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

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

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

CTSU モジュールは Touch モジュール向けの CTSU ドライバです。CTSU モジュールは Touch ミドルウェアレイヤからのアクセスを想定していますが、ユーザアプリケーションからもアクセスできます。

CTSU ペリフェラルは CTSU,CTSU2L,CTSU2SL の 3 種類のバージョンがあります。それぞれ機能が異なります。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。

CTSU2SL：RX140-256KB, RX140-128KB

CTSU2L：RX140-64KB

CTSU：RX113, RX130, RX230, RX231, RX23W, RX671

CTSU と CTSU2L、CTSU2SL 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 は下記のように表現します。

・CTSU、CTSU2L、CTSU2SL 共通の説明　→　CTSU

・CTSU のみの説明　→　CTSU1

・CTSU2L と CTSU2SL 共通の説明　→　CTSU2L

・CTSU2SL のみの説明　→ CTSU2SL

1.1 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。

1.1.1 QE for Capacitive Touch との連携

このモジュールはコンフィグレーション設定により様々な静電容量計測を提供します。コンフィグレーション設定は QE for Capacitive Touch によって生成されます。

コンフィグレーション設定の一部であるタッチインタフェース構成は、計測する端子（以下、TS）の組み合わせとその計測モードを表します。複数のタッチインタフェース構成が必要となる場合は、製品内で異なる計測モードの組み合わせが存在するときや、アクティブシールド機能を使用するときです。

1.1.2 計測とデータ取得

計測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（ELCL）で起動された外部イベントのいずれかによって開始できます。

計測処理は、CTSU ペリフェラルが処理するため、メインプロセッサの処理時間を消費しません。

計測中に発生する INTCTSUWR と INTCTSURD は本モジュールが処理します。これらの処理に対して DTC を使用することも可能です。

計測完了割り込みである INTCTSUFN の処理が完了したときにコールバック関数でアプリケーションに通知します。計測が完了したら内部処理も実行するために、次の計測までに計測結果を取得してください。

API 関数の R_CTSU_ScanStart()で計測を開始できます。

API 関数の R_CTSU_DataGet()で計測結果を取得できます。

1.1.3 センサ ICO 補正

CTSU ペリフェラルはセンサ ICO 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、一時的に補正処理に移行します。補正処理では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得られた計測値に対して補正計算をします。

CTSU2L の温度補正が有効な場合は、TS 端子に接続された外部抵抗を使用して、定期的に補正係数を更新します。温度依存の無い外部抵抗を基準とすることで、センサ ICO の温度ドリフトに対しても補正することができます。

1.1.4　初期オフセット調整

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 ICO 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計測値がターゲット値になるように自動調整します。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	Default target value
Self-capacitance	15360 (37.5%)
Self-capacitance using active shield	6144 (15%)
Mutual-capacitance	10240 (20%)

パーセンテージは CCO の入力制限によるものです。100%は計測値 40960 です。デフォルトのターゲット値は、526us（CTSU1）または 256us（CTSU2）にです。計測時間を変更すると、基準時間との比率で目標値が調整されます。

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 は 3 の推奨値であり、CSTUPRMODE は 2 の推奨値です。CTSUPRRATIO および CTSUPRMODE を推奨値から変更する場合の計測時間についてはハードウェアマニュアルを参照してください。

・CTSU2（自己容量モード）

Target value	Target value (multi frequency)	CTSUSNUM	Measurement time
7680	15360 (128us + 128us)	0x7	128usec
15360	30720 (256us + 256us)	0xF	256usec
3840	7680 (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 を 0.5MHz 以外に設定する場合の計測時間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。

1.1.5　ランダムパルス周波数計測(CTSU1)

CTSU1 ペリフェラルは 1 つのドライブ周波数で計測します。

ドライブ周波数は電極への電流量を決めるもので、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。

ドライブ周波数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。例えば、PCLK が 32MHz、CTSUCLK で 1/2 周期を選択、CTSUSDPA で 16 分周を選択するとドライブ周波数は 0.5MHz となります。なお、CTSUSDPA は TS 毎に変更する事が可能です。

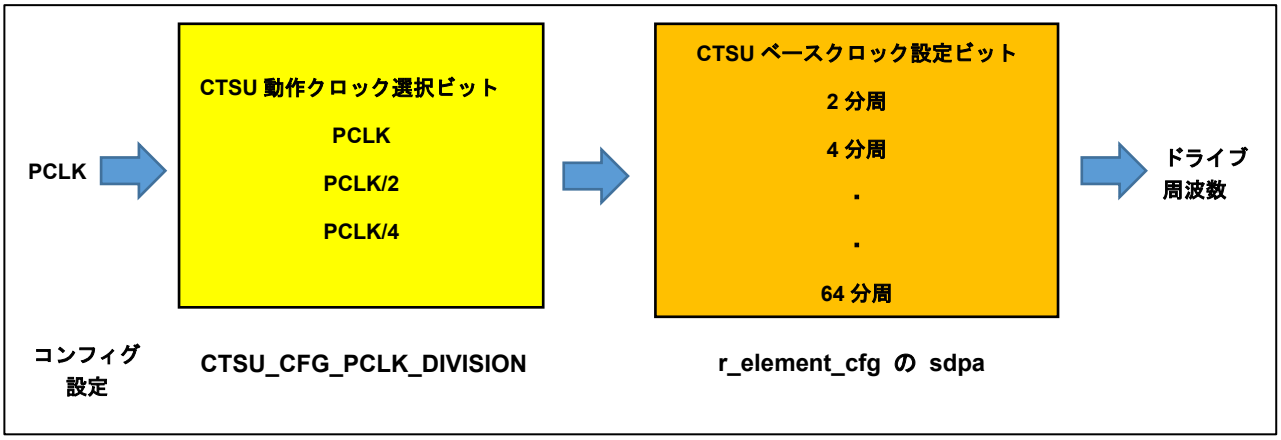

図 1　ドライブ周波数設定

実際のドライブパルスは外部環境ノイズ対策として、ドライブ周波数をベースとした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になります。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

1.1.6　マルチ周波数計測(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 種類のドライブ周波数で計測できます。

このモジュールはデフォルトでは多数決判定が可能な 3 種類の周波数で計測し、得られた 3 周波数での結果を第 1 周波数基準値に標準化および多数決判定して計測値を確定します。

また、多数決判定前のデータも取得できます。このデータを使用して独自のノイズフィルタ処理をすることも可能です。処理したデータをモジュールのバッファに書き戻せば 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。

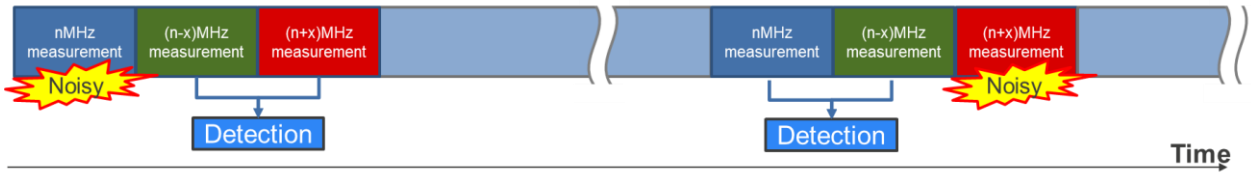

図 2　マルチ周波数計測

ドライブ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 種類のドライブ周波数を設定します。

ドライブ周波数は下記計算式となります。

(PCLKB 周波数 / CLK / STCLK) x SUMULTIn / 2 / SDPA　　:　n = 0, 1, 2

以下に PCLKB 周波数が 32MHz のときにドライブ周波数 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。

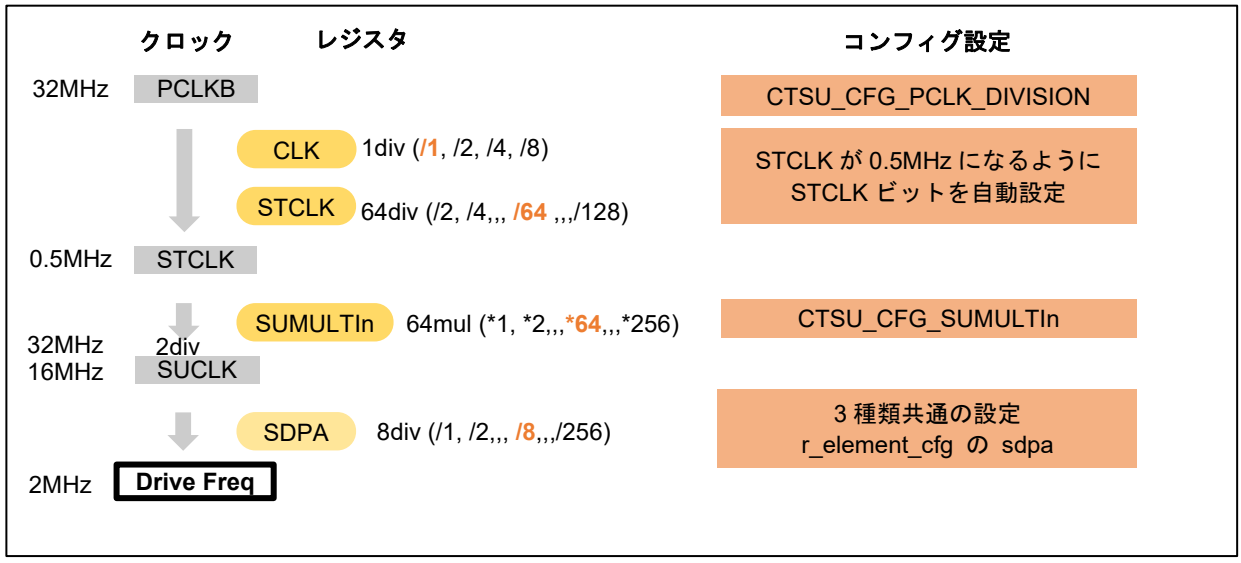

図 3 ドライブ周波数設定

1.1.7 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 4 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,
```

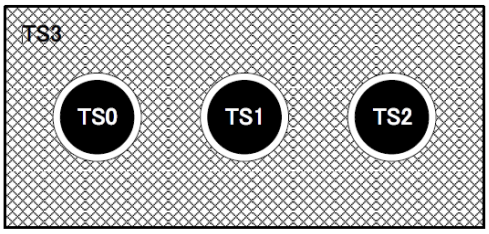

図 4 シールド電極構成例

1.1.8 計測エラー通知

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。

1.1.9　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。

1.1.10　診断機能

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。この診断機能は、内部回路が正常に動作することを診断するための API として提供します。

診断内容は、CTSU1 と CTSU2L で異なり、CTSU1 用に 5 種類、CTSU2L 用に 9 種類の診断を提供します。

診断機能は、API 関数を呼び出すことによって実行できます。これは通常の測定とは別に独立して実行されますので、通常計測に影響を与える事はありません。

診断機能を有効にするには、CTSU_CFG_DIAG_SUPPORT_ENABLE を 1 に設定します。

CTSU1 の場合、外部に 27pF のコンデンサ接続が必要になります。診断機能の計測後に通常のタッチ計測を行う場合は、約 1ms の待ち時間後にタッチ計測を開始してください。

CTSU2L の場合、ADC FIT (r_s12ad_rx)を使用します。診断機能で使用する ADC モジュールでエラーが発生した場合、R_CTSU_DataGet 関数の戻り値は FSP_ERR_ABORTED が返ります。ADC モジュールエラーについては、ADC FIT (r_s12ad_rx)を参照してください。

以下の点について注意して CTSU2L の診断機能を使用して下さい。

1.　CTSU2L の診断機能を使用する場合、必ず ADC を使用した測定をします。その為、ADC FIT をアプリケーション上でも使用する際は診断機能を使用する前に、ADC FIT を必ずクローズして下さい。

2.　上記 1.でクローズを行わなかった場合、FSP_ERR_ABORTED が発生します。下記サンプルを参考に ADC FIT をクローズして、次の診断機能実行時に CTSU ドライバ内の ADC 計測が実施出来るようにして下さい。

```
        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)
        {
            diag_err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
            if ( FSP_SUCCESS == diag_err )
            {
                /* TODO: Add your own code here. */
            }
        }
        else if (FSP_ERR_ABORTED == err)
        {
            adc_err = R_ADC_Close(0);
            if (ADC_SUCCESS != adc_err)
            {
                while (true) {}
            }
        }
```

3.　RTOS アプリケーションを作成する場合、CTSU モジュールの診断機能のタスクと ADC モジュールのタスクのスケジューリングに注意してください。

1.1.11　MEC 機能(CTSU2SL)

CTSU2SL ペリフェラルは複数の電極を接続し 1 つの電極として計測する MEC（Multi Electrode Connection)機能を持っています。この機能は自己容量モードのみ使用可能です。

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 5 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。

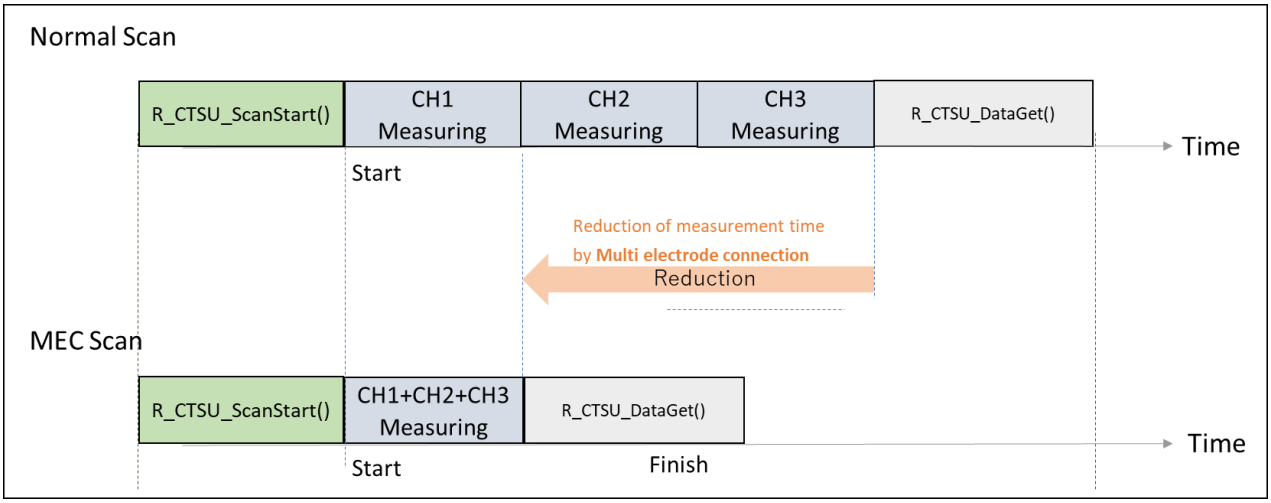

図 5 通常計測と MEC 機能を使用した計測の計測時間比較

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 機能を使用する場合は、同じ TS 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 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 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 6 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,
```

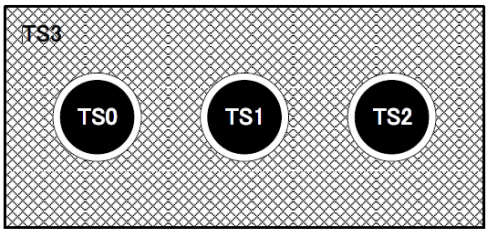

図 6 MEC＋シールド電極構成例

1.1.12　自動補正機能(CTSU2SL)

CTSU2SL ペリフェラルはハードウェアでセンサ ICO 補正する自動補正機能を持っています。センサ ICO 補正についての詳細は、1.1.3 章を参照してください。

CTSU2SL ペリフェラルが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。

1.1.13　自動判定機能(CTSU2SL)

CTSU2SL ペリフェラルはハードウェアでボタンのタッチ判定をする自動判定機能を持っています。

CTSU2SL ペリフェラルがボタンのタッチ判定を処理するため、メインプロセッサの処理時間を消費しません。

計測と判定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 (ELCL) で起動された外部イベントのいずれかによって開始できます。API 関数の R_CTSU_ScanStart()を使用してください。

計測中に発生する INTCTSUWR と INTCTSURD は本モジュールが処理します。これらの処理に対して DTC を使用するため、DTC が必須となります。

INTCTSUFN の処理が完了したときにコールバック関数でアプリケーションに通知します。次の計測までに判定結果を取得してください。API 関数の R_CTSU_AutoJudgeDataGet()を使用してください。

判定結果はマルチ周波数計測の結果を多数決した結果です。

第 1 周波数	第 2 周波数	第 3 周波数	判定結果
“タッチ”	“タッチ”	“タッチ”	“タッチ”
“タッチ”	“タッチ”	“非タッチ”	“タッチ”
“タッチ”	“非タッチ”	“タッチ”	“タッチ”
“非タッチ”	“タッチ”	“タッチ”	“タッチ”
“タッチ”	“非タッチ”	“非タッチ”	“非タッチ”
“非タッチ”	“タッチ”	“非タッチ”	“非タッチ”
“非タッチ”	“非タッチ”	“タッチ”	“非タッチ”
“非タッチ”	“非タッチ”	“非タッチ”	“非タッチ”

自動判定機能を有効にするには CTSU_CFG_AUTO_JUDGE_ENABLE = 1 を設定して下さい。

(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。

マルチ周波数計測のそれぞれに(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

非タッチ状態の計測結果からベースラインを設定します。R_CTSU_OffsetTuning()で初期オフセット調整を完了後、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周囲環境変化に追従します。設定した計測回数で”非タッチ”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。”タッチ”判定するとクリアします。

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) 閾値

ベースラインから任意のオフセットをつけた閾値を利用してタッチ判定をします。

閾値はヒステリシスを付与して設定します。”タッチ”から”非タッチ”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”から”非タッチ”に遷移しにくくなるので注意してください。

ctsu_auto_button_cfg_t の threshold と hysteresis でボタン毎の閾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閾値と下側閾値を計算して、CTSUAJTHR レジスタに設定します。

図 7 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、上側閾値を超えた際に”タッチ”判定します。

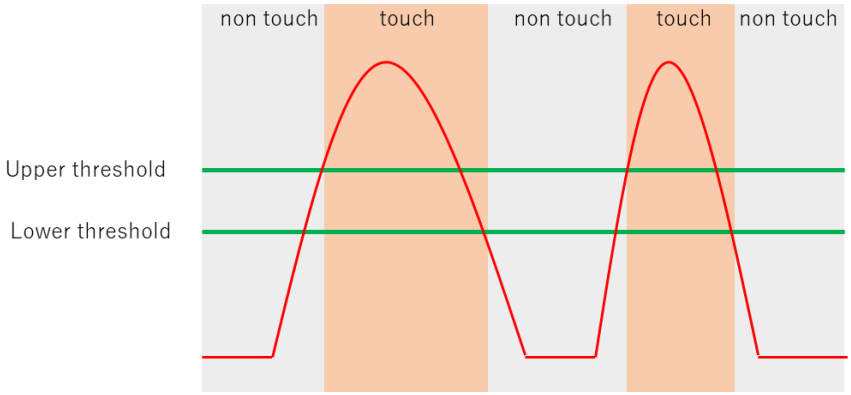

図 7 自己容量の判定

図 8 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、下側閾値を超えた際に”タッチ”判定されます。

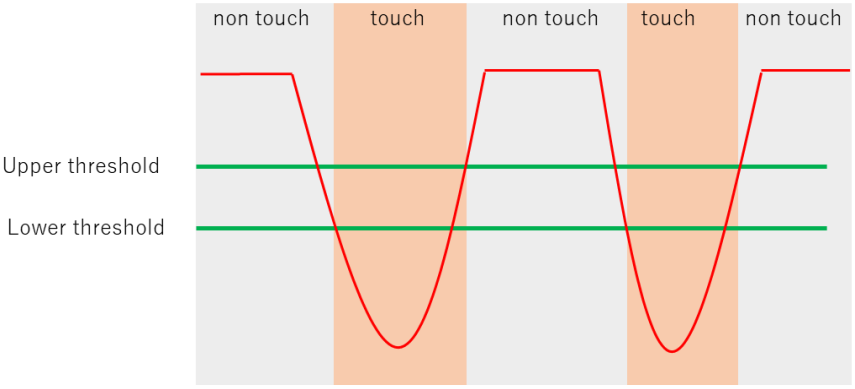

図 8 相互容量の判定

(d) 非タッチ判定連続回数 / タッチ判定連続回数

“タッチ”または”非タッチ”状態が一定回数継続した時に”タッチ”または”非タッチ”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。連続回数を増やす程チャタリング対策には効果がありますが、反応速度が低下するので注意してください。

(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**図 9** に示します。

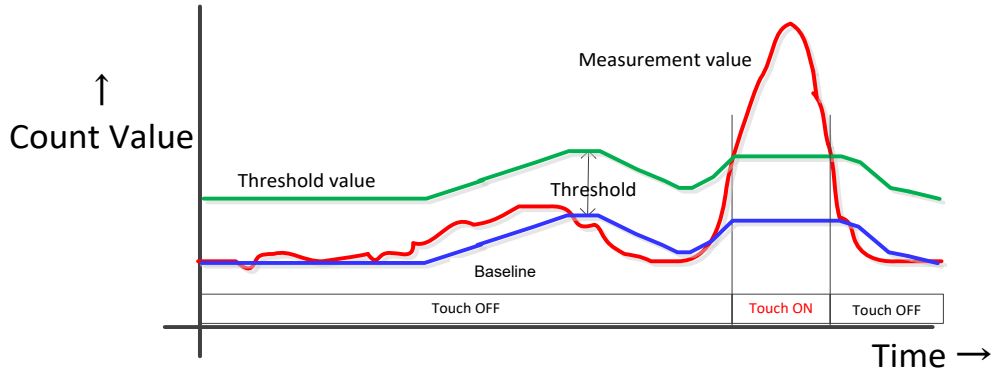

図 9 ボタンの判定

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。

```
.tlot = 2,          // 非タッチ判定連続回数＝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
.mtucfen = 1,       // 相互容量演算許可
.ajfen = 1,         // 自動判定有効
```

1.2　計測モード

このモジュールは、CTSU ペリフェラルが提供する自己容量、相互容量、CTSU2L ペリフェラルはそれに加えて電流計測の各計測モードに対応しています。また、CTSU2L は温度補正モードとして補正係数を更新するモードを提供します。

1.2.1　自己容量モード

自己容量モードでは、各 TS の静電容量を計測します。

CTSU ペリフェラルは TS 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 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。

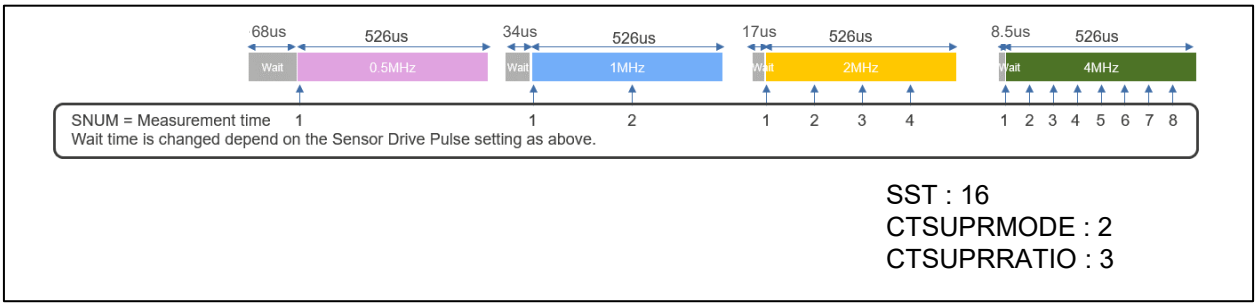

図 10 自己容量計測時間(CTSU1)

[CTSU2L]

1 つの TS の計測時間はデフォルト設定の場合、約 576us です。

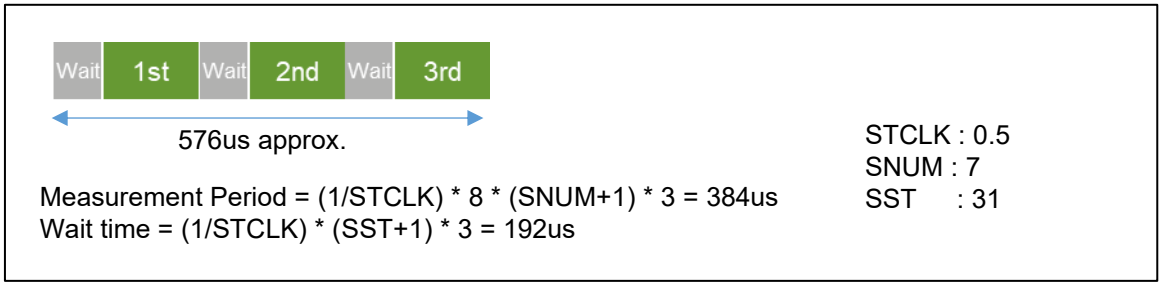

図 11 自己容量計測時間(CTSU2L)

1.2.2　相互容量モード

相互容量モードでは、受信 TS（Rx）と送信 TS（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 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 回目計測値と 1 回目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。このモジュールは差分計算をせずに 2 回の計測値を出力します。

[CTSU1]

1 つの TS 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。

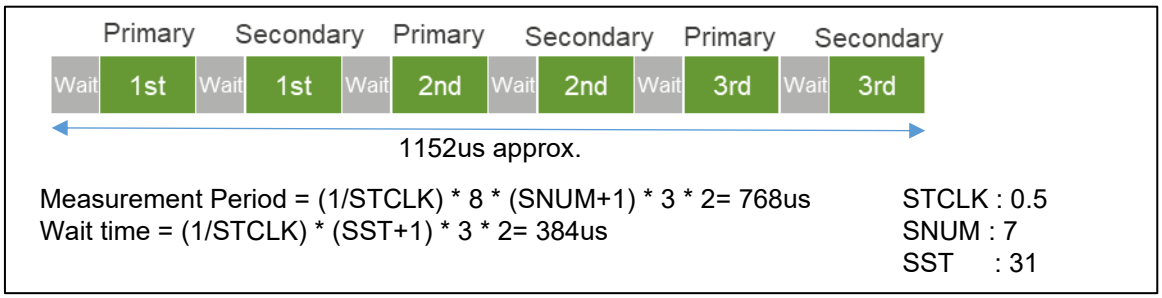

図 12 相互容量計測時間(CTSU2L)

1.2.3 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 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。

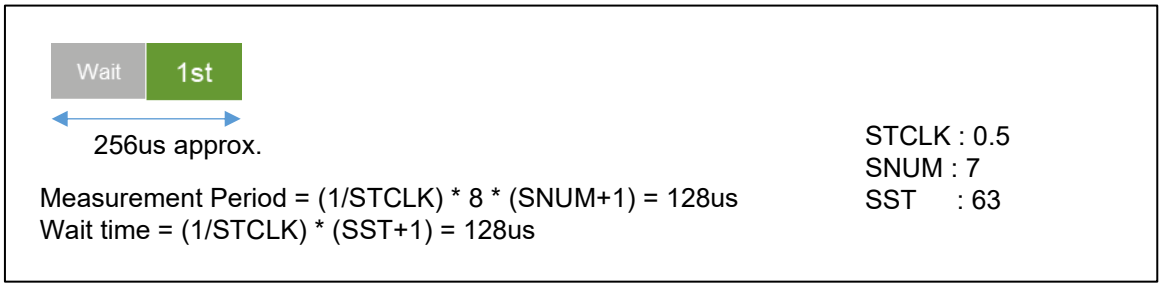

図 13 電流計測時間

1.2.4 温度補正モード(CTSU2L)

CTSU2L の温度補正モードでは、TS 端子に接続された外部抵抗を使用して、定期的に補正係数を更新します。これには 3 種類の処理があります。図 14 のタイミングチャートを参照してください。

1. 補正用回路の計測をします。12 回の計測で 1 セットになります。

2. 外部抵抗に TSCAP 電圧を印加したときの電流を計測して、温度依存性の無い外部抵抗を基準とした補正係数を作成します。1 の計測が 1 セット完了した後の計測で実行します。

3. 外部抵抗にオフセット電流を流して ADC で電圧を計測することで RTRIM レジスタを調整し、内部基準抵抗の温度ドリフトに対応します。2 を何回実行したときに実行するかの回数をコンフィグで設定できます。

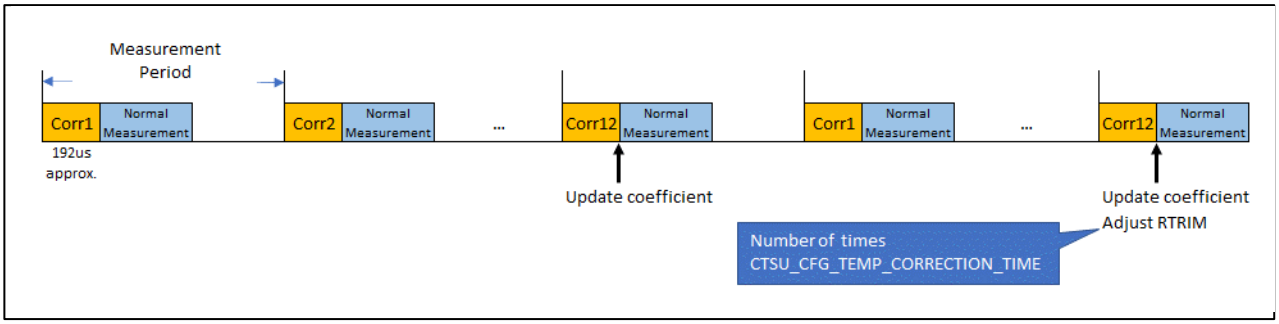

図 14 温度補正計測タイミングチャート

温度補正モードでは ADC FIT (r_s12ad_rx)を使用しています。温度補正モードで使用する ADC モジュールでエラーが発生した場合、R_CTSU_DataGet 関数の戻り値は FSP_ERR_ABORTED が返ります。ADC モジュールエラーについては、ADC FIT (r_s12ad_rx)を参照してください。

以下の点について特に注意してください。

1. CTSU2L の温度補正モードを使用する場合、必ず ADC を使用した測定をします。その為、ADC FIT をアプリケーション上でも使用する際は温度補正を使用する前に、ADC FIT を必ずクローズして下さい。

2. 上記 1.でクローズを行わなかった場合、FSP_ERR_ABORTED が発生します。下記サンプルを参考に ADC FIT をクローズして、次の温度補正実行時に CTSU ドライバ内の ADC 計測が実施出来るようにして下さい。

```
R_CTSU_ScanStart(g_qe_ctsu_instance_temp_correction.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;

err = R_CTSU_DataGet(g_qe_ctsu_instance_temp_correction.p_ctrl, &dummyD);
if (FSP_SUCCESS == err)
{
    /* TODO: Add your own code here. */
}
else if (FSP_ERR_ABORTED == err)
{
    adc_err = R_ADC_Close(0);
    if (ADC_SUCCESS != adc_err)
    {
        while (true) {}
    }
}
```

3. RTOS アプリケーションを作成する場合、CTSU モジュールの温度補正のタスクと ADC モジュールのタスクのスケジューリングに注意してください。

1.2.5　診断モード

診断モードは、この診断機能を使って様々な内部計測値をスキャンするモードです。

詳細については、1.1.10 項を参照してください。

1.3　計測タイミング

1.1.1 章で説明した通り、計測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できます。

一般的な使用方法は、タイマを用いて定期的に計測します。タイマの間隔はすべての計測と内部値の更新が完了できるように設定してください。計測時間はタッチインタフェース構成と計測モードによって異なります。1.2 章を参照してください。

ソフトウェアトリガと外部トリガには、微小な実行タイミングの差異があります。

ソフトウェアトリガは R_CTSU_ScanStart()でタッチインタフェース構成に対する設定をしてから開始フラグをセットするので、タイマイベント発生から若干の遅延が発生します。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、こ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推奨となります。外部トリガで複数のタッチインタフェース構成の場合は、一つの計測が完了したときに R_CTSU_ScanStart()で別のタッチインタフェース構成の設定をしてください。

1.4　API 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()	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。

- CTSU1
- CTSU2L
- CTSU2SL

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モジュール (r_bsp) v6.10 以降

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 v3.80 以降（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_v4.90 以降 （温度補正モードまたは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 V3.1.0 以降

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 v3.04.00
- IAR RX Toolchain v4.20.3
- GCC RX Toolchain v 8.3.0.202202

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、このドライバの使用と競合する可能性があります。
CTSU_CFG_AUTO_JUDGE_ENABLE	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) ー 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) ー 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) ー 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。	
CTSU_CFG_NUM_SELF_ELEMENTS	自己容量、電流計測、温度補正の合計 TS 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_LOW_VOLTAGE_MODE	Low Voltage モードを有効にするか選択します。この値が CTSU1 は CTSUCR1 レジスタの CTSUATUNE0、CTSU2L は CTSUCRAL レジスタの ATUNE0 ビットに設定されます。
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	マルチ周波数計測の数を設定します。

CTSU_CFG_SUMULTI0	マルチ周波数計測の 1 つ目の周波数の逓倍率を設定します。 0x3F が推奨値です。
CTSU_CFG_SUMULTI1	マルチ周波数計測の 2 つ目の周波数の逓倍率を設定します。 0x36 が推奨値です。
CTSU_CFG_SUMULTI2	マルチ周波数計測の 3 つ目の周波数の逓倍率を設定します。 0x48 が推奨値です。
CTSU_CFG_TEMP_CORRECTION_SUPPORT	温度補正を有効にするか選択します。
CTSU_CFG_TEMP_CORRECTION_TS	温度補正端子の番号を設定します。
CTSU_CFG_TEMP_CORRECTION_TIME	温度補正の補正係数更新間隔を設定します。温度補正モードでの計測 13 回を 1 セットとして、何セット実行毎に更新するかの回数を設定します。
CTSU_CFG_CALIB_RTRIM_SUPPORT	温度補正の RTRIM 補正を有効にするか選択します。 本設定を有効にして動作するには、ADC の設定が必要となります。
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	自動補正処理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。

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。

[CTSU1]

ROM、RAM のコードサイズ 自己容量 1 エレメント	
CTSU_CFG_PARAM_CHECKING_ENABLE 0 CTSU_CFG_DTC_SUPPORT_ENABLE 0	ROM: 2569 bytes RAM: 61 bytes

ROM、RAM のコードサイズ 各モードとエレメント追加での増加分				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	2569 bytes	+16 bytes	2836 bytes	+12 bytes
RAM	61 bytes	+21 bytes	71 bytes	+31 bytes

[CTSU2L]

ROM、RAM のコードサイズ ボタン 1 固定でコンフィグレーションオプション変更	
CTSU_CFG_PARAM_CHECKING_ENABLE 0 CTSU_CFG_DTC_SUPPORT_ENABLE 0	ROM: 4199 bytes RAM: 195 bytes

ROM、RAM のコードサイズ 各モードとエレメント追加での増加分 CTSU_CFG_PARAM_CHECKING_ENABLE 0、CTSU_CFG_DTC_SUPPORT_ENABLE 0				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	4199 bytes	+16 bytes	4335 bytes	+22 bytes
RAM	195 bytes	+35 bytes	211 bytes	+51 bytes

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ともに r_ctsu_qe.h、r_ctsu_qe_api.h に定義されています。

タッチインタフェース構成のコントロール構造体です。アプリケーションからの設定は不要です。
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、このモジュールの API の第一引数に設定してください。

```
typedef struct st_ctsu_instance_ctrl
{
    uint32_t                open;               ///< Whether or not driver is open.
    volatile ctsu_state_t   state;              ///< CTSU run state.
    ctsu_cap_t              cap;                ///< CTSU Scan Start Trigger Select
    ctsu_md_t               md;                 ///< CTSU Measurement Mode Select(copy to cfg)
    ctsu_tuning_t           tuning;             ///< CTSU Initial offset tuning status.
    uint16_t                num_elements;       ///< Number of elements to scan
    uint16_t                wr_index;           ///< Word index into ctsuwr register array.
    uint16_t                rd_index;           ///< Word index into scan data buffer.
#if (BSP_FEATURE_CTSU_VERSION == 2)
    uint8_t * p_frequency_complete_flag;        ///< Pointer to complete flag of each frequency. g_ctsu_frequency_complete_flag[] is set by Open API.
#endif
    int32_t               * p_tuning_diff;      ///< Pointer to difference from base value of each element. g_ctsu_tuning_diff[] is set by Open API.
    uint16_t                average;            ///< CTSU Moving average counter.
    uint16_t                num_moving_average; ///< Copy from config by Open API.
    uint8_t                 ctsucr1;            ///< Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW, and PON is set by HAL driver.
    ctsu_ctsuwr_t         * p_ctsuwr;           ///< CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API.
    ctsu_self_buf_t       * p_self_raw;         ///< Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API.
    uint16_t              * p_self_corr;        ///< Pointer to Self correction data. g_ctsu_self_corr[] is set by Open API.
    ctsu_data_t           * p_self_data;        ///< Pointer to Self moving average data. g_ctsu_self_data[] is set by Open API.
    ctsu_mutual_buf_t     * p_mutual_raw;       ///< Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open API.
    uint16_t              * p_mutual_pri_corr;  ///< Pointer to Mutual primary correction data. g_ctsu_self_corr[] is set by Open API.
    uint16_t              * p_mutual_snd_corr;  ///< Pointer to Mutual secondary correction data. g_ctsu_self_corr[] is set by Open API.
    ctsu_data_t           * p_mutual_pri_data;  ///< Pointer to Mutual primary moving average data. g_ctsu_mutual_pri_data[] is set by Open API.
    ctsu_data_t           * p_mutual_snd_data;  ///< Pointer to Mutual secondary moving average data. g_ctsu_mutual_snd_data[] is set by Open API.
    ctsu_correction_info_t * p_correction_info; ///< Pointer to correction info
    ctsu_txvsel_t           txvsel;             ///< CTSU Transmission Power Supply Select
    ctsu_txvsel2_t          txvsel2;            ///< CTSU Transmission Power Supply Select 2 (CTSU2 Only)
    uint8_t                 ctsuchac0;          ///< TS00-TS07 enable mask
    uint8_t                 ctsuchac1;          ///< TS08-TS15 enable mask
    uint8_t                 ctsuchac2;          ///< TS16-TS23 enable mask
    uint8_t                 ctsuchac3;          ///< TS24-TS31 enable mask
    uint8_t                 ctsuchac4;          ///< TS32-TS39 enable mask
    uint8_t                 ctsuchtrc0;         ///< TS00-TS07 mutual-tx mask
    uint8_t                 ctsuchtrc1;         ///< TS08-TS15 mutual-tx mask
    uint8_t                 ctsuchtrc2;         ///< TS16-TS23 mutual-tx mask
    uint8_t                 ctsuchtrc3;         ///< TS24-TS31 mutual-tx mask
    uint8_t                 ctsuchtrc4;         ///< TS32-TS39 mutual-tx mask
    uint16_t                self_elem_index;    ///< self element index number for Current instance.
    uint16_t                mutual_elem_index;  ///< mutual element index number for Current instance.
    uint16_t                ctsu_elem_index;    ///< CTSU element index number for Current instance.
#if (BSP_FEATURE_CTSU_VERSION == 2)
    uint8_t * p_selected_freq_self;             ///< Frequency selected by self-capacity
    uint8_t * p_selected_freq_mutual;           ///< Frequency selected by mutual-capacity
#endif
#if (BSP_FEATURE_CTSU_VERSION == 1)
 #if (CTSU_CFG_DIAG_SUPPORT_ENABLE == 1)
    ctsu_diag_info_t * p_diag_info;             ///< pointer to diagnosis info
 #endif
#endif

#if (BSP_FEATURE_CTSU_VERSION == 2)
```

```
    ctsu_range_t range;                         ///< According to atune12. (20uA : 0, 40uA : 1, 80uA : 2, 160uA : 3)
    uint8_t      ctsucr2;                       ///< Copy from (posel, atune1, md) by Open API. FCMODE and SDPSEL and LOAD is set by HAL driver.
 #if (CTSU_CFG_NUM_CFC != 0)
    uint64_t             cfc_rx_bitmap;         ///< Bitmap of CFC receive terminal.
    ctsu_corrcfc_info_t * p_corrcfc_info;       ///< pointer to CFC correction info
 #endif
 #if (CTSU_CFG_DIAG_SUPPORT_ENABLE == 1)
    ctsu_diag_info_t * p_diag_info;             ///< pointer to diagnosis info
 #endif
#endif
#if (CTSU_CFG_AUTO_JUDGE_ENABLE == 1)
    ctsu_auto_judge_t * p_auto_judge;            ///< Array of automatic judgement register write variables. g_ctsu_auto_judge[] is set by Open API.
    uint32_t            adress_auto_judge;       ///< Automatic judgement Variable start address
    uint32_t            adress_ctsuwr;           ///< CTSUWR variable start address for automatic judgement
    uint32_t            adress_self_raw;         ///< Self raw variable start address for automatic judgement
    uint32_t            adress_mutual_raw;       ///< Mutual raw variable start address for automatic judgement
    uint32_t            count_auto_judge;        ///< Automatic judgement transfer count
    uint32_t            count_ctsuwr_self_mutual;///< CTSUWR, Self raw, Mutual raw transfer count for automatic judgement
    uint8_t             blini_flag;              ///< Flags for controlling baseline initialization bit for automatic judgement
    uint8_t             ajmmat;                  ///< Copy from config by Open API for automatic judgement
    uint8_t             ajbmat;                  ///< Copy from config by Open  for automatic judgement
 #endif
#endif
    ctsu_cfg_t const * p_ctsu_cfg;              ///< Pointer to initial configurations.
    void (* p_callback)(ctsu_callback_args_t *);///< Callback provided when a CTSUFN occurs.
    uint8_t               interrupt_reverse_flag;///< Flag in which read interrupt and end interrupt are reversed
    ctsu_event_t          error_status;         ///< error status variable to send to QE for serial tuning.
    ctsu_callback_args_t * p_callback_memory;   ///< Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
    void const           * p_context;           ///< Placeholder for user data.
    bool     serial_tuning_enable;              ///< Flag of serial tuning status.
    uint16_t serial_tuning_mutual_cnt;          ///< Word index into ctsuwr register array.
    uint16_t tuning_self_target_value;          ///< Target self value for initial offset tuning
    uint16_t tuning_mutual_target_value;        ///< Target mutual value for initial offset tuning
    uint8_t                   tsod;             ///< Copy from tsod by Open API.
    uint8_t                   mec_ts;           ///< Copy from mec_ts by Open API.
    uint8_t                   mec_shield_ts;    ///< Copy from mec_shield_ts by Open API.
} ctsu_instance_ctrl_t;
```

タッチインタフェース構成のコンフィグレーション設定の構造体です。

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および初期設定値が 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。

```
typedef struct st_ctsu_cfg
{
    ctsu_cap_t                cap;                  ///< CTSU Scan Start Trigger Select
    ctsu_txvsel_t             txvsel;               ///< CTSU Transmission Power Supply Select
    ctsu_txvsel2_t            txvsel2;              ///< CTSU Transmission Power Supply Select 2 (CTSU2 Only)
    ctsu_atune1_t             atune1;               ///< CTSU Power Supply Capacity Adjustment (CTSU Only)
    ctsu_atune12_t            atune12;              ///< CTSU Power Supply Capacity Adjustment (CTSU2 Only)
    ctsu_md_t                 md;                   ///< CTSU Measurement Mode Select
    ctsu_posel_t              posel;                ///< CTSU Non-Measured Channel Output Select (CTSU2 Only)
    uint8_t                   tsod;                 ///< TS all terminal output control for multi electrode scan
    uint8_t                   mec_ts;               ///< TS number used when using the MEC function
    uint8_t                   mec_shield_ts;        ///< TS number of active shield used when using MEC function
    uint8_t                   tlot;                 ///< Number of consecutive judgements exceeding the threshold L for automatic judgement
    uint8_t                   thot;                 ///< Number of consecutive judgements exceeding the threshold H for automatic judgement
    uint8_t                   jc;                   ///< judgement condition for automatic judgement
    uint8_t                   ajmmat;               ///< Measured value moving average number of times for automatic judgement
    uint8_t                   ajbmat;               ///< Average number of baselines for automatic judgement
    uint8_t                   mtucfen;              ///< Mutual capacity operation for automatic judgement
    uint8_t                   ajfen;                ///< Automatic judgement function enabled for automatic judgement
    uint8_t                   autojudge_monitor_num;///< Method number for QE monitor for automatic judgement
    uint8_t                   ctsuchac0;            ///< TS00-TS07 enable mask
    uint8_t                   ctsuchac1;            ///< TS08-TS15 enable mask
    uint8_t                   ctsuchac2;            ///< TS16-TS23 enable mask
    uint8_t                   ctsuchac3;            ///< TS24-TS31 enable mask
    uint8_t                   ctsuchac4;            ///< TS32-TS39 enable mask
    uint8_t                   ctsuchtrc0;           ///< TS00-TS07 mutual-tx mask
    uint8_t                   ctsuchtrc1;           ///< TS08-TS15 mutual-tx mask
```

```
    uint8_t                  ctsuchtrc2;             ///< TS16-TS23 mutual-tx mask
    uint8_t                  ctsuchtrc3;             ///< TS24-TS31 mutual-tx mask
    uint8_t                  ctsuchtrc4;             ///< TS32-TS39 mutual-tx mask
    ctsu_element_cfg_t const * p_elements;           ///< Pointer to elements configuration array
    uint8_t                  num_rx;                 ///< Number of receive terminals
    uint8_t                  num_tx;                 ///< Number of transmit terminals
    uint16_t                 num_moving_average;     ///< Number of moving average for measurement data
    bool tunning_enable;                             ///< Initial offset tuning flag
    void (* p_callback)(ctsu_callback_args_t * p_args); ///< Callback provided when CTSUFN ISR occurs.
    void const * p_context;                          ///< User defined context passed into callback function.
    void const * p_extend;                           ///< Pointer to extended configuration by instance of interface.
    uint16_t     tuning_self_target_value;           ///< Target self value for initial offset tuning
    uint16_t     tuning_mutual_target_value;         ///< Target mutual value for initial offset tuning
} ctsu_cfg_t;
```

上記構造体で使用している列挙型を示します。

```
/** 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;
```

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 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 no yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,       ///< Diagnosis of LDO over voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH         = 6005,       ///< Diagnosis of CCO into 19.2uA failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW          = 6006,       ///< Diagnosis of CCO into 2.4uA failed.
    FSP_ERR_CTSU_DIAG_SSCG             = 6007,       ///< Diagnosis of SSCG frequency failed.
    FSP_ERR_CTSU_DIAG_DAC              = 6008,       ///< Diagnosis of non-touch count value failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE   = 6009,       ///< Diagnosis of LDO output voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE     = 6010,       ///< Diagnosis of over voltage detection circuit failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT     = 6011,       ///< Diagnosis of over current detection circuit failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE  = 6012,       ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE   = 6013,       ///< Diagnosis of Current source value failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN     = 6014,       ///< Diagnosis of SENSCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN       = 6015,       ///< Diagnosis of SUCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY   = 6016,       ///< Diagnosis of SUCLK clock recovery function failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN         = 6017,       ///< Diagnosis of CFC oscillator gain failed.
} fsp_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。

2.12　IEC 60730 準拠

安全規格である IEC 60335-1 と IEC60730 クラス B の両方に準拠しています。
認証の状況などの最新情報は、Web ページ（家電向け機能安全ソリューション (IEC/UL60730)｜Renesas）に公開しています。

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_cfg

コンフィグレーション構造体へのポインタ (通常は QE for Capacitive Touch によって生成)

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_qe.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 120ms 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。

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

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_qe.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。

自動判定機能使用時は、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計測設定の初期化が行われます。

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_data

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	*/* 初期オフセット調整中 */*
FSP_ERR_ABORTED	*/* ADC を使用したデータ収集でエラー発生 */*

Properties

r_ctsu_qe.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計測値を指定されたバッファに読み込みます。計測モードにより必要なバッファサイズが異なります。自己容量モードと電流計測モードでは使用する TS 数、相互容量モードではマトリックス数の 2 倍を用意してください。

温度補正モードでは計測値を格納しません。RTRIM 調整をした場合には、RTRIM 値を格納します。このときは本関数内で ADC の設定を変更しているので、使用している ADC 設定に戻す処理をしてください。それ以外の場合は 0xFFFF を格納します。

初期オフセット調整がオンの場合は、調整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計測値を格納しません。初期オフセット調整については、1.1.6 章を参照してください。

計測値は、この関数内でセンサ ICO 補正、多数決判定（オンの場合）、移動平均の処理をした値となります。

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_api_ctrl

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_callback

コールバック関数ポインタ

p_context

コールバック関数の引数に送るポインタ

p_callback_memory

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_qe.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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

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_qe.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:

なし

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

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	*/ * LDO 過電圧の診断に失敗しました* / *
FSP_ERR_CTSU_DIAG_CCO_HIGH	*/ *19.2uA への CCO の診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_CCO_LOW	*/ *2.4uA への CCO の診断に失敗しました。* / *
FSP_ERR_CTSU_DIAG_SSCG	*/ * SSCG 頻度の診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_DAC	*/ *非タッチカウント値の診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE	*/ * LDO 出力電圧の診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE	*/ *過電圧検出回路の診断に失敗しました。* / *
FSP_ERR_CTSU_DIAG_OVER_CURRENT	*/ *過電流検出回路の診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE	*/ * LDO 内部抵抗値の診断に失敗しました。* / *
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE	*/ *現在のソース値の診断に失敗しました。* / *
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN	*/ *SENSCLK 周波数ゲインの診断に失敗しました。 * / *
FSP_ERR_CTSU_DIAG_SUCLK_GAIN	*/ *SUCLK 周波数ゲインの診断に失敗しました。
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY	*/ *SUCLK クロックリカバリ機能の診断に失敗しました。* / *

Properties

r_ctsu_qe.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。

Example:

```
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 succssed. */
   }
}
```

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

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_qe.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:

なし

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_specific_data

データ種別に応じた計測値を格納するバッファへのポインタ

specific_data_type

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  /* 初期オフセット調整中 */
FSP_ERR_NOT_ENABLED         /* CTSU_SPECIFIC_SELECTED_FREQ を CTSU1 で指定 */
```

Properties

r_ctsu_qe.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 RAW データを格納します。1.1.3 のセンサ ICO 補正の計算前のデータとなります。

specific_data_type に CTSU_SPECIFIC_CORRECTION_DATA を設定した場合、p_specific_data に補正済データを格納します。1.1.3 のセンサ ICO 補正の計算後のデータとなります。

これらは CTSU2 ではチャネル数とマルチ周波数の数を乗算した数のデータを格納します。

specific_data_type に CTSU_SPECIFIC_SELECTED_DATA 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。CTSU2 のみ有効です。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。

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_CORRECTION_DATA);
```

Special Notes:

なし

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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_insert_data

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	*/* 初期オフセット調整中 */*

Properties

r_ctsu_qe.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。自己容量モードの場合は、p_ctrl->p_self_data に格納します。相互容量の場合は、p_ctrl->p_mutual_pri_data および p_crtl->p_mutual_snd_data に格納します。

Example:

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

/* Get Specific Data */
err = R_CTSU_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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

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	*/* 初期オフセット調整中 */*

Properties

r_ctsu_qe.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、1.1.4 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

コントロール構造体へのポインタ(通常は QE for Capacitive Touch によって生成)

p_button_status

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_qe.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;
 uint16_t buf[CTSU_CFG_NUM_SELF_ELEMENTS];

 /* Open Touch middleware */
 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, btn_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 初期オフセット調整に内容追記
		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_SpecifivDataGet を新規作成
		35~36	3.9 R_CTSU_DataInset 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

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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