RENESAS　アプリケーションノート

RX ファミリ

QE CTSU モジュール Firmware Integration Technology

要旨

本アプリケーションノートは 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
・RX230 グループ
・RX231 グループ
・RX23W グループ
・RX671 グループ
・RX140 グループ
・RX260 グループ
・RX261 グループ

CTSU ペリフェラルは CTSU, CTSUa, CTSU2L, CTSU2SL, CTSU2SLa と、機能が違う 5 種類のバージョンがあります。各デバイスに搭載している CTSU ペリフェラルのバージョンは以下の通りです。

CTSU2SLa	: RX260, RX261
CTSU2SL	: RX140-256KB, RX140-128KB
CTSU2L	: RX140-64KB
CTSUa	: RX130, RX671
CTSU	: RX113, RX230, RX231, RX23W

CTSU と CTSUa は本書では機能の違いに対する説明の違いは無いため、どちらも CTSU と表記します。CTSU と CTSU2L、CTSU2SL、CTSU2SLa は機能に違いがあるため、説明の都合上、本書では CTSU と CTSU2L、CTSU2SL、CTSU2SLa は下記のように表記します。

・CTSU、CTSU2L、CTSU2SL、CTSU2SLa 共通の説明	→ CTSU
・CTSU のみの説明	→ CTSU1
・CTSU2L, CTSU2SL, CTSU2SLa 共通の説明	→ CTSU2L
・CTSU2SL と CTSU2SLa 共通の説明	→ CTSU2SL
・CTSU2SLa のみの説明	→ CTSU2SLa

本書を理解するために事前に静電容量タッチ導入ガイドおよび静電容量タッチソフトウェア概要を参照することを推奨します。

本モジュールを用いてアプリケーションを開発する際は QE for Capacitive Touch の利用を推奨します。

関連ドキュメント

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)
静電用容量センサマイコン 静電容量タッチ導入ガイド(R30AN0424)
静電用容量センサマイコン 静電容量タッチソフト概要(R30AN0470)

目次

1. 概要 ... 3
1.1 CTSU モジュールとは ... 3
1.2 CTSU モジュールの位置づけ ... 3
1.3 CTSU モジュールの使い方 ... 4
1.4 計測モード ... 6
1.4.1 自己容量計測モード ... 6
1.4.2 相互容量計測モード ... 6
1.4.3 電流計測モード(CTSU2L) ... 7
1.4.4 自動判定モード(CTSU2SL) ... 8
1.4.5 診断モード ... 12
1.5 機能 ... 15
1.5.1 ランダムパルス周波数計測(CTSU1) ... 15
1.5.2 マルチクロック計測(CTSU2L) ... 15
1.5.3 センサ CCO 補正 ... 17
1.5.4 イニシャルオフセットチューニング ... 17
1.5.5 移動平均 ... 18
1.5.6 コールバック関数 ... 18
1.5.7 シールド機能(CTSU2L) ... 18
1.5.8 複数電極接続（MEC : Multiple Electrode Connection）機能 (CTSU2SL) ... 19
1.5.9 自動 CCO 補正機能(CTSU2SL) ... 20
1.5.10 自動周波数補正機能(CTSU2SLa) ... 20
1.6 他の FIT モジュールとの連携 ... 21
1.6.1 計測開始トリガ ... 21
1.6.2 低消費電力動作 ... 21
1.6.3 データ転送 ... 21
1.6.4 診断機能 ... 22
1.7 API 概要 ... 23

2. API 情報 ... 24
2.1 ハードウェアの要求 ... 24
2.2 ソフトウェアの要求 ... 24
2.3 サポートされているツールチェーン ... 24
2.4 制限事項 ... 24
2.5 ヘッダファイル ... 24
2.6 整数型 ... 24
2.7 コンパイル時の設定 ... 25
2.8 コードサイズ ... 27
2.9 引数 ... 28
2.10 戻り値 ... 33
2.11 コールバック関数 ... 34
2.12 API の処理時間 ... 35
2.13 FIT モジュールの追加方法 ... 36
2.13.1 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 ... 36
2.13.2 Smart Configurator を使用しない場合のモジュールオプションの設定 ... 36

3. API 関数 ... 37
3.1 R_CTSU_Open ... 37
3.2 R_CTSU_ScanStart ... 38
3.3 R_CTSU_DataGet ... 39
3.4 R_CTSU_CallbackSet ... 40
3.5 R_CTSU_Close ... 41
3.6 R_CTSU_Diagnosis ... 42
3.7 R_CTSU_ScanStop ... 44
3.8 R_CTSU_SpecificDataGet ... 45
3.9 R_CTSU_DataInsert ... 46
3.10 R_CTSU_OffsetTuning ... 47
3.11 R_CTSU_AutoJudgementDataGet ... 48

1. 概要

1.1 CTSU モジュールとは

CTSU モジュールとは、CTSU ペリフェラルを制御するドライバです。CTSU モジュールは、コンフィグレーション設定により、TS 端子に接続された電極に応じた計測をして、静電容量の計測値を取得します。

主に以下の機能があります。

- CTSU ペリフェラルを制御して取得した静電容量の計測値に対して、複数の補正処理をして補正計測値を算出します。また CTSU2L ペリフェラルは静電容量の代わりに電流を計測することも可能です。
- CTSU2SL ペリフェラルからボタンのタッチ判定結果を取得します。
- CTSU ペリフェラルが正常に動作しているか内部回路を診断します。

1.2 CTSU モジュールの位置づけ

タッチシステムのソフトウェア構成を図 1.1 に示します。

通常、CTSU モジュールは CTSU ペリフェラルを制御し、補正計測値を返します。そのため、ボタンの ON/OFF、スライダ、ホイールのタッチ位置をアプリケーションで利用したい際は TOUCH モジュールと組み合わせての利用を推奨しています。
その際、TOUCH モジュールは CTSU モジュールを呼び出し、取得した補正計測値をボタンの ON/OFF やタッチ位置に変換します。具体的な動作については TOUCH モジュールのアプリケーションノートを参照してください。

CTSU モジュールから取得した補正計測値をもとに独自にタッチ判定を行う場合は、以降の章を参照してください。
また、CTSU2SL ペリフェラルが搭載している HW 判定と組み合わせることで（自動判定モード）、CTSU モジュールを使ってタッチ ON/OFF 結果を取得することが可能です。
CTSU モジュールを使って CTSU ペリフェラルの内部回路を診断し、その診断結果を取得可能です。

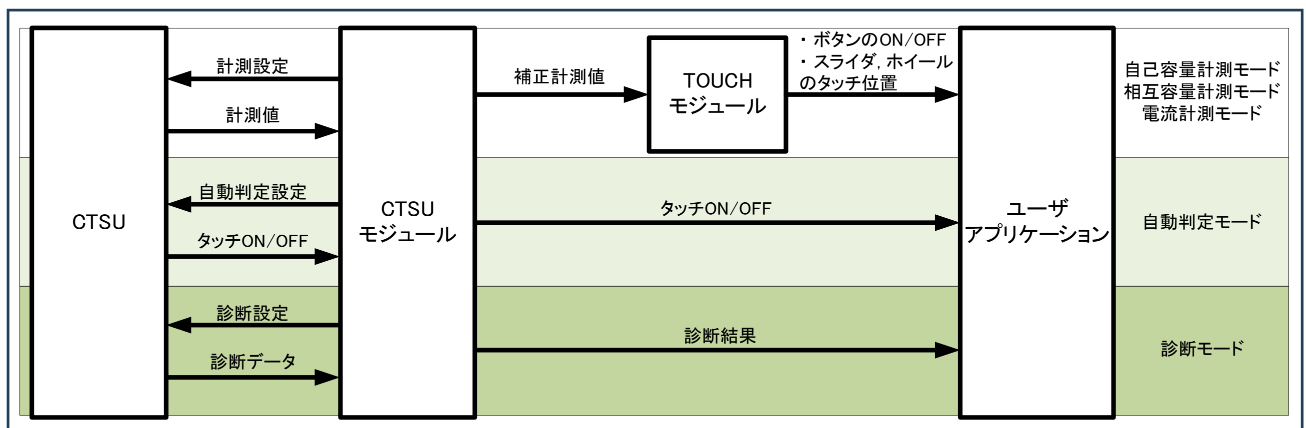

図 1.1 CTSU モジュールとユーザアプリケーションの関係図

1.3 CTSU モジュールの使い方

図 1.2 は、CTSU モジュールの使い方を示した、ソフトウェアシステムの全体図です。

CTSU モジュールは、各種 API コール群と、それに応じた構造体情報を合わせて、計測モードが定義されています。自己容量/相互容量/電流計測モードで呼び出される API コール群を基本動作モデルとしています。自動判定、診断モードの動作モデルについては 1.4 をご参照ください。

基本動作モデルでは、まず、R_CTSU_Open()を呼び出し、CTSU ペリフェラルの初期化と、コンフィグレーション設定をリードしてコントロール構造体に設定情報を作成します。その後、R_CTSU_ScanStart()（計測開始処理）と、R_CTSU_DataGet()（計測結果データ取得処理）を周期的に呼び出すことで、定期的な計測処理を実行します。

計測中は、INTCTSUWR(レジスタ設定要求割り込み)と INTCTSURD(計測結果読み出し要求割り込み)、INTCTSUFN(計測終了割り込み)が発生します。これらの割り込みは CTSU ペリフェラルから発生し、本モジュールが処理します。INTCTSUFN の処理完了後、コールバック関数を通じてアプリケーションへ計測完了を通知します。通知を受けた後に計測結果取得処理を行ってください。

なお、INTCTSUWR と INTCTSURD の処理には DTC を使用することも可能です。DTC を使用する場合の詳細は 1.6.3 をご参照ください。

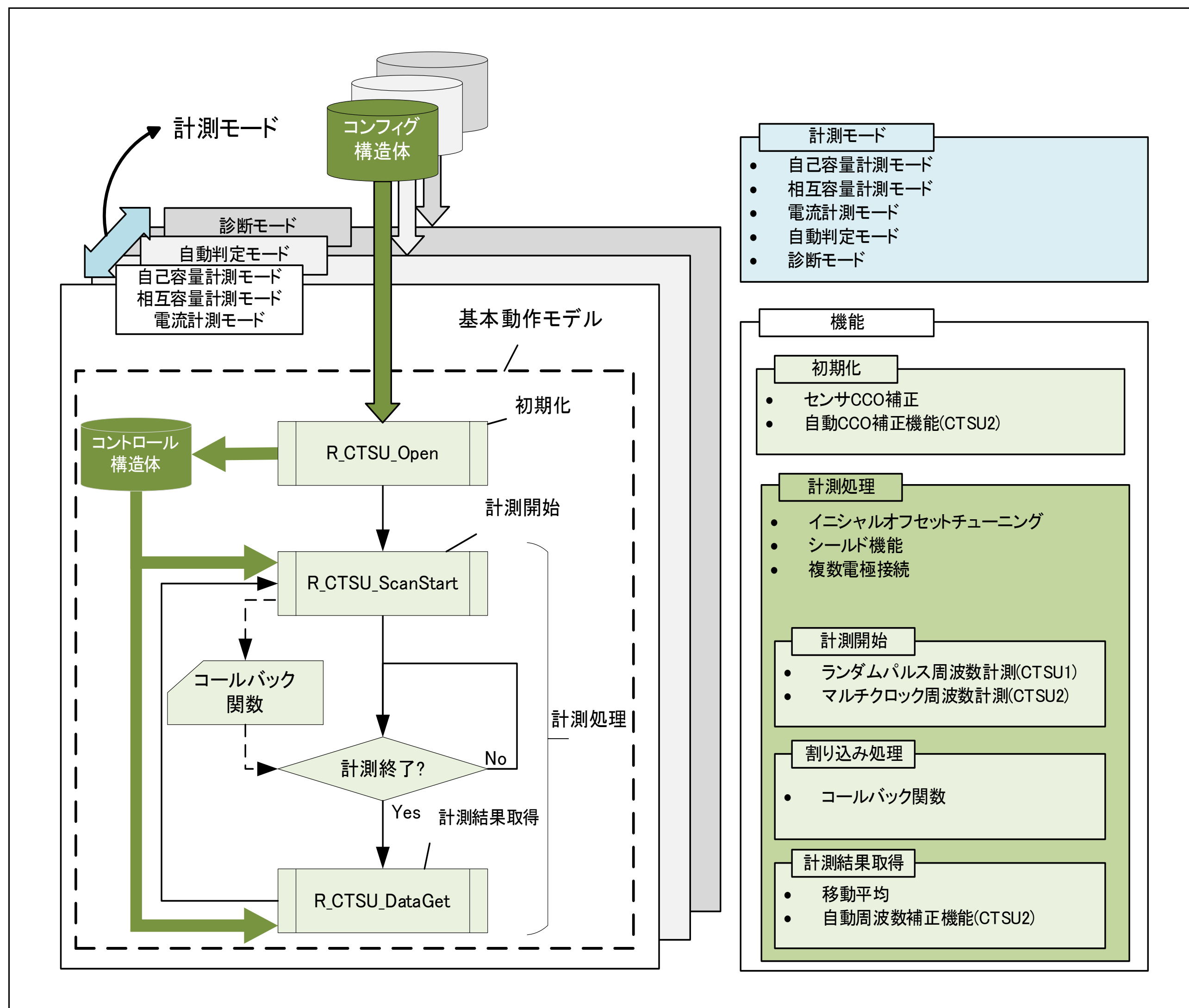

図 1.2 ソフトウェアシステムの全体図

本モジュールにおけるデータ管理は、コンフィグ構造体とコントロール構造体の二つの構造体を中心に構成されています。

コンフィグ構造体では計測対象および計測条件を設定します。静電容量を計測する際の計測対象をエレメントと呼び、自己容量方式で計測を行う場合は TS 端子が、相互容量計測方式では送信 TS 端子と、受信 TS 端子のマトリクスになります。計測条件には、計測モード、計測時間、計測レンジ、トリガ設定など、CTSU ペリフェラルによる計測実行に必要な各種パラメータが含まれます。

これらコンフィグ構造体に設定された計測情報は、R_CTSU_Open()の実行時にコントロール構造体へコピーされ、同時に計測結果を格納するためのバッファが割り当てられるとともに、計測状態を管理するための内部フラグやステート情報が初期化されます。初期化後の計測処理ではコントロール構造体のみを参照することで、計測設定、計測状態、および計測結果を一元的に管理することが可能です。

これらの構造体を複数用意することで、異なる計測モードを独立して並行に動作させることや、同一の計測モードであっても計測対象および計測条件が異なる複数の計測処理を独立して並行に動作させることが可能です。

これらの計測設定を簡単に設定できるように、ルネサスでは QE for Capacitive Touch という開発支援ツールを提供しています。本ドライバを用いたアプリケーションを作る際も、QE for Capacitive Touch を利用し作成したタッチインタフェース構成を利用することを推奨します。

1.4 計測モード

計測対象や、計測の方式、またその条件によって定まる計測処理を総じて計測モードと言います。CTSU ペリフェラルの機能に応じて、利用可能な計測モードが異なります。

CTSU ペリフェラルは自己容量計測モード及び相互容量計測モードと診断モードをサポートしています。CTSU2L ペリフェラルはこれらに加えて電流計測モードを、また CTSU2SL ペリフェラルでは自動判定モードをサポートしています。各計測モードの詳細は以下をご参照ください。

1.4.1 自己容量計測モード

自己容量計測モードでは、各 TS 端子の静電容量を計測します。

CTSU ペリフェラルは TS 端子番号に従って昇順に計測してデータを格納します。例えば、アプリケーションで TS5、TS8、TS2、TS3、TS6 の順番で使用したい場合でも、TS2、TS3、TS5、TS6、TS8 の順に計測してデータを格納するので、バッファのインデックスは[2]、[4]、[0]、[1]、[3] を参照してください。

[CTSU1]

1 つの TS 端子の計測時間はデフォルト設定の場合、安定待ち時間＋約 526us です。

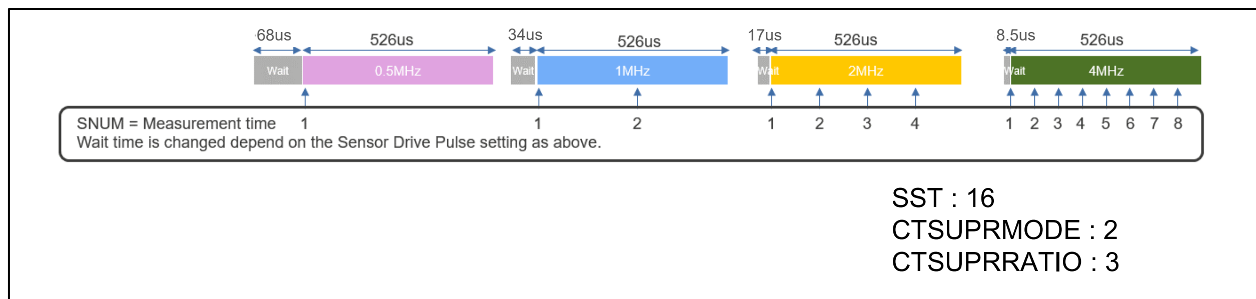

図 1.3 自己容量計測時間(CTSU1)

[CTSU2]

1 つの TS 端子の計測時間はデフォルト設定の場合、約 576us です。

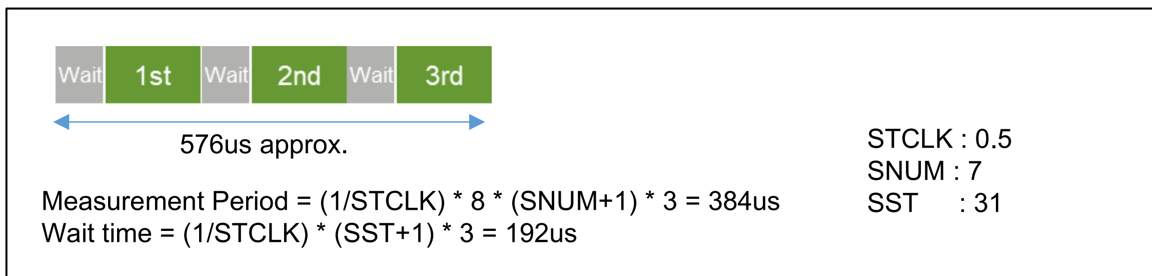

図 1.4 自己容量計測時間(CTSU2L)

1.4.2 相互容量計測モード

相互容量計測モードでは、受信 TS 端子（Rx）と送信 TS 端子（Tx）の間に発生する静電容量を計測します。そのため、2 つ以上の TS 端子を必要とします。

CTSU2L ペリフェラルは、設定された Rx と Tx の全ての組み合わせを計測します。例えば、TS10、TS3 が Rx、TS2、TS7、TS4 が Tx の場合、以下の組み合わせの順に計測してデータを格納します。

TS3-TS2, TS3-TS4, TS3-TS7, TS10-TS2, TS10-TS4, TS10-TS7

電極間に発生している相互容量を計測するため、CTSU2L ペリフェラルは同一電極に対して 2 回の計測処理をします。1 回目（Primary）と 2 回目（Secondary）でパルス出力とスイッチドキャパシタの位相関係を反転させて計測し、2 回目補正計測値と 1 回目補正計測値の差分を求めることで相互容量値を得る事ができます。CTSU モジュールは差分計算をせずに 2 回の補正計測値を出力します。

[CTSU1]

1 つの TS 端子の計測時間はデフォルト設定の場合、安定待ち時間＋約 526us の 2 倍です。

[CTSU2]

1 つの電極の計測時間はデフォルト設定の場合、約 1152us です。

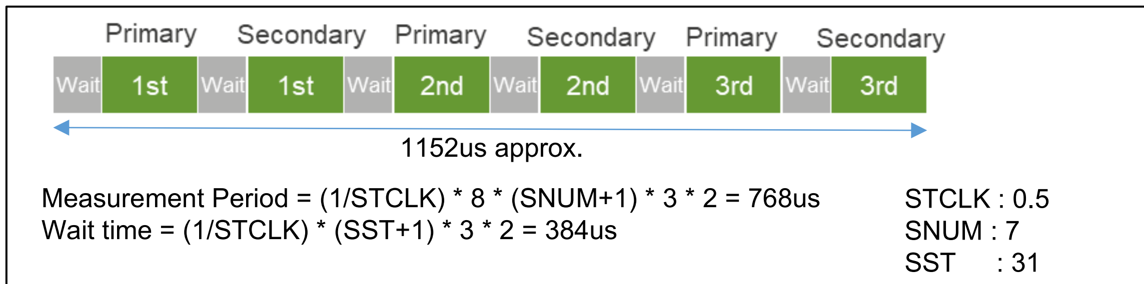

図 1.5 相互容量計測時間(CTSU2L)

1.4.3 電流計測モード(CTSU2L)

CTSU2L の電流計測モードでは、TS 端子に入力される微小な電流を計測します。

計測とデータ格納の順は自己容量と同様です。

スイッチドキャパシタ動作ではないので計測回数は 1 回になるため、1 つの TS 端子の計測時間はデフォルト設定の場合、約 256us です。電流計測モードでは他のモードより長い安定待ち時間が必要となるため、SST を 63 に設定しています。

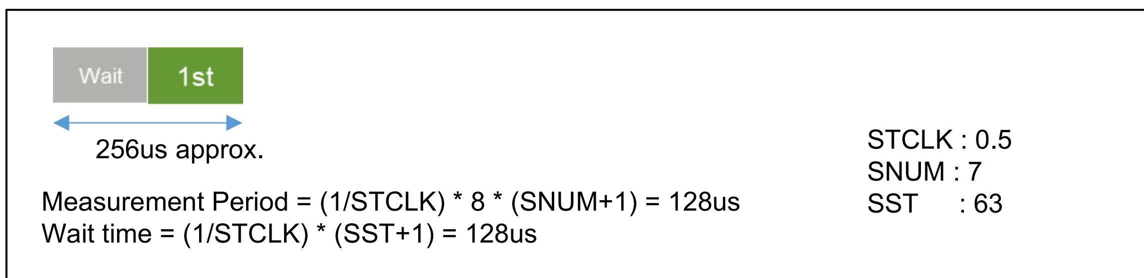

図 1.6 電流計測時間

1.4.4 自動判定モード(CTSU2SL)

自動判定モードでは、CTSU2SL ペリフェラルを制御して、タッチ判定結果を取得することができます。タッチ判定処理は CTSU2SL ペリフェラルによって実施されるため、CPU の処理を使わずにタッチ判定が可能になります。ただし、計測時に発生する割り込みに対して DTC 転送が必要であるため、DTC モジュールが必須となります。DTC については 1.6.3 章を参照ください。

自動判定モードの動作フローを下記に示します。まず、R_CTSU_ScanStart()と R_CTSU_OffsetTuning()を用いて、イニシャルオフセットチューニング処理を行います。イニシャルオフセットチューニングについては、1.5.4 を参照してください。イニシャルオフセットチューニングの完了後、R_CTSU_ScanStart()と R_CTSU_AutoJudgementDataGet()を用いて自動判定の計測処理を行います。このとき、1 回目の計測と、2 回目以降の計測で処理が異なるため注意が必要です。

1 回目の計測は、タッチ判定のベースラインの設定をします。このときの補正計測値がベースラインとなり、タッチ判定結果は OFF となります。

2 回目以降の計測は、コントロール構造体の計測設定に従って計測して、タッチ判定結果を取得します。

自動判定モードを使用すると、外部トリガによる計測と組み合わせて、低消費電力でタッチ判定を行うアプリケーションを作成可能です。その際の詳細と注意事項については(1)を参照してください。

自動判定の詳細について確認が必要な場合は(2)を参照してください。

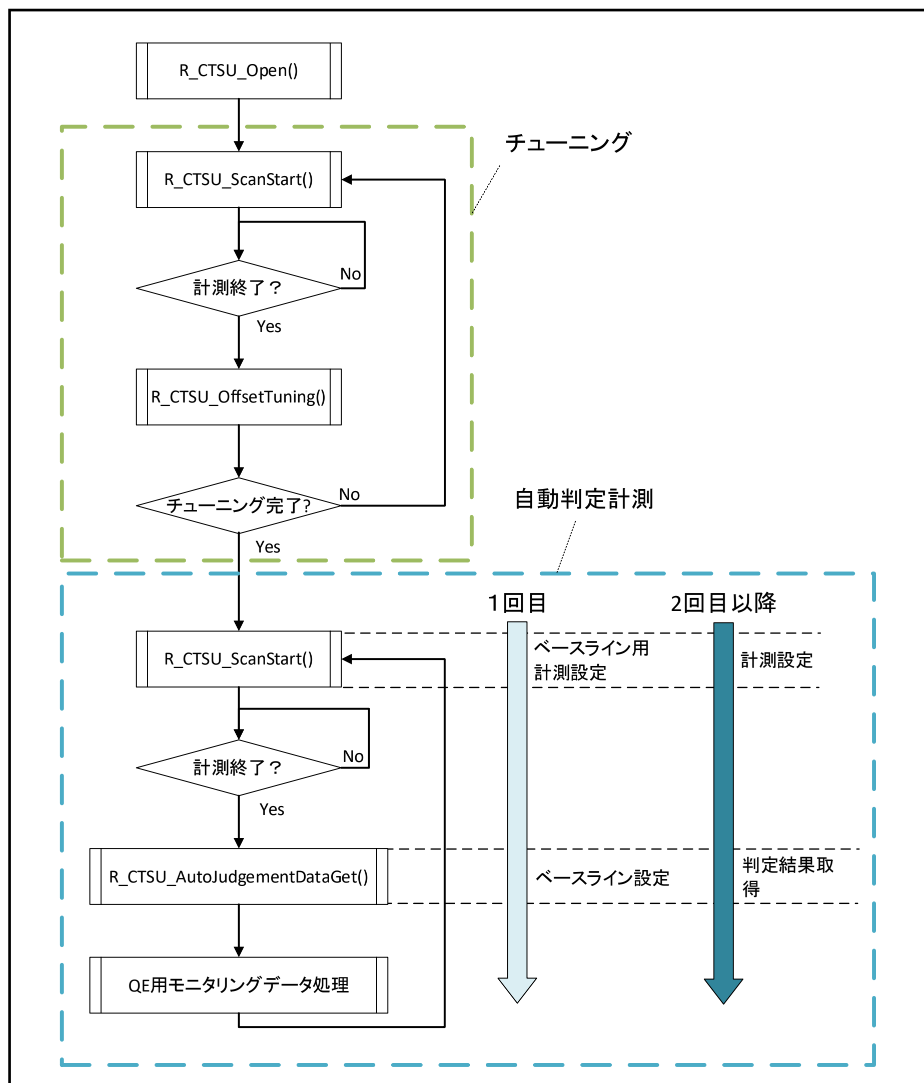

図 1.7 自動判定モードの基本動作モデル

(1)　低消費電力でタッチ判定を行う際の注意点

CPU のスヌーズモードおよびソフトウェアスタンバイモードと組み合わせた低消費電力タッチアプリケーションでは、CPU を低消費電力モードの状態でタッチ判定を行い、タッチ判定結果が ON であれば CPU を起動するという動作が可能です。

ただし、ベースライン設定までは CPU 処理が必要になります。そのため、1 回目の R_CTSU_ScanStart()と R_CTSU_AutoJudgementDataGet()を実行した後に、低消費電力動作モードに移行してください。

具体的な動作例については、アプリケーションノートの「スマートウェイクアップソリューション」を参照してください。

(2)　機能と設定方法

自動判定モード利用時には、通常計測の設定に加え、マクロ定義の変更と、コンフィグ構造体に自動判定モード用の設定をする必要があります。また、その設定は自動判定の計測方式である JMM と VMM によって異なります。

まず、自動判定機能を有効にするには CTSU_CFG_AUTO_JUDGE_ENABLE = 1 を設定してください。またその際に一緒に CTSU_CFG_AUTO_CORRECTION_ENABLE = 1 を設定して自動 CCO 補正機能も有効にしてください。VMM は CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE = 1 を設定して自動周波数補正機能も有効にしてください。
次に通常計測に加えて設定が必要なコンフィグ構造体の設定例を JMM として以下に示します。
VMM を利用する際は jc を 0、majirimd を 1 に設定してください。

```
.tlot = 2,      // タッチ OFF 判定連続回数 = 3 回
.thot = 2,      // タッチ判定連続回数 = 3 回
.jc = 1,        // 2 つ以上でタッチ判定
.ajmmat = 2,    // 移動平均 2^2 回
.ajbmat = 7,    // ベースライン平均回数 2^(7+1) 回
.majirimd = 0,  // JMM
.mtucfen = 1,   // 相互容量演算許可
.ajfen = 1,     // 自動判定有効
```

次に、(a) ~ (e)に自動判定機能の説明とその設定方法を記載しています。

JMM のときは、マルチクロック計測のそれぞれに(a) ~ (e)の設定を行います。

(a) 計測モード

ctsu_auto_button_cfg_t の mtucfen で自己容量か相互容量の選択をします。自己容量は 0 を設定してください。相互容量は 1 を設定してください。

(b) ベースライン

タッチ OFF 状態の補正計測値からベースラインを設定します。R_CTSU_OffsetTuning()でイニシャルオフセットチューニングを完了後、最初に R_CTSU_ScanStart()をコールしたときにベースラインを初期設定します（BLINI ビットをセット）。その後に、R_CTSU_AutoJudgementDataGet()をコールしたときにベースライン初期化を解除（BLINI ビットをクリア）して、ベースライン更新処理を開始します。

ベースラインは設定した計測回数毎に更新して外部環境変化に追従します。設定した計測回数で”タッチ OFF”状態が継続した場合、その平均値にベースラインを更新します。期間中に”タッチ ON”判定するとそれまでの平均回数と平均値をクリアします。

ctsu_cfg_t の ajbmat で計測回数（ベースライン更新間隔）を設定できます。タッチインタフェース構成内のボタン共通です。周囲環境変化への追従性を調整できます。

(c) タッチしきい値

ベースラインから任意のオフセットをつけたタッチしきい値を利用してタッチ判定をします。

タッチしきい値はヒステリシスを付与して設定します。"タッチ ON"から"タッチ OFF"の遷移にヒステリシスを持たせることでチャタリングを防止します。大きくするほどチャタリング対策に効果がありますが、タッチ OFF 判定をしづらくなります。

ctsu_auto_button_cfg_t の threshold と hysteresis でボタン毎のタッチしきい値とヒステリシスを設定できます。本モジュールはこれらから上側タッチしきい値と下側タッチしきい値を計算して、CTSUAJTHR レジスタに設定します。

以下に自己容量の判定を示します。自己容量ボタンはタッチ時に電極容量が増加するため、上側タッチしきい値を超えた際に"タッチ ON"判定します。

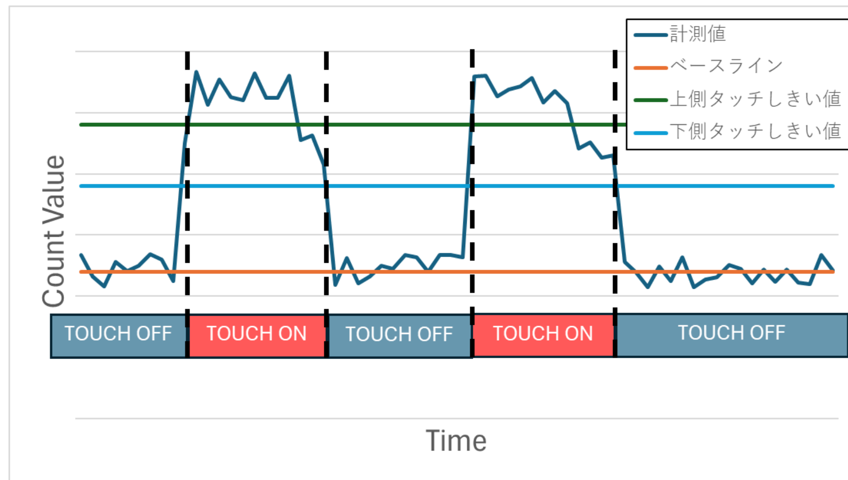

図 1.8 自己容量の判定

以下に相互容量の判定を示します。相互容量ボタンはタッチ時に電極間容量が減少するため、下側タッチしきい値を超えた際に"タッチ ON"判定されます。

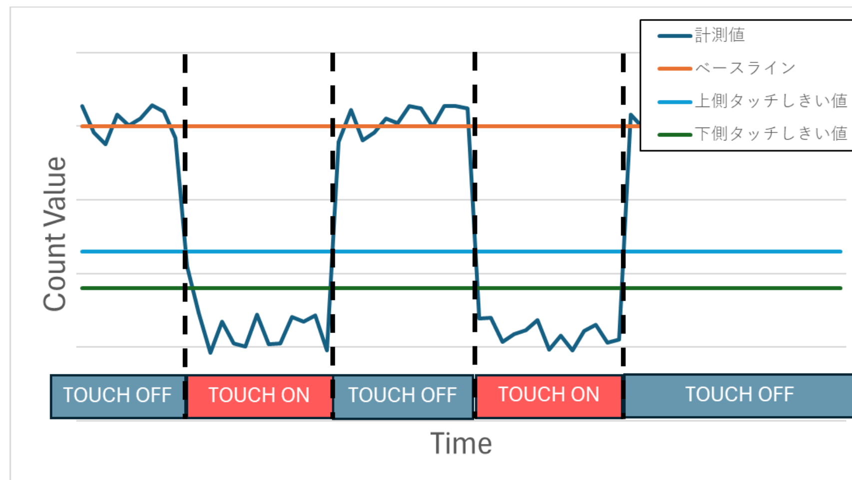

図 1.9 相互容量の判定

(d) 非タッチ判定連続回数 / タッチ判定連続回数

"タッチ ON"または"タッチ OFF"状態が一定回数継続した時に"タッチ ON"または"タッチ OFF"を判定するためのフィルタ機能です。

ctsu_cfg_t の tlot および thot で回数を設定できます。タッチインタフェース構成内のボタン共通です。連続回数を増やすほどチャタリング対策には効果がありますが、反応速度が低下します。

(e) 移動平均

自動判定機能では ctsu_cfg_t の ajmmat で移動平均回数を設定できます。タッチインタフェース構成内のボタン共通です。

これまで説明したボタン判定の動作を以下に示します。

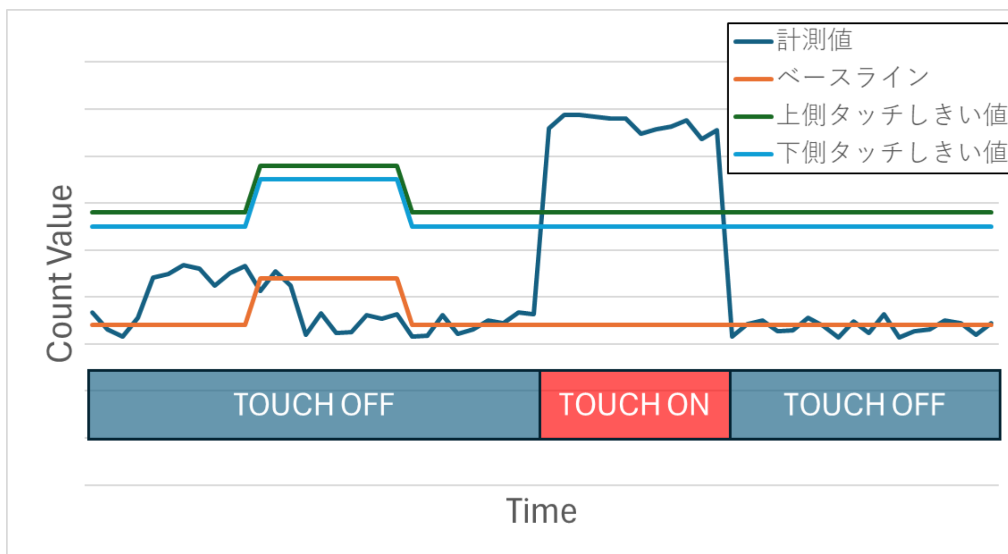

図 1.10 ボタンの判定

1.4.5 診断モード

CTSU ペリフェラルは自身の内部回路を診断する機能を持っています。診断モードは、内部回路が正常に動作することを診断するための API を提供します。

本モードでは CTSU 内部回路を診断し、異常発生時にエラーとして結果を取得できます。

診断モードは CTSU1 と CTSU2 で診断内容が違い、それにより動作モデル(API コール順)も変化します。

診断モードを有効にする共通設定として CTSU_CFG_DIAG_SUPPORT_ENABLE を 1 にしてください。

(1) CTSU1

(a) 診断項目

CTSU1 の診断項目を表 1.1 に示します。

表 1.1 診断項目一覧 (CTSU1)

順番	診断内容
1	過電圧検出診断
2	CCO 上限診断
3	CCO 下限診断
4	SSCG 発振器診断
5	センサオフセット診断

(b) 想定アプリケーション

CTSU1 の場合、診断用のインスタンスに対して、基本モデルで計測と計測データ処理をして、計測データ処理の戻り値が正常終了の場合に、診断 API をコールして診断処理をします。 1.11 に示します。診断モードの計測後に他の計測モードを利用する場合は、約 1ms の待ち時間後にタッチ計測を開始してください。

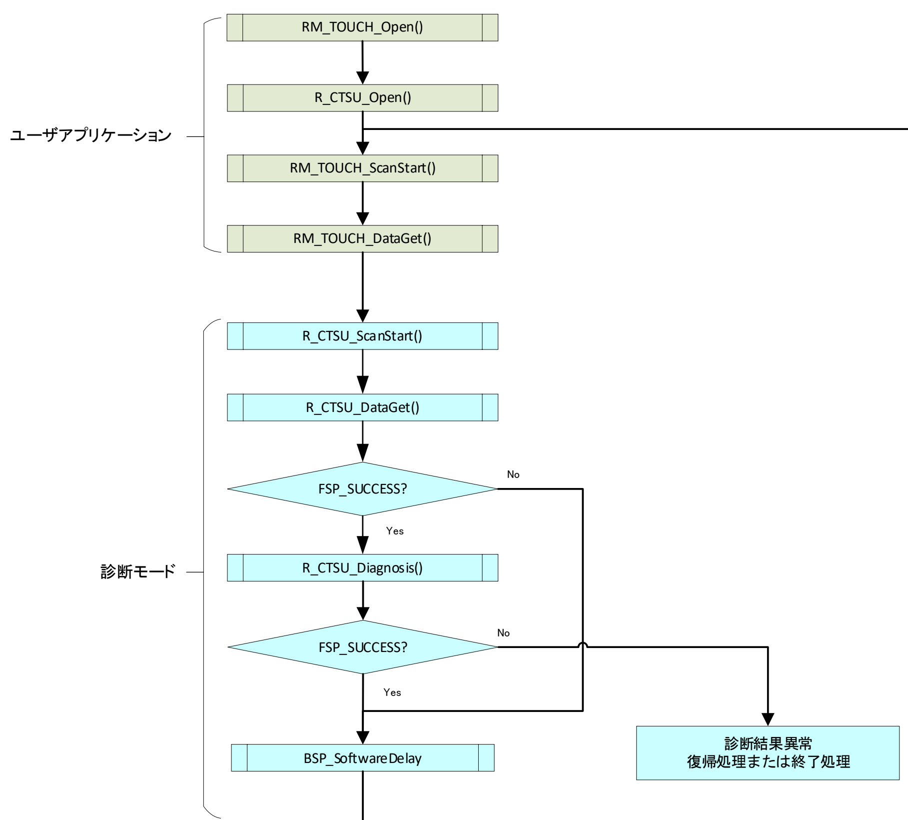

図 1.11 診断モード利用時の想定フロー (CTSU1)

(c) コンフィグレーション設定

本モードのコンフィグ構造体について、メンバである md を CTSU_MODE_DIAGNOSIS_SCAN に設定してください。

また、任意の TS 端子にコンデンサを接続してください(27pF 推奨)。接続した TS 端子番号をマクロ定義 CTSU_CFG_DIAG_DAC_TS に割り当ててください。

(2) CTSU2L

(a) 診断項目

CTSU2L の診断項目を表 1.2 に示します。

表 1.2 診断項目一覧 (CTSU2L)

順番	診断内容	ワンタイム/リピート(※1)
1	出力電圧診断	リピート
2	過電圧検出診断	ワンタイム
3	過電流検出診断	ワンタイム
4	内部抵抗診断	リピート
5	センサオフセット診断	ワンタイム
6	SENSCLK 周波数診断	リピート
7	SUCLK 周波数診断	リピート
8	SUCLK クロックリカバリ診断	リピート

※1 ワンタイムはリセット後に 1 回のみ診断すること想定しています。リピートは、周期的に計測することを想定しています。

(b) 想定アプリケーション

CTSU2L の診断モードでは、システム起動時またはリセット時に、その他計測モードを開始する前に本モードによる診断を 1 度行うことを想定しています。その他計測モード使用中であっても、本モードを利用することで、内部回路の定期的な診断を行うことが可能です。

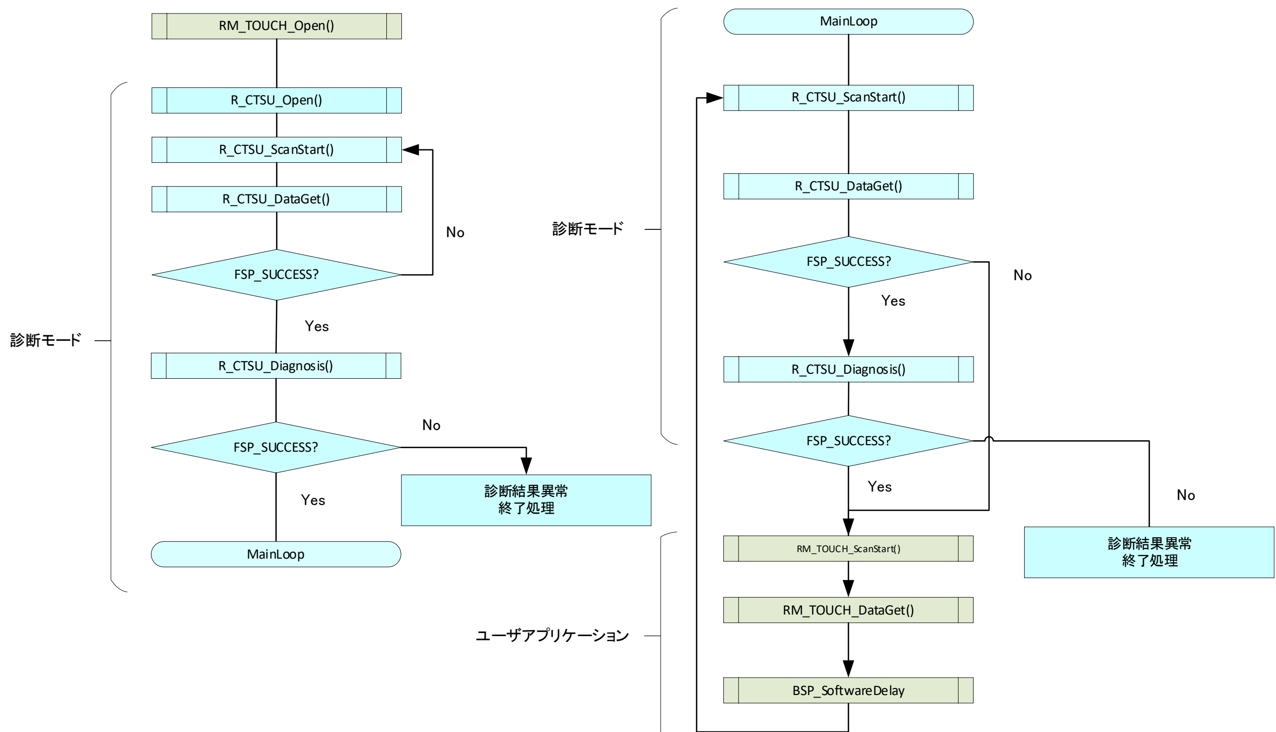

図 1.12 診断モードのフローチャート (CTSU2L)

(c)　コンフィグレーション設定

診断モードのコンフィグ構造体のメンバに対して、必要な設定を表 1.3 に示します。

これらの設定は、QE for Capacitive Touch を利用する場合は自動的に生成されます。QE for Capacitive Touch を使用しない場合は 1.4.5(2)(d)のコンフィグレーションの注意点を参照してください。

表 1.3 診断モード(CTSU2L)のコンフィグレーション設定

メンバ名	設定
ctsu_element_cfg_t	
so	0x00
snum	0x07
sdpa	過電流検知用の計測設定
ctsu_cfg_t	
md	CTSU_MODE_DIAGNOSIS_SCAN
ctsuchacN (N=0～4)	過電流検知用の計測端子

本モードでは AD 計測をするために r_s12ad_rx モジュールを追加する必要があります。詳細は、1.6.4 をご参照ください。

(d)　コンフィグレーション設定の注意点

診断モードでは過電流検知動作を確認するため、通常計測の対象である TS 端子の中から 1 つを選択し、コンフィグ構造体の ctsuchacN にその TS 端子を設定する必要があります。
設定後、TS 端子に接続された CTSU2SL ペリフェラル内部で過電流を検知できる電流量を生成するため、以下の手順に従ってセンサドライブパルス周波数を調整します。

1.　TS 端子に流れる電流量の計算

選択した TS 端子に任意のセンサドライブパルス周波数fと TSCAP 電圧Vで計測を行い、その端子の寄生容量C を求めます。その際、CTSU2SL ペリフェラル内部に流れる電流量 I を以下の式で計算します。

$$I = fCV$$

2.　診断用の SDPA の設定

過電流を検出できる条件になるようにセンサドライブパルスを設定します。

寄生容量計測時の SDPA を $SDPA_0$、1 で算出した電流量を I_0 として、診断用の SDPA($SDPA_1$)を求めてください。

$$SDPA_1 = \frac{(SDPA_0 + 1)I_0}{100} - 1$$

この際、$SDPA_1$ によって設定されるセンサドライブパルス周波数が対象の TS 端子を充放電するのに十分な時間を確保できることを確認してください。

1.5 機能

CTSU モジュールがサポートする機能は以下のとおりです。

1.5.1 ランダムパルス周波数計測(CTSU1)

計測に使用するドライブパルスは、設定したベースクロックに対して位相シフトと周波数拡散をしたパルスです。

ドライブパルス周波数は、基本的には QE for Capacitive Touch によって調整された値を使用します。

このモジュールは初期化時に固定で下記設定をします。

CTSUSOFF = 0, CTSUSSMOD = 0, CTSUSSCNT = 3

ベースクロックの設定方法は下記となります。

CTSU に入力される PCLK の周波数、CTSU 動作クロック選択ビット(CTSUCLK)および CTSU ベースクロック設定ビット(CTSUSDPA)で決定されます。例として、PCLK が 32MHz、CTSUCLK で 1/2 周期を選択、CTSUSDPA で 16 分周を選択するとベースクロックは 0.5MHz となります。なお、CTSUSDPA は TS 端子毎に変更する事が可能です。

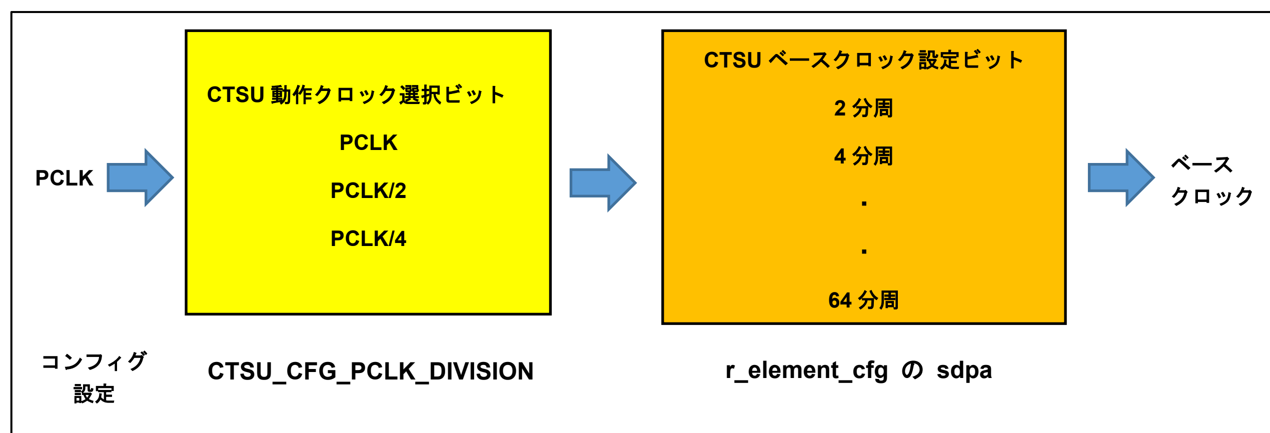

図 1.13 ベースクロック設定(CTSU1)

1.5.2 マルチクロック計測(CTSU2L)

CTSU2L ペリフェラルは同期ノイズを回避するために最大 4 種類のドライブパルス周波数で計測できます。本モジュールはデフォルトで 3 種類の周波数で計測し、得られた 3 つの計測結果を多数決判定します。

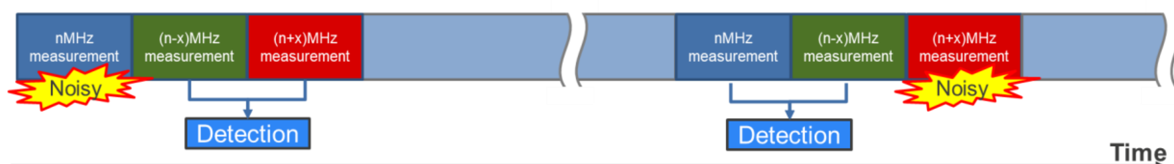

図 1.14 マルチクロック計測

3 つの計測結果に対するタッチ判定方法には JMM（判定多数決モード）と VMM（計測値多数決モード）の 2 種類があります。JMM は自己容量ボタンと相互容量のボタンのみ対応しています。

Touch モジュールと合わせた JMM と VMM の処理フローを以下に示します。

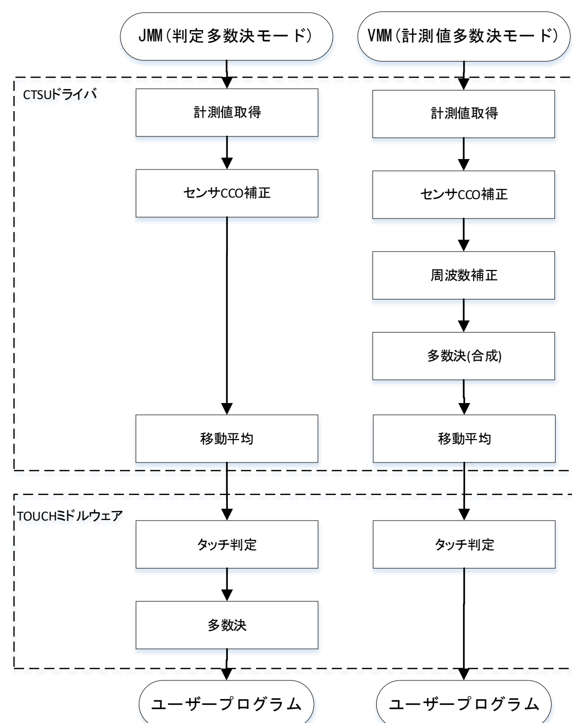

図 1.15 JMM（判定多数決モード）と VMM（計測値多数決モード）の処理フロー

JMM は 3 つの計測値を CCO 補正し、各々タッチ判定した結果を多数決して最終タッチ判定をします。

VMM は 3 つの計測値を CCO 補正し、第 1 周波数での計測値に補正する周波数補正を行い、値の近い 2 つの計測値を加算した結果を改めて計測時間が 2 倍の計測値とします。この補正計測値でタッチ判定をします。

VMM の計算例

周波数補正した補正計測値 1、補正計測値 2、補正計測値 3 から、各ペアの差分値 1～3 を計算して、その差分値の絶対値で比較して小さいペアを選択します。補正計測値のバラつき防止のため補正計測値 1 と補正計測値 2 の組み合わせを選びやすくしています。補正計測値 3 を比較する際は差分値 2 を 2 倍、差分値 3 を 1.5 倍して比較します。

計測値補正 1	補正計測値 2	補正計測値 3	差分値 1	差分値 2	差分値 3	判定結果	加算値
7734	7734	7663	0	71	71	補正計測値 1+2	15468
7689	7739	7666	50	23	73	補正計測値 1+3	15355
7734	7679	7664	55	70	15	補正計測値 2+3	15343
7721	7719	7694	2	27	25	補正計測値 1+2	15440
7716	7747	7693	31	23	54	補正計測値 1+2	15463

タッチインタフェース構成ごとに JMM か VMM かを設定可能です。ctsu_cfg_t のメンバ「majority_mode」を 1 にすると JMM、0 にすると VMM で動作します。

R_CTSU_DataGet()は移動平均を実施した後のデータを取得できます。その前の各処理をしたデータを取得するには R_CTSU_SpecificDataGet()を使用してください。これらのデータを使用して独自のノイズフィルタ処理をしたデータを Touch モジュールで判定することも可能です。詳細は 3.8 章と 3.9 章を参照してください。

RENESAS

ドライブパルス周波数はコンフィグ設定によって決定します。このモジュールはコンフィグ設定に応じたレジスタ設定をして、3 種類のドライブパルス周波数を設定します。

ドライブパルス周波数は下記計算式となります。

$$f_{DrivePulse} = \frac{F_{PCLKB}}{2^{\mathrm{CLK}} \times 2(STCLK + 1)} \frac{\mathrm{SUMULTIn} + 1}{2(\mathrm{SDPA} + 1)}, \quad \mathrm{n} = 0, 1, 2$$

以下に PCLKB 周波数が 32MHz のときにドライブパルス周波数 2MHz を生成する設定を示します。SDPA はタッチインタフェース構成毎に設定可能です。

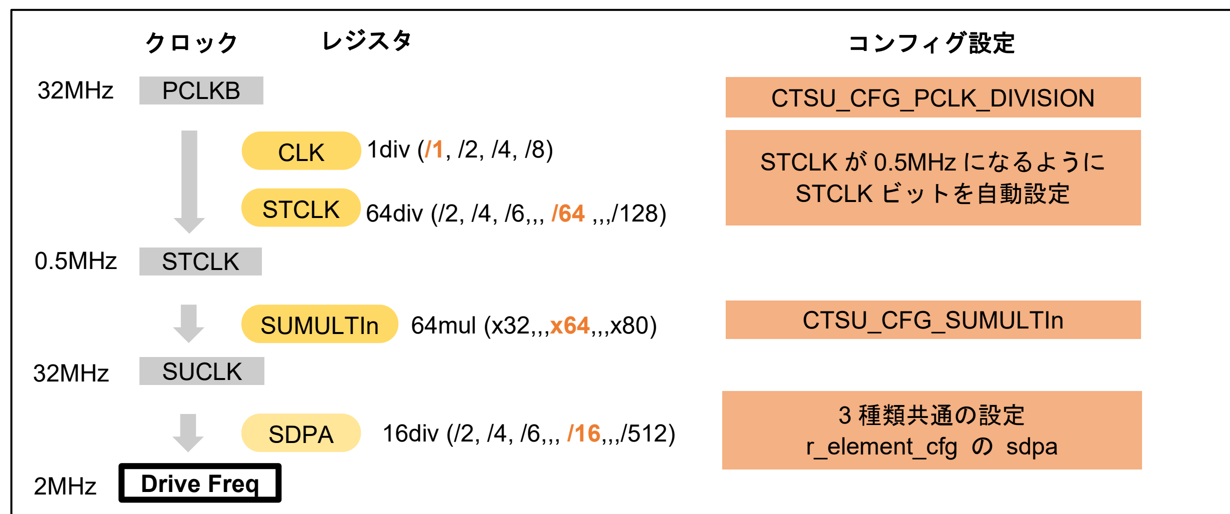

図 1.16 ドライブパルス周波数設定(CTSU2L)

1.5.3 センサ CCO 補正

CTSU ペリフェラルはセンサ CCO の MCU 製造プロセスにおける潜在的な微小バラつきに対応するため補正用回路を内蔵しています。

このモジュールは電源投入後の初期化時、補正用回路を使用して、正確なセンサ測定値を確保するための補正係数を生成します。この補正係数を用いて、取得した計測値の補正を行います。

1.5.4 イニシャルオフセットチューニング

CTSU ペリフェラルはタッチによって変化する電流量を考慮して、センサ CCO のダイナミックレンジ内に収まるように寄生容量をキャンセルするためのオフセット電流回路を内蔵しています。

このモジュールはオフセット電流設定を補正計測値がターゲット値になるように調整します。この調整は通常の計測プロセスを使用するので、起動後数回の R_CTSU_ScanStart() と R_CTSU_DataGet() が必要です。ctsu_element_cfg_t のメンバ「so」は調整の開始点として使用されるため、この値が適切であれば少ない回数で完了することができます。通常、この値は QE for Capacitive Touch によって調整された値を使用します。

CTSU2L の場合、この機能はコンフィグでオフすることが可能です。

デフォルトのターゲット値

Mode	CTSU1 target value	CTSU2L target value
Self-capacitance	15360 (37.5%)	11520 (37.5%)
Self-capacitance using active shield	-	4608 (15%)
Mutual-capacitance	10240 (25%)	7680 (25%)

パーセンテージは CCO へ最大入力電流を印加した時を 100%としたときのものです。
CTSU1 での 100%は計測時間が 526usec（基準時間）のときの補正計測値 40960 です。
CTSU2L での 100%は計測時間が 256usec（基準時間）のときの補正計測値 30720 です。
計測時間を変更すると、基準時間との比率でターゲット値を調整して適用されます。

CTSUSNUM と CTSUSDPA を組み合わせたターゲット値の例

・CTSU1（CTSU クロック= 32MHz、自己容量計測モード）

Target value	CTSUSNUM	CTSUSDPA	Measurement time
15360	0x3	0x7	526usec
30720	0x7	0x7	1052usec
30720	0x3	0xF	1052usec
7680	0x1	0x7	263usec
7680	0x3	0x3	263usec

CTSUSNUM と CTSUSDPA の組み合わせにより、計測時間は変化します。上記の表で、CTSUPRRTIO と CTSUPRMODE は推奨値を使用しています。この値を変更することは非推奨です。詳細はハードウェアマニュアルを参照してください。

・CTSU2L（自己容量計測モード）

Target value	Target value (multi frequency)	CTSUSNUM	Measurement time
5760	11520 (128us + 128us)	0x7	128usec
11520	23040 (256us + 256us)	0xF	256usec
2880	5760 (64us + 64us)	0x3	64usec

計測時間は CTSUSNUM によって異なります。STCLK を 0.5MHz に設定できない場合、上記の表はサポートされません。STCLK についてはハードウェアマニュアルに参照してください。

1.5.5 移動平均

計測結果を移動平均する機能です。コンフィグで移動平均回数を設定できます。

1.5.6 コールバック関数

CTSU ペリフェラルは計測の異常を検出するとステータスレジスタに設定します。

このモジュールは計測完了割り込み処理で CTSU1 の場合はステータスレジスタの CTSUSOVF、エラーステータスレジスタの CTSUICOMP を、CTSU2L の場合はステータスレジスタの ICOMP1、ICOMP0、SENSOVF をリードして、コールバック関数で通知します。リードした後にステータスレジスタはリセットします。異常内容はコールバック関数の引数 ctsu_callback_args_t のメンバの event を参照してください。

1.5.7 シールド機能(CTSU2L)

CTSU2L ペリフェラルは寄生容量の増加を抑えつつ外部からの影響をシールドするために、シールド用端子および非計測端子からドライブパルスと同相のシールド信号を出力する機能を内蔵しています。この機能は自己容量計測のときのみ使用可能です。

このモジュールは各タッチインタフェース構成に対してシールドを設定できます。

例えば、図 1.17 のような電極構成の場合、以下に示すように ctsu_cfg_t のメンバを設定してください。ここでは、その他のメンバは省略しています。

```
    .txvsel         = CTSU_TXVSEL_INTERNAL_POWER,
    .txvsel2        = CTSU_TXVSEL_MODE,
    .md             = CTSU_MODE_SELF_MULTI_SCAN,
    .posel          = CTSU_POSEL_SAME_PULSE,
    .ctsuchac0      = 0x0F,
    .ctsuchtrc0     = 0x08,
```

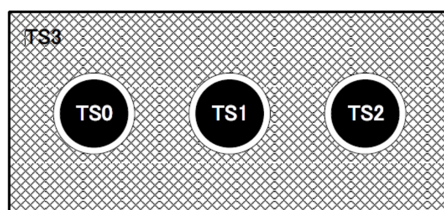

図 1.17 シールド電極構成例

1.5.8 複数電極接続（MEC : Multiple Electrode Connection）機能（CTSU2SL)

MEC は複数の電極を接続し、1 つの電極として計測する機能です。本機能は自己容量計測モード時のみ使用可能です。

例えば 3 つの電極を使用する場合、通常時は通常計測して 3 チャネルの計測をしてそれぞれの補正計測値を取得し、省電力時は MEC 計測して 3 チャネルを合わせた 1 チャネルの計測をして 1 つの補正計測値を取得するということが可能です。MEC 計測時は接続した電極の容量が合成されるので注意してください。

図 1.18 に通常計測と MEC 機能を使用した計測の計測時間の比較を示します。複数チャネルを同時に計測するため、計測時間が短くなります。

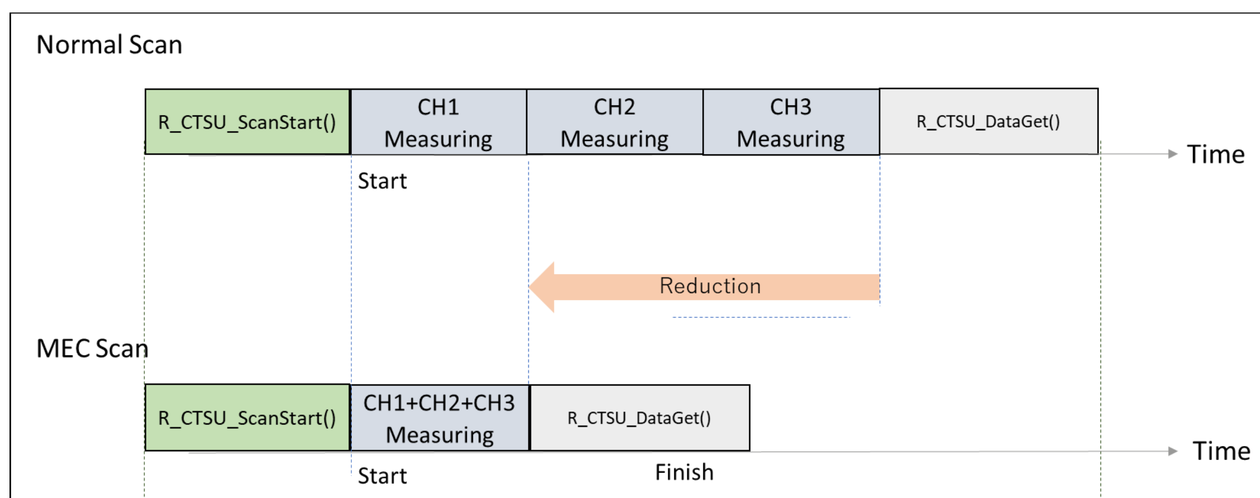

図 1.18 通常計測と MEC 機能を使用した計測の計測時間比較

MEC 機能のコードを有効にするには、CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE を 1 に設定します。

MEC 機能を使用する際、同じ TS 端子に対して通常のタッチインタフェース構成とは別のタッチインタフェース構成を作成します。MEC 計測用のタッチインタフェース構成では下記の設定が必要となります。

ctsu_cfg_t の tsod を 1 に設定することでタッチインタフェース構成に対して MEC 機能を有効できます。

ctsu_cfg_t の mec_ts を計測する TS 番号のいずれかに設定してください。

シールド機能も同時に使用する場合は、ctsu_cfg_t の mec_shield_ts にシールド端子の TS 端子番号を設定してください。この場合、シールド端子として使用できる TS 端子は 1 つです。

ctsu_cfg_t の num_rx は 1 に設定してください。

例えば、図 1.19 の電極構成の場合、以下に示すように ctsu_cfg_t のメンバを設定してください。ここでは、その他のメンバは省略しています。

```
    .tsod = 1,
    .mec_ts = 0,
    .mec_shield_ts = 3,
    .num_rx = 1,
```

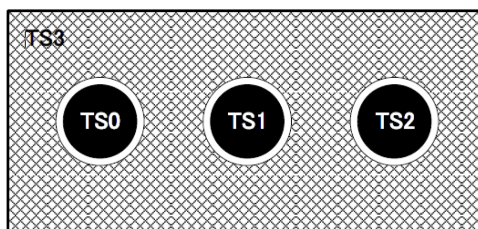

図 1.19 MEC＋シールド電極構成例

1.5.9 自動 CCO 補正機能(CTSU2SL)

自動 CCO 補正機能とは、CTSU2SL ペリフェラル内部でセンサ CCO 補正の計算を実施する機能です。センサ CCO 補正についての詳細は、1.1.3 章を参照してください。

CTSU2SL が内部でセンサ CCO 補正計算を処理するため、ソフトウェアの補正計算処理を使用することなく補正済データを計算することができ、メインプロセッサの処理時間を消費しません。

自動 CCO 補正機能を有効にするには、CTSU_CFG_AUTO_CORRECTION_ENABLE を 1 に設定します。

1.5.10 自動周波数補正機能(CTSU2SLa)

自動周波数補正機能とは、CTSU2SLa ペリフェラル内部でマルチクロック補正を行う機能です。

CTSU2SLa ペリフェラルが内部でマルチクロック補正計算を処理するため、ソフトウェアの補正計算処理を使用することなく補正済データを計算することができ、メインプロセッサの処理時間を消費しません。

自動周波数補正機能を有効にするには、CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE を 1 に設定します。

自動周波数補正機能を使用する場合は、自動 CCO 補正機能も有効にしてください。

RENESAS

1.6 他の FIT モジュールとの連携

CTSU モジュールは他の FIT モジュールと連携する必要がある機能があります。

表 1.4 連携する FIT モジュール一覧

FIT モジュール名	用途	連携方法
CMT, LPT などのタイマ	外部トリガでの計測開始	CMT FIT モジュールなどの使用するタイマの FIT モジュールを追加 使用するタイマに応じてアプリケーションで設定
ELC	外部トリガでの計測開始	ELC FIT モジュールを追加 使用するタイマに応じてアプリケーションで設定
LPC	低消費電力動作のタッチ動作	LPC FIT モジュールを追加
DTC	計測時の INTCTSUWR での書き込みと INTCTSURD での読み出し	DTC FIT モジュールを追加 CTSU モジュール内で DTC FIT モジュールの API を設定済み
S12ADC	診断機能(CTSU2L)の出力電圧診断	S12ADC FIT モジュールを追加 CTSU モジュール内で r_s12ad_rx モジュールの API を設定済み

1.6.1 計測開始トリガ

計測はソフトウェアトリガまたは外部トリガのいずれかによって開始できます。システムに応じて選択してください。

ソフトウェアトリガは、R_CTSU_ScanStart()で計測動作開始ビットに 1 を設定することで計測を開始します。タッチインタフェース構成に対する設定をしてから、計測動作開始ビットをセットするので、API コールから実際の計測開始までにわずかな遅延が発生します。ただ、これは計測時間に比べて非常に小さいので、通常はシンプルな設定となるソフトウェアトリガを推奨します。

外部トリガは、R_CTSU_ScanStart()で計測開始トリガ選択ビットに 1 を設定した後に計測動作開始ビットに 1 を設定してトリガ待ち状態にして、イベント入力で計測を開始します。一般的にはタイマを使用して定期的に計測します。ソフトウェアトリガのわずかな遅延を許容できないような場合や低消費電力での動作が必要な場合に使用してください。外部トリガで複数の計測モードを使用する場合は、1 つの計測モードの計測が完了したときに R_CTSU_ScanStart()で別の計測モードの計測開始を設定してください。

1.6.2 低消費電力動作

LPC は各動作制御モードへの遷移や、低消費電力状態を制御し、消費電力を低減します。LPC モジュールと、CTSU2SL が搭載する自動判定モードと組み合わせることで、低消費電力動作のタッチ動作を実現可能です。詳しくは、各 MCU のスマートウェイクアップの APN をご参照ください。

1.6.3 データ転送

CTSU モジュールでは、計測中に発生する INTCTSUWR および INTCTSURD の処理に対して、CPU の代わりに DTC を使用することができます。

自動判定モードでは DTC モジュールを必ず利用する必要がありますが、その他計測モードでは DTC を任意に設定可能です。

DTC モジュールを利用する際は、スマート・コンフィグレータ―上で、以下の設定を行ってください。

- コンポーネントタブから r_dtc_rx をインポートする。
- r_dtc_rx の設定の“DMAC FIT check”を” DMAC FIT module is not used with DTC FIT module”にする。
- r_ctsu_qe の設定の”Data transfer of INTCTSUWR and INTCTSURD” を”DTC”に設定する。

- r_bsp の設定のヒープサイズを 0x1000 以上に設定する。

1.6.4 診断機能

CTSU2L の診断モードでは ADC 計測を行うタイミングがあります。よって、スマート・コンフィグレータ上で、コンポーネントタブから r_s12ad_rx を追加してください。追加後は、r_s12ad_rx モジュール内で個別の設定を行う必要はありません。

ADC FIT を診断モード以外にアプリケーション上でも使用する際は以下のタイミングで ADC モジュールをクローズしてリソースを開放して下さい。

- リセット後、最初に診断モードの R_CTSU_ScanStart()をコールする前。
- R_CTSU_Diagnosis()実行後、最初に診断モードの R_CTSU_ScanStart()をコールする前。

なお、ADC モジュールのクローズは R_ADC_Close()を実行することで処理できます。

クローズを行わなかった場合、R_CTSU_Diagnosis()の戻り値として FSP_ERR_ABORTED が返ります。その場合、ADC FIT をクローズして、次の診断モード実行時に CTSU モジュール内の ADC 計測が実施出来るようにして下さい。

1.7 API 概要

本モジュールの API 関数一覧を以下に記載します。

全ての API 関数の第一引数にはコントロール構造体のポインタを渡してください。それ以外の引数についてポインタを渡す場合は NULL でないこと、および各 API 関数で必要なサイズを確保していることを確認してください。ただし、R_CTSU_CallbackSet()は例外ですので、API 関数の詳細説明 3.4 を参照してください。

表 1.5 API 関数一覧

関数	説明
R_CTSU_Open()	指定したタッチインタフェース構成を初期化します。
R_CTSU_ScanStart()	指定したタッチインタフェース構成の計測を開始します。
R_CTSU_DataGet()	指定したタッチインタフェース構成の全ての補正計測値を取得します。
R_CTSU_CallbackSet()	指定したタッチインタフェース構成のコールバック関数を設定します。
R_CTSU_Close()	指定したタッチインタフェース構成を終了します。
R_CTSU_Diagnosis()	診断を実行します。
R_CTSU_ScanStop()	指定したタッチインタフェース構成の計測を停止します。
R_CTSU_SpecificDataGet()	指定したタッチインタフェース構成の指定したデータ種別の計測値または補正計測値を読み込みます。
R_CTSU_DataInsert()	指定したデータを指定したタッチインタフェース構成の補正計測値バッファに格納します。
R_CTSU_OffsetTuning()	指定したタッチインタフェース構成のオフセットレジスタ（SO）を調整します。
R_CTSU_AutoJudgementDataGet()	自動判定機能を使用して、指定したタッチインタフェース構成の全てのボタン判定結果を取得します。

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 が以下の機能のいずれかをサポートしている必要があります。

- CTSU
- CTSUa
- CTSU2L
- CTSU2SL
- CTSU2SLa

2.2 ソフトウェアの要求

このモジュールは以下の 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モジュール (r_bsp) v7.70

コンフィグレーション設定によって以下のモジュールにも依存します。

- DTC モジュール r_dtc v4.50（DTC 転送を使用する場合）
  DTC 転送を使用する際は、r_bsp のプロパティの Heap size を 0x1000 以上にしてご使用ください。
  GCC コンパイラを使用する場合は Heap size を 0x1600 推奨しています。
- ADC モジュール r_s12ad_rx_v5.41（診断モードを使用する場合）

また、以下のツールの使用を想定しています。

- 静電容量式タッチセンサ対応開発支援ツール QE for Capacitive Touch V4.3.0

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開発環境とコンパイラで動作確認を行っています。

開発環境

- Renesas e^2 studio 2025-12
- IAR Embedded Workbench for Renesas RX 5.20.1

コンパイラ

- Renesas CC-RX Toolchain v3.07.00
- GCC RX Toolchain v14.2.0.202511
- IAR C/C++ Compiler for Renesas RX version 5.20.1

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

このモジュールは 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設定の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。

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“BSP_CFG_PARAM_CHECKING_ENABLE”	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。 “BSP_CFG_PARAM_CHECKING_ENABLE”の場合、BSP での設定に依存します。
CTSU_CFG_DTC_SUPPORT_ENABLE ※ デフォルト値は“0”	“1”を設定することで CTSU2L の CTSUWR 割り込みおよび CTSURD 割り込み処理にメインプロセッサではなく DTC を使用します。 注意事項: 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、このモジュールの使用と競合する可能性があります。
CTSU_CFG_AUTO_JUDGE_ENABLE ※ デフォルト値は“0”	“1”に設定することで自動判定のコードを有効にします。
CTSU_CFG_INTCTSUWR_PRIORITY_LEVEL ※ デフォルト値は“2”	CTSUWR 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 0 (低) ― 15 (高)になります。
CTSU_CFG_INTCTSURD_PRIORITY_LEVEL ※ デフォルト値は“2”	CTSURD 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 0 (低) ― 15 (高)になります。
CTSU_CFG_INTCTSUFN_PRIORITY_LEVEL ※ デフォルト値は“2”	CTSUFN 割り込みの優先度レベルを設定します。範囲は 0 (低) ― 15 (高)になります。
以下のコンフィグレーションは、タッチインタフェース構成に応じた設定となるため、Smart Configurator では設定しません。 QE for Capacitive Touch を使用すると設定されます。その場合、プロジェクトに QE_TOUCH_CONFIGURATION が定義され、r_ctsu_qe_config.h の定義は無効になり、代わりに qe_touch_define.h に定義されます。	
QE_TOUCH_VERSION	QE のバージョン
CTSU_CFG_NUM_SELF_ELEMENTS	自己容量、電流計測、診断機能の合計 TS 端子数を設定します。
CTSU_CFG_NUM_MUTUAL_ELEMENTS	相互容量の合計マトリックス数を設定します。
CTSU_CFG_NUM_AUTOJUDGE_SELF_ELEMENTS	自動判定自己容量エレメントの総数を設定します。
CTSU_CFG_NUM_AUTOJUDGE_MUTUAL_ELEMENTS	自動判定相互容量エレメントの総数を設定します。
CTSU_CFG_LOW_VOLTAGE_MODE	Low Voltage モードを有効にするか選択します。この値が CTSUCRAL レジスタの ATUNE0 ビットに設定されます。 注意事項： 本ソフトは CTSU1 の Low Voltage モードをサポートしていないため、CTSU1 の場合は 0 を設定してください。
CTSU_CFG_PCLK_DIVISION	PCLK の分周率を設定します。この値が CTSU1 は CTSUCR1 レジスタの CTSUCLK ビット、CTSU2L

	は CTSUCRAL レジスタの CLK ビットに設定されます。
CTSU_CFG_TSCAP_PORT	TSCAP 端子を設定します。 例えば、P30 の場合 0x0300 を設定します。
CTSU_CFG_VCC_MV	VCC の電圧を設定します。 例えば、5.00V の場合 5000 を設定します。
CTSU_CFG_NUM_SUMULTI	マルチクロック計測の数を設定します。
CTSU_CFG_SUMULTI0	マルチクロック計測の 1 つ目の周波数の逓倍率を設定します。 0x3F が推奨値です。(RX260/RX261 は 0x2F を推奨)
CTSU_CFG_SUMULTI1	マルチクロック計測の 2 つ目の周波数の逓倍率を設定します。 0x36 が推奨値です。(RX260/RX261 は 0x28 を推奨)
CTSU_CFG_SUMULTI2	マルチクロック計測の 3 つ目の周波数の逓倍率を設定します。 0x48 が推奨値です。(RX260/RX261 は 0x36 を推奨)
CTSU_CFG_DIAG_SUPPORT_ENABLE	診断機能を有効にするか選択します。
CTSU_CFG_DIAG_DAC_TS	CTSU1 で診断に使用する TS 端子の番号を指定します。
CTSU_CFG_AUTO_CORRECTION_ENABLE	自動 CCO 補正を有効にするか選択します。
CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE	自動周波数補正を有効にするか選択します。
CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE	MEC 機能を有効にするか選択します。
CTSU_CFG_MAJORITY_MODE	タッチ判定方法処理のビットマップ。1 ビット目は VMM、2 ビット目は JMM を示します。タッチインタフェース構成に合わせて設定します。 1 : VMM 2 : JMM 3 : VMM と JMM の混在

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。

Renesas CC-RX Toolchain v3.07.00 を使用して、下記のコンパイル時の設定でのサイズを示します。サイズに関わる設定のみ示しています。

- CTSU_CFG_PARAM_CHECKING_ENABLE 0
- CTSU_CFG_DTC_SUPPORT_ENABLE 0
- CTSU_CFG_AUTO_JUDGE_ENABLE 0
- CTSU_CFG_LOW_VOLTAGE_MODE 0
- CTSU_CFG_TEMP_CORRECTION_SUPPORT 0
- CTSU_CFG_CALIB_RTRIM_SUPPORT 0
- CTSU_CFG_AUTO_CORRECTION_ENABLE 0
- CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE 0
- CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE 0

自己容量と相互容量それぞれ 1 エレメントでのサイズと、1 エレメント追加で増えるサイズを示します。QE が出力する qe_touch_config.c も含みます。

[CTSU1]

- CTSU_CFG_NUM_SUMULTI 1

Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	2678 bytes	+24 bytes	2995 bytes	+22 bytes
RAM	223 bytes	+23 bytes	241 bytes	+35 bytes

[CTSU2L] VMM

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 1

Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	4320 bytes	+24 bytes	4693 bytes	+22 bytes
RAM	457 bytes	+41 bytes	491 bytes	+63 bytes

[CTSU2L] JMM

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 2

Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	4453 bytes	+24 bytes	4809 bytes	+22 bytes
RAM	465 bytes	+49 bytes	507 bytes	+79 bytes

※ FIT QE CTSU モジュール Firmware Integration Technology Rev.3.20 の APN からコードの計測方法を変更しています。

2.9 引数

API 関数の引数である構造体および列挙型を示します。API 関数で使用するパラメータの多くは、列挙型で定義しています。これは型チェックを行い、エラーを減少させるためです。

これらの構造体や列挙型はプロトタイプ宣言と共に r_ctsu_qe.h、r_ctsu_qe_api.h で定義されています。

表 2.1 に ctsu_ctrl_t 構造体（コントロール構造体）を示します。この構造体で使用しているデータ型については r_ctsu_qe.h を参照してください。各タッチインタフェース構成の計測設定や計測結果を管理します。QE for Capacitive Touch を使用することで、タッチインタフェース構成に応じたコントロール構造体の変数が qe_touch_config.c に出力されるので、このモジュールの API の第一引数に設定してください。

表 2.1 ctsu_ctrl_t 構造体

データ型	メンバ	内容
uint32_t	open	Openフラグ
volatile ctsu_state_t	state	計測状態
ctsu_cap_t	cap	計測トリガ
ctsu_md_t	md	計測モード
ctsu_tuning_t	tuning	イニシャルオフセットチューニングフラグ
uint16_t	num_elements	エレメント数
uint16_t	wr_index	CTSUWR割り込みインデックス
uint16_t	rd_index	CTSURD割り込みインデックス
uint8_t *	p_element_complete_flag	エレメントのオフセットチューニング完了フラグのポインタ
int32_t *	p_tuning_diff	目標値との差分値のポインタ
uint16_t	average	移動平均実施回数
uint16_t	num_moving_average	移動平均フィルタの平均サンプル数
uint8_t	ctsucr1	CTSUCR1の設定
ctsu_ctsuwr_t *	p_ctsuwr	CTSUWRの設定
ctsu_self_buf_t *	p_self_raw	自己容量計測値バッファのポインタ
uint16_t *	p_self_corr	自己容量補正値バッファのポインタ
uint16_t *	p_self_mfc	自己容量マルチクロック補正値バッファのポインタ
ctsu_data_t *	p_self_data	自己容量補正計測値バッファのポインタ
ctsu_mutual_buf_t *	p_mutual_raw	相互容量計測値バッファのポインタ
uint16_t *	p_mutual_pri_corr	相互容量プライマリ補正値バッファのポインタ
uint16_t *	p_mutual_snd_corr	相互容量セカンダリ補正値バッファのポインタ
uint16_t *	p_mutual_pri_mfc	相互容量プライマリマルチクロック補正値バッファのポインタ
uint16_t *	p_mutual_snd_mfc	相互容量セカンダリマルチクロック補正値バッファのポインタ
ctsu_data_t *	p_mutual_pri_data	相互容量プライマリ補正計測値バッファのポインタ
ctsu_data_t *	p_mutual_snd_data	相互容量セカンダリ補正計測値バッファのポインタ
ctsu_correction_info_t *	p_correction_info	補正情報のポインタ
ctsu_txvsel_t	txvsel	TXVSELの設定
ctsu_txvsel2_t	txvsel2	TXVSEL2の設定
uint8_t	ctsuchac0	CHAC0の設定
uint8_t	ctsuchac1	CHAC1の設定
uint8_t	ctsuchac2	CHAC2の設定
uint8_t	ctsuchac3	CHAC3の設定
uint8_t	ctsuchac4	CHAC4の設定
uint8_t	ctsuchtrc0	CHTRC0の設定
uint8_t	ctsuchtrc1	CHTRC1の設定
uint8_t	ctsuchtrc2	CHTRC2の設定
uint8_t	ctsuchtrc3	CHTRC3の設定
uint8_t	ctsuchtrc4	CHTRC4の設定
uint16_t	self_elem_index	自己容量のエレメントインデックス

データ型	メンバ	内容
uint16_t	mutual_elem_index	相互容量のエレメントインデックス
uint16_t	ctsu_elem_index	エレメントインデックス
ctsu_cfg_t const *	p_ctsu_cfg	コンフィグ構造体のポインタ
void	(* p_callback) (ctsu_callback_args_t *)	コールバック関数のポインタ
uint8_t	interrupt_reverse_flag	割り込み反転フラグ
ctsu_event_t	error_status	エラーステータス
ctsu_callback_args_t *	p_callback_memory	コールバック関数保存 (TrustZone用途)
void const *	p_context	Contextのポインタ
bool	serial_tuning_enable	シリアルチューニング有効フラグ
uint16_t	serial_tuning_mutual_cnt	シリアルチューニング
uint16_t	tuning_self_target_value	自己容量のオフセットチューニングターゲット値
uint16_t	tuning_mutual_target_value	相互容量のオフセットチューニングターゲット値
uint8_t	tsod	TSOD設定
uint8_t	mec_ts	MECのTS端子番号
uint8_t	mec_shield_ts	MECシールドのTS端子番号
CTSU_CFG_DIAG_SUPPORT_ENABLE == 1		
ctsu_diag_info_t *	p_diag_info	診断情報のポインタ
BSP_FEATURE_CTSU_VERSION == 2		
uint8_t *	p_frequency_complete_flag	マルチクロックのオフセットチューニング完了フラグのポインタ
uint8_t *	p_selected_freq_self	選択した周波数番号のポインタ（自己容量）
uint8_t *	p_selected_freq_mutual	選択した周波数番号のポインタ（相互容量）
ctsu_range_t	range	電流レンジ
uint8_t	ctsucr2	CTSUCR2の設定
(BSP_FEATURE_CTSU_VERSION == 2 && CTSU_CFG_AUTO_JUDGE_ENABLE == 1)		
ctsu_auto_judge_t *	p_auto_judge	自動判定情報のポインタ
uint32_t	address_auto_judge	p_auto_judgeのアドレス
uint32_t	address_ctsuwr	p_ctsuwrのアドレス
uint32_t	address_self_raw	p_self_rawのアドレス
uint32_t	address_mutual_raw	p_mutual_rawのアドレス
uint32_t	count_auto_judge	自動判定でのDTC転送回数
uint32_t	count_ctsuwr_self_mutual	自動判定でのCTSUWR割り込み回数
uint8_t	blini_flag	BLINI設定フラグ
uint8_t	ajmmat	AJMMAT設定
uint8_t	ajbmat	AJBMAT設定
(BSP_FEATURE_CTSU_VERSION == 2 && CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE == 1)		
uint32_t	p_mcact1	MCACT1設定のポインタ
uint32_t	p_mcact2	MCACT2設定のポインタ
uint8_t	mcact_flag	自動周波数補正設定フラグ

表 **2.2** に ctsu_cfg_t 構造体（コンフィグ構造体）を示します。

QE for Capacitive Touch を使用することで、タッチインタフェース構成に応じた変数が qe_touch_config.c に出力されるので、R_CTSU_Open()の第二引数に設定してください。コンフィグの値は Smart Configurator または QE for Capacitive Touch が設定する前提となっており、処理の効率化のために本ソフトウェアではエラー確認をしません。手動で修正する場合は注意してください。

表 2.2 ctsu_cfg_t 構造体

データ型	メンバ名	内容	値の範囲
ctsu_cap_t	cap	CTSU スキャン開始トリガ選択	CTSU_CAP_SOFTWARE：ソフトウェアトリガ CTSU_CAP_EXTERNAL：外部トリガ
ctsu_txvsel_t	txvsel	送信電源選択	CTSU_TXVSEL_VCC：VCC 選択 CTSU_TXVSEL_INTERNAL_POWER：VDD 選択
ctsu_txvsel2_t	txvsel2	送信電源選択 2 (CTSU2 のみ)	CTSU_TXVSEL_MODE：TXVSEL に従う CTSU_TXVSEL_VCC_PRIVATE：VCC(専用)選択
ctsu_atune1_t	atune1	電源容量調整 (CTSU のみ)	CTSU_ATUNE1_NORMAL：通常出力 CTSU_ATUNE1_HIGH：高電流出力
ctsu_atune12_t	atune12	電源容量調整 (CTSU2 のみ)	CTSU_ATUNE12_80UA：80uA モード CTSU_ATUNE12_40UA：40uA モード CTSU_ATUNE12_20UA：20uA モード CTSU_ATUNE12_160UA：160uA モード
ctsu_md_t	md	CTSU 計測モード選択	CTSU_MODE_SELF_MULTI_SCAN：自己容量マルチスキャンモード CTSU_MODE_MUTUAL_FULL_SCAN：相互容量フルスキャンモード CTSU_MODE_MUTUAL_CFC_SCAN：相互容量 CFC スキャンモード (CTSU2 のみ) CTSU_MODE_CURRENT_SCAN：電流スキャンモード(CTSU2L) CTSU_MODE_DIAGNOSIS_SCAN：Diagnosis スキャンモード
ctsu_posel_t	posel	非計測端子出力選択	CTSU_POSEL_LOW_GPIO：L 出力(GPIO) CTSU_POSEL_HI_Z：Hi-Z 出力 CTSU_POSEL_LOW：L 出力(TXVSEL/TXVSEL2 の設定) CTSU_POSEL_SAME_PULSE：同相(送信)パルス出力(TXVSEL/TXVSEL2 の設定)
uint8_t	tsod	TS 端子固定出力選択	0：静電容量計測モード 1：TS 端子固定出力(High/Low 出力)
uint8_t	mec_ts	MEC 機能で使用する TS 端子番号	0 to 35
uint8_t	mec_shield_ts	MEC 機能で使用するアクティブシールドの TS 端子番号	0 to 35
uint8_t	tlot	自動判定のタッチしきい値 L を超えた連続判定回数	0 to 255
uint8_t	thot	自動判定のタッチしきい値 H を超えた連続判定回数	0 to 255
uint8_t	jc	自動判定の判定条件	0：判定結果の 1 つがしきい値 H 超えでタッチ判定 1：判定結果の 2 つがしきい値 H 超えでタッチ判定 2：判定結果の 3 つがしきい値 H 超えでタッチ判定 3：判定結果の 4 つがしきい値 H 超えでタッチ判定
uint8_t	ajmmat	自動判定の補正計測値移動平均回数	0 to 11 （2^設定値）
uint8_t	ajbmat	自動判定のベースライン平均回数	0 to 15 （2^(設定値+1)、0 はベースライン更新停止）
uint8_t	mtucfen	自動判定の相互容量演算	0：減算しない 1：2 回目計測データから 1 回目計測データを減算する
uint8_t	ajfen	自動判定機能有効選択	0：自動判定機能無効 1：自動判定機能有効
uint8_t	autojudge_monitor_num	自動判定用 QE モニタ構成番号	0 to 7
uint8_t	ctsuchac0	TS00-TS07 有効マスク	0x00 to 0xFF
uint8_t	ctsuchac1	TS08-TS15 有効マスク	0x00 to 0xFF
uint8_t	ctsuchac2	TS16-TS23 有効マスク	0x00 to 0xFF
uint8_t	ctsuchac3	TS24-TS31 有効マスク	0x00 to 0xFF
uint8_t	ctsuchac4	TS32-TS35 有効マスク	0x00 to 0xFF

データ型	メンバ名	内容	値の範囲
uint8_t	ctsuchtrc0	TS00-TS07 相互送信マスク	0x00 to 0xFF
uint8_t	ctsuchtrc1	TS08-TS15 相互送信マスク	0x00 to 0xFF
uint8_t	ctsuchtrc2	TS16-TS23 相互送信マスク	0x00 to 0xFF
uint8_t	ctsuchtrc3	TS24-TS31 相互送信マスク	0x00 to 0xFF
uint8_t	ctsuchtrc4	TS32-TS35 相互送信マスク	0x00 to 0xFF
ctsu_element_cfg_t *	p_elements	エレメント・コンフィグレーションのポインタ	-
uint8_t	num_rx	受信端子数	0 to 36
uint8_t	num_tx	送信端子数	0 to 36
uint16_t	num_moving_average	計測データ移動平均数	0 to 65535
bool	tunning_enable	イニシャルオフセットチューニングフラグ	true : Enable false : Disable
void *	p_callback	CTSUFN 割り込みコールバック	-
void *	p_context	コンテキストのポインタ	-
void *	p_extend	拡張構成のポインタ	-
uint16_t	tuning_self_target_value	自己容量イニシャルオフセットターゲット値	0 to 65535
uint16_t	tuning_mutual_target_value	相互容量イニシャルオフセットターゲット値	0 to 65535
ctsu_auto_button_cfg_t *	p_ctsu_auto_buttons	自動判定ボタン設定の配列へのポインタ	-

上記構造体で使用している型を示します。

```
/** CTSU Events for callback function */
typedef enum e_ctsu_event
{
    CTSU_EVENT_SCAN_COMPLETE = 0x00,   ///< Normal end
    CTSU_EVENT_OVERFLOW      = 0x01,   ///< Sensor counter overflow (CTSUST.CTSUSOVF set)
    CTSU_EVENT_ICOMP         = 0x02,   ///< Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set)
    CTSU_EVENT_ICOMP1        = 0x04    ///< Abnormal sensor current (CTSUSR.ICOMP1 set)
} ctsu_event_t;

/** CTSU Scan Start Trigger Select */
typedef enum e_ctsu_cap
{
    CTSU_CAP_SOFTWARE,              ///< Scan start by software trigger
    CTSU_CAP_EXTERNAL               ///< Scan start by external trigger
} ctsu_cap_t;

/** CTSU Transmission Power Supply Select */
typedef enum e_ctsu_txvsel
{
    CTSU_TXVSEL_VCC,                ///< VCC selected
    CTSU_TXVSEL_INTERNAL_POWER       ///< Internal logic power supply selected
} ctsu_txvsel_t;

/** CTSU Transmission Power Supply Select 2 (CTSU2 Only) */
typedef enum e_ctsu_txvsel2
{
    CTSU_TXVSEL_MODE,               ///< Follow TXVSEL setting
    CTSU_TXVSEL_VCC_PRIVATE,         ///< VCC private selected
} ctsu_txvsel2_t;

/** CTSU Power Supply Capacity Adjustment (CTSU Only) */
typedef enum e_ctsu_atune1
{
    CTSU_ATUNE1_NORMAL,             ///< Normal output (40uA)
    CTSU_ATUNE1_HIGH                ///< High-current output (80uA)
} ctsu_atune1_t;

/** CTSU Power Supply Capacity Adjustment (CTSU2 Only) */
typedef enum e_ctsu_atune12
{
    CTSU_ATUNE12_80UA,              ///< High-current output (80uA)
    CTSU_ATUNE12_40UA,              ///< Normal output (40uA)
    CTSU_ATUNE12_20UA,              ///< Low-current output (20uA)
```

```
    CTSU_ATUNE12_160UA                ///< Very high-current output (160uA)
} ctsu_atune12_t;

/** CTSU Measurement Mode Select */
typedef enum e_ctsu_mode
{
    CTSU_MODE_SELF_MULTI_SCAN  = 1,    ///< Self-capacitance multi scan mode
    CTSU_MODE_MUTUAL_FULL_SCAN = 3,    ///< Mutual capacitance full scan mode
    CTSU_MODE_MUTUAL_CFC_SCAN  = 7,    ///< Mutual capacitance cfc scan mode (CTSU2 Only)
    CTSU_MODE_CURRENT_SCAN     = 9,    ///< Current scan mode (CTSU2 Only)
    CTSU_MODE_CORRECTION_SCAN  = 17,   ///< Correction scan mode (CTSU2 Only)
    CTSU_MODE_DIAGNOSIS_SCAN   = 33    ///< Diagnosis scan mode
} ctsu_md_t;

/** CTSU Non-Measured Channel Output Select (CTSU2 Only) */
typedef enum e_ctsu_posel
{
    CTSU_POSEL_LOW_GPIO,              ///< Output low through GPIO
    CTSU_POSEL_HI_Z,                  ///< Hi-Z
    CTSU_POSEL_LOW,                   ///< Output low through the power setting by the TXVSEL[1:0] bits
    CTSU_POSEL_SAME_PULSE             ///< Same phase pulse output as transmission channel through the
power setting by the TXVSEL[1:0] bits
} ctsu_posel_t;

/** CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) */
typedef enum e_ctsu_ssdiv
{
    CTSU_SSDIV_4000,                  ///< 4.00 <= Base clock frequency (MHz)
    CTSU_SSDIV_2000,                  ///< 2.00 <= Base clock frequency (MHz) < 4.00
    CTSU_SSDIV_1330,                  ///< 1.33 <= Base clock frequency (MHz) < 2.00
    CTSU_SSDIV_1000,                  ///< 1.00 <= Base clock frequency (MHz) < 1.33
    CTSU_SSDIV_0800,                  ///< 0.80 <= Base clock frequency (MHz) < 1.00
    CTSU_SSDIV_0670,                  ///< 0.67 <= Base clock frequency (MHz) < 0.80
    CTSU_SSDIV_0570,                  ///< 0.57 <= Base clock frequency (MHz) < 0.67
    CTSU_SSDIV_0500,                  ///< 0.50 <= Base clock frequency (MHz) < 0.57
    CTSU_SSDIV_0440,                  ///< 0.44 <= Base clock frequency (MHz) < 0.50
    CTSU_SSDIV_0400,                  ///< 0.40 <= Base clock frequency (MHz) < 0.44
    CTSU_SSDIV_0360,                  ///< 0.36 <= Base clock frequency (MHz) < 0.40
    CTSU_SSDIV_0330,                  ///< 0.33 <= Base clock frequency (MHz) < 0.36
    CTSU_SSDIV_0310,                  ///< 0.31 <= Base clock frequency (MHz) < 0.33
    CTSU_SSDIV_0290,                  ///< 0.29 <= Base clock frequency (MHz) < 0.31
    CTSU_SSDIV_0270,                  ///< 0.27 <= Base clock frequency (MHz) < 0.29
    CTSU_SSDIV_0000                   ///< 0.00 <= Base clock frequency (MHz) < 0.27
} ctsu_ssdiv_t;

/** CTSU select data type for slect data get */
typedef enum e_ctsu_specific_data_type
{
    CTSU_SPECIFIC_RAW_DATA,
    CTSU_SPECIFIC_CORRECTION_DATA,
    CTSU_SPECIFIC_SELECTED_FREQ,
} ctsu_specific_data_type_t;

/** Callback function parameter data */
typedef struct st_ctsu_callback_args
{
    ctsu_event_t event;               ///< The event can be used to identify what caused the callback.
    void const * p_context;           ///< Placeholder for user data. Set in ctsu_api_t::open function
in ::ctsu_cfg_t.
} ctsu_callback_args_t;

/** CTSU Control block. Allocate an instance specific control block to pass into the API calls.
 * @par Implemented as
 * - ctsu_instance_ctrl_t
 */
typedef void ctsu_ctrl_t;

/** CTSU Configuration parameters. */
/** Element Configuration */
typedef struct st_ctsu_element
{
    ctsu_ssdiv_t ssdiv;               ///< CTSU Spectrum Diffusion Frequency Division Setting (CTSU
Only)
    uint16_t    so;                   ///< CTSU Sensor Offset Adjustment
    uint8_t     snum;                 ///< CTSU Measurement Count Setting
    uint8_t     sdpa;                 ///< CTSU Base Clock Setting
} ctsu_element_cfg_t;
```

```
/** Configuration of each automatic judgement button */
typedef struct st_ctsu_auto_button_cfg
{
    uint8_t  elem_index;            ///< Element number used by this button for automatic judgement.
    uint16_t threshold;             ///< Touch/non-touch judgement threshold for automatic
judgement.
    uint16_t hysteresis;            ///< Threshold hysteresis for chattering prevention for
automatic judgement.
} ctsu_auto_button_cfg_t;
```

2.10 戻り値

API 関数の戻り値を示します。この列挙型は、fsp_common_api.h で記載されています。

```
** Common error codes */
typedef enum e_fsp_err
{
    FSP_SUCCESS = 0,

    FSP_ERR_ASSERTION           = 1,        ///< A critical assertion has failed
    FSP_ERR_INVALID_POINTER      = 2,       ///< Pointer points to invalid memory location
    FSP_ERR_INVALID_ARGUMENT     = 3        ///< Invalid input parameter
    FSP_ERR_INVALID_CHANNEL      = 4,       ///< Selected channel does not exist
    FSP_ERR_INVALID_MODE        = 5,        ///< Unsupported or incorrect mode
    FSP_ERR_UNSUPPORTED         = 6,        ///< Selected mode is not supported by this API
    FSP_ERR_NOT_OPEN            = 7,        ///< Requested channel is not configured or API not open
    FSP_ERR_ABORTED             = 18,       ///< An operation was aborted

    /* Start of CTSU Driver specific */
    FSP_ERR_CTSU_SCANNING             = 6000,         ///< Scanning.
    FSP_ERR_CTSU_NOT_GET_DATA         = 6001,         ///< Not processed previous scan data.
    FSP_ERR_CTSU_INCOMPLETE_TUNING    = 6002,         ///< Incomplete initial offset tuning.
    FSP_ERR_CTSU_DIAG_NOT_YET         = 6003,         ///< Diagnosis of data collected not yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,        ///< Diagnosis of Output Voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH        = 6005,         ///< Diagnosis of CCO High failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW         = 6006,         ///< Diagnosis of CCO Low failed.
    FSP_ERR_CTSU_DIAG_SSCG            = 6007,         ///< Diagnosis of SSCG Oscillator failed.
    FSP_ERR_CTSU_DIAG_DAC             = 6008,         ///< Diagnosis of Sensor Offset failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE   = 6009,        ///< Diagnosis of Output Voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE     = 6010,        ///< Diagnosis of Over Voltage Detection failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT     = 6011,        ///< Diagnosis of Over Current Detection failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE  = 6012,        ///< Diagnosis of Load Resistance failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE   = 6013,        ///< Diagnosis of Current Offset failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN     = 6014,        ///< Diagnosis of SENSCLK Frequency failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN      = 6015,         ///< Diagnosis of SUCLK Frequency failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY   = 6016,        ///< Diagnosis of SUCLK Clock Recovery failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN        = 6017,         ///< Diagnosis of CFC oscillator failed.
  } fsp_err_t;
```

2.11 コールバック関数

本 FIT モジュールは、計測完了割り込みの処理が完了したときに、登録されたコールバック関数をコールします。コンフィグ構造体のメンバ p_callback に設定してください。QE の出力コードでは設定済です。R_CTSU_CallbackSet()で設定することも可能です。3.4 章を参照してください。

コールバック関数は、アプリケーションで用意してください。QE を使用してチューニング結果を出力すると、下記のコールバック関数のサンプルコードも出力します。ソフトウェア判定と自動判定のときで、出力する関数が変わります。両方の構成があるときは、両方出力します。

ソフトウェア判定

```
void qe_touch_callback(touch_callback_args_t * p_args)
{
    g_qe_touch_flag = 1;
    g_qe_ctsu_event = p_args -> event;
}
```

自動判定

```
void qe_ctsu_auto_callback(ctsu_callback_args_t * p_args)
{
    g_qe_touch_flag = 1;
    g_qe_ctsu_event = p_args -> event;
}
```

下記に示す通り、R_CTSU_ScanStart()と R_CTSU_DataGet()の間で、g_qe_touch_flag をポーリングすることを想定しています。

```
R_CTSU_ScanStart(g_qe_ctsu_instance.p_ctrl);

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

R_CTSU_DataGet(g_qe_ctsu_instance.p_ctrl, &data);
```

コールバック関数の引数については、2.9 章の ctsu_callback_args_t を参照してください。touch_call_back_arg_t は Touch モジュールで ctsu_callback_args_t を typedef した型です。1.5.5 章で説明した通り、構造体メンバの event で計測でのエラー有無を確認できます。

2.12 API の処理時間

本章では、API の処理時間の参考値を示します。

処理時間を測定した条件を以下に示します。

- コンパイラ　: Renesas CC-RX v3.07.00
- コンパイラ最適条件　: レベル 2
- メインクロック　: 32 MHz

基本動作モデルの API 処理時間を示します。

例として、自己容量計測モードでエレメント 1 つを計測した場合の API の処理時間を以下に示します。なお、R_CTSU_Open は、リセット後の初回実行時は、CCO 補正の補正係数を作成するための処理をするため、処理時間が長くなります。

表 2.3 基本動作モデルの API の処理時間

関数名	計測時間(us)
R_CTSU_Open（初回）	102251
R_CTSU_Open（2 回目以降）	52
R_CTSU_ScanStart	8
R_CTSU_DataGet	24

診断モード利用時の API の処理時間を示します。R_CTSU_Open()は基本動作モデルと同じです。

診断モードでは、診断項目によって処理内容が異なるため、R_CTSU_ScanStart()と R_CTSU_DataGet()の処理時間が異なります。表 2.4 と表 2.5 を参照してください。出力電圧診断は R_CTSU_DataGet()関数を使用しないため、対象外となります。

また、表 2.4 の診断名は表 1.2 と対応しています。

表 2.4 診断モード実行時の API の処理時間

関数名	計測時間(us)
R_CTSU_Diagnosis	1

表 2.5 診断モード実行時の API の処理時間

診断名	R_CTSU_ScanStart	R_CTSU_DataGet
出力電圧診断	849	
過電圧検出診断	16	7
過電流検出診断	15	7
内部抵抗診断	16	14
センサオフセット診断	17	12
SENSCLK 周波数診断	17	7
SUCLK 周波数診断	16	7
SUCLK クロックリカバリ診断	18	7

2.13 FIT モジュールの追加方法

2.13.1 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) e[2] studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e[2] 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 e[2] 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) e[2] studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e[2] 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 e[2] 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 e[2] 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。

2.13.2 Smart Configurator を使用しない場合のモジュールオプションの設定

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。

3. API 関数

3.1 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。タッチインタフェース構成毎に実行してください。

Format

```
fsp_err_t R_CTSU_Open (ctsu_ctrl_t * const p_ctrl,
                        ctsu_cfg_t const * const p_cfg)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ
p_cfg : [in] コンフィグレーション構造体へのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_ALREADY_OPEN	*/* Close()のコールなしに Open()がコールされました */*
FSP_ERR_INVALID_ARGUMENT	*/* コンフィグレーションのパラメータが不正です */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、引数 p_cfg に従ってコントロール構造体の初期設定、レジスタの初期設定、割り込みの設定および有効化をします。

また、最初のタッチインタフェース構成に対する処理のときには、補正係数の作成処理を実行します。この処理には約 100ms を必要とします。

最初のタッチインタフェース構成に対する処理かつ CTSU_CFG_DTC_SUPPORT_ENABLE が有効になっている場合には DTC の初期化をします。

Example

```
fsp_err_t err;

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

/* Initialize the API. */
err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);

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

Special Notes:

この関数のコール前にポートを初期化する必要があります。ポート初期化関数は SmartConfigurator によって作成される R_CTSU_PinSetInit() の使用を推奨します。

タッチインタフェース構成が診断モードのときは、他のタッチインタフェース構成の R_CTSU_Open()を先に実行してください。

3.2 R_CTSU_ScanStart

この関数は、指定したタッチインタフェース構成の計測を開始します。

Format

```
fsp_err_t R_CTSU_ScanStart (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

Return Values

FSP_SUCCESS / 成功しました */*
FSP_ERR_ASSERTION / 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN / Open()のコールなしにコールされました */*
FSP_ERR_CTSU_SCANNING / スキャン中 */*
FSP_ERR_CTSU_NOT_GET_DATA / 前の結果を取得していません */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、ソフトウェアトリガの場合、タッチインタフェース構成に応じた計測設定をして計測を開始します。外部トリガの場合、計測設定をしてトリガ待ち状態にします。

CTSU_CFG_DTC_SUPPORT_ENABLE が有効になっている場合、DTC 設定もします。

計測結果は INTCTSUFN 割り込みハンドラから発行されるコールバック関数で通知します。

自動判定モード使用時は、オフセットチューニング完了後、最初にこの関数が呼ばれたときはベースライン設定をします。

Example

```
fsp_err_t err;

/* Initiate a sensor scan by software trigger */
err = R_CTSU_ScanStart(&g_ctsu_ctrl);

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

Special Notes:

なし

3.3 R_CTSU_DataGet

この関数は、指定したタッチインタフェース構成の前回計測した全ての補正計測値を読み込みます。

Format

```
fsp_err_t R_CTSU_DataGet (ctsu_ctrl_t * const p_ctrl, uint16_t * p_data)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ
p_data : [out] 補正計測値を格納するバッファへのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*
FSP_ERR_CTSU_SCANNING	*/* スキャン中 */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/* イニシャルオフセットチューニング中 */*
FSP_ERR_ABORTED	*/* ADC を使用したデータ収集でエラー発生 */*
FSP_ERR_CTSU_DIAG_NOT_YET	*/* 診断用データの収集が完了していません */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、前回計測した全ての補正計測値を指定されたバッファ p_data に読み込みます。
CTSU1：センサ CCO 補正、移動平均の処理をした値
CTSU2L JMM：センサ CCO 補正、移動平均の処理をした値
CTSU2L VMM：センサ CCO 補正、マルチクロック補正、移動平均をした値

計測モードにより必要なバッファサイズが異なります。自己容量計測モードと電流計測モードでは使用する TS 端子数、相互容量計測モードではマトリックス数の 2 倍を用意してください。CTSU2 の JMM の場合は、3 周波数のデータが格納されるため、さらに 3 倍を用意してください。

イニシャルオフセットチューニングが完了するまでの数回は FSP_ERR_CTSU_INCOMPLETE_TUNING を返します。このときはバッファに補正計測値を格納しません。

診断モードの場合、データ収集が未完了の場合は FSP_ERR_CTSU_DIAG_NOT_YET を返します。データ収集が完了の場合は FSP_SUCCESS を返しますので、R_CTSU_Diagnosis()をコールして診断結果の確認をしてください。また、出力電圧診断で ADC の計測でエラーが出た場合は FSP_ERR_ABORTED を返します。

Example:

```
fsp_err_t err;
uint16_t buf[CTSU_CFG_NUM_SELF_ELEMENTS];

/* Get all sensor values */
err = R_CTSU_DataGet(&g_ctsu_ctrl, buf);
```

Special Notes:

なし

3.4 R_CTSU_CallbackSet

この関数は、計測完了コールバック関数に指定した関数を設定します。

Format

```
fsp_err_t R_CTSU_CallbackSet (ctsu_ctrl_t * const p_api_ctrl,
                              void (* p_callback)(ctsu_callback_args_t *),
                              void const * const p_context,
                              ctsu_callback_args_t * const p_callback_memory)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

p_callback : [in] コールバック関数ポインタ

p_context : [in] コールバック関数の引数に送るポインタ

p_callback_memory : [in] NULL を設定してください

Return Values

FSP_SUCCESS　*/* 成功しました */*

FSP_ERR_ASSERTION　*/* 引数のポインタが指定されていません */*

FSP_ERR_NOT_OPEN　*/* Open()のコールなしにコールされました */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、計測完了コールバック関数に指定した関数を設定します。デフォルトでは、コールバック関数は ctsu_cfg_t のメンバ p_callback の関数が設定されているので、動作中に別の関数に変更したい場合に使用してください。

また、コンテキストポインタも設定可能です。使用しない場合は、p_context に NULL を設定してください。

p_callback_memory は NULL を設定してください。

Example:

```
fsp_err_t err;

/* Set callback function */
err = R_CTSU_CallbackSet(&g_ctsu_ctrl, ctsu_callback, NULL, NULL);
```

Special Notes:

なし

3.5 R_CTSU_Close

この関数は、指定したタッチインタフェース構成を終了します。

Format

```
fsp_err_t R_CTSU_Close (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、指定したタッチインタフェース構成を終了します。

Example:

```
fsp_err_t err;

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

Special Notes:

本 API を診断モードと組み合わせて使用する場合は以下の点に注意してください。

・R_CTSU_ScanStart()を実施後、と R_CTSU_DataGet()をコールするまでの間で本 API をコールしないこと。

3.6 R_CTSU_Diagnosis

この関数は、CTSU の内部回路を診断する機能を提供する API 関数です。

Format

```
fsp_err_t R_CTSU_Diagnosis (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

Return Values

FSP_SUCCESS	*/*すべての診断は正常です。*/*
FSP_ERR_ASSERTION	*/*引数ポインタがありません。*/*
FSP_ERR_NOT_OPEN	*/* Open（）を呼び出さずに呼び出されます。 */*
FSP_ERR_CTSU_NOT_GET_DATA	*/*以前のスキャンデータは処理されませんでした。 */*
FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE	*/*出力電圧診断で異常を検知しました*/*
FSP_ERR_CTSU_DIAG_CCO_HIGH	*/* CCO 上限診断で異常を検知しました。 */*
FSP_ERR_CTSU_DIAG_CCO_LOW	*/* CCO 下限診断で異常を検知しました。*/*
FSP_ERR_CTSU_DIAG_SSCG	*/* SSCG 発振器診断で異常を検知しました。 */*
FSP_ERR_CTSU_DIAG_DAC	*/*センサオフセット診断で異常を検知しました。 */*
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE	*/*出力電圧診断で異常を検知しました。 */*
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE	*/*過電圧検出診断で異常を検知しました。*/*
FSP_ERR_CTSU_DIAG_OVER_CURRENT	*/*過電流検出診断で異常を検知しました。 */*
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE	*/*内部抵抗診断で異常を検知しました。*/*
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE	*/*センサオフセット診断で異常を検知しました。*/*
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN	*/* SENSCLK 周波数診断で異常を検知しました。*/*
FSP_ERR_CTSU_DIAG_SUCLK_GAIN	*/* SUCLK 周波数診断の診断で異常を検知しました。*
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY	*/* SUCLK クロックリカバリ診断で異常を検知しました。*/*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は CTSU の内部回路を診断する機能を提供する API 関数です。R_CTSU_DataGet 関数の戻り値が FSP_SUCCESS のときに呼び出してください。各診断で異常を検知した場合、その診断のエラーを返します。全てが正常の場合は、FSP_SUCCESS を返します。

Example:

```
/* CTSU1 の場合 */
fsp_err_t err;
uint16_t  dummy;

/* Open Diagnosis function */
R_CTSU_Open(g_qe_ctsu_instance_diagnosis.p_ctrl, g_qe_ctsu_instance_diagnosis.p_cfg);

/* Scan Diagnosis function */
R_CTSU_ScanStart(g_qe_ctsu_instance_diagnosis.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;

err = R_CTSU_DataGet(g_qe_ctsu_instance_diagnosis.p_ctrl,&dummy);
if (FSP_SUCCESS == err)
{
    err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
    if ( FSP_SUCCESS  == err )
    {
        /* Diagnosis was succeeded. */
    }
}
```

```
/* CTSU2 の場合 */
void R_CTSU_PinSetInit(void);
void qe_touch_main(void);
uint16_t  dummyD;

void qe_touch_main(void)
{
    fsp_err_t err;
    uint8_t initial_diag_flag = 0;

    /* 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);

    /* Open CTSU driver for [Diagnosis] */
    err = R_CTSU_Open(g_qe_ctsu_instance_diagnosis.p_ctrl, g_qe_ctsu_instance_diagnosis.p_cfg);

    /* Initial Diagnosis loop */
    while (0 == initial_diag_flag)
    {
        /* for [Diagnosis] configuration */
        err = R_CTSU_ScanStart(g_qe_ctsu_instance_diagnosis.p_ctrl);
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = R_CTSU_DataGet(g_qe_ctsu_instance_diagnosis.p_ctrl, &dummyD);
        if (FSP_SUCCESS == err)
        {
            err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
            if (FSP_SUCCESS == err)
            {
                initial_diag_flag = 1;
            } else {
                while(1);
            }
        }
    }

    /* Main loop */
    while (true)
    {
        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        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);

        /* for [Diagnosis] configuration */
        err = R_CTSU_ScanStart(g_qe_ctsu_instance_diagnosis.p_ctrl);
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = R_CTSU_DataGet(g_qe_ctsu_instance_diagnosis.p_ctrl, &dummyD);
        if (FSP_SUCCESS == err)
        {
            err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
            if (FSP_SUCCESS != err)
            {
                while(1);
            }
        }

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

Special Notes:

なし

3.7 R_CTSU_ScanStop

この関数は、指定したタッチインタフェース構成の計測を停止します。

Format

```
fsp_err_t R_CTSU_ScanStop (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、指定したタッチインタフェース構成を停止します。

Example:

```
fsp_err_t err;

/* Stop CTSU module */
err = R_CTSU_ScanStop(&g_ctsu_ctrl);
```

Special Notes:

本 API を診断モードと組み合わせて使用する場合は以下の点に注意してください。

・R_CTSU_ScanStart()を実施後、と R_CTSU_DataGet()をコールするまでの間で本 API をコールしないこと。

3.8 R_CTSU_SpecificDataGet

この関数は、指定したタッチインタフェース構成の指定したデータ種別の計測値または補正計測値を読み込みます。

Format

```
fsp_err_t R_CTSU_SpecificDataGet (ctsu_ctrl_t * const       p_ctrl,
                                  uint16_t                * p_specific_data,
                                  ctsu_specific_data_type_t specific_data_type)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ
p_specific_data : [out] データ種別に応じた計測値または補正計測値を格納するバッファへのポインタ
specific_data_type : [in] 取得するデータ種別

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*
FSP_ERR_CTSU_SCANNING	*/* スキャン中 */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/* イニシャルオフセットチューニング中 */*
FSP_ERR_NOT_ENABLED	*/* 非対応のタイプ指定 */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

specific_data_type に CTSU_SPECIFIC_RAW_DATA を設定した場合、p_specific_data に計測値を格納します。CTSU1 ではエレメント数、CTSU2 ではエレメント数とマルチクロックの数を乗算した数のバッファを用意してください。

specific_data_type に CTSU_SPECIFIC_CORRECTION_DATA を設定した場合、p_specific_data にセンサ CCO 補正データを格納します。CTSU1 ではエレメント数、CTSU2 ではエレメント数とマルチクロックの数を乗算した数のバッファを用意してください。

specific_data_type に CTSU_SPECIFIC_CORRECTION_DATA を設定した場合、p_specific_data にマルチクロック補正データを格納します。CTSU2 の VMM のみ有効です。エレメント数のバッファを用意してください。

specific_data_type に CTSU_SPECIFIC_SELECTED_FREQ を設定した場合、p_specific_data に多数決で使用された周波数のビットマップを格納します。1 番目の周波数がビット 0、2 番目の周波数がビット 1、3 番目の周波数がビット 2 に対応します。例えば、1 番目と 3 番目の周波数が使用されていた場合は 0x05 を格納します。CTSU2 の VMM のみ有効です。

Example:

```
fsp_err_t err;
uint16_t specific_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI]

/* Get Specific Data */
err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, &specific_data[0],
CTSU_SPECIFIC_RAW_DATA);
```

Special Notes:

specific_data_type を CTSU_SPECIFIC_RAW_DATA 以外に設定する場合は、R_CTSU_DataGet()をコールした後に本 API を実行してください。

3.9 R_CTSU_DataInsert

この関数は、指定したタッチインタフェース構成のタッチ計測結果のバッファに指定したデータを格納します。

Format

```
fsp_err_t R_CTSU_DataInsert (ctsu_ctrl_t * const p_ctrl,
                             uint16_t * p_insert_data)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

p_insert_data : [in] 格納するデータのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*
FSP_ERR_CTSU_SCANNING	*/* スキャン中 */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/* イニシャルオフセットチューニング中 */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

ユーザアプリケーションで R_CTSU_SpecificDataGet() で取得したデータにノイズ対策などの処理をして、そのデータを本関数で格納することを想定しています。p_insert_data に格納するデータ配列の先頭アドレスを設定してください。そのデータを補正計測値バッファに格納します。（自己容量計測モードの場合は、p_ctrl->p_self_data、相互容量の場合は、p_ctrl->p_mutual_pri_data および p_crtl->p_mutual_snd_data）

Example:

```
fsp_err_t err;
uint16_t specific_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI]

/* Get Specific Data */
err = R_CTSU_DataGet(&g_ctsu_ctrl, &specific_data[0],
CTSU_SPECIFIC_CORRECTION_DATA);

/* Noise filter process */

/* Insert data */
err = R_CTSU_DataInsert(&g_ctsu_ctrl, &specific_data[0]);
```

Special Notes:

なし

3.10 R_CTSU_OffsetTuning

この関数は、指定したタッチインタフェース構成のオフセットレジスタ（SO）を調整します。

Format

```
fsp_err_t R_CTSU_OffsetTuning (ctsu_ctrl_t * const p_ctrl);
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ

Return Values

FSP_SUCCESS	*/* 成功しました */*
FSP_ERR_ASSERTION	*/* 引数のポインタが指定されていません */*
FSP_ERR_NOT_OPEN	*/* Open()のコールなしにコールされました */*
FSP_ERR_CTSU_SCANNING	*/* スキャン中 */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/* イニシャルオフセットチューニング中 */*

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、前回計測した全ての補正計測値を使用してオフセット調整を行います。計測が完了した後にコールしてください。この関数を一度実行すると調整が完了するまでは FSP_ERR_CTSU_INCOMPLETE_TUNING を返します。オフセット調整が完了すると FSP_SUCCESS を返します。オフセット調整が完了するまで計測と本関数コールを繰り返してください。オフセット調整については、1.5.4 章を参照してください。

自動判定が有効の場合、オフセット調整完了後にベースライン初期化ビットフラグを設定します。

Example:

```
fsp_err_t err;
err = R_CTSU_ScanStart (g_qe_ctsu_instance_config01.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;
err = R_CTSU_OffsetTuning (g_qe_ctsu_instance_config01.p_ctrl);
```

Special Notes:

なし

3.11 R_CTSU_AutoJudgementDataGet

この関数は、指定したタッチインタフェース構成の自動判定ボタンの結果を取得します。

Format

```
fsp_err_t R_CTSU_AutoJudgementDataGet (ctsu_ctrl_t * const p_ctrl,
                                        uint64_t * p_button_status)
```

Parameters

p_ctrl : [in] コントロール構造体へのポインタ
p_button_status : [out] ボタン状態を格納するバッファへのポインタ

Return Values

FSP_SUCCESS /* 成功しました */
FSP_ERR_ASSERTION /* 引数のポインタが指定されていません */
FSP_ERR_NOT_OPEN /* Open()のコールなしにコールされました */
FSP_ERR_CTSU_SCANNING /* スキャン中 */
FSP_ERR_INVALID_MODE /* 自動判定オフのモードは無効です。*/

Properties

r_ctsu_api.h にプロトタイプ宣言されています。

Description

この関数は、自動判定ボタンの結果を取得します。 計測が完了した後に呼び出します。結果は 64 ビットビットマップであり、指定されたタッチインタフェース構成の TS 番号の順に格納されます。

オフセットチューニング完了後に初めてこの関数が呼ばれたとき、ベースライン平均値演算を開始する設定を行います。

Example:

```
fsp_err_t err;
uint64_t button_status;

/* Open CTSU Driver */
err = R_CTSU_Open (&g_ctsu_ctrl, &g_ctsu_cfg);

/* Initial Offset Tuning */
while (true)
{
    err = R_CTSU_ScanStart (&g_ctsu_ctrl);
    while (0 == g_qe_touch_flag) {}
    g_qe_touch_flag = 0;

    err = R_CTSU_OffsetTuning (&g_ctsu_ctrl);
}

/* Main loop */
while (true)
{
    /* for [CONFIG01] configuration */
    err = R_CTSU_ScanStart (&g_ctsu_ctrl);
    while (0 == g_qe_touch_flag) {}
    g_qe_touch_flag = 0;


    /* Get all sensor values */
    err = R_CTSU_AutoJudgementDataGet(&g_ctsu_ctrl, &button_status);
}
```

Special Notes:

この関数は CTSU2SL のみに対応します。

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
1.11	2019/12/11	3~6	「基準値」の定義を追加しました。 スキャンの完了後にカスタムコールバック関数が 2 回呼び出される問題を修正しました。
1.20	2021/2/1	1	対象デバイスに RX671 を追加
		10	バージョンを更新
		12	コードサイズを更新
		12、19	API の戻り値を更新
2.00	2021/7/30	-	全面改訂
2.01	2021/12/17	3,4	1.1.4 イニシャルオフセットチューニングに内容追記
		5	1.1.6 マルチ計測周波数(CTSU2L)に内容追記
		6	1.1.7 シールド機能(CTSU2L)について内容追記
		9	1.2.4 温度補正モード(CTSU2L)に内容追記
		11	1.4 API 概要追記
		12	2.2 ソフトウェア要求に DTC 使用時 Heap size の記述を更新
		15	2.8 コードサイズの内容修正
		16~19	2.9 引数の構造体更新
		30	3.6 R_CTSU_Diagnosis に内容追記
		33~34	3.8 R_CTSU_ SpecificDataGet を新規作成
		35~36	3.9 R_CTSU_DataInsert を新規作成
2.10	2022/4/20	3	1.概要に内容追記
		7	1.1.11 MEC 機能(CTSU2SL)を追加
		7	1.1.12 自動判定機能(CTSU2SL)を追加
		7,8,9	1.1.13 自動機能(CTSU2SL)を追加
		16	2.7 コンパイル時の設定に内容追記
		19,20	2.9 引数に内容追記
		39	3.10 R_CTSU_OffsetTuning を追加
		40	3.11 R_CTSU_AutoJudgementDataGet を追加
2.20	2022/12/28	3	1 概要を更新
		7	1.1.10 診断機能に追記
		12	1.2.1 自己容量モードの図を差し替え
		14	1.2.4 温度補正モード(CTSU2L)に追記
		16	2.2 ソフトウェアの要求を更新
		16	2.3 サポートされているツールチェーンを更新
		19	2.8 コードサイズを更新
		20	2.9 引数を更新
		24	2.10 戻り値を更新

		29	3.3 R_CTSU_DataGet を更新
3.00	2024/10/15	1,3	対象デバイスに RX260/RX261 を追加
		4	1.1.4 デフォルトのターゲット値(CTSU2L)を更新
		5~7	1.5.2 マルチクロック計測(CTSU2L)を追記、図 **1.16** を更新
		10	1.5.10 自動周波数補正機能(CTSU2SLa)を追加
		16	2.1 ハードウェアの要求を更新
		16	2.2 ソフトウェアの要求を更新
		16	2.3 サポートされているツールチェーンを更新
		19	2.7 コンパイル時の設定を更新
		20	2.8 コードサイズを更新
		21	2.9 引数を更新
		30	3.3 R_CTSU_DataGet の Description を更新
		36	3.8 R_CTSU_SpecificDataGet の Description を更新
3.10	2025/2/19	15	1.5.1 ランダムパルス周波数計測(CTSU1)の説明を修正
		8	1.4.4 自動判定モード(CTSU2SL)の説明を修正
		24	2.2 ソフトウェアの要求を更新
		24	2.3 サポートされているツールチェーンを更新
		25	2.7 コンパイル時の設定を更新
		27	2.8 コードサイズを更新
		28	2.9 引数を更新
3.11	2025/3/21	-	コードファイルの Disclaimer コメントの変更による更新。 この APN の内容に変更はありません。
3.20	2025/7/31	23	1.1 に説明を追加
		24	2.2 ソフトウェアの要求を更新
		24	2.3 サポートされているツールチェーンを更新
		27	2.8 コードサイズを更新
		34	2.11 コールバック関数 を追加
		37~48	3 API 関数 の引数に入出力の情報を追加
3.30	2026/3/24	3	1.1 CTSU モジュールとは を追加
		3	1.2 CTSU モジュールの位置づけ を追加
		4	1.3 CTSU モジュールの使い方を 追加
		23	API 概要を 1.7 へ移動
		8~9	1.4.4 自動判定モード(CTSU2SL)を追加
		12~14	1.4.5 診断モードを追加
		15	1.5 機能を並び替えて整理
		21~23	1.6 他の FIT モジュールとの連携を追加
		-	温度補正モード(CTSU2L)の章を削除
		24	2.2 ソフトウェアの要求を更新
		24	2.3 サポートされているツールチェーンを更新
		27	2.8 コードサイズを更新
		35~37	2.12 API の処理時間を追加
		-	IEC60730 準拠の章を削除
		39	3.3 R_CTSU_DataGet 戻り値と診断モードの説明を追記
		42	3.6 R_CTSU_Diagnosis モジュールのアップデートに合わせて説明を追加
		-	sdpx ファイルと json ファイルをパッケージに同梱

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、ノイズ輻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

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。

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/

© 2026 Renesas Electronics Corporation. All rights reserved.