RL78 ファミリ

QE と SIS を使用した静電容量タッチアプリケーションの開発

要旨

本アプリケーションノートでは、RL78 MCU を使用した静電容量タッチセンシングを応用したアプリケーションの作成に必要な手順を説明します。

動作確認デバイス

静電容量センサユニット(CTSU)をサポートする RL78 ファミリ

目次

1. 概要

本アプリケーションノートでは、RL78 MCU を使用した静電容量タッチ機能をシステムに組み込む、以下の手順を説明します。

- RL78/G23 MCU ボードを使用したスマート・コンフィグレータによるプロジェクト作成
- QE for Capacitive Touch によるタッチインタフェース作成とチューニング、モニタリング

2. 関連ドキュメント

本アプリケーションノートでは、実際に動作するアプリケーションを作成する手順を簡単に紹介します。このアプリケーション例で使用されている各ツールに関する質問、より詳細な使用方法に関しては、

e^2 studio /スマート・コンフィグレータ、Software Integration System (SIS)のドライバ/ミドルウェア、Renesas Code Generator や QE for Capacitive Touch のヘルプ(e^2 studio のヘルプに含まれています)などのドキュメントを参照してください。

3. 静電容量タッチアプリケーション開発手順

以下は、プロジェクトにタッチセンサ検出を統合するために必要な手順の概要です。これらの手順は一般的なユーザアプリケーション開発に適用可能です。

- e^2 studio のプロジェクト作成ウィザードを使用して新規プロジェクトを作成します。
- スマート・コンフィグレータを使用して必要なモジュールを、作成したプロジェクトに追加します。
- QE for Capacitive Touch を使用して静電容量タッチインタフェースを作成します。
- QE for Capacitive Touch を使用してプロジェクトをチューニングします。
- 必要な SIS のモジュールの API コールをプロジェクトに追加し、静電容量タッチ制御を有効にします。
- QE for Capacitive Touch を使用してプロジェクトをモニタし、静電容量タッチ検出を確認します。

4. 開発ツールとソフトウェアコンポーネント

このプロジェクトでは以下の開発環境を使用します。

- RL78/G23 静電容量タッチ評価システム (RTK0EG0030S01001BJ)
  - 使用マイコン: RL78/G23 (R7F100GSN2DFB)
- 統合開発環境 e^2 studio (V2021-07 (21.7.0)以降)
  - Renesas Software Integration System (SIS) (version: 1.0.0 以降)、Renesas Code Generator
  - Renesas QE for Capacitive Touch e^2 studio plug-in (version: 2.0.0 以降)
- コンパイラ: Renesas CCRL (v1.10.00 以降)
- エミュレータ: Renesas E2 emulator Lite (RTE0T0002LKCE00000R)

5. アプリケーション例の概要

アプリケーション例のメインループの実装は以下のとおりです。

- rm_touch ミドルウェアの処理を実行すべきか判断するグローバルフラグをチェックします。
  - グローバルフラグが TRUE の場合(準備完了)
    - グローバルフラグを FALSE にリセットします。
    - rm_touch ミドルウェアの API をコールし、前回計測時のデータの処理と必要なデータを更新し、次のスキャンを開始します。
    - rm_touch ミドルウェアの API をコールし、ユーザが作成したグローバル変数に、ターゲットボード上のセンサのタッチ有無をバイナリ形式で格納します。

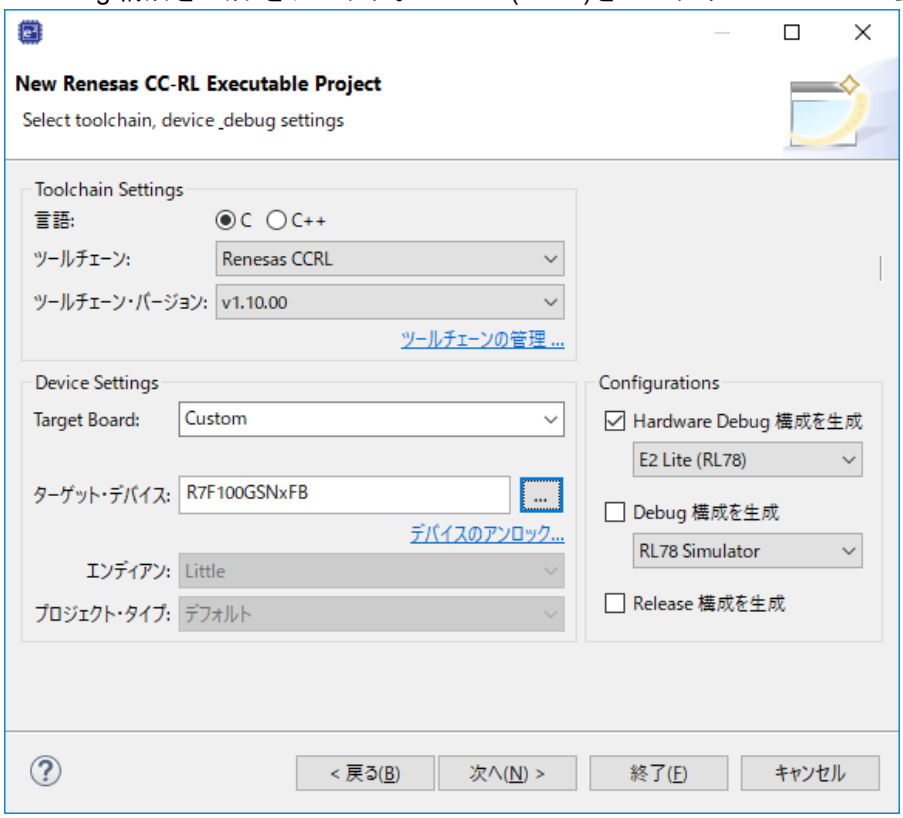
完成したアプリケーション例のコードリストに関しては「**16. qe_touch_sample.c のリスト**」章を参照してください。

6. プロジェクト作成

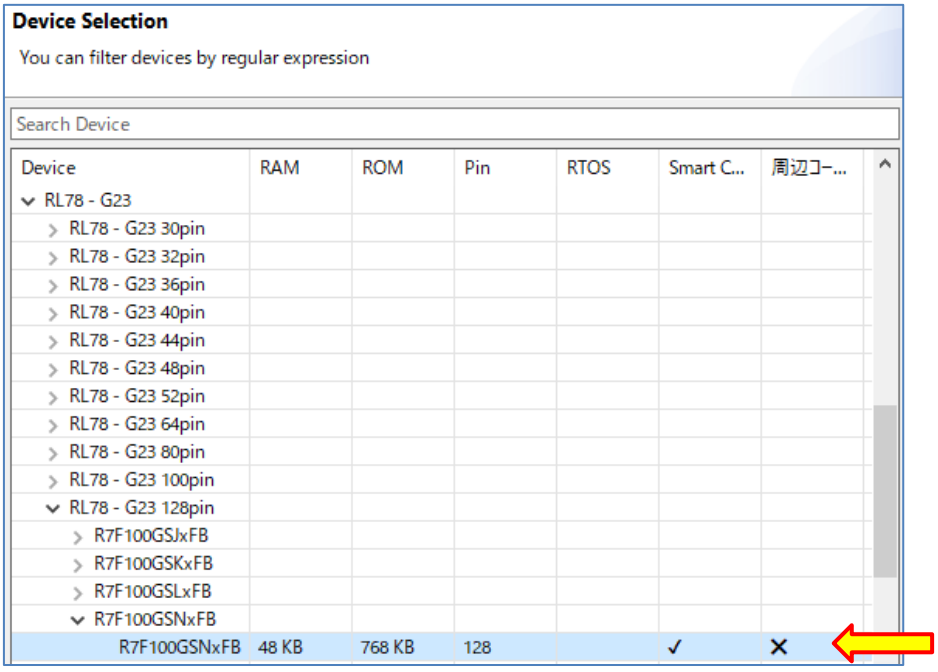
1. Windows のスタートメニューまたはデスクトップのショートカットから e² studio を起動します。ダイアログが表示されたら、任意の場所にワークスペースを作成します。
2. e² studio のメニュー[ファイル] - [新規] - [**C/C++ Project**]を選択し、新規プロジェクトの作成を開始します。
3. ダイアログが表示されたら“**Renesas RL78**” - ”**Renesas CC-RL C Executable Project**”を選択し、[**次へ**]をクリックします。
4. 次のダイアログで”**プロジェクト名(P):**”に任意のプロジェクト名を入力します。このアプリケーション例では”**Capacitive_Touch_Project_Example**”を入力します。入力完了後、[**次へ**]をクリックします。

5. 次のダイアログでは、以下を選択します。

- 言語: C
- ツールチェーン: Renesas CCRL
- ツールチェーン・バージョン: v1.10.00
- Target Board: Custom
- ターゲット・デバイス: R7F100GSNxFB
- “Hardware Debug 構成を生成”をチェック。E2 Lite (RL78)をプルダウンメニューから選択。



注.　“ターゲット・デバイス”の選択には[…]ボタンの押下で表示されるデバイスを使用します。



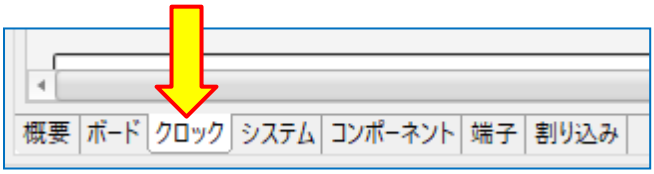
6. 完了したら、[**次へ**]をクリックします。

7. 次のダイアログが表示されたら、“**Use Smart Configurator**”をチェックし、[**終了**]をクリックしてください。

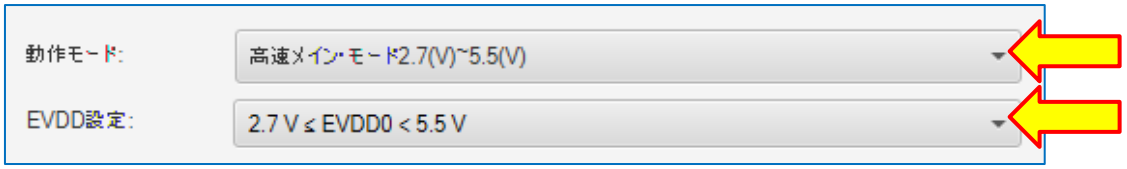
完了すると e^2 studio のデフォルトウィンドウがスマート・コンフィグレータのパースペクティブで表示され、プロジェクトの設定が可能な状態になります。これで新規プロジェクトの作成は完了です。

7. スマート・コンフィグレータによるモジュール追加

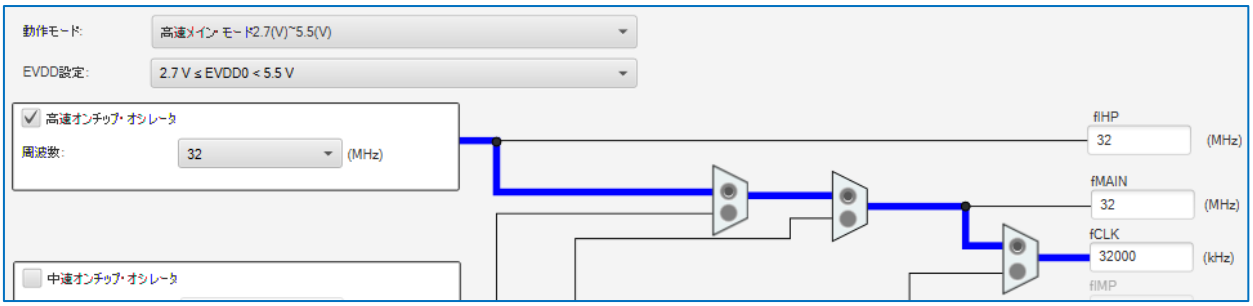
1. e² studio の中央下部のタブから[**クロック**]タブを選択し、RL78 MCU のクロック設定を表示します。



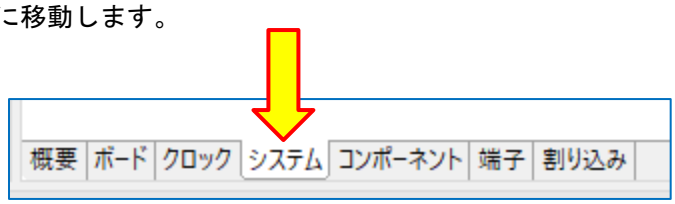
2. このアプリケーション例では、MCU の V_{DD} を 3.3 V で使用します。下の画像に示すように、“**動作モード:**”と“**EV_{DD} 設定:**”を選択します。



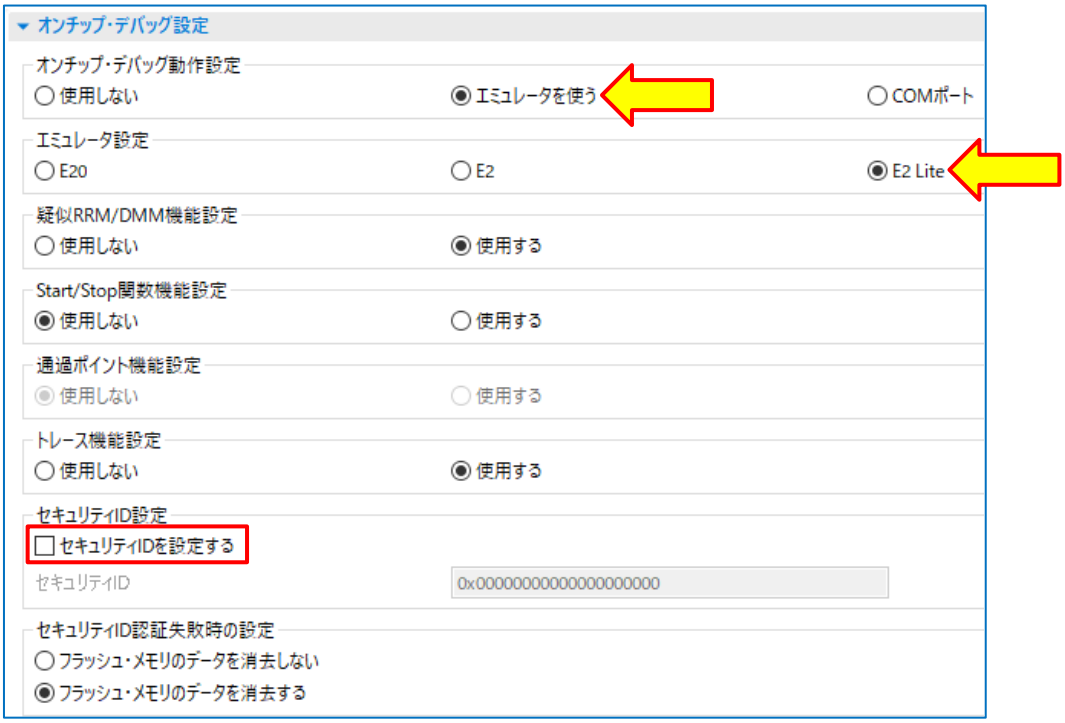
3. このアプリケーション例で使用するクロック設定を以下に示します。



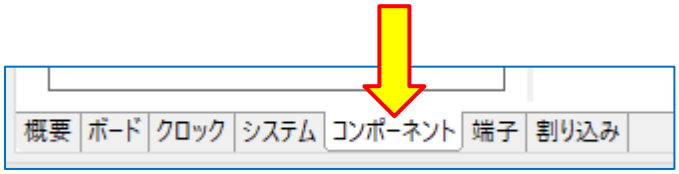
4. 次に[**システム**]タブに移動します。



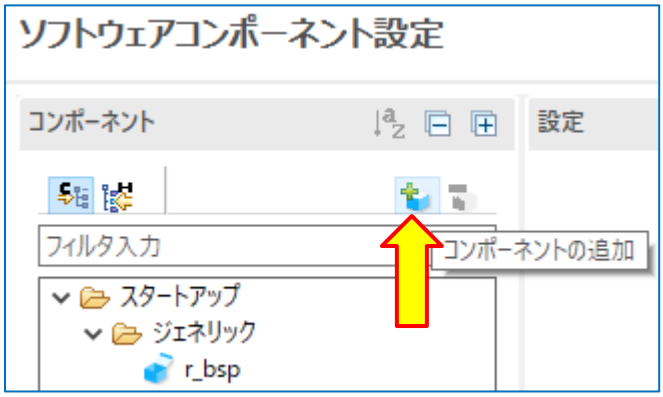
5. 以下のように“**エミュレータを使う**”および“**E2 Lite**”を選択し、“**セキュリティ ID を設定する**”のチェックを外します。



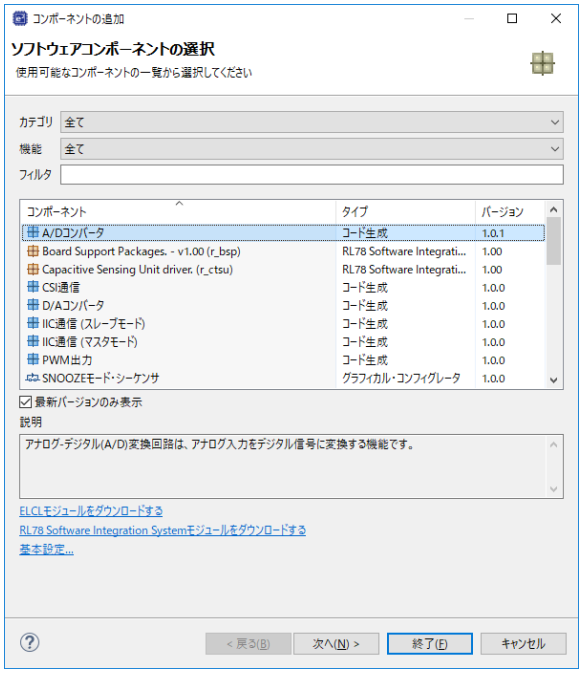
6. 次に[**コンポーネント**]タブに移動します。



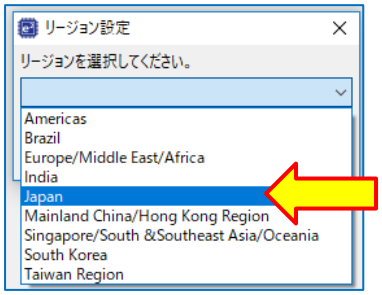
7. アプリケーションで使用するモジュールをプロジェクトに追加します。ソフトウェアコンポーネント設定の  アイコンをクリックし、モジュールを追加してください。

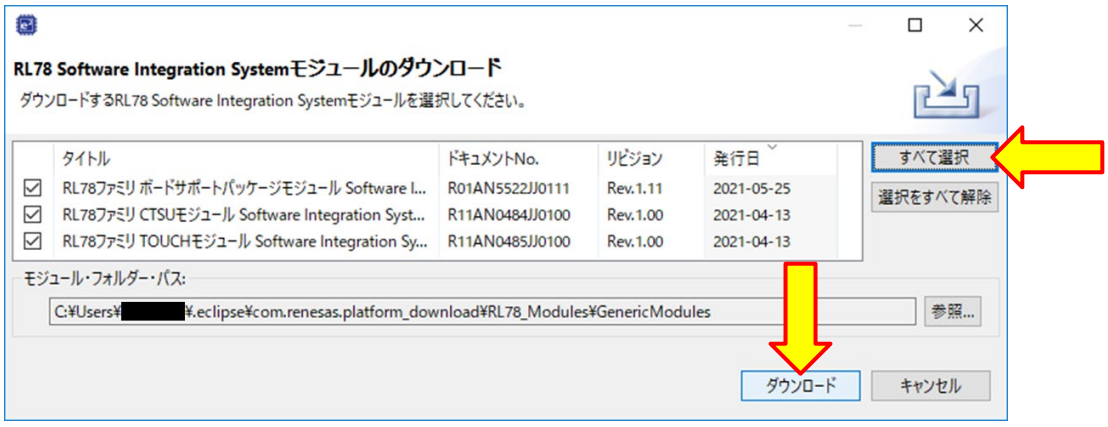


8. “ソフトウェアコンポーネントの選択”ダイアログが表示され、プロジェクトに追加可能なモジュールが表示されます。

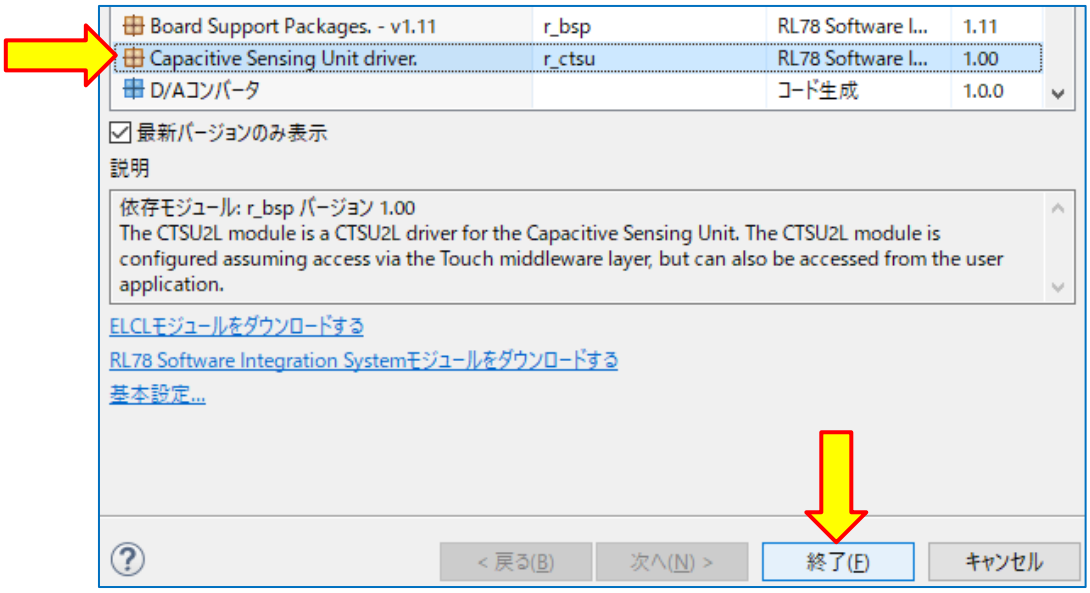


注.　SIS モジュールを初めて使用する場合は、以下の手順でダウンロードしてください。

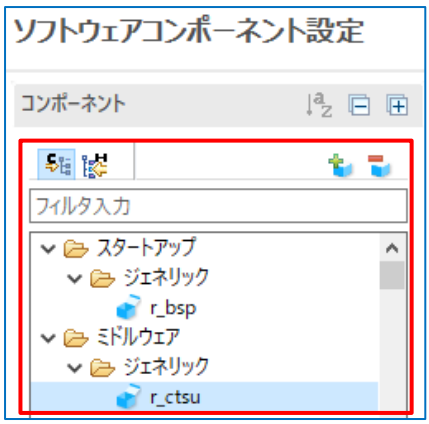




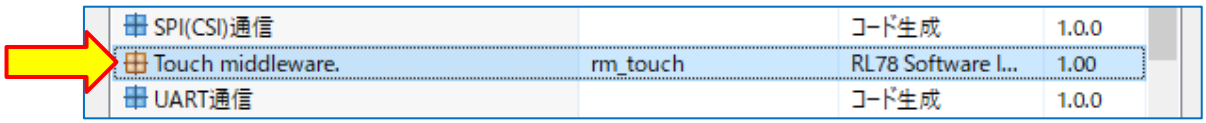
9. “**r_ctsu**”モジュールを選択し、ダイアログ下部の[**終了**]をクリックします。



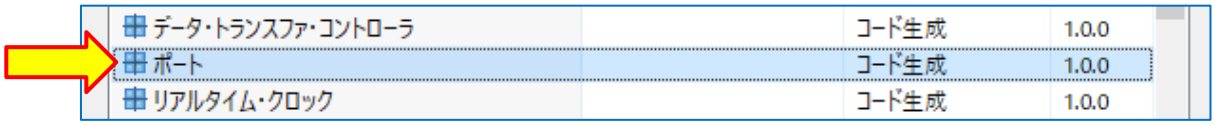
10. 下の図のように追加したモジュールがソフトウェアコンポーネント設定に表示されます。



11. 次に、**rm_touch** モジュールを前の **r_ctsu** と同じ手順で追加します。“**rm_touch**”を選択し、[**終了**]をクリックします。

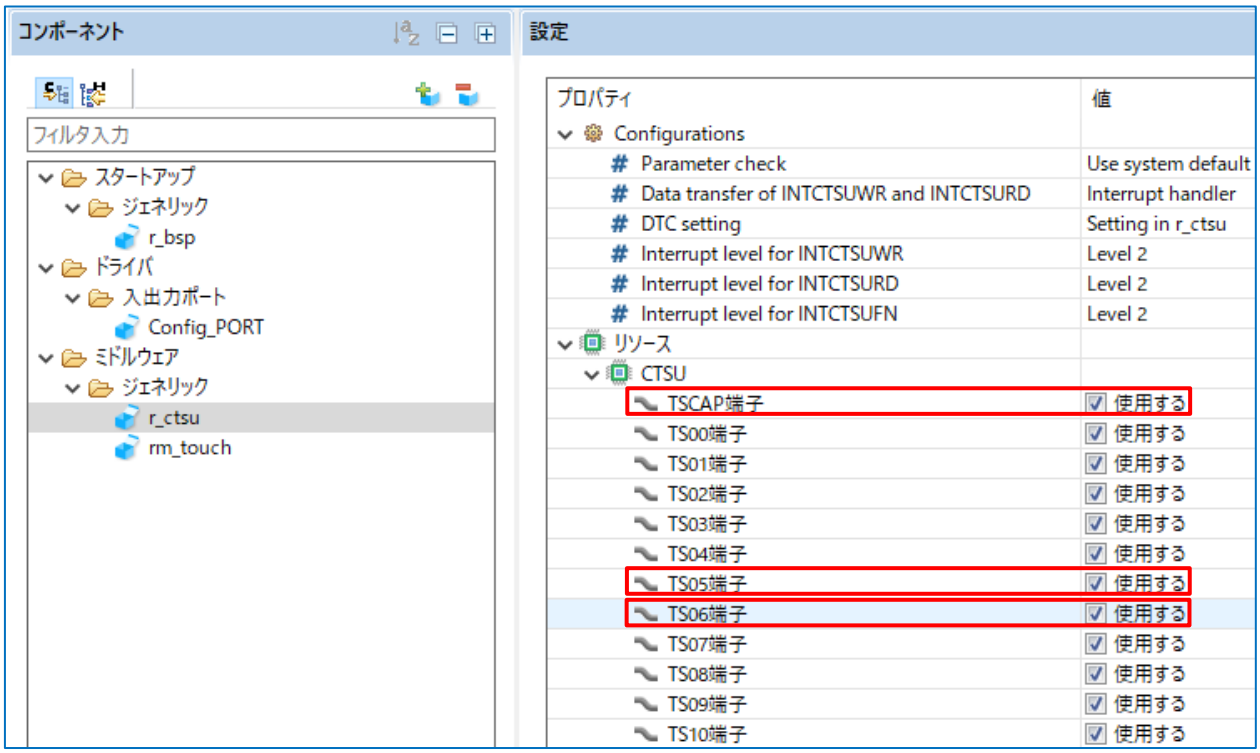


12. 最後に、**ポート**モジュールを追加します。“**ポート**”を選択し、[**終了**]をクリックします。



13. 次に、使用する TSxx 端子を有効にします。**r_ctsu** モジュールを選択し、同モジュールに関連するポートを設定パネルに表示します。このアプリケーション例では 2 つのボタン(**TS05** と **TS06**)を割り付けます。以下のセンサポートを設定パネルでクリックし、プロジェクトで使用できるようにします。

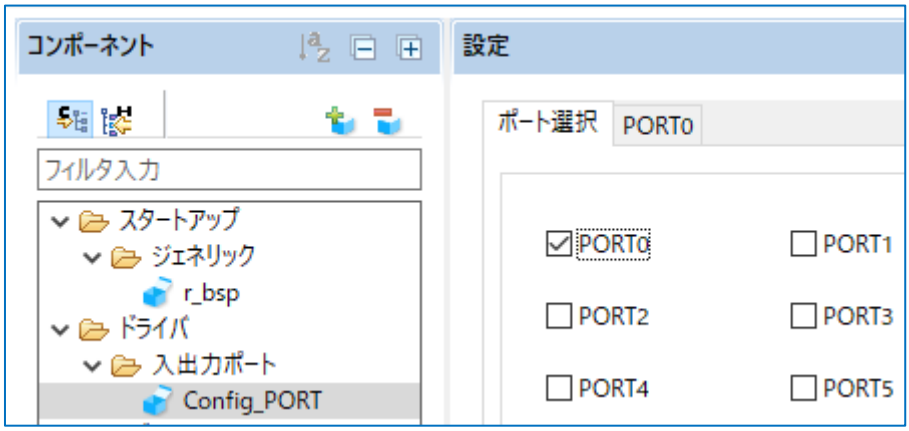
    - TSCAP 端子
    - TS05 端子
    - TS06 端子



注.　タッチアプリケーションで使用しない TSxx 端子は、ロウ・レベル出力駆動に設定することを推奨します。CTSU2 ではタッチアプリケーションで使用しない TSxx 端子を“☑ 使用する”で有効にした場合、非計測端子としてロウ・レベル出力の設定になります。実際に回路を作成される場合は、端子処理などを適切に行い、電気的特性を満たすように設計してください。

14. 次に、アプリケーションで使用しないポートをスタートアップでロウ・レベル出力駆動に設定することを推奨します。前の手順での **r_ctsu** モジュールのセンサポートを選択したのと同じように **Config_PORT** モジュールを選択し、**PORT0** のチェックボックスをクリックします。これにより、設定パネルに以下のようなタブが表示されます。

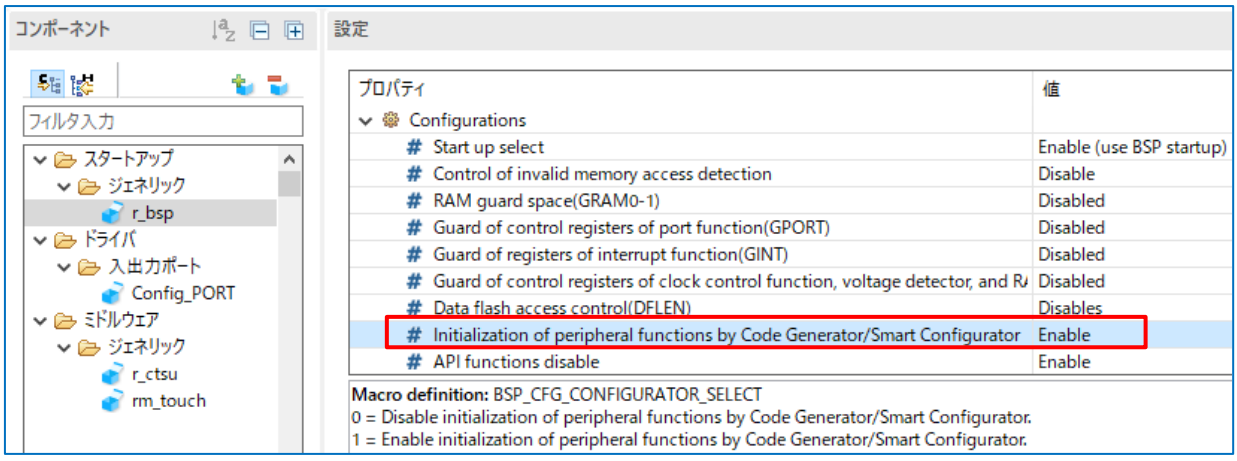
注.　ここでは使用例として一部のポートを設定しています。実際に回路を作成される場合は、端子処理などを適切に行い、電気的特性を満たすように設計してください。



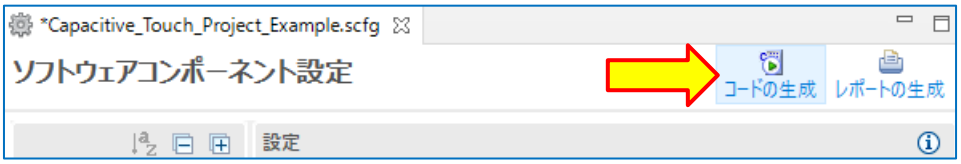
15. **PORT0** タブを選択し、P07 を“**出力**”に設定します。設定パネルは以下の図のようになります。

注.　ここでは使用例として一部のポートを設定しています。実際に回路を作成される場合は、端子処理などを適切に行い、電気的特性を満たすように設計してください。

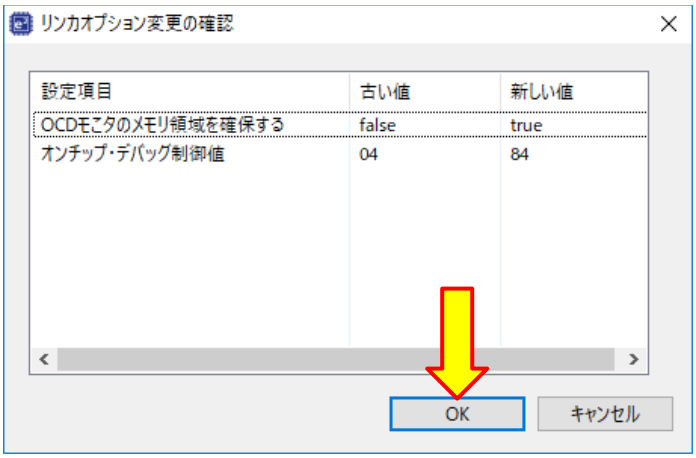
16. 以下のように、“**Initialization of peripheral functions by Code Generator/Smart Configurator**”が“**Enable**”に設定されていることを確認します。



17. 以下の図のように、スマート・コンフィグレータの右上の アイコンをクリックして、プロジェクトに必要なモジュールのコードを追加します。



注.　スマート・コンフィグレータでオンチップ・デバッグ設定またはオプション・バイト設定を変更した場合、以下のメッセージが表示されます。 変更内容を確認した後に、[OK]をクリックします。



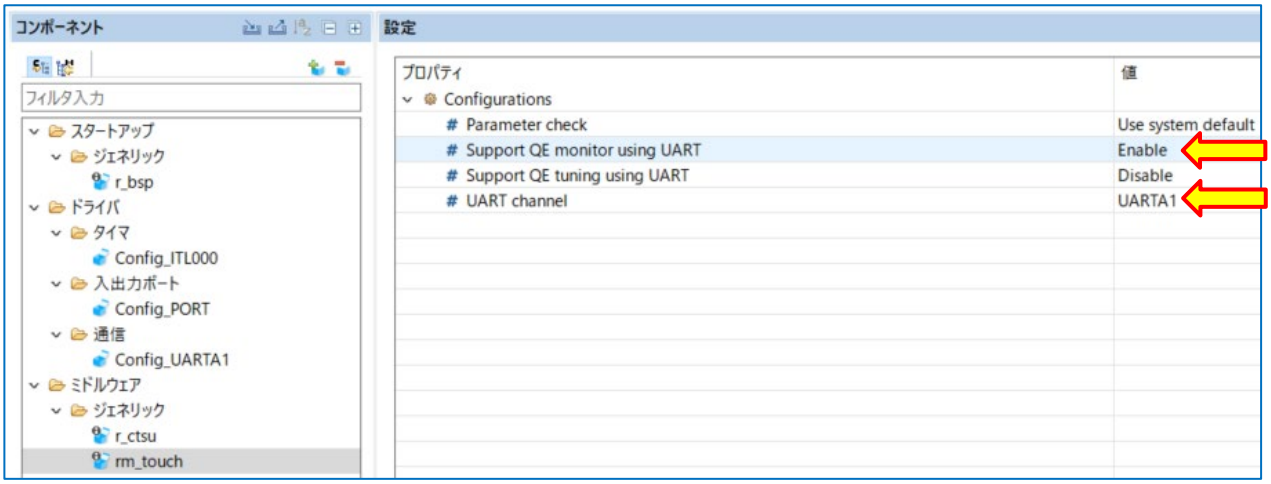
8.　[追加機能] UART を使用したシリアル通信モニタの設定 (1/3)

注.　タッチアプリケーションのタッチ性能のモニタリングは、OCD(On-Chip Debugging)エミュレータを介した通信によって確認できます。ただし、RL78 ファミリの場合、モニタリングパフォーマンスは RL78 ファミリの OCD 機能によって制限されます。
また一方で、タッチ性能のモニタリングは、シリアル通信を介して行うこともできます。 したがって、スムーズにモニタリングを行いたい場合は、シリアル通信を介したモニタリング機能を追加してください。

以下に示す 8 章、13 章および 15 章(本章を含む)では、UART を使用したシリアル通信モニタの設定について説明します。

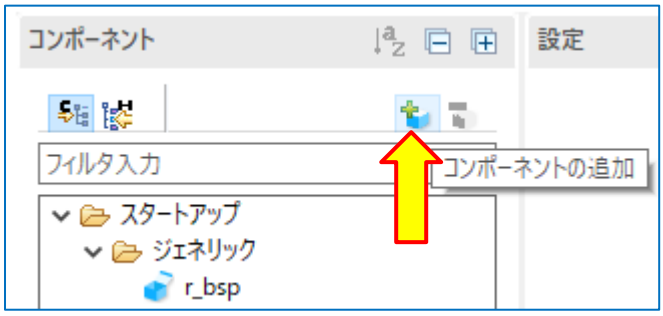
- 8. [追加機能] UART を使用したシリアル通信モニタの設定 (1/3)
- 13. [追加機能] UART を使用したシリアル通信モニタの設定 (2/3)
- 15. [追加機能] UART を使用したシリアル通信モニタの設定 (3/3)

1. 以下のように、[**コンポーネント**]タブで、“**rm_touch**”モジュールを選択し、“**Support QE monitor using UART**”を“**Enable**”に、“**UART channel**”を“**UARTA1**”に、設定します。

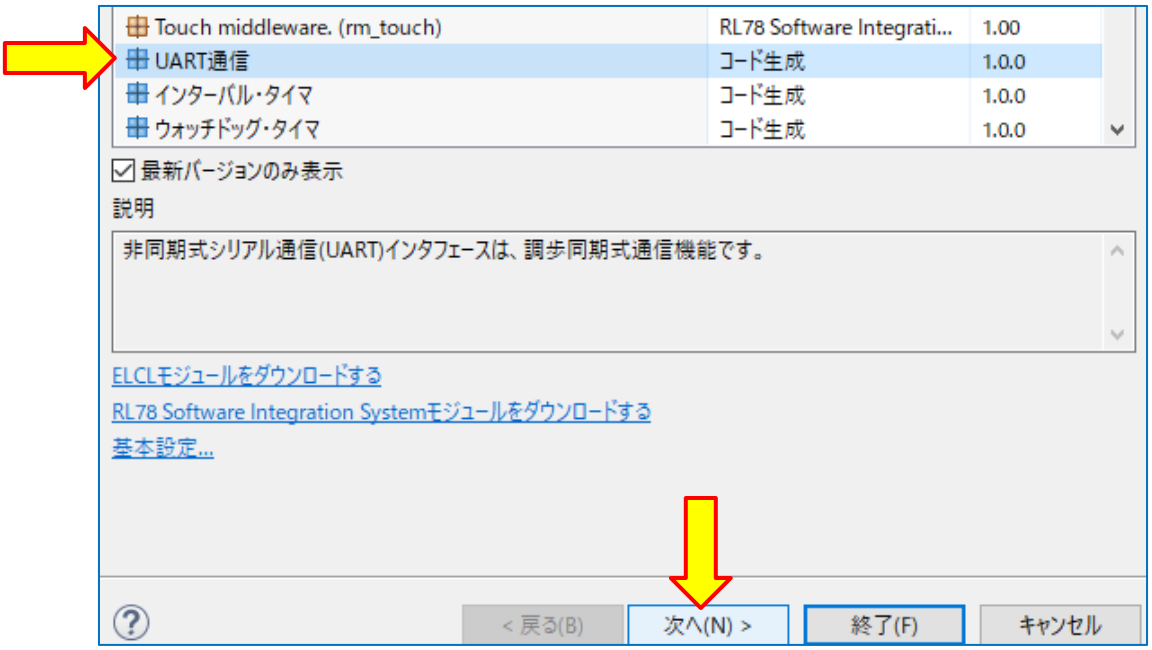


注.　ツールで設定する UART チャネルおよびポートは、使用するターゲットボードによって異なります。

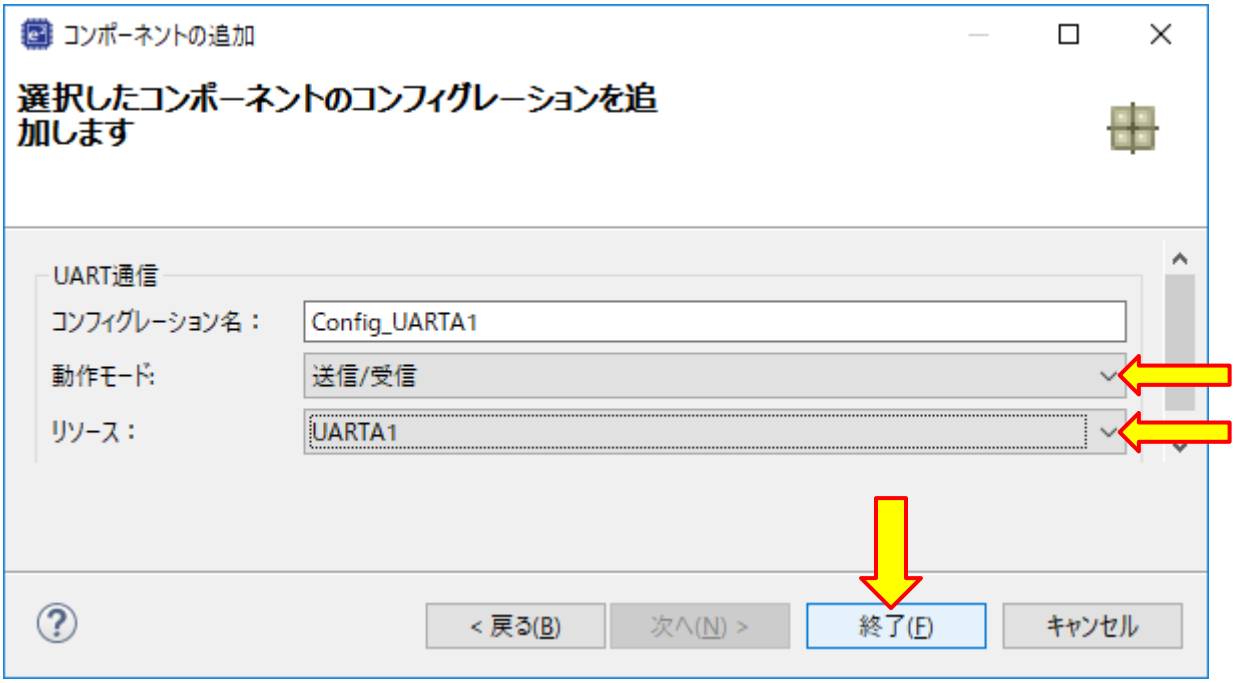
2. ソフトウェアコンポーネント設定の アイコンをクリックし、モジュールを追加してください。



3. “**UART 通信**”モジュールを選択し、ダイアログ下部の[**次へ**]をクリックします。



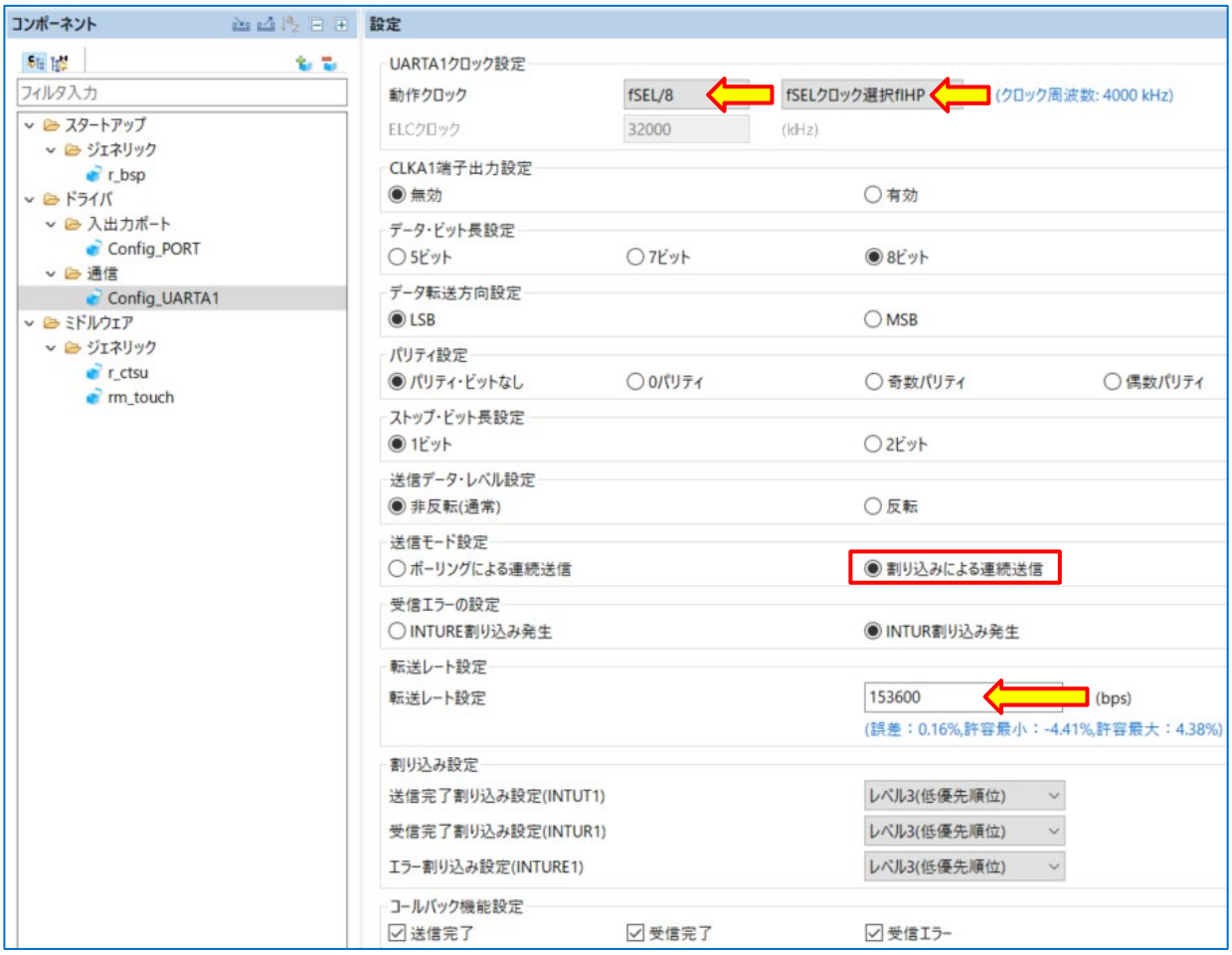
4. “**動作モード:** ”を“**送信/受信**”に、“**リソース:** ”を“**UARTA1**”に設定し、ダイアログ下部の[**終了**]をクリックします。



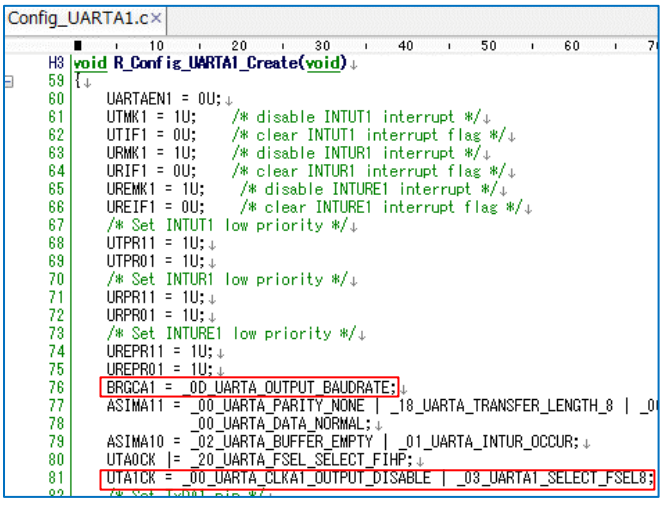
注.　ツールで設定する UART チャネルおよびポートは、使用するターゲットボードによって異なります。

5. [**コンポーネント**]タブで、“**Config_UARTA1**”モジュールを選択します。

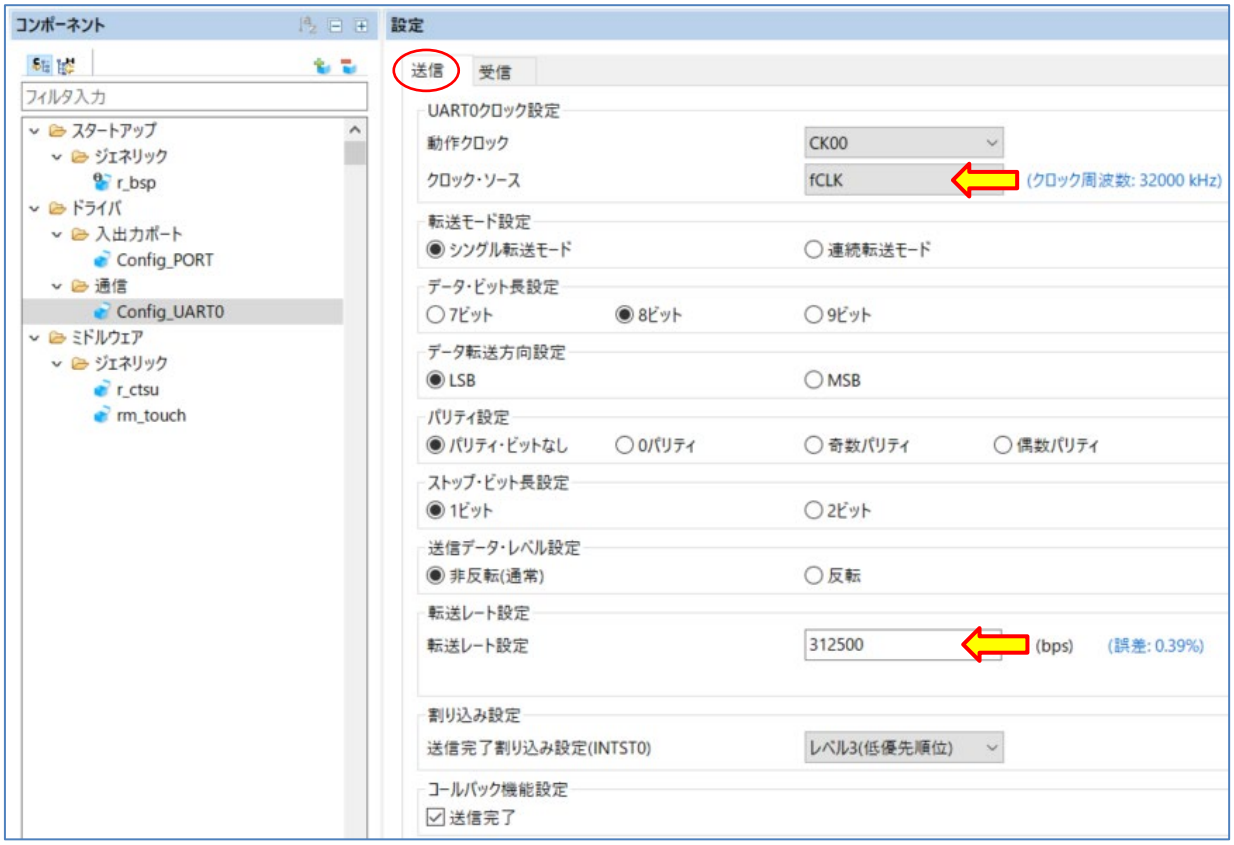
次に、“**動作クロック**”を“**fSEL/8**”と“**fSEL クロック選択 fIHP**”に、“**割り込みによる連続送信**”を選択し、“**転送レート設定**”を“**153600**”(bps)に設定します。

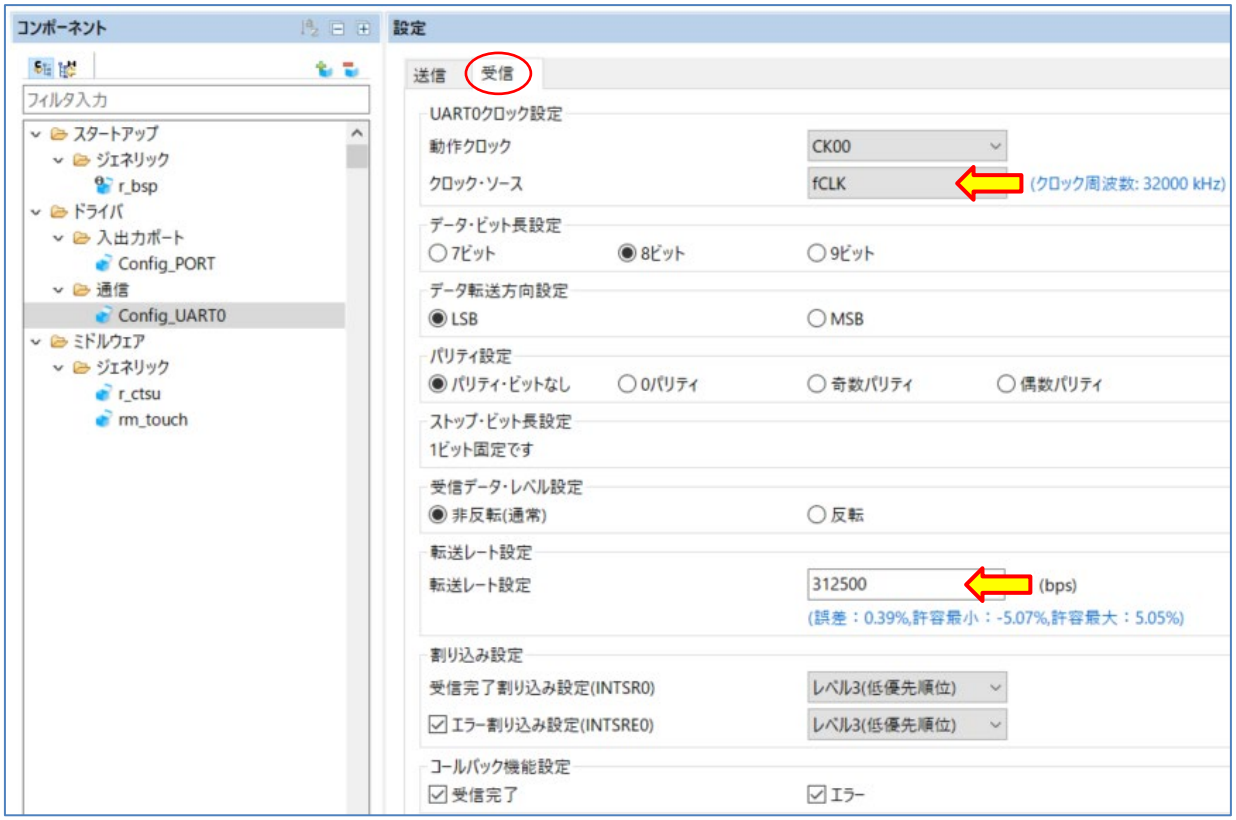


注 1. スマート・コンフィグレータのバージョンによっては、“転送レート設定”でエラーが発生することがあります。このエラーが発生した場合は、コード生成後に下記赤枠部分のプログラムを変更して転送レートを設定してください。

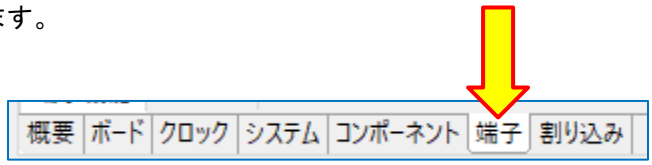


注 2.　ツールで設定する UART チャネルおよびポートは、使用するターゲットボードによって異なります。例えば、UART0 を使用する場合、下記画像のように設定が異なります。

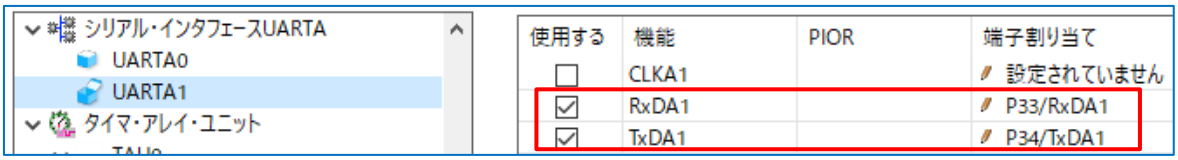




6. [**端子**]タブに移動します。

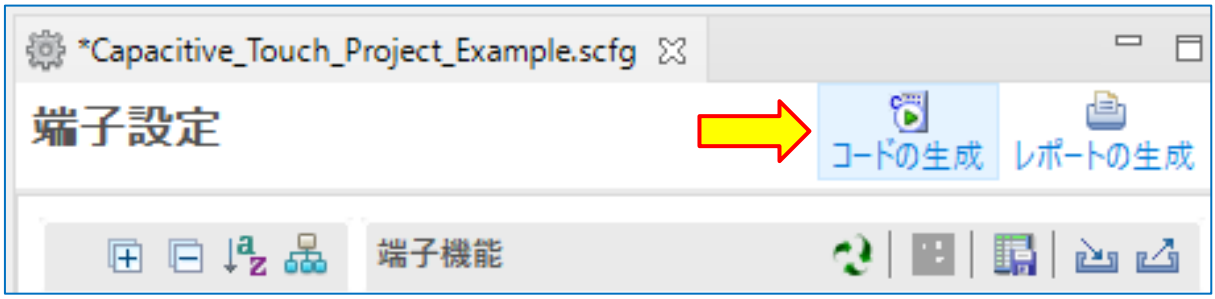


7. “**RxDA1**”機能を“**P33**”に、“**TxDA1**”機能を“**P34**”に割り当てます。



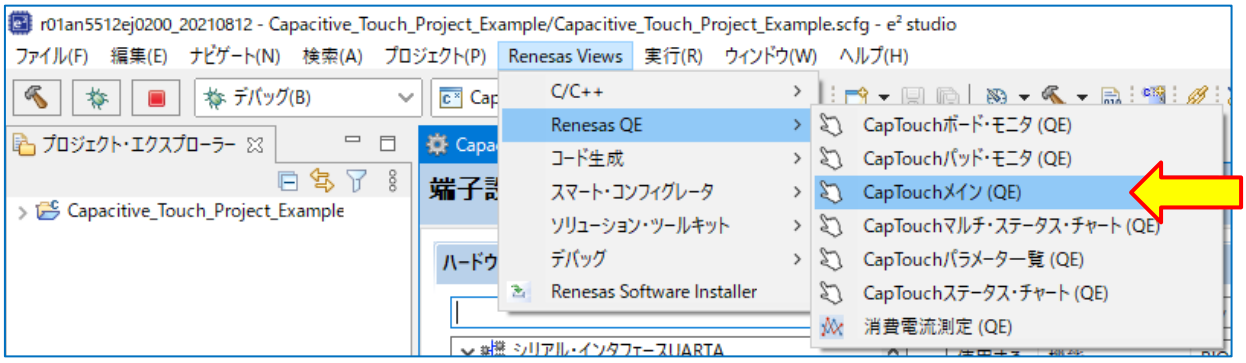
注.　ツールで設定する UART チャネルおよびポートは、使用するターゲットボードによって異なります。例えば、UART0 を使用する場合、下記画像のように設定が異なります。

8. 以下の図のように、スマート・コンフィグレータの右上の  アイコンをクリックして、プロジェクトに必要なモジュールのコードを追加します。

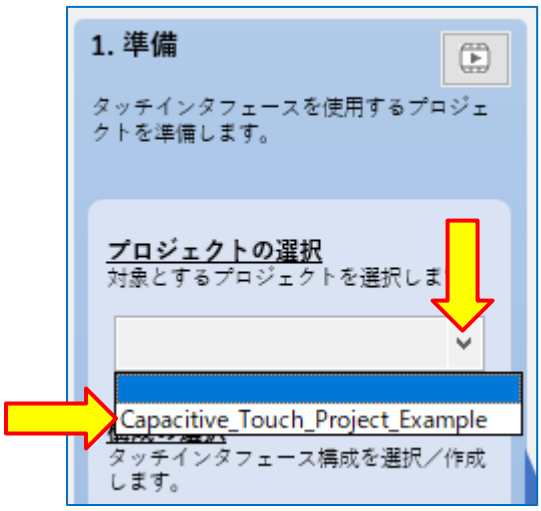


9. 静電容量タッチインタフェース作成

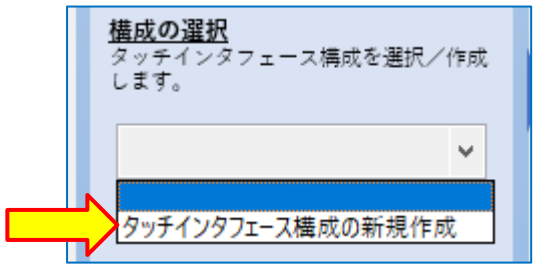
1. e² studio のメニュー[**Renesas Views**] - [**Renesas QE**] - [**CapTouch メイン (QE)**]／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]を選択し、プロジェクトに静電容量タッチの設定をするためのメインウィンドウを表示します。



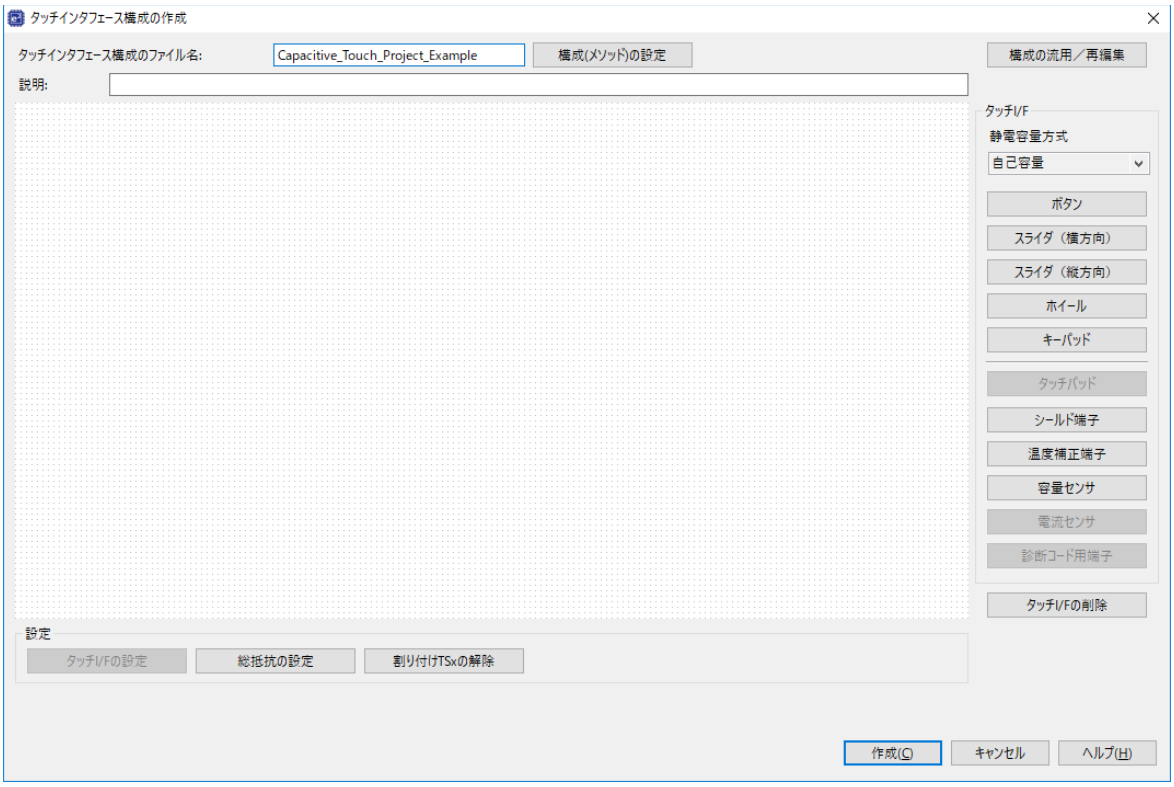
2. “**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”の“プロジェクトの選択”のプルダウンメニューから”**Capacitive_Touch_Project_Example**”を選択し、タッチインタフェースを設定するプロジェクトを選択します。



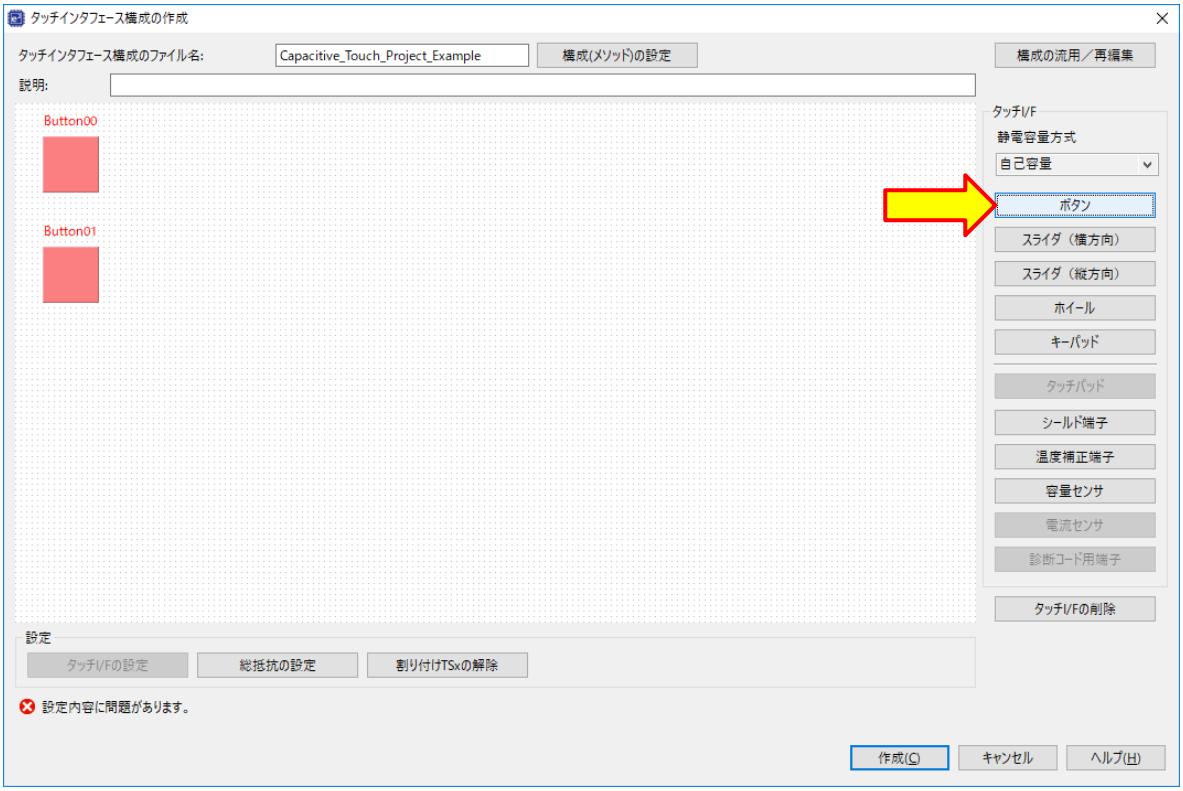
3. 次に、“構成の選択”のプルダウンメニューから[**タッチインタフェース構成の新規作成**]を選択し、新しいタッチインタフェース構成を生成します。



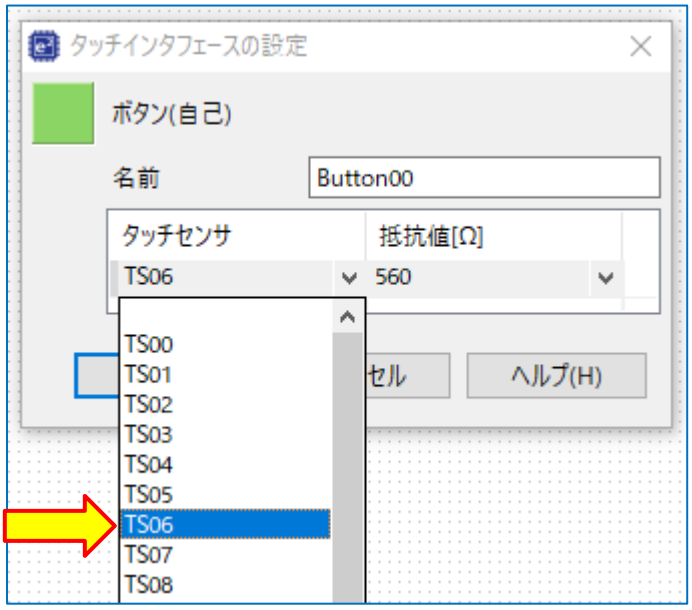
4. “タッチインタフェース構成の作成”ウィンドウが開き、タッチインタフェースを配置する領域が表示されます。



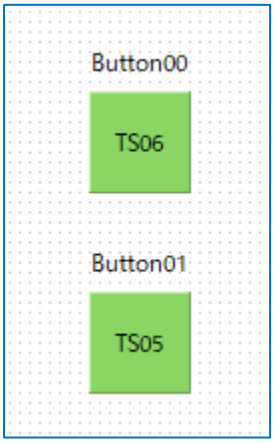
5. “タッチインタフェース構成の作成”ウィンドウの右側から[**ボタン**]を選択して 2 つのボタンをタッチインタフェースの配置領域に追加します。2 つのボタンを追加し、キーボードの[**ESC**]キーを押してタッチインタフェースの追加を終了すると以下のようになります。



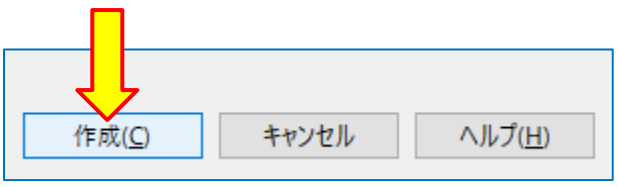
6. “**Button00**”をダブルクリックし、“タッチインタフェースの設定”ダイアログを表示します。ここではプルダウンメニューから、このボタンに割り当てる MCU のセンサポートに **TS06** を選択します。



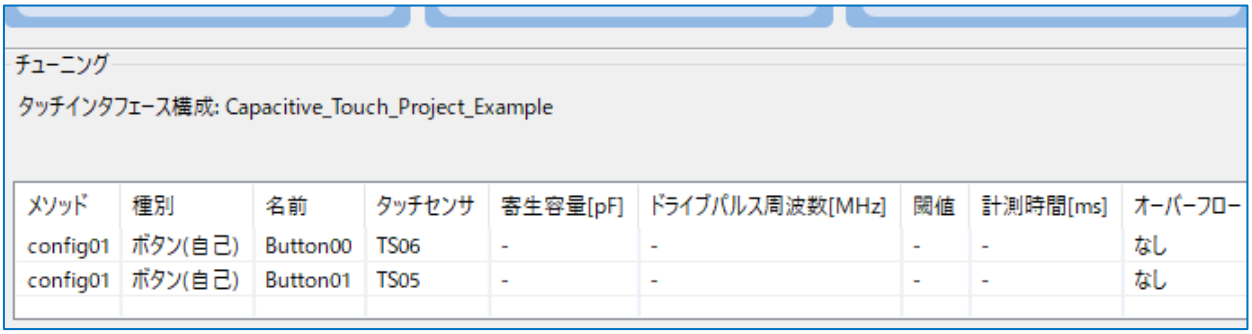
7. “**Button00**”に対する前の手順と同じように、“**Button01**”に **TS05** を割り当てます。タッチインタフェースの配置領域は以下のようになります。スマート・コンフィグレータで有効にしたセンサポートに従って、割り付けが正しく設定されると設定エラーの表示がなくなります。



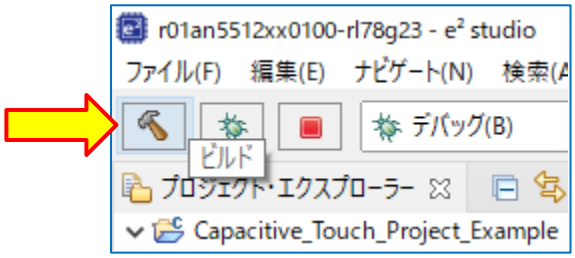
8. “タッチインタフェース構成の作成”ウィンドウの[**作成**]をクリックします。これでタッチインタフェースが設定されます。



9. “**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”の“チューニング”パネルにタッチインタフェースの構成が表示されます。

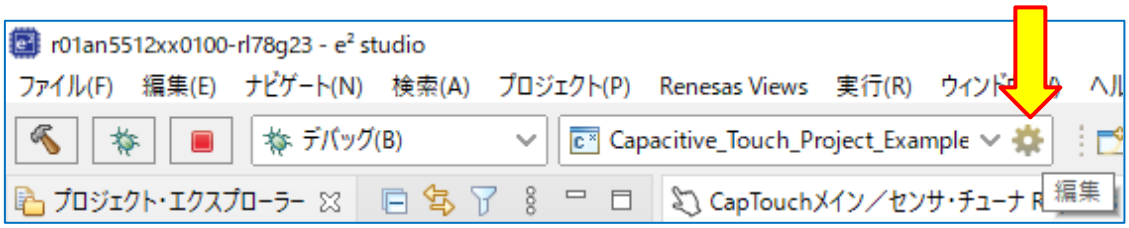


10. e² studio 左上の アイコンをクリックしてビルドを開始します。プロジェクトはエラーやワーニングなしでビルドされます。

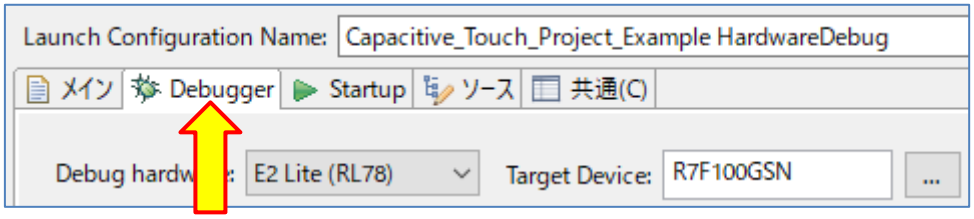


10. 静電容量タッチセンサ・チューニング向けデバッグ構成の設定変更

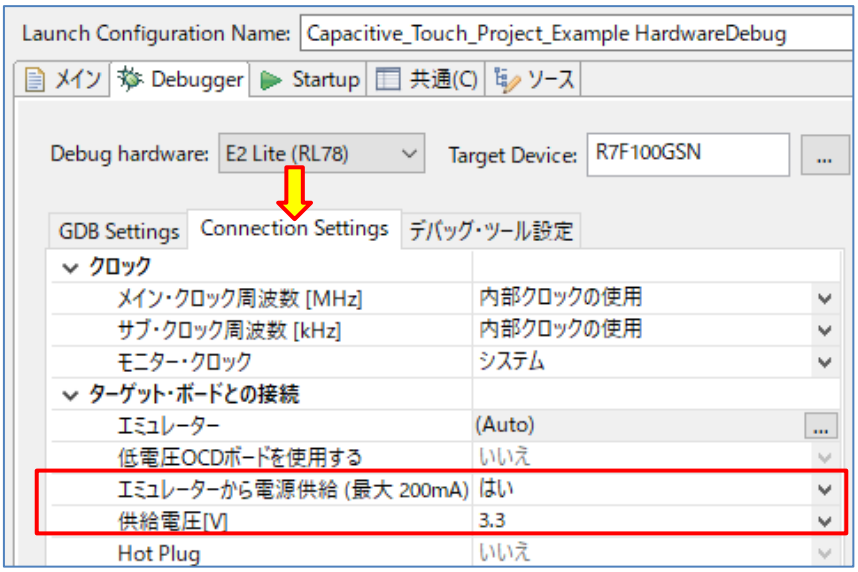
1. デバッグセッション開始後にチューニングカーネルを MCU の RAM にダウンロードできるように、デバッグ構成を変更する必要があります。 アイコンをクリックし、デバッグ構成を選択してください。



2. 表示されたパネルから“**Debugger**”タブを選択します。

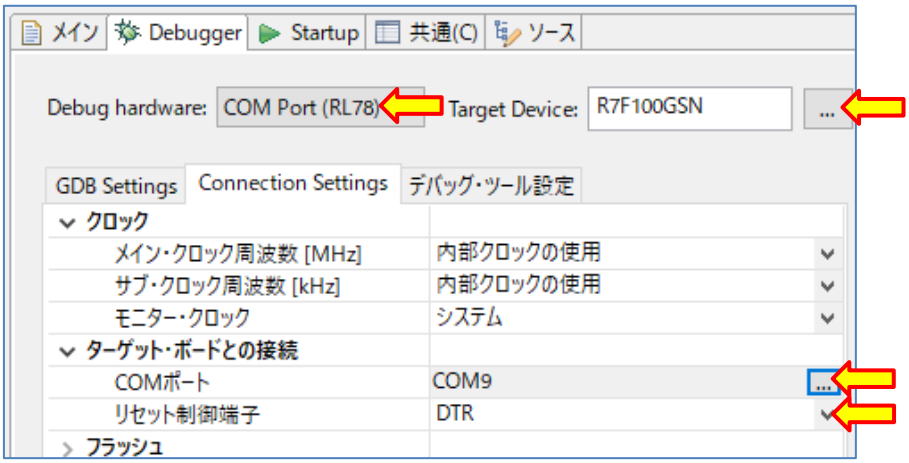


3. “**Connection Settings**”タブを選択します。このアプリケーション例では、ターゲットボードの電源はエミュレータの電源から供給します。“エミュレータから電源供給(最大 200mA)”と“供給電圧[V]”が以下の赤枠部分のように設定されていることを確認します。

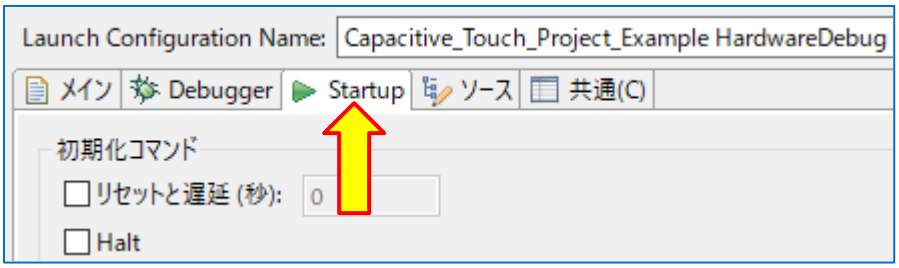


注 1.　動作確認を簡単に行うために、このアプリケーション例では、ターゲットボードの電源はエミュレータの電源から供給します。PC の USB ポートから、E2 emulator Lite を経由してターゲットボードに電源を供給することは可能ですが、ルネサスではターゲットボードで生成する電源を使用することを推奨しています。

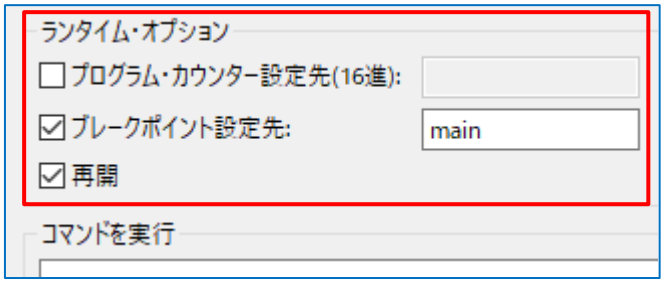
注 2.　どのデバッグ方法が使用可能かは、ターゲットボードの仕様によります。使用するデバッグ方法に応じて“Debug hardware:”を選択し、項目を設定してください。例えば、COM port デバッグを行う場合、下記画像のように設定が異なります。



4. “**Startup**”タブを選択します。



5. “**ブレークポイント設定先:**”と“**再開**”のチェックボックスが以下のようにチェックされていることを確認します。これらのチェックボックスを表示するにはダイアログを下にスクロールする必要があります。

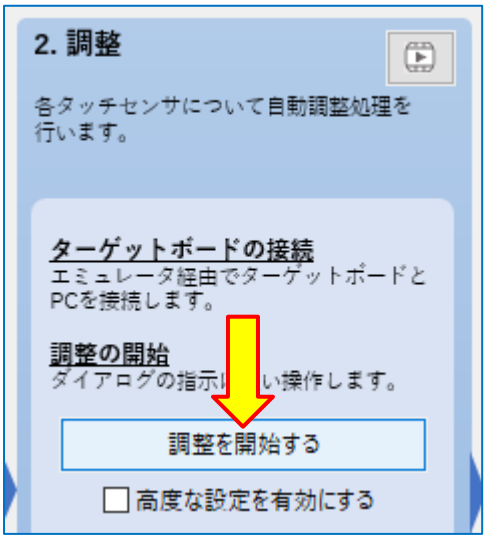


6. [**OK**]をクリックし、変更した設定を有効にします。これでチューニングのためのプロジェクトの設定とデバッグ構成の設定は終了です。

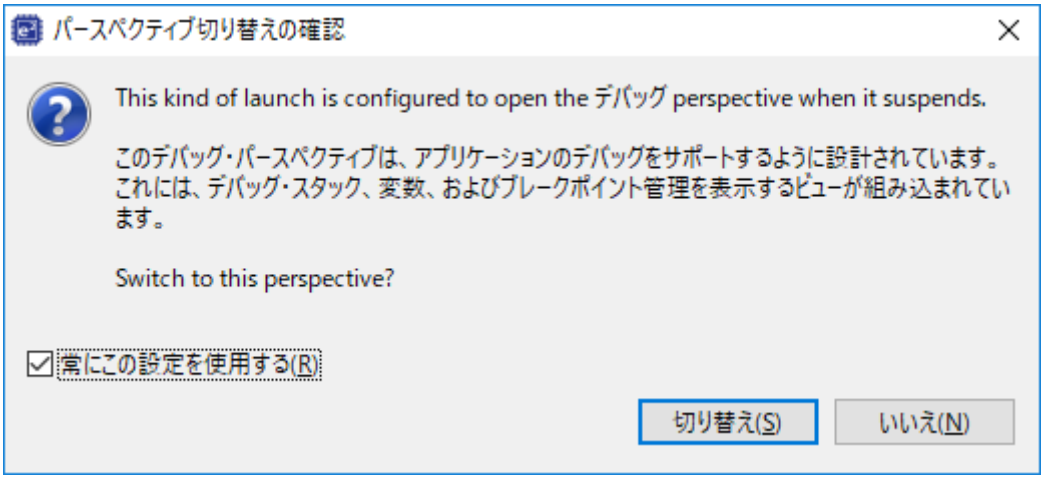
11. QE for Capacitive Touch を使用した静電容量タッチセンサ・チューニング

1. “**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”の[**調整を開始する**]をクリックし、自動チューニングを開始します。

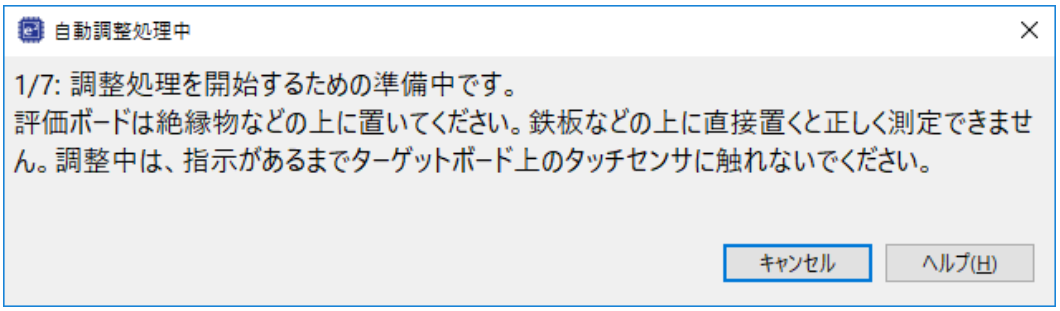
注.　動作確認を簡単に行うために、このアプリケーション例では、ターゲットボードの電源はエミュレータの電源から供給します。PC の USB ポートから、E2 emulator Lite を経由してターゲットボードに電源を供給することは可能ですが、ルネサスではターゲットボードで生成する電源を使用してチューニングすることを推奨しています。



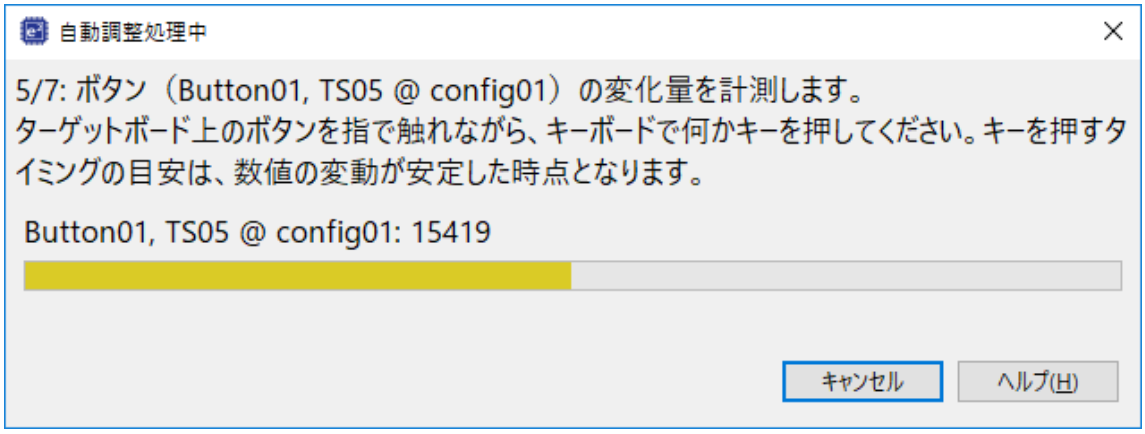
2. デバッグセッションの開始時、e² studio はデバッグ・パースペクティブに切り替える旨のメッセージを表示することがあります。[**常にこの設定を使用する(R)**]をチェックし、[**切り替え**]をクリックしてデバッグセッションと QE for Capacitive Touch の自動チューニングを続行してください。



3. QE for Capacitive Touch の自動チューニングが開始されます。チューニングプロセスをガイドする”自動調整処理中”ダイアログを適宜、確認してください。表示例を以下に示します。通常、初期のチューニングプロセス中は操作を必要としません。

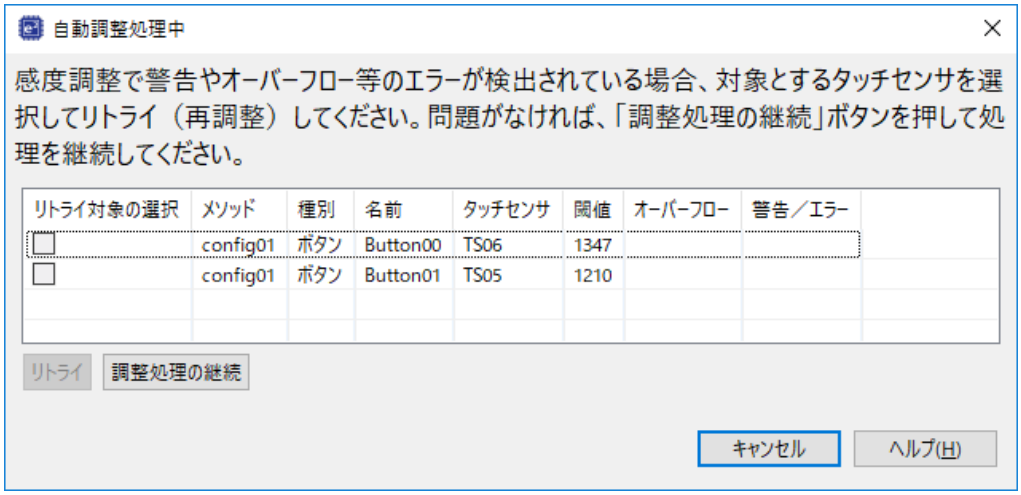


いくつかの工程を経て、以下のようなダイアログが表示されます。ここではチューニングプロセスにおけるタッチ感度の計測をします。ダイアログで表示されているセンサ(**Button01**/**TS05**)を通常の圧力でタッチします。センサに触れているとき、バーグラフは右に増加し、数値で示すタッチカウント値が増えます。センサに触れたまま、PC のキーボードのいずれかのキーを押して計測を確定します。

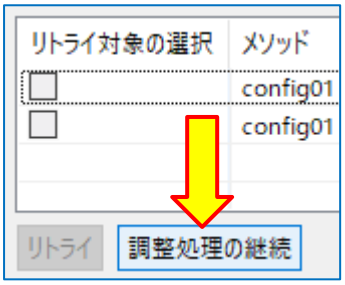


4. **Button00**/**TS06** に対して、前の手順を繰り返します。

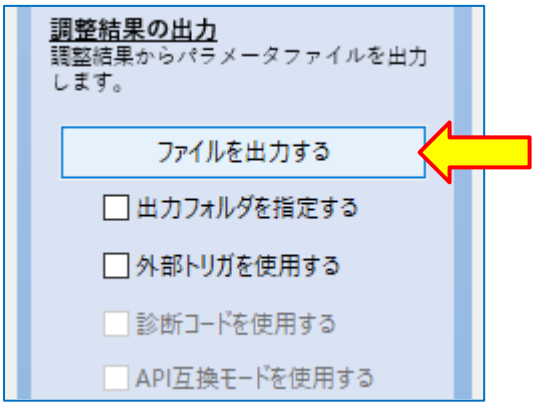
5. チューニングが完了すると、以下のようなダイアログが表示され閾値を確認できます。この閾値はミドルウェアでタッチのイベント判定に使用されます。



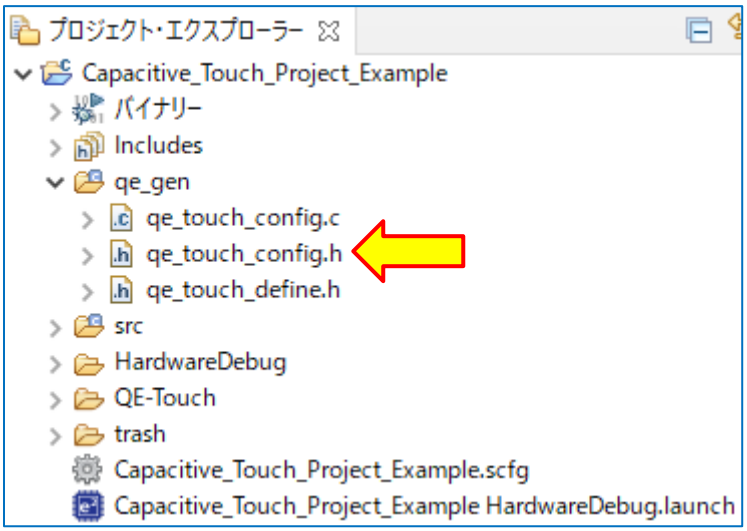
6. 表示されたダイアログの[**調整処理の継続**]をクリックします。これでチューニングプロセスは終了し、ターゲットボードとのデバッグセッションを切断します。“**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”に戻ります。



7. 残る手順はチューニングされたパラメータファイルの出力だけです。[**ファイルを生成する**]をクリックします。



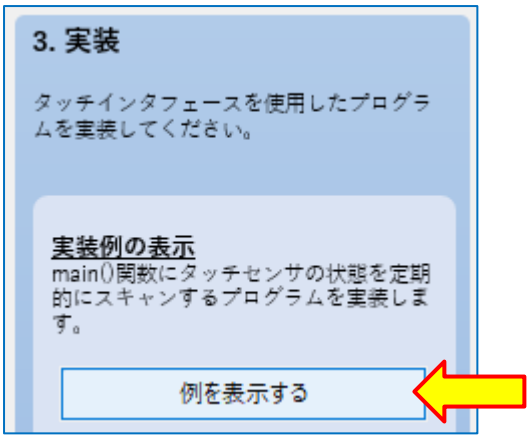
8. “**プロジェクト・エクスプローラー**”ウィンドウで **qe_touch_config.c** と **qe_touch_config.h**、**qe_touch_define.h** が追加されたことを確認できます。これらのファイルにはドライバを使用したタッチ検出を有効にするために必要なチューニング情報が含まれています。



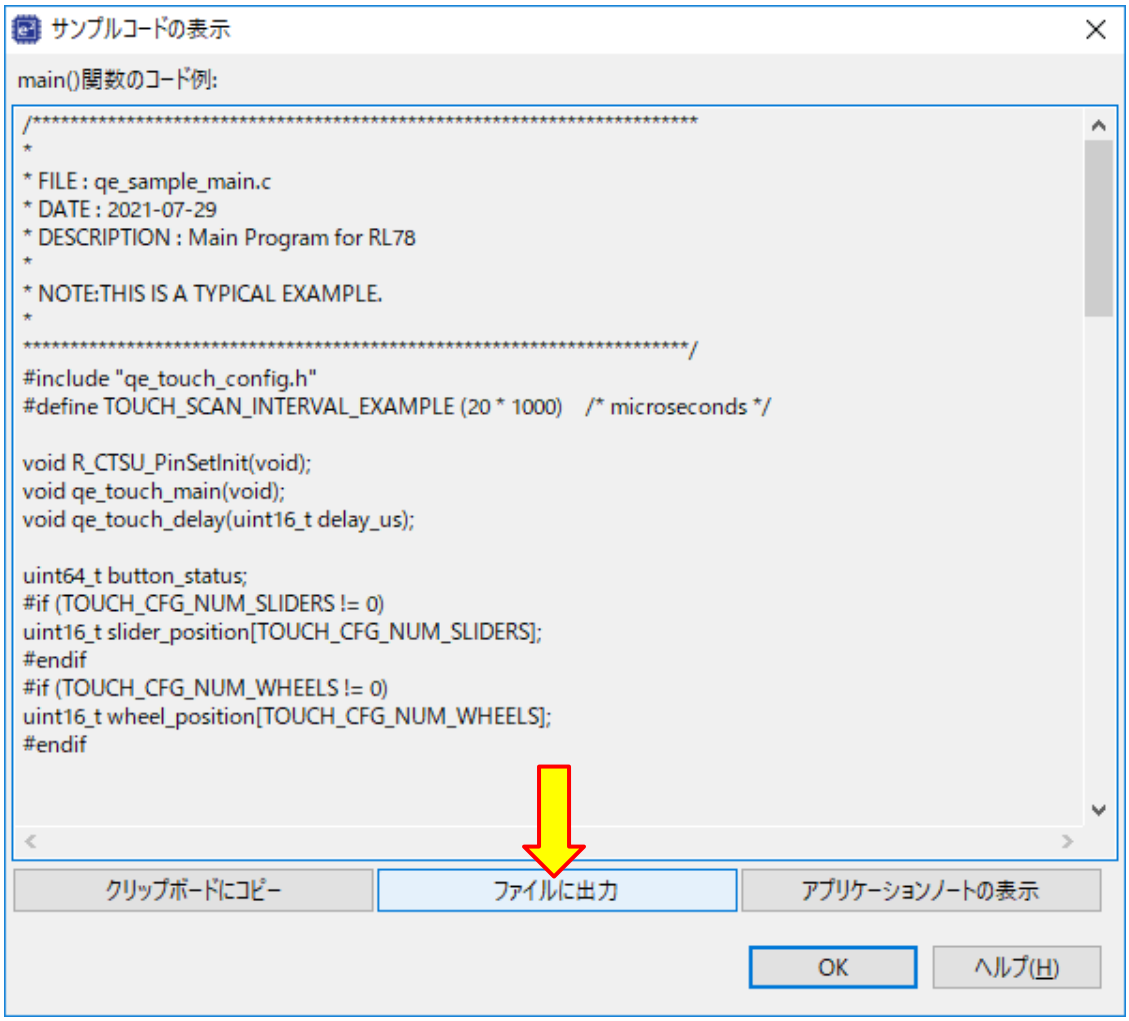
9. e² studio の左上の アイコンをクリックしてプロジェクトをビルドします。“コンソール”ウィンドウで、ビルドした結果にエラーがないことが確認できます。

12. アプリケーションに rm_touch ミドルウェアの API コールを追加

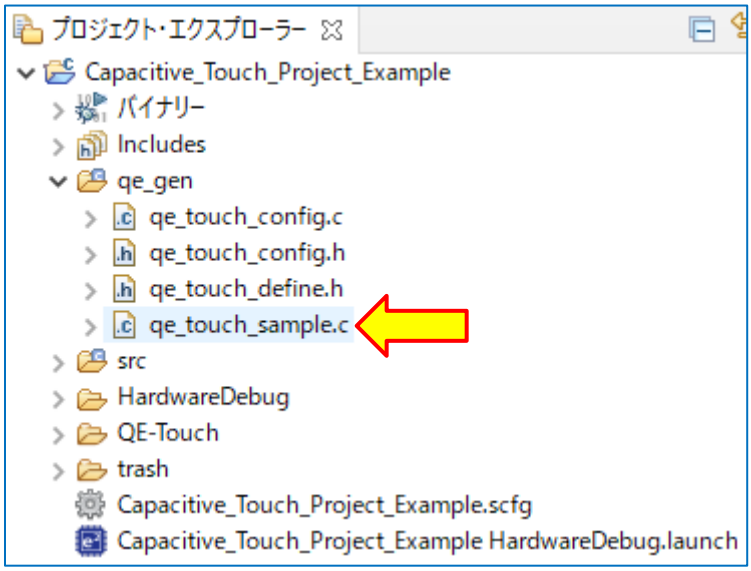
1. タッチセンサの状態をスキャンするプログラムを実装するために、“**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”の[**例を表示する**]をクリックします。



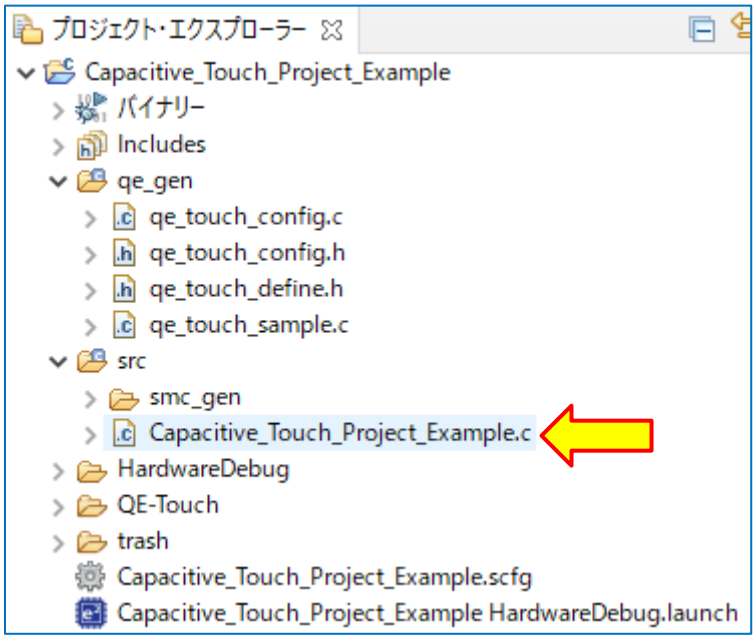
2. “サンプルコードの表示”ウィンドウが開き、サンプルコードが表示されます。サンプルコードを出力するために[**ファイルに出力**]をクリックしてください。



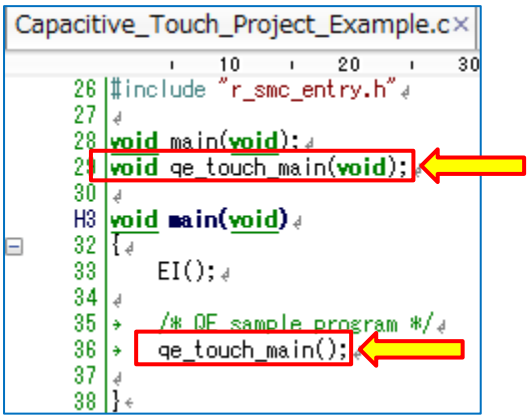
3. “プロジェクト・エクスプローラー”より“**qe_touch_sample.c**”ファイルが生成されたことを確認してください。



4. “**Capacitive_Touch_Project_Example.c**”ファイルを開きます。



5. main()関数から qe_touch_main()関数をコールします。“**Capacitive_Touch_Project_Example.c**”ファイルへ下記画像のように赤枠部分のコード(“**void qe_touch_main(void);**”および“**qe_touch_main();**”)を追加してください。



6. これで、アプリケーション例で必要なコードの変更はすべて終了です。アプリケーション例のプロジェクトをビルドすると、エラーまたはワーニングなしで終了することを確認できます。

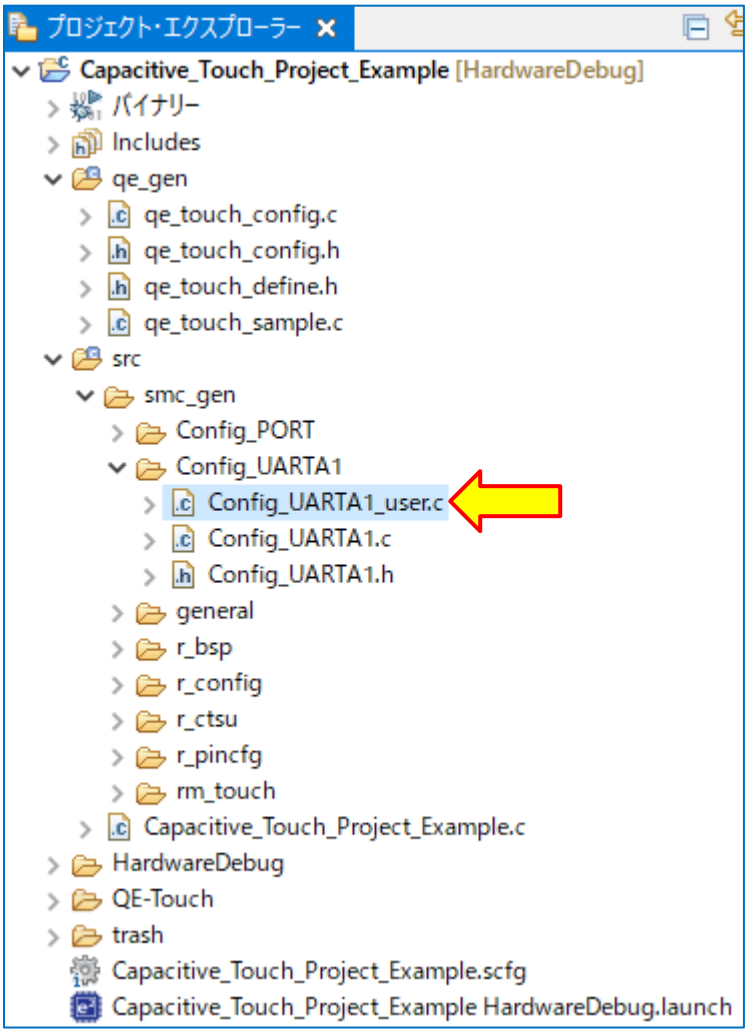
13. [追加機能] UART を使用したシリアル通信モニタの設定 (2/3)

注.　タッチアプリケーションのタッチ性能のモニタリングは、OCD(On-Chip Debugging)エミュレータを介した通信によって確認できます。ただし、RL78 ファミリの場合、モニタリングパフォーマンスは RL78 ファミリの OCD 機能によって制限されます。
また一方で、タッチ性能のモニタリングは、シリアル通信を介して行うこともできます。 したがって、スムーズにモニタリングを行いたい場合は、シリアル通信を介したモニタリング機能を追加してください。

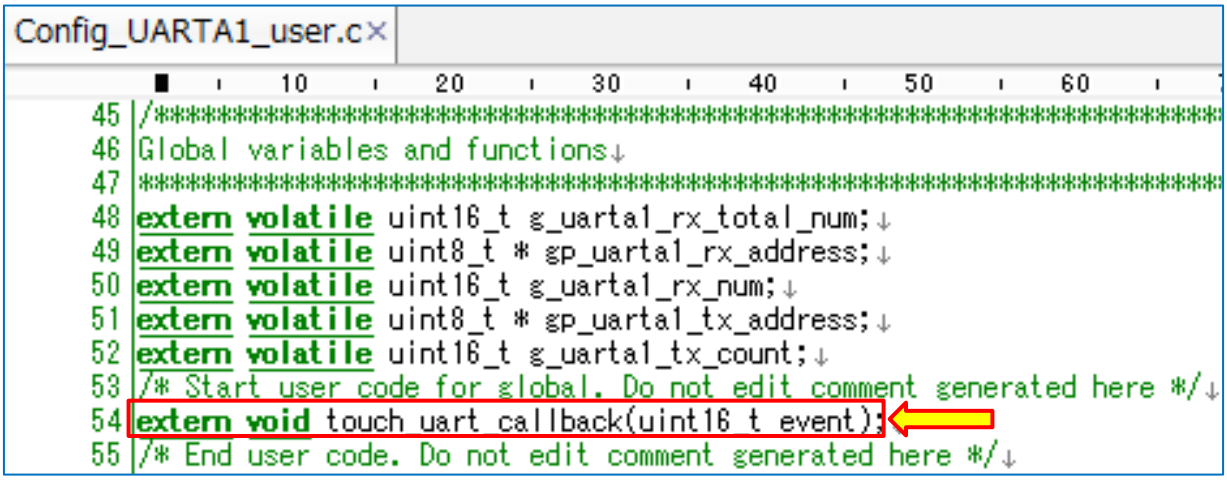
以下に示す 8 章、13 章および 15 章(本章を含む)では、UART を使用したシリアル通信モニタの設定について説明します。

- 8. [追加機能] UART を使用したシリアル通信モニタの設定 (1/3)
- 13. [追加機能] UART を使用したシリアル通信モニタの設定 (2/3)
- 15. [追加機能] UART を使用したシリアル通信モニタの設定 (3/3)

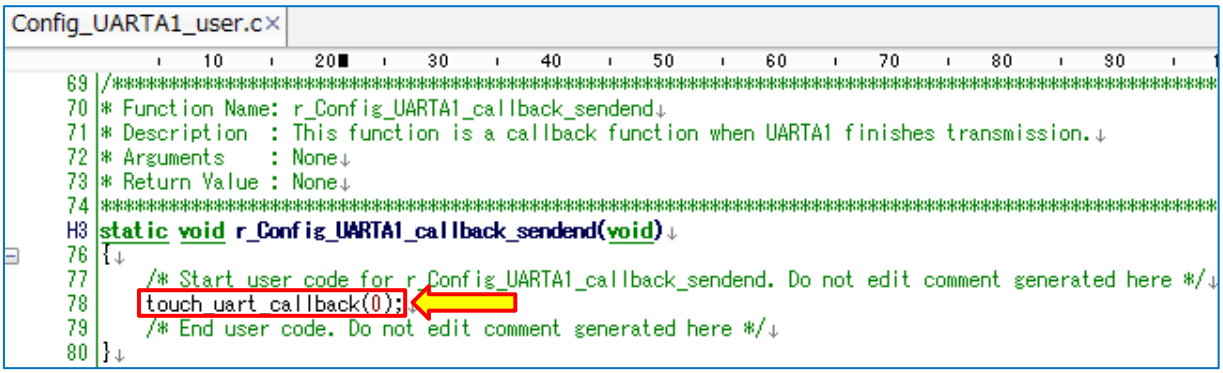
1. “**Config_UARTA1_user.c**”ファイルを開きます。



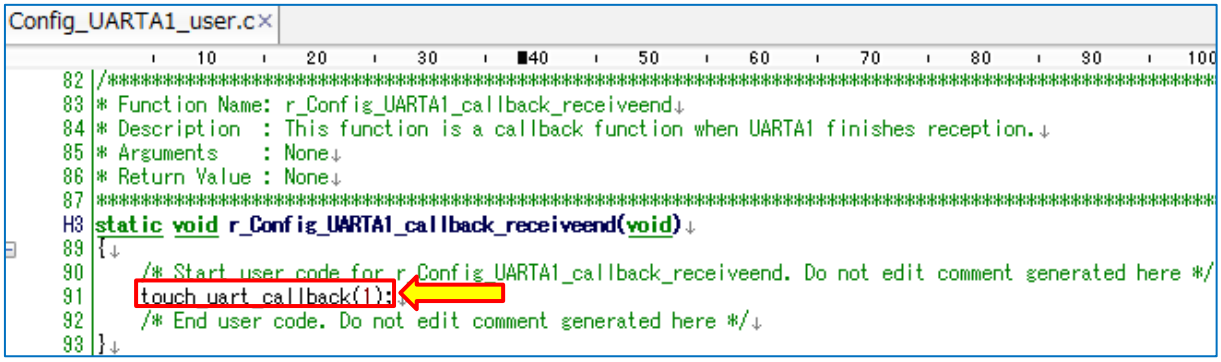
2. “**Config_UARTA1_user.c**”ファイルへ下記画像のように赤枠部分のコード“**extern void touch_uart_callback(uint16_t event);**”を追加してください。



3. “**Config_UARTA1_user.c**”ファイルへ下記画像のように赤枠部分のコード“**touch_uart_callback(0);**”を追加してください。



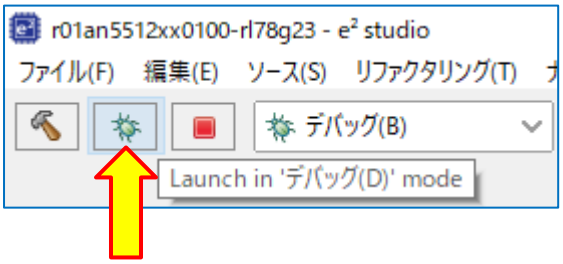
4. “**Config_UARTA1_user.c**”ファイルへ下記画像のように赤枠部分のコード“**touch_uart_callback(1);**”を追加してください。



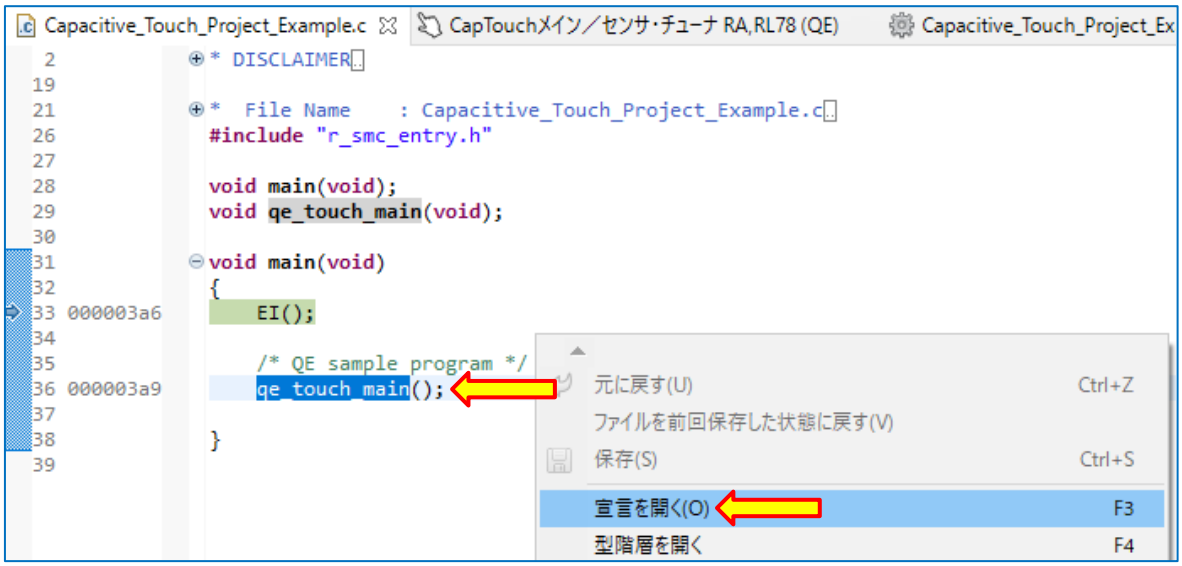
5. e² studio の左上の アイコンをクリックしてプロジェクトをビルドします。“コンソール”ウィンドウで、ビルドした結果にエラーがないことが確認できます。

14. “式”ウィンドウと QE for Capacitive Touch によるモニタリング

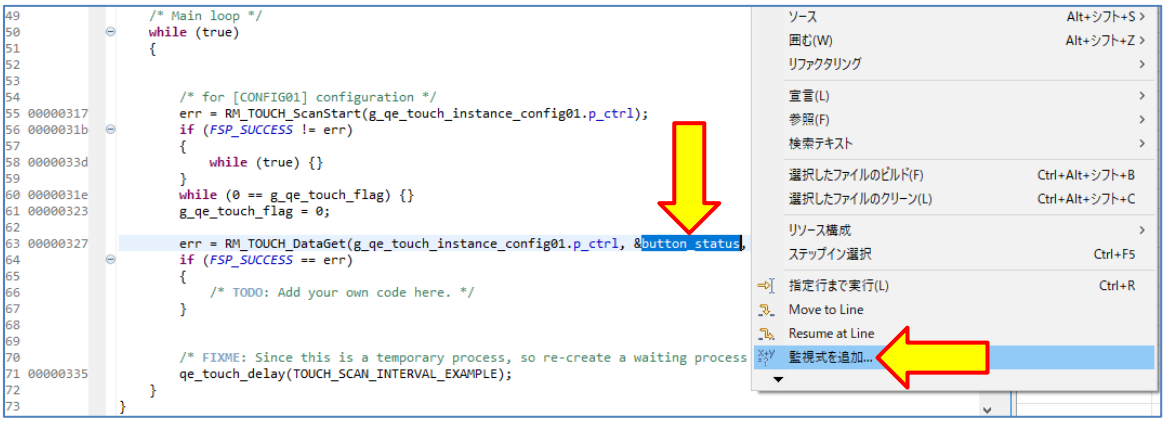
1. e² studio の左上にある  アイコンをクリックしてデバッグセッションを開始します。



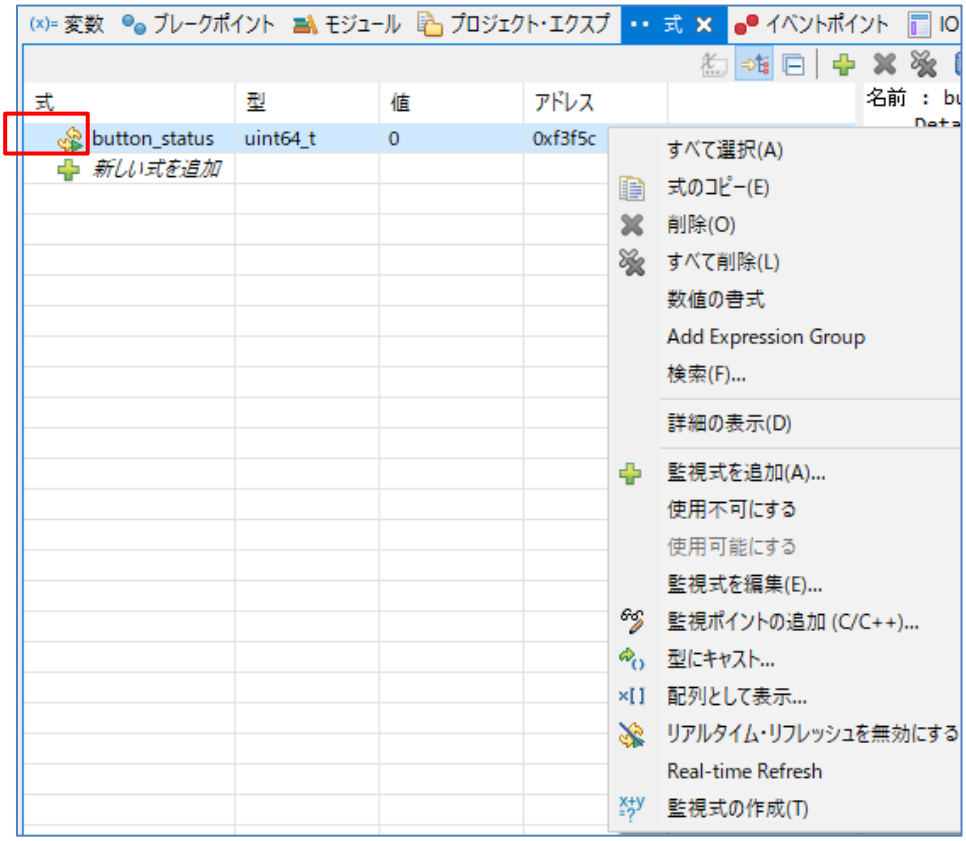
2. デバッグセッションは **main()**関数コールでストップします。
3. **qe_touch_main()**関数の宣言を開きます。



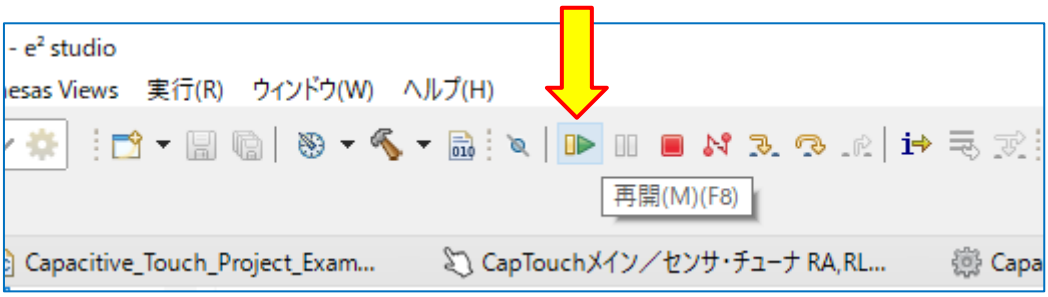
4. **qe_touch_sample.c** ファイルを下にスクロールして while(true)ループの **RM_TOUCH_DataGet()**関数を表示し、“式”ウィンドウに変数 **button_status** を追加します。



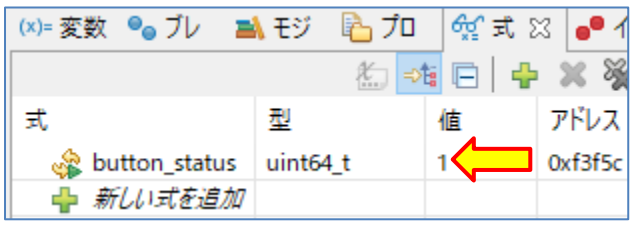
5. “式”ウィンドウで追加した変数のリアルタイム・リフレッシュが有効に設定されていることを確認します。



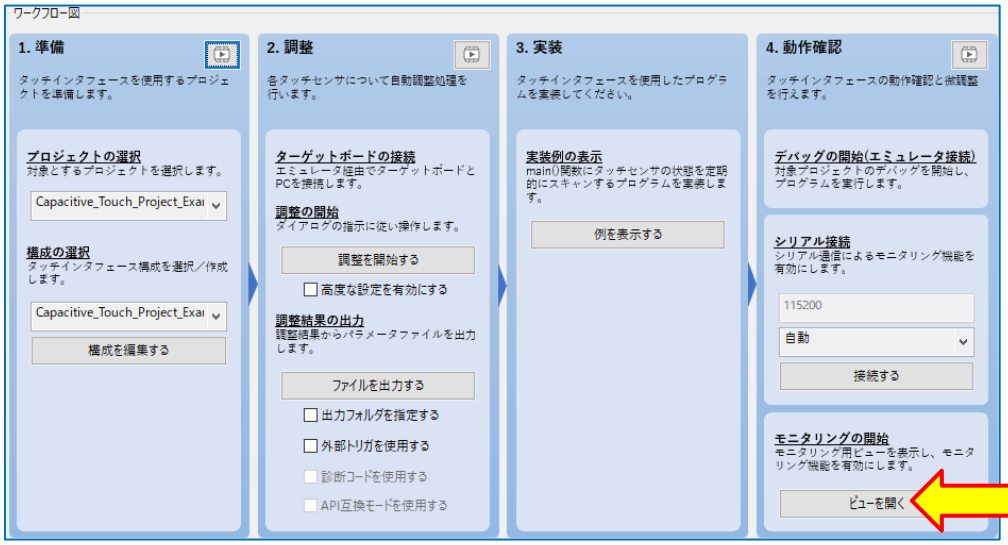
6. e² studio のツールバーボタンのほぼ中央にある“**再開**”ボタンをクリックしプログラム実行を続行します。



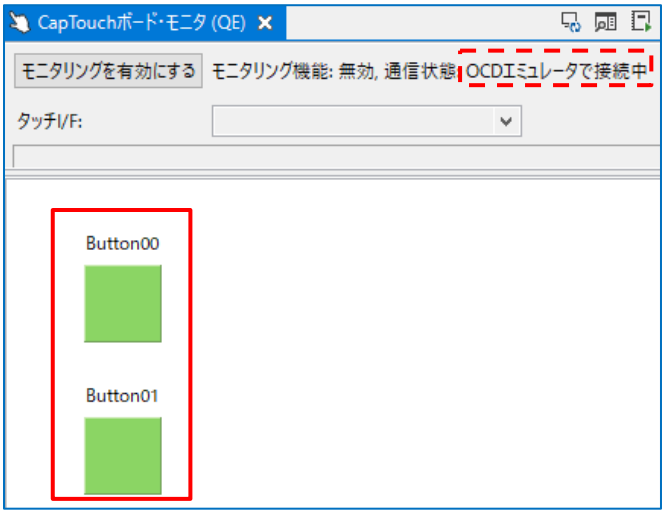
7. このアプリケーションノートの「**9. 静電容量タッチインタフェース作成**」章で“**Button01**”として定義した、ボード上の **TS05** をタッチします。“**Button01**”をタッチすると、“式”ウィンドウの **button_status** の値が 1 になることを確認できます。



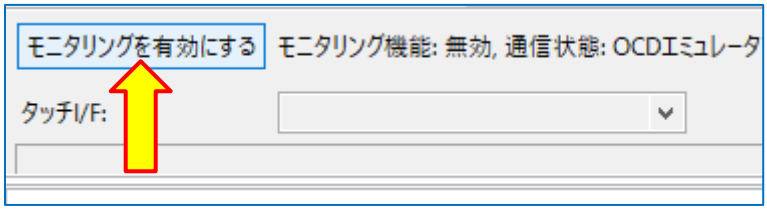
8. [**CapTouch メイン (QE)**]／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]から動作確認の“**ビューを開く**”を選択し、[**CapTouch ボード・モニタ (QE)**]を起動します。



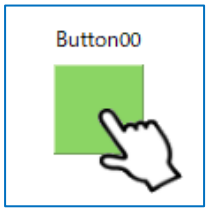
9. 見やすくするためウィンドウをドラッグする必要があるかもしれませんが、“**CapTouch ボード・モニタ (QE)**”は以下のように表示されます。



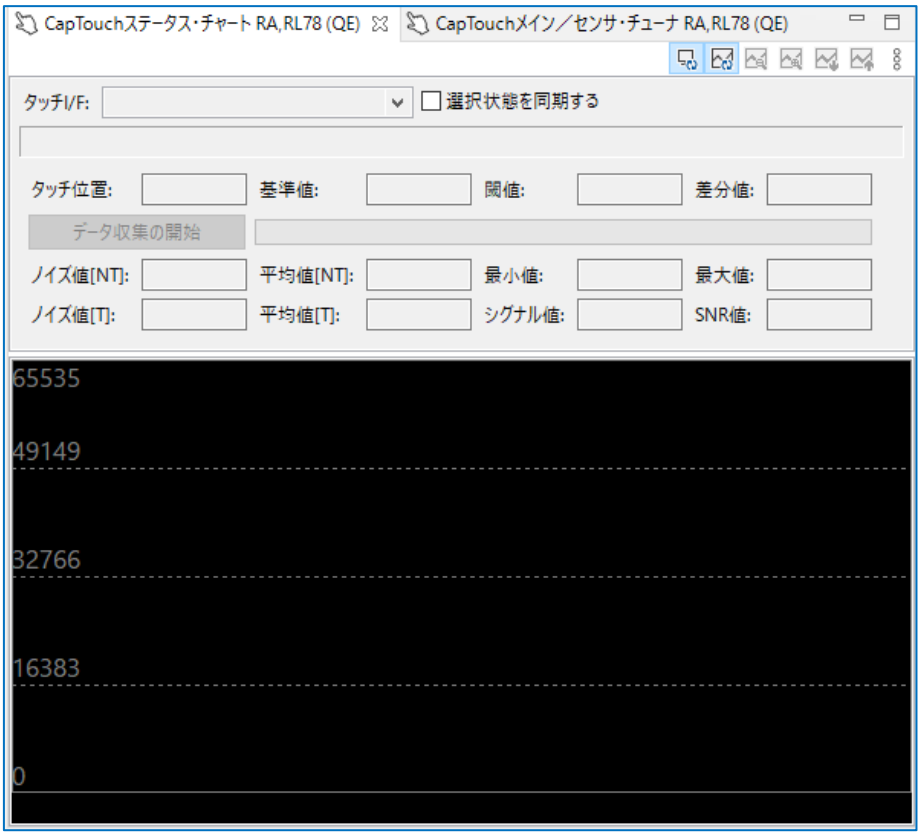
10. [**モニタリングを有効にする**]をクリックします。“**モニタリング機能: 無効**”が“**モニタリング機能: 有効**”に切り替わります。



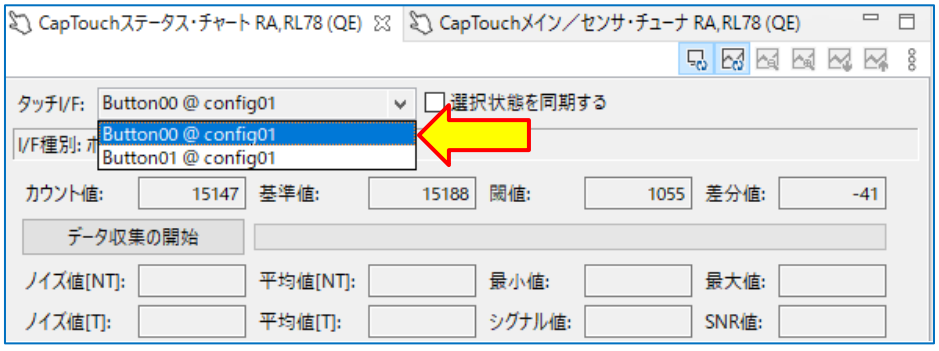
11. ターゲットボード上のボタン **TS06** をタッチします。“**CapTouch ボード・モニタ (QE)**”では、以下のように、タッチした状態をボタン上の指アイコンで表します。



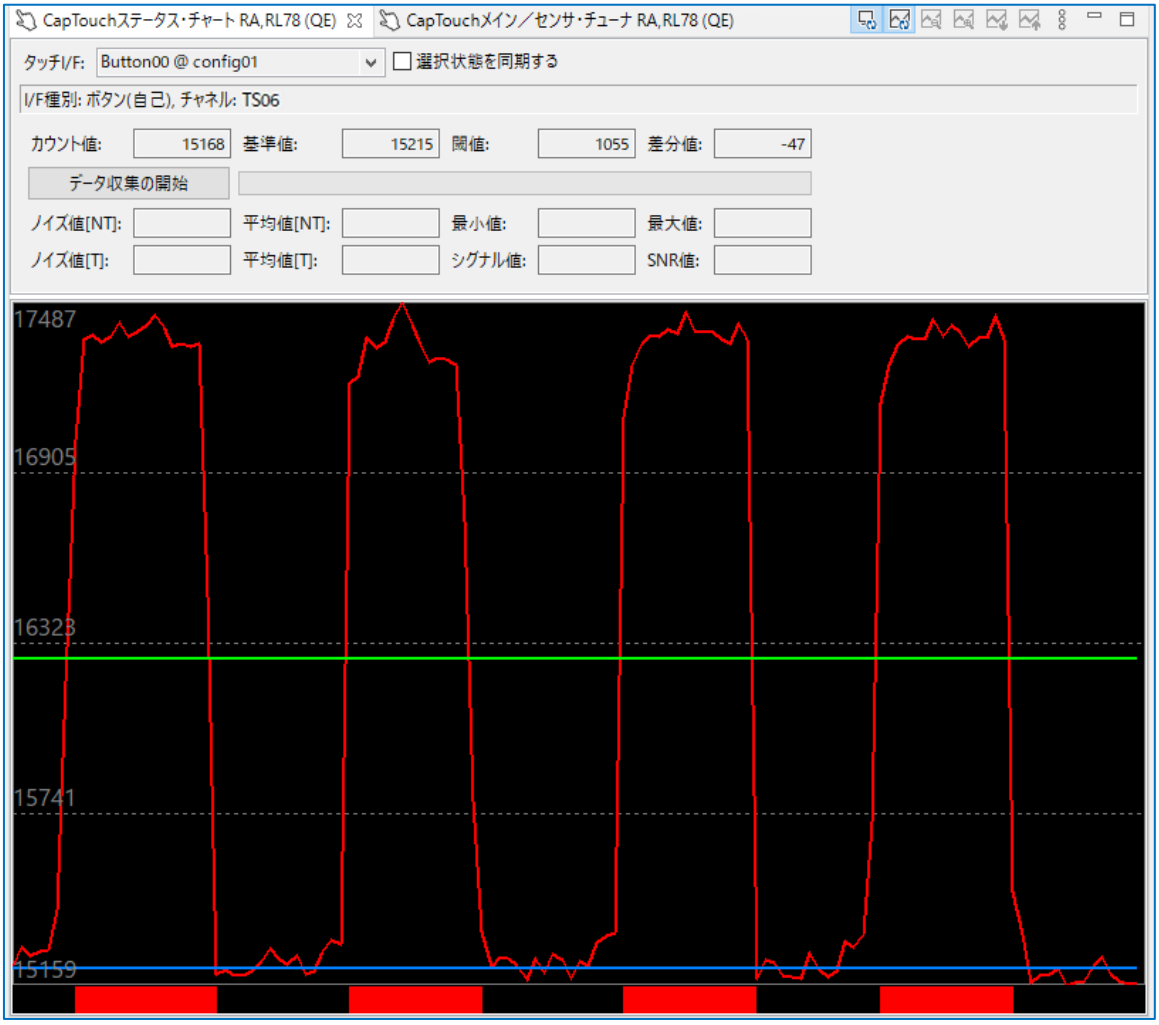
12. タッチカウント値をグラフィカルに表示するには、“**CapTouch ステータス・チャート (QE)**”を起動します。



13. プルダウンメニューから“**Button00 @ config01**”を選択します。

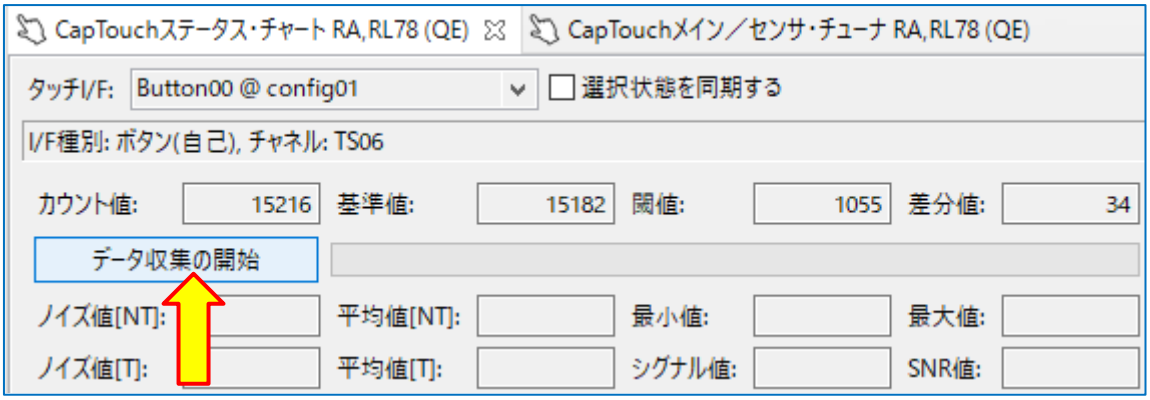


14. グラフには実行中の値が表示されます。ボード上の **TS06** をタッチすると、タッチカウント値がグラフ上で変化することを確認できます。緑のラインは閾値を表し、**rm_touch** ミドルウェアはボタンが操作/タッチされているかどうかを判断するために使用されます。グラフ下部の赤い矩形はタッチカウント値が閾値を超えてタッチが検出されたことを表示しています。



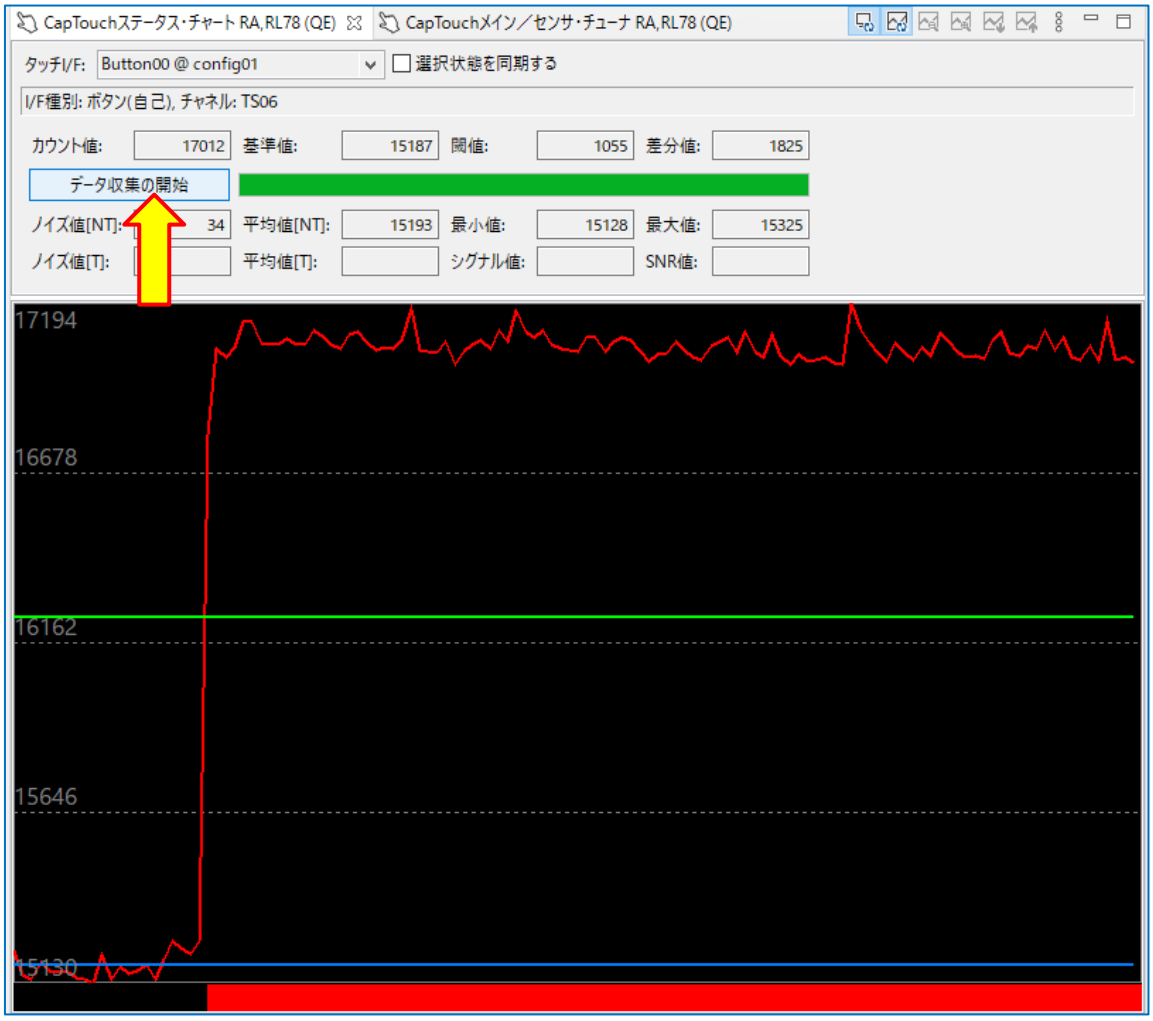
注.　15～18 項は標準偏差の表示および測定する際に設定する必要があります。

15. 次に標準偏差の測定方法についてです。“**データ収集の開始**”をクリックします。タッチオフ状態を収集している間は電極に触れないでください。緑色のバーはデータ収集率を表しています。緑色のバーが右端まで達するとタッチオフ状態のデータ収集は完了します。

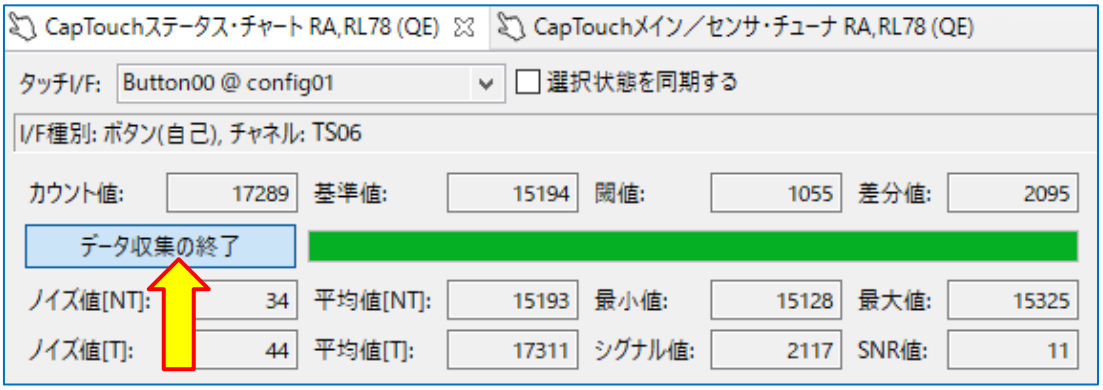


16. 緑色のバーが右端まで達したときに“**データ収集の終了**”をクリックしてください。

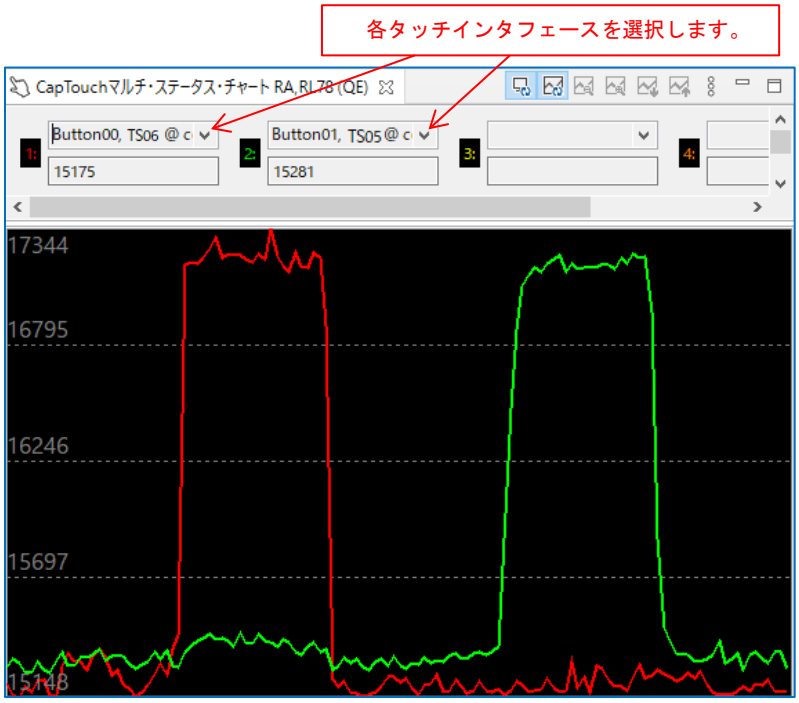
17. 次にタッチオン状態のデータを収集するために電極に触れてください。電極に触れている状態で“**データ収集の開始**”をクリックしてください。



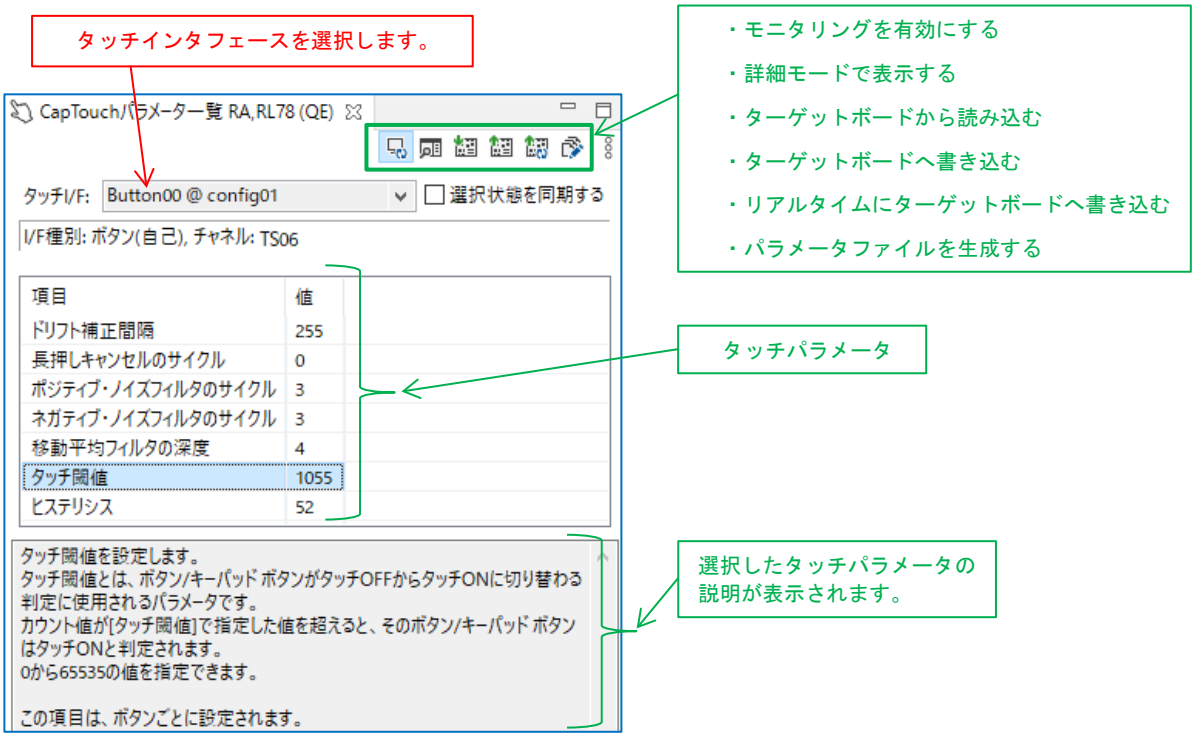
18. 緑色のバーが右端まで達したときに“**データ収集の終了**”をクリックしてください。データ収集が完了すると SNR が表示されます。



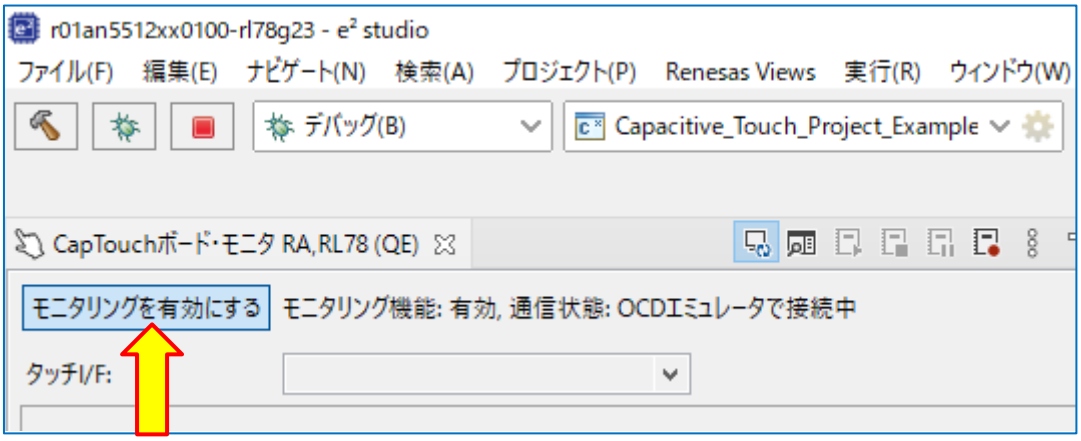
19. 複数のタッチセンサのタッチカウント値をグラフィカルに表示するには、“**CapTouch マルチ・ステータス・チャート (QE)**”を使用します。



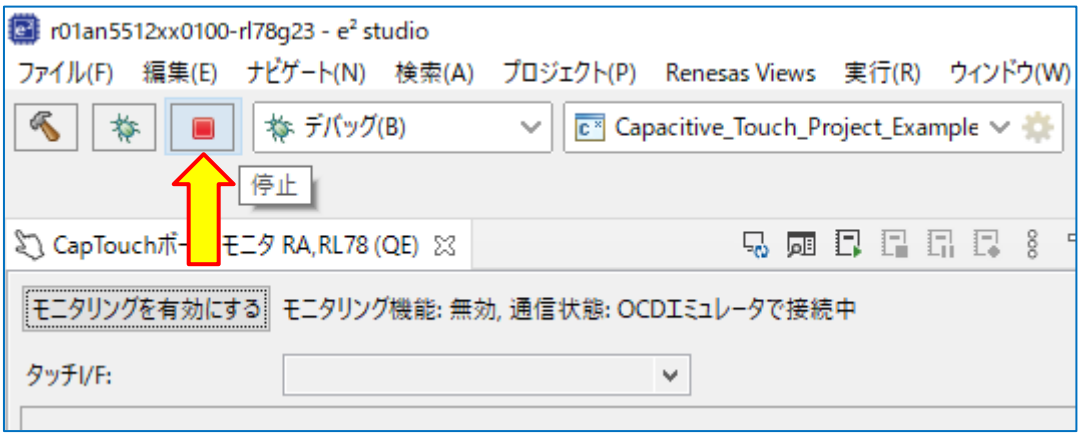
20. タッチパラメータを確認および調整する場合は、“**CapTouch パラメーター覧 (QE)**”を使用します。



21. モニタリングを終了する場合は、[**モニタリングを有効にする**]をクリックします。“**モニタリング機能: 有効**”が“**モニタリング機能: 無効**”に切り替わります。



22. デバッグセッションを終了する場合は、停止アイコンをクリックします。



15. [追加機能] UART を使用したシリアル通信モニタの設定 (3/3)

注.　タッチアプリケーションのタッチ性能のモニタリングは、OCD(On-Chip Debugging)エミュレータを介した通信によって確認できます。ただし、RL78 ファミリの場合、モニタリングパフォーマンスは RL78 ファミリの OCD 機能によって制限されます。
また一方で、タッチ性能のモニタリングは、シリアル通信を介して行うこともできます。 したがって、スムーズにモニタリングを行いたい場合は、シリアル通信を介したモニタリング機能を追加してください。

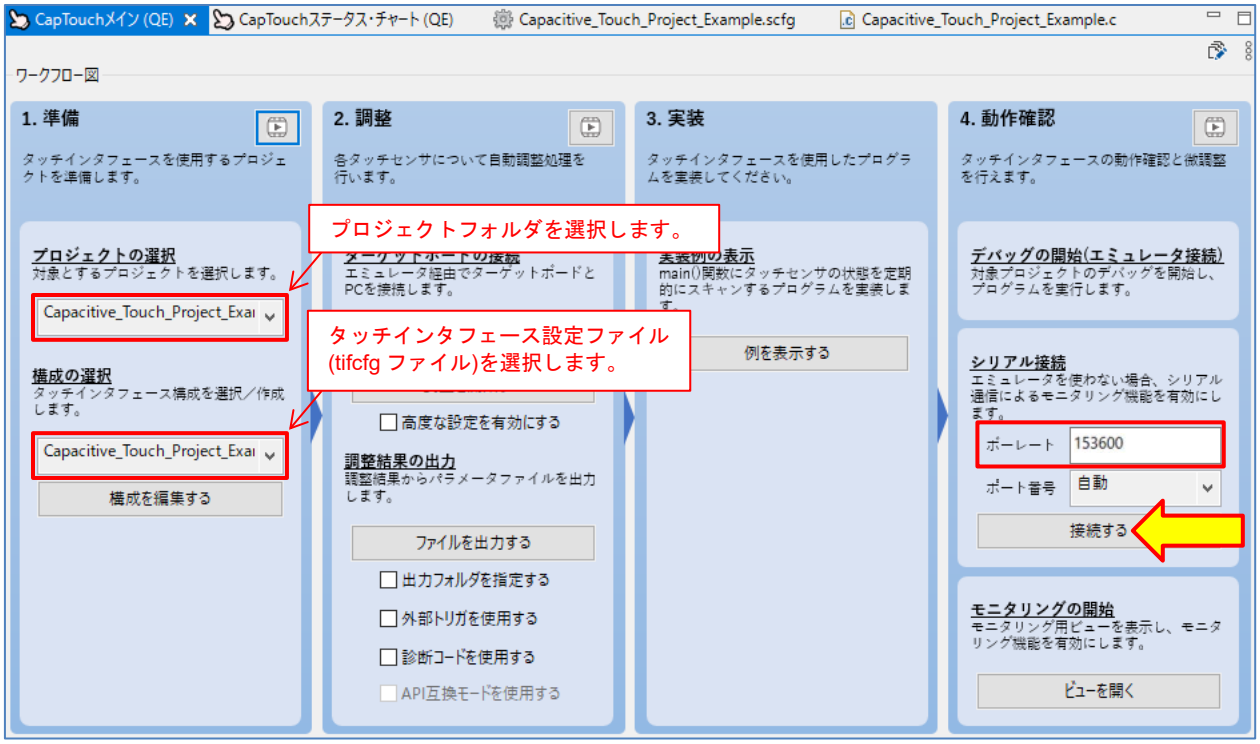
以下に示す 8 章、13 章および 15 章(本章を含む)では、UART を使用したシリアル通信モニタの設定について説明します。

- 8. [追加機能] UART を使用したシリアル通信モニタの設定 (1/3)
- 13. [追加機能] UART を使用したシリアル通信モニタの設定 (2/3)
- 15. [追加機能] UART を使用したシリアル通信モニタの設定 (3/3)

1. シリアル接続(UART/USB)を介してターゲットボードを PC に接続します。

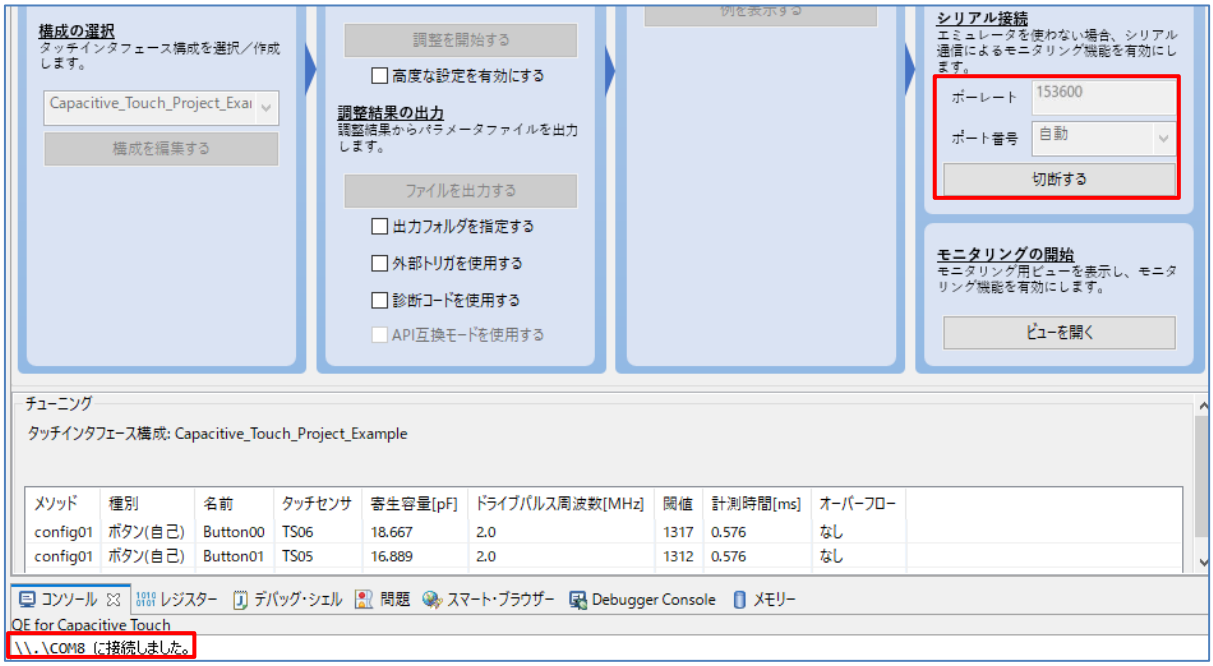
2. ターゲットボードでタッチアプリケーションプログラムを実行します。

3. “**CapTouch メイン (QE)**／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]”パネルを開きます。プロジェクトフォルダ(**Capacitive_Touch_Project_Example**)および tifcfg ファイル(**Capacitive_Touch_Project_Example.tifcfg**)が下図の赤枠のように設定されていることを確認した後、“**ボーレート**”を“**153600**”(bps)に設定し、[**接続する**]ボタンをクリックします。

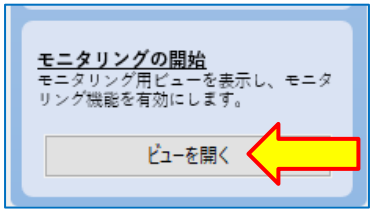


注.　ボーレート(bps)の設定値は、「8. [追加機能] UART を使用したシリアル通信モニタの設定 (1/3)」章の手順 5 で設定したボーレート(bps)を入力してください。

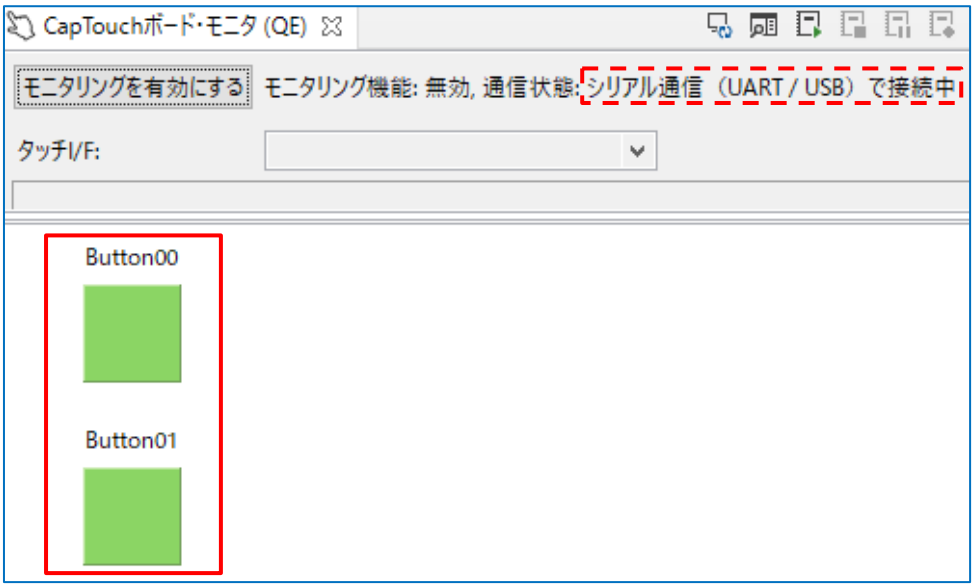
4. シリアル接続を実行すると、以下の表示が変わります。表示されているメッセージを確認します。



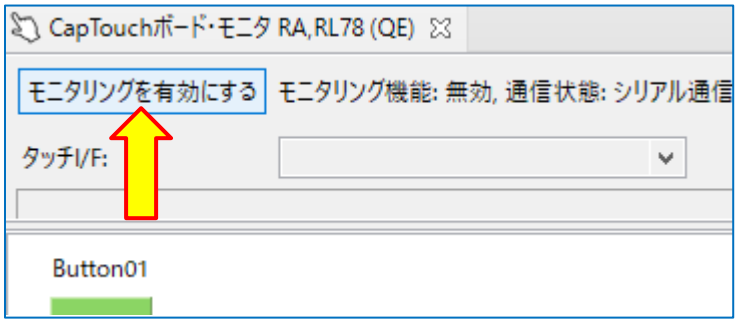
5. [**CapTouch メイン (QE)**]／QE V3.2.0 以降では[**CapTouch ワークフロー (QE)**]から動作確認の“**ビューを開く**”を選択し、[**CapTouch ボード・モニタ (QE)**]を起動します。



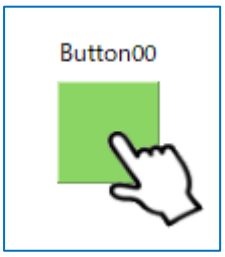
6. “**CapTouch ボード・モニタ (QE)**”は以下のように表示されます。



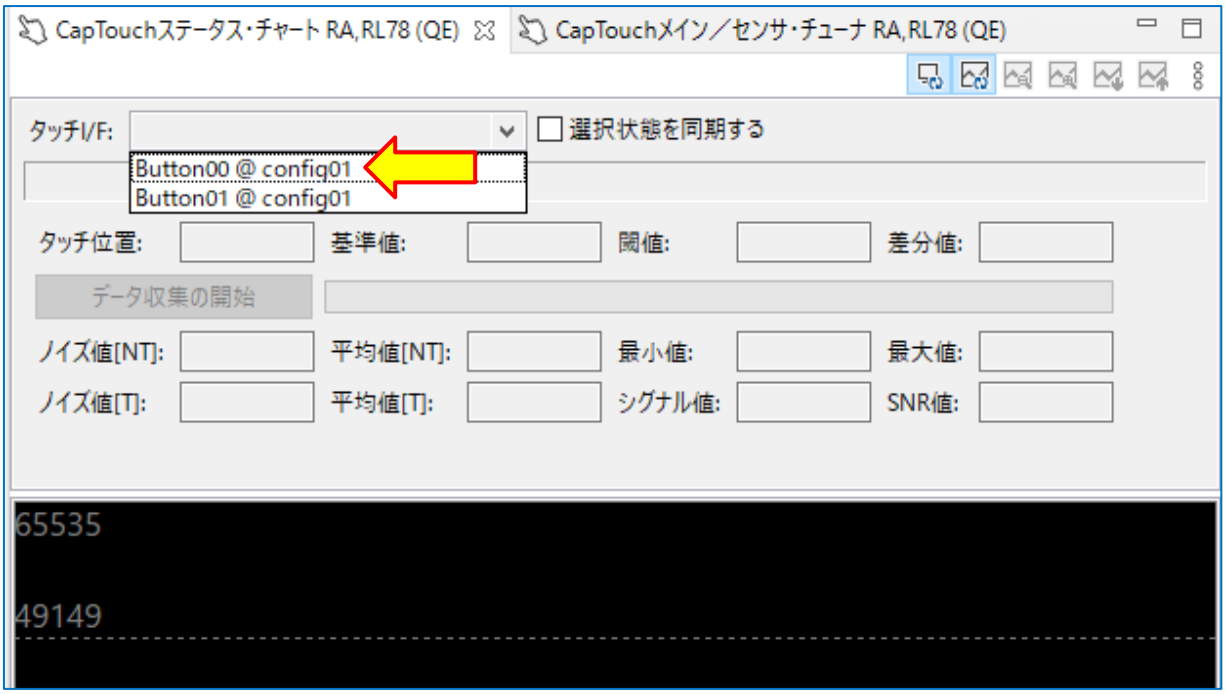
7. [**モニタリングを有効にする**]をクリックします。“**モニタリング機能: 無効**”が“**モニタリング機能: 有効**”に切り替わります。



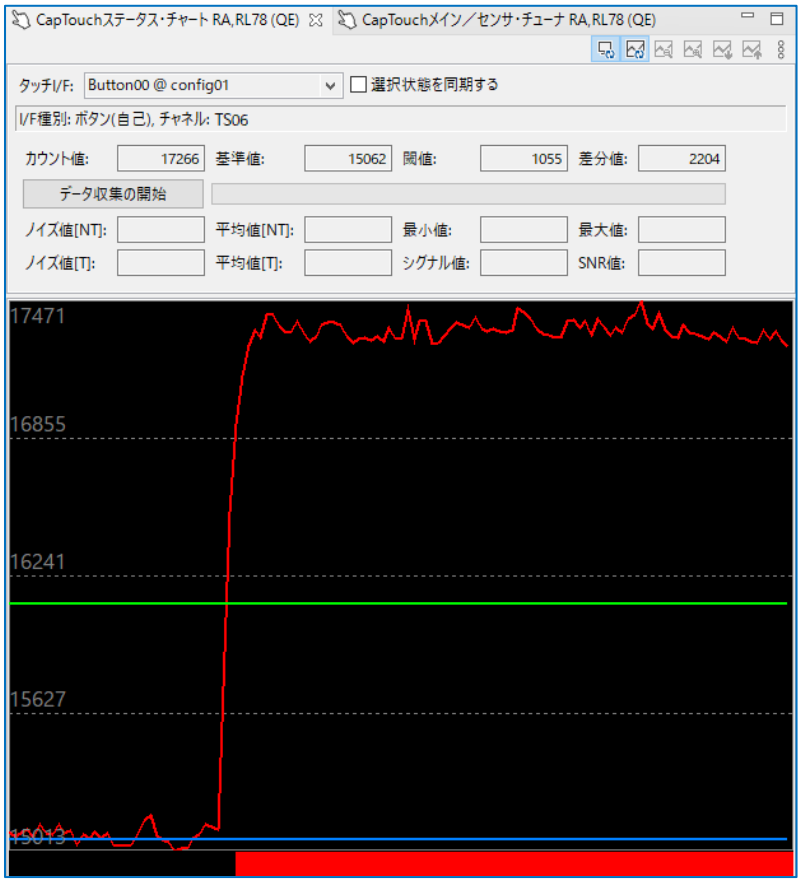
8. ターゲットボード上のボタン **TS06** をタッチします。“**CapTouch ボード・モニタ (QE)**”では、以下のように、タッチした状態をボタン上の指アイコンで表します。



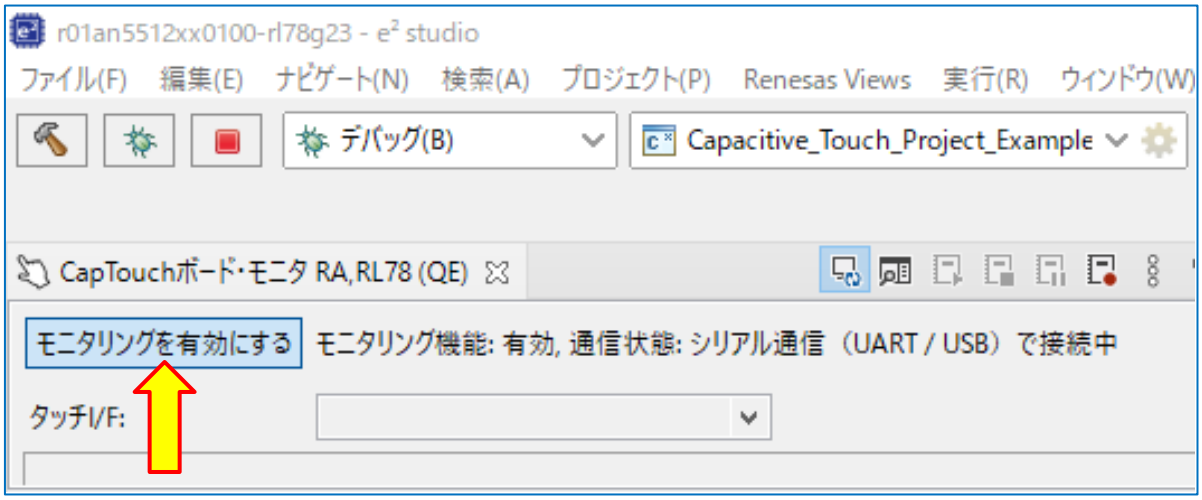
9. タッチカウント値をグラフィカルに表示するには、“**CapTouch ステータス・チャート (QE)**”を起動します。プルダウンメニューから“**Button00 @ config01**”を選択します。



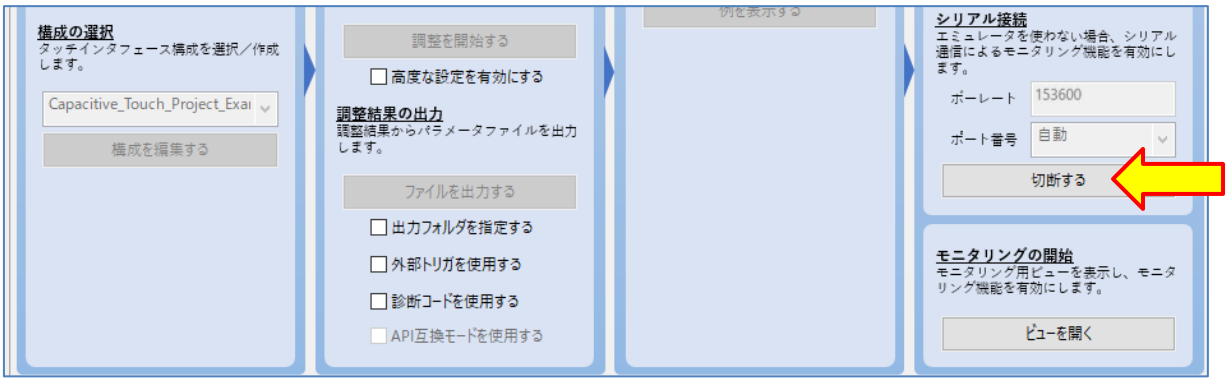
10. グラフには実行中の値が表示されます。ボード上の **TS06** をタッチすると、タッチカウント値がグラフ上で変化することを確認できます。



11. モニタリングを終了する場合は、[**モニタリングを有効にする**]をクリックします。“**モニタリング機能: 有効**”が“**モニタリング機能: 無効**”に切り替わります。



12. シリアル接続を終了する場合は、[**切断する**]ボタンをクリックします。



16. qe_touch_sample.c のリスト(ソフトウェアタイマ使用例)

QE for Capacitive Touch から出力されるサンプルプログラムを以下に示します。

```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2021-07-29
* DESCRIPTION : Main Program for RL78
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#define TOUCH_SCAN_INTERVAL_EXAMPLE (20 * 1000)    /* microseconds */

void R_CTSU_PinSetInit(void);
void qe_touch_main(void);
void qe_touch_delay(uint16_t delay_us);

uint64_t button_status;
#if (TOUCH_CFG_NUM_SLIDERS != 0)
uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
#endif
#if (TOUCH_CFG_NUM_WHEELS != 0)
uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];
#endif




void qe_touch_main(void)
{
    fsp_err_t err;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();




    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }
```

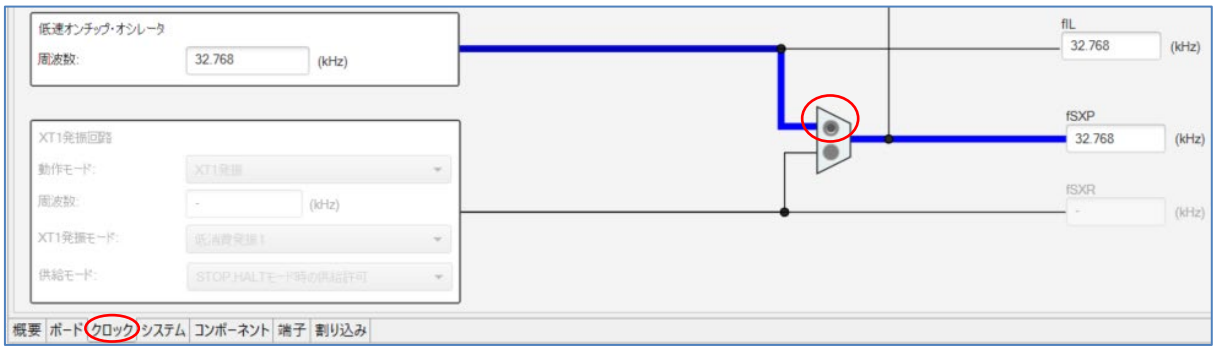
```
    /* Main loop */
    while (true)
    {


        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            /* TODO: Add your own code here. */
        }


        /* FIXME: Since this is a temporary process, so re-create a waiting process yourself. */
        qe_touch_delay(TOUCH_SCAN_INTERVAL_EXAMPLE);
    }
}

void qe_touch_delay(uint16_t delay_us)
{
    uint32_t i;
    uint32_t loops_required;
    uint16_t clock_mhz;

    clock_mhz = (uint16_t)(R_BSP_GetFclkFreqHz() / 1000000);
    if (0 == clock_mhz)
    {
        clock_mhz = 1;
    }

    loops_required = ((uint32_t)delay_us * (uint32_t)clock_mhz);
    loops_required /= 20;
    for (i = 0; i < loops_required; i++)
    {
        BSP_NOP();
    }
}
```

17. [付録] ハードウェアタイマでのタッチ計測

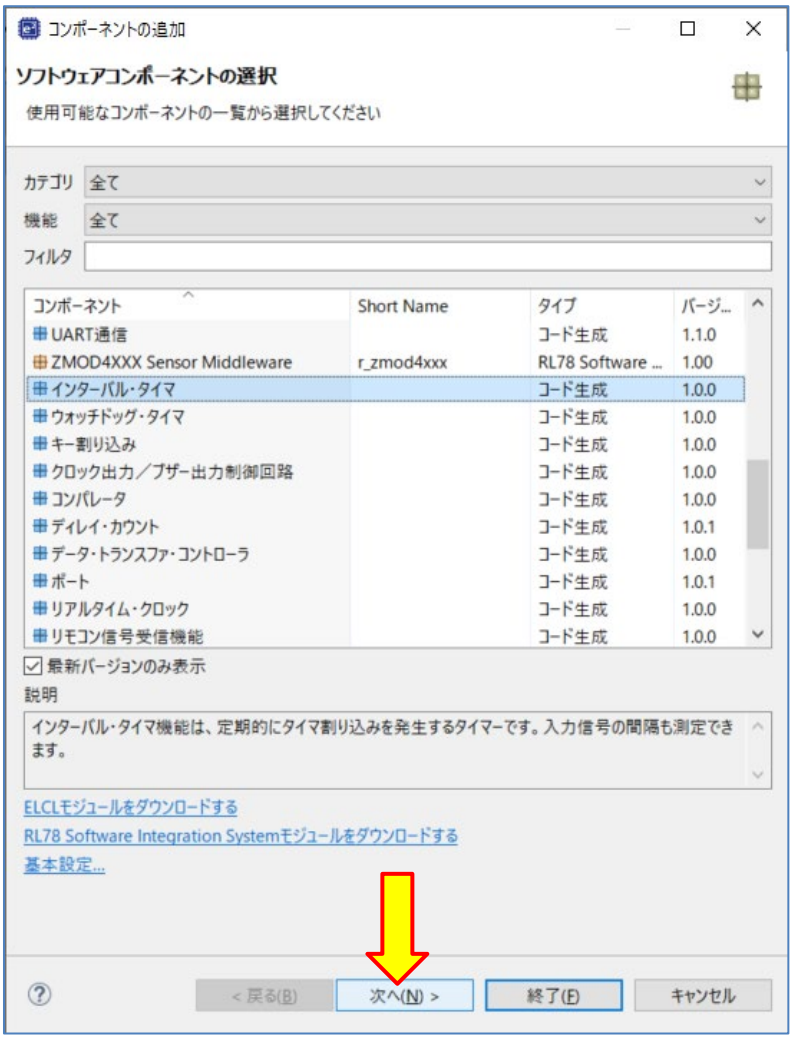
ハードウェアタイマによるタッチ計測周期の実装例を説明します。

17.1 ハードウェアタイマの設定手順

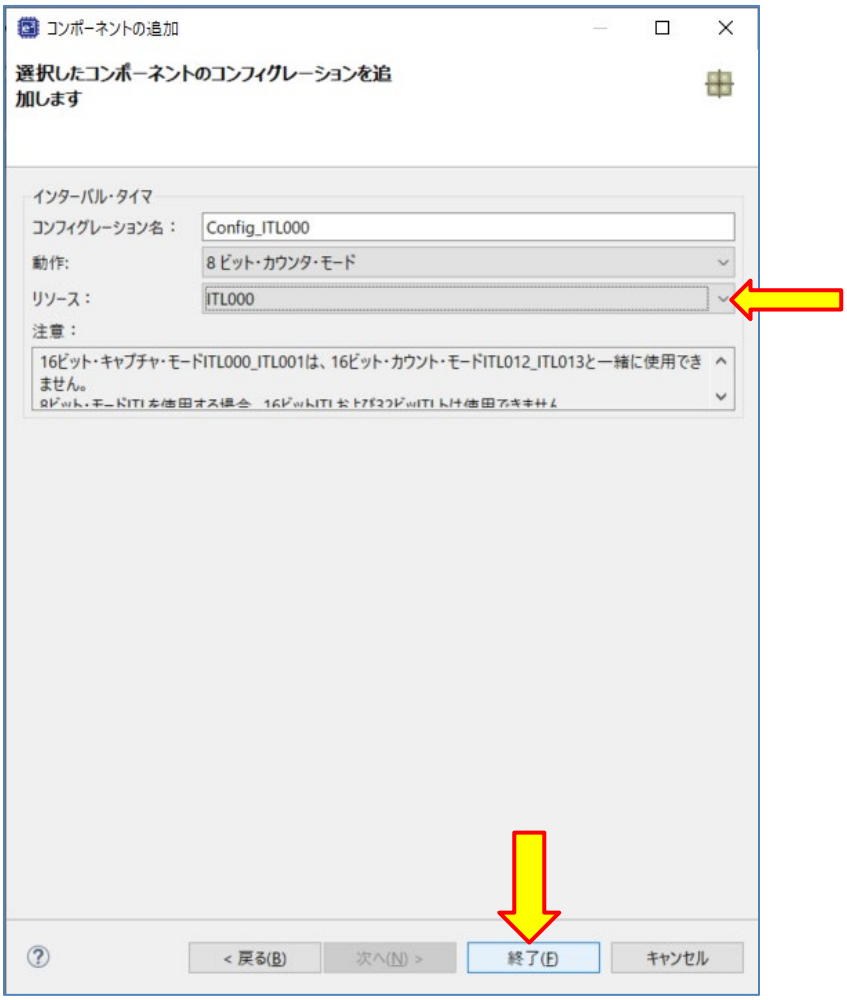
1. タッチ計測周期に使用するクロックを以下のように設定します。



2. ソフトウェアコンポーネント設定で“**インターバル・タイマ**”モジュールを選択し、ダイアログ下部の[**次へ**]をクリックします。

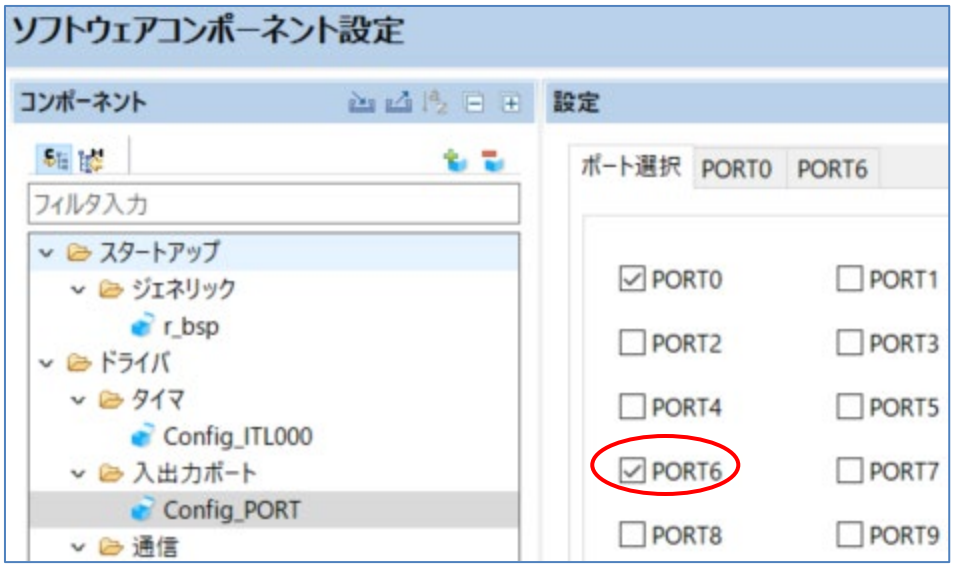


3. インターバル・タイマの構成を以下のように設定し、ダイアログ下部の[**終了**]をクリックします。

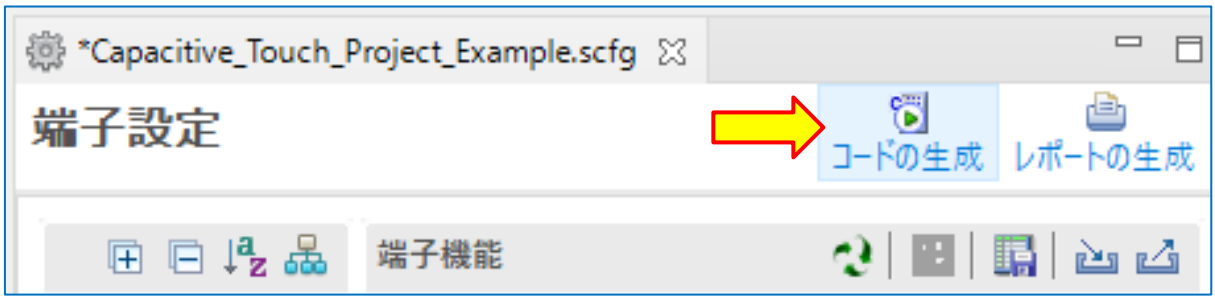


4. インターバル・タイマを以下のように設定します。

5. 動作確認のための LED 点灯/消灯に使用するポートを以下のように設定します。



6. 以下の図のように、スマート・コンフィグレータの右上の アイコンをクリックして、プロジェクトに必要なモジュールのコードを追加します。



7. プログラムの実装例は、「17.2 タッチ計測プログラム(ハードウェアタイマ使用例)」を参照してください。

17.2 タッチ計測プログラム(ハードウェアタイマ使用例)

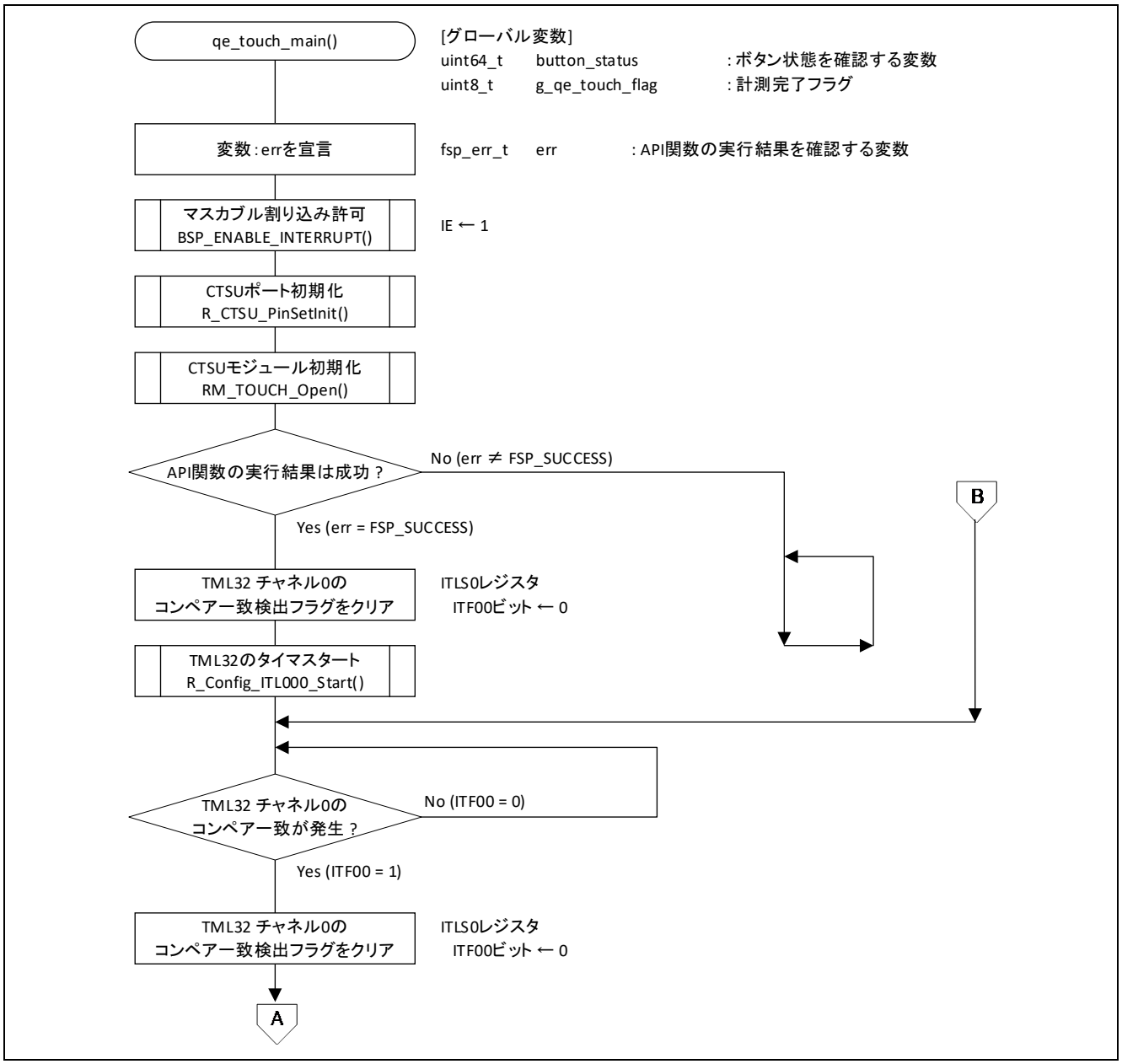
ハードウェアタイマでのタッチ計測のプログラム実装例を以下に示します。

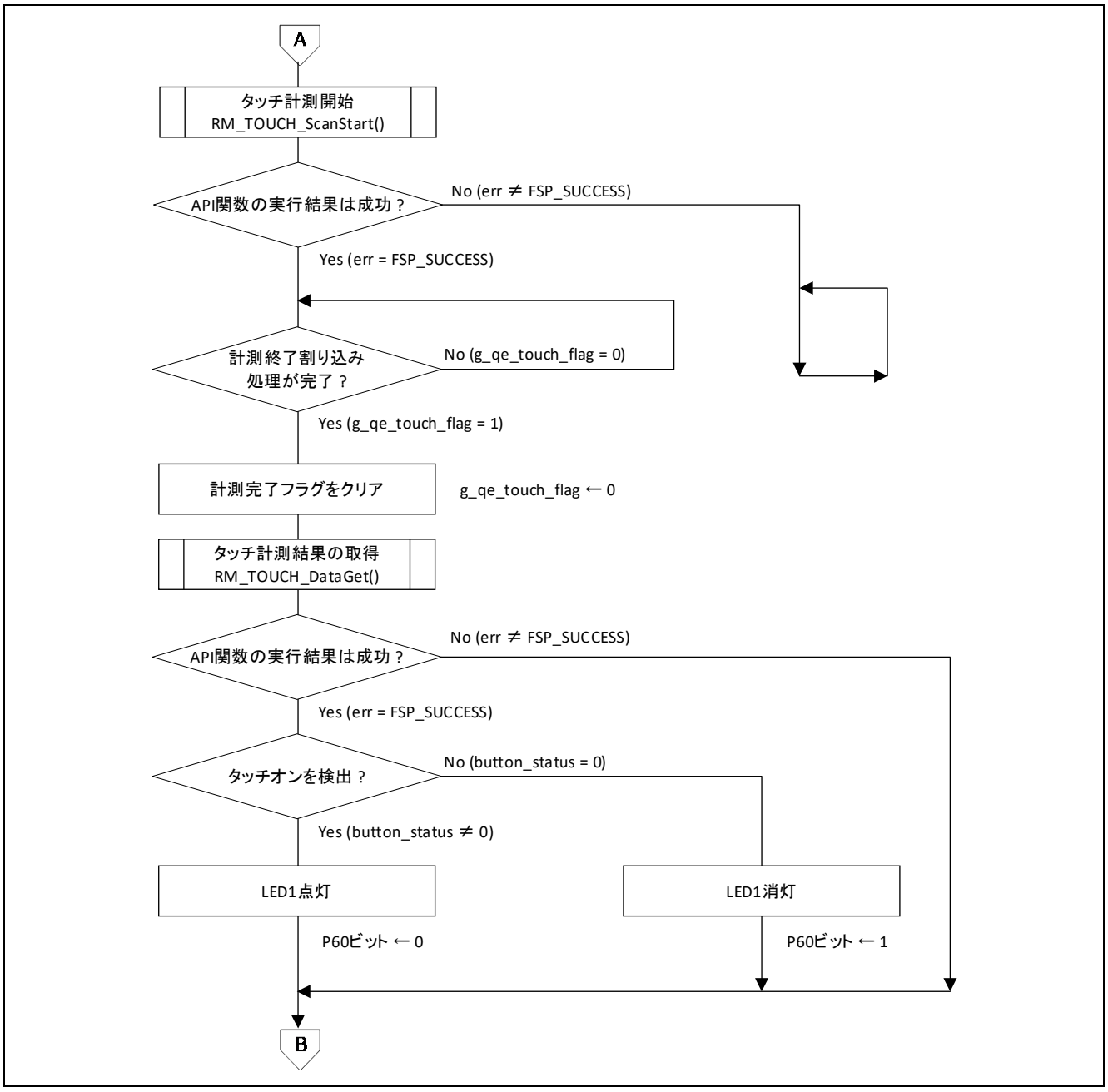
```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2022-05-09
* DESCRIPTION : CTSU2L Program for RL78
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#include "Config_ITL000.h"

void R_CTSU_PinSetInit(void);
void qe_touch_main(void);

uint64_t button_status;
#if (TOUCH_CFG_NUM_SLIDERS != 0)
uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
#endif
#if (TOUCH_CFG_NUM_WHEELS != 0)
uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];
#endif

void qe_touch_main(void)
{
    fsp_err_t err;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();

    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    ITLS0 &= ~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    R_Config_ITL000_Start();

    /* Main loop */
    while (true)
    {
        while (_00_ITL_CHANNEL0_COUNT_MATCH_NOT_DETECTE == (ITLS0 & _01_ITL_CHANNEL0_COUNT_MATCH_DETECTE)) {}
        ITLS0 &= ~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
```

```
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {

            /* TODO: Add your own code here. */
            if (0 != button_status)
            {
                P6_bit.no0 = 0;
            }
            else
            {
                P6_bit.no0 = 1;
            }
        }
    }
}
```

17.3 フローチャート(ハードウェアタイマ使用例)

ハードウェアタイマでのタッチ計測のフローチャートを以下に示します。




A
タッチ計測開始
RM_TOUCH_ScanStart()
API関数の実行結果は成功？
No (err ≠ FSP_SUCCESS)
Yes (err = FSP_SUCCESS)
計測終了割り込み
処理が完了？
No (g_qe_touch_flag = 0)
Yes (g_qe_touch_flag = 1)
計測完了フラグをクリア
g_qe_touch_flag ← 0
タッチ計測結果の取得
RM_TOUCH_DataGet()
API関数の実行結果は成功？
No (err ≠ FSP_SUCCESS)
Yes (err = FSP_SUCCESS)
タッチオンを検出？
No (button_status = 0)
Yes (button_status ≠ 0)
LED1点灯
LED1消灯
P60ビット ← 0
P60ビット ← 1
B

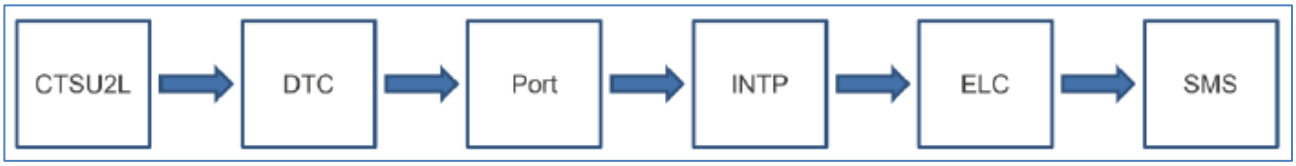
18. [応用例] 自動判定機能(SMS 使用)と MEC 機能の設定

注.　「自動判定機能(SMS 使用)と MEC 機能」は、CTSU2La を搭載した製品で対応しています。

本章では、RL78/G22 の自動判定機能(SMS 使用)と MEC（Multi Electrode Connection)機能の設定について説明します。

1. RL78/G22 の場合、SMS 計測使用時に使用するモジュールのフロー

   以下は、RL78/G22 で SMS 計測使用時に使用するモジュールのフローです。CTSU2La から DTC を使用してポート出力を行います。ポートから出力された信号を用いて割り込み信号を発生させます。割り込み信号によって ELC をトリガさせ SMS 処理を開始します。



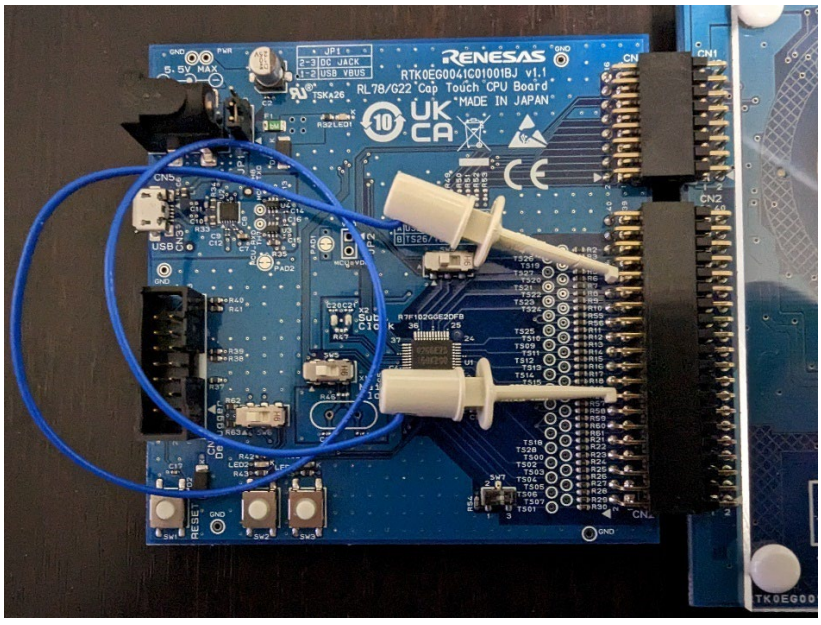
2. ターゲットボード

   RL78/G22 静電容量タッチ評価システム(製品型名: RTK0EG0042S01001BJ)を使用した場合で説明します。

3. 外部トリガの結線

   RL78/G22 で SMS を使用した自動判定計測を行う場合は、下記の設定をしてください。

   MCU ボード側の CN2 の 32 番ピン(P22/TS20)と 16 番ピン(P16/INTP5/TS17)を以下のように結線します。

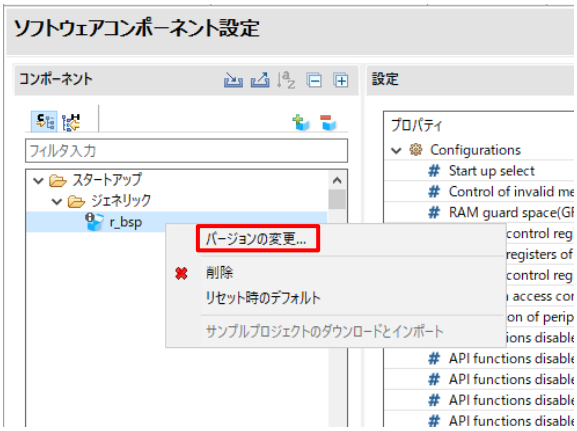


4. 「6. プロジェクト作成」章および「7. スマート・コンフィグレータによるモジュール追加」章と同じ手順で“プロジェクト作成とモジュールの追加を行います。

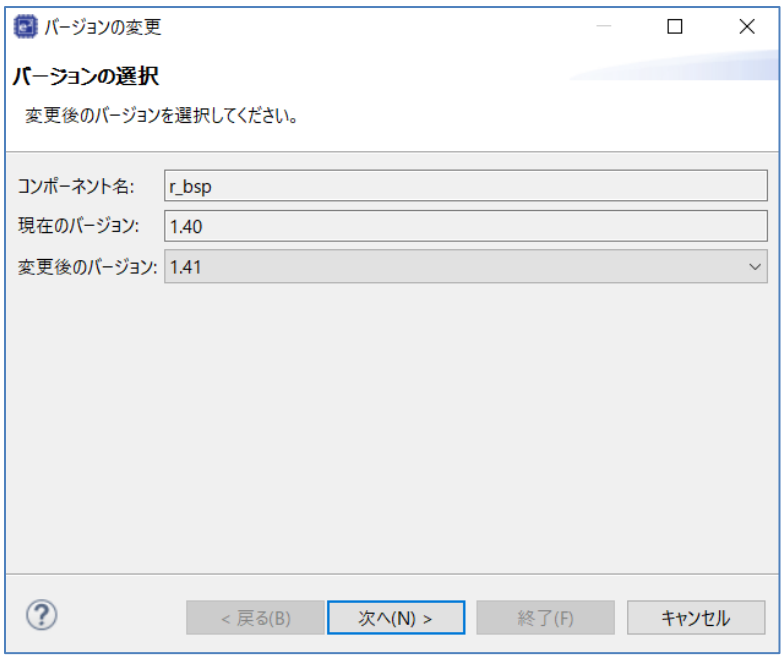
5.　コンポーネントの追加

5.1.　r_bsp のバージョン確認

「コンポーネント」タブを選択し、r_bsp のバージョンを確認します。
r_bsp を選択し右クリック、「バージョンの変更」(赤枠)をクリックします。

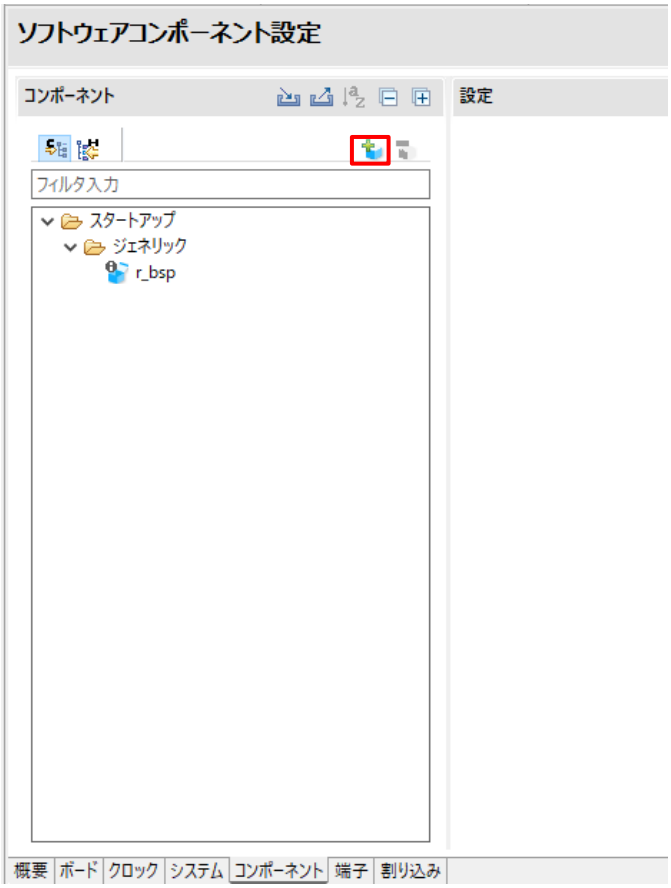


「現在のバージョン」が 1.40 以降と異なる場合は「変更後のバージョン」で 1.40 以降を選択して「次へ」をクリックでバージョンを変更してください。

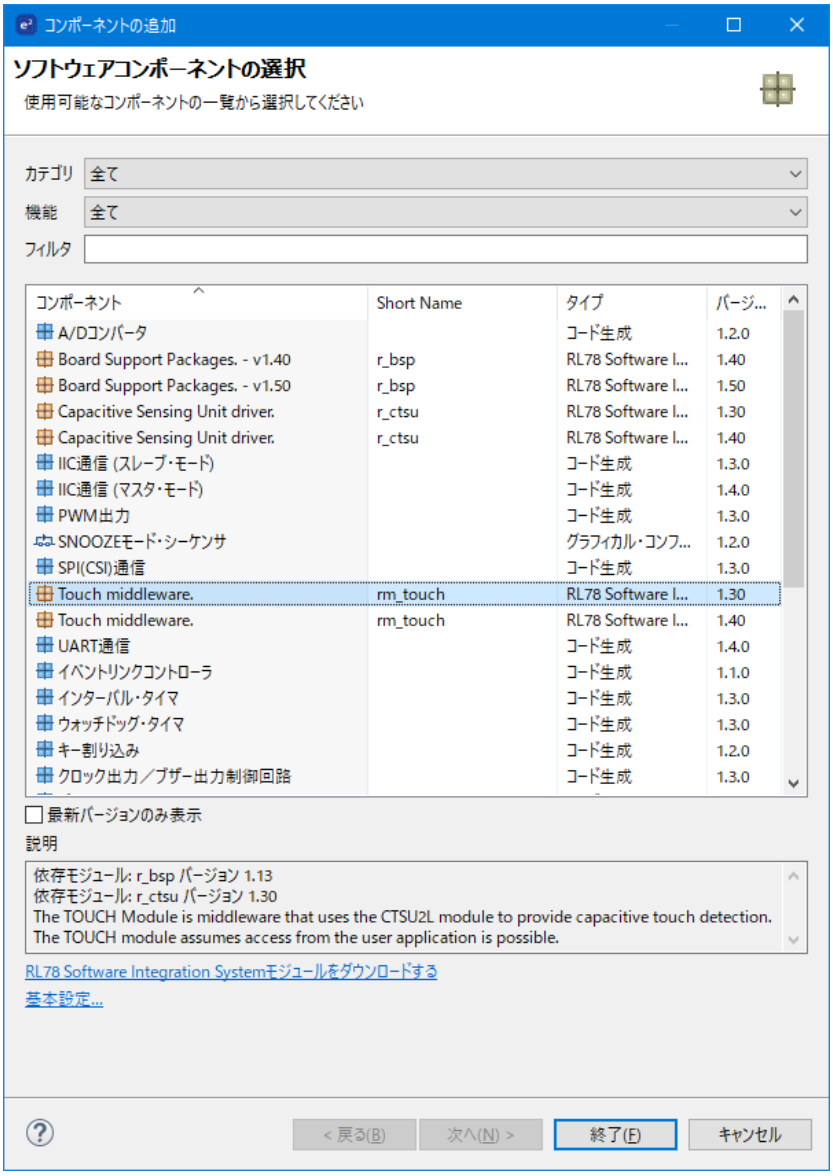


5.2.　rm_touch, r_ctsu コンポーネントの追加

コンポーネントの追加ボタン(赤枠)をクリックし、一覧から追加するコンポーネントを選択します。



「Touch middleware」のバージョン 1.30 以降を選択し、「終了」をクリックします。



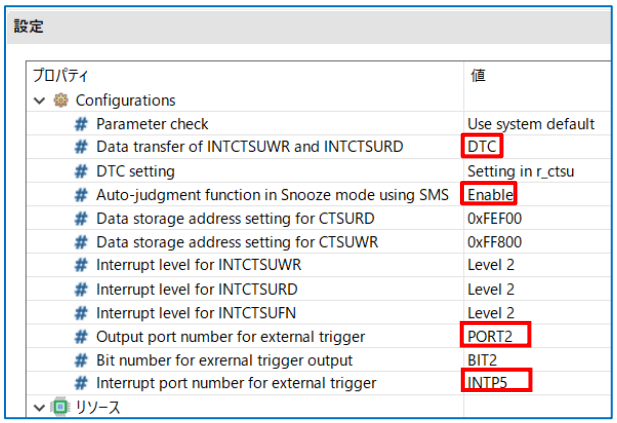
rm_touch を追加すると、依存モジュールである r_ctsu も同時に追加されます。
r_ctsu を選択し、「設定」を開いて、以下の設定を変更します。

「Data transfer of INTCTSUWR and INTCTSURD」の「Interrupt handler」を「DTC」へ。

「Auto-judgment function in Snooze mode using SMS」の「Disable」を「Enable」へ。

「Output port number for external trigger」で出力ポートを設定します。

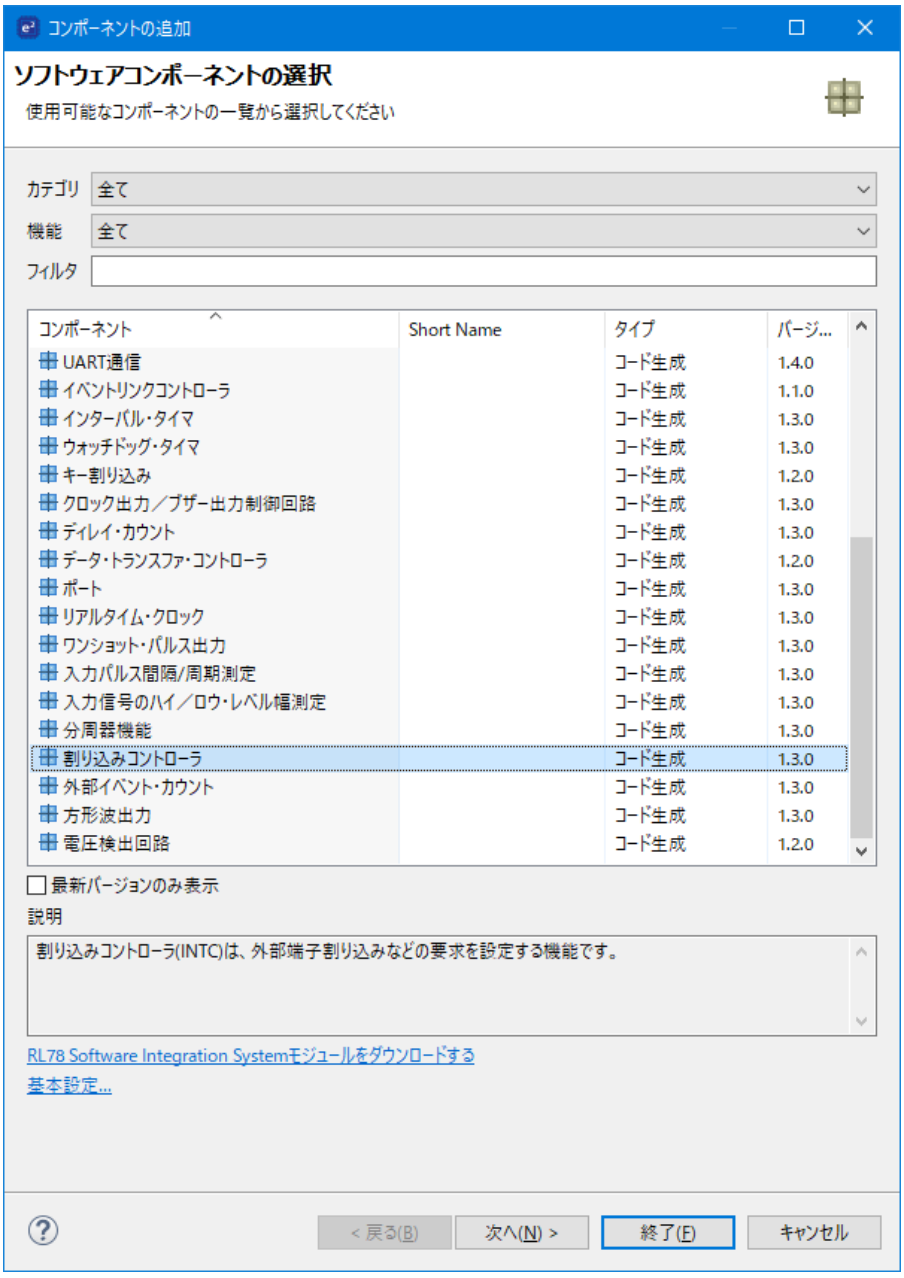
「Interrupt port number for external trigger」で割り込み信号を設定します。



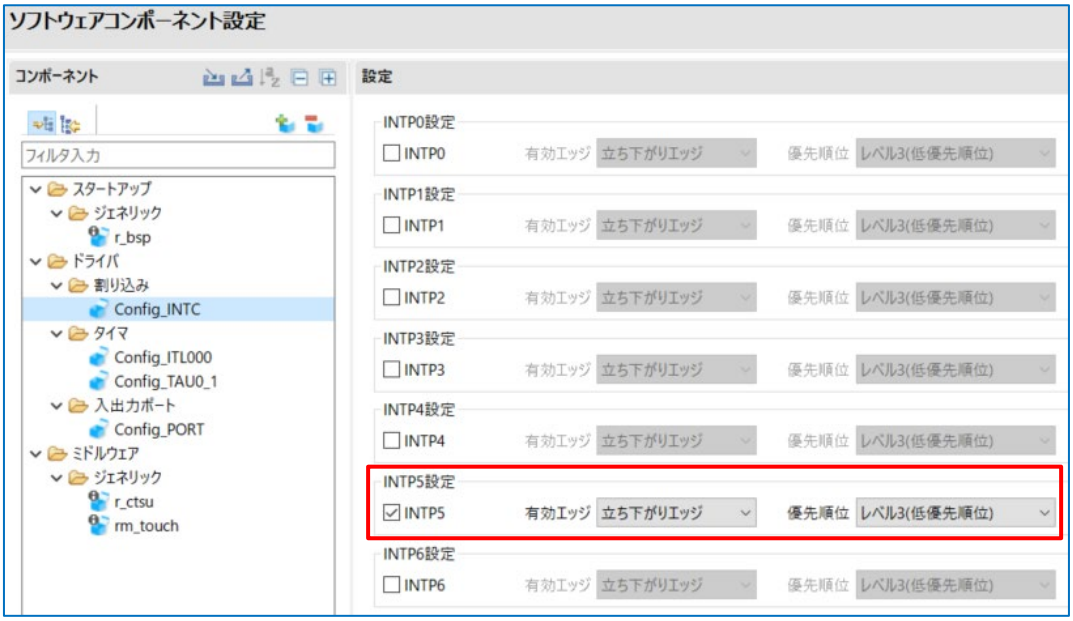
5.3.　割り込みコントローラコンポーネントの追加

同様に、割り込みコントローラを追加します。

一覧より「割り込みコントローラ」を選択して「終了」をクリックします。



「Config_INTC」を選択し、設定より「INTP5」にチェックを入れます。



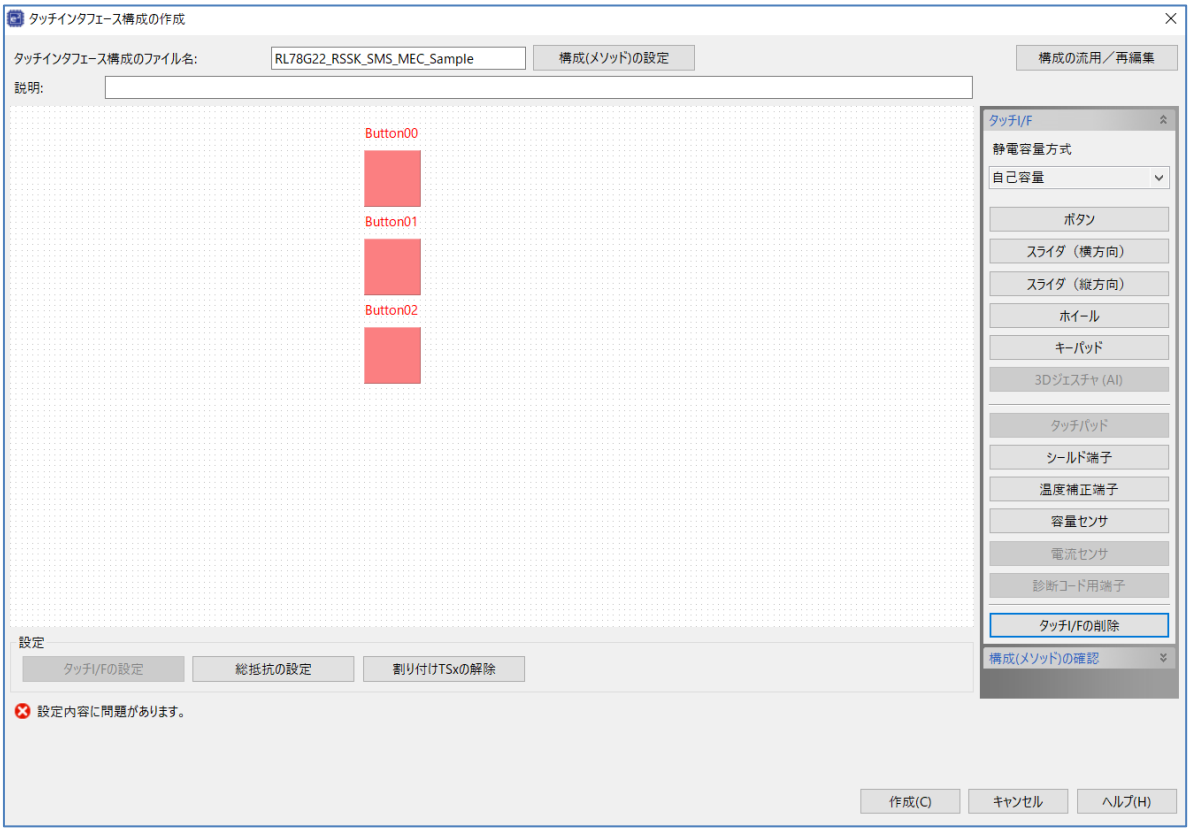
5.4.　ソースコード生成とビルド

「コードの生成」(赤枠)をクリックしソースコードを出力します。

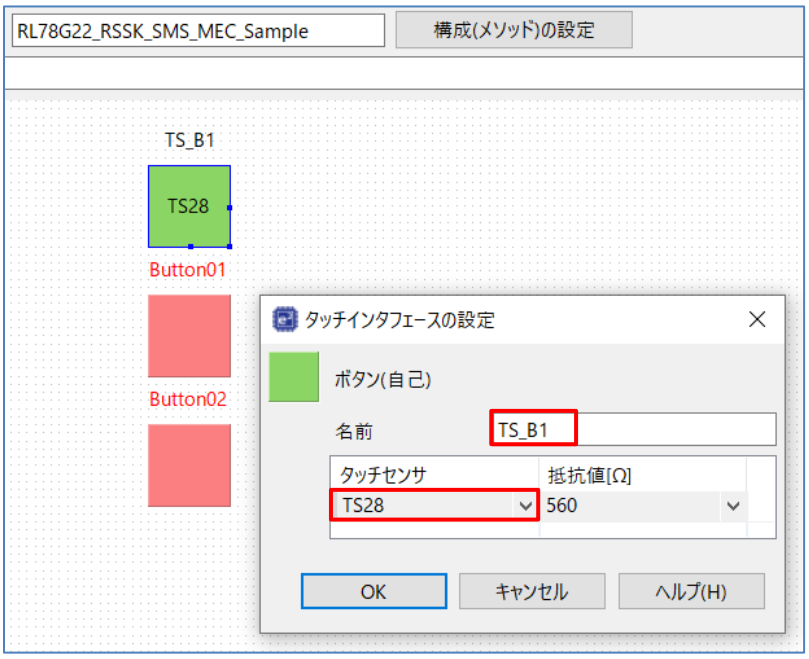
6.　インタフェース構成の作成

「9. 静電容量タッチインタフェース作成」章と同じ手順で、タッチインターフェースを設定します。

SMS 起動用の外部トリガとして一部のポートを使用するため、そのポートと対応するタッチセンサのホイールやボタンが使用できなくなります。また、今回は 3 つの電極を MEC として使用しますので、以下のようにボタンを配置します。
右の一覧から「ボタン」を選択し、画面上に配置していき、配置が終わったら「Esc」キーを押下して配置のモードを抜けます。

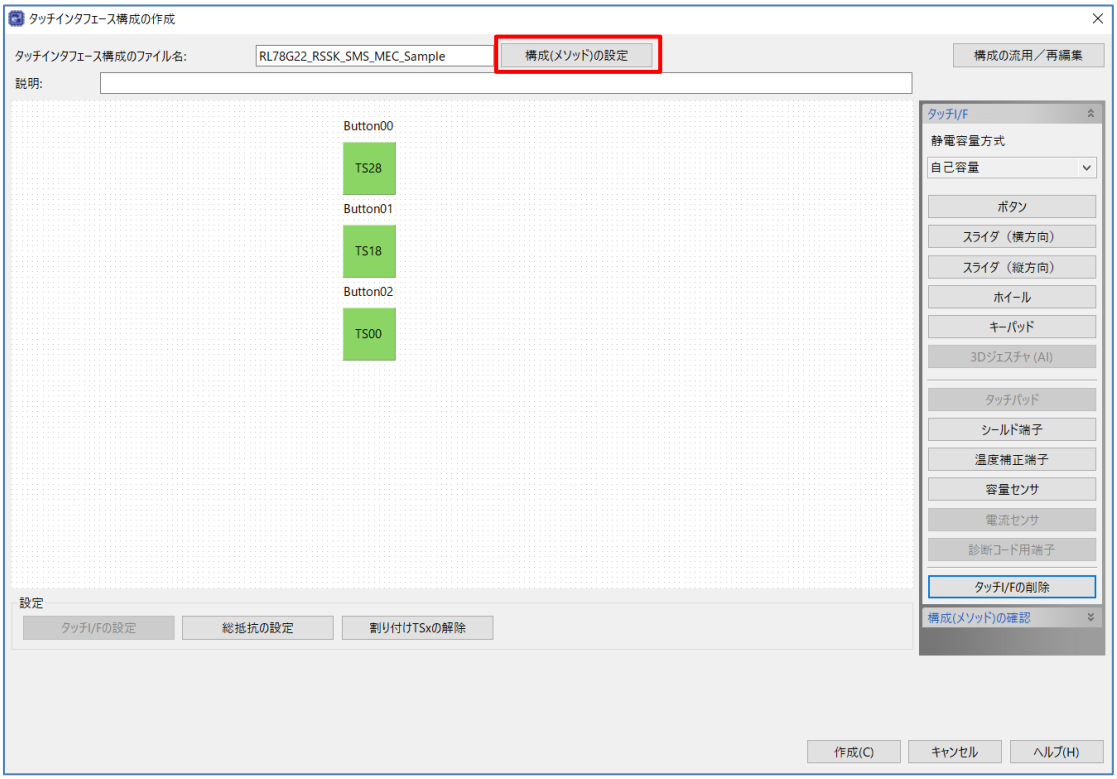


それぞれのボタンの名前とタッチセンサの割り当てを行います。
配置したボタンをダブルクリックし、設定ウインドウを開き、名前とタッチセンサ(赤枠)を変更して「OK」をクリックします。

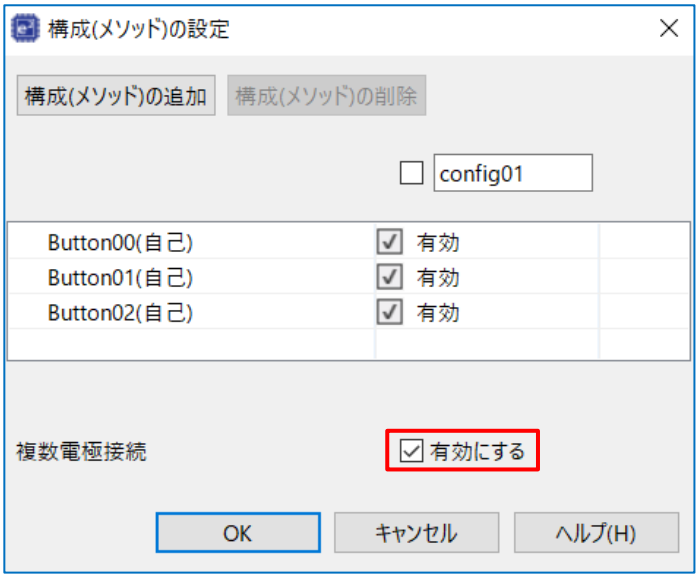


それぞれのボタンに以下のように名前とタッチセンサを設定して下さい。

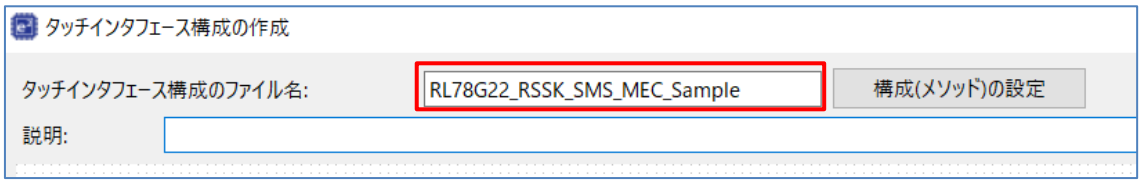
設定が完了したら、「構成(メソッド)の設定」クリックします。



「複数電極接続」を有効にし(赤枠)、「OK」をクリックします。



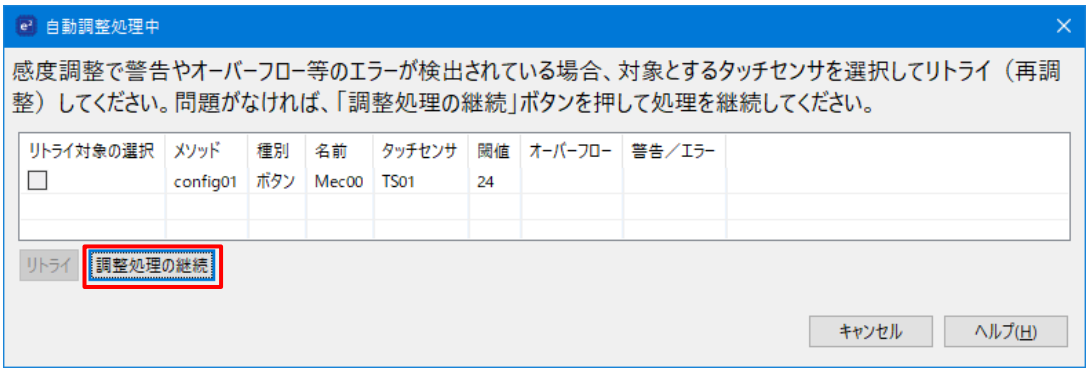
最後に、「タッチインタフェース構成のファイル名」を設定(赤枠)し、「作成」ボタンをクリックします。



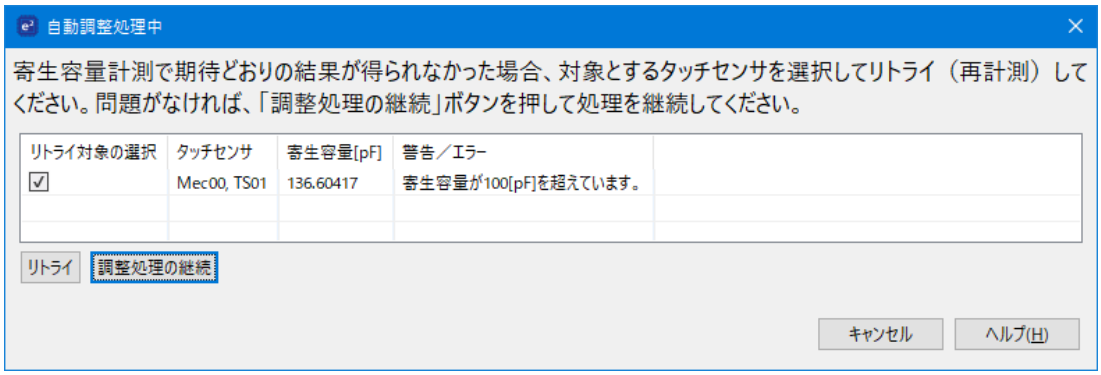
7.　調整

「10. 静電容量タッチセンサ・チューニング向けデバッグ構成の設定変更」章以降と同じ手順で静電容量タッチセンサ・チューニングが実行できます。

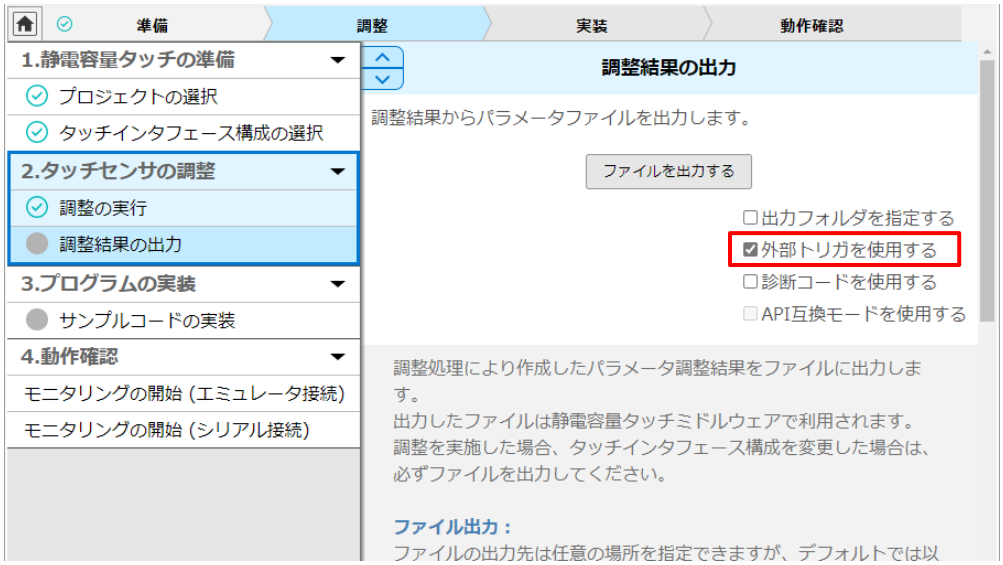
調整が完了すると以下のウインドウが表示されます。「調整処理の継続」(赤枠)をクリックして終了します。



途中で以下のような計測した寄生容量に対する警告が表示されることがありますが、「調整処理の継続」をクリックして先へ進めてください。



ワークフローの「調整結果の出力」を選択し、「外部トリガを使用する」(赤枠)にチェックを入れて、「ファイルを出力する」ボタンをクリックしてください。

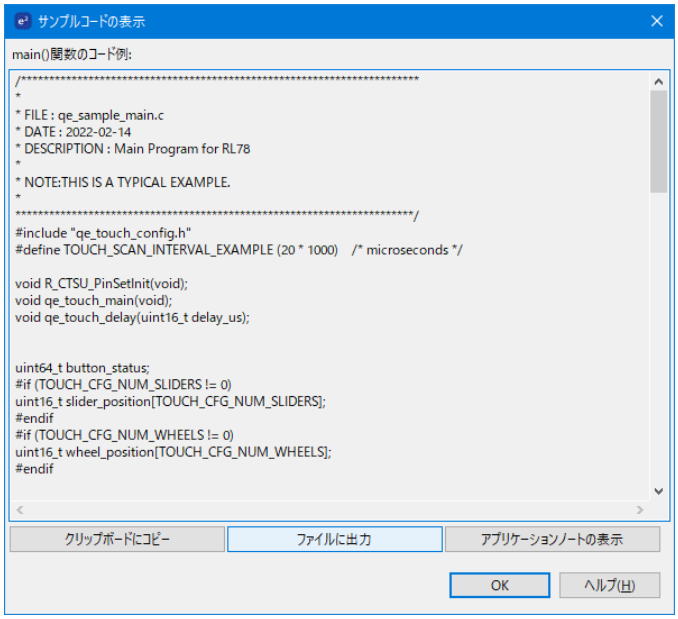


これによりプロジェクトに qe_gen フォルダが作成され、そこに設定等が定義されたソースコードを出力します。

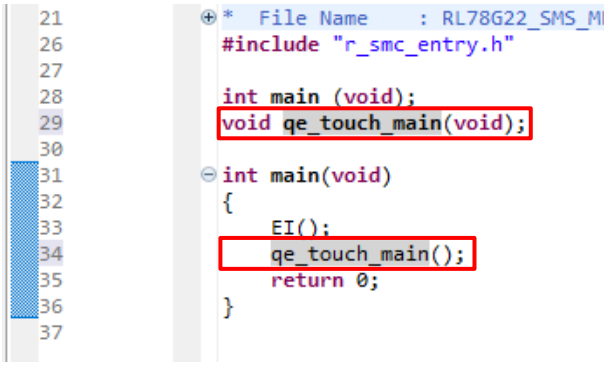
8.　サンプルコード出力

ワークフローより「サンプルコードの実装」を選択し、「例を表示する」ボタンをクリックします。

以下のように qe_touch_main()関数のコード例が表示され、qe_gen フォルダ以下に出力することも可能ですが、ここで出力されるコードは外部トリガや、RL78 の STOP モードに対応したコードではないため、ファイルに出力ボタンで qe_gen/qe_touch_sample.c を出力したうえで、ソースコード(qe_touch_sample.c)を上書きしてください。



qe_touch_main()関数の呼び出しを main()関数に追加(赤枠)します。



サンプルコード上書きの詳細

```
void qe_touch_main(void)
{
    fsp_err_t err;
    uint16_t b_status = 0;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();

    /*    LED     */
    /* P62 GPIO Low */
    PM6_bit.no2 = 0;
    LED2 = LED_OFF;
    /* P63 GPIO Low */
    PM6_bit.no3 = 0;
    LED3 = LED_OFF;

    /* P140 setting */
    PMCT14_bit.no0 = 0;
    PM14_bit.no0 = 0;
    P14_bit.no0 = 0;

    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    MK2H |= 0x40;    /* Disable interrupt servicing of "write request interrupt for setting registers for each channel" */
    MK3L |= 0x01;    /* Disable interrupt servicing of "measurement data transfer request interrupt" */

    /* Select the event source for the CTSU in the ELCL */
    ELSELR10 = 0x06;    /* ELCITL0(32bit interval timer0) -> CTSU */
```

Initialize CTSU PINS

Initialize CPU BOARD LED

Initialize PORT using SMS

Reduce standby power consumption

ELC setting for CTSU measurement triggered by interval timer

```
    /* initial offset tuning */
    /* for [CONFIG01] configuration */
    err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    g_qe_touch_flag = 0;

    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    R_Config_ITL000_Start();

    /* Main loop */
    while (true)
    {
        __stop();

        ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            R_Config_ITL000_Stop();
            break;
        }
    }

    button_status = 0;
    g_qe_touch_flag = 0;

    LED2 = LED_ON;
    LED3 = LED_ON;
```

Initial Offset tuning Process

Flag clear

CPU BOARD LED turn on

```c
/* normal snooze mode */
/* for [CONFIG01] configuration */
err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
if (FSP_SUCCESS != err)
{
    while (true) {}
}

g_qe_touch_flag = 0;

ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

R_Config_ITL000_Start();

while (true)
{
    __stop();

    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    while (0 == g_qe_touch_flag) {}
    g_qe_touch_flag = 0;

    err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
    if (FSP_SUCCESS == err)
    {
        if(1 == button_status)
        {
            LED2 = LED_ON;
            LED3 = LED_OFF;
            b_status = 1;
        }

        if (b_status == 1)
        {
            if (!button_status)
            {
                R_Config_ITL000_Stop();
                LED2 = LED_ON;
                LED3 = LED_ON;
                b_status = 0;
                break;
            }
        }
    }
}
```

SNOOZE mode scan process

```c
/* SMS setting */
err = RM_TOUCH_SmsSet(g_qe_touch_instance_config01.p_ctrl);
err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
if (FSP_SUCCESS != err)
{
    while (true) {}
}

R_Config_INTC_INTP1_Start();
R_Config_ITL000_Start();

while (true)
{
    while (true)
    {
        __stop();

        /* When the threshold is exceeded, callback function is called. */
        if (g_qe_touch_flag)
        {
            break;
        }
    }
    err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
    if (FSP_SUCCESS == err)
    {
        if (!button_status)
        {
            LED2 = LED_ON;
            LED3 = LED_OFF;

        }
```

SNOOZE + SMS mode Scan process

LED extinguishing process after waking up from SNOOZE + SMS mode

```c
            RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
            LED2 = LED_ON;
            LED3 = LED_ON;

            while (true)
            {
                __stop();

                ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

                while (0 == g_qe_touch_flag) {}    /* Wait until the interrupt servicing of "measurement end interrupt" is completed */
                g_qe_touch_flag = 0;

                err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
                if (FSP_SUCCESS == err)
                {
                    if (button_status)
                    {
                        LED2 = LED_OFF;
                        LED3 = LED_ON;
                        b_status = 1;
                    }
```

Snooze mode scan

LED turn off process in ActiSnooze mode scan

```
                    if (b_status == 1)
                    {
                        if (!button_status)
                        {
                            R_Config_ITL000_Stop();
                            R_Config_INTC_INTP1_Stop();
                            LED2 = LED_ON;
                            LED3 = LED_ON;
                            b_status = 0;

                            /* Start SMS measurement */
                            RM_TOUCH_SmsSet(g_qe_touch_instance_config01.p_ctrl);
                            err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
                            if (FSP_SUCCESS != err)
                            {
                                while (true) {}
                            }

                            R_Config_INTC_INTP1_Start();
                            R_Config_ITL000_Start();
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

Return processing to SNOOZE + SMS mode

19. 参考ドキュメント

RL78/Gxx ユーザーズマニュアル ハードウェア編
RL78 ファミリ ユーザーズマニュアル ソフトウェア編 (R01US0015)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

テクニカルアップデート／テクニカルニュース
(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

アプリケーションノート RL78 ファミリ 静電容量センサユニット（CTSU2L）動作説明 (R01AN5744)
アプリケーションノート RL78 ファミリ CTSU モジュール Software Integration System (R11AN0484)
アプリケーションノート RL78 ファミリ TOUCH モジュール Software Integration System (R11AN0485)
アプリケーションノート 静電容量センサマイコン 静電容量タッチ電極デザインガイド (R30AN0389)
アプリケーションノート RL78 ファミリ
RL78/G23 静電容量タッチ低消費電力ガイド(SNOOZE 機能) (R01AN5886)
アプリケーションノート RL78/G23 グループ
静電容量タッチ低消費電力ガイド(SMS 機能) (R01AN6670)
アプリケーションノート RL78/G22 静電容量タッチ低消費電力ガイド(SMS/MEC 機能) (R01AN6847)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://www.renesas.com/

静電容量センサユニット関連ページ
https://www.renesas.com/solutions/touch-key
https://www.renesas.com/qe-capacitive-touch

お問い合わせ
http://www.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.4.30	-	初版発行
2.00	2021.8.31	-	開発環境を更新
2.10	2022.5.20	-	「8. [追加機能] UART を使用したシリアル通信モニタの設定(1/3)」章を更新。 「15. [追加機能] UART を使用したシリアル通信モニタの設定(3/3)」章を更新。 「17. [付録] ハードウェアタイマでのタッチ計測」章を追加。
3.00	2023.5.22	-	「18. [応用例] 自動判定機能(SMS 使用)と MEC 機能の設定」章を追加。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。