ブラシレスモータスタータキット(RL78G1F)
[ソフトウェア編]
取扱説明書

ルネサス エレクトロニクス社 RL78/G1F(QFP-64 ピン)搭載
ブラシレスモータスタータキット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.0.0.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

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

【限定保証】

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

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

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

【免責事項】

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

安全上のご注意

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

表記の意味

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

絵記号の意味

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

⚠警告

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

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

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

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

⚠注意

以下のことをされると故障の原因となる場合があります。

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

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

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

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

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

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

アクセスランプがある製品では、アクセスランプが点灯中に電源を切ったり、パソコンをリセットをしないでください。

製品の故障の原因となったり、データが消失する恐れがあります。

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

1.1. はじめに

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「RL78/G1F 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 CS+の環境向けに作成されていますので、CS+の環境を PC にインストールしておいてください。なお、CS+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。

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

1.2. マイコンボード初期設定

参照プロジェクト：TUTORIAL1

本キットに付属のマイコンボード(HSBRL78G1F-64)は、RL78/G1F グループのマイコンを搭載しており、ボードに搭載されている水晶振動子使用時はクロック周波数 20MHz、マイコン内蔵の高速オンチップ・オシレータを使用したときは、クロック周波数 32MHz で動作させる事ができます。ここでは、ボード上に搭載されている水晶振動子を使用して、クロック周波数 20MHz で動作させる場合の設定を行ってみます。

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）

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR1_RL78G1F64¥RL78_G1F_64.mtpj

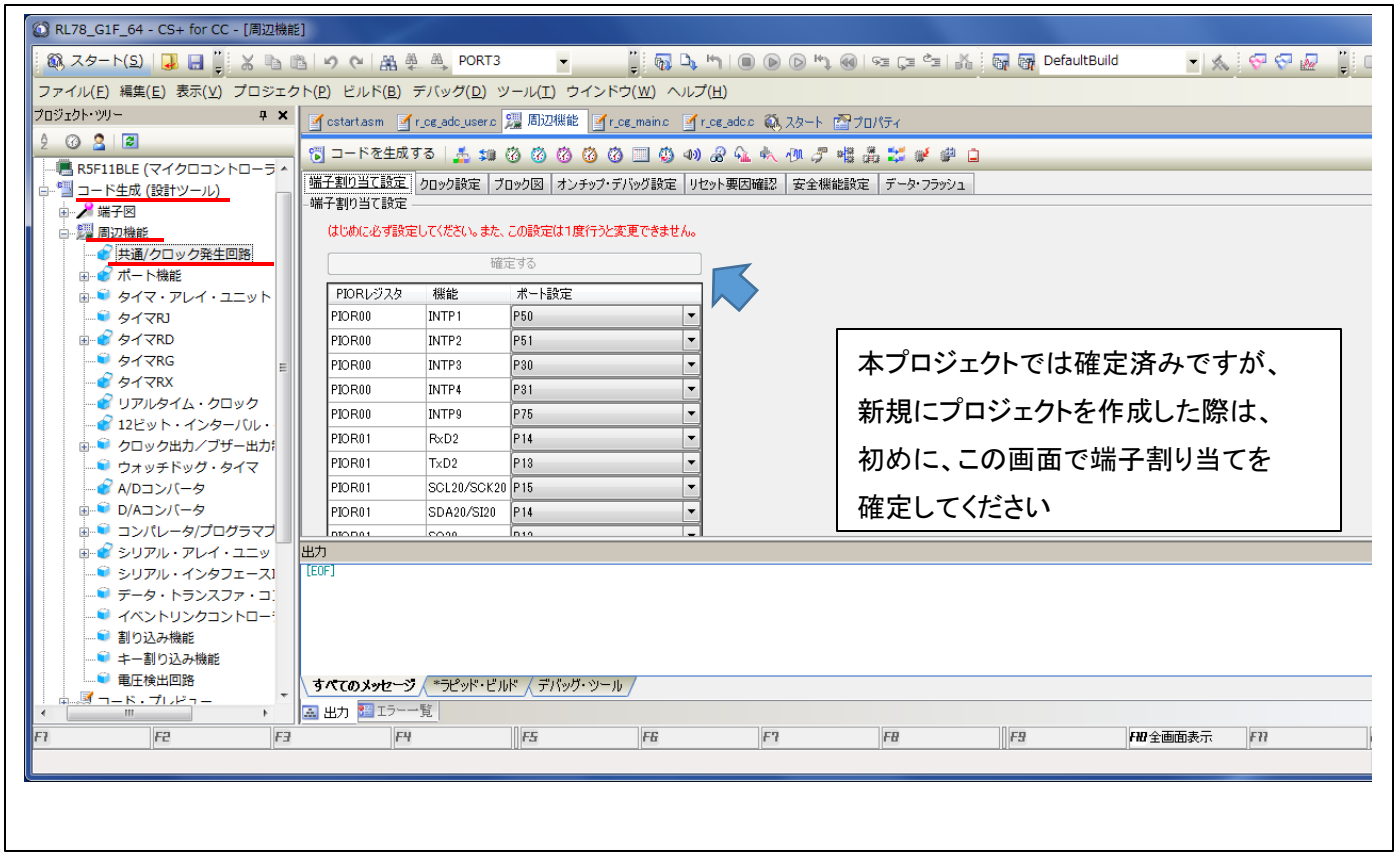
を、ダブルクリック（または、CS+から「プロジェクト」ー「プロジェクトを開く」で）して、プロジェクトを開きます。

・USB-OCE を使用している場合
マイコンボードと USB-OCE を接続してください。
モータドライバボードに電源を投入してください。

・E1 を使用している場合
マイコンボードと E1 を接続してください。
モータドライバボードに電源を投入してください。

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。CS+では、コード生成機能があり、面倒なレジスタ設定のプログラムコードをメニューで選択して生成する事ができます。

プロジェクトを開き、メニューを展開し



コード生成

周辺機能

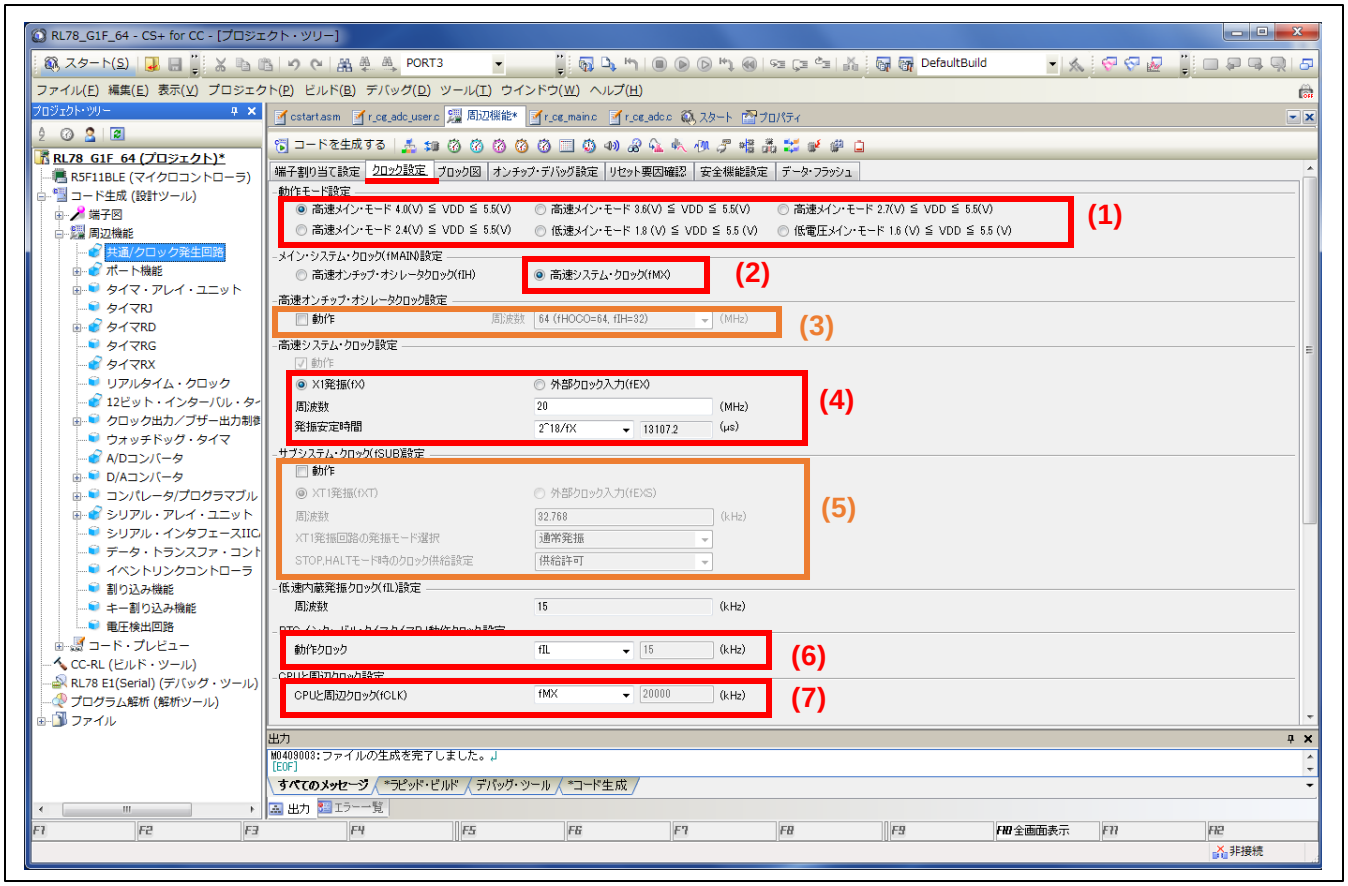
共通/クロック発生回路

をダブルクリックしてください。（メニューが展開していないときは、＋を押して展開してください）

※新規にプロジェクトを作成する際は、「端子割り当て設定」のタブを開きポート設定を確定させてください

本プロジェクトでは確定済みになっていますが、デフォルト値から変更は行っておりません。

次に、「クロック設定」のタブを開いて設定します。



選択項目は

(1)動作モード設定　「高速メインモード(4V≦VDD≦5.5V)」にチェック

　モータドライバボードと組み合わせて使用する場合は、VDD=5V となりますので、この項目を選択します。

(2)メイン・システム・クロック　「高速システム・クロック(MX)」を選択

　ボード上に搭載されている 20MHz の水晶振動子を使用する設定です。

(3)高速オンチップ・オシレータクロック設定　動作のチェックを外す

　本プロジェクトでは、使用しないのでチェックを外します。

(4)高速システム・クロック設定　「X1 発振(fX)」を選択

　周波数　「20」を入力

　発振安定時間　「2^{18}」を選択

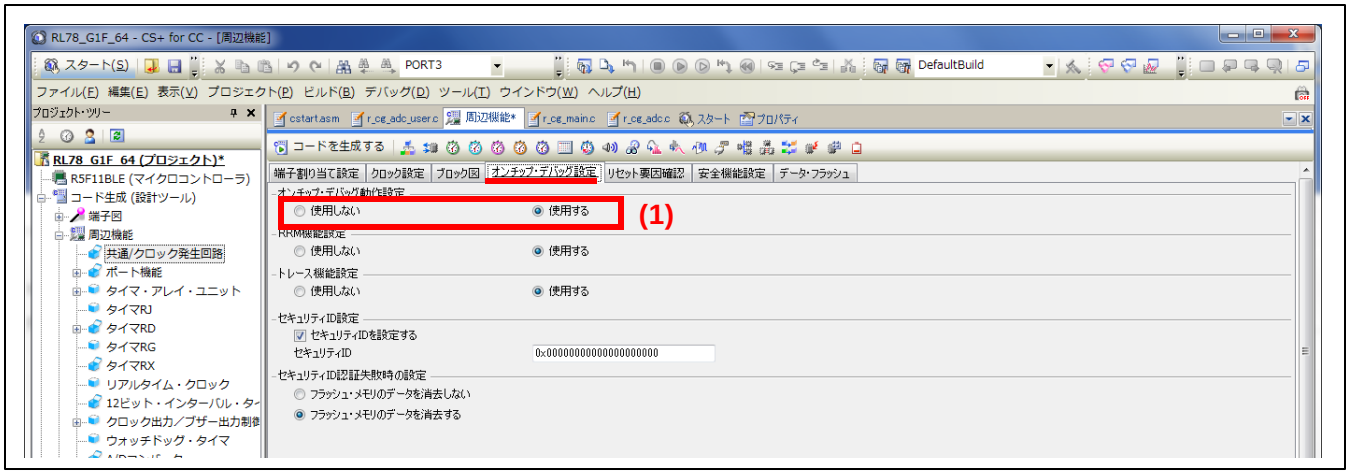
(5)サブシステム・クロック設定　動作のチェックを外す

　本ボードには、サブクロック用の 32.768kHz の水晶振動子が搭載されていますので動作させる事もできます。リアルタイムクロック(カレンダ機能等)を使用する場合は、チェックを入れて、動作させてください。

(6)RTC 動作クロック　「fiL」を選択(fiL 以外は選択できないようになっています)

(7)CPU と周辺クロック　「fMX」を選択(fMX 以外は選択できないようになっています)

次に、「オンチップ・デバッグ設定」のタブを開いて設定します。

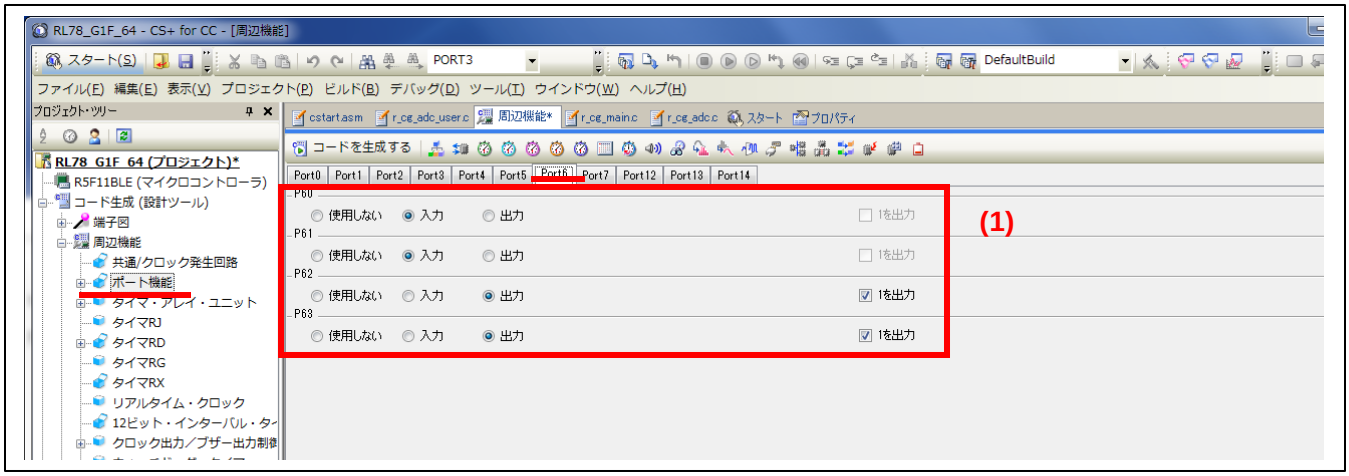


選択項目は

(1)オンチップ・デバッグ動作設定　「使用する」にチェック

デバッガ接続を行わない場合は、本設定は不要です。

次に、プログラムの動作に関連する設定を行います。



コード生成

周辺機能

ポート機能

「Port6」タブ

を開いて、

設定項目

(1)P60　「入力」

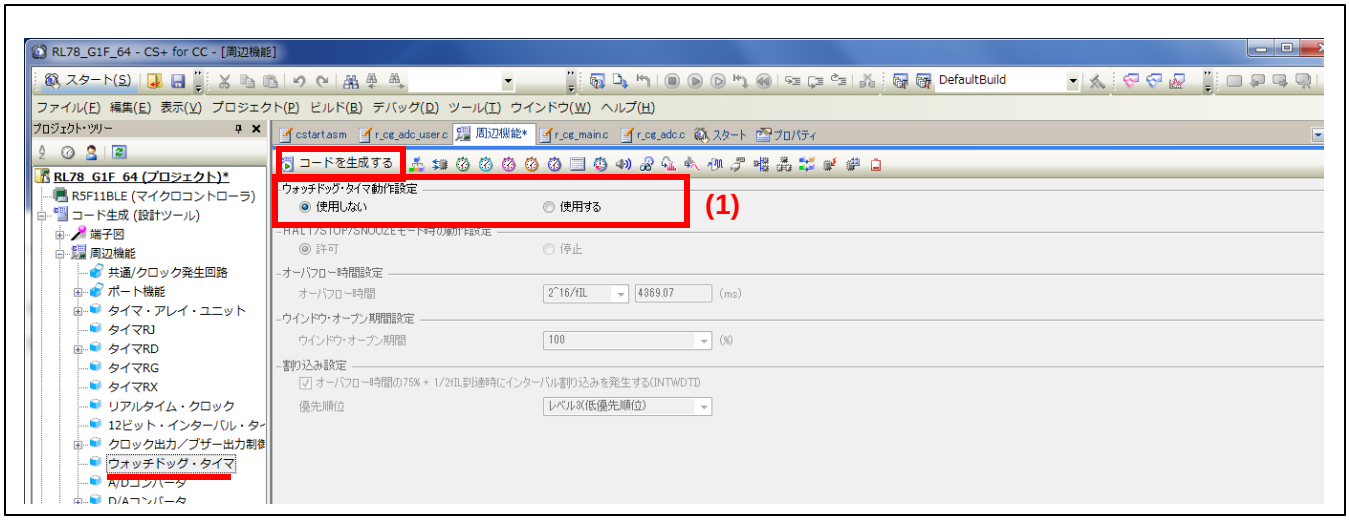
　P61　「入力」

　P62　「出力」　「1 を出力」

　P63　「出力」　「1 を出力」

を選択、及びチェックを入れてください。

次に、ウオッチドッグ・タイマに関連する設定を行います。



設定項目

(1)ウォッチドッグ・タイマ動作設定　「使用しない」にチェック

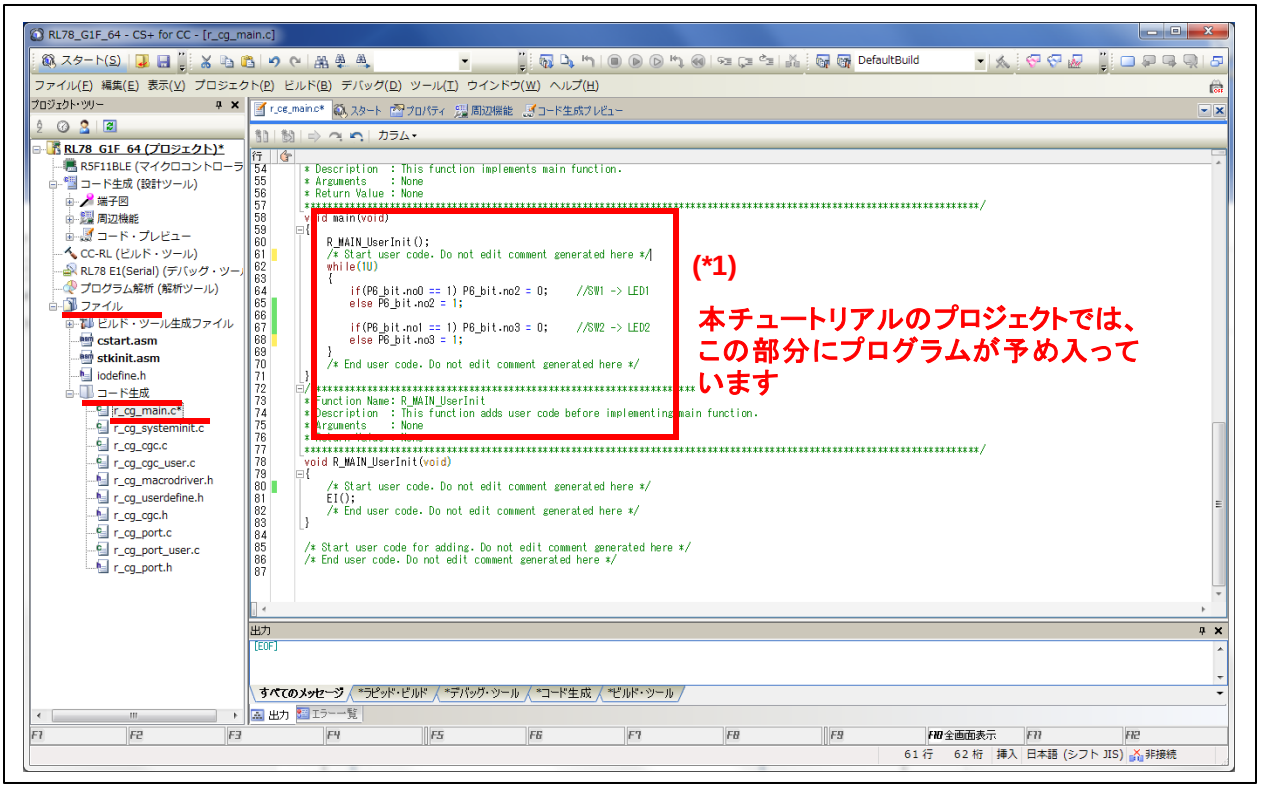
　一通り設定が終わったら

コードを生成する

のボタンを押してください。(設定に変更を加えた際は、必ず最後にこのボタンを押してください。)

「コードを生成する」ボタンにより、CS+がクロック設定等のプログラムコードを自動で生成します。(コード生成プラグインをインストール及び有効にする必要があります。詳細は CS+のマニュアルを参照ください。)

基本的なプログラムコード（クロックの設定等）は、CS+により生成されましたので、ユーザプログラムの部分を追加します。



ファイル

コード生成

r_cg_main.c

をダブルクリックします。

CS+のコード生成機能により、自動生成されたソースコードにユーザ側でソースを追加する場合は、(*1)の部分

```
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
```

Start user code. から End user code. の中に入れる必要があります。（この領域から外れた場所に追加すると、コード生成ボタンを押したときのコード再生成で、ユーザ側で追加したものが消えてしまいます。

上記、while の部分に、本チュートリアルの目的である、スイッチの読み取り及び LED の点灯制御のソースをいれてみます。（TUTORIAL1 のプロジェクトには予め、下記ソースが入っています。）

```
/* Start user code. Do not edit comment generated here */
while(1U)
{
    if(P6_bit.no0 == 1) P6_bit.no2 = 0;//SW1 -> LED1
    else P6_bit.no2 = 1;

    if(P6_bit.no1 == 1)P6_bit.no3 = 0;//SW2 -> LED2
    else P6_bit.no3 = 1;
}
/* End user code. Do not edit comment generated here */
```

P60(SW1)が H(ON)なら
　P62 を L(LED1:ON)
上記以外
　P62 を H(LED1:OFF)
SW2, LED2 も
同様の処理を行う

while ループ（本プログラムでは永遠にこの中をループ）

-ビルド
-ビルド・プロジェクト （もしくは F7 キーを押す）

エラーが出なければ、プログラムのビルドは成功です。

1.3. プログラムの実行

実際に作成したプログラムをマイコンボードの ROM 領域に書き込みを行い、プログラムを実行するためには以下のいずれかの方法があります。

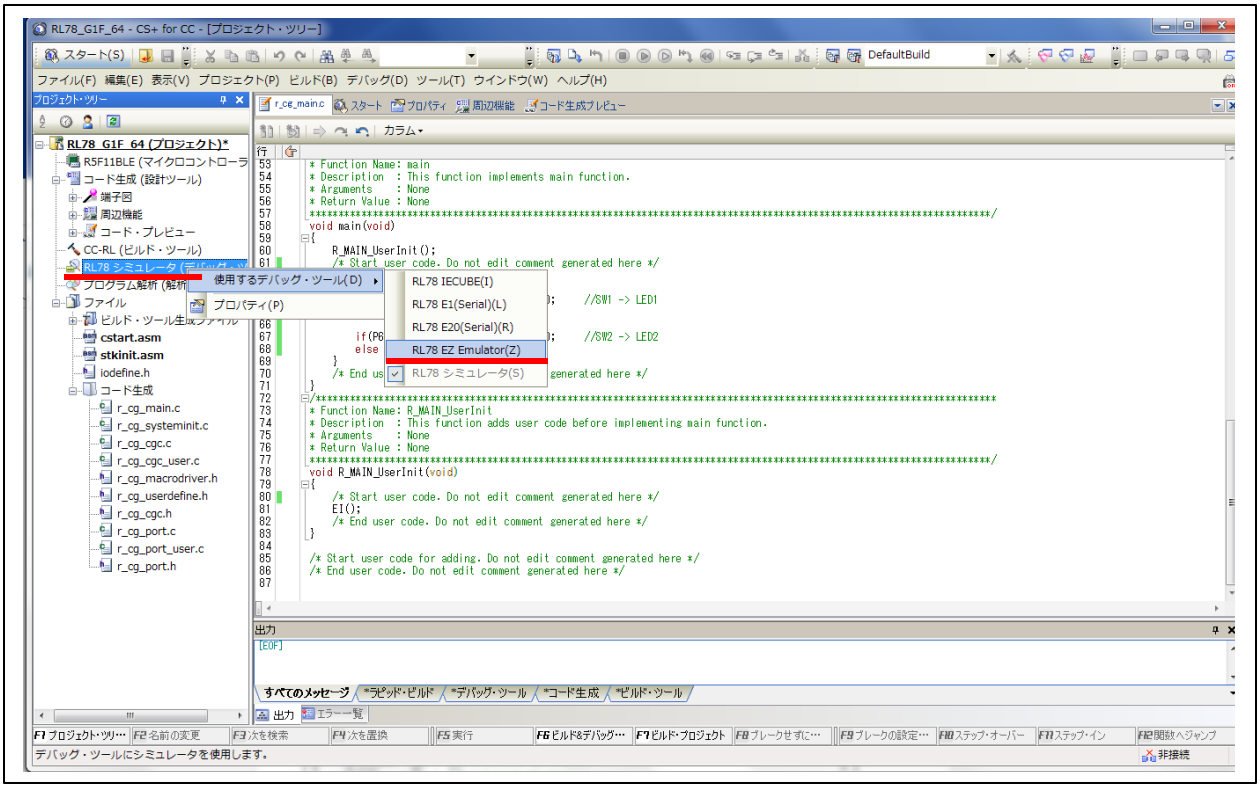
1.3.1. USB-OCE を使う方法

当社製品、USB-OCE（製品型名：USBOCE-RL78-3-14）を使用して、プログラムをマイコンボードに書き込む方法を説明します。

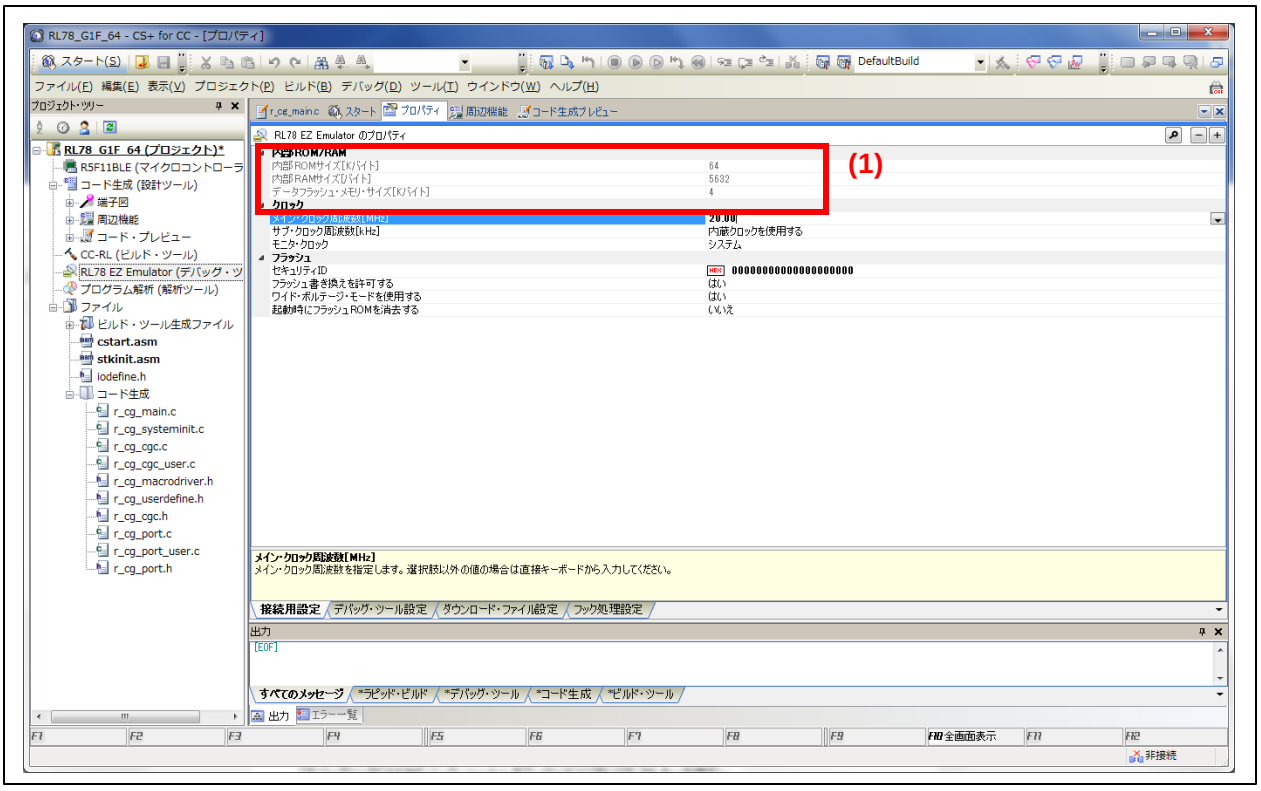
・USB-OCE 付属の 14P フラットケーブルで、USB-OCE とマイコンボード(HSBRL78G1F-64)を接続します。
・USB-OCE を USB-miniB ケーブル(*1)で PC と接続します

(*1)USB-mimiB ケーブルは、USB-OCE には付属致しませんので、市販のものをご用意ください
※USB-OCE のドライバソフトがインストールされている事が必要です

CS+で、



デバッグツール （右クリック）
使用するデバッグツール
RL78 EZ Emulator
を選択してください。



デバッグツール　（右クリック）

プロパティ

設定項目

(1)メイン・クロック周波数　「20.00」を選択してください

次に

-デバッグ

-デバッグ・ツールへダウンロード

の操作を行うと、プログラムがマイコンボードに書き込まれます。

-デバッグ

-実行　（もしくは F5）

の操作で、プログラムが実行されます。

SW1（上側）を ON/OFF させてみてください。SW1 に連動して、LED(P62)が、点灯・消灯した場合プログラムは、マイコンボードにダウンロードされ、実行されていることを示します。（なお、本プログラムでは SW2 と LED(P63)も連動しています）

プログラムを停止する場合は、

-デバッグ

-停止（もしくは Shift＋F5）

させて、

-デバッグ
-デバッグツールから切断（もしくは Shift＋F6）
とすると、USB-OCE とマイコンボードは切り離されます。（ケーブルを抜いたり、電源を落としたりできる状態となります）

1.3.2. E1 を使う方法（デバッガ接続）

ルネサスエレクトロニクス製 E1（もしくは E20）を使用する場合は、上記 USB-OCE を使う場合のデバッグの

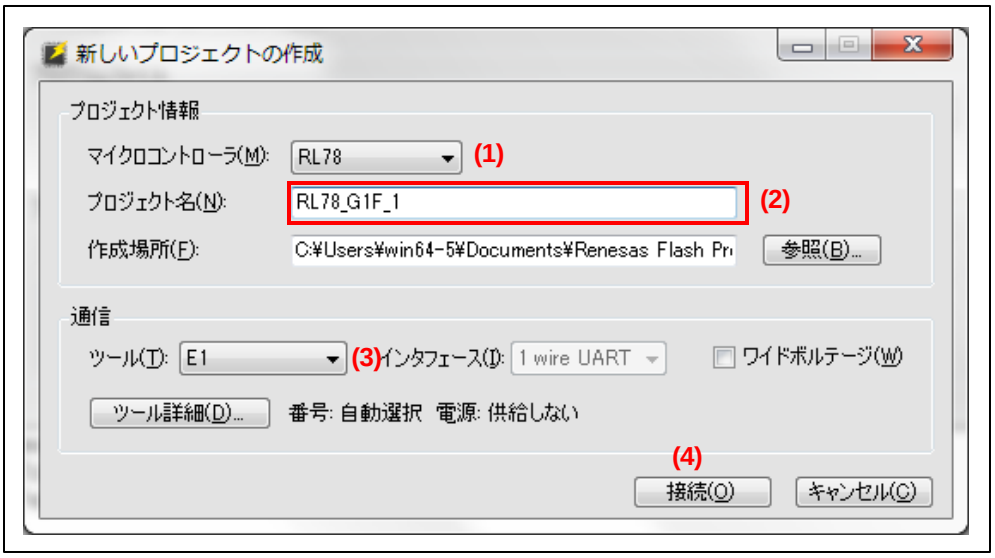
デバッグツール　（右クリック）
使用するデバッグツール
RL78 E1(Serial)

を選択してください。デバッグツールの選択が異なるだけで、他は USB-OCE を使用した場合と同様です。

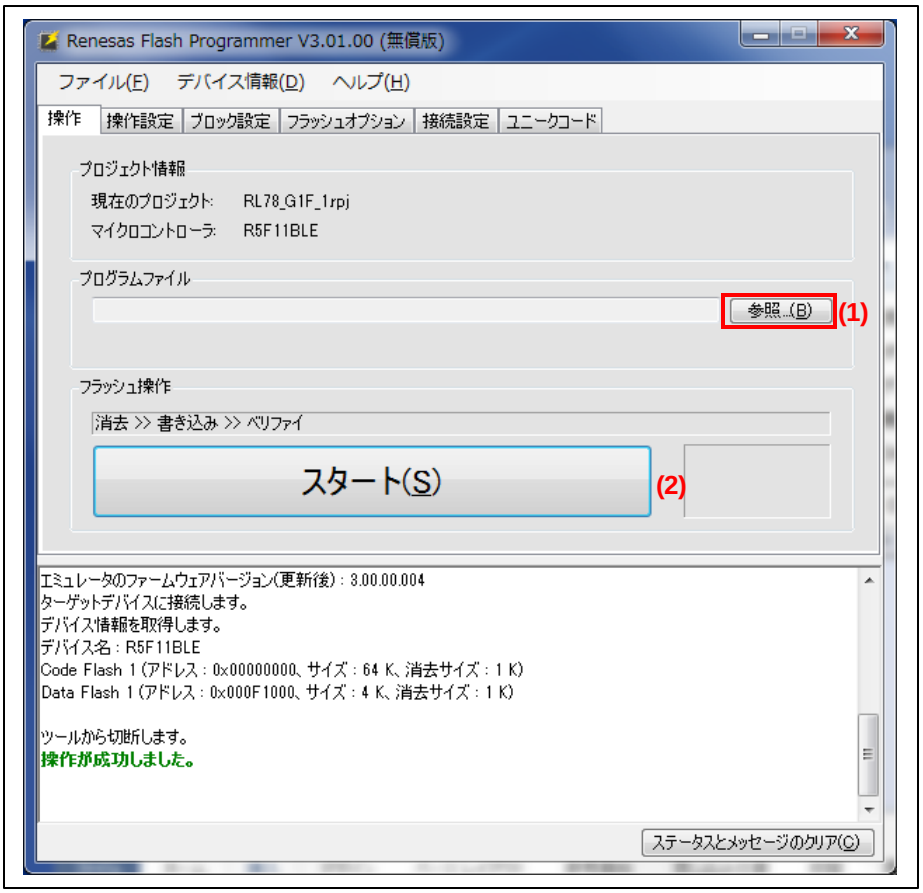
1.3.3. E1 を使用して書き込みのみを行う方法

※デバッガと接続しない状態でプログラムを動かす方法です

RenesasFlashProgrammer(RFP)を起動し、
「ファイル」ー「新しいプロジェクトを作成」



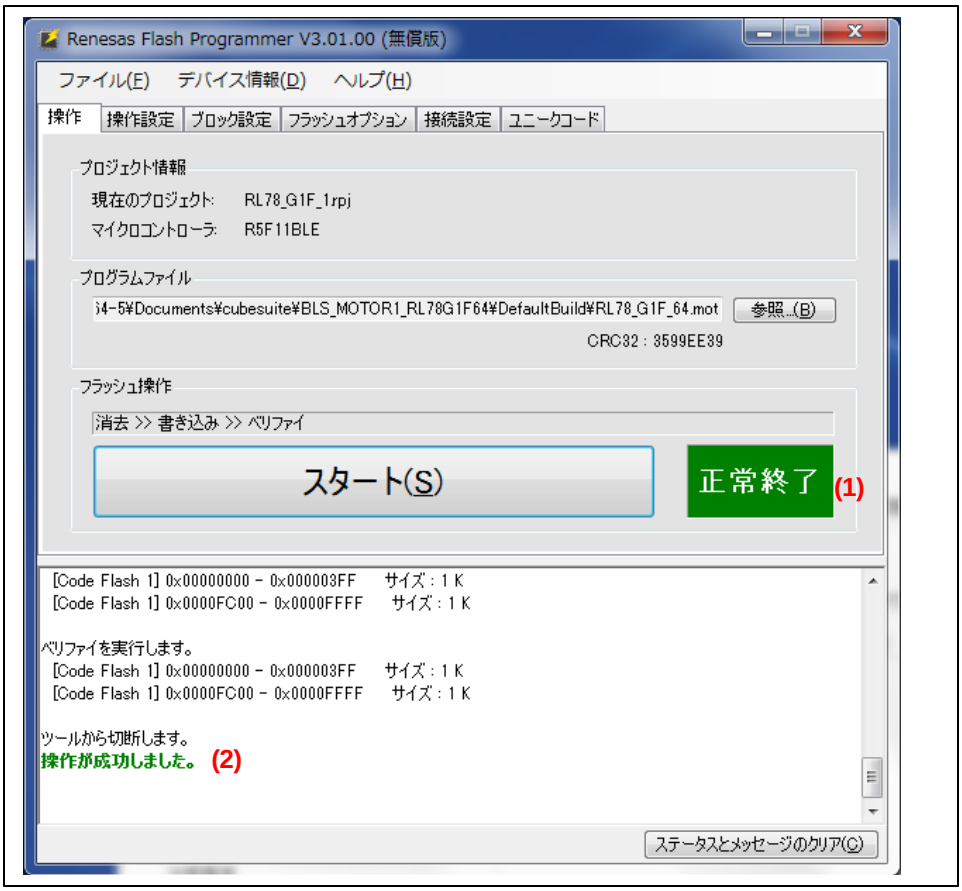
(1)RL78　を選択
(2)適当な名称を入力
(3)E1　を選択
(4)接続ボタンを押す



(1)TUTORIAL1 で生成した mot ファイルを指定

（BLS_MOTOR1_RL78G1F64¥DefaultBuild の下に、RL78_G1F_64.mot というファイル名でファイルがあるはずです）

(2)スタートボタンを押す



(1)正常終了
(2)操作が成功しました

の表示となれば良い
※フラッシュ操作は、デフォルト
（消去>>書き込み>>ベリファイ）の
ままで構いません

・E1 を取り外す

プログラムの動作としては、デバッガ接続（1.3.1 及び 1.3.2）と同じです。

いかがでしょう、スイッチと LED が連動するという簡単なプログラムですが、動作したでしょうか。

動作しない場合は、
・電源が入っているか（モータドライバボード上の PL(D5)が点灯している、マイコンボード上の PL が点灯しているか）
・デバッガ（USB-OCE または E1）との接続エラーとなっていないか
・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。

今回は、マイコンのクロックの設定は、CS+のコード生成機能が行いましたので、ハードウェアマニュアルを見てレジスタの値を設定する必要がありませんでした。

細かな設定をするのであれば、コード生成ではカバーが出来ないケースもありますが、メニューで選ぶだけでレジスタの設定が完了するコード生成を活用してみてください。

1.4. モータに電流を流す

参照プロジェクト：TUTORIAL2

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR2_RL78G1F64¥ RL78_G1F_64.mtpj

を開いてください。プログラムのビルドと、マイコンへの書き込み・実行は、1.3 と同様に行ってください。

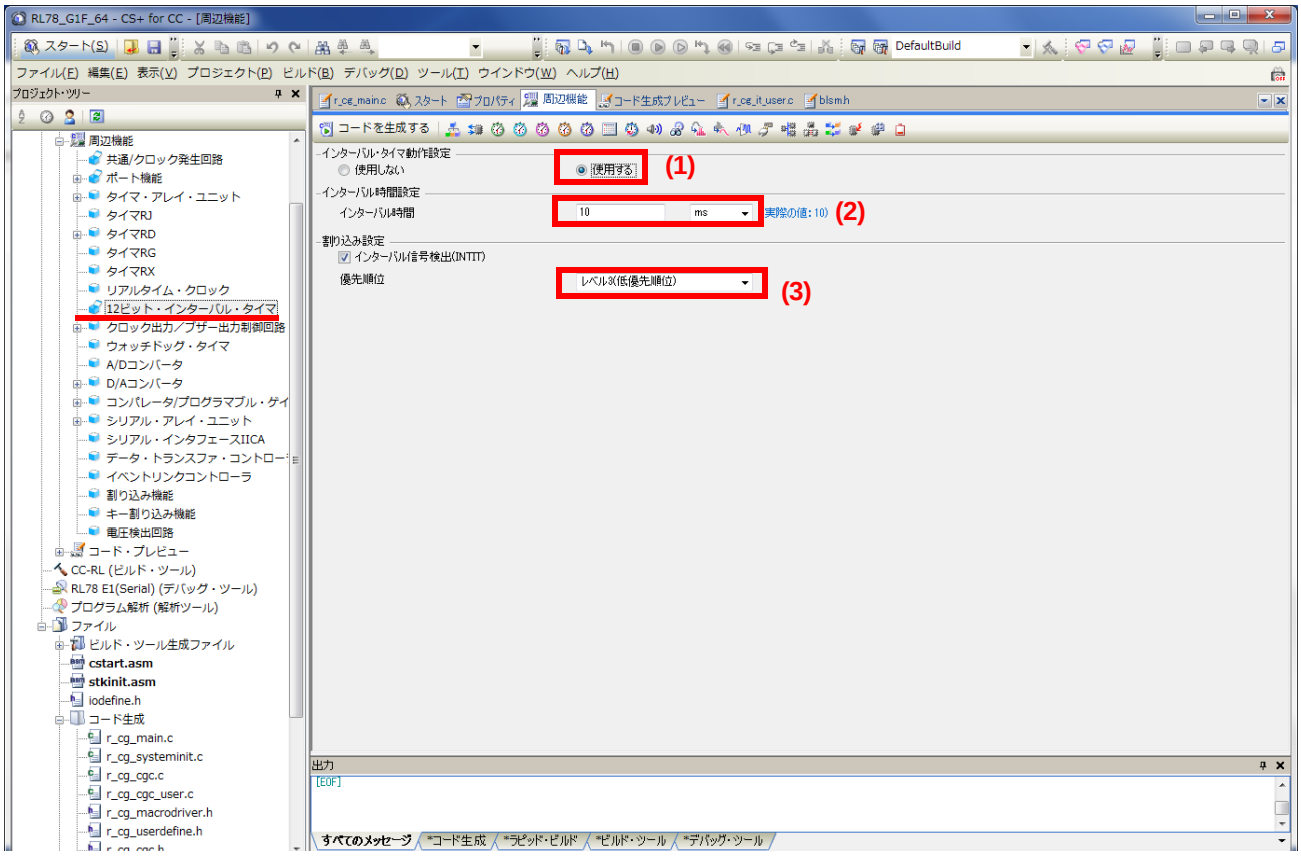
プログラムを実行し、SW1 を ON にすると、LED1 が点灯。LED2 が 1 秒間隔で一瞬点灯すると思います（ほんの短い間ですので、判りづらいですが）。LED2 が点灯するタイミングで、モータから「カチッ」という音が聞こえてくると思います。（よく見ると、モータの軸がかすかに動いているのが見えると思います）（SW2 は本プログラムでは未使用です）

本プログラムでは、

・タイマ割り込み

の機能を使っています。

〇タイマ割り込み（12 ビット・インターバル・タイマ）



コード生成

周辺機能

12 ビット・インターバル・タイマ

(1)インターバルタイマ　使用する

(3)インターバル時間　10　ms　（10ms 毎に特定の処理を行いたい）

(4)割り込みを許可　レベル 3

上記設定後、「コード生成」すると、

BLS_MOTOR2_RL78G1F64¥cg_src¥r_cg_it_user.c が生成されます。

```
static void __near r_it_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

    /* End user code. Do not edit comment generated here */
}
```

コード生成ではプログラムの枠組みのみ生成されますので、実体（中身）を上記、Start - End の中に記述します。

この仕組みは、TUTORIAL1 の r_cg_main.c でも同じで、コード生成機能を使用した場合で共通です。

BLS_MOTOR2_RL78G1F64¥cg_src¥r_cg_it_user.c

```
static void r_cmt_cmi0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    static unsigned long timer_counter=0;
    static unsigned char state=0;

    const unsigned long width=2000;

    unsigned char p1;

    volatile unsigned long x;

    timer_counter++;

    if((timer_counter % 100) == 0)
    {
        if(active == 1)
        {
            p1 = P1 & 0xc0;//b7, b6 keep
            switch(state)
            {
                case 0:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = W_V_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                break;
                case 1:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = W_U_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                    break;
                case 2:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = V_U_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                break;
                case 3:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = V_W_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                break;
                case 4:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = U_W_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                break;
                case 5:
                    P6_bit.no3 = 0;//LED2=ON
                    P1 = U_V_DIRECTION | p1;
                    for(x=0; x<width; x++);
                    P1 = OFF_DIRECTION | p1;
                    P6_bit.no3 = 1;//LED2=OFF
                break;
                default:
                break;
                }
            }
        state++;
        if(state == 6) state=0;
    }
```

10ms 割り込み処理の 100 回に 1 回
（=10ms × 100=1s:1 秒に一回）

active は SW1=ON のとき 1 に main()でセットされる

state は実行の度にインクリメントされて 5 までいくと 0 に戻る

モータに電流を流す（W 相から V 相に対して）
width　で定義した時間空ループ（ウェイト）
電流を切る

W_U_DIRECTION_1
は、定数定義で後述

モータに電流を流す定義(BLS_MOTOR2_RL78G1F64¥blsm¥blsm.h)

```
//電流を U→V に流す設定, P15 : U(P) -> P11 : V(N) P15=H, P11=H
#define U_V_DIRECTION 0x22

//電流を U→W に流す設定, P15 : U(P) -> P10 : W(N) P15=H, P10=H
#define U_W_DIRECTION 0x21

//電流を V→U に流す設定, P13 : V(P) -> P14 : U(N) P13=H, P14=H
#define V_U_DIRECTION 0x18

//電流を V→W に流す設定, P13 : V(P) -> P10 : W(N) P13=H, P10=H
#define V_W_DIRECTION 0x09

//電流を W→U に流す設定, P12 : W(P) -> P14 : U(N) P12=H, P14=H
#define W_U_DIRECTION 0x14

//電流を W→V に流す設定, P12 : W(P) -> P11 : V(N) P12=H, P11=H
#define W_V_DIRECTION 0x06

//電流を流さない設定
#define OFF_DIRECTION ~0x3f
```

ここで、モータに電流を U→V に流すとき（付属ケーブルでは、青→黄の方向の電流）

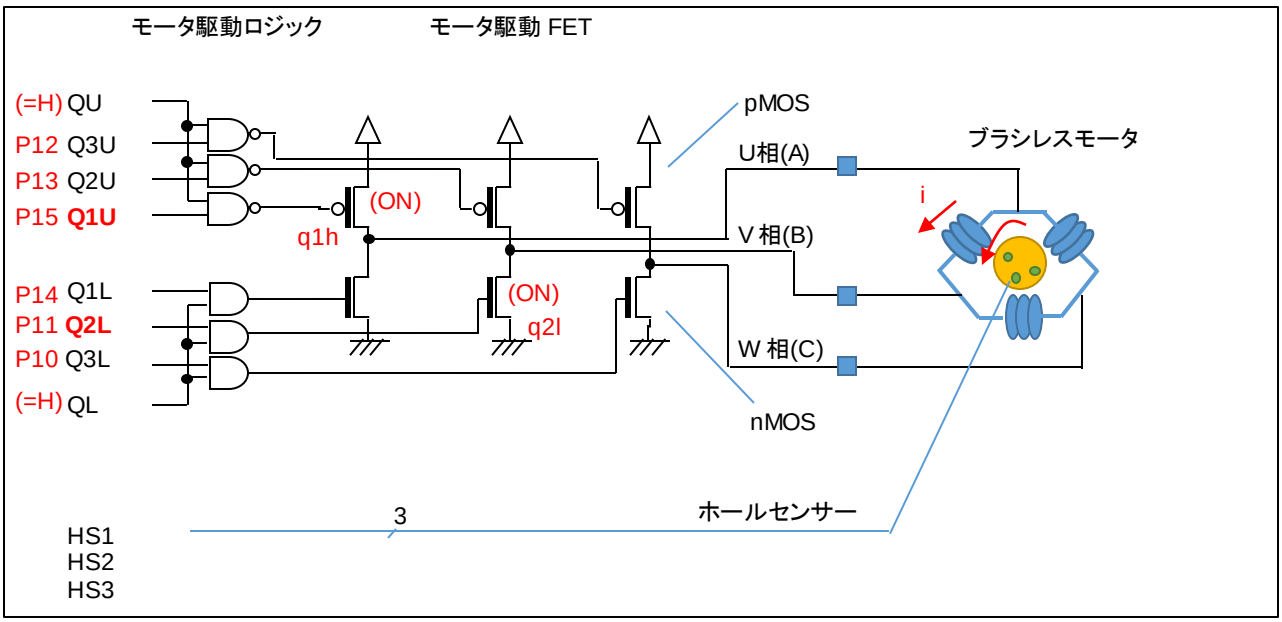
U_V_DIRECTION 0x22

という定義を使用しています。

これは、

P1	b7	b6	b5	b4	b3	b2	b1	b0
0x22	0	0	1	0	0	0	1	0

P1 の b5(P15)と b1(P11)を ON させ、他は OFF するという意味となります。



P15 は、U の H 側を制御しており（Q1U につながっている）、P11 は V の L 側を制御しています（Q2L につながっている）。P15 と P11 を 1 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 以外の定義もに、3 相ある電極の 2 相間に電流を流す設定となります。

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。

1.5. A/D 変換と PWM を試す

参照プロジェクト：TUTORIAL3

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR3_RL78G1F64¥RL78_G1F_64.mtpj

を開いてください。

次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。

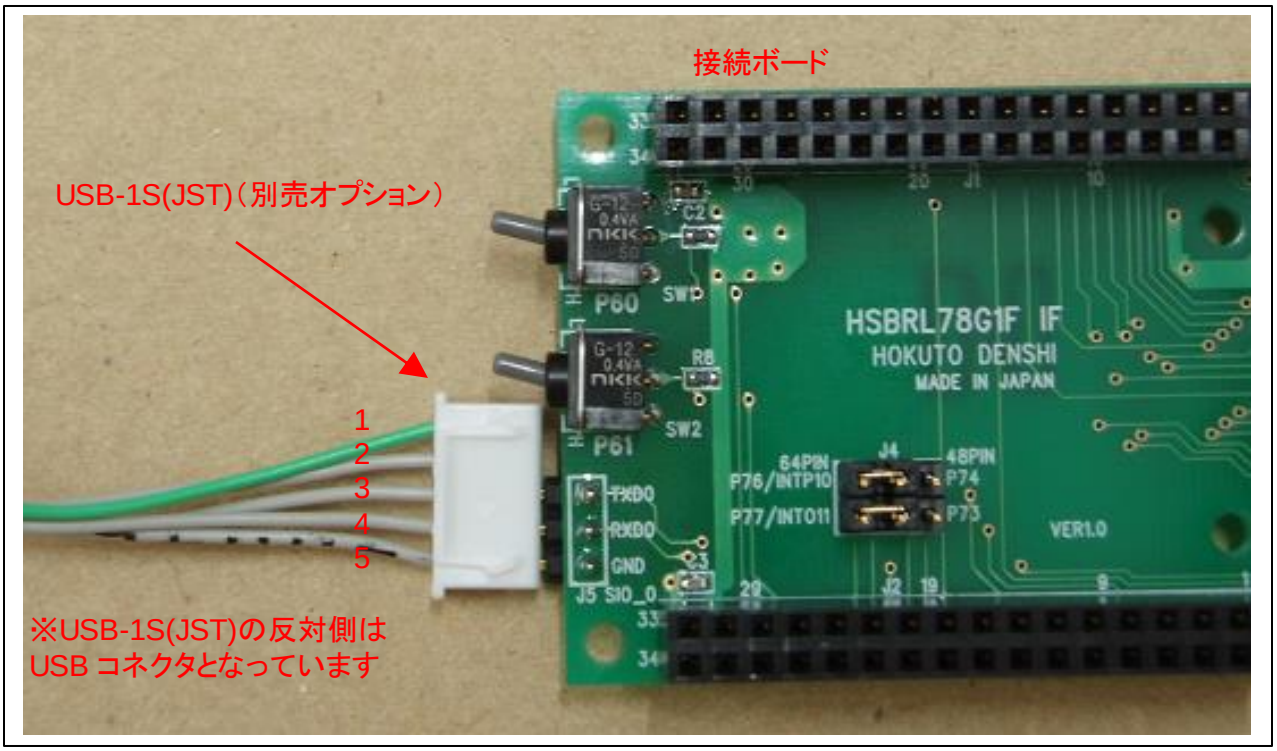
本プログラムでは、

・VR の回転角に応じたパルス幅の信号が、P17/QL から出力される。

・モータドライバボード上の温度センサ(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 UART0(TXD0:P51, RXD0:P50)に、シリアル通信で出力します。USB-1S（別売オプション）, USB-Adapter（別売オプション）または市販の USB-Serial 変換機器を接続ボード J5 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）。

－接続例(USB-1S)－

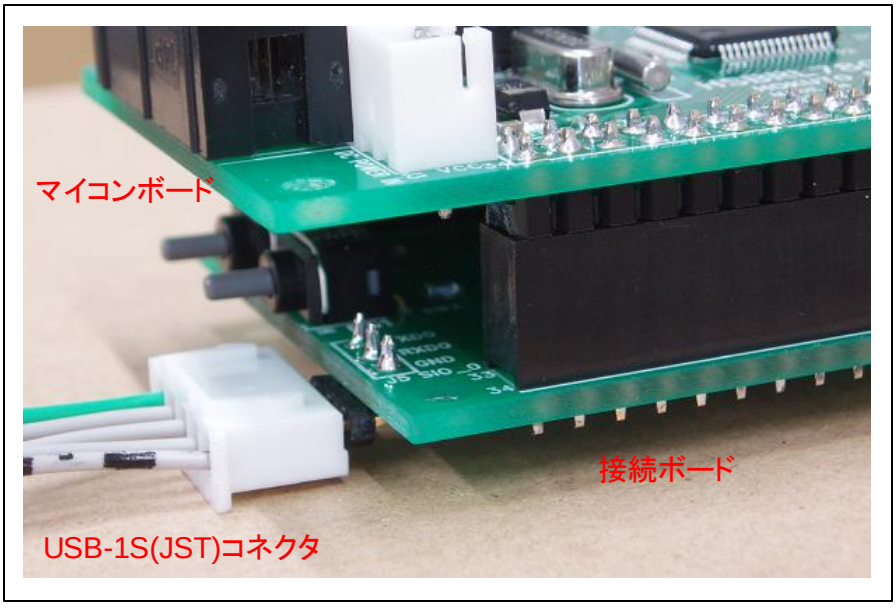


※見やすくするため写真ではマイコンボードを取り外していますが
実際は、写真の接続ボードの上にマイコンボードが接続されます

UART 端子	-	USB-1S(JST)
1(TX)	-	3(TXD) (*1)
2(RX)	-	4(RXD) (*1)
3(GND)	-	5(GND)

※USB-1S(JST)の 1,2 番ピンは未接続となります、3~5 番ピンが接続ボード端子と接続されるようにしてください

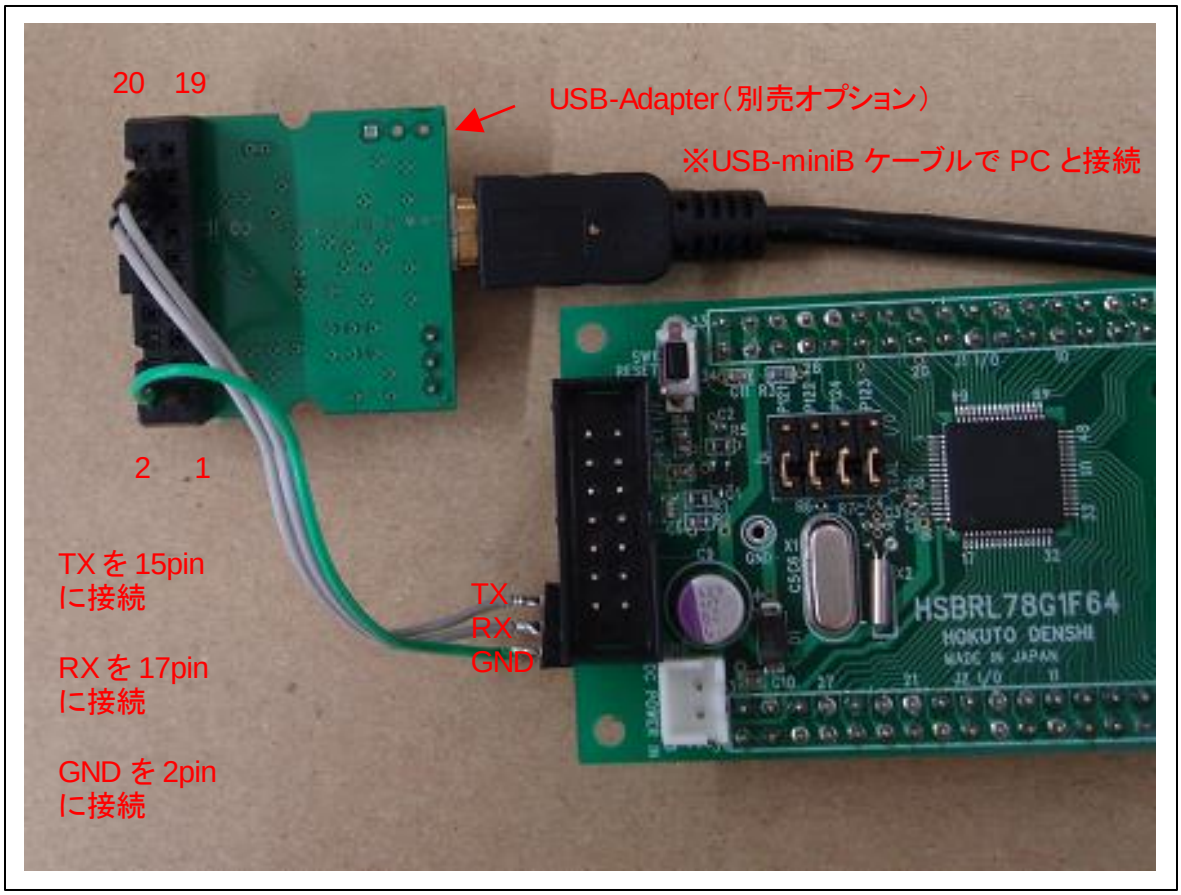
(*1)USB-1S(JST)の信号名は接続先基準で命名されているためこの場合は、TX-TX, RX-RX の接続となりますが、市販の USB-Serial 変換機器を使用する際は、TX-RX, RX-TX となるように接続してください。



実際の接続例

本来は接続ボードの上にマイコンボードが接続された状態で使用します。

ー接続例(USB-Adapter)ー



変換ボードの UART 端子を、USB-Adapter（北斗電子製オプション製品）に接続する場合の、接続例を示します。
※ケーブルは自作する必要があります。

UART 端子	-	USB-Adapter
1(TX)	-	15(TXD) (*1)
2(RX)	-	17(RXD) (*1)
3(GND)	-	2(GND)

(*1)USB-Adapter の信号名は接続先基準で命名されているためこの場合は、TX-TX, RX-RX の接続となりますが、市販の USB-Serial 変換機器を使用する際は、TX-RX, RX-TX となるように接続してください。

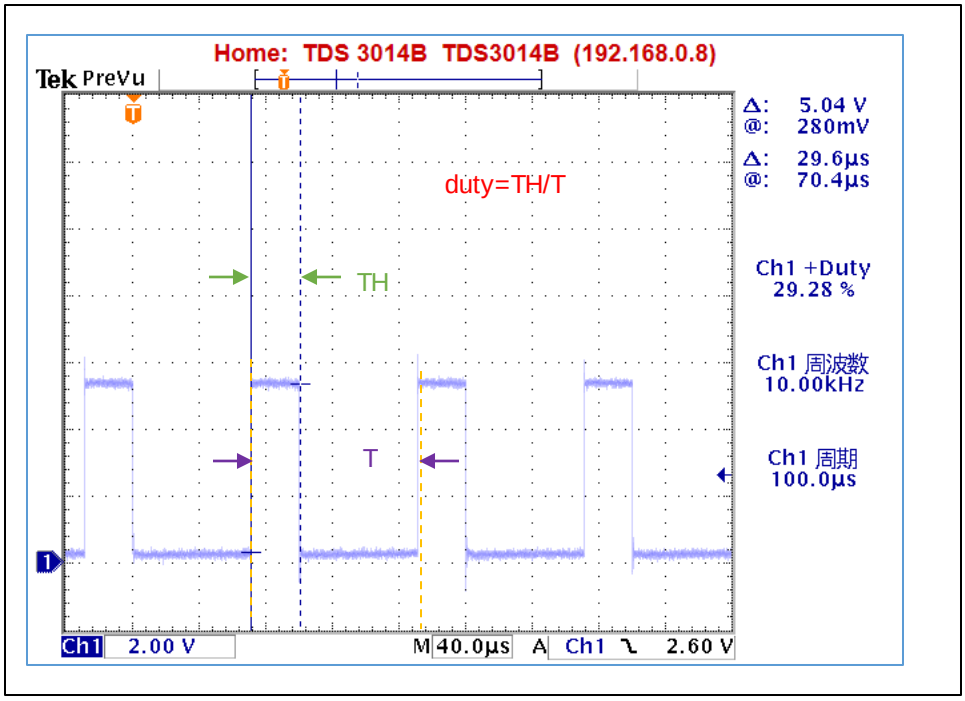
・シリアル端末から出力される情報（5 秒に 1 回更新）

Tempatature(A/D value)	: 551
Tempatature(degree)	: 29
VR(A/D value)	: 300
P17 duty	: 586 / 2000

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

38400bps, 8bit, none, 1bit の設定で表示できます

・オシロスコープで QL 端子を観測した波形



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、マイコンの P27/ANI7 に接続されており、ここでは当該端子を ANI7 の機能（A/D 変換入力）として使っています。RL78/G1F の A/D 変換機能は、10bit となっているので、0~1023 までの値を取ります。（この値は、約 25℃のときに、512 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、ブラシレスモータスタータキット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、29℃となっていますが、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、930(1023×10/11)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。

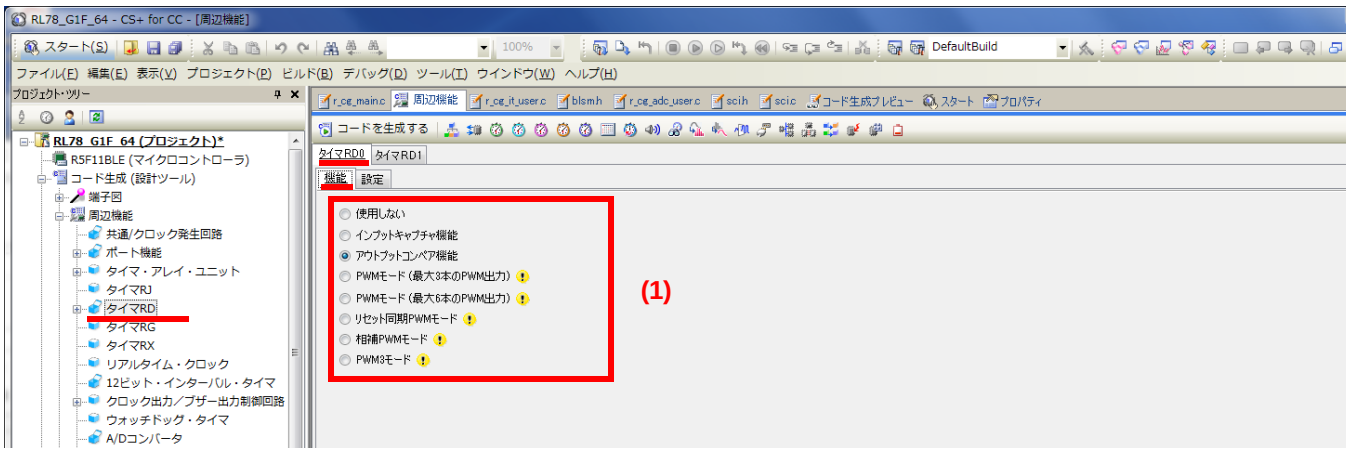
P17 duty は、P17/QL 端子の duty 値となります。この値は、VR の読み取り値に連動して変更を行っています。（VR の読み取り値/1023×100）0~90%程度まで設定可能です（本プログラムでは）。この値と、実際に P17/QL 端子から出力されるパルス波形は連動しています。

ここで、VR の読み取り値をプログラムで処理して、P17/QL 端子の duty 比を決めている事に注意願います。(プログラムの介在なしに、VR と duty が連動しているわけではない)

TRDGRC0 のレジスタ値を変えると、P17 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。

なお、VR→duty の変化はリアルタイムに追随(10ms に 1 回)し、画面表示は 5 秒に 1 回行っています。

○PWM 波形出力(タイマ RD)

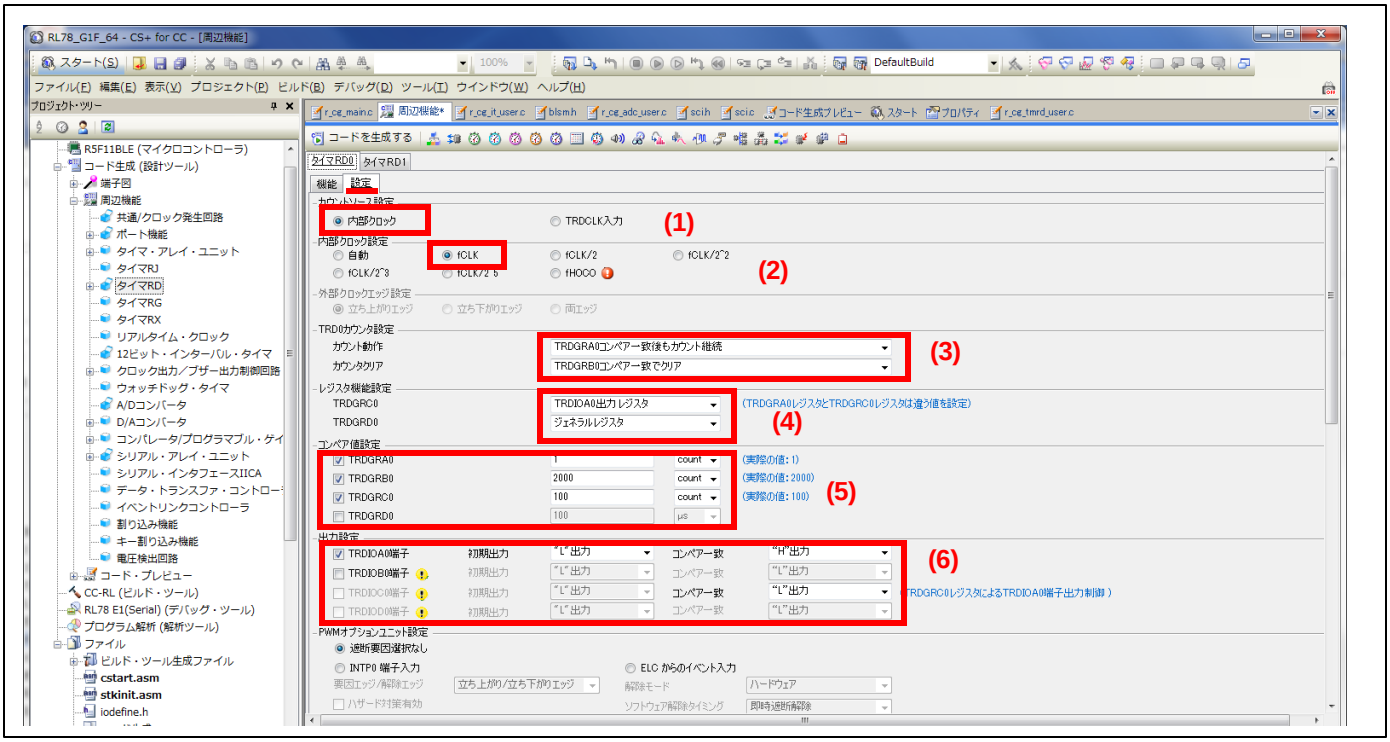


タイマ RD

タイマ RD0 タブ

機能タブ

(1)アウトプットコンペア機能



設定タブ

(1)カウントソース設定　内部クロック

(2)内部クロック設定　fCLK（20MHz）(*1)

(3)TRD0 カウンタ設定

　カウント動作　TRDGRA0 コンペアー致後もカウント継続

　カウンタクリア　TRDGRB0 コンペアー致でクリア

(4)レジスタ機能設定

　TRDGRC0　TRDIOA 出力レジスタ

　TRDGRD0　ジェネラルレジスタ

(5)コンペア値設定

　TRDGRA0　1 count

　TRDGRB0　2000 count

　TRDGRC0　100 count

(6)出力設定

　TRDIOA 端子

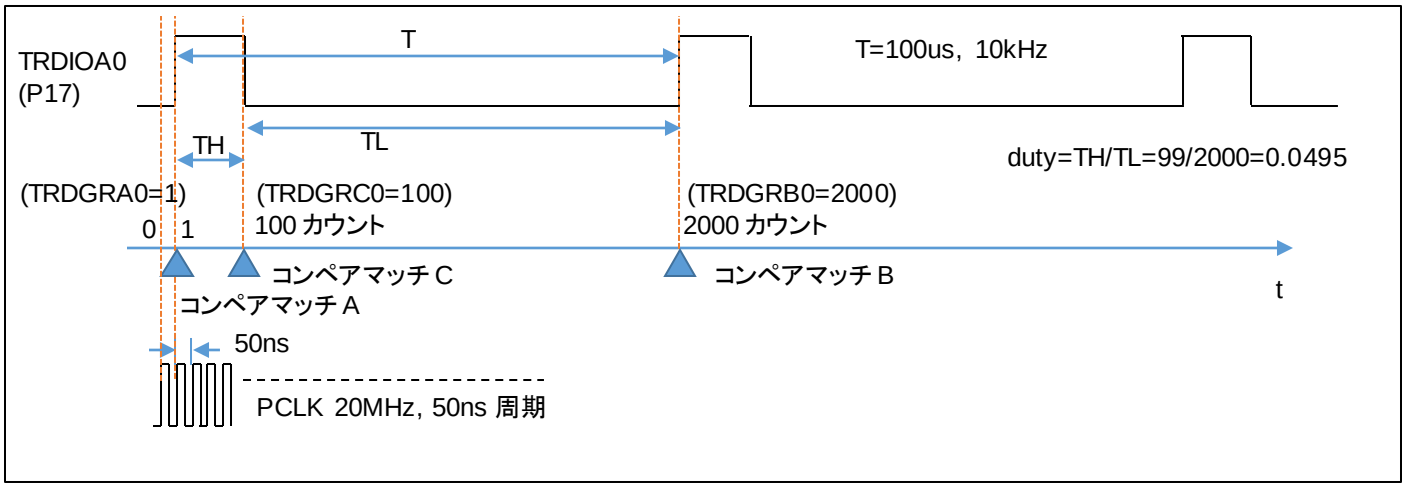
　　初期出力　L 出力

　　コンペアー致 H 出力

　TRDIOC 端子（TRDGRC0 レジスタによる TRDIOA 端子出力制御）

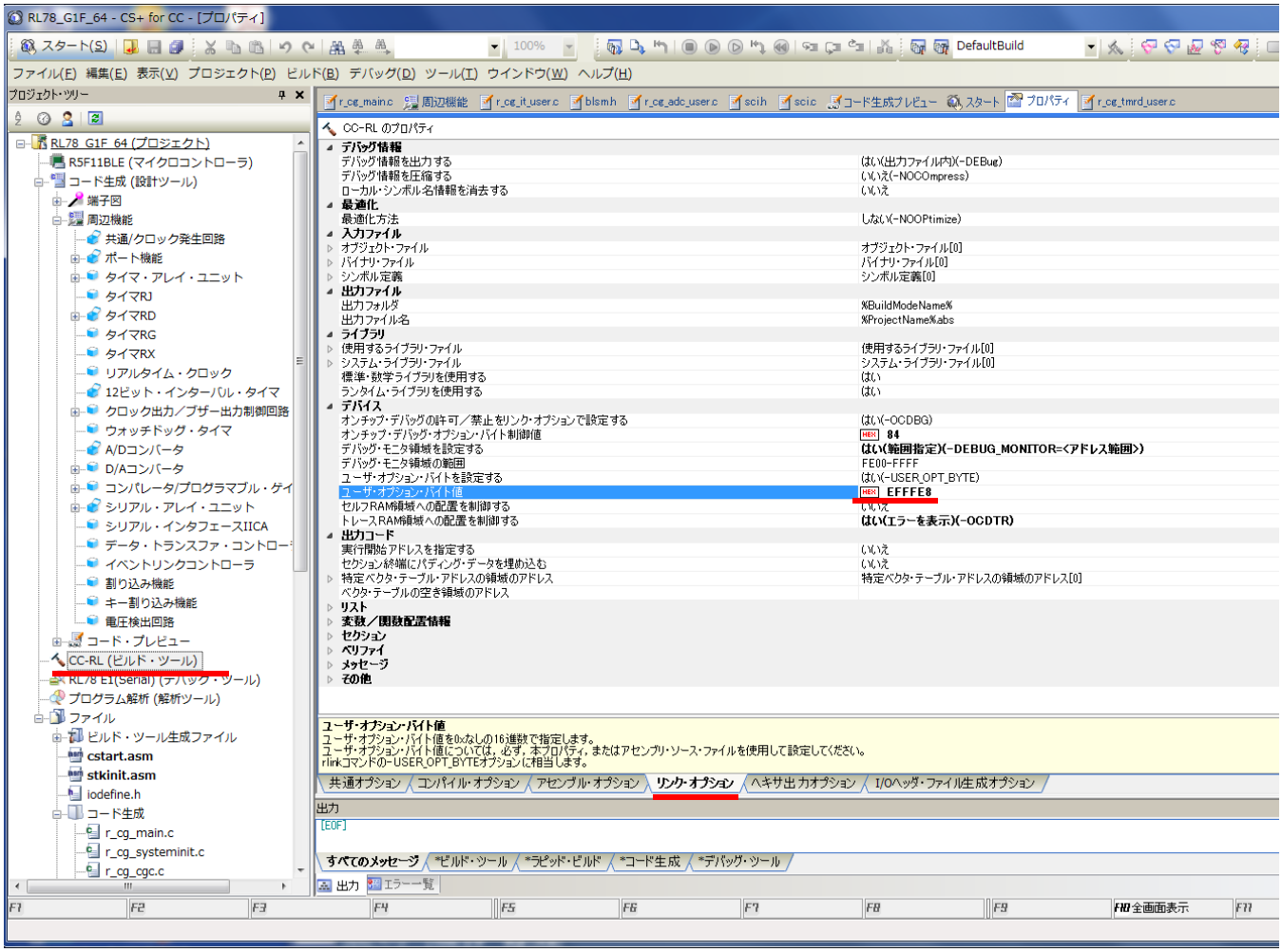
　　コンペアー致 L 出力

(*1)ユーザオプションバイト値でクロックソースを 20MHz にするか、32MHz にするかが変わります。



タイマ RD のアウトプットコンペア機能を用い、PWM 波形を生成することができます。全体の周期を決めるのが、TRDGRB0 となり、TRDGRA0, TRDGRC0 の 2 つの値で出力を切り替え、任意の duty の波形を TRDIOA0(P17)から出力可能です。ここでは、周期(TRDGRB0)は固定とし、TRDGRC0 の値を変える事により duty を変更しています。

ークロックソースに関してー



タイマ RD（タイマ RX も同様）のクロックソースは、ユーザオプションバイト値で変化します。

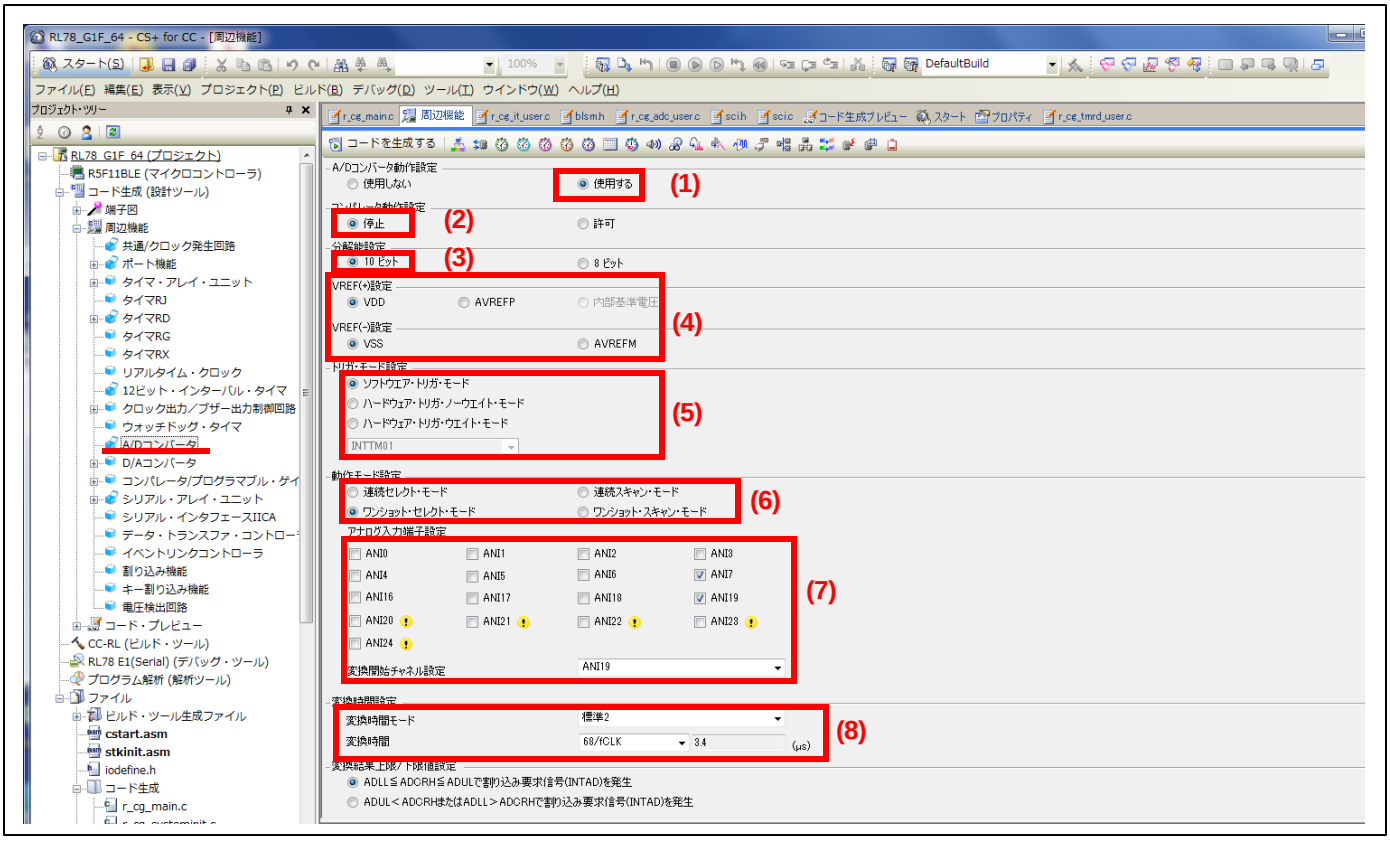
CC-RL（ビルド・ツール）
リンクオプションタブ
ユーザ・オプション・バイト値

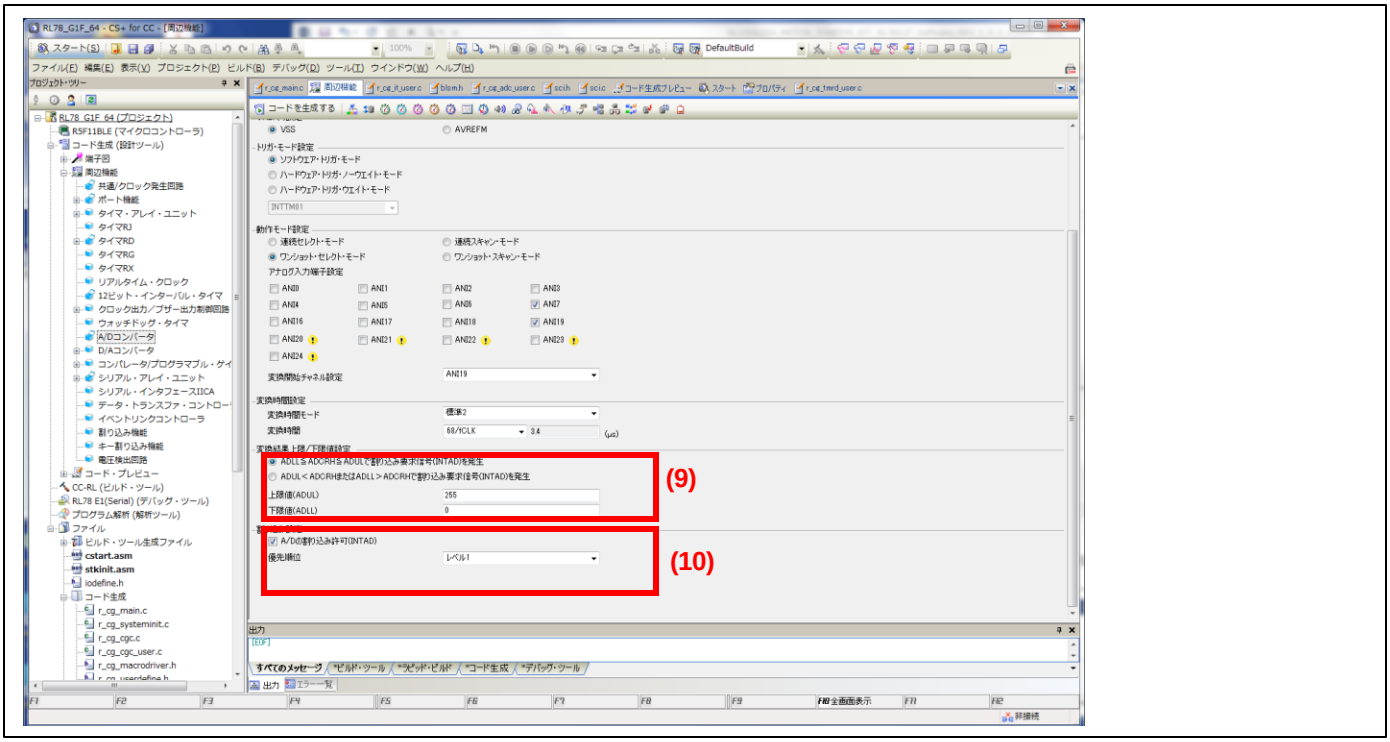
(1) EFFF**F**8 ：FRQSEL4=1, クロックソースは、64MHz の高速オンチップ・オシレータ
(2) EFFF**E**8 ：FRQSEL4=0 クロックソースは、20MHz の高速システム・クロック

(1)を選択した場合は、PWM 波形の周期が 33.25us となります。周期は変わりますが、duty は基本的には変わりません。本チュートリアルでは、オプションバイトとしてどちらを選択しても動作します。

なお、サンプルプログラムでは、回転数の検出を行う際 20MHz をベースに計算を行っているので、ユーザ・オプション・バイト値は、(2)の EFFF**E**8 である必要があります。

○A/D 変換（A/D コンバータ）





A/D コンバータ

(1)A/D コンバータ動作設定　使用する

(2)コンパレータ動作設定　停止

(3)分解能設定　10 ビット

(4)VREF 設定

VREF+設定　VDD

VREF-設定　VSS

(5)トリガ・モード設定　ソフトウェア・トリガ・モード

(6)動作モード設定　ワンショット・セレクト・モード

※別なチュートリアルでは、連続スキャンモードも使用していますが、A/D 変換実行時にその都度設定しています

(7)アナログ入力端子設定

ANI7

ANI19

変換開始チャネル設定　ANI19　※A/D 変換実行時にその都度設定しています

(8)変換時間設定

変換時間モード　標準 2

変換時間　68/fCLK

(9)変換結果上限／下限値設定　ADLL ≦ ADCRH ≦ ADUL で割り込み要求信号(INTAD)を発生

上限値(ADUL)　255

下限値(ADLL)　0

(10)割り込み設定　A/D の割り込み許可(INTAD)　チェック

優先順位　レベル 1

※A/D 変換の終了を割り込みで通知する

BLS_MOTOR3_RL78G1F64¥cg_src¥r_cg_it_user.c

```
static void __near r_it_interrupt(void)          10ms に 1 回呼び出される
{
    /* Start user code. Do not edit comment generated here */
    unsigned short temparature_ad_val, vr_ad_val;
    unsigned short temparature;
    unsigned char i;
    static unsigned long timer_counter=0;

    extern unsigned char adc_flag;
    unsigned long adc_wait;

    timer_counter++;

    EI();//multiple interrupt      多重割り込み許可（AD 変換の終了時の割り込みを受け付けるため）

    //VR ADC
    vr_ad_val=0;
    ADM0_bit.no6 = 0;//ADMD=0, select mode          VR（可変抵抗）の
    ADS = 19;//ANI19 select                        A/D 変換
    adc_flag=0;
    adc_wait=0;
    R_ADC_Start();//Conversion start
    while( (adc_flag == 0) && (adc_wait < ADC_WAIT_MAX) ) adc_wait++;//wait

    if(adc_flag == 1) R_ADC_Get_Result(&vr_ad_val);

    //tempatature ADC
    temparature_ad_val=0;
    ADM0_bit.no6 = 0;//ADMD=0, select mode          温度センサの
    ADS = 7;//ANI7 select                          A/D 変換
    adc_flag=0;
    adc_wait=0;
    R_ADC_Start();//Conversion start
    while( (adc_flag == 0) && (adc_wait < ADC_WAIT_MAX) ) adc_wait++;//wait

    if(adc_flag == 1) R_ADC_Get_Result(&temparature_ad_val);

    //tempatature calc
    for(i=0; i<80; i++)          温度値への変換
    {
        if(temparature_ad_val < g_blsm_ad_val_table[i]) break;
    }
    temparature = g_blsm_temp_table[i];

    TRDGRC0 = (unsigned short)((float)(vr_ad_val)/1023.0*2000.0);

    if((timer_counter % 500) == 0)          VR の A/D 変換値より duty 値を決める
    {
        sciPrint("¥n");                     シリアル端末への表示は 5 秒に 1 回行う
        sciPrint("Tempatature(A/D value) : ");   （A/D 変換及び duty の更新は 10ms 毎）
        sciPrintShortInt(temparature_ad_val);
        sciPrint("¥n");

        sciPrint("Tempatature(degree)    : ");
        sciPrintShortInt(temparature);
        sciPrint("¥n");

        sciPrint("VR(A/D value)          : ");
        sciPrintShortInt(vr_ad_val);
        sciPrint("¥n");

        sciPrint("P17 duty               : ");
        sciPrintShortInt(TRDGRC0);
        sciPrint(" / 2000 ¥n");
    }
    /* End user code. Do not edit comment generated here */
}
```

1.6. モータを回してみる

参照プロジェクト：TUTORIAL4

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR4_RL78G1F64¥RL78_G1F_64.mtpj

を開いてください。

プログラムの動作としては、以下となります。

SW1 を OFF にしてください。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。

その後、SW1 を ON にしてください。

※ブルーの矢印はボード下方向から見た場合の向きです

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%で、電流は、0.15A~0.25A 程度です、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 1.2A 程度となります、本プログラムの duty は最大 18%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

本チュートリアルでは、12 ビット・インターバルタイマを使用しています。

以前のチュートリアルでは、インターバル期間を 10ms としていましたが、本チュートリアルでは、133.34us としています。

BLS_MOTOR4_RL78G1F64¥cg_src¥r_cg_it_user.c

```
static void __near r_it_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

    //every 133us

    unsigned short temparature_ad_val, vr_ad_val;
    unsigned short temparature;
    unsigned char i;

    extern unsigned char adc_flag;
    unsigned long adc_wait;

    static unsigned char state=0;

    unsigned short p1_reg;

    EI();//multiple interrupt

    intr_count++;

    if((intr_count == 50) || (intr_count == 100))//every 6.67ms
    {

        P6_bit.no3 = ~P6_bit.no3;//LED2:reverse

        p1_reg = P1 & 0x40;//b6 keep (b7 -> 0)

        if(active == 1)
        {
            switch(state)
            {
                case 0:
                    P1 = W_V_DIRECTION | p1_reg;
                break;
                case 1:
                    P1 = U_V_DIRECTION | p1_reg;
                break;
                case 2:
                    P1 = U_W_DIRECTION | p1_reg;
                break;
                case 3:
                    P1 = V_W_DIRECTION | p1_reg;
                break;
                case 4:
                    P1 = V_U_DIRECTION | p1_reg;
                break;
                case 5:
                    P1 = W_U_DIRECTION | p1_reg;
                break;
                default:
                break;
            }
            state++;
            if(state == 6) state=0;
        }
        else
        {
            P1 = OFF_DIRECTION | p1_reg;
        }
    }
```

本割り込みルーチンは 133.34us 毎に呼び出される

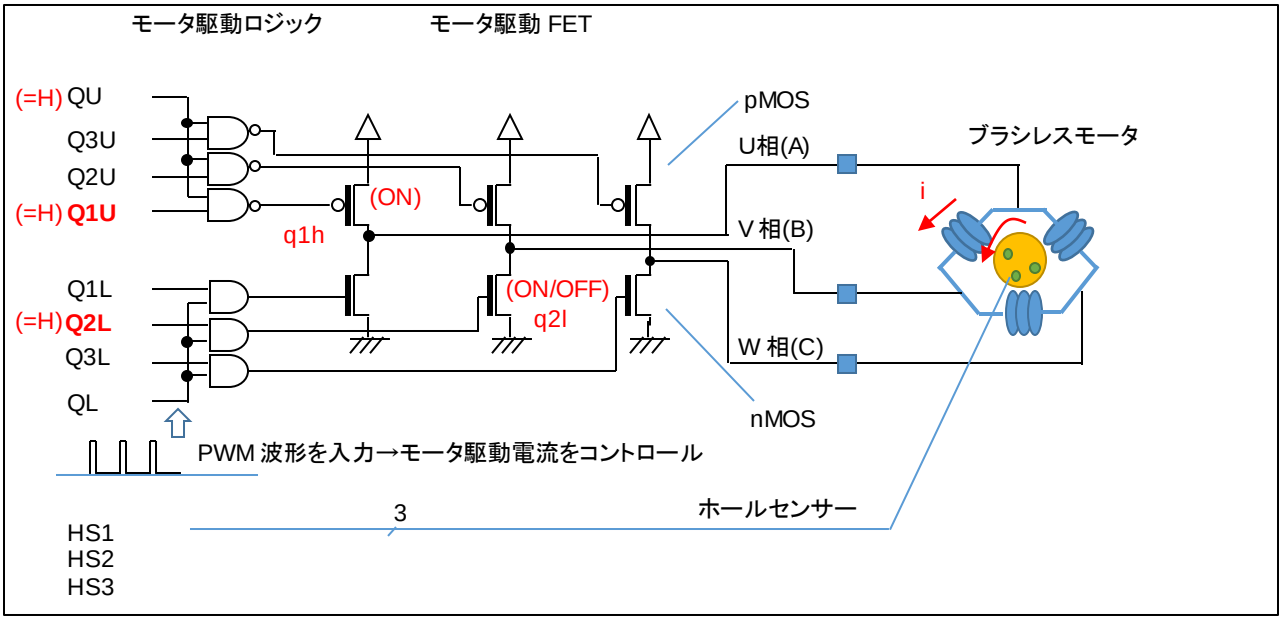
本割り込みルーチンが 50 回呼び出される毎に
LED2 を反転（点灯→消灯， 消灯→点灯）
及び、モータに印加する磁界を変更させる処理
を行う

6.67ms で次の状態に移行

SW1=OFF のときは、電流を止める

この後 13.34ms 毎に、VR の読み取りと P17(QL)の duty 設定を行っている（TUTORIAL3 と同様、
記載は省略）

本プログラムは、タイマ割り込み（12 ビット・インターバル・タイマ）を T=133.34us 毎に呼び出し、50T(6.67ms)毎にモータの印加磁界を次の相に変更。100T(13.34ms)毎に VR を読み取って QL の duty に反映させています。（LED2 は、50T 毎に ON/OFF が切り替わります）



プログラム的に"P1 = U_V_DIRECTION | p1_reg;"のとき、QL には PWM 波形が入力されています。このとき、q1h は ON(6.67ms 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 比が平均電流となるので、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。

なお、本プログラムでは、6.67ms 毎に 1/6 回転する（回転数は固定）としています。1 回転で、40ms なので、回転数としては、25 回転/s。1 分あたりの回転数は、1500rpm 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、1500rpm 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 1500rpm に制限されているため）。

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。

1.7. ホールセンサの値をみる

参照プロジェクト:TUTORIAL5

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR5_RL78G1F64¥RL78_G1F_64.mtpj

を開いてください。

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=OFF とします。

・シリアル端末から出力される情報

```
BLUSHLESS  MOTOR STARTER  KIT TUTORIAL5
motor pos = 01
```

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

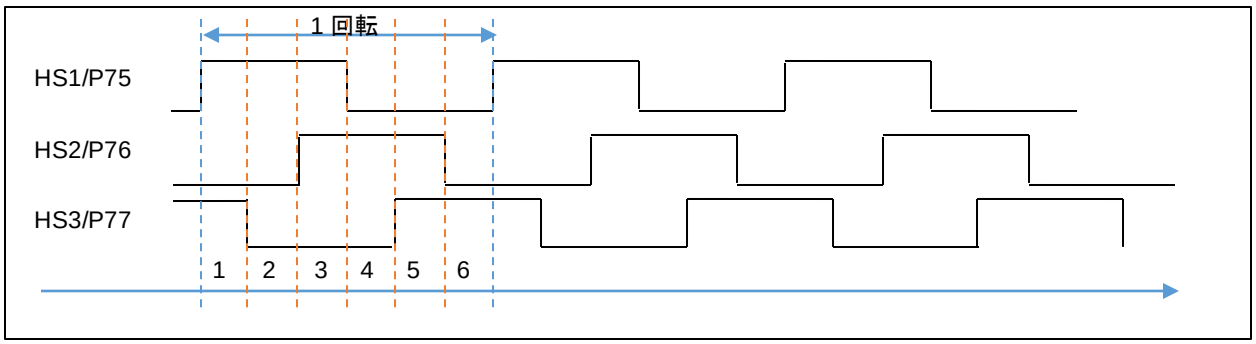
38400bps, 8bit, none, 1bit の設定で表示できます

太字の表示上をカーソルが動いていると思います。表示は、13.34m 秒間隔で更新しています。ここで、モータの軸を手でまわしてみてください。数字が 01 から 06 まで変化したと思います(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。モータの軸を

01 [←方向回転] → 05 [→方向回転] → 01

上記の様に左右に動かすと、数値は 01→05→01 の様に元に戻ると思います。この数値は、軸の絶対位置を表しています。



本プログラムで表示される数値は、

数値(motor pos)	上図の 1~6	P77 (b2)	P76 (b1)	P75 (b0)
05	1	1	0	1
01	2	0	0	1
03	3	0	1	1
02	4	0	1	0
06	5	1	1	0
04	6	1	0	0

motor_pos = P77 × 4 + P76 × 2 + P75
となります。(プログラムの処理を容易にするため、P75, P76, P77 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, SW2 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 03→02 または 03→01(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。

ここで、SW1 を ON にしてみてください。(SW1 を ON にすると、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 1500rpm ぐらいで回る(40ms で 1 回転)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(正確には、電流の向きを変えるのは、12 ビット・インターバル・タイマ割り込みの 133.34us 刻みですが)電流の向きを変化させます。

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 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(motor pos=05→01 等)、そこ(motor_pos=01)で同じ場所(motor_pos=01)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)

BLS_MOTOR5_RL78G1F64¥cg_src¥r_cg_it_user.c

133.34us に 1 回呼び出される

```
static void __near r_it_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

    //every 133us

    unsigned short temparature_ad_val, vr_ad_val;
    unsigned short temparature;
    unsigned char i;

    extern unsigned char adc_flag;
    unsigned long adc_wait;

    unsigned short p1_reg;

    unsigned char pos;

    EI();//multiple interrupt

    intr_count++;


    P6_bit.no3 = ~P6_bit.no3;//LED2:reverse

    //     P75:b0        P76:b1            P77:b2
    pos = (P7_bit.no5) | (P7_bit.no6 << 1) | (P7_bit.no7 << 2);
                                                    ホールセンサ値の読み取り
    p1_reg = P1 & 0x40;//b6 keep (b7 -> 0)

    if(active == 1)
    {
        switch(pos)
        {
            case 1:
                P1 = W_V_DIRECTION | p1_reg;          現在のモータ回転子の位置に応じて
            break;                                    流す電流の向きを決める
            case 3:
                P1 = U_V_DIRECTION | p1_reg;
            break;
            case 2:
                P1 = U_W_DIRECTION | p1_reg;
            break;
            case 6:
                P1 = V_W_DIRECTION | p1_reg;
            break;
            case 4:
                P1 = V_U_DIRECTION | p1_reg;
            break;
            case 5:
                P1 = W_U_DIRECTION | p1_reg;
            break;
            default:
            break;
        }
    }
    else
    {
        P1 = OFF_DIRECTION | p1_reg;
    }
```

1.8. 過電流・過熱保護の動作

参照プロジェクト：TUTORIAL6

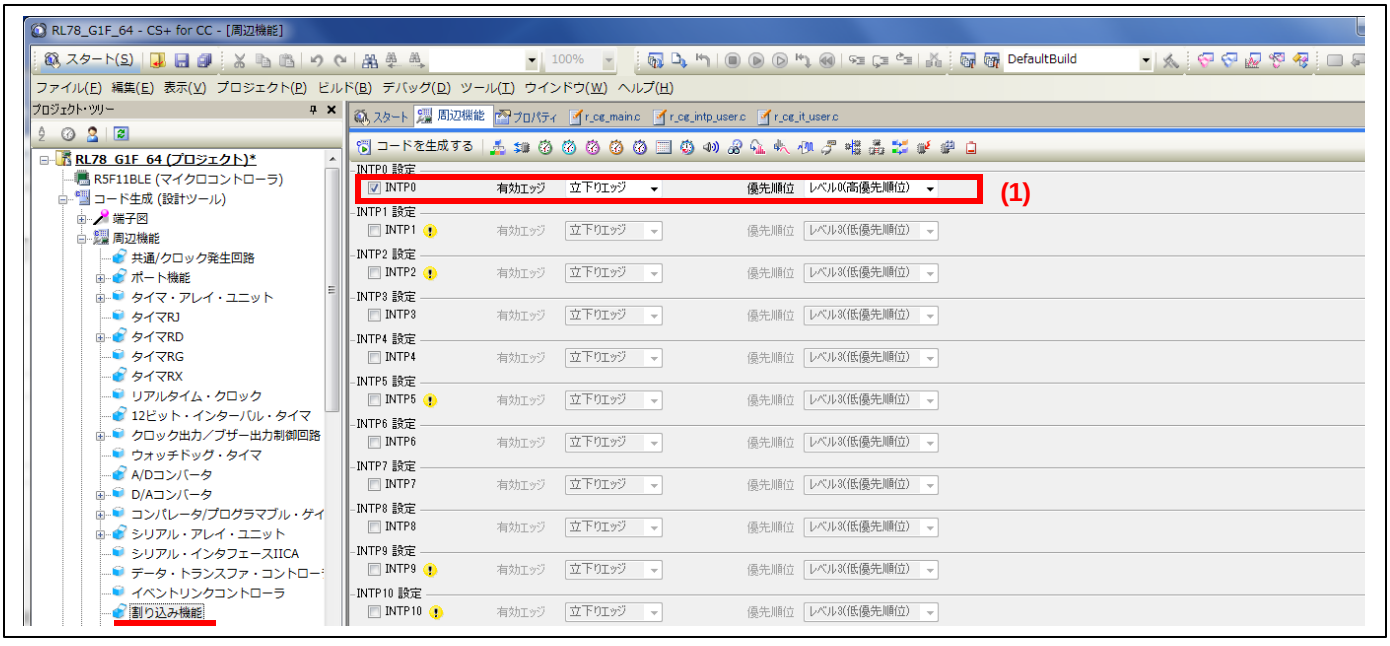
プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR6_RL78G1F64¥RL78_G1F_64.mtpj

を開いてください。

SW1 を ON にし、VR を回すとモータが回転します。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。このとき、LED2 が点滅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で、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）。マイコン側で、この信号は、P137/INTP0 につながっており、本プログラムでは INTP0 の端子割り込みが発生すると、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、LED2 が点滅します）

〇端子割り込み(INTP0)



割り込み機能

(1)INTP0 設定　　INTP0 にチェック

有効エッジ　　立下りエッジ

優先順位　　レベル 0（高優先順位）

・過電流保護設定部

BLS_MOTOR6_RL78G1F64¥cg_src¥r_cg_intp_user.c

```
static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    volatile unsigned long x;

    //over current detect

    P1 = OFF_DIRECTION;

    sciPrint("***OVER CURRENT STOP***¥n");

    while(1)
    {
        P6_bit.no3 = ~P6_bit.no3;//LED2:reverse
        for(x=0; x<500000; x++);
    }
    /* End user code. Do not edit comment generated here */
}
```

INTP0 の割り込みが入った際
本関数が呼び出される

モータに流れる電流を止める処理
モータ動作で使用しているタイマを止める処理

LED2 を点滅させる

また、温度に関しても(13.3ms に一度)監視をしており、設定温度を超えた場合、モータを止める処理を行います。

duty を高めに設定し、しばらくモータを回した状態を維持するか、ドライヤ等でヒートシンク下部を温めてみてください。シリアル端末を接続しているときは、随時(2.6 秒に 1 回)duty, 温度, 回転数が表示されます。温度が徐々に上がっていき、50℃を超えた時に、モータは停止するはずです。(過熱停止のサインとして LED2 が、2 回点滅・消灯を繰り返します)

・シリアル端末から出力される情報

```
BLUSHLESS  MOTOR STARTER KIT TUTORIAL6
duty                          : 422 / 2000
temparature                   : 33 [deg]
rotation speed                : 5387 [rpm]
rotation speed(average) : 4537 [rpm]
```

duty，温度，回転数を表示します

```
***OVER  TEMPARATURE  STOP***
temparature    : 51 [deg]
```

過熱停止のとき

```
***OVER  CURRENT  STOP***
```

過電流停止のとき

・過熱保護処理

BLS_MOTOR6_RL78G1F64¥cg_src¥r_cg_it_user.c

（前略）

133.34us のタイマ割り込み関数の 100 回に 1 回実行

```
if(intr_count == 100)//every 13.3ms
{
    intr_count=0;
    timer_counter++;

    //VR ADC
    vr_ad_val=0;
    ADM0_bit.no6 = 0;//ADMD=0, select mode
    ADS = 19;//ANI19 select
    adc_flag=0;
    adc_wait=0;
    R_ADC_Start();//Conversion start
    while( (adc_flag == 0) && (adc_wait < ADC_WAIT_MAX) ) adc_wait++;//wait

    if(adc_flag == 1) R_ADC_Get_Result(&vr_ad_val);

    //tempatature ADC
    temparature_ad_val=0;
    ADM0_bit.no6 = 0;//ADMD=0, select mode
    ADS = 7;//ANI7 select
    adc_flag=0;
    adc_wait=0;
    R_ADC_Start();//Conversion start
    while( (adc_flag == 0) && (adc_wait < ADC_WAIT_MAX) ) adc_wait++;//wait

    if(adc_flag == 1) R_ADC_Get_Result(&temparature_ad_val);

    //tempatature calc
    for(i=0; i<80; i++)
    {
        if(temparature_ad_val < g_blsm_ad_val_table[i]) break;
    }
    temparature = g_blsm_temp_table[i];

    if(temparature > BLSM_UPPER_TEMP)//over tempature detect
    {
        P1 = OFF_DIRECTION;
        sciPrint("***OVER TEMPARATURE STOP***¥n");
        sciPrint("temparature  : ");
        sciPrintShortInt(temparature);
        sciPrint(" [deg]¥n¥n");

        while(1)
        {
            P6_bit.no3 = 0;//LED2:ON
            for(x=0; x<500000; x++);
            P6_bit.no3 = 1;//LED2:OFF
            for(x=0; x<500000; x++);
            P6_bit.no3 = 0;//LED2:ON
            for(x=0; x<500000; x++);
            P6_bit.no3 = 1;//LED2:OFF
            for(x=0; x<5000000; x++);
        }
    }

    duty = (unsigned short)((float)(vr_ad_val)/1023.0*2000.0);
    TRDGRC0 = duty;
```

A/D 変換値を温度に変換するところは、1.5 と同じ

過電流停止

```
//温度上限
#define BLSM_UPPER_TEMP 50
```

LED が 2 回点滅・消灯を繰り返す処理

VR に応じた duty を設定

（後略）

本チュートリアルでは、回転数の検出及び表示を行っています。回転数の検出は、タイマ RX で、ホールセンサが切り替わるまでの時間をカウントします。

ホールセンサは、モータが 1/6 回転した際に、出力が切り替わります。タイマの周期は、1.6us（20MHz, 1/32 分周）に設定しています。例えば、ホールセンサ出力が切り替わった際、カウンタ値が 1000(1.6us x 1000 = 1.6ms)の場合は、1 回転に要する時間が 1.6ms x 6 = 9.6ms となり、1 秒あたりの回転数が 104、1 分あたりの回転数が 6250[rpm]と計算されます。

※タイマ RX のクロックソースに関して

ユーザオプションバイトが

EFFF**F**8

となっている場合は、タイマ RX のクロックソースが 64MHz となりますので、観測される回転数が 1/3.2 となります。

（計算式は、1.6us 周期で計算しているが、実際は 0.5us 周期でカウントされる）

正しい回転数を取得できない場合は、ユーザオプションバイト値を

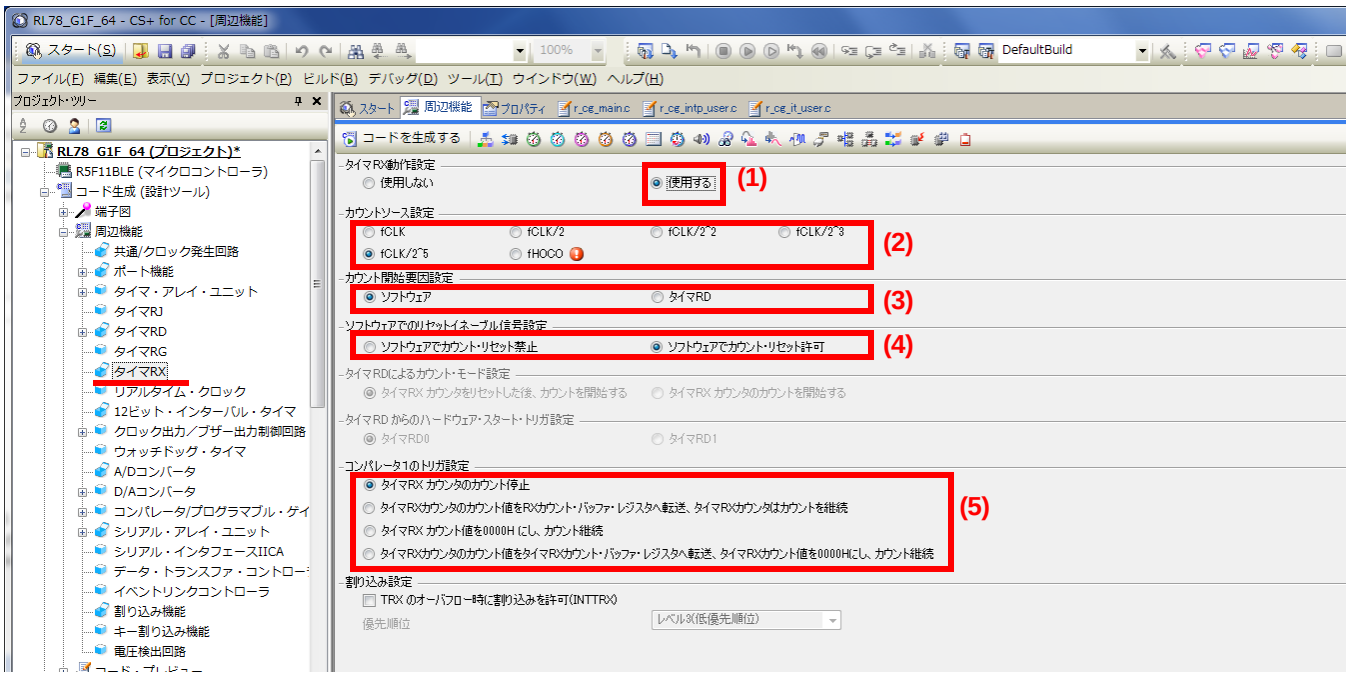
EFFF**E**8

に設定してください。

回転数を回転が止まる寸前まで落とした場合の回転数は約 800rpm 程度です。300rpm 前後の値が表示された場合は、ユーザオプションバイト値が EFFF**F**8 となっている可能性が高いです。

タイマのクロックソースに関しては、1.5 章「クロックソースに関して」の部分を参照のこと。

○タイマ RX



タイマ RX

(1)タイマ RX 動作設定　使用する

(2)カウントソース設定　fCLK/2^5 (1.6us)

(3)カウント開始要因設定　ソフトウェア

(4)ソフトウェアでのリセットイネーブル信号設定　ソフトウェアでカウント・リセット許可

(5)コンパレータ 1 のトリガ設定　タイマ RX カウンタのカウント停止

・回転数の取得

BLS_MOTOR6_RL78G1F64¥blsm¥blsm.c

```
void blsm_rpm_calc(void)      本関数は、133.34us 周期で実行されるインターバルタイマ内で呼び出される
{
    unsigned short pos;
    float rpm_tmp;

    //     P75:b0        P76:b1            P77:b2
    pos = (P7_bit.no5) | (P7_bit.no6 << 1) | (P7_bit.no7 << 2);

    if(pos != g_blsm_prev_pos)
    {
        if((1 <= pos) && (pos <= 6))
        {
            rpm_tmp = 6.25e6 / (float)TRX;//rpm = 1/(T*1.6u)*60/6 = 6.25e6/T
            g_blsm_rpm = (unsigned short)rpm_tmp;
            TRX = 0;//TRXリスタート

            g_blsm_direction=127;//127:不正値
            switch(pos)
            {
                case 5://pos0
                    if(g_blsm_prev_pos == 4) g_blsm_direction=1;//pos5
                    if(g_blsm_prev_pos == 1) g_blsm_direction=-1;//pos1
                break;
                case 1://pos1
                    if(g_blsm_prev_pos == 5) g_blsm_direction=1;//pos0
                    if(g_blsm_prev_pos == 3) g_blsm_direction=-1;//pos2
                break;
                case 3://pos2
                    if(g_blsm_prev_pos == 1) g_blsm_direction=1;//pos1
                    if(g_blsm_prev_pos == 2) g_blsm_direction=-1;//pos3
                break;
                case 2://pos3
                    if(g_blsm_prev_pos == 3) g_blsm_direction=1;//pos2
                    if(g_blsm_prev_pos == 6) g_blsm_direction=-1;//pos4
                break;
                case 6://pos4
                    if(g_blsm_prev_pos == 2) g_blsm_direction=1;//pos3
                    if(g_blsm_prev_pos == 4) g_blsm_direction=-1;//pos5
                break;
                case 4://pos5
                       if(g_blsm_prev_pos == 6) g_blsm_direction=1;//pos4
                       if(g_blsm_prev_pos == 5) g_blsm_direction=-1;//pos0
                break;
                }
            }
    }

    if(TRX > 62500)//100msセンサーの遷移を観測できない場合
    {
        g_blsm_rpm=0;
        g_blsm_direction=0;
        TRX = 0;//TRXリスタート
    }

    g_blsm_prev_pos=pos;

}
```

回転数の計算
（タイマの時間から rpm 値を計算
タイマの刻みは 1.6us）

回転方向の取得

一定期間センサの出力が切り替わらないときは
停止していると判断する

1.9. 相電圧・相電流の観測

参照プロジェクト：TUTORIAL7

プロジェクト一式をローカルのディスクにコピーし、

BLS_MOTOR7_RL78G1F64¥RL78_G1F_64.mtpj

を開いてください。

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 と同じです。

但し、TUTORIAL6 と異なる点があり、（モータが回っているときに）SW2 を ON にすると ON にした瞬間から 200 ポイント（133.34us × 100=13.3ms 間）分の相電圧、相電流のデータを取得して表示するという動作となります。（データ表示に伴いモータの回転は止まります。

・シリアル端末から出力される情報

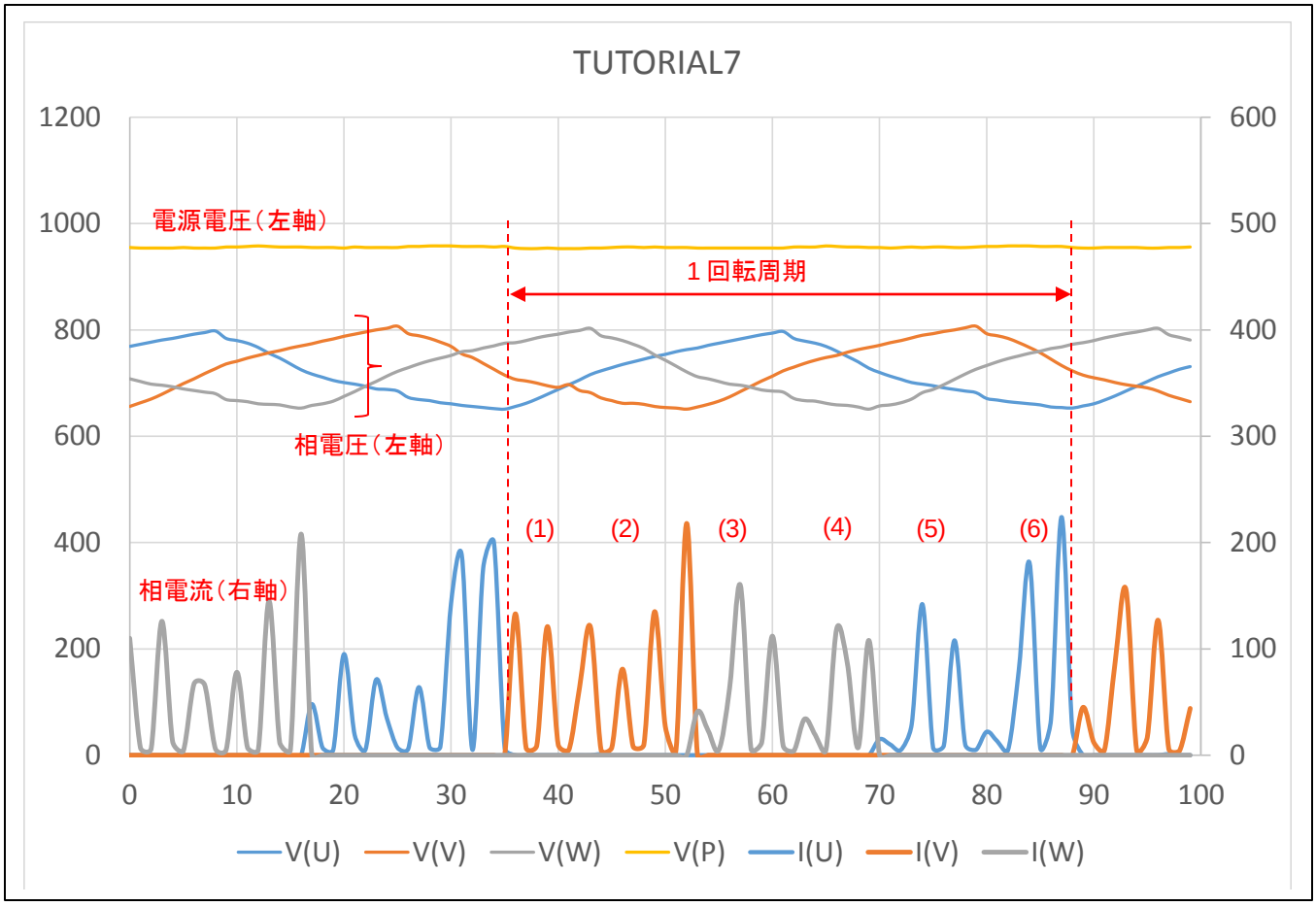
```
duty                          : 678 / 2000
temparature                   : 31 [deg]
rotation speed                : 6883 [rpm]
rotation speed(average) : 7401 [rpm]

                                                    ここで SW2 を ON に切り替える
index,V(U),V(V),V(W),V(P),I(U),I(V),I(W)
0,751,807,691,957,1,0,0
1,743,811,693,957,64,0,0
2,738,815,697,957,3,0,0
3,735,818,703,957,5,0,0
4,730,822,709,958,39,0,0
…
```

duty，温度，回転数を表示します
（TUTORIAL6 と同じ動作）

一瞬（13.3ms のデータ取得）後
100 ポイントのデータが画面に
表示されます
（データは、CSV 形式となっているので
Excel 等に貼り付けてください）

・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



※縦軸は、A/D 変換値(0~1023)

BLS_MOTOR1_RL78G1F64¥cg_src¥r_cg_it_user.c

回転制御のプログラム

```
switch(pos)
{
    case 1:
        P1 = W_V_DIRECTION | p1_reg;        (1)
    break;
    case 3:
        P1 = U_V_DIRECTION | p1_reg;        (2)
    break;
    case 2:
        P1 = U_W_DIRECTION | p1_reg;        (3)
    break;
    case 6:
        P1 = V_W_DIRECTION | p1_reg;        (4)
    break;
    case 4:
        P1 = V_U_DIRECTION | p1_reg;        (5)
    break;
    case 5:
        P1 = W_U_DIRECTION | p1_reg;        (6)
    break;
    default:
    break;
}
```

W→V に電流を流す：上のグラフの(1)に対応

時系列

波形は、端末に表示された数値をプロットしたものである。数値は 10bit A/D 変換の生データ（0-1023）となっています。相電圧（モータドライバボード上でフィルタリングされている）は、sin 波に近い波形。相電流は、この場合 PWM 駆動（パルス制御）しているので、電流が流れている状態と止まっている状態の繰り返しとなります。

※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できます。

モータ制御のチュートリアル編はここまでとなります。モータドライバボードに備わっている機能は一通り試しましたので、この後はマイコンの機能を活用し、様々なアルゴリズムでブラシレスモータを制御してみてください。

なお、RL78/G1F のチュートリアルでは、可能な限り CS+の「コード生成」の機能を使用しています。「コード生成」機能は、プログラムの枠組みやレジスタ初期設定を自動で生成してくれますので、活用すれば便利な機能だと思います。もちろん、ある程度 RL78 のプログラムに慣れてきたら、「コード生成」機能を使わずにレジスタ設定等を書き下す事もできますので、必要に応じて（初めて使用する機能はとりあえず「コード生成」に任せる等）使用してみて下さい。

2. サンプルプログラム

2.1. プログラム仕様

サンプルプログラムは、チュートリアルの内容を踏まえ、回転数や回転方向を制御できるものとなっています。

本プログラム(BLSM_SAMPLE1_HSBRL78G1F64)は、以下の仕様を実装しています。

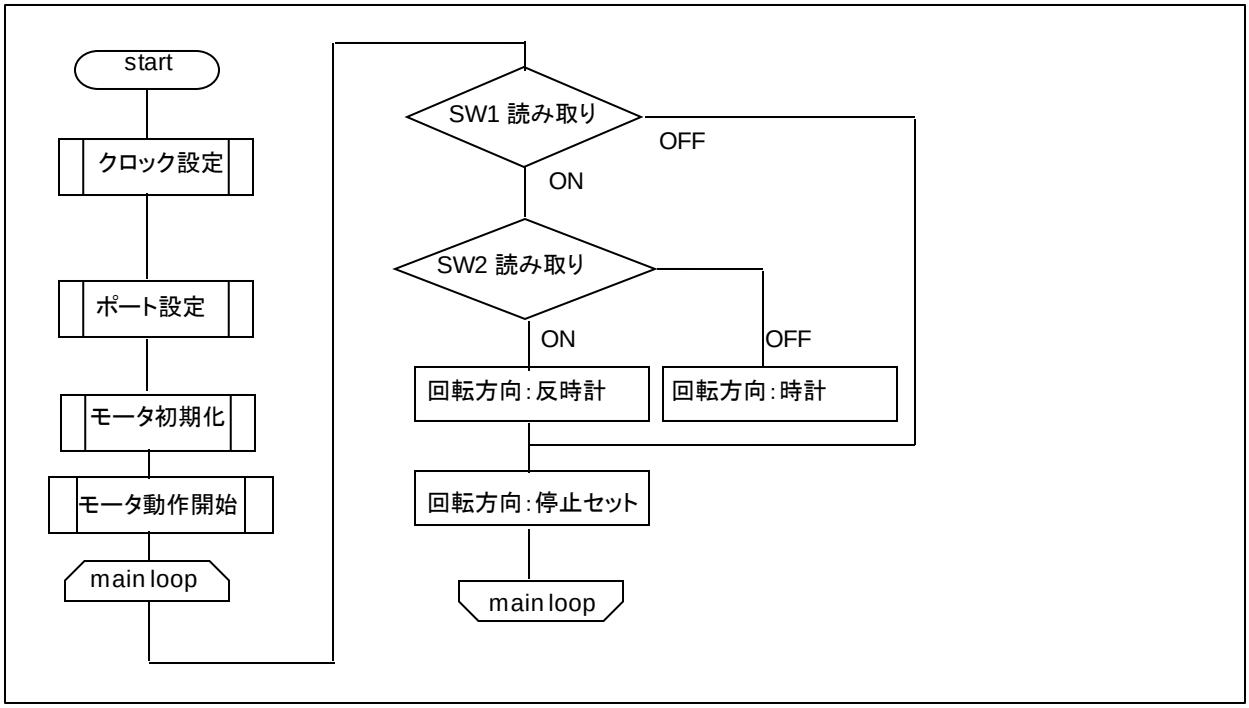
・ホールセンサを使用
・VR にて回転数を変更
・SW1 で回転・停止を制御
・SW2 で回転方向を制御
・過電流保護停止機能
・過熱停止機能

SW1　chA ON:モータ回転, OFF:モータ停止
SW2　chA ON:反時計回り, OFF:時計回り

モータドライバボード上 VR　回転数制御

2.2. 関数仕様

2.2.1. 全体及び main 関数



2.2.2. モータ制御関数

blsm_motor_init

概要:初期化関数
宣言:void blsm_motor_init(void)
説明:
・変数の初期化
を行います
引数:なし
戻り値:なし

blsm_motor_start

概要:モータ制御開始関数
宣言:void blsm_motor_start (void)
説明:
・タイマーのカウント開始
を行います
引数:なし
戻り値:なし

blsm_motor_stop

概要:モータ制御停止関数
宣言:void blsm_motor_stop (void)
説明:
・タイマー停止
・変数の初期化
を行います
引数:なし
戻り値:なし

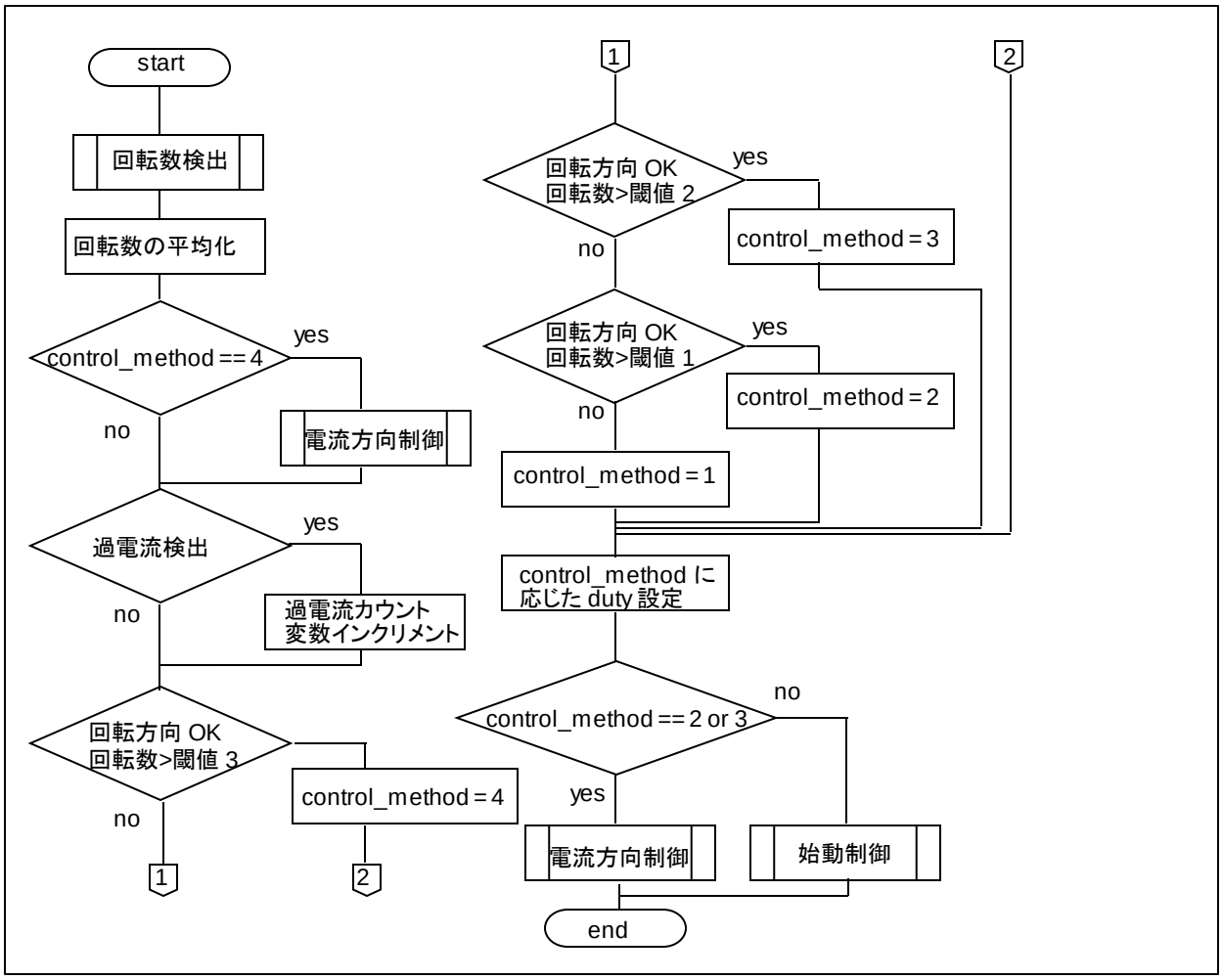
2.2.3. 割り込み関数

モータ制御プログラム本体の処理は、割り込みを使用して行います。

int_timer1

概要:タイマー割り込み(133.34us 周期)
宣言:void int_timer1 (void)
説明:
・回転数・回転方向検出
・モータ制御(センサー位置に応じた磁界をかける)
・始動制御(始動時のみ)
を行います
引数:なし
戻り値:なし

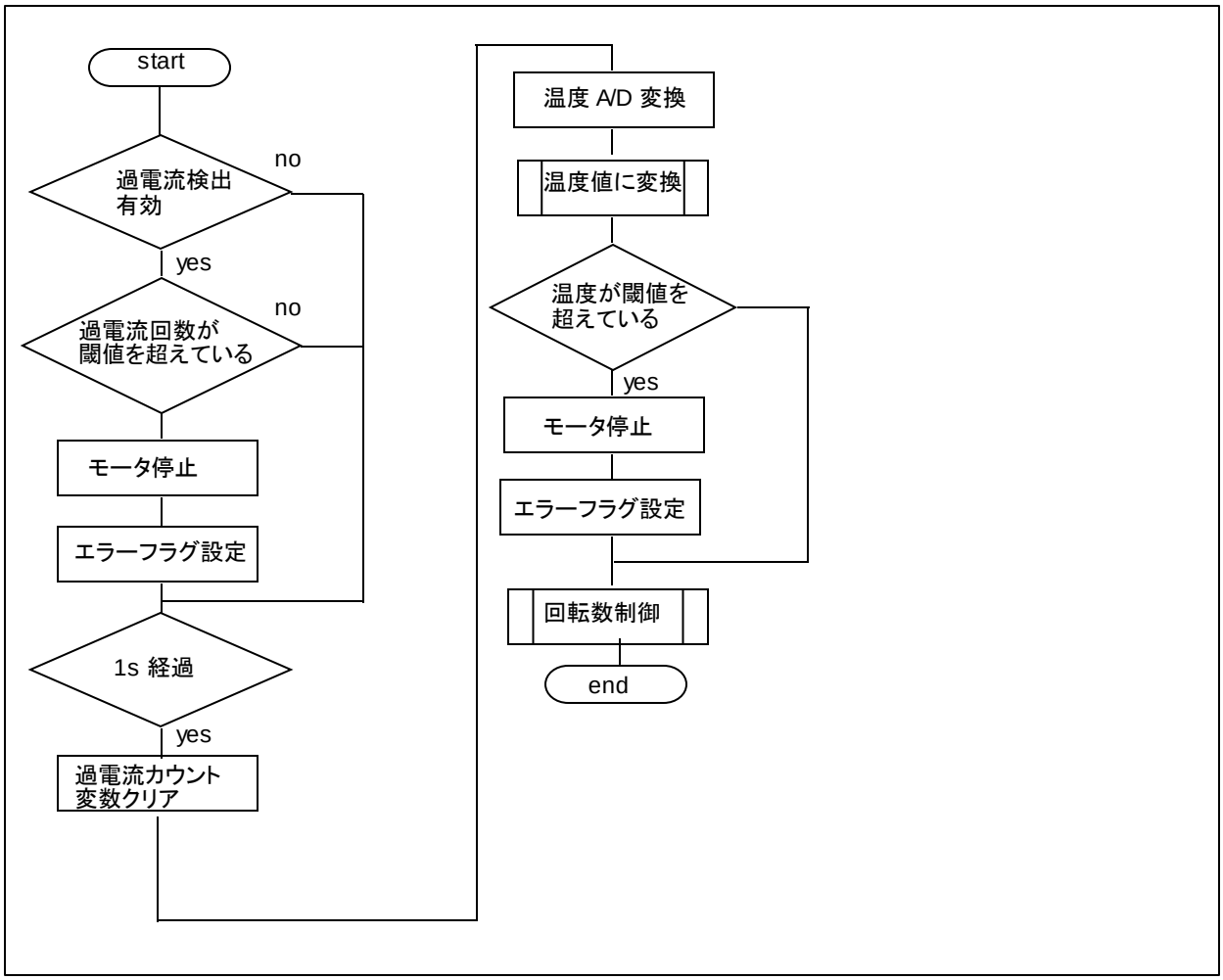
ーフローチャートー



int_timer0

概要:タイマー割り込み(13.3ms 周期)
宣言:void int_timer0 (void)
説明:
・過電流検出停止処理
・過熱停止処理・始動制御(始動時のみ)
・回転数制御(目標回転数と現時点の回転数を比較して、モータ電流を増減させる)
を行います
引数:なし
戻り値:なし

ーフローチャートー



2.2.4. その他内部で呼び出される関数

blsm_motor_control

概要:モータ制御関数
宣言:void blsm_motor_control (void)
説明:
・センサー位置と回転方向に応じた磁界をかける処理
を行います
引数:なし
戻り値:なし

blsm_motor_control_pos_phase

概要:モータ始動制御関数
宣言:void blsm_motor_control_pos_phase (void)
説明:
・センサー位置と回転方向に応じた、1ステップ先の磁界をかける処理
を行います
始動時により大きなテンションをかけるために使用
引数:なし
戻り値:なし

blsm_motor_control_neg_phase

概要:モータ制御関数
宣言:void blsm_motor_control_neg_phase (void)
説明:
・センサー位置と回転方向に応じた、逆方向の磁界をかける処理
を行います
始動時に逆方向のテンションをかけるために使用
引数:なし
戻り値:なし

blsm_rpm_calc

概要:モータ制御関数
宣言:void blsm_rpm_calc (void)

説明:
・現時点の回転数
・回転方向
を取得します
引数:なし
戻り値:なし(グローバル変数に格納)

2.2.5. グローバル変数

(1)本プログラムの入力となる変数

g_blsm_target_direction

概要:モータの回転方向を決める変数
宣言:short g_blsm_target_direction
説明:
0:モータは停止します
1:モータは反時計回りします
-1:モータは時計回りします

g_blsm_target_rpm

概要:モータの回転数を決める変数
宣言:unsigned short g_blsm_target_rpm
説明:モータの目標回転数(rpm)を指定します
2000~12000 の値が有効です。

上記変数に値をセットすると、セットされた値に応じてモータの回転が制御されます。

(2)本プログラムで書き換えられる変数（抜粋）

以下の変数は任意のタイミングで参照可能です。

g_blsm_direction

概要：モータの回転方向を示す変数
宣言：short g_blsm_direction
説明：
0: モータは停止しています
1: モータは反時計回りで回転しています
-1: モータは時計回りで回転しています
127: 回転方向が検出できない

g_blsm_rpm

概要：モータの回転数(rpm)を示す変数
宣言：unsigned short g_blsm_rpm
説明：モータの回転数を rpm で示します

g_blsm_rpm_ave

概要：モータの回転数(rpm)の平均値示す変数
宣言：unsigned short g_blsm_rpm_ave
説明：モータの回転数を rpm で示します

g_blsm_temparature

概要：モータドライバボードの温度を示す変数
宣言：short g_blsm_temparature
説明：基板の温度[℃]を示します

g_blsm_control_method

概要:制御のステージを示す変数
宣言:unsigned char g_blsm_control_method
説明:
0: 停止状態
1: 始動制御(<100rpm)
2: 始動開始(1)(<500rpm)
3: 始動開始(2)(<1000rpm)
4: 通常制御(>1000rpm)

g_blsm_error_code

概要:エラー種別を示す変数
宣言:unsigned char g_blsm_error_code
説明:
0: エラーが発生していない
1: 過電流検出
2: 過熱検出

※エラーが発生した場合は、本変数にエラー種別を設定してモータは停止します(g_blsm_target_direction=0 とります)
※エラーフラグのクリアは本変数に 0 を書き込むか、blsm_motor_stop()で行ってください
※エラーフラグが立った状態でも、g_blsm_target_direction=1, -1 をセットする事によりモータは再び回転します

2.2.6. プログラムの動作を制御する定義値

#define BLSM_POS_PHASE_CONTROL_METHOD

定義時:モータ回転時印加する磁界を 1/6 位相進める
未定義時:通常の制御を行う

#define BLSM_OVER_CURRENT_CONTROL 1

過電流検出を行いモータを停止させる場合"1"を定義。

#define BLSM_OVER_CURRENT_COUNT_THRESHOLD 1000

過電流停止の閾値を定義する。133.34us 毎に過電流検出の端子レベルをチェックし、1.3s 間にこの回数以上過電流となっていた場合、モータを停止させる。(大きい数値を設定した方が、過電流によりモータが止まりにくくなる、最大値は 10000)
※10000 以上の値を設定すると、過電流でモータが止まることはない事に注意

#define BLSM_TEMPARATURE_CONTROL 1

過熱検出を行いモータを停止させる場合"1"を定義。

#define BLSM_TEMPARATURE_THRESHOLD 50

過熱停止を行う場合の閾値。[℃]で定義。

#define BLSM_ROTATION_SPEED_CONTROL 1

回転数制御を行う場合"1"を定義

#define BLSM_DUTY_MAX 1800

設定される duty の最大値を定義(1800/2000=90%)

#define BLSM_DUTY_MIN 100

設定される duty の最小値を定義（100/2000=5%）

#define BLSM_RPM_MAX 12000

設定可能な回転数の最大を定義[rpm]

#define BLSM_RPM_MIN 2000

設定可能な回転数の最小を定義[rpm]

#define BLSM_RPM_FEEDBACK_CONST 0.1

回転数制御のフィードバック係数を定義
0.1 の場合は、目標に対し 10%ずつ近づけていく

#define BLSM_RPM_FEEDBACK_THRESHOLD 0.1

回転制御の閾値を定義
0.1 の場合は、目標回転数との誤差 10%以内であれば、回転数（duty）を変更しない

#define BLSM_CONTROL_DUTY0 0

ステージ 0（停止時）の duty

#define BLSM_CONTROL_THRESHOLD1 100

ステージ 1（初期始動）を抜ける回転数[rpm]

#define BLSM_CONTROL_DUTY1_START (340-5)

ステージ 1 の開始時の duty 340(/2000)
※プログラムで最初に 5 を加算するので、予め 5 を減算している

#define BLSM_CONTROL_DUTY1_MAX 375

ステージ 1 の duty の最大値(/2000)

#define BLSM_CONTROL_CYCLE1 200

始動時 1 サイクル持続時間 133.34us × 200 = 26.6ms

#define BLSM_CONTROL_CYCLE1_PRE 80

始動時に逆向きのテンションを加える時間 133.34us × 80 = 10.7ms

#define BLSM_CONTROL_THRESHOLD2 500

ステージ 2 を抜ける回転数[rpm]

#define BLSM_CONTROL_DUTY2 470

ステージ 2 の duty(/2000)

#define BLSM_CONTROL_THRESHOLD3 1000

ステージ 3 を抜ける回転数[rpm]

#define D_BLSM_CONTROL_DUTY3 625

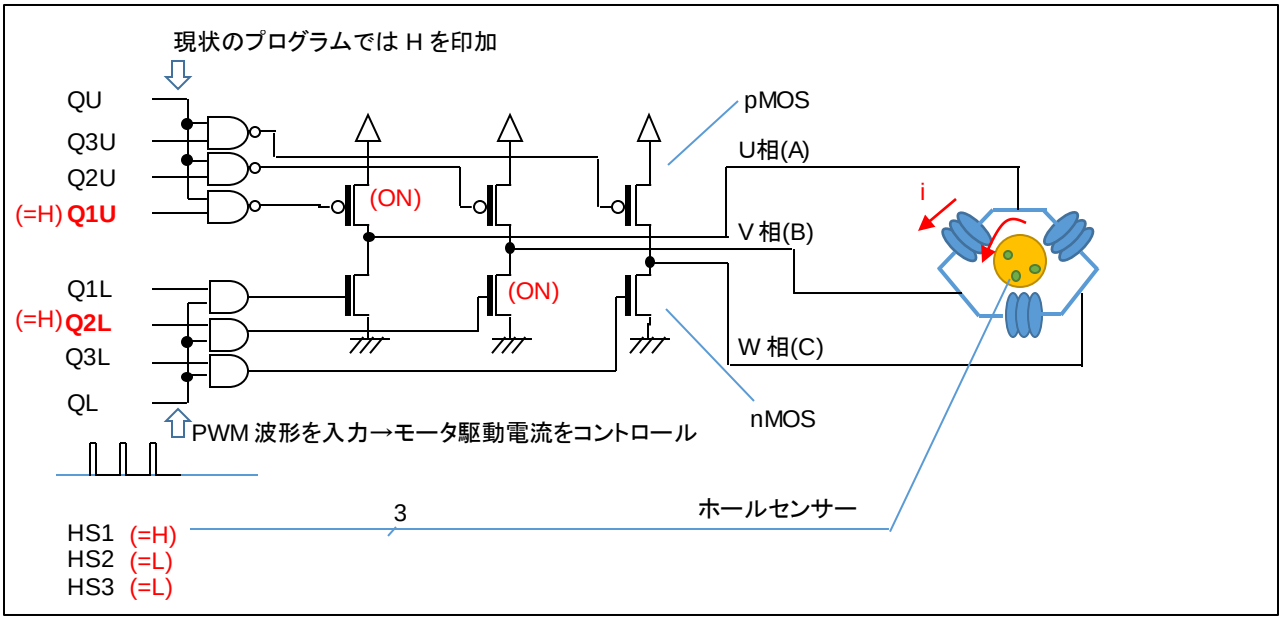
ステージ 3 の duty(/2000)

2.2.1. プログラムで使用しているマイコン機能

機能名	分類	用途
12 ビット・インターバル・タイマ	タイマ	モータ制御タイマー割り込み(133.34us 周期)
タイマ RD	タイマ	PWM 制御 chA
タイマ RX	タイマ	モータ回転数カウント
INTP0(*1)	端子割り込み	過電流検出信号入力
A/D コンバータ	A/D 変換	温度検出, VR 読み取り

(*1)現状のプログラムは割込みとしては使用していません
（割り込みとしてではなく、入力端子として使用）

2.3. モータ制御アルゴリズム



ホールセンサーの出力に応じて、pMOS, nMOS を 1 つずつ ONさせることにより、特定の相電流（上図では、U→V 相に電流を流している）を発生させ、モータのロータにトルクを発生させます。ホールセンサーの出力が切り替わるタイミングで、次の相電流に切り替える事により、回転を維持させます。

回転数に関しては、QL に印加する波形の duty 比を変えてモータに流す電流を変化させる事により制御しています（PWM 波形を生成）。QU は H 固定で使用しています。（QU 側も PWM 制御することは可能）

－ホールセンサ位置と電流の方向の関係－

HS2/HS1/HS0	正転（前進）	逆転（後進）
1 0 1	W→U	V→W
0 0 1	W→V	V→U
0 1 1	U→V	W→U
0 1 0	U→W	W→V
1 1 0	V→W	U→V
1 0 0	V→U	U→W

2.3.1. 始動制御

－始動制御－

ステージ	状態	回転数[rpm]	制御
ステージ 0	停止	0	
ステージ 1	始動制御	<100	(1)10.4ms 正方向のトルクをかける (2)通常の回転制御をかける -----26.6ms 以内に回転数が 100rpm に達すればステージ 1 に移行 (3)10.4ms 逆方向のトルクをかける (4)通常の回転制御をかける -----26.6ms 以内に回転数が 100rpm に達すればステージ 1 に移行
ステージ 2	始動開始(1)	<500	duty 制限（電流制限）がある通常の回転制御
ステージ 3	始動開始(2)	<1000	duty 制限（電流制限）がある通常の回転制御
ステージ 4	通常制御	>1000	目標の回転数に対し duty（電流）制御

始動制御は、上記の方式で行う。一度回転を始めると、ホールセンサーの値に応じた磁界をかければ良いが、始動時はモータの軸に揺さぶりをかける方向の磁界をかける事により初期の回転を促す。

2.3.2. 回転数制御

回転数制御は単純なアルゴリズムで行っています。
13.3ms 周期で、
目標の回転数 / 現在の回転数
を算出し、0.9~1.1（目標に対し、10%以内の差異）であれば、duty（電流）の変更は行わず、目標との差分が 10%以上であれば、差分の 10%値を増減させます。（例えば、目標 10,000rpm で、現在 5,000rpm の場合、電流値を 2 倍(10,000/5,000)の 10%である 20%増加させるよう制御を行います。）

2.4. 安全機構

2.4.1. 過電流保護

133.34us の割り込みルーチン内で、過電流フラグ（*INT=L）が立っていた場合、変数(g_blsm_over_current_count)をインクリメントする。
1 秒毎に、変数の値を定数(BLSM_OVER_CURRENT_COUNT_THRESHOLD)値と比較し、値が定数を超えていた場合、過電流状態と判断して、モータを停止させる。
g_blsm_over_current_count 変数は、1.3 秒毎にクリアされる。
1.3 秒間の平均で過電流を判断している理由は、1 回の過電流フラグで過電流の判断を行った場合、モータの突入電流で保護機構が働いてしまうケースがあるためです。

2.4.2. 過熱保護

13.3ms の割り込みルーチン内で、温度をモニタ（A/D 変換及びテーブル比較）し、温度が定数(BLSM_TEMPARATURE_THRESHOLD)を超えていた場合、過熱状態と判断してモータを停止させる。

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	2016.8.29	—	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:http://www.hokutodenshi.co.jp

ルネサス エレクトロニクス RL78/G1F(QFP-64 ピン)搭載
ブラシレスモータスタータキット

ブラシレスモータスタータキット(RL78G1F)
[ソフトウェア編]
取扱説明書

株式会社 北斗電子
 2016 年 8 月 29 日改訂 REV.1.0.0.0 (160829)