ブラシレスモータスタータキット(RA6T2)
[ソフトウェア　チュートリアル編]
取扱説明書

ルネサス エレクトロニクス社　RA6T2(QFP-100 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、アクセスランプが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
・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. チュートリアル

本チュートリアルでは、マイコンの基本的なプログラムを説明致しますが、ルネサスエレクトロニクスの Web から「RA6T2 グループ　ユーザーズマニュアル　ハードウェア編」を予めダウンロードして、手元に用意してください。マイコンの詳細な設定は、このマニュアル（以下「ハードウェアマニュアル」と記載する）に記載されています。

また、本マニュアルで説明するプログラムは、ルネサスエレクトロニクス e2studio+FSP の環境向けに作成されていますので、RA 向けの e2studio+FSP の環境を PC にインストールしておいてください。なお、e2studio+FSP のインストール等は、ルネサスエレクトロニクス社のマニュアルを参照してください。
（チュートリアル、サンプルは FSP Ver3.7.0 で作成されていますので、Ver3.7.0 以上の FSP の使用が必要となります）

マイコンの基本的なプログラムは行えるが、モータ制御は初めてという方を想定した構成となっており、ホールセンサを使用してブラシレスモータを回転させる方法。モータが動作しているときの、電圧や電流を観測する方法。また、モータドライバボードの保護回路の使用方法の習得を目的にしています。

以下が、本チュートリアルで学べる事柄となります。

・マイコンボードを動作させるための初期設定（クロック設定やモジュールスタンバイの解除）
・モータのコイルに流す電流を制御する方法
・A/D 変換
・PWM（パルス幅変調）
・ホールセンサの値を取得する方法
・温度値の取得
・過電流検出の方法
・モータが動作しているときの相電圧・相電流の取得
・ベクトル型制御

本キット付属の「接続ボード」には、モータドライバボードを接続するボックスコネクタが、3 箇所（CH-1, CH-2, CH-3）あります。モータドライバボードは、どのコネクタに接続しても問題ありません。

チュートリアル 2~7, A, B, B2 では、3ch のモータの制御に対応しています。別売の「ブラシレスモータ拡張キット」をご用意頂ければ、最大 3ch の動作を確認する事ができます。

－チュートリアル一覧－

チュートリアル	プロジェクト名	内容
チュートリアル 1	RA6T2_BLMKIT_TUTORIAL1	マイコンボードを動作させる初期設定 スイッチの読み取りと LED の制御
チュートリアル 2	RA6T2_BLMKIT_TUTORIAL2	モータへの通電方法 （モータは回転しません）
チュートリアル 3	RA6T2_BLMKIT_TUTORIAL3	A/D 変換と PWM 波形の生成方法
チュートリアル 4	RA6T2_BLMKIT_TUTORIAL4	実際にモータを回転させる方法
チュートリアル 5	RA6T2_BLMKIT_TUTORIAL5	ホールセンサの値を利用する方法
チュートリアル 6	RA6T2_BLMKIT_TUTORIAL6	過電流・過熱保護
チュートリアル 7	RA6T2_BLMKIT_TUTORIAL7	相電圧・相電流の観測

チュートリアル 1~7 までは、つながりのある内容となっており、例えばチュートリアル 6 はチュートリアル 5 の内容に「過電流・過熱保護」の機構を追加する内容になっています。一つ前のチュートリアルに少しずつ機能追加を行っていき、モータ制御プログラムを組み立てていく内容です。

チュートリアル	プロジェクト名	内容
チュートリアル A	RA6T2_BLMKIT_TUTORIAL_A	マイコンのハードウェア制御機構を用いたモータ制御 （CH-1 のみ）

チュートリアル A は、チュートリアル 7 をベースに、RA6T2 マイコンが持っている機能である出力相切り替え機能(GPT_OPS)を使ってモータを回す内容です。
※CH-1 のみ
※CH-2, CH-3 はチュートリアル 7 の内容と変わりません

チュートリアル	プロジェクト名	内容
チュートリアル B	RA6T2_BLMKIT_TUTORIAL_B	相補 PWM を使ったモータ制御（回転数固定）
チュートリアル B2	RA6T2_BLMKIT_TUTORIAL_B2	相補 PWM を使ったモータ制御（回転数可変）

チュートリアル B は、チュートリアル 7 をベースに、モータ制御の部分をベクトル制御とも呼ばれる相補 PWM 駆動に変えたものです。

チュートリアル 1~7 は連続したチュートリアル。モータを回す上で基本的な内容を 1 ステップずつ追加していく内容。A と B はチュートリアル 7 の内容から枝分かれするイメージです。

1.1. マイコンボード初期設定

参照プロジェクト:RA6T2_BLMKIT_TUTORIAL1

（アーカイブファイル:RA6T2_BLMKIT_TUTORIAl1.zip）

本キットに付属のマイコンボード(HSBRA6T2F100)は、RA6T2 グループのマイコンを搭載しており、クロック周波数 240MHz で動作させることが出来ます。

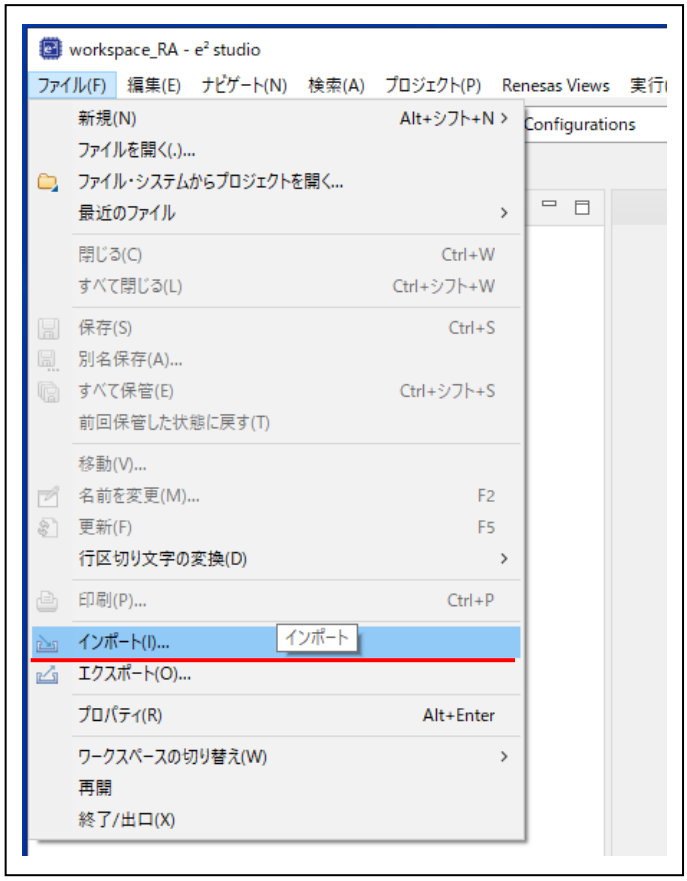
マイコンの動作モードやクロック周波数は、プログラムで設定する必要があります。

本プロジェクトでは、

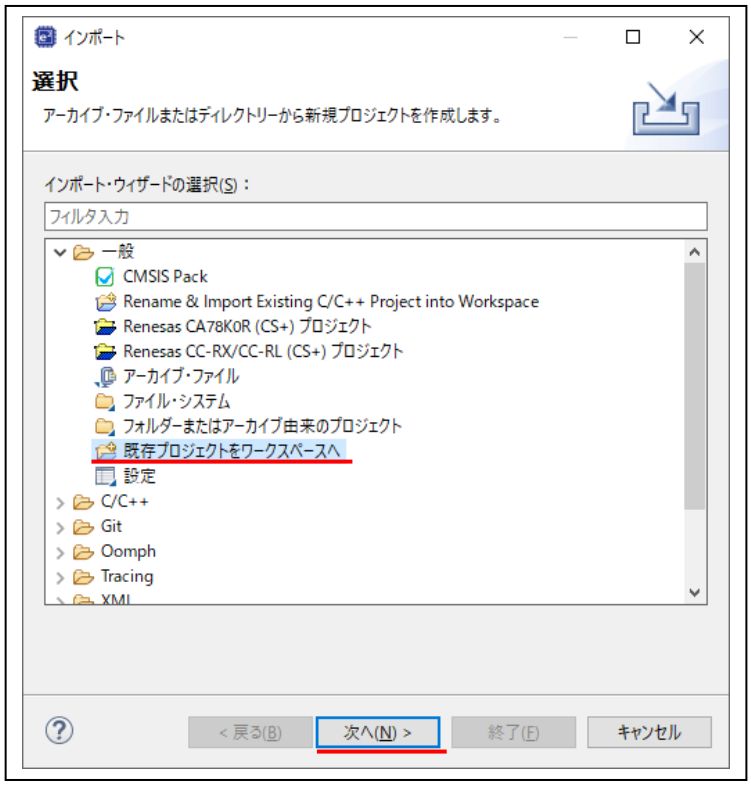
・マイコンボードの初期設定
・基本的な動作

を確認します。（※本プロジェクトは、モータドライバボード・接続ボードをつないだ状態で実行してください）

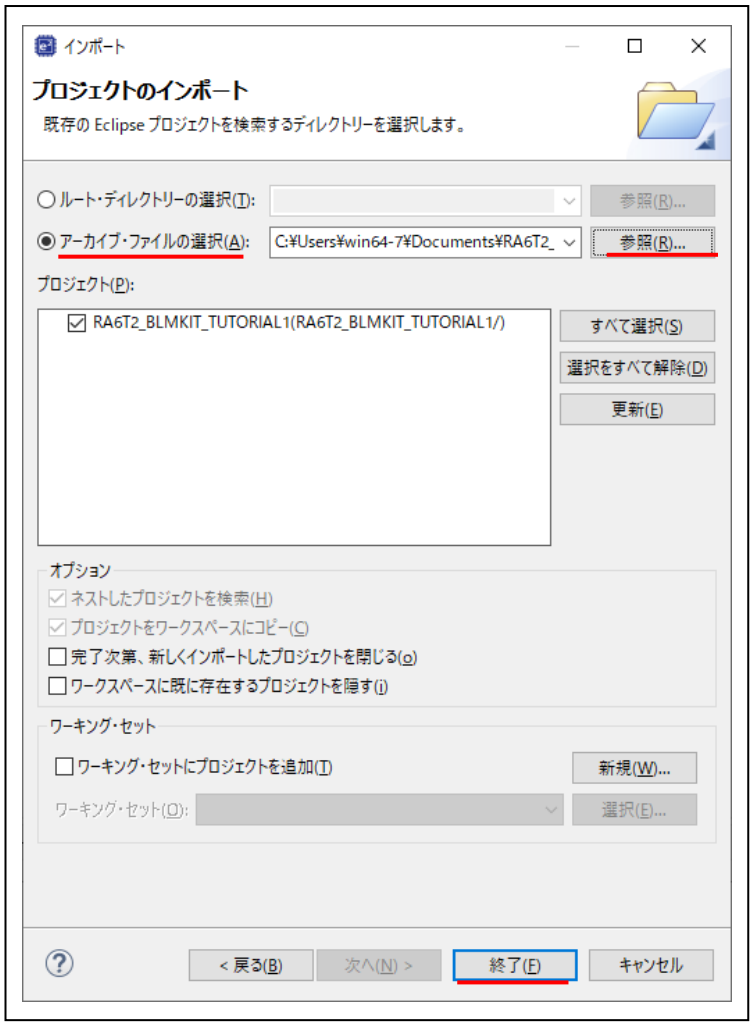
CD 内のアーカイブ（e2studio のプロジェクトをアーカイブした）ファイル、RA6T2_BLMKIT_TUTORIAl1.zip を、e2studio のプロジェクトにインポートしてください。



e2studio の「ファイル」メニューで「インポート」を選択します。

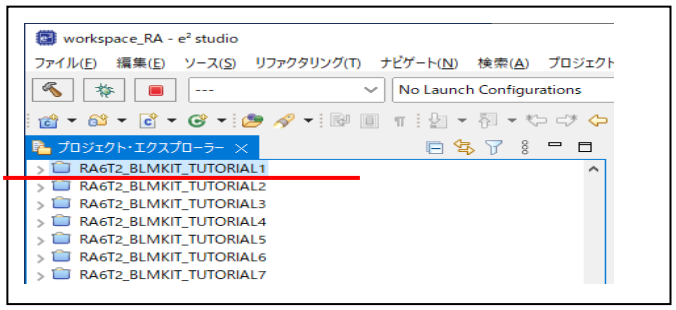


「一般」グループ内の「既存プロジェクトをワークスペースへ」を選択。「次へ」。



「アーカイブ・ファイルの選択」を選ぶ。
参照ボタンを押し、(RA6T2_BLMKIT_TUTORIAL1.zip)を選択。
プロジェクト
RA6T2_BLMKIT_TUTORIAL1 がインポート対象に選択されている事を確認し、「終了」を押す。

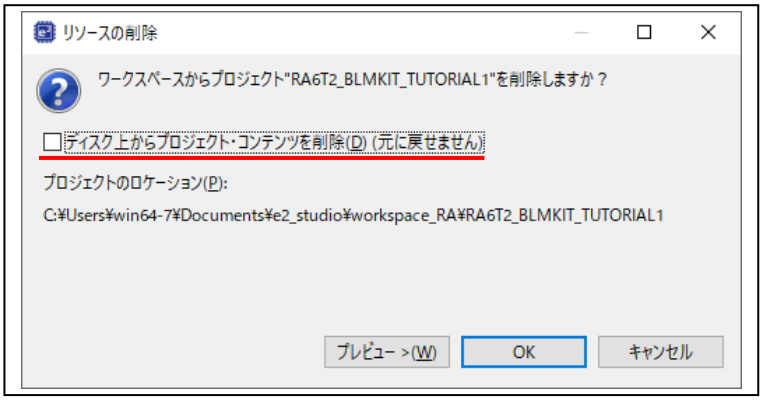
プロジェクト・エクスプローラ上で、プロジェクト「RA6T2_BLMKIT_TUTORIAL1」が見えていればプロジェクトのインポートは成功です。



※上記スクリーンショットは他のチュートリアルもインポートした状態です

※同一名称のプロジェクトが存在している場合インポートが出来ませんので、ワークスペースを別に作成するか、既存のプロジェクトを削除後にインポートしてください

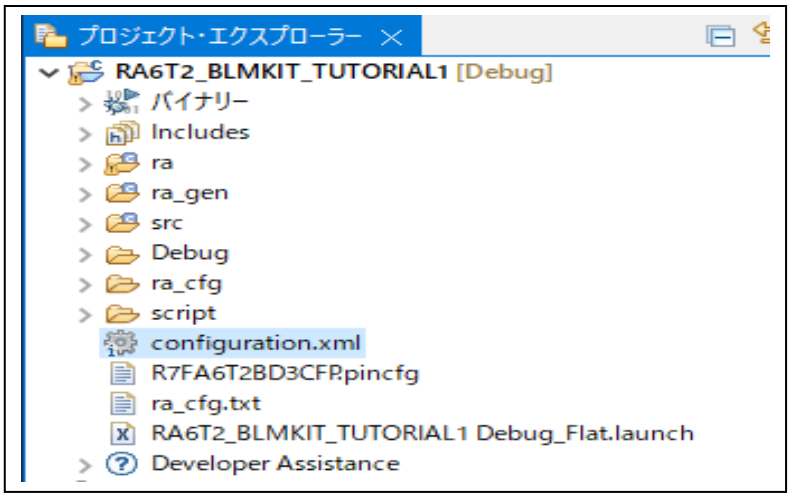
[参考]
※プロジェクトを削除する際に、「ディスク上からプロジェクト・コンテンツを削除」のチェックを入れないで削除した場合は、プロジェクトの実体（フォルダ、ファイル類）は残ったままとなりますので、一からプロジェクトのインポートをやり直したい場合は、「ディスク上からプロジェクト・コンテンツを削除」にチェックを入れてプロジェクトを削除してください



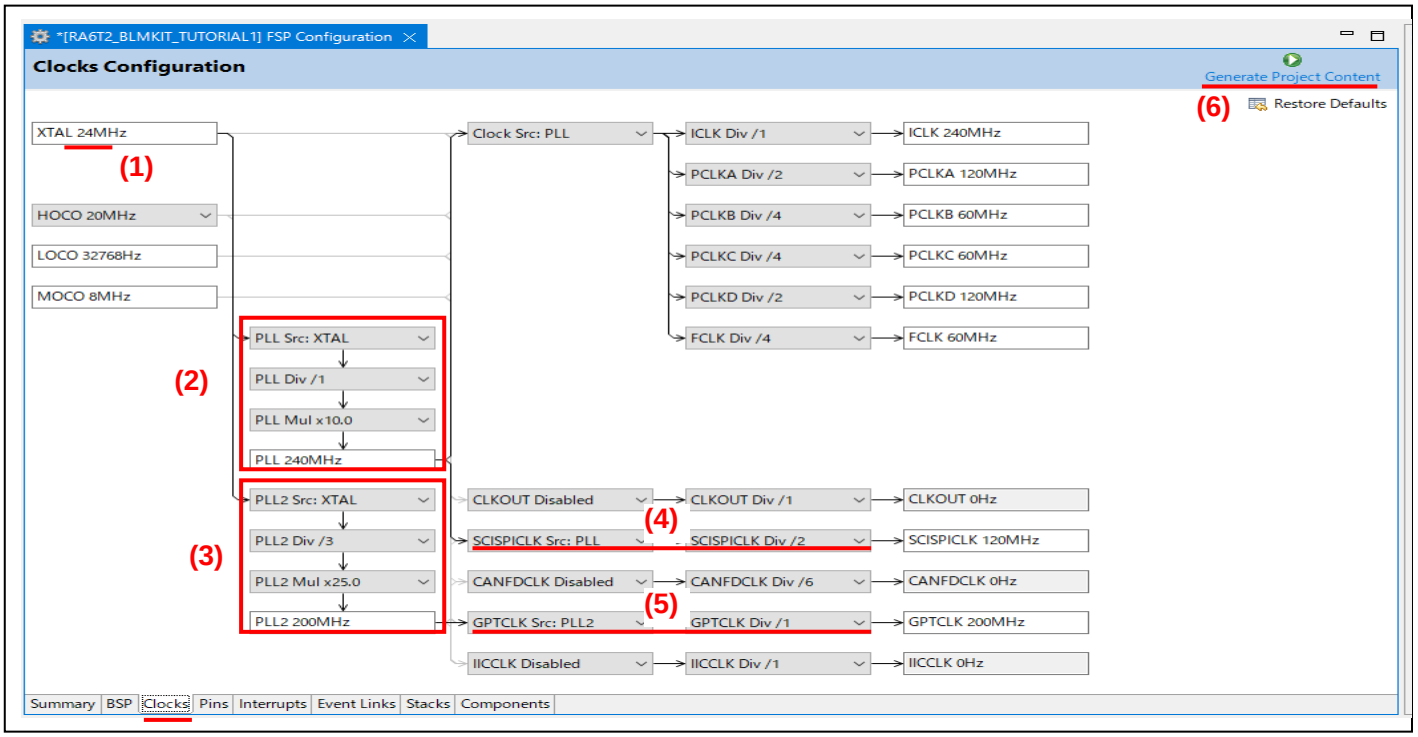
プロジェクトのインポート後、プロジェクトの基本的な設定を行う項目を次ページ以降に示します。
※RA6T2_BLMKIT_TUTORIAL1 プロジェクトをインポートした場合、次ページ以降の設定は済んだ状態となっています。プロジェクトを新規作成した場合、どのような設定を行う必要があるかの参考のために記載しています。
（RA6T2_BLMKIT_TUTORIAL1 プロジェクトを作成した手順を示しています）

マイコンで動作するプログラムを作成する場合、クロック等の初期設定が必要です。ルネサスエレクトロニクス社がリリースしている FSP を使用すると、GUI 画面でクロックの設定を行い、コード生成ボタンを押す事で、初期設定のプログラムコードが出力されます。

プロジェクト・エクスプローラ内の、RA6T2_BLMKIT_TUTORIAL1 のフォルダアイコンをダブルクリックして、プロジェクトを開いた状態として、



configuration.xml（歯車のアイコンのファイル）をダブルクリック



「Clocks」タブを選択。

(1)XTAL 24MHz (24 000 000) を入力

(2)PLL 設定　ソース XTAL，入力分周比　PLL Div/1，逓倍比 x10 を選択（PLL 出力周波数 240MHz）

(3)PLL2 設定　ソース XTAL，入力分周比　PLL Div/3，逓倍比 x25 を選択（PLL 出力周波数 200MHz）

(4)SCISPI クロックソース　PLL を選択

(5)GPT クロックソース　PLL2 を選択

[(6)コード生成ボタン(Generate Project Content)を押す]

(1)は、メインクロックの発振周波数を入力します。HSBRA6T2F100 には、24MHz の水晶振動子が搭載されていますので、「24 000 000」を入力します。

(2)は、PLL の入力側の分周比（ここでは、24MHz の 1/1 である 24MHz）を設定します。PLL の逓倍比は、x 10 を設定します。

これにより、

PLL の出力周波数：24[MHz] /1 x10 = 240 [MHz]

の設定となります。

(3)は 2 つ目の PLL の設定で、24MHz / 3 x 25 = 200 [MHz]の設定とします。

(4)は、SCI（シリアル通信）, SPI（シリアルペリフェラル通信）の設定クロックで、PLL(240MHz)ベースで 1/2 = 120MHz を設定します。

(5)は、GPT（モータ制御に使用するタイマ）の設定クロックで、PLL2(200MHz)ベースで、1/1=200MHz を設定します。

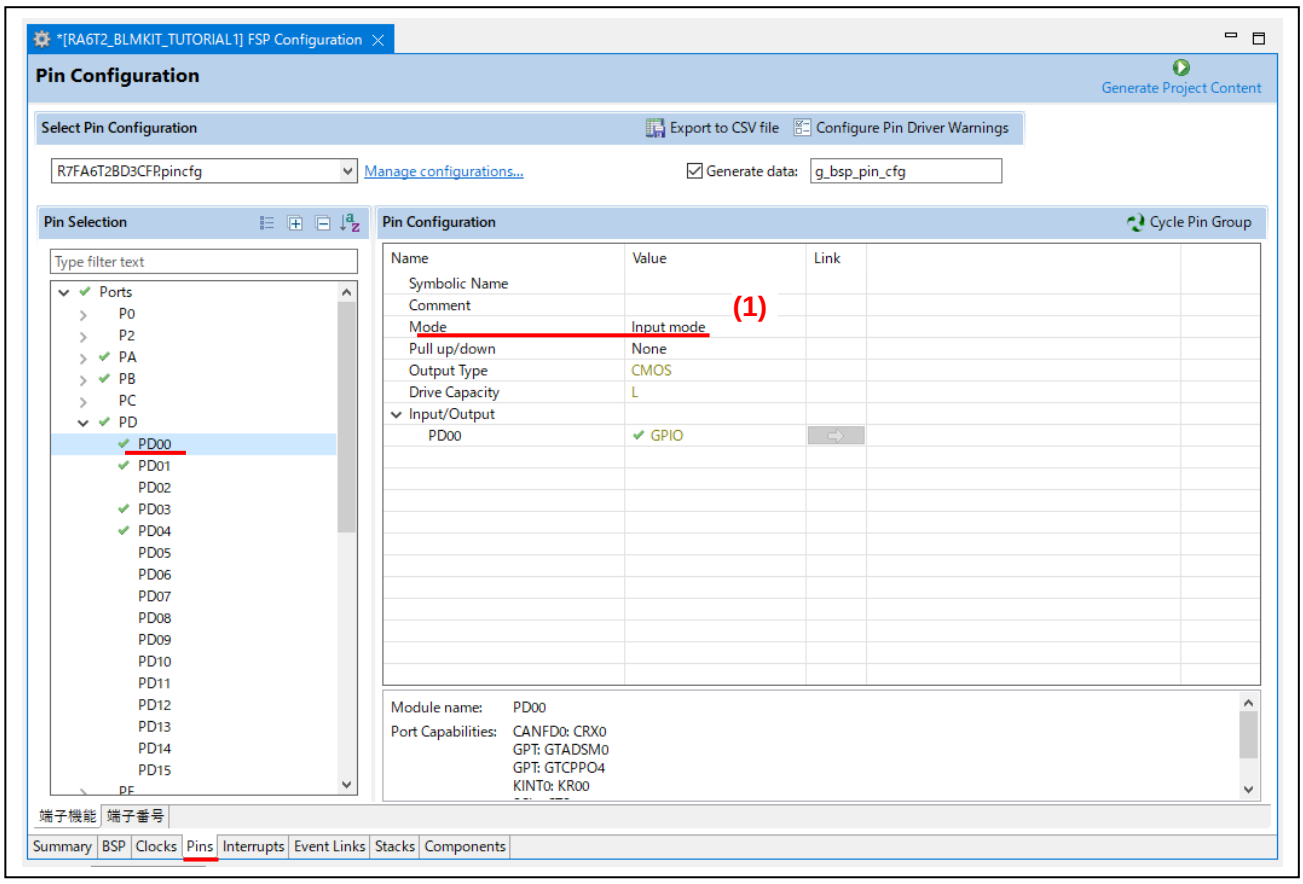
・主要なクロック設定値

ICLK	[MHz]	クロック種
ICLK	240	CPU コアクロック
PCLKA	120	周辺クロック
PCLKB	60	周辺クロック
PCLKC	60	A/D 変換クロック
PCLKD	120	タイマクロック(*1)
SCISPICLK	120	シリアル通信クロック
GPTCLK	200	タイマクロック(*1)

(*1)モータ制御に使用する GPT タイマは、PCLKD=120MHz と GPTCLK=200MHz のどちらかを選択して使用します
※チュートリアル 2 以外では基本的に、GPTCLK=200Mz を選択しています

ここでは、RA6T2 マイコンの設定周波数を最大値で動かす場合の設定をしています。

設定後、Generate Project Content のボタンを押すと、設定した項目に従い、クロック設定のコードが自動生成されます。(他にも設定項目がありますので、ここでは Generate Project Content のボタンを押さずに、次へ進んで良いです。)

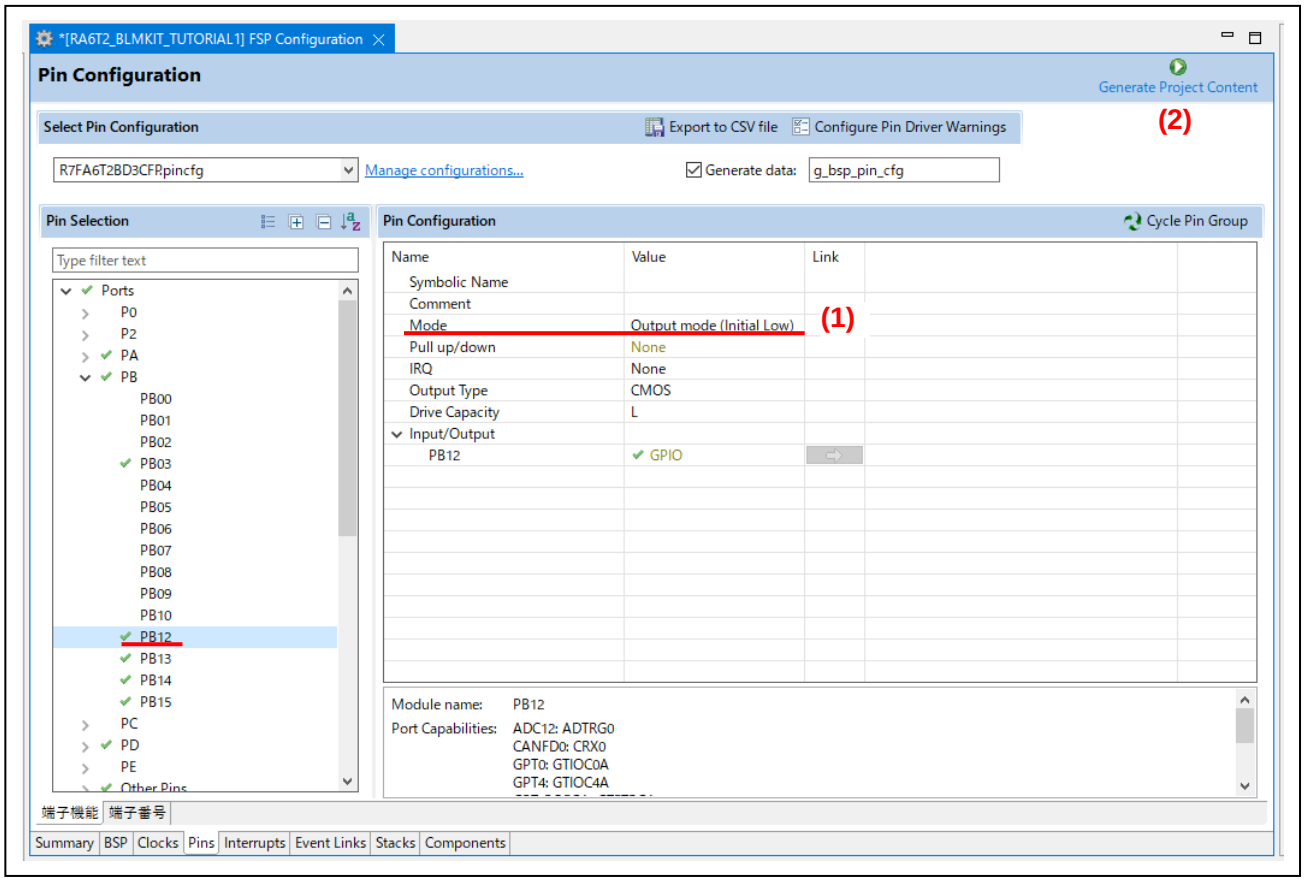


「Pins」タブを選択。
(1)PD00(SW1)を選択し、Mode を Input mode に設定する

PD01(SW2), PD03(SW2), PD04(SW2)も同様に設定。

モータインタフェースボードで、PD00, PD01, PD03, PD04 がスイッチ(マイコンから見ると入力)に接続されていますので、これらを入力端子に設定します。

(マイコンボード上のスイッチを使用する場合は、PD08(マイコンボード SW2)も入力に設定)



(1)PB12(LED1)を選択し、Mode を Onput mode(Initial Low) に設定する

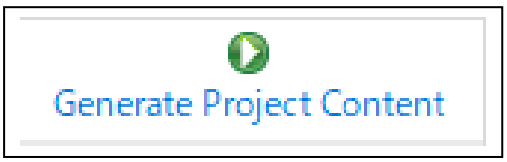
LED は H 出力時点灯なので、Initial Low に設定します。(起動後は LED が消灯)

PB13(LED2), PB14(LED3), PB15(LED4)も同様に設定。

モータインタフェースボードで、PB12~PB15 が LED(マイコンから見ると出力)に接続されていますので、これらを出力端子に設定します。

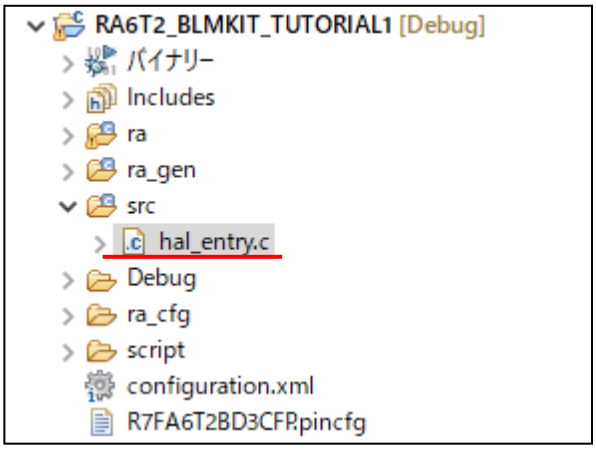
※Drive Capacity は L の設定で問題ありません
(L:2mA, M:4mA, H:10mA, HC:10mA の設定です。LED は、1mA 程度の駆動電流なので、Low 設定で十分です。
※Output type は、CMOS に設定してください(nch open drain では、LED を点灯制御できません)

マイコンボード上の LED(D1)を使う場合は、PD09 も出力設定にしてください。

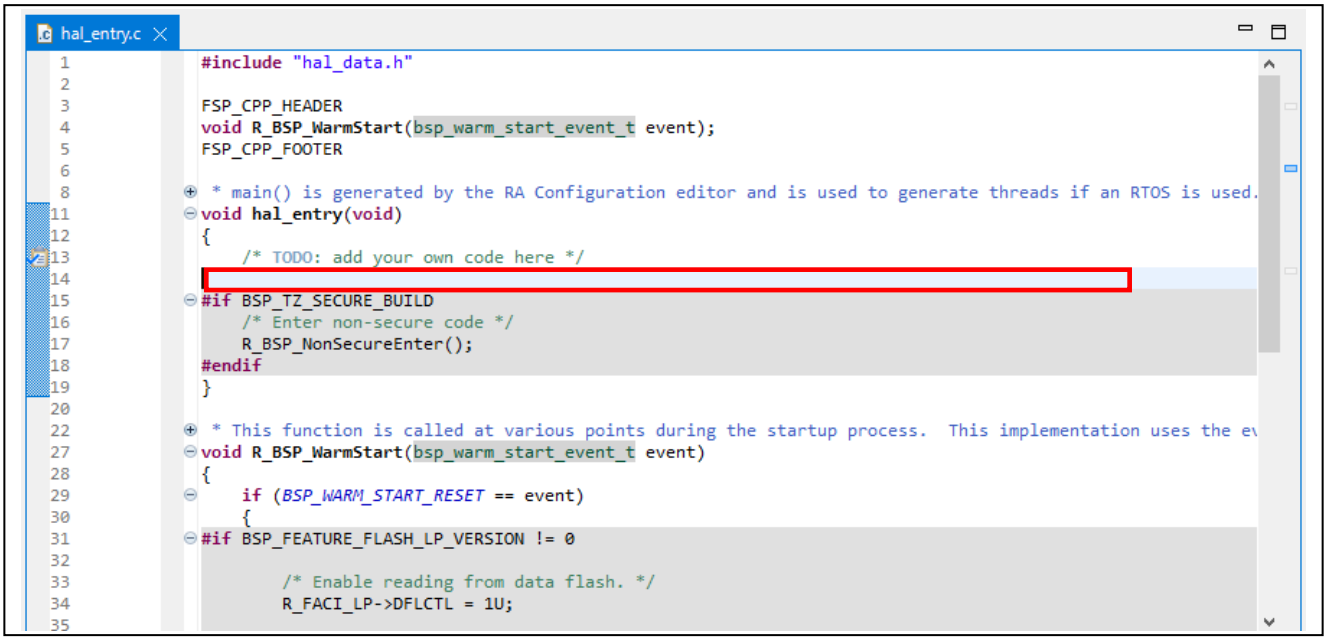


(2)一通り設定が終わった後、Generate Project Content のボタンを押します。

次に、ユーザプログラム本体を書き下します。



src の下の hal_entry.c がユーザプログラムのエントリーポイント（プログラムのスタート地点）を記載するファイルとなります。



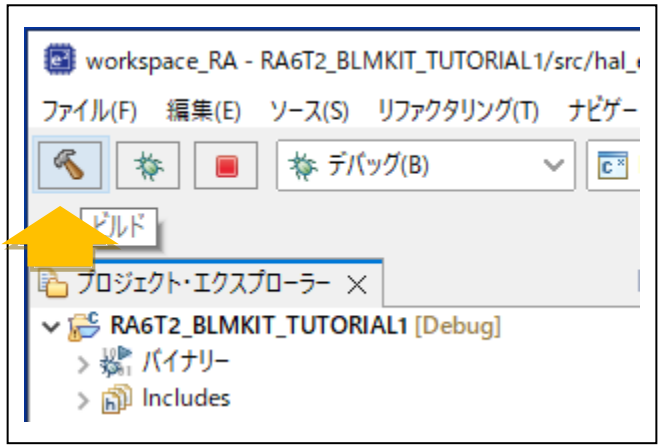
赤枠部分（add your own code here の下）に、ユーザプログラムを追加します。

```
        /* TODO: add your own code here */
#ifndef USE_PCNTR3
    //PODRを使用した制御方法

    //PODRは
    //0を設定:L出力
    //1を設定:H出力
    //となります

    while(1)
    {
        //SW1 PD00 -> LED1 PB12
        if(R_PORT13->PIDR_b.PIDR0 == 0) R_PORT11->PODR_b.PODR12 = 1;
        else R_PORT11->PODR_b.PODR12 = 0;

        //SW2 PD01 -> LED2 PB13
        if(R_PORT13->PIDR_b.PIDR1 == 0) R_PORT11->PODR_b.PODR13 = 1;
        else R_PORT11->PODR_b.PODR13 = 0;

        //SW3 PD03 -> LED3 PB14
        if(R_PORT13->PIDR_b.PIDR3 == 0) R_PORT11->PODR_b.PODR14 = 1;
        else R_PORT11->PODR_b.PODR14 = 0;
```

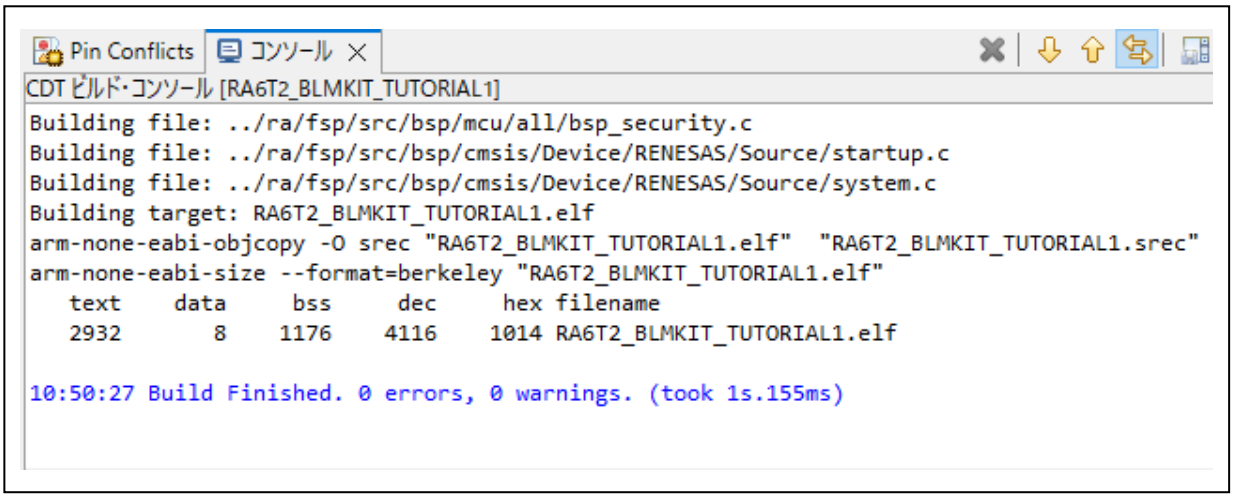
※参考
プログラム内には、PODR を使った制御方法（左記）と、PCNTR3 を使った制御方法が併記されています。
（#define でどちらを使用するか変更可能です）

RA6T2_BLMKIT_TUTORIAL1 は、SW の ON/OFF で LED を ON/OFF させるサンプルプログラムになっています。（SW1~SW4 が LED1~LED4 に対応）

プログラムを記載後、



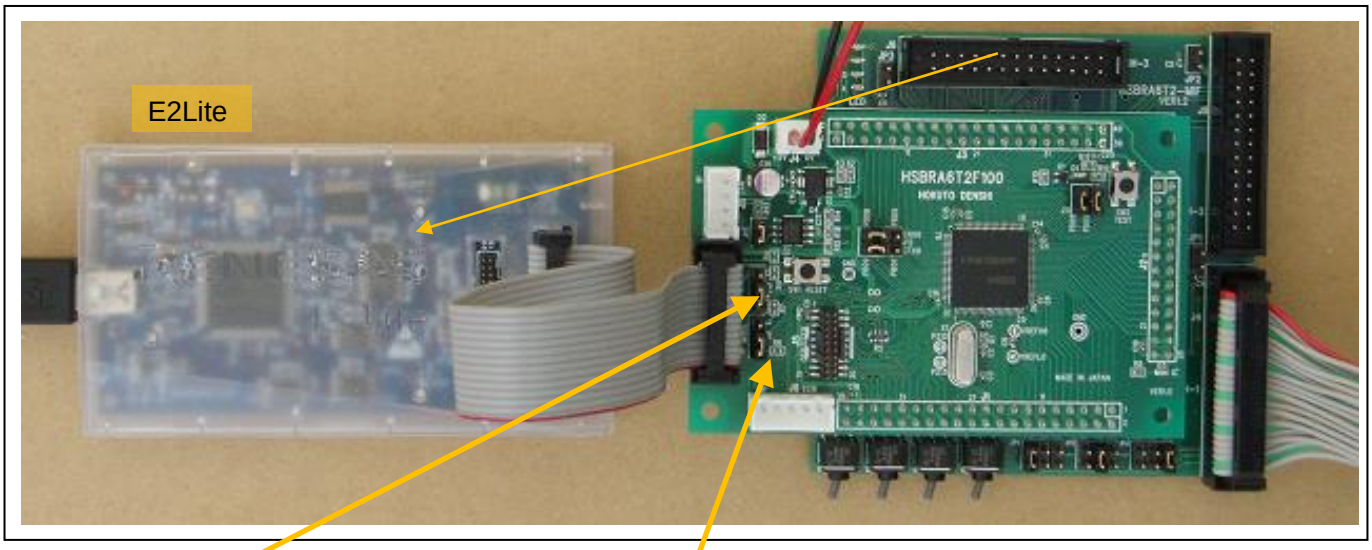
e2studio の Build（トンカチのアイコン）を押します。



e2studio のコンソール上で、処理が進み、0 errors となれば問題ありません。

次に、ビルドしたプログラムをマイコンボード（正確には、マイコンボード上に搭載されているマイコンチップ）に書き込む必要があります。書き込みは、デバッガ（ルネサス E2, E2Lite）か USB-Serial 変換機器を使って行います。

ーデバッガ（E2, E2Lite）を使用してプログラムの書き込み（デバッグ実行）ー



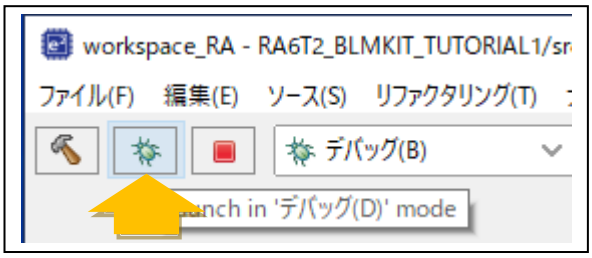
J10 ジャンパ

E2 の場合、上 2 ピンをショート
E2Lite の場合、下 2 ピンをショート

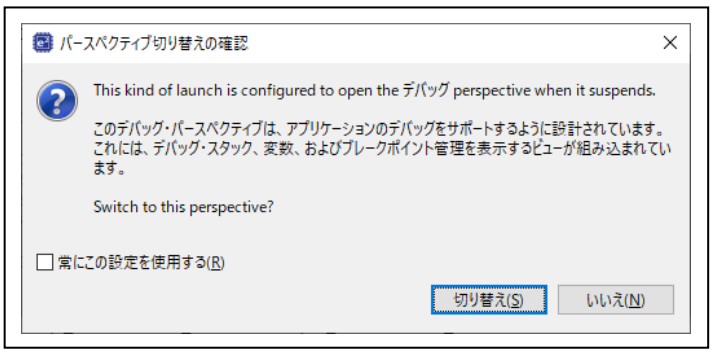
J9 ジャンパ

下 2 ピンをショート
（デバッグ、及び E2/E2Lite で書き込みの設定）

ルネサス E2 または E2Lite をお持ちの場合は、E2（もしくは E2Lite）をマイコンボード、J7(14)に接続します。J10 ジャンパを、E2 の場合上側、E2Lite の場合下側ショートに設定。

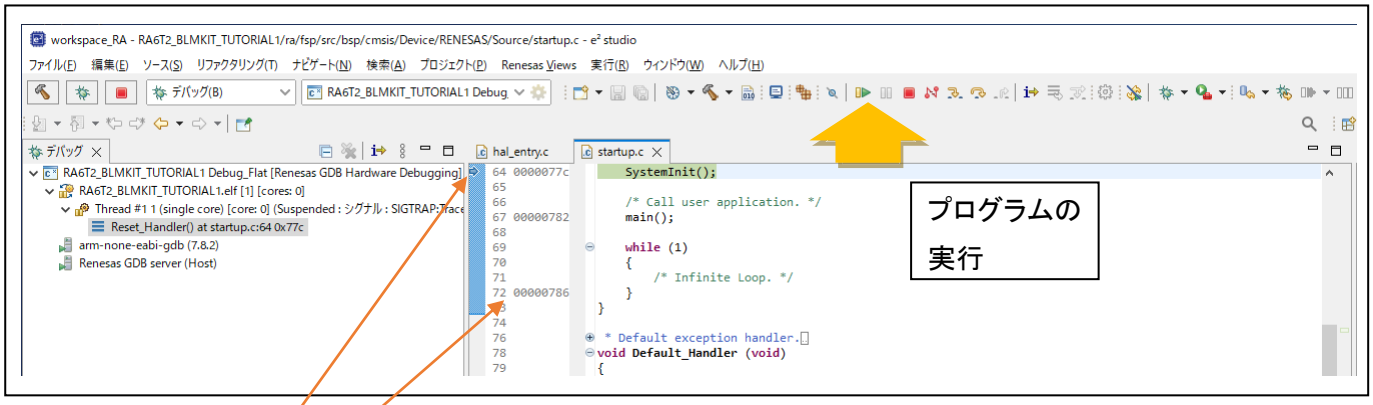


e2studio の Debug（虫のアイコン）を押します。



上記ダイアログが表示された場合は「切り替え」を押す。

**サンプルプロジェクトでは、E2Lite デバッガの設定となっています。
E2 デバッガを使用する際は、後述のデバッガの設定で、使用するデバッガを変更してください**



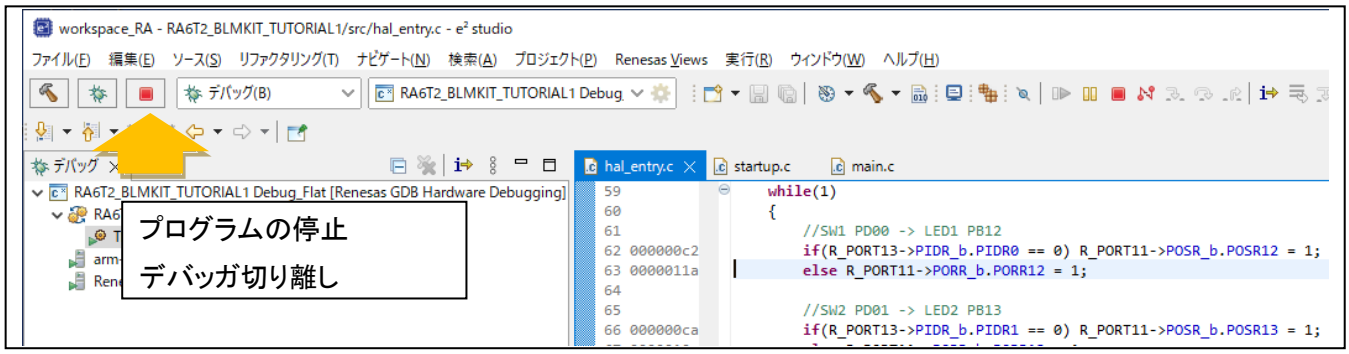
プログラムの停止位置
アドレスが表示される

プログラムのダウンロード（コンパイル・ビルドしたプログラムがマイコンチップの ROM に書き込まれる）が成功すると、上記の様な画面となります。

右上の右三角のアイコンを押すとプログラムは実行されます。

（hal_entry()で一度停止しますので、もう一度プログラムの実行ボタンを押してください）

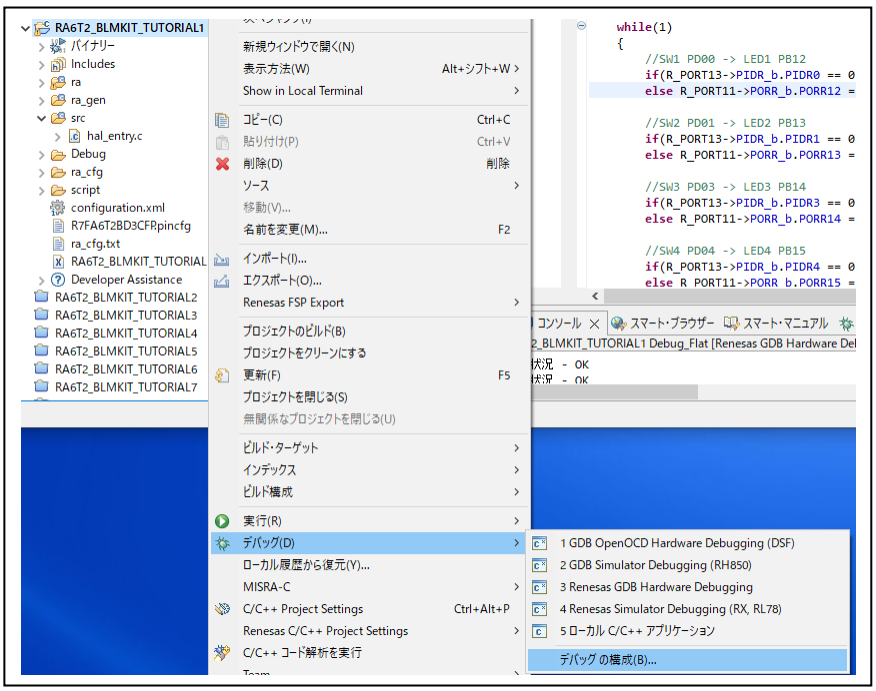
この状態でプログラムは実行されていますので、SW1~4 を切り替えてみてください。SW を ON 側に倒すと、対応する LED1~4 が点灯となれば、プログラムは期待通りの動作となっています。



赤の四角のアイコンを押すと、プログラムは停止となり、デバッガ接続も切り離されます。E2/E2Lite の取り外しや、電源を落とすことが可能となります。

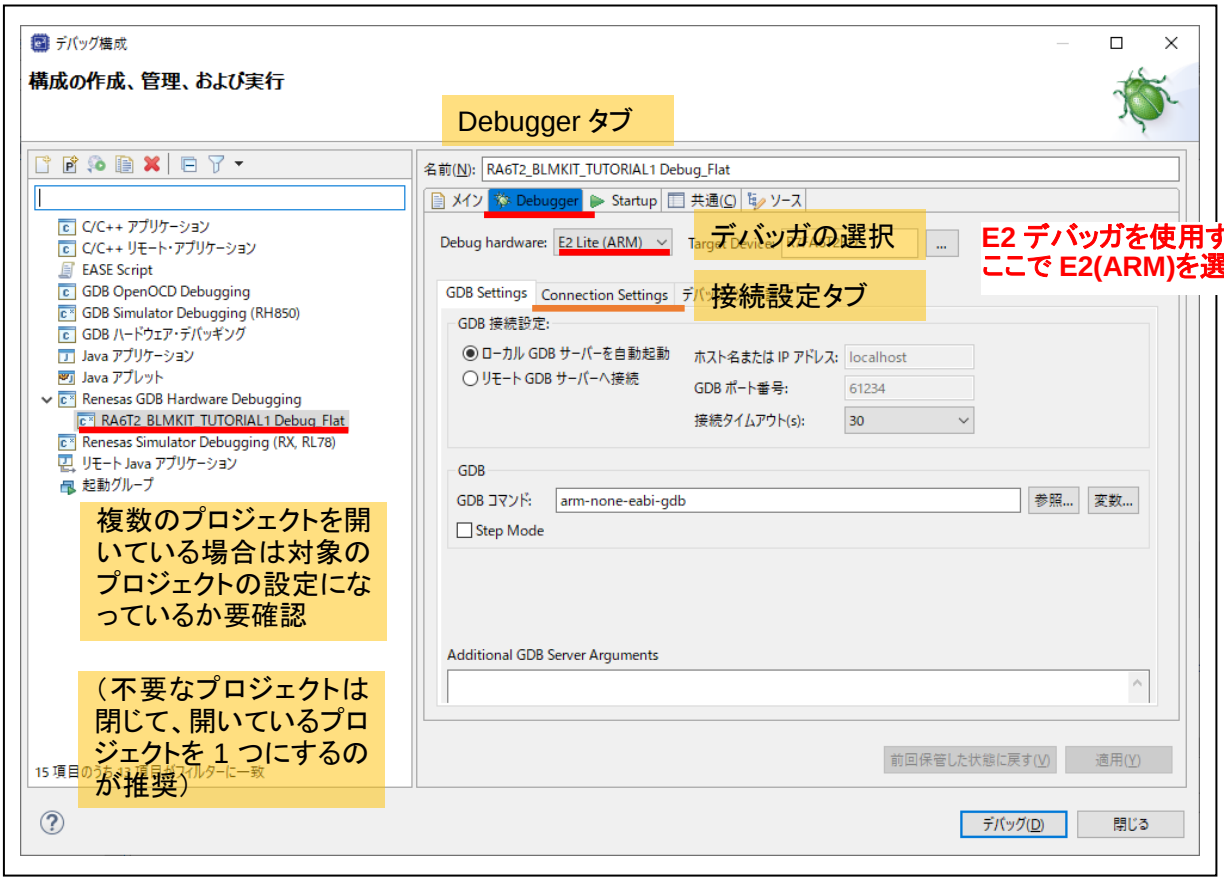
[参考]

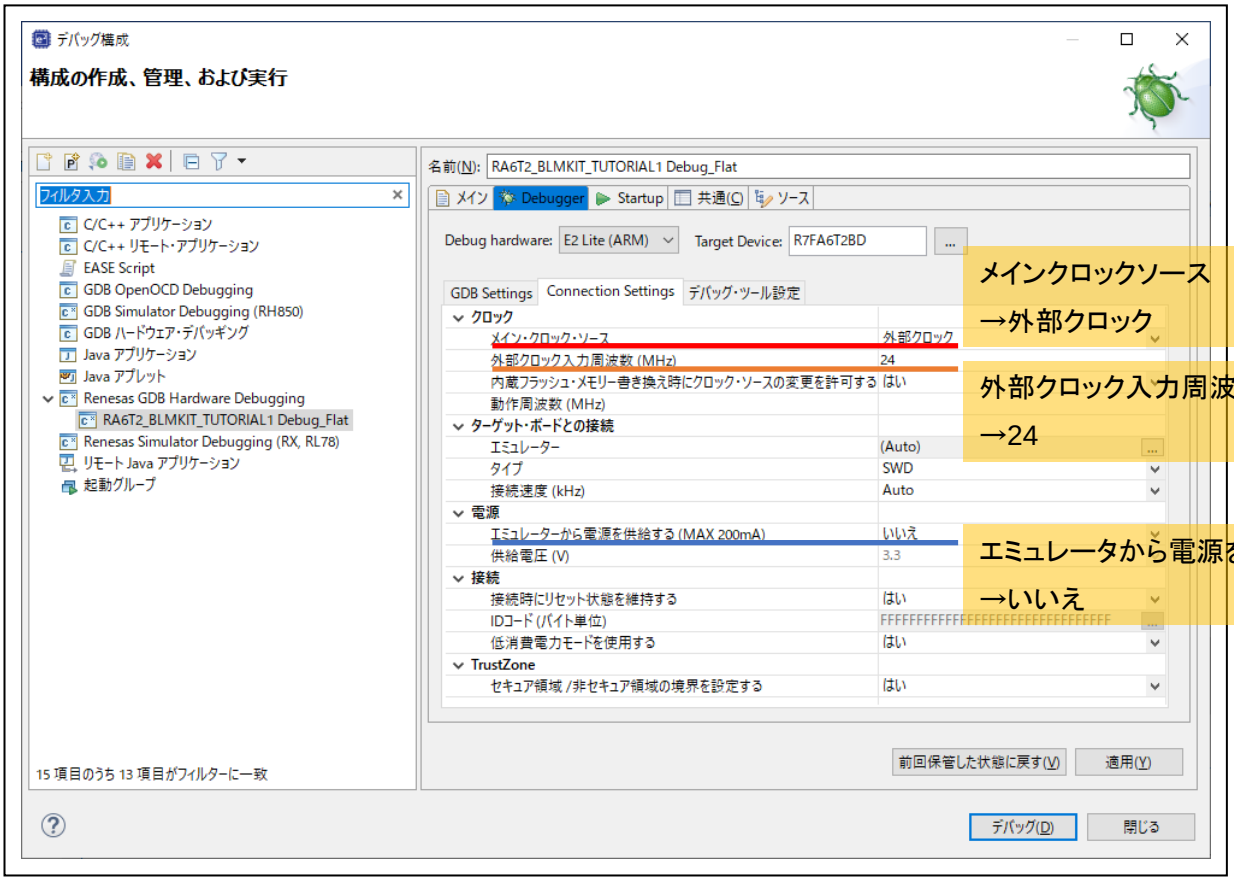
新規に作成したプロジェクトで、デバッガ接続の設定を行う場合は、以下の手順で行って頂きたく。



プロジェクトを右クリック

デバッグ―デバッグの構成





メインクロックソース
→外部クロック

外部クロック入力周波数
→24

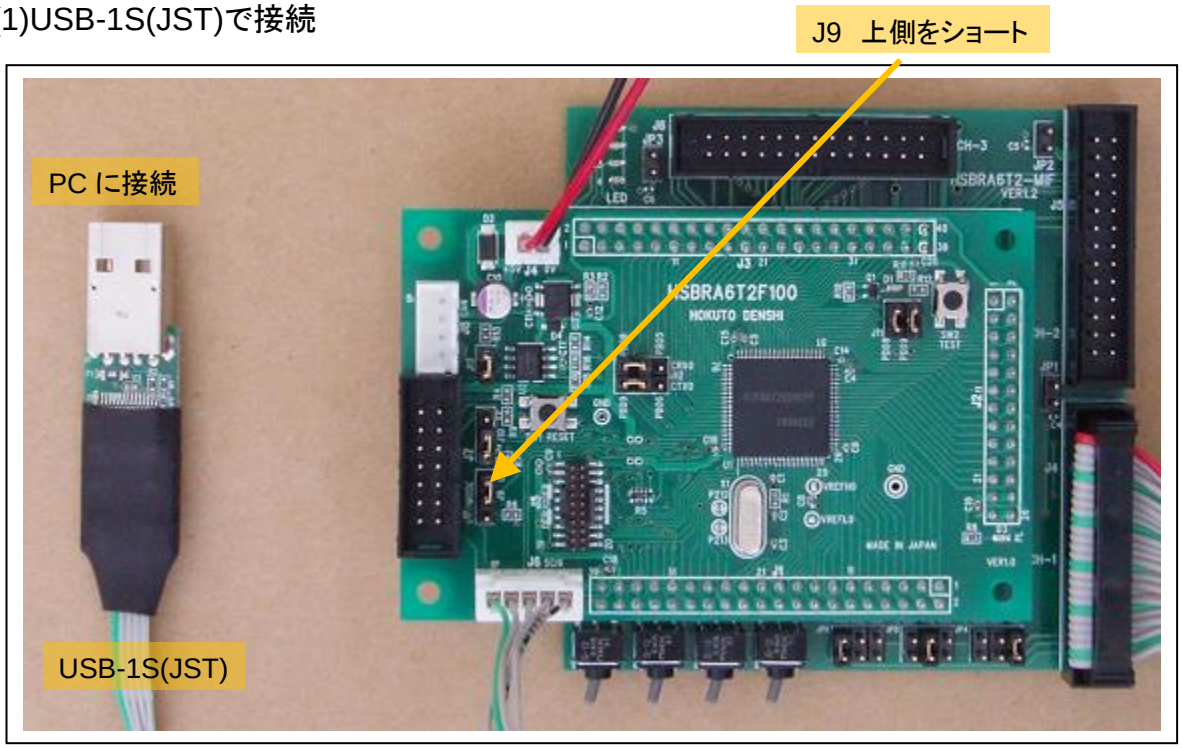
エミュレータから電源を供給する
→いいえ

接続設定を上記の様に変更してください。

(1)J6 に USB-1S(JST)を接続
(2)J7 に USB-ADAPTER-RX14 を接続
(3)USB-Serial 変換機器を J9 または J7 に接続
(4)E2(または E2Lite)を J7 に接続

(1)~(4)のいずれかで PC とマイコンボードを接続する。

(1)USB-1S(JST)で接続

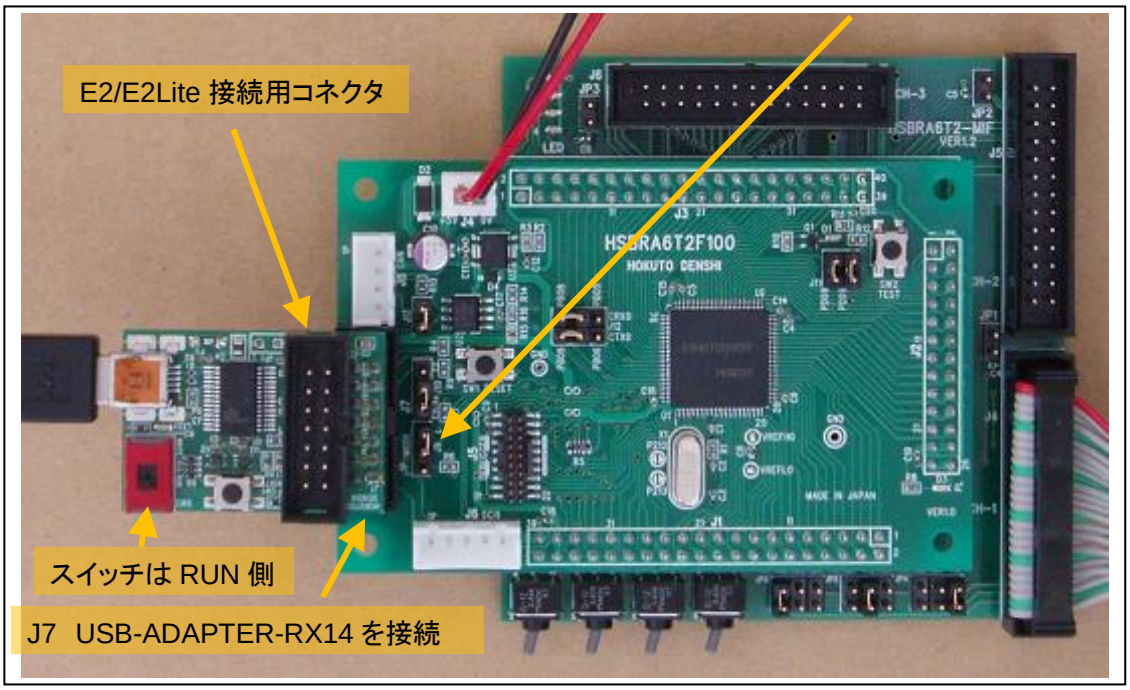


※USB-1S(JST)は、別売品です

J9 を 1-2 ショート(上側 2 ピンがショートされるように挿す)

※書込み後、プログラム実行時は、J9 を「2-3 ショート(下側 2 ピンをショート)」、または「オープン(ジャンパピンを抜く)」としてください

(2)USB-ADAPTER-RX14 で接続



※USB-ADAPTER-RX14 は、別売品です

J9 を 1-2 ショート（上側 2 ピンがショートされるように挿す）

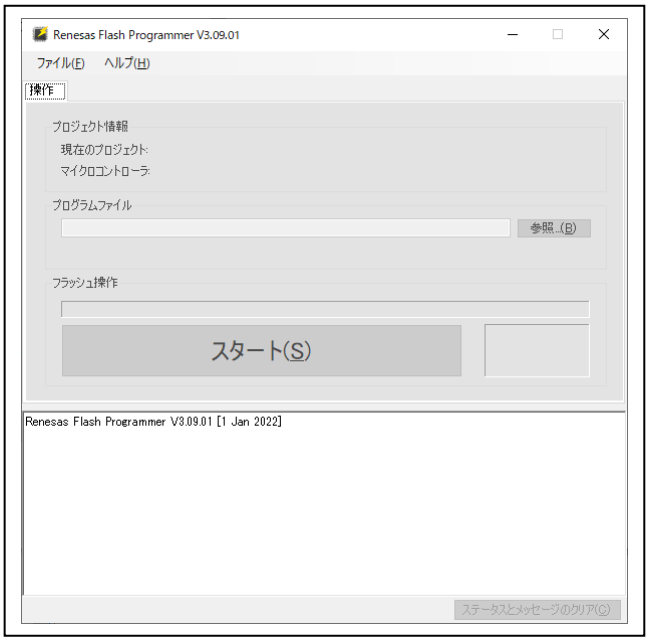
※書込み後、プログラム実行時は、J9 を「2-3 ショート（下側 2 ピンをショート）」、または「オープン（ジャンパピンを抜く）」としてください

USB-ADAPTER-RX14 をマイコンボードに挿したまま、USB-ADAPTER-RX14 上の 14P コネクタに E2, E2Lite を接続可能です。

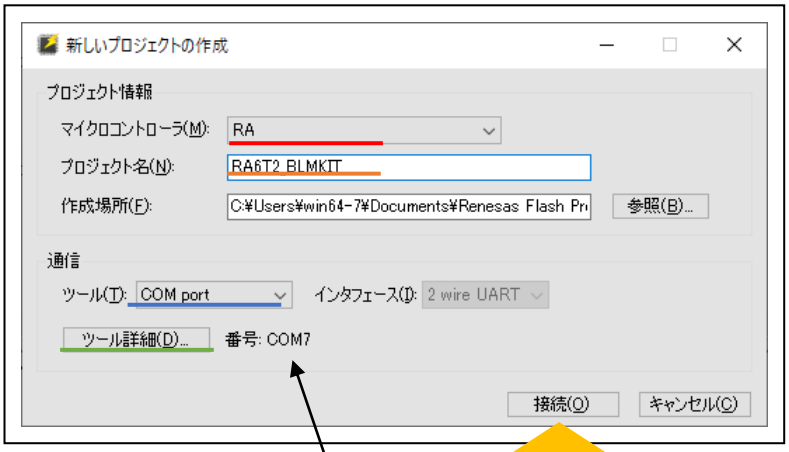
※E2, E2Lite を使って書き込む際は、J9 2-3 ショート（下側 2 ピンをショート）でも問題ありません

※USB-ADAPTER-RX14 と E2 の両方接続する場合は、E2 の接続モードを SWD としてください。E2 の JTAG 接続と、USB-ADAPTER-RX14 の併用はできません。
（E2Lite の場合は接続モードは、SWD しかありませんので接続モードの設定は不要です）

RFP(RenesasFlashProgrammer)を起動。



ファイルー新しいプロジェクトを作成



接続を押す,

USB-1S, USB-ADAPTER-RX14, USB-Serial 変換機器を使用する場合は、予め仮想 COM ポート番号をデバイスマネージャ等で確認してください。

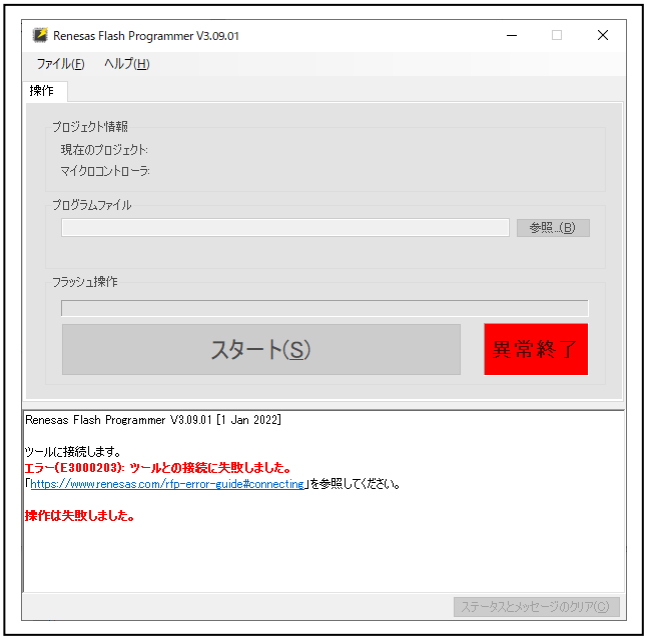
マイクロコントローラ
→RA

プロジェクト名
→任意の名称を入力

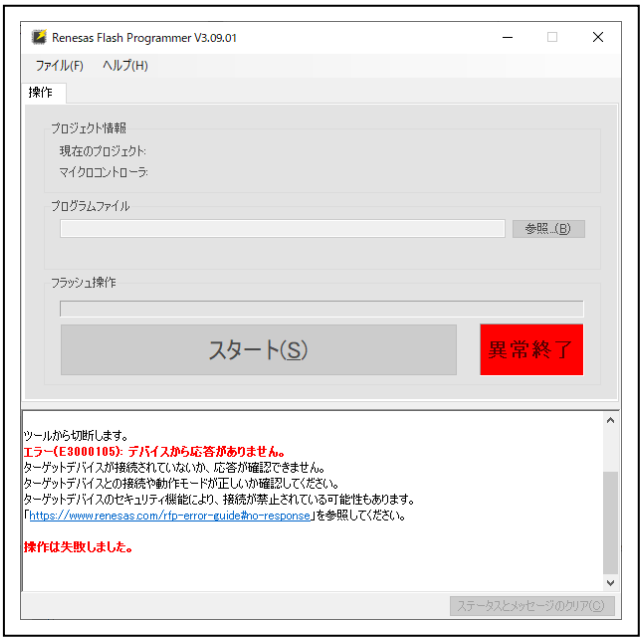
ツール
→COM port を選択
※E2 使用時は E2 emulator
　E2 Lite 使用時は E2 emulator Lite を選択

ツール詳細
→COM 番号を選択
（E2, E2Lite 以外の場合）

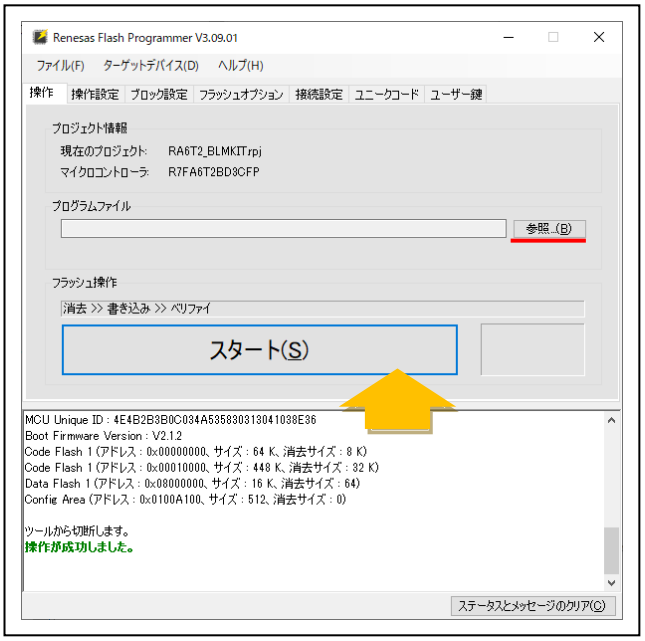
[参考]エラー例



「ツールとの接続に失敗しました」
→選択した COM ポートが使用中、または存在しない
（仮想端末ソフトが開いている等）仮想端末を閉じる



「デバイスから応答がありません」
→J9 の設定ミス、またはリセットされていない
ジャンパの確認後、電源の切断・投入を行う



問題がなければ、操作に成功しましたとなります。

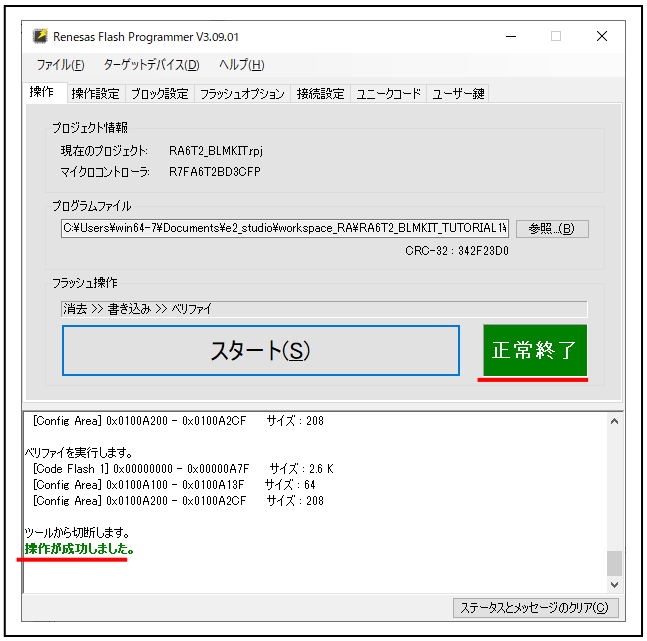
※e2studio のワークスペースフォルダの場所を確認願います（デフォルトから変えていなければ、下記となると思います）

参照で、プログラムファイル
（C:¥Users¥ユーザ名
¥e2_studio¥workspace¥*RA6T2_BLMKIT_TUTORIAL1*¥Debug¥*RA6T2_BLMKIT_TUTORIAL1*.srec）
↑ *プロジェクト名*
を選択。（e2studio のワークスペースフォルダ以下の、「プロジェクト名」¥Debug¥「プロジェクト名」.srec ファイルを指定）

マイコンボード上のリセットボタン(SW1)を押す　　※重要

スタート(S)を押す

「デバイスから応答がありません」というエラーが出た場合は、リセットボタンを押す（もしくは電源を一度切断して再投入）してください。



正常終了　操作が成功しました　となれば、問題ありません。プログラムが、マイコンチップの ROM に書き込まれています。

J9 ジャンパを抜きリセット（または電源再投入）
※もしくは J9 ジャンパを下側に切り替える

この状態でプログラムは実行されていますので、SW1~4 を切り替えてみてください。SW を ON 側に倒すと、対応する LED1~4 が点灯となれば、プログラムは期待通りの動作となっています。

本チュートリアルの目的は、簡単なプログラムの作成と、実際にマイコンチップに書き込んで動作させるところまでとなります。

ソースコードを変えてみて、動作が変わるか等確かめてみてください。

動作しない場合は、
・電源が入っているか（マイコンボード上の PL(D3)が点灯しているか）
・マイコンボードが動作モードになっているか（J8 のジャンパが抜けているか、下側ショートになっているか）
・プログラムの書込みで失敗していないか
等を確認してください。

ーコラムー　TUTORIAL1 の PODR と PCNTR3 を使ったポート制御例に関して

TUTORIAL1 では、LED の出力ポートを制御する方法として、2 種類のコードを記載しています。

(1)PODR を使った制御

```
        //SW1 PD00 -> LED1 PB12
        if(R_PORT13->PIDR_b.PIDR0 == 0) R_PORT11->PODR_b.PODR12 = 1;
        else R_PORT11->PODR_b.PODR12 = 0;
```

RX マイコンのプログラムを行ったことのある方なら、良く使うレジスタかと思います。PODR に 0 を書き込むと L 出力で、1 を書き込むと H 出力になるというものです。

※R_PORT13 は Port D(PDxx), R_PORT11 は Port B(PBxx)です

(2)PCNTR3 を使った制御

```
        //SW1 PD00 -> LED1 PB12
        if(R_PORT13->PIDR_b.PIDR0 == 0) R_PORT11->POSR_b.POSR12 = 1;
        else R_PORT11->PORR_b.PORR12 = 1;
```

PCNTR3 は、POSR（ポートのセット）, PORR（ポートのリセット）という 2 つのレジスタで構成されています。POSR に 1 を書き込むと、H 出力。PORR に 1 を書き込むと L 出力になるというレジスタです。

(1)と(2)の相違点としては、赤字の部分が 0 か 1 かのみで、どちらを使用してもそう大差ないかと思います。

ここで、LED1(PB12)を L, LED2(PB13)を H、その他は変えないという制御を考えてみます。

(1)PODR を使った制御

```
unsigned short tmp;

tmp = R_PORT11->PODR;
tmp &= 0xefff;          //bit12(PB12)をL (0xefff = ~0x1000)
tmp |= 0x2000;           //bit13(PB13)をH
R_PORT11->PODR = tmp;
```

色々なやり方はあるかと思いますが、現状のレジスタ値を読み込み、0 にしたいところを&(and)演算で、1 にしたいところを|(or)演算で変更して、書き戻すというやり方です。（リード、モディファイ、ライト）

(2)PCNTR3 を使った制御

```
R_PORT11->PORR = 0x1000;  //bit12(PB12)をL
R_PORT11->POSR = 0x2000;  //bit13(PB13)をH
```

PCNTR3(PORR, POSR)を使った場合は、変更したいところを 1 にするというレジスタアクセスとなるので、この様に記載できます。さらに、PORR, POSR は PCNTR3 として、32bit アクセスも可能なので、下記の記載でも等価です。

(3)PCNTR3 を使った制御（32bit 単位でのアクセス）

```
R_PORT11->PCNTR3 = 0x10002000;
```

上位 16bit に PORR，下位 16bit に POSR を設定すると、1 行で「変更しない」「1 に変更」「0 に変更」の処理が行えます。

RX のプログラムに慣れている方ですと、PODR を使いたくなりますが、PCNTR3 は便利なレジスタですので、本チュートリアルでも積極的に使用しています。

1.2. モータに電流を流す

参照プロジェクト:RA6T2_BLMKIT_TUTORIAL2

（アーカイブファイル:RA6T2_BLMKIT_TUTORIAL2.zip）

をワークスペースに展開してください。1.2 章同様に、マイコンボードにプログラムを書き込んで実行してください。

接続ボード上の SW1~SW3 は OFF 側に切り替えてください。

プログラムを実行すると、1 秒毎に LED1~4 の点灯が切り替わります。

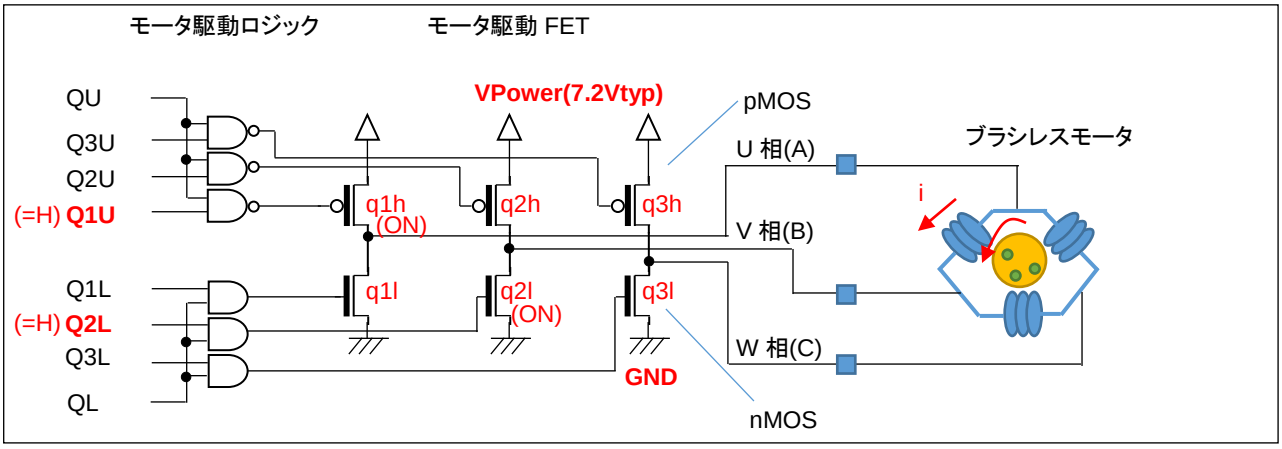
このとき、SW1 を ON すると、CH-1 側に接続したモータドライバボードに、モータに電流を流す信号が送られます。SW1 を OFF にすると、信号は止まります。（同様に、SW2 は CH-2 側、SW3 は CH-3 側を駆動します）

SW を ON とすると、LED が切り替わるタイミングで、モータから「カチッ」という音が聞こえてくると思います。このとき、モータに電流を流す制御を行っています。モータは、U(A), V(B), W(C)の 3 本のワイヤでモータドライバボードとつながっています。

※モータ側では、端子に A, B, C と書かれていますが、以降の解説では U, V, W 相という記載を用います、U=A, V=B, W=C です。

3 本のワイヤに対し、モータドライバボード上で、U に H 側の電源(7.2V)を接続し、V に L 側の電源(GND=0V)を接続した場合 U から V に対して電流が流れます。単純に、3 本のワイヤ(UVW)のうち 2 本をアクティブ（片方を電源、もう一方を GND に接続）とする場合、電流の流れ方としては 6 通りあります。

・U 相から V 相に電流を流す設定



トランジスタの駆動パターンですが、モータに電流を流す際は、

H 側	U 相(q1h)	V 相(q2h)	W 相(q3h)
L 側	U 相(q1l)	V 相(q2l)	W 相(q3l)

合計 6 個のトランジスタの内、H 側 1 箇所、L 側 1 箇所を ON させます。例えば、

q1h と q2l を ON させた場合、Vpower→モータの U 相端子→モータの V 相端子→GND に電流が流れます。…(a)

また、

q2h と q1l を ON させた場合、Vpower→モータの V 相端子→モータの U 相端子→GND に電流が流れます。…(b)

(a)と(b)では、電流が逆方向となります。

q1h と q1l（U 相の H 側と L 側）を ON させる制御は禁止です（モータには電流が流れず、電源間がショートする）。

禁止の組み合わせを省くと、下記の 6 通りとなります。

	(1)	(2)	(3)	(4)	(5)	(6)
H 側	q1h=ON	q1h=ON	q2h=ON	q2h=ON	q3h=ON	q3h=ON
L 側	q2l=ON	q3l=ON	q3l=ON	q1l=ON	q1l=ON	q2l=ON
電流の方向	U→V	U→W	V→W	V→U	W→U	W→V

上記(1)~(6)の様に制御する事により、U, V, W の 3 相の内 2 本にどちらの向きでも電流を流す事が可能です。

本チュートリアルプログラムでは、6 通りの電流を 1 秒毎に切り替えて流すようにしています。

LED	動作
LED1	U 相から V 相に電流を流す
LED2	U 相から W 相に電流を流す
LED3	V 相から W 相に電流を流す
LED4+LED1	V 相から U 相に電流を流す
LED4+LED2	W 相から U 相に電流を流す
LED4+LED3	W 相から V 相に電流を流す

なお、電流は 1 秒間流すわけではなく、LED の点灯パターンが変化した瞬間に 50us の間流す様にしています。

モータに電流を流しているときに、モータの軸に触ると、「カチッ」と音がするタイミングで、わずかに動く感じが指に伝わってくるかと思います。電流が流れる時間は、一瞬のため、モータの軸が動くまではいきませんが、電流を流す時間を増やすとモータの軸の動きが大きくなるというイメージです。また、6 通りの電流の切り替えのタイミング（本チュートリアルでは 1 秒）は、モータの回転数に影響するイメージです。

プログラムでは、

・電流の流す方向の切り替えタイミング(1s)

・電流を流す時間(50us)

を変更する事が出来ます。

blm_main.c 内で、

```
#if 0

    //タイマ周期を変える場合

    timer_info_t info;

    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    unsigned long gpt0_period_counts = info.period_counts;

    (void) R_GPT_InfoGet(&g_timer1_ctrl, &info);
    unsigned long gpt1_period_counts = info.period_counts;

    //GPT0(50us)を変更
    //※長時間ONするとモータのコイルに過大な電流が流れますので、200us以下程度が目安
    const float gpt0_period_time = 200e-6f; //200[us]
    gpt0_period_counts = (unsigned long)((float)gpt0_period_counts * (gpt0_period_time / 50.0e-6f));
    (void) R_GPT_PeriodSet(&g_timer0_ctrl, gpt0_period_counts);

    //GPT1(1s)を変更
    const float gpt1_period_time = 100e-3f; //100[ms]
    gpt1_period_counts = (unsigned long)((float)gpt1_period_counts * gpt1_period_time);
    (void) R_GPT_PeriodSet(&g_timer1_ctrl, gpt1_period_counts);

#endif
```

・「#if 0」 →「#if **1**」に変更

・電流を流す時間：200e-6f の部分

・電流の切り替えタイミング：100e-3f の部分

上記部分を変えると、タイミングを変えることができますので試してみてください。

（50us の方は、あまり大きな値にしないでください。～200us 以下ぐらいに設定する事を推奨致します。）

（※モータ内部はコイル(インダクタンス）で構成されており、長時間（＝DC 的に）電圧を印加すると、コイルのインピーダンスが下がり過大な電流が流れるためです）

本プログラムでは、2 つのタイマ（50us と 1s）を使っています。50us の方は、GPT0, 1s の方は GPT1 です。

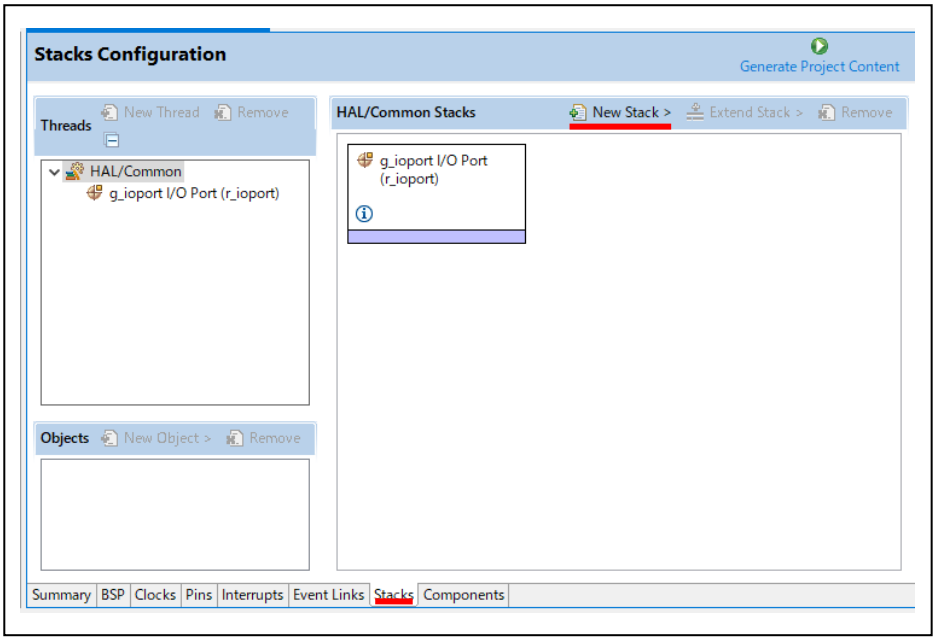
タイマ	用途	設定時間	クロック源	分解能	カウンタ
GPT0	通電時間	50us	PCLKD=120MHz	8.3ns	32bit
GPT1	電流切り替わりタイミング	1s	PCLKD=120MHz	8.3ns	32bit

GPT タイマは、汎用的に使えるタイマで、カウンタは 32bit です。クロック源は、120MHz の PCLKD か、200MHz の GPTCLK から選択できます（それらの周波数を分周してタイマに入力する事も可能です）。

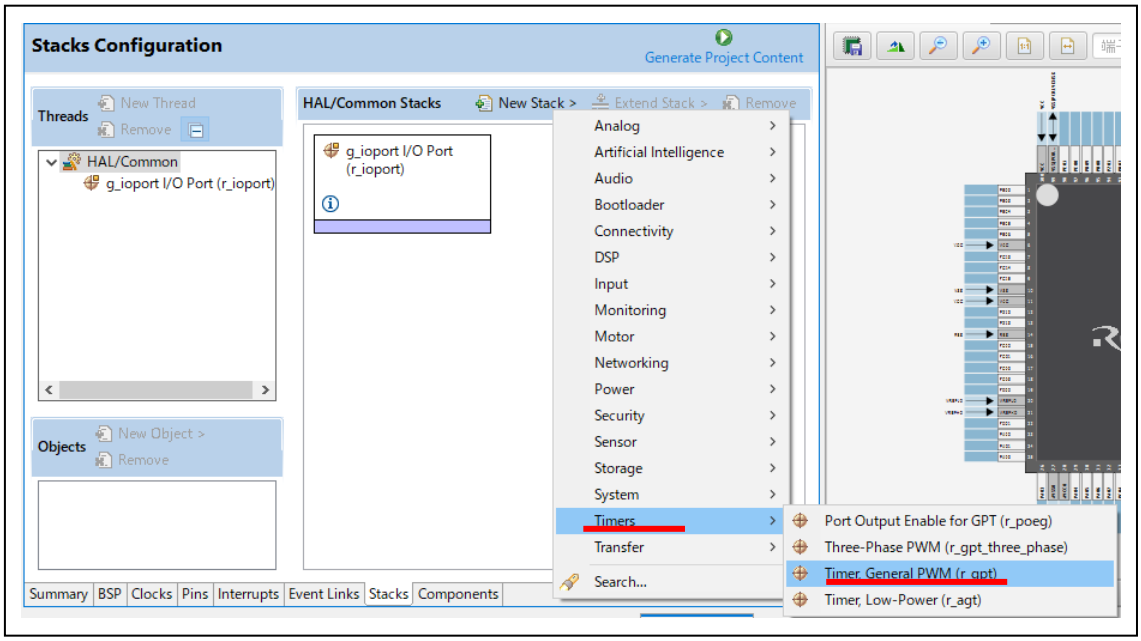
モータ制御プログラムでは、タイマは必ず使用する機能となりますので、マイコンのハードウェアマニュアルを参照し、タイマの分解能（1 カウントの時間）や、設定範囲から、適切なタイマを選択していく事となります。
（本チュートリアルでは、GPT を使っていますが、以降のチュートリアルでは別なタイマも使用しています。）

モータに電流を流す処理は、電流の方向を設定後、モータに通電し、タイマ(GPT0=50us)時間経過後に電流を止めるというものです。

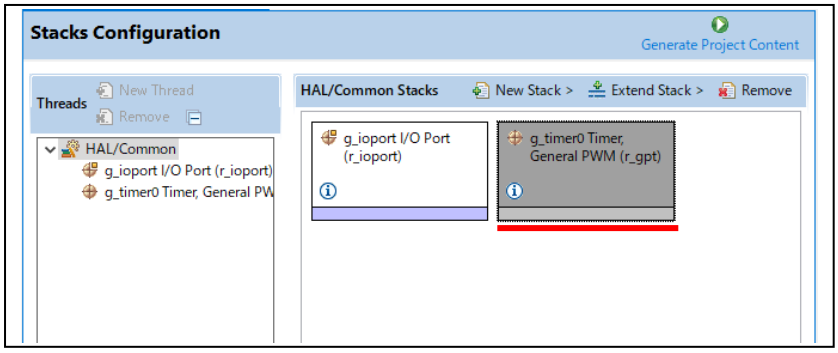
タイマ(GPT)の設定は、FSP を使用して行っています。

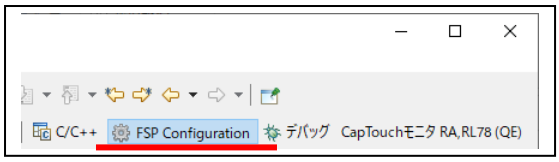


FSP の設定（プロジェクトエクスプローラの Configuration.xml，歯車のアイコン、1.1 でクロックの設定を行った場所）で、Stacks タブ NewStack を押す。



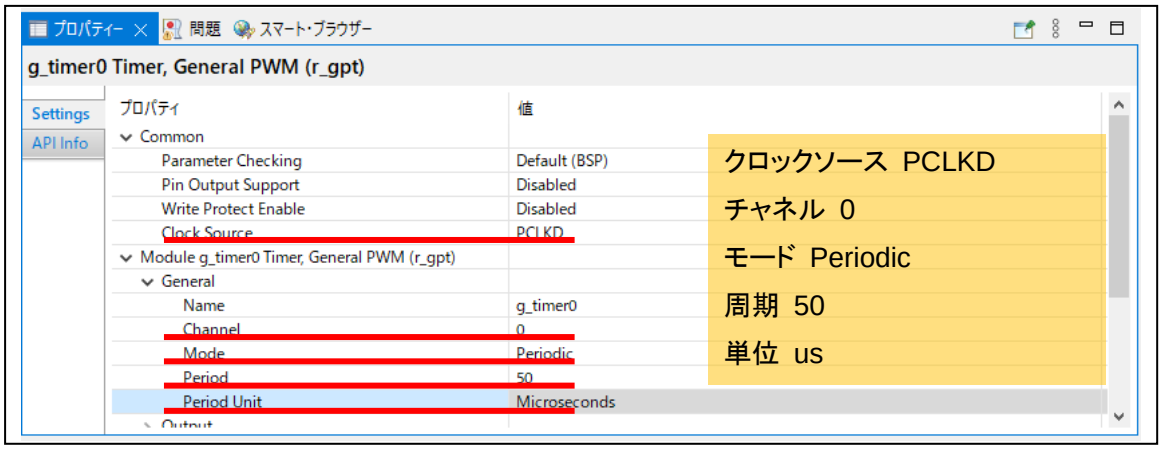
Timers - Timer General PWM(r_gpt) を選択





e2studio のペイン切り替えで
FSP Configuration
を選択
（e2studio の画面右上）

追加された、g_timer0 のボックスを選択。



FSP でタイマを使用する場合、GUI で動作モードや周期等設定が行えますので、タイマ機能を制御するプログラムコードを書き下す必要はありません。

上記で設定しているのは初期値ですので、50us や 1s という時間を変える場合、
・前出のプログラムコードを変更する
・初期値を FSP 上で変更する
のどちらでも有効です。

※FSP の設定を変更した場合「Generate Project Content」のボタンを押すのを忘れない様にしてください
（このボタンを押した際に、FSP で生成されるプログラムコードが更新されます）

・src¥blm¥blm_main.c（モータ制御メイン関数=blm_main()内）

```
//1秒に1回行う処理

//モータの通電方向の遷移
switch(loop)
{
    case 0:
        direction = U_V_DIRECTION;
        blm_led_out(0x1);          //LED1=ON
        break;

    case 1:
        direction = U_W_DIRECTION;
        blm_led_out(0x2);          //LED2=ON
        break;

    case 2:
        direction = V_W_DIRECTION;
        blm_led_out(0x4);          //LED3=ON
        break;

    case 3:
        direction = V_U_DIRECTION;
        blm_led_out(0x9);          //LED4,LED1=ON
        break;

    case 4:
        direction = W_U_DIRECTION;
        blm_led_out(0xA);          //LED4,LED2=ON
        break;

    case 5:
        direction = W_V_DIRECTION;
        blm_led_out(0xC);          //LED,LED3=ON
        break;

    default:
        direction = OFF_DIRECTION;
        blm_led_out(0x0);          //LED=OFF
        break;
}

loop++;
if(loop >= 6) loop=0;

//モータの通電パターンの設定
blm_drive_ch1(direction);
blm_drive_ch2(direction);
blm_drive_ch3(direction);
```

枠内のコードは、1 秒（GTP1 で設定した周期）に 1 回実行される

LED1 を点灯

LED2 を点灯

LED4 と LED1 を点灯

0→1→2→3→4→5→0
の繰り返し

通電開始

・src¥blm¥blm_main.c（続き）

```
//モータの通電時間カウント（GPT0，50usタイマスタート）
g_gpt0_flag = false;
(void) R_GPT_Start(&g_timer0_ctrl);

while(1)                                    g_gpt0_flag は
{                                           GPT(50us)の周期終わりでクリアされる変数
    //モータの通電時間ウェイト
    if(g_gpt0_flag == true) break;
}

//モータの通電OFF
blm_drive_ch1(OFF_DIRECTION);   電流を止める処理
blm_drive_ch2(OFF_DIRECTION);
blm_drive_ch3(OFF_DIRECTION);

//タイマ停止
(void) R_GPT_Stop(&g_timer0_ctrl);

//GPT1（1sタイマ）フラグクリア
g_gpt1_flag = false;
```

blm_drive_ch1(direction) ~ blm_drive_ch1(OFF_DIRECTION)

の期間が、モータに電流を流している期間です。

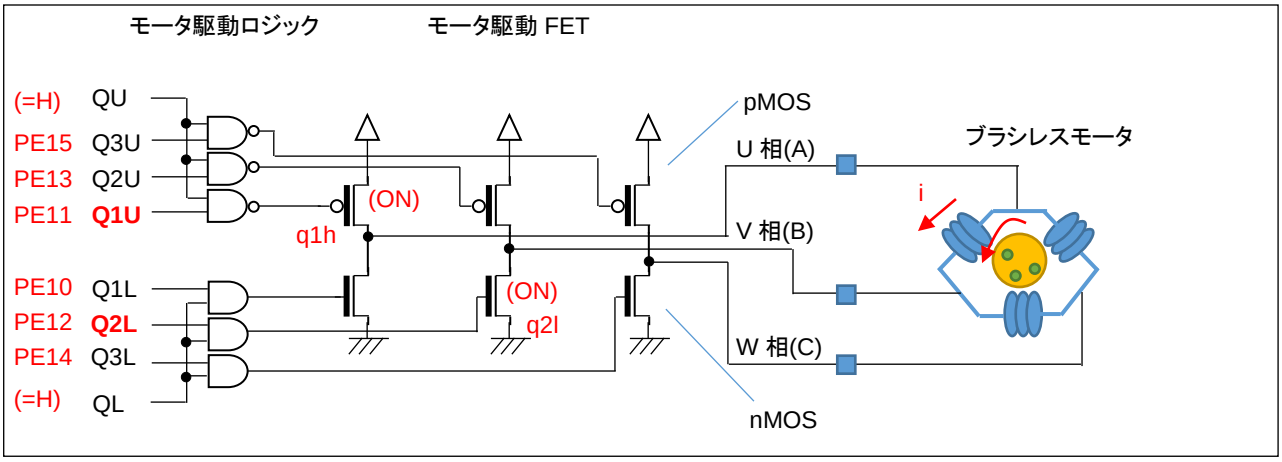
blm_drive_ch?()関数は、モータに引数に応じた方向に電流を流す制御を行う関数です。

・src¥blm¥blm.c

```
void blm_drive_ch1(unsigned short direction)
{
    //ブラシレスモータCH-1制御関数

    //引数：
    // unsigned short direction
    // OFF_DIRCTION  : 電流OFF
    // U_V_DIRECTION : U→Vに電流を流す様制御
    // U_W_DIRECTION : U→Wに電流を流す様制御
    // V_U_DIRECTION : V→Uに電流を流す様制御
    // V_W_DIRECTION : V→Wに電流を流す様制御
    // W_U_DIRECTION : W→Uに電流を流す様制御
    // W_V_DIRECTION : W→Vに電流を流す様制御

    //戻り値：なし

    switch(direction)
    {
        case OFF_DIRECTION:
            //PE11, PE10, PE13, PE12, PE15, PE14 = L
            R_PORT14->PCNTR3 = 0xfc000000;
            break;

        case U_V_DIRECTION:
            //電流をU→Vに流す設定, PE11(Q1U)=H, PE12(Q2L)=H （他はL）
            R_PORT14->PCNTR3 = 0xe4001800;
            break;

        case U_W_DIRECTION:
            //電流をU→Wに流す設定, PE11(Q1U)=H, PE14(Q3L)=H
            R_PORT14->PCNTR3 = 0xb4004800;
            break;

        case V_U_DIRECTION:
            //電流をV→Uに流す設定, PE13(Q2U)=H, PE10(Q1L)=H
            R_PORT14->PCNTR3 = 0xd8002400;
            break;

        case V_W_DIRECTION:
            //電流をV→Wに流す設定, PE13(Q2U)=H, PE14(Q3L)=H
            R_PORT14->PCNTR3 = 0x9c006000;
            break;

        case W_U_DIRECTION:
            //電流をW→Uに流す設定, PE15(Q3U)=H, PE10(Q1L)=H
            R_PORT14->PCNTR3 = 0x78008400;
            break;

        case W_V_DIRECTION:
            //電流をW→Vに流す設定, PE15(Q3U)=H, PE12(Q2L)=H
            R_PORT14->PCNTR3 = 0x6c009000;
            break;

        default:
             break;
    }
}
```

1.1 章のコラムで記載した、PCNTR3 レジスタを使用して、1 行で端子の L/H 設定を行っています

モータドライバボード側では、CH-1 は、PE11, PE10, PE13, PE12, PE15, PE14 の 6 端子で電流を制御する方式です。PE11 と PE12 を H 制御すると、U→V の方向に電流を流す制御となるといった具合です。



PE11 は Q1U につながっていて、U 相の H 側を制御しています。PE12 は Q2L につながっていて、V 相の L 側を制御しています。残りも同様で、6 本の信号線が 6 個のモータ駆動 FET を 1 対 1 で制御する形となっています。PE11 と PE12 を H とすると、q1h, q2l の 2 つの FET が ON し、モータのコイルを経由して図の i の電流が流れます。U→V 以外の定義も同様に、3 相ある電極の 2 相間に電流を流す設定となります。

なお、SW1 を ON にすると、上図の QU=QL=H に制御され、6 本の信号（PE10~PE15）が有効になります。SW1 を OFF にすると、QU=QL=L に制御され、6 本の信号が無効化（PE10~PE15 の信号レベルに拘わらず 6 個のモータ駆動 FET が全て OFF 制御）となります。

本チュートリアルでは、1.1 章で簡単に説明した、PCNTR3 レジスタを使って、6 本の信号線の L, H 出力を切り替えています。

本プログラムでは、モータの軸が一瞬振れる感覚がありますが、回転には程遠いと思います。モータを回転させる制御まで、あといくつかのステップがあります。ここでは、モータ内の任意のコイルの任意の方向に電流を流す事が、プログラムで行える事を理解してください。

本書では、CH-1 の端子制御に関して記載していますが、CH-2, CH-3 も端子名が変わるだけで同様の構成です。

モータドライバボードを、CH-2 につないだ場合は、SW2 で ON/OFF。CH-3 は、SW3 で ON/OFF となり、どの CH でも同じような動作となります。

1.3. A/D 変換と PWM を試す

参照プロジェクト:RA6T2_BLMKIT_TUTORIAL3

（アーカイブファイル:RA6T2_BLMKIT_TUTORIAL3.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

このチュートリアルでは、マイコンの A/D 変換（Analog to Digital 変換）機能と、PWM（Pulse Width Modulation: パルス幅変調）を試してみます。モータの制御から一旦離れますが、どちらもモータを動かすのに必要な機能となります。
本プログラムでは、

・VR の回転角に応じたパルス幅の信号が、QL (CH-1)PC14, (CH-2)PD11, (CH-3)PE01 から出力
・モータドライバボード上の温度センサ（サーミスタ, R54）の値を拾う

という動作を行います。本プログラムでは、情報を SCI9(PB03/TXD9)に、シリアル通信(UART)で出力します。USB-1S(JST)（別売オプション）を、J6 に接続する。USB-ADAPTER-RX14（別売オプション）を、J7 に挿す、または、市販の USB-Serial モジュールを PB03 に接続してください（シリアル通信のモニタは必須ではありませんが、接続した場合、プログラムの動作が判り易くなると思います）。

・シリアル端末から出力される情報（3 秒に 1 回更新）

```
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

RA6T2 / BLUSHLESS MOTOR STARTERKIT TUTORIAL3

Motor driver board connection check...
  CH-1 Connected.
  CH-2 NOT connected.
  CH-3 NOT connected.


                        CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     x         x         x
Temperature(A/D value) :  2155         0         0
Temperature(degree)    :    28         0         0
VR(A/D value)          :   723         0         0
QL duty[%]             :   0.0       0.0       0.0
```

PC 上では、teraterm 等のシリアル端末ソフトで表示してください

115200bps, 8bit, none, 1bit の設定で表示できます

上記は、CH-1 にのみモータドライバボードを接続した場合です。モータドライバボードが接続されていない場合は、各値が 0 となります。

Active は、SW が OFF（CH-1 は SW1, CH-2 は SW2, CH-3 は SW3）の時は、x になります。

ここで、SW1 を ON にします。

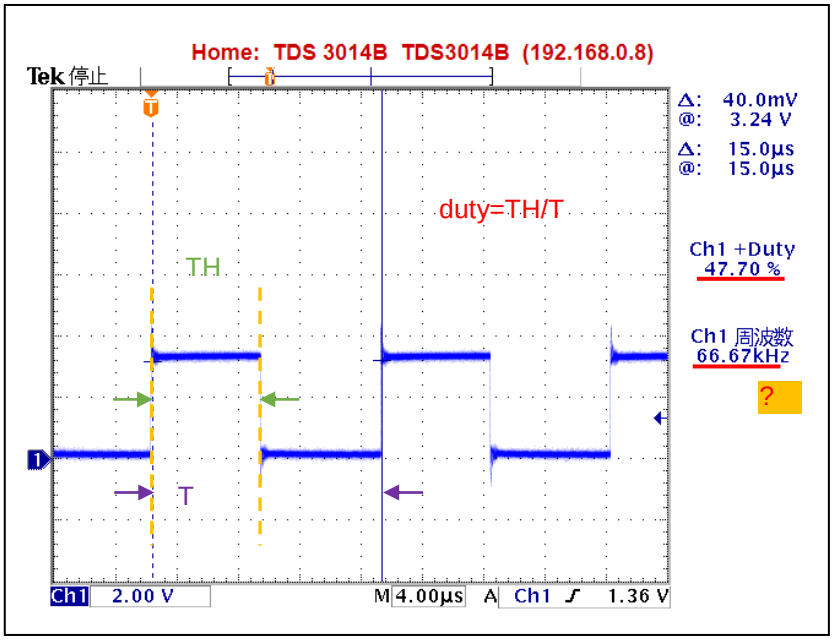
```
                        CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2148         0         0
Temperature(degree)    :    28         0         0
VR(A/D value)          :   261         0         0
QL duty[%]             :   6.4       0.0       0.0

                        CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2157         0         0
Temperature(degree)    :    28         0         0
VR(A/D value)          :  1956         0         0
QL duty[%]             :  47.7       0.0       0.0
```

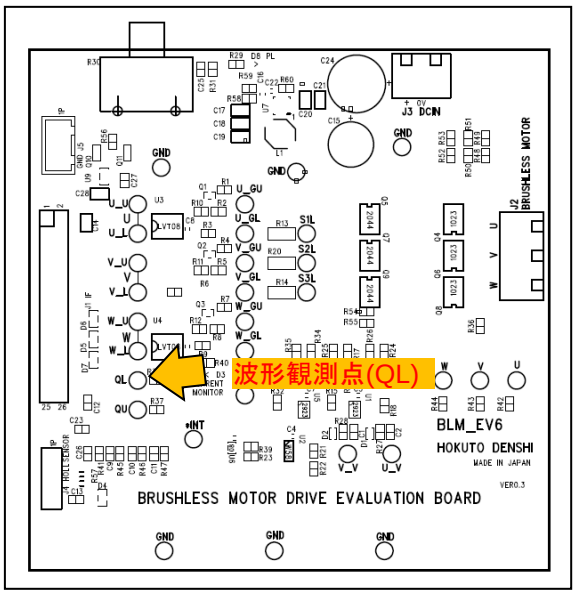
SW を ON にすると、QL duty が 0 からモータドライバボード上の VR に応じた数値に変わります

→実際に PWM 波形が出力されます

上記では、温度センサの A/D 変換値は 2157 で温度に変換すると、28℃。モータドライバボードの、VR の A/D 変換値は 1956 で、duty は 47.7%に設定されているという情報が出力されています。

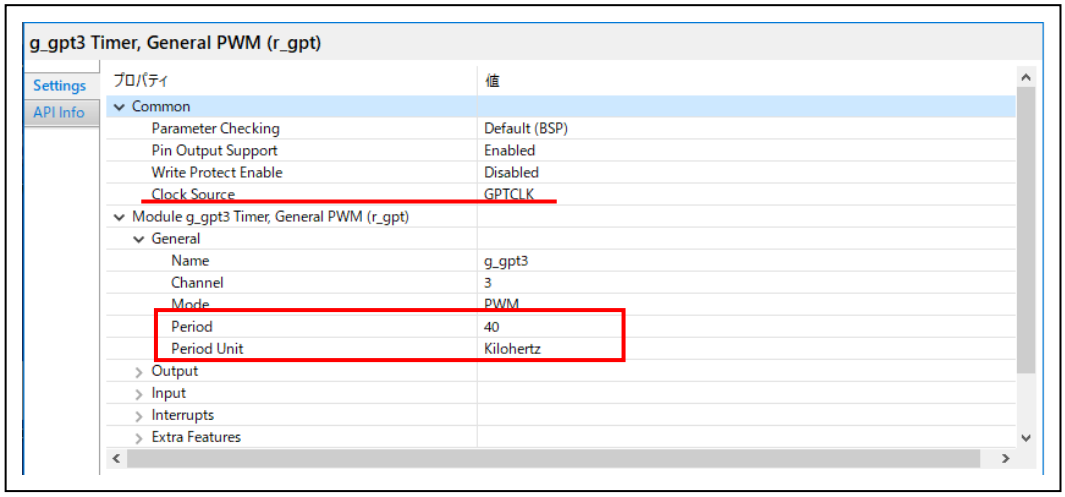
・オシロスコープで QL 端子（モータドライバボード QL 端子）を観測した波形





duty は、VR の回転角度に応じて、0~90%程度の範囲で変化します。QL 波形の周波数は、40kHz です。
（オシロで観測すると、66.67kHz になっています?…次ページに理由を示します）

－GPT の周波数設定に関して－



FSP3.7 では、GPT の周波数設定にバグがあり、40kHz 設定の場合、実際に出力されるのは 66.67kHz となります。

（FSP3.8 で修正予定）

※クロックソースに GPTCLK(200MHz)を選択した場合に設定周波数がずれます、なおチュートリアル 2 では PCLKD を選択していますので設定値（50us と 1s）通りのタイミングとなっています

プログラムでは、PWM のキャリア周波数の設定を 40kHz としています。

→FSP3.7 以下　実際には 66.67kHz で出力

→FSP3.8 以降でコード生成ボタンを押して再ビルドした場合、出力は 40kHz になると思われます

※FSP のバージョン変更後動作が多少変わる事が考えられます

PWM のキャリア周波数は、モータの動作に関してそれ程大きな影響を与えないと考えて、FSP のバージョンにより変わってしまう事は許容しています。FSP3.8 以降でモータ動作の挙動が変わってしまった際は、FSP の GPT の周期値設定を

Period　40　→ 67 （現行バージョンの FSP では、整数値しか入力不可）

に変更すれば、現状と同じ動作となると考えられます。

（もしくは、

Period 15

Period Unit Microseconds

に設定すれば、66.67kHz ジャストに設定可能です。）

Tempatature(A/D value)は、温度センサーの出力です。温度センサーの出力は、信号名では AD6、マイコンの(CH-1)PA6/AN006, (CH-2)PB00/AN008, (CH-3)PE09/AN021 に接続されています。マイコンの当該端子を A/D 変換入力に設定し、A/D 変換機能を使って温度値を取得します。RA6T2 の A/D 変換機能は、12bit となっているので、0~4095 までの値を取ります。(この値は、約 25℃のときに、2048 になります。温度が高いほど数値は大きくなります)

Tempatature(degree)は、温度センサーの A/D 変換値を摂氏温度に変換したものです。温度の変換は、モータドライバボードの取扱説明書に計算式を示してあります。(計算式は、exp の計算を含む手間のかかるものとなっているので、本プログラムでは予め A/D 変換値と温度の 80 点のテーブルを作成しておき、テーブルから温度を換算しています)ここでは、28℃とっていますが、ドライヤー等でモータドライバボードの温度センサ部(R54:黒いヒートシンクの下側付近にある)を暖めると、画面表示上の温度も上昇するかと思います。

VR(A/D value)は、モータドライバボード上の VR(ボリューム)を回すと、値が変わるはずです。時計回りに目一杯回すと 0。反時計回りに目一杯回すと、3722(4095×10/11)近傍になるはずです。VR は、プログラム上で値を拾いアナログ入力デバイスとして、任意の用途で使用する事が出来ます。

QL duty は、QL 端子の duty 値となります。この値は、VR の読み取り値に連動して変更を行っています。本プログラムでは(VR の読み取り値/4095×100=)0~90%程度まで設定可能です。この値と、実際に QL 端子から出力されるパルス波形は連動しています。

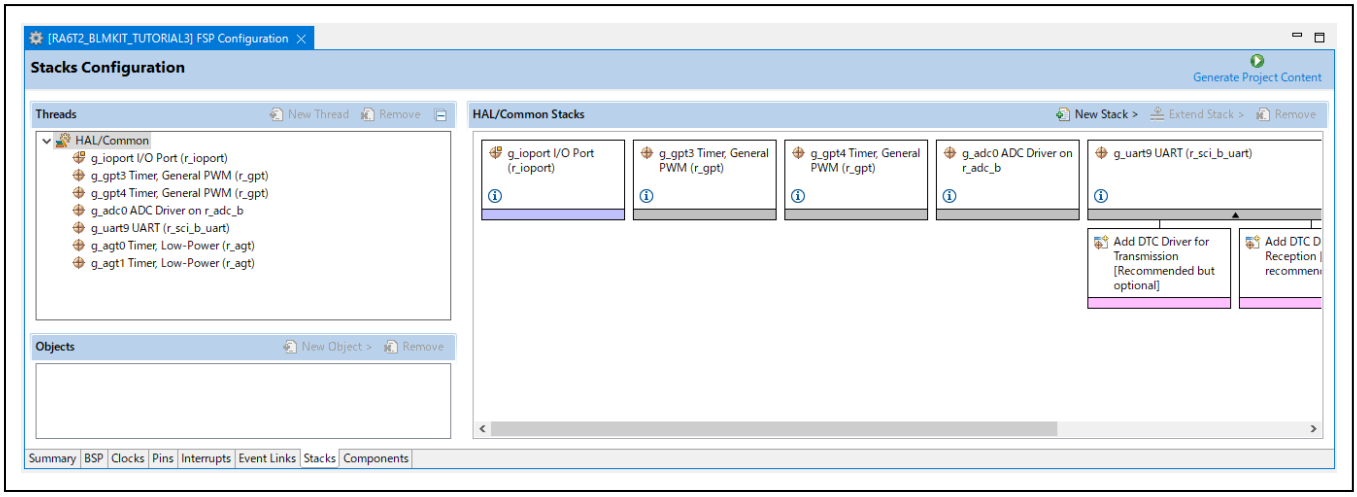
ここでは、VR の読み取り値をプログラムで処理して、QL 端子の duty 比を決めている事に注意願います。

QL 端子は、

CH-1	CH-2	CH-3
PC14/GTIOC3A	PD11/GTIOC3B	PE01/GTIOC4B

上記の端子に接続されており、32 ビットタイマ(GPT)の出力端子に設定する事により、H/L の繰り返し波形(矩形波)出力を得る事ができます。

CH-1 では、GPT3A の duty 設定値を変えると、QL(PC14)に出力される、H パルスの幅が変わります。この様に、パルス幅を変える制御を、PWM(パルス幅変調)といい、モータの制御ではよく使用される方式です。

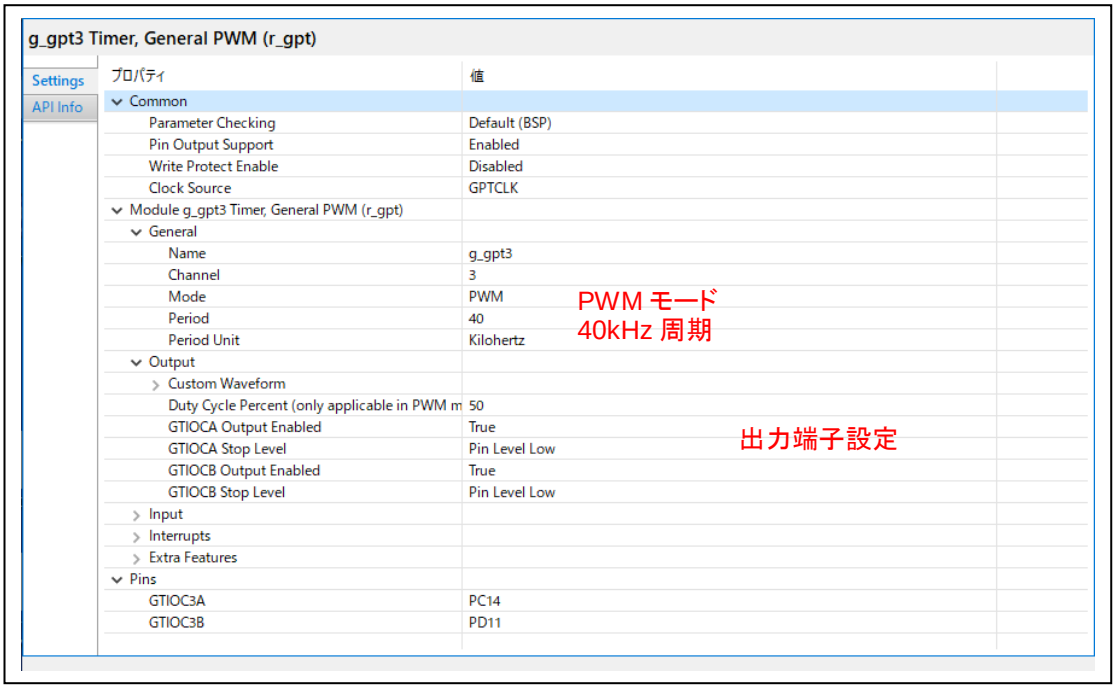


FSP Configuratuon の画面を開き、Stacks でマイコンの使用したい機能を追加していきます。ここでは、

Stack 種	内容	用途	備考
g_ioport	I/O ポート設定	I/O 端子設定	デフォルトで追加済み
g_gpt3	GPT3 タイマ	PWM 波形生成(CH-1, CH-2)	
g_gpt4	GPT4 タイマ	PWM 波形生成(CH-3)	
g_adc0	ADC(ch0)	A/D 変換	
g_uart9	UART	UART 文字表示	
g_agt0	AGT0 タイマ	ADC 起動	50us 周期
g_agt1	AGT1 タイマ	画面表示タイミング	10ms 周期

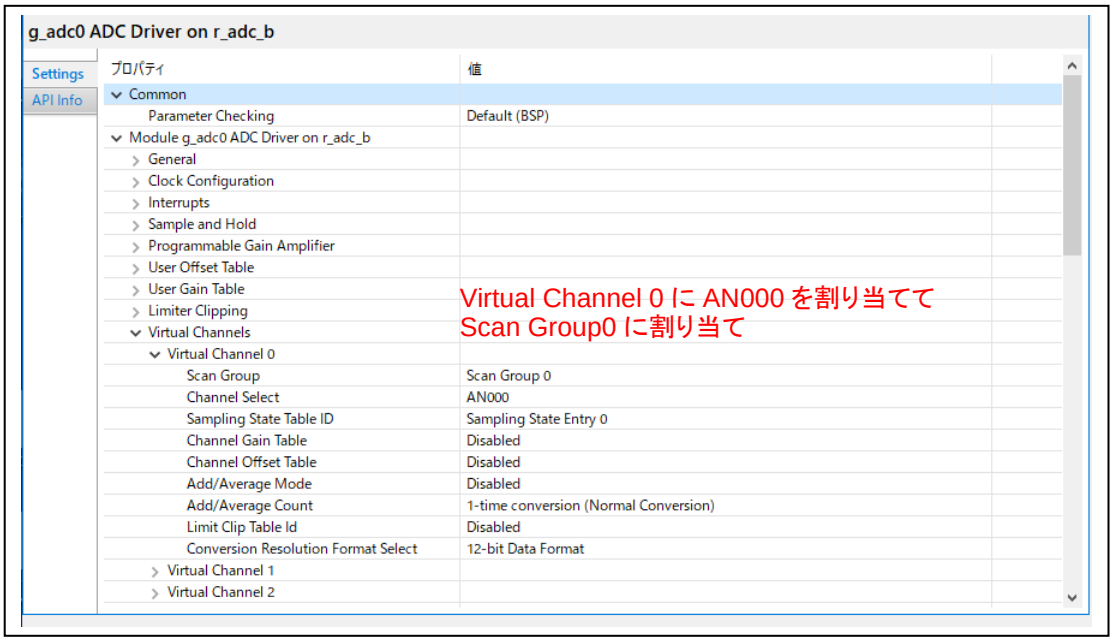
上記の Stack を追加して使用しています。

・g_gpt7(GPT7)設定



PWM モードであることや、周期 40kHz、PC14(CH-1, QL), PD11(CH-2, QL)から出力する設定等を行っています。

・g_adc0(ADC0)設定

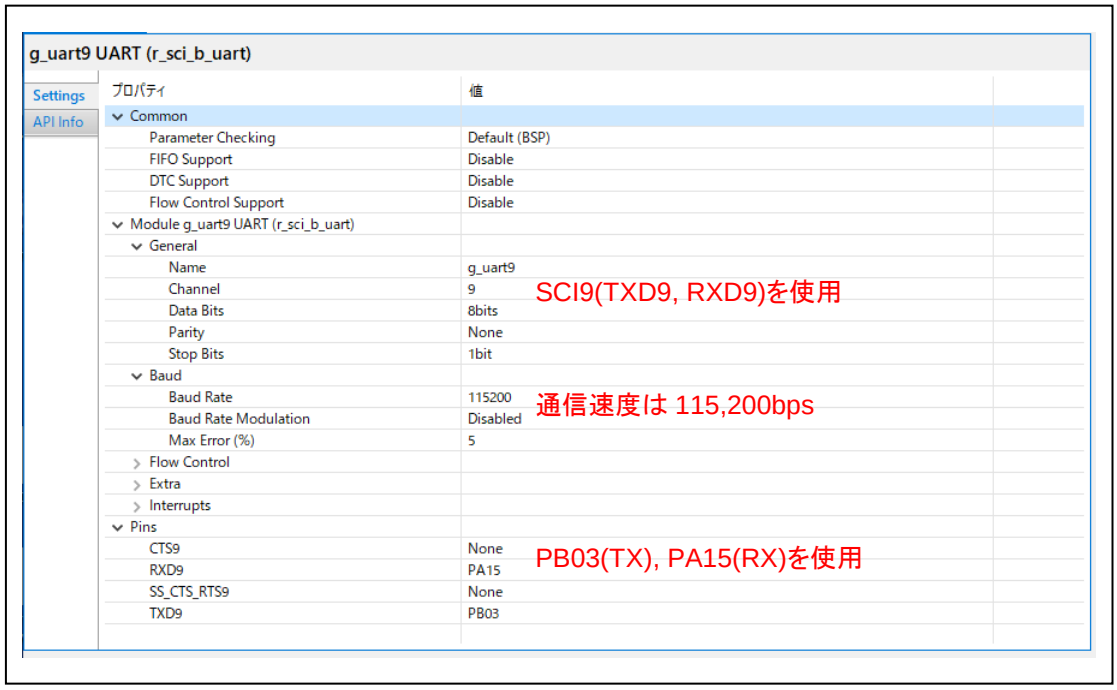


stack の設定で、動作モードや、端子の設定、A/D 変換終了時に呼び出される割り込み関数、割り込み優先度等を設定しています。

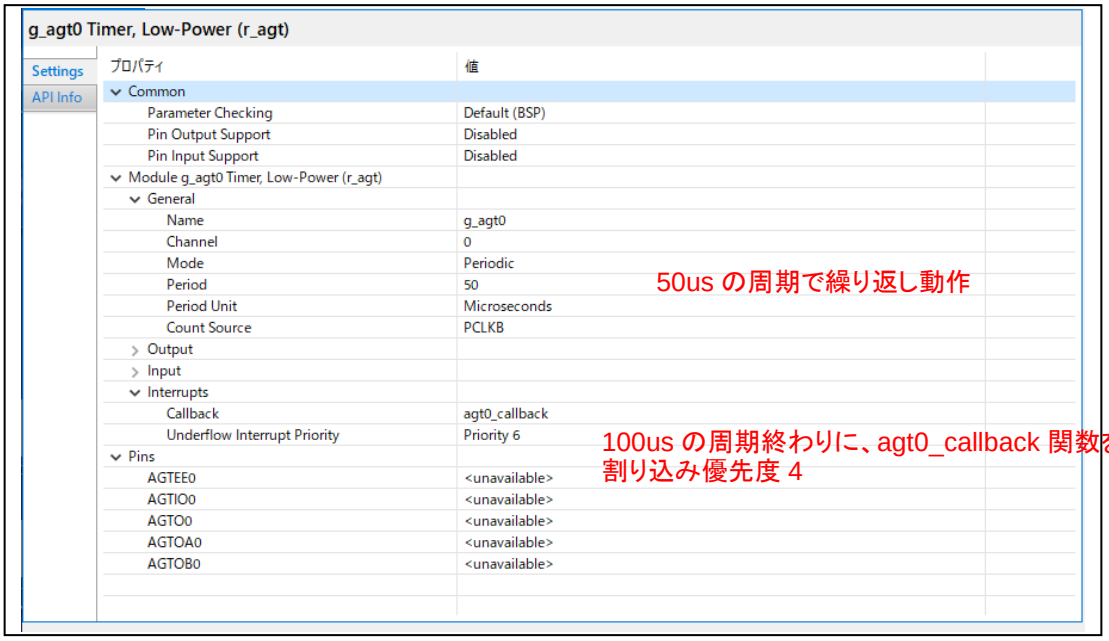
スキャングループ	割り当て端子
Scan Group0	AN000~AN005
Scan Group1	AN006~AN011
Scan Group2	AN012~AN017
Scan Group3	AN018~AN021, AN028

本プログラムでは、4 つのスキャングループを定義して、合計 23 端子の A/D 変換を行っています。
（1 つのスキャングループは 8 端子まで、AD ユニットの 0/1 が混在する組み合わせは同一スキャンググループには入れられない等の制約があります。制約内でグルーピングを行い、グループ毎に A/D 変換の実行が可能です。）

・g_uart9(SCI9)設定



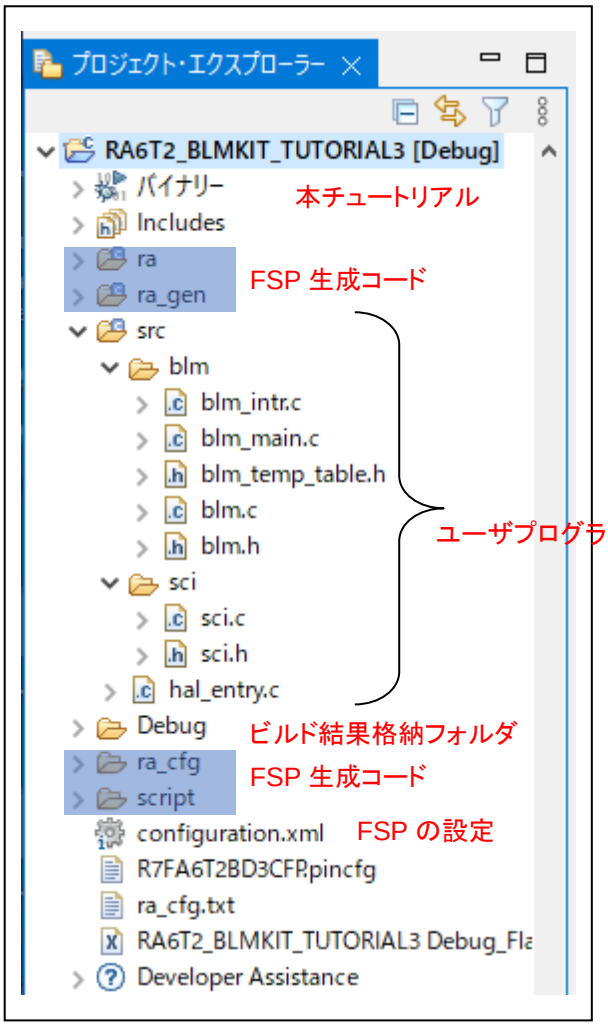
・g_agt0(AGT0)設定



AGT では、周期や割り込み関数の定義等を行っています。

FSP を使用すると、GUI で各種設定を行う事で、設定に応じたプログラムコードが出力されます。

ここで、プログラムのソースの構成を示します。



Configuration.xml は FSP の設定です。クロック設定、端子設定、stack の追加等はここで行います。

・ユーザプログラム（src フォルダ以下）

blm	blm_intr.c	割り込みプログラムコード向けソースファイル
	blm_main.c	ユーザメイン関数
	blm_temp_table.h	温度変換テーブル
	blm.c	関数類を記載したソースファイル
	blm.h	共通ヘッダ
sci	sci.c	SCI(UART)通信ソースファイル
	sci.h	SCI(UART)ヘッダ
hal_entry.c		ユーザプログラムの開始ポイント

src 以下がユーザプログラムです。ソースは上記構成となっており、以降のチュートリアルやサンプルプログラムでも同様のツリー構造です。

src 以下には、ユーザが作成した C のソースファイル(.c)やヘッダファイル(.h)を追加して構いません。

ra, ra_gen, ra_cfg フォルダは FSP が生成したコードが格納されていますので、基本的にはユーザが編集しない前提です。script はリンカ設定。Debug 以下はビルドにより生成されるファイル（mot ファイルや map ファイル）が格納されます。

本チュートリアルでは、AGT タイマを用いて定期的に割り込みを掛けるようにしています。

・src¥blm¥blm_intr.c 内 AGT0 コールバック（割り込み）関数

50us に 1 回呼び出される

```
void agt0_callback (timer_callback_args_t * p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */

        //50usの周期割り込み処理

        //カウンタ変数
        g_agt0_counter++;
        g_agt0_counter_2++;

        //50us ADC起動
        if (g_adc_scan_flag == 0)
        {
            g_adc_scan_flag = ADC_GROUP_MASK_0 | ADC_GROUP_MASK_1 | ADC_GROUP_MASK_2 | ADC_GROUP_MASK_3;
            (void) R_ADC_B_ScanGroupStart(&g_adc0_ctrl, g_adc_scan_flag);
        }
    }
}
```

50us×n 回を観測するためのモニタ変数

前回の A/D 変換が終わっていたら
A/D 変換開始指示

本チュートリアルでは、A/D 変換は AGT0 タイマの周期終わりに呼ばれる割り込み関数（上記）中で実行（A/D 変換開始指示）しています。
A/D 変換が終了すると、A/D 変換終了の割り込みが入ります。（入るように設定しています）

A/D 変換完了時の処理は、以下の様になっています。

・src¥blm¥blm_intr.c 内 AD0 コールバック（割り込み）関数

A/D 変換完了割り込み

```
void adc_callback (adc_callback_args_t * p_args)
{
    if (p_args->event == ADC_EVENT_SCAN_COMPLETE)
    {

        switch (p_args->group_mask)
        {
            case ADC_GROUP_MASK_0:
                g_adc_result[BLM_CH1].v_u_phase = R_ADC_B->ADDR_b[0].DATA;
                g_adc_result[BLM_CH1].v_v_phase = R_ADC_B->ADDR_b[1].DATA;
                g_adc_result[BLM_CH1].v_w_phase = R_ADC_B->ADDR_b[2].DATA;
                g_adc_result[BLM_CH1].i_u_phase = R_ADC_B->ADDR_b[3].DATA;
                g_adc_result[BLM_CH1].i_v_phase = R_ADC_B->ADDR_b[4].DATA;
                g_adc_result[BLM_CH1].i_w_phase = R_ADC_B->ADDR_b[5].DATA;
                g_adc_scan_flag &= (unsigned short)~ADC_GROUP_MASK_0;
                break;
[後略]
```

AD 変換結果を
グローバル変数にコピー

A/D 変換結果レジスタ値を、グローバル変数に代入する処理となっています。

stack で設定した機能の起動等は、blm_init()関数で行っています。

・src¥blm¥blm.c 内初期化関数

```
void blm_init(void)
{
    //ブラシレスモータ初期化関数

    //引数：なし
    //戻り値：なし

    //CH 出力ポート設定
    //FSPのPins Configurationで設定済み

    //LED, SW
    //FSPのPins Configurationで設定済み

    //AGT0 : 50us ADC変換スタート
    //AGT1 : 10ms 画面表示更新タイミング
    (void) R_AGT_Open(&g_agt0_ctrl, &g_agt0_cfg);
    (void) R_AGT_Open(&g_agt1_ctrl, &g_agt1_cfg);

    //GPT3 : CH-1, CH-2 QL PWM
    //GPT9 : CH-4 QL PWM
    (void) R_GPT_Open(&g_gpt3_ctrl, &g_gpt3_cfg);
    (void) R_GPT_Open(&g_gpt4_ctrl, &g_gpt4_cfg);

//ADC
    (void) R_ADC_B_Open(&g_adc0_ctrl, &g_adc0_cfg);
    (void) R_ADC_B_Calibrate(&g_adc0_ctrl, NULL);

    //ADCキャリブレーション処理終了待ち
    fsp_err_t err = FSP_SUCCESS;
    adc_status_t status = {.state = ADC_STATE_SCAN_IN_PROGRESS};
    while ((ADC_STATE_SCAN_IN_PROGRESS == status.state) &&
           (FSP_SUCCESS == err))
    {
        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
        err = R_ADC_B_StatusGet(&g_adc0_ctrl, &status);
    }

    (void) R_ADC_B_ScanCfg(&g_adc0_ctrl, &g_adc0_scan_cfg);

#ifdef ADC_B_BUG_FIX_TEMPORARY_CODE
    //FSP3.6ではA/D変換のグループスキャン実行の際のグループ選択にバグがあるので一時的
な修正コードを追加する
    //FSPのバージョンアップに伴いバグが解消された場合は
ADC_B_BUG_FIX_TEMPORARY_CODE を未定義としてください
    g_adc0_ctrl.cached_adsystr = ADC_GROUP_MASK_0 | ADC_GROUP_MASK_1 |
ADC_GROUP_MASK_2 | ADC_GROUP_MASK_3;

    //FSP3.6ではシングルスキャンモー時、1つのADCチャネル(AD0, AD1)に対して、2スキャン
グループを割り当てた場合のレジスタ設定が行われないので
    //追加する
    //本プログラムでは
    //ADC0 : スキャングループ0, スキャングループ2
    //ADC1 : スキャングループ1, スキャングループ3
    //を割り当てている
```

AGT, GPT タイマの設定

ADC の設定
ADC のキャリブレーション

ADC のスキャン設定

一時的なコード
（FSP のバージョンアップに伴い
不要となる予定）

```
    R_ADC_B->ADGSPCR_b.PGS0 = 1;
    R_ADC_B->ADGSPCR_b.PGS1 = 1;
    R_ADC_B->ADGSPCR_b.RSCN0 = 1;
    R_ADC_B->ADGSPCR_b.RSCN1 = 1;
    R_ADC_B->ADGSPCR_b.LGRRS0 = 1;
    R_ADC_B->ADGSPCR_b.LGRRS1 = 1;;
#endif

    //AGTタイマスタート
    (void) R_AGT_Start(&g_agt0_ctrl);
    (void) R_AGT_Start(&g_agt1_ctrl);

    //GPTタイマスタート
    (void) R_GPT_Start(&g_gpt3_ctrl);
    (void) R_GPT_Start(&g_gpt4_ctrl);
}
```

AGT タイマのスタート

GPT タイマのスタート

FSP の API 関数を使い、初期化や機能の起動等を行っています。FSP の API 関数の仕様は、e2studio がインストールされているフォルダ内に html のマニュアルの形で格納されていますので、必要に応じて参照してください。

・src¥blm¥blm.c 内 hal_entry()

```
void hal_entry(void)
{
    /* TODO: add your own code here */

    blm_main();

    while(1);

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

モータ制御の処理にバトンを渡す

この while は実行されない

hal_entry()関数が、一般的なメイン関数のイメージです。(main()は FSP が生成するコード内で定義されており、そこから hal_entry()が呼ばれる)この部分に、ユーザの処理を記載します。ここでは、blm_main()に処理を引き継ぐ形としています。

・src¥blm¥blm_main.c 内 blm_main()

```
void blm_main(void)
{
    //ブラシレスモータメイン関数

    const unsigned long duty_change_interval = (unsigned long)(0.1 / 50.0e-6);          //0.1[s]毎（50usで何カウントか）
    const unsigned long sw_read_interval = (unsigned long)(500e-6 / 50.0e-6);           //500us 毎にスイッチの状態をスキャン（50usで何カウントか）
    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）

    unsigned short prev_state[BLM_CH_NUM] = { BLM_CH_STATE_INACTIVE, BLM_CH_STATE_INACTIVE, BLM_CH_STATE_INACTIVE };
                                                                                        変数定義
    unsigned short i;
    unsigned short ret;

    sci_start();            SCI(UART)の初期化

    sci_write_str("¥nCopyright (C) 2022 HokutoDenshi. All Rights Reserved.¥n¥n");
    sci_write_str("RA6T2 / BLUSHLESS MOTOR STARTERKIT TUTORIAL3¥n");

    blm_init();         //初期化          ブラシレスモータ処理の初期化（前頁に記載）
    blm_led_out(0);     //LED 全消灯
[後略]
```

blm_main()関数が、モータ制御の処理を行っているプログラムのスタート地点です。

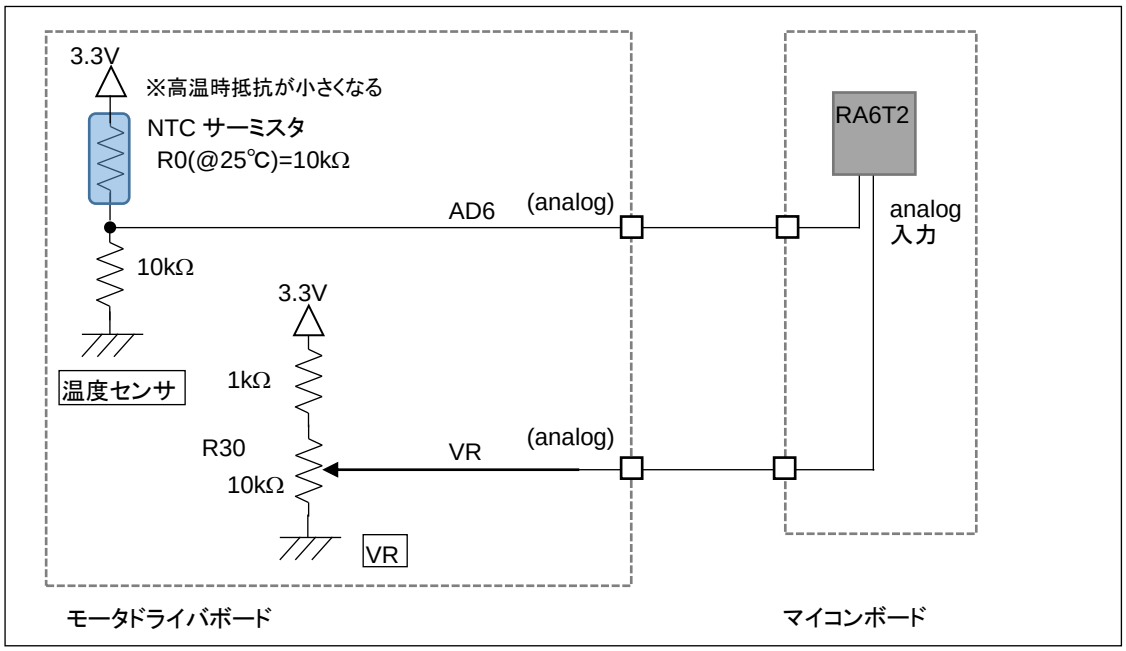
・プログラムのメインループ

```
    //メインループスタート
    while (1)
    {
        //スイッチの読み取り(500us毎）→出力ON/OFF
        if (g_agt0_counter >= sw_read_interval)
        {
            //チャタリング防止
            //スイッチは、50us毎×sw_read_interval(10)=500us毎に読み取りを行う

            blm_sw_to_state();                    SW を読み取りグローバル変数に代入

            //状態が変化した際スタート・ストップ
            for (i=0; i<BLM_CH_NUM; i++)
            {                                     SW の状態が変化した際スタート･ストップの処理
                if (g_state[i] != prev_state[i])
                {
                    if (g_state[i] == BLM_CH_STATE_ACTIVE)
                    {
                        blm_start(i);
                        sci_write_str("¥n  CH-");
                        sci_write_uint16(i+1);
                        sci_write_str(" START¥n");
                    }
                    else
                    {
                        blm_stop(i);
                        sci_write_str("¥n  CH-");
                        sci_write_uint16(i+1);
                        sci_write_str(" STOP¥n");
                    }
                    prev_state[i] =  g_state[i];    //現在の状態を保存
                }
            }
            g_agt0_counter = 0; //カウンタ初期化
        }

        //VRをdutyに反映（0.1秒毎）
        if (g_agt0_counter_2 >= duty_change_interval)
        {
            blm_duty_change();                    0.1 秒に 1 回 VR の読み取り値を duty に反映させる

            g_agt0_counter_2 = 0;
        }

        //画面表示（3秒に1回）
        if (g_agt1_counter >=  information_display_interval)
        {
            blm_information_display();        3 秒に 1 回画面表示を行う

            g_agt1_counter = 0;
        }

    }  //while
```

本チュートリアルで使用端子をまとめると、以下となります。

	CH-1	CH-2	CH-3
VR 読み取り(VR)	PC04/AN010	PC05/AN011	PB010/AN028
温度センサ(AD6)	PA06/AN006	PB00/AN008	PE09/AN021
PWM 波形出力(QL)	PC14/GTIOC3A	PD11/GTIOC3B	PE01/GTIOC4B

温度センサと VR の読み取りは、以下の様になっています。



温度センサは、25℃の時サーミスタは 10kΩとなりますので、AD6 端子の電位は 1.65V となります。高温時は電圧が上がる方向に変化します。VR は、軸を(軸方向から見て)反時計回りに回した際出力電位が上がり、最大 3.0V 程度(A/D 変換値で 3720 程度)、最小 0V((A/D 変換値で 0)となります。

本チュートリアルでは、モータ制御の周辺機能を使うもので、モータを駆動するという観点からは一旦離れましたが、FSP を使ってマイコンの種々の機能を動かすというチュートリアルで、RA マイコンで開発を行う上では重要な部分です。

次のチュートリアルでは、実際にモータを動かしてみます。

1.4. モータを回してみる

参照プロジェクト：RA6T2_BLMKIT_TUTORIAL4

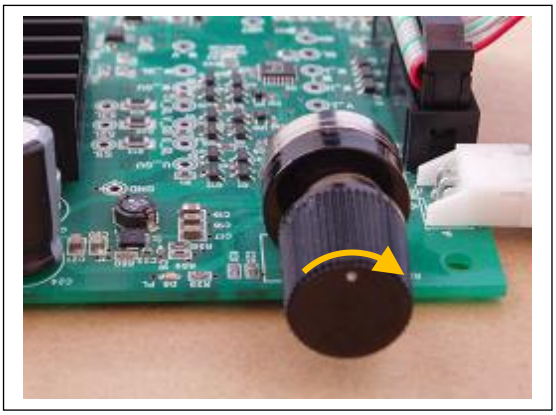
（アーカイブファイル：RA6T2_BLMKIT_TUTORIAL4.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

プログラムの動作としては、以下となります。

電源投入前に、SW1~SW2 のトグルスイッチを OFF に倒した状態としてください。

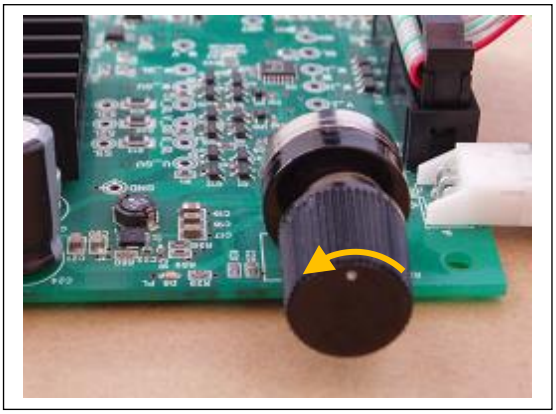
そして、VR を目一杯時計回り（軸を正面からみて）に回してください。



モータドライバボードに電源を投入してください。

CH-1 にモータドライバボードを接続している場合、SW1 を ON に倒してください。

徐々に VR を反時計回りに回していきます。



（オシロスコープがある場合は、QL 端子の duty を観測してください）
最初は変化がないと思いますが（ここで電源から流れ出る電流がモニタ出来る場合は、徐々に電流値が増していると思います）、ある程度 VR を回すとモータの軸が振動を始めるはずです。

VR をもっと回すと、モータの軸の振動が大きくなり、いずれスムーズに回転を始めると思います（このときの、duty は約 10%程度で、電流は、0.2A 程度です）。

回転前のモータの軸が振動している状態の時は、消費電流が大きく、回転を始めると消費電流は減ります。

VR を目一杯回すと、モータからブーンとうなる音が聞こえてくると思います。（消費電流は最大 2A 程度となります、本プログラムの duty は最大 19%程度に設定しています）
※この状態はモータドライバボード上の FET が発熱しますので、あまり長い時間維持しないでください

本プログラムでは、モータに流す電流の向き（＝印加磁界の向き)は、一定周期(6ms)で変化させ、モータに流す電流の大きさは、VR の回転角度に連動させています。

SCI9（J6 または J7）端末を接続すると、以下の情報が出力されます。

・シリアル端末から出力される情報（3 秒に 1 回更新）

```
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

RA6T2 / BLUSHLESS MOTOR STARTERKIT TUTORIAL4

Motor driver board connection check...
  CH-1 Connected.
  CH-2 NOT connected.
  CH-3 NOT connected.


                       CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     x         x         x
Temperature(A/D value) :  2185         0         0
Temperature(degree)    :    29         0         0
VR(A/D value)          :     0         0         0
QL duty[%]             :   0.0       0.0       0.0

  CH-1 START


                       CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2186         0         0
Temperature(degree)    :    29         0         0
VR(A/D value)          :     0         0         0
QL duty[%]             :   0.0       0.0       0.0

                       CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2187         0         0
Temperature(degree)    :    29         0         0
VR(A/D value)          :  1112         0         0
QL duty[%]             :   5.4       0.0       0.0

                       CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2185         0         0
Temperature(degree)    :    29         0         0
VR(A/D value)          :  1837         0         0
QL duty[%]             :   9.0       0.0       0.0

                       CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
Temperature(A/D value) :  2188         0         0
Temperature(degree)    :    29         0         0
VR(A/D value)          :  1885         0         0
QL duty[%]             :   9.2       0.0       0.0
```

起動

モータドライバボードの
接続チェック

SW1=OFF の状態

SW1=ON

Active = o になります

VR を回して duty を増やしていく

回転が安定した状態

QL duty の値と回転の様子に着目してください。スムーズに回っている状態ですと、10%ぐらいの duty になるのではないかと思います。それより小さいと、軸が振動するだけで回らず。それより大きいと、モータの振動が大きくなり、スムーズに回る感じではなくなると思います。モータ制御においては、duty の制御（=モータに与える電力）が重要であると言えるかと思います。

本チュートリアルでは、TUTORIAL3 で使用している機能に加え、GPT0 タイマを使用しています。

Stack 種	内容	用途	備考
g_gpt0	GPT0 タイマ	モータに印加する電流（磁界の方向）のシフト	6ms 周期

6ms 毎に流す電流方向を変えていき（印加磁界の方向を変えていき）、モータを回す力を生み出しています。

・src¥blm¥blm_intr.c 内 GPT0 割り込み関数 gpt0_callback　　6ms 毎に呼び出される関数

```
void gpt0_callback (timer_callback_args_t * p_args)
{
    //6ms, モータ回転周期タイマ

    //モータに与える磁界を回転させてゆく処理
    //6ms毎に、電流を流す方向を変えてゆく

    static unsigned short loop = 0;
    unsigned short motor_phase_control;
    unsigned short i;

    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */

        switch(loop)
        {
            case 0:
                motor_phase_control = U_V_DIRECTION;
                break;

            case 1:
                motor_phase_control = U_W_DIRECTION;
                break;

            case 2:
                motor_phase_control = V_W_DIRECTION;
                break;

            case 3:
                motor_phase_control = V_U_DIRECTION;
                break;

            case 4:
                motor_phase_control = W_U_DIRECTION;
                break;

            case 5:
                motor_phase_control = W_V_DIRECTION;
                break;

            default:
                motor_phase_control = OFF_DIRECTION;
                break;
        }
```

6ms で次の状態に移行

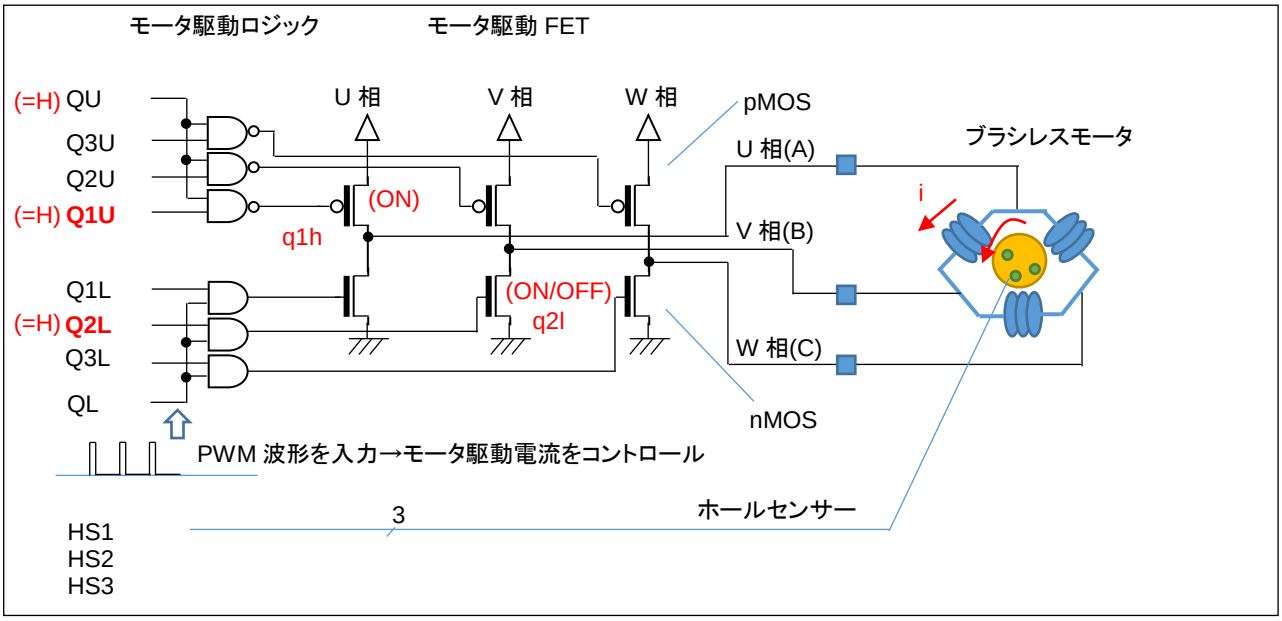
```
        loop++;
        if (loop >= 6) loop = 0;

        for (i=0; i<BLM_CH_NUM; i++)
        {
            if(g_state[i] == BLM_CH_STATE_ACTIVE)
            {
                blm_drive[i](motor_phase_control);
            }
            else
            {
                blm_drive[i](OFF_DIRECTION);
            }
        }
    }
}
```

ループ変数をインクリメント

SW=ON(CH-n:アクティブ）のとき
モータ駆動端子をドライブ

SW が OFF の時は駆動 OFF
（U, V, W 相の pMOS, nMOS の
6 素子全て OFF）

本プログラムでは、スイッチが ON ならば、モータに流す電流の向きを 6ms 毎に次の方向に切り替えて行きます。VR に連動した duty は、QL の信号に与えています。



QL には常に、（VR 回転角度に応じた）PWM 波形が入力されています。

プログラム的に g_motor_phase_control = U_V_DIRECTION のとき、q1h は ON（6ms の期間ずっと）していますが、q2l は、ON/OFF を断続的に繰り返しています。（ON している割合が duty 比）モータの U→V に流れる電流も、断続的に流れる・止まるを繰り返しています。duty 比が平均電流となりますので、duty 比を大きくした方がモータの軸を回す力も大きくなります。

（U_V_DIRECTION の時は、U 相の H 側(pMOS)と、V 相の L 側(nMOS)のスイッチング素子(MOS FET)が ON します。その他の方向も同様に、H 側と L 側を 1 つずつ ON させます。電流を流す方向は、1.2 章で説明した 6 パターンとなります。）

本プログラムでは、6ms 毎に 1/6 回転する制御としています（回転数は固定です）。1 回転で、36ms なので、回転数としては、28 回転/s。1 分あたりの回転数は、1667rpm です。モータの回転方向は反時計回り（軸方向から見て）です。

これは、回転数を固定とし、モータに流れる平均電流を変化させモータを回転させる手法で、電流が小さいときはモータが回らず、電流が大きいときには（モータが発する音が大きくなり）無駄なエネルギーが消費されています（モータの軸を回す力が大きく、1667rpm より速く回す能力があるにも拘わらず、回転数がプログラムで 1667rpm に固定されているためです）。

特定の回転数でモータを回すためには、モータに流す平均電流を制御する必要があります。ここでは duty 比で電流を変化させていますので、回転数に応じた duty 比の制御が必要になると考えてください。
（なお、モータの軸に負荷をつなげている場合は、負荷の大きさに対応して duty を増やす必要があります。）

本チュートリアルでは、モータに流す電流方向を変化させ、適切な duty を与えればモータが回転する事を示しています。次のチュートリアルでは、モータに内蔵されたセンサを用い、モータが回っているのか、止まっているのか。また、現在の軸の絶対位置を読み取る方法を示します。

1.5. ホールセンサの値をみる

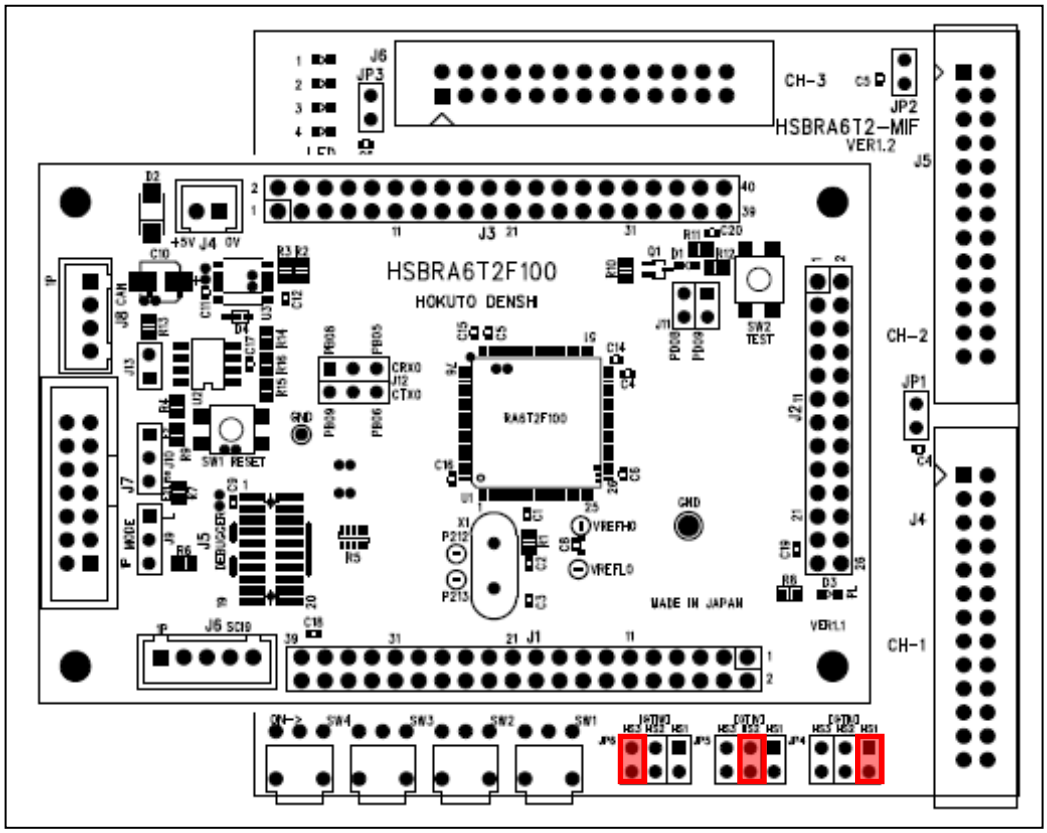
参照プロジェクト:RA6T2_BLMKIT_TUTORIAL5

（アーカイブファイル:RA6T2_BLMKIT_TUTORIAL5.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

本チュートリアルでは、ホールセンサの値を読み取っています。
TUTORIAL3 同様、シリアル端末を接続してください。
SW1, SW2, SW3=OFF とします。

・ジャンパ設定



接続ボードの JP4~6（ボード右下）のジャンパを

JP4(GTIU)	HS1
JP5(GTIV)	HS2
JP6(GTIW)	HS3

に設定してください。（モータのホールセンサとマイコンの入力端子を接続するジャンパです。）

・シリアル端末から出力される情報

```
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

RA6T2 / BLUSHLESS MOTOR STARTERKIT TUTORIAL5

Motor driver board connection check...
   CH-1 Connected.
   CH-2 NOT connected.
   CH-3 NOT connected.

pos = 1   7   7
```

モータドライバボードが接続されていない場合は 7 となります

← CH-1 CH-2 CH-3 のホールセンサ位置情報

PC 上では、teraterm 等のシリアル端末ソフトで表示してください

115200bps, 8bit, none, 1bit の設定で表示できます

電源を投入すると、上記の表示がシリアル端末に出力されます。

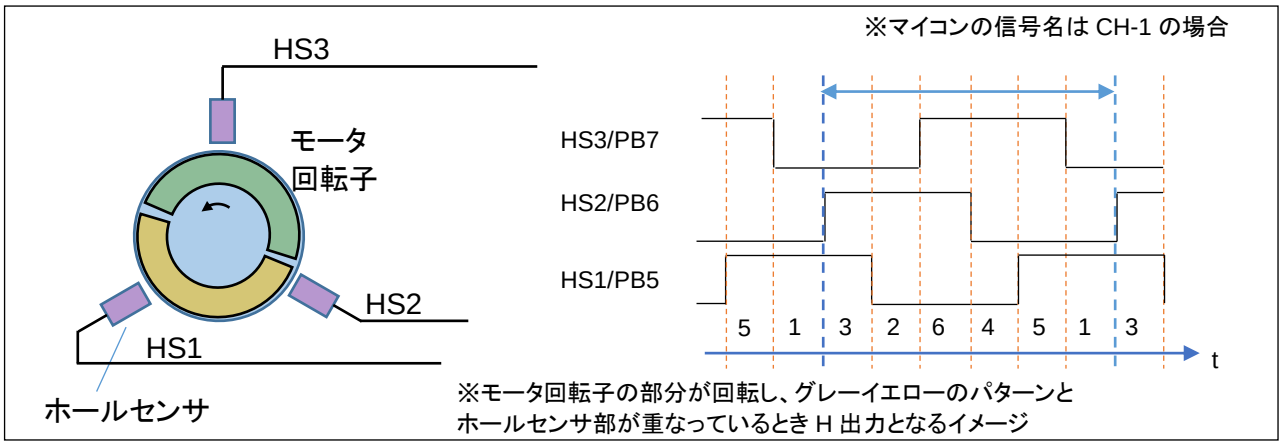
表示は、0.1s 間隔で更新しています。ここで、モータの軸を手で回してみてください。数字が 1 から 6 まで変化すると思います（数値の変化の仕方は、一見順不同に見えると思います）。

この数値は、ホールセンサの位置を示しています。モータの軸は 1 回転あたり、6 回クリック（止まるところ）があると思います。モータの軸が 1 のとき

1　[時計回りに軸を回転] → 5

1　[反時計回りに軸を回転] → 3

となるはずです。この数値は、軸の絶対位置を表しています。



ホールセンサの値は、基本的には duty=50%の矩形波が 120°ずれて出力されるイメージです。

本プログラムで表示される数値は、

数値(pos)	[HS3] PB07 (b2)	[HS2] PB06 (b1)	[HS1] PB05 (b0)
3	0	1	1
2	0	1	0
6	1	1	0
4	1	0	0
5	1	0	1
1	0	0	1

0=L
1=H

pos = PB7×4 + PB6×2 + PB5 （※CH-1 の場合）

となります。（プログラムの処理を容易にするため、PB7, PB6, PB5 に重み付けを行い加算した数値）

ホールセンサの出力は、接続されているマイコンのピンを単純に入力回路に設定して、デジタル的に読み取っています。(SW1~SW4 を読み取るのと同様の手法)

モータの軸を手で回すと、上記表の数値に従い変化 1→5 または 1→3(回転方向による)すると思います。これは、プログラムで「いまモータの回転軸がどの位置にあるのか」を把握できていることを示します。

本モータのホールセンサは、軸 1 回転につき、出力が 6 値変わりますので、軸位置が 60 度変化すると、ホールセンサの出力が変化するという事となります。

※モードライバボードが接続されていない CH は、数値が 7 となります

ここで、SW1 を ON してみてください(CH-2/CH-3 を使っている場合は SW2/SW3 を ON)。
(スイッチを ON にすると、pos の画面表示は止まります)

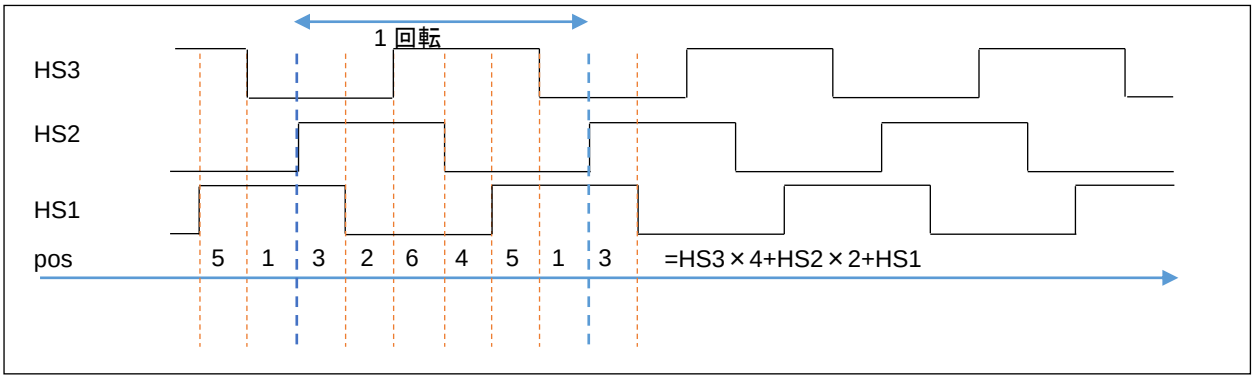
VR を回すと、TUTORIAL4 のプログラム同様、モータが回転を始めると思います。しかし、このプログラムでは、VR の回転に応じてモータの回転数が変わる事(モータの回転数は、音からある程度判断するしかないのですが)にお気付きでしょうか。TUTORIAL4 のプログラムは、回るか・回らないか。回ったときは常に 1670rpm ぐらいで回る(36ms で 1 回転)という動作でした。しかし、このプログラムでは、ホールセンサーの読み取り値が変化したときに電流の向きを変化させます。(正確には、電流の向きを変えるのは、タイマ割り込みの 50us 刻みのタイミングです。)

なお、VR を回す→QL 信号の duty 比を変えるという動作は、TUTORIAL4 と同じです。TUTORIAL4 と異なるのは、電流の向きを変えるタイミングです。TUTORIAL4 では、6ms という決まったタイミングでしたが、本チュートリアルでは、ホールセンサが切り替わったタイミングが電流の向きを切り替えるトリガとなります。

モータの軸が 1/6 回転して、ホールセンサの値が変わった時点で、モータの回転軸を引っ張る(押し出す)磁界を生む電流にスイッチできるため、流れる電流(このプログラムでは、VR に連動した duty)を大きくすると、モータの回転軸にかかる力が大きくなり、速く次のホールセンサ切り替え(1/6 回転)に達するため、VR(duty)とモータの回転数が連動する動作となります。

言い換えると、TUTORIAL4 のプログラムは、duty を大きくすると、モータの回転軸は速く次の 1/6 回転に達しますが(pos=5→1 等)、そこ(pos=1)で同じ場所(pos=1)に引っ張る力をキープしますので、エネルギーが無駄に消費され、電流は増加するものの回転数は変わらないという動作です。

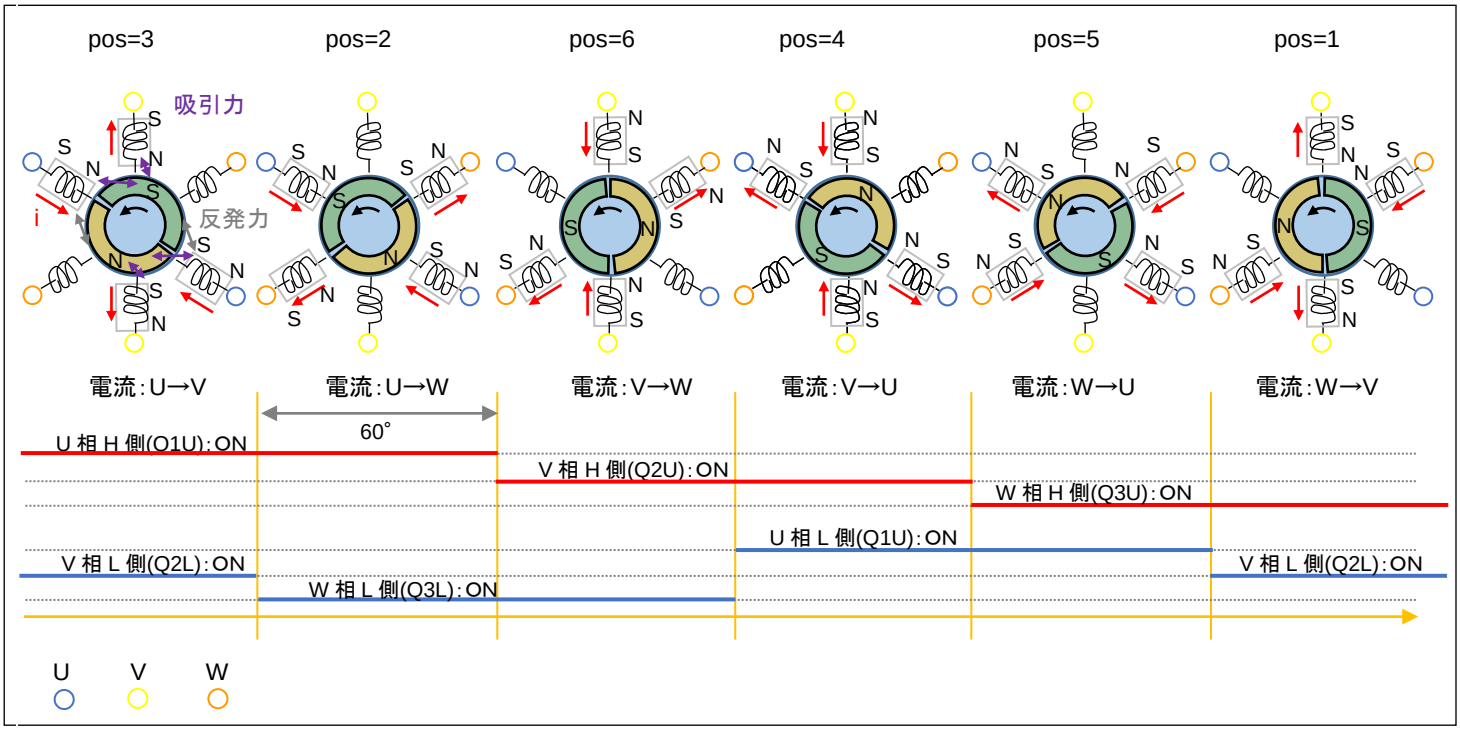
・ホールセンサ位置と電流印加方向に関して



pos	反時計回り(CCW)	時計回り(CW)
3	U→V	W→U
2	U→W	W→V
6	V→W	U→V
4	V→U	U→W
5	W→U	V→W
1	W→V	V→U

※矢印の向きは時系列、本チュートリアルでは回転方向は「反時計回り(CCW)」固定です

・反時計回り(CCW)



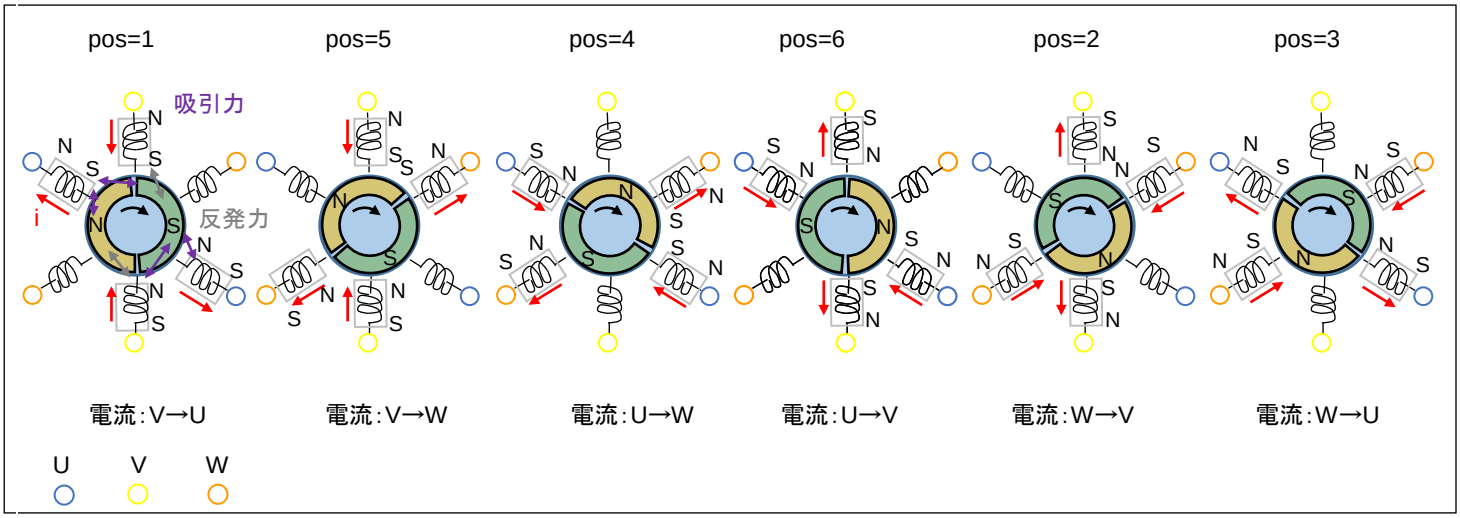
モータが 60 度回転した時点で(60 度回転した事はホールセンサで判断します)、次の電流パターンにシフトさせます。UVW の 3 相、H 側/L 側の計 6 本の制御信号を 120° 毎に ON/OFF を切り替えていく制御となりますので、このような制御方法は「120 度制御」と呼ばれます。

※コイルの数や配置はイメージです(実際のモータ内部の構成とは必ずしも同じではありません)

[参考]

本チュートリアルでは、回転方向は固定ですが、逆回転（時計回り(CW)）させる場合は、以下の様になります。

・時計回り(CW)



※本チュートリアルでは、時計回り(CW)の回転方向のプログラムは実装されていませんが、次ページのプログラムの pos 値と電流印加方向のテーブルを上図の対応に変えればモータは逆回転となります

※チュートリアル A(RA6M2_BLMKIT_TUTORIAL_A)のプログラムでは、反時計回り(CCW)と時計回り(CW)の両方の回転方向が定義されています（2.1 章参照）

・src¥blm¥blm_intr.c 内 AGT1 割り込み関数 agt1_callback　　50us 毎に呼び出される関数

```
        //モータ回転制御
        for (i=0; i<BLM_CH_NUM; i++)
        {
            g_sensor_pos[i] = blm_holl_sensor_pos(i);

            //blm_drive[i] は関数ポインタで、blm_drive_ch1() ~ blm_drive_ch3()
            if(g_state[i] == BLM_CH_STATE_ACTIVE)
            {
                switch(g_sensor_pos[i])
                {
                    case 3:
                        blm_drive[i] (U_V_DIRECTION);
                        break;
                    case 2:
                        blm_drive[i] (U_W_DIRECTION);
                        break;
                    case 6:
                        blm_drive[i] (V_W_DIRECTION);
                        break;
                    case 4:
                        blm_drive[i] (V_U_DIRECTION);
                        break;
                    case 5:
                        blm_drive[i] (W_U_DIRECTION);
                        break;
                    case 1:
                        blm_drive[i] (W_V_DIRECTION);
                        break;
                    default:
                        blm_drive[i] (OFF_DIRECTION);
                        break;
                }
            }
```

ホールセンサ値の読み取り

ホールセンサにより算出された
現在のモータ回転子の位置
(g_sensor_pos[i])に応じて
流す電流の向きを決める

ホールセンサの情報を使うことにより、最適なタイミングでモータの軸を引っ張る磁界を生成できますので、本プログラムでは、duty（平均電流:トルクに対応）を増やすと、モータの回転数が上がります。

なお、本チュートリアルでは、回転方向は固定です（軸方向から見て、反時計回り）。実際のアプリケーションで、回転方向を変える必要がある場合は、プログラムのテーブル（pos=3 のとき、U→V に電流を流す）を前ページの図の対応に変える必要があります。

・チュートリアル 5 での画面表示（3 秒に 1 回）

```
                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       o           x           x
rotation speed([rpm])     :    1980           0           0
Temperature(A/D value)    :    2272           0           0
Temperature(degree)       :      31           0           0
VR(A/D value)             :    2325           0           0
QL duty[%]                :    11.4         0.0         0.0
```

本チュートリアルでは、回転数の表示が追加されています。

50us 毎に、変数値をインクリメントしていき、ホールセンサの値が変化した際、

・変数値を保存

・変数値のリセット（0 代入）

しています。そして、画面表示のタイミングで、変数値を 1 分間あたりの回転数([rpm])に変換しています。

```
period = (float)g_rotation_counter[i] * 50e-6f * 6.0f;          //1回転の周期
rpm[i] = (long)(1.0f / period) * 60L;                          //[rpm]変換
```

ホールセンサは、1/6 回転で値が変化するので、1 回転あたりの周期は、

50us で（ホールセンサ値が変化するまで）何回カウントされたか　×　50us　×6 ...(1)

で求まります。

回転数は、周期の逆数なので、(1)の逆数が 1 秒あたりの回転数。モータ等の回転数は、rpm, 1 分間あたりの回転数で表すことが多いため、1 秒間あたりの回転数×60 で計算しています。

1.6. 過電流・過熱保護の動作

参照プロジェクト：RA6T2_BLMKIT_TUTORIAL6

（アーカイブファイル：RA6T2_BLMKIT_TUTORIAL6.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

モータドライバボードが接続されている CH に対応したスイッチを ON にし、VR を回していくとモータが回転し、回転数が上がっていきます。

基本的な動作は、TUTORIAL5 と同じです。TUTORIAL5 のプログラムは、duty の最大値を 20%弱に制限していますが、本プログラムでは 90%程度まで duty を上げられます。VR を回していくと、回転数が上がりますが、一定以上まで上げた場合、急にモータの回転が止まるはずです(*1)。このとき、LED4 が点灯していると思います。これは過電流保護機構が働いたためです。モータドライバボード側では、過電流保護機構が働くと、*INT が L になります（L パルスが出ます）。マイコンボード側で、この信号は、CH-1 では PB04/IRQ13 につながっており、本プログラムでは以下の条件でモータに流れる電流を遮断し、モータを止める制御を行います。（過電流停止のサインとして、LED4 が点灯します）

(a)1 回でも PB04/IRQ13 の信号が L になった場合
(b)50us 毎に PB04 の信号をチェックし、10ms 間に 100 回以上過電流である場合
(c)50us 毎に PB04 の信号をチェックし、1 秒間に 1000 回以上過電流である場合

過電流の判定基準は、(a)~(c)のどの条件を有効にするかは任意の組み合わせで設定可能です。(a)が一番厳しい判定基準です。(a)を有効にした場合は、(b)(c)の判定前にモータが止まりますので、実質的には

(1) (a)を有効にする
(2) (b)及び(c)を有効にする
(3) (b)を有効化する
(3) (c)を有効化する

のいずれかの選択となります。(b)はある程度短期の判定基準。(c)は平均電流の判定基準のイメージです。(b)は 200 未満（10ms 間に 50us 毎に、200 回の判定となるので、200 以上の数値を指定すると、過電流エラー検出される事がない）。(c)は、20,000 未満の任意の値を設定可能です。

(*1)電流リミットの掛けられる電源装置をお使いの場合は、電流リミットを 2A~に設定してください。
（電流リミットを 1A 程度に設定すると、電源装置側の電流リミットに引っかかり、過電流検出される電流まで達しません。）

過熱保護機能に関しては、モータドライバボード上に搭載されているサーミスタ（1.3 章を参照）の温度のモニタリングを定期的に行い、設定した閾値を超えた場合に、モータを停止させるというものです。

プログラムでは、10ms 間隔で温度のモニタリングを行っており、1 回でも閾値を超えた場合モータが停止します。

・シリアル端末から出力される情報（過電流停止）

```
   CH-1 START

                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       o           x           x
rotation speed([rpm])     :       0           0           0
Temperature(A/D value) :    2137           0           0
Temperature(degree)       :      28           0           0
VR(A/D value)             :       0           0           0
QL duty[%]                :     0.0         0.0         0.0

                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       o           x           x
rotation speed([rpm])     : 14280           0           0
Temperature(A/D value) :    2135           0           0
Temperature(degree)       :      28           0           0
VR(A/D value)             :    2862           0           0
QL duty[%]                :    69.9         0.0         0.0

   CH-1 STOP

 *** OVER CURRENT ( COUNT = ONCE(interrupt) ) ***
```

PC 上では、teraterm 等のシリアル端末ソフトで表示してください

115200bps, 8bit, none, 1bit の設定で表示できます

VR を回して duty を上げていくと

過電流検出で停止

上記は(a)の 1 回の過電流信号の割り込みで停止した場合です。

```
                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       o           x           x
rotation speed([rpm])     : 14280           0           0
Temperature(A/D value) :    2160           0           0
Temperature(degree)       :      28           0           0
VR(A/D value)             :    2773           0           0
QL duty[%]                :    67.9         0.0         0.0

   CH-1 STOP

 *** OVER CURRENT ( COUNT = 113 / 10[ms]) ***
```

(b)(c)の設定（(a)は無効化）した場合は上記の様な表示となります。

・シリアル端末から出力される情報（過熱停止）

```
                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       x           x           x
rotation speed([rpm])     :   1620            0           0
Temperature(A/D value)    :   2882            0           0
Temperature(degree)       :      51            0           0
VR(A/D value)             :    374            0           0
QL duty[%]                :    0.0          0.0         0.0

  CH-1 STOP

 *** OVER TEMP ( TEMP = 51 [deg]) ***
```

過熱検出で停止

PC 上では、teraterm 等のシリアル端末ソフトで表示してください

115200bps, 8bit, none, 1bit の設定で表示できます

過熱停止の場合の表示例です。

エラーとなった場合は、5 秒間は一旦表示が停止し、その後動作は継続されます。エラーとなった CH のスイッチを OFF するとエラーはリセットされます。

（LED4 はエラー表示ですが、一度エラーが出ると点灯となり、リセットや電源再投入まで点灯の状態を維持します。）

・src¥blm¥blm.c

```
    //過熱停止有効，1回の過電流検出で停止
    g_error_check_flag = BLM_ERROR_OVER_TEMP_STOP | BLM_ERROR_OVER_CURRENT_STOP1;

    //過熱停止有効，10ms，1sの間規定回数以上の過電流検出で停止
    //g_error_check_flag      =      BLM_ERROR_OVER_TEMP_STOP      |      BLM_ERROR_OVER_CURRENT_STOP2      |
BLM_ERROR_OVER_CURRENT_STOP3;
```

プログラム内では、

g_error_check_flag

変数の値で、(a)(b)(c)及び過熱停止(d)の有効化を行います。

(a) BLM_OVER_CURRENT_STOP1(=0x2)　1 回の過電流検出信号で停止
(b) BLM_OVER_CURRENT_STOP2(=0x4)　10ms 間に規定の回数（デフォルト 100 回）以上過電流検出で停止
(c) BLM_OVER_CURRENT_STOP3(=0x8)　1 秒間に規定の回数（デフォルト 1000 回）以上過電流検出で停止
(d) BLM_OVER_TEMP_STOP1(=0x1)　　過熱停止

過熱停止と 1 回の過電流検出で停止を有効にする場合。
g_error_check_flag = BLM_OVER_TEMP_STOP1 | BLM_OVER_CURRENT_STOP1;
(g_error_check_flag = 0x3)

g_error_check_flag には、有効にしたい停止方法を OR(|)で与えてください。

・src¥blm¥blm.h

```
//過電流停止カウント回数
#define BLM_OVER_CURRENT_COUNT_10MS     100     //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）

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

(b)(c)の電流検出の(d)の過熱停止の閾値は、上記で定義されていますので別な値に変える事も可能です。

・過電流検出で使用している端子

モータドライバボード側の信号名は *INT です。この信号がマイコンボード側では以下の端子に接続されています。

	端子名	備考
CH-1	PB014/IRQ13	(a)の場合は IRQ13, (b)(c)の場合は汎用入力として使用
CH-2	PB02/IRQ9-DS	(a)の場合は IRQ9, (b)(c)の場合は汎用入力として使用
CH-3	PA12/IRQ12	(a)の場合は IRQ12, (b)(c)の場合は汎用入力として使用

※モータドライバボード側の過電流検出は、U 相と V 相の電流が両方 8A ピークを越えた場合*INT=L となります

(a)の IRQ を使用する場合は立ち下がりエッジの検出。(b)(c)の場合は、50us 間隔で端子のレベルを読み取り、10ms, 1s 間の L の回数をカウントします。

・過熱保護で使用している端子

モータドライバボード側の信号名は AD6 です。この信号がマイコンボード側では以下の端子に接続されています。

	端子名	備考
CH-1	PA06/AN006	A/D 入力として使用
CH-2	PB00/AN008	A/D 入力として使用
CH-3	PE09/AN021	A/D 入力として使用

(a)1 回でも *INT の信号が L になった場合停止させる処理。

・src¥blm¥blm_intr.c

```
void irq13_callback (external_irq_callback_args_t * p_args)
{
    //CH-1過電流割り込み

    blm_drive_ch1(OFF_DIRECTION);
    g_error[BLM_CH1].status |= BLM_ERROR_OVER_CURRENT_STOP1;
    g_state[BLM_CH1] = BLM_CH_STATE_INACTIVE;

}
```

CH-1 の場合

IRQ13 は立下りエッジ検出に設定

CH-1 側は、IRQ13 の割り込みが入った場合、即モータを停止させる処理です。

(b)(c)50us 毎に過電流をチェックする処理。

・src¥blm¥blm.c

```
bool blm_current_monitor_ch1(void)
{
    //戻り値
    //  過電流検出なし : 1 (true)
    //  過電流検出あり : 0 (false)

    return R_PORT11->PIDR_b.PIDR4;
}
```

CH-1 の場合

単純に端子のレベルを読む処理
（50us 毎に実行）

50us 毎に電流をチェックするのは、AGT0（50us タイマ）の割り込み処理内で実行しています。

10ms 毎のチェック(b)や 1 秒毎のチェック(c)、過熱停止のチェック(d)は、AGT1(10ms タイマ）の割り込み処理内で実行しています。

1.7. 相電圧・相電流の観測

参照プロジェクト：RA6T2_BLMKIT_TUTORIAL7

（アーカイブファイル：RA6T2_BLMKIT_TUTORIAL7.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

本プログラムでは、A/D 変換の機能を使い、U, V, W の各相電圧および U, V, W の L 側の電流観測用抵抗に流れている電流を取得します。プログラムの動作としては、TUTORIAL6 と同様です。

但し、本プログラムは、モータが回っているときにマイコンボードの SW2 を押すと押した瞬間に 400 ポイント（50us ×400=20ms 間）分の相電圧、相電流のデータを表示してモータが停止するという動作となります。

・シリアル端末から出力される情報

```
                              CH-1        CH-2        CH-3
Motor Driver Board        :   Connect     NoConnect NoConnect
Active                    :       o           x           x
rotation speed([rpm])     :    6660           0           0
Temperature(A/D value) :       2199           0           0
Temperature(degree)       :      29           0           0
VR(A/D value)             :    1472           0           0
QL duty[%]                :    36.0         0.0         0.0


   --- A/D information ---マイコンボード上の SW2 を押下

CH-1 A/D Conversion result

serial V(U) V(V) V(W) I(U) I(V) I(W) V(Power) temp volume
0 2858 3113 2885 56 1 1 3682 2199 1472
1 2858 3112 2889 215 1 1 3675 2201 1474
2 2846 3113 2897 320 0 0 3676 2191 1470
3 2839 3099 2899 24 0 0 3679 2198 1472
4 2835 3095 2904 36 0 1 3678 2198 1472
5 2830 3093 2910 99 0 0 3680 2200 1471
```

VR
U 相電圧, V 相電圧, W 相電圧
U 相電流, V 相電流, W 相電流,
温度，電源電圧

合計 9 種のデータを出力します

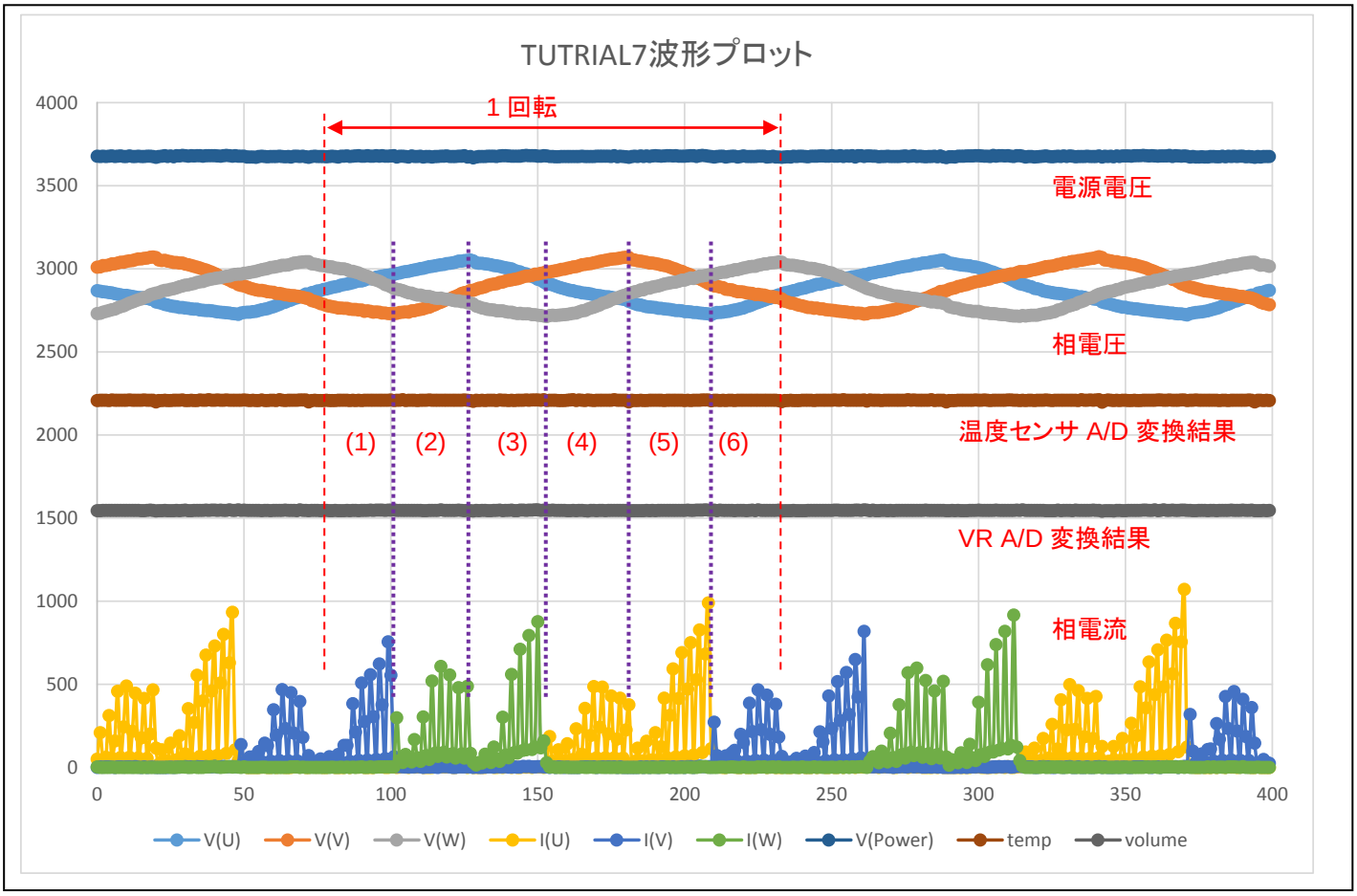
※CH-3 側は
VR
U 相電圧, V 相電圧, W 相電圧
温度
の 5 種のデータ出力

データのサンプリングレートは 20kHz
（50us 間隔）です

※データは、常時サンプリングされており、SW2 を押した時に、バッファリングされているデータ（400 点分）が表示されます

※CH-3 では、計測していないデータの値は 0 となります（マイコン A/D 変換ポート数の制約）

・シリアル端末から出力される情報を Excel でプロットした波形



波形は、端末に表示された数値をプロットしたものとなります。縦軸は、A/D 変換値となります。RA6T2 の A/D コンバータは、12bit 精度のため、値は 0~2^{12}-1(=0~4095)の範囲の値を取ります。相電圧は、モータの駆動端子を、RC でなだらかにした（LPF 通過後の波形）です。相電流は、GND 側に、電流センスの抵抗がある回路なので、I(U)がプラス方向に振れている＝他から U 相に電流が流れ込んでいる事を示しています。

本チュートリアルでは、前チュートリアル同様、1 回転で 6 回電流の向きを切り替えており、

	電流の流れる方向
(1)	U→V
(2)	U→W
(3)	V→W
(4)	V→U
(5)	W→U
(6)	W→V

に対応します。

※電流は PWM 駆動しているので、断続的に ON/OFF を繰り返しています。PWM のキャリア周波数と、A/D 変換周期(50us)の関係で波形の見え方は変わります。（RA6T2 マイコンでは、PWM 波形出力のタイミングで A/D 変換をキックする設定も可能です。）

・src¥blm¥blm_intr.c 内 AGT1 割り込み関数 agt1_callback

```
            if(g_state[i] == BLM_CH_STATE_ACTIVE)
            {
                switch(g_sensor_pos[i])
                {
                    case 3:
                        blm_drive[i] (U_V_DIRECTION);  (1)に対応
                        break;
                    case 2:
                        blm_drive[i] (U_W_DIRECTION);  (2)に対応
                        break;
                    case 6:
                        blm_drive[i] (V_W_DIRECTION);  (3)に対応
                        break;
                    case 4:                                          時系列
                        blm_drive[i] (V_U_DIRECTION);  (4)に対応
                        break;
                    case 5:
                        blm_drive[i] (W_U_DIRECTION);  (5)に対応
                        break;
                    case 1:
                        blm_drive[i] (W_V_DIRECTION);  (6)に対応
                        break;
                    default:
                        blm_drive[i] (OFF_DIRECTION);
                        break;
                }
            }
```

※回転方向は反時計回りです

※制御プログラム次第で、波形は変わります

本プログラムでは、A/D 変換の生データを一旦メモリに格納し、それをシリアル端末経由で出力する処理を行っていますが、実際のモータ制御プログラムでは、得られたデータをリアルタイムに処理して、モータの制御に活用する事が出来ます。

チュートリアル７までが、基本的にモータを回す上で必須となるであろうマイコンの機能を使ったチュートリアルとなります。

2. チュートリアル（応用編）

1~7 までのチュートリアルの内容を踏まえ、チュートリアル 7 のプログラムに機能を加えたのが 2 章で説明するチュートリアル（応用編）となります。

2.1. ハードウェアでの電流方向切り替え

参照プロジェクト：RA6T2_BLMKIT_TUTORIAL_A

（アーカイブファイル：RA6T2_BLMKIT_TUTORIAL_A.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

RA6T2 マイコンには、ホールセンサ接続端子が用意されており、ホールセンサの出力に応じて、出力の電流方向を切り替える事ができます。

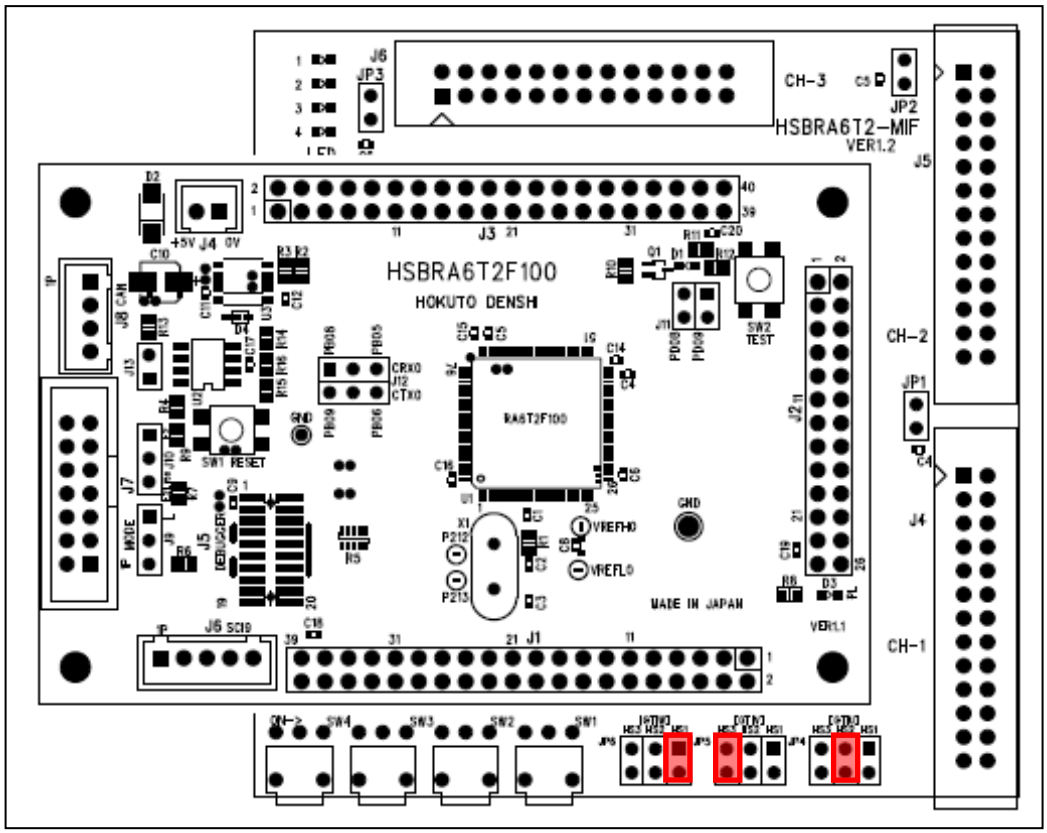
制御方式としては、120° 制御となりますが、タイマ(GPT0)を組み合わせて 6 相全ての信号を PWM 制御しています。

この方式の利点としては、マイコン側でハード的に電流方向の切り替えが行われる事です。（電流方向の設定をユーザプログラムで制御する必要がありません）

本チュートリアルでは、CH-1 のみ使用します。（CH-2, CH-3 は、この機能は使用できません。CH-2, CH-3 は、チュートリアル 7 と同じ手法でモータを回します。）

接続ボードの JP4~JP6 を以下の設定としてください、

・ジャンパ設定



接続ボードの JP4~6（ボード右下）のジャンパを

JP4(GTIU)	HS2	(HS3) (*1)
JP5(GTIV)	HS3	(HS1) (*1)
JP6(GTIW)	HS1	(HS2) (*1)

に設定してください。※本チュートリアル限定

本チュートリアルでは、JP4~JP6 ジャンパの挿し位置で回転方向が決まります。（上記以外の組み合わせでは、モータは回転しません(*1)）

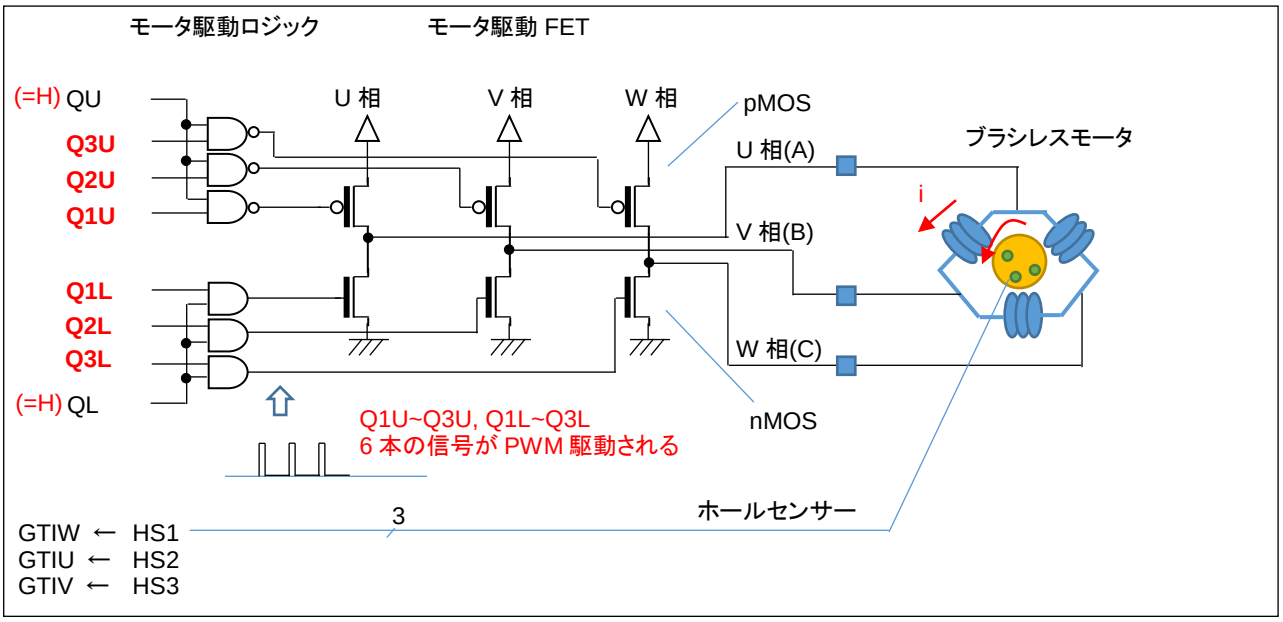
(*1)の組み合わせとするとモータは逆回転します。

JP4~6 ジャンパ設定後、VR を絞った状態で、SW1 を ON にしてください。その後、徐々に VR を回してみてください。VR の回転角に応じて回転数が変わります。（VR を半分ぐらいまで回さないと回転を始めませんが、一度回転が始まると、duty を絞っても回転を維持します。）

プログラムの動作は、TUTORIAL5~7 とそう変わらない動作ですが、本チュートリアルでは、ホールセンサの出力値がマイコンのハードウェアで直に処理されている点が異なります。

TUTORIAL5~7 では、50us 毎にホールセンサの値を見て、それに応じた電流方法の切り替えをプログラム内で行っていましたが、本チュートリアルでは電流方向の切り替えを行ってる処理のプログラムコードは存在しません。(50us 毎にホールセンサの値を見ていますが、これは回転数の算出のためです。ホールセンサの値を見ている部分の処理を消しても、モータの回転には影響しません。)

GTIU(PB05), GTIV(PB06), GTIW(PB07)は、マイコンのホールセンサ入力端子で、この 3 端子の L/H レベルの組み合わせで、出力の電流方向が決まります。

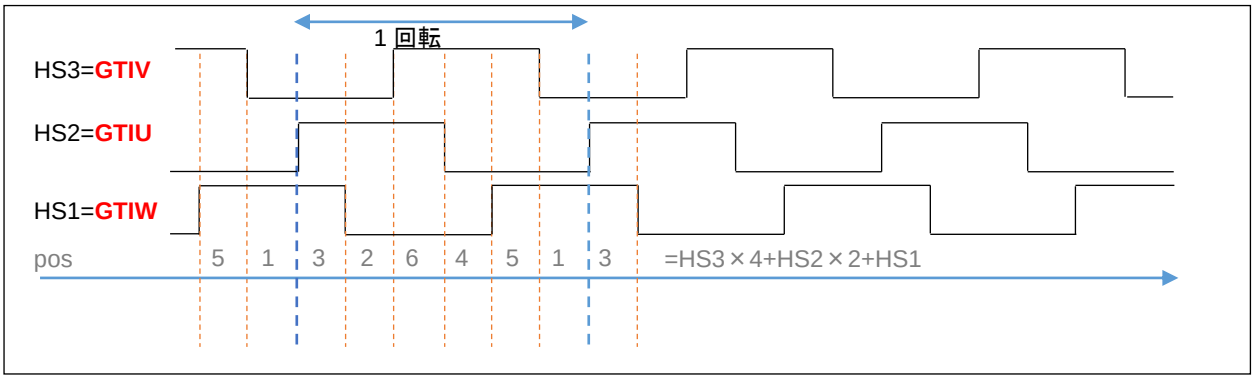


本チュートリアルでは、QU=QL=H とします。Q1U~Q3U, Q1L~Q3L の 6 本の信号がホールセンサ値(HS1~3)に応じて、アクティブ(H)になります。VR の回転角に応じて、PWM の duty は変わります。PWM は、Q1U~Q3U, Q1L~Q3L の 6 相に適用されます。

[参考]

・ホールセンサ位置と電流印加方向に関して（GPT_OPS 機能）

GPT_OPS の機能を使用してモータを回す際の、「ホールセンサパターン」－「電流印加方向」の関係ですがマイコンのハードウェアマニュアルを見ると、HS1~3 と GTIV~GTIW の関係が下記の様になっています（定義されています）。
※正しくは GTIV~GTIW（ホールセンサ入力）と 6 相の出力端子の関係がマイコン側で定義されています

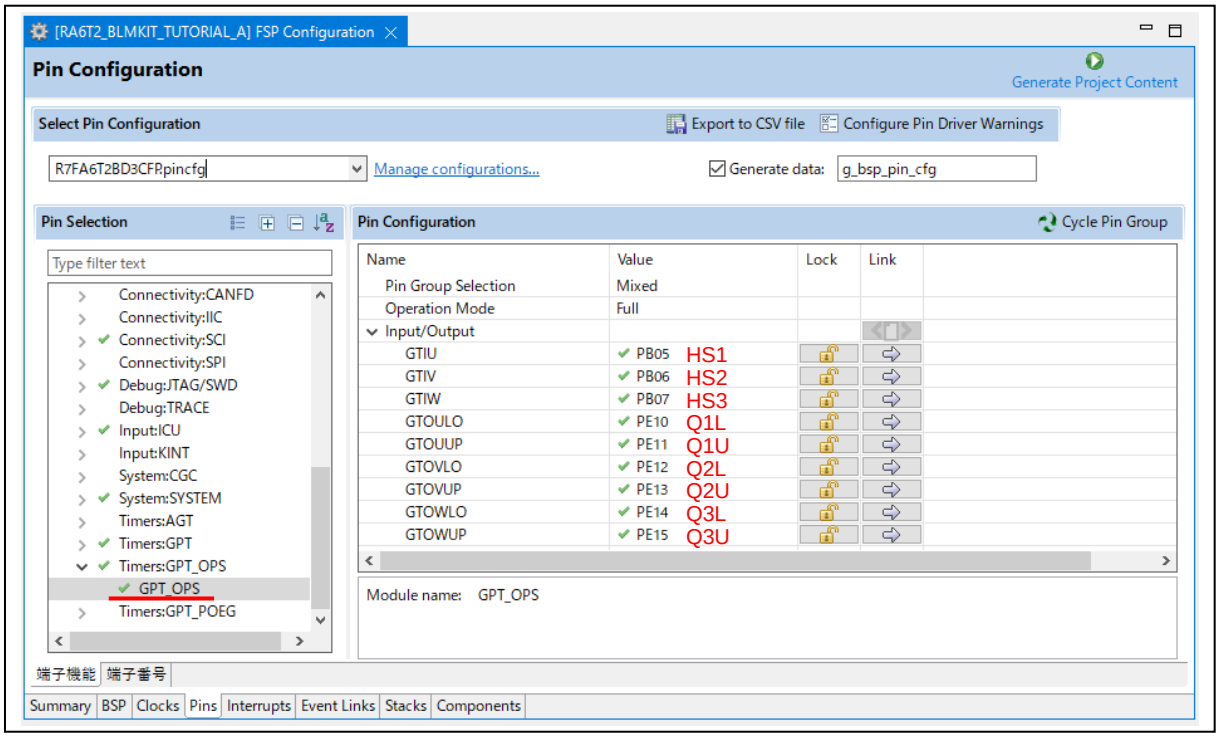


pos	反時計回り(CCW)	時計回り(CW)
3	U→V	W→U
2	U→W	W→V
6	V→W	U→V
4	V→U	U→W
5	W→U	V→W
1	W→V	V→U

そのため、本チュートリアルではジャンパピンを差し替えて、HS1~3とGTIU~GTIW の対応を、今までのチュートリアルと変更する必要があります。
HS1, HS2, HS3 はモータのホールセンサ出力端子の並び順に割り振った信号名です。（HS1=GTIU, HS2=GTIV, HS3=GTIW という接続ですと、単純で判り易いのですが）U, V, W 相のコイルの配置とホールセンサーの絶対位置の関係で、どのホールセンサ出力(HS1~HS3)を GTIx に接続するかが決まります。これは、電流方向がハードウェアで決まるので、そのハードウェアの挙動に合わせてモータ駆動回路を構成する必要があるという事かと考えます。（本キットの接続ボードでは、実験的にどの組み合わせでも構成出来る様、JP4~JP6 のジャンパを用意しています。）

※本チュートリアル以外のプログラムは、ソフトウェアでホールセンサパターンと電流印加方向の対応を決められるので、HS1(=PB5), HS2(=PB6), HS3(=PB7)の対応で処理を行っています

・FSP GPT_OPS の設定



FSP の設定は、GPT_OPS をモータ出力端子、ホールセンサ入力端子に割り当てています。

この、GPT_OPS というマイコンの機能を使うと、ほとんどマイコンのハードウェアのみでモータを回す事ができますので、マイコンがこのような機能を持っていて必要に応じて使うことができると認識して頂ければと思います。

・src¥blm.c（blm_init()内　初期設定）

```
//GPT0 : CH-1 PWM
//GPT3 : CH-2 QL PWM
//GPT4 : CH-3 QL PWM
(void) R_GPT_Open(&g_gpt0_ctrl, &g_gpt0_cfg);
(void) R_GPT_Open(&g_gpt3_ctrl, &g_gpt3_cfg);
(void) R_GPT_Open(&g_gpt4_ctrl, &g_gpt4_cfg);

//CH-1をハードで制御する設定(GPT_OPS)
R_GPT_OPS->OPSCR_b.UF = 0;      //入力端子設定有効(U)
R_GPT_OPS->OPSCR_b.VF = 0;      //入力端子設定有効(V)
R_GPT_OPS->OPSCR_b.WF = 0;      //入力端子設定有効(W)
R_GPT_OPS->OPSCR_b.FB = 0;      //外部フィードバック有効
R_GPT_OPS->OPSCR_b.P = 1;       //P側PWM制御
R_GPT_OPS->OPSCR_b.N = 1;       //N側PWM制御
R_GPT_OPS->OPSCR_b.INV = 0;     //アクティブH
R_GPT_OPS->OPSCR_b.RV = 0;      //正回転（仮設定値）
R_GPT_OPS->OPSCR_b.ALIGN = 0;   //入力をGTCLKで読み取り
R_GPT_OPS->OPSCR_b.GRP = 0;     //出力禁止要因選択
R_GPT_OPS->OPSCR_b.GODF = 0;    //グループ出力禁止無効
R_GPT_OPS->OPSCR_b.NFEN = 1;    //入力にノイズフィルタを適用
R_GPT_OPS->OPSCR_b.NFCS = 0x1;  //ノイズフィルタ:GTCLK/4
```

CH-1 PWM 設定

CH-2, CH-3 PWM 設定
（CH-2, CH-3 は、QL 信号 1 本のみ PWM）

マイコンが持つハード(GPT_OPS)でモータを回す設定

・src¥blm.c（blm_init()内　初期設定）

```
void blm_start(unsigned short ch)
{
    //ブラシレスモータ有効化
                                                          CH-1 PWM 設定
    //引数：
    //  unsigned short ch ： 有効化するチャンネル番号          CH-2, CH-3 PWM 設定
    //戻り値：なし                                          （CH-2, CH-3 は、QL 信号 1 本のみ PWM）

    switch(ch)
    {
        case BLM_CH1:
            if (g_target_direction[BLM_CH1] == BLM_CCW)
            {                                                マイコンが持つハード
                R_GPT_OPS->OPSCR_b.RV = 0;      //正回転       (GPT_OPS)でモータを回す設定
            }
            else
            {
                R_GPT_OPS->OPSCR_b.RV = 1;      //逆回転
            }
            //PC15(QU) = PC14(QL) = H
            R_PORT12->PCNTR3_b.POSR = 0xc000;
            //Q1U-Q3U, Q1L-Q3L
            R_GPT_OPS->OPSCR_b.EN = 1;          //GPT OPS出力有効
            (void) R_GPT_Start(&g_gpt0_ctrl);   //本チュートリアルではGPT0でQU1~3，QL1~3の6相をPWM駆動す
る
            break;                                           CH-2，CH-3 は、チュートリアル 7
（[後略]）                                                    と同じ手法で制御しています
```

・シリアル端末から出力される情報

```
                        CH-1      CH-2      CH-3
Motor Driver Board     :  Connect   NoConnect NoConnect
Active                 :     o         x         x
rotation speed([rpm])  :  4500         0         0
target direction       :  CCW       CCW       CCW
Temperature(A/D value) :  2231         0         0
Temperature(degree)    :    30         0         0
VR(A/D value)          :  1927         0         0
QL duty[%]             :  47.0       0.0       0.0
```

基本的には、チュートリアル 7 と変わりませんが、回転方向(target direction)の表示、機能が追加されています。

	ON	OFF
SW1	CH-1 の回転	CH-1 の停止
SW2	CH-2 の回転	CH-2 の停止
SW3	CH-3 の回転	CH-3 の停止
SW4	回転方向時計回り(CW)	回転方向反時計回り(CCW)

SW1~3 を ON に倒したときの SW4 の方向によって回転方向が変わります。CH-1 は GPT_OPS の OPSCR.RV レジスタの値で回転方向を変えています。CH-2, CH-3 は、ホールセンサ位置と電流印加方向を回転方向によって切り替えています。

[参考]正回転(CCW)・逆回転(CW)に対応したホールセンサと電流印加方向のテーブル

・src¥blm_intr.c（agt0_callback(), 50us 周期割り込み関数内）

```
            if (g_target_direction[i] == BLM_CCW)
            {
                //回転方向CCW
                switch (g_sensor_pos[i])
                {
                    case 3:
                        blm_drive[i] (U_V_DIRECTION);
                        break;
                    case 2:
                        blm_drive[i] (U_W_DIRECTION);
                        break;
                    case 6:
                        blm_drive[i] (V_W_DIRECTION);
                        break;
                    case 4:
                        blm_drive[i] (V_U_DIRECTION);
                        break;
                    case 5:
                        blm_drive[i] (W_U_DIRECTION);
                        break;
                    case 1:
                        blm_drive[i] (W_V_DIRECTION);
                        break;
                    default:
                        blm_drive[i] (OFF_DIRECTION);
                        break;
                }
            }
            else if (g_target_direction[i] == BLM_CW)
            {
                //回転方向CW
                switch (g_sensor_pos[i])
                {
                    case 3:
                        blm_drive[i] (W_U_DIRECTION);
                        break;
                    case 2:
                        blm_drive[i] (W_V_DIRECTION);
                        break;
                    case 6:
                        blm_drive[i] (U_V_DIRECTION);
                        break;
                    case 4:
                        blm_drive[i] (U_W_DIRECTION);
                        break;
                    case 5:
                        blm_drive[i] (V_W_DIRECTION);
                        break;
                    case 1:
                        blm_drive[i] (V_U_DIRECTION);
                        break;
                    default:
                        blm_drive[i] (OFF_DIRECTION);
                        break;
                }
            }
```

回転方向：反時計回り(CCW)

ホールセンサ位置 3(HS3..HS1=011)の時
U 相から V 相に電流を流す

回転方向：時計回り(CW)

ホールセンサ位置 3(HS3..HS1=011)の時
W 相から U 相に電流を流す

※1.5 章で[参考]として記載している
電流印加方法

※チュートリアル 5（1.5 章）のプログラムの両方向の回転に対応したバージョンアップ版です

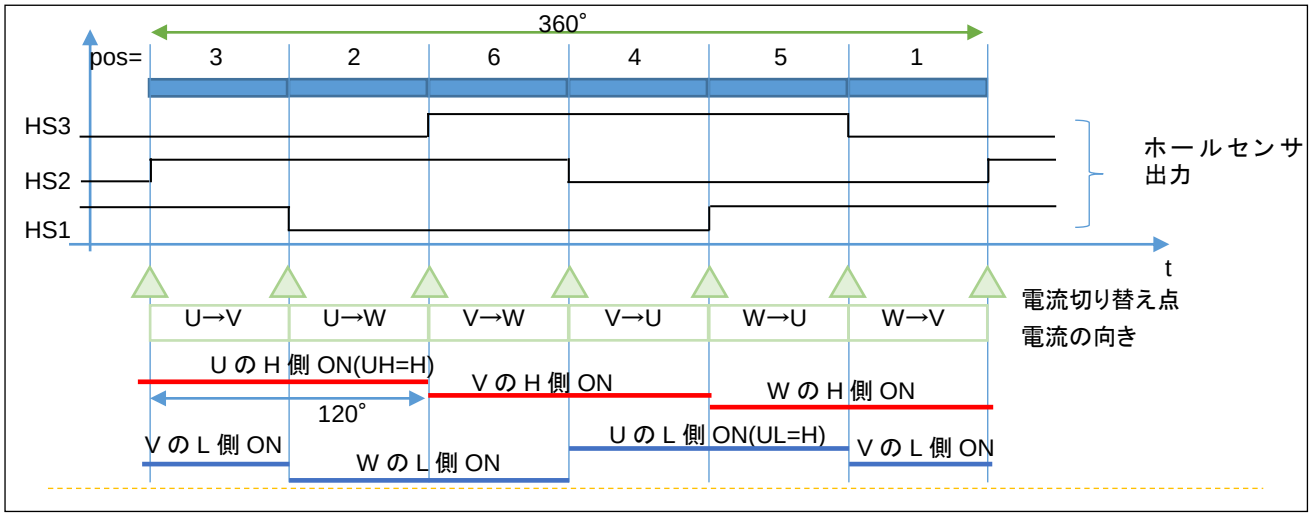
2.2. 相補 PWM 信号での駆動

参照プロジェクト:RA6T2_BLMKIT_TUTORIAL_B

（アーカイブファイル:RA6T1_BLMKIT_TUTORIAL_B.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

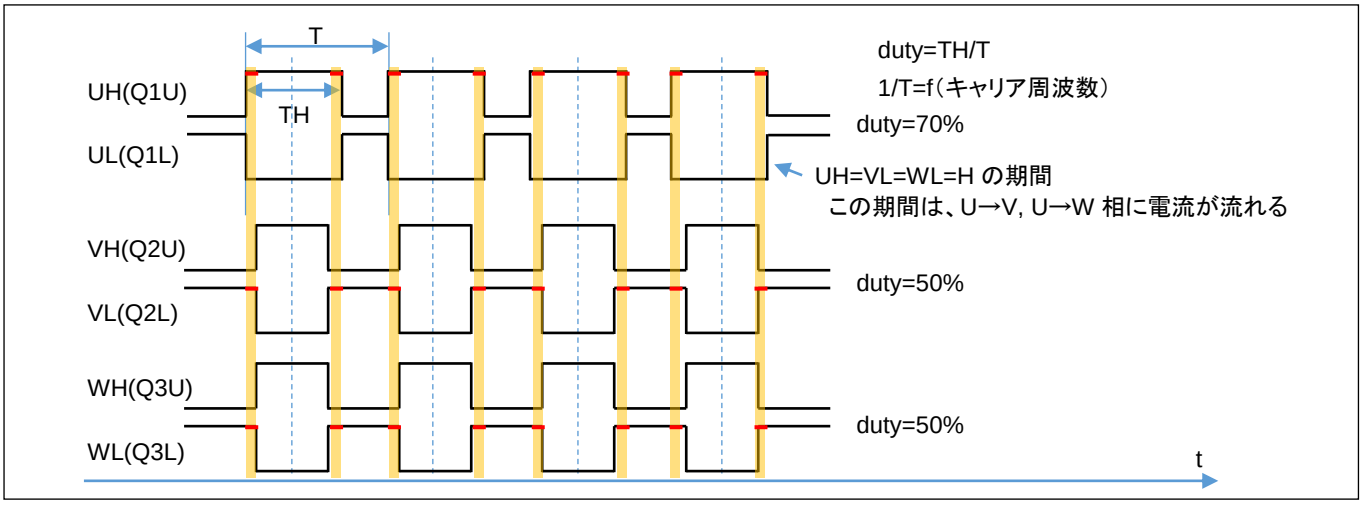
・120 度制御



TUTORIAL7, TUTORIAL_A でモータを駆動している方式は、H 側と L 側の ON 期間を 60°(1/6 周期)ずらし、H 側の ON の期間, L 側 ON の期間をそれぞれ 120°として、電流の方向を切り替えていく方式で、120°制御です。

それに対し、H 側と L 側の波形を反転信号で駆動する方式は、相補 PWM と呼ばれます。

・相補 PWM



U 相 H 側(UH)と U 相の L 側(UL)は、常に逆相で駆動します。V 相と、W 相も同様です。

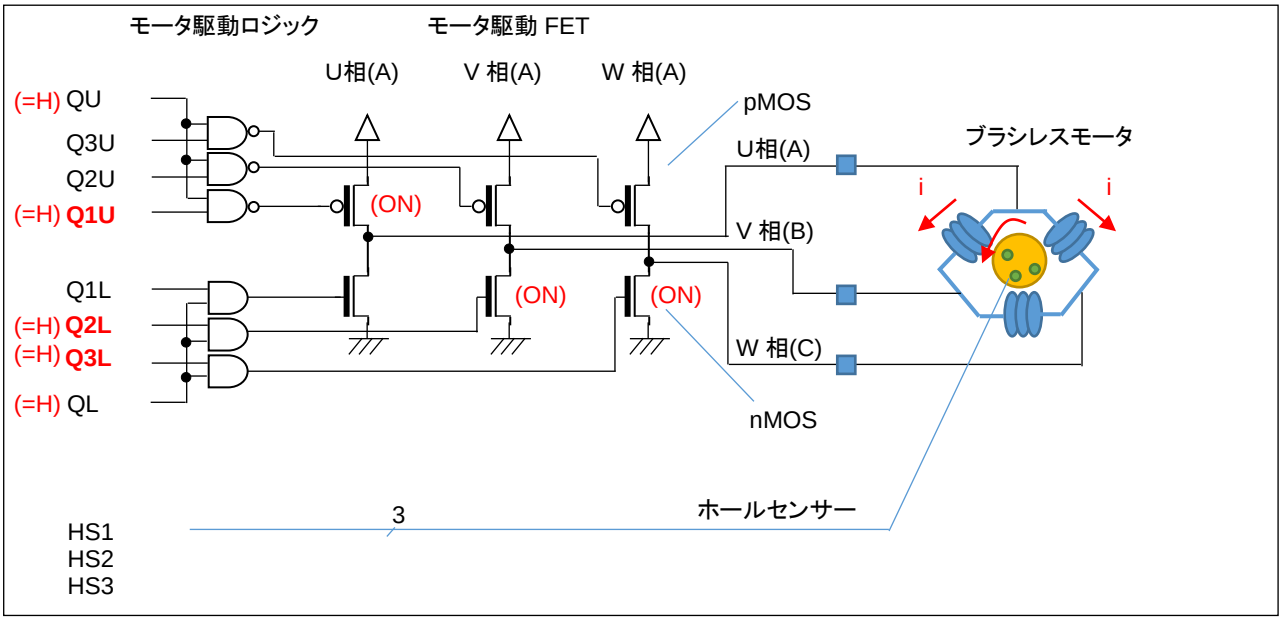
上図の相補 PWM では、

U 相　duty70%
V 相　duty50%
W 相　duty50%

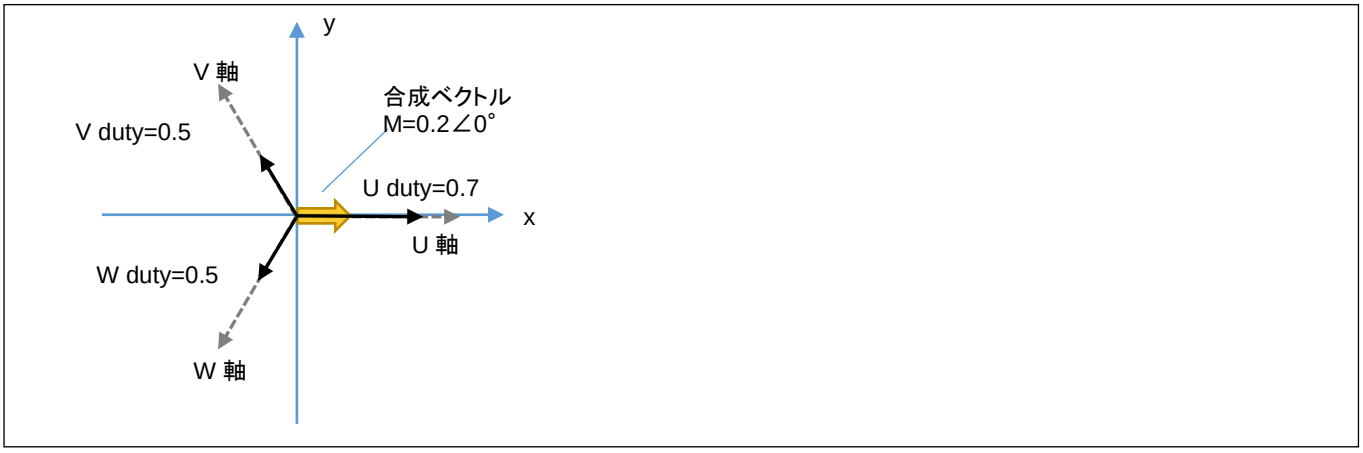
の、PWM 信号となっています。この例では、オレンジ塗りつぶしのタイミングで、（U 相の H 側と V 相の L 側と W 相の L 側が ON しており）

U 相 H　→　V 相 L
U 相 H　→　W 相 L

に電流が流れます。120°　制御では、電流の流れるパスは 1 通りでしたが、相補 PWM では、基本的に 2 通りのパスがあります。（duty の高い相から duty の低い相への電流が生じる。）



ここで、duty の差分（70-50=20%）の時間だけ、U→V, U→W に電流が流れる事となります。



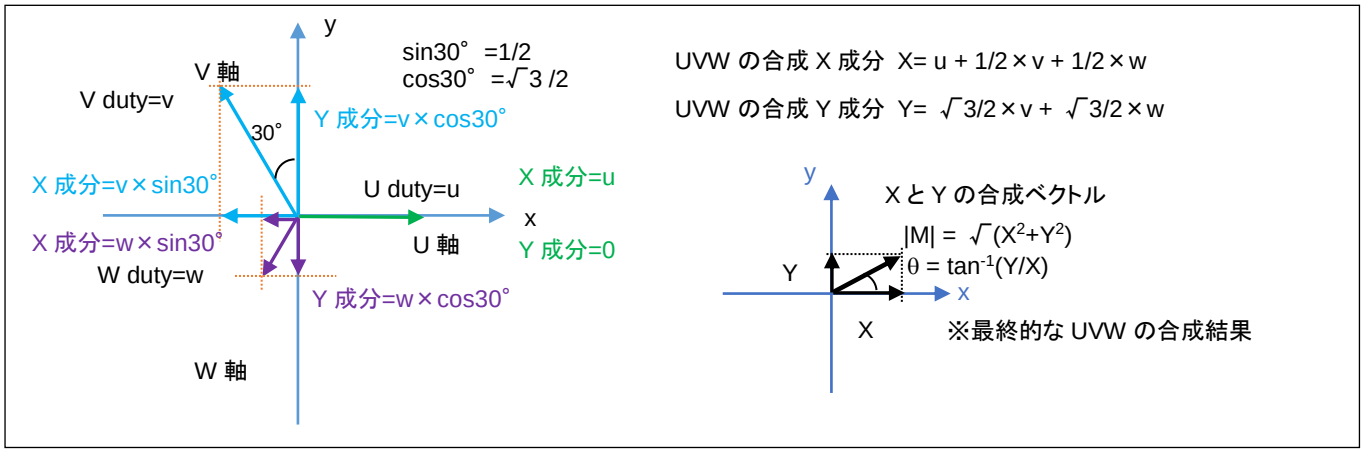
図で考えると、U,V,W 相は 120° の差分があり、それぞれの方向に 0.7,0.5,0.5 の強さで引っ張っているイメージです。

U=0.7, V=0.5, W=0.5 の強さで引っ張ると、この場合の合成ベクトルは、U 軸方向に 0.2 の力で引っ張る事となります。

U, V, W の 3 相の duty を調整する事により、「印加する磁界の大きさ（＝電流の大きさ）」と「どの向きに磁界を印加するか（＝UVW のどの方向に電流を流すか）」を自由に設定できます。

先の例では、3 本のベクトルの合成ベクトルは、U 軸方向に 0.2(20%)となります。

ここで、x 軸に U 軸を重ねる様にし、V 軸を 120° 、W 軸を 240° の位置に取ります。



$$x = U - \frac{V}{2} - \frac{W}{2}$$

$$y = (V - W)\frac{\sqrt{3}}{2}$$

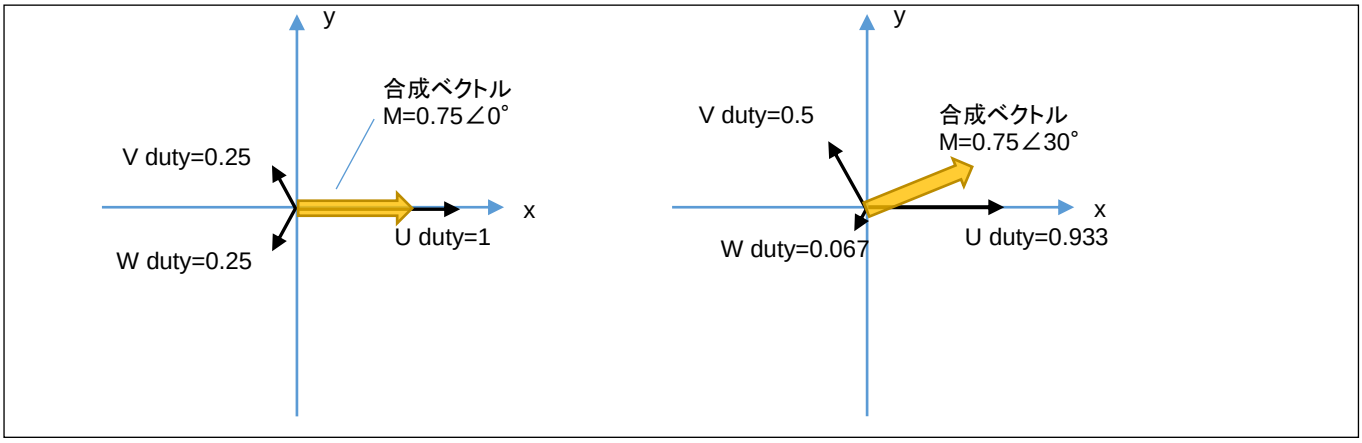
U, V, W の大きさから、x-y 軸（直交座標系）のベクトルに変換することができ、U,V,W の合成ベクトルを M、x 軸を基準とした角度をθとすると、

$$|M| = \sqrt{x^2 + y^2}$$

$$\theta = tan^{-1}\frac{y}{x}$$

となります。

ここで、U 相の duty を 1 として、V, W 相の duty を 0.25 とすると、合成ベクトルとしては、U 軸方向に 0.75 となります。同様に V,V,W=(0.933, 0.5, 0.067)とすると、合成ベクトルは強さは 0.75 で角度は U 軸から 30°ずれた位置となります。



120°制御の場合は、60°単位での切り替え（1 回転で 6 段階、電流の流れる方向＝磁界の方向は 6 パターンしか存在しない）となりますが、相補 PWM では磁界の向きをもっと細かな角度で切り替える事が可能です。

本チュートリアルのプログラムの動作としては以下となります。

SW1~SW3 のスイッチを OFF に、VR を絞った状態で電源を投入してください。
モータドライバボードを接続した CH のスイッチを ON にしてください。
VR を上げてみてください。どこかでモータが回りだすタイミングがあるはずです。

プログラム動作としては、TUTORIAL4 と同じです。回転数は 2000rpm 固定で、VR により duty を変化させていくプログラムとなっています。（モータの制御に、ホールセンサの値は、使っていません。）

合成ベクトルの大きさ(|M|)=duty は、VR の回転角度に応じて変化します。合成ベクトルの角度(θ)は、50us 毎に動かしていきます。

2000rpm / 60 = 33.3 回転/s
1/33.3 = 30ms …1 回転（2π[rad]で 30ms かかる）
30ms / 50us ×2π = 10.47×10^{-3}[rad]

本チュートリアルでは、50us(AGT0)の周期割り込みで、磁界の印加角度（合成ベクトルの角度）を変えていく処理としているので、50us 毎に印加角度を 10.47×10^{-3}[rad]ずつ変化させていけば良い（＝2000rpm の速度で回転するように印加する磁界を動かす）事となります。

回転数は固定で duty を VR のツマミの回転に応じて変化させる手法は、TUTORIAL4 と同様です、

duty（合成ベクトルの大きさ|M|）が小さいときはモータは回転せず、大きすぎても効率が下がります（このあたりの関係も TUTORIAL4 と同様です）。最適な duty のとき、モータはスムーズに回ります。

モータを制御する側からすると、

・合成ベクトルの大きさ|M|
・合成ベクトルの角度θ

を与えたいのですが、マイコン側からすると、

・U 相の duty(0~100%)
・V 相の duty(0~100%)
・W 相の duty(0~100%)

の 3 値を与える事となります。

|M|, θを与えた際、U,V,W のそれぞれの強さに分解する方式は一通りではないのですが、本チュートリアルのプログラムでは、正弦波駆動（UVW がそれぞれ、正弦波で変化）としています。

U(duty) = (|M| ・cos θ) /2 + 0.5
V(duty) = (|M| ・cos (θ-120°)) /2 + 0.5
W(duty) = (|M| ・cos (θ-240°)) /2 + 0.5

（0-1 の範囲に正規化）

|M|, θの値から上式で、U,V,W のそれぞれの duty 値に変換しています。

－合成ベクトルの UVW への分解に関して－

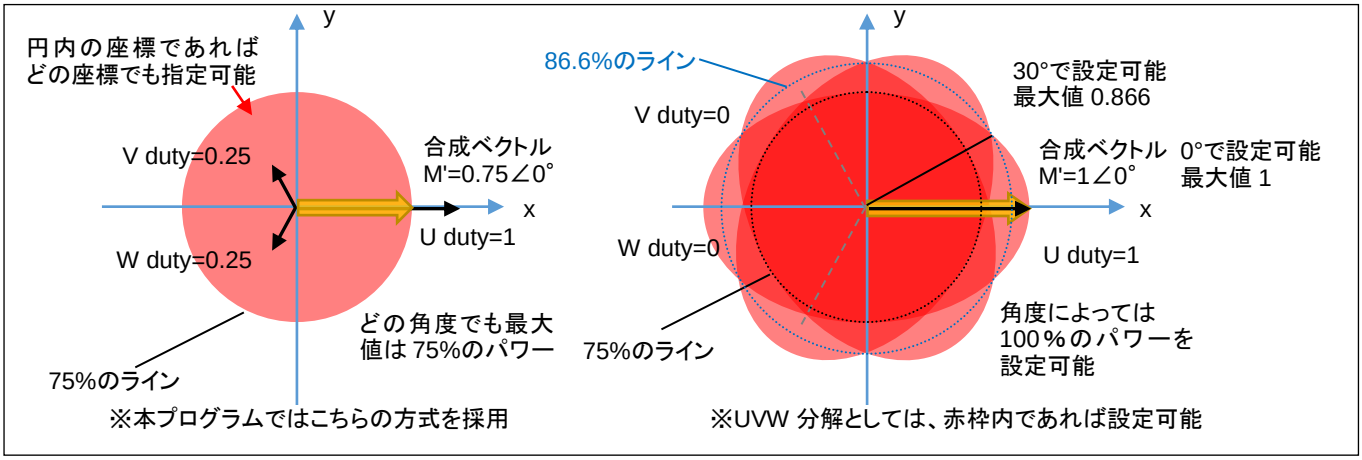
上記手法、計算式（正弦波駆動）を用いると、|M|=1, θ=0 を与えて各相の duty を計算すると、
(U, V, W) = (1.0, 0.25, 0.25)
となり、この合成ベクトルは、
0.75∠0°
となります。0°（U 軸）方向に最大限のパワーを与えたい場合、結果的には 75%のパワーで 0° 方向に力を印加する事となります。

同様に、|M|=1, θ=30° での各相の duty は、
(U, V, W) = (0.933, 0.5, 0.067)
となり、この合成ベクトルは
|M'|∠θ = 0.75∠30°
です。30° 方向に最大限のパワーを与えたい場合でも上記同様、結果的には 75%(=|M'|)のパワーとなります。
（入力として与えた、|M|が|M'|に変換されます）

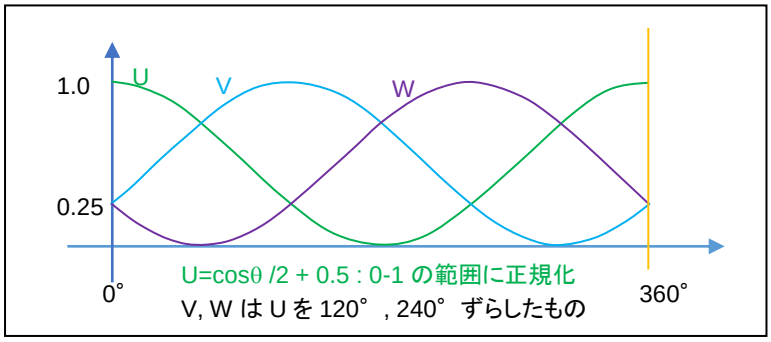
本手法で、UVW の分解を計算した場合、どの角度においても、最大値は 0.75 になります。（0.75 に制限されます）

例えば、3 相の duty を以下の様にすれば、
(U, V, W) = (1.0, 0.0, 0.0) → 1.0∠0°
となります。（|M'| = |M|とする事が可能です）

別な計算方法を用いると、0° 方向に 100%のパワーを与える事も可能です。但し、全角度に対して、100%のパワーを与える事は、UVW（0°, 120°, 240°）の 3 本のベクトルの合成では不可能です。（100%のパワーを与えることのできる角度は、0°, 120°, 240° とその反対側の 60°,180°,300° のみ。）
30° では、(U, V, W) = (1.0, 0.5, 0.0) → 0.866∠30°、86.6%が理論上の最大パワーとなります。

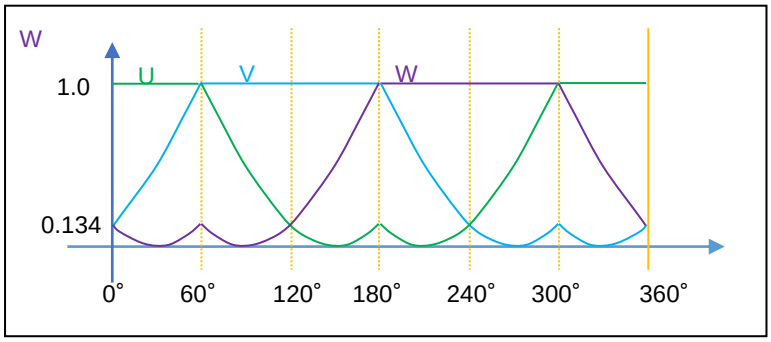


・本プログラムの UVW 分解（正弦波駆動）



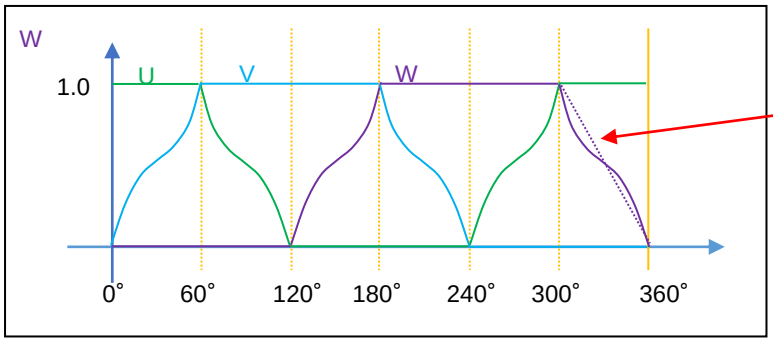
本プログラムの UVW 分解は、各相を正弦波で連続的に変化させる手法で、最大のパワーがどの角度においても、75%に制限されるが、UVW の変化に連続性があり、三角関数（コサイン）の計算は必要であるが、計算式は単純です。コサインは-1~1 の範囲の値を取りますので、PWM duty（設定可能なのは 0~100%, 0~1）に対応させるために、2 で割り 0.5 を加算して、0~1 の範囲にシフト（1 に正規化）させています。

・UVW 分解（別バージョン）



例えばですが、どの角度でも設定可能な最大のパワーは、86.6%(=√3/2)であると考えます。上記の様な UVW のカーブとすると（かなり変則的なカーブですが）、設定可能な最大パワーが 75→86.6%に増加します。（但し領域で分けて三角関数と、1 固定で UVW 値を計算する必要があります。）

・UVW 分解（別バージョン 2）



計算を簡単にするためにこのカーブを直線近似するという「別バージョン 2'」も考えられます
（直線近似した場合、入力の角度に対し、UVW 変換した結果の角度は最大 1.2° 程度ずれますが、それ程大きなずれにはなりません）

上記別バージョンでも、モータに 100%の電力を与える事はできません。（120 度制御では、刻みは 60° であるが、最大 100%の電力を与える事ができる。）前ページの右図の赤枠の外周をなぞるように UVW 分解を行うと、0,60,120,180,240,300° の位置では 100%の電力を与える事が可能です。

別バージョン 2 では、1∠0°→1∠0°（0°方向には 100%のパワーで引っ張る）1∠30°→0.866∠30°（30°方向には 86.6%のパワーになってしまう）変換です。

入力 （プログラムで与える |M|と角度）	UVW 分解の結果（|M'|と角度）		
	正弦波変換 （全角度に 75%のパワー） ※本チュートリアルで採用	別バージョン （全角度に 86.6%のパワー）	別バージョン 2 （角度により上限を変える） [別バージョン 2'（直線近似）]
1∠0°	0.75∠0°	0.866∠0°	1∠0°　[1∠0°]
1∠10°	0.75∠10°	0.866∠10°	0.922∠10°　[0.928∠8.9°]
1∠20°	0.75∠20°	0.866∠20°	0.879∠20°　[0.882∠19.1°]
1∠30°	0.75∠30°	0.866∠30°	0.866∠30°　[0.866∠30°]
0.1∠0°	0.075∠0°	0.0866∠0°	0.1∠0°
0.1∠30°	0.075∠30°	0.0866∠10°	**0.0866∠30°　(*1)**

別バージョン 2 では角度により最大パワーが異なる（120 度駆動と PWM のハイブリッド？）の様な方式です。

(*1)この部分の変換は、0.1∠30°にする事も可能です。（考え方次第です）角度により設定上限が異なるだけで、上限に達しない範囲では入力の duty 値を変えずに変換するという考え方も取り得ます。

－合成ベクトルの UVW への分解に関して(2)－

正弦波駆動の UVW 分解で、

$U=(\cos\theta \times |M|) / 2 + 0.5$　(1)

（V, W は q に 120°, 240°を加えて算出）

/ 2 + 0.5 は cos（-1~1 を取る）を 0~1（各相の duty として設定できる範囲）に変換する操作です（1 に正規化）。(1)式では、ベクトルの大きさ|M|を乗算した後で、正規化を行っています。（本プログラムで採用している計算式）

$U=\{(\cos\theta \times \mathbf{1}) / 2 + 0.5\} \times |M|$　(2)

ここで、U の値は|M|=1 で計算し、正規化後に|M|を乗算するとどうなるか考えてみます。

|M|=1（入力の duty=100%）の時は、(1)と(2)で計算結果は変わりません。

|M|=0.5, θ=30°のケースで考えてみます。

(1)式で U, V, W を計算すると、(U, V, W) = (0.717, 0.5, 0.283)　(1')

(2)式で U, V, W を計算すると、(U, V, W) = (0.467, 0.25, 0.033)　(2')

となります。

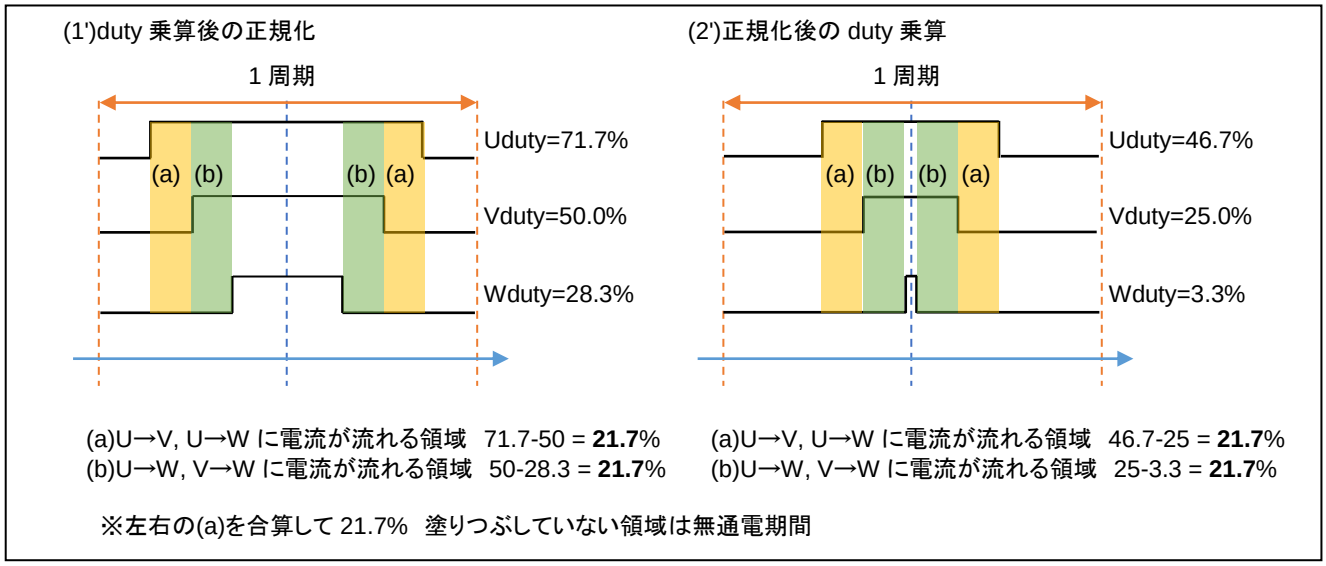
(1')と(2')で、UVW 各相の値は異なりますが、合成結果は

|M'| = 0.375
θ = 30°

で変わりません。

角度は、（当たり前ですが）入力の通り。ベクトルの大きさは、入力の 75%になるので、0.5×0.75=0.375 です。

この、(1')と(2')の違いは？



1 周期における電流の流れるタイミングは異なります。2 相間に流れる電流の大きさは同じです。

※上図は、1 周期（キャリア周波数 40kHz の場合は、25us）を示しており同じ波形が繰り返されるイメージです（50us 毎に角度を変えていくので、各相の duty は徐々に変化しますが、基本的には 1 周期の左右に同じ 1 周期の波形が来るイメージです。）

(1')(2')で電流が流れない時間は、
(1') : 100-71.7 + 28.3 = 56.6%
(2') : 100-46.7 + 3.3 = 56.6%
と同じですが、
(1') : 28.3%と 28.3%で分割
(2') : 53.3%と 3.3%で分割
となります。(2')の場合は、(b)と(b)の間がほとんど空かず、電流の流れない時間が長く続きます（この例だと 53.3%）。

この、(1')と(2')の違いが、モータのトルクや回転の滑らかさ、消費電力等に影響を与える事は考えられます。

しかし、基本的には、モータにどの程度の電力を与えるか、どの方向に引っ張るかという事が重要ですので、UVW 分解法における無電流期間の分布はそれ程大きな問題にはならないと考えます。

（PWM のキャリア周波数を変える事でも無電流期間と電流を流す期間の分布は変わります。…周波数を上げると全体が圧縮される。キャリア周波数の変更と、(1')と(2')どちらの式とするかは同じような影響かと思います。）

・通電期間の時系列(1')

	n-1 周期	←			1 周期			→	n+1 周期
	無通電	無通電	(a)	(b)	無通電	(a)	(b)	無通電	無通電
(1')	14.15%	14.15%	10.85%	10.85%	28.3%	10.85%	10.85%	14.15%	14.15%
	28.3%		21.7%		28.3%	21.7%		28.3%	
									→t

周期の 28.3%無通電、21.7%通電の繰り返し。

・通電期間の時系列(2')

	n-1 周期	←			1 周期			→	n+1 周期
	無通電	無通電	(a)	(b)	無通電	(a)	(b)	無通電	無通電
(2')	26.65%	26.65%	10.85%	10.85%	3.3%	10.85%	10.85%	26.65%	26.65%
	53.3%		21.7%		3.3%	21.7%		53.3%	
									→t

周期の 53.3%無通電、21.7%通電、3.3%無通電、21.7%通電、53.3%無通電の繰り返し。

※塗りつぶしが通電期間

通電時間と無通電期間のバランスが取れるのが(1')の方式。短い無通電期間が中央（＝三角波 PWM の頂点）に来るのが(2')の方式です。

なお、30° で duty を変える場合、

	(1')	(2')
0.1∠30°	(0.543, **0.5**, 0.457)	(0.093, 0.05, 0.007)
0.2∠30°	(0.587, **0.5**, 0.413)	(0.187, 0.1, 0.013)
0.3∠30°	(0.630, **0.5**, 0.370)	(0.280, 0.15, 0.02)

(1)式を使った場合、V 相の duty は常に 50%となります。この場合、0.5∠30° の場合同様に、無通電の期間が 1/2 分割されて分布する事となります。（バランスの関係は、上記 0.5∠30° と同様になります。）

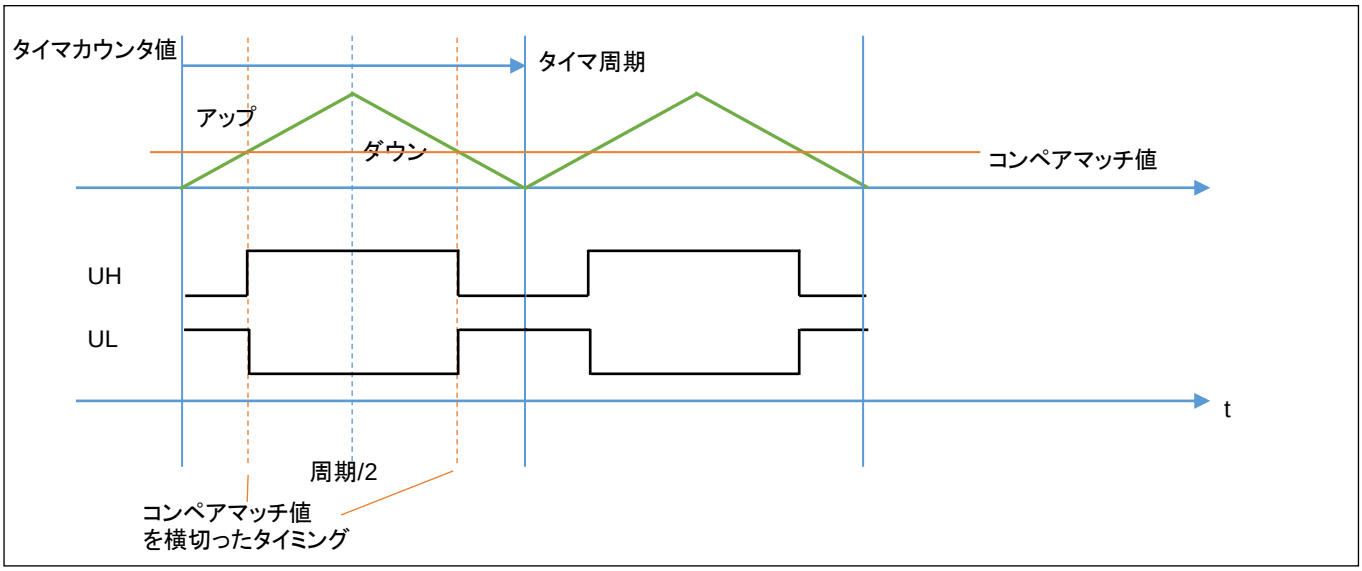
本チュートリアル、サンプルプログラムでは、(1)式を使い(1')になる様な分解法ですが、duty, θを UVW 相に分解する方法は無数に存在します（そもそも正弦波変換なのか、他の変換方法を採るのか。正規化前に duty を乗じるのか、正規化後に乗じるのか、です）。どの手法が正解なのかは、一概にはなんとも言えません。

本キットでは、最大パワーが 75%に制限される UVW 分解法としてますが、この場合 120 度制御に比べて最高回転数が劣ります。最高回転数を稼ぐ事を目的とするのであれば、120 度制御とするか、相補 PWM では UVW の分解方法がポイントになるかと考えます。(UVW 分解方法は、モータ制御の目的に応じ色々なバリエーションが考えられます。)

※サンプルプログラム内(*1)には、
「正弦波駆動」(デフォルト) `blm_angle_to_uvw_duty`
コメントアウトで
(2)式を使った正規化後に duty を乗じる方式 `blm_angle_to_uvw_dutyx`
「別バージョン 1」(全角度に 86.6%のパワー) `blm_angle_to_uvw_duty1`
「別バージョン 2」(60° 刻みで 100%のパワー) `blm_angle_to_uvw_duty2`
「別バージョン 2'」(別バージョン 2 の直線近似版] `blm_angle_to_uvw_duty2x`
を用意しています。
(blm.c, blm_dutyset()関数内で、blm_angle_to_uvw_duty()関数を呼び出している部分を変更)
(サンプルプログラムでは、UVW 分解法の動作の違いを見ることができます。)

(*1)チュートリアルではなく、
RA6T2_BLMKIT_SAMPLE
プロジェクト内に含まれています。

・相補 PWN 波形をカウンタを使って生成する方法



相補 PWM を構成するタイマは、周期の 1/2 まではアップカウント、周期の 1/2 から 1 周期まではダウンカウントとなる動作です。設定したコンペマッチ値を横切った際、出力は反転します。周期は固定とし、コンペアマッチ値を、U, V, W でそれぞれ別な値とすることで、U, V, W それぞれの相で別個の duty の矩形波を得ることができます。

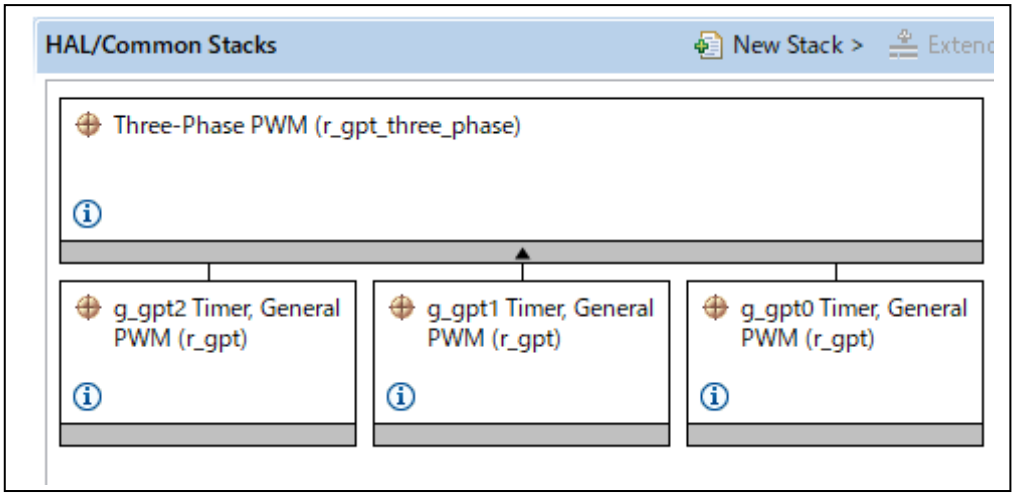
U_コンペアマッチ値 = (1.0 - U(duty)) × 周期/2 （U(duty)は 1 に正規化済みの値）
※V, W も同様

	CH-1	CH-2	CH-3
U 相	GPT2	GPT4	GPT7
V 相	GPT1	GPT5	GPT8
W 相	GPT0	GPT6	GPT9

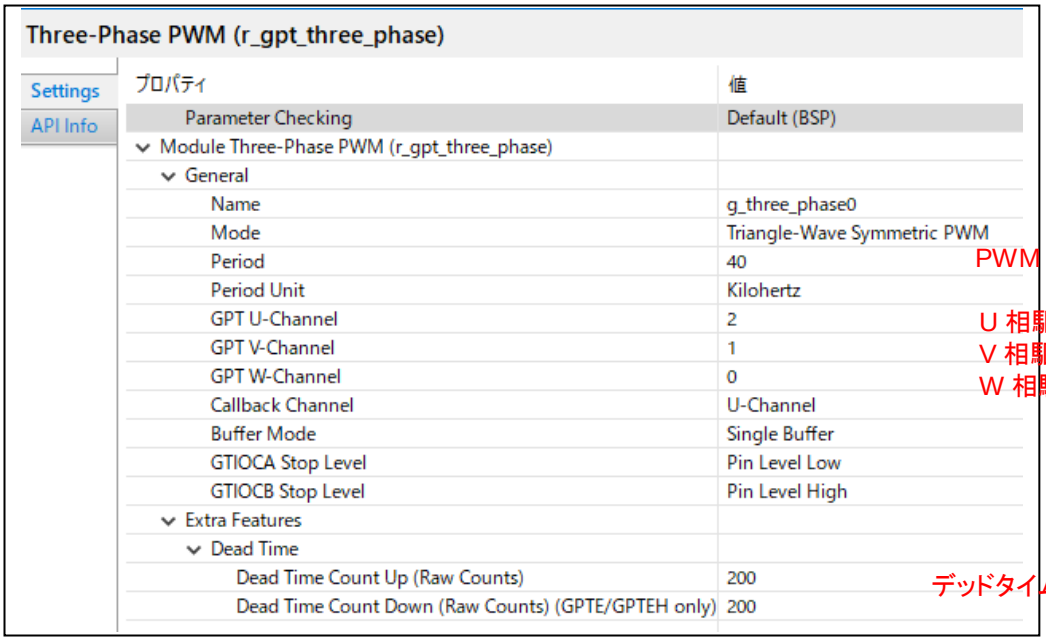
各 CH のタイマは、上記を使用しています。

※3 相の相補 PWM の生成には、タイマを 3 つ使用します

FSP では、Three-Phase PWM Driver という API が用意されています。



前出のタイマの様に、New Stack から、プロジェクトに追加可能です。

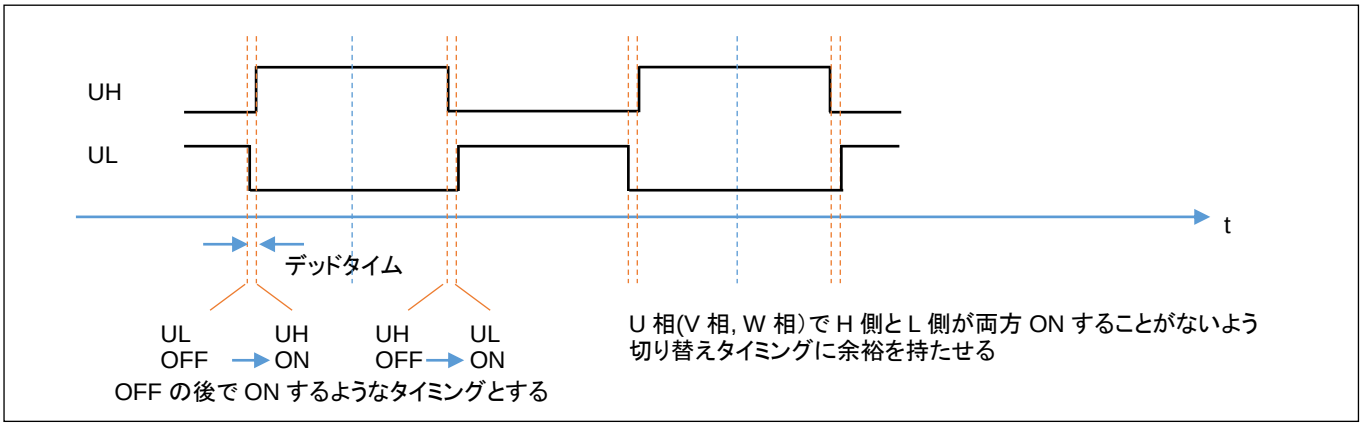


PWM 周波数は 40kHz

U 相駆動には、GPT2
V 相駆動には、GPT1
W 相駆動には、GPT0 を割り当て

デッドタイムも指定可能

Three-Phase PWM Driver では、デットタイムを設定しています。
これは、相補 PWM で、Posi（=UH 側信号）と Nega（=UL 側信号）を、単に反転させる訳ではなく、時間差を付けて切り替える制御となります。



UH の信号と UL の信号が同時に ON すると、電流はモータのコイルではなく、出力回路部の MOS FET のみに流れます（電源が pMOS－nMOS を経由して GND にショート）ので、その様な状態を避けるための制御となります。

3 相の相補 PWM 駆動を行う場合は、使用するタイマ(GPT)は、1 個のモータあたり 3 つ必要ですが、マイコンから見るとタイマはハードウェア（CPU リソースを消費することなく、一定タイミングで切り替わる）ですので、マイコン側のプログラムとしては、適切なタイミングで PWM の duty（UVW の 3 相の duty には、ベクトルの大きさ|M|と角度θの情報が含まれます）を設定すれば良い事となります。

ブラシレスモータの制御としては、単純な 120° 制御と、相補 PWM を用いたベクトル制御の 2 通りがメジャーな制御方式です。本チュートリアルのプログラムは、ホールセンサは回転数の算出に用いているだけで、回転の制御には使っていません。TUTORIAL_B2 では、相補 PWM とホールセンサを組み合わせて、モータを制御しています。

TUTORIAL5 に対応した相補 PWM が次の TUTORIAL_B2 となります。

※デッドタイムを指定すると、設定 duty には誤差が生じますが、本プログラムではデッドタイム指定に伴う誤差の補正は行わない事とします。

2.3. 相補 PWM 信号での駆動（ホールセンサ使用）

参照プロジェクト：RA6T2_BLMKIT_TUTORIAL_B2

（アーカイブファイル：RA6T1_BLMKIT_TUTORIAL_B2.zip）

をワークスペースに展開してください。1.1 章同様に、マイコンボードにプログラムを書き込んで実行してください。

※チュートリアル A で JP4~6 を変更した場合は初期値(GTIU=HS1, GTIV=HS2, GTIW=HS3)に戻してください

TUTORIAL_B では、VR のツマミを動かすとスムーズに回転する領域がありますが、duty を増やしても回転数が増加する事はありません（TURORIAL4 の相補 PWM 版です）。

duty を増やすと回転数が duty に応じて増加する（TUTORIAL5 の相補 PWM 制御版）のが、本チュートリアルです。

ー制御方式に関してー

	制御方式	ホールセンサ （回転制御に使用しているか）
TUTORIAL4（1.5 章）	120°+PWM	未使用
TUTORIAL5（1.6 章）	120°+PWM	使用
TUTORIAL_B（2.2 章）	相補 PWM	未使用
TUTORIAL_B2（本章）	相補 PWM	使用

・チュートリアル B での src¥blm¥blm_intr.c（50us 割り込み関数内）

```
if (g_state[i] == BLM_CH_STATE_ACTIVE)
{
    //UVWの磁界印加割合を設定
    blm_dutyset(i, g_angle[i], g_duty[i]);

    //印加磁界角度を進める
    if (g_target_direction[i] == BLM_CCW)
    {
        g_angle[i] += g_angle_diff[i];
        if (g_angle[i] > PI2)
        {
            g_angle[i] -= PI2;
        }
    }
    else if (g_target_direction[i] == BLM_CW)
    {
        g_angle[i] -= g_angle_diff[i];
        if (g_angle[i] < 0.0f)
        {
            g_angle[i] += PI2;
        }
    }
```

g_angle_diff[i] = 10.47×10^{-3}

50us 毎に決まった角度
（2000rpm に相当する 10.47×10^{-3}[rad]）
を加算する

角度が際限なく大きくなるといずれオーバフロー するので、PI2（定数で 2π）に制限する

逆回転の場合は、角度を減算する

チュートリアル B では角度の増分は、常に一定値（2000rpm に相当する角度）としています。そのため、回転した場合の回転数は設定した duty に拘わらず、大体 2000rpm です。

それに対し、本チュートリアルではホールセンサの値を見て、
(1)50us 毎の角度増分を決定する（回転が速いときは角度増分を増やす）
(2)相補 PWM の印加角度(θ)をホールセンサの位置に合わせる
という処理を行っています。

・チュートリアル B2 での src¥blm¥blm_intr.c（50us 割り込み関数内）

```
            //回転速度の算出
            if (prev_sensor_pos[i] != g_sensor_pos[i])
            {
                //センサ位置が変化
                g_rotation_counter[i] = rotion_counter[i];
                rotion_counter[i] = 0;                          //ホールセンサ位置が変化したタイミングでリセット

                if (g_state[i] == BLM_CH_STATE_ACTIVE)
                {
                    //理想位置を算出
                    ideal_angle = blm_ideal_angle(g_sensor_pos[i], g_target_direction[i], g_angle_forward[i]);
                                                                ホールセンサが切り替わった際の理想的な角度を算出

                    //sci_write_str("¥n *** ");
                    //sci_write_int16((short)((ideal_angle - g_angle[i])/PI*180.0f));
                    //sci_write_str(" ***¥n");

                    //速度のずれを g_angle_diff に反映
                    g_angle_diff[i] = blm_angle_diff_calc(g_angle_diff[i], ideal_angle, g_angle[i], g_target_direction[i]);
                                                (1)理想的な角度と現在の角度を比較して 50us 毎の角度増分値を
                                                理想的な角度となるように近づけていく
                    //印加角度をセンサから算出される理想位置にずらす
                    g_angle[i] = ideal_angle;               (2)印加角度を理想値に上書き
                }

                prev_sensor_pos[i] = g_sensor_pos[i];           //現在のホールセンサ位置を保存
            }
```

blm_ideal_angle()関数はホールセンサの位置（チュートリアル 4 の pos=1~6）に応じた、そのタイミングでの理想的な角度を算出する関数です。

・src¥blm¥blm.c（blm_ideal_angle()関数内）

```
    if(direction == BLM_CCW)
    {
        switch(pos)
        {
            case 3:
                ret = RAD_330_DEGREE;
                break;
            case 2:
                ret = RAD_30_DEGREE;
                break;
            case 6:
                ret = RAD_90_DEGREE;
                break;
            case 4:
                ret = RAD_150_DEGREE;
                break;
            case 5:
                ret = RAD_210_DEGREE;
                break;
            case 1:
                ret = RAD_270_DEGREE;
                break;
            default:
                return 0;
                break;
        }
```

RAD_330_DEGREE = 2π/360×330= 5.76 [rad]
…330°をラジアン変換した値

単純にホールセンサ位置と角度の対応テーブルです。

・src¥blm¥blm.c

```
float blm_angle_diff_calc(float diff_angle, float ideal_angle, float angle, short target_direction)
{
    //制御周期(50us)毎の角度増分を計算する関数

    //引数
    //  diff_angle 現状の角度差分
    //  ideal_angle センサ切り替わり時の理想的な角度
    //  angle 現状の角度
    //  target_direction 回転方向

    //戻り値
    //  計算後の diff_angle

    /*
     * ideal_angle = angle
     * となる様に、diff_angleを微調整する
     *
     * ideal_angle - angle が
     *
     *  プラス : 現状のdiff_angleが遅い
     *  マイナス : 現状のdiff_angleが速い
     */

    float angle_sub;

    angle_sub = ideal_angle - angle;

    //PI(180[°])より大きな場合はdiff_angleを変更しない
    if (angle_sub > PI)
    {
        return diff_angle;
    }

    //-PI2(-360[°])より小さな場合はdiff_angleを変更しない
    if (angle_sub < -PI2)
    {
        return diff_angle;
    }

    //角度は、-PI ~ PI（-180°~180°）の範囲内に変換する
    if (angle_sub < -PI) angle_sub += PI2;

    //理想との差分をBLM_ANGLE_DIFF_FEEDBACKの割合で埋めていく

    return diff_angle + angle_sub * BLM_ANGLE_DIFF_FEEDBACK * target_direction;
}
```

PI = π = 3.141592...（円周率）

PI2 = 2π

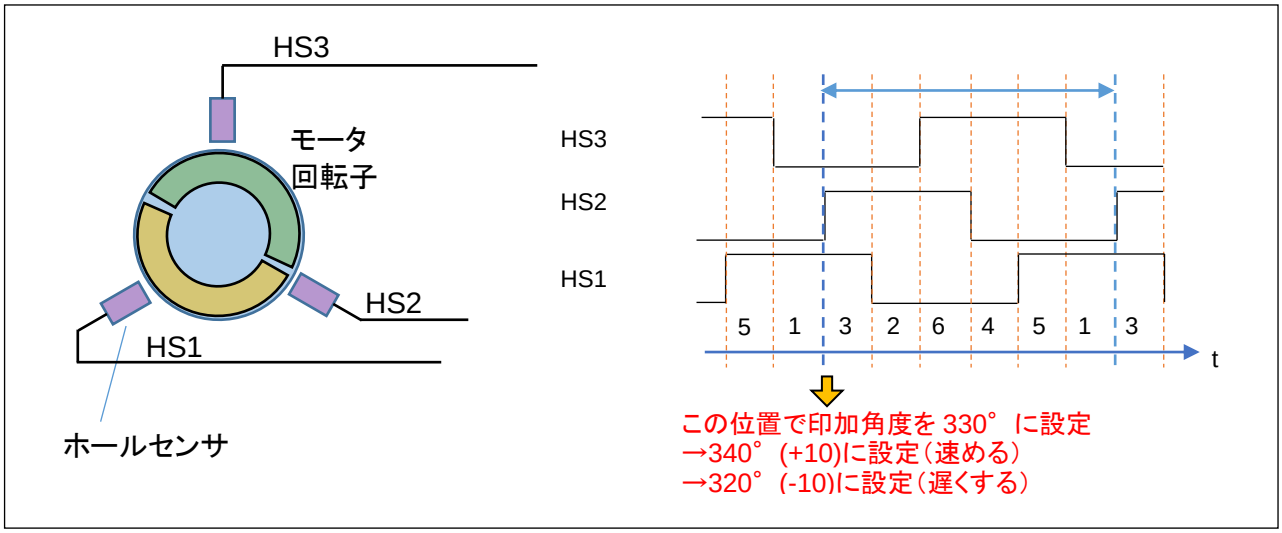
=0.01f (1%)

50us 毎の角度増分を決定する部分は、現状の角度増分(diff_angle)に対し、理想値とのずれ(angle_sub)を加算するのですが、1 回で理想値にしてしまうのではなく、BLM_ANGLE_DIFF_FEEDBACK(=0.01)(1%)ずつ差分を埋めていく（diff_angle を滑らかに変化させる）方式です。

・シリアル端末から出力される情報

```
                          CH-1      CH-2      CH-3
Motor Driver Board       :  Connect   NoConnect NoConnect
Active                   :     o         x         x
diff angle -> speed([rpm]):  1320      1980      1980
forward angle([deg])     :     0         0         0
target direction         :   CCW       CCW       CCW
rotation speed([rpm])    :  1440         0         0
Temperature(A/D value)   :  2349         0         0
Temperature(degree)      :    34         0         0
VR(A/D value)            :   965         0         0
QL duty[%]               :  23.6       0.0       0.0
```

基本的には、いままでのチュートリアルと大きくは変わりませんが、進角(forward angle)の表示、機能が追加されています。進角調整はホールセンサ位置と磁界の印加角度の関係を調整する機能です。



ホールセンサの出力が切り替わるタイミングで磁界の印加角度を設定していますが、この角度を速めたり遅くしたりする機能が進角調整です。この進角の値は、シリアル端末のキーボードで設定を行います。

	+1°	-1°	リセット(=0)
CH-1	q	w	e
CH-2	a	s	d
CH-3	z	x	c

CH-1 のモータは、キーボードから'q'を入力すると角度が 1° 速くなります。'w'で 1° 遅くする方向です。'e'で初期値(=0)に戻します。調整範囲は±45° の範囲です（blm.h 内で定義、変更は可能）。モータが停止しているときでも変更は可能です。一般に高速回転時は、進角を+の方向（エンジンでしたら点火タイミングを速めるイメージでしょうか、モータであれば磁界の引っ張る角度を少し前の方に設定する）にすると、消費電流が減る（効率が上がる）事が期待できます。

ー三角関数(cos)の計算に関してー

相補 PWM のプログラムでは、制御周期(50us)毎に cos の計算を行って UVW 相の duty（UVW 分解）を計算しています。以前の RA6T1 向けのサンプルプログラムでは、cos はプログラムの先頭で 0-360° の範囲で 1° 刻みで計算してテーブル化して、cos の値はテーブル参照で利用していました。それに対し、RA6T2 のプログラムではリアルタイムで計算しています。

これは、RA6T2 は三角関数の計算をハードで行う機能（TFU:三角関数ユニット）を持っているためです。cos の計算は、14 サイクルで計算できるので、都度計算させても処理が間に合わなくなる事がありません。

・src¥blm¥blm.c（blm_angle_to_uvw_duty()関数内）

```
    //COS計算 （TFU使用）
    /*
     * FSP RA6T2の環境では、cosf(angle) を使ってもTFU （三角関数ハードウェア演算） が使用されます
     * cos(angle)は、倍精度浮動小数点計算、ソフトウェアライブラリでの計算となり極端に計算速度が落ちますので
注意
     */
    R_TFU->SCDT1 = angle;   SCDT1 に角度(rad)を書く

    phase_duty->u = R_TFU->SCDT0 * duty / 2.0f + 0.5f;  //U相のduty値 SCDT0 に結果が格納される

    R_TFU->SCDT1 = angle - RAD_120_DEGREE;              //U相から120°ずれた点をV相とする

    phase_duty->v = R_TFU->SCDT0 * duty / 2.0f + 0.5f;  //V相のduty値

    R_TFU->SCDT1 = angle - RAD_240_DEGREE;              //U相から240°ずれた点をW相とする

    phase_duty->w = R_TFU->SCDT0 * duty / 2.0f + 0.5f;  //W 相の duty 値
```

TFU の使用は非常に単純で、レジスタに値（rad 値）を書けば、レジスタに結果（cos 値）が格納されますので、読むだけです。

計算結果の回収方法は 2 通りあります。

上記手法ですと、SCDT0 をリードアクセスした段階で、計算が終わるまでの間バスアクセスが止まるので、計算中に割り込みで他の処理を行う場合等待ちが発生します。「計算が終わったであろうタイミングでリードする」「計算中フラグを観測し、フラグが落ちたタイミングでリードする」という手法もあります。（本チュートリアルのプログラムでは、14 サイクル程度のバスロックは問題ないと考えて素直に待つ手法としています。）

※本プログラムではレジスタ書込みとしていますが、cosf()関数で計算しても、TFU が使われますので高速で演算されます（e2studio+GCC ARM C Embedded 環境の場合）

ー浮動小数点数の計算に関してー

float a;

a = a * 2.0;

上記の様なコードを書くと、コンパイル時

warning: conversion from 'double' to 'float' may change value [-Wfloat-conversion]

というワーニングが出ます。double 型の数値を float 型の変数に代入したので値が変わる（float で表現できない桁数の場合や double の精度が失われる）という内容で、気にしなければ、気にならない内容かもしれません。

ワーニングを消す目的で、

a = (float)(a * 2.0);　　//基本的には違う

としたくなりますが、

a = a * 2.0f;

とするのが正解かと思います。

PC でのプログラムに慣れている方（組み込み系はあまり触らない方）なら、

2　（整数）
2.0　（浮動小数点数）

の使い分けは行うが、あえて

2.0f　（float 型の浮動小数点数）

を使うケースがあまりないかもしれません。

また、RX マイコンのプログラムに慣れている方では、RX では「RX231/RX651 等のマイコン」では、コンパイラ(CC-RX)オプションで、

double 型、及び long double 型の精度　単精度として扱う(-dbl_size=4)

がデフォルトになっています。

そのため、double と float どちらでも、4 バイト精度(float)で計算されます（float で扱うようなコードが生成されます）。（a = a * 2.0 は float の演算として処理されます）

また、「RX72N」等のマイコンでは、倍精度浮動小数点命令を使用する(-dpfpu)がデフォルトで、double 型をハードウェアで計算する事ができるので、a= a * 2.0 の計算も高速に処理できます。

そのため RX では、2.0 と 2.0f を厳密に区別しなくても、それ程問題にならないケースが多いと思われます。

それに対し、RA マイコンは

float　マイコンに FPU が搭載されているので、ハードウェアで高速に演算
double　ハードウェアでは一発で計算できないので、ソフトウェアライブラリでの演算（極端に演算速度が落ちる）

となります。これは、double 型の変数を使った場合、計算が遅いというだけの話ではなく、float 型の変数 a を使った計算においても、

```
a = a * 2.0;
```

2.0 が double 型の定数として取り扱われるので、乗算の部分が double 型の演算となり、計算が遅くなります。

よって、上記の計算は

```
a = a * 2.0f;
```

である必要があります。

モータ制御の様なリアルタイム制御では、演算速度は重要な要素となりますので、double 型の精度が必要のない計算は、2.0f の様に f サフィックスを付ける様にしてください。

以上で、チュートリアル編は終了となります。

チュートリアルの内容を踏まえたサンプルプログラムの解説は、別資料「ソフトウェア　サンプルプログラム編」を参照してください。

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	2022.6.23	—	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

ルネサス エレクトロニクス RA6T2(QFP-100 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(RA6T2)
[ソフトウェア　チュートリアル編]
取扱説明書

株式会社 北斗電子
 2022 年 6 月 23 日改訂 REV.1.0.0.0 (220623)