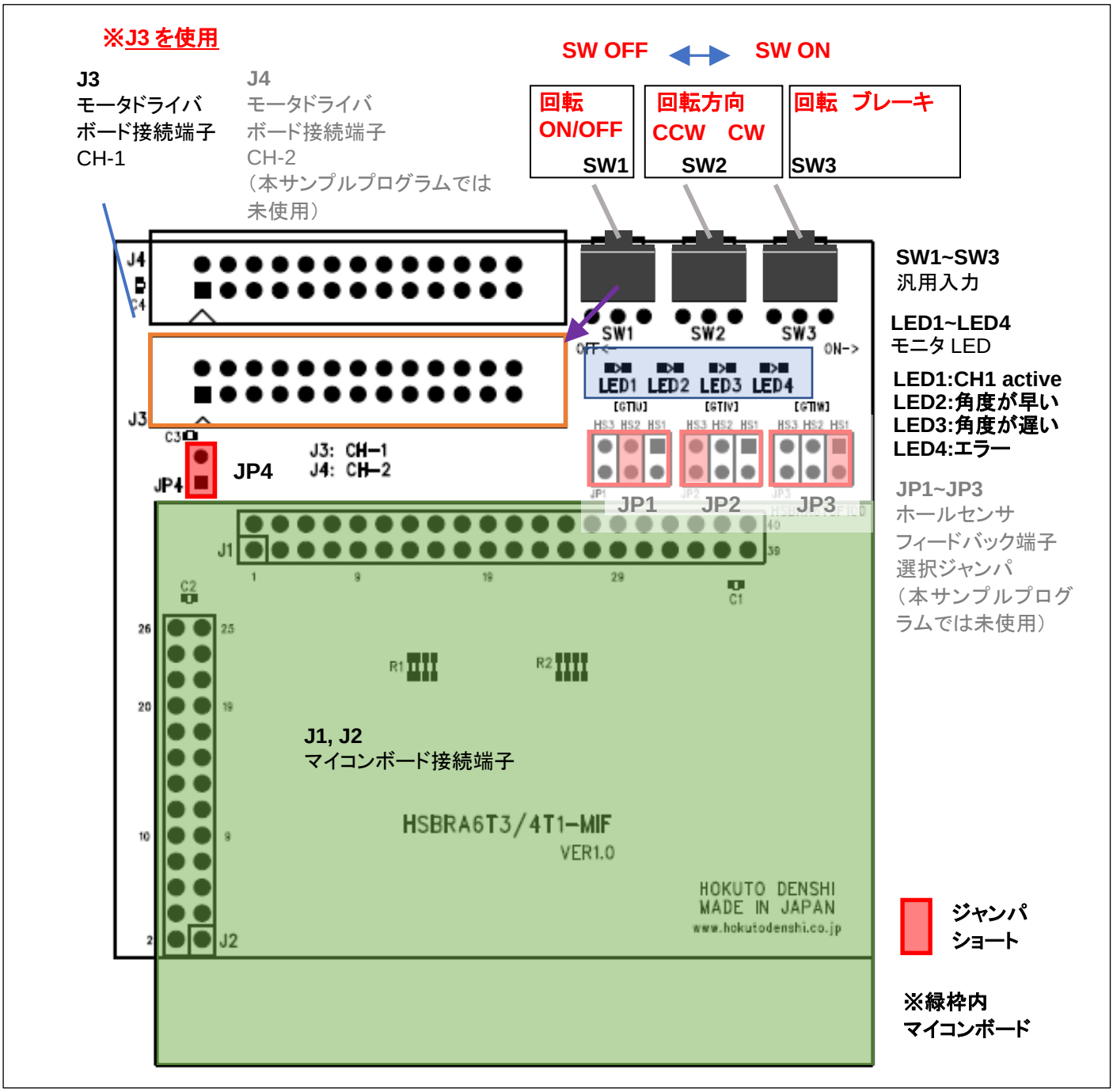
(*1)
・1 回でも 8A(peak)を超えると停止（デフォルト無効化）
・10ms 間に 180 回以上過電流が検出された場合停止（デフォルト有効化）
・1s 間に 1000 回以上過電流が検出された場合停止（デフォルト有効化）

	OFF	ON
SW1	モータ停止	モータ運転
SW2	回転方向 CCW	回転方向 CW
SW3	通常（回転）	ブレーキ

	最低	最大
VR	1,000[rpm]	12,000[rpm]

チュートリアル B2, C2 では VR で duty を直接制御していましたが、本サンプルプログラムでは VR は回転数を制御します。

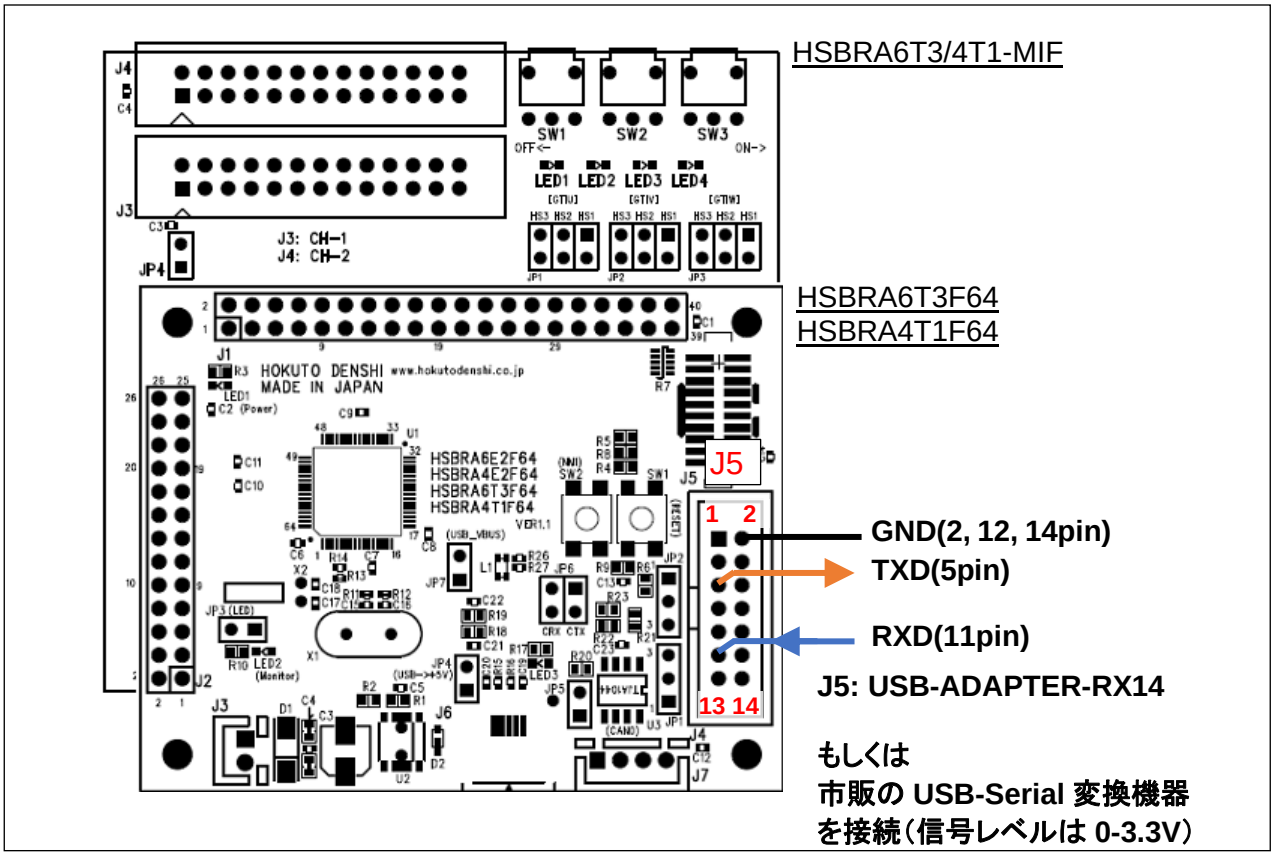
－モータドライバボードの設定等－



※マイコンボード上の LED2 は通常制御移行で ON、停止、始動制御時は OFF

ーシリアルポートの接続ー

※シリアルポートの接続は、モータを動かす上で必須ではありませんが、回転数や温度のモニタに必要です。



・起動時のシリアル端末の表示

```
Copyright (C) 2023 HokutoDenshi. All Rights Reserved.

RA6T3/RA4T1 / BLUSHLESS MOTOR STARTERKIT SAMPLE

Motor driver board connection check...
   CH-1 Connected.
```

115,200bps
の設定で端末を開いてください

・情報表示（3 秒に 1 回） SW1 を ON 方向に倒しモータを回転させた状態

```
                                CH-1
Motor Driver Board          :   Connect
Active                      :   o
holl sensor                 :   Motor inside
target speed([rpm])         :   5666
current target speed([rpm]):   5666
forward angle([deg])        :       0
target direction            :   CCW
rotation speed([rpm])       :   5220
rotation speed[ave]([rpm]) :   5527
Temperature(A/D value)      :   2010
Temperature(degree)         :      25
VR(A/D value)               :   1564
duty[%]                     :   44.9
```

各種情報を表示
（一部チュートリアルと相違点があります）

・画面表示内容

項目	表示例	内容	備考
Mortor Driver Board	Connect / NoConnect	モータドライバボード接続有無	起動時に CH1 のコネクタ(J3)にモータドライバボードが接続されているかをチェックしてその結果を表示
Active	o / x	SW1 が ON で回転制御対象の時は o	
holl sensor	Motor inside / Phase voltage	モータ内蔵のホールセンサを使用するか、相電圧で回転を検出するか（センサレス駆動）	blm.c 内でどちらの方式とするか選択
target speed([rpm])	5666	VR ツマミで設定した目標回転数([rpm])	
current target speed([rpm])	5666	現在の制御回転数[rpm]	磁界印加角度増分に対応
forward angle([deg])	0	進角調整値[°]	
target direction	CCW / CW / BREAK/ STOP	回転方向	
rotation speed([rpm])	5220	現在の回転数[rpm]	
rotation speed(ave)([rpm])	5527	現在の平均回転数	直近の 10ms 毎 16 値の平均
Temperature(A/D value)	2010	温度センサーA/D 変換値	0~4095
Temperature(degree)	25	温度センサー摂氏温度[℃]	
VR(A/D value)	1564	VR の A/D 変換値	0~4095
duty[%]	44.9	現在の duty 比[%]	

・キーボードからの操作

シリアル端末にキーボードから進角調整を行う事が可能です。（キーボードの qwe を使用）

	進角調整 （進み方向） +1°	進角調整 （遅れ方向） -1°	リセット(=0)
CH-1	q	w	e

キーボードの q を 5 回押すと、

```
                              CH-1
Motor Driver Board          :  Connect
Active                      :  o
holl sensor                 :  Motor inside
target speed([rpm])         : 10309
current target speed([rpm]): 10309
forward angle([deg])        :  0
target direction            :  CCW
rotation speed([rpm])       : 11100
rotation speed[ave]([rpm]) : 10192
Temperature(A/D value)      :  2227
Temperature(degree)         :  30
VR(A/D value)               :  3119
duty[%]                     :  78.5
```

```
                              CH-1
Motor Driver Board          :  Connect
Active                      :  o
holl sensor                 :  Motor inside
target speed([rpm])         : 10300
current target speed([rpm]): 10300
forward angle([deg])        :  5
target direction            :  CCW
rotation speed([rpm])       : 11760
rotation speed[ave]([rpm]) : 10068
Temperature(A/D value)      :  2198
Temperature(degree)         :  29
VR(A/D value)               :  3118
duty[%]                     :  75.5
```

進角調整値(forward angle)の値が増え、進角調整が掛かった状態となります。

1.2. モータ制御アルゴリズム

TUTORIAL_B2「相補 PWM 信号での駆動」をベースとしています。

※「チュートリアル編」のマニュアルも併せて参照ください

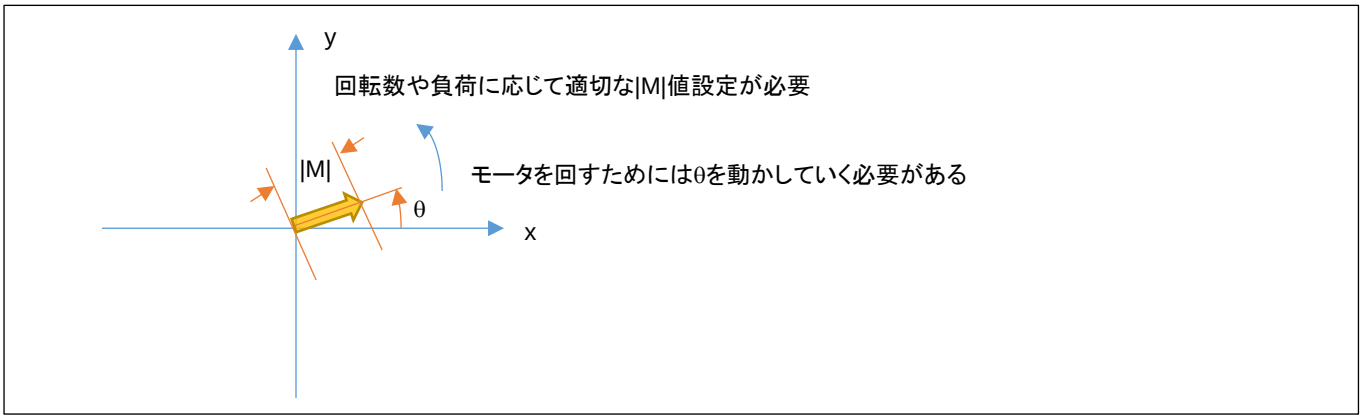
TUTORIAL_B2 のプログラムに、
・回転数(duty)制御
・始動制御
を追加したものが、本サンプルプログラムとなります。

基本的な制御は相補 PWM なので、プログラムとしてはその時々の

・ベクトルの大きさ|M| （duty 比…g_duty 変数値）
・ベクトルの角度θ （磁界印加角度…g_angle 変数値）

を決めて設定するという動作です。

（|M|とθの値は、実際には UVW 3 値の PWM 値に変換されて、タイマが駆動されます。しかし、その部分は機械的な処理を行うだけですので、プログラムの肝は「いま、このタイミングでの|M|,θ値を決定」する事となります。）



※|M|,θ値を U, V, W の各相の duty 値に変換するアルゴリズムは正弦波駆動です
（この変換の話は「ソフトウェア チュートリアル編」のマニュアルを参照ください。本サンプルプログラム内には、「ソフトウェア チュートリアル編」のマニュアルで示している、正弦波以外の変換アルゴリズムのソースが含まれていますので、アルゴリズムの違いを見ることができます。）

－正弦波駆動以外のアルゴリズムを選択する場合－

blm.c 内の

```
//UVW 分解方法:(1)の正弦波駆動を指定 ※7 行の内いずれかのコメントアウトを外す(関数ポインタなのでプログラムの実行中に切り替えても OK)
blm_angle_to_uvw_duty = blm_angle_to_uvw_duty_sin;                  //(1)正弦波駆動(デフォルト)
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty_sin_post;           //(2)正弦波駆動, duty を後から乗算
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty_sin_3harmonic;      //(3)正弦波 3 倍高調波重畳
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty_sin_3harmonic_post; //(4)正弦波 3 倍高調波重畳, duty を後から乗算
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty1;                   //(5)別バージョン 1, 全方向に 86.6%のパワー
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty2;                   //(6)別バージョン 2, 60°で割り切れる角度では 100%のパワー
//blm_angle_to_uvw_duty = blm_angle_to_uvw_duty2x;                  //(7)別バージョン 2' 別バージョン 2 を直線近似したもの
```

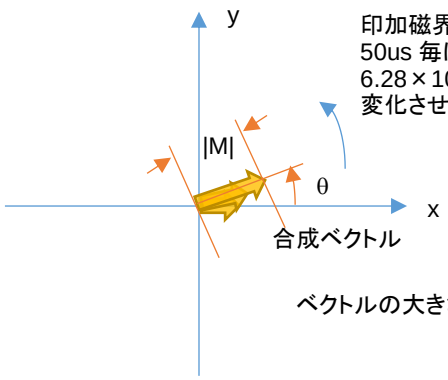
blm_angle_to_uvw_duty をどの関数に紐付けするかを変えてください。

7 種類の変換方法の関数が用意されています。

※関数ポインタとしているので、プログラム実行中でも変更する事が可能です

1.2.1. 始動制御

フェーズ 1（g_phase = BLM_PHASE_1(=1) のとき）の制御、SW1 を ON にして、モータを停止状態から立ち上げる際の動作です。



印加磁界は
50us 毎に
6.28×10^{-3}[rad]
変化させてゆく

6.28×10^{-3}
→回転数は固定
→1200rpm に相当する角度

→1200rpm: 20 回転/s, 50ms/1 回転
50us 毎に角度を動かしていくので、50us 毎に 1/1000 回転ずつ印加角度を動かしていく（50ms/50us = 1000)
1 回転で、2π[rad]なので、50us あたり $2\pi/1000 = 6.28 \times 10^{-3}$ [rad]となる

ベクトルの大きさ=duty を基本的に増加させていく

・印加磁界の角度(θ)は 1200rpm に相当する角度（6.28×10^{-3}[rad]）を 50us 毎に）動かしていく
・duty(|M|)は 10ms 毎に調整（条件により増加または減少）

となるような動作です。

始動制御の方式は、TUTORIAL_B で行っている事と同様のものとなります。

TUTORIAL_B では、
・回転数（＝印加磁界の切り替わりタイミング）は一定
・duty は VR に連動して変わる
という方式でした。

サンプルプログラムの、始動制御は
・回転数（＝印加磁界の切り替わりタイミング）は一定
・duty は回転が安定するまで duty 値を増やしていく（場合によっては減らす）
という方式です。TUTORIAL_B で VR ツマミを回して手動で変えていた duty をプログラムで増加させているだけで、行っている事は同一です。

duty は最初は 0 ですが、徐々に増やしていきます。

このとき、
・回転安定回数
→ホールセンサ切り替わり時、
目標の回転方向とあっていればカウンタをインクリメント
目標の回転方向と逆であればカウンタをデクリメント
・回転数（速度）
の 2 項目をモニタリングします。

始動制御中に、

回転数が BLM_START_RPM_LOWER(=800rpm)未満
または、回転安定回数が BLM_DIRECTION_STABLE_THRESHOLD1(=6)以下であれば duty 増加

回転数が、BLM_START_RPM_UPPER(=1500rpm)以上
かつ、回転安定回数が BLM_DIRECTION_STABLE_THRESHOLD1(=6)以上であれば duty 減少
（手動で調整する際の「ツマミを回しすぎたので戻す作業」のイメージ）

の様に duty を調整します。

また、

回転数が BLM_START_RPM_LOWER(=800rpm)以上
かつ、回転安定回数が BLM_DIRECTION_STABLE_THRESHOLD2(=24)以上

であれば、始動制御の状態を抜けて通常制御に移行します。

回転安定回数は、最低回転数(BLM_START_RPM_LOWER=800[rpm])に達した後、

・目標回転方向と実際の回転方向が一致

の際に、カウンタ値が加算され、カウンタ値が BLM_DIRECTION_STABLE_THRESHOLD2(=24)（1200rpm で 4 回転、0.2 秒に相当）以上であれば、phase=2（通常制御）に移行させます。

duty を 0 から増加させていくと、duty の小さな領域では軸が振動するだけで、モータは回転しない。duty が増えてくると振動の振幅が大きくなり正回転（目標とする回転方向）と逆回転を短時間で繰り返す。duty をさらに大きくすると、安定して回転するようになる。duty を大きくしすぎると回転数が上がりすぎるので、duty は戻す（減少させる）という動作です。

blm.h 内の

```
//始動回転数
#define BLM_START_RPM          1200
#define BLM_START_RPM_LOWER     800
#define BLM_START_RPM_UPPER     1500

//回転安定
#define BLM_DIRECTION_STABLE_THRESHOLD1 6       //1 回転分
#define BLM_DIRECTION_STABLE_THRESHOLD2 24      //1200rpm の場合 0.2s 以上回転安定時 phase2 に移行
#define BLM_DIRECTION_STABLE_THRESHOLD3 30      //回転安定検出の最大値
```

上記で、始動時の回転数や回転安定とみなす回数等を定義しています。

始動時の duty ですが、最初は大きく値を変更、徐々に変更する値を小さくしていく様にしています。

blm.c 内

```
//始動時の phase1 での duty 変化量
//                                0.20%   0.2%    0.23475%    0.46875%    0.9375%    1.875%    3.75%
7.5%
const float g_phase1_diff_array[8] = {0.002f, 0.002f, 0.00234375f, 0.0046875f, 0.009375f, 0.01875f,
0.0375f, 0.075f};
//                                0       1       2           3           4          5         6
7
volatile unsigned short g_phase1_diff_index = 7;    //7.5%から変化量を調整
```

最初は、0%→7.5%に増加させます。次は、7.5%→11.25%→13.125%の様に徐々に増分値を減らしていきます。変化量は、blm.c 内に定義しています。この値を小さくすると始動に時間が掛かる様になり、大きくし過ぎると始動時に発振の様な挙動（duty を増やし過ぎて戻る）を示します。

1.2.2. 通常制御

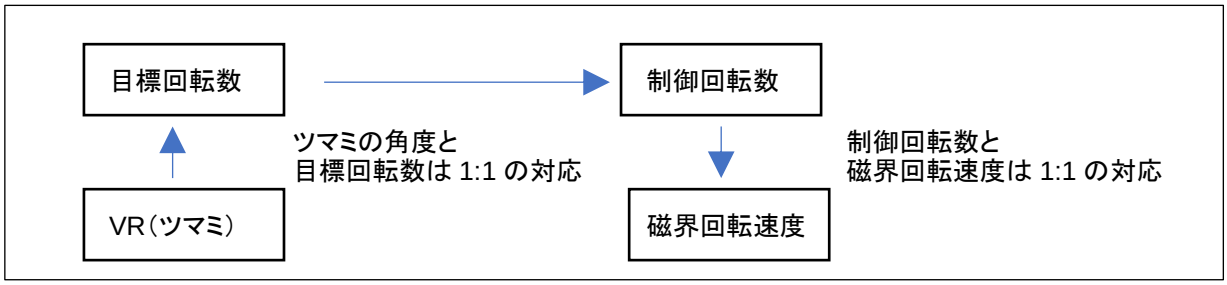
フェーズ 2（g_phase = BLM_PHASE_2(=2)　のとき）の制御です。

通常制御においても、印加磁界角度(θ)と duty(|M|)をどういう値とするのか、という話となります。

TUTORIAL_B2 では、VR ツマミで duty を変化させていたので、duty はツマミの角度によって一意に決まるという方式です。本サンプルプログラムでは、VR ツマミと対応するのは、目標回転数です。

目標回転数と印加磁界角度を 1:1 で対応させた場合、目標回転数を短時間で変更した場合、回転と印加磁界角度の整合が取れなくなります。そのため、本サンプルプログラムでは、制御回転数という概念（変数）を用意しています。

目標回転数：ユーザが VR ツマミで設定した回転数
制御回転数：プログラムで決める「今現在の設定」の回転数



最終的には、制御回転数＝目標回転数としますが、目標回転数は急激に変化することもあるので、プログラム上の設定回転数（＝制御回転数）は別に値を設けています。

モータ動作は制御回転数に制御され、制御回転数と目標回転数のすり合わせは別に行うという方式です。

1.2.2.1. 印加磁界角度の設定

・制御回転数(g_current_target_rpm)から、50us 毎の角度増分値(g_angle_diff)を算出（10ms 毎）

・角度増分値を印加磁界角度(g_angle)に加算（50us 毎）

・ホールセンサから導出される、理想印加角度に補正（ホールセンサ切り替わり時）

印加磁界角度に関しては、50us 毎に制御回転数に対応する分だけ動かしていき、ホールセンサ切り替わりのタイミングでリセット（ホールセンサ位置に応じた角度に設定）しています。

実際の回転数と制御回転数が合っていない場合でも、ホールセンサ切り替わりのタイミングで印加磁界角度はリセットされます。（ずれが累積する事がない様にしています）

1.2.2.2. duty の設定

・目標回転数と制御回転数のずれが 5%以内

→PI 制御を行う（duty の増分積算値を一定時間毎に duty に反映させる）

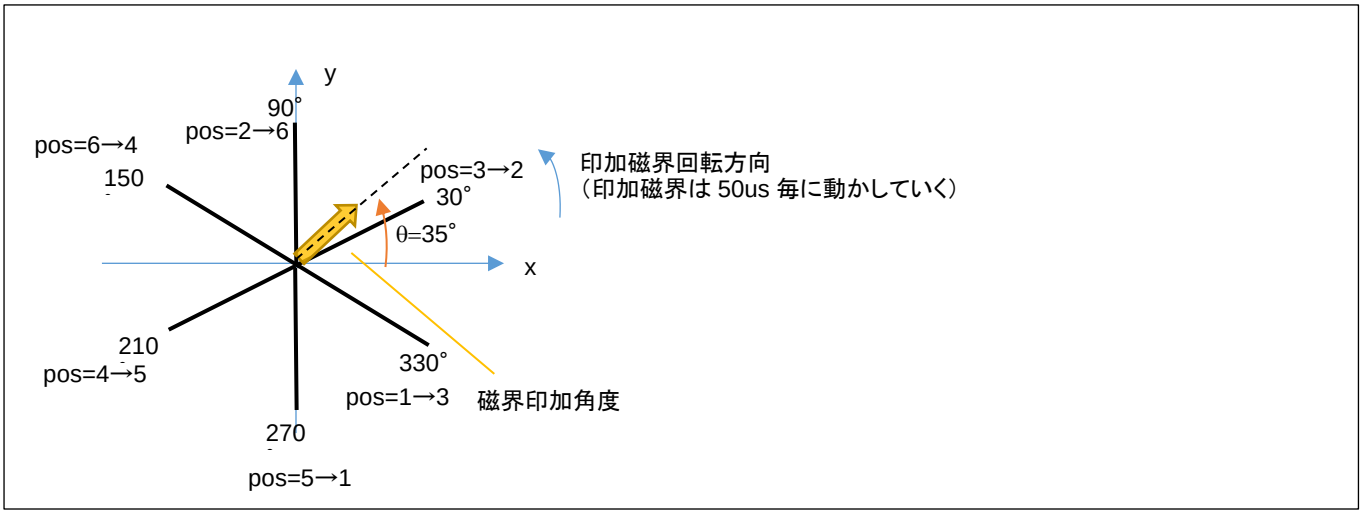
ホールセンサが切り替わったタイミングで、理想的な印加磁界角度を算出。

理想的な印加磁界角度に対し現在の印加磁界角度が BLM_ANGLE_DIFF_THRESHOLD2(=30)度以上大きい
→duty 増分積算値から BLM_DUTY_DIFF2(0.2%)を減算（ホールセンサ切り替わり毎） [LED3 が点灯]

理想的な印加磁界角度に対し現在の印加磁界角度が BLM_ANGLE_DIFF_THRESHOLD2(=30)度以上小さい
→duty 増分積算値に BLM_DUTY_DIFF2(0.2%)を加算（ホールセンサ切り替わり毎） [LED2 が点灯]

理想的な印加磁界角度に対し現在の印加磁界角度が BLM_ANGLE_DIFF_THRESHOLD1(=15)度以上大きい
→duty 増分積算値から BLM_DUTY_DIFF1(0.05%)を減算（ホールセンサ切り替わり毎） [LED3 が点灯]

理想的な印加磁界角度に対し現在の印加磁界角度が BLM_ANGLE_DIFF_THRESHOLD1(=15)度以上小さい
→duty 増分積算値に BLM_DUTY_DIFF1(0.05%)を加算（ホールセンサ切り替わり毎） [LED2 が点灯]



例えば、ホールセンサの読み取り値が 3→2 に変わった際、印加磁界角度が 35° だとすると、ホールセンサの切り替わりタイミングが遅い（実際のモータの回転数が印加磁界角度より遅れている）事となります。
ここで、印加磁界角度に対し、ホールセンサ切り替わりが BLM_ANGLE_DIFF_THRESHOLD1(=15)度以上遅れている場合は、モータの回転数を上げたいので duty 増分積算値の値に BLM_DUTY_DIFF1(=0.05%)を加算します。
逆に、印加磁界角度に対し、ホールセンサ切り替わりが BLM_ANGLE_DIFF_THRESHOLD1(=15)度以上進んでいる場合は、duty 増分積算値の値から BLM_DUTY_DIFF1(=0.05%)を減算します（回転数を落とす方向）。
印加磁界角度とホールセンサのずれがない場合は、duty が適切だとして、duty を変更しません。

印加磁界角度に対し、ホールセンサ切り替わりタイミングがもっと大きくずれている（BLM_ANGLE_DIFF_THRESHOLD2(=30)度以上）の場合は、duty 増分積算値をより大きく変化（BLM_DUTY_DIFF2(=0.2%)）させます。

duty を増加させる際には LED3 が点灯、duty を減少させる場合は LED2 が点灯します。

その後、印加磁界角度（図の黄色矢印）を、図では 30°（理想値）に変更します。ホールセンサ切り替わりのタイミングで、常にホールセンサ（＝モータの軸位置）に応じた印加磁界角度に調整します。（この動作は duty 値の設定ではなく、印加磁界角度の設定の話です。）

上記で求めた duty 増分積算値(g_duty_diff_integral)を duty 値(g_duty)に加算（10ms 毎）。

blm.h 内の

```
#define BLM_DUTY_DIFF1          0.0005f     //0.05%, ANGLE_DIFF_THRESHOLD1 のずれを検出した際の増分
#define BLM_DUTY_DIFF2          0.0002f     //0.2%, ANGLE_DIFF_THRESHOLD2 のずれを検出した際の増分
```

```
//角度のずれ
#define BLM_ANGLE_DIFF_THRESHOLD1   RAD_15_DEGREE   //15 度以上ずれている場合は、duty の微調整を行う
#define BLM_ANGLE_DIFF_THRESHOLD2   RAD_30_DEGREE   //30 度以上ずれている場合は、duty の調整を行う
```

上記で、角度のずれや角度ずれがある場合の duty 比の増減値を決めています。

・目標回転数と制御回転数のずれが 5%以上

→ずれが大きいので、duty を大きく調整する

・目標回転数と現在の回転数の割合×フィードバック係数(BLM_DUTY_FEEDBACK_RATE)を duty に乗算(10ms 毎)（比例制御）

blm.h 内

```
#define BLM_DUTY_FEEDBACK_RATE  0.01f       //1%
```

1.2.2.3. 制御回転数の設定

・目標回転数と制御回転数のずれが 5%以内

→制御回転数 = 目標回転数に設定

※ほぼずれがないので、制御回転数を最終的な回転数の目標である目標回転数に設定してしまう

・目標回転数と制御回転数のずれが 5%以上

・目標回転数と現在の回転数の割合×フィードバック係数(BLM_RPM_FEEDBACK_RATE)を制御回転数に乗算(10ms 毎)

blm.h 内

```
//回転数フィードバック
#define BLM_RPM_FEEDBACK_RATE   0.20f          //20%
```

※徐々に制御回転数を目標回転数に近づけていく動作

制御回転数(＝印加磁界を進める速度)は、

目標回転数(例えば、10,000rpm)、回転数(平均)(5,000rpm)の場合、(差が 5%以上のケース)

目標に対しての現状値の割合： rate = 1-(1000/5000) = 0.5
5000 × (rate × 0.2 +1.0) = 5500rpm
(0.2 はフィードバック係数, BLM_RPM_FEEDBACK_RATE)

が、次の制御回転数として設定されます。

目標回転数と制御回転数に大きく乖離がある場合は、一気に制御回転数を変更せず少しずつ目標に近づけていくイメージとなります。

なお、前節で説明している duty は、

目標回転数(例えば、10,000rpm)、回転数(平均)(5,000rpm)の場合、現在の duty=0.3 の場合、(差が 5%以上のケース)

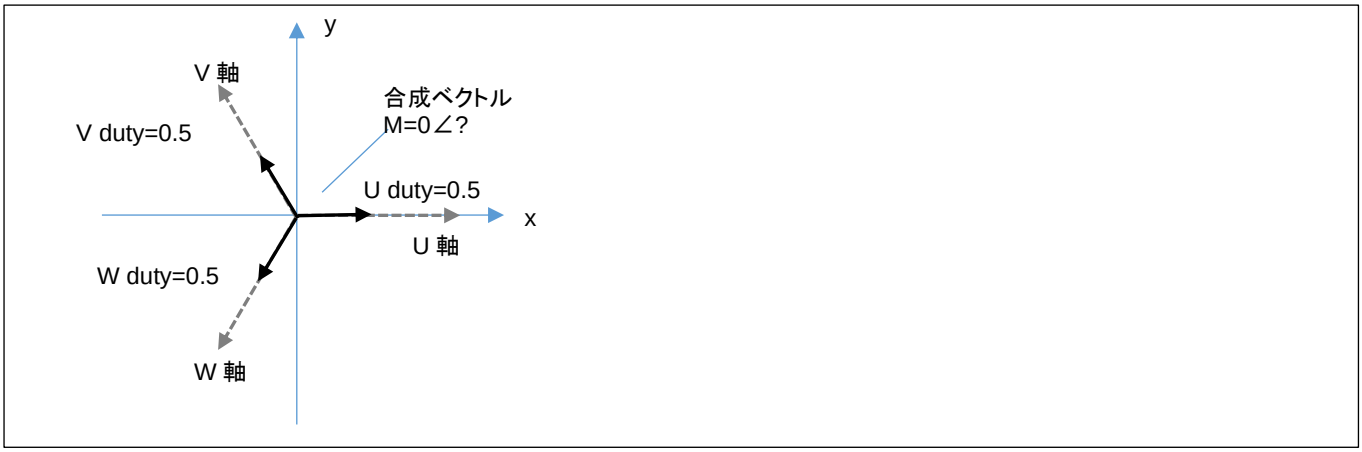
目標に対しての現状値の割合： rate = 1-(1000/5000) = 0.5
0.3 × (rate × 0.01 + 1.0) = 0.3015(30.15%)
(0.01 はフィードバック係数, BLM_DUTY_FEEDBACK_RATE)

が、新しい duty として設定されます。

制御回転数, duty とも、10ms 毎に新しい値が算出され、目標に近づけていく事となります。

1.2.2.4. ブレーキの制御

モータ回転時に、SW3 を ON 側に倒すとブレーキが掛かる様になります。



U,V,W に同じ duty を設定する事でブレーキを掛ける事が出来ます。サンプルプログラムでは、(U, V, W) = (0.5, 0.5, 0.5)としています。

0.5=50%でなくても、UVW 相に同じ duty を設定すればブレーキは掛かりますが、duty が大きいほど強いブレーキが掛かります。（サンプルプログラムでは、各相 50%としています。）

モータ回転時に SW1=OFF にしてモータを停止する場合と、SW3=ON にしてモータを停止する場合で違いが出るはずです。また、SW1=OFF 時（モータに印加する電圧 OFF）と、SW1=ON, SW3=ON 時（モータに電圧を印加）で、軸の空転に掛かる力が異なっています。

モータ回転時に、ブレーキを掛けた場合、瞬間的に電流が流れるため、ブレーキ=ON の場合は、10ms 毎の過電流検出を一時的に無効化する様にしています。

1.3. ホールセンサパターン取得アルゴリズム

モータ内蔵のホールセンサを使用する場合は、TUTORIAL5 で行っている方式、ホールセンサを汎用 I/O ポートに接続し、I/O ポートの 0/1 で読み取ります。

相電圧を使用してホールセンサパターンを生成する（疑似ホールセンサパターンの使用、センサレス駆動の）場合は、TUTORIAL_C, TUTORIAL_C2 で説明している方式です。

blm.c 内の

```
//ホールセンサ
volatile unsigned short g_holl_sensor = BLM_HOLL_MOTOR;     //モータのホールセンサを使用
//volatile unsigned short g_holl_sensor = BLM_HOLL_PSEUDO;    //UVW の相電圧から生成した疑似ホールセンサパターンを使用
```

g_holl_sensor 変数の値により、どちらのセンサを使用するかを選択可能です（変数定義なので、プログラムの実行中に変更する事も可能です。）

疑似ホールセンサパターンを使用する場合は、

blm.h 内で

```
//疑似ホールセンサパターン
#define BLM_HOLL_PSEUDO_SENSOR_AVERAGE              //定義時:電圧の移動平均をホールセンサパターンとする,未定義時:その時の電圧をホールセンサパターンとする
#define BLM_HOLL_PSEUDO_SENSOR_HYS                  //定義時ヒステリシスを有効にする
#define BLM_HOLL_PSEUDO_SENSOR_HYS_VAL          25  //25 = 20mV/3300mV*4096, 20mV 程度ヒステリシスを付ける
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_1   15  //疑似ホールセンサパターン使用時は速度に応じて 15-45°のオフセットを付ける
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_2   25
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_3   35
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_4   45
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_1     2000    //2000[rpm]までは 15°
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_2     6000    //6000[rpm]までは 25°
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_3     8000    //8000[rpm]までは 35°, それ以上は 45°
```

設定があり、サンプルプログラムではデフォルトで

・相電圧は移動平均（BLM_HOLL_PSEUDO_SENSOR_AVERAGE を定義）

・ヒステリシス有効化（BLM_HOLL_PSEUDO_SENSOR_HYS を定義）

としています。

※本サンプルプログラムでは、マイコンの A/D 変換機能による平均化は未使用です、ソフトウェアによる平均化を使用するか使用しないか選択可能としています（デフォルトでは使用する）

また、TUTORIAL_C2 では疑似ホールセンサパターンを使用した際は、磁界印加角度に 30° のオフセットを付けていましたが、サンプルプログラムでは制御回転数に応じてオフセットの量を変えています。

回転数	オフセット量
~2000[rpm]	15°
~6000[rpm]	25°
~8000[rpm]	35°
8000[rpm]~	45°

UVW の相電圧の平均化に関しては、blm.h 内で

(1)相電圧の長期的な平均値の算出

```
//ADC 相電圧長周期移動平均値算出
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_CALC              //定義時長周期の平均を算出する
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_POW2       10     //2**10 = 1024 点の平均を求める
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE            1024   //※変更時は 2 つの定数を同時に変更
(2**BLM_ADC_LONG_TERM_MOVING_AVERAGE_POW2 = BLM_ADC_LONG_TERM_MOVING_AVERAGE)
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_HIST_POW2   3      //2**3=8, 1024 点の平均値の 8 点の移動平均を取り最終的な平均値を求める
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_HIST        8      //※変更時は 2 つの定数を同時に変更
(2**BLM_ADC_AVERAGE_HIST_POW2 = BLM_ADC_AVERAGE_HIST)
```

(2)相電圧の短期的な移動平均の算出

```
//ADC 相電圧短周期移動平均値算出
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_CALC             //定義時短周期の平均を算出する
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_HIST_POW2   3    //2**3=8, 8 点の移動平均を取る
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_HIST        8    //※変更時は 2 つの定数を同時に変更
(2**BLM_ADC_MOVING_AVERAGE_HIST_POW2 = BLM_ADC_MOVING_AVERAGE_HIST)
```

※BLM_HOLL_PSEUDO_SENSOR_AVERAGE 定義時に使用

(1)で求めた長期的（DC 的）な平均と、(2)で求めた短期的な移動平均（ノイズ除去が目的）の比較で、疑似的なホールセンサパターンを生成しています。この辺りは、TUTORIAL_C, TUTORIAL_C2 で説明している話ですので、チュートリアル編のマニュアルを参照してください。

TUTORIAL_C, TUTORIAL_C2 では、回転方向 CCW に対応した疑似ホールセンサパターン生成としていましたが、本サンプルプログラムでは、CCW, CW の両方の回転方向に対応したパタン生成としています。

CCW : V 相値 × 4 + U 相値 × 2 + W 相値 × 1
CW: $\overline{\text{W 相値}}$ × 4 + $\overline{\text{V 相値}}$ × 2 + $\overline{\text{U 相値}}$ × 1

（U 相値は、U 相の電圧が U 相の平均電圧より高い場合は 1，低い場合は 0）

1.4. 安全機構

過電流状態と(モータドライバボードの)過熱状態を検知してモータを停止する機能を持たせています。
g_error_check_flag 変数に、有効化したい定数の値を設定する事で、どの安全機構を有効化するか決められます。

1.4.1. 過電流保護

(1)1 回の過電流検出停止を有効化

g_error_check_flag |= BLM_ERROR_OVER_CURRENT_STOP1;

この場合は割り込みでモータを停止させます。過電流検出端子(*INT)が 1 回でも L(過電流状態)に切り替わった際に、モータは停止となります。

(2)10ms 間に一定回数の過電流検出での停止を有効化

g_error_check_flag |= BLM_ERROR_OVER_CURRENT_STOP2;

50us の割り込みルーチン内で、過電流検出端子(*INT)が L(過電流状態)であった場合、変数(g_over_current_counter1)をインクリメントします。
10ms 毎に、変数の値を定数(BLM_OVER_CURRENT_THRESHOLD_10MS)値と比較し、値が定数を超えていた場合、過電流状態と判断して、モータを停止させます。
g_over_current_counter1 変数は、10ms 毎にクリアされます。
BLM_OVER_CURRENT_THRESHOLD_10MS の初期値は、180 です。50us 毎に 10ms なので、200回チェックした中で、180 回以上なので、10ms 間に 90%以上過電流状態となった場合過電流保護で停止となります。

(3)1s 間に一定回数の過電流検出での停止を有効化

g_error_check_flag |= BLM_ERROR_OVER_CURRENT_STOP3;

50us の割り込みルーチン内で、過電流検出端子(*INT)が L(過電流状態)であった場合、変数(g_over_current_counter2)をインクリメントします。
1s 毎に、変数の値を定数(BLM_OVER_CURRENT_THRESHOLD_1S)値と比較し、値が定数を超えていた場合、過電流状態と判断して、モータを停止させます。
g_over_current_counter2 変数は、1s 毎にクリアされます。
BLM_OVER_CURRENT_THRESHOLD_1S の初期値は、1000 です。50us 毎に 1s なので、20,000回チェックした中で、1,000 回以上なので、1s 間に 5%以上過電流状態となった場合過電流保護で停止となります。

過電流検出は、(1)を有効化した場合、(2)(3)で停止する前に(1)で止まります。(2)(3)の設定は、(1)を無効化した場合有効です。

blm.h 内

```
//過電流停止カウント回数
#define BLM_OVER_CURRENT_COUNT_10MS     180     //50us 毎にチェックを行い 10ms あたり 180 回以上過電流検出で停止（最大 200）
#define BLM_OVER_CURRENT_COUNT_1S       1000    //50us 毎にチェックを行い 1s あたり 1000 回以上過電流検出で停止（最大 20,000）
```

1.4.2. 過熱保護

g_error_check_flag |= BLM_ERROR_OVER_TEMP;

10ms の割り込みルーチン内で、温度をモニタ（A/D 変換及びテーブル比較）し、温度が定数(BLM_UPPER_TEMP)を超えていた場合、過熱状態と判断してモータを停止させます。

BLM_UPPER_TEMP の初期値は 50 です（摂氏 50 度）。

```
//過熱停止[℃]
#define BLM_OVER_TEMP   50
```

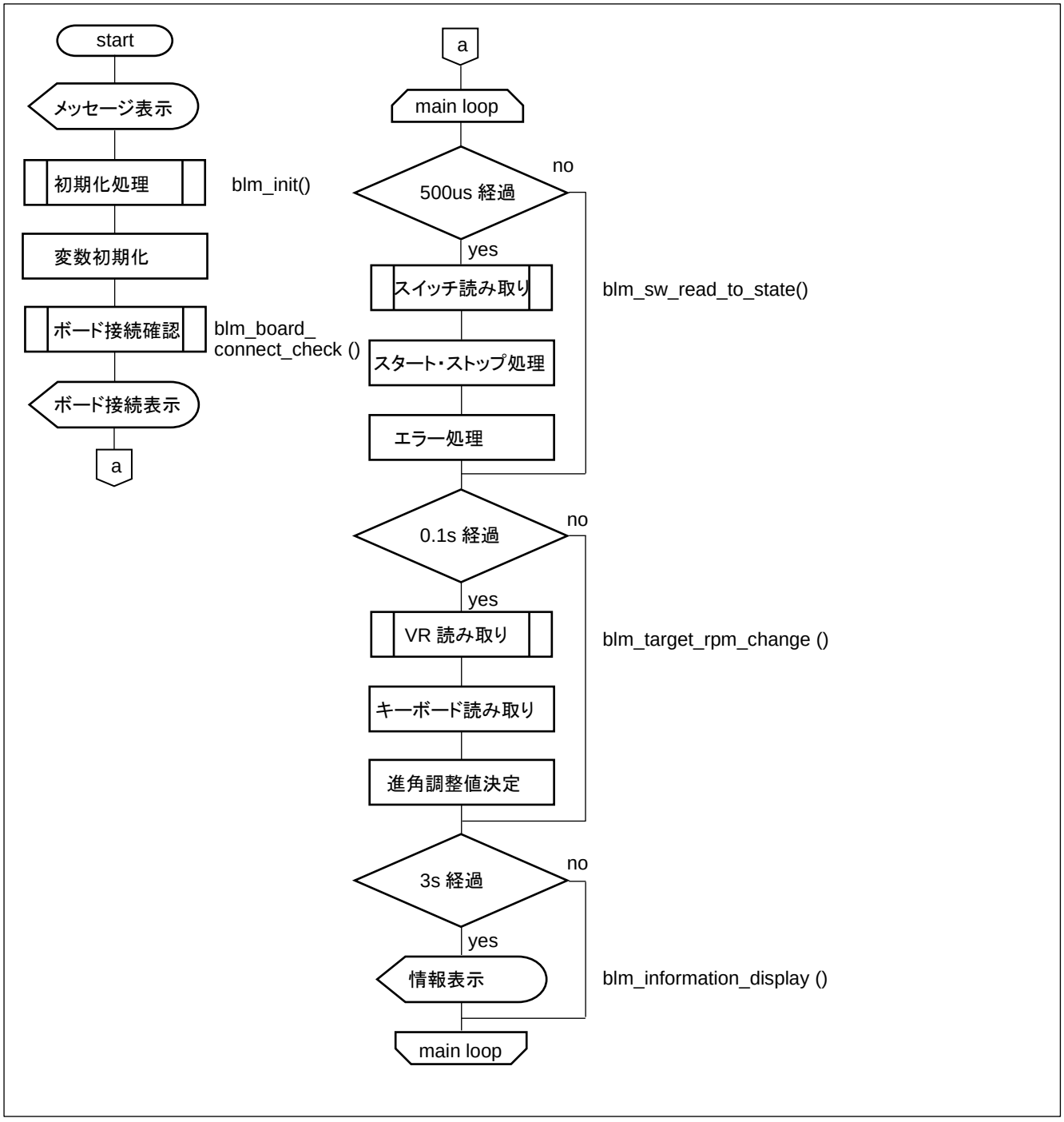
過電流停止と過熱保護のやり方は、TUTORIAL6 で行っている方式と同じです。

1.5. 関数仕様

1.5.1. 全体及び main 関数

main 関数[blm_main.c 内 blm_main()]

ーフローチャートー



メイン関数では、

・初期化
・モータドライバボードの接続確認
・スイッチの読み取り
・モータのスタート・ストップ処理
・エラー表示
・VR（ターゲット回転数）の読み取り
・進角調整（キーボード読み取り）
・画面表示

を行います

メッセージ表示は、今までのチュートリアル同様、シリアル(SCI9)に、115,200bps で出力されます。

モータドライバボードが接続されていないと判断された場合は、SW を ON 側に倒しても反応しません。

1.5.2. main 関数内で実行される関数（blm_main.c に含まれる関数）

blm_sw_to_state

概要：スイッチ読み取り関数
宣言：void blm_sw_to_state(void)
説明：
・スイッチの読み取り
・スイッチの状態をグローバル変数に格納
・動作中の ch の LED の点灯／非動作 ch の LED の消灯
を行います
引数：なし
戻り値：なし
補足：
SW1 が ON 時：　g_state = BLM_CH_STATE_ACTIVE;　　//モータは動作状態
SW1 が OFF 時：　g_state = BLM_CH_STATE_INACTIVE;　//モータは停止状態
とします。（g_state がモータの動作状態を決める変数）
また,SW1 が ON に切り替わった際に、
SW2 が OFF：　g_target_direction = BLM_CCW;　//回転方向は反時計回り(CCW)
SW2 が ON：　g_target_direction = BLM_CW;　　//回転方向は時計回り(CW)
の設定を行います。
SW3 が ON の場合は、g_target_direction = BLM_BREAK;　　//モータはブレーキを掛けた状態
　とします。（モータ回転時にブレーキを掛けた場合は、一時的に電流が流れるので、短期的な過電流検出（10ms 毎の過電流検出）は OFF としています。（SW3 を OFF にした際に、過電流検出の設定を元に戻します。）

blm_target_rpm_change

概要：目標回転数設定関数
宣言：void blm_target_rpm_change(void)
説明：
・VR の読み取り値を目標回転数に変換
を行います。
引数：なし
戻り値：なし
補足：
　VR の角度に応じて、g_target_rpm　を設定します

blm_information_display

概要:情報表示関数
宣言:void blm_information_display(void)
説明:
・シリアル端末への情報表示
を行います。
引数:なし
戻り値:なし
補足:
・モータドライバボードの接続有無　Motor Driver Board
・回転中かどうか　Active
・使用するホールセンサ　holl sensor
・目標回転数　target speed([rpm])
・現在のターゲット回転数　current target speed([rpm])
・進角　forward angle([deg])
・ターゲット回転方向　target direction
・回転数　rotation speed([rpm])
・回転数（平均）　rotation speed[ave]([rpm])
・温度センサー値（A/D 変換データ値）　Temperature(A/D value)
・温度　Temperature(degree)
・VR の A/D 変換データ値　VR(A/D value)
・duty 値　duty[%]
を表示します。

blm_error_display

概要:情報表示関数
宣言:blm_error_display(void)
説明:
・シリアル端末へのエラー情報表示
を行います。
引数:なし
戻り値:なし
補足:
停止理由と過電流の場合は過電流のカウント数、過熱の場合は温度を表示します。

1.5.3. モータ制御関数（blm.c に含まれる関数）

blm_init

概要：モータ初期化関数
宣言：void blm_init(void)
説明：
・変数の初期化
・タイマ、ADC 等のマイコン周辺機能の動作開始
を行います
引数：なし
戻り値：なし

blm_start

概要：モータ動作開始関数
宣言：void blm_start(void)
説明：
・モータの動作開始
を行います
引数：なし
戻り値：なし

blm_stop

概要：モータ停止関数
宣言：void blm_stop(void)
説明：
・モータの動作停止
を行います
引数：なし
戻り値：なし

blm_dutyset

概要： duty 設定関数
宣言：void blm_dutyset(float angle, float duty)
説明：
・合成ベクトルの duty と印加角度を指定することで、相補 PWM の UVW 各相の duty に変換して設定
を行います

引数:
　float angle:　印加角度（ラジアン値で指定）
　float duty:　合成ベクトルの duty(0-1)
戻り値:　なし

blm_angle_to_uvw_duty

概要:UVW 相の duty を求める関数
宣言:void blm_angle_to_uvw_duty(float angle, float duty, blm_uvw *phase_duty)
説明:
・合成ベクトルの duty と印加角度から UVW 相の duty に分解した値の算出
を行います
引数:
　float angle:　印加角度（ラジアン値で指定）
　float duty:　合成ベクトルの印加 duty(0-0.75)
　blm_uvw *phase_duty　各相の印加ベクトル（計算結果,戻り値）（blm_uvw 構造体）
戻り値:　なし
補足:
　テスト用に、
　(1)blm_angle_to_uvw_duty_sin,　デフォルト、正弦波変換
　(2)blm_angle_to_uvw_duty_sin_post,　正弦波変換の duty を後で乗算するパターン
　(3)blm_angle_to_uvw_duty_sin_3harmonic,　3 倍高調波を重畳した正弦波変換
　(4)blm_angle_to_uvw_duty_sin_3harmonic_post,　3 倍高調波を重畳した正弦波変換の duty を後で乗算
　(5)blm_angle_to_uvw_duty1,　どの角度にも 86.6%のパワーとする
　(6)blm_angle_to_uvw_duty2,　角度により最大 100%のパワーとする
　(7)blm_angle_to_uvw_duty2x,　blm_angle_to_uvw_duty2 の直線近似版
の別バージョンを用意（UVW 変換のアルゴリズムの違いを見ることが出来ます）

blm_angle_to_uvw_duty = blm_angle_to_uvw_duty_sin_3harmonic;
（関数ポインタ）=（関数の実体）
の様に、7 種類の関数（実体）のどれを使用するか代入してください。

blm_ideal_angle

概要:ホールセンサ切り替わり時の理想的な印加磁界角度を求める関数
宣言:float blm_ideal_angle(unsigned short pos, short direction, float angle_adjust, bool *err_flag)
説明:
・ホールセンサが切り替わったタイミングでの理想的な印加磁界角度の算出
を行います

引数:

unsigned short pos: ホールセンサ位置(1~6)

short direction: 回転方向(CCW=1, CW=-1)

float angle_adjust: 印加角度補正値（ラジアン値で指定）

bool *err_flag: エラーフラグ

戻り値:

理想的な角度（ラジアン値）

*err_flag ホールセンサが 1~6 の値の場合は false（エラーなし）、それ以外 true（エラーあり）を返す

blm_direction

概要:モータの回転方向を求める関数

宣言:short blm_direction(unsigned char pos, unsigned char prev_pos)

説明:

・ホールセンサ位置（現位置と、1 つ前の位置）からモータ回転方向の算出
を行います

引数:

unsigned char pos: 現在のホールセンサ位置

unsigned char prev_pos: 1 つ前のホールセンサ位置

戻り値:

1: 回転方向は CCW（反時計回り）

-1: 回転方向は CW（時計回り）

-127: 回転方向不明（異常）

blm_pseudo_holl_sensor_pos

概要:疑似ホールセンサパターンの値を得る関数

宣言:unsigned short blm_pseudo_holl_sensor_pos(short target_direction)

説明:

・UVW の相電圧の値から、現在のホールセンサ位置の推定
を行います

引数:

short target_direction: 設定している回転方向

戻り値:

1~6: モータ内蔵のホールセンサと同じ位置情報

0, 7: ホールセンサパターンの取得に失敗

deg2rad

概要:角度変換関数
宣言:float deg2rad(short deg)
説明:
・度単位からラジアン単位への変換
を行います
引数:
　short deg:　角度(°)
戻り値:
　角度(ラジアン)

rad2reg

概要:角度変換関数
宣言:short rad2reg(float rad)

説明:
・ラジアン単位から度単位への変換
を行います
引数:
　float rad:　角度(ラジアン)
戻り値:
　角度(°)
補足:
　引数は変換結果が short で表現できる範囲の値である必要があります(引数の-π~πの範囲へのマッピングは行いません)

1.5.4. その他の関数（blm_common.c に含まれる関数）

blm_board_connect_check

概要：モータドライバボード接続確認関数
宣言：unsigned short blm_connect_check(void)
説明：
・モータドライバボードの接続状態の確認
を行います
引数：なし
戻り値：
　0 :モータドライバボード非接続,
　1 :モータドライバボード接続
補足：
　BLM_ACTIVE_CH　定数値により、CH-1, CH-2 のどちらのボード接続結果を返すか変わります。

blm_holl_sensor_pos

概要：ホールセンサ位置関数
宣言：unsigned short blm_holl_sensor_pos(void)
説明：
・ホールセンサ位置の算出
を行います
引数：なし
戻り値：
　HS3×4+HS2×2+HS1 で算出される 1~6 のホールセンサ値（7 の場合はモータドライバボード未接続）
補足：
　BLM_ACTIVE_CH　定数値により、CH-1, CH-2 のどちらのホールセンサ値を返すか変わります。

blm_current_monitor

概要：過電流検出関数
宣言：bool blm_current_monitor(void)
説明：
・過電流検出
を行います
引数：なし
戻り値：
　true(=1)　　過電流検出なし
　false(=0)　　過電流状態

blm_board_temp

概要:温度算出関数
宣言:short blm_board_temp(unsigned short adc_val)
説明:
・温度センサ A/D 変換値から温度値の算出
を行います
引数:
　unsigned short adc_val:　　温度センサ A/D 変換値
戻り値:
　温度値（摂氏温度、1 度単位）

blm_sw_read

概要:スイッチ読み取り関数
宣言:unsigned short blm_sw_read(void)
説明:
・変換ボード上のスイッチ(SW1~SW3)の読み取り
を行います
引数:なし
戻り値:
　bit0:　SW1=ON の時 1, OFF の時 0
　bit1:　SW2=ON の時 1, OFF の時 0
　bit2:　SW3=ON の時 1, OFF の時 0

blsm_led_out
blsm_led_out2

概要:LED 設定関数
宣言:void blm_led_out(unsigned short pattern)
宣言:void blm_led_out2(unsigned short pattern)
説明:
・変換ボード上の LED の点灯、消灯制御(blm_led_out)
・マイコンボード上の LED の点灯、消灯制御(blm_led_out2)
を行います
引数:(blm_led_out)
　unsigned short pattern:
　　bit0　1:LED1 点灯，0:LED1 消灯
　　bit1　1:LED2 点灯，0:LED2 消灯
　　bit2　1:LED3 点灯，0:LED3 消灯

bit3　1:LED4 点灯，0:LED4 消灯

引数：(blm_led_out2)

unsigned short pattern:

bit0　1:LED1 点灯，0:LED1 消灯

戻り値：なし

blm_led_change

概要：LED 設定関数

宣言：void blm_led_change(unsigned short on_pattern, unsigned short off_pattern)

説明：

・（変換ボード上の）LED の点灯、消灯の切り替え

を行います

引数：

unsigned short on_pattern:

bit0　1:LED1 点灯

…

bit3　1:LED4 点灯

unsigned short off_pattern:

bit0　1:LED1 消灯

…

bit3　1:LED4 消灯

戻り値：なし

補足:

on_pattern に 1 を立てた LED は点灯に切り替わります。off_pattern に 1 を立てた LED は消灯に切り替わります。

blm_led_change(0x1, 0x8);

→LED1 が点灯となり、LED4 は消灯となります

1.5.5. 割り込み関数

割り込み関数は、blm_intr.c 内で定義され、タイマや ADC 処理等を行っています。

agt0_callback

概要：AGT0 タイマ割り込み（50us 周期）
宣言：void agt0_callback (timer_callback_args_t * p_args)
説明：
・A/D 変換の実行
・モータ回転制御
・過電流モニタ
・回転速度の算出
・磁界印加角度の補正
・duty 補正値の算出
を行います
補足：
モータ制御の基本周期割り込みです。本サンプルプログラムでは、モータ制御に関わるリアルタイムで処理を本関数内で行っています。

agt1_callback

概要：AGT1 タイマ割り込み（10ms 周期）
宣言：void agt1_callback (timer_callback_args_t * p_args)
説明：
・過電流停止処理
・過熱停止処理
・動作フェースの遷移
・duty 補正（フィードバック）
・回転数の計算
を行います
補足：
モータ制御の定期処理の周期割り込みです。本サンプルプログラムでは、モータ制御に関わる定期的に実行したい処理を本関数内で行っています。

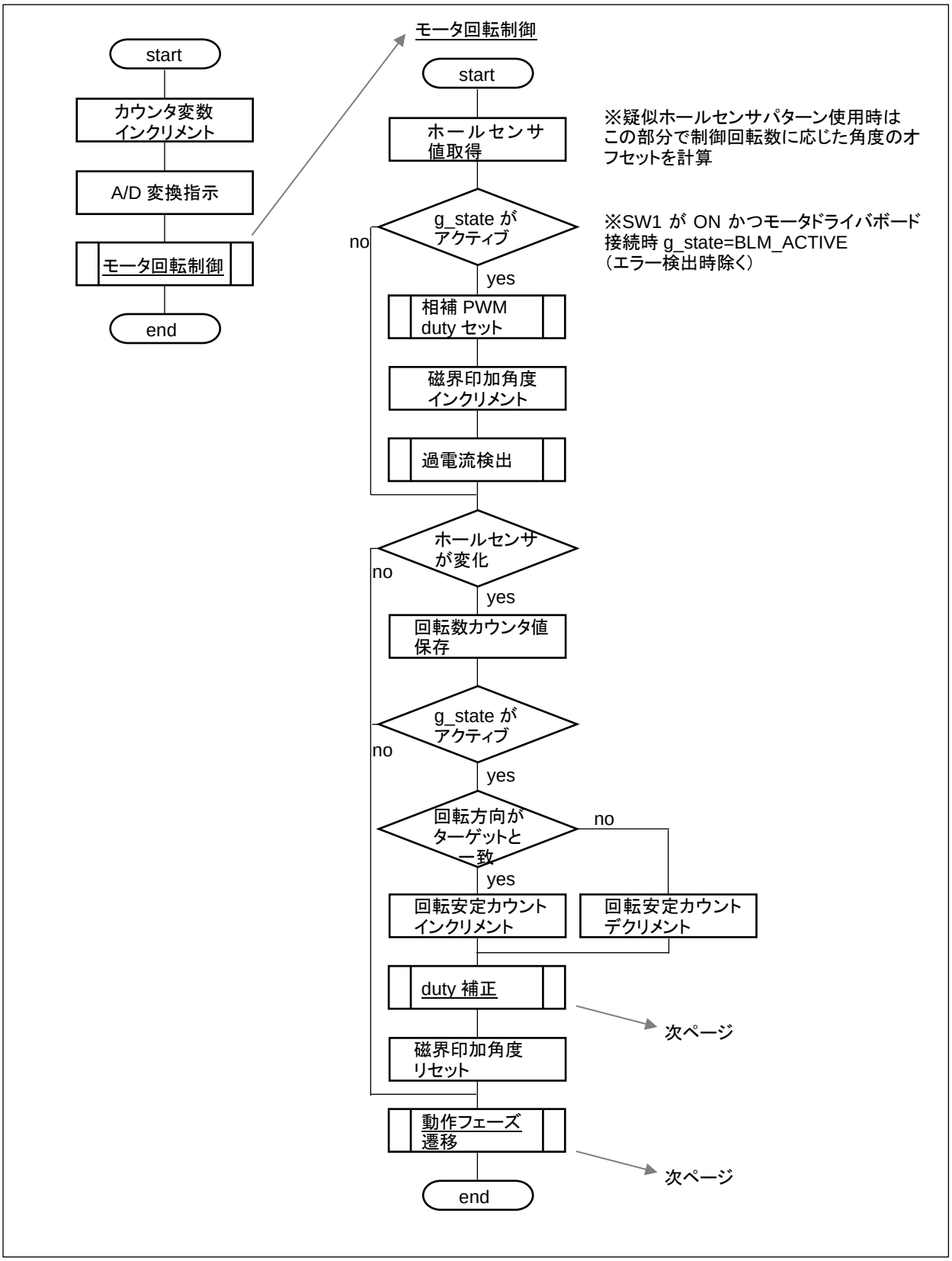
adc_callback

概要：A/D 変換完了割り込み
宣言：void adc_callback (adc_callback_args_t * p_args)
説明：
・A/D 変換結果の回収
を行います
補足：
　AGT0（50us 周期）で、A/D 変換開始指示を出しており、A/D 変換終了時、本割り込み関数にて、A/D 変換結果をグローバル変数に保存しています。
　平均値の算出等も本関数内で行われます。

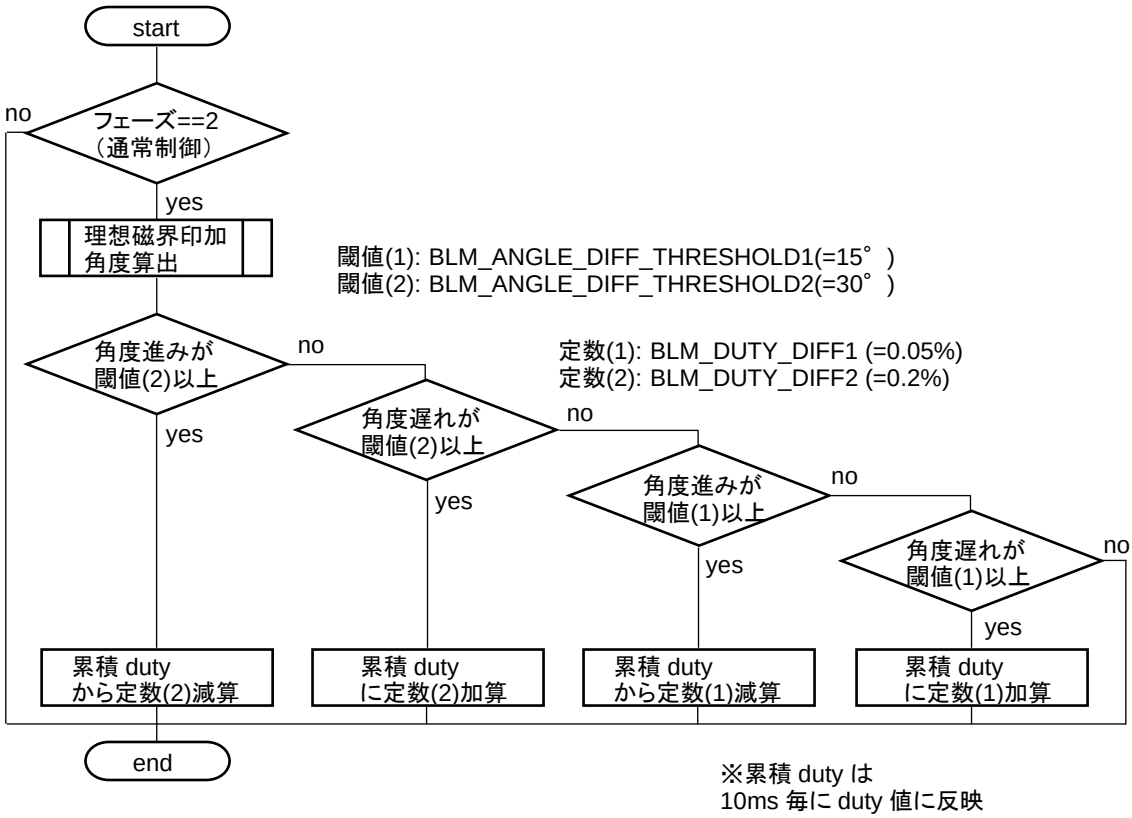
irq1_callback

概要：端子割り込み
宣言：void irq1_callback (external_irq_callback_args_t * p_args)
説明：
・*INT（過電流信号）の検出
を行います
補足：
　1 回の過電流信号でモータを停止させる設定の場合有効な割り込み
（デフォルトでは、10ms 毎, 1 秒毎の過電流停止を使用しているため、本関数は使用されない）

－agt0_callback（50us 毎の割り込み）全体フローチャート－

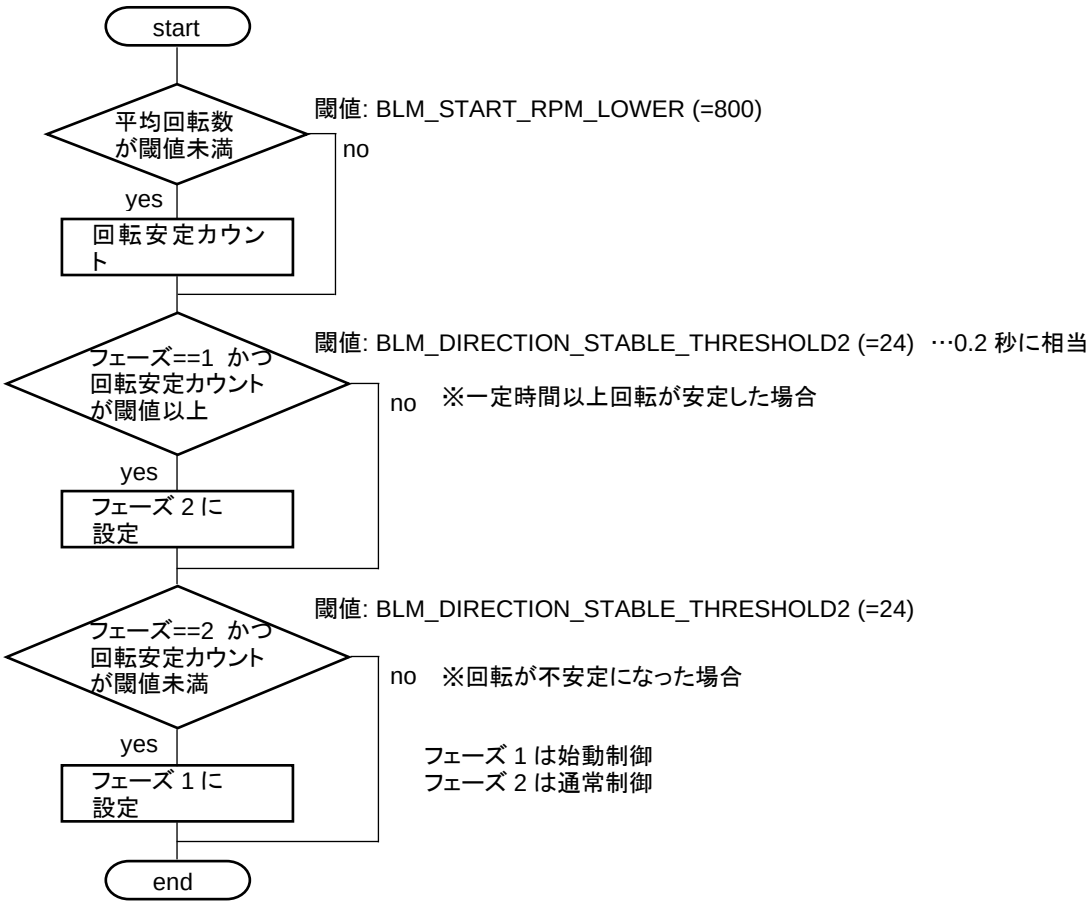


duty 補正



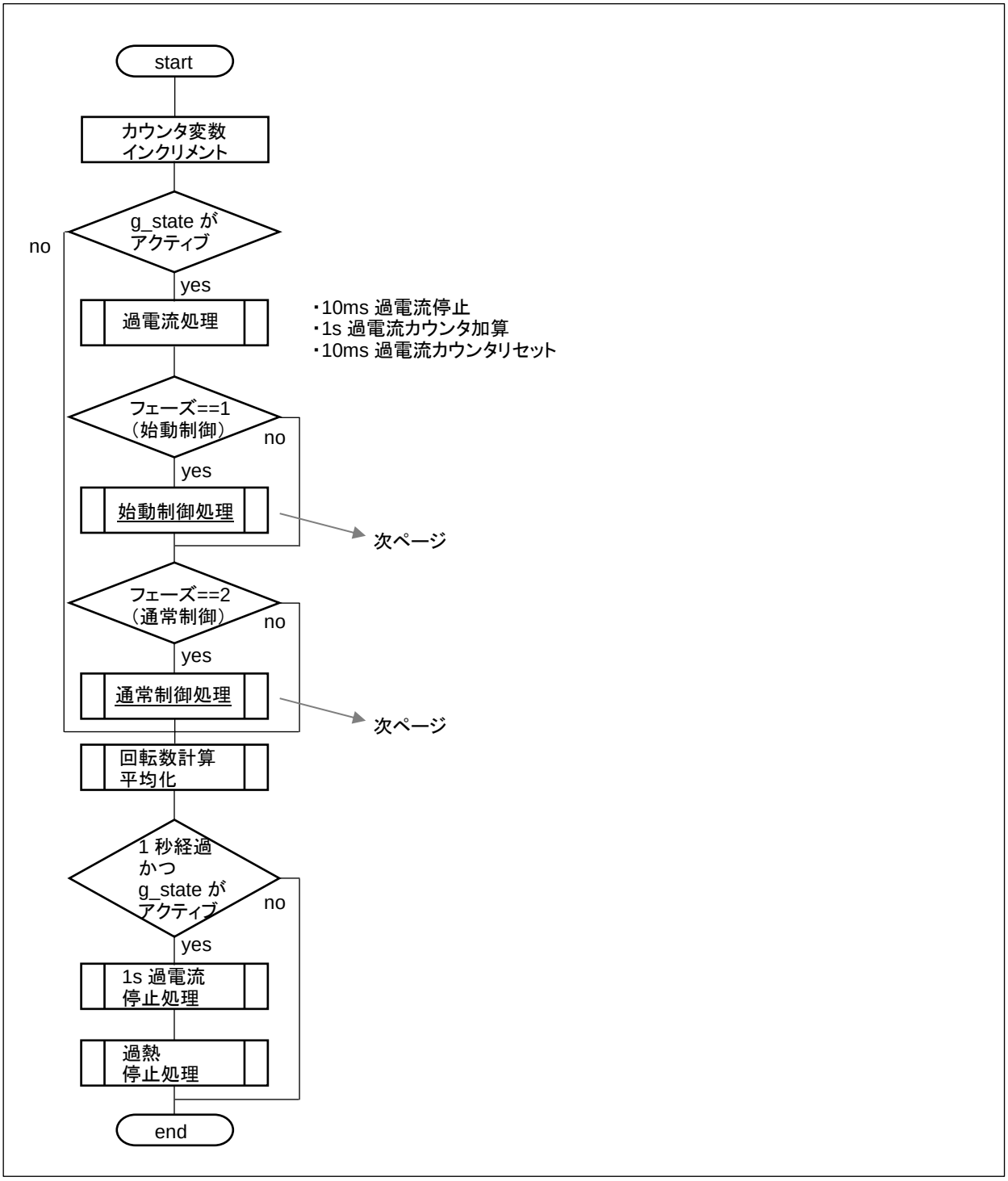
※累積 duty は
10ms 毎に duty 値に反映

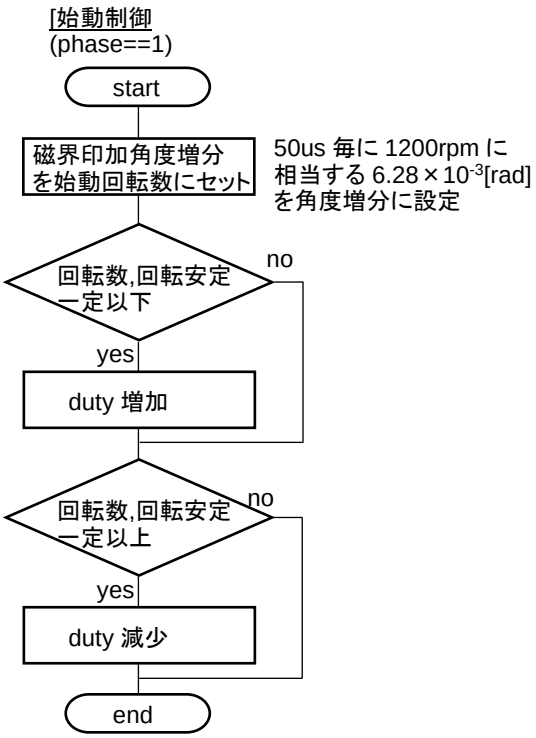
動作フェーズ遷移



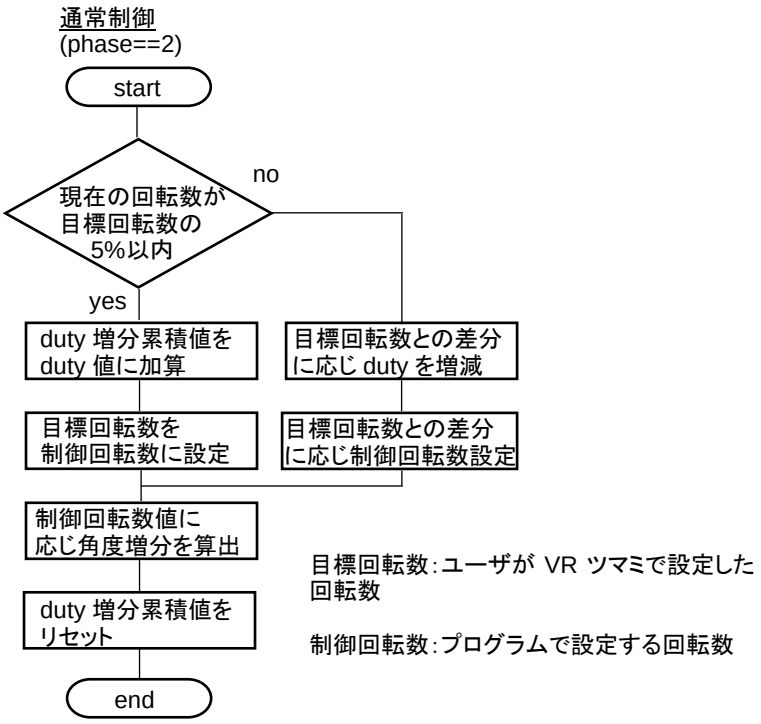
フェーズ 1 は始動制御
フェーズ 2 は通常制御

－agt1_callback（10ms 毎の割り込み）全体フローチャート－





※磁界を進める角度（回転数）は
BLM_START_RPM(=1200rpm)
固定



目標回転数：ユーザが VR ツマミで設定した回転数

制御回転数：プログラムで設定する回転数

目標回転数 5%以内の時は、ホールセンサ切り替わり毎の duty 増分累積値を適用
5%以上の時は目標回転数との差分に応じた duty を計算して適用

1.6. グローバル変数

g_adc_scan_flag

概要:A/D 変換中を示す変数
宣言:volatile unsigned short g_adc_scan_flag
説明:
A/D 変換開始時、1 に設定され、A/D 変換終了の割り込みルーチンの最後に、0(フラグクリア)とされる変数。本変数が 1 の時には、A/D 変換実行中であるので、A/D 変換の開始をスキップする。

blm_adc g_adc_result

概要:A/D 変換結果を格納する変数
宣言:volatile blm_adc g_adc_result
説明:
g_adc_result には、A/D 変換結果が格納される(50us 毎に更新)
補足:
blm_adc は、構造体で、以下のメンバを含む。

```
typedef struct{
    unsigned short volume;          //VRの値
    unsigned short v_u_phase;    //U相の電圧
    unsigned short v_v_phase;    //V相の電圧
    unsigned short v_w_phase;    //W相の電圧
    unsigned short i_u_phase;    //U相の電流
    unsigned short i_v_phase;    //V相の電圧
    unsigned short i_w_phase;    //W相の電圧
    unsigned short temp;            //温度センサの値
    unsigned short v_power;      //電源電圧
} blm_adc;
```

U 相の電圧の A/D 変換値は、g_adc_result.v_u_phase。

g_adc_result_buf

概要:A/D 変換結果の履歴を格納する変数
宣言:blm_adc g_adc_result_buf[BLM_ADC_HIST]　　// BLM_ADC_HIST =400
説明:
g_adc_result_buf[ ]には、A/D 変換結果が(直前の 400 ポイントが)格納される(50us 毎に更新)
リングバッファ状にデータが格納。最新のデータは、インデックスが g_adc_result_buf_index-1 のデータ。

g_adc_result_buf_index

概要:A/D 変換結果の履歴を格納する変数のインデックス
宣言:unsigned short g_adc_result_buf_index

g_adc_v_phase_long_term_average
g_adc_v_phase_short_term_average

概要:相電圧の平均値を格納する変数
宣言:blm_uvw_uint16 g_adc_v_phase_long_term_average
　　blm_uvw_uint16 g_adc_v_phase_short_term_average
説明:
　g_adc_v_phase_long_term_average　は、相電圧の長期間の平均（1024　点の平均値の、8　点の移動平均。約 400ms の平均）。
　g_adc_v_phase_short_term_average は、相電圧の短期間の平均、直近 8 点の移動平均。
※下線部の値は定数値（変更可能）
補足:

```
typedef struct{
    unsigned short u;          //U 相
    unsigned short v;          //V 相
    unsigned short w;          //W 相
} blm_uvw_uint16;
```

　g_adc_v_phase_long_term_average.u　が、U 相の平均電圧。

g_state

概要:チャネル状態変数
宣言:volatile unsigned short g_state
説明:
　現在の CH 毎の状態を示す変数
　　BLM_CH_STARE_INACTIVE(0):　停止
　　BLM_CH_STARE_ACTIVE(1):　回転

g_board_connect_info

概要:モータドライバボード接続状態変数
宣言:volatile unsigned short g_board_connect_info
説明:
　CH 毎のモータドライバボード接続状態を示す変数

BLM_NO_CONNECT(0): モータドライバボード未接続
BLM_CONNECT(1): モータドライバボード接続

g_error

概要:エラー状態変数
宣言:volatile blm_error g_error
説明:
g_error.status
BLM_NO_ERROR(0): エラーなし
BLM_ERROR_OVER_TEMP_STOP(0x1): 過熱停止
BLM_ERROR_OVER_CURRENT_STOP1(0x2): 過電流停止 1(1 回の割り込みで停止)
BLM_ERROR_OVER_CURRENT_STOP2(0x4): 過電流停止 2(10ms 間に規定回数オーバ)
BLM_ERROR_OVER_CURRENT_STOP3(0x8): 過電流停止 3(1s 間に規定回数オーバ)
g_error.temp 過熱停止時の温度
g_error.over_current_count_1 10ms 間の過電流カウント数
g_error.over_current_count_1 1s 間の過電流カウント数
補足:
blm_error は、構造体で、以下のメンバを含む。

```
typedef struct{
    unsigned short status;
    short temp;                                 //温度
    unsigned long over_current_count_1;         //10msの過電流検出回数
    unsigned long over_current_count_2;         //1sの過電流検出回数
} blm_error;
```

g_error_check_flag

概要:エラーチェック項目設定変数
宣言:volatile unsigned short g_error_check_flag
説明:
エラーチェック対象を指定
補足:
g_error_check_flag = BLM_ERROR_OVER_TEMP_STOP | BLM_ERROR_OVER_CURRENT_STOP3;
の場合、過熱停止と、1 秒毎の過電流停止を有効化。
BLM_ERROR_OVER_TEMP_STOP(0x1) 過熱停止
BLM_ERROR_OVER_CURRENT_STOP1(0x2) 1 回の過電流検出で停止
BLM_ERROR_OVER_CURRENT_STOP2(0x4) 10ms 毎一定回数の過電流検出で停止
BLM_ERROR_OVER_CURRENT_STOP3(0x8) 1s 毎一定回数の過電流検出で停止
を OR で指定。

g_sensor_pos

概要:ホールセンサ位置変数
宣言:volatile unsigned short g_sensor_pos
説明:
50us 毎に読み取ったホールセンサの情報を 1~6 の数値に変換した値を格納

g_rotation_counter

概要:回転数計カウンタ変数
宣言:volatile unsigned long g_rotation_counter
説明:
50us 毎にインクリメント、ホールセンサの値が変化した際にリセットされる変数

g_rpm

概要:回転数計算変数
宣言:volatile unsigned short g_rpm
説明:
現在の回転数(ホールセンサが変化した際に計算され更新)

g_rpm_ave

概要:回転数計算変数
宣言:volatile unsigned short g_rpm_ave
説明:
現在の平均回転数(10ms 毎の回転数の過去 16 値の平均値)
過去の値を保存するのに、g_rpm_hist[16], g_rpm_hist_index を使用。

g_over_current_counter_1
g_over_current_counter_2

概要:過電流検出回数を示す変数
宣言:volatile unsigned long g_over_current_counter_1
volatile unsigned long g_over_current_counter_2
説明:
g_over_current_counter_1: 10ms 間に 50us 毎に過電流チェックを行い過電流検出された回数(10ms でリセット)
g_over_current_counter_2: 1s 間に 50us 毎に過電流チェックを行い過電流検出された回数(1s でリセット)

g_target_direction

概要:回転方向の設定変数
宣言:volatile short g_target_direction
説明:
BLM_CCW(1): 反時計回り(Counter Clock Wise)
BLM_CW(-1): 時計回り(Clock Wise)
BLM_BREAK(2): ブレーキ
BLM_STOP(0): 停止

g_direction

概要:現在の回転方向を示す変数
宣言:volatile short g_direction
説明:
BLM_CCW(1): 反時計回り(Counter Clock Wise)
BLM_CW(-1): 時計回り(Clock Wise)
BLM_BREAK(2): ブレーキ
BLM_STOP(0): 停止
BLM_UNKNOWN(-127): センサの切り替わりが異常

g_stable

概要:回転方向の安定を示す変数
宣言:volatile unsigned short g_stable
説明:
ホールセンサ切り替わり時、切り替わり前の値との比較で回転方向(g_direction)を判断します。回転方向が、設定した回転方向(g_target_direction)と一致しているとき、回転は安定していると判断して、g_stable をインクリメントします。不一致の時は、g_stable をデクリメントします。

g_target_rpm

概要:設定回転数を示す変数
宣言:volatile unsigned short g_target_rpm
説明:
ユーザが VR ツマミで設定した回転数を格納する変数。単位は、rpm 値で格納されます。この回転数に近づくように回転数を制御します。

g_current_target_rpm

概要:現在の制御回転数を示す変数
宣言:volatile unsigned short g_current_target_rpm
説明:
現在の制御回転数を格納する変数で、50us 毎に磁界印加角度を進める際の角度は、この変数から算出されます。g_current_target_rpm（制御回転数）と g_target_rpm（目標回転数）の差分が小さいときには、g_current_target_rpm = g_target_rpm となる様に設定し、差分が大きいときは一定のレート(BLM_RPM_FEEDBACK_RATE)で、g_current_target_rpm を g_target_rpm に近づけていきます。

g_phase

概要:現在の動作フェーズを示す変数
宣言:volatile unsigned short g_phase
説明:
BLM_PHASE_0(0): 停止
BLM_PHASE_1(1): 始動制御
BLM_PHASE_2(2): 通常制御
BLM_PHASE_3(3): ブレーキ

g_duty

概要:duty 比設定変数
宣言:float g_duty
説明:
duty 比を設定する変数。0-1。
本変数値を適切な値（目標回転数に合った値）となるように制御を行う。

g_phase1_diff_array

概要:始動時の duty を変化させるテーブル変数（CH 毎）
宣言:const float g_phase1_diff_array[8]
説明:
{0.002f, 0.002f, 0.00234375f, 0.0046875f, 0.009375f, 0.01875f, 0.0375f, 0.075f};
0.20% 0.2% 0.23475% 0.46875% 0.9375% 1.875% 3.75% 7.5%
始動制御時の立ち上がりを速くするためのテーブル。始動制御時は、本テーブルに書かれている値で duty の立ち上がりのカーブを決定する。
g_phase1_diff_index は、本変数のインデックスで使用される。

g_duty_diff_integral

概要:duty 比設定変数
宣言:volatile float g_duty_diff_integral
説明:
通常制御時、duty の累積差分値を格納する変数。ホールセンサ切り替わりのポイントで、回転数が速い(duty を減少させる)、回転数が遅い(duty を増加させる)に応じた duty の変更値を本変数に格納し、10ms のタイミングで g_duty に本変数の値を加算する。

g_angle

概要:印加角度設定変数
宣言:volatile float g_angle
説明:
印加角度を設定する変数。ラジアン単位。50us 毎に、本変数の値と duty 比の値(g_duty)を UVW 分解して、実際の PWM 波形に反映させる。

g_angle_diff

概要:印加角度差分設定変数
宣言:volatile float g_angle_diff
説明:
50us 毎に本変数を、g_angle に加算(CCW 時)、減算(CW 時)する。50us 毎の磁界印加角度の増分値。g_current_target_rpm から算出される。

g_holl_sensor_count

概要:印加角度差分設定変数
宣言:volatile float g_angle_diff
説明:
50us 毎に本変数を、g_angle に加算(CCW 時)、減算(CW 時)する。50us 毎の磁界印加角度の増分値。g_current_target_rpm から算出される。

g_angle_forward

概要:進角調整設定変数
宣言:volatile float g_angle_forward

説明:
　ホールセンサ切り替わり時の理想印加角度の調整を行う変数です。初期値 0。ラジアン単位。
　正の値の時は、進み方向（速めに磁界印加角度をスイッチング）、負の値は遅れ方向です。
（キーボードから進角調整を行うと、度単位での調整となりますが、プログラム内ではラジアンに変換されて、進角値が調整されます。）

g_timer_half_count

概要:タイマ周期の半分を示す変数
宣言:unsigned long g_timer_half_count
説明:
　相補 PWM を構成するタイマーの谷から山まで（山から谷まで）のカウント値を保持する変数。PWM キャリア周波数を途中で変更しない限りは、決め打ちの値でも良いが、変数としている。

g_timer_half_count

概要:PWM 周期の半分のカウント数
宣言:unsigned long g_timer_half_count
説明:
　PWM 周期の谷から山までのカウント数。duty 値を PWM 周期に変換する際に使用。

g_holl_sensor

概要:ホールセンサ区分変数
宣言:volatile unsigned short g_holl_sensor
説明:
　モータ内蔵のホールセンサを使用するか、相電圧から計算される疑似ホールセンサパターンを使うかを選択する変数。
補足:
　BLM_HOLL_MOTOR(1):　モータ内蔵のホールセンサを使用
　BLM_HOLL_PSEUDO(2):　疑似ホールセンサパターンを使用

1.7. プログラムの動作を制御する定義値

```
#define PI 3.14159265358979f
#define PI2 (PI*2.0f)
#define SQRT3_DIV2 0.866025403784f
#define SQRT3 1.73205080757f
#define N2_DIV_SQRT3 1.15470053838f
```

円周率(π)の値と、2πの値。√3 /2, √3, 2/√3 の値。

```
#define U_V_DIRECTION 1
#define U_W_DIRECTION 2
#define V_U_DIRECTION 3
#define V_W_DIRECTION 4
#define W_U_DIRECTION 5
#define W_V_DIRECTION 6
#define OFF_DIRECTION 0
```

モータ制御の電流方向を定義する定数ですが、相補 PWM 制御の場合未使用です。

```
#define BLM_CH_NUM    1
#define BLM_CH1       0
#define BLM_CH2       1
#define BLM_ACTIVE_CH BLM_CH1    //使用する CH を選択
```

ch 数（本キットでは、同時に使用可能なのは 1ch）と、ch 番号の定数。どちらの ch を使用するか。

```
#define BLM_CH_STATE_INACTIVE    0    //スイッチ OFF
#define BLM_CH_STATE_ACTIVE      1    //スイッチ ON
```

スイッチ(SW1)の ON/OFF（モータが回転制御状態かどうか）を示す定数。

```
#define BLM_PHASE_0 0    //停止
#define BLM_PHASE_1 1    //始動制御
#define BLM_PHASE_2 2    //通常制御
#define BLM_PHASE_3 3    //ブレーキ
```

動作フェーズを示す定数。

```
#define BLM_NO_ERROR                          0    //エラーなし
#define BLM_ERROR_OVER_TEMP_STOP              0x0001    //過熱停止
#define BLM_ERROR_OVER_CURRENT_STOP1          0x0002    (*1)
#define BLM_ERROR_OVER_CURRENT_STOP2          0x0004    (*2)
#define BLM_ERROR_OVER_CURRENT_STOP3          0x0008    (*3)
```

(*1)　//割り込みを使い 1 回でも過電流が観測されると停止
(*2)　//10ms の間 BLM_OVER_CURRENT_COUNT_10MS 回数を超えたら停止
(*3)　//1s の間 BLM_OVER_CURRENT_COUNT_1S 回数を超えたら停止

エラーステータスを示す定数。

```
#define BLM_NO_CONNECT 0    //モータドライバボード未接続
#define BLM_CONNECT    1    //モータドライバボード接続
```

モータドライバボードの接続状況を示す定数。

```
#define  BLM_ADC_HIST       400          //400 ポイント分を保存(2bytesx9x3ch=21kB)...RA6T3/4T1 は RAM40kB
```

A/D 変換の履歴を保存する数。

```
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_CALC
```

本定数定義時は、A/D 変換結果の回収（A/D 変換割り込みルーチン）で、長周期の平均値を計算する。l

```
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_POW2          10
    //2**10 = 1024 点の平均を求める
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE               1024
    //※変更時は 2 つの定数を同時に変更
```

長周期の平均値算出の際の平均値は、1024 点の平均値を取る。
BLM_ADC_LONG_TERM_MOVING_AVERAGE_POW2 は 2 の何乗か、
BLM_ADC_LONG_TERM_MOVING_AVERAGE は何点かを示す定数。

変更する場合は、2 つの定数（2^10=1024)の整合性が取れる様に変更してください。

```
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_HIST_POW2     3
    //2**3=8, 1024 点の平均値の 8 点の移動平均を取り最終的な平均値を求める
#define BLM_ADC_LONG_TERM_MOVING_AVERAGE_HIST          8
    //※変更時は 2 つの定数を同時に変更
```

長周期の平均値は、1024 点の平均値をさらに 8 点の移動平均を求める計算を行っており、移動平均のポイント数の定義です。変更する場合は、2 つの定数（2^3=8)の整合性が取れる様に変更してください。

```
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_CALC
```

本定数定義時は、A/D 変換結果の回収（A/D 変換割り込みルーチン）で、短周期の平均値を計算する。

```
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_HIST_POW2     3
    //2**3=8, 8 点の移動平均を取る
#define BLM_ADC_SHORT_TERM_MOVING_AVERAGE_HIST          8
    //※変更時は 2 つの定数を同時に変更
```

短周期の平均値は、8 点の移動平均を求める計算を行っており、移動平均のポイント数の定義です。変更する場合は、2 つの定数（2^3=8)の整合性が取れる様に変更してください。

```
#define BLM_OVER_CURRENT_COUNT_10MS        180
     //50us 毎にチェックを行い 10ms あたり 100 回以上過電流検出で停止（最大 200）
#define BLM_OVER_CURRENT_COUNT_1S          1000
    //50us 毎にチェックを行い 1s あたり 1000 回以上過電流検出で停止（最大 20,000）
```

過電流検出の閾値。

BLM_ERROR_OVER_CURRENT_STOP2 有効時 BLM_OVER_CURRENT_COUNT_10MS の値が使用され、BLM_ERROR_OVER_CURRENT_STOP3 有効時 BLM_OVER_CURRENT_COUNT_1S の値が使用されます。

50us 間隔で 10ms 間に 200 回の過電流チェックが行われるが、BLM_OVER_CURRENT_COUNT_10MS を 180 にした場合、10ms 間に 90%(180/200)以上の過電流が検出された場合に停止となります。

```
#define BLM_OVER_TEMP     50
```

過熱停止の閾値。定義値 50 の場合は、モータドライバボード上の温度センサ（サーミスタ）が 50℃を超えた時に、モータは停止となります。

```
#define BLM_DUTY_MAX                    1.0f
#define BLM_DUTY_MIN                    0.0f
```

duty 比の設定値の最大、最小。1.0f と 0.0f の場合は 0~100%の設定となり、duty を最大値まで設定する状態。

```
#define BLM_DUTY_DIFF1                  0.0005f
      //0.05%, ANGLE_DIFF_THRESHOLD1  のずれを検出した際の増分
#define BLM_DUTY_DIFF2                  0.0002f
      //0.2%, ANGLE_DIFF_THRESHOLD2  のずれを検出した際の増分
```

通常制御において、ホールセンサ切り替わり時、15°(ANGLE_DIFF_THRESHOLD1)以上のずれがあった場合は、duty に 0.05%(BLM_DUTY_DIFF1)を加減算。30°(ANGLE_DIFF_THRESHOLD2)以上のずれがあった場合は、duty に 0.2%(BLM_DUTY_DIFF2)を加減算。

```
#define BLM_DUTY_FEEDBACK_RATE   0.01f     //1%
```

通常制御において、回転数が目標回転数から大きく離れている（5%以上）場合、duty を 1%(BLM_DUTY_FEEDBACK_RATE)ずつ近づけていく。

```
#define BLM_CCW       1    //反時計回り
#define BLM_CW       -1    //時計回り
#define BLM_STOP      0    //停止
#define BLM_BREAK     2    //ブレーキ
#define BLM_UNKNOWN -127    //不明（異常）
```

モータの回転方向を定義する定数。

```
#define BLM_MAX_RPM       12000    //上限
#define BLM_MIN_RPM       1000     //下限
```

最高、最低回転数。VR ツマミを回した際の目標回転数の上限と下限。

```
#define BLM_START_RPM                   1200    //始動回転数
#define BLM_START_RPM_LOWER             800     //始動制御時の最低回転数
#define BLM_START_RPM_UPPER             1500    //始動制御時の最高回転数
```

始動制御（フェーズ=1）時の固定回転数 1200rpm(BLM_START_RPM)。及び、始動制御における、この値未満は回転が安定していないとみなす最低回転数 800rpm(BLM_START_RPM_LOWER)。始動制御で、この回転数を越えた場合 duty を減らす最高回転数 1500rpm(BLM_START_RPM_UPPER)の設定値。

```
#define BLM_DIRECTION_STABLE_THRESHOLD1 6          //1 回転分
#define BLM_DIRECTION_STABLE_THRESHOLD2 24
        //1200rpm の場合 0.2s 以上回転安定時 phase2 に移行
#define BLM_DIRECTION_STABLE_THRESHOLD3 30
        //回転安定検出の最大値
```

始動制御時、回転安定が 6(BLM_DIRECTION_STABLE_THRESHOLD1)に達していない場合、duty を増やします。同じく、回転安定が 24(BLM_DIRECTION_STABLE_THRESHOLD2)を超えた場合、通常制御（フェーズ 2）に移行します。回転安定の最大値が、BLM_DIRECTION_STABLE_THRESHOLD3=30 です。回転安定が 30 を超えている場合、回転安定カウントのインクリメントは行いません。

```
#define BLM_RPM_FEEDBACK_RATE     0.20f                //20%
```

回転数のフィードバック係数。目標回転数と制御回転数が乖離している場合、この定数の割合で制御回転数を目標回転数に近づけていきます。

```
#define BLM_FORWARD_ANGLE_MAX     45
#define BLM_FORWARD_ANGLE_MIN     -45
```

進角調整範囲。単位° 。-45~45° の範囲で進角調整が有効です。

```
#define RAD_0_DEGREE (0.0f)
#define RAD_15_DEGREE (15.0f/180.0f*PI)
…
#define RAD_345_DEGREE (345.0f/180.0f*PI)
```

角度（° 単位）をラジアン変換した定数です。

```
#define BLM_ANGLE_DIFF_THRESHOLD1     RAD_15_DEGREE
    //15 度以上ずれている場合は、duty の微調整を行う
#define BLM_ANGLE_DIFF_THRESHOLD2     RAD_30_DEGREE
    //30 度以上ずれている場合は、duty の調整を行う
```

通常制御時、15°(BLM_ANGLE_DIFF_THRESHOLD1)以上理想角度と現在の印加角度がずれた場合、duty の微調整を行い、30°(BLM_ANGLE_DIFF_THRESHOLD2)以上理想角度と現在の印加角度がずれた場合、duty の調整を行う。

```
#define BLM_HOLL_MOTOR   1
#define BLM_HOLL_PSEUDO 2
```

ホールセンサとして、モータ内蔵のホールセンサを使用する(1)か、疑似ホールセンサパターンを使用する(2)かを定義する定数。

```
#define BLM_HOLL_PSEUDO_SENSOR_AVERAGE
```

定義時、疑似ホールセンサパターンを使用する場合に、短期間の平均値を使用する。未定義の場合は、その時のA/D 変換値を使用する。

```
#define BLM_HOLL_PSEUDO_SENSOR_HYS
```

定義時、疑似ホールセンサパターンを使用する場合に、ヒステリシスを有効にする。未定義の場合は、ヒステリシを使用しない。

```
#define BLM_HOLL_PSEUDO_SENSOR_HYS_VAL              25
    //25 = 20mV/3300mV*4096, 20mV 程度ヒステリシスを付ける
```

ヒステリシスを付ける場合の、ヒステリシス値。上記の場合は、平均値+20mV で、0→1 に変化し、平均値-20mV で、1→0 に変化する。

```
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_1    15
    //疑似ホールセンサパターン使用時は速度に応じて 15-45°のオフセットを付ける
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_2    25
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_3    35
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_ANGLE_4    45
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_1      2000    //2000[rpm]までは 15°
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_2      6000    //6000[rpm]までは 25°
#define BLM_HOLL_PSEUDO_SENSOR_OFFSET_RPM_3      8000    //8000[rpm]までは 35°, それ以上は 45°
```

疑似ホールセンサパターンを使用する場合、回転速度に応じて磁界印加角度に定義値のオフセットを付ける設定。

```
#define BLM_CONTROL_PERIOD 50.0e-6f
```

基本制御周期、50us。

#define BLM_PORT_DEBUG

定義時、ポートデバッグを有効にする。P400~P403 をデバッグに使用。

```
#define BLM_DEBUG_PRINT1      //簡易デバッグ表示
//#define BLM_DEBUG_PRINT2    //詳細デバッグ表示（50us の割り込み関数内でのデバッグ表示）
//#define BLM_DEBUG_PRINT3    //詳細デバッグ表示（10ms の割り込み関数内でのデバッグ表示）
//#define BLM_DEBUG_PRINT4    //詳細デバッグ表示（ホールセンサの情報）
```

定義時デバッグ表示を有効化。

1.8. プログラムで使用している機能と割り込み

1.8.1. プログラムで使用しているマイコン機能

機能名	分類	用途
AGT0	タイマ	基本制御周期（50us 周期）
AGT1	タイマ	定期処理（10ms 周期）
GPT1,2,3	タイマ	CH-1 相補 PWM 波形生成
SCI9	通信	メッセージ表示
ADC0	A/D 変換	温度検出, VR 読み取り，相電圧・電流取得
IRQ1	端子割り込み	過電流検出(*1)

(*1)設定に応じて使用

1.8.2. プログラムで使用している割り込み

機能名	分類	割り込みレベル
IRQ1	端子割り込み	1
AGT0	タイマ	2
AGT1	タイマ	3
ADC0	A/D 変換	4
SCI9	通信	12

※割り込みレベル

0:最優度高い

15:優先度低い

優先度が n の割り込み処理実行中に、n より小さなレベルの割り込みは多重割り込みで処理されます。

1.9. デバッグ補助機能

モータの制御においては、デバッガを使用したデバッグが難しい場合があります。ブレークポイントを張って、プログラムを停止した場合、「磁界印加角度のインクリメント動作が止まりモータが回らない」等の状態となってしまうためです。

また、出力端子がアクティブな状態（U 相から V 相に電流を流している状態等）でプログラムが停止すると、過大な電流が流れて、モータドライバボードが破損します。（モータは PWM 駆動等、適切なタイミングで電流を ON/OFF しながら駆動する必要があります。モータ内部はコイル（インダクタンス）で構成されていますので、電流が流れる状態を維持すると（＝DC 的に電流を印加すると）、過大な電流が流れます。…インダクタンスは周波数に比例したインピーダンスを持つため、DC 的には理想的なインダクタンスのインピーダンスは 0 となります。…実際はコイルの配線抵抗もありますので、0 ではないですが。）
（マイコンのタイマ機能を使用して PWM 波形を生成している場合等、デバッガでプログラムの動作を止めても波形出力が継続されるケースもあります。）

モータのデバッグに関しては、デバッガで安易にブレークできない（ケースが多い）事に留意ください。

本サンプルプログラムでは、デバッグの補助機能として、
・UART(SCI)による情報表示
・端子を使ったデバッグ
を用意しています。

1.9.1. UART(SCI)を使用した情報表示

UART では、3 秒に 1 回、回転数や duty、温度等の情報を表示しています。

blm¥blm.h 内で

```
//デバッグ表示
#define BLM_DEBUG_PRINT1    //簡易デバッグ表示
//#define BLM_DEBUG_PRINT2  //詳細デバッグ表示（50us毎の割り込み内）
//#define BLM_DEBUG_PRINT3  //詳細デバッグ表示（10ms 毎の割り込み内）
//#define BLM_DEBUG_PRINT4  //詳細デバッグ表示（ホールセンサの情報）
```

BLM_DEBUG_PRINT1 を定義すると、簡易的な情報の表示。

```
  CH-1 START
d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+d+<<
```

黒字の部分が追加される表示

d+ ：始動制御(BLM_PHASE_1)で duty を増加
d- ：始動制御(BLM_PHASE_1)で duty を減少

```
> > > > > > > > > > > > > > >> > > >> > > >> > > > > > > > > > > > > > > > > > > > < < < < < < <
< < < < < < < < < < < < < < < < < < < < < < < < < < < < < < <
```

<< :通常制御(BLM_PHASE_2)で回転が速すぎるので duty 値を大きく減少

< :通常制御(BLM_PHASE_2)で回転が速いので duty 値を減少

>> :通常制御(BLM_PHASE_2)で回転が遅すぎるので duty 値を大きく増加

> :通常制御(BLM_PHASE_2)で回転が遅いので duty 値を増加

といった情報が表示に追加されます。

元々の表示(3 秒に 1 度の情報表示)は

blm¥blm_main.c 内で

```
const unsigned long information_display_interval = (unsigned long)(3.0 / 10.0e-3);  //3 秒毎に画面に情報を表示（10ms で何カウントか）
```

3 秒に 1 回となる様設定されているので、3.0 のところを変更する。

```
const unsigned long information_display_interval = 0;
```

ゼロに設定して、3 秒に 1 回の画面表示を止めてしまう事も可能です。

BLM_DEBUG_PRINT2 を定義すると、50us の割り込みルーチンでの詳細情報の表示。

```
  CH-1 START
d+d+d+d+d+d+d+d+d+d+D2 *REV*D2 C:0, H:6->2@304
d+d+D2 C:0, H:2->6@142                    duty を増加させていく過程
d+D2 C:0, H:6->4@207
d+D2 C:0, H:4->5@276
d+d+D2 *REV*D2 C:0, H:5->4@65             d+は BLM_DEBUG_PRINT1
d+d+D2 C:0, H:4->5@189                    で表示される情報
（中略）
D2 C:0, H:1->3@284
D2 C:0, H:3->2@339

************************************************
                          始動制御から通常制御
                          切り替わり
->phase2                                  <<は BLM_DEBUG_PRINT1
<< D2 C:0, H:2->6@51>90                   で表示される情報
D2 C:0, H:6->4@143>150
D2 C:0, H:4->5@208>210
（中略）
D2 C:0, H:6->4@143>150
D2 C:0, H:4->5@217>210
```

d+d+D2 ***REV***D2 C:0, H:5->4@65
設定回転方向と逆回転した場合
（始動時はモータ軸が振動するので逆回転も起こります）

D2 ： BLM_DEBUG_PRINT2 で表示される情報（50us の割り込み内での情報表示）

C:0 ：CH-1 の情報

H4->5 ：ホールセンサが 4 から 5 に切り替わった

@217 ：その時の印加磁界角度（217°）

>210 ：その時の印加磁界角度を（217°）、センサー切り替わり時の理想角度である（210°）に補正

※通常制御の場合

BLM_DEBUG_PRINT3 を定義すると、10ms の割り込みルーチンでの詳細情報の表示。

```
CH-1 START
d+D3 P:1 C:0, rpm=0, duty=75, stable=0
d+D3 P:1 C:0, rpm=0, duty=112, stable=0
d+D3 P:1 C:0, rpm=0, duty=131, stable=0
d+D3 P:1 C:0, rpm=0, duty=140, stable=0
（中略）
d+D3 P:1 C:1, rpm=228, duty=225, stable=0
d+D3 P:1 C:1, rpm=318, duty=227, stable=0
d+D3 P:1 C:1, rpm=397, duty=229, stable=0
d+D3 P:1 C:1, rpm=476, duty=231, stable=0
（中略）
D3 P:1 C:0, rpm=1237, duty=195, stable=21
D3 P:1 C:0, rpm=1222, duty=195, stable=22
D3 P:1 C:0, rpm=1207, duty=195, stable=23
D3 P:2 C:0, rpm=1380/1237/1397/3512, duty=196
<< D3 P:2 C:0, rpm=1320/1233/1393/3512, duty=198
D3 P:2 C:0, rpm=1560/1245/1405/3512, duty=199
D3 P:2 C:0, rpm=1440/1252/1413/3512, duty=200
（中略）
D3 P:2 C:0, rpm=3660/3371/3476/3476, di=0, duty=312
D3 P:2 C:0, rpm=3840/3420/3476/3476, di=0, duty=312
D3 P:2 C:0, rpm=3120/3416/3476/3476, di=0, duty=312
```

duty を増加させていくが
回転は始まらない

d+, <<は BLM_DEBUG_PRINT1
で表示される情報

徐々に回転が速くなる
（回転判定判定はされていない）

duty を上げなくても安定して回転
（始動制御を抜ける直前）

通常制御に移行後
目標回転数に近づけていく段階

目標回転数に達し PI 制御移行後

D3 ： BLM_DEBUG_PRINT3 で表示される情報（10ms の割り込み内での情報表示）

P:1 ：始動制御(BLM_PHASE_1)

C:0 ：CH-1

rpm= ：回転数（平均）

duty= ：設定 duty 値×1,000（処理速度向上のため整数値での取り扱い）

stable= ：回転安定数（24 を超えると通常制御に移行）

P:2 ：通常制御(BLM_PHASE_2)

rpm= ：回転数/回転数（平均）/制御回転数/目標回転数

di= ：duty の累積加算値×10,000（処理速度向上のため整数値での取り扱い）

（di=0 の時は duty を加算も減算もしない状態）

BLM_DEBUG_PRINT4 を定義すると、ホールセンサの情報の表示。

```
pos:6,6
pos:6,6
pos:4,6
pos:4,4
pos:4,4
pos:5,4
pos:5,5
pos:5,5
pos:1,5
```

n,n ホールセンサ，疑似ホールセンサ

pos:
6 モータ内蔵のホールセンサの値が 6
,6 UVW の相電圧から算出した疑似ホールセンサパターンの値が 6

※上記では、疑似ホールセンサパターンの値が多少遅れている（先にホールセンサの値が変化している）

デフォルトでは、1ms 毎のホールセンサ値を表示します。

※UART の出力のバッファに関して

UART 向けに 1024 バイト（文字）のバッファを 2 つ設けています（出力用と格納用）。出力が終わった段階で、出力用と格納用のバッファを差し替えます。（交互に使用します）
文字出力は、115,200bps なので、1 文字の出力に約 87us 程度掛かります。バッファが溢れた場合、データを捨てる設定としています。

blm¥blm_main.c

```
    g_sci_send_nowait_flag = FLAG_SET;  //UART の表示が間に合わない場合はデータを捨てる（リアルタイム処理優先）
```

上記フラグを設定しない場合、バッファが溢れた場合、UART の文字出力が終わりバッファに空きができるまで、プログラムの処理が止まりますので、モータ駆動の場合はフラグを設定する事が推奨です。

※バッファが溢れた場合、データを捨てる設定ですので、出力する情報量が多い場合、途中で表示が切れたようになります。

※バッファの 1024 バイトを超える情報を表示させる場合、（モータに電圧を印加する制御を止めた後で、）
g_sci_send_nowait_flag = FLAG_CLEAR;
とすると、バッファに空きが生じるまで、文字出力を待つようになりますので、全ての情報が出力可能です
（メモリ容量に空きがあれば、sci.h 内で定義されている 1024 という値を増やすという対応もできます。）

UART の文字表示は、sci¥sci.c に定義されている関数で行っています。

・UART 関連関数

sci_start

SCI の初期化を行います。最初に実行してください。

使用例:
sci_start();

sci_write_str

文字列の表示を行います。

使用例:
sci_write_str("display string¥n");
→表示文字:display string[改行]

sci_write_uint16

数値表示を行います。

使用例:
unsigned short a = 12345;
sci_write_uint16(a);
→表示文字:12345

バリエーション:

関数名	引数	説明
sci_write_uint8	unsigned char	符号なし 8bit 数値表示
sci_write_uint16	unsigned short	符号なし 16bit 数値表示
sci_write_uint32	unsigned long	符号なし 32bit 数値表示
sci_write_int8	char	符号付き 8bit 数値表示（負数の場合のみ-を表示します）
sci_write_int16	short	符号付き 16bit 数値表示（負数の場合のみ-を表示します）
sci_write_int32	long	符号付き 32bit 数値表示（負数の場合のみ-を表示します）

sci_write_uint16_hex

16 進数で数値表示を行います。("0x"等の表示は行いません)

使用例:
unsigned short a = 0x1234;
sci_write_uint16_hex(a);
→表示文字:1234

バリエーション:

関数名	引数	説明
sci_write_uint8_hex	unsigned char	符号なし 8bit hex 表示
sci_write_uint16_hex	unsigned short	符号なし 16bit hex 表示
sci_write_uint32_hex	unsigned long	符号なし 32bit hex 表示

sci_write_flush

出力バッファに溜まっているデータを吐き出させます。(出力バッファが空になるまで、プログラムの実行が止まります。)

使用例:
sci_write_flush();

補足:
長いデータ(バッファ容量の 1024 バイト)を超えるデータを出力させる場合、バッファ溢れによりデータが捨てられますので、定期的に本関数を実行するか、
g_sci_send_nowait_flag = FLAG_CLEAR;
として、バッファ溢れ時に出力を待つようにしてください。

また、デバッガでプログラムを停止させた場合、文字出力の処理も停止しますので、プログラムのブレーク前に画面表示を終わらせたい場合は、本関数実行後にブレークを掛けてください。

sci_read_char

キーボードから入力した文字の読み出しを行います。

使用例:

```
unsigned short ret;
unsigned char c;
ret = sci_read_char(&c);
if (ret != SCI_RECEIVE_DATA_EMPTY)
{
   if (c == 's') blm_stop(BLM_CH1);       //キーボードから s が入力された際、CH1 のモータを停止させる
}
```

補足:

入力バッファは(初期設定値で)16 バイト(文字)用意しています。sci_read_char ではバッファに格納されている一番古いデータが取り出せますので、複数の文字(例えば 123 等の数値入力させた場合)を読み出す場合は、関数の戻り値が SCI_RECEIVE_DATA_EMPTY になるまで複数回本関数を呼び出してください。

float2str

float 型の変数から文字列への変換を行います。

使用例:

```
float a = 1.2345;
char buf[20];
float2str(a, 2, buf);
sci_write_str(buf);
```

→表示文字:1.23

引数:

第 1 引数　表示させる数値
第 2 引数　表示させる小数点以下の桁数
第 3 引数　文字列格納バッファ

バリエーション:

関数名	第 1 引数	説明
float2str	float	浮動小数点数(float)の文字列への変換
double2str	double	浮動小数点数(double)の文字列への変換
float2str_eformat	float	浮動小数点数(float)の文字列への変換(e 形式) ※1.23e-3 等
double2str_eformat	double	浮動小数点数(double)の文字列への変換(e 形式) ※1.23e-3 等

補足:

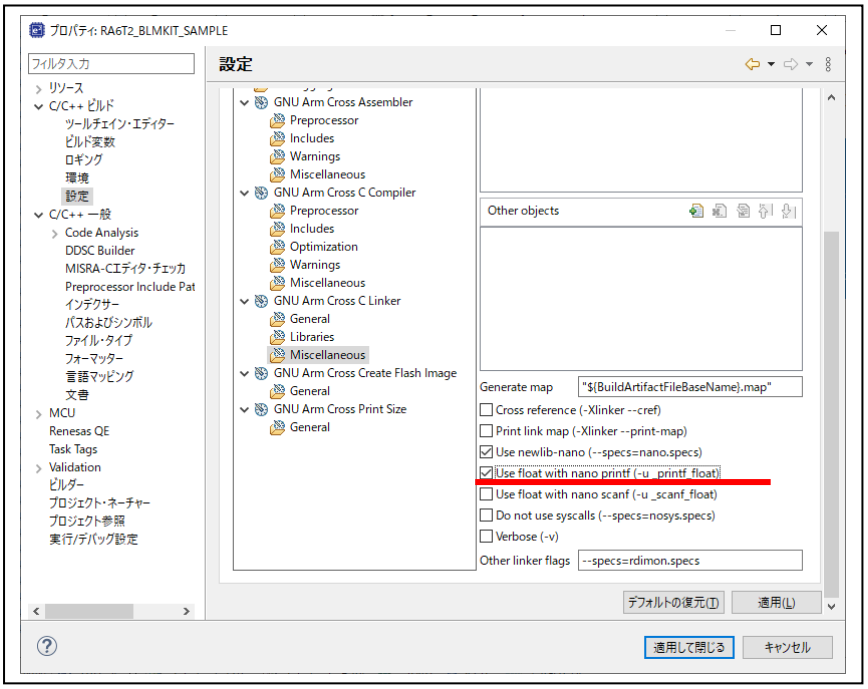
浮動小数点数の表示(文字列への変換)は、標準のライブラリ関数を使用して、

sprintf(buf, "%.2f", a);

でも行えますが、この場合

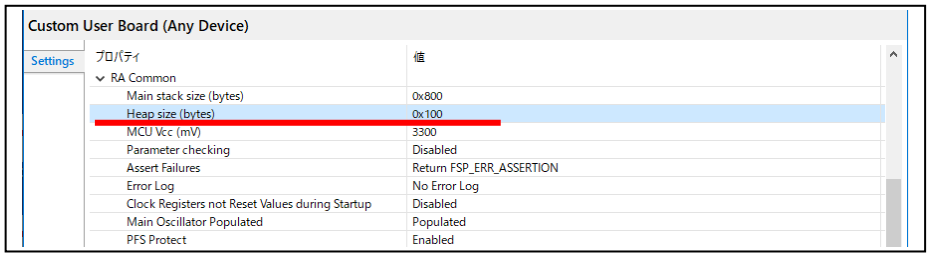
(1)リンカオプションの指定

C/C++ビルド - 設定 - GNU Arm Cross C Linker - Miscellaneous - Use float with nano printf(-u printf_float) にチェックを入れる



(2)HEAP メモリの確保

FSP Configuration BSP タブ RA common - Heap size(bytes) "0x0"を"0x100"等に設定する



の設定が必要です。

1.9.2. 端子を使ったデバッグ

モータ制御で特定のアクションが起こった際に UART 表示よりもリアルタイムで情報を出力する用途で使用頂ける機能です。

本機能を有効化する場合、

¥blm¥blm.h

#define BLM_PORT_DEBUG

上記定義を有効化してください。(デフォルトで有効)

ポート名(基板端子)	当該ポート:L 出力	当該ポート:H 出力	当該ポート:トグル出力
P400(J2-11)(*1)	BLM_DEBUG_PORT1_L	BLM_DEBUG_PORT1_H	BLM_DEBUG_PORT1_T
P401(J2-10)	BLM_DEBUG_PORT2_L	BLM_DEBUG_PORT2_H	BLM_DEBUG_PORT2_T
P402(J2-9)	BLM_DEBUG_PORT3_L	BLM_DEBUG_PORT3_H	BLM_DEBUG_PORT3_T
P403(J2-8)	BLM_DEBUG_PORT4_L	BLM_DEBUG_PORT4_H	BLM_DEBUG_PORT4_T

(*1)マイコンボード上の LED2 にも接続されています

プログラム内で、

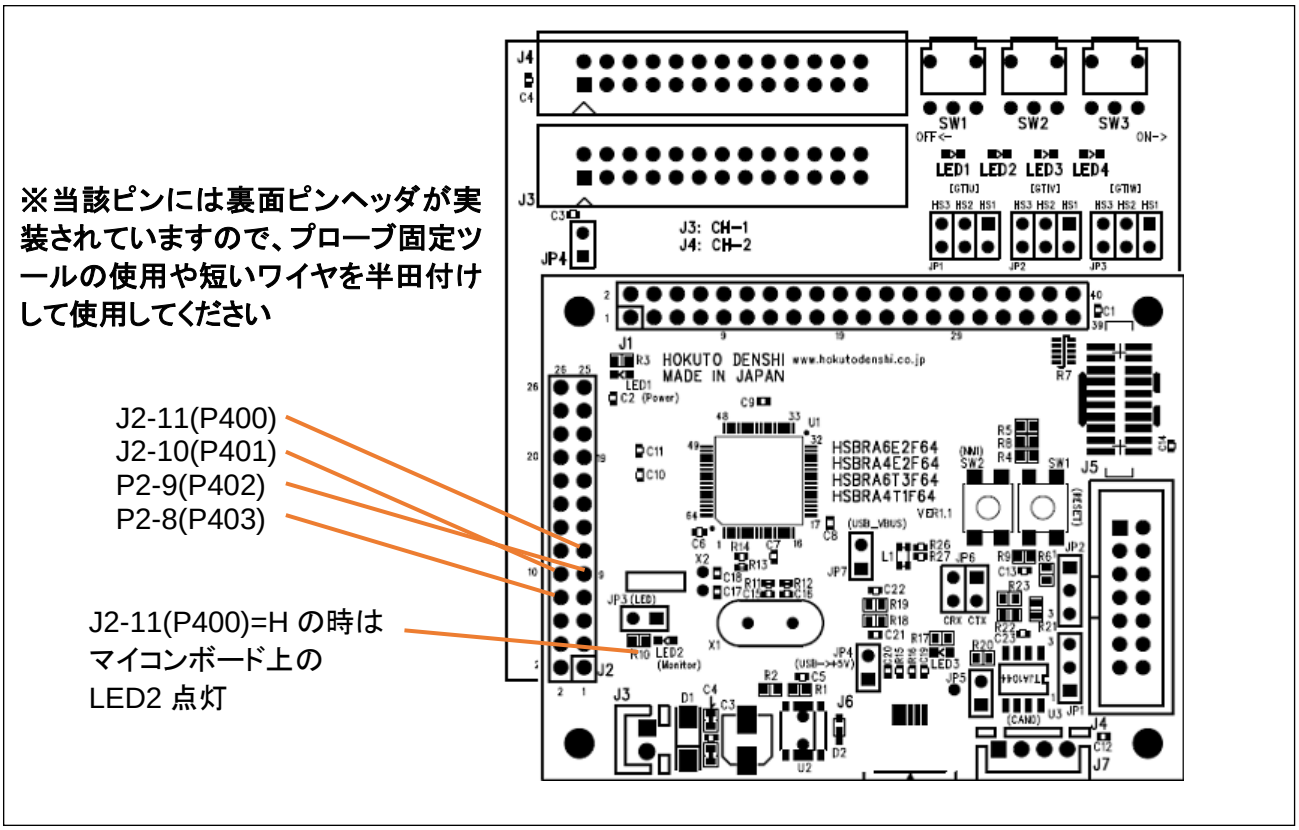
BLM_DEBUG_PORT1_H
特定の処理
BLM_DEBUG_PORT1_L

とすると、P400 端子をオシロスコープ等でモニタする事により、特定の処理に掛かる時間を計測可能です。

サンプルプログラムでは、以下の設定としています。

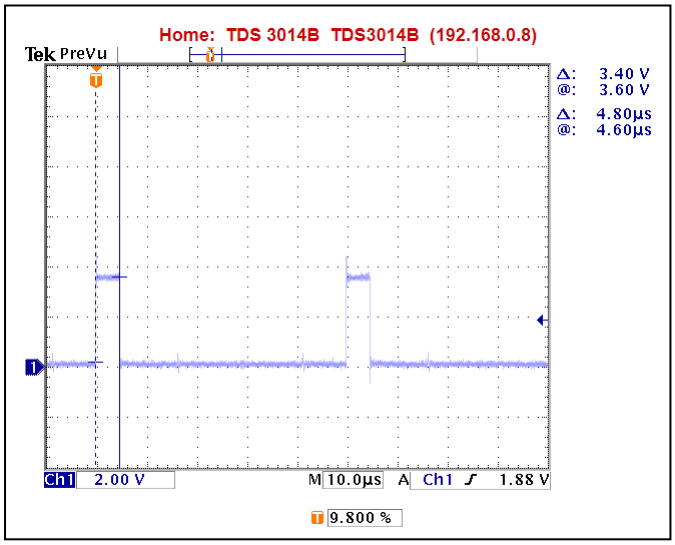
ポート名(基板端子)	観測内容
P400(J2-11)	PHASE2(通常制御)に移行した際に H(マイコンボード上の LED2 点灯)
P401(J2-10)	50us の割り込み処理の先頭で H, 終了時に L
P402(J2-9)	10ms の割り込み処理の先頭で H, 終了時に L
P403(J2-8)	A/D 変換処理開始時に H、終了時に L

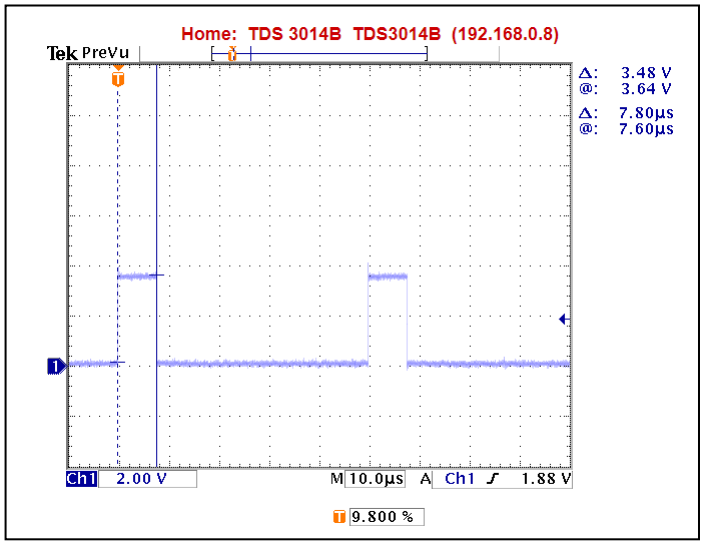
ポート名	観測内容
LED2	通常制御で印加磁界が進んでいる時 LED2 点灯(duty を減らす動作)
LED3	通常制御で印加磁界が遅れている時 LED3 点灯(duty を増やす動作)



端子デバッグ機能を使用する際には、プログラム内に端子を変化させる命令（L 出力、H 出力、トグル出力のいずれか）を埋め込んでおき、オシロスコープ等を使って端子の変化を観測してください。

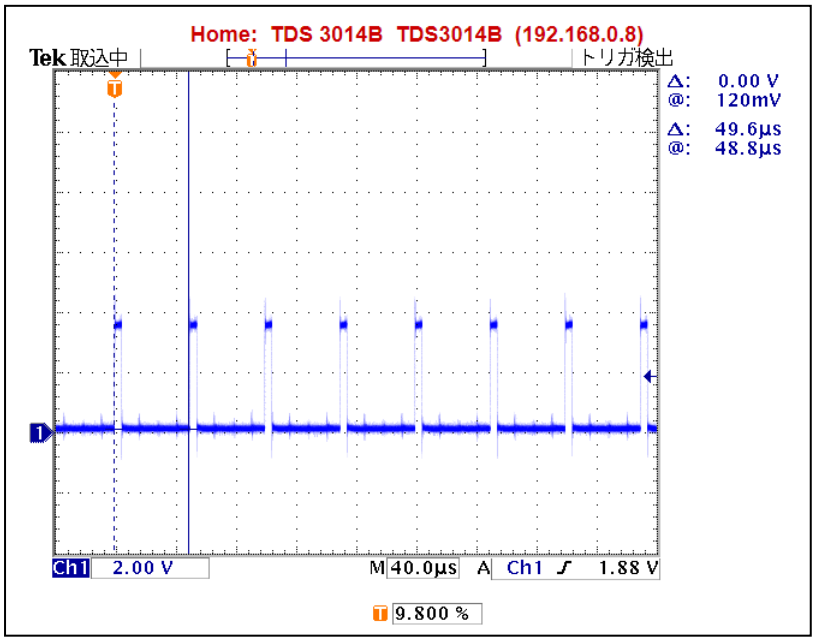
・50us の割り込みモニタ例(P401), RA6T3　　RA4T1





上記は、50us の割り込み処理（割り込みの先頭で H、割り込みの終了で L）をモニタした例です。処理時間は、4.8us 程度（RA6T3 の場合）なので、50us 周期に対して余力があり、もう少し複雑な処理を追加する事も可能ですし、基本制御周期(50us)を短くする事が可能なことが見て取れます。（RA4T1 の場合は 7.8us 程度）

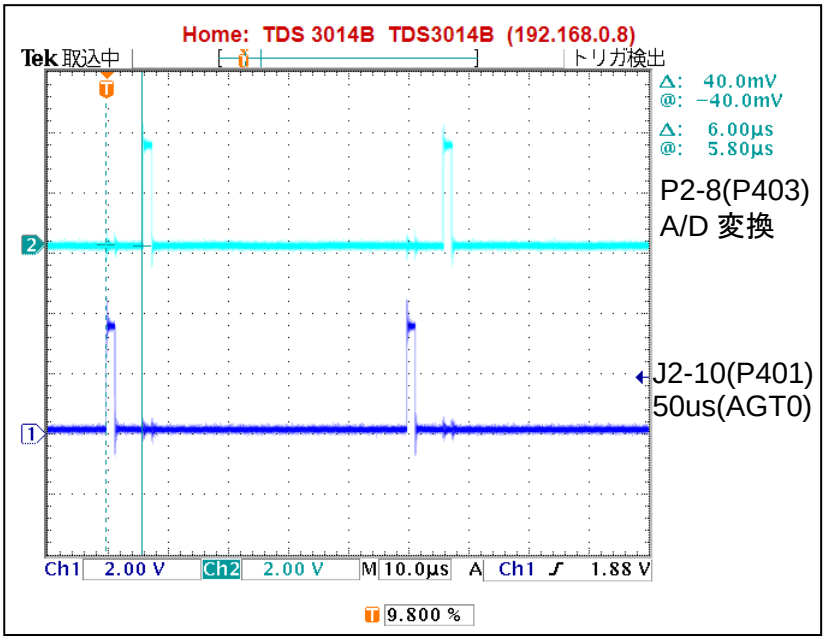
・50us の割り込みを長周期で観測(P401), RA6T3



処理が歯抜け（50us 毎に実行されないタイミングがある）になっていないか、1 回の処理時間に大きなばらつきがないか等の情報が得られます。もし、50us の割り込み処理において、実行時間が長い処理があれば、10ms の定期処理に追い出す等の対応が考えられます。）

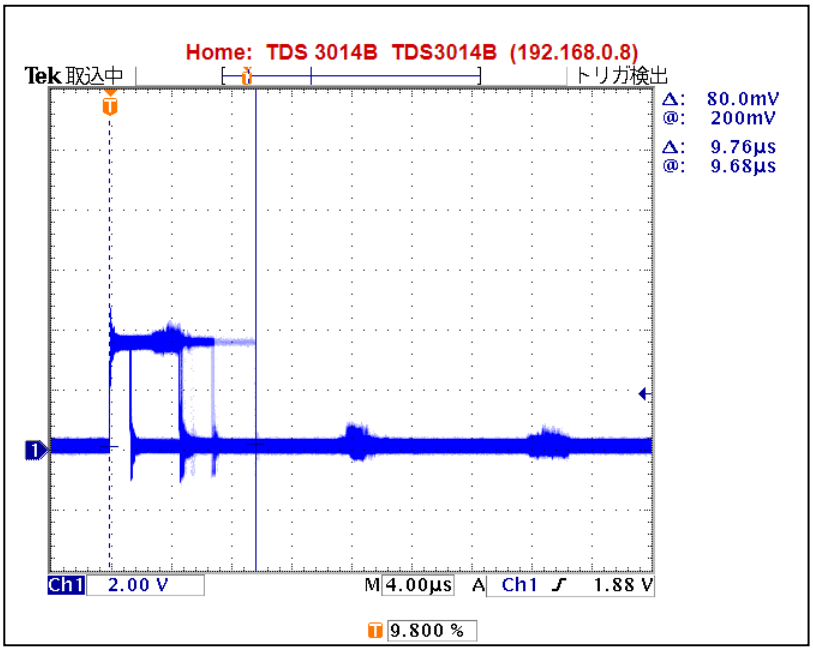
（モータが上手く回らない場合は、定期的に割り込みが実行されていないという事も考えられます。端子モニタがデバッグで役に立つ事もあります。）

・50us の割り込みと A/D 変換割り込み(P401, P403), RA6T3



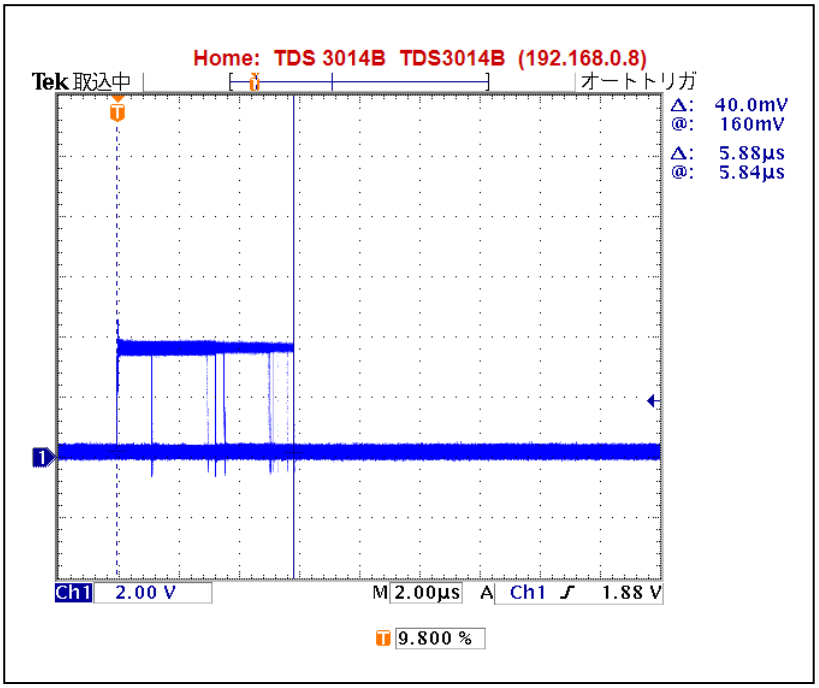
A/D 変換は 50us の割り込みで起動していますが、起動後 6us 程度で A/D 変換が終了しており、A/D 変換の処理（長周期と短周期の移動平均の算出）も、次の A/D 変換起動までに終わっている事が確認できます。

・50us の割り込みを重ね書きした場合(P401), RA6T3



50us の割り込みを重ね書きした場合の波形例です。通常 5us 程度ですが、最大で 10us 程度掛かるケースもある様に見受けられます。（RA4T1 の場合は、最大 12us 程度）（50us 毎の割り込み処理に、50us 以上の時間が掛かって、割り込みがスキップされる様な事は無い様に見受けられます。）

・10ms の割り込みを重ね書きした場合(P402), RA6T3



10ms の割り込みを同じように重ね書きすると、最大 6us 程度（通常 3us）掛かるケースがある様に見受けられます。（10ms に 1 回の処理ですので、処理を大幅に追加しても問題ないように見受けられます。）

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2023.12.1	—	初版発行

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:https://www.hokutodenshi.co.jp

商標等の表記について

- 全ての商標及び登録商標はそれぞれの所有者に帰属します。
- パーソナルコンピュータを PC と称します。

ルネサス エレクトロニクス RA6T3/RA4T1(QFP-64 ピン)搭載
ブラシレスモータスタータキット

ブラシレスモータスタータキット (RA6T3/RA4T1) [ソフトウェア サンプルプログラム編] 取扱説明書

株式会社 北斗電子

©2023 北斗電子 Printed in Japan 2023 年 12 月 1 日改訂 REV.1.0.0.0 (231201)