RA ファミリ

QEとFSP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、RA 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

静電容量式タッチセンサユニット(CTSU、CTSU2)をサポートする RA ファミリ

目次

1. 概要

本アプリケーションノートでは、RA 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際に行う、以下の手順について説明します。

- RA6M2 MCU ボードまたは RA2L1 MCU ボードを使用した RA 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。本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 / RA スマート・コンフィグレータ、Flexible Software Package (FSP)のドライバ/ミドルウェア、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. 開発手順の概要

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。

1. 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。
2. RA 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。
3. 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。
4. 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。
5. 必要な FSP 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。
6. 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

本プロジェクトでは以下の開発環境を使用します。

表1　開発環境

開発ツール ソフトウェアコンポーネント	RA6M2 (CTSU)	RA2L1 (CTSU2)
ボード・キット	RA6M2 MCU グループ評価キット (EK-RA6M2)	RA2L1 搭載 静電容量タッチ評価システム (RTK0EG0022S01001BJ)
統合開発環境 e^2 studio	2023-07 以降	
GCC ARM Embedded compiler	12.2.1.20230214 以降	
Renesas QE for Capacitive Touch	3.3.0 以降	
Flexible Software Package (FSP)	4.6.0 以降	

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。

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「7. 変更後のqe_touch_sample.cのリスト」を参照してください。

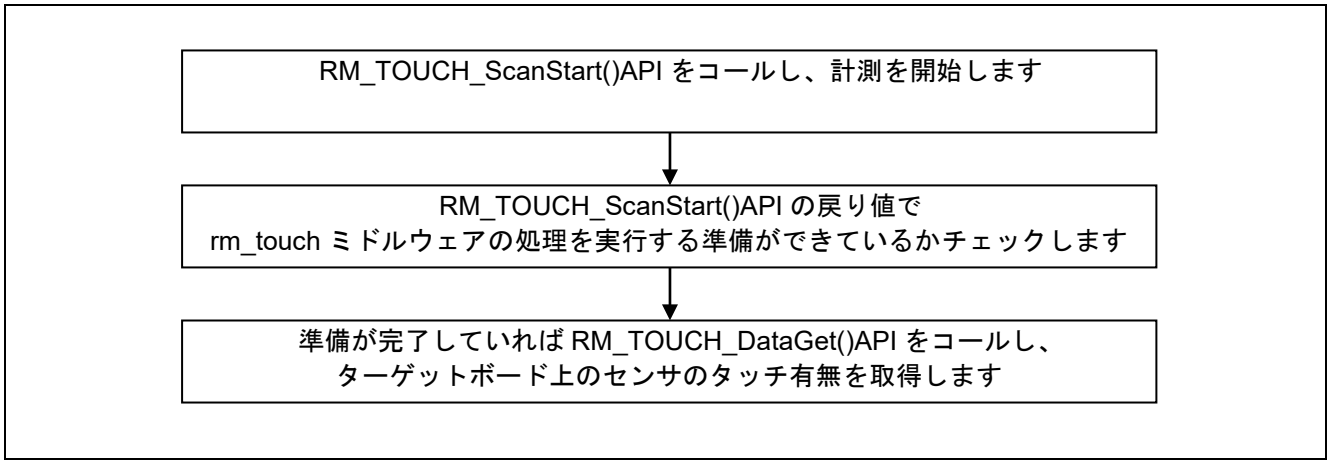

図1　アプリケーション例の概要

6. 静電容量タッチアプリケーション開発手順

6.1 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 e^2 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. e^2 studio のメニュー[ファイル] – [新規] – [Renesas C/C++ Project] – [Renesas RA]を選択し、新規プロジェクトの作成を開始します。

3. [New C/C++ Project]ダイアログが表示されたら”Renesas RA C/C++ Project”を選択し、[次へ]をクリックします

4. 次の[Renesas RA C/C++ Project]ウィザードの[Project Name and Location]ページで[Project name]に任意のプロジェクト名を入力します。本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[Device and Tools Selection]ページでは、以下を選択します。

表2　デバイス、ツールチェーン選択

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
FSP Version	4.6.0	
Board	EK-RA6M2	RSSK-RA2L1
Device	R7FA6M2AF3CFB	R7FA2L1AB2DFP
Toolchains	GNU ARM Embedded 12.2.1.20230214	
Debugger	J-Link ARM	J-Link ARM, E2 (ARM), E2 Lite (ARM)のいずれか

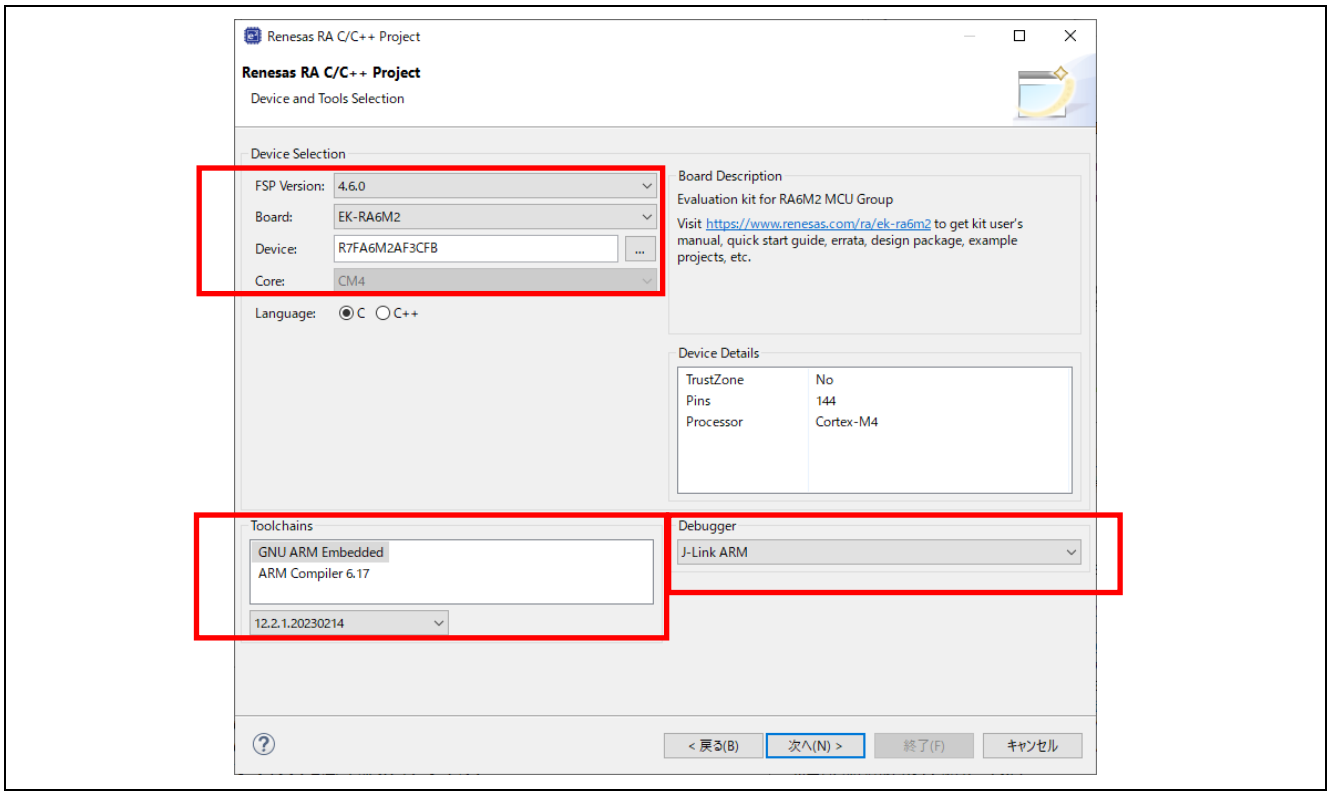

図2　デバイス、ツールチェーン（RA6M2）

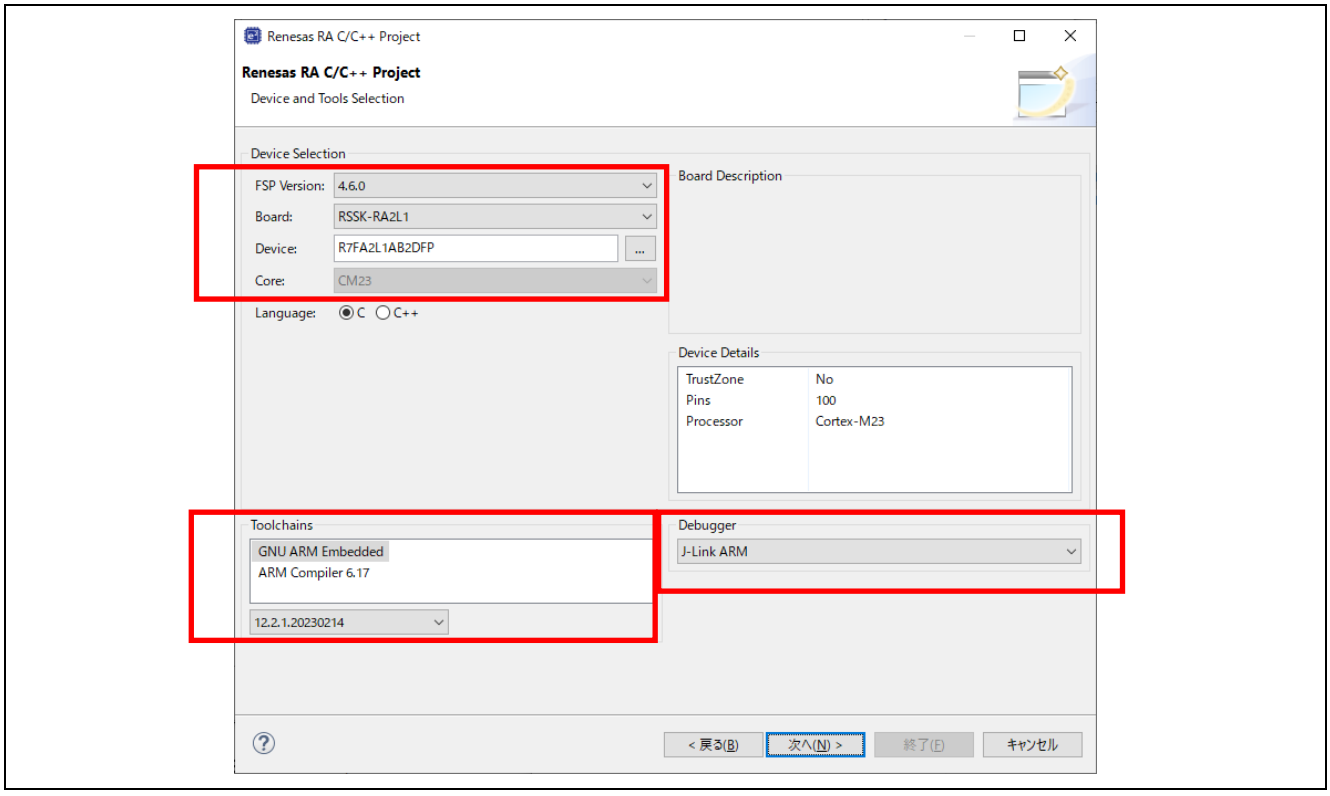

図3　デバイス、ツールチェーン（RA2L1）

6.　完了したら、[次へ]をクリックします。

7. 次の[Build Artifact and RTOS Selection]ページでは、以下を選択します。

表3　ビルド、RTOS 選択

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
Build Artifact Selection	Executable	
RTOS Selection	No RTOS	

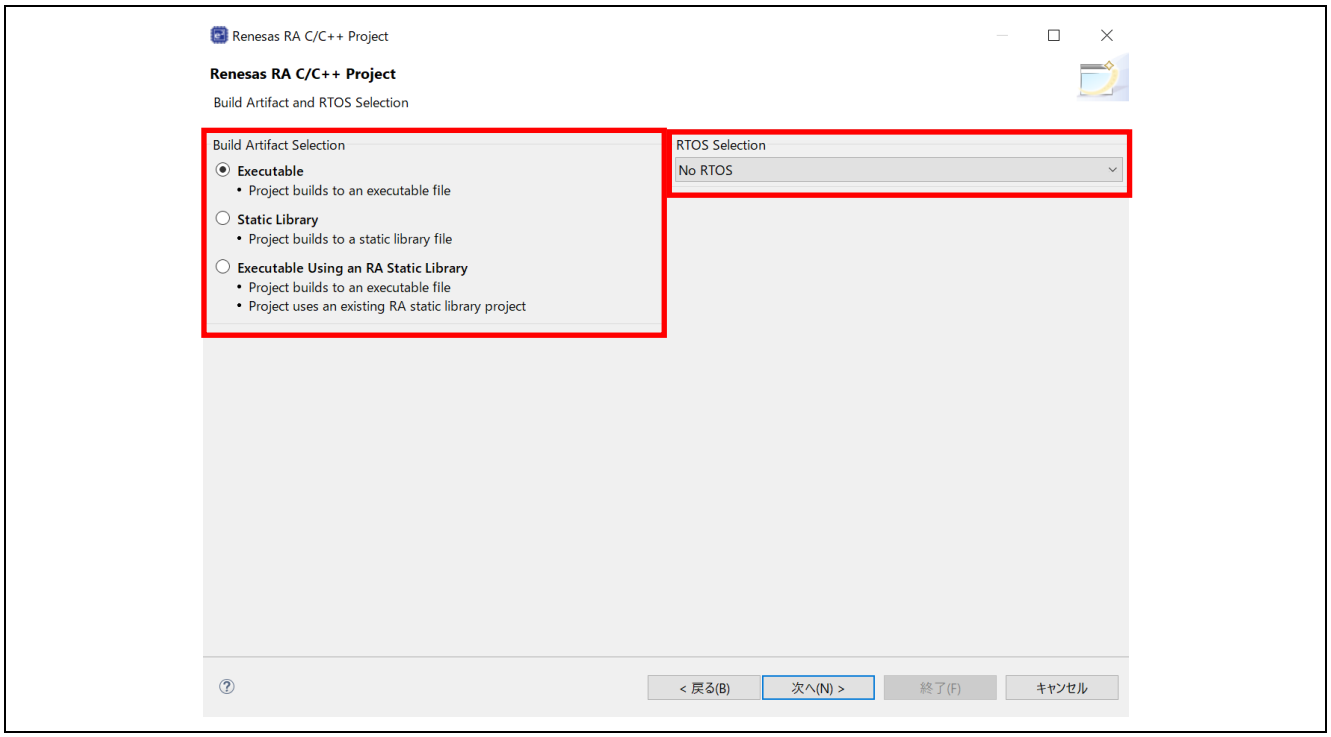

図4　ビルド、RTOS 選択

8. 完了したら、[次へ]をクリックします。

9. 次の[Project Template Selection]ページでは、”Bare Metal - Minimal”をチェックし、[終了]をクリックします。

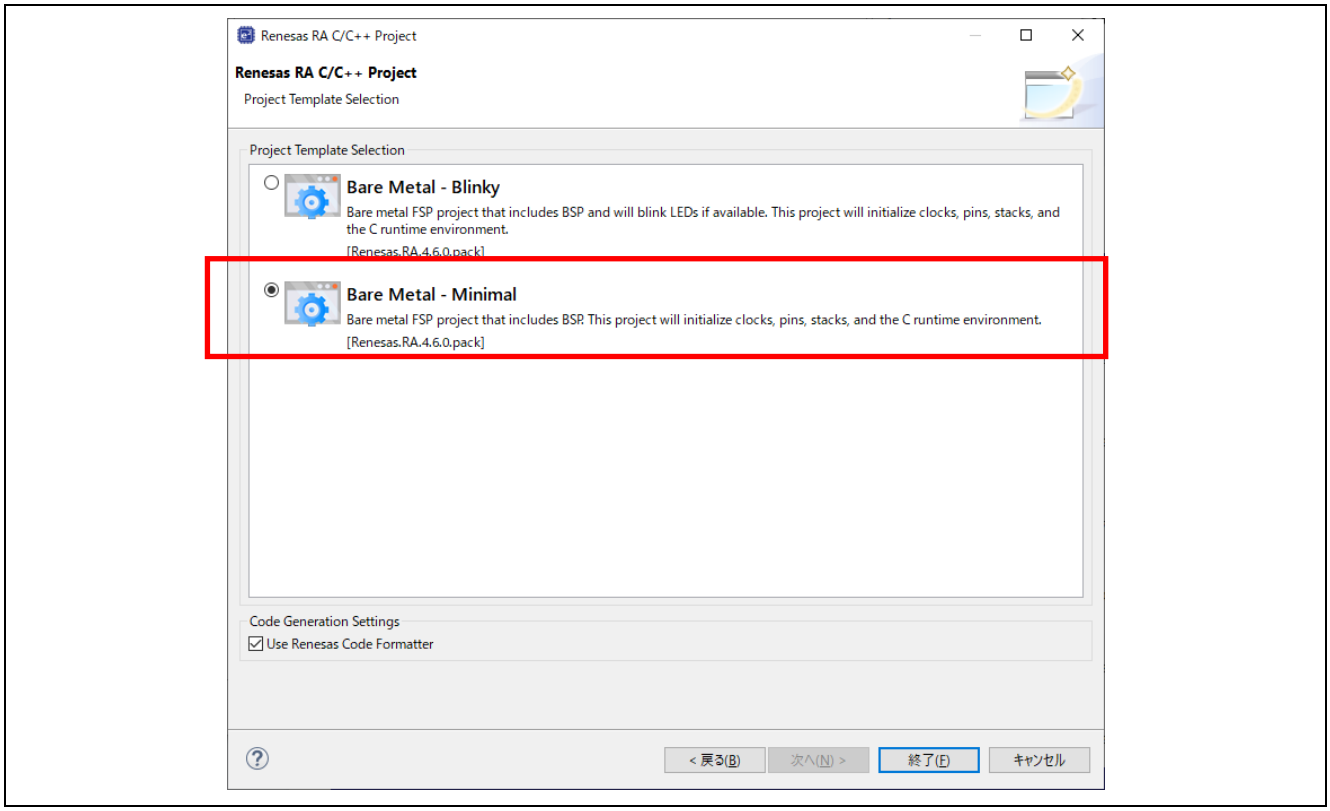

図5　プロジェクトテンプレート選択

完了すると e² studio のデフォルトウィンドウが RA 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。

6.2 RA 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 の中央下部から[BSP]タブを選択し、BSP 構成を表示します。

図6　[BSP]タブ

2. e² studio の左下に表示される[プロパティー] – [Settings] – [RA Common] – [MCU Vcc (mV)]から電源供給電圧を設定します。本アプリケーション例では”3300”(mV)に設定しています。

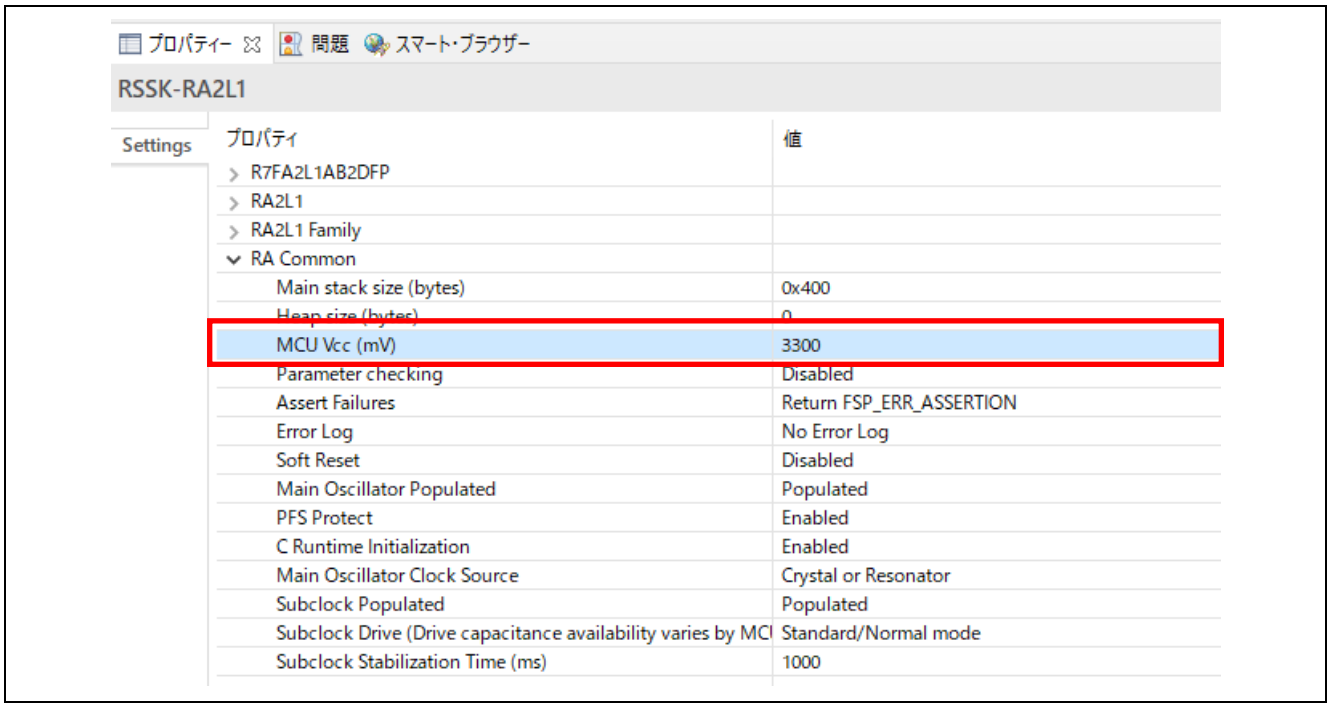

図7　電源供給電圧

3. 次に[Clocks]タブを選択し、クロック設定を表示します。

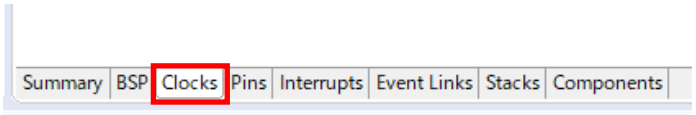

図8　[Clocks]タブ

4. 本アプリケーション例では、以下を選択します。

表4　クロック設定

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
PLL Src	XTAL	-（設定項目なし）
Clock Src	PLL	HOCO
PCLKB	PCLKB Div /4	PCLKB Div /2

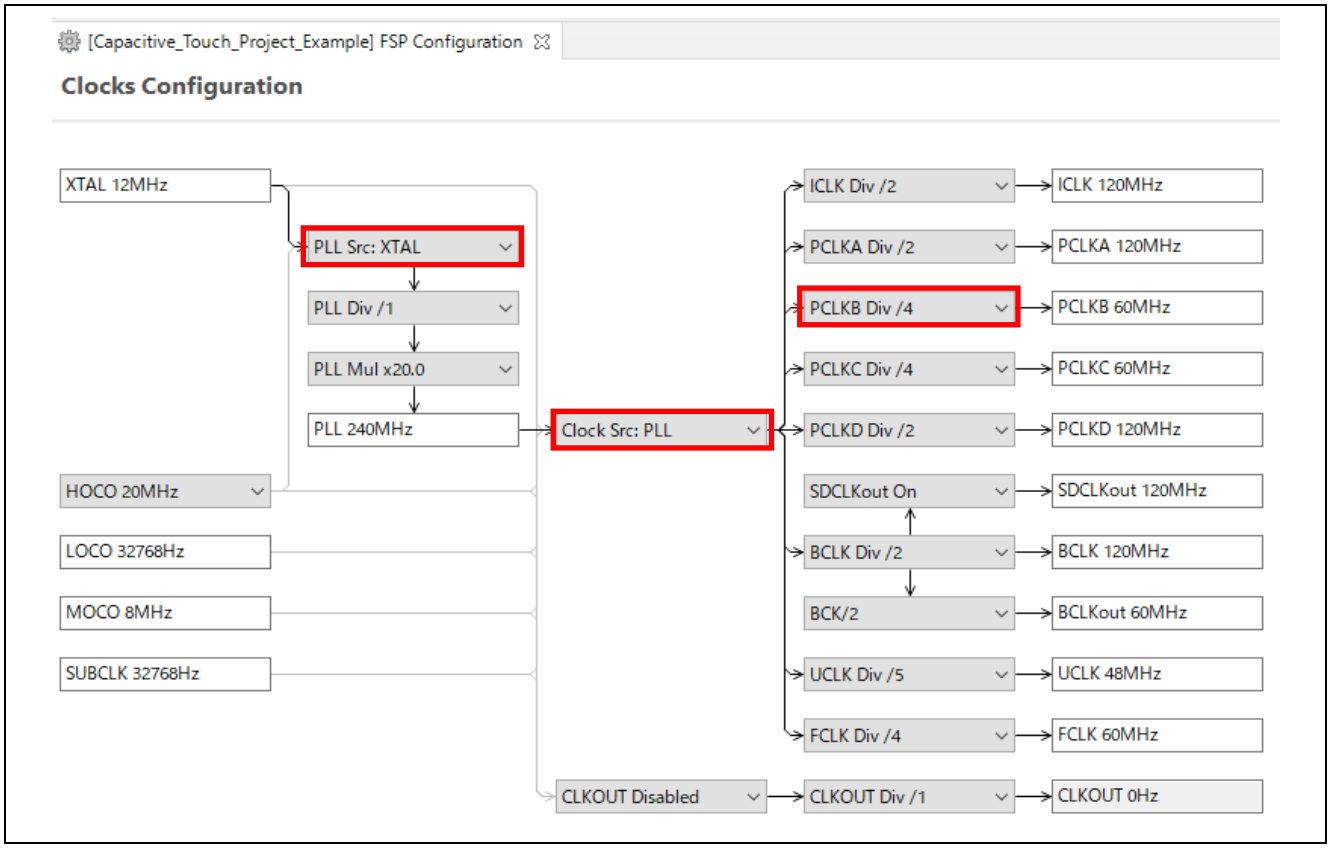

図9　クロック設定（RA6M2）

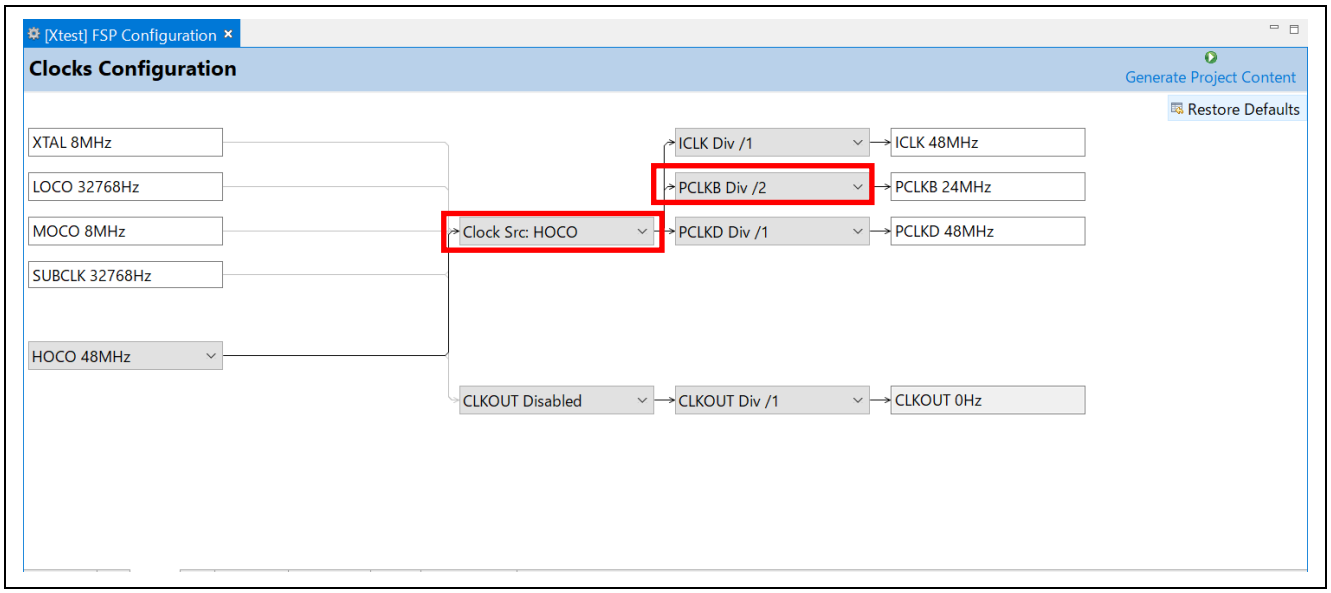

図10　クロック設定（RA2L1）

5. 次に[Pins]タブを選択します。MCU のセンサポートと静電容量タッチアプリケーションボードを接続するための割り当てを行います。

   【注】：プロジェクト作成時、[Device Selection]の[Board]で選択した項目によっては、デフォルトで TS 端子が設定済みです。

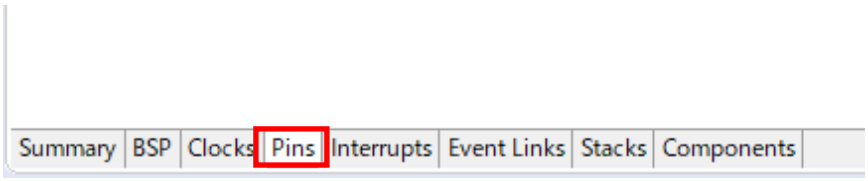

図11　[Pins]タブ

6. [Pin Selection]の[Peripherals]から[Input:CTSU]を開き、[CTSU0]を選択します。

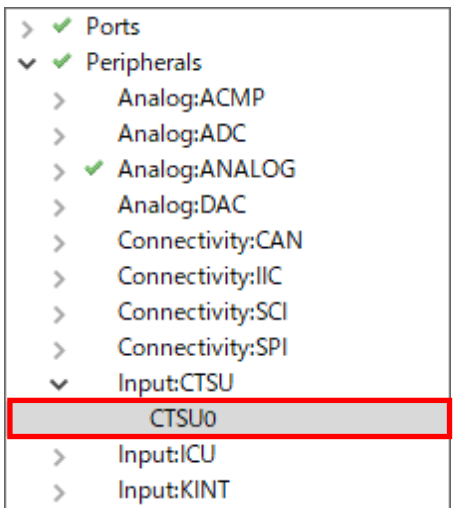

図12　Peripherals 選択（CTSU0）

7. [Pin Configuration]の[Operation Mode]を”Disabled”から”Enabled”に変更します。

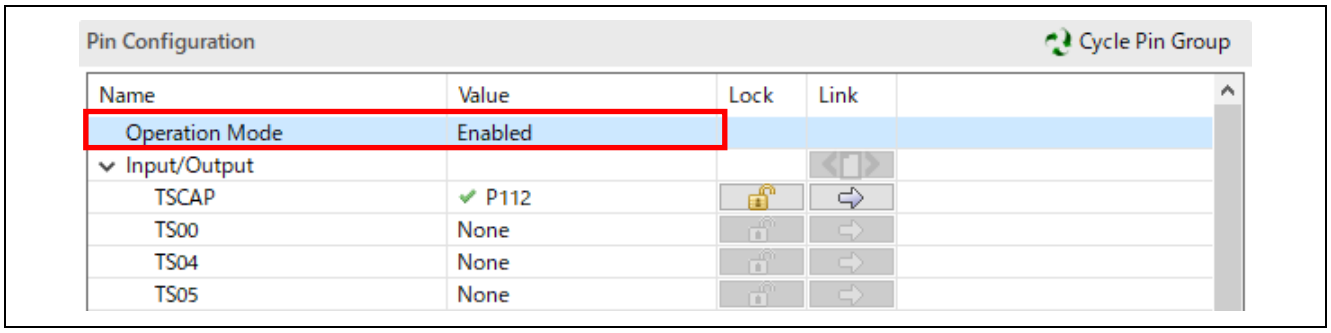

図13　[Operation Mode]（CTSU0）

8. 使用する TS 端子を有効にします。RA2L1（CTSU2）では非計測の TS 端子の出力を一括で制御できるため、全ての TS 端子を有効にします。なお、本アプリケーション例で静電容量タッチインタフェースなどに使用する TS 端子は以下のとおりです。

表5　使用する TS 端子

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
使用する TS 端子	● TSCAP 端子 ● TS2 端子	● TSCAP 端子 ● TS0 端子 ● TS11-CFC 端子

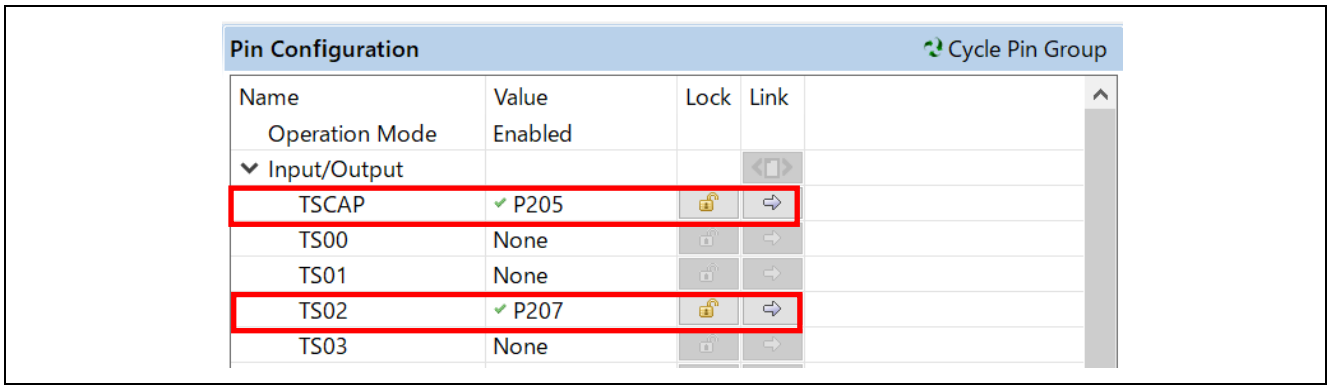

図14　TS 端子設定（RA6M2）

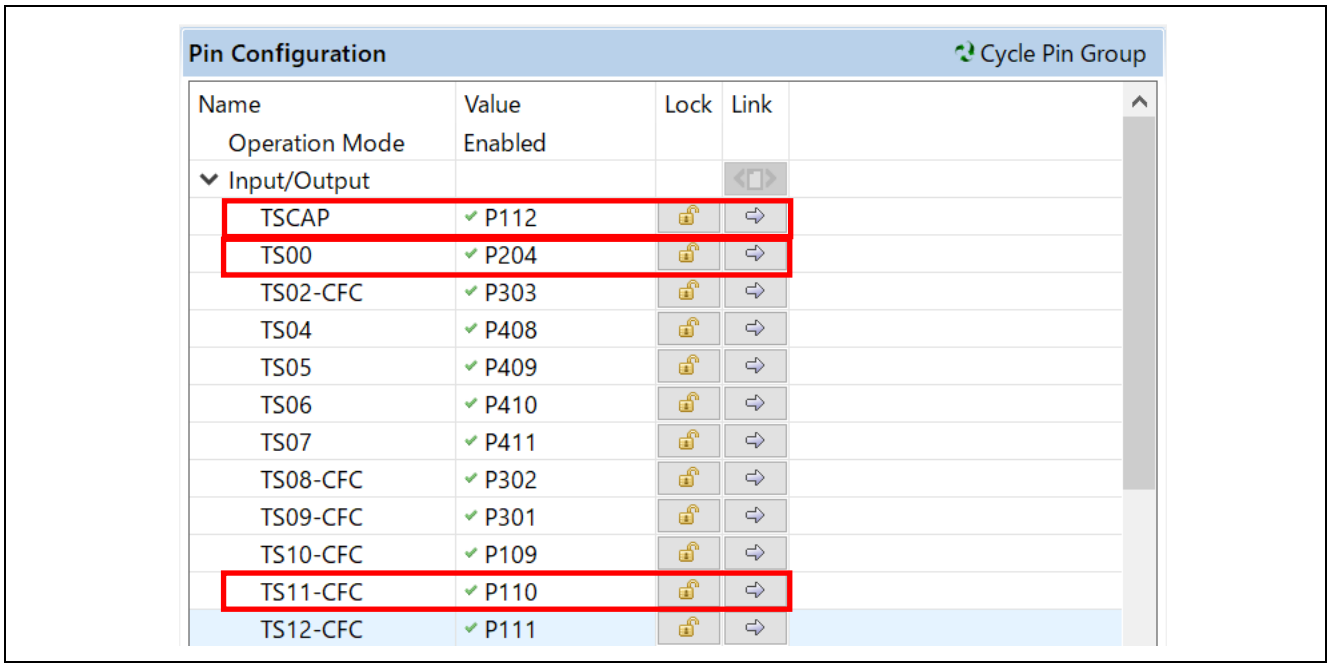

図15　TS 端子設定（RA2L1）

9. RA6M2（CTSU）ではアプリケーションで使用しないセンサポートをスタートアップで Low 出力駆動に設定することを推奨します。[Pin Selection]の[Ports]から[P2] – [P206]を選択し、[Mode]を”Disable”から”Output mode (Initial Low)”に変更します。

   【注】：ここでは使用例として一部のポートのみ設定しています。使用しないすべてのセンサポートに対して同様の設定をしてください。

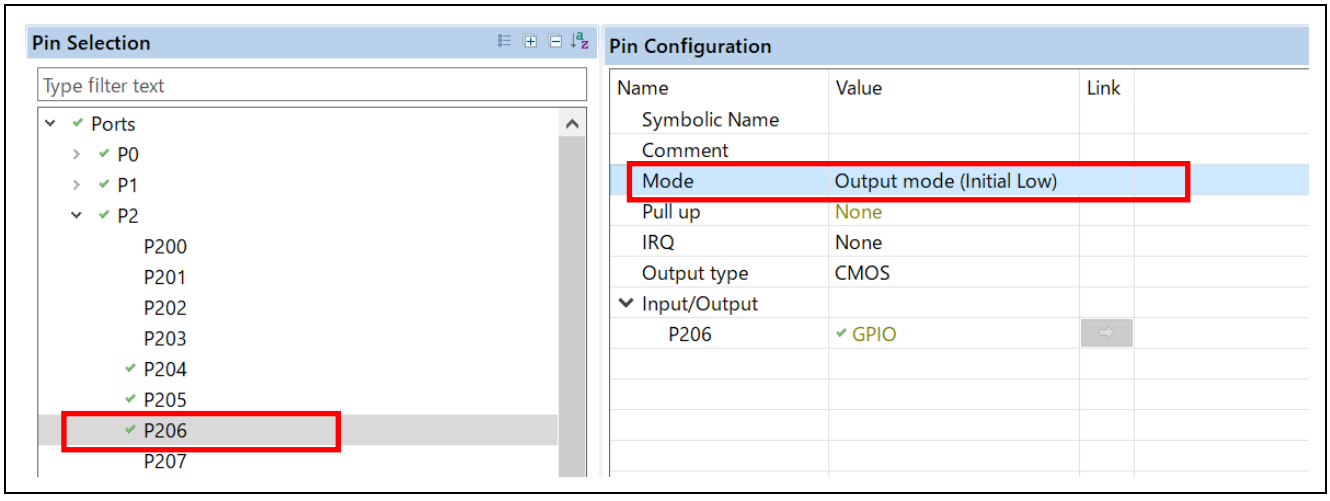

図16　Low 出力駆動設定

10. [Pin Selection]の[Peripherals]から[Connectivity:SCI]を開き、[SCI9]を選択します。

    【注】：10~11項はシリアル通信によるモニタリングを使用する場合のみ設定してください。シリアル通信によるモニタリングを使用しない場合は12項へ進んでください。

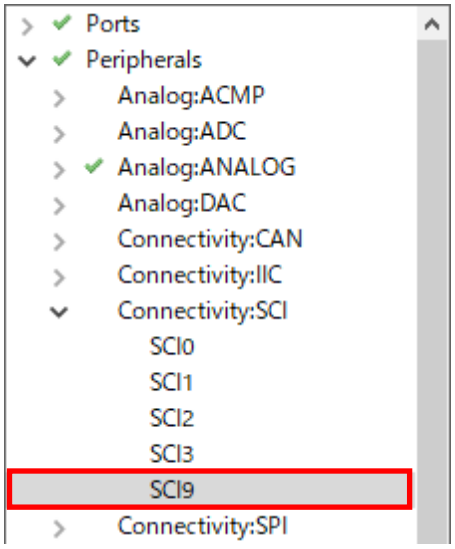

図17　Peripherals 選択（SCI9）

11. [Pin Configuration]の[Operation Mode]を”Disabled”から”Asynchronous UART”に変更します。また、[TXD]に”P203”、[RXD]に”P202”が選択されていることを確認します。

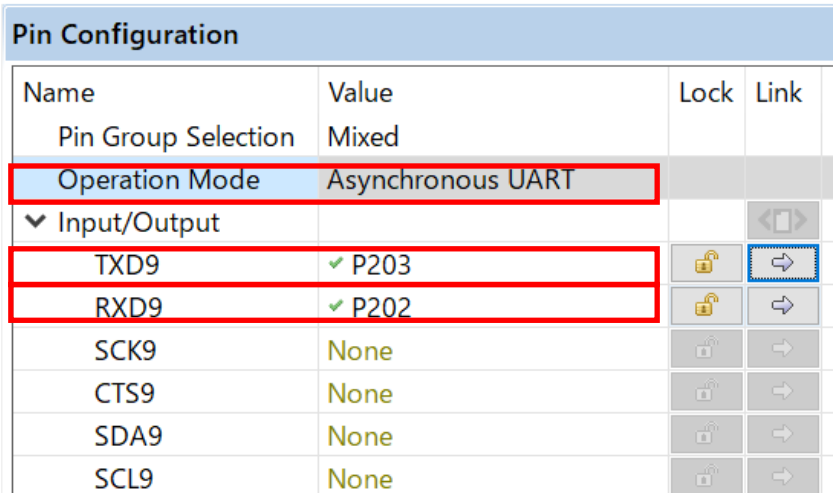

図18　[Operation Mode]（SCI9）

12. 次に[Stacks]タブを選択します。

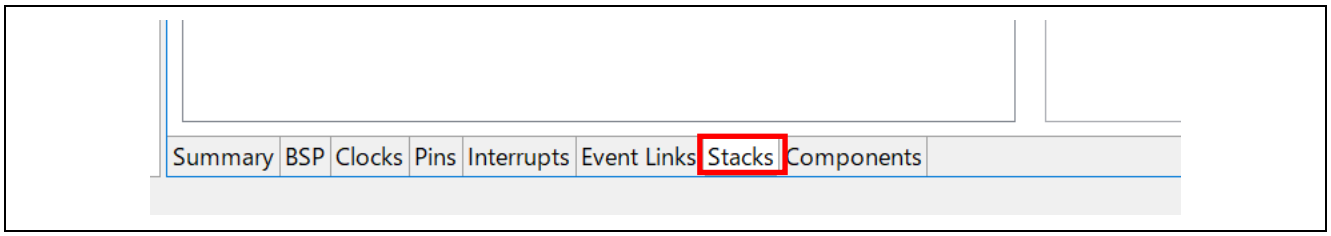

図19　[Stacks]タブ

13. [New Stack] -> [Cap Touch] -> [TOUCH (rm_touch)]を選択し、CTSU ドライバとミドルウェアを追加します。

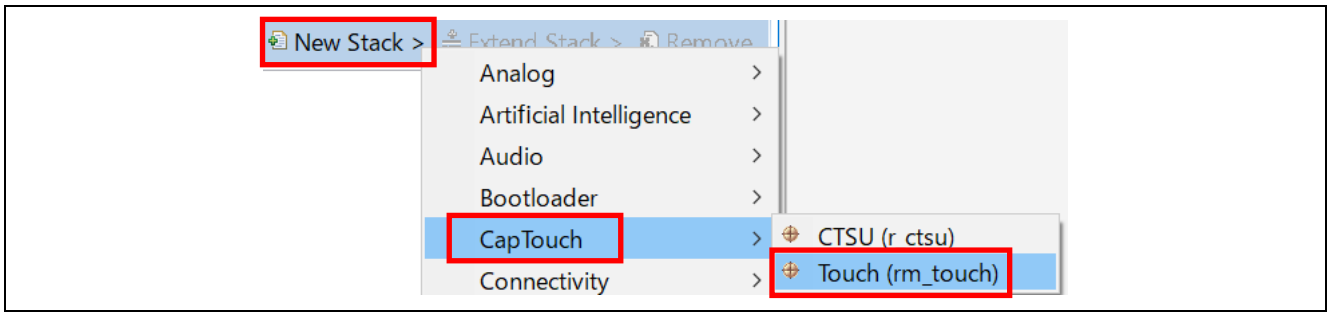

図20　CTSU ドライバ、ミドルウェア追加

14. 以下の図のように、[HAL/Common Stacks]が表示されます。

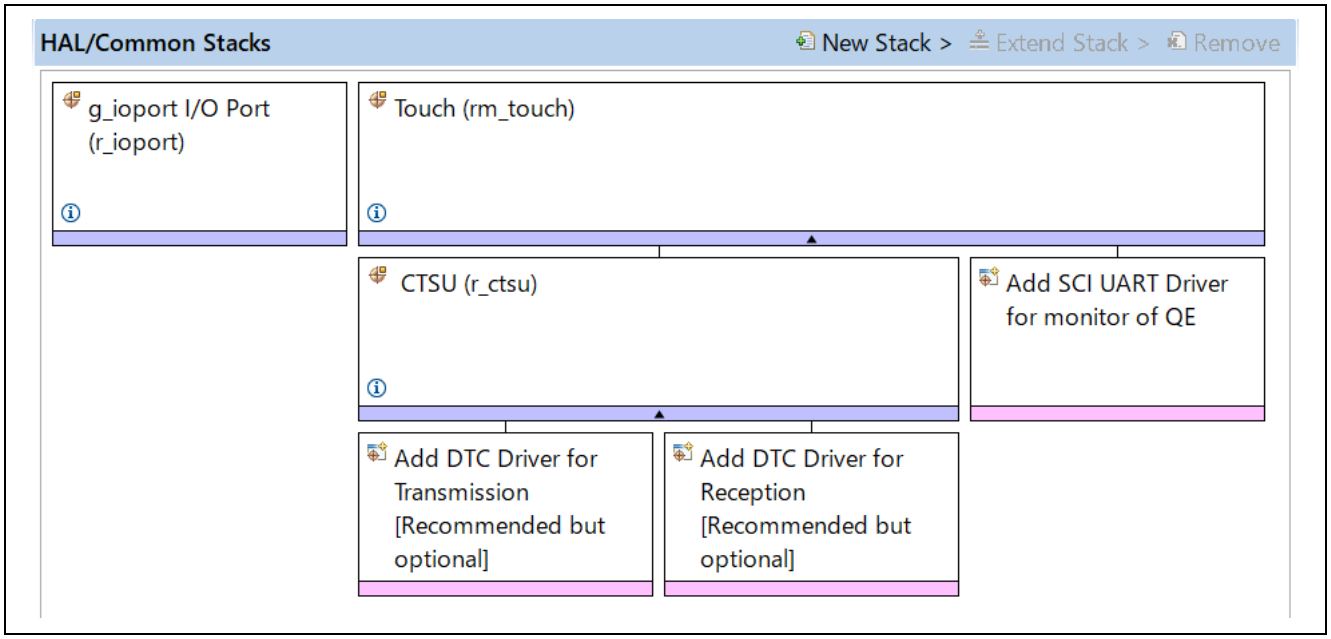

図21　[HAL/Common Stacks]（CTSU 追加後）

15. 次に[CTSU Driver on r_ctsu]モジュールをクリックして[プロパティー]を表示します。

    【注】：15~18項は DTC を使用する場合のみ設定してください。DTC により CPU を経由せずにメモリ、レジスタ間でデータを転送します。DTC を使用しない場合は19項へ進んでください。

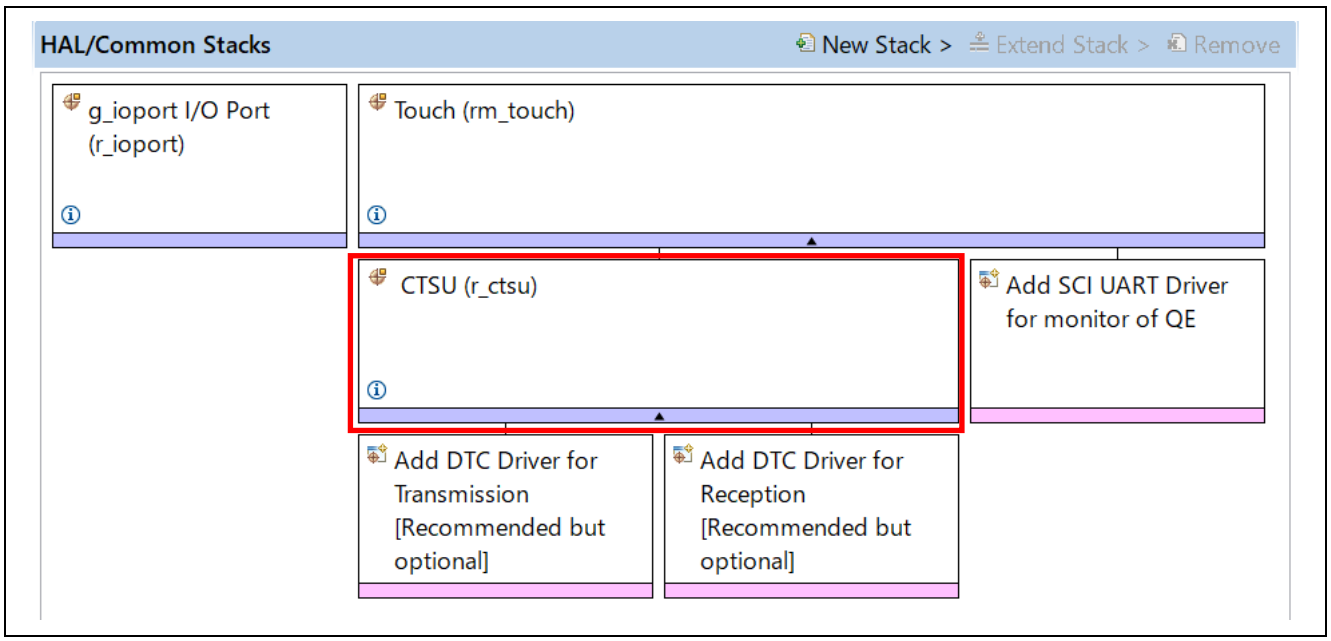

図22　[CTSU Driver on r_ctsu]モジュール

16. DTC の設定をします。画面左下に表示される[プロパティー] – [Settings] – [Common] – [Support for using DTC]を”Disabled”から”Enabled”に変更します。

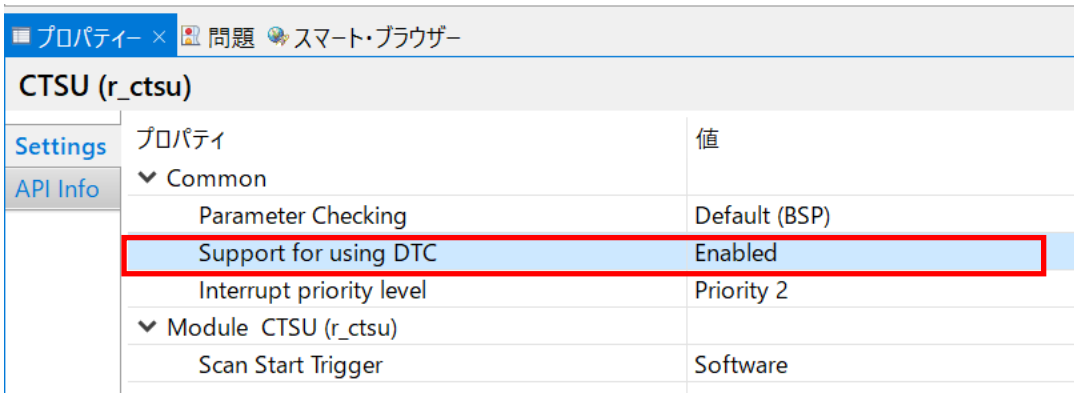

図23　[Support for using DTC]

17. [Add DTC Driver for Transmission]モジュールをクリックし、[New]->[Transfer (r_dtc)]を選択して送信用の DTC ドライバを追加します。

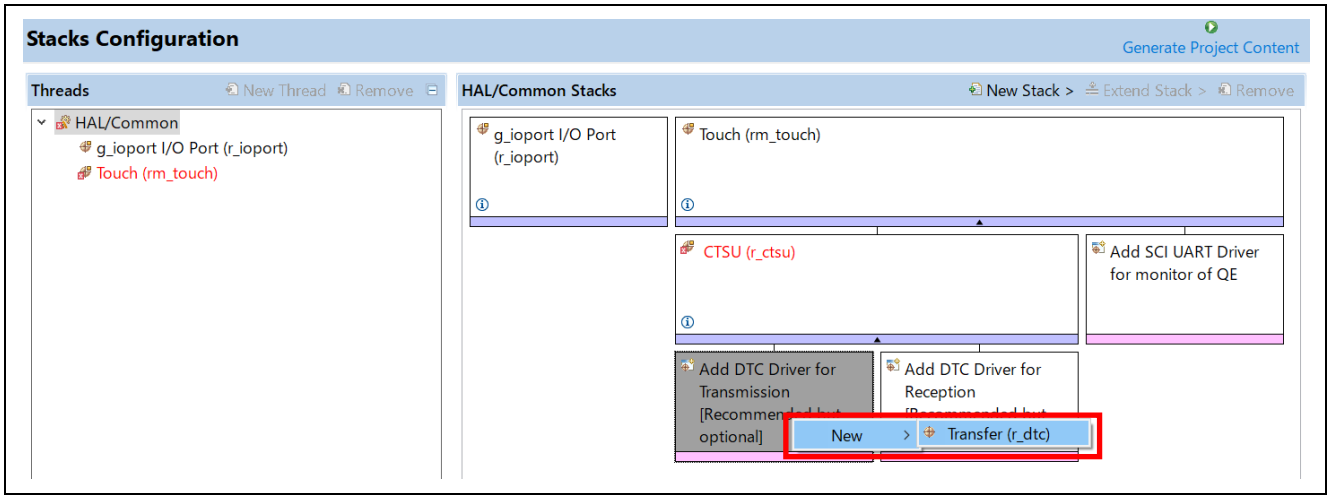

図24　送信用 DTC ドライバ追加

18. 同様に[Add DTC Driver for Reception]モジュールをクリックし、[New]->[Transfer (r_dtc)]を選択して受信用の DTC ドライバを追加します。

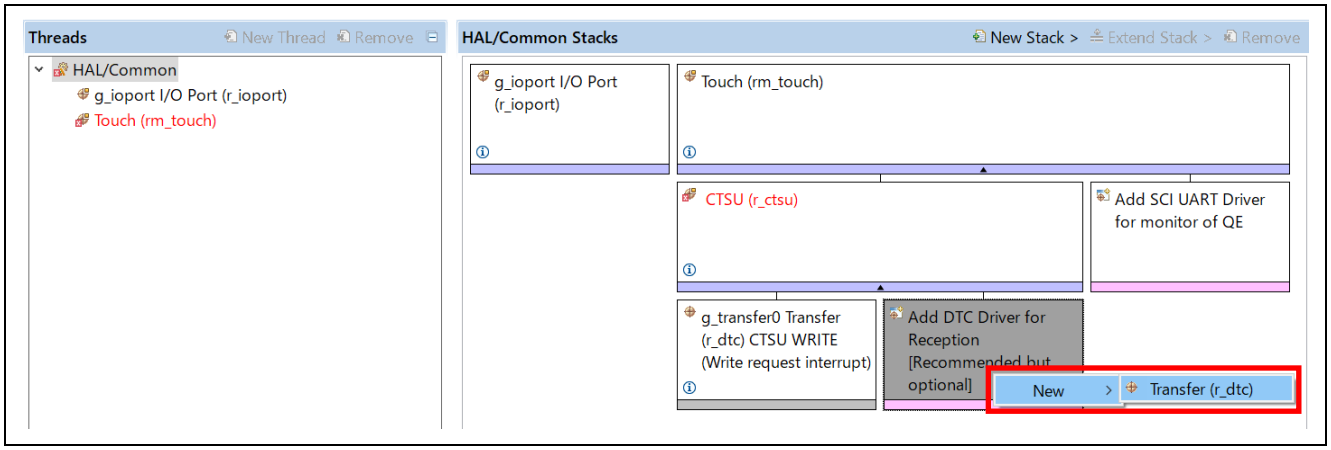

図25　受信用 DTC ドライバ追加

19. [TOUCH (rm_touch)]モジュールをクリックして[プロパティー]を表示します。

    【注】：19~26項はシリアル通信によるモニタリングを使用する場合のみ設定してください。シリアル通信によるモニタリングを使用しない場合は27項へ進んでください。

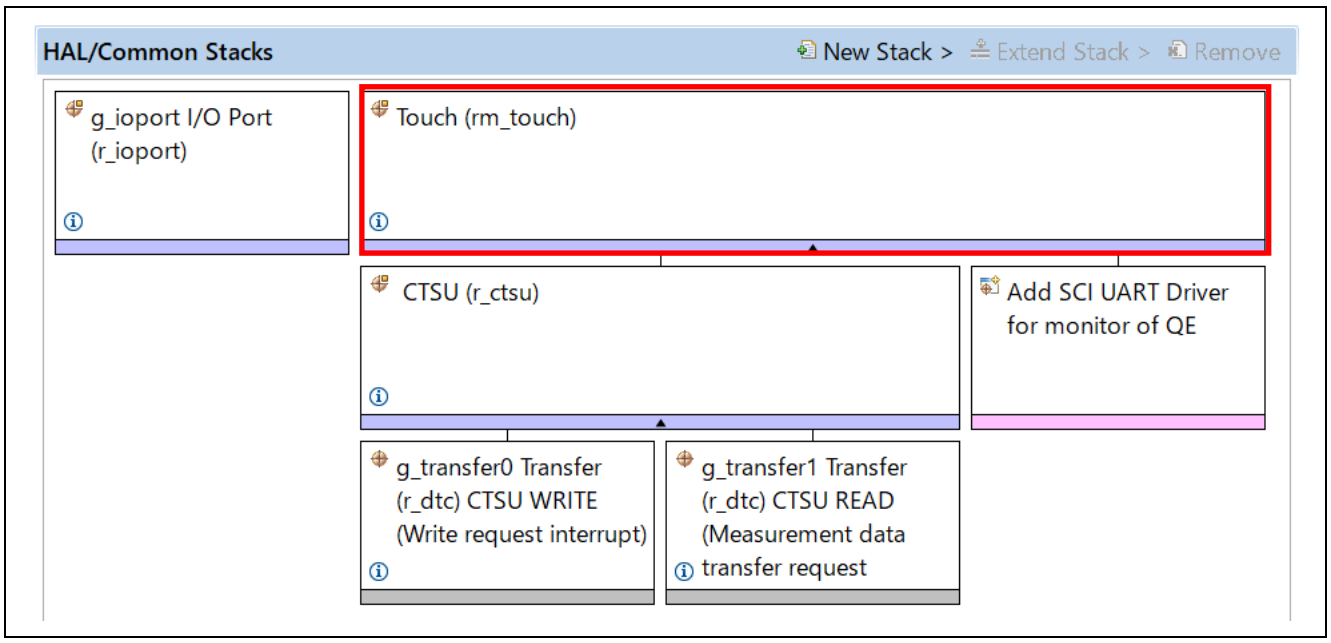

図26　[TOUCH Driver on rm_touch]モジュール

20. 画面左下に表示される[プロパティー] – [Settings] – [Common] – [Support for QE monitoring using UART]を”Disabled”から”Enabled”に変更します。

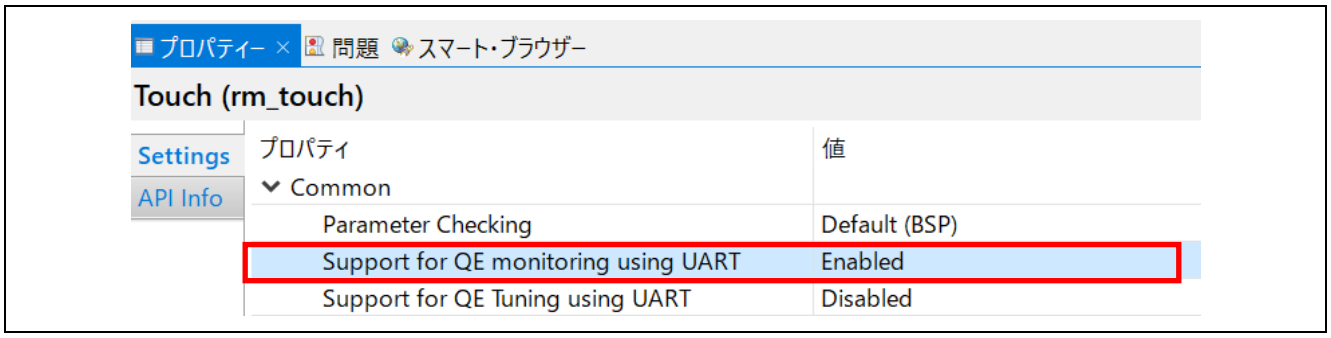

図27　[Support for QE monitoring using UART]

21. [Add SCI UART Driver]モジュールをクリックし、[New]->[UART (r_sci_uart)]を選択して UART ドライバを追加します。

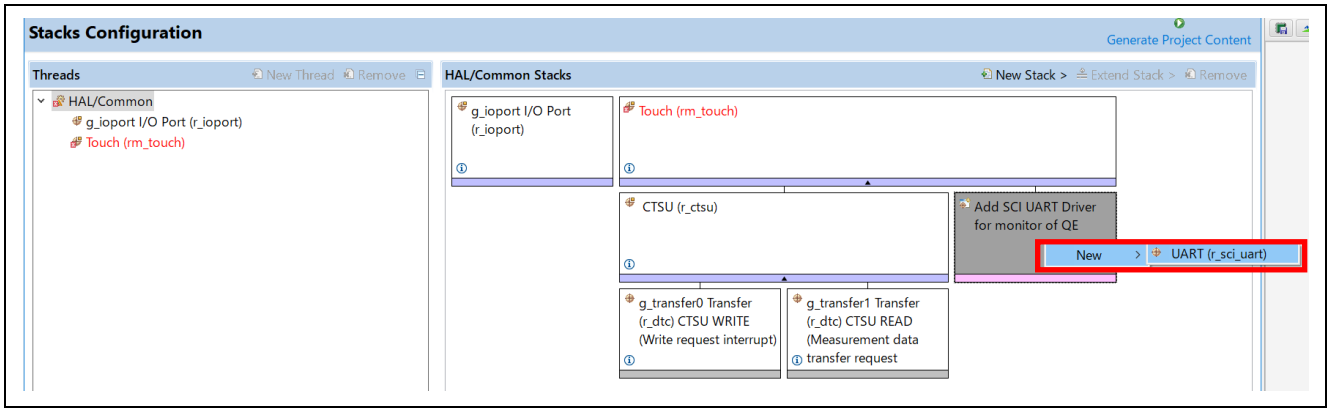

図28　UART ドライバ追加

22. 次に[g_uart_qe UART Driver on r_sci_uart]モジュールをクリックして[プロパティー]を表示します。

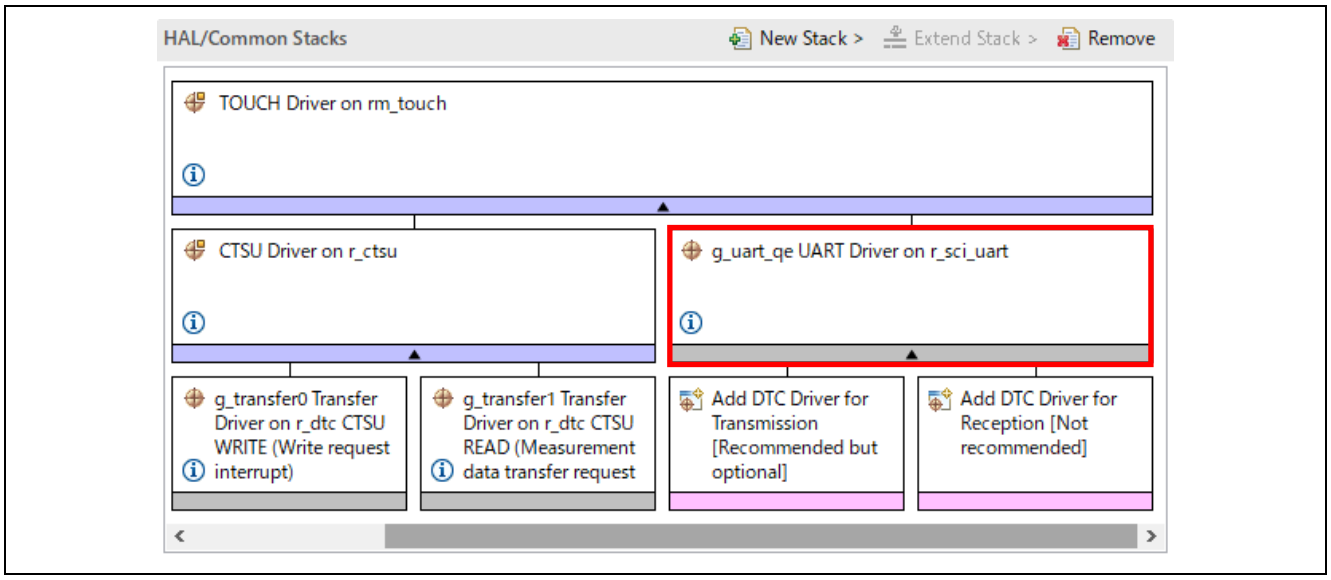

図29　[g_uart_qe UART Driver on r_sci_uart]モジュール

23. SCI のチャネルの設定をします。画面左下に表示される[プロパティー] – [Settings] – [Module g_uart_qe UART Driver on r_sci_uart] – [General] – [Channel]を”0”から”9”に変更します。

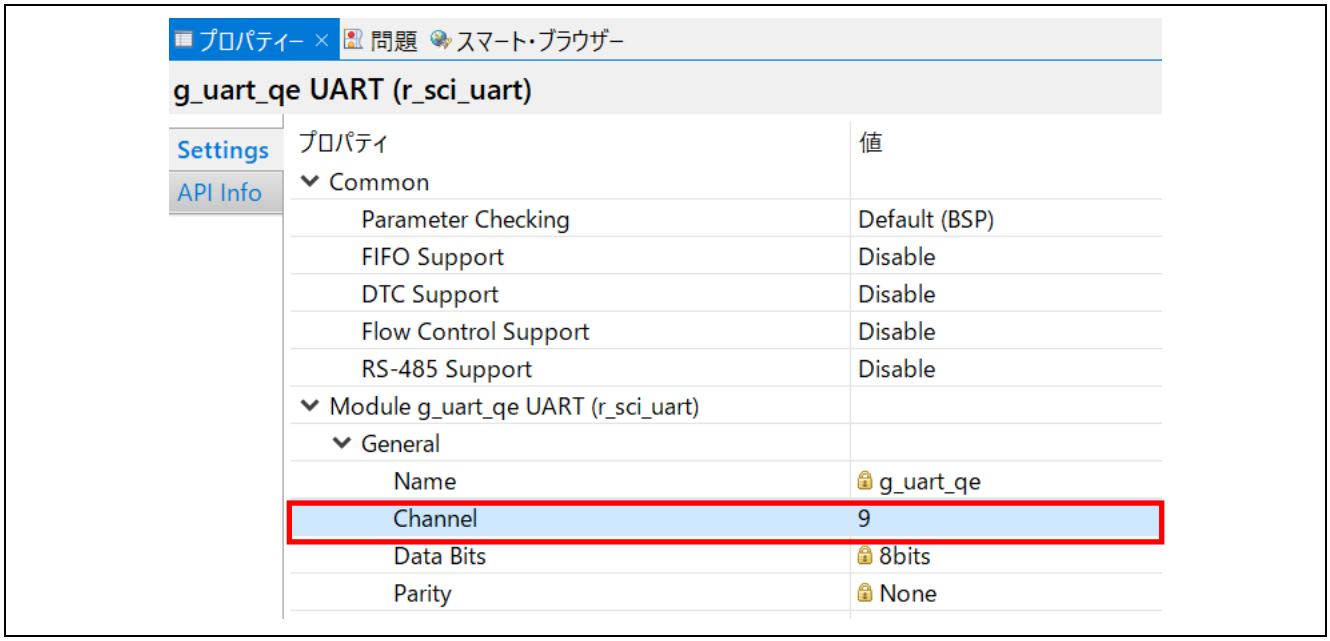

図30　[Channel]

24. 続けて[プロパティー]から DTC の設定をします。[Common] – [DTC Support]を”Disable”から”Enable”に変更します。

【注】：24~26項は DTC を使用する場合のみ設定してください。DTC により CPU を経由せずにメモリ、レジスタ間でデータを転送します。DTC を使用しない場合は27項へ進んでください。

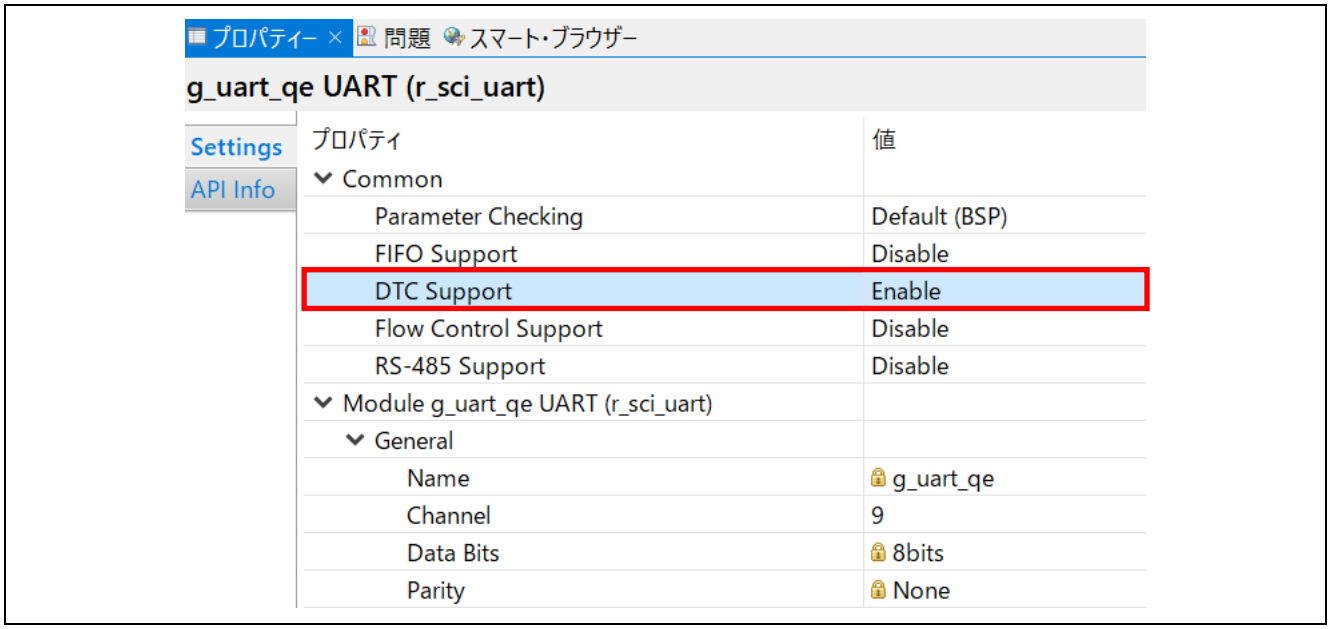

図31　[DTC Support]

25. [Add DTC Driver for Transmission]モジュールをクリックし、[New]->[Transfer (r_dtc)]を選択して送信用の DTC ドライバを追加します。

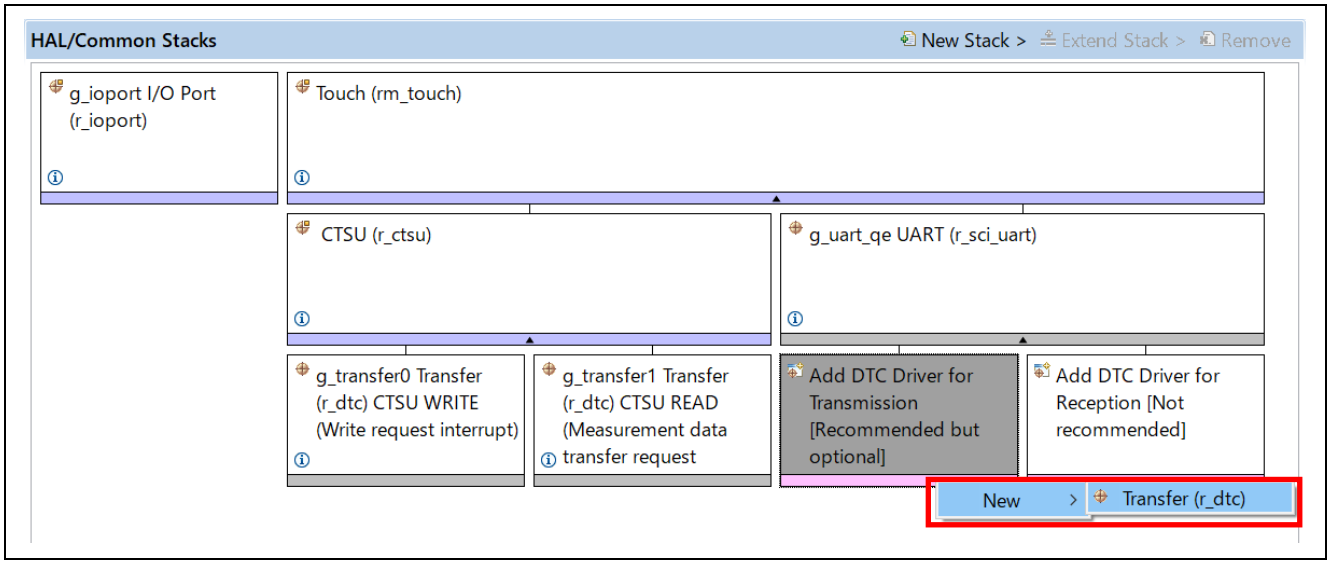

図32　送信用 DTC ドライバ追加（SCI9）

26. 同様に[Add DTC Driver for Reception]モジュールをクリックし、[New]->[Transfer (r_dtc)]を選択して受信用の DTC ドライバを追加します。

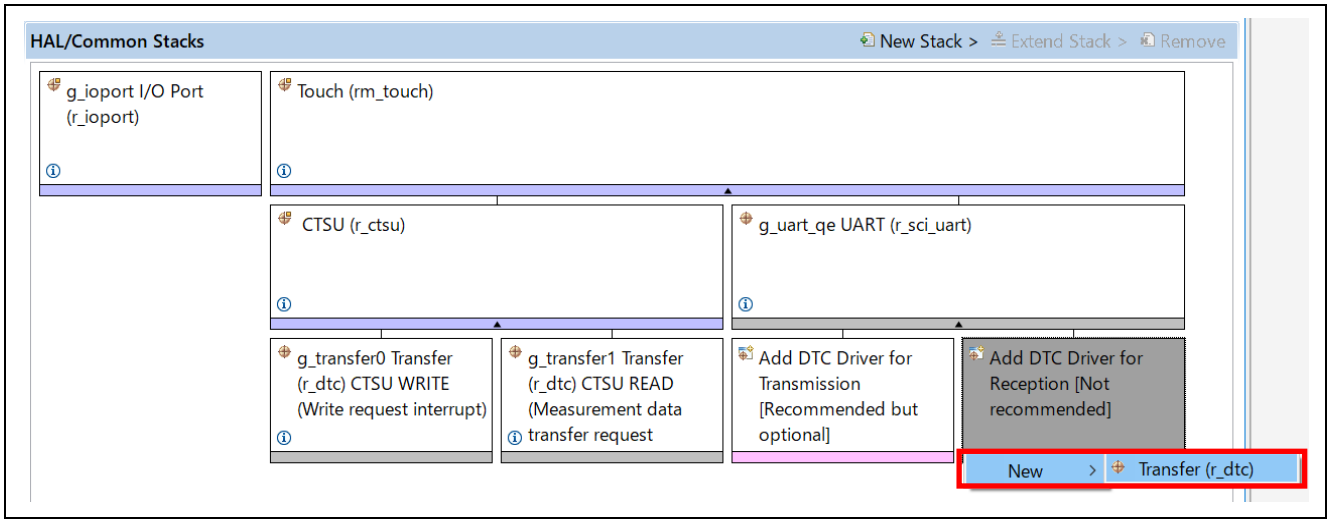

図33　受信用 DTC ドライバ追加（SCI9）

27. [New Stack] -> [Timers] -> [Timer Low-Power (r_agt)]を選択し、AGT ドライバを追加します。

【注】：27~33項は CTSU 計測動作開始トリガに外部トリガを使用する場合のみ設定してください。計測開始トリガに外部トリガを使用しない場合（ソフトウェアトリガの場合）は34項へ進んでください。

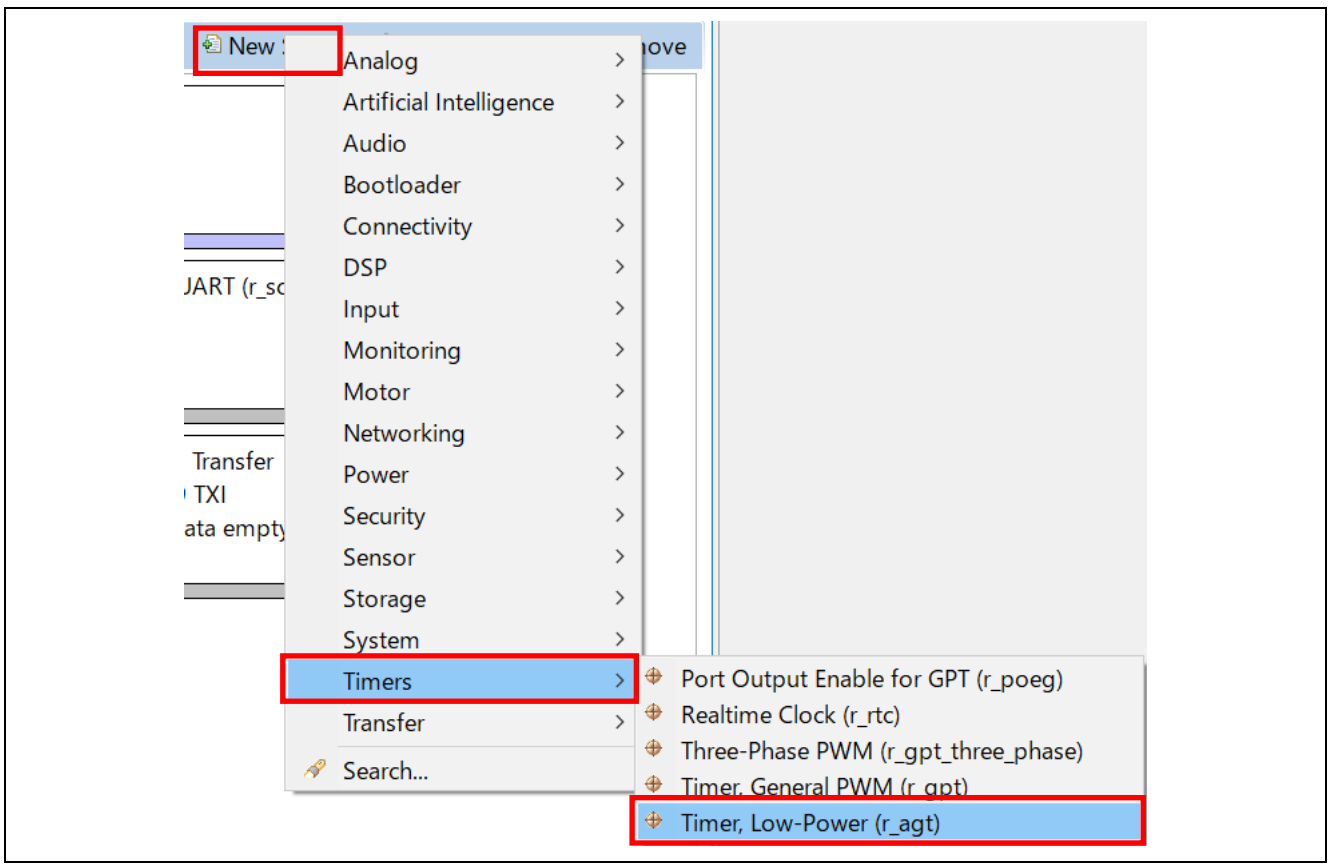

図34　AGT ドライバ追加

28. 次に[g_timer0 Timer,Low-Power (r_agt)]モジュールをクリックし、[プロパティー]を表示します。

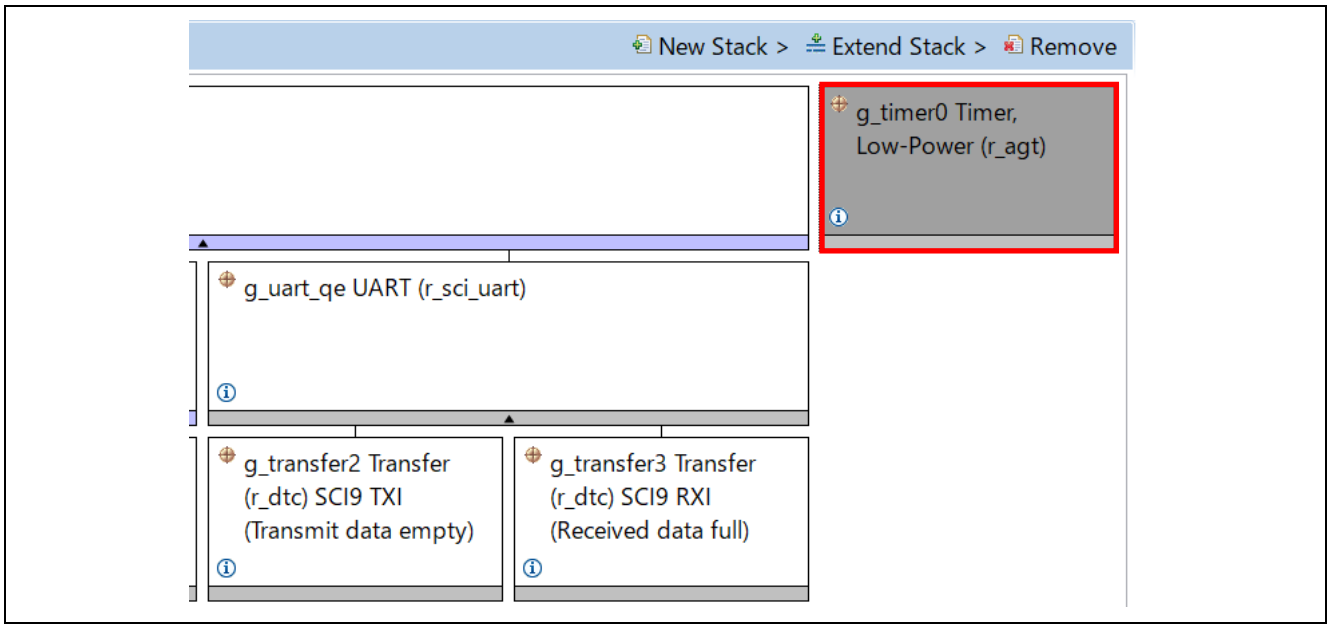

図35　[g_timer0 Timer Driver on r_agt]モジュール

29. AGT の設定をします。画面左下に表示される[プロパティー] – [Settings] – [Module g_timer0 Timer Driver on r_agt] – [General] – [Period]に”100”を入力、[Period Unit]に”Milliseconds”を選択します。

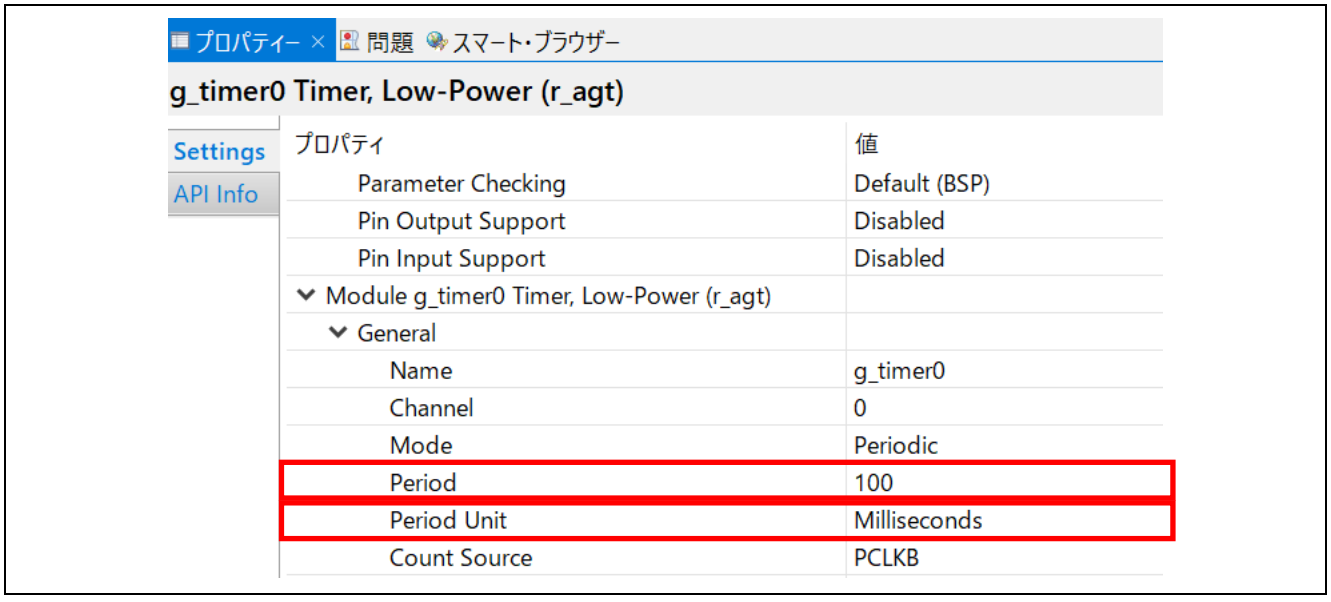

図36　[Period]、[Period Unit]

30. 続けて AGT の割り込みを設定します。[Module g_timer0 Timer Driver on r_agt] – [Interrupts] – [Underflow Interrupt Priority]に”Disabled”以外を選択してください。ここでは”Priority 2”を選択します。この設定により 100ms 周期で AGT0 の割り込みが発生します。

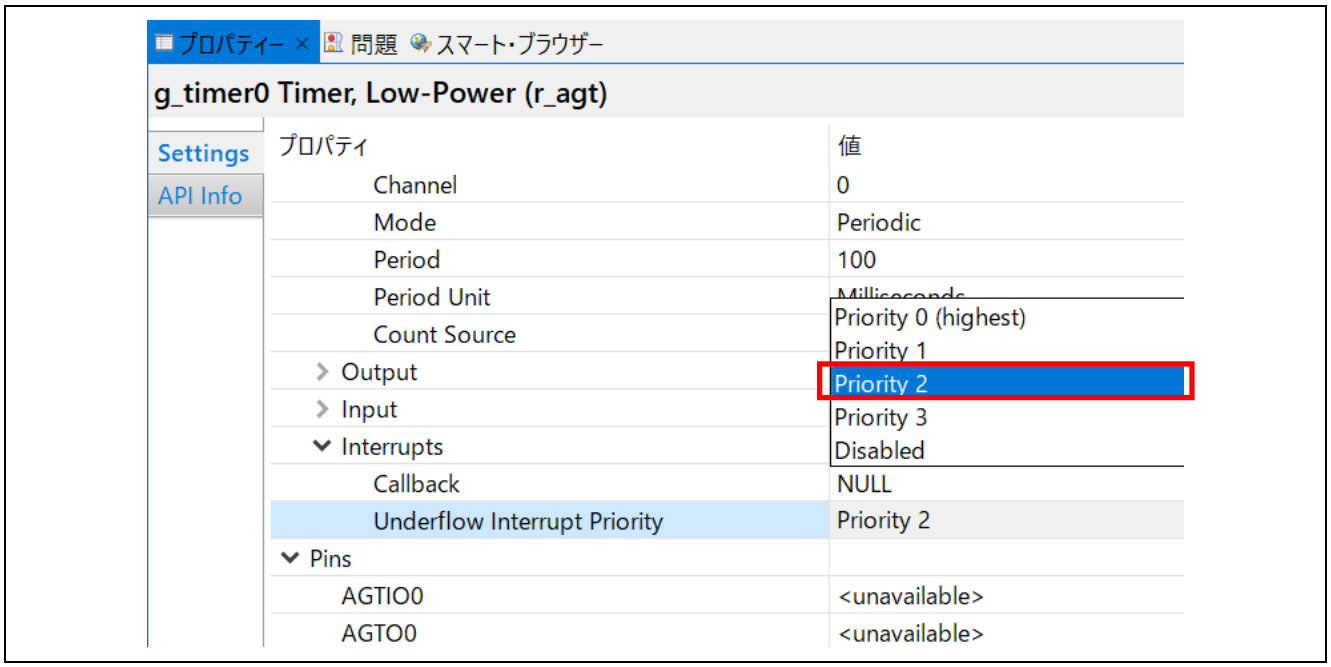

図37　[Underflow Interrupt Priority]

31. [New Stack] -> [System] -> [Event Link Controller (r_elc)]を選択し、ELC ドライバを追加します。

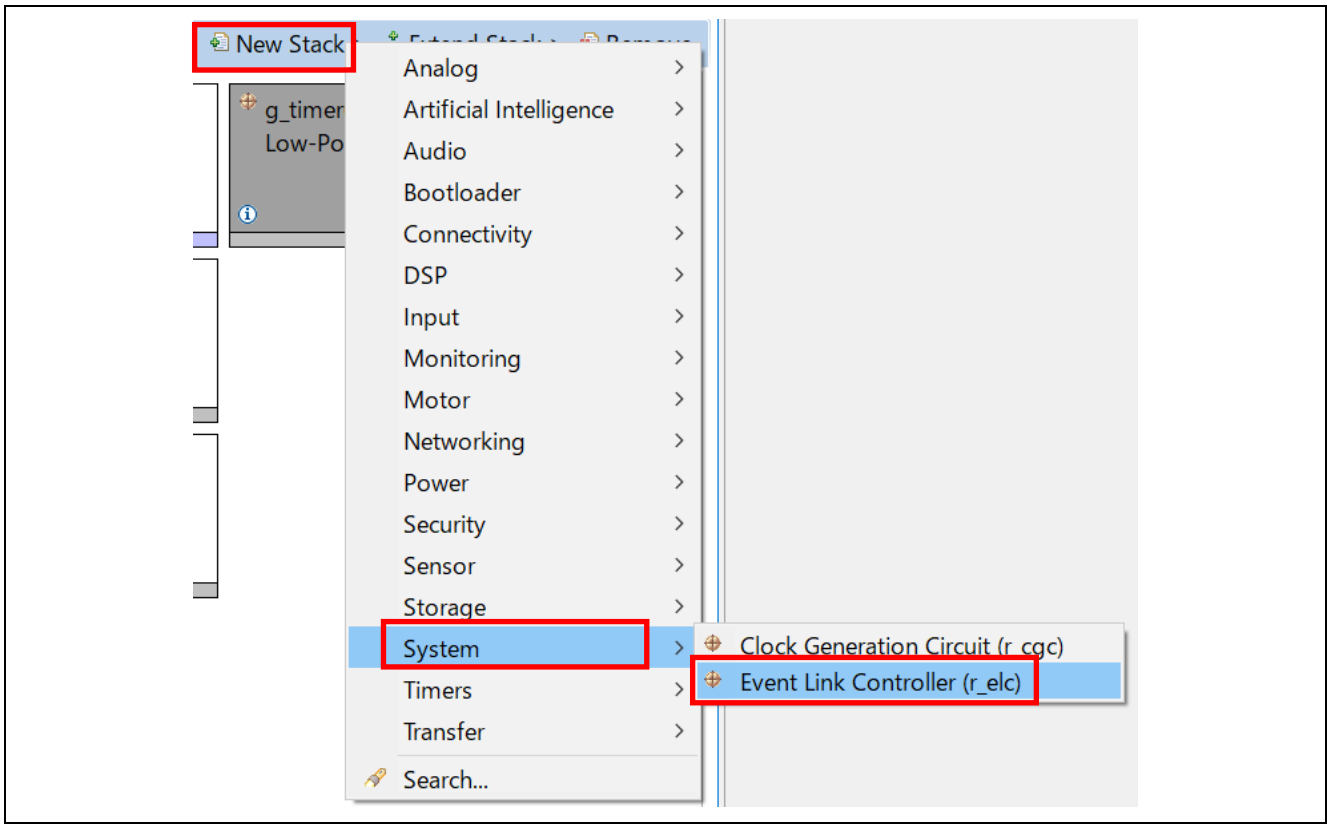

図38　ELC ドライバ追加

32. [CTSU (r_ctsu)]モジュールをクリックし、[プロパティー]を表示します。

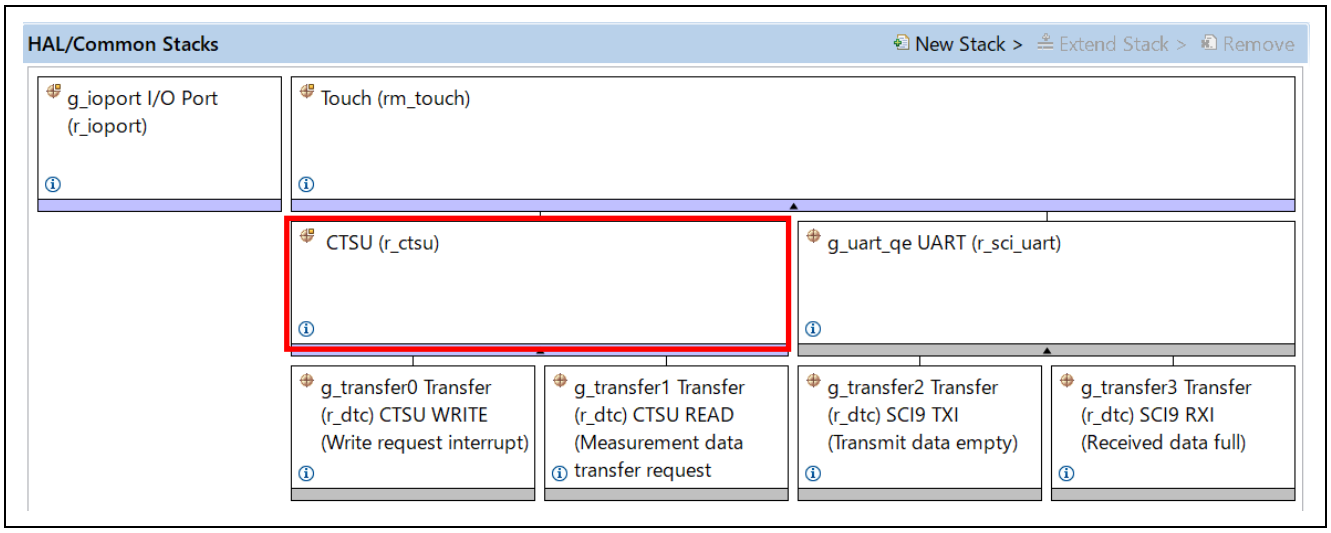

図39　[CTSU Driver on r_ctsu]モジュール

33. CTSU 計測動作開始トリガの設定をします。画面左下に表示される[プロパティー] – [Settings] – [Module CTSU Driver on r_ctsu] – [Scan Start Trigger]を”Software”から”AGT0 INT (AGT interrupt)”に変更します。この設定により CTSU は、AGT0 の割り込みで計測を開始します。

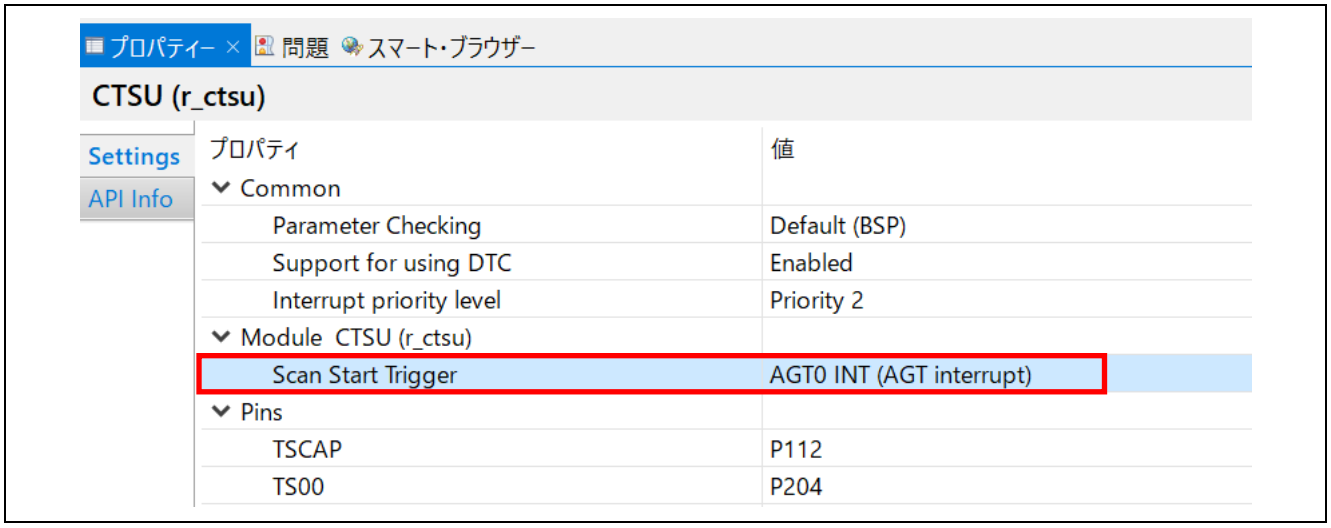

図40　[Scan Start Trigger]

34. これで静電容量タッチに必要なモジュールが追加されました。最後にプロジェクトに必要なモジュールのコードを生成します。以下の図のように、画面右上にある[Generate Project Content]をクリックします。

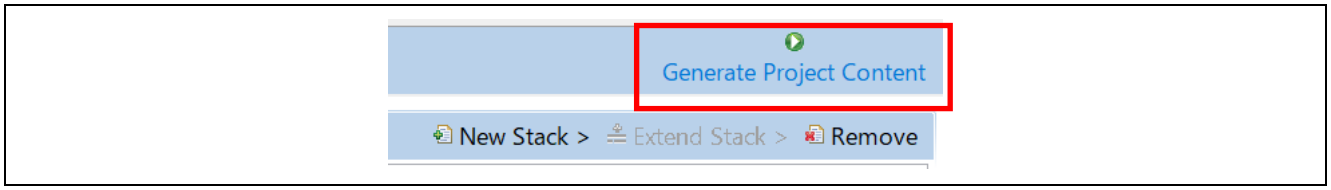

図41　[Generate Project Content]

6.3　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 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。

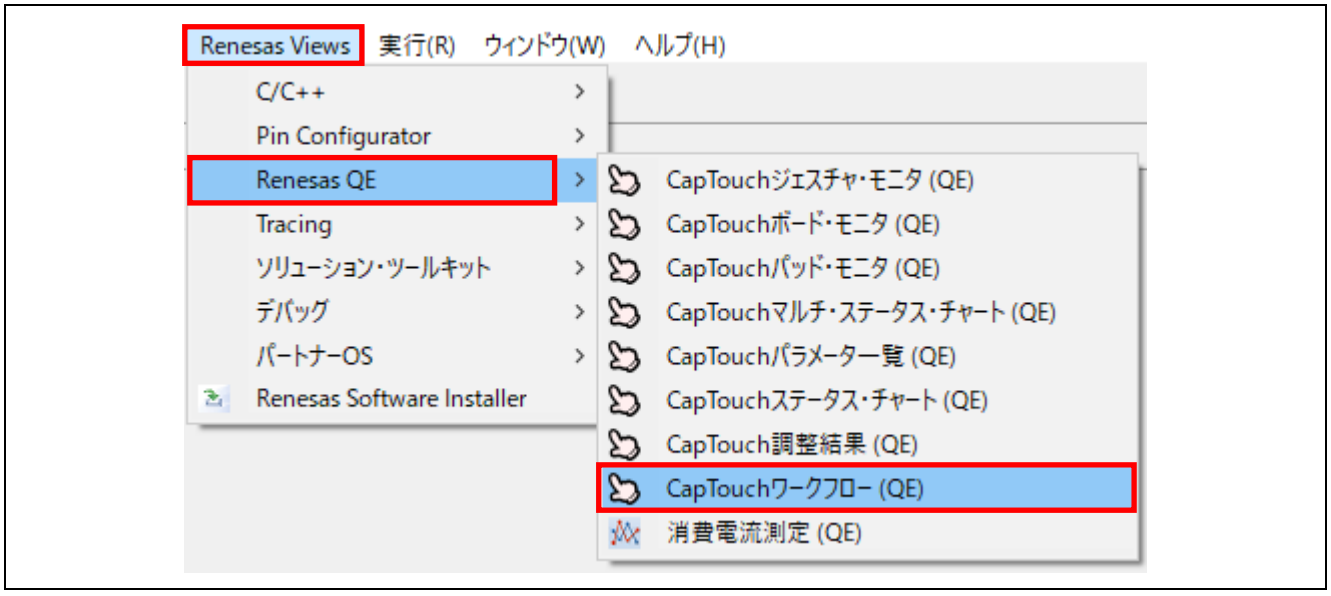

図42　[CapTouch ワークフロー (QE)]メニュー

2. [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。

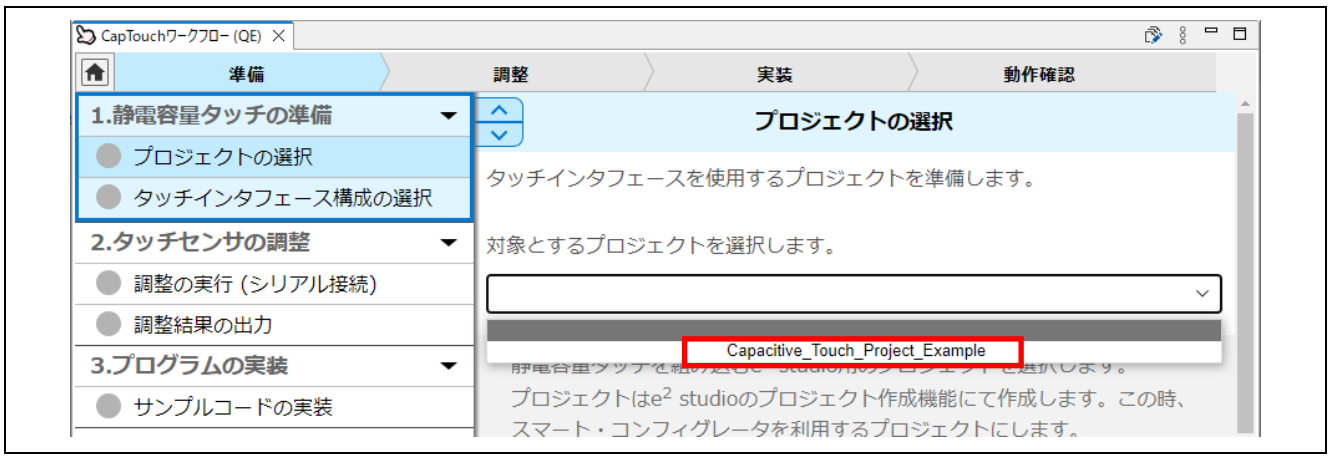

図43　プロジェクトの選択

3. 次に、“CapTouch ワークフロー (QE)” 左のメニューで” タッチインタフェース構成の選択”を選択し、設定項目を表示します。[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。

図44　構成の選択

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。

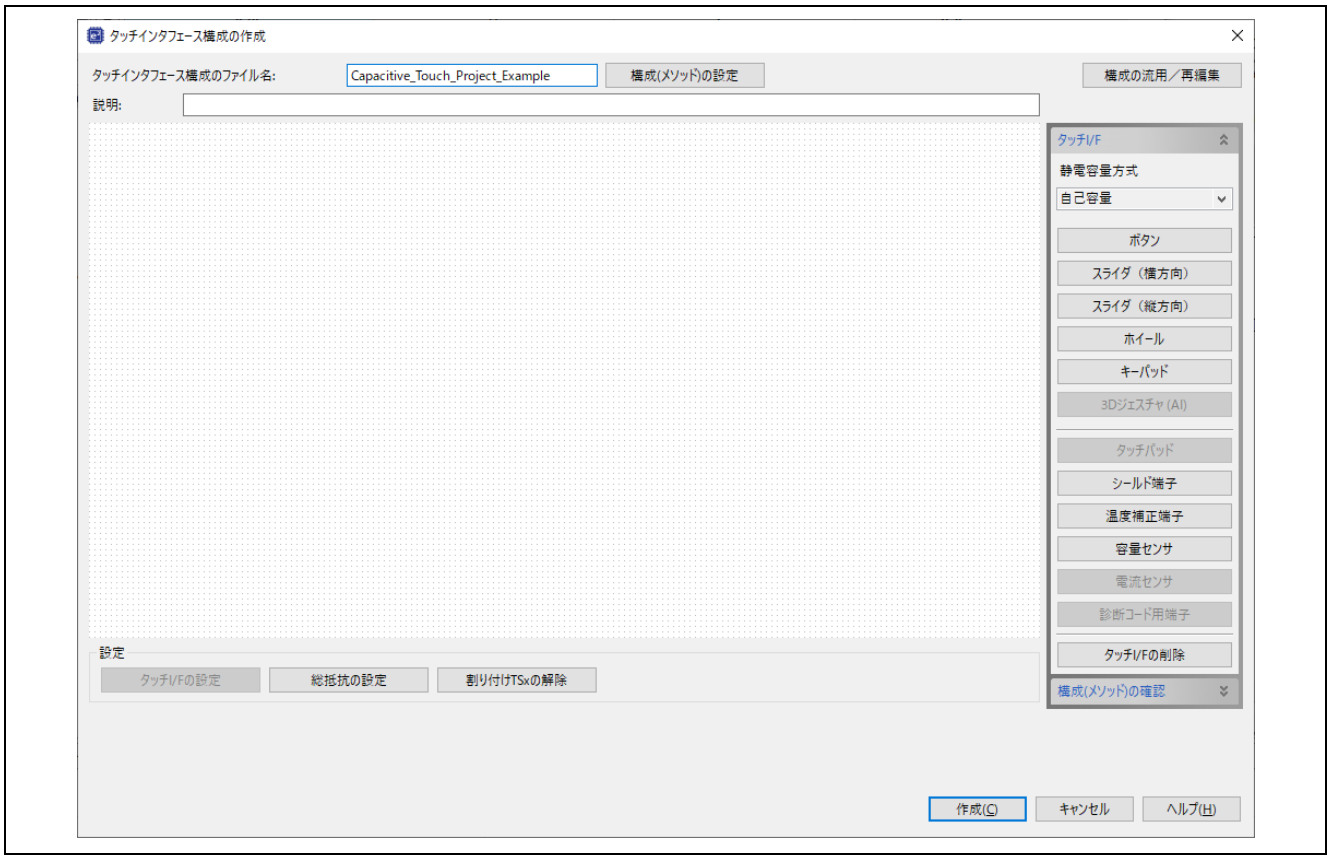

図45　[タッチインタフェース構成の作成]ウィンドウ

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、タッチインタフェースの配置領域をクリックしてボタンを追加します。キーボードの ESC キーを押すか、再度[ボタン]を選択するとボタンの追加を終了します。追加したボタンはタッチセンサが設定されていないので、設定エラー(赤色)のままです。

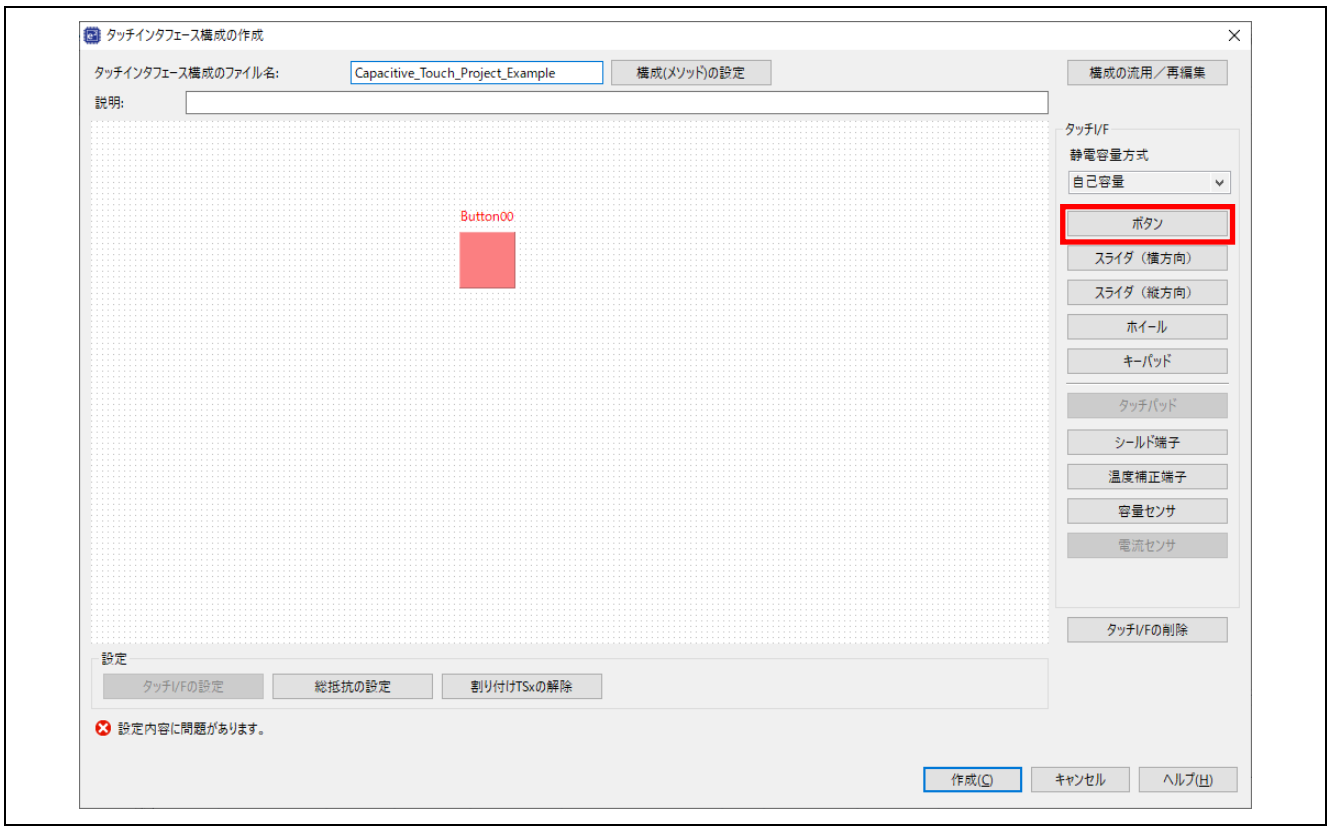

図46　ボタン追加

6. タッチインタフェースの配置領域に追加した“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 のセンサポートを選択します。RA 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、ボタンが緑色になります。

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
選択するタッチセンサ	TS02	TS11

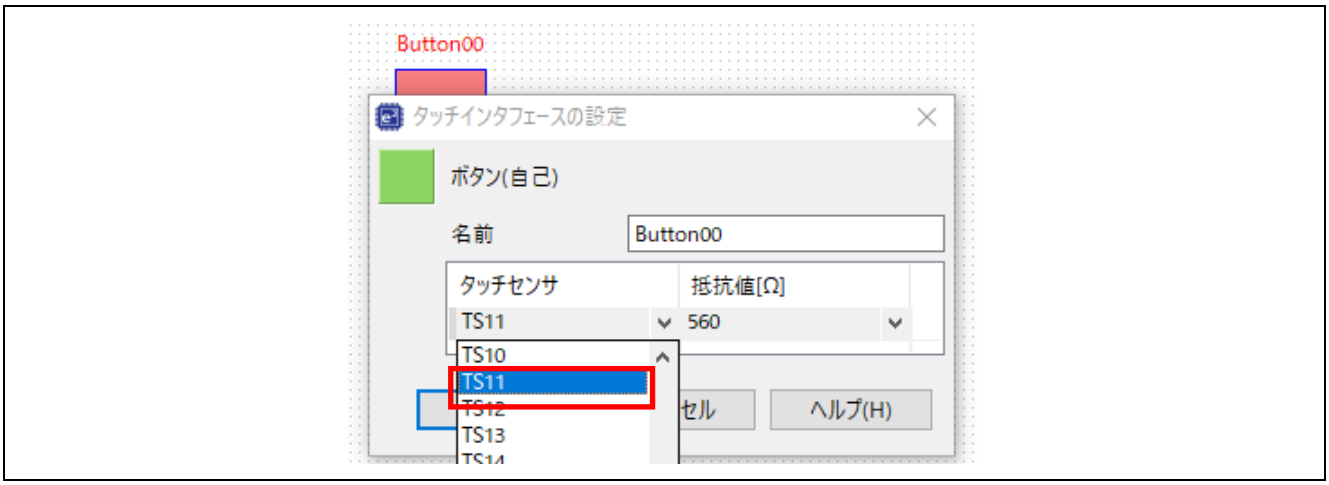

図47　タッチセンサの設定（ボタン）

7. 次に[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[シールド端子]を選択し、タッチインタフェースの配置領域をクリックしてシールド端子を追加します。キーボードの ESC キーを押すか、再度[シールド端子]を選択するとシールド端子の追加を終了します。追加したシールド端子はタッチセンサが設定されていないので、設定エラー(赤色)のままです。

   【注】：プロジェクト作成時、[Device Selection]の[Board]で”EK-RA6M2”(RA6M2 MCU グループ評価キット)を選択した場合は、[シールド端子]が無効のため追加することはできません。

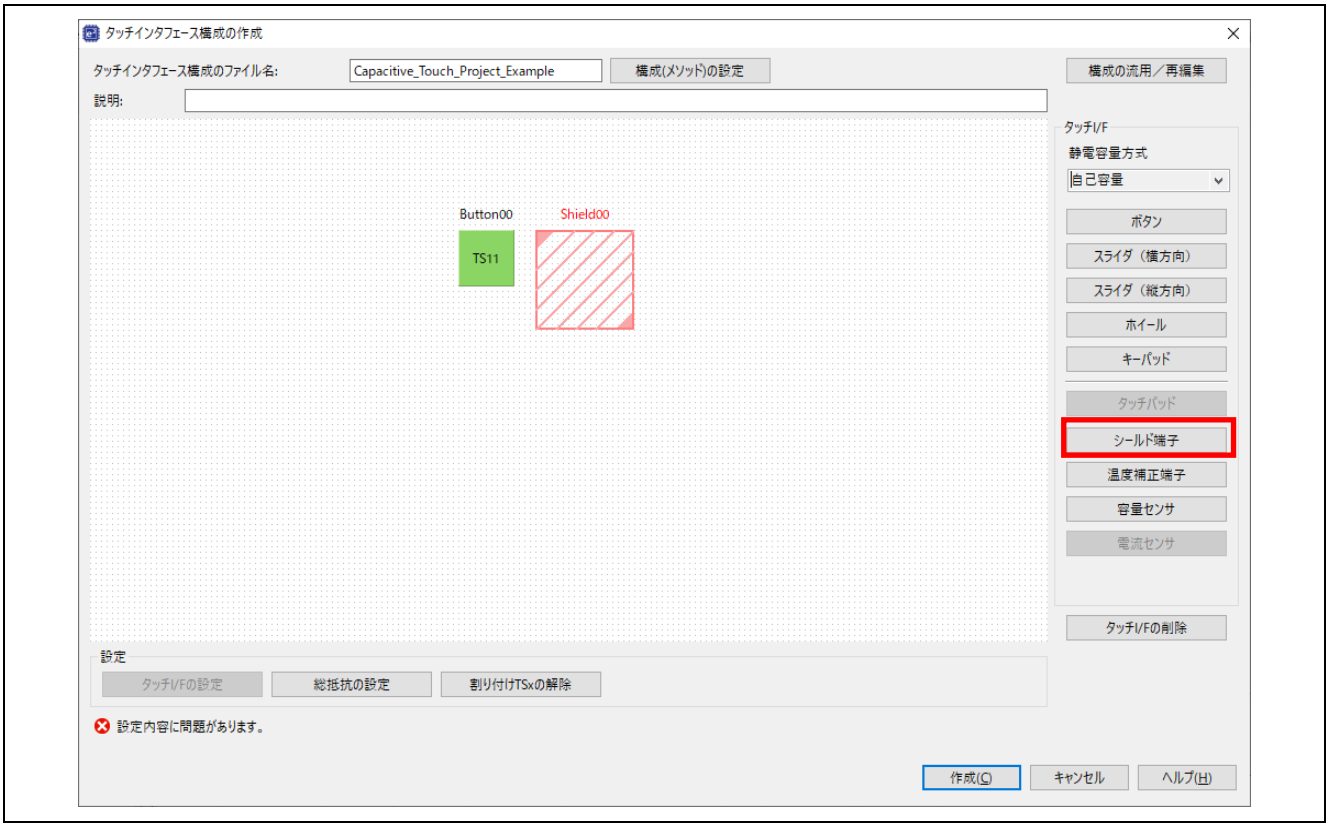

図48　シールド端子追加

8. タッチインタフェースの配置領域に追加した“Shield00”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、このシールド端子に割り当てるMCU のセンサポートを選択します。RA 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、シールド端子が緑色になります。

項目	RA6M2 (CTSU)	RA2L1 (CTSU2)
選択するタッチセンサ	-（EK-RA6M2 では設定不可）	TS00

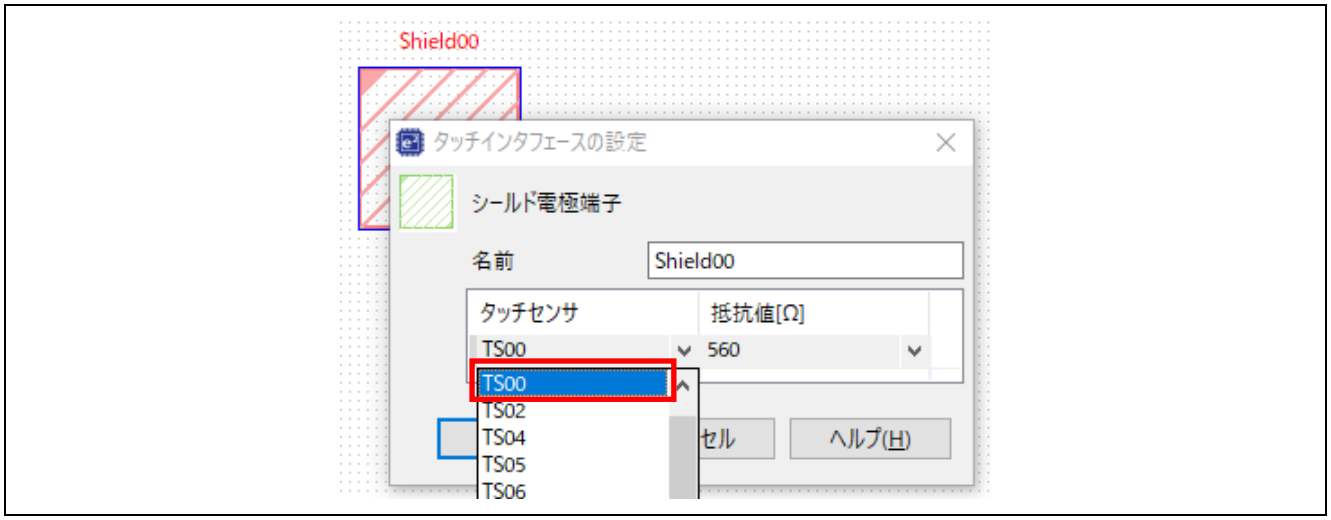

図49　タッチセンサの設定（シールド端子）

9. [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。

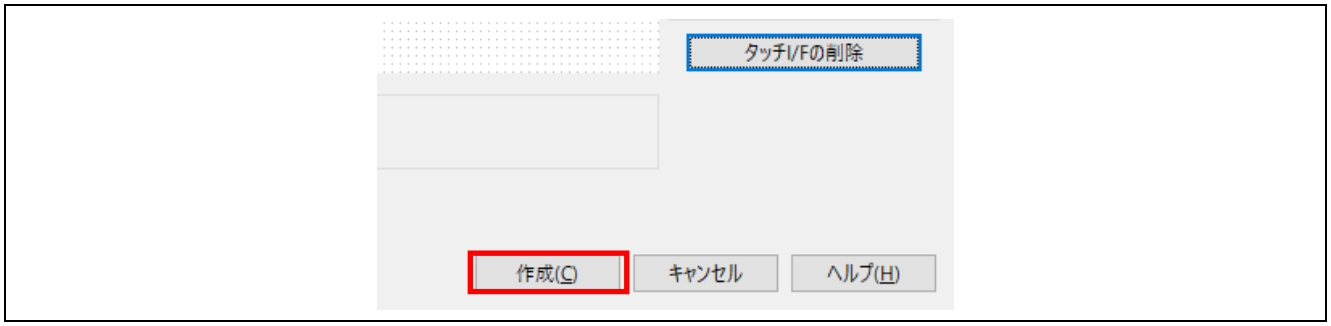

図50　[作成]ボタン

10. e² studio からメニュー[Renesas Views] – [Renesas QE] – [CapTouch 調整結果（QE)] を選択し、調整結果ビューを開くと、[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。

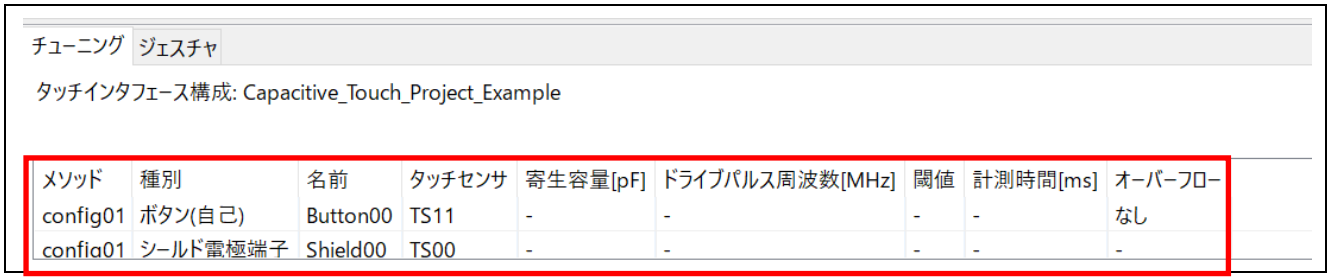

図51　[チューニング]パネル

11. 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。

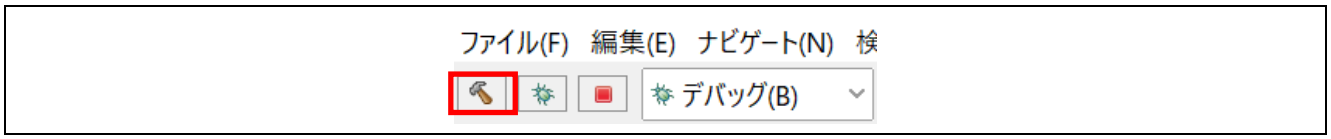

図52　ビルドボタン

6.4 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。 のドロップダウンリストボタンをクリックし、デバッグ構成を選択してください。

図53　デバッグ構成の編集ボタン

2. [Startup]タブを選択します。

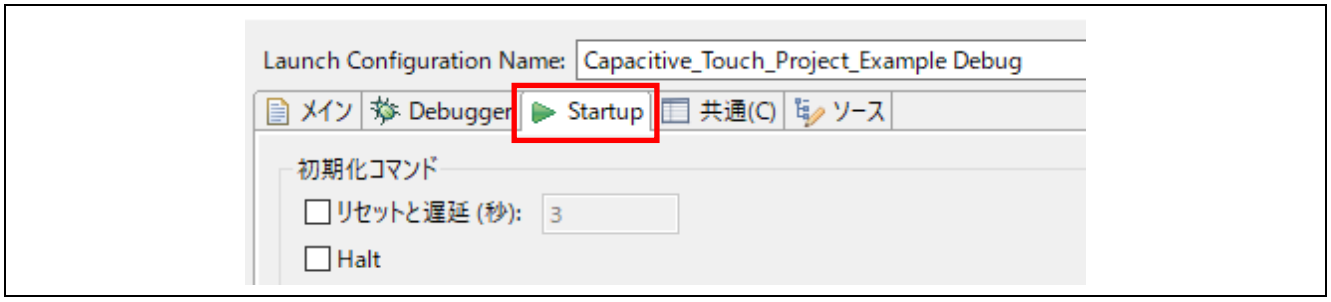

図54　[Startup]タブ

3. [ランタイム・オプション]の[ブレークポイント設定先]と[再開]のチェックボックスを以下のようにチェックします。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。

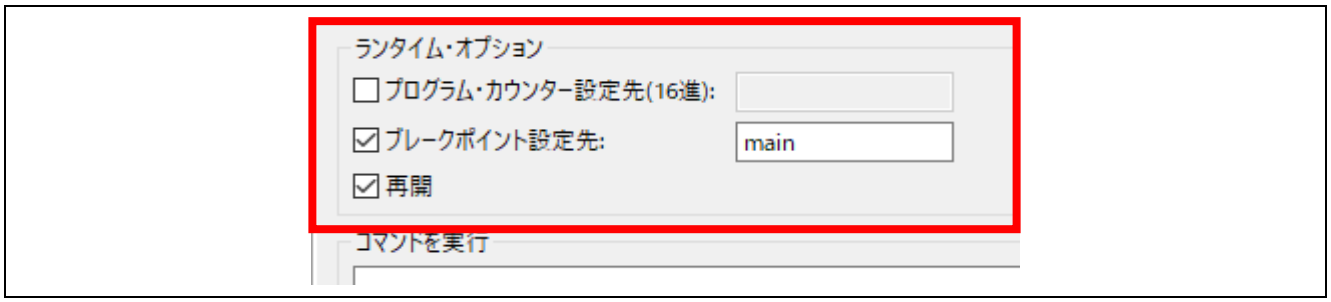

図55　[ランタイム・オプション]

4. [適用]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。

6.5 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. エミュレータがボードと PC に正しく接続されていることを確認します。

2. [CapTouch ワークフロー (QE)]の左のメニューで”調整の実行 (エミュレータ接続)”を選択し、”タッチセンサの調整”の項目設定を開きます。[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。

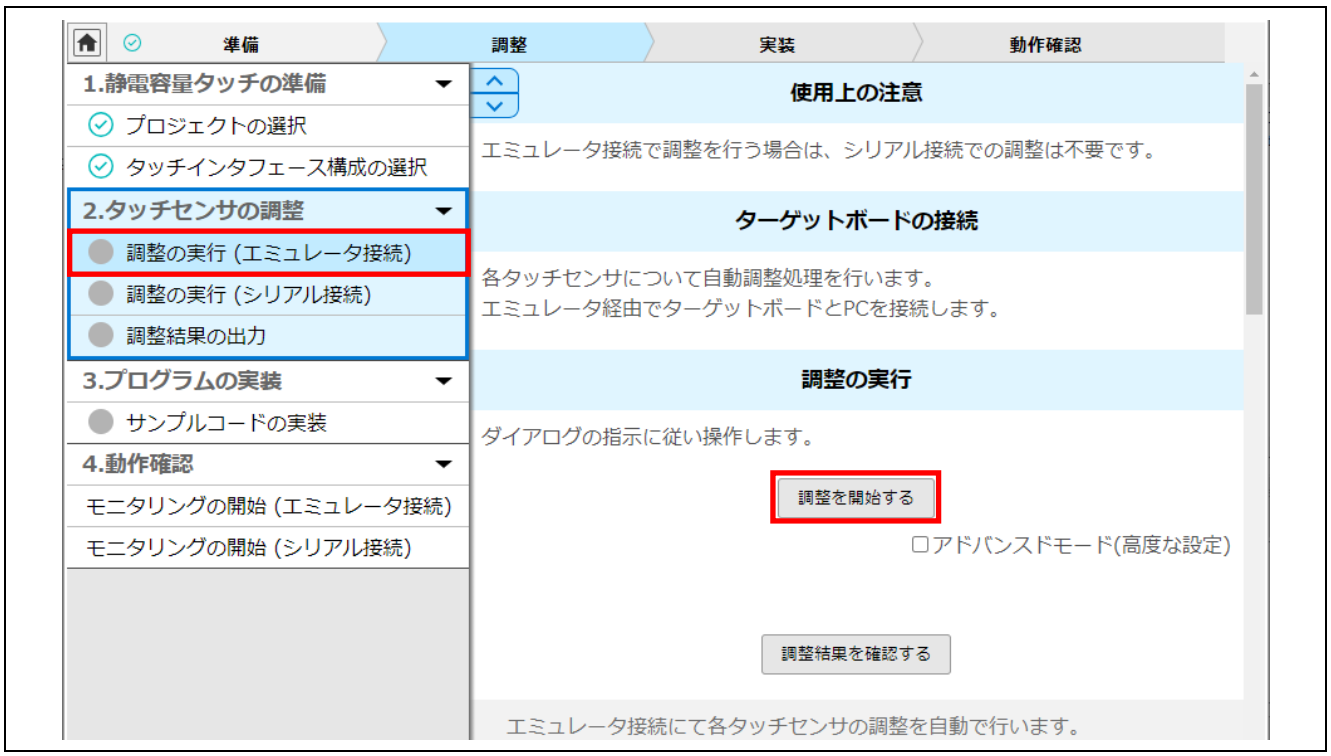

図56　チューニングの開始

3. 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]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。

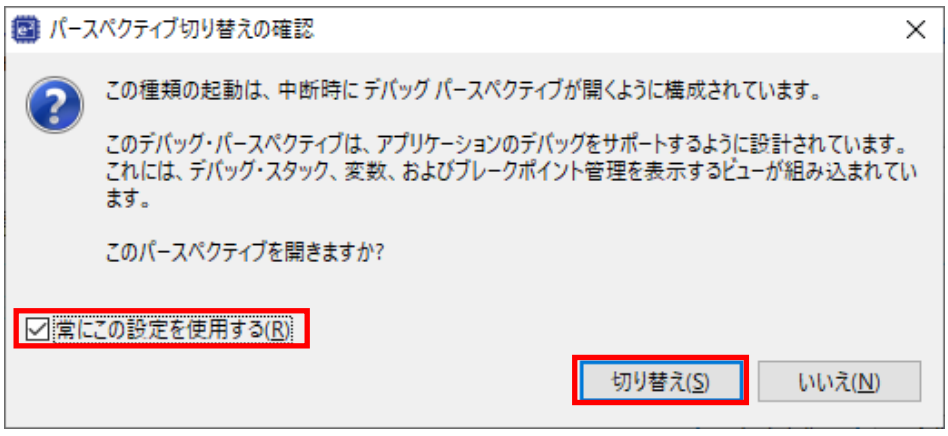

図57　パースペクティブ切り替えの確認

4. 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。

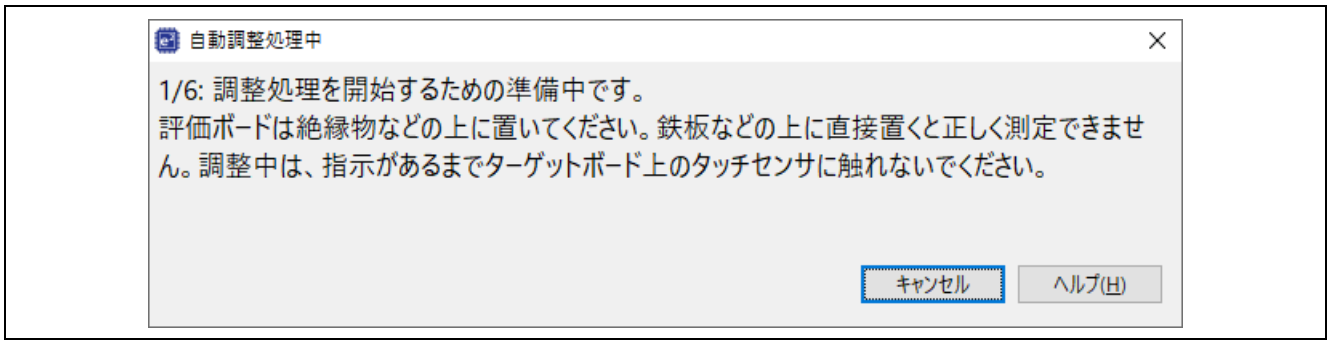

図58　[自動調整処理中]ダイアログ（準備中）

5. いくつかの自動工程を経て、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”Button00”（RA6M2 は TS02、RA2L1 は TS11-CFC）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。

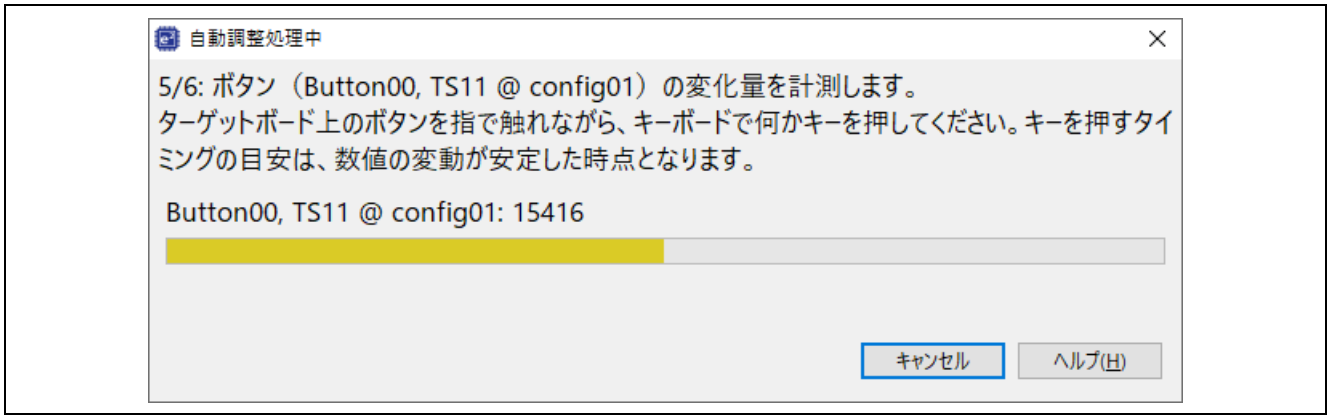

図59　[自動調整処理中]ダイアログ（計測中）

6. 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。

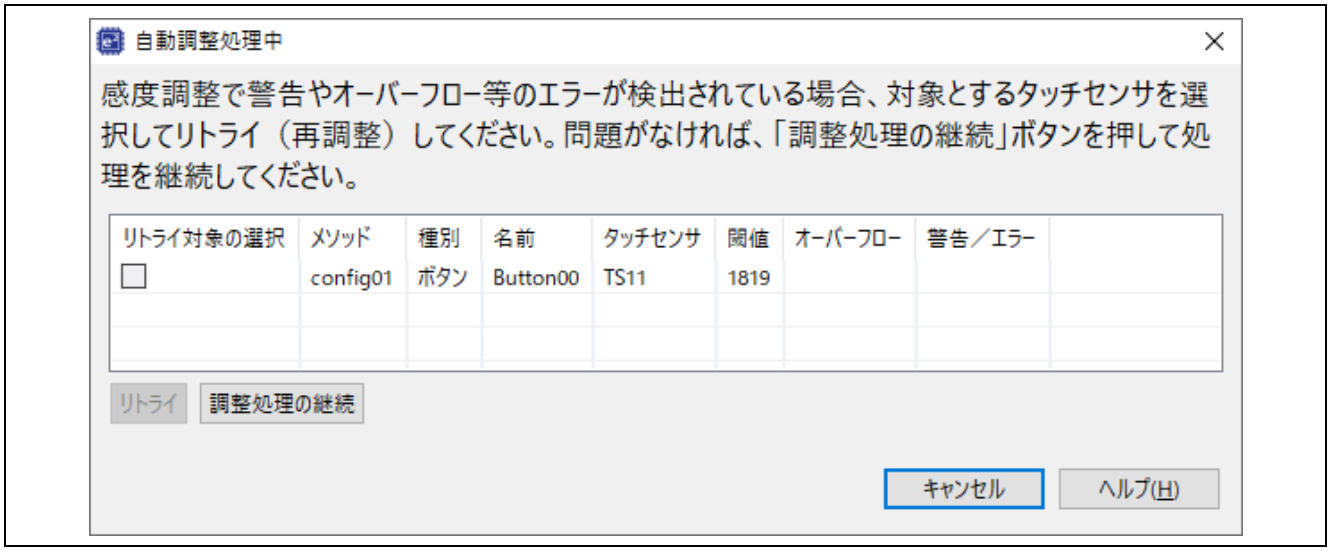

図60　[自動調整処理中]ダイアログ（調整完了）

7. 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 ワークフロー (QE)]に戻ります。

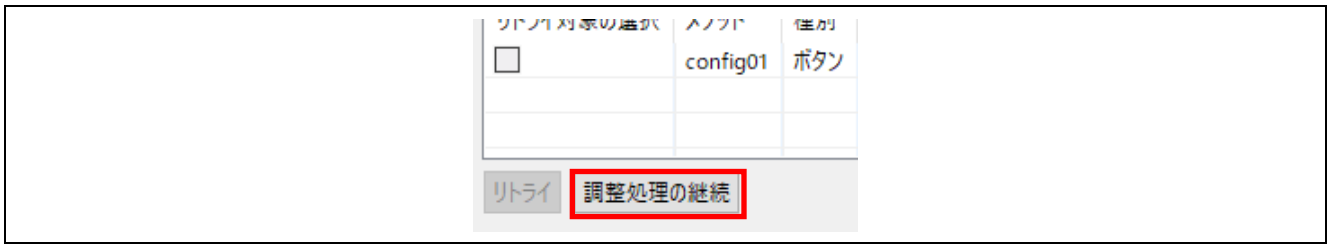

図61　[調整処理の継続]ボタン

8. 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。“CapTouch ワークフロー (QE)” 左のメニューで ”調整結果の出力”を選択し、[ファイルを出力する]をクリックします。

【注】：当項は、CTSU 計測動作開始トリガに外部トリガを使用しない場合（ソフトウェアトリガの場合）に実行してください。

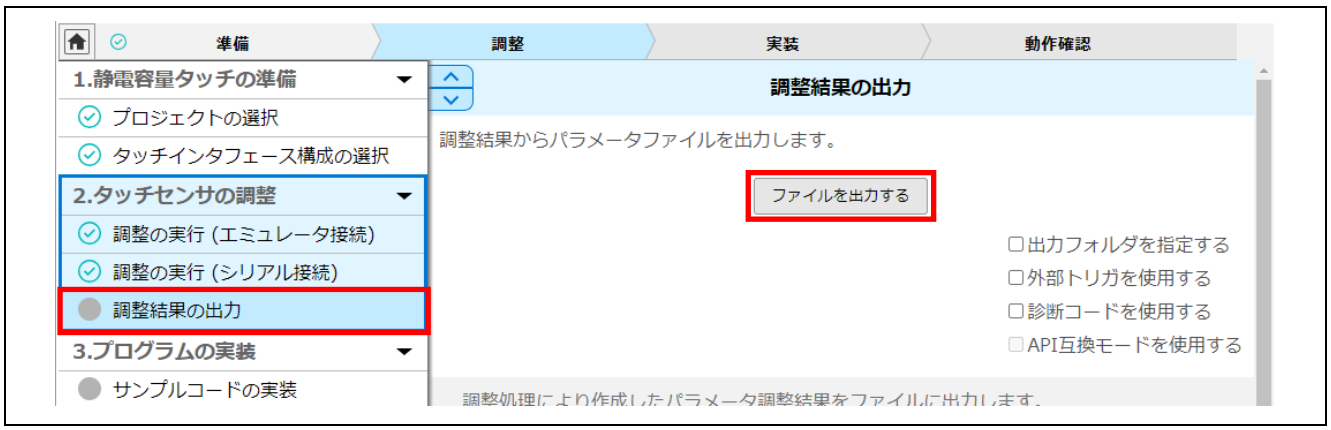

図62　調整結果の出力

9. CTSU 計測動作開始トリガに外部トリガを使用する場合、[外部トリガを使用する]をチェックし、[ファイルを出力する]をクリックします。

   【注】：当項は、CTSU 計測動作開始トリガに外部トリガを使用する場合に実行してください。

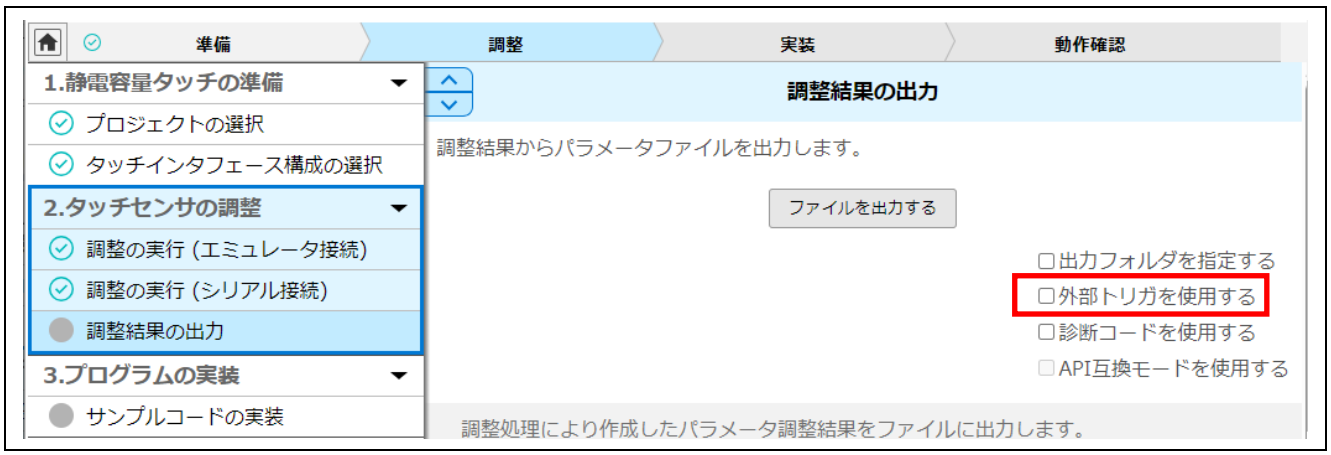

図63　パラメータファイルの出力設定

10. [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。

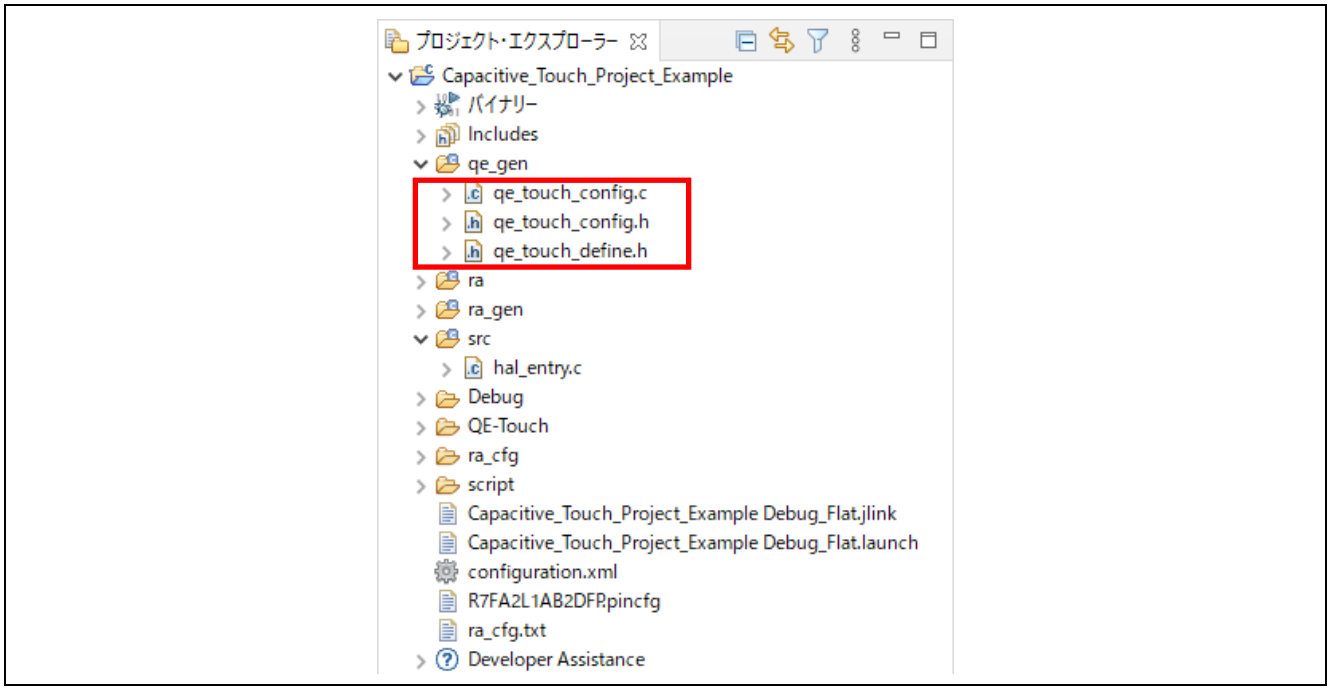

図64　パラメータファイル

11. 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]ウィンドウにビルドした結果、エラーがないことが確認できます。

    【注】: Flexible Software Package (FSP) V3.0.0 にてシリアル通信によるモニタリングを有効にした場合、ビルドエラーが発生します。詳細はhttps://github.com/renesas/fsp/issues/78 を参照してください。

6.6　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 ワークフロー (QE)]左メニューで”サンプルコードの実装”を選択し、[例を表示する]をクリックします。

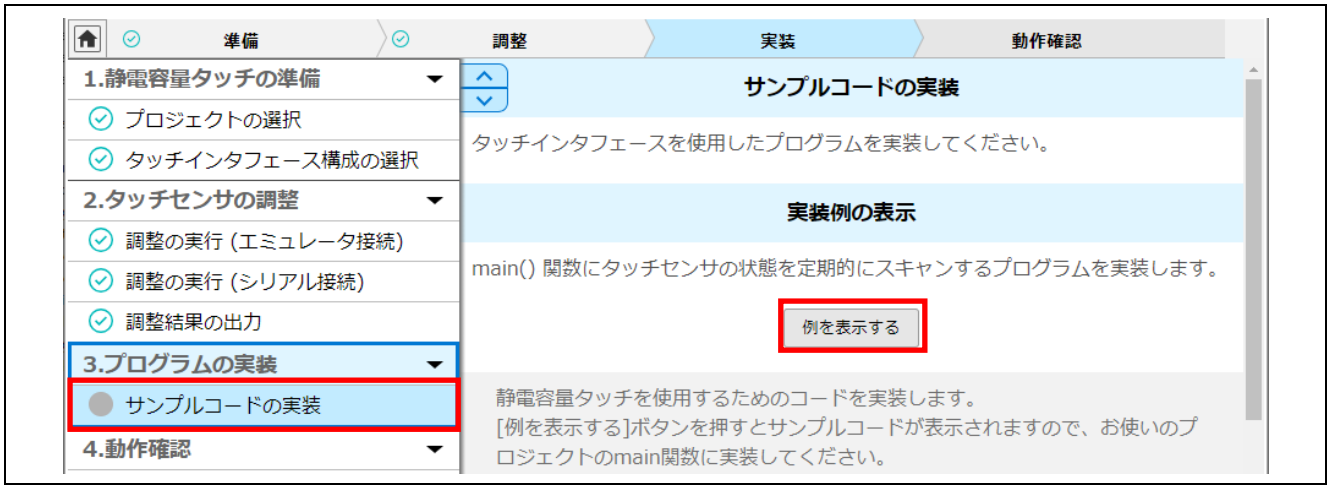

図65　実装例の表示

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]をクリックします。次に[OK]をクリックして、ウィンドウを閉じます。

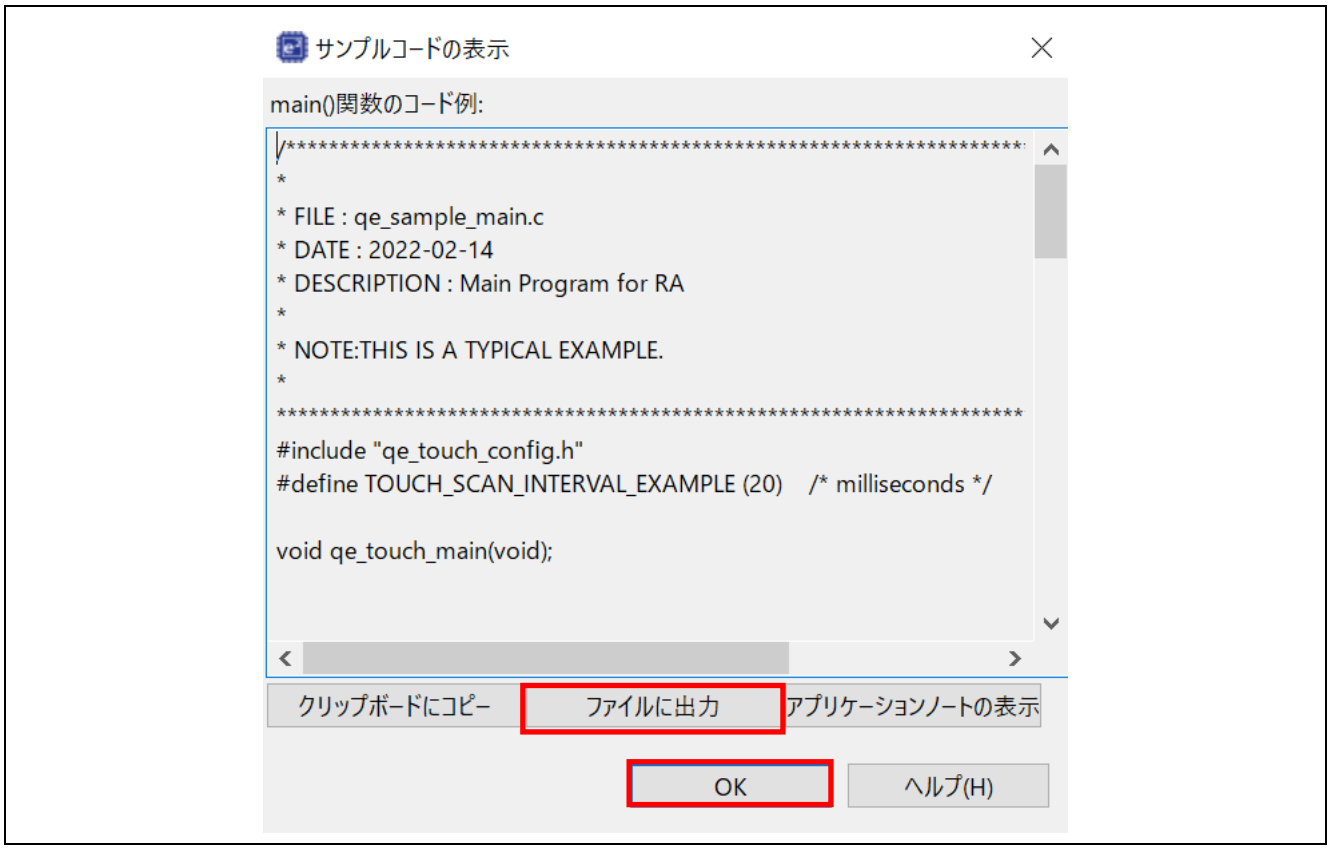

図66　[サンプルコードの表示]ウィンドウ

3. [プロジェクト･エクスプローラー]で qe_touch_sample.c ファイルが生成されたことが確認できます。

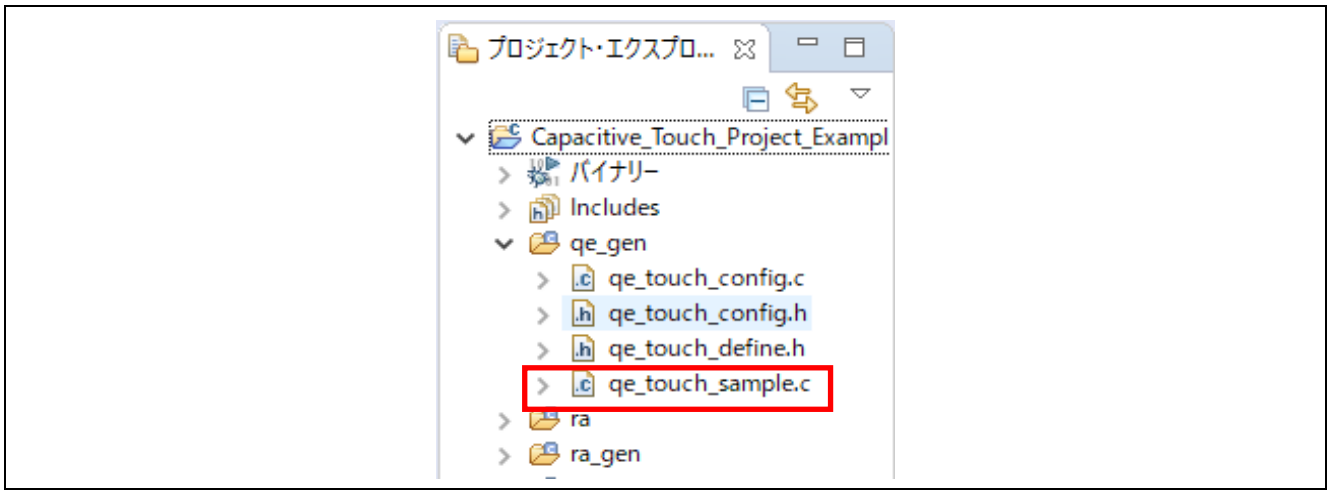

図67　qe_touch_sample.c

4. [プロジェクト・エクスプローラー]の ra_gen -> main.c をダブルクリックしてファイルを開きます。main() 関数内の”hal_entry”を選択した状態で右クリックし[宣言を開く]を選択すると、hal_entry () 関数の宣言が開きます。

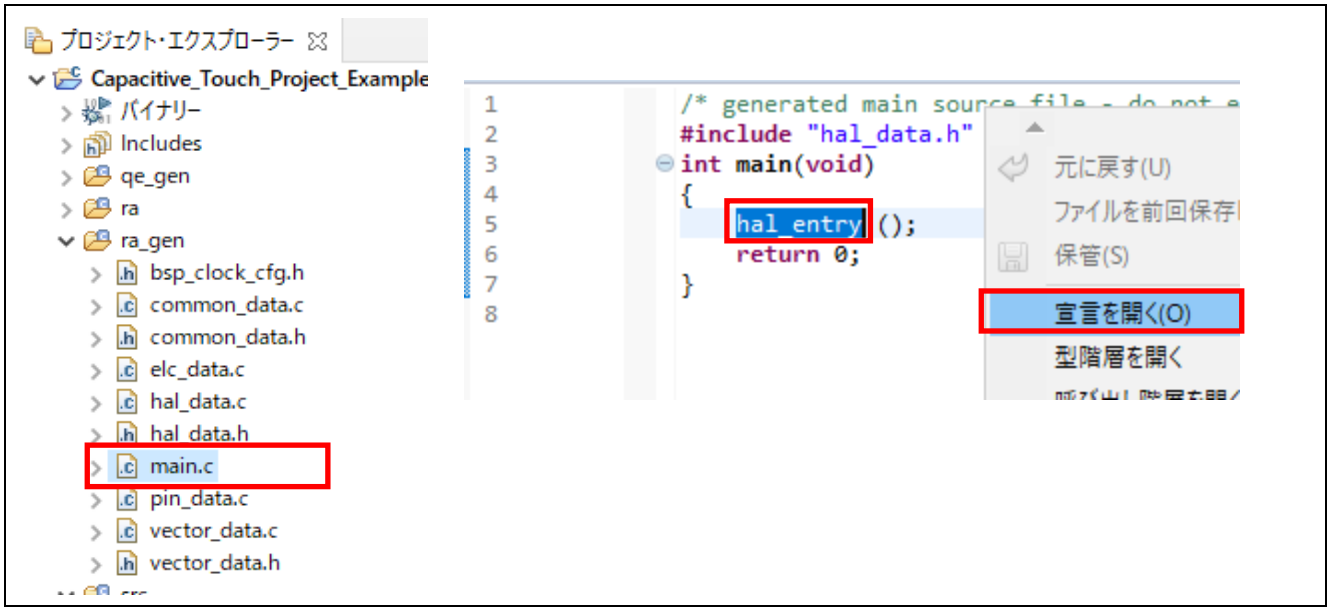

図68　宣言を開く(hal_entry 関数)

5. hal_entry() 関数の以下の箇所に qe_touch_main() 関数をコールするコードとプロトタイプ宣言を追加します。

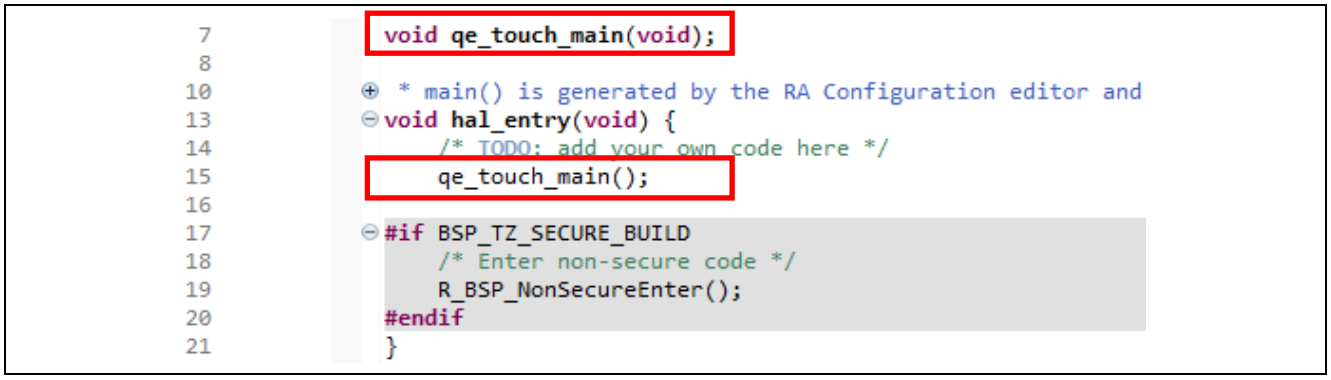

図69　関数コール（qe_touch_main 関数）

6. hal_entry()関数内の”qe_touch_main”を選択した状態で右クリックし[宣言を開く]を選択すると、qe_touch_main () 関数の宣言が開きます。

【注】：6～7項は CTSU 計測動作開始トリガに外部トリガを使用する場合のみ実行してください。計測開始トリガに外部トリガを使用しない場合（ソフトウェアトリガの場合）は8項へ進んでください。

図70　宣言を開く（qe_touch_main 関数）

7. CTSU 計測動作開始トリガに外部トリガを使用する場合、qe_touch_main()関数の AGT ドライバを制御する API のコメントを外してコードを有効にします。

```
27  void qe_touch_main(void)
28  {
29      fsp_err_t err;
30
31      /* Initializes the software and sets the links defined in the control structure. */
32      R_ELC_Open(g_elc.p_ctrl, g_elc.p_cfg);
33      /* Globally enable event linking in the ELC. */
34      R_ELC_Enable(g_elc.p_ctrl);
35
36      /* Open Touch middleware */
37      err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
38      if (FSP_SUCCESS != err)
39      {
40          while (true) {}
41      }
42
43      /* Open AGT driver */
44      R_AGT_Open(g_timer0.p_ctrl, g_timer0.p_cfg);
45      /* Start AGT. */
46      R_AGT_Start(g_timer0.p_ctrl);
47
48      /* Main loop */
49      while (true)
50      {
```

図71　コードの修正

8. 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。

6.7 [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 hal_entry() 関数コールでストップします。これは main() 関数の最初のコードポイントです。

3. hal_entry() 関数を選択し、宣言を開きます。

図72　宣言を開く（hal_entry 関数）

4. qe_touch_main() 関数を選択し、宣言を開きます。

図73　宣言を開く（qe_touch_main 関数）

5. qe_touch_main.c ファイルを下にスクロールして while(true)ループの RM_TOUCH_DataGet() 関数を表示します。引数の” button_status”を選択した状態で右クリックし[監視式を追加…]を選択すると、[監視式を追加]ダイアログが開くので[OK]を選択し、[式]ウィンドウに変数 button_status を追加します。

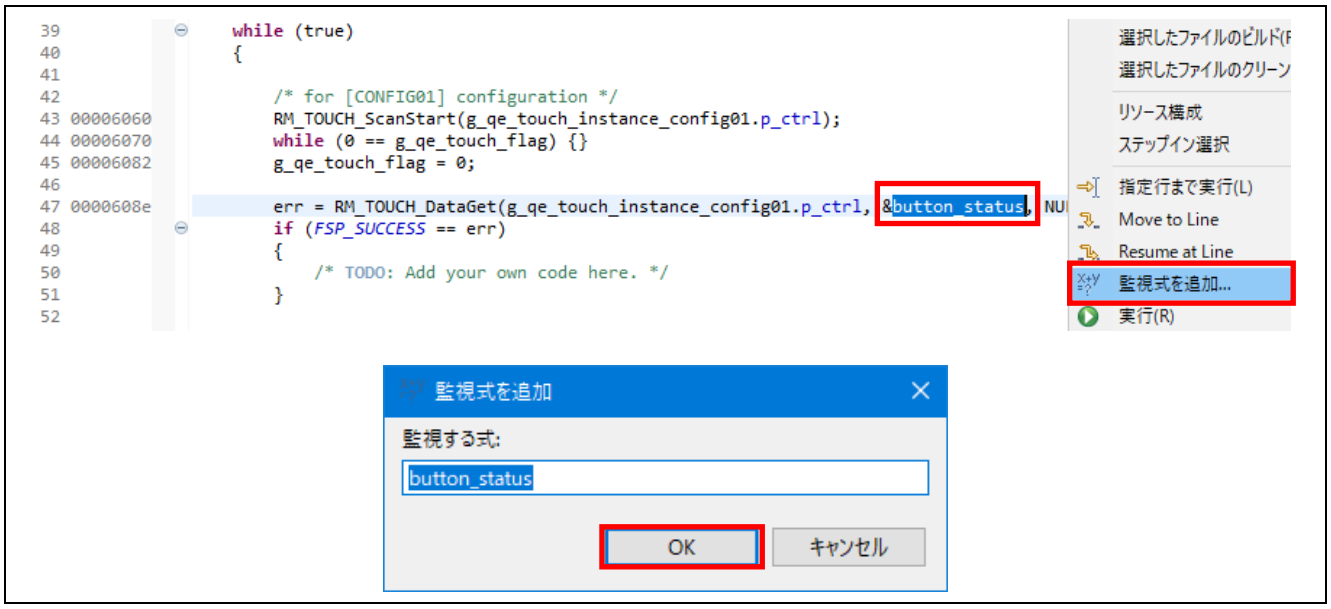

図74　[監視式を追加]メニュー、[監視式を追加]ダイアログ

6. [式]ウィンドウで右クリックし[Enable Real-time Refresh]を選択すると、追加した変数のリアルタイムリフレッシュが有効になります。

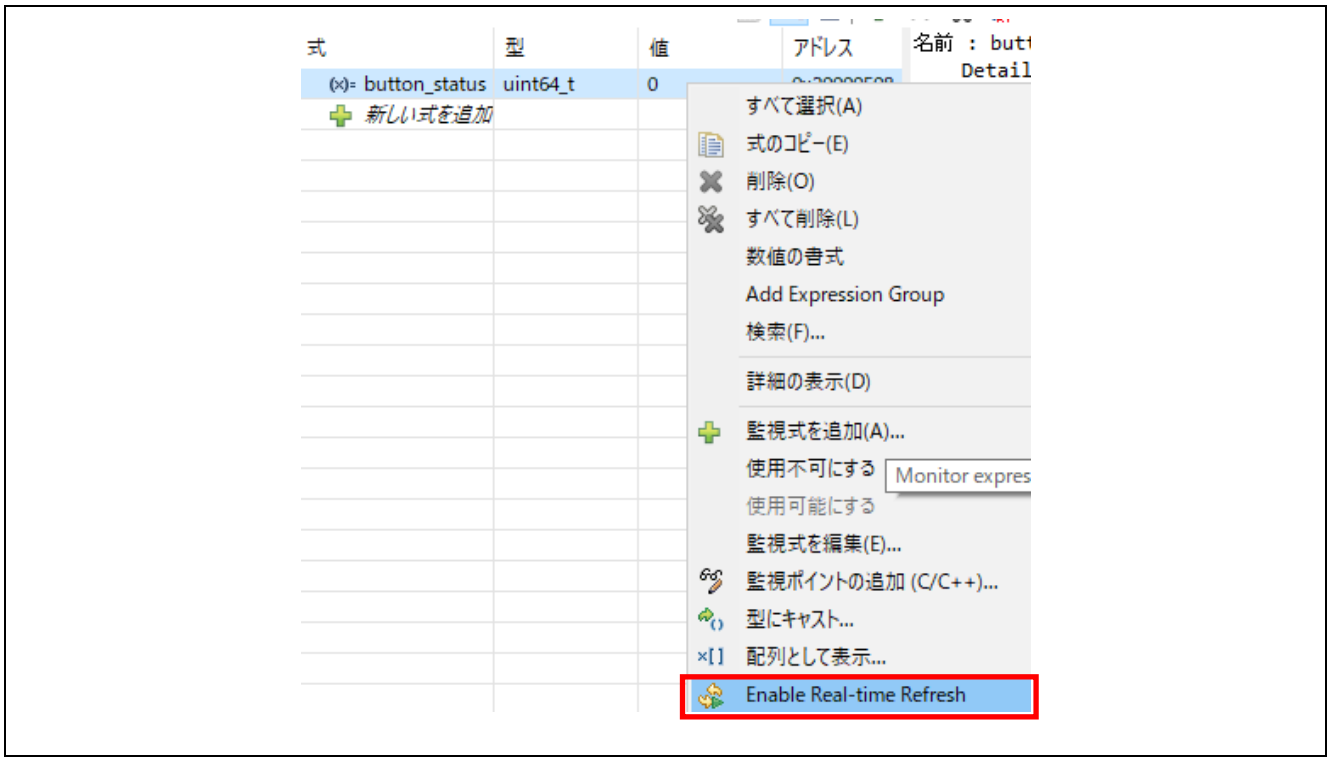

図75　[Enable Real-time Refresh]メニュー

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

8. 「6.3 静電容量タッチインタフェース作成」で”Button00”として定義した、ボード上のセンサ（RA6M2 は TS02、RA2L1 は TS11-CFC）をタッチします。”Button00”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。

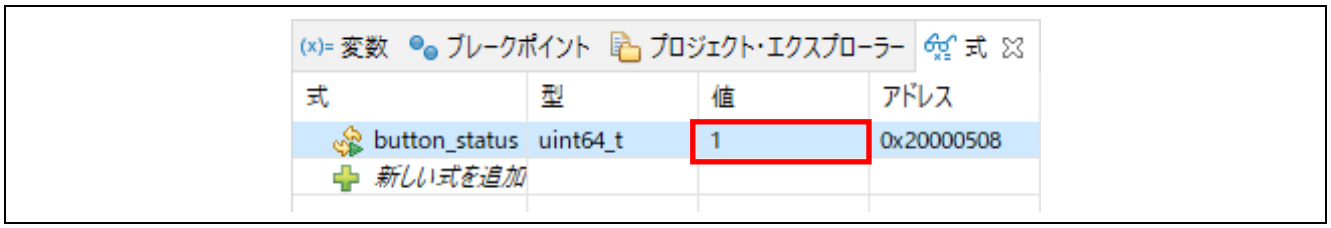

図76　[式]ウィンドウでのタッチ状態の確認

9. [CapTouch ワークフロー (QE)] 左のメニューで ”モニタリングの開始 (エミュレータ接続)”を選択します。[ビューを開く]をクリックし、[CapTouch ボード・モニタ(QE)]を起動します。

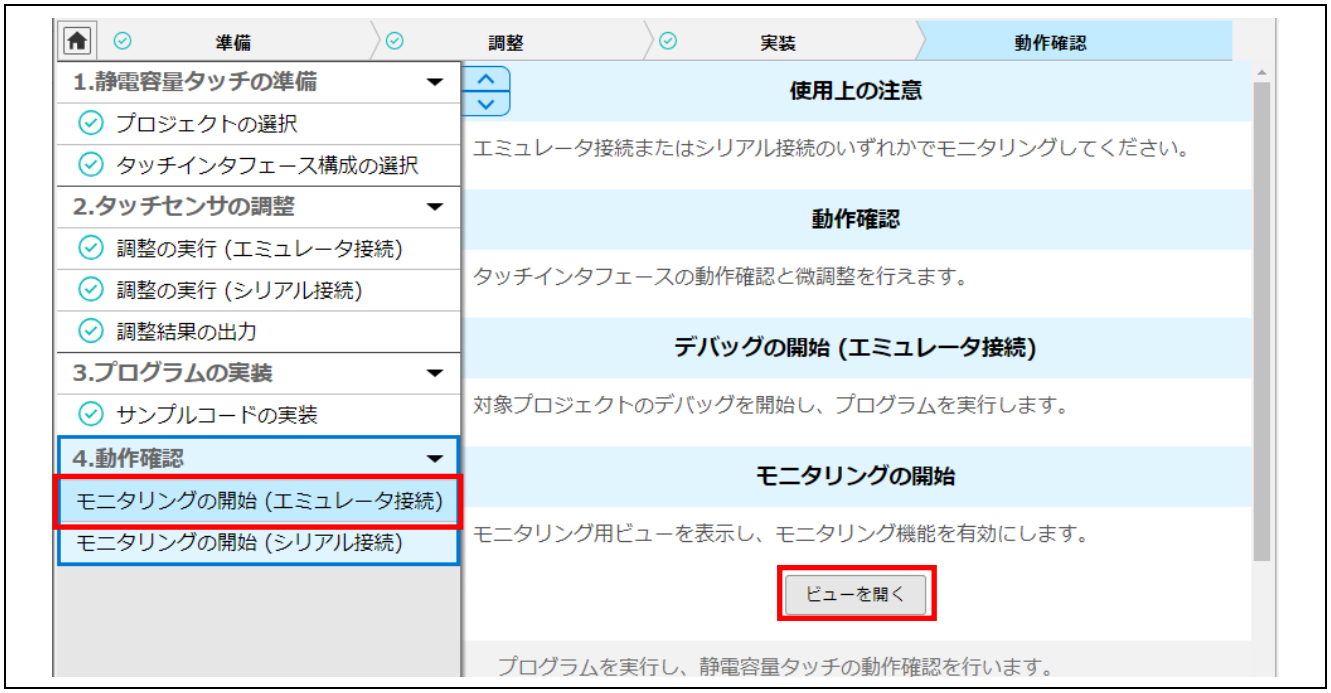

図77　モニタリングの開始

10. [CapTouch ボード・モニタ(QE)]は以下のように表示されます。

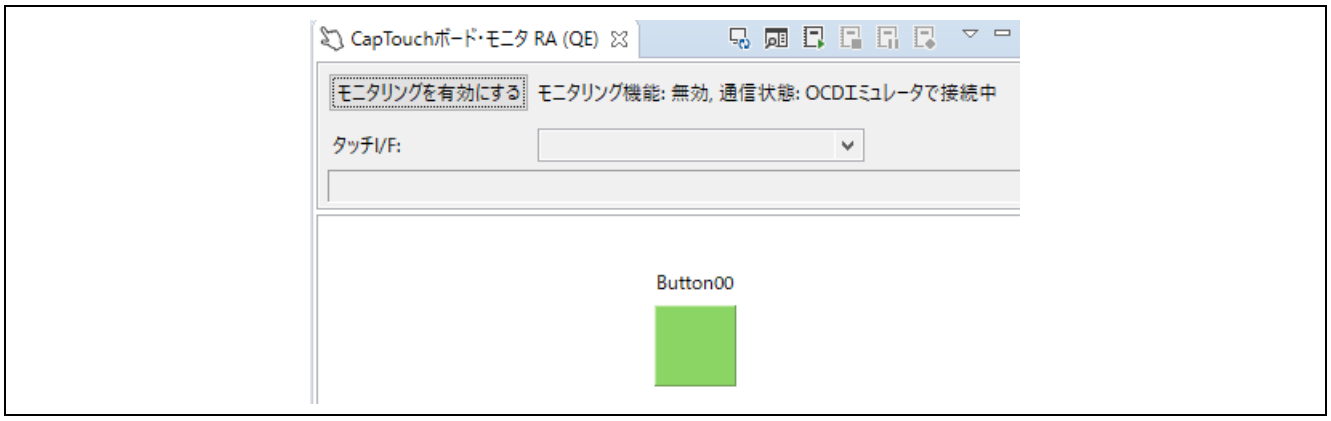

図78　[CapTouch ボード・モニタ(QE)]ウィンドウ（RA6M2）

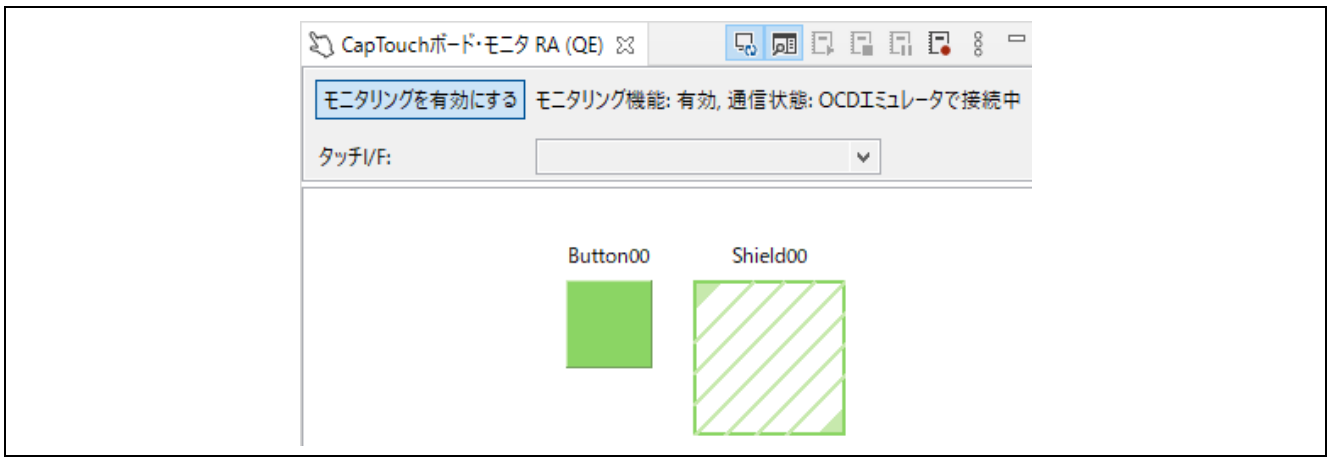

図79　[CapTouch ボード・モニタ (QE)]ウィンドウ（RA2L1）

11. [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。

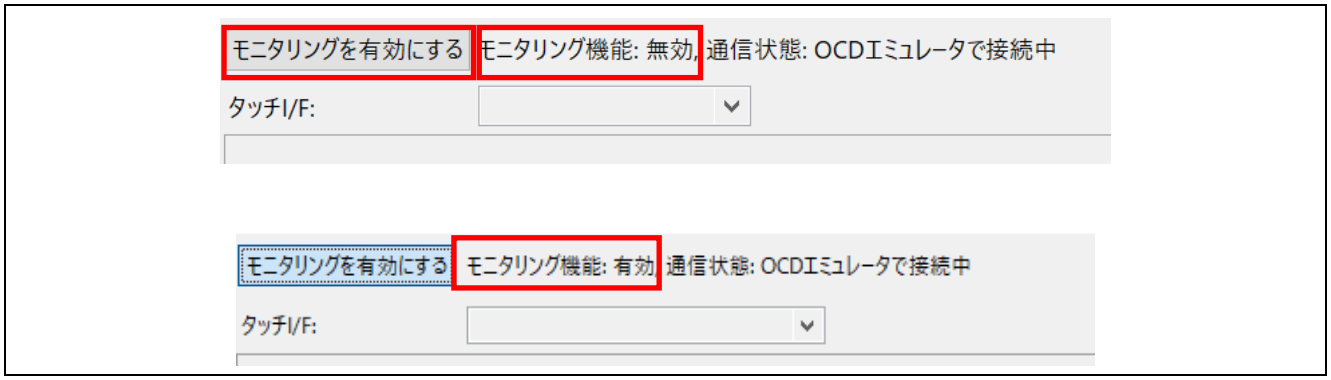

図80　モニタリング機能を有効

12. 静電容量タッチアプリケーションボード上の”Button00”（RA6M2 は TS02、RA2L1 は TS11-CFC）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。

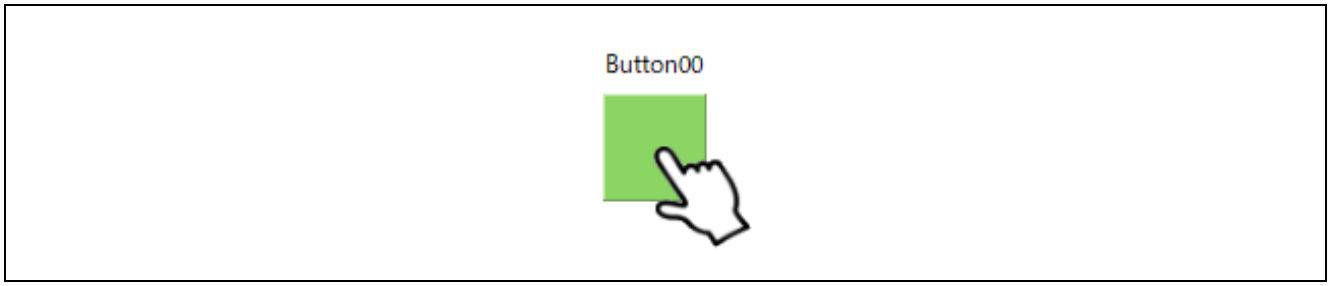

図81　[CapTouch ボード・モニタ (QE)]ウィンドウでのタッチ状態の確認

13. 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。

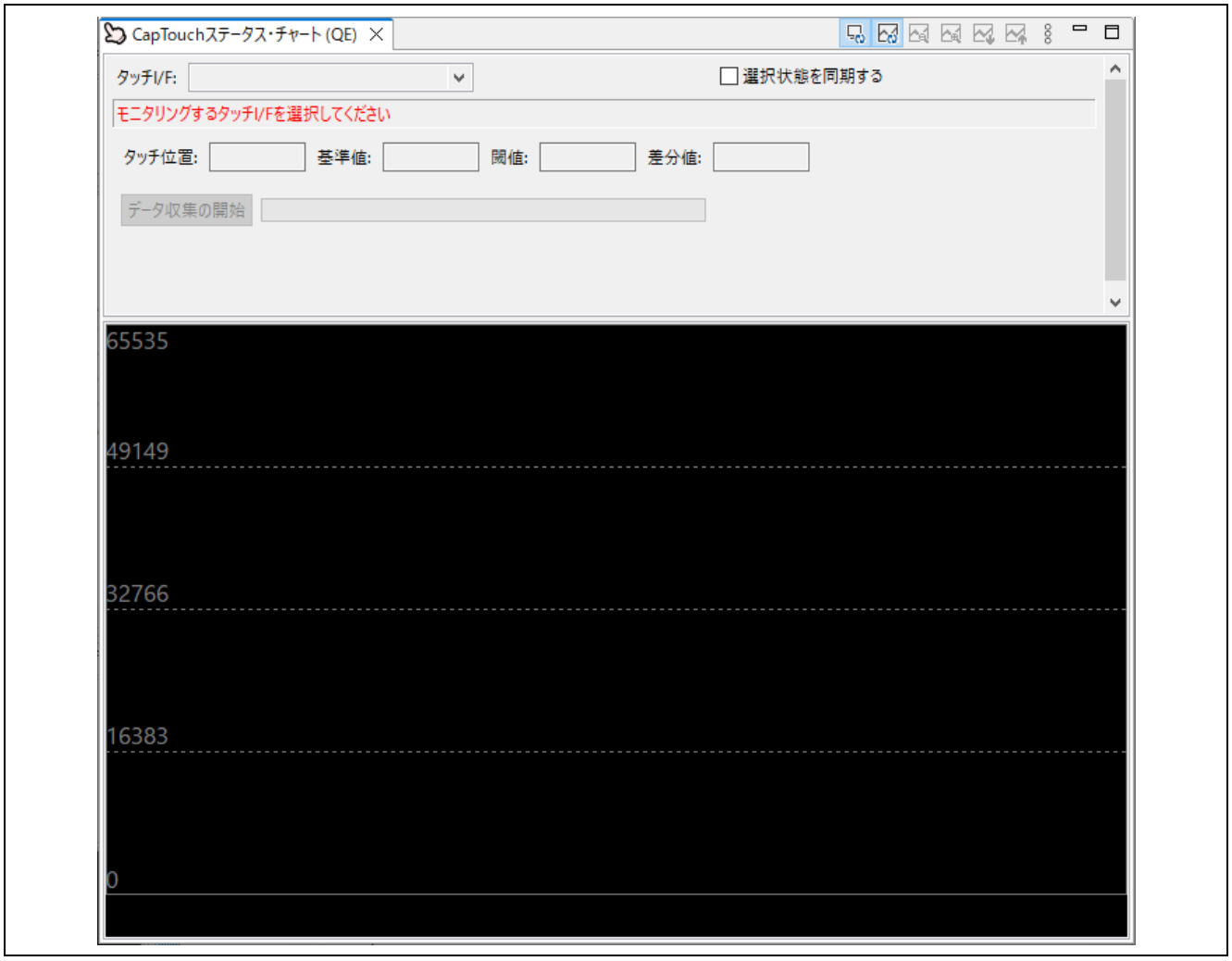

図82　[CapTouch ステータス・チャート (QE)]ウィンドウ

14. [タッチ I/F]のプルダウンメニューから”Button00 @ config01”を選択します。

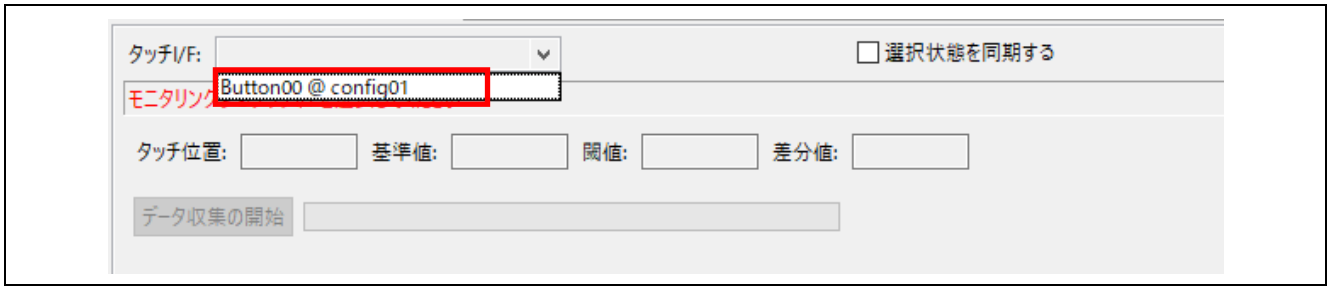

図83　タッチ I/F 選択

15. グラフには実行中の値が表示されます。ボード上の”Button00”（RA6M2 は TS02、RA2L1 は TS11-CFC）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。

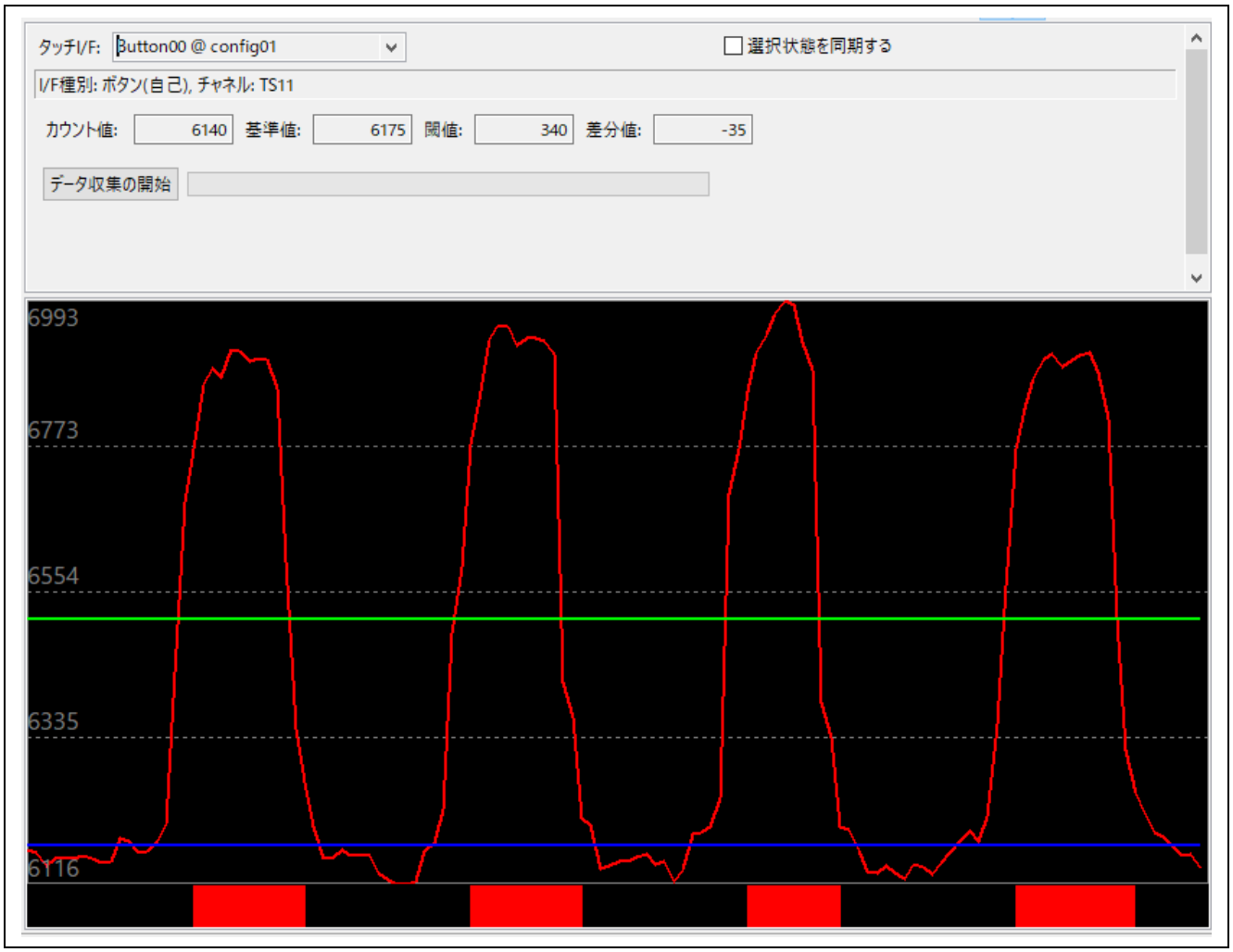

図84　[CapTouch ステータス・チャート (QE)]ウィンドウでのタッチ状態の確認

【注】：16～19 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。

16. 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”をクリックすると、データ収集設定ダイアログが表示されます。データ収集ダイアログのデータ収集数をプルダウンメニューから選択し、データ収集対象は、”非タッチ+タッチ時” を選択し、”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。

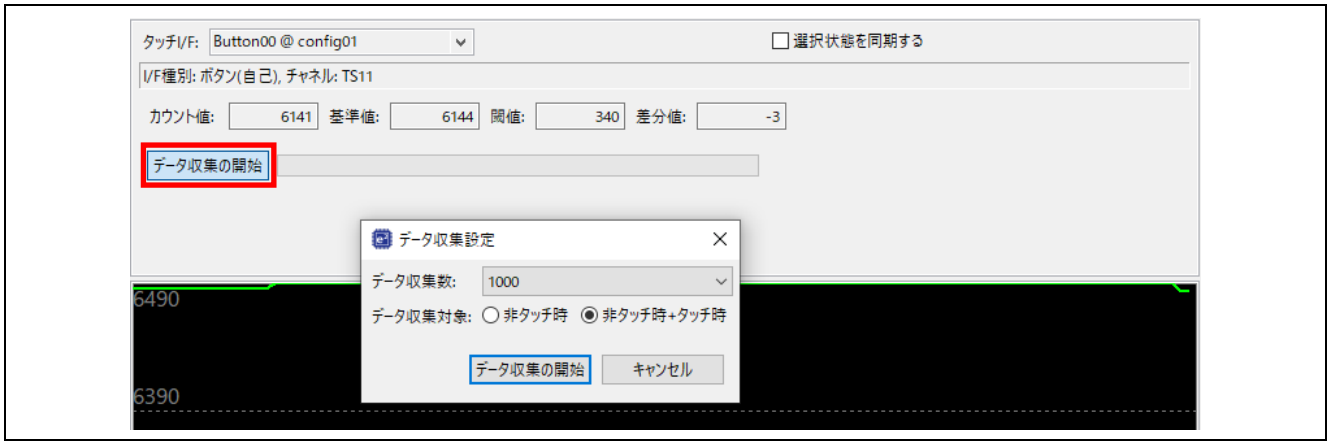

図85　[データ収集の開始]ボタン（タッチオフ状態）

17. 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、ダイアログが表示されます。[OK]をクリックしてダイアログを閉じます。

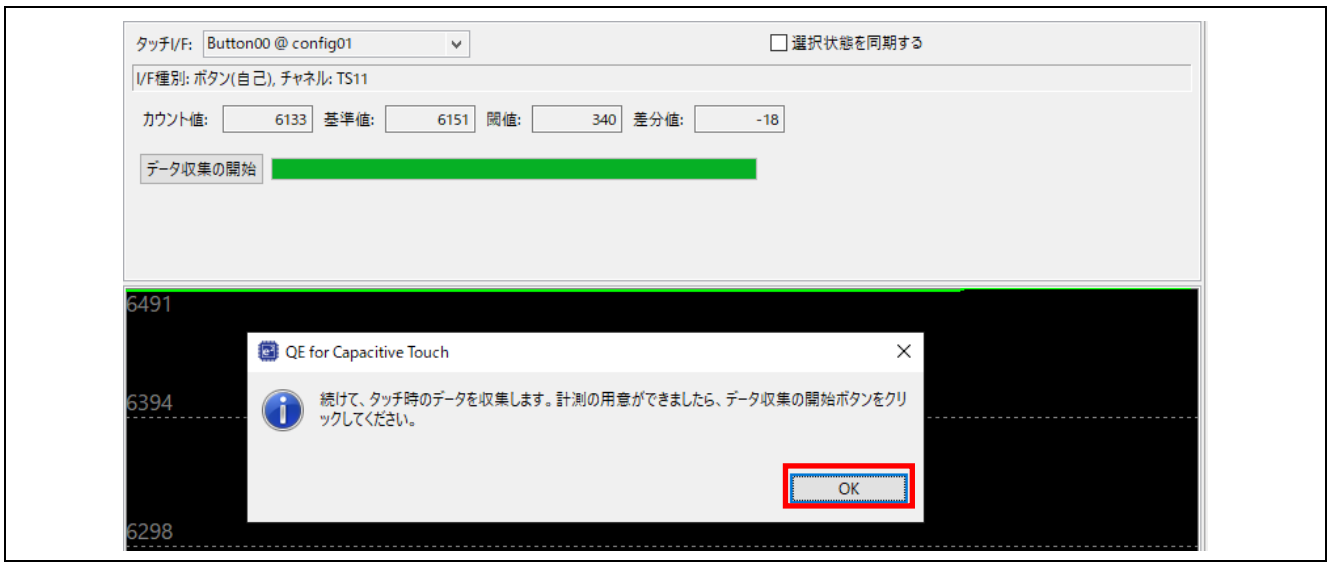

図86　[データ収集の終了]ボタン（タッチオフ状態）

18. 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。データ取集が完了するまで、電極に触れている状態を維持してください。

図87　[データ収集の開始]ボタン（タッチオン状態）

19. 緑色のバーが右端まで達し、データ収集が完了すると”標準偏差の測定結果”ダイアログが表示されます。このダイアログにはノイズ標準偏差値と SNR が表示されます。

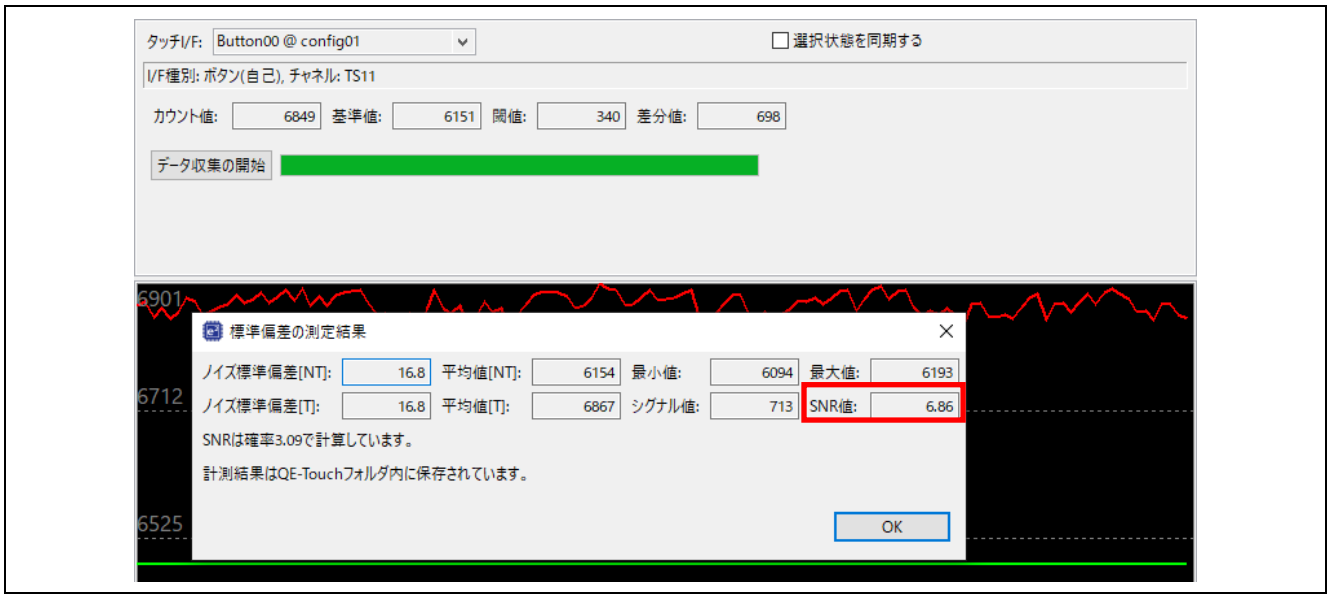

図88　測定結果の表示

6.8　シリアル通信を利用した QE for Capacitive Touch によるモニタリング

1. モニタリングを実行中の場合は、選択中の[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。

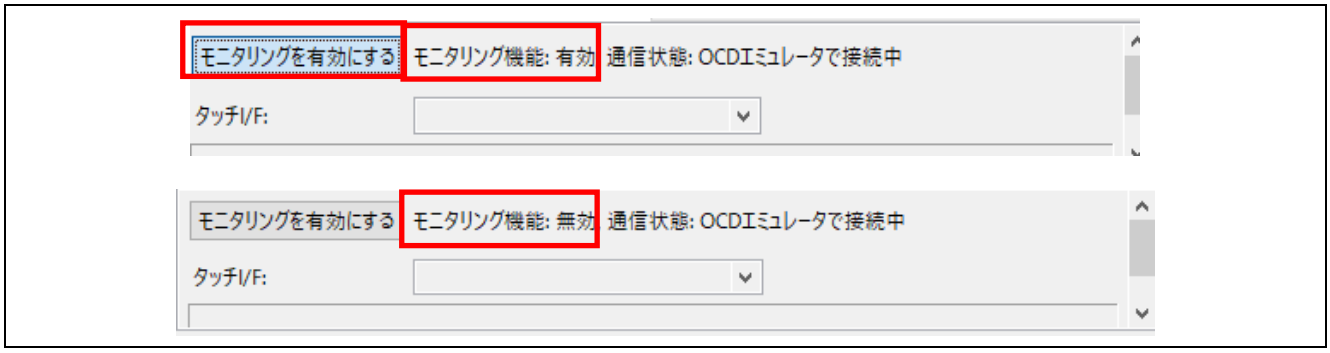

図89　モニタリング機能を無効

2. 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終了します。

3. エミュレータを PC とボードから外します。USB ケーブルがボードと PC に正しく接続されていることを確認します。

   【注】：エミュレータを接続した状態でも動作可能ですが、ここではエミュレータなしでモニタリングが可能なことを確認するため、エミュレータを外します。

4. ボードの RESET スイッチを押して、リセットします。

5. [CapTouch ワークフロー QE)]を表示し、[プロジェクトの選択]で”Capacitive_Touch_Project_Example”、[構成の選択]で”Capacitive_Touch_Project_Example.tifcfg”が選択されていることを確認します。

図90　プロジェクトの選択、構成の選択

6. [CapTouch ワークフロー (QE)]の左メニューで”モニタリングの開始(シリアル接続)”を選択します。[接続]を選択し、シリアル通信によるモニタリング機能を有効にします。

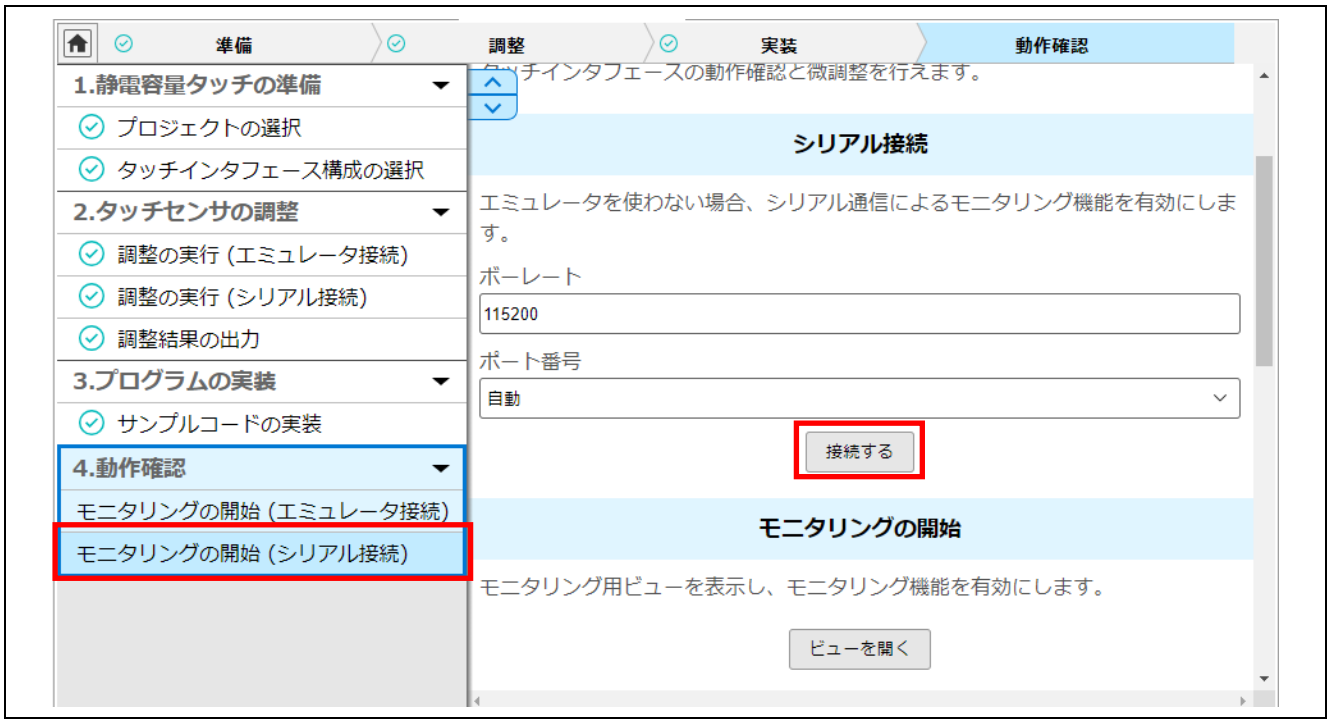

図91　[接続]ボタン

7. 画面下の[コンソール]ウィンドウに”COM*n* に接続しました。” のメッセージが表示されることを確認します。

   【注】：接続先の COM ポートの番号 *n* は、PC の環境によって異なります。

図92　[コンソール]ウィンドウの出力

8. 以降の手順は、「6.7 [式]ウィンドウとQE for Capacitive Touchによるモニタリング」の9項以降と同様です。

7. 変更後の qe_touch_sample.c のリスト

7.1 CTSU 計測動作開始トリガにソフトウェアトリガを使用する場合

```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2020-09-10
* DESCRIPTION : Main Program
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#define TOUCH_SCAN_INTERVAL_EXAMPLE (20)    /* milliseconds */

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;



    /* 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. */
        R_BSP_SoftwareDelay(TOUCH_SCAN_INTERVAL_EXAMPLE,
  BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

7.2　CTSU 計測動作開始トリガに外部トリガを使用する場合

```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2020-09-10
* DESCRIPTION : Main Program
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#define TOUCH_SCAN_INTERVAL_EXAMPLE (20)    /* milliseconds */

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;

    /* Initializes the software and sets the links defined in the control structure.
*/
    R_ELC_Open(g_elc.p_ctrl, g_elc.p_cfg);
    /* Globally enable event linking in the ELC. */
    R_ELC_Enable(g_elc.p_ctrl);

    /* 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) {}
    }

    /* Open AGT driver */
    R_AGT_Open(g_timer0.p_ctrl, g_timer0.p_cfg);
    /* Start AGT. */
    R_AGT_Start(g_timer0.p_ctrl);
```

```
    /* 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. */
        R_BSP_SoftwareDelay(TOUCH_SCAN_INTERVAL_EXAMPLE,
BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

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

https://www.renesas.com/

静電容量式タッチセンサユニット関連ページ

https://www.renesas.com/solutions/touch-key

https://www.renesas.com/fsp

https://www.renesas.com/qe-capacitive-touch

技術サポート問合せ

https://www.renesas.com/support

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	2020.2.28	-	初版発行
1.10	2020.4.6	2	アプリケーション例の概要の簡略
		3	ファイル生成場所の変更
		3	プロジェクト構成の設定の変更
		4	電源供給電圧設定方法の追加
		5	クロック周波数設定方法の変更
		5	タッチセンサに使用しないポート番号の変更
		8,9	DTC の設定方法を追加
		13	デバッグ方法の変更
		14	チューニングプロセスの省略
		18	アプリケーシ例のコード追加方法の変更
		19,20	モニタリングビューの開き方について変更
		23	標準偏差の測定方法の追加
		25,26	サンプルコードの更新
2.00	2022.6.7	全体	RA2L1(CTSU2)の環境、手順を追加 開発環境のバージョン、名称を更新 図表番号を追加 図の黒色矢印を赤色枠に変更
		3	2 章、4 章 Renesas Code Generator を削除 3 章 タイトルを変更
		4	5 章 実装内容を図に変更 6 章 タイトルを変更、以降の開発手順の内容を章から節に移動
		5	図 3 デバイス、ツールチェーン（RA2L1）追加
		6	6.1 7 ビルド、RTOS 選択の手順を追加
		7	図 5 プロジェクトテンプレート選択 追加
		10	図 10 クロック設定（RA2L1）追加
		12	図 15 TS 端子設定（RA2L1）追加
		13	6.2 10～11 シリアル通信によるモニタリングを使用する場合の手順を追加
		17～21	6.2 19～26 シリアル通信によるモニタリングを使用する場合の手順を追加
		21～24	6.2 27～33 CTSU 計測動作開始トリガに外部トリガを使用する場合の手順を追加
		28	6.3 7～8 シールド端子を作成する手順を追加
		31	6.4 1 エミュレータの接続を確認する手順を追加
		35	6.5 9 CTSU 計測動作開始トリガに外部トリガを使用する場合の手順を追加 6.5 11 ビルドエラーに関する注意事項を追加
		38, 39	6.6 6～7 CTSU 計測動作開始トリガにソフトウェアトリガを使用する場合の手順を追加
		41	図 75 [Enable Real-time Refresh]メニュー 追加
		43	図 79 [CapTouch ボード・モニタ RA,RL78,Synergy (QE)]ウィンドウ（RA2L1）追加
		49～51	6.8 シリアル通信を利用した QE for Capacitive Touch [RA,RL78,Synergy]によるモニタリング 追加

		52, 53	7.1 ソースコードを更新
		54, 55	7.2 CTSU 計測動作開始トリガに外部トリガを使用する場合を追加
		56	URL 変更
3.00	2024.1.11	3～7	開発環境情報の更新
		25, 26, 29, 32, 34～36, 42, 50	QE for Capacitive Touch のバージョンアップによるワークフロー関連の更新
		31	e² studio バージョンアップによるツールバーの更新
		45～48	QE for Capacitive Touch のバージョンアップによる、標準偏差の測定方法の更新
		56	URL 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。