RL78 ファミリ

TOUCH モジュール Software Integration System

要旨

本アプリケーションノートは TOUCH モジュールについて説明します。

対象デバイス

RL78/G23 Group

RL78/G22 Group

RL78/G16 Group

関連ドキュメント

RL78 ファミリ CTSU モジュール (R11AN0484)

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 計測とデータ処理 3
1.1.3 ボタンのタッチ判定 3
1.1.4 スライダ・ホイールのタッチ位置検出 5
1.1.5 タッチ判定閾値の調整 6
1.1.6 SMS を使用した自動判定計測 8
1.2 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 11
2.6 整数型 11
2.7 コンパイル時の設定 12
2.8 コードサイズ 13
2.9 引数 14
2.10 戻り値 16

3. API 関数 17
3.1 RM_TOUCH_Open 17
3.2 RM_TOUCH_ScanStart 18
3.3 RM_TOUCH_DataGet 19
3.4 RM_TOUCH_CallbackSet 20
3.5 RM_TOUCH_SmsSet 21
3.6 RM_TOUCH_Close 24
3.7 RM_TOUCH_ScanStop 25
3.8 RM_TOUCH_SensitivityRatioGet 26
3.9 RM_TOUCH_ThresholdAdjust 27
3.10 RM_TOUCH_DriftControl 28
3.11 RM_TOUCH_MonitorAddressGet 29

1. 概要

TOUCH モジュールは CTSU モジュールを使用して、静電容量方式のタッチ検出を提供するミドルウェアです。TOUCH モジュールはユーザアプリケーションからのアクセスを想定しています。

1.1 機能

TOUCH 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 CTSU モジュールと同様にコンフィグレーション設定により様々なタッチ検出を提供します。コンフィグレーション設定は QE for Capacitive Touch（以下、QE）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、CTSU とのリンク情報とボタン、スライダ、ホイールの構成情報を表します。複数のタッチインタフェース構成が必要となる場合は、製品内で自己容量ボタンと相互容量ボタンが両方存在するときや、アクティブシールド機能を使用するときです。

また、QE のモニタ機能もサポートします。モニタはデバッガ通信と UART 通信のいずれかで実施されますが、QE からの情報を判別して必要なデータを送信します。

また、スタンドアロン版 QE とのシリアルチューニング機能もサポートしています。QE と UART 通信してコンフィグレーション設定ファイルを生成します。

1.1.2 計測とデータ処理

静電容量の変化からボタンがタッチされたかどうかを判定、スライダ・ホイールの位置検出をします。そのため、定期的に静電容量を計測し続ける必要があります。アプリケーションで R_TOUCH_ScanStart() と R_TOUCH_DataGet()を定期的にコールしてください。詳細はサンプルアプリケーションを参照してください。

CTSU2L は多数決判定方法として JMM（判定多数決モード）と VMM（計測値多数決モード）の 2 種類があります。JMM は 1 つのボタンに対して 3 つの計測結果を CTSU モジュールから取得し、それぞれタッチ判定をして、3 つのタッチ判定結果を多数決して最終タッチ判定をします。VMM は 1 つのボタンに対して CTSU モジュールで多数決判定した 1 つの計測結果を取得してタッチ判定をします。

スライダとホイールは VMM のみ対応しています。

JMM と VMM の詳細は CTSU モジュールの APN を参照してください。

1.1.3 ボタンのタッチ判定

(a)～(f)にタッチ判定機能の説明とその設定方法を示します。

(a) 基準値、閾値の作成

まず、非タッチ状態の計測結果から基準値を作ります。最初の基準値は最初の計測値となります。この基準値から任意のオフセットを設けて閾値とし、計測値が閾値を超えたか超えていないかで ON/OFF の判定をします。

自己容量ボタンと相互容量ボタンの処理は概ね同様です。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、計測値の減少方向に閾値を設定して ON/OFF を判定します。

コンフィグレーション設定（touch_button_cfg_t の threshold）で閾値をボタン毎に設定できます。

実際の判断にはチャタリング防止や外部環境の変化の対応が必要となるので、以降説明する機能も搭載しています。

(b) ポジティブ・ノイズフィルタ / ネガティブ・ノイズフィルタ

チャタリング対策処理の一つです。ON または OFF 状態が一定回数継続した時に ON または OFF を確定します。

コンフィグレーション設定（touch_cfg_t の on_freq, off_freq）で連続一致計測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。

(c) ヒステリシス

チャタリング対策処理の一つです。ON 後の閾値に定数をオフセットして OFF から ON、ON から OFF の閾値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。

コンフィグレーション設定（touch_button_cfg_t の hysteresis）でヒステリシス値をボタン毎に設定できます。大きくするほどチャタリング対策に効果がありますが、ON から OFF または OFF から ON に復帰しにくくなるので注意してください。

(d) チャタリング抑制（ビルドオプション）

チャタリング対策処理の一つです。このビルドオプションは、タッチ判定を行うための機能（ポジティブ・ノイズフィルタ / ネガティブ・ノイズフィルタとヒステリシス）を補完する機能です。閾値を超えた回数をカウントする、カウンタの処理方法を TypeA または TypeB に設定します。

TypeA チャタリング抑制：閾値を超えた回数のカウンタをヒステリシス範囲内は保持します。

TypeB チャタリング抑制：閾値を超えた回数のカウンタをヒステリシス範囲内はリセットします。

図 1 にチャタリング抑制の動作例を示します。

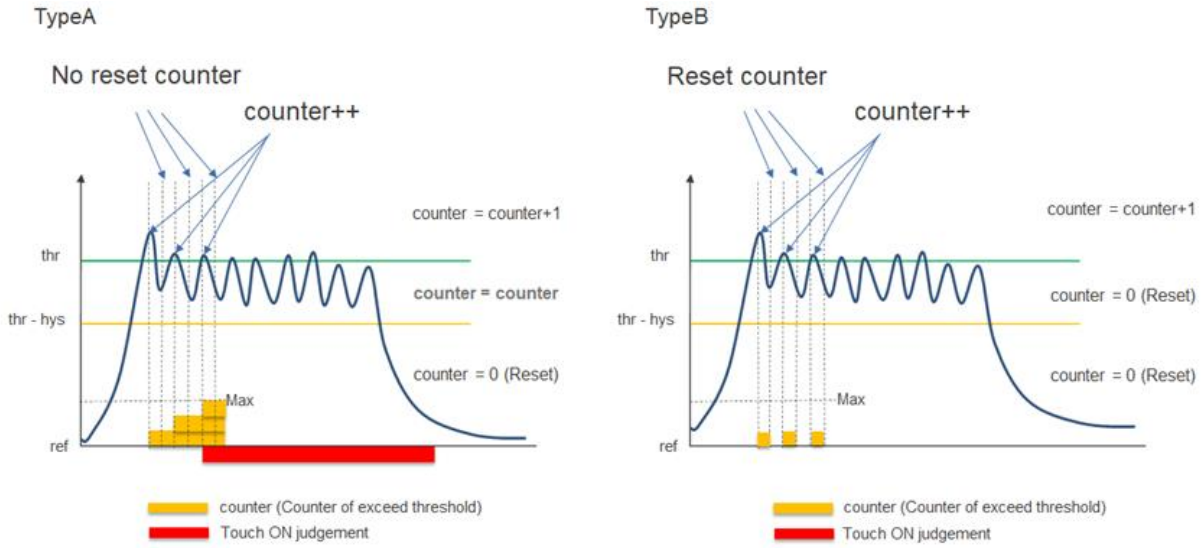


図 1 チャタリング抑制動作例

(e) ドリフト補正処理

外部環境変化に対する対策です。基準値を更新する処理をドリフト補正処理と表します。

OFF 状態での計測値を一定期間で平均化して、一定期間後もタッチ OFF 状態であれば、基準値を更新します。ドリフト補正処理は OFF 状態でのみ実行し、タッチ ON 判定するとクリアします。

コンフィグレーション設定（touch_cfg_t の drift_freq）で期間を設定できます。タッチインタフェース構成内のボタン共通です。環境変化への追従性を調整できます。

図 2 にドリフト補正処理の動作を示します。

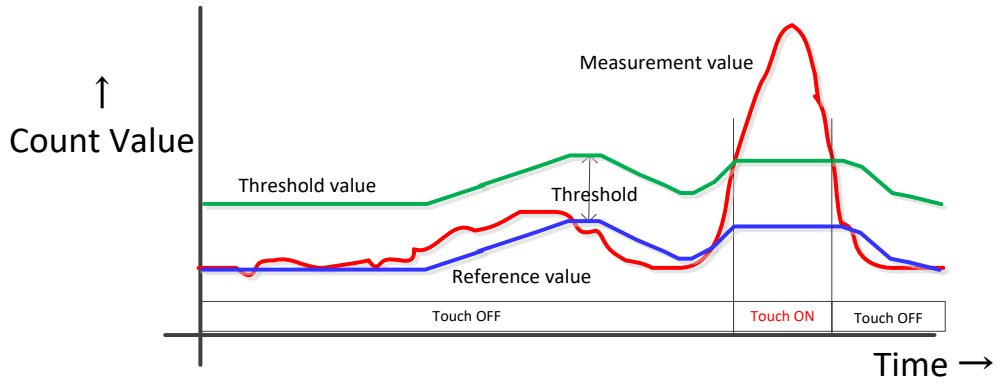


図 2 ボタンの判定

(f) 長押しキャンセル

強いノイズなどの急激な環境変化により、ドリフト補正処理が追従できずに ON 状態から復帰できなくなることがあります。この状態から復帰するために、一定期間 ON 状態が継続した時に強制的に OFF 状態にしてドリフト補正処理を動作させる機能です。

コンフィグレーション設定（touch_cfg_t の cancel_freq）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。

1.1.4 スライダ・ホイールのタッチ位置検出

複数の TS を物理的に直線で配置して使用することでスライダを構成、円形に配置して使用することでホイールを構成します。

タッチ位置は、構成する TS の計測値から計算します。スライダとホイールの計算方法はほぼ同じです。

1. 構成する TS の中での最大値（TS_MAX）を検出します。
2. TS_MAX とその両隣の計測値の差分（d1, d2）を計算します。（スライダで左端が TS_MAX の場合は右隣とその右隣を使用します。右端の場合はその逆です。）
3. d1 と d2 の加算値が閾値を超えていれば、位置計算を開始します。超えていなければ、位置検出無しとして計算処理を終了します。
4. TS_MAX の位置を中心として、d1 と d2 の比率から位置を計算します。スライダは 1 から 100、ホイールは 1 から 360 の範囲になります。

1.1.5 タッチ判定閾値の調整

環境変化などでタッチ変化量に変化が生じた場合のために、QE での再チューニングではなくユーザプログラムで動的にタッチ判定閾値を調整するための機能です。

この機能は API 関数 RM_TOUCH_SensitivityRatioGet()と RM_TOUCH_ThresholdAdjust()を提供します。第 2 引数には touch_sensitivity_info_t 構造体のポインタを設定します。

表 1 touch_sensitivity_info_t 構造体

データ型	メンバ名	内容
uint16_t *	p_touch_sensitivity_ratio	タッチ変化量比率配列のポインタ
uint16_t	old_threshold_ratio	従来の閾値比率
uint16_t	new_threshold_ratio	新しい閾値比率
uint8_t	new_hysteresis_ratio	新しいヒステリシス比率

- タッチ変化量の比率の取得

  計測値と基準値の比較を行い、計測値の方が小さい場合はタッチ変化量比率に 0 を設定し、大きい場合はタッチ変化量比率を計算して出力します。この機能は API 関数 RM_TOUCH_SensitivityRatioGet()で提供します。

  タッチ変化量比率 = (計測値－基準値)×新しい閾値比率÷従来の閾値

- 比率変更による閾値とヒステリシスの調整

  ユーザプログラムから従来の閾値比率と新しい閾値比率と新しいヒステリシス比率を受け取って、閾値とヒステリシスを変更します。この機能は RM_TOUCH_ThresholdAdjust()で提供します。QE チューニングでの閾値のデフォルトはタッチ変化量の 60%です。これをベースに変更する場合は、old_threshold_ratio を 60 に設定してください。

  新しい閾値 = 従来の閾値×新しい閾値比率÷従来の閾値比率

  新しいヒステリシス = 新しい閾値×新しいヒステリシス比率

  これに加えて、タッチ変化量の比率による変更機能もあります。上記の「タッチ変化量の比率の取得」で説明した機能で取得した結果を使用することを推奨します。この機能を使用しない場合は、タッチ変化量比率は 100 にしてください。

  ユーザプログラムからタッチ変化量比率を受け取って、閾値を変更します。

  新しい閾値 = 設定したい閾値×現在のタッチ変化量比率

  新しい閾値を使用して、ヒステリシスを変更します。

  新しいヒステリシス = 新しいヒステリシス比率×新しい閾値

[調整例 1] タッチ感度より EMC ノイズによる誤判定防止の対策を優先する場合

QE チューニングでの閾値のデフォルトはタッチ変化量の 60%です。これを 70%に調整します。

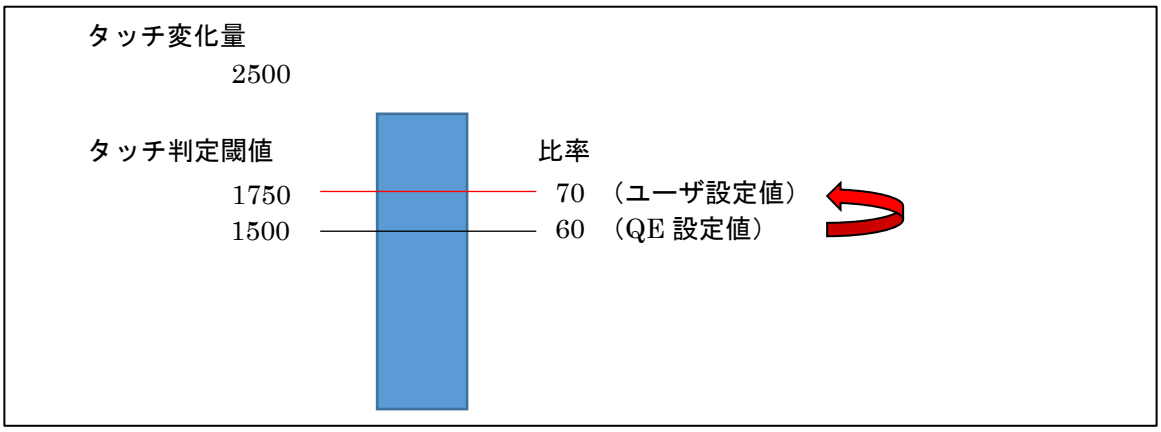


図 3 閾値の比率変更例

この場合、touch_sensitivity_info_t 構造体のメンバを下記のように設定してください。
*p_touch_sensitivity_ratio = 100, old_threshold_ratio = 60, new_threshold_ratio = 70, new_hysteresis_ratio = 5

[調整例 2] オーバレイパネルの種類を変えたいが、QE での再チューニングができない場合

チューニング時より厚いオーバレイパネルを使用するとタッチ変化量が小さくなってしまうため、ソフトウェアを再チューニングせずにそのまま使用するとタッチ判定ができない可能性があります。このような場合に、チューニングしたときのタッチ変化量とオーバレイパネル変更後のタッチ変化量の比率からタッチ判定閾値を調整します。

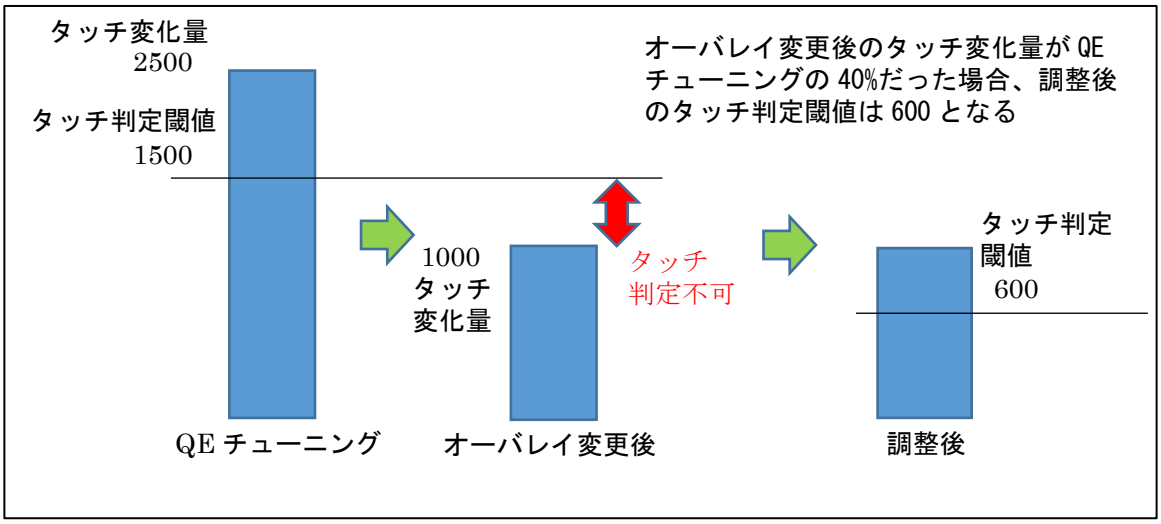


図 4 タッチ変化量の変化における閾値調整の例

この場合、touch_sensitivity_info_t 構造体のメンバを下記のように設定してください。
*p_touch_sensitivity_ratio = 40, old_threshold_ratio = 60, new_threshold_ratio = 60, new_hysteresis_ratio = 5

[実用例] 再チューニングやソフトウェア書き換え不要で調整するアプリケーション

PC と UART 通信できるようにして「調整モード」に入れられるようにします。「調整モード」では、MCU からタッチした状態でのタッチ変化量の比率を PC にリアルタイム送信します。ユーザは PC でモニタしながら比率を確定するコマンドを送信します。MCU は受信したらデータフラッシュに保存します。ソフト起動時にはデータフラッシュの保存した比率を読み込むようにしておき、この保存値からタッチ判定閾値を調整します。

1.1.6 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。

この機能を使用したいタッチインタフェース構成に対して、RM_TOUCH_SmsSet()をコールした後に RM_TOUCH_ScanStart()で計測を開始してください。タッチ ON 判定するまで 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。RM_TOUCH_SmsSet()ではポジティブ・ノイズフィルタを on_freq の設定回数に、ネガティブ・ノイズフィルタを 0 に設定しているため、アプリケーションでは、通常動作と同様に RM_TOUCH_DataGet()をコールしてボタン判定結果を取得することができます。

詳細は 3.5 章および RL78 ファミリ CTSU モジュール (R11AN0484)を参照してください。

1.2 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
RM_TOUCH_Open()	指定したタッチインタフェース構成を初期化します。
RM_TOUCH_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。
RM_TOUCH_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。
RM_TOUCH_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。
RM_TOUCH_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。
RM_TOUCH_Close()	指定したタッチインタフェース構成を終了します。
RM_TOUCH_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。
RM_TOUCH_SensitivityRatioGet()	指定したタッチインタフェース構成のチューニング時のタッチ変化量に対する現在のタッチ変化量の比率を取得します。
RM_TOUCH_ThresholdAdjust()	指定したタッチインタフェース構成のタッチ判定閾値とヒステリシスを調整します。
RM_TOUCH_DriftControl()	指定したタッチインタフェース構成のドリフト補正の設定を変更します。
RM_TOUCH_MonitorAddressGet()	指定したタッチインタフェース構成の QE モニタに使用される変数のアドレスを取得します。

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。

- CTSUb
- CTSU2L
- CTSU2La

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) v1.70 以降
- CTSU モジュール（r_ctsu） v2.00 以降

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 V4.0.0 以降

2.3 サポートされているツールチェーン

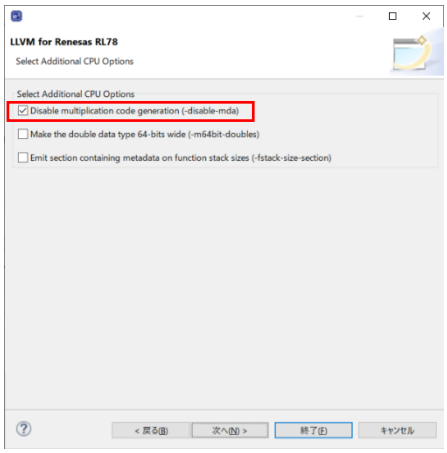
本モジュールは以下に示すツールチェーンで動作確認を行っています。

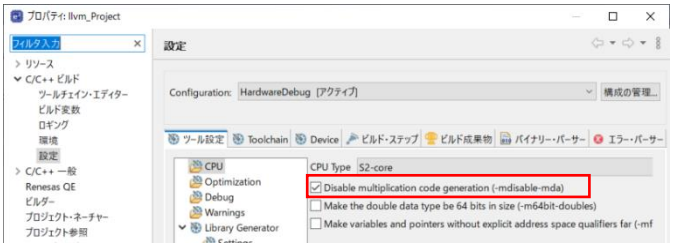
- Renesas CC-RL Toolchain v1.14.00
- IAR Embedded Workbench for Renesas RL78 v5.10.3
- LLVM for Renesas RL78 v17.0.1.202406

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。

RL78/G16 グループで LLVM コンパイラを使用する場合、プロジェクト作成時に以下の CPU Option にチェックしてください。プロジェクト作成後も、プロジェクトのプロパティから設定することができます。





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“rm_touch_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“rm_touch_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	
TOUCH_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。
TOUCH_CFG_MONITOR_ENABLE ※ このオプションは rm_touch_config.h にはありません。QE が出力する qe_touch_define.h に定義されます。その際のデフォルト値は”1”	1 を設定することで QE モニタのためのデータ生成を有効します。
TOUCH_CFG_UART_MONITOR_SUPPORT ※ デフォルト値は“0”	このオプションは TOUCH_CFG_MONITOR_ENABLE が有効のときに使用されます。 1 を設定することで QE とのシリアル通信を有効にします。 注意事項: UART モジュールは SmartConfigurator で生成してください。
TOUCH_CFG_UART_TUNING_SUPPORT ※ デフォルト値は“0”	UART チューニングを設定します。 0 は無効、1 は有効です。
TOUCH_CFG_UART_NUMBER	UART のチャネル番号を設定します。
TOUCH_CFG_CHATTERING_SUPPRESSION__TYPE ※ デフォルト値は“0”	チャタリング抑制タイプを設定します。 “0”の場合、TypeA に設定され、閾値を超えた回数のカウンタをヒステリシス範囲内は保持します。 “1”の場合、TypeB に設定され、閾値を超えた回数のカウンタをヒステリシス範囲内はリセットします。
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 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、rm_touch_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_BUTTONS	ボタンの総数を設定します。
CTSU_CFG_NUM_SLIDERS	スライダの総数を設定します。
CTSU_CFG_NUM_WHEELS	ホイールの総数を設定します。

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。

アプリケーションとボタン、スライダ、ホイールの数によっては RAM サイズをオーバーする可能性があります。特に RL78/G16 グループは RAM が 2KB のためご注意ください。

ROM、RAM のコードサイズ　自己容量ボタン 1（VMM）	
TOUCH_CFG_PARAM_CHECKING_ENABLE 0 TOUCH_CFG_MONITOR_ENABLE 0 TOUCH_CFG_UART_MONITOR_SUPPORT 0	ROM: 1979 bytes RAM: 98 bytes

ROM、RAM のコードサイズ
各設定でのサイズ、追加による増加分（VMM）

	自己容量ボタン 1	+自己容量ボタン	+ホイール	+スライダ	相互容量ボタン 1	+相互容量ボタン
ROM	1979 bytes	+ 6 bytes	+568 bytes	+586 bytes	2169 bytes	+6 bytes
RAM	98 bytes	+24 bytes	+13 bytes	+15 bytes	100 bytes	+26 bytes

ROM、RAM のコードサイズ　自己容量ボタン 1 個（JMM）	
TOUCH_CFG_PARAM_CHECKING_ENABLE 0 TOUCH_CFG_MONITOR_ENABLE 0 TOUCH_CFG_UART_MONITOR_SUPPORT 0	ROM: 2563 bytes RAM: 138 bytes

ROM、RAM のコードサイズ
各設定でのサイズ、追加による増加分（JMM）

	自己容量ボタン 1	+自己容量ボタン	+ホイール	+スライダ	相互容量ボタン 1	+相互容量ボタン
ROM	2563 bytes	+18 bytes	—	—	2699 bytes	+18 bytes
RAM	138 bytes	+64 byte	—	—	144 bytes	+70 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 rm_touch_api.h に定義されています。

表 2 に touch_ctrl_t 構造体（コントロール構造体）を示します。この構造体で使用しているデータ型については rm_touch_qe.h を参照してください。各タッチインタフェース構成のタッチ判定設定やタッチ判定結果を管理します。QE for Capacitive Touch を使用することで、タッチインタフェース構成に応じたコントロール構造体の変数が qe_touch_config.c に出力されるので、このモジュールの API の第一引数に設定してください。

表 2 touch_ctrl_t 構造体

データ型	データ名	内容
uint32_t	open	Open フラグ
touch_button_info_t	binfo	ボタン情報
touch_slider_info_t	sinfo	スライダ情報
touch_wheel_info_t	winfo	ホイール情報
bool	serial_tuning_enable	シリアルチューニング有効フラグ
touch_cfg_t const *	p_touch_cfg	コンフィグ構造体のポインタ
ctsu_instance_t const *	p_ctsu_instance	CTSU コントロール構造体のポインタ
touch_mm_info_t	p_touch_mm_info	多数決判定情報構造体へのポインタ

touch_button_info_t 構造体を示します。ボタンの判定結果および各ボタンの判定のためのデータを管理します。

表 3 touch_button_info_t 構造体

データ型	データ名	内容
uint64_t	status	ボタン判定結果
uint16_t *	p_threshold	しきい値バッファのポインタ
uint16_t *	p_hysteresis	ヒステリシスバッファのポインタ
uint16_t *	p_reference	基準値バッファのポインタ
uint16_t *	p_on_count	しきい値を超えた回数
uint16_t *	p_off_count	しきい値を下回った回数
uint32_t *	p_drift_buf	ドリフト補正バッファのポインタ
uint16_t *	p_drift_count	ドリフトカウントバッファのポインタ
uint8_t	on_freq	ポジティブ・ノイズフィルタの判定回数
uint8_t	off_freq	ネガティブ・ノイズフィルタの判定回数
uint16_t	drift_freq	ドリフト補正の設定回数
uint16_t	cancel_freq	長押しキャンセルの設定回数

touch_slider_info_t 構造体を示します。スライダの位置結果および各スライダのしきい値を管理します。

表 4 touch_slider_info_t 構造体

データ型	データ名	内容
uint16_t *	p_position	位置結果バッファのポインタ
uint16_t *	p_threshold	しきい値バッファのポインタ

touch_wheel_info_t 構造体を示します。ホイールの位置結果および各ホイールのしきい値を管理します。

表 5 touch_wheel_info_t 構造体

データ型	データ名	内容
uint16_t *	p_position	位置結果バッファのポインタ
uint16_t *	p_threshold	しきい値バッファのポインタ

表 6 に touch_cfg_t 構造体（コンフィグ構造体）を示します。

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

表 6 touch_cfg_t 構造体

データ型	メンバ名	内容	値の範囲
touch_button_cfg_t *	p_buttons	ボタンコンフィグレーションへのポインタ	-
touch_slider_cfg_t *	p_sliders	スライダコンフィグレーションへのポインタ	-
touch_wheel_cfg_t *	p_wheels	ホイールコンフィグレーションへのポインタ	-
touch_pad_cfg_t *	p_pad	パッドコンフィグレーションへのポインタ	-
uint8_t	num_buttons	ボタン数	0 to 64
uint8_t	num_sliders	スライダ数	0 or 7
uint8_t	num_wheels	ホイール数	0 or 7
uint8_t	on_freq	ON 判定の累積数	0 to 255（0 は機能を無効）
uint8_t	off_freq	OFF 判定の累積数	0 to 255（0 は機能を無効）
uint16_t	drift_freq	ベース値ドリフト周期	0 to 65535 （0 は機能を無効）
uint16_t	cancel_freq	最大連続 ON 判定回数	0 to 65535 （0 は機能を無効）
uint8_t	number	QE モニタの構成番号	0 to 255
ctsu_instance_t *	p_ctsu_instance	CTSU instance のポインタ	-
void *	p_context	コンテキストのポインタ	-
void *	p_extend	拡張構成のポインタ	-

表 7 touch_button_cfg_t 構造体

データ型	メンバ名	内容	値の範囲
uint8_t	elem_index	ボタンのエレメントインデックス	0 to 63
uint16_t	threshold	タッチ閾値	1 to 65535
uint16_t	hysteresis	チャタリング抑制用ヒステリシス値	0 to 65534

表 8 touch_slider_cfg_t 構造体

データ型	メンバ名	内容	値の範囲
uint8_t *	p_elem_index	スライダのエレメントインデックスへのポインタ	-
uint8_t	num_elements	スライダで使用しているエレメント数	1 to 10
uint16_t	threshold	位置計算の閾値	1 to 65535

表 9 touch_wheel_cfg_t 構造体

データ型	メンバ名	内容	値の範囲
uint8_t *	p_elem_index	ホイールのエレメントインデックスへのポインタ	-
uint8_t	num_elements	ホイールで使用しているエレメント数	1 to 8
uint16_t	threshold	位置計算の閾値	1 to 65535

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 = 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_NOT_OPEN             = 7,                    ///< Requested channel is not configured or API not open
      FSP_ERR_ALREADY_OPEN         = 14,                   ///< Requested channel is already open in a different
configuration
      FSP_ERR_INVALID_HW_CONDITION  = 27,                  ///< Detected hardware is in invalid condition

      /* 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;
```

3. API 関数

3.1 RM_TOUCH_Open

この関数は、本モジュールの初期化をする関数です。この関数は他の API 関数を使用する前に実行する必要があります。タッチインタフェース構成毎に実行してください。

Format

```
fsp_err_t RM_TOUCH_Open (touch_ctrl_t * const p_ctrl,
                         touch_cfg_t const * const p_cfg)
```

Parameters

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

Return Values

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

Properties

rm_touch_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をした後、R_CTSU_Open()をコールして CTSU モジュールを初期化します。TOUCH_CFG_MONITOR_ENABLE が有効のときは、モニタバッファの初期設定をします。さらに TOUCH_CFG_UART_MONITOR_SUPPORT が有効のときは、UART モニタの初期設定と UART モジュールの初期化をします。

Example

```
fsp_err_t err;

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

/* Initialize the API. */
err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_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。

この関数で CTSU モジュールの R_CTSU_Open()をコールしています。R_CTSU_Open()も参照してください。

3.2 RM_TOUCH_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 RM_TOUCH_ScanStart (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

Return Values

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

Properties

rm_touch_api.h にプロトタイプ宣言されています。

Description

この関数は、R_CTSU_ScanStart()をコールして計測を開始します。

Example

```
fsp_err_t err;

/* Initiate a sensor scan by software trigger */
err = RM_TOUCH_ScanStart(&g_touch_ctrl);

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

Special Notes:

この関数で CTSU モジュールの R_CTSU_ScanStart()をコールしています。R_CTSU_ScanStart()も参照してください。

3.3 RM_TOUCH_DataGet

この関数は、指定したタッチインタフェース構成の状態を読み込みます。

Format

```
fsp_err_t RM_TOUCH_DataGet (touch_ctrl_t * const p_ctrl,
                        uint64_t          * p_button_status,
                        uint16_t          * p_slider_position,
                        uint16_t          * p_wheel_position)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

p_button_status : ボタン状態を格納するバッファへのポインタ

p_slider_position : スライダ位置を格納するバッファへのポインタ

p_wheel_position : ホイール位置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_INVALID_HW_CONDITION	*/*測定値が異常 */*

Properties

rm_touch_api.h にプロトタイプ宣言されています。

Description

この関数は、R_CTSU_DataGet()をコールして前回計測した全ての計測値を読み込み、タッチ判定および位置検出をします。TOUCH_CFG_MONITOR_ENABLE が有効のときは、モニタバッファにデータを格納します。さらに TOUCH_CFG_UART_MONITOR_SUPPORT が有効のときは、モニタバッファのデータを UART 送信します。

Example:

```
  fsp_err_t err;
  uint64_t button_status;
  uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
  uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];

  /* Get all sensor values */
  err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, slider_position,
wheel_position);
```

Special Notes:

この関数で CTSU モジュールの R_CTSU_DataGet()をコールしています。R_CTSU_DataGet()も参照してください。

3.4 RM_TOUCH_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 RM_TOUCH_CallbackSet (touch_ctrl_t * const p_api_ctrl,
                          void (* p_callback)(touch_callback_args_t *),
                          void const * const p_context,
                          touch_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

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

rm_touch_api.h にプロトタイプ宣言されています。

Description

この関数は、R_CTSU_CallbackSet()をコールしてコールバック関数を設定します。

Example:

```
fsp_err_t err;

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

Special Notes:

この関数で CTSU モジュールの R_CTSU_CallbackSet()をコールしています。R_CTSU_CallbackSet()も参照してください。

3.5 RM_TOUCH_SmsSet

この関数は、指定したタッチインタフェース構成に対して 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 RM_TOUCH_SmsSet (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

Return Values

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

Properties

rm_touch_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。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 RM_TOUCH_ScanStart()をコールしてください。タッチ ON 判定されると、コールバック関数がコールされます。RM_TOUCH_DataGet()をコールして、ボタンのステータスを取得してください。

Example:

```
    fsp_err_t err;
    uint64_t button_status;

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

    /* for ExternalTrigger */
    ELISEL7 = 0x17;
    ELL1SEL0 = 0x08;
    ELL1LNK0 = 0x01;
    ELOSEL6 = 0x01;
    ELOENCTL = 0x40;

    /* Open Touch middleware */
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    /* Offset tuning cannot be performed by SMS measurement, so it should be
performed in advance. */

    /* for ExternalTrigger */
    err = RM_TOUCH_ScanStart(&g_touch_ctrl);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }
    R_ITL_Start_Interrupt();
    R_Config_ITL000_Start();

    /* Measurement loop */
    while (true)
    {
        /* for [CONFIG01] configuration */
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            R_Config_ITL000_Stop();
            break;
        }
    }

    /* Start SMS measurement */
    if (0 == button_status)
    {
        err = RM_TOUCH_SmsSet(&g_touch_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
    }
    err = RM_TOUCH_ScanStart(&g_touch_ctrl);
    if (FSP_SUCCESS != err)
    {
```

```
    while (true) {}
}

R_Config_ITL000_Start();

/* Measurement loop for low power consumption */
__stop();

err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, NULL, NULL);
if (FSP_SUCCESS == err)
{
   if (button_status)
   {
      /* LED ON  */
   }
   else
   {
      /* LED OFF */
   }
}
```

Special Notes:

この関数はタッチ OFF を確認してコールしてください。タッチ ON でコールすると、タッチ ON でのベースライン設定となり、ベースラインドリフト機能で更新されるまでタッチ判定ができなくなります。

この関数で CTSU モジュールの R_CTSU_SmsSet()をコールしています。R_CTSU_SmsSet()も参照してください。

3.6 RM_TOUCH_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 RM_TOUCH_Close (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

Return Values

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

Properties

rm_touch_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 = RM_TOUCH_Close(&g_touch_ctrl);
```

Special Notes:

この関数で CTSU モジュールの R_CTSU_Close()をコールしています。R_CTSU_Close()も参照してください。

3.7 RM_TOUCH_ScanStop

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。

Format

```
fsp_err_t RM_TOUCH_ScanStop (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

Return Values

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

Properties

rm_touch_api.h にプロトタイプ宣言されています。

Description

この関数は、指定したタッチインタフェース構成を停止します。

Example:

```
fsp_err_t err;

/* Stop CTSU module */
err = RM_TOUCH_ScanStop(&g_touch_ctrl);
```

Special Notes:

この関数で CTSU モジュールの R_CTSU_ScanStop()をコールしています。R_CTSU_ScanStop()も参照してください。

3.8 RM_TOUCH_SensitivityRatioGet

この関数は、指定したタッチインタフェース構成のチューニング時のタッチ変化量に対する現在のタッチ変化量の比率を取得します。

Format

```
fsp_err_t RM_TOUCH_SensitivityRatioGet (touch_ctrl_t * const p_ctrl,
                        touch_sensitivity_info_t * p_touch_sensitivity_info);
```

Parameters

p_ctrl : コントロール構造体へのポインタ

p_touch_sensitivity_info : タッチ変化量比率計算のテーブル情報を格納する変数へのポインタ

Return Values

FSP_SUCCESS /* タッチ変化量の比率の取得に成功しました */

FSP_ERR_INVALID_POINTER /* ポインタが無効なメモリ位置を指しています */

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

rm_touch_api.h にプロトタイプ宣言されています。

Description

現在の閾値と閾値比率から計算したタッチ変化量を 100%として、計測結果のタッチ変化量の比率を出力します。指定したタッチインタフェース構成の全てのボタンについて出力するので、ボタン数分のバッファを用意して p_touch_sensitivity_info->p_touch_sensitivity_ratio にバッファの先頭アドレスを設定してください。

Example:

```
  qe_err_t err;
  touch_sensitivity_info_t touch_sensitivity_table[QE_NUM_METHODS];
  uint16_t touch_sensitivity_first[CONFIG01_NUM_BUTTONS ] = { 100 };

  touch_sensitivity_table[QE_METHOD_CONFIG01].p_touch_sensitivity_ratio =
touch_sensitivity_first;
  touch_sensitivity_table[QE_METHOD_CONFIG01].old_threshold_ratio = 60;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_threshold_ratio = 60;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_hysteresis_ratio = 5;

  err = RM_TOUCH_SensitivityRatioGet(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);
```

3.9 RM_TOUCH_ThresholdAdjust

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判定閾値とヒステリシスを調整します。

Format

```
fsp_err_t RM_TOUCH_ThresholdAdjust  (touch_ctrl_t * const p_ctrl,
                        touch_sensitivity_info_t * p_touch_sensitivity_info);
```

Parameters

p_ctrl : コントロール構造体へのポインタ

p_sensitivity_info : タッチ変化量比率計算のテーブル情報を格納する変数へのポインタ

Return Values

FSP_SUCCESS	*/* タッチ判定閾値の変更に成功しました */*
FSP_ERR_INVALID_POINTER	*/* ポインタが無効なメモリ位置を指しています */*
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

rm_touch_api.h にプロトタイプ宣言されています。

Description

タッチ変化量の比率（各ボタン）、新しい閾値の比率、従来の閾値の比率、新しいヒステリシス値の比率を入力として、閾値とヒステリシスを調整します。

Example:

```
  qe_err_t err;
  touch_sensitivity_info_t touch_sensitivity_table[QE_NUM_METHODS];
  uint16_t touch_sensitivity_first[CONFIG01_NUM_BUTTONS ] = { 100 };

  touch_sensitivity_table[QE_METHOD_CONFIG01].p_touch_sensitivity_ratio =
touch_sensitivity_first;
  touch_sensitivity_table[QE_METHOD_CONFIG01].old_threshold_ratio = 60;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_threshold_ratio = 70;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_hysteresis_ratio = 5;

  err = RM_TOUCH_SensitivityRatioGet(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);

  err = RM_TOUCH_ThresholdAdjust(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);
```

Special Notes:

なし

3.10 RM_TOUCH_DriftControl

この関数は、指定したタッチインタフェース構成のドリフト補正の設定を変更します。

Format

```
fsp_err_t RM_TOUCH_DriftControl(touch_ctrl_t * const p_ctrl,
                        uint16_t input_drift_freq);
```

Parameters

p_ctrl : コントロール構造体へのポインタ

input_drift_freq : ドリフト補正の間隔、有効／無効

Return Values

FSP_SUCCESS	*/* ドリフト補正の変更に成功しました */*
FSP_ERR_ASSERTION	*/* “引数のポインタが指定されていません*/*

Properties

rm_touch_api.h にプロトタイプ宣言されています。

Description

ドリフト補正の設定を input_drift_freq に設定した回数にします。回数を 0 以外に設定した場合にドリフト補正機能は有効になり、0 に設定するとドリフト補正機能は無効になります。

本 API の使用例としては、RM_TOUCH_SensitivityRatioGet()を使用してタッチ変化量の比率を求める際に、オーバレイが厚くなったことでタッチ変化量が減り、タッチしても閾値を超えなくなってしまった場合に基準値がドリフトすることを防ぎます。

Example:

```
qe_err_t  err;

err = RM_TOUCH_DriftControl(g_qe_touch_instance_config01.p_ctrl, 0);
```

Special Notes:

なし

3.11 RM_TOUCH_MonitorAddressGet

この関数は、指定したタッチインタフェース構成の QE モニタに使用される変数のアドレスを取得します。

Format

```
fsp_err_t RM_TOUCH_MonitorAddressGet (touch_ctrl_t * const p_ctrl,
                                      uint8_t ** pp_monitor_buf,
                                      uint8_t ** pp_monitor_id,
                                      uint16_t ** pp_monitor_size)
```

Parameters

p_ctrl : コントロール構造体へのポインタ

pp_monitor_buf : モニタバッファの先頭アドレスを格納する変数へのポインタ

pp_monitor_id : モニタ ID 変数のアドレスを格納する変数へのポインタ

pp_monitor_size : モニタサイズの先頭アドレスを格納する変数へのポインタ

Return Values

FSP_SUCCESS	*/* QE モニター変数アドレスが取得に成功しました */*
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_NOT_ENABLED	*/* 要求された操作が有効になっていません */*

Properties

rm_touch_api.h にプロトタイプ宣言されています。

Description

この機能は、自動判定とソフトウェア判定の両方のタッチインタフェース構成がある場合の QE モニタ機能のために使用します。第 2 引数でモニタバッファの先頭アドレス、第 3 引数でモニタ ID 変数のアドレス、第 4 引数でモニタサイズの先頭アドレスを取得できます。

Example:

```
qe_err_t  err;
uint8_t * gp_monitor_buf;
uint8_t * gp_monitor_id;
uint16_t * gp_monitor_size;

err =  RM_TOUCH_MonitorAddressGet(g_qe_touch_instance_config01.p_ctrl,
                                  &gp_monitor_buf,
                                  &gp_monitor_id,
                                  &gp_monitor_size);
```

Special Notes:

QE モニタ機能のために使用します。

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	3	1.1.1 QE for Capcitive Touch との連携 を更新
		5-6	1.1.5 タッチ判断閾値の調整 を追加
		6	SMS を使用した自動判定計測 を追加
		9	2.7 コンパイル時の設定 を更新
		10	2.8 コードサイズ を更新
		10	2.9 引数 を更新
		11	2.10 戻り値 を更新
		-	RM_TOUCH_VersionGet を削除
		18-20	3.5 RM_TOUCH_SmsSet を追加
		22	3.7 RM_TOUCH_ScanStop を追加
		23-24	3.8 RM_TOUCH_SensitivityRatioGet を追加
		25-26	3.9 RM_TOUCH_ThresholdAdjust を追加
		28-29	3.10 RM_TOUCH_DriftControl を追加
1.11	2022/1/18	8	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
		10	2.8 コードサイズ を更新
1.20	2022/4/20	6	1.1.6 SMS を使用した自動判定計測 に追記
		24	3.5 RM_TOUCH_SmsSet の Example:を修正
1.30	2023/2/14	1	1 概要の対象デバイスに RL78/G22 を追加
		4	1.1.3 ボタンのタッチ判定を追加
		9	2.2 ソフトウェアの要求を更新
		9	2.3 サポートされているツールチェーンを更新
		11	2.8 コードサイズを修正
		13	2.10 戻り値を更新
		17	3.3RM_TOUCH_DataGet を更新
1.40	2023/6/14	1	対象デバイスに RL78/G16 グループを追加
		9	2.1 ハードウェアの要求
		9	2.2 ソフトウェアの要求
		9	2.4 制限事項を更新
		12	2.8 コードサイズを修正
1.50	2024/5/31	9	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
		23	3.5 RM_TOUCH_SmsSet の Special Notes を更新
2.00	2024/10/15	3	1.1.2 計測とデータ処理を更新
		6	1.1.5 タッチ判定閾値の調整を更新
		9	2.1 ハードウェアの要求を更新
		9	2.2 ソフトウェアの要求を更新
		9	2.3 サポートされているツールチェーンを更新
		10	2.7 コンパイル時の設定を更新
		11	2.8 コードサイズを更新
		12	2.9 引数を更新
		22	3.8 RM_TOUCH_SensitivityRatioGet の Descripription を更新
		23	3.9 RM_TOUCH_ThresholdAdjust の Descripription を更新
		25	3.11 RM_TOUCH_MonitorAddressGet を追加

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。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。