

RL78 ファミリ

CTSU モジュール Software Integration System

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

RL78/G23 グループ

関連ドキュメント

RL78 ファミリ TOUCH モジュール (R11AN0485)

目次

1. 概要	3
1.1 機能	3
1.1.1 QE for Capacitive Touch との連携	3
1.1.2 計測とデータ取得	3
1.1.3 センサ ICO 補正	3
1.1.4 初期オフセット調整	3
1.1.5 マルチ周波数計測	4
1.1.6 シールド機能	5
1.1.7 計測エラー通知	5
1.1.8 移動平均	5
1.1.9 診断機能	6
1.1.10 SMS を使用した自動判定計測	6
1.2 計測モード	7
1.2.1 自己容量モード	7
1.2.2 相互容量モード	7
1.2.3 電流計測モード	7
1.2.4 温度補正モード	8
1.2.5 診断モード	8
1.3 計測タイミング	9
1.4 API 概要	9
2. API 情報	10
2.1 ハードウェアの要求	10
2.2 ソフトウェアの要求	10
2.3 サポートされているツールチェーン	10
2.4 制限事項	10
2.5 ヘッダファイル	10
2.6 整数型	10
2.7 コンパイル時の設定	11
2.8 コードサイズ	13
2.9 引数	13
2.10 戻り値	16
3. API 関数	17
3.1 R_CTSU_Open	17
3.2 R_CTSU_ScanStart	19
3.3 R_CTSU_DataGet	20
3.4 R_CTSU_CallbackSet	22
3.5 R_CTSU_SmsSet	23
3.6 R_CTSU_Close	25
3.7 R_CTSU_Diagnosis	26
3.8 R_CTSU_ScanStop	28
3.9 R_CTSU_SpecificDataGet	29
3.10 R_CTSU_DataInsert	31
3.11 R_CTSU_OffsetTuning	33

1. 概要

CTSU モジュールは Touch モジュール向けの CTSU2L 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。

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、CTSU2L 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

CTSU2L 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。

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

CTSU2L 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。`cts_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。

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 (CTSUS1) または 256us (CTSUS2) 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

- ・ CTSUS2 (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 マルチ周波数計測

CTSUS2L 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 [3.9 章](#)と [3.10 章](#)を参照してください。



図 1 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。

$$(f_{CLK} \text{ 周波数} / CLK / STCLK) \times SUMMULTIn / 2 / SDPA \quad : n = 0, 1, 2$$

以下に f_{CLK} 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。

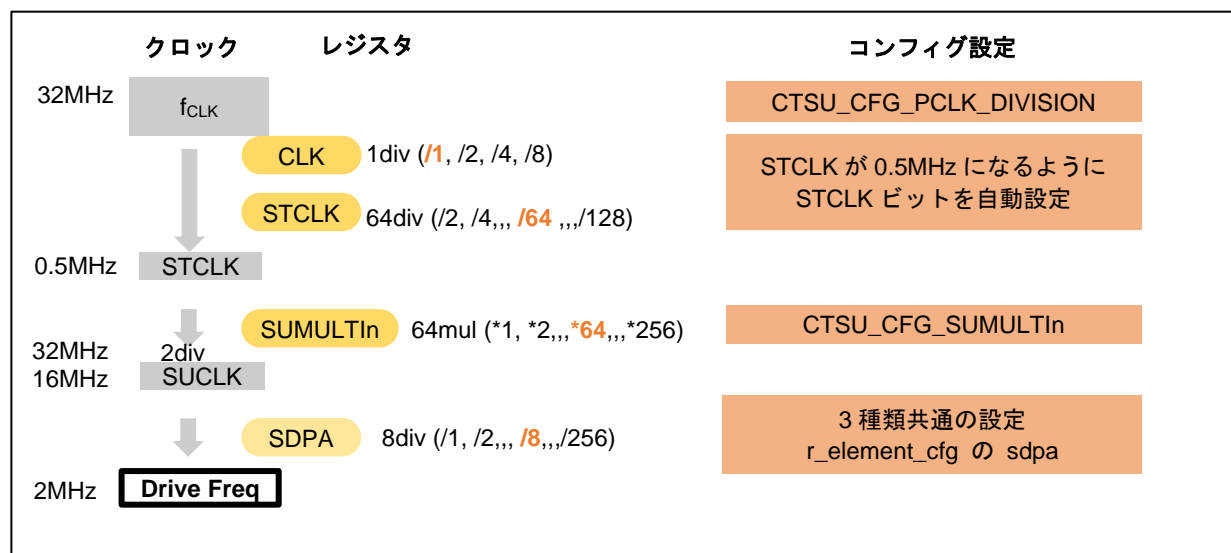


図 2 ドライブ周波数設定

1.1.6 シールド機能

CTS2L 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 3 のような電極構成の場合、以下に示すように `cts_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,
.pose1     = CTSU_POSEL_SAME_PULSE,
.ctsuchac0 = 0x0F,
.ctsuchtrc0 = 0x08,
```

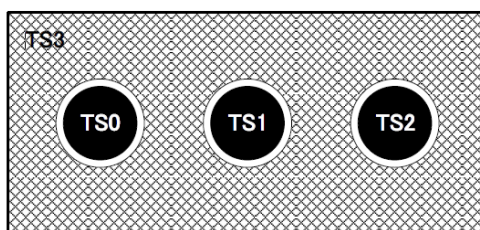


図 3 シールド電極構成例

1.1.7 計測エラー通知

CTS2L 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 `cts_callback_args_t` のメンバの `event` を参照してください。

1.1.8 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。

1.1.9 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、内部回路を診断するための API を提供します。

診断内容は、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。

CTSU2L の場合、ADC を使用します。

1.1.10 SMS を使用した自動判定計測

SMS を使用して CPU 動作無しに計測からタッチ判定まで動作させる機能です。CPU が STOP モードと SNOOZE モードのみで動作するため、低消費電力で計測できます。

外部トリガ設定かつ DTC 転送設定のみサポートします。外部トリガには f_{SXP} を選択した 32 ビット・インターバル・タイマを使用してください。

この機能を使用したいタッチインタフェースに対して、R_CTSU_SmsSet() をコールした後に R_CTSU_ScanStart() で計測を開始してください。初期オフセット調整完了後に実行することを推奨します。

CTSU ペリフェラルが外部トリガで計測して結果をリードする度に SMS は R_CTSU_DataGet() 相当の処理およびタッチ判定処理をします。

タッチ ON 判定すると INTSMSE 割り込みを発生して、通常計測と同様のコールバック関数をコールして SMS 計測設定を解除します。そのときにアプリケーションは、通常動作と同様に R_CTSU_DataGet() をコールすることで計測結果を取得することができます。

本機能を使用しているときはシステムの他処理で SMS を使用することはできません。

本機能を有効にするには、外部トリガでの計測設定と CTSU_CFG_DTC_SUPPORT_ENABLE を 1 に設定かつ CTSU_CFG_SMS_SUPPORT_ENABLE を 1 に設定します。DTC リピート転送を使用しているため、リピート・エリアに指定している変数の下位 8 ビットを 00H とする必要があります。そのため、CTSU_CFG_SMS_TRANSFER_ADDRESS と CTSU_CFG_SMS_CTSUWR_ADDRESS には RAM 領域のアドレスかつ下位 8 ビットを 00H となるアドレスを設定してください。CTSU_CFG_SMS_TRANSFER_ADDRESS に置かれる変数は 544 バイトを使用します。CTSU_CFG_SMS_CTSUWR_ADDRESS に置かれる変数は (4 * element 数 * マルチ周波数の数) を使用します。例えば、自己容量 3 ボタンで 3 周波数計測の場合は、36 バイトを使用します。

QE for Capacitive Touch でのチューニングを行うためには、CTSU_CFG_SMS_TRANSFER_ADDRESS は 0xFE00~0xFC800 以外の値、CTSU_CFG_SMS_CTSUWR_ADDRESS は 0xFF200~0xFCB00 以外の値に設定してください。

1.2 計測モード

このモジュールは、CTS2L ペリフェラルが提供する自己容量、相互容量、電流計測の各計測モードに対応しています。また、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。

CTS2L 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 の順番で使いたい場合でも、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。

1 つの TS の計測時間はデフォルト設定の場合、約 576us です。

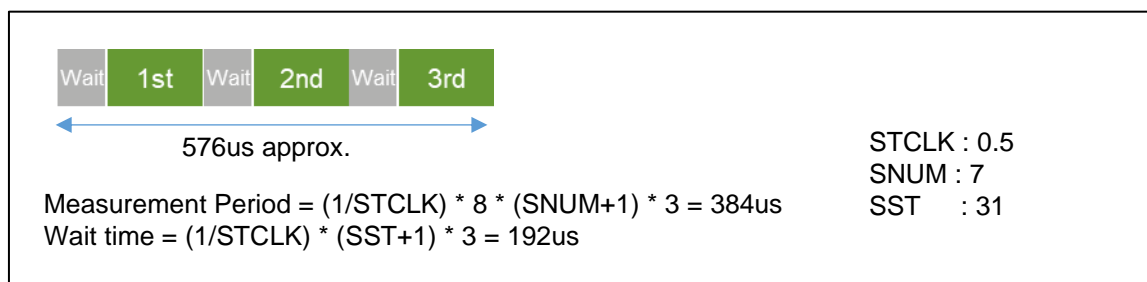


図 4 自己容量計測時間

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。

CTS2L 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、CTS2L 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。そのため、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。

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。

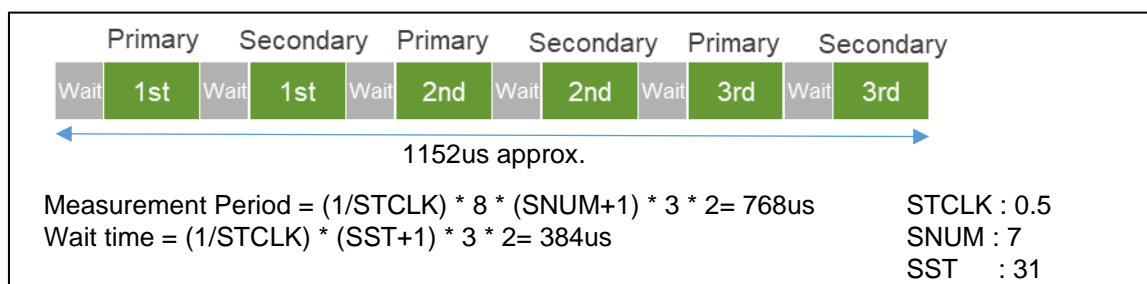


図 5 相互容量計測時間

1.2.3 電流計測モード

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。

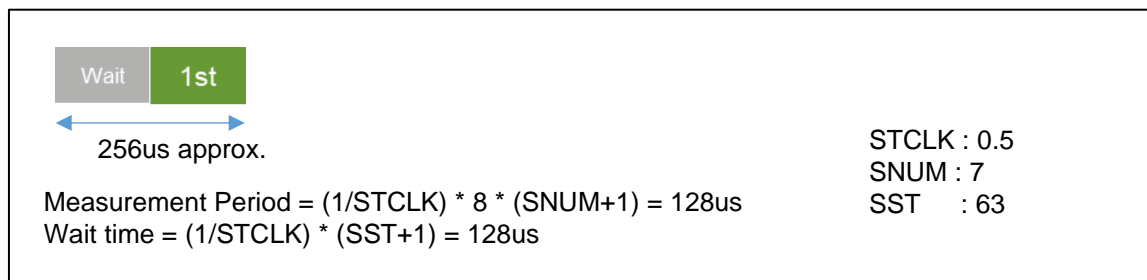


図 6 電流計測時間

1.2.4 温度補正モード

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7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。

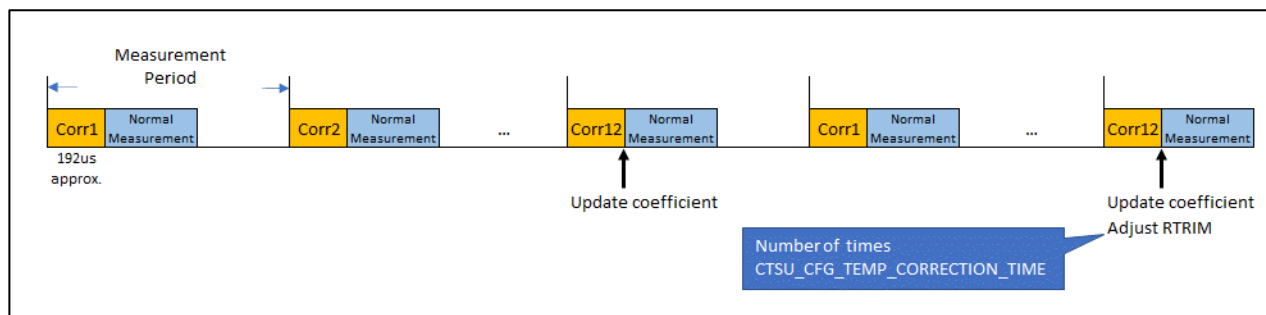


図 7 温度補正計測タイミングチャート

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、1.1.9 項を参照してください。

1.3 計測タイミング

1.1.2 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 (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。

一般的な使用法は、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_StartScan()	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_SmsSet()	指定したタッチインタフェースに対して SMS を使用した自動判定計測するための設定をします。
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_StartStop()	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。

2. API 情報

本モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- CTSU2L

2.2 ソフトウェアの要求

このドライバは以下のモジュールに依存しています。

- ボードサポートパッケージ (r_bsp) v1.13 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。

- コード生成 DTC v1.00 以降

また、静電容量式タッチセンサ開発支援ツールの使用を想定しています。

- QE for Capacitive Touch V2.0.0 以降 (V3.0.2 以降推奨)

2.3 サポートされているツールチェーン

本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-RL Toolchain v1.11.00
- IAR Embedded Workbench for Renesas RL78 v4.21.1
- LLVM for Renesas RL78 v10.0.202203

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_api.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_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_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_DTC_USE_SC ※ デフォルト値は “0”	DTC を使用する場合に、Smart Configurator の DTC 設定を使用するかどうかを選択します。 “0” の場合、CTSU モジュール内部での DTC 設定を使用します。 “1” の場合、Smart Configurator での DTC 設定を使用します。 CTSUWR を 22 番、CTSURD を 23 番に割り当てて、ノーマルモード、16 ビット転送を設定してください。
CTSU_CFG_SMS_SUPPORT_ENABLE ※ デフォルト値は “0”	SMS を使用した自動判定計測の機能を有効にするか選択します。
CTSU_CFG_SMS_TRANSFER_ADDRESS ※ デフォルト値は “0xFF800”	DTC リピート転送で使用するリピート・エリアのアドレス設定です。1.1.10 章を参照してください。
CTSU_CFG_SMS_CTSUWR_ADDRESS ※ デフォルト値は “0xFFB00”	DTC リピート転送で使用するリピート・エリアのアドレス設定です。1.1.10 章を参照してください。
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 (高) – 3 (低) 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 (高) – 3 (低) 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 (高) – 3 (低) になります。
以下のコンフィギュレーションは、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_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 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 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 つ目の周波数の通倍率を設定します。 0x3F が推奨値です。
CTSU_CFG_SUMULTI1	マルチ周波数計測の 2 つ目の周波数の通倍率を設定します。 0x36 が推奨値です。
CTSU_CFG_SUMULTI2	マルチ周波数計測の 3 つ目の周波数の通倍率を設定します。 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。

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 CC-RL 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。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

ROM、RAM のコードサイズ ボタン 1 固定でコンフィギュレーションオプション変更

CTSU_CFG_PARAM_CHECKING_ENABLE 0	ROM: 5501 bytes
CTSU_CFG_DTC_SUPPORT_ENABLE 0	RAM: 282 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	4806 bytes	+28 bytes	5183 bytes	+17 bytes
RAM	274 bytes	+34 bytes	290 bytes	+52 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_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_md_t         md;             ///< CTSU Measurement Mode Select(copy from 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.
    uint8_t           * p_tuning_count; ///< Pointer to tuning count of each element. g_ctsu_tuning_count[]
is set by Open API.
    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           ctsucrl;        ///< 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.
    uint16_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_mutual_pri_corr[] is set by Open API.
```

```

uint16_t          * p_mutual_snd_corr; ///< Pointer to Mutual secondary correction data.
g_ctsu_mutual_snd_corr[] is set by Open API.
uint16_t          * p_mutual_pri_data; ///< Pointer to Mutual primary moving average data.
g_ctsu_mutual_pri_data[] is set by Open API.
uint16_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
uint16_t           mutual_elem_index;      ///< Mutual element index
uint16_t           ctsu_elem_index;       ///< CTSU element index
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. FGMODE and SDPSEL and
LOAD is set by HAL driver.
uint8_t            sms;                   ///< Whether or not SMS use.
#if (CTSU_CFG_DIAG_SUPPORT_ENABLE == 1)
ctsu_diag_info_t   * p_diag_info;         ///< pointer to diagnosis info
#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.
ctsu_event_t        error_status;         ///< Error status variable.
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
} 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
    ctsu_atune12_t      atune12;           ///< CTSU Power Supply Capacity Adjustment
    ctsu_md_t           md;                ///< CTSU Measurement Mode Select
    ctsu_posel_t        posel;             ///< CTSU Non-Measured Channel Output Select
    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
    uint8_t             tuning_enable;      ///< Initial offset tuning flag
    uint8_t             judge_multifreq_disable; ///< Disable to judge multi frequency
    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 (CTSUS2 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 (CTSUS2 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_CURRENT_SCAN    = 9,    ///< Current scan mode
    CTSU_MODE_CORRECTION_SCAN = 17,    ///< Correction scan mode
} ctsu_md_t;

/** CTSU Non-Measured Channel Output Select (CTSUS2 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
} ctsu_posel_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;

/** Element Configuration */
typedef struct st_ctsu_element
{
    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、API 関数のプロトタイプ宣言とともに fsp_common_api.h で記載されています。

```

/* Return error codes */
typedef enum e_fsp_err
{
    FSP_SUCCESS,
    FSP_ERR_ASSERTION,          ///< A critical assertion has failed
    FSP_ERR_INVALID_POINTER,    ///< Pointer points to invalid memory location
    FSP_ERR_INVALID_ARGUMENT,   ///< Invalid input parameter
    FSP_ERR_NOT_OPEN,           ///< Requested channel is not configured or API not open
    FSP_ERR_ALREADY_OPEN,       ///< Requested channel is already open in a different configuration
    FSP_ERR_NOT_ENABLED,        ///< Requested operation is not enabled
    FSP_ERR_INVALID_STATE,      ///< API or command not valid in the current state
    FSP_ERR_CTSU_SCANNING,       ///< Scanning.
    FSP_ERR_CTSU_NOT_GET_DATA,   ///< Not processed previous scan data.
    FSP_ERR_CTSU_INCOMPLETE_TUNING, ///< Incomplete initial offset tuning.
    FSP_ERR_CTSU_DIAG_NOT_YET,   ///< Diagnosis of data collected no yet.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE, ///< Diagnosis of LDO output voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE, ///< Diagnosis of over voltage detection circuit failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT, ///< Diagnosis of over current detection circuit failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE, ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE, ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN, ///< Diagnosis of SENSCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN,  ///< Diagnosis of SUCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY, ///< Diagnosis of SUCLK clock recovery function failed.
} fsp_err_t;

```

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

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 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_USE_DTC 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_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_USE_DTC 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。

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	<i>/* 成功しました */</i>
FSP_ERR_ASSERTION	<i>/* 引数のポインタが指定されていません */</i>
FSP_ERR_NOT_OPEN	<i>/* Open()のコールなしにコールされました */</i>
FSP_ERR_CTSU_SCANNING	<i>/* スキャン中 */</i>
FSP_ERR_CTSU_INCOMPLETE_TUNING	<i>/* 初期オフセット調整中 */</i>

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。多数決判定をオフにしている場合は、それぞれさらに CTSU_CFG_NUM_SUMULTI の倍数分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_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、1.1.5 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_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_SmsSet

この関数は、指定したタッチインタフェースに対して SMS を使用した自動判定計測するための設定をします。

Format

```

fsp_err_t R_CTSU_SmsSet (ctsu_ctrl_t * const p_ctrl,
                        uint16_t * p_threshold,
                        uint16_t * p_hysteresis,
                        uint16_t count filter)

```

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_threshold

タッチ判定閾値のポイント

p context

タッチ判定ヒステリシスのポイント

count filter

タッチ回数一致フィルタの値（上位 8 ビットは OFF 用、下位 8 ビットは ON 用）

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

rcsu_api.h にプロトタイプ宣言されています。

Description

この関数は、下記を設定します。タッチ判定処理に引数設定を使用します。

- ・ CTSUFN 割り込みを禁止
- ・ SMS モジュールを Enable
- ・ SMS 設定
- ・ ELCL 設定
- ・ SMS 開始

自動判定計測を開始するには、本関数コール後に同じタッチインタフェースに対して

R_CTSU_ScanStart()をコールしてください。タッチ ON 判定すると、INTSMSE が発生して CTSU モジュールの割り込みハンドラで下記の設定をします。

- ・計測ステータスをコールバック関数の引数に設定
- ・計測値を変数に設定
- ・コールバック関数コール
- ・CTSUFN 割り込みを許可

- ・ SMS モジュールを Disable

Example:

```
fsp_err_t err;
uint16_t threshold[3] = {1000, 1500, 2000};
uint16_t hysteresis[3] = {50, 75, 100};
uint16_t buf[3];

/* Start SMS measurement */
err = R_CTSU_SmsSet(&g_ctsu_ctrl, threshold, hysteresis[3], 0x0303);
err = R_CTSU_ScanStart(&g_ctsu_ctrl);
__stop();

err = R_CTSU_DataGet(&g_ctsu_ctrl, buf);
```

Special Notes:

低消費電力動作をするために計測トリガとなる 32 ビット・インターバル・タイマの割り込みを許可し、そのタイミングでアプリケーションにて STOP 命令を実行してください。

3.6 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

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

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:

なし

3.7 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_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

3.8 R_CTSU_ScanStop

この関数は、指定されたタッチインタフェース構成の測定を停止します。

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

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

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、指定されたタッチインタフェース構成の測定を停止します。

Example:

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

Special Notes:

なし

3.9 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

<i>FSP_SUCCESS</i>	<i>/* 成功しました */</i>
<i>FSP_ERR_ASSERTION</i>	<i>/* 引数のポインタが指定されていません */</i>
<i>FSP_ERR_NOT_OPEN</i>	<i>/* Open()のコールなしにコールされました */</i>
<i>FSP_ERR_CTSU_SCANNING</i>	<i>/* スキャン中 */</i>
<i>FSP_ERR_CTSU_INCOMPLETE_TUNING</i>	<i>/* 初期オフセット調整中 */</i>
<i>FSP_ERR_NOT_ENABLED</i>	<i>/* CTSU_SPECIFIC_SELECTED_FREQ を CTSU1 で指定 */</i>

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 ではチャン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_cts_ctrl, &specific_data[0],
CTSU_SPECIFIC_CORRECTION_DATA);
```

Special Notes:

なし

3.10 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	<i>/* 成功しました */</i>
FSP_ERR_ASSERTION	<i>/* 引数のポインタが指定されていません */</i>
FSP_ERR_NOT_OPEN	<i>/* Open()のコールなしにコールされました */</i>
FSP_ERR_CTSU_SCANNING	<i>/* スキャン中 */</i>
FSP_ERR_CTSU_INCOMPLETE_TUNING	<i>/* 初期オフセット調整中 */</i>

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_ctrl->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

3.11 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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021/4/13	-	初版発行
1.10	2021/8/31	5	1.1.9 診断機能
		5	1.1.10 SMS を使用した自動判定計測を追加
		8	1.2.5 診断モードを追加
		9	1.4 API 概要を更新
		11	2.7 コンパイル時の設定を更新
		13	2.8 コードサイズを更新
		13	2.9 引数を更新
		16	2.10 戻り値を更新
		-	R_CTSU_VersionGet を削除
		24	3.5 R_CTSU_SmsSet を追加
		27	3.7 R_CTSU_Diagnosis を追加
		29	3.8 R_CTSU_ScanStop を追加
1.11	2022/1/18	3,4	1.1.4 初期オフセット調整に内容追記
		4	1.1.6 マルチ計測周波数(CTSU2L)に内容追記
		9	1.4 API 概要を更新
		10	2.2 ソフトウェアの要求 2.3 サポートされているツールチェーン を更新
		13	2.8 コードサイズを更新
		13~14	2.9 引数の構造体更新
		30~31	3.8 R_CTSU_SpecificDataGet を追加
		32~33	3.9 R_CTSU_DataInsert を追加
1.20	2022/4/20	6	1.1.10 SMS を使用した自動判定計測に説明を追記
		4,5	PCLKb を f _{CLK} に修正
		9	1.4 API 概要を更新
		10	2.2 ソフトウェアの要求 2.3 サポートされているツールチェーン を更新
		33	3.11 R_CTSU_OffsetTuning を追加
		24	3.5 R_CTSU_SmsSet の Example: を修正

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、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た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（豊洲フォレストシア）

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。