RX ファミリ

QE CTSU モジュール Firmware Integration Technology

R01AN4469JU0110
Rev.1.10
2019.07.09

要旨

Firmware Integration Technology (FIT)を使用した QE CTSU モジュールは QE Touch モジュール向けのドライバレイヤとして開発されています。CTSU モジュールはユーザアプリケーションから直接アクセスされるものではなく、Touch ミドルウェアレイヤからのみアクセスされます。本アプリケーションノートは QE CTSU モジュールについて説明します。

対象デバイス

・RX113 グループ

・RX130 グループ

・RX23W、RX230、RX231 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

関連ドキュメント

QE と FIT を使用した静電容量タッチアプリケーションの開発 (R01AN4516)
QE Touch Diagnostic API Users' Guide (R01AN4785EU)
Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
RX100 Series VDE Certified IEC60730 Self-Test Code (R01AN2061ED)
RX v2 Core VDE Certified IEC60730 Self-Test Code for RX v2 MCU (R01AN3364EG)

目次

1. 概要

QE CTSU FIT モジュールは QE Touch FIT モジュールのドライバレイヤとして提供され、自己容量と相互容量のモードをサポートします。スキャンはソフトウェアトリガまたは外部トリガをサポートします。

1.1 機能

QE CTSU FIT モジュールがサポートする機能は以下のとおりです。

- Open () に指定するすべての引数は、QE for Capacitive Touch によって生成されます。
- センサは、自己容量または相互容量モードで構成できます。
- スキャンはソフトウェアトリガまたは外部トリガによって開始できます。

1.2 自己容量モード

自己容量モードの場合、ドライバは以下の用語を定義します。

自己容量モード

自己容量モードでは、1 つのボタン/キーを使用するために 1 つの CTSU のセンサを必要とします。一連のセンサを物理的に直線で配置して使用することでスライダを構成可能です。一連のセンサを円形に配置して使用した場合はホイールとなります。

スキャン順序

自己容量モードでは、ハードウェアは指定されたセンサの番号に従って昇順にスキャンします。例えば、アプリケーションでセンサ 5、8、2、3、6 が指定されている場合、ハードウェアは 2、3、5、6、8 の順にスキャンします。

要素

自己容量モードでの要素とは、スキャン順序におけるセンサのインデックスを表します。先の例でいうとセンサ 5 は要素 2 となります。

バッファ内容

ドライバ層では、CTSUSC センサカウンタと CTSURC リファレンスカウンタのレジスタ値を、スキャン構成内の各センサに対応するバッファに読み込みます。リファレンスカウンタ は使用されませんが、以下の 2 つの理由により、両方のレジスタ値をバッファに格納します。

1) スキャンのシーケンスに従い、両方のレジスタ値を読みだす必要があります。

2) これによって、以降の処理が割り込みと DTC で共通になります。

注意: ただし、R_CTSU_ReadData()等、このバッファにアクセスする API はセンサカウンタの値のみを読み込みます。

スキャン時間

センサのスキャンは、CTSU 周辺回路によってバックグラウンドで処理されるため、メインプロセッサの処理時間を消費しません。1 つのセンサをスキャンするには約 500us かかります。DTC が使用されていない場合、 2.2 us のオーバーヘッド (システムクロック 54MHz の場合) がレジスタから、またはレジスタへのデータ転送時にメインプロセッサで発生します。

メソッド/スキャン構成

これらの用語は、同じ意味で使用されます。1 つのメソッド (スキャン構成) は、スキャンするセンサの組み合わせであると同時に、スキャンするモード (自己容量または相互容量) を表します。QE for Capacitive Touch を使用する場合、アプリケーションは 1 ~ 8 のメソッドを使用できます。通常、アプリケーションは 1 つのメソッドを定義します。一方、より高度なアプリケーションでは、製品内でさまざまな機能が必要とされるときや、自己容量と相互容量のセンサ構成の組み合わせが存在するときに、異なるスキャン構成が必要となります。

補正

CTSU は電源投入での初期化時、一時的に補正モードに移行します。補正モードでは、MCU 個々での様々なタッチ入力値をシミュレーションし、理想的なセンサカウント値との比較を行います。カウント値は、理論値では直線に沿うはずですが、実際には微小な補正が必要になります。補正処理では CTSU 内部のレジスタ値を変更し、可能な限り最も正確なセンサ測定値を確保するためのタッチ層の補正係数を生成します。この補正により、MCU 製造プロセスにおける潜在的な微小バラつきに対応します。

オフセット調整

オフセット調整は Open()プロセス中に Touch ミドルウェア層によって実行されます。この調整では、温度、湿度、その他の電子部品やデバイスとの近接など、周囲の環境による微小な静電容量の違いに対応するために、QE for Capacitive Touch で生成された CTSU レジスタの設定値を微小調整します。このプロセスが完了すると、システムは調整されて操作の準備ができていると見なされます。

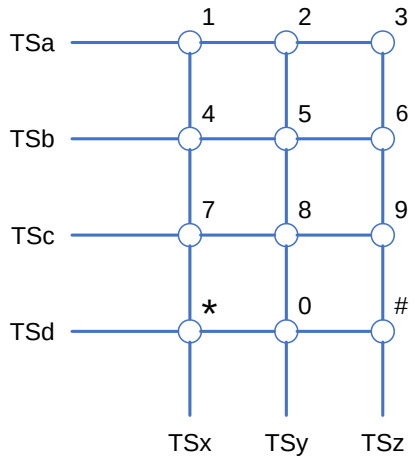
1.3　相互容量モード

相互容量モードの場合、ドライバは以下の用語を定義します。

相互容量モード

相互容量モードでは、1 つのタッチボタン/キーを使用するために 2 つの CTSU のセンサを必要とします。このモードは、通常 4 つ以上のキーを持つマトリックス/キーパッドが必要な場合に使用します。相互容量モードでは、自己容量モード時より少ないセンサで動作する必要があるためです。

ここでは、一般的な電話機のボタンを 4 行 3 列のマトリックスに置き換える例で説明します。ある行の 1 つの TS センサが各列の TS センサに接続され、ある列の 1 つの TS センサが各行の TS センサに接続されます。それぞれのボタン/キーはセンサの組み合わせ (以降、センサ・ペア) として識別され、センサ・ペアを構成するセンサは、RX、TX の順で表されます。行または列に RX と TX のセンサが混在することはありません。CTSU 周辺回路はセンサ・ペアにおける RX と TX 両方をスキャンし、その差分の演算結果は、ボタン/キーがタッチされたかどうかの判定に使用されます。



行を"RX"、列を "TX" とした場合、
ボタン 4 はTSb とTSx のセンサの組み合わせとなります。

行を"RX"、列を "TX" とした場合、
ボタン 8 はTSc とTSy のセンサの組み合わせとなります。

このように 12 のボタンのスキャンに 7 つのセンサしか必要ありません。対して、自己容量モードでは 12 のセンサが必要となります。

なお現時点で、スライダとホイールは相互容量モードでサポートされません。

スキャン順序

相互容量モードでは、CTSU 周辺回路はセンサ・ペアを、RX に設定されたセンサはプライマリ、TX に設定されたセンサはセカンダリとして昇順にスキャンします。例えば、センサ 10、11、3 が RX センサに設定され、センサ 2、7、4 が TX センサに設定された場合、CTSU 周辺回路は以下のセンサ・ペアの順にスキャンします。

3,2 - 3,4 - 3,7

10,2 - 10,4 - 10,7

11,2 - 11,4 - 11,7

要素

相互容量モードでの要素とは、スキャン順序におけるセンサ・ペアにインデックスを表します。先の例でいうとセンサ・ペア 10、7 は要素 5 となります。

バッファ内容

ドライバ層では、CTSUSC センサカウンタと CTSURC リファレンスカウンタのレジスタ値を、スキャン構成内のセンサ・ペアに対応するバッファに読み込みます。リファレンスカウンタは使用されませんが、以下の 2 つの理由により、両方のレジスタ値をバッファに格納します。

1) スキャンのシーケンスに従い、両方のレジスタ値を読みだす必要があります。

2) これにより、以降の処理が割り込みと DTC で共通になります。

注意: ただし、R_CTSU_ReadData()等、このバッファにアクセスする API はセンサカウンタの値のみを読み込みます。

スキャン時間

センサのスキャンは、CTSU 周辺回路によってバックグラウンドで処理されるため、メインプロセッサの処理時間を消費しません。1 つのセンサ・ペアのスキャンに約 1000us (1ms) かかります。DTC が使用されていない場合、 4.4 us のオーバーヘッド (システムクロック 54MHz の場合) が、レジスタから、またはレジスタへのデータ転送時にメインプロセッサで発生します。

メソッド/スキャン構成

これらの用語は、同じ意味で使用されます。1 つのメソッド (スキャン構成) は、スキャンするセンサの組み合わせであると同時に、スキャンするモード (自己容量または相互容量) を表します。QE for Capacitive Touch を使用する場合、アプリケーションは 1 ~ 8 のメソッドを使用できます。通常、アプリケーションは 1 つのメソッドが定義します。一方、より高度なアプリケーションでは、製品内でさまざまな機能が必要とされるときや、自己容量と相互容量のセンサ構成の組み合わせが存在するとき、異なるスキャン構成が必要となります。

補正

CTSU は電源投入での初期化時、一時的に補正モードに移行します。補正モードでは、MCU 個々での様々なタッチ入力値をシミュレーションし、理想的なセンサカウント値との比較を行います。カウント値は、理論値では直線に沿うはずですが、実際には微小な補正が必要になります。補正処理では CTSU 内部のレジスタ値を変更し、可能な限り最も正確なセンサ測定値を確保するためのタッチ層の補正係数を生成します。この補正により、MCU 製造プロセスにおける潜在的な微小バラつきに対応します。

オフセット調整

オフセット調整は Open()プロセス中に Touch ミドルウェア層によって実行されます。この調整では、温度、湿度、その他の電子部品やデバイスとの近接など、周囲の環境による微小な静電容量の違いに対応するために、QE for Capacitive Touch で生成された CTSU レジスタの設定値を微小調整します。このプロセスが完了すると、システムは調整されて操作の準備ができていると見なされます。

1.4　トリガソース

センサのスキャンはソフトウェアトリガまたはイベントリンクコントローラ (ELC) で起動された外部イベントのいずれかによって開始します。通常、ソフトウェアトリガが使用されます。一般的な使用方法は、タイマを用いて定期的にスキャンを開始させます。定期的なソフトウェアトリガと外部トリガの微小な実行タイミングの差異は、下記で説明します。

ソフトウェアトリガの場合、CMT のような周期タイマが設定され、その間隔はすべてのセンサをスキャンしデータが更新されるように設定されます (スキャン時間に関しては「1.2 自己容量モード」、「1.3 相互容量モード」を参照)。タイマが終了すると下記の一連のイベントが発生します。

- タイマ割り込みルーチンでフラグを設定します。
- メインアプリケーションがフラグの設定を検出するまでわずかな遅延が発生します (「図 1-1 定期的なソフトウェアトリガタイミング」の赤いバーを参照)。
- Touch ミドルウェアドライバを使用してメインアプリケーションはスキャンされたデータをロードし、内部の値 (移動平均など) を更新、別のスキャンを開始するためにソフトウェアトリガを発行する API をコールします。QE Touch FIT モジュールでは API 関数 R_TOUCH_UpdateDataAndStartScan()を使用します。

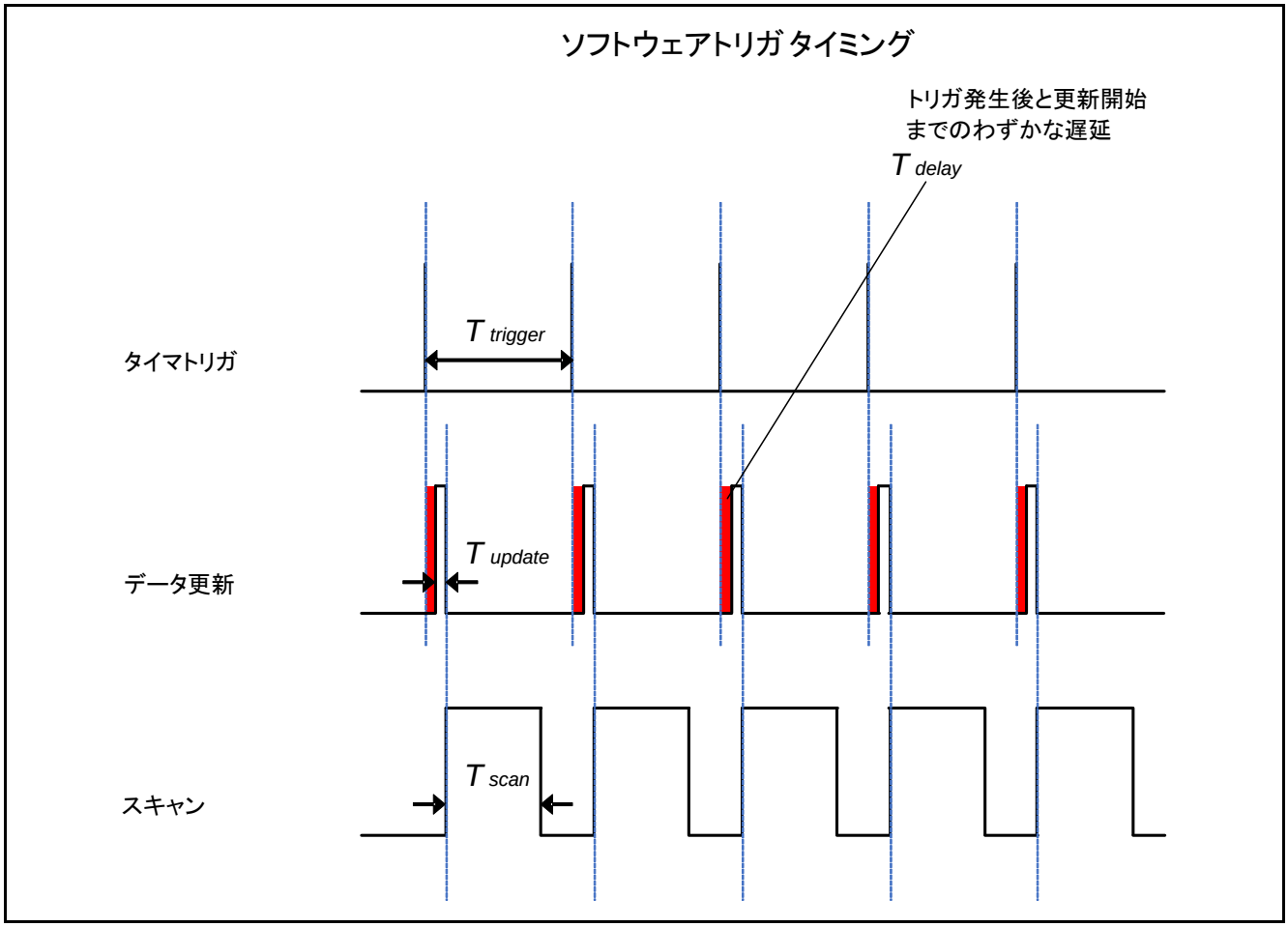


図 1-1 定期的なソフトウェアトリガタイミング

外部トリガの使用はソフトウェアトリガ使用時とほぼ同じです。通常、MTU のような周期タイマが設定され、その間隔はすべてのセンサをスキャンしデータが更新されるように設定されます (スキャン時間に関しては「1.2 自己容量モード」、「1.3 相互容量モード」を参照)。このタイマは次に、CTSU にリンクされている ELC にリンクされます。タイマが終了すると、下記の一連のイベントが発生します。

- CTSU のスキャンを開始するように設定された ELC が起動されます。
- スキャンが完了すると、CTSU スキャン完了割り込みルーチンでフラグがセットされます。
- メインアプリケーションがフラグの設定を検出までわずかな遅延が発生します (「図 1-2 定期的な外部トリガタイミング」の赤いバーを参照)。
- Touch ミドルウェアドライバを使用してメインアプリケーションはスキャンされたデータをロードし、内部の値 (移動平均など) を更新する API をコールします。FIT QE Touch FIT モジュールでは API 関数 R_TOUCH_UpdateData()を使用します。

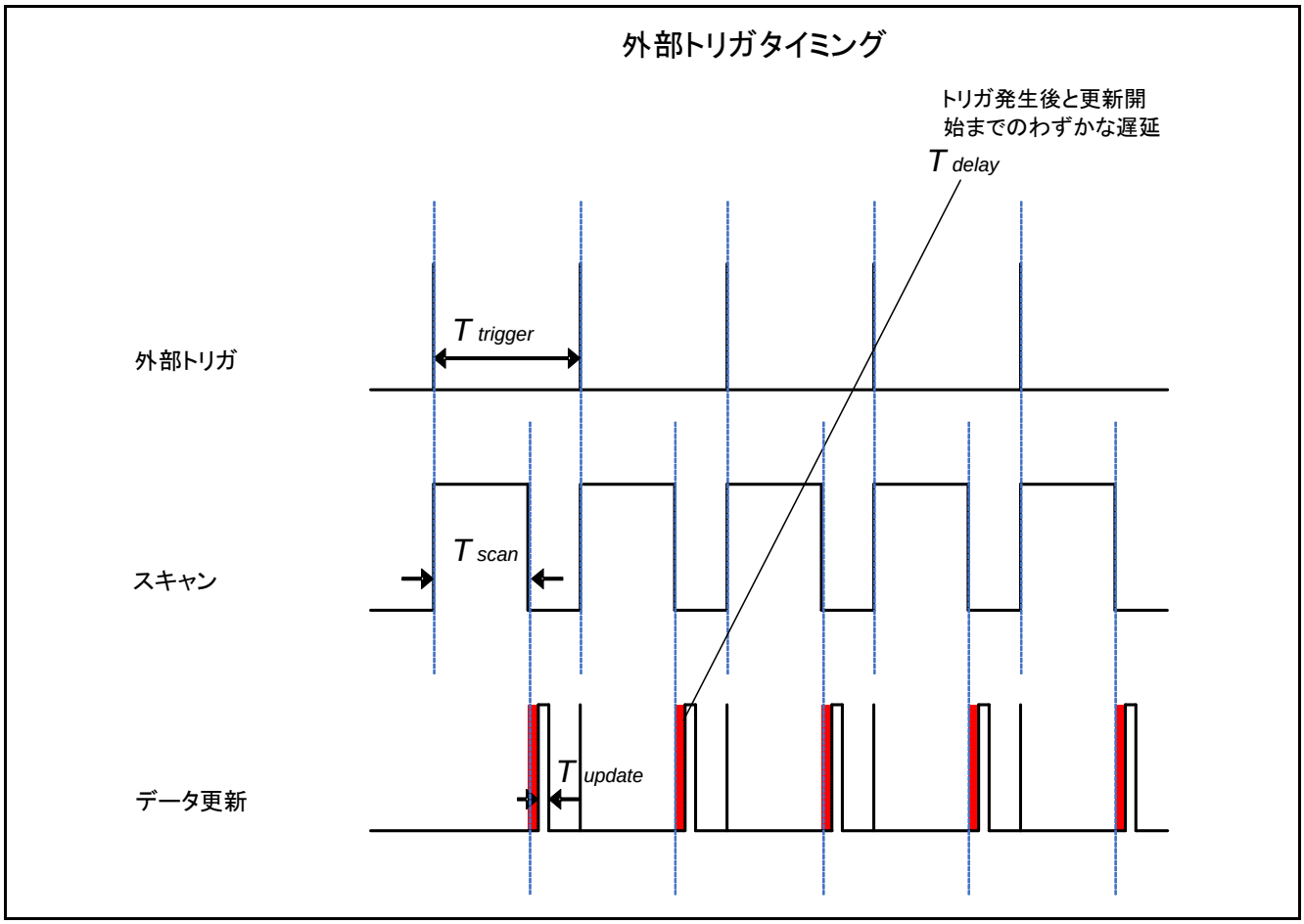


図 1-2 定期的な外部トリガタイミング

どちらのトリガメカニズムでもメインアプリケーションが割り込みルーチンにおけるフラグの設定を検出するために要する時間はユーザのポーリングアルゴリズムに応じて数マイクロ秒ずつ変化します。またスキャンされたデータの更新するために要する時間はセンサがタッチされたかどうか (elf-if ステートメントによる異なるパス)などによって数マイクロ秒変化する可能性があります。一般にこれらの微妙な違いはスキャンごとに最小限に留められ、スキャンの実行に費やされる全体の時間に比べて非常に小さくなっている。これらの理由により大多数のユーザはよりシンプルな設定のためソフトウェアトリガを選びます。しかしスキャン間隔のわずかなばらつきを許容できないようなケースでは外部トリガは有効な選択肢となります。ただし、外部トリガを使用する場合、DTC は使用できません。

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- CTSU

2.2 ソフトウェアの要求

このドライバは以下の FIT モジュールに依存しています。

- ボードサポートパッケージ (r_bsp) v4.24 以降（GCC/IAR サポートの場合、v5.20 以降）
- 静電容量式タッチセンサ対応開発支援ツール QE for Capacitive Touch V1.1.0

2.3 サポートされているツールチェーン

本 FIT モジュールは以下に示すツールチェーンで動作確認を行っています。

- Renesas CC-RX Toolchain v3.01.00
- IAR RX Toolchain v4.11.1
- GCC RX Toolchain v4.8.4.201801

2.4 制限事項

このコードはリエントラントではなく、複数の同時関数のコールを保護します。

2.5 ヘッダファイル

すべての API 呼び出しと使用されるインタフェース定義は“r_ctsu_qe_if.h”に記載されています。
ビルドごとの構成オプションは“r_ctsu_qe_config.h”で選択します。

2.6 整数型

このドライバは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.7　コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_ctsu_qe_config.h で行います。

オプション名および設定値に関する説明を、下表に示します。ビルド時に設定可能なコンフィグレーションオプションは r_ctsu_qe_config.h に含まれます。下表に設定の概要を示します。

r_ctsu_qe_config.h のコンフィギュレーションオプション	
CTSU_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は“1”	パラメータチェック処理をコードに含めるか選択できます。 “0”を選択すると、パラメータチェック処理をコードから省略できるため、コードサイズが削減できます。 “0”の場合、パラメータチェック処理をコードから省略します。 “1”の場合、パラメータチェック処理をコードに含めます。
CTSU_CFG_USE_DTC ※ デフォルト値は“0”	1 を設定することで CTSU の書き込みおよび読み込みの割り込みを処理にメインプロセッサではなく DTC を使用します。 警告: DTC がアプリケーションの他の場所で使用されている場合、このドライバの使用と競合する可能性があります。 DTC が使用されている場合、スキャンのトリガ種別に外部トリガを使用できません。
CTSU_CFG_INT_PRIORITY_LEVEL ※ デフォルト値は“8”	CTSU 割り込みの優先度レベル (DTC を使用する場合も必要) を設定します。範囲は 1 (低) – 15 (高) になります。
CTSU_CFG_SAFETY_LINKAGE_ENABLE ※ デフォルト値は“0”	0 を設定することで標準動作になります（セーフティモジュールは使用されません）。 1 を設定することでセーフティモジュールが使用するセクション情報が表示されます。

2.8　コードサイズ

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。掲載した値は、「2.3 サポートされているツールチェーン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル:2、最適化のタイプ:サイズ優先、データ・エンディアン:リトルエンディアンです。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

ROM、RAM のコードサイズ　例：2 つのボタンと 4 電極構成のスライダ 1 つを作成	
ROM の使用: CTSU_PARAM_CHECKING_ENABLE 1 > CTSU_PARAM_CHECKING_ENABLE 0 CTSU_CFG_USE_DTC 1 > CTSU_CFG_USE_DTC 0	
最小サイズ	ROM: 3191 バイト
	RAM: 484 バイト
最大サイズ	ROM: 3605 バイト
	RAM: 524 バイト

2.9　API データ型

API データ型は、r_typedefs_qe.h に記載されています。詳細は「3 API 関数」を参照ください。

2.10　戻り値

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_typedefs_qe.h で記載されています。

```
/* Return error codes */
typedef enum e_qe_err
{
    QE_SUCCESS = 0,
    QE_ERR_NULL_PTR,            // missing argument
    QE_ERR_INVALID_ARG,
    QE_ERR_BUSY,
    QE_ERR_ALREADY_OPEN,
    QE_ERR_CHAN_NOT_FOUND,
    QE_ERR_SENSOR_SATURATION,   // sensor value detected beyond linear portion
                                //  of correction curve
    QE_ERR_TUNING_IN_PROGRESS, // offset tuning for method not complete
    QE_ERR_ABNORMAL_TSCAP,      // abnormal TSCAP detected during scan
    QE_ERR_SENSOR_OVERFLOW,     // sensor overflow detected during scan
    QE_ERR_OT_MAX_OFFSET,       // CTSU SO0 offset reached max value and
                                //   sensor offset tuning incomplete
    QE_ERR_OT_MIN_OFFSET,       // CTSU SO0 offset reached min value and
                                //   sensor offset tuning incomplete
    QE_ERR_OT_WINDOW_SIZE,      // offset tuning window too small for sensor
                                //   to establish a reference count
    QE_ERR_OT_MAX_ATTEMPTS,     // 1+ sensors still not tuned for method
    QE_ERR_OT_INCOMPLETE,       // 1+ sensors still not tuned for method
    QE_ERR_TRIGGER_TYPE,        // function not available for trigger type
} qe_err_t;
```

2.11 FIT モジュールの追加方法

2.11.1 ソースツリーへの追加とプロジェクトインクルードパスの追加

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.11.2 Smart Configurator を使用しない場合のドライバオプションの設定

CTSU 固有のオプションは r_config¥r_ctsu_qe_config.h で確認および編集できます。

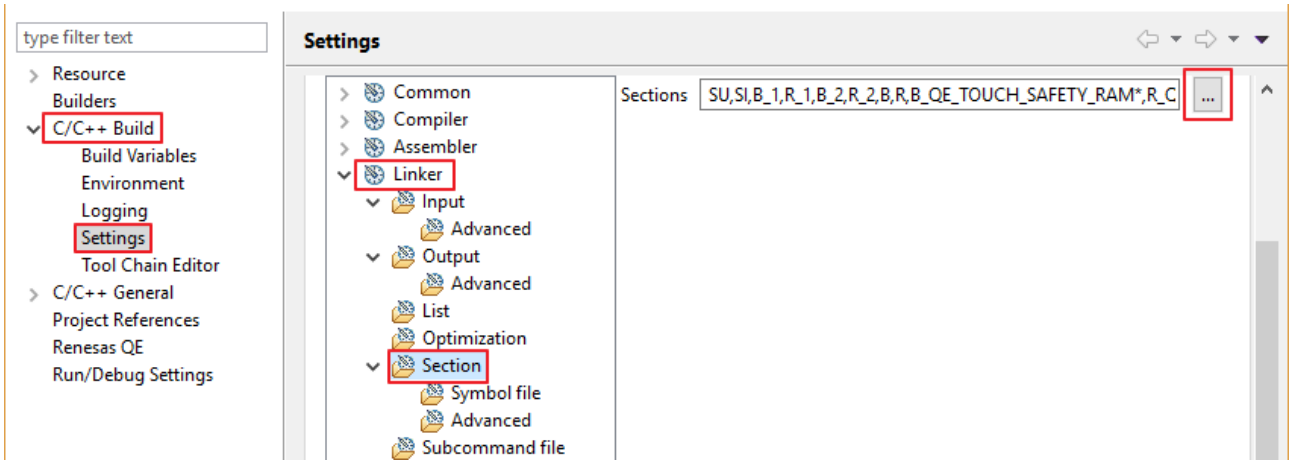
デフォルト値を含むリファレンスのヘッダファイル (編集用ではありません) は r_ctsu_qe¥ref¥r_ctsu_qe_config_reference.h として格納されています。

2.12 セーフティモジュールの設定

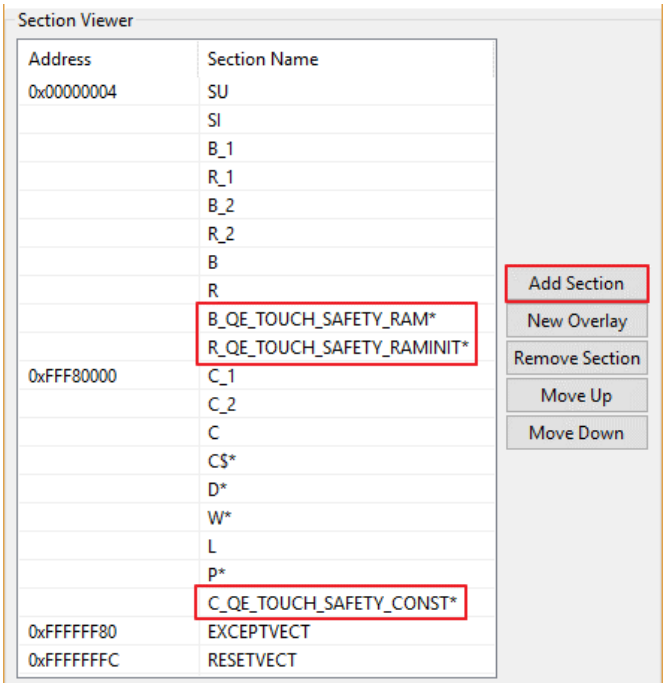
機能安全のクラス B に対応したセーフティモジュールを構築する場合(アプリケーションノート R01AN4785EU を参照)、プロジェクトのリンカスクリプトに特別なセクションを追加する必要があります。そして、それらのセクションに対応する#pragma section (セクション切り替え)には、ドライバ構成ファイルの設定が必要です。

2.12.1 Renesas Toolchain

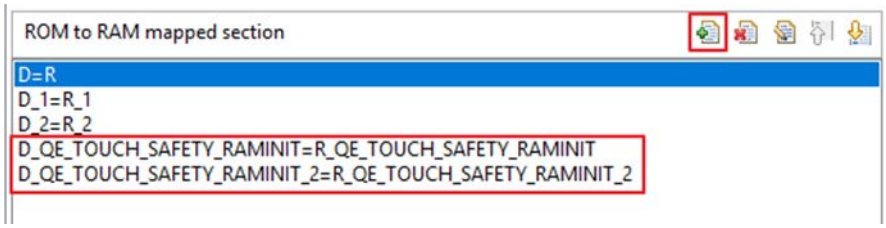
1) e² studio 上で、プロジェクトの[プロパティ] - > [C / C ++ビルド] - > [設定] - > [Linker] - > [セクション]の順に選択し、[セクション]テキストボックスの右側にある[…]ボタンをクリックしてセクションビューアを開きます。



2) セクションビューア内にセクション“B_QE_TOUCH_SAFETY_RAM*”、“R_QE_TOUCH_SAFETY_RAMINIT*”、“C_QE_TOUCH_SAFETY_CONST*”を追加します。

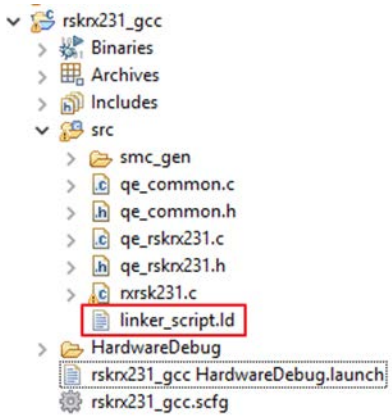


3) プロジェクトの[プロパティ] - > [C / C ++ビルド] - > [設定] - > [Linker] - > [セクション] - > [シンボル]に移動し、以下の割り当てを追加します。



2.12.2　GCC Toolchain

1) テキストエディタでプロジェクトリンカファイル「linker_script.ld」を開きます。このファイルは、e2 studio でのプロジェクトの「src」ディレクトリ内にあります。



2)　“.text”ROM 領域にセクション“P_QE_TOUCH_DRIVER”を追加します。

```
19
20
21
22
23
24
25
26
27
28
29
```

```
.text 0xFFF80000: AT(0xFFF80000)
{
    *(.text)
    *(.text.*)
    *(P)

    /* Adding P sections for QE safety code */
    *(P_QE_TOUCH_DRIVER)

    etext = .;
} > ROM
```

3) “.rodata”ROM 領域にセクション“C_QE_TOUCH_SAFETY_CONSTANT”、“C_QE_TOUCH_SAFETY_CONSTANT_1”および“C_QE_TOUCH_SAFETY_CONSTANT_2”を追加します。

```
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
```

```
.rodata :
{
    *(.rodata)
    *(.rodata.*)
    *(C_1)
    *(C_2)
    *(C)

    /* Adding C sections for QE safety code */
    *(C_QE_TOUCH_SAFETY_CONSTANT)
    *(C_QE_TOUCH_SAFETY_CONSTANT_1)
    *(C_QE_TOUCH_SAFETY_CONSTANT_2)

    _erodata = .;
} > ROM
```

4) 初期化された“.data”RAM 領域にセクション“D_QE_TOUCH_SAFETY_RAMINIT”および“D_QE_TOUCH_SAFETY_RAMINIT_2”を追加します。

```
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
```

```
.data : AT(_mdata)
{
    _data = .;
    *(.data)
    *(.data.*)
    *(D)
    *(D_1)
    *(D_2)

    /* Adding D sections for QE safety code */
    *(D_QE_TOUCH_SAFETY_RAMINIT)
    *(D_QE_TOUCH_SAFETY_RAMINIT_2)

    _edata = .;
} > RAM
```

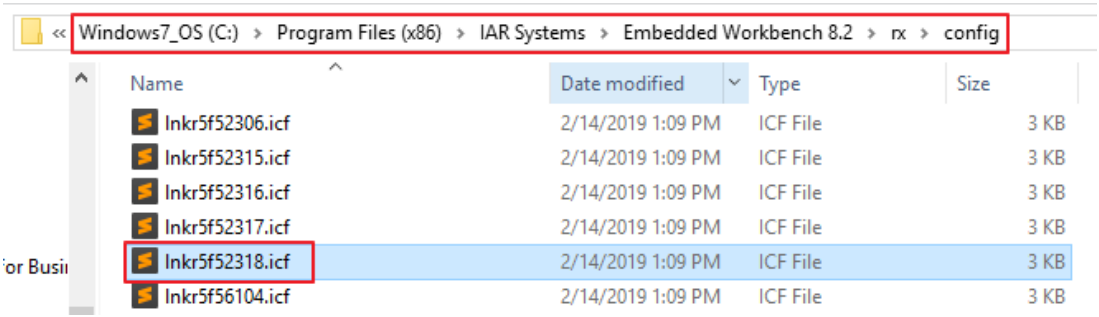
5) 初期化されていない“.bss”RAM 領域にセクション“B_QE_TOUCH_SAFETY_RAM_2”を追加します。

```
143         .bss :
144         {
145             _bss = .;
146             *(.bss)
147             *(.bss.**)
148             *(COMMON)
149             *(B)
150             *(B_1)
151             *(B_2)
152
153             /* Adding B sections for QE safety code */
154             *(B_QE_TOUCH_SAFETY_RAM_2)
155
156             _ebss = .;
157             _end = .;
158         } > RAM
```

6) 注：リンカスクリプトで追加または変更する可能性のあるセクションについては、BSP アプリケーションノートを参照してください。

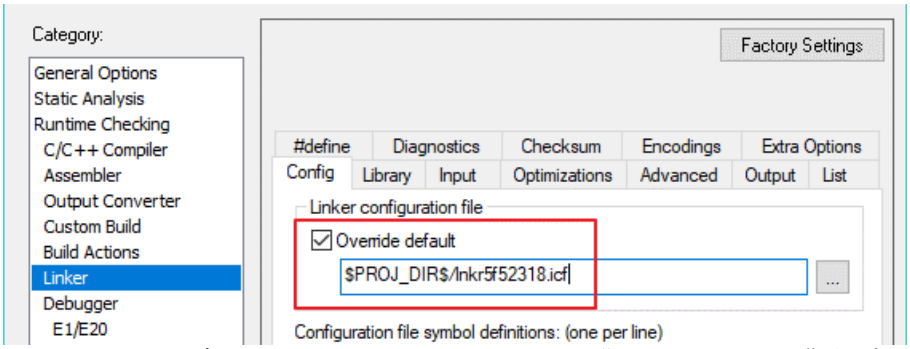
2.12.3 IAR Toolchain

1) Windows Explorer 上で、ターゲット MCU のテンプレートリンカファイルを IAR Workbench プロジェクトの最上位ディレクトリにコピーします。以下は、RSK RX231 用のサンプルリンカファイルを含むデフォルトの IAR ツールチェーンインストールパス(概略)です。



2) リンカファイルをプロジェクトに含めるためには、プロジェクト名を右クリックして[Add] - > [Add File]を選択します。

3) プロジェクトにファイルを追加したら、プロジェクト名を右クリックして[Options]を選択します。表示されたダイアログボックスで、左側の“Category”欄から“Linker”を選択し、右側表示枠の“Override default”にチェックを入れて、追加したファイルへのパスを変更します。



4) テキストエディタでリンカファイルを開いて“initialize by copy”領域にセクション“D_QE_TOUCH_SAFETY_RAMINIT”と“D_QE_TOUCH_SAFETY_RAMINIT_2”を追加します。

```
22
23  initialize by copy {    rw,
24                          ro section D,
25                          ro section D_1,
26                          ro section D_2,
27                          ro section D_2,
28                          ro section D_QE_TOUCH_SAFETY_RAMINIT,
29                          ro section D_QE_TOUCH_SAFETY_RAMINIT_2,
30                                                  };
```

5) 以下に示すように、“ROM32”領域にセクション“C”と“P”を追加し、“RAM32”領域にセクション“D”と“B”を追加します。“ro”と“rw”の種類の違いに注意してください。

```
55  "ROM32":place in ROM_region32       {  ro,
56                                          ro section C_QE_TOUCH_SAFETY_CONSTANT,
57                                          ro section C_QE_TOUCH_SAFETY_CONSTANT_1,
58                                          ro section C_QE_TOUCH_SAFETY_CONSTANT_2,
59                                          ro section P_QE_TOUCH_DRIVER
60                                                                          };
61  "RAM32":place in RAM_region32       { rw,
62                                          ro section D,
63                                          ro section D_1,
64                                          ro section D_2,
65                                          ro section D_QE_TOUCH_SAFETY_RAMINIT,
66                                          ro section D_QE_TOUCH_SAFETY_RAMINIT_2,
67                                          rw section B_QE_TOUCH_SAFETY_RAM_2,
68                                          block HEAP };
```

2.12.4　ドライバ構成ファイル

セクション名を正しく生成するには、ドライバ構成ファイルで次の設定を行う必要があります。

File “r_ctsu_qe_config.h”:

```
#define CTSU_CFG_SAFETY_LINKAGE_ENABLE    (1)
```

File “r_touch_qe_config.h”:

```
#define TOUCH_CFG_SAFETY_LINKAGE_ENABLE   (1)
```

2.13　IEC 60730 準拠

安全規格である IEC 60335-1 と IEC60730 クラス B の両方への準拠について、診断コードモジュールと共に CTSU ドライバと Touch ドライバは、最終アプリケーションに適用される MCU に対して、RX 60730 セルフテストコードライブラリに使用する必要があります。RX 60730 セルフテストコードライブラリは、ルネサス Web サイト上（https://www.renesas.com/us/en/）にあります。または、詳細についてはルネサスサポート窓口にお問い合わせください。

これらのセルフテストソフトウェアライブラリとアプリケーションノートは、ルネサス Web サイトにあります。

・RX100 Series VDE Certified IEC60730 Self-Test Code (R01AN2061ED)
・RX v2 Core VDE Certified IEC60730 Self-Test Code for RX v2 MCU (R01AN3364EG)

ファイル「r_ctsu_qe_pinset.c」および「r_ctsu_qe_pinset.h」は、Smart Configurator によって生成されます。これらのファイルは、CTSU ドライバおよび Touch ドライバパッケージの一部とは見なされず、IEC60730 準拠の一環として必須ではないことにご注意ください。

3. API 関数

3.1 概要

本モジュールには以下の関数が含まれます。

関数	説明
R_CTSU_Open()	本モジュールを初期化します。
R_CTSU_StartScan()	ソフトウェアトリガでセンサのスキャンを開始します。
R_CTSU_ReadButton()	指定したボタンの計測値を取得します。
R_CTSU_ReadSlider()	指定したスライダまたはホイールのすべての計測値を取得します。
R_CTSU_ReadElement()	指定したセンサ/センサ・ペア(相互容量モード時)の計測値を取得します。
R_CTSU_ReadData()	すべての計測値を取得します。
R_CTSU_ReadReg()	レジスタ値を取得します。
R_CTSU_WriteReg()	レジスタ値を書き込みます。
R_CTSU_Control()	特別な操作を実行します。
R_CTSU_Close()	本モジュールを終了します。
R_CTSU_GetVersion()	本モジュールのバージョン番号を返します。

3.2　R_CTSU_Open

この関数は、本モジュールの初期化をする関数です。この関数は他の API 関数を使用する前に実行する必要があります。

Format

```
qe_err_t R_CTSU_Open(ctsu_cfg_t *p_ctsu_cfgs[],
                     uint8_t    num_methods,
                     qe_trig_t  trigger);
```

Parameters

p_ctsu_cfgs

スキャン構成の配列へのポインタ (QE for Capacitive Touch によって生成された gp_ctsu_cfgs[])

num_methods

スキャン構成の配列の数 (QE for Capacitive Touch によって生成された QE_NUM_METHODS)

trigger

スキャントリガソース (QE_TRIG_SOFTWARE または QE_TRIG_EXTERNAL)

Return Values

QE_SUCCESS	*/* CTSU 初期化に成功しました */*
QE_ERR_NULL_PTR	*/* 引数のポインタが指定されていません */*
QE_ERR_INVALID_ARG	*/* “num_methods”または“trigger”が不正です。（QE ツールが生成した配列 p_ctsu_cfgs [ ]の内容は検査されません） */*
QE_ERR_BUSY	*/* 別の CTSU の操作が実行中のため実行できません*/*
QE_ERR_ALREADY_OPEN	*/* Close()のコールなしに Open()がコールされました */*
QE_ERR_ABNORMAL_TSCAP	*/* 補正中に TSCAP エラーが検出されました */*
QE_ERR_SENSOR_OVERFLOW	*/* 補正中にセンサオーバフローエラーが検出されました */*
QE_ERR_SENSOR_SATURATION	*/* 初期センサ値が補正曲線の直線部を超えています */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

この関数は、QE CTSU FIT モジュールを初期化します。初期化には、レジスタの初期化、割り込みの有効化、“r_ctsu_qe_config.h”で有効になっている場合 DTC の初期化が含まれます。

この関数はまた QE for Capacitive Touch でチューニングされたパラメータを最適化する初期補正アルゴリズムを実行します。これは、静電容量タッチのアプリケーションシステムにおける小さな環境的もしくは物理的な違いといったボードごとの小さなばらつきを吸収するために実行されます。

この関数は他の API 関数を使用する前にコールする必要があることに注意してください。

Reentrant

不可

Example

```
qe_err_t err;

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

/* Initialize the API. */
err = R_CTSU_Open(gp_ctsu_cfgs, QE_NUM_METHODS, QE_TRIG_SOFTWARE);

/* Check for errors. */
if (err != QE_SUCCESS)
{
    . . .
}
```

Special Notes:

当関数のコール前にポートを初期化する必要があります。

3.3　R_CTSU_StartScan

ソフトウェアトリガによるセンサスキャンの開始をする関数です。

Format

```
qe_err_t  R_CTSU_StartScan(void);
```

Parameters

なし

Return Values

QE_SUCCESS	*/* スキャンを開始しました */*
QE_ERR_TRIGGER_TYPE	*/* Open() で外部トリガを指定されました */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

DTC の使用が“r_ctsu_qe_config.h”で有効になっている場合、この関数は最初に DTC へ CTSU レジスタ転送アドレスを設定後、ハードウェアセンサスキャンを開始します。

Reentrant

なし

Example

```
qe_err_t err;

/* Initiate a sensor scan by software trigger */
err = R_CTSU_StartScan();

/* Check for errors. */
if (err != QE_SUCCESS)
{
    . . .
}
```

Special Notes:

外部トリガモードで使用時、当関数をコールしないでください。

3.4　R_CTSU_ReadButton

前回スキャン時のボタンの計測値 (補正は未適用) を読み込む関数です。

Format

```
qe_err_t  R_CTSU_ReadButton(btn_ctrl_t *p_btn_ctrl,
                            uint16_t   *p_value1,
                            uint16_t   *p_value2);
```

Parameters

p_btn_ctrl

ボタン制御向け構造体 (QE for Capacitive Touch で生成される配列 g_buttons_xxx[]、”xxx”は”config01” など QE for Capacitive Touch で設定されたメソッド名を示す) へのポインタ。

p_value1

プライマリ計測値を格納する変数へのポインタ。

p_value2

セカンダリ計測値を格納する変数へのポインタ。相互容量モード時のみ使用されます。

Return Values

QE_SUCCESS	*/* ボタンの計測値を読み込ました */*
QE_ERR_NULL_PTR	*/* 引数のポインタが指定されていません */*
QE_ERR_INVALID_ARG	*/* ボタン制御向け構造体で指定された要素のインデックスが不正です */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

当関数は指定されたボタンの計測値 (補正は未適用) を読み込みます。

Reentrant

なし

Example: 構成名 PANEL、ボタン名 ONOFF、自己容量モード

```
qe_err_t err;
uint16_t sensor_val;

/* Load sensor value for button */
err = R_CTSU_ReadButton(&g_buttons_panel[PANEL_INDEX_ONOFF],
                        &sensor_val, NULL);
```

Example: 構成名 PANEL、ボタン名 ONOFF、相互容量モード

```
qe_err_t err;
uint16_t primary,secondary,value;

/* Load sensor values for button */
err = R_CTSU_ReadButton(&g_buttons_panel[PANEL_INDEX_ONOFF],
                    &primary, &secondary);

value = secondary - primary;
```

Special Notes:

なし

3.5　R_CTSU_ReadSlider

前回スキャン時のスライダまたはホイールの計測値 (補正は未適用) を読み込む関数です。

Format

```
qe_err_t  R_CTSU_ReadSlider(sldr_ctrl_t *p_sldr_ctrl,
                            uint16_t    *p_buf1,
                            uint16_t    *p_buf2);
```

Parameters

p_slider_ctrl

スライダ制御向け構造体 (QE for Capacitive Touch で生成される配列 g_sliders_xxx[]、”xxx” は”config01”など QE for Capacitive Touch で設定されたメソッド名を示す) へのポインタ。

p_buf1

プライマリ計測値を格納するバッファへのポインタ。

p_buf2

セカンダリ計測値を格納するバッファへのポインタ。相互容量モードのために予約 (現時点ではサポートしていません)

Return Values

```
QE_SUCCESS          /* スライダの計測値を読み込みました */
QE_ERR_NULL_PTR     /* 引数のポインタが指定されていません */
QE_ERR_INVALID_ARG  /* スライダ制御向け構造体で指定された要素のインデックスが不正です */
QE_ERR_BUSY         /* 別の CTSU 操作が実行中のため実行できません */
```

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

当関数は指定されたスライダまたはホイールの計測値 (補正は未適用) を読み込みます。

Reentrant

なし

Example: 構成名 PANEL、スライダ名 DIMMER、自己容量モード

```
qe_err_t err;
uint16_t data[QE_MAX_SLDR_ELEM_USED];

/* Load sensor values from low to high for slider */
err = R_CTSU_ReadSlider(&g_sliders_panel[PANEL_INDEX_DIMMER], data, NULL);
```

Example: 構成名 PANEL、ホイール名 SERVO、自己容量モード

```
qe_err_t err;
uint16_t data[QE_MAX_WHEEL_ELEM_USED];

/* Load sensor values from low to high for wheel */
err = R_CTSU_ReadSlider(&g_wheels_panel[PANEL_INDEX_SERVO], data, NULL);
```

Special Notes:

相互容量モード時のスライダは現時点でサポートされておりません。

3.6　R_CTSU_ReadElement

前回スキャン時の指定要素の計測値 (補正は未適用) を読み込む関数です。

Format

```
qe_err_t  R_CTSU_ReadElement(uint8_t    element,
                             uint16_t   *p_value1,
                             uint16_t   *p_value2);
```

Parameters

element

取得対象のセンサ/センサ・ペアのインデックス。

p_value1

プライマリ計測値を格納する変数へのポインタ。

p_value2

セカンダリ計測値を格納する変数へのポインタ。相互容量モード時のみ使用。

Return Values

QE_SUCCESS	*/* 要素の計測値を読み込みました */*
QE_ERR_NULL_PTR	*/* 引数のポインタが指定されていません */*
QE_ERR_INVALID_ARG	*/* 要素のインデックスが不正です */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

当関数は指定された要素のインデックスの計測値 (補正は未適用) を取得します。

Reentrant

なし

Example: 自己容量モード

```
qe_err_t err;
uint16_t sensor_val;

/* Load sensor value for element 4 */
err = R_CTSU_ReadElement(4,&sensor_val, NULL);
```

Example: 相互容量モード

```
qe_err_t err;
uint16_t primary,secondary,value;

/* Load sensor values for element 7 */
err = R_CTSU_ReadElement(7, &primary, &secondary);

value = secondary – primary;
```

Special Notes:

なし

3.7　R_CTSU_ReadData

前回スキャン時のすべての計測値 (補正は未適用) を読み込む関数です。

Format

```
qe_err_t  R_CTSU_ReadData(uint16_t   *p_buf,
                          uint16_t   *p_cnt);
```

Parameters

p_buf

すべてのセンサの計測値を格納するバッファへのポインタ。(基準値は含みません)

p_cnt

取得した計測値の数を格納する変数へのポインタ。相互容量時は自己容量時の 2 倍の数が格納されます。

Return Values

QE_SUCCESS	*/* 計測値を読み込みました */*
QE_ERR_NULL_PTR	*/* 引数のポインタが指定されていません */*
QE_ERR_INVALID_ARG	*/* 要素のインデックスが不正です */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

この関数は、前回スキャン時のすべての計測値 (補正は未適用) をバッファに読み込み、読み込んだデータの数を返します。データの数は、自己容量モードでは使用されるセンサの数に等しく、相互容量モードではセンサ・ペアの数の 2 倍になります。

Reentrant

なし

Example: 構成名 PANEL、自己容量モード

```
qe_err_t err;
uint16_t buf[PANEL_NUM_ELEMENTS];
uint16_t cnt;

/* Load all sensor values */
err = R_CTSU_ReadData(buf,&cnt);
```

Example: 構成名 PANEL、相互容量モード

```
qe_err_t err;
uint16_t buf[PANEL_NUM_ELEMENTS*2];
uint16_t cnt;

/* Load all sensor values */
err = R_CTSU_ReadData(buf, &cnt);
```

Special Notes:

なし

3.8　R_CTSU_ReadReg

現在のレジスタ値を取得する関数です。

Format

```
qe_err_t  R_CTSU_ReadReg(ctsu_reg_t reg,
                         uint8_t element,
                         uint16_t *p_value);
```

Parameters

reg

読み込み対象のレジスタの ID。

element

CTSUSSC、SO0、SO1 のみ。要求されたセンサ/センサ・ペア(相互容量時)に関連する要素。

p_value

レジスタの値を格納する変数へのポインタ。

Return Values

QE_SUCCESS	*/* レジスタの値を読み込みに成功しました */*
QE_ERR_NULL_PTR	*/* 引数のポインタが指定されていません */*
QE_ERR_INVALID_ARG	*/* レジスタの ID または要素のインデックスが不正です */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

通常この関数は、“reg”で指定された CTSU ハードウェアレジスタの内容を“p_value”で指定された変数に読み込みます。

レジスタ CTSUSSC、SO0 および SO1 の場合、各センサ/センサ・ペア向けに保存された設定値が読み込まれます (ハードウェアレジスタには最後のセンサ/センサ・ペアの値のみが格納されています)。引数“element”には読み込む設定値が格納されている要素を指定します。要素に関しては「1.2 自己容量モード」および「1.3 相互容量モード」を参照してください。

Reentrant

なし

Example: 共通レジスタを取得する場合

```
  qe_err_t err;
  uint16_t value;

  /* Read the CTSU Error Status Register into “value” (“element” unused) */
  err = R_CTSU_ReadReg(CTSU_REG_ERRS, 0, &value );
```

Example: センサ固有のレジスタを取得する場合

```
  qe_err_t err;
  uint16_t value;

  /* Read the CTSUSSC configured value associated with element 4 into “value”
*/
  err = R_CTSU_ReadReg(CTSU_REG_SSC, 4, &value );
```

Special Notes:

なし

3.9　R_CTSU_WriteReg

この関数は CTSUSSC、SO0、SO1 を除き、引数の値を指定されたハードウェアレジスタに書き込みます。CTSUSSC、SO0、SO1 の場合、指定された引数（element）に対応する各センサ/センサ・ペア向けに保存された設定値に引数の値を書き込みます。

Format

```
qe_err_t  R_CTSU_WriteReg(ctsu_reg_t reg,
                          uint8_t element,
                          uint16_t value);
```

Parameters

reg

書き込み対象のレジスタの ID。

element

CTSUSSC、SO0、SO1 のみ。要求されたセンサ/センサ・ペア(相互容量時)に関連する要素。

value

レジスタに書き込む値。

Return Values

QE_SUCCESS	*/* レジスタの書き込みに成功しました */*
QE_ERR_INVALID_ARG	*/*レジスタの ID または要素のインデックスが不正です */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

通常この関数は、“reg”で指定された CTSU ハードウェアレジスタに“value”で指定された値に書き込みます。

レジスタ CTSUSSC、SO0 および SO1 の場合、“value”はドライバによって“element”(構成に関しては「1.2 自己容量モード」および「1.3 相互容量モード」を参照してください) で指定されたセンサ/センサ・ペア向けに内部的に保存されます。この値はのちにセンサ/センサ・ペアがスキャンされる直前にハードウェアレジスタに書き込まれます。

Reentrant

なし

Example: 共通レジスタに書き込む場合

```
qe_err_t err;
uint16_t value = 0x04;

/* Write “value” to CTSU_REG_CR0 (“element” unused) */
err = R_CTSU_WriteReg(CTSU_REG_CR0, 0, value );
```

Example: センサ固有のレジスタに書き込む場合

```
qe_err_t err;
uint16_t value = 0x1D21;

/* Update CTSUSO0 value associated with element 4 */
err = R_CTSU_WriteReg(CTSU_REG_SO0, 4, value );
```

Special Notes:

なし

3.10　R_CTSU_Control

ドライバに特殊操作コマンドを実行する関数です。

Format

```
qe_err_t  R_CTSU_Control(ctsu_cmd_t cmd,
                         uint8_t method,
                         uint16_t *p_arg);
```

Parameters

cmd

実行するコマンド。

method

コマンドを実行する対象のメソッド。

p_arg

コマンドの引数へのポインタまたは NULL。

Return Values

QE_SUCCESS　　*/* コマンドの実行に成功しました */*

QE_ERR_NULL_PTR　　*/* 引数のポインタ (p_arg) が指定されていません */*

QE_ERR_INVALID_ARG　　*/* コマンドが不正または未知のメソッドです */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

この関数は CTSU ドライバに特殊な操作を実行するために使用されます。“cmd”コマンドは以下のとおりです。

CTSU_CMD_SET_METHOD:

現在使用しているメソッド/スキャン構成を変更するときに使用します。“method”と同等のものは“qe_common.h”で定義され“QE_METHOD_xxx”の形式で定義されています。また“p_arg”は使用しません。

CTSU_CMD_SET_CALLBACK:

外部トリガ有効時に、デフォルトとして QE for Capacitive Touch で生成され、各スキャン終了時にコールされる qetouch_timer_callback()をオーバーライドします。“method”は使用しません。“p_arg”は新しいコールバック関数の名前となります。

CTSU_CMD_GET_STATE:

ドライバ (スキャン中) の現在の状態を取得します。“method”は使用しません。“p_arg”には

“ctsu_state_t”型の変数へのポインタを指定します。

CTSU_CMD_GET_STATUS:

指定されたメソッドのエラー、プロセス完了フラグを取得します。“method”と同等のものは“qe_common.h”で定義され“QE_METHOD_xxx”の形式で定義されています。“p_arg”は“ctsu_status_t”型の変数へのポインタを指定します。

CTSU_CMD_GET_METHOD_MODE:

指定された“method”引数に対して、自己容量または相互容量モードを示します。“ p_arg”は、“ctsu_mode_t”型の変数へのポインタを指定します。このコマンドは、診断と安全コードとして使用します。

CTSU_CMD_GET_SCAN_INFO:

指定された“method”引数に対して、最後に完了したスキャンのエラーレジスタ値を取得します。“ p_arg”は、“scan_info_t”型の変数へのポインタを指定します。このコマンドは、診断と安全コードとして使用します。

Reentrant

なし

Example: CTSU_CMD_SET_METHOD

```
qe_err_t err;

/* Change method/scan configuration (methods defined in qe_common.h) */
err = R_CTSU_Control(CTSU_CMD_SET_METHOD, QE_METHOD_FULLPOWER, NULL );
```

Example: CTSU_CMD_SET_CALLBACK

```
qe_err_t err;

/* Change default callback function to ext_trig_scan_complete_cb() */
err = R_CTSU_Control(CTSU_CMD_SET_CALLBACK, 0, ext_trig_scan_complete_cb)
```

Example: CTSU_CMD_GET_STATE

```
qe_err_t    err;
ctsu_state_t current_state;

/* Get the current scan state of the driver */
err = R_CTSU_Control(CTSU_CMD_GET_STATE, 0, &current_state);
```

Example: CTSU_CMD_GET_STATUS

```
qe_err_t     err;
ctsu_status_t current_status;

/* Get status flags for method (methods defined in qe_common.h) */
err = R_CTSU_Control(CTSU_CMD_GET_STATUS, QE_METHOD_PANEL2, &current_status);
```

Example: CTSU_CMD_GET_METHOD_MODE

```
qe_err_t     err;
ctsu_mode_t   scan_mode;

/* Get scan capacitance-mode for method QE_METHOD_PANEL1 */
err = R_CTSU_Control(CTSU_CMD_GET_METHOD_MODE, QE_METHOD_PANEL1, &scan_mode);
```

Example: CTSU_CMD_GET_SCAN_INFO

```
qe_err_t     err;
scan_info_t   scan_flags;

/* Get scan error flags for method QE_METHOD_PANEL1 */
err = R_CTSU_Control(CTSU_CMD_GET_SCAN_INFO, QE_METHOD_PANEL1, &scan_flags);
```

Special Notes:

なし

3.11　R_CTSU_Close

本モジュールを終了する関数です。

Format

```
qe_err_t  R_CTSU_Close(void);
```

Parameters

なし

Return Values

QE_SUCCESS	*/* CTSU 周辺機能を正常に終了しました */*
QE_ERR_BUSY	*/* 別の CTSU 操作が実行中のため実行できません */*

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

この関数は CTSU に関連する割り込みとクロックを無効にして CTSU ドライバを終了します。

Reentrant

なし

Example:

```
qe_err_t err;

/* Shut down peripheral and close driver */
err = R_CTSU_Close();
```

Special Notes:

なし

3.12　R_CTSU_GetVersion

CTSU FIT モジュールのバージョン情報を返します。

Format

```
uint32_t R_CTSU_GetVersion(void);
```

Parameters

なし

Return Values

バージョン情報を返します。

Properties

r_ctsu_qe_if.h にプロトタイプ宣言されています。

Description

当関数はインストールされている QE CTSU ドライバのバージョン情報を返します。バージョン情報は上位 2 バイトにメジャーバージョン番号、下位 2 バイトにマイナーバージョン番号としてエンコードされます。例えばバージョンが 4.25 の場合、0x000040019 のように返します。

Reentrant

この関数は再入可能(リエントラント)です。

Example:

```
uint32_t cur_version;

/* Get version of installed CTSU API. */
cur_version = R_CTSU_GetVersion();

/* Check to make sure version is new enough for this application's use. */
if (MIN_VERSION > cur_version)
{
   /* This QE CTSU API version is not new enough and does not have XXX feature
      that is needed by this application. Alert user. */
   ...
}
```

Special Notes:

本関数は#pragma ディレクティブを使ったインライン関数となります。

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://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018/10/4	-	初版発行
1.10	2019/7/9	1	対象デバイスに RX23W を追加
		4～6	補正とオフセット調整の定義を追加
		12、19	API の戻り値を更新
		35～36	CTSU_CMD_GET_METHOD_MODE、 CTSU_CMD_GET_SCAN_INFO の関数を追加
		11、 13～17	セーフティモジュール用ドライバ（GCC / IAR サポートを含む）に#pragma section マクロと設定オプションを追加
		1、17	IEC 60730 準拠に関する内容を追加

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。お客様の機器・システムの設計において、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には、お客様の責任において行ってください。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、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、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。
5. 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。
6. 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。
7. 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。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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.4.0-1　2017.11)

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。