HSBRX65-IO-BOARD
サンプルソフトウェア　マニュアル

ルネサス エレクトロニクス社　RX651/RX671(QFP-144 ピン)搭載 HSB シリーズマイコンボード向け I/O ボード

-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.1.2.0.0

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、アクセスランプの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。

製品の故障や、データ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。

概要

本書は、RX651 マイコンを搭載したマイコンボード(HSBRX651F144A)、及び RX671 マイコンを搭載したマイコンボード(HSBRX671F144)向けの、I/O ボード（スイッチや LED, LCD 等が搭載されており、マイコンのプログラムの動作をモニタするためのボード）である、HSBRX65-IO-BOARD を動作させるサンプルプログラムの説明を行っている資料です。

サンプルプログラムは、あくまで I/O ボードを動作させる一例です。サンプルプログラムで使用しているものとは、別なタイマ機能を用いたり、別な手法を用いてプログラミングを行ってみる事を推奨致します。

開発環境

本サンプルプログラムは、CS+向けのプロジェクトとなっていますので、予め

CS+forCC（Ver8.07 以降） [ルネサスエレクトロニクス製]

をダウンロードし、インストールを行ってください。本サンプルプログラムをビルドする目的であれば、無償版で十分です。（無償版のリンクサイズ制約が問題となる様な大規模なプログラムではありません）

本書では、CS+と連携して使用できる、スマート・コンフィグレータを使用しています。スマート・コンフィグレータは、自動的にコードを生成してくれる便利なツールですので、「RX スマート・コンフィグレータ（Ver2.12 以降）」も合わせてダウンロード、インストールを行ってください。

作成したプログラムをデバッグする際は、デバッガ

E1
E2
E2Lite
E20

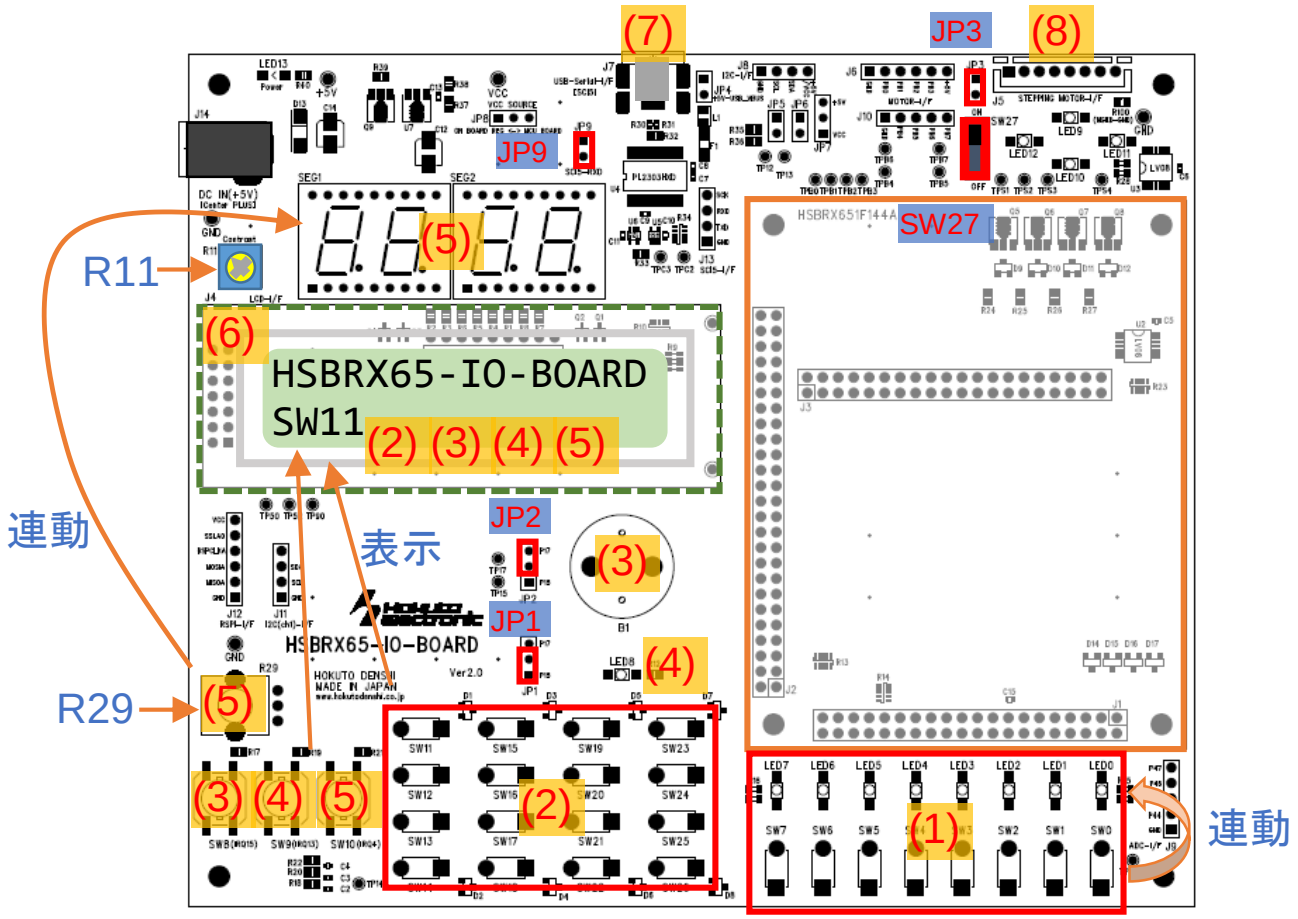
（いずれか）があると便利です。デバッガは、マイコンボードへのプログラムの書き込みと、デバッグ、両方の用途で使用可能です。

なお、デバッガをお持ちでなくても、マイコンボードへのプログラムの書き込みは、付属の USB ケーブル（USB A-miniB)で行う事ができます。USB ケーブルでプログラムの書き込みを行う際は、RenesasFlashProgrammer（Ver3 以降）が必要です。（基本的には、CS+のインストール時に、同時にインストールされます）

本書では、CS+以外の開発環境を使用する場合の説明はありませんが、e2studio やサードパーティ製の開発環境を使用する事も可能です。

サンプルプログラムの動作説明

本サンプルプログラムをマイコンボードに書き込んだ場合の動作に関して説明します。



(1)SW-LED 連動

ボード上のプッシュスイッチ SW0-7 を押すと、押している間、LED0-7 が点灯します。

（これは、SW0-7 と LED0-7 がボード上で接続されているわけではなく、マイコンボードで実行されているプログラムが、SW の読み取りと、LED の制御を行っています。）

(2)マトリックススイッチの読み取り

マトリックススイッチ SW11-26 を押すと、LCD 画面に押しているスイッチが表示されます。

※LCD 画面が薄い、または濃く塗りつぶされている場合は、ボード上の R11（水色の中に黄色の十字がある部品）を、精密ドライバ等で調整してください

※複数のキーを同時に押すと、「SW26*」の様に、末尾にアスタリスク(*)が表示されます（SWxx の xx は、押されているキーの中で番号の一番大きなものが表示されます）

(3)ブザー鳴動

※JP2 を 2-3 ショート（上側の 2 ピンにショートジャンパを挿す）にしてください

SW8 を押すと、ブザーが鳴ります。SW8 はトグル動作になっており、もう一度押すとブザーは止まります。
ブザーが鳴っている間、SW0-7 を押すと、ブザーの音程が変わります。

(4)LED 点滅

※JP1 を 1-2 ショート（下側の 2 ピンにショートジャンパを挿す）にしてください

SW9 を押すと、LED8 が点滅します。（1 秒周期）もう一度 SW9 を押すと、LED8 は消灯します（SW9 はトグル動作です）。
LED8 が点滅している間、R29 を回すと、点灯時間が変わります。

(5)7 セグメント LED

7 セグメント LED は、R29 の角度に応じた数値が表示されます（0~3FF）。
SW10 を押す度に、「表示数値が固定(HOLD)」「消灯(OFF)」「R29 に応じた値(ON)」の様に切り替わります。

(6)LCD

LCD 画面には、1 行目には
HSBRX65-IO-BOARD
の表示が出力されます。2 行目は、マトリックススイッチを押している間「SW11-26」の表示、及び SW8-10 を押した際「IRQ15,IRQ13, IRQ4」の表示（IRQ 表示は、他の情報が出力されるまで、表示は維持）が出力されます。

(7)USB-Serial

J7(USB-miniB)を PC と接続。PC 側で、

115,200bps, 8bit, パリティなし, ストップビット 1bit, フロー制御なし

で端末（Teraterm 等）を開くと、端末にメッセージが表示されます。

端末から、ブザーの鳴動を ON/OFF したり、7 セグメント LED のデモを実行したりする事が可能です。

※USB-Serial を使用する際は、PC 側に Prolific の USB-Serial のドライバがインストールされている必要があります（ドライバのインストールに関しては、取扱説明書参照）

※仮想 COM ポート番号（COM4 等）は、PC 環境によって異なりますので、USB ケーブルの挿抜で、PC 上に出現する COM ポート番号を選んで、端末を接続してください

※Ver2.0 以降のボードでは USB-Serial 変換使用時、JP9 をショートに設定してください（~Ver1.1 ボードではジャンパ設定は不要です）

(8)ステッピングモータ

※JP3 をショートに設定してください

J5 1-6 番ピンに、ステッピングモータ（推奨モータ：PF25-48C1[日本パルスモータ製]）を、接続し端末からステッピングモータを制御するコマンド("s" or "r")を入力すると、ステッピングモータを右回転または左回転させる事ができます。

ーUSB-Serial に接続した端末からのコマンドー

HSBRX671F144 ボード向けのサンプルプログラムでは
この部分の表示が HSBRX671F144 になります

ボードを起動すると、端末には下記のメッセージが表示されます。

```
HSBRX65-IO-BOARD[since Ver1.1](HSBRX651F144A) Sample program [Rev1.2].
Copyright (C) 2019-2022 HokutoDenshi. All Rights Reserved.
TEST MENU:
  b : buzzer(P17) [ON/OFF toggle]
  l : LED8(P15) [ON/OFF toggle]
  m : key matrix
  s : stepping motor[CCW]
  r : stepping motor[CW]
  7 : 7seg Demo
  h : help
>
```

・b コマンド（SW8 と等価）

キーボードから、b（1 文字、エンター不要）を入力すると、ブザー(P17)を鳴らします。もう一度 b コマンドを入力すると、ブザーは止まります。

```
>b
 buzzer (P17)>
 ->buzzer ON

>b
 buzzer (P17)>
 ->buzzer OFF
```

・l コマンド（SW9 と等価）

キーボードから、l コマンドを入力すると、LED8 が点滅します。もう一度 l コマンドを入力すると、LED8 は OFF となります。

```
>l
 LED8 (P15)>
 ->LED8 blink ON

>l
 LED8 (P15)>
 ->LED8 OFF
```

・m コマンド

キーボードから、m コマンドを入力すると、マトリックススイッチの状態を表示します。何かキーを押すと停止します。

```
>m
 -matrix sw (press any key to exit)-
SW11 SW12 SW13 SW14 SW15 SW16 SW17 SW18 SW19 SW20 SW21 SW22 SW23 SW24 SW25 SW26
  o    -    -    -    o    -    -    -    -    -    -    -    o    -    -    -
```

押している、キーが o で表示されます。(複数キーの同時押しも認識します)

※m コマンド実行中は、LCD 画面には、SW11-26 の情報が表示されません

・s コマンド

キーボードから、s コマンドを入力すると、ステッピングモータ端子(J5)から、ステッピングモータを左回りに回転させる信号が出力されます。何かキーを押すと停止します。

```
>s
 -stepping motor drive [CCW] (press any key to exit)-
```

・r コマンド

キーボードから、r コマンドを入力すると、ステッピングモータ端子(J5)から、ステッピングモータを右回りに回転させる信号が出力されます。何かキーを押すと停止します。

```
>r
 -stepping motor drive [CW] (press any key to exit)-
```

・7 コマンド

キーボードから、7 コマンドを入力すると、7 セグメント LED のデモ(全表示から、1111, 2222, …FFFF まで順次表示)が実行されます(デモの後は、7 セグメント LED が ON となります)。

```
>7
 7seg LED
 ->7seg led DEMO -> ON
```

・h コマンド

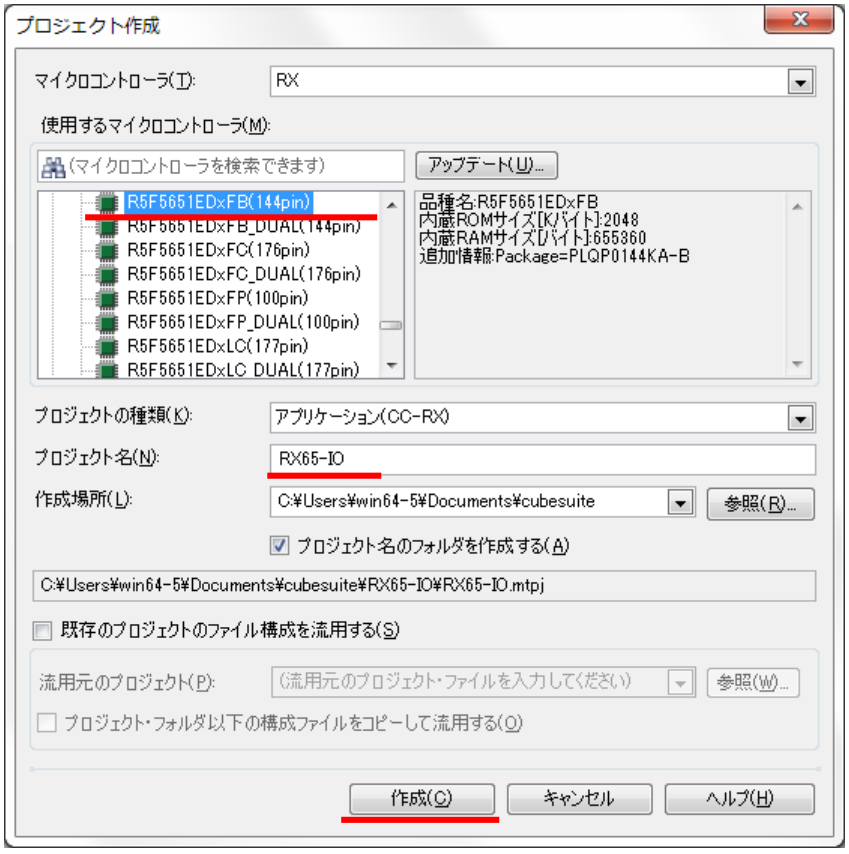
キーボードから、h コマンドを入力すると、ヘルプの画面が出力されます。

```
>h
  b : buzzer(P17) [ON/OFF toggle]
  l : LED8(P15) [ON/OFF toggle]
  m : key matrix
  s : stepping motor[CCW]
  r : stepping motor[CW]
  7 : 7seg Demo
  h : help
```

スマート・コンフィグレータを使用した初期設定

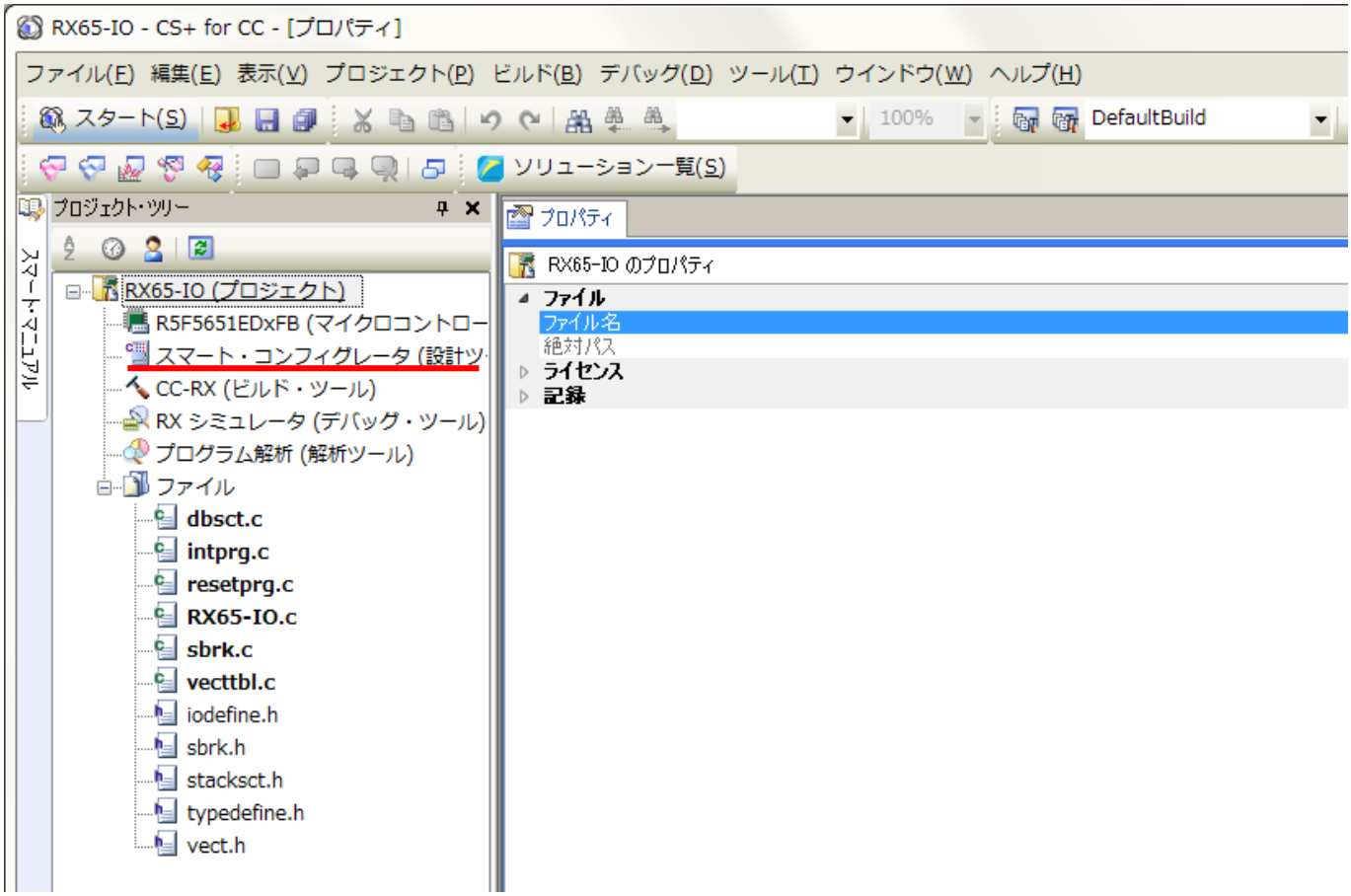
CS+を使用して、本ボード向けのプログラムを作成する際は、スマート・コンフィグレータを使用すると、クロックの初期設定や割り込みの設定等のコードを自動生成してくれますので、手間の掛かる初期設定のコードを書く手間が省けます。

CS+を起動し、
ファイルー新規作成ー新しいプロジェクトを作成

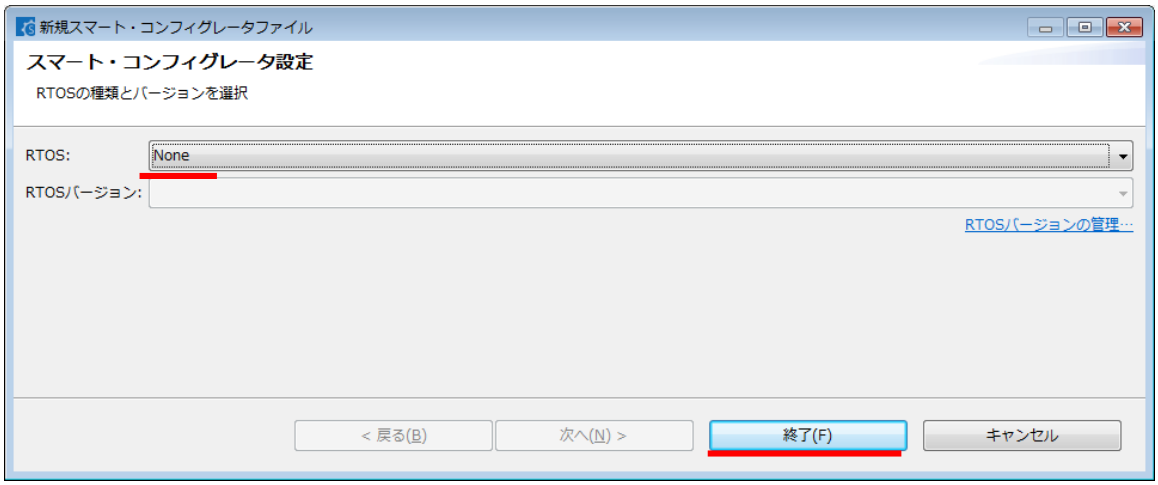


使用するマイクロコントローラ： R5F5651EDxFB …HSBRX651F144A の場合
R5F5671EHxFB …HSBRX671F144 の場合
プロジェクト名： 任意の名称

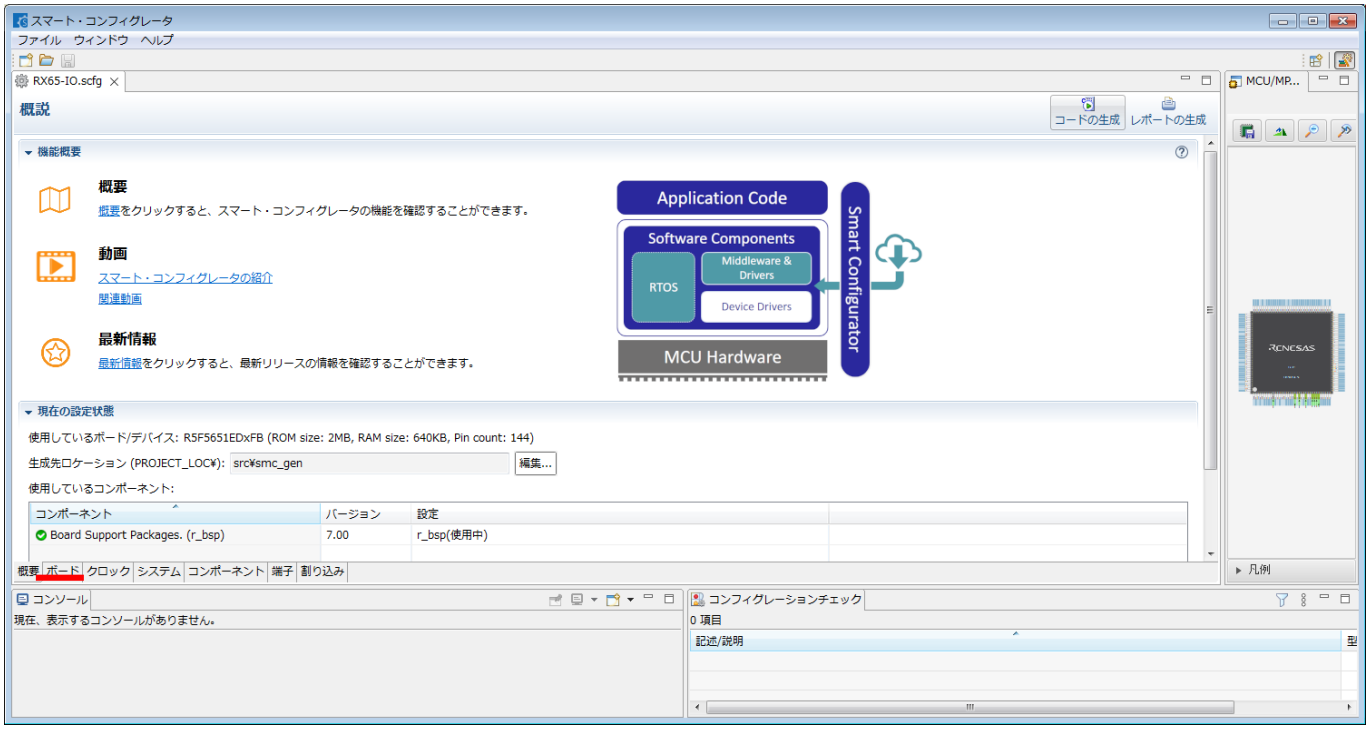
「作成」ボタンを押す。



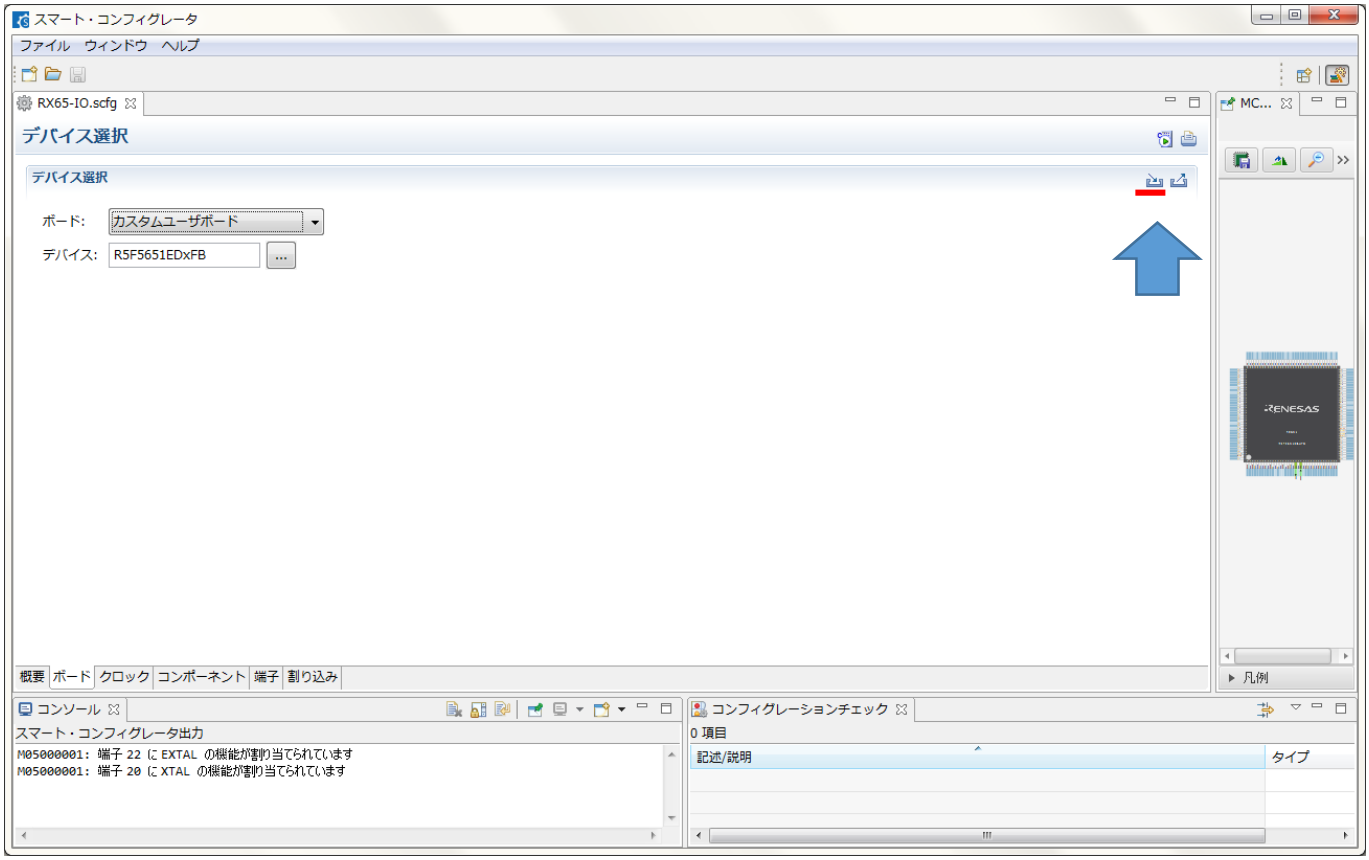
「スマート・コンフィグレータ（設計ツール）」をダブルクリックで起動。



RTOS は None を選択して「終了」を押す。



「ボード」タブを選択。



右上の「読み込み」ボタンを押す。

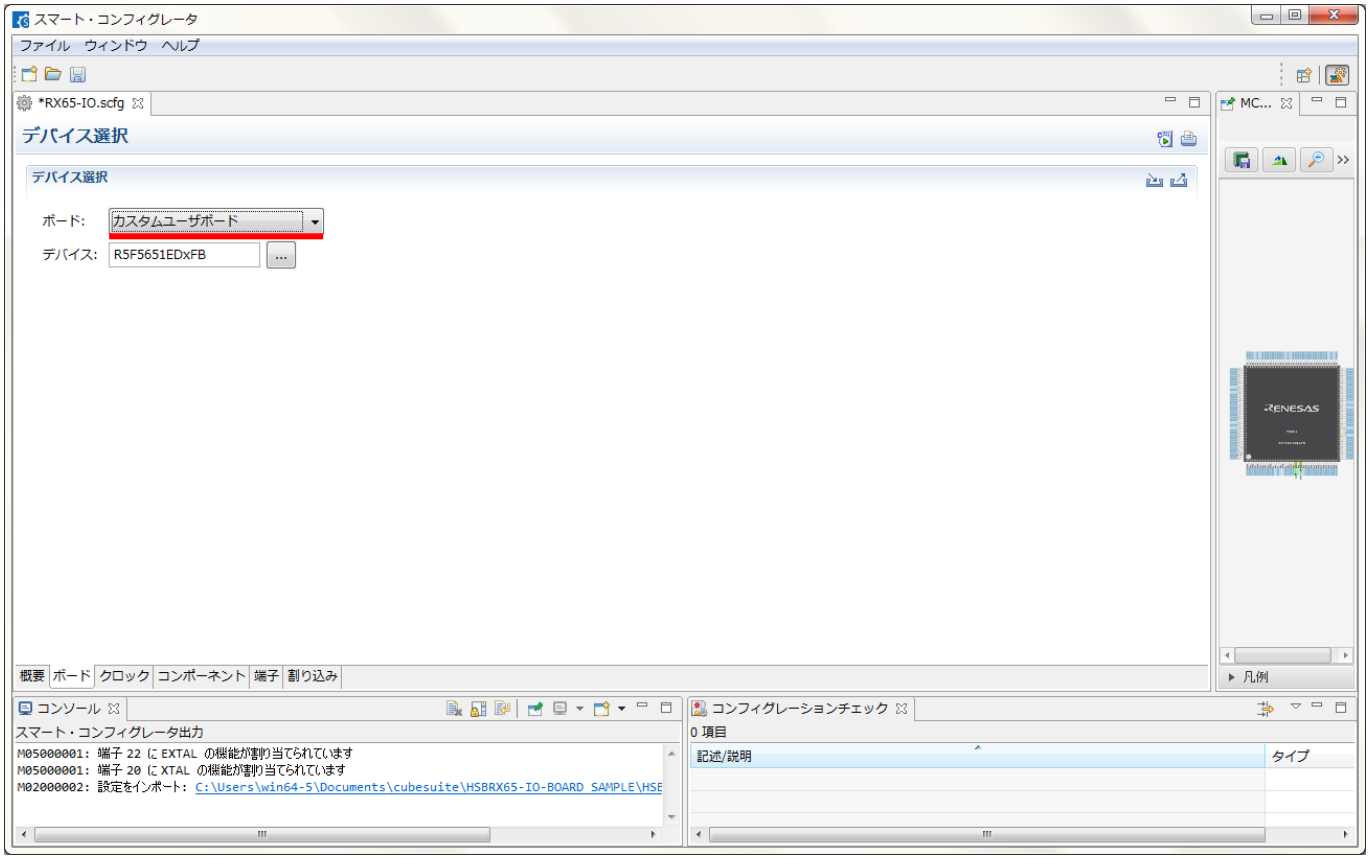
サンプルプロジェクトフォルダ(HSBRX65-IO-BOARD_SAMPLE/HSBRX65-IO-BOARD_RX671)内の、.bdf ファイルを指定して読み込みます

読み込むファイル：

HSBRX65-IO-BOARD_SAMPLE¥HSBRX651F144A_IO-BOARD_V1.00.bdf …HSBRX651F144A の場合

HSBRX65-IO-BOARD_SAMPLE_RX671¥HSBRX671F144_IO-BOARD_V1.00.bdf …HSBRX671F144 の場合

※V の後のバージョン番号は変わる可能性があります

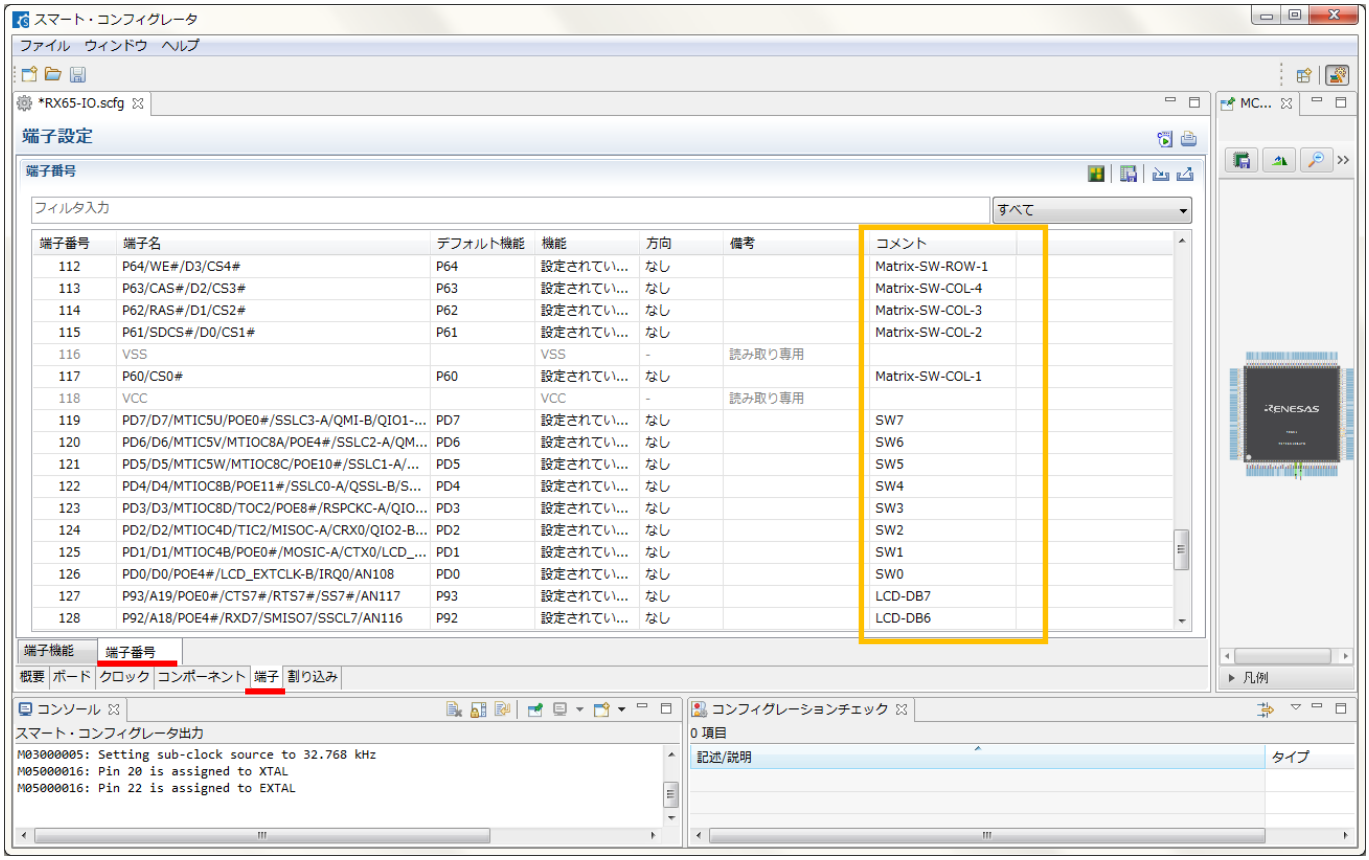


ボード欄、「HSBRX651F144A_IO-BOARD_V1.00」を選択。…HSBRX651F144A の場合

「HSBRX671F144_IO-BOARD_V1.00」を選択。…HSBRX671F144 の場合

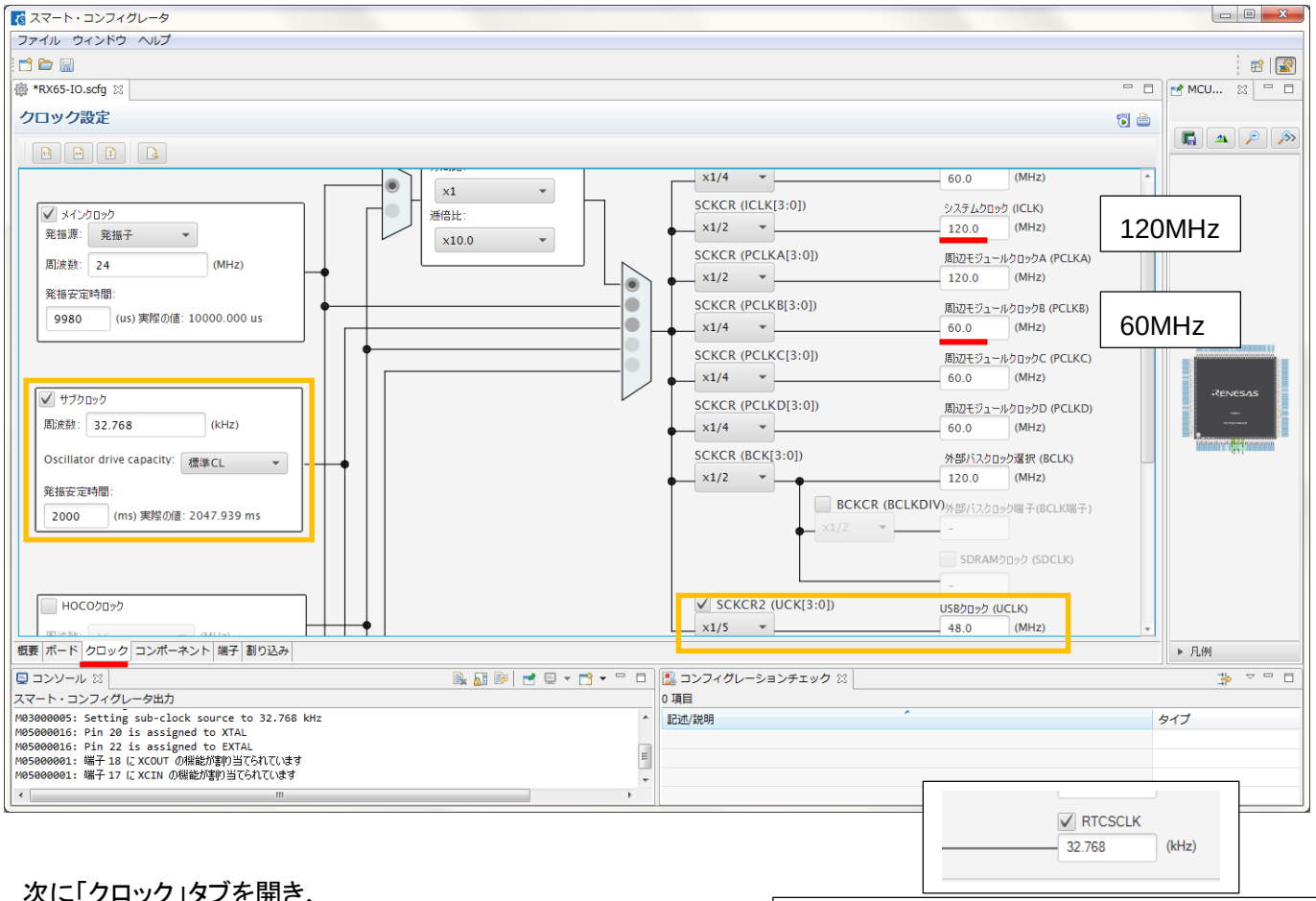
（V の後のバージョン番号は変わる可能性があります）

確認画面がでますので、「続ける」（もしくは「保存して続ける」）を押してください。



端子タブ－端子番号タブ

を選択すると、コメントの欄に、どの端子がどの信号に接続されているかのコメントがありますので、端子機能の割り当てを行った際等、確認の目安としてください。



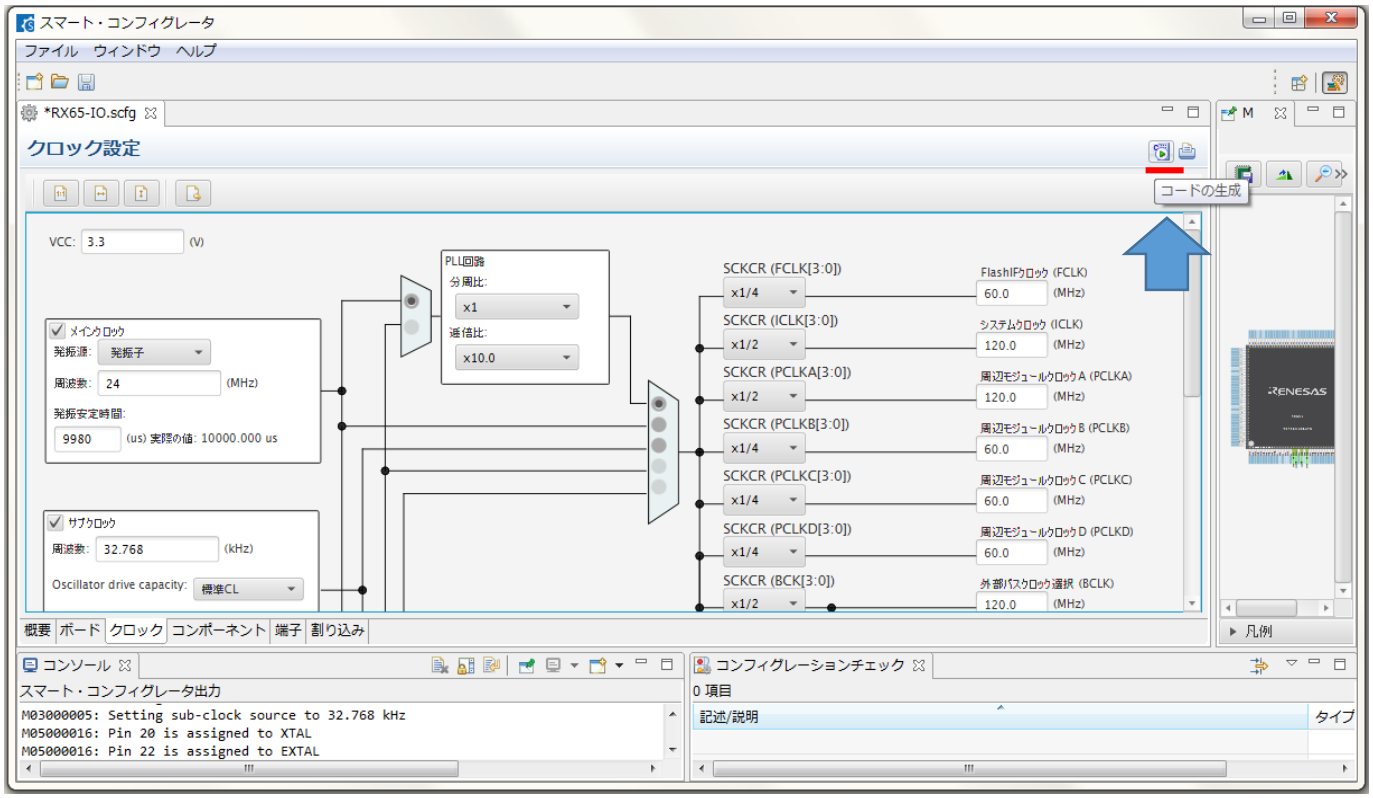
次に「クロック」タブを開き、
システムクロック：120MHz
周辺モジュールクロック：60MHz
になっている事を確認してください。（基本的には初期設定のまま変更不要です）

※本サンプルでは未使用ですがリアルタイムクロック（時計機能）を使用する際は RTCSCLK にチェックを入れてください

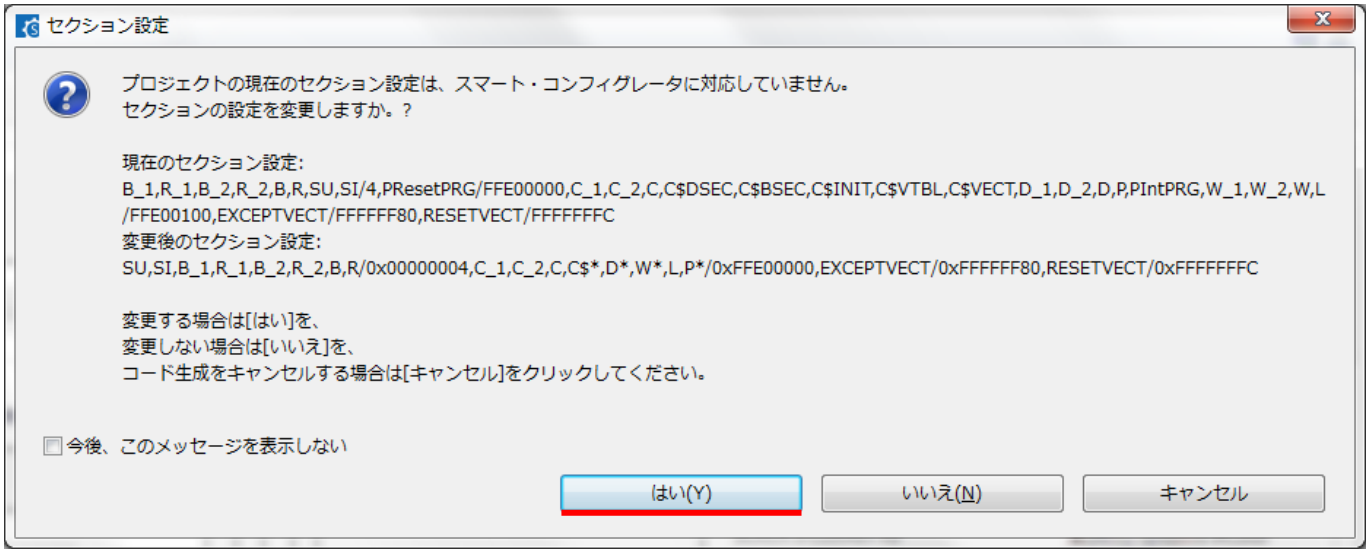
必要に応じて、以下設定してください。

サブクロック（チェックを入れる）
※マイコンボードにはサブクロックが実装されています、プログラムで RTC の機能を使用しないときはチェックは必要ありません

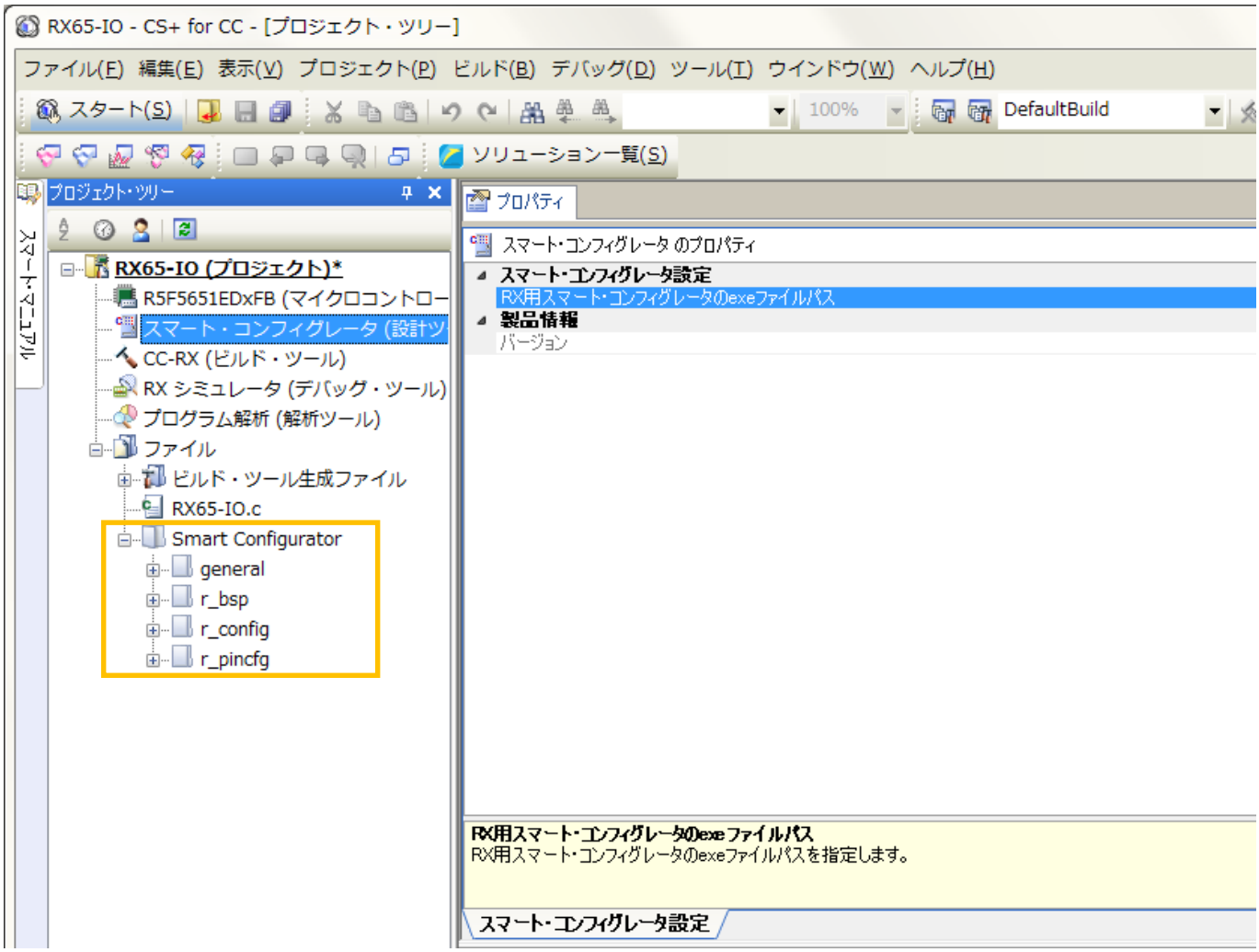
SCKCR2（チェックを入れる）
※マイコンボードには USB 関連の回路が実装されています、プログラムでマイコンボード側の USB（ホスト、ファンクション）の機能を使用しないときはチェックは必要ありません
（USB クロック使用時は、48MHz になるよう設定してください）



一通り設定が終わったら、右上の「コードの生成」ボタンを押してください。

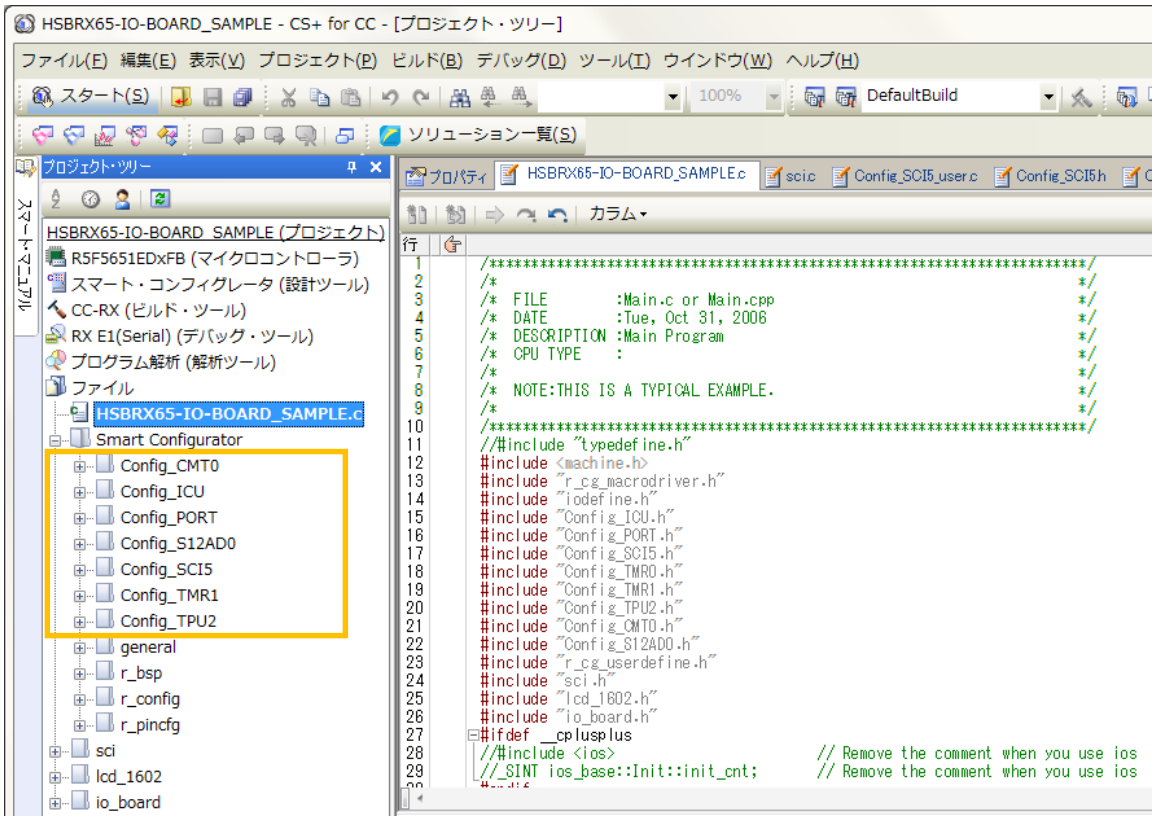


セクションの変更のメッセージが表示された際は、「はい」を選択してください。

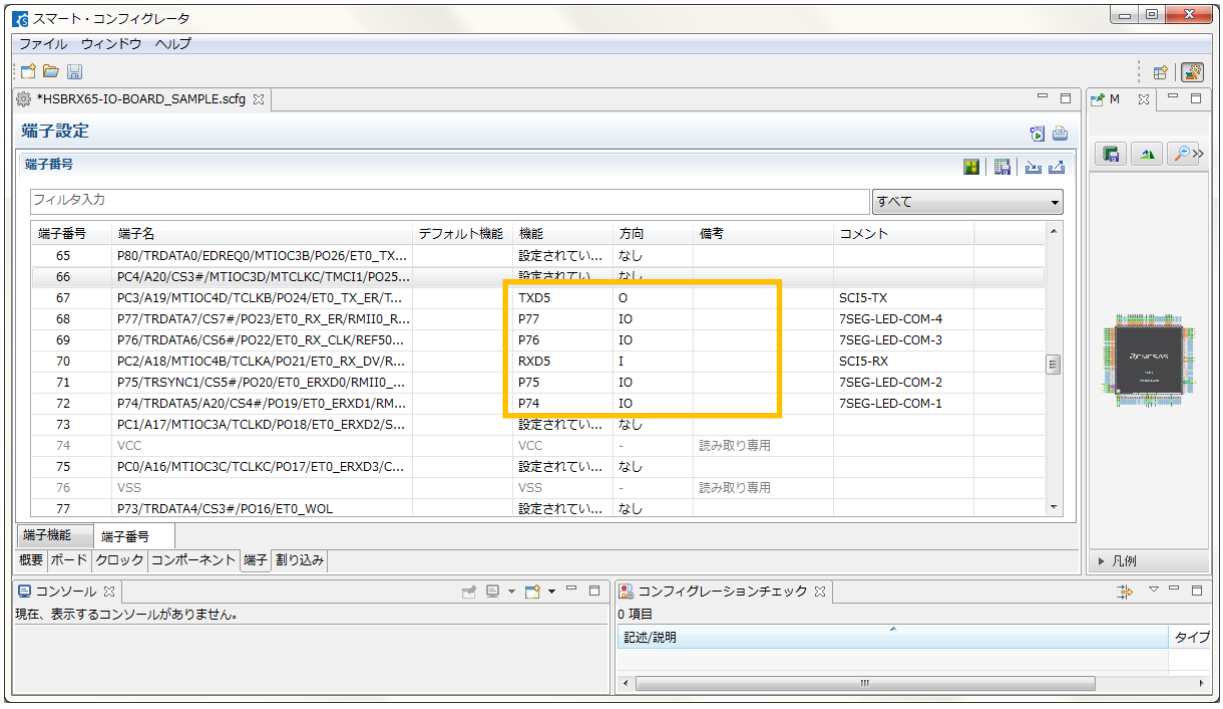


上記枠内が、スマート・コンフィグレータが生成したソースコードです。（この時点では、クロックの設定程度しか入っていませんが、スマート・コンフィグレータでコンポーネントを追加していくと、この部分が増えていきます）

[参考]



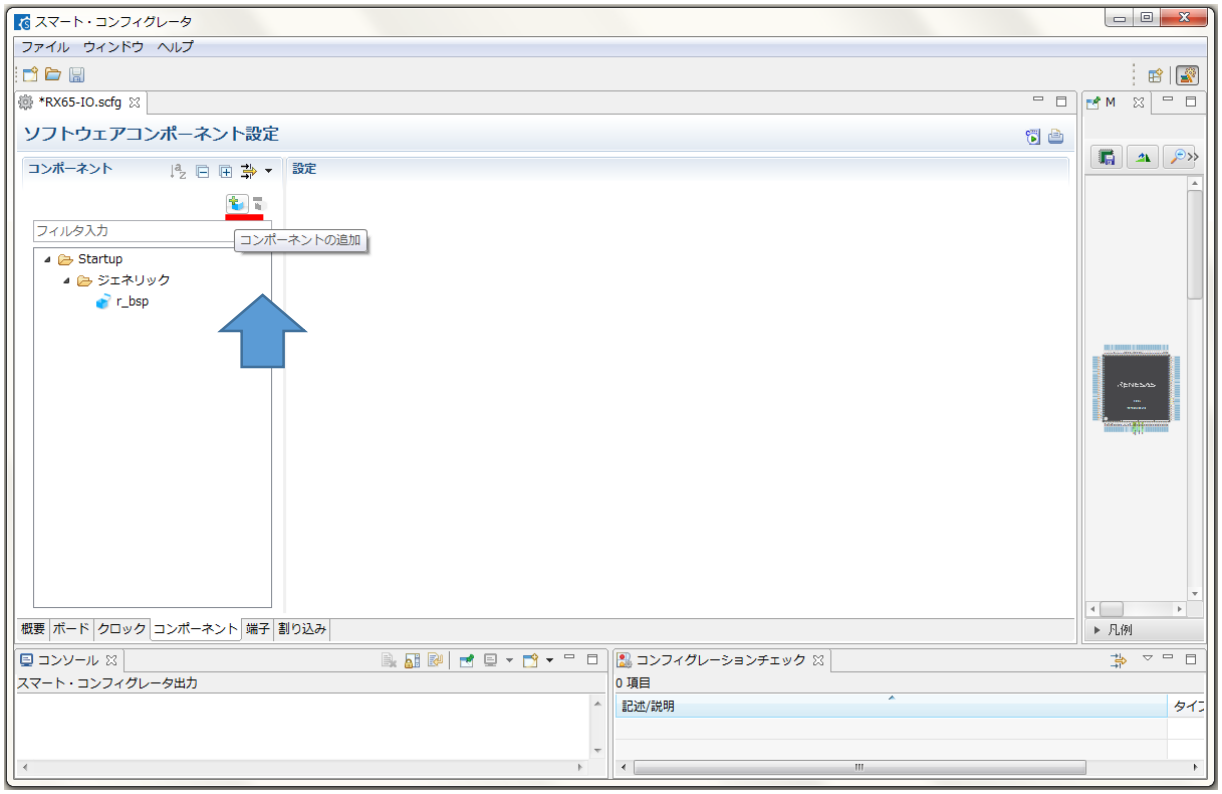
スマート・コンフィグレータでコンポーネントを追加すると、上記の枠内にソースコードが追加されます。



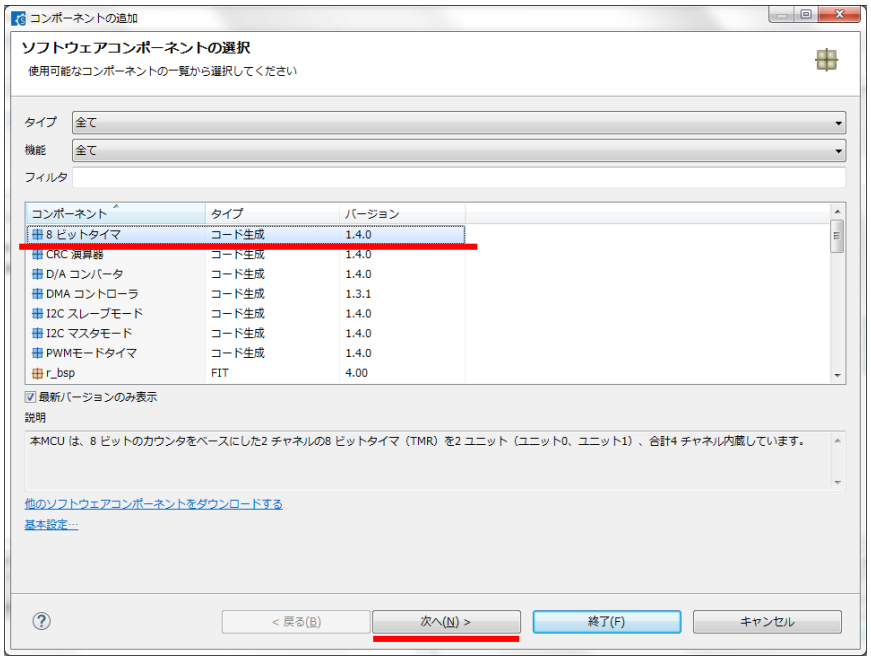
また、端子の「機能」「方向」の欄に選択している機能が反映されます。

－スマート・コンフィグレータによるブザー向けタイマ TMR1 の設定に関して－

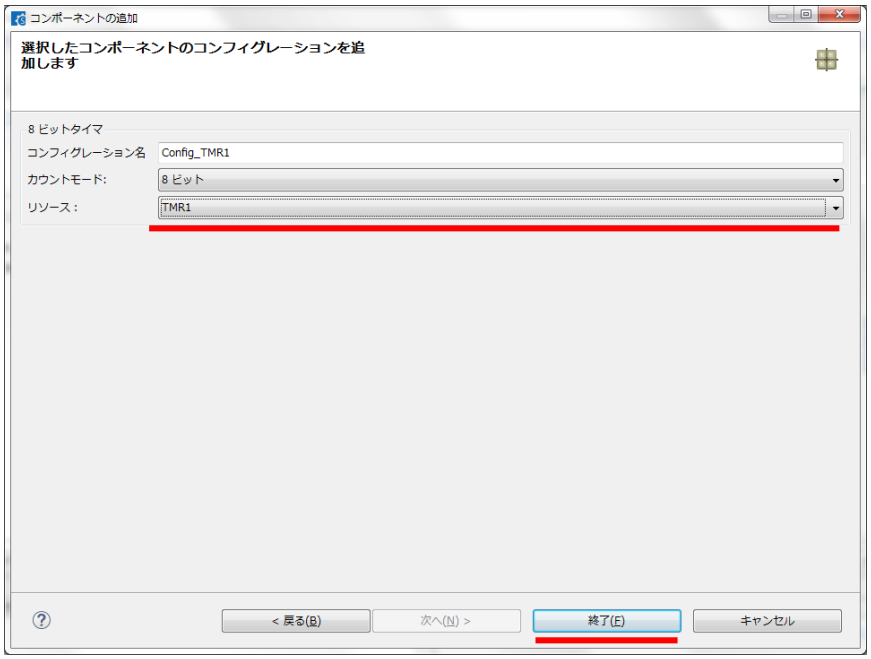
スマート・コンフィグレータを使用して、ブザーを鳴らすためのタイマ(TMR1)を設定する例を示します。



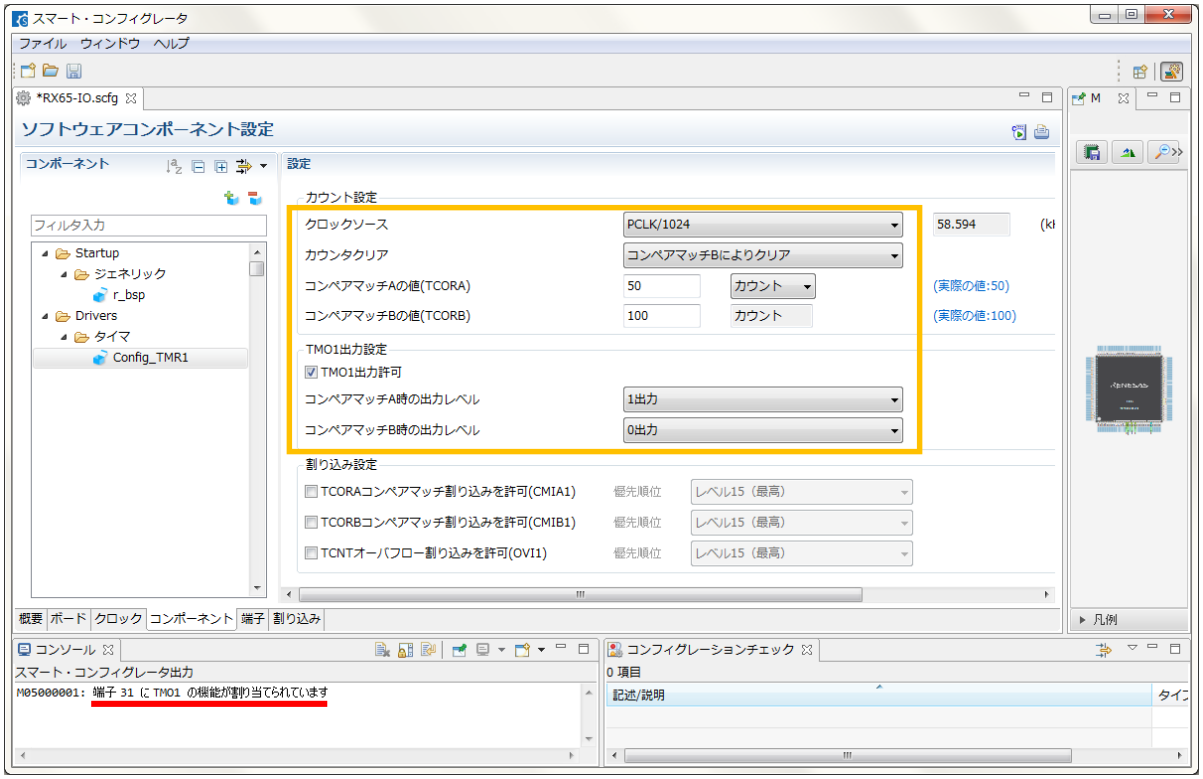
「コンポーネントの追加」を選択。



「8 ビットタイマ」を選択して、「次へ」。



「TMR1」を選択して、「終了」。



クロックソース「PCLK/1024」

カウンタクリア「コンペアマッチ B によるクリア」

コンペアマッチ A の値「50」カウント　※仮値

コンペアマッチ B の値「100」カウント　※仮値

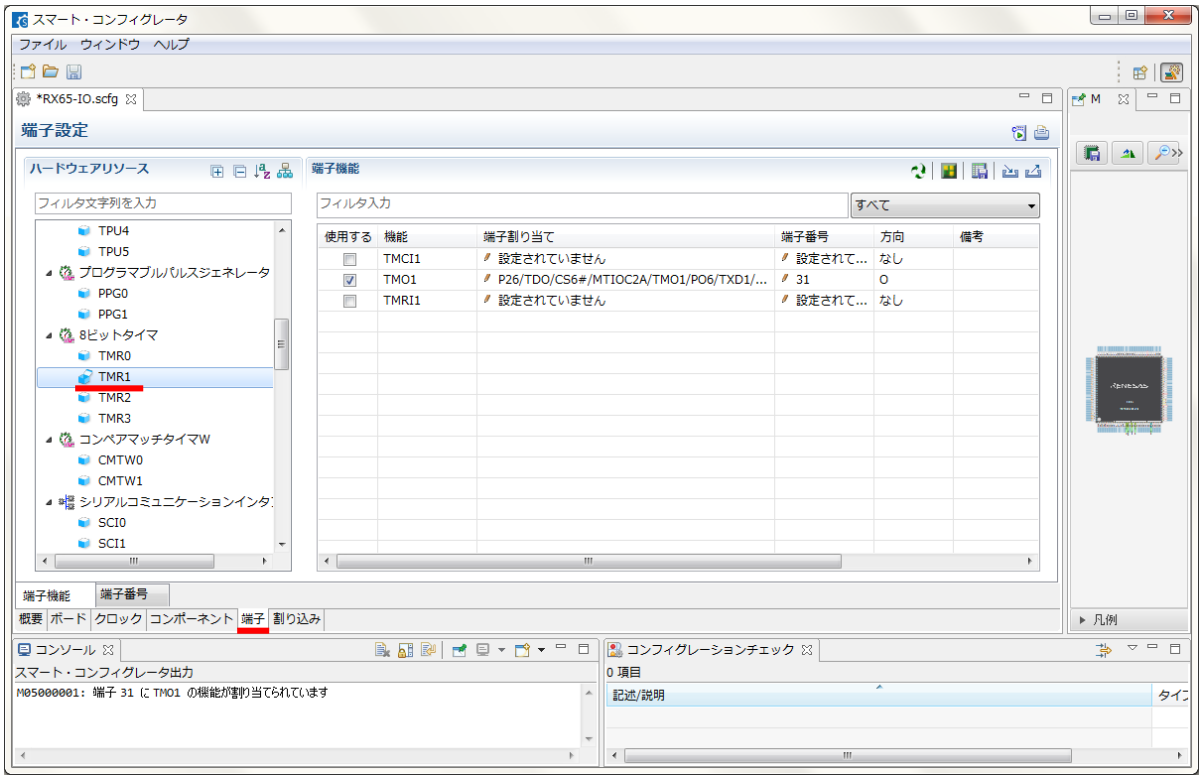
TMO1 出力許可

　コンペマッチ A の出力レベル「1 出力」

　コンペマッチ B の出力レベル「0 出力」

端子 31 に TMO1 の機能が割り当てられています

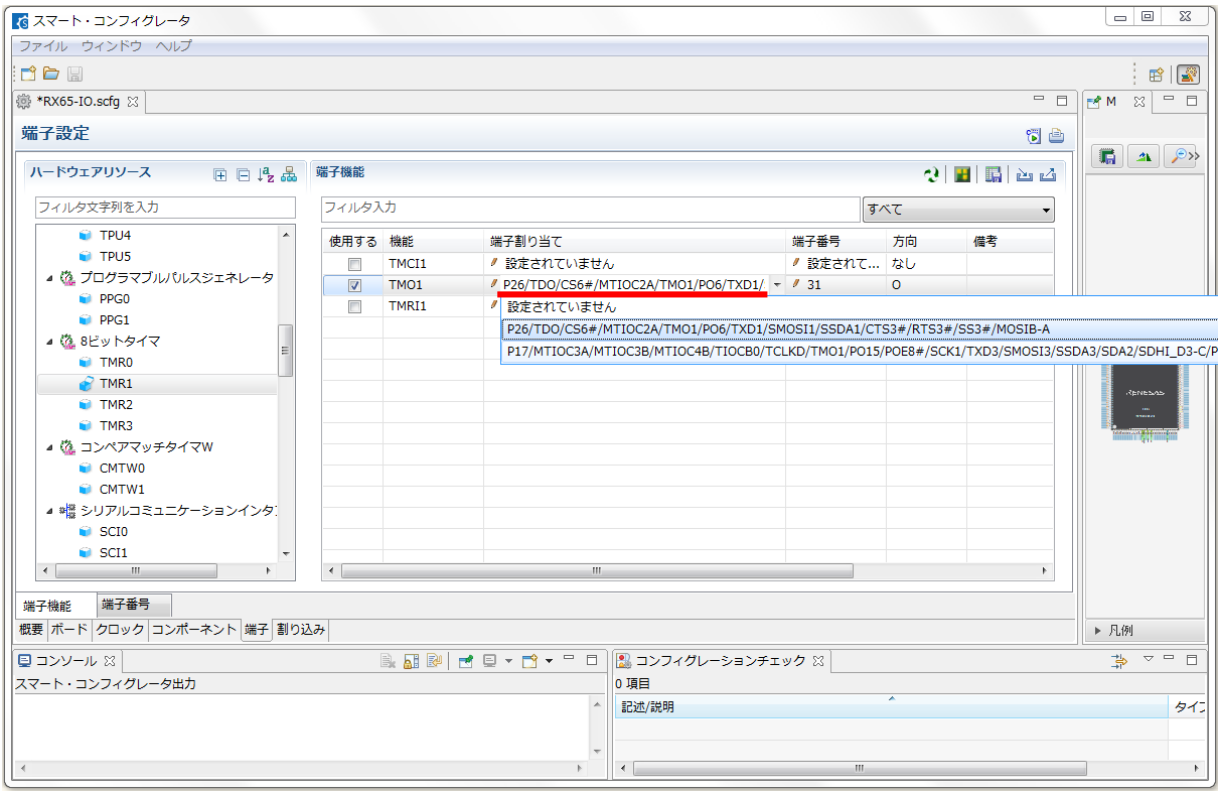
→31 番ピンは、P26 です。ここでは、P17 に、TMO1 を割り当てたいので、変更します。



「端子」タブを選択

「TMR1」の設定を開く

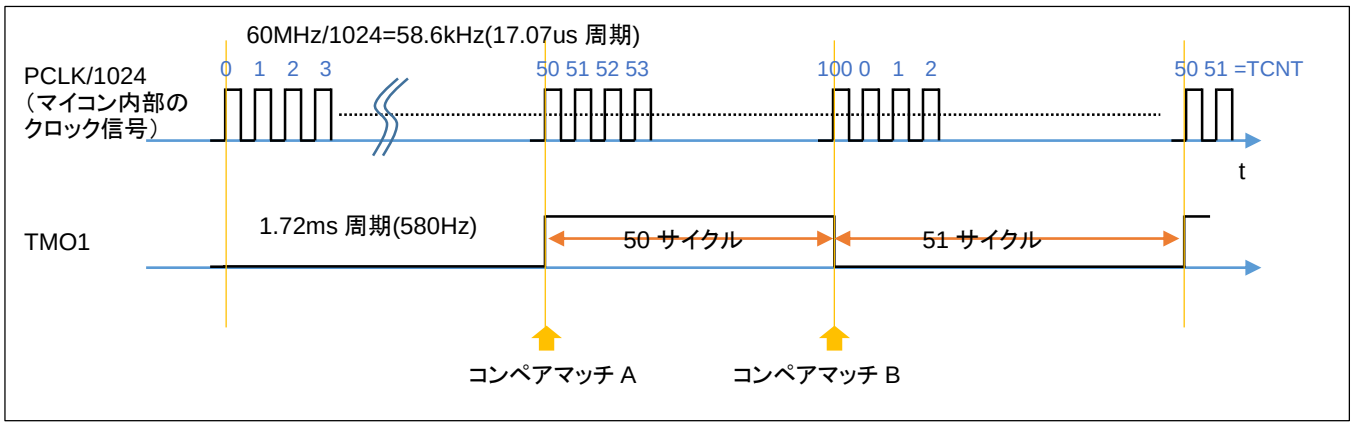
TMO1 の端子割り当て



TMO1 の端子割り当ての欄をクリックして、P17 側に変更する。

以上で、スマート・コンフィグレータを使用して、P17 に TMO1 を割り当てる設定は完了です。

（スマート・コンフィグレータ上の「コード生成」を行うと、TMO1 を P17 から出力するコードが生成されます）



コンペアマッチ A の値(TCORA)を 50, コンペアマッチ A の値(TCORB)を 100 に設定した場合、TMO1(P17)からは、上記のような波形が出力されます。

※デューティ比を 50%に設定したい場合は、TCORA=n, TCORB=2n-1 に設定してください

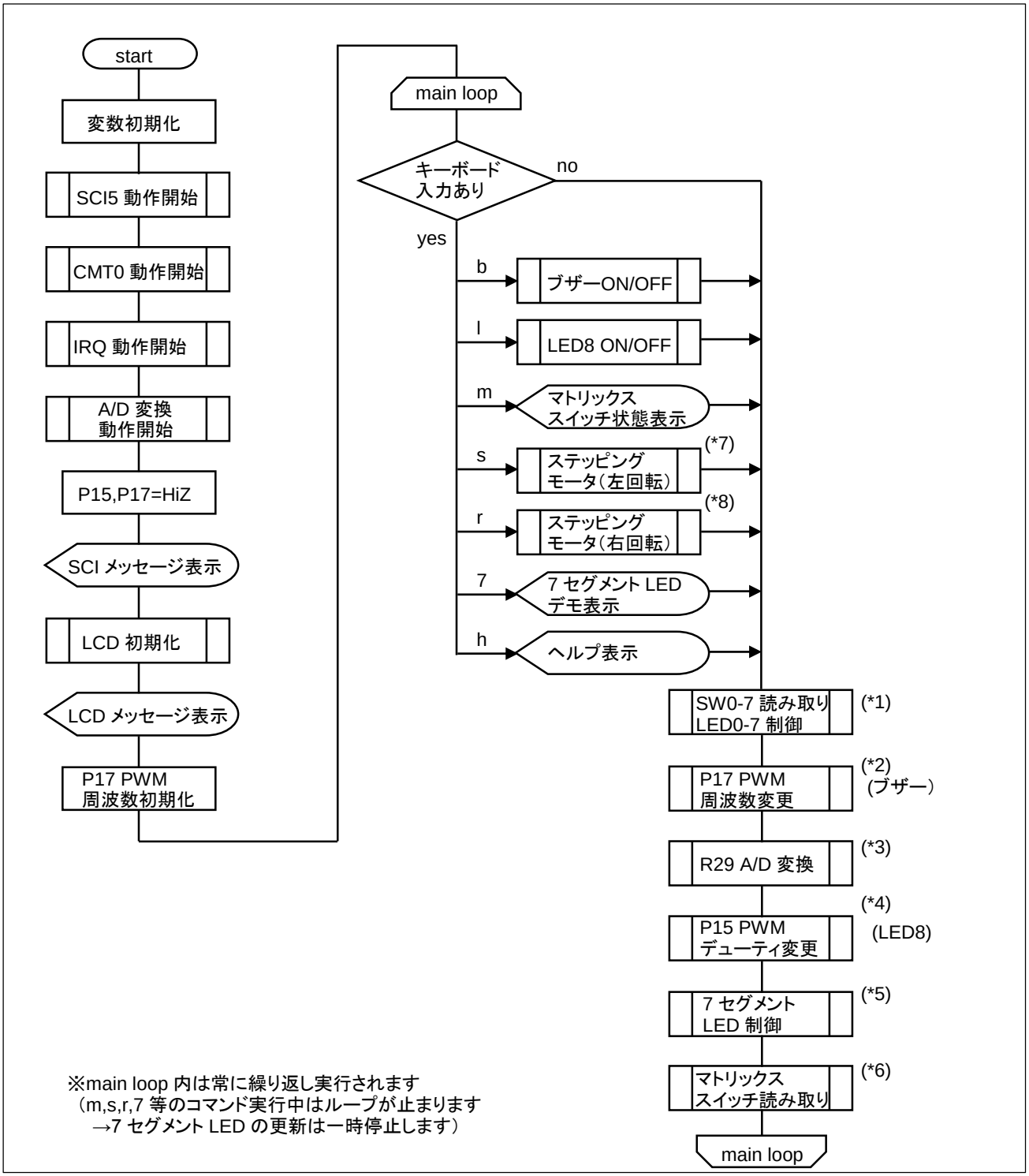
1. サンプルプログラム説明(V1.2)

1.1. メイン関数

サンプルプログラムのメイン関数の処理内容を以下に示します。

ーフローチャートー

メイン関数 main()



メイン関数は各種初期化後、
・キーボードからの指令読み取り
・プッシュスイッチ SW0-7 の読み取り→LED0-7 の ON/OFF 制御
・P17（ブザー）の周波数を SW0-7 の押したキーに応じて変更（ブザー鳴動時のみ)
・R29（ボリューム)の回転角度を A/D 変換
（7 セグメント LED の表示値、及び P15 のデューティ比に結果を使用）
・P15(LED8)のデューティ比をボリュームの回転角度に連動（LED8 点滅時のみ）
・7 セグメント LED の表示値をボリュームの回転角度に連動（7 セグメント LED ON, HOLD 無効時）
・マトリックススイッチ読み取り、LCD 画面に表示
上記の処理を無限ループします。

メイン関数とは別に
・タイマ(CMT0)による定期的な割り込み処理（100us 毎）
・SW8~SW10 押下による割り込み処理
が実行されます。

main loop は、処理内容により実行時間は前後しますが、最短で 6.5us 程度で 1 ループします。
（LCD 出力の処理が入ると、実行時間は 200us 程度掛かります）

1.1.1. SW0-7 読み取り LED0-7 制御(*1)

SW0-7 の状態を読み取り、LED0-7 を ON/OFF します。

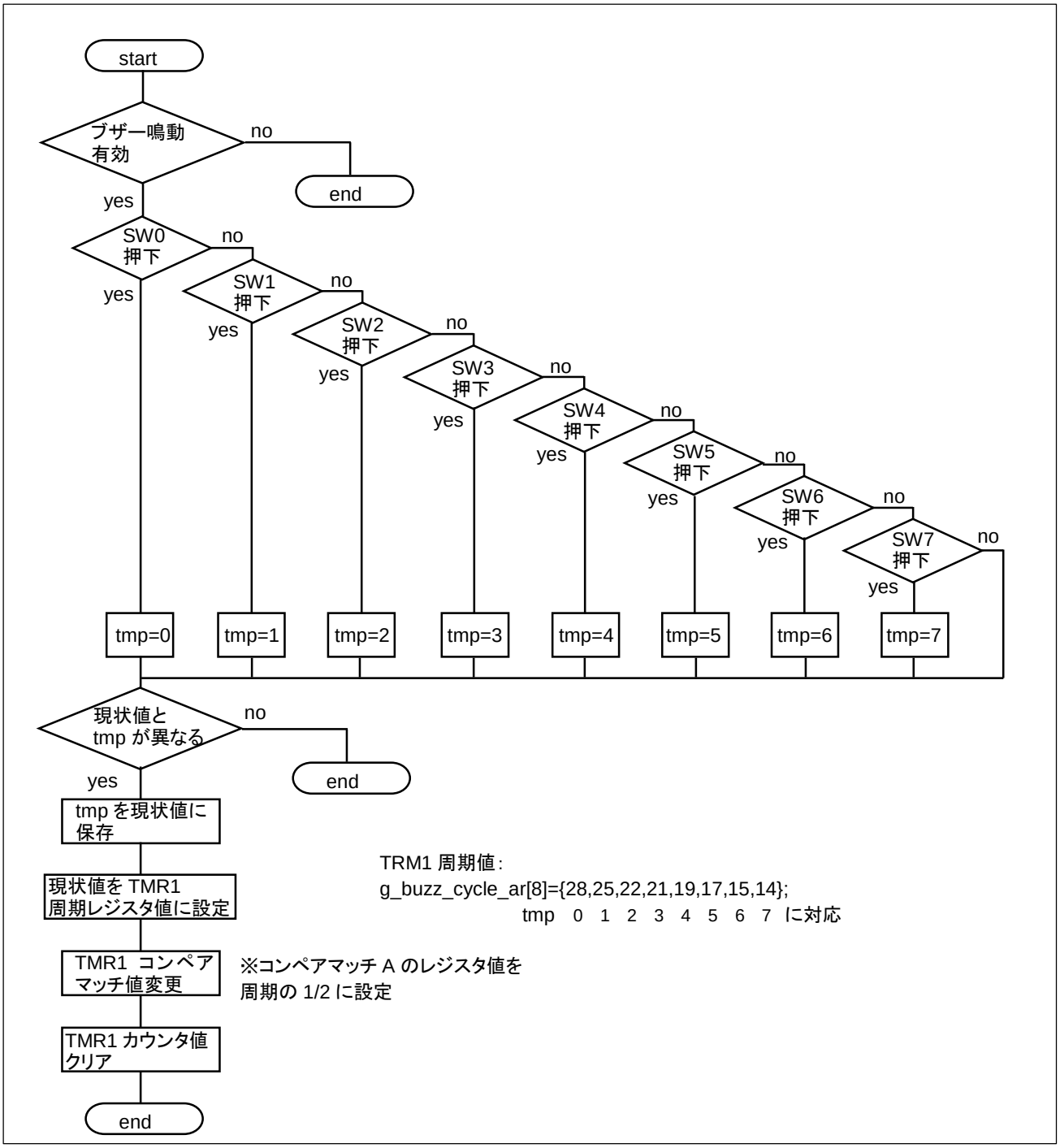
SW0-7 の端子状態を示すレジスタ(PORTD.PIDR)値を、LED0-7 の出力ポート(PORTE.PODR)にコピーしています。プログラムのコードとしては、1 行のみです。

※マイコンのプログラムは始めてという方は、この部分の処理の変更を行ってみると、ポートの入出力の手法という、マイコンの制御で最も単純な制御に関しての理解が深まるかと思います

※SW の押している個数を LED の点灯状態に反映させるサンプルが、オリジナルのソースにコメントアウトされて記載されています

1.1.2. ブザーの周波数変更(*2)

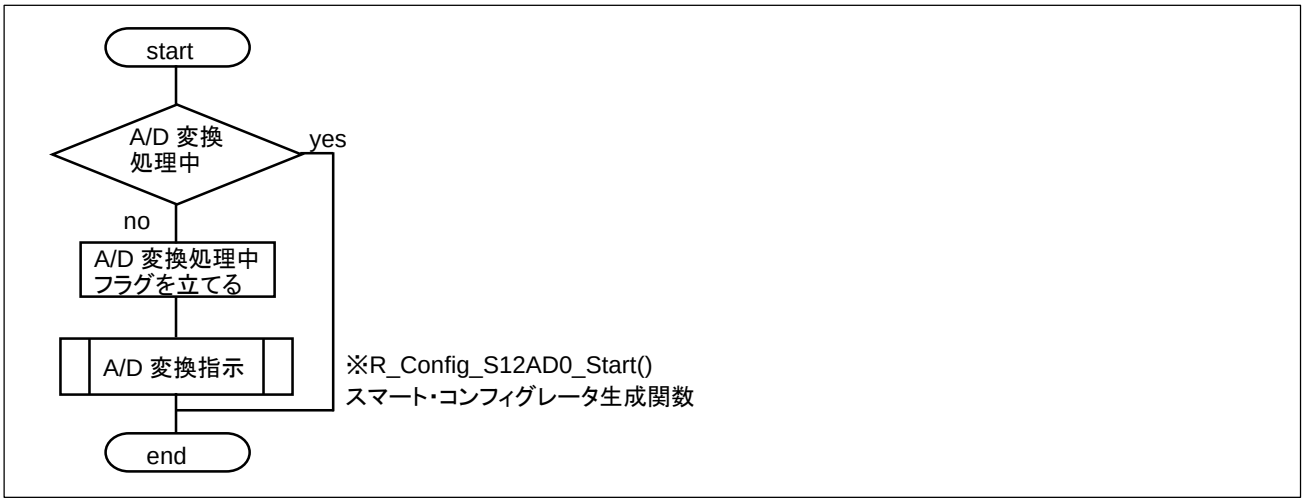
SW0-7 の状態を読み取り、ブザーの周波数（音程）を変化させます。



ブザー鳴動時に、SW0-7 を押すと、ブザーの音程が変化します。音程に対応する周期は、予め配列変数に設定してあり、押したスイッチによって選ぶようになっています。
（音程を変えたい場合は、g_buzz_cycle_ar の定義値を変えてみてください。指定可能なのは~255 以内です。周期の値なので、値が小さいほど高音で、28 で 2kHz 位です。）

1.1.3. A/D 変換(*3)

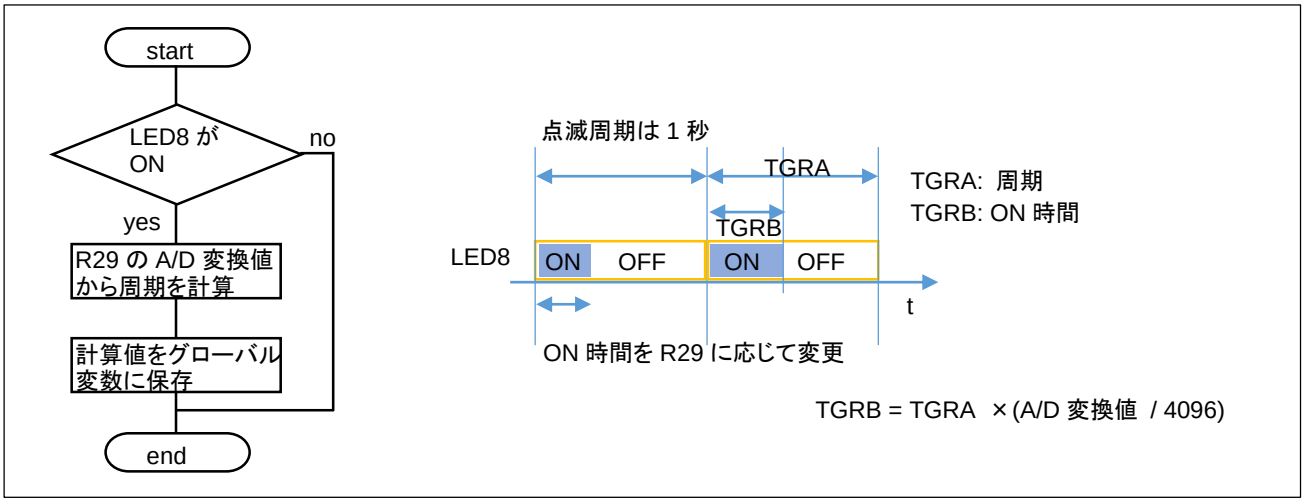
R29 の回転角度を、A/D 変換で読み取ります。



プログラム内では、A/D 変換処理中のフラグを確認し、A/D 変換実行中でなければ、A/D 変換を行うよう指示を出します。A/D 変換処理中を示すフラグのクリアと、A/D 変換結果の回収は、A/D 変換の処理が終了した時に呼び出される、割り込み処理内で行っています。

1.1.4. LED8　PWM デューティ変更(*4)

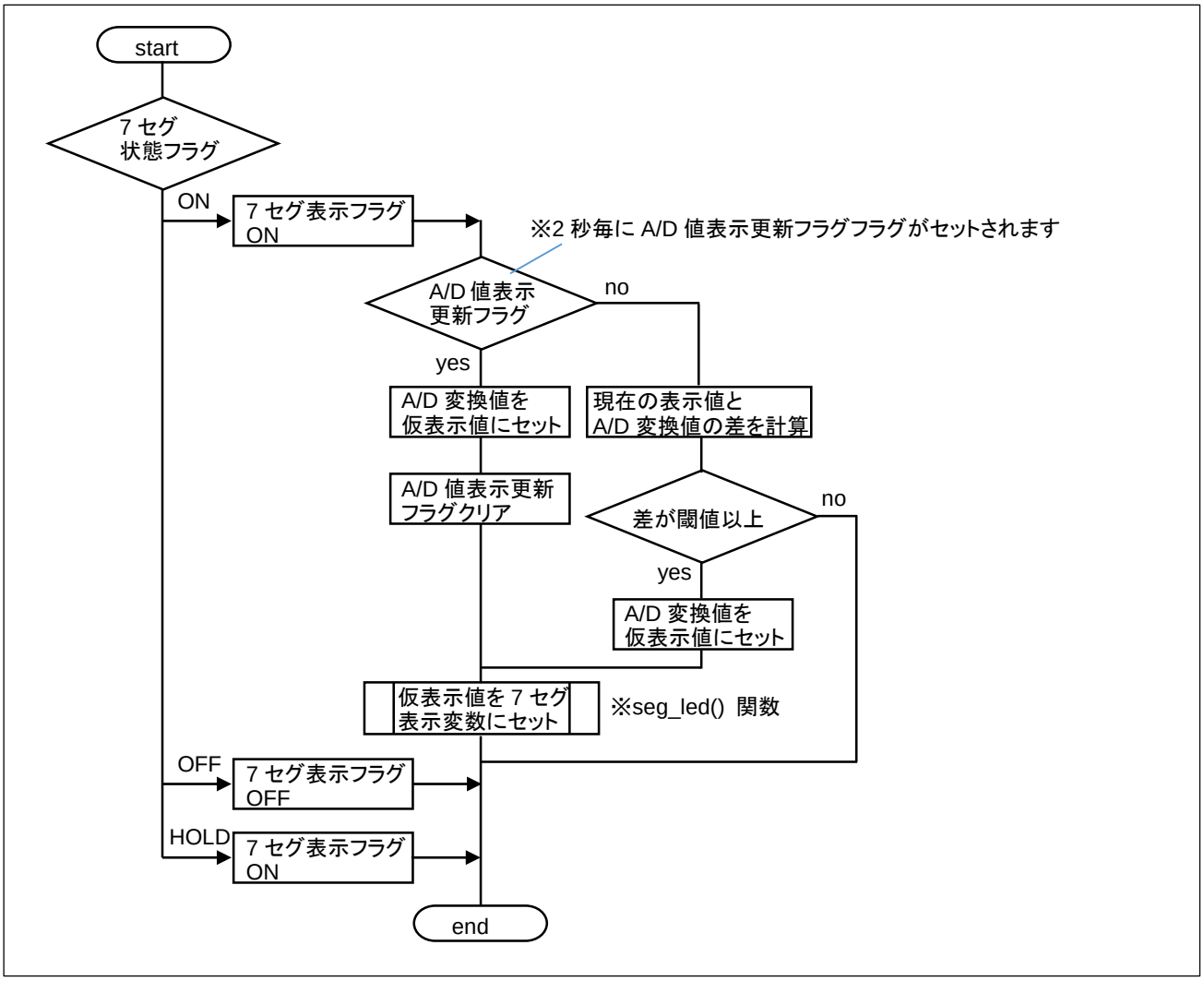
LED8 点滅中は、R29 の回転角度に応じて点灯時間が変化します。



LED8 が点灯している時間を、R29 の A/D 変換値(0~4095)に応じた値に設定しています。
（LED8 は、TPU2 タイマを使用しており、周期レジスタを TGRA, ON 時間を TGRB で設定しています）
※TGRB レジスタの設定は、決められたタイミング（カウンタ値リセットされるタイミング）で行っています
（詳細は後述）

1.1.5. 7 セグメント LED 制御(*5)

7 セグメント LED の表示値を設定します。



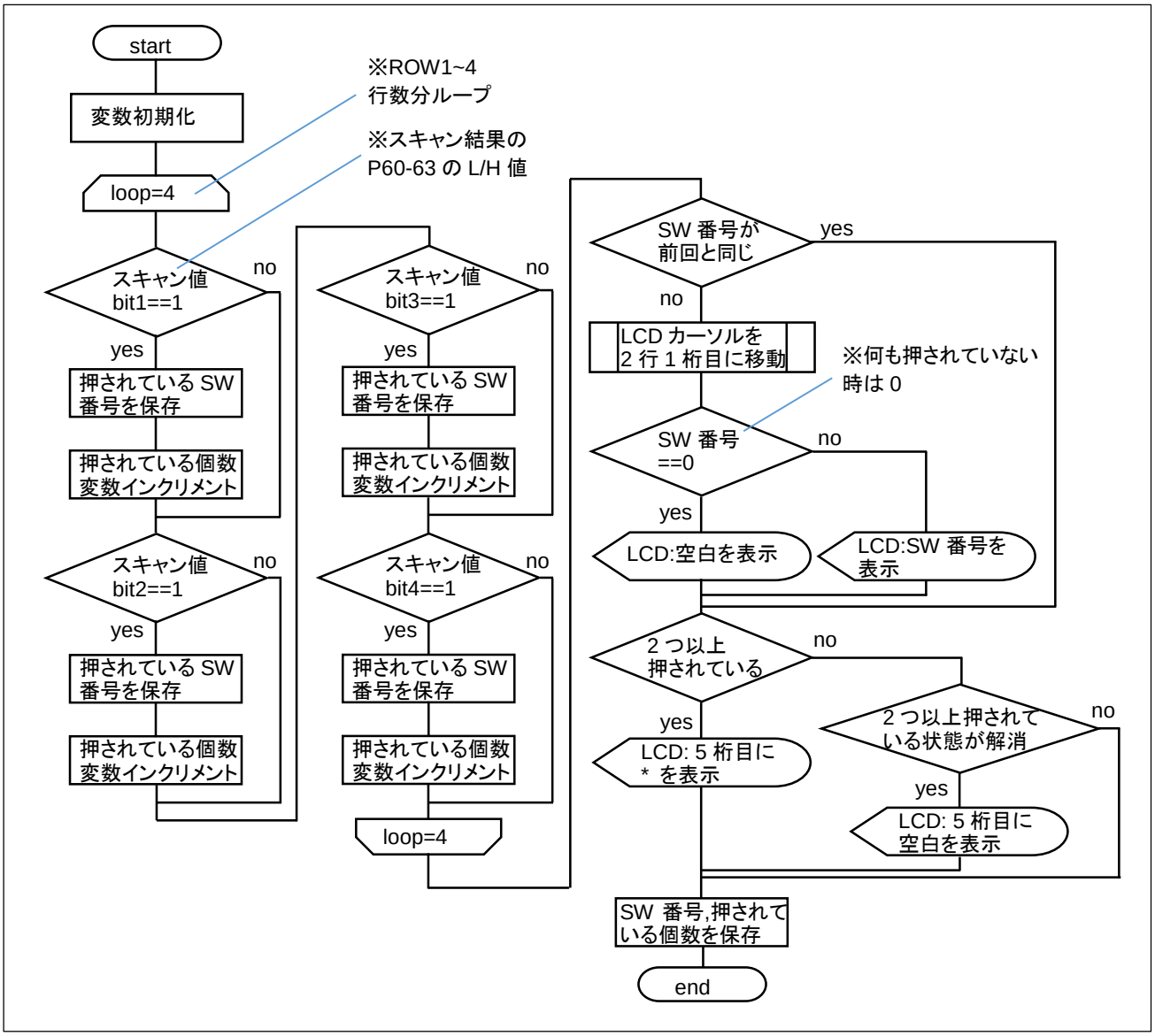
7 セグメント LED の点灯制御は、100us 毎に定期的に実行されるコンペアマッチタイマ(CMT0)内で行われています。メイン関数内の本処理の目的は、「7 セグ表示変数」に表示値を設定する事です。

A/D 変換値=表示値とすると、A/D 変換値の微妙な変動(0x123→0x124→0x123→…等)が、表示値に反映され、値が読みにくいので、現在の表示値と一定値(プログラム内の設定値は 3)以上変化した時のみ、表示値を更新する事としています。なお、この様な処理を行うと、0x002→0x000 の様な変更が永遠に反映されないので、強制的に一定時間(設定値は 2 秒)毎に表示更新を行っています。

7 セグメント LED の状態フラグが、OFF の時は表示フラグをクリア。HOLD の時は値の更新処理は実行せず、表示フラグのセットのみ行っています。

1.1.6. マトリックススイッチ(*6)

マトリックススイッチのスキャン結果を処理します。

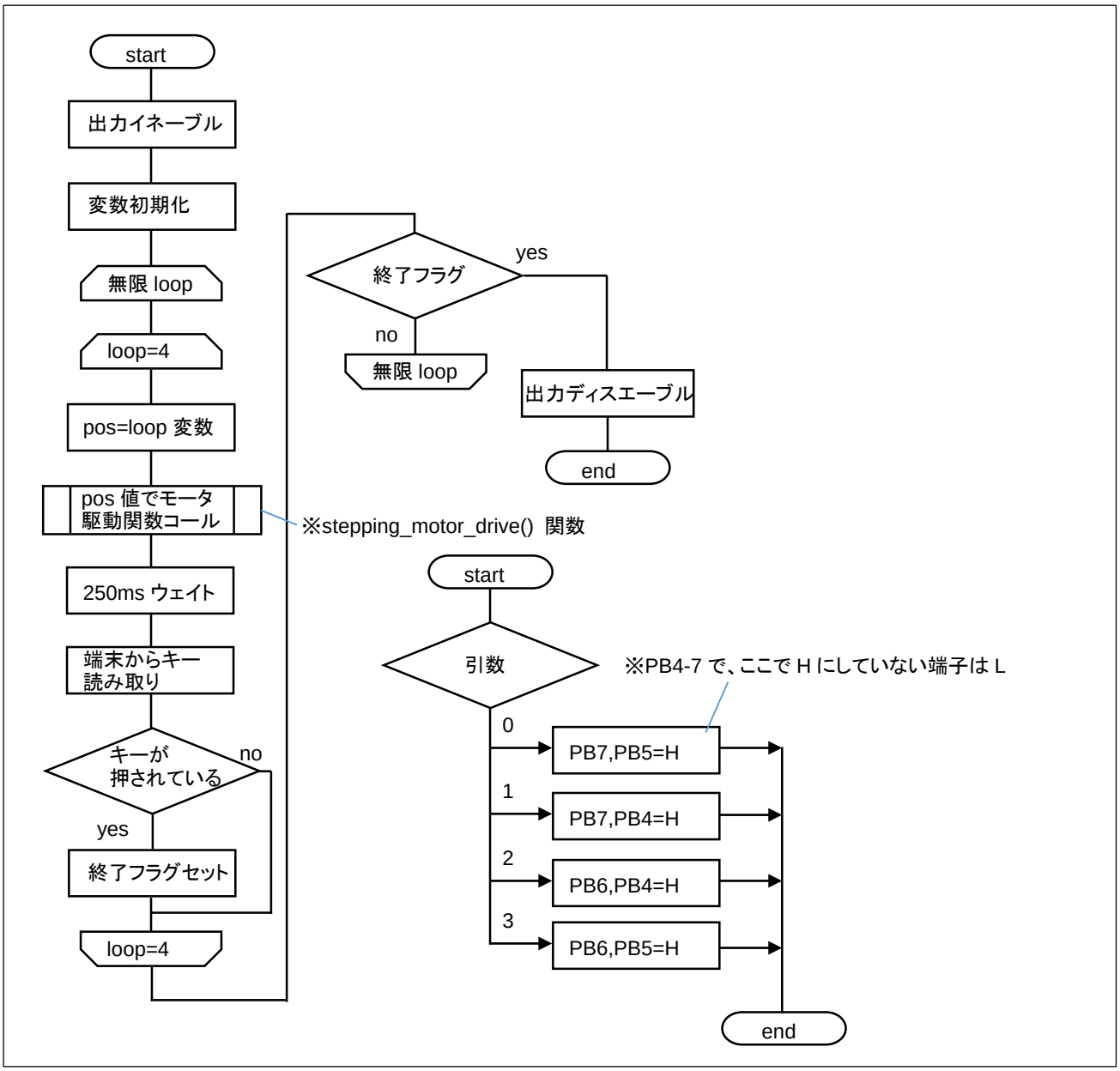


マトリックススイッチのスキャン自体は、100us 毎のコンペアマッチタイマ(CMT0)処理内で行われており、スキャン結果がスキャン値に格納済みとなっています。メイン関数内では、スキャン値を押されているスイッチ番号(11~26)に変換し、LCD への表示を行います。

また、2 つ以上のキーが押されているかを判断し、2 つ以上のキーが押されている場合は、LCD に*（アスタリスク）を表示します。

1.1.7. ステッピングモータ（左回転）(*7)

ステッピングモータを左回転させます。



PB4-7 で駆動されている 4 つのトランジスタ(Q5-8)の内、2 素子を L にドライブします。L にドライブする素子は、250ms 毎に、切り替えを行います。

pos	(PB7) Q5	(PB6) Q6	(PB5) Q7	(PB4) Q8
0	L		L	
1	L			L
2		L		L
3		L	L	

（PBn=H のとき、Qx=L）

1.1.8. ステッピングモータ（右回転）(*8)

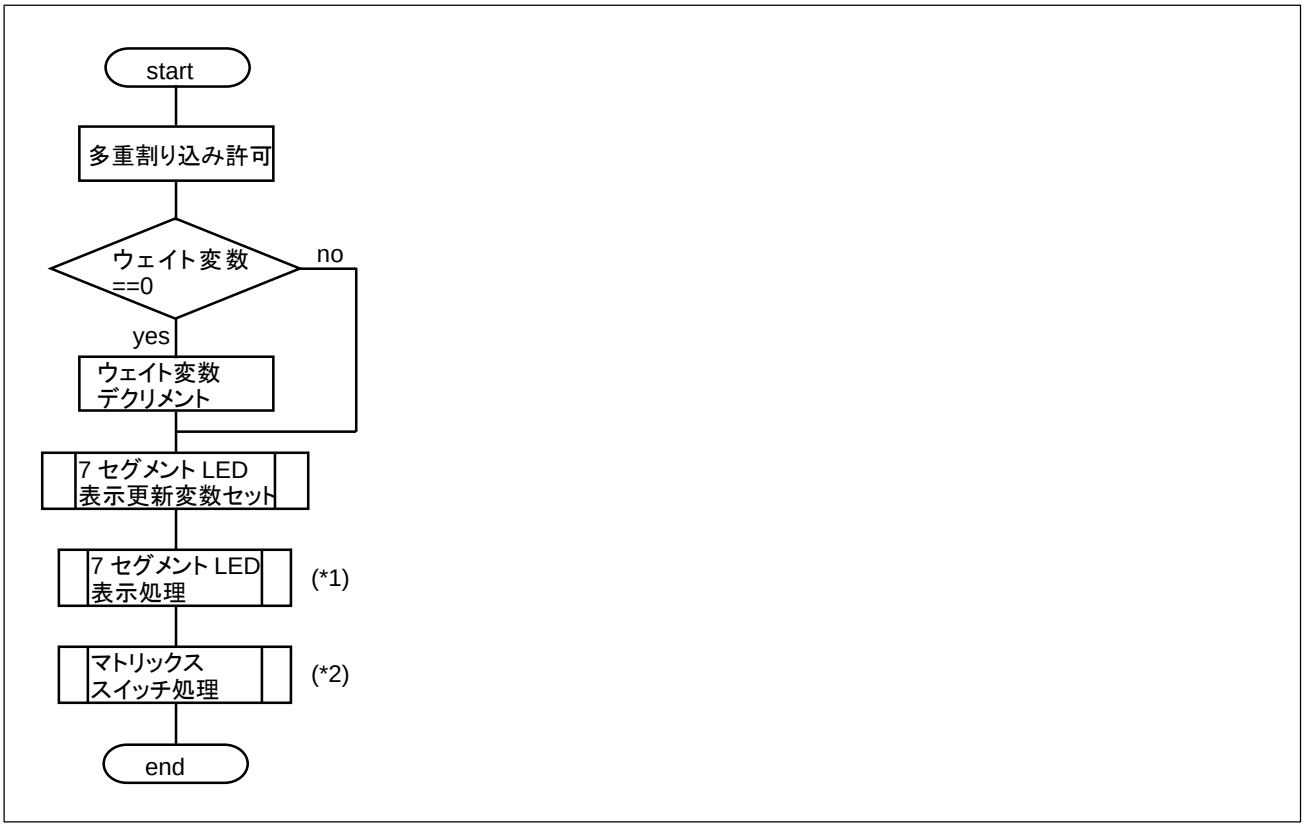
ステッピングモータを右回転させます。

左回転の場合は、pos を 0,1,2,3 の順で変化させますが、逆回転の場合は、pos=3,2,1,0 の順で stepping_motor_drive() 関数を呼び出すという違いのみです。

1.2. 割り込み関数

1.2.1. タイマ(CMT0)割り込み処理

・r_Config_CMT0_cmi0_interrupt()



タイマ割り込みは、100us 毎に実行される割り込み関数です。定期的に実行したい処理をこの関数内で実行しています。

※CMT0 の割り込み優先度は、6 に設定しています

6 より優先度が高い割り込みは、本ルーチン実行中でも割り込みルーチンに飛びます（多重割り込み許可）

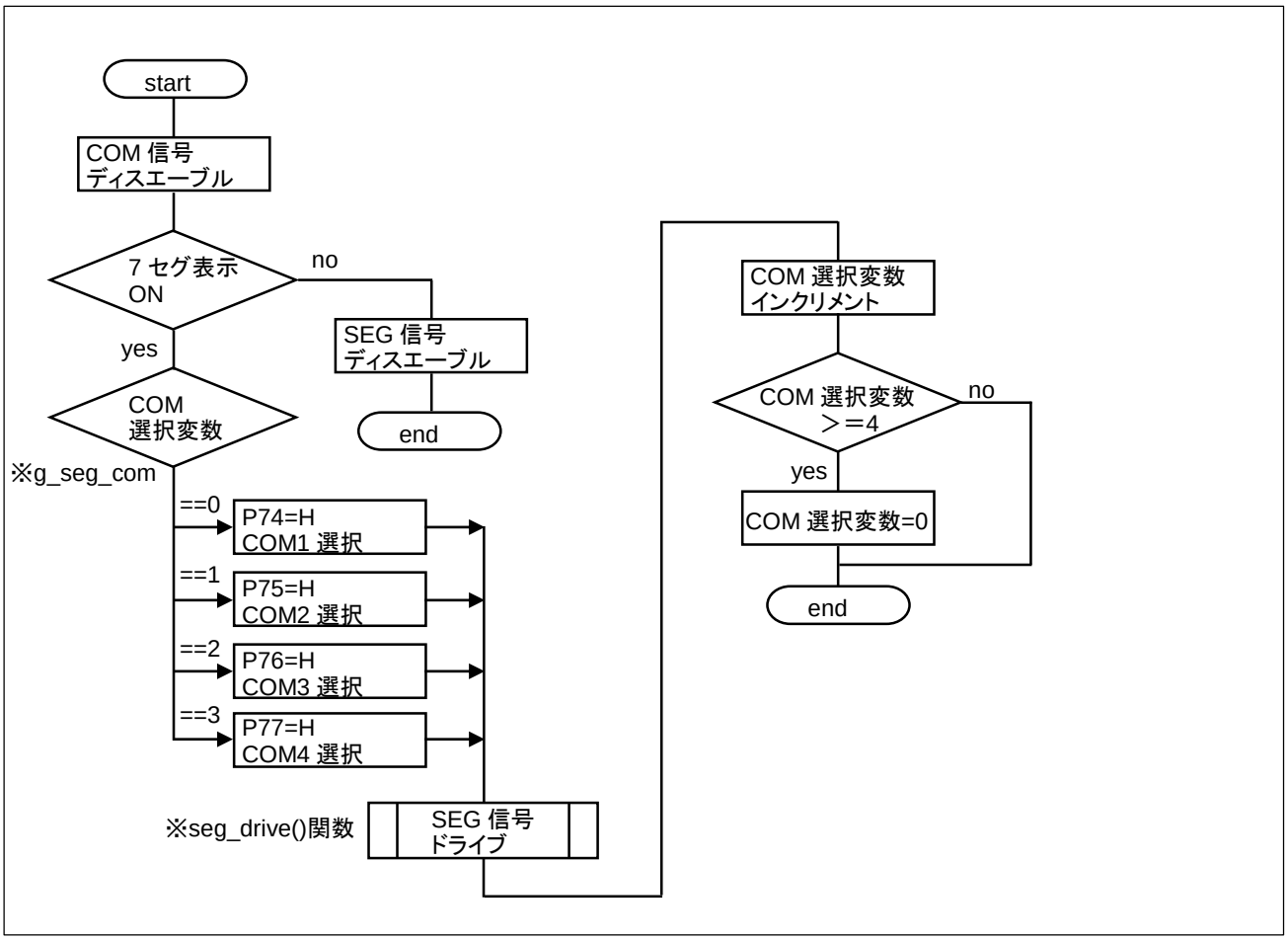
※V1.0 では、CMT0 内で SCI5 の文字表示を行っていましたが、V1.2 では割り込み（スマートコンフィグレータで出力する SCI5 の API 関数を使用）で文字表示を行う様に変更しました

1.2.2. 7 セグメント LED 表示処理(*1)

※1234 を表示する場合

(1)COM1 を選択(P74=H, P75-77=L)、セグメント(PA)を表示値"4"にセット
(2)COM2 を選択(P75=H, P74,P75-77=L)、セグメント(PA)を表示値"3"にセット
(3)COM3 を選択(P76=H, P74-75,P77=L)、セグメント(PA)を表示値"2"にセット
(4)COM4 を選択(P77=H, P74-76=L)、セグメント(PA)を表示値"1"にセット

100us 毎に、(1)→(2)→(3)→(4)→(1)…の繰り返しとしています。

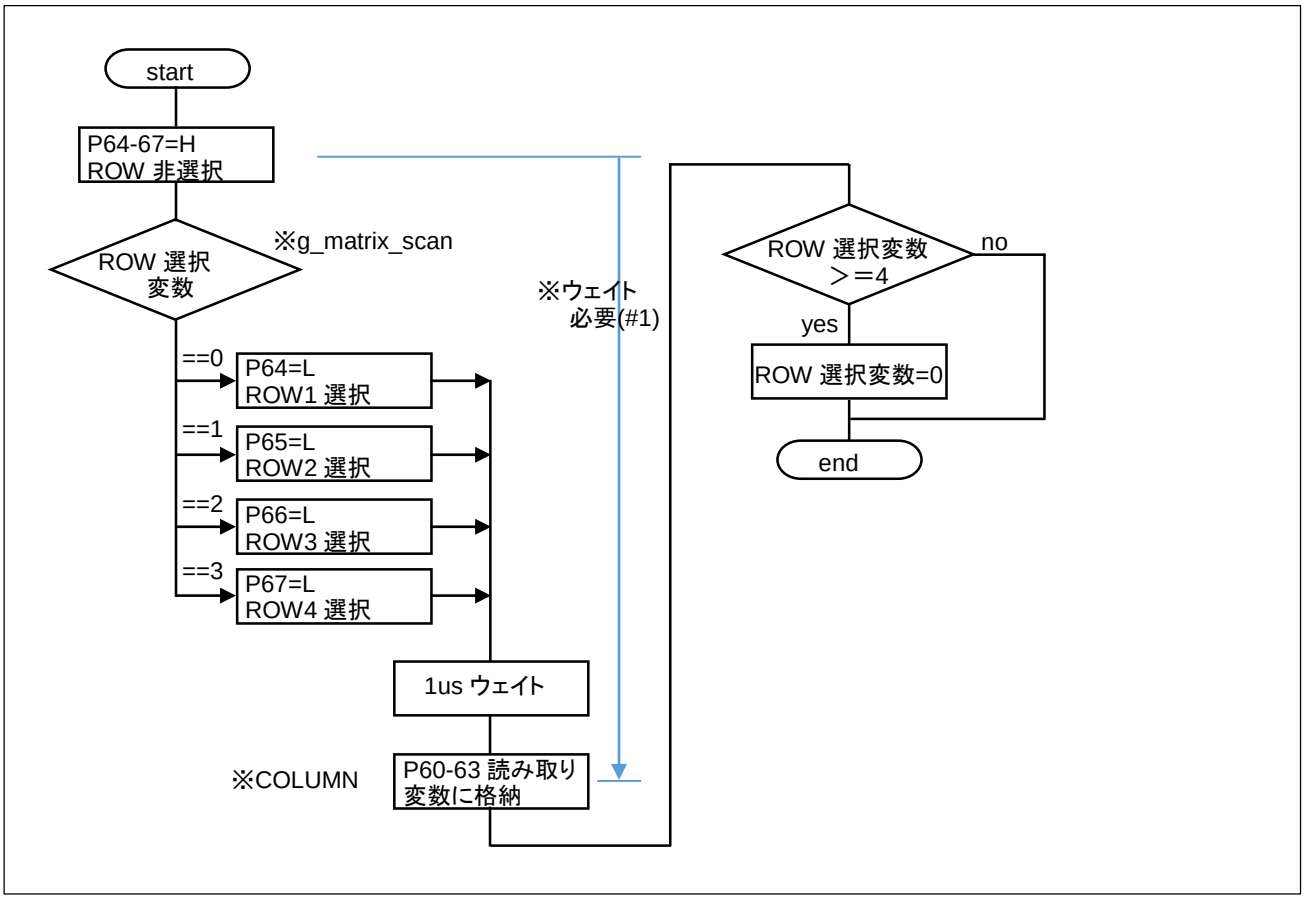


COM 選択変数は、0→1→2→3→0→…の様に変化します。選択された COM に対応する、SEG 信号をドライブ（設定）するという動作となります。

1.2.3. マトリックススイッチスキャン処理(*2)

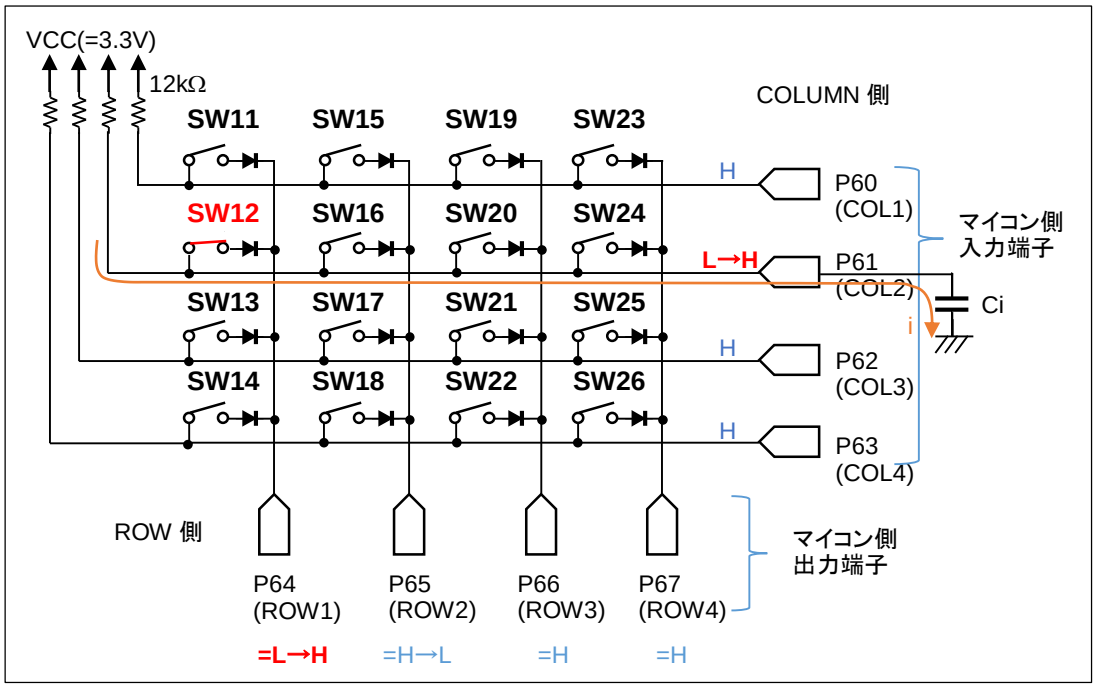
(1)ROW-1 を選択(P64=L, P65-67=H), P60-63 を読み取り（SW11-14 の状態を読み取り）
(2)ROW-2 を選択(P65=L, P64, P66-67=H), P60-63 を読み取り（SW15-18 の状態を読み取り）
(3)ROW-3 を選択(P66=L, P64-65, P67=H), P60-63 を読み取り（SW19-22 の状態を読み取り）
(4)ROW-4 を選択(P67=L, P64-66=H), P60-63 を読み取り（SW23-26 の状態を読み取り）

100us 毎に、(1)→(2)→(3)→(4)→(1)…の繰り返しとしています。



ROW 選択変数は、0→1→2→3→0→…の様に変化します。選択された ROW に対応する P60-63(COLUMN)の状態を読み取り、変数に格納します。

(#1)マトリックススイッチ読み取りのウェイトに関して



SW12 を押している状態で、ROW 選択が P64 から P65 に移るケースを考えると、以下の様になります。

・P64 が選択(=L)の時は、P61 は L

・P64 を H 設定

・P61 が H に遷移

※12kΩ のプルアップ抵抗で、マイコンの入力端子を H に引き上げるので、ある程度の時間が必要

※マイコンの入力端子の端子容量（8pF(max)）他で、トータル 20pF の場合　12kΩ　×　20pF = 240ns 程度の時定数となる

・P65 を L に設定

※この時点では P61 の H への遷移は完了していない

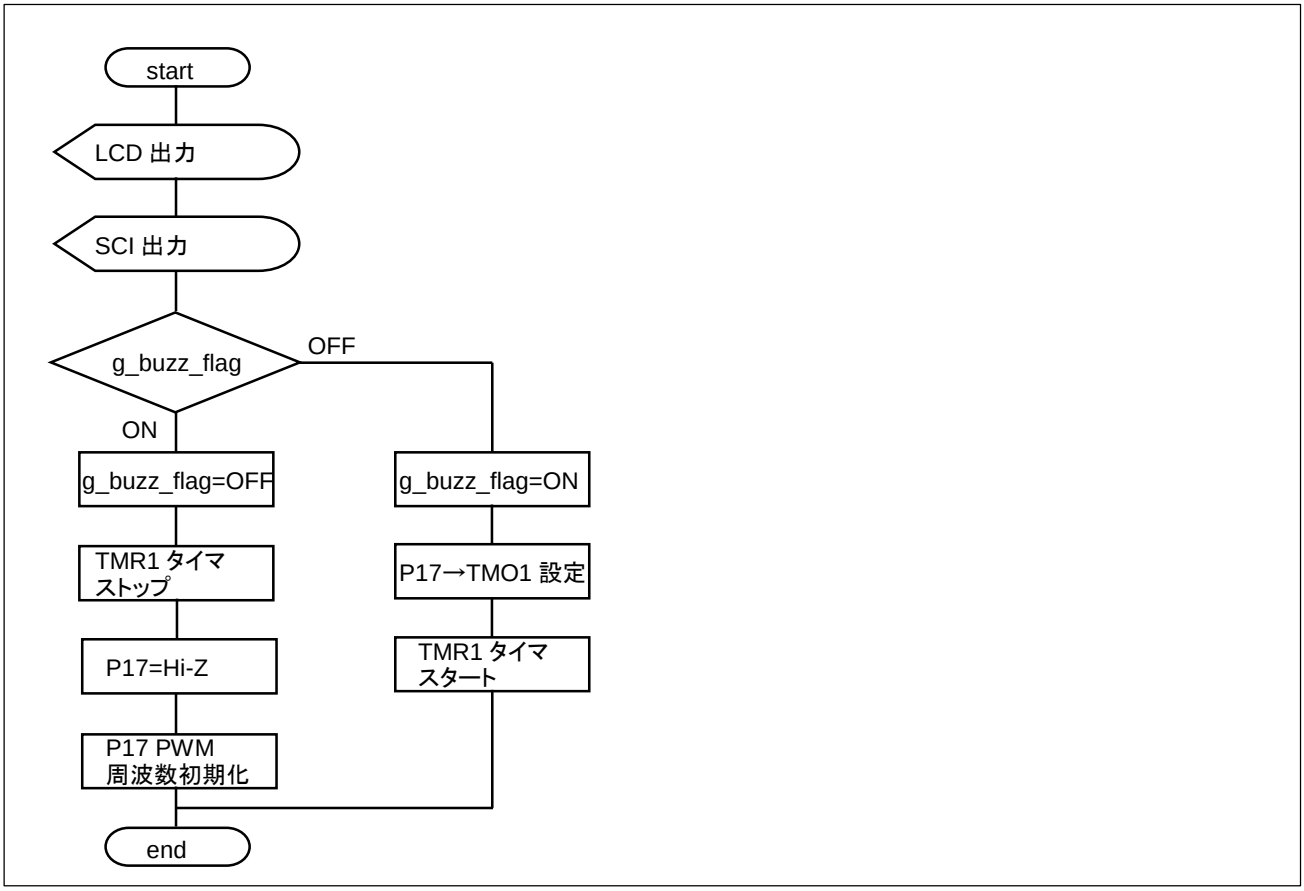
・1us のウェイト（240ns の時定数に対し、余裕を持った時間）

※P61 は H に遷移済み

・P60-63 を読み取り

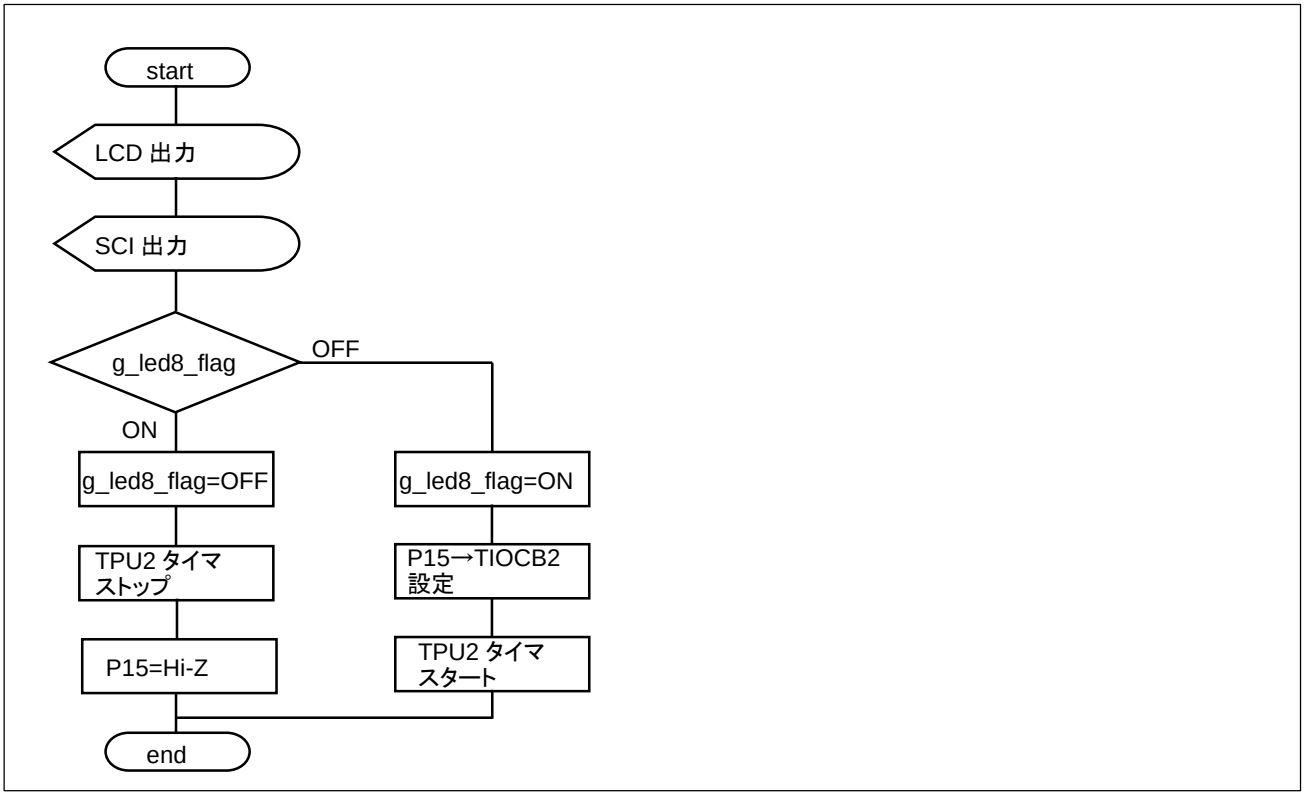
1.2.1. SW8(IRQ15)割り込み処理

・r_Config_ICU_irq15_interrupt()



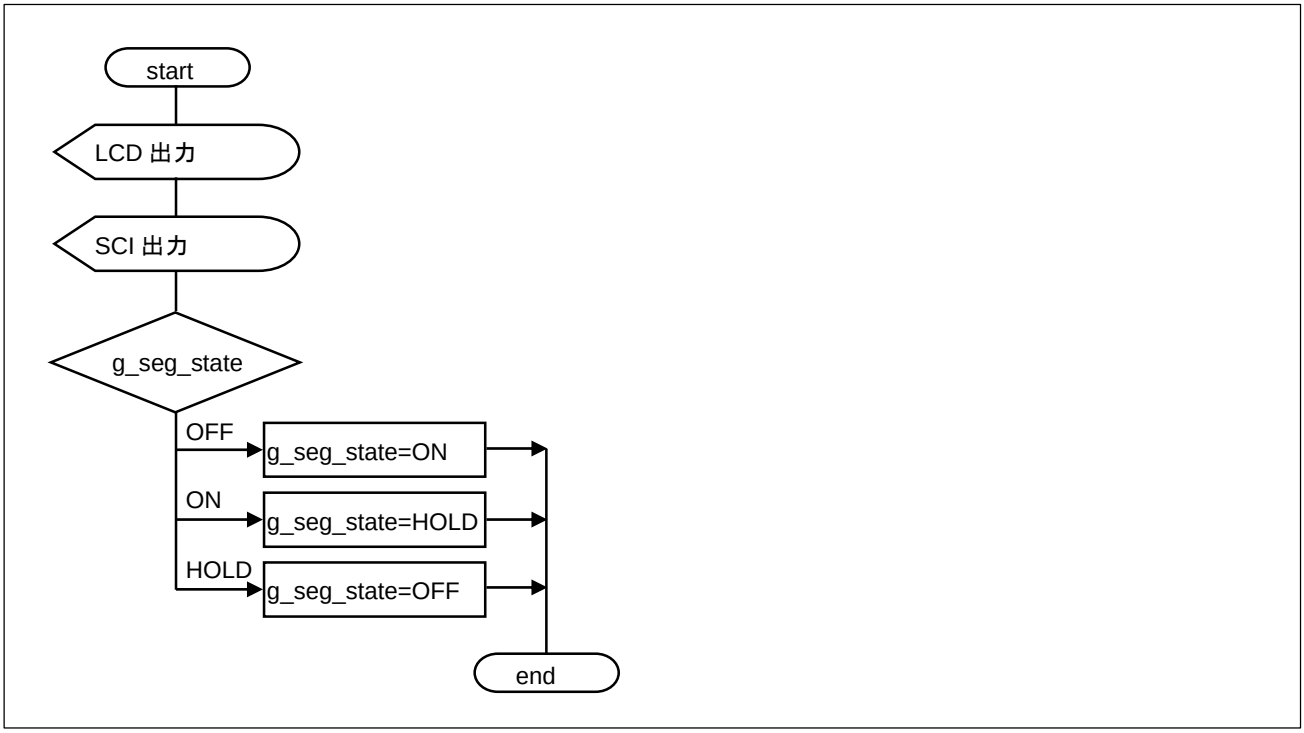
1.2.2. SW9(IRQ13)割り込み処理

・r_Config_ICU_irq13_interrupt()



1.2.3. SW10(IRQ4)割り込み処理

・r_Config_ICU_irq4_interrupt()



SW8-10 の割り込み処理は、

・LCD 表示

・SCI 表示

・フラグ変数の切り替え（トグル）

となります。

SW8(IRQ15)は、ブザー鳴動の ON/OFF。SW9(IRQ13)は、LED8 の ON/OFF。SW10(IRQ4)は、7 セグメント LED の、ON/HOLD/OFF の切り替えとなります。

1.3. ユーザ定義関数

ユーザ定義関数は、

HSBRX65-IO-BOARD_SAMPLE(_RX671)¥src¥user

以下に格納されているものです。

1.3.1. io_board.c に含まれる関数

mdelay

概要:ウェイト関数

宣言:

void mdelay(unsigned long msec);

説明:

指定された時間待ちます

※100us(0.1ms)刻みのタイマでカウントしているので、精度としては 0.1ms オーダです

引数:

msec: 待ち時間、ms 単位

戻り値:

なし

使用例:

```
mdelay(500);    //この行で 500ms 待ちます
```

sw_led

概要:SW0-7 読み取り LED0-7 制御関数

宣言:

void sw_led(void);

説明:

SW0-7 を読み取り、押されている SWn(n=0-7)に対応する、LEDn(n=0-7)を点灯させます

引数:

なし

戻り値:

なし

seg_led

概要:7 セグメント LED 制御関数

宣言:

void seg_led(unsigned short val, unsigned short dp);

説明:

指定された値を処理してグローバル変数に代入します

引数:

val: 表示値 ※hex で指定、0x0000~0xFFFF

dp: 少数点 ※0x0~0xf

4 つの少数点を全点灯させる場合は、0xf を指定

1.234 の様に最上位桁の小数点を表示させる場合は、0x8

戻り値:

なし

その他:

val を桁毎に分解し、g_seg_char[0]~ g_seg_char[3]に代入します（g_seg_char[]は、0x0~0xf）

dp を桁毎に分解し、g_seg_dp[0]~ g_seg_dp[3]に代入します（g_seg_dp[]は、0 または 1）

使用例:

```
seg_led(0x1234, 0);     //7 セグメント LED に、"1234"を表示させる(*1)（※指定するのが hex 値な事に注意）
seg_led(0xabcd, 0);     //7 セグメント LED に、"ABCD"を表示させる(*1)
seg_led(0x0123, 0x8);     //7 セグメント LED に、"0.123"を表示させる(*1)
seg_led(0x0123, 0x4);     //7 セグメント LED に、"01.23"を表示させる(*1)
```

(*1)正しくは、本関数では表示用の変数に値を代入するのみです

seg_drive

概要:7 セグメント LED セグメント信号ドライブ関数

宣言:

```
unsigned char seg_drive(unsigned char seg_char, unsigned char dp);
```

説明:

7 セグメント LED のセグメント信号をドライブするレジスタ値を返します

引数:

seg_char: 表示させる文字(0x0~0xf)

dp: 小数点（点灯:1, 消灯:0）

戻り値:

セグメント信号の、PORTA.PODR.BYTE に相当するレジスタ値

使用例:

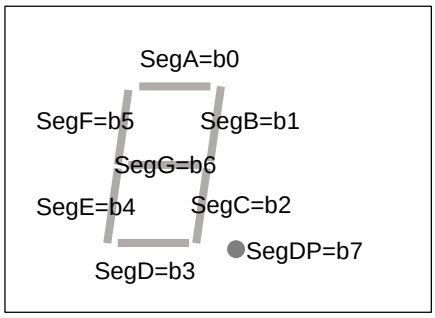
```
PORT7.PODR.BYTE=0x10;     //P74 のみ H（COM1 を選択）
PORTA.PODR.BYTE = seg_drive(0x3, 1);     //1 桁目に、"3."を表示
```

seg_char	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
7 セグ表示								
戻り値	0x3f	0x06	0x5b	0x4f	0x66	0x6d	0x7d	0x7

seg_char	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
7 セグ表示	8	9	A	b	C	d	E	F
戻り値	0x3f	0x6f	0x77	0x7c	0x39	0x5e	0x79	0x71

※戻り値は、dp=1（少数点点灯）の場合は、上記の値に 0x80 を OR した値となります

・セグメントと戻り値の関係



数値の 1 を表示させる場合は、SegB と SegC を点灯させれば良いので、bit1, bit2 を 1 に設定する。
→0b0000 0110 = 0x6

－seg_drive 関数ではサポートされていない表示に関して－

符号のマイナス(-)を表示させる場合
→bit6 のみ 1 に設定　0b0100 0000 = 0x40　PORTA.PODR.BYTE=0x40;

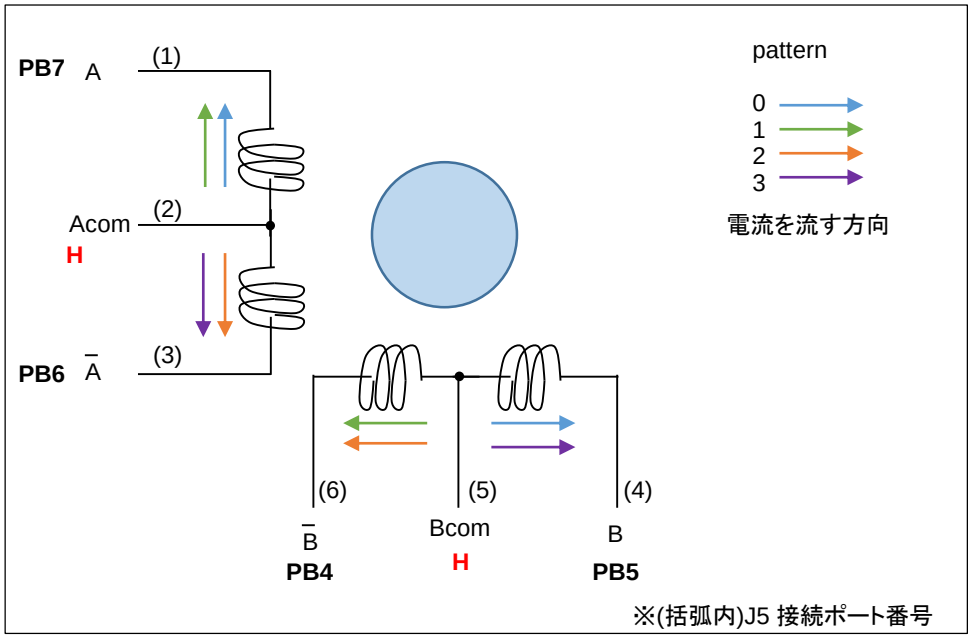
アルファベットの H を表示させる場合
→bit5, bit4, bit6, bit1, bit2 を 1 に設定　0b0111 0110 = 0x76　PORTA.PODR.BYTE=0x76;

上記の様に、7 つのセグメントで表示可能なものであれば、表示可能です。

stepping_motor_drive

概要：ステッピングモータ駆動関数
宣言：
　　void stepping_motor_drive(unsigned char pattern);
説明：
　　指定された値に応じて、ステッピングモータ駆動端子を ON/OFF します
引数：
　　pattern：　0~3 を指定
戻り値：
　　なし

－指定する pattern とステッピングモータに流す電流の方向の関係－



pattern	PB7	PB6	PB5	PB4
	A	$\overline{A}$	B	$\overline{B}$
0	L		L	
1	L			L
2		L		L
3		L	L	

pattern 引数に、0~3 の値を入れて本関数を呼び出した場合、上記の方向の電流を流します。

使用例:

```
while(1)
{
    stepping_motor_drive(0);
    mdelay(250);
    stepping_motor_drive(1);
    mdelay(250);
    stepping_motor_drive(2);
    mdelay(250);
    stepping_motor_drive(3);
    mdelay(250);
}
```

250ms 毎に、モータ軸を回転させる例

（pattern を 3,2,1,0 の順で変化させた場合は、回転方向は逆となります。）

stepping_motor_en

概要:ステッピングモータ出力イネーブル関数

宣言:

void stepping_motor_en(unsigned char on_off);

説明:

ステッピングモータ駆動端子の出力モード／Hi-Z を切り替えます

引数:

on_off: OFF(0), ON(1)を指定

戻り値:

なし

buzz_pitch

概要:ブザーの音程関数

宣言:

void buzz_pitch(unsigned char cycle);

説明:

TMR1 タイマ（P17/TMO1）の出力周期を変更します

引数:

cycle: 周期(~255)を指定（1/60MHz×1024=17.07us, cycle×17.07us の周期となります）

戻り値:

なし

1.3.2. lcd_1602.c に含まれる関数

lcd_init

概要:LCD 初期化関数

宣言:

void lcd_init(void);

説明:

LCD(SC1602)の初期化を行います

引数:

なし

戻り値:

なし

lcd_cmd

概要:LCD コマンド送信関数

宣言:

void lcd_cmd(unsigned char c);

説明:

LCD にコマンドを送信します

引数:

c: 送信するコマンド

戻り値:

なし

使用例:

lcd_cmd(0xc4);

カーソルを 2 行目の 5 桁目に移動

－LCD のカーソル移動（任意の場所に文字表示する方法）－

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x40	0x41	0x42	0x43	**0x44**	0x45	0x46	0x47	0x48	0x49	0x4a	0x4b	0x4c	0x4d	0x4e	0x4f

LCD を、2 行モードで使用した場合、2 行×16 文字表示となりますが、文字（キャラクタ）データを保管するメモリ(DDRAM)のアドレスは、上記となります。（2 行目が 0x40 から始まります）

カーソル移動のコマンドは、0b1nnn nnnn(n:address)ですので、n のところに 0x44(0b0100 0100)を設定すると、コマンドとしては、(0b1100 0100) 0xc4 となります。

lcd_hs1
lcd_hs2

概要:LCD カーソル移動関数

宣言:

void lcd_hs1(void);

void lcd_hs2(void);

説明:

LCD のカーソル（文字を表示する位置）を変更します

lcd_hs1 は、1 行目 1 文字目、lcd_hs2 は、2 行目 1 文字目です

引数:

なし

戻り値:

なし

lcd_clear

概要:LCD 表示クリア関数

宣言:

void lcd_clear(void);

説明:

LCD 表示をクリアします

引数:

なし

戻り値:

なし

lcd_write_char

概要:LCD 文字表示関数

宣言:

void lcd_write_char(unsigned char c);

説明:

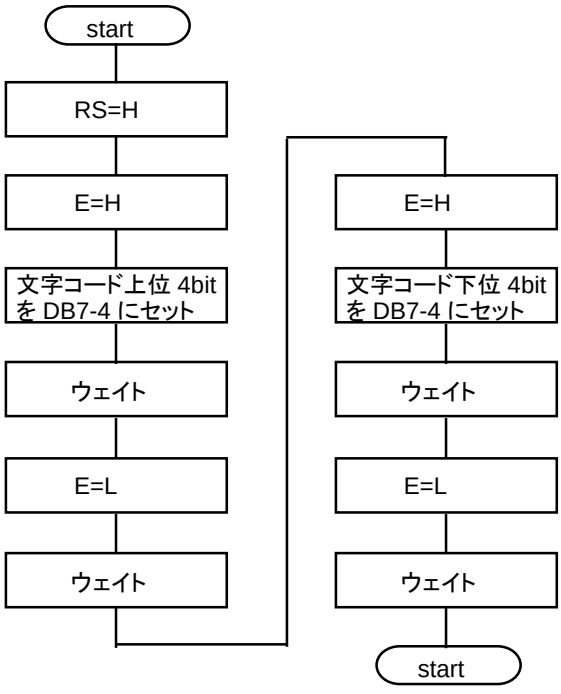
LCD に 1 文字表示させます（表示は、現在のカーソル位置です）

引数:

c: 表示する文字を指定

戻り値:

なし



LCD に文字を送る場合は、以下の手順で行います。

・RS=H に設定

・DB7-4 に送信データ(1)をセット

・E=L に変化したタイミングで LCD（SC1602 モジュール）がデータを取り込みます

・DB7-4 に送信データ(2)をセット

・E=L に変化したタイミングで LCD（SC1602 モジュール）がデータを取り込みます

※4bit モードでは、8bit（1 文字）を 2 回に分けて送信します

lcd_write_hex

概要:LCD 表示関数

宣言:

void lcd_write_hex(unsigned char c);

説明:

LCD に 1 バイトの hex コードを表示させます

引数:

c: 表示するコード

戻り値:

なし

使用例:

lcd_write_hex(0x2D); //LCD画面に"2D"を表示させます

lcd_write_byte_int

概要:LCD 表示関数

宣言:

void lcd_write_byte_int(unsigned char num);

説明:

LCD に 1 バイトを数値で表示します

引数:

num: 表示する数値(0~255)

戻り値:

なし

使用例:

lcd_write_byte_int(125); //LCD画面に"125"を表示させます

lcd_write_short_int

概要:LCD 表示関数

宣言:

void lcd_write_short_int(unsigned short num);

説明:

LCD に 2 バイトを数値で表示します

引数:

num: 表示する数値(0~65535)

戻り値:

なし

使用例:

lcd_write_byte_int(1200); //LCD画面に"1200"を表示させます

lcd_write_str

概要:LCD 文字列表示関数

宣言:

void lcd_write_str(unsigned char *str);

説明:

LCD に文字列を表示します

引数:

*str: 表示する文字列のポインタ(¥0 終端)

戻り値:

なし

使用例:

```
lcd_write_byte_int("ABC");    //LCD画面に"ABC"を表示させます
```

1.3.3. sci.c に含まれる関数

※V1.2 で変更

sci_start

概要:SCI 初期化関数

宣言:

```
void sci_start( void );
```

説明:

SCI の動作開始を行います

引数:

なし

戻り値:

なし

sci_write_char

概要:SCI 1 文字出力関数

宣言:

```
void sci_write_char( unsigned char c );
```

説明:

SCI に 1 文字出力行います

引数:

c: 表示する文字の文字コード(1 を表示する場合は、0x31 または'1')

戻り値:

なし

sci_write_str

概要:SCI 文字列出力関数

宣言:

```
void sci_write_str( const char *str );
```

説明:

SCI に文字列を出力します

引数:

*str: 文字列(¥0 終端)

戻り値:

なし

説明:

文字列に、"¥n"(0x0a)が含まれる場合は、"¥r¥n"(0x0d,0x0a)に変換されます

sci_write_uint8_hex
sci_write_uint16_hex
sci_write_uint32_hex

概要:SCI hex 出力関数

宣言:

```
void sci_write_uint8_hex( unsigned char c );
void sci_write_uint16_hex( unsigned short s );
void sci_write_uint32_hex( unsigned long l );
```

説明:

SCI に hex コードを出力します

引数:

c: hex コード(8bit)

s: hex コード(16bit)

l: hex コード(32bit)

戻り値:

なし

使用例:

```
sci_write_str("0x");
sci_write_uint8_hex( 0x5a );
```

端末に、"0x5a"が表示されます。

sci_write_uint8
sci_write_uint16
sci_write_uint32

概要:SCI 数値出力関数

宣言:

```
void sci_write_uint8( unsigned char c );
void sci_write_uint16( unsigned short s );
void sci_write_uint32( unsigned long l );
```

説明:

SCI に符号なしで数値を出力します

引数:

c: 表示させる数値(8bit)

s: 表示させる数値(16bit)

l: 表示させる数値(32bit)

戻り値:

なし

使用例:

```
unsigned short s = 0x8000;
sci_write_uint16( s );
```

端末に、"32768"が表示されます。

sci_write_int8
sci_write_int16
sci_write_int32

概要:SCI 数値出力関数

宣言:

```
void sci_write_int8( char c );
void sci_write_int16( short s );
void sci_write_int32( long l );
```

説明:

SCI に(負の数値の場合)符号付きで数値を出力します

引数:

c: 表示させる数値(8bit)

s: 表示させる数値(16bit)

l: 表示させる数値(32bit)

戻り値:

なし

使用例:

```
short s = 0x8000;
sci_write_int16( s );
```

端末に、"-32768"が表示されます

floart2str

float2str_eformat

double2str

double2str_eformat

概要:浮動小数点数ー文字列変換関数

宣言:

void float2str(float value, int num, char *str);

void float2str_eformat(float value, int num, char *str);

void double2str(double value, int num, char *str);

void double2str_eformat(double value, int num, char *str);

説明:

浮動小数点数を文字列に変換します

（_eformat は 1.23e-3 の様に e 形式に変換を行う関数です）

引数:

value: 表示させる数値(float/double 型)

num: 小数点以下の桁数

*str: 文字列変換後の先頭ポインタ

戻り値:

なし

使用例:

```
float f = 1.23456f;
char buf[20];
float2str(f, 2, buf);
sci_write_str(buf);
```

端末に、"1.23"が表示されます。

sci_read_char

概要:SCI 1 文字読み出し関数

宣言:

unsigned short sci_read_char(unsigned char *c);

説明:

SCI から 1 文字読み出しを行います

引数:

c: 読み出し結果(ポインタ)

戻り値:

0: 受信データがあり、*c に読み出した文字(ポインタ)を格納

0xffff(SCI_RECEIVE_DATA_EMPTY): バッファに溜まっている受信データがない

sci_read_str

概要:SCI 文字列読み出し関数

宣言:

unsigned short sci_read_str(char *str, unsigned short size);

説明:

SCI から文字列を読み出します

引数:

str: 読み出し結果(ポインタ)

size: 読み出すバイト数

戻り値:

0: 受信データが size バイト以上あり、*str に読み出した文字の先頭ポインタを格納

0xffff(SCI_RECEIVE_DATA_EMPTY): バッファに size バイトの受信データが溜まっていない

1.4. グローバル変数

1.4.1. io_board.c で定義している変数

・ウェイト関数向け

unsigned long g_wait_x100_us_counter

ウェイト関数(mdelay)向け変数。0以外の時は、100us毎に減算されます。mdelay関数では、この変数値が0になるのを待ちます。

・7セグメントLED向け

unsigned short g_seg_state　　//OFF, ON, HOLD

7セグメントLEDをON(1), OFF(0), HOLD(2)制御する状態制御フラグ変数。

unsigned short g_seg_flag　　//OFF, ON

7セグメントLEDの表示・非表示を制御するフラグ変数。

unsigned short g_seg_com

7セグメントLEDのCOM選択を保存する変数。0~3。

unsigned char g_seg_char[4]

7セグメントLEDの表示値を保存しておく変数。4:桁数分。

unsigned char g_seg_dp[4]

7セグメントLEDの小数点の表示値を保存しておく変数。4:桁数分。

・マトリックススイッチ

unsigned short g_matrix_scan

マトリックススイッチのROW選択を保存する変数。0~3。

unsigned short g_matrix_sw[4]

マトリックススイッチの読み取り値を保存する変数。4:ROW数分。

・A/D変換

unsigned short g_adc_flag

A/D変換中を示すフラグ。A/D変換実行前に1にセットし、A/D変換終了時に0にクリアされます。

unsigned short g_adc_val[1]

A/D変換結果を代入する変数。A/D変換終了時に、A/D変換値が代入されます。1:1ch分。

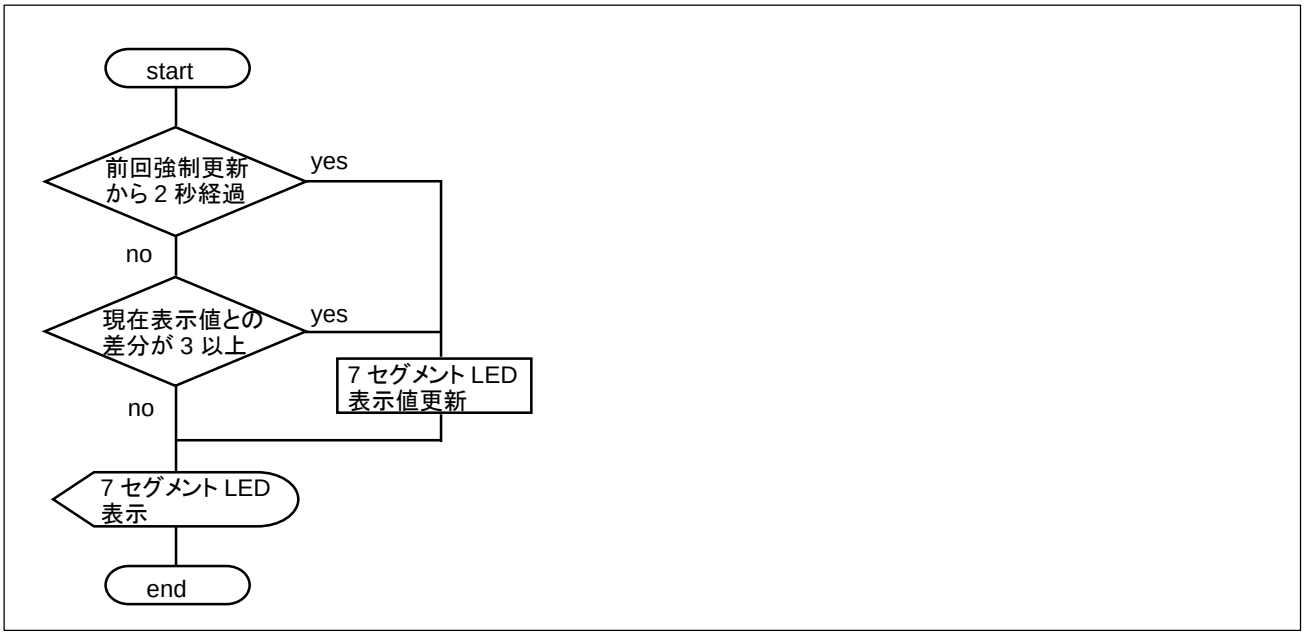
unsigned short g_adc_val_update

A/D変換結果を強制的に7セグメントLEDに反映させる変数。

unsigned long g_adc_val_update_counter

A/D変換結果を強制的に7セグメントLEDに反映させるためのカウンタ変数。

ーA/D変換値を7セグメントLEDに表示させるフローー



ボリューム(R29)の回転角度に応じた A/D 変換値が、7 セグメント LED に表示される動作となっていますが、A/D 変換値は、ばらつきが出るため、「A/D 変換値=7 セグメント LED 表示値」とした場合、表示が高速に切り替わり、値が読みにくいケースがあります。

そのため、

・現在の表示値から 3 以上差分が付いた場合

または

・2 秒に 1 回

表示値を更新するプログラムコードとしています。

・ブザー、LED8

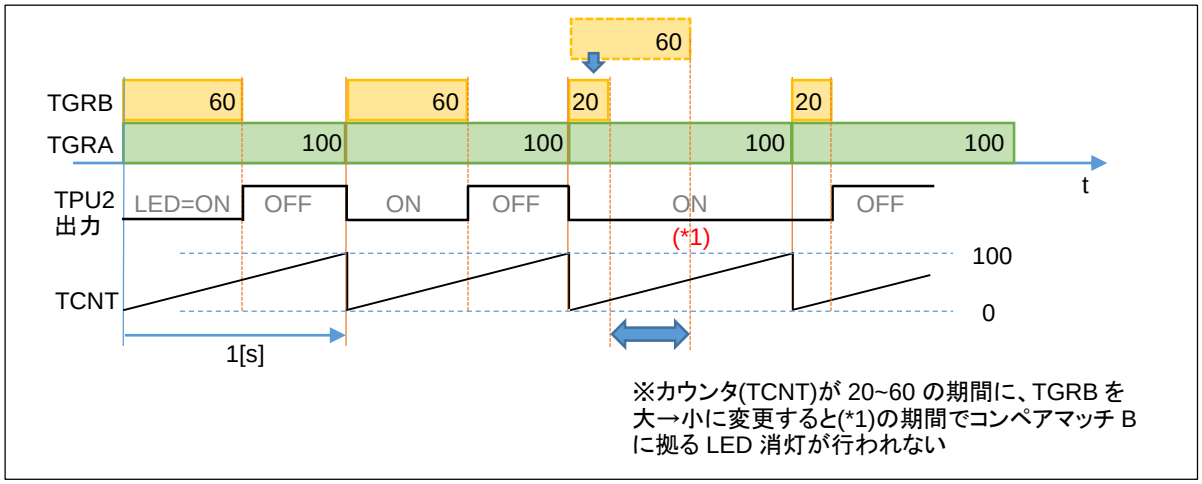
unsigned short g_buzz_flag
unsigned short g_led8_flag

ブザーとLED8をON/OFFする変数。

unsigned short g_tpu2_tgrb

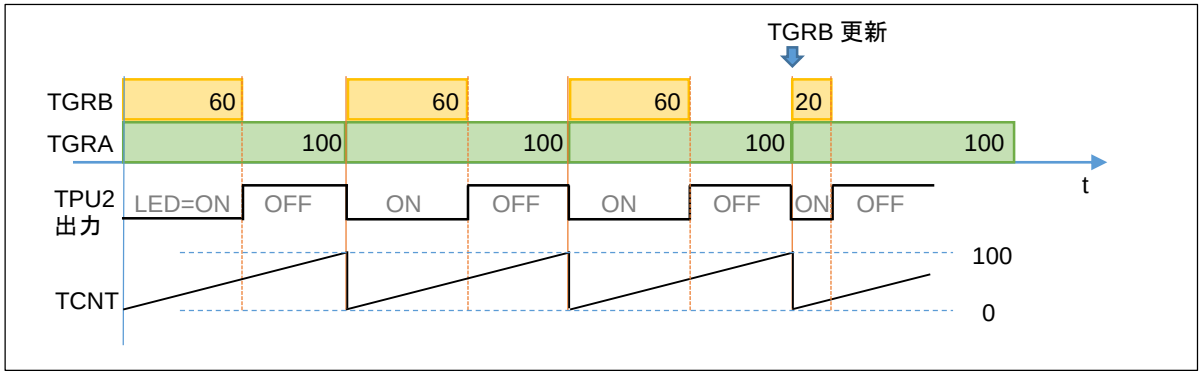
TPU2のコンペアマッチB（TPU2.TGRB=PWMのデューティ比を決めるレジスタ値）の設定値を代入する変数。

・任意のタイミングでTPU2.TGRBを更新した場合



任意のタイミングでTGRBを更新した場合、点滅の周期として設定している1秒より長く点灯するタイミングが来るケースがあります。（点灯時間を、60→20へと変更したにも拘わらず、120となるケースがでてきます）

・TGRAコンペアマッチのタイミングでTPU2.TGRBを更新



任意のタイミングで、グローバル変数(g_tpu2_tgrb)に値をセットしておき、レジスタ値(TPU2.TGRB)に反映させるタイミングを、TGRAのコンペアマッチ割り込みのタイミングとする事により、設定周期より長いON時間となる事が起きないようにしています。

```
const unsigned char g_buzz_cycle_ar[8]={28,25,22,21,19,17,15,14};
```

ブザー周期（周波数）の設定値。1 サイクル=1/60MHz × 1024=17.07[us]

28 : 2093[Hz]　※初期値

25 : 2344[Hz]

22 : 2663[Hz]

21 : 2790[Hz]

19: 3084[Hz]

17: 3447[Hz]

15: 3906[Hz]

14: 4185[Hz]

1.4.2. sci.c で定義している変数

```
unsigned char sci_send_buf[2][SCI_SEND_BUF_SIZE]
```

SCI表示バッファ（デフォルトでは、1024文字×2）。

```
unsigned char sci_recv_buf[SCI_RECV_BUF_SIZE]
```

SCI受信バッファ（デフォルトでは、16文字）。

1.5. 定義値

1.5.1. io_board.h で定義している値

```
#define OFF     0
#define ON      1
#define HOLD    2
```

ブザー, LED, 7セグメントLED等のON/OFFフラグ変数に設定、比較する定数。

```
#define ADC_VAL_HIST     3          //3以上変化した場合表示を更新
#define ADC_VAL_UPDATE_INTERVAL   20000          //2秒毎に強制的に更新
```

7 セグメント LED の表示安定化動作を決める定数。

1.5.1. lcd_1602_RX65.h で定義している値

```
#define LCD_1602_RS_PFS     MPC.P50PFS.BYTE=0x00;
#define LCD_1602_E_PFS     MPC.P52PFS.BYTE=0x00;
#define LCD_1602_DB4_PFS     MPC.P90PFS.BYTE=0x00;

#define LCD_1602_DB5_PFS     MPC.P91PFS.BYTE=0x00;
#define LCD_1602_DB6_PFS     MPC.P92PFS.BYTE=0x00;
#define LCD_1602_DB7_PFS     MPC.P93PFS.BYTE=0x00;

#define LCD_1602_RS_PMR     PORT5.PMR.BIT.B0=0;
#define LCD_1602_E_PMR     PORT5.PMR.BIT.B2=0;
#define LCD_1602_DB4_PMR     PORT9.PMR.BIT.B0=0;
```

```
#define LCD_1602_DB5_PMR     PORT9.PMR.BIT.B1=0;
#define LCD_1602_DB6_PMR     PORT9.PMR.BIT.B2=0;
#define LCD_1602_DB7_PMR     PORT9.PMR.BIT.B3=0;

#define LCD_1602_RS_PDR     PORT5.PDR.BIT.B0
#define LCD_1602_E_PDR     PORT5.PDR.BIT.B2
#define LCD_1602_DB4_PDR     PORT9.PDR.BIT.B0
#define LCD_1602_DB5_PDR     PORT9.PDR.BIT.B1
#define LCD_1602_DB6_PDR     PORT9.PDR.BIT.B2
#define LCD_1602_DB7_PDR     PORT9.PDR.BIT.B3

#define LCD_1602_RS_PODR     PORT5.PODR.BIT.B0
#define LCD_1602_E_PODR     PORT5.PODR.BIT.B2
#define LCD_1602_DB4_PODR     PORT9.PODR.BIT.B0
#define LCD_1602_DB5_PODR     PORT9.PODR.BIT.B1
#define LCD_1602_DB6_PODR     PORT9.PODR.BIT.B2
#define LCD_1602_DB7_PODR     PORT9.PODR.BIT.B3
```

LCD の各制御端子の割付を指定。

1.5.2. sci.h で定義している値

```
#define SCI_SEND_BUF_SIZE     1024
```

SCI の送信バッファサイズ（バイト数）を定義。このサイズのバッファを 2 つ使用しますので、実際に消費されるメモリは、1024×2=2048[bytes]になります。

```
#define SCI_RECV_BUF_SIZE     16
```

SCI の受信バッファサイズ（バイト数）を定義。

（※キーボードからの入力を受け取るバッファを想定しているので、16文字としています。データの受信に使用する場合等バッファサイズを増やす事は可能です。）

2. マイコン使用機能

2.1. 使用機能一覧

機能	用途	備考
CMT0 コンペアマッチタイマ 0	各種(7 セグメント LED、マトリックススイッチ) 周期処理	100us 周期
IRQ4	SW10 押下処理	立下りエッジ、デジタルフィルタ使用
IRQ13	SW9 押下処理	立下りエッジ、デジタルフィルタ使用
IRQ15	SW8 押下処理	立下りエッジ、デジタルフィルタ使用
汎用ポート	7 セグメント LED マトリックススイッチ プッシュスイッチ LED ステッピングモータ	
S12AD0 12 ビット A/D コンバータ	R29 の回転角度読み取り	
SCI5 シリアルコミュニケーション インタフェース	PC との通信	
TMR1 8 ビットタイマ	ブザー	
TPU2 16 ビットパルスタイマユニット	LED8	

2.2. I/O ボードの各機構に割り当てられているマイコン機能

I/O ボード機構	端子	マイコン機能 (1)	マイコン機能 (2)	マイコン機能 (3)	マイコン機能 (4)
7 セグメント LED(SEG1-2)	PA0-7, P74-77	**汎用ポート**			
キャラクタ LCD(J4)	P50,52,P90-93	**汎用ポート**			
ブザー(B1)	P17 P15	汎用ポート	**TMO1** TIOCB2	TIOCB0 MTIOC0B	MTIOC3A
LED(LED8)	P17 P15	汎用ポート	TMO1 **TIOCB2**	TIOCB0 MTIOC0B	MTIOC3A
ボリューム(R29)	P40	汎用ポート	**AN000**		
割り込み用スイッチ(SW8)	P07	汎用ポート	**IRQ15**	*ADTRG0	
割り込み用スイッチ(SW9)	P05	汎用ポート	**IRQ13**		
割り込み用スイッチ(SW10)	P14	汎用ポート	**IRQ4**		
マトリックススイッチ (SW11-26)	P60-67	**汎用ポート**			
プッシュスイッチ(SW0-7)	PD0-7	**汎用ポート**			
LED(LED0-7)	PE0-7	**汎用ポート**			
ステッピングモータ(J5)	PB4-7	**汎用ポート**	PO28-31		
USB-Serial(J7)	PC2-3	汎用ポート	**RXD5,TXD5**		

※**赤太字**は、本サンプルプログラムでの使用機能

2.3. 使用割り込み一覧

本サンプルプログラムで使用している割り込みの一覧を示します。

割り込み（機能名）	優先度	用途
CMI0(CMT0) コンペアマッチタイマ	6	7 セグメント LED，マトリックススイッチ, SCI バッファ出力
IRQ4(ICU) 端子割り込み	10	SW10 押下検出
IRQ13(ICU) 端子割り込み	10	SW9 押下検出
IRQ15(ICU) 端子割り込み	10	SW8 押下検出
RXI5(SCI5) SCI5 受信割り込み	8	USB-Serial 受信
TXI5(SCI5) SCI5 送信データエンプティ	8	USB-Serial 送信処理
TEI5(SCI5) SCI5 送信完了割り込み	8	USB-Serial 送信処理
ERI5(SCI5) SCI5 エラー割り込み	8	USB-Serial エラー処理
TGI2A(TPU2) TPU2 コンペアマッチ	4	TPU2.TGRB レジスタ更新
S12ADI(S12AD) A/D 変換完了	12	A/D 変換値を変数に格納

※優先度は、数値の大きなほうが優先です

2.4. I/O ポート初期設定

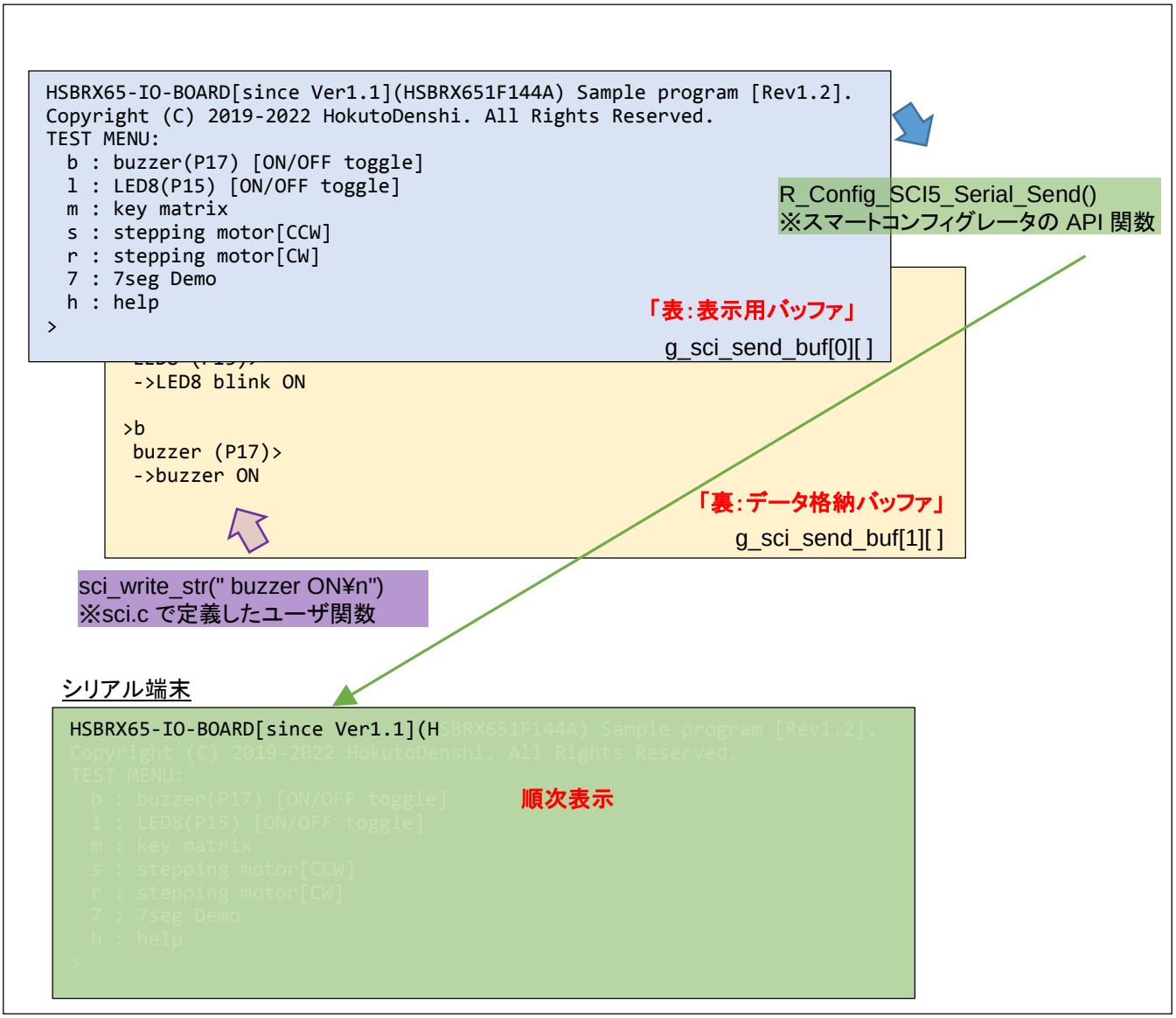
汎用ポートの初期設定を以下に示します。（初期設定は、スマート・コンフィグレータで設定しています）

I/O ボード機構	端子	ポート設定	備考
7 セグメント LED(SEG1-2)	PA0-7, P74-77	出力（初期値 0）	消灯
キャラクタ LCD(J4)	P50,52,P90-93	出力（初期値 0）	
マトリックススイッチ (SW11-26)	P60-63 P64-67	入力 出力（初期値 1）	 ROW 非選択
プッシュスイッチ(SW0-7)	PD0-7	入力	
LED(LED0-7)	PE0-7	出力（初期値 1）	LED は消灯
ステッピングモータ(J5)	PB4-7	出力（初期値 0）	

2.5. SCI5(USB-Serial)を使用したキャラクタの送受信

サンプルプログラム V1.2 で SCI の使用方法を大きく変えています。

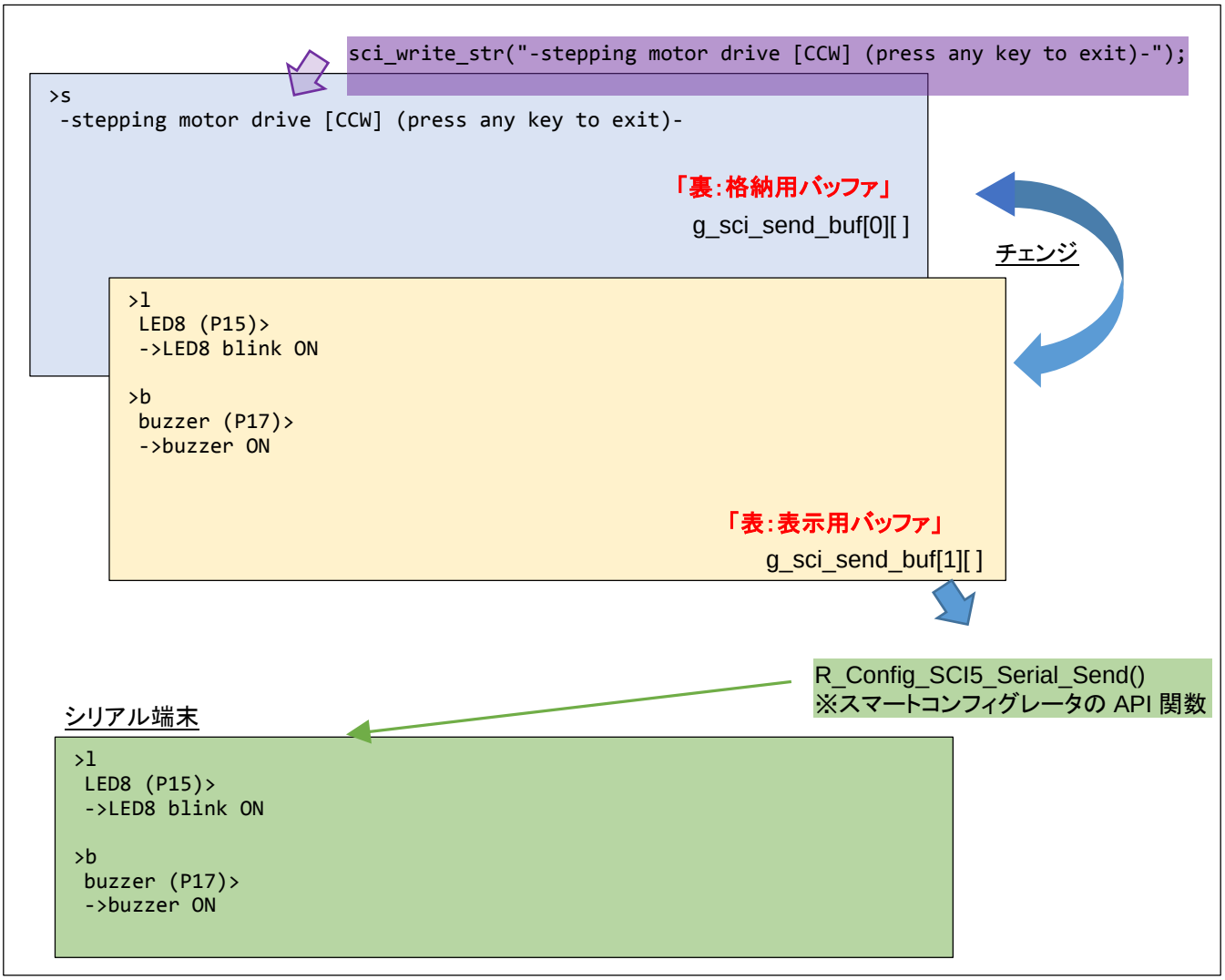
~V1.1 では、SCI の初期化のみスマートコンフィグレータで行い、文字出力のところはオリジナルのリングバッファ構成としていました。V1.2 では、スマートコンフィグレータの API 関数（文字出力、文字入力）を使う構成としています。



文字出力用のバッファは 2 面（2 つ）あり、表示用のバッファは API 関数(R_Config_SCI5_Send)に渡され、シリアル端末に順次データが送られます。シリアルは、115,200bps 設定なので、1 文字を送信するのに、87us（1/115,200×10）掛かります。これは、マイコンの処理（120MHz 動作、1 クロックあたり 8.3ns）に比べて非常に遅いので、文字出力が終わるまで待っていると効率が悪いためバックグラウンドで送信処理を行わせています。

※UART(SCI)では、スタートビット=1，データ（1 文字）=8bit，ストップビット=1 で合計 10bit で 1 文字送信します

表示用バッファのデータを全て出力した時点で、今度はデータ格納バッファのデータを API 関数に渡して、バッファの表（表示用）と裏（データ格納用）を入れ替えます。

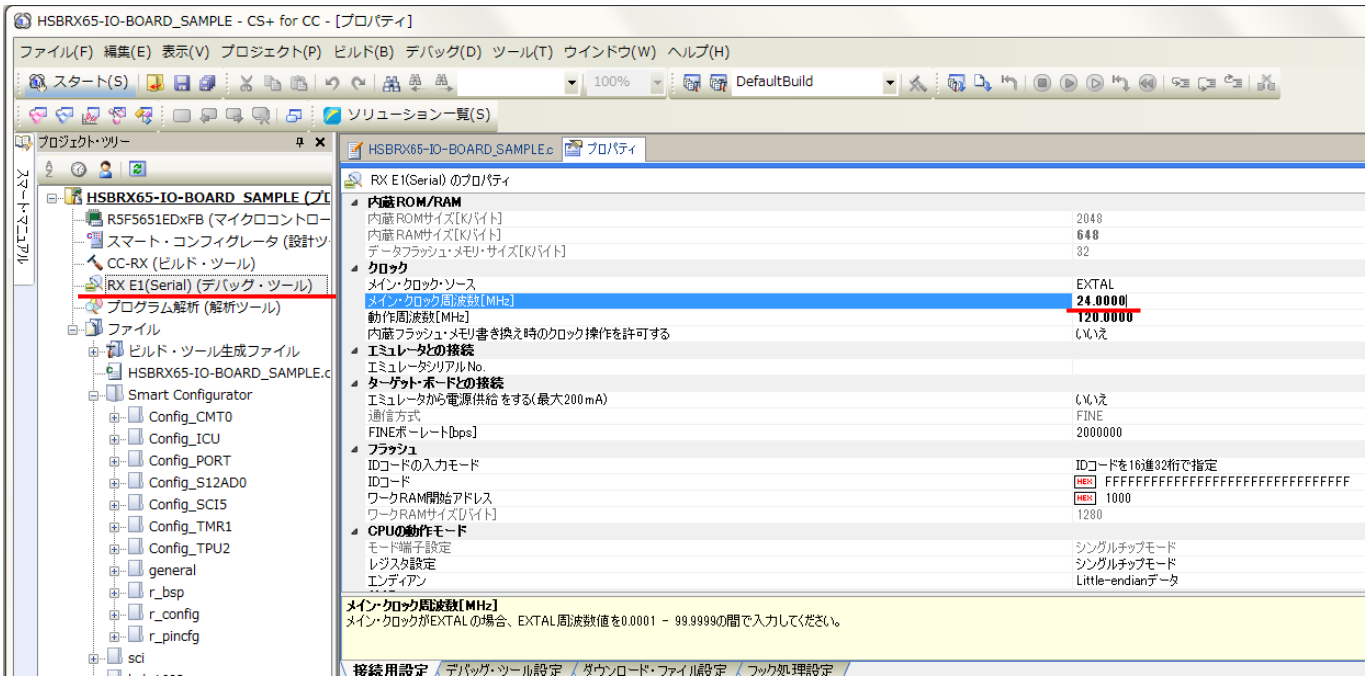


API 関数を使用する事で移植性（サンプルコードでは、条件分けを行い、RX 向けのコードを有効化していますが、RL78 や RA マイコンでも同じユーザ関数を使える様になっています）が高まります。

3. マイコンボードへのプログラム書き込み方法

3.1. デバッガを使用した書き込み

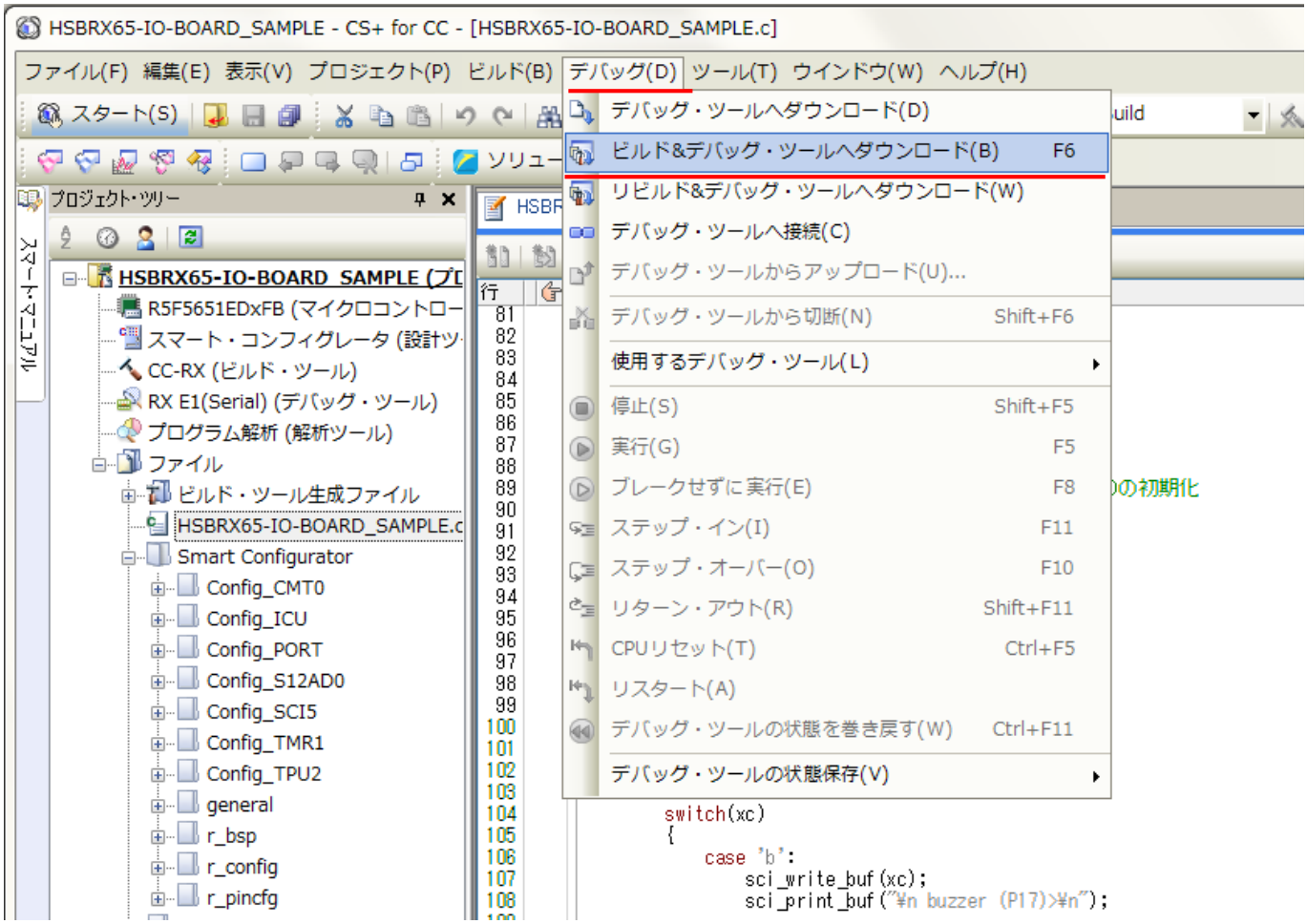
CS+を起動し、



デバッグ・ツール（右クリック）ー使用するデバッグツール
で、お手持ちのデバッガ(E1, E2, E2Lite, E20)を選択してください。
※E1, E20 の場合は、Serial と JTAG が選べますが、どちらを選んでも問題ありません
※E2, E2Lite の場合は、デバッガ選択後のプロパティで、通信方式："JTAG", "FINE"のどちらでも選択可能です

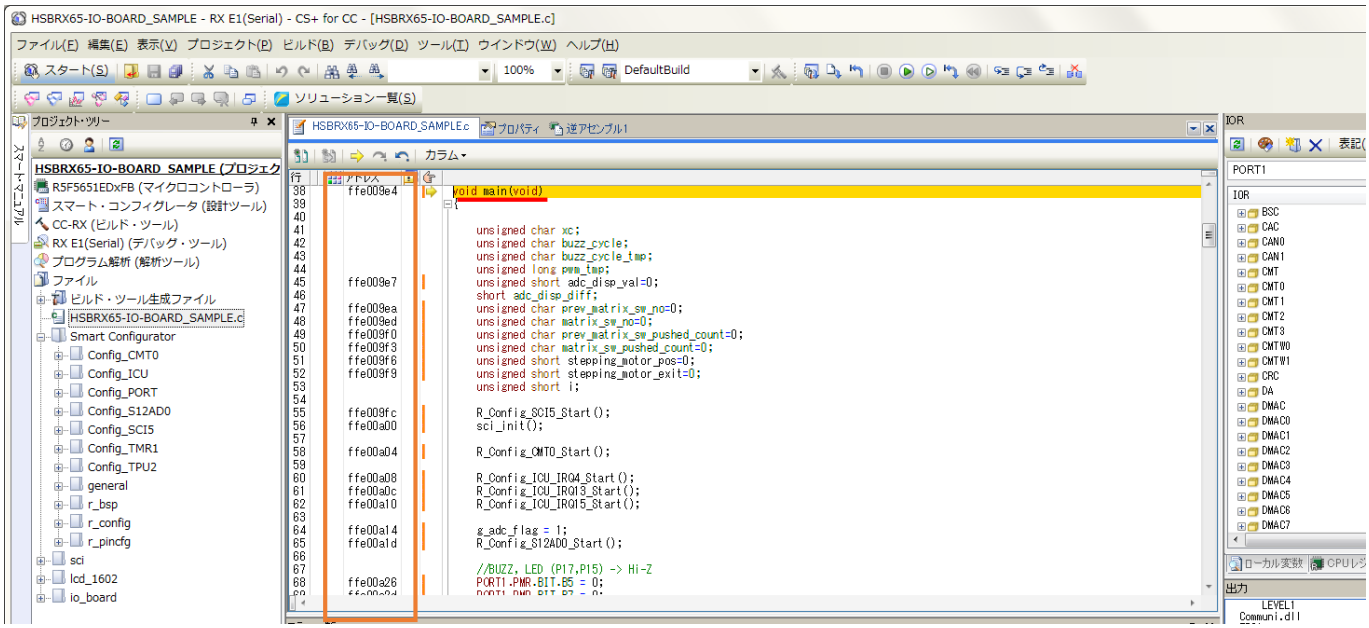
メイン・クロック周波数[MHz] **24.0000**
動作周波数[MHz] **120.0000**

クロック設定で、上記値を入力してください。

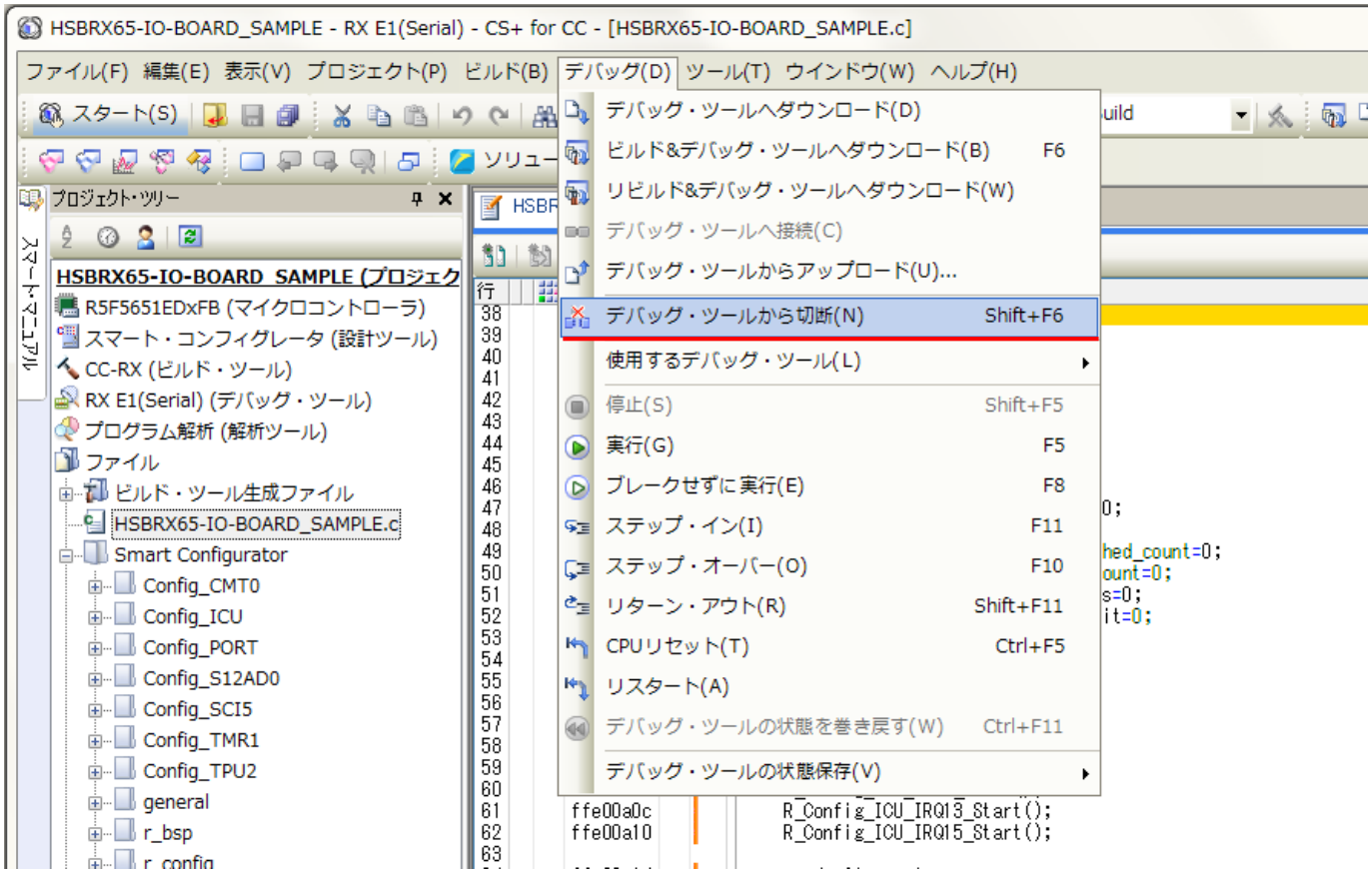


デバッグ－ビルド＆デバッグ・ツールへダウンロード

※プログラムのビルドとマイコンボードへのプログラムのダウンロード（書き込み）を同時に行う場合



メイン関数（void main(void)）の行でプログラムが停止、アドレスのところにプログラムのアドレスが表示されていれば、プログラムのダウンロードは成功しています。（プログラムがマイコンボードに書き込まれています）

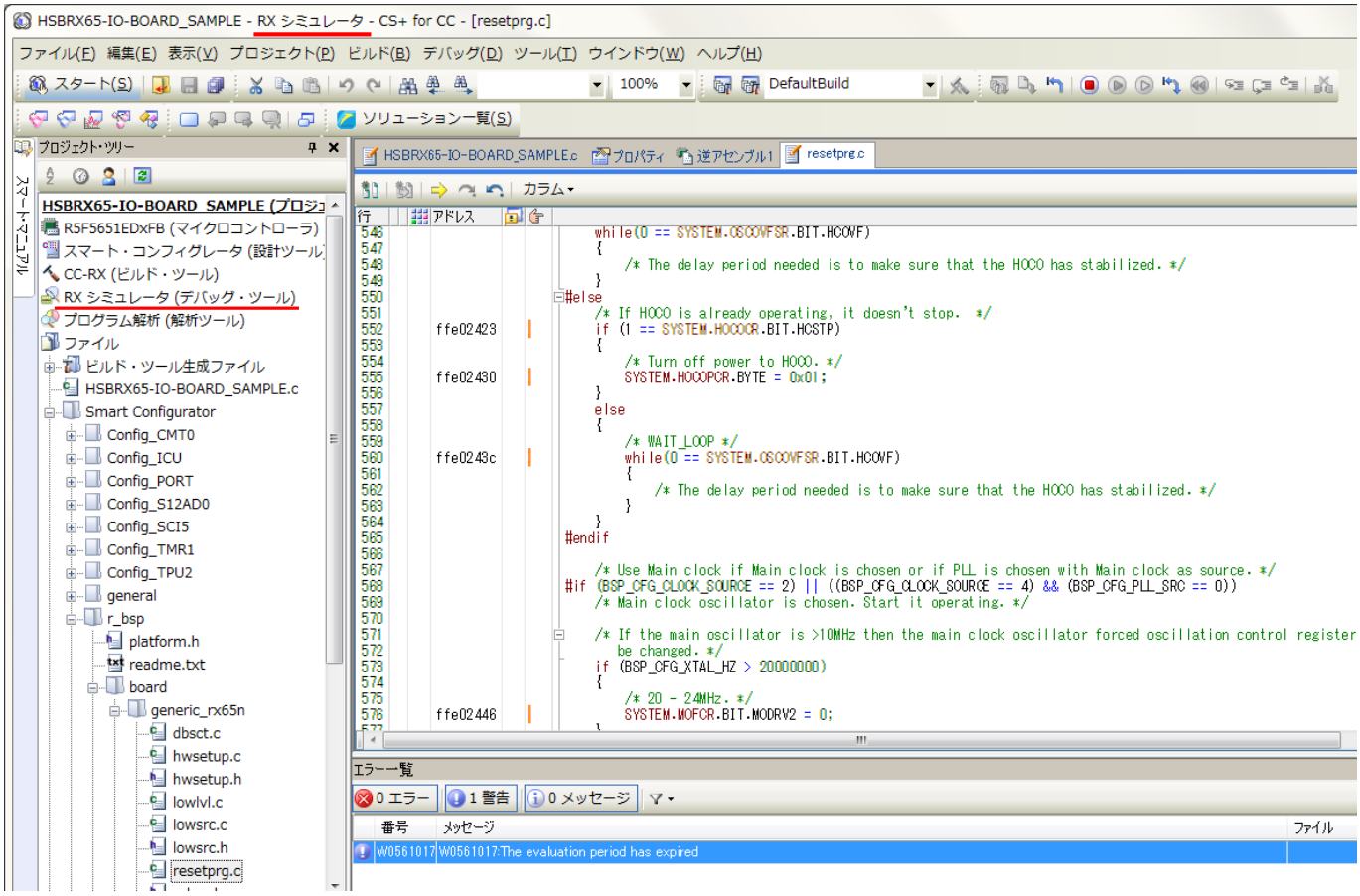


デバッグを終了させる場合は、

デバッグ－デバッグ・ツールから切断

を選んで、デバッガとマイコンボードの接続を切った後で、ボード電源断等行ってください。

ーデバッガの設定に問題があるケースー



デバッガの設定で「RX シミュレータ」を選択した場合、プログラムのダウンロードは実行されますが、マイコンボードに対しての書き込みは行われていません。
（プログラムは、メイン関数実行より前で止まります）

RX シミュレータの場合は、ツールがマイコンの動作を模擬するモードですので、実際のマイコンボードに対しての書込み、マイコンボード上でのプログラム実行等は行われません。

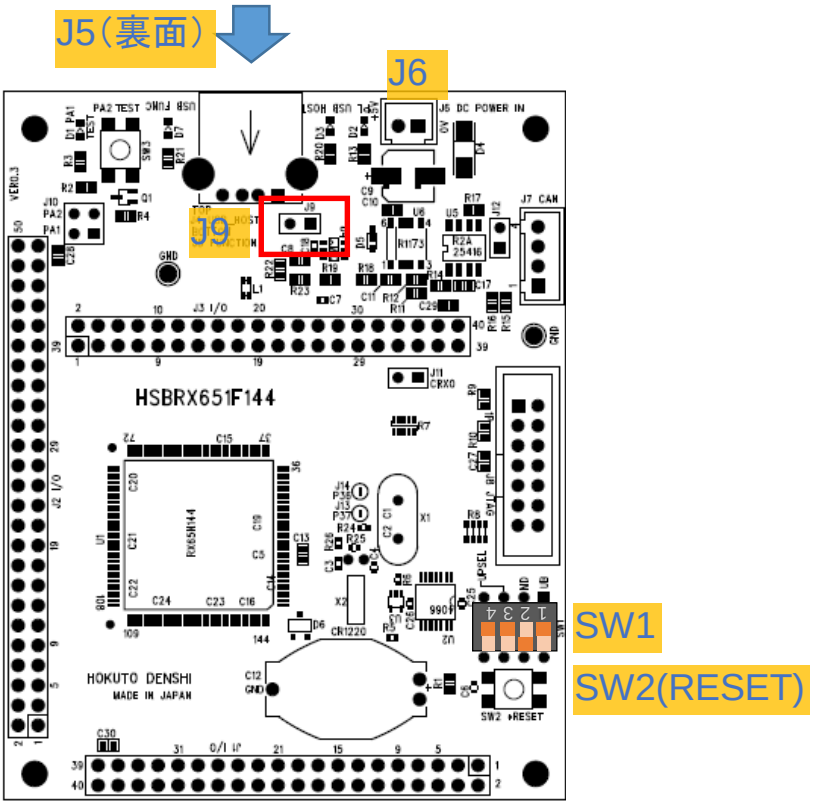
3.2. USB ケーブルを使用した書き込み

製品付属の USB ケーブル(USB-A-miniB)を、

マイコンボードの J5

に挿して、PC と接続してください。

J5（ボード裏面）に USB ケーブルを挿す



マイコンボードに給電を行ってください。

(1)I/O ボード J14 に AC アダプタを接続

(2)マイコンボード J6 に 5V 電源を接続

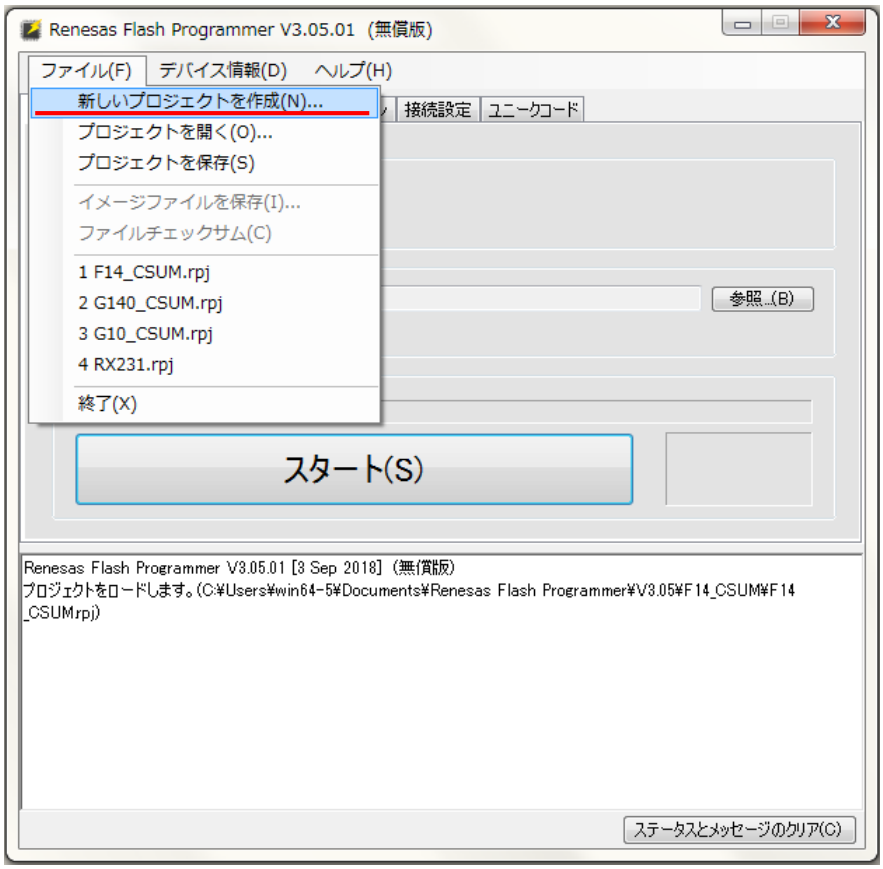
(3)マイコンボード J9 をショートに設定し、USB ケーブル経由で電源を供給

※(1)~(3)のいずれかを選んで、給電を行ってください

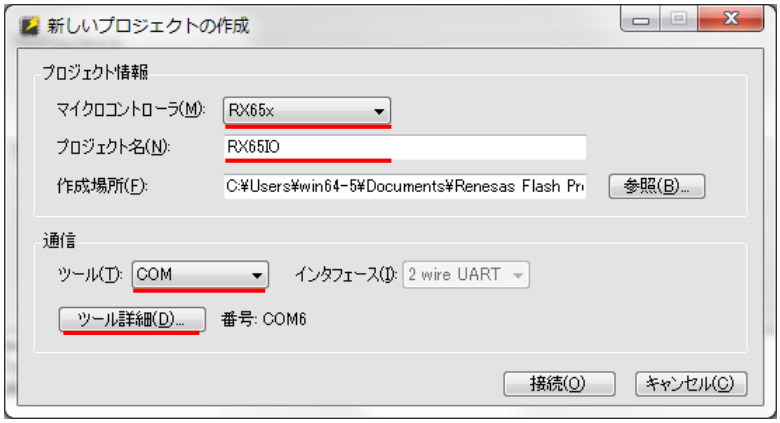
マイコンボード、SW1(DIP-SW)の 2 のみ、ON（下側）に設定（他は OFF）。

SW2（リセット）を押す。

RFP(RenesasFlashProgrammer)を起動



ファイルー新しいプロジェクトを作成

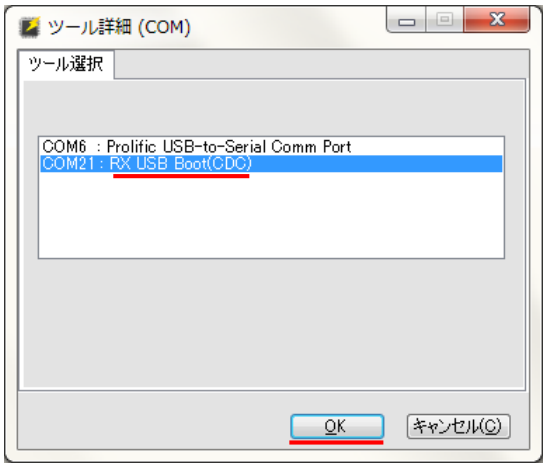


マイクロコントローラ　RX65x　を選択（HSBRX671F144 の場合は RX67x を選択）

プロジェクト名　任意の名称　を入力

ツール　COM　を選択

ツール詳細ボタンを押す

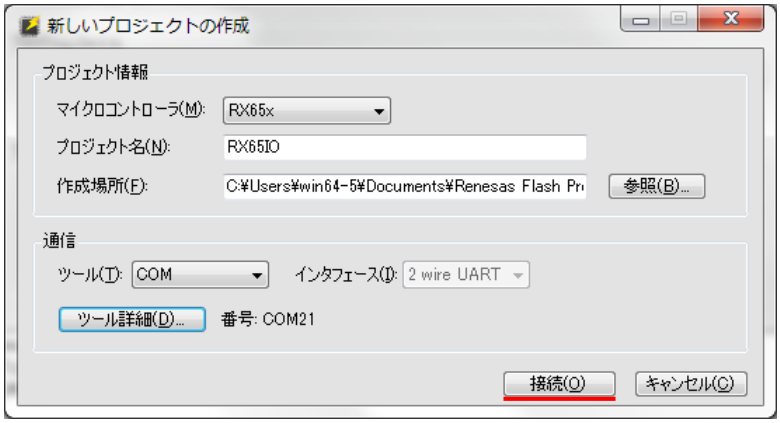


PC 上に、COM ポートが複数存在している場合は、選択ができますので、

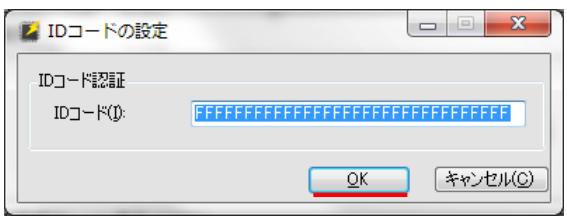
RX USB Boot(CDC)

となっている、COM ポート（上記では、COM21）を選択してください。
※RX USB Boot のデバイスが見えない場合は、マイコンボードの DIP-SW(SW1)の設定を確認後、マイコンボードをリセット（もしくは電源の再投入）してください

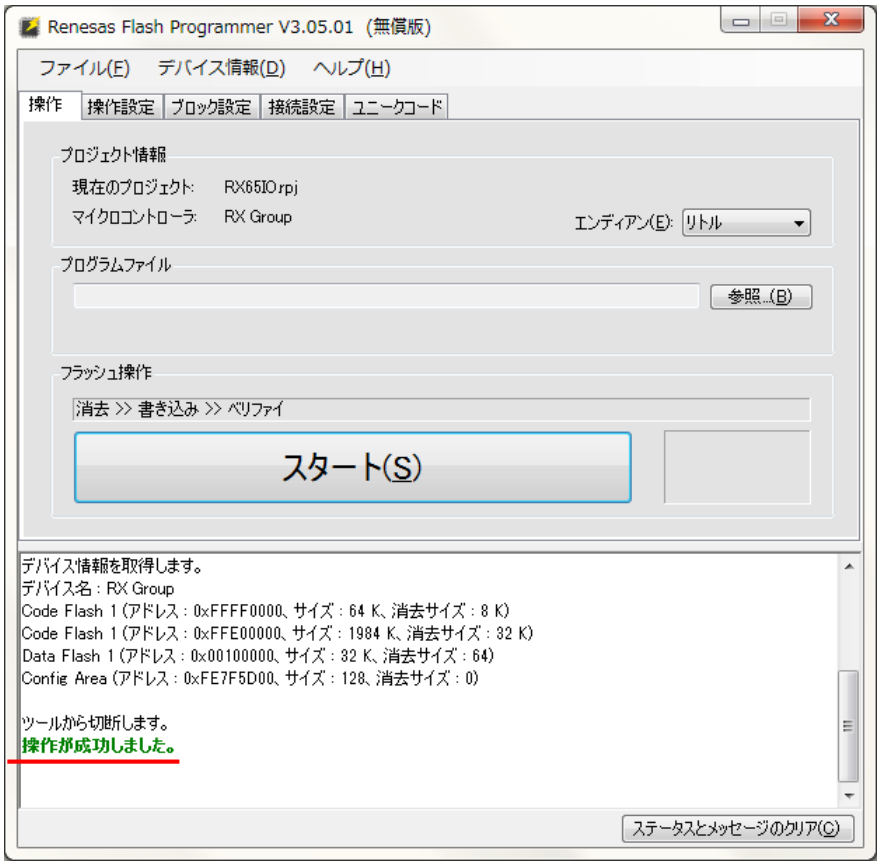
「OK」を押す。



「接続ボタン」を押す。



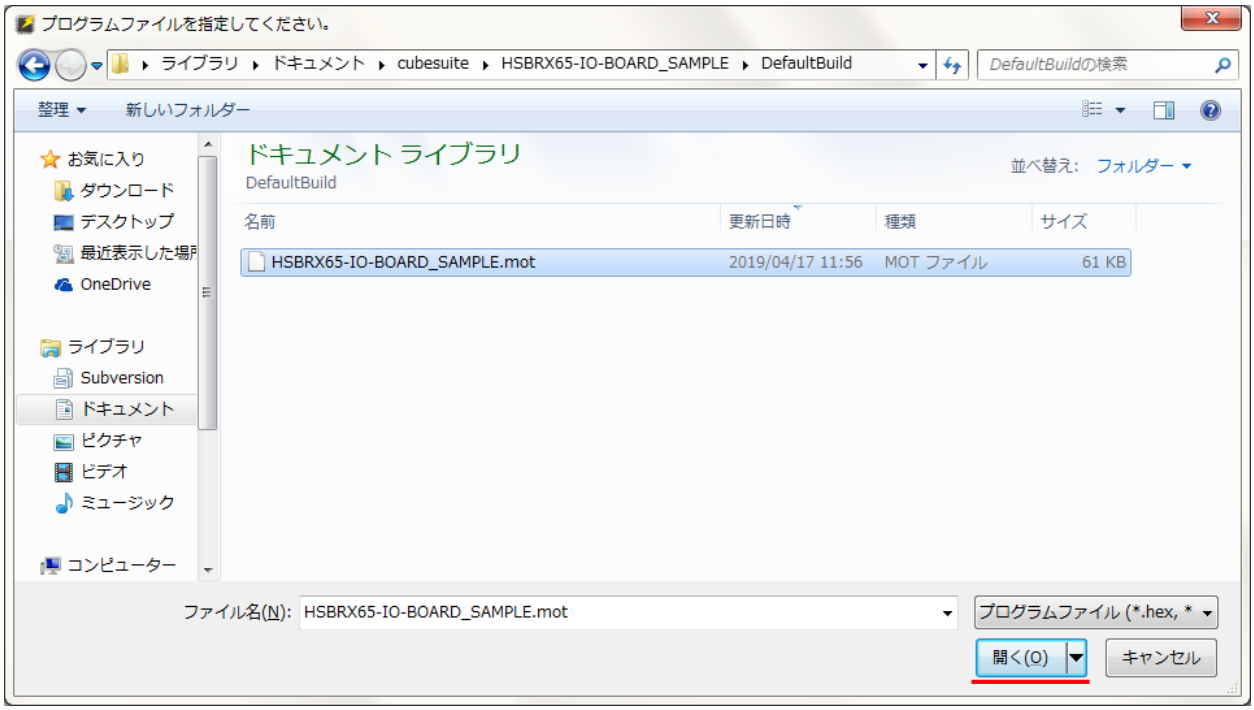
出荷時状態、特段 ID コードを設定していない状態ですと、デフォルトで FF(x16)となっていますので、「OK」を押す。



デバイス情報取得後、「操作が成功しました」という表示が出れば、問題ありません。

※接続に失敗した場合は、マイコンボードをリセット（もしくは電源の再投入）後に、再度「接続」を行ってください

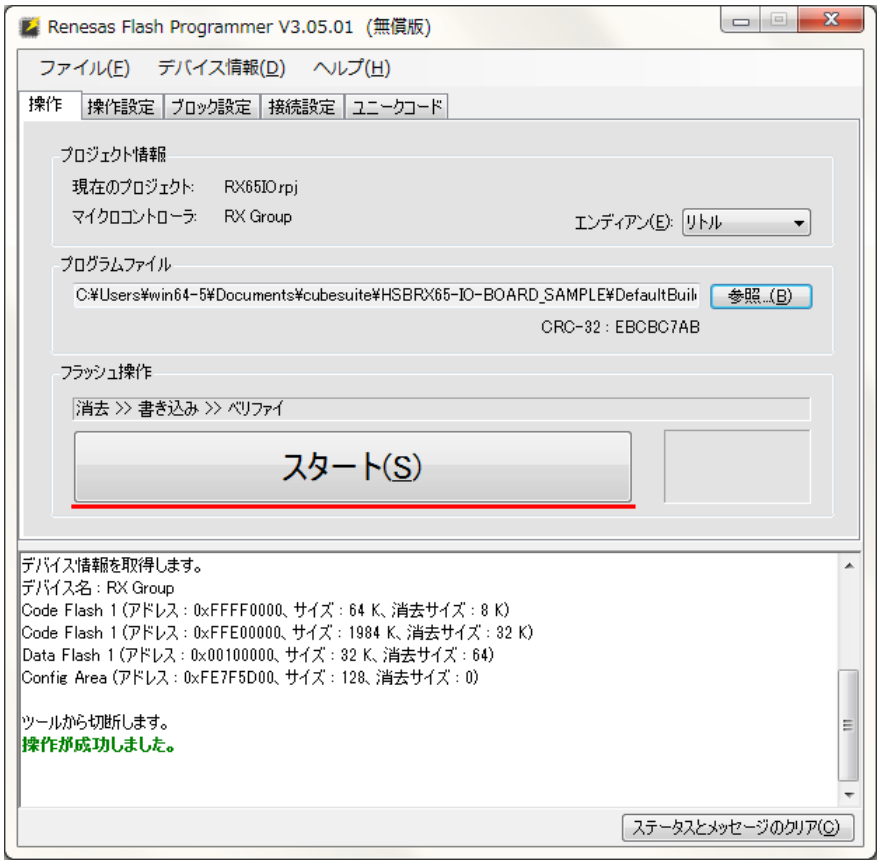
プログラムファイル 「参照」ボタンを押し、ビルドした mot ファイルを指定してください。



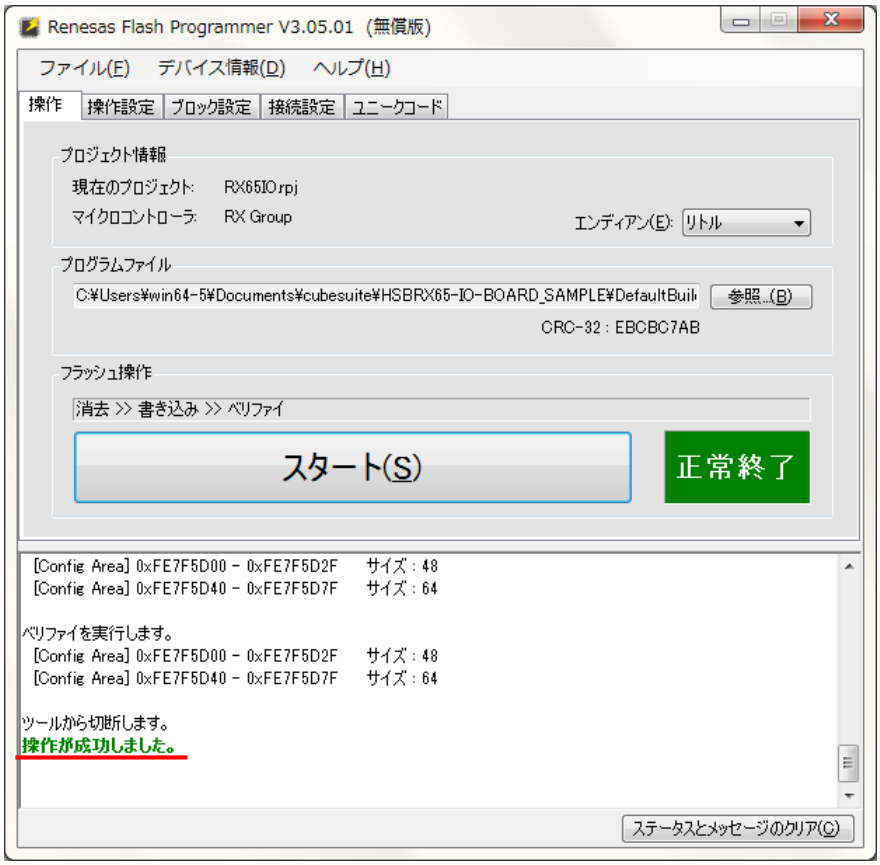
mot ファイル（ビルドしたプログラムのバイナリを書き込み用にテキストフォーマット化したファイル）は、プロジェクトフォルダの、DefaultBuild の下にあります。

マイコンボードをリセット（マイコンボードの SW2 を押す）（もしくは電源再投入）してください。 ※重要

※作成済みのプロジェクトを読み出して再度実行する場合は、この時点でのリセットは不要です



スタートボタンを押します。



消去、書き込み、ベリファイ後、「操作が成功しました」の表示が出力されれば、プログラムの書き込みは成功しています。

※書き込みに失敗した場合は、マイコンボードをリセット（もしくは電源の再投入）後に、再度「スタート」を行ってください

USB ケーブルをマイコンボード、J5 から抜いてください。（マイコンボードの電源を切断してください）

マイコンボードの SW1(DIP-SW)を全て OFF 側に設定してください。

マイコンボードに電源を接続してください。

上記手順で、マイコンボードに書き込まれたプログラムが実行を開始します。

デバッガが手元にないといった場合でも、USB ケーブルのみでプログラムの書き込みを行う事が可能です。

付録

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	2019.4.18	—	初版発行
REV.1.1.0.0	2022.4.13	P8,10-13,36,64 P15 P30,44-48,53,54,56-58	HSBRX671F144 の記載を追加 コメント追加 SCI 処理変更に伴う修正
REV.1.2.0.0	2022.5.31	P6,7	ボード Ver2.0 の注意事項追加

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 エレクトロニクス社　RX651/RX671(QFP-144 ピン)搭載 HSB シリーズマイコンボード向け I/O ボード

HSBRX65-IO-BOARD
サンプルソフトウェア　マニュアル

株式会社 北斗電子
 2022 年 5 月 31 日改訂 REV.1.2.0.0 (220531)