RX ファミリ

QE Touch モジュール Firmware Integration Technology

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

・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

QE Touch Diagnostic API Users' Guide (R01AN4785EU)
Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
RX100 Series VDE Certified IEC60730 Self-Test Code (R01AN2061ED)
RX v2 Core VDE Certified IEC60730 Self-Test Code for RX v2 MCU (R01AN3364EG)

目次

1. 概要

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シリアル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。シリアル通信を用いて実施されます。QE とデータ通信を行い計測周波数・計測時間・閾値の決定を行います。

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 RM_TOUCH_ScanStart() と RM_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。

1.1.3 ボタンのタッチ判定

(a) 基準値、閾値の作成

タッチボタンでは、機械接点ボタンのようなハードウェアによる ON/OFF 状態は存在しません。このため、ソフトウェアで ON/OFF の判定をします。

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) 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 1 にドリフト補正処理の動作を示します。

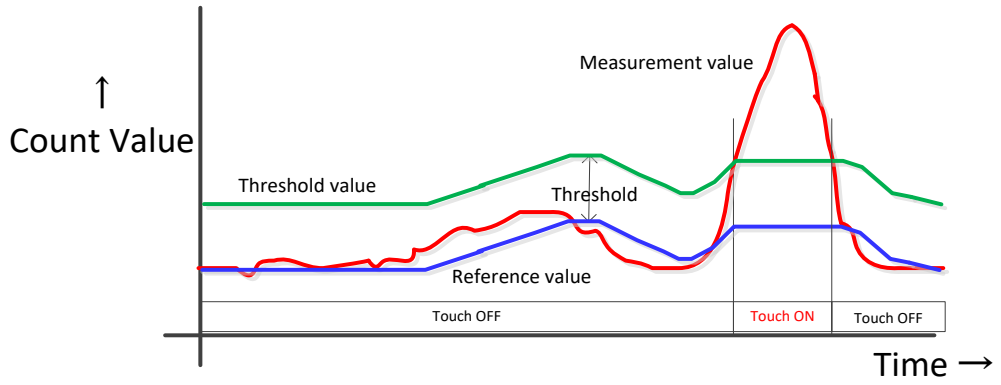


図 1 ボタンの判定

(e)　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。

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　タッチ判断閾値の調整

QE チューニングでは、指でボタンを押した状態で測定してコンフィグファイルに出力します。その設定値はタッチ／非タッチ状態の間でのタッチ変化量の 60%を閾値、ヒステリシス係数は閾値の 5%です。

本モジュールは、この閾値とヒステリシス係数を動的に調整する機能を提供しています。

下記の 2 つの機能になります。

1. 閾値とヒステリシス係数を任意の割合に調整

RM_TOUCH_ThresholdAdjust() を使用してください。

[使用例]

EMC ノイズ対策のために、閾値をタッチ変化量の 70%に変更したい。

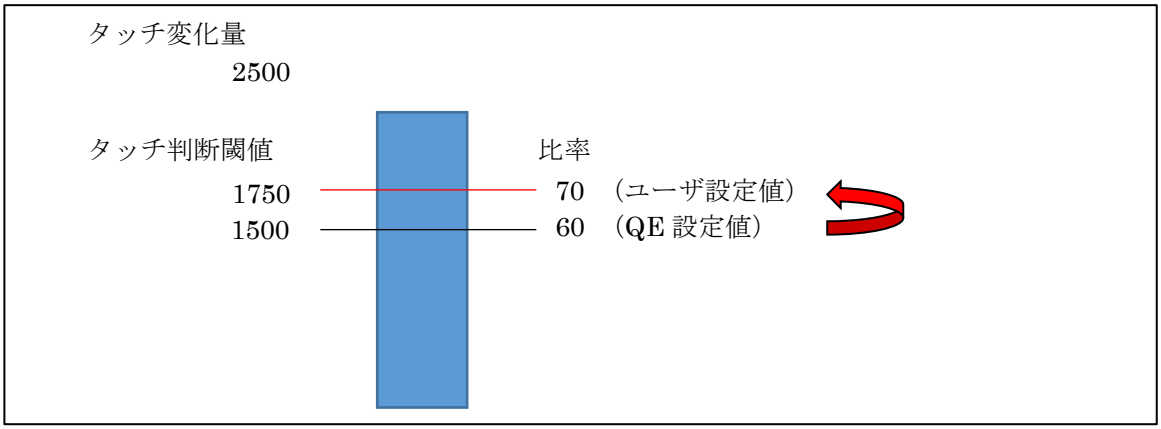


図 2 閾値の比率変更例

2. 閾値とヒステリシス係数を現在のタッチ変化量に合わせて調整

RM_TOUCH_SensitivityRatioGet()、RM_TOUCH_ThresholdAdjust ()、RM_TOUCH_DriftControl() を使用してください。

[使用例]

オーバレイパネルの種類を変えた際、タッチ変化量がチューニングしたときと異なってしまう。ソフトウェアは再チューニングせずにそのまま使用したい。QE チューニング時より厚いオーバレイパネルを使用した場合、タッチ変化量は小さくなるが、タッチ判断閾値は同一のため、タッチ判定ができない可能性がある。QE チューニングのタッチ変化量とオーバレイ変更後のタッチ変化量の比率からタッチ判断閾値を調整する。

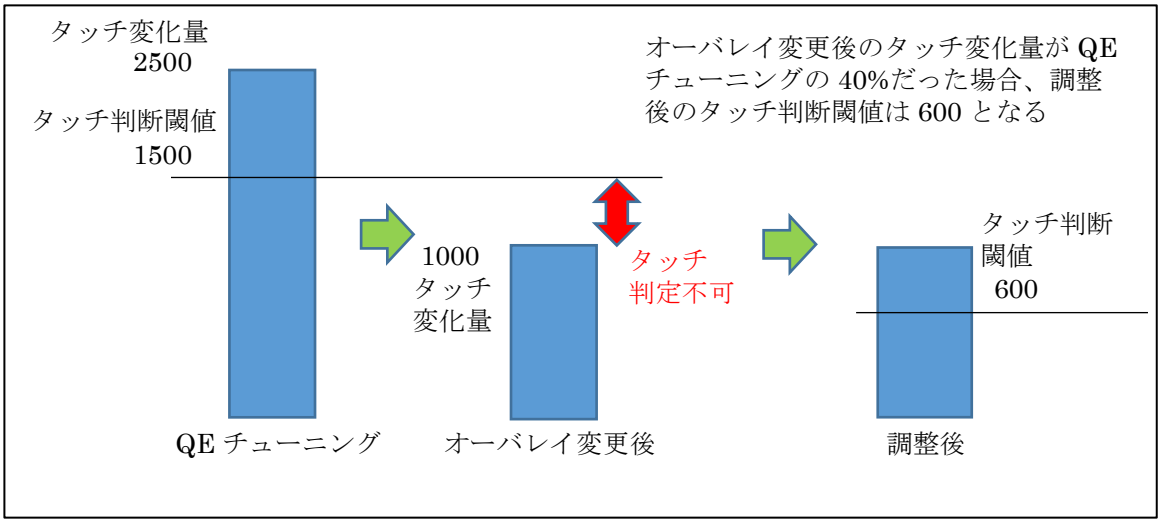


図 3 タッチ変化量の変化における閾値調整の例

データフラッシュを利用して、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例を示します。

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断閾値を調整します。

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_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。
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_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_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_GetSensitivityRatio()	QE 設定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算出します。
RM_TOUCH_AdjustThresholdRatio()	タッチ変化量に対するタッチ判断閾値の比率、ヒステリシス値の比率を変更し、現在のタッチ変化量の比率からタッチ判断閾値、ヒステリシス値を調整します。
RM_TOUCH_DriftControl()	ドリフト補正の設定を変更します。
RM_TOUCH_MonitorAddressGet()	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 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 が以下の機能のどちらかをサポートしている必要があります。

- CTSU
- 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 (r_bsp) v6.10 以降
- QE CTSU FIT モジュール (r_ctsu_qe) v2.10
- SCI module (r_sci_rx) v3.90 以降

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 V2.0.0 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 v.3.04.00
- IAR RX Toolchain v4.20.3
- GCC RX Toolchain v8.3.0.202104

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“rm_touch_qe.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_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。

rm_touch_qe_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_qe_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 での生成してください。
TOUCH_CFG_UART_MONITOR_SUPPORT	UART モニタを設定します。 0 は無効、1 は有効です。
TOUCH_CFG_UART_TUNING_SUPPORT	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_UART_BAUDRATE	UART のボーレートを設定します。
TOUCH_CFG_UART_PRIORITY	UART の割り込み優先度を設定します。
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_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。	
TOUCH_CFG_NUM_BUTTONS	ボタンの総数を設定します。
TOUCH_CFG_NUM_SLIDERS	スライダの総数を設定します。
TOUCH_CFG_NUM_WHEELS	ホイールの総数を設定します。
TOUCH_CFG_PAD_ENABLE	タッチパッドの使用有無を設定します。0 は無効、1 は有効です。

2.8　コードサイズ

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

ROM、RAM のコードサイズ　自己容量ボタン 1	
TOUCH_CFG_PARAM_CHECKING_ENABLE 0 TOUCH_CFG_MONITOR_ENABLE 0 TOUCH_CFG_UART_MONITOR_SUPPORT 0	ROM: 1209 バイト RAM: 21 バイト

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

	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	1209 バイト	+0 バイト	+376 バイト	+424 バイト	1225 バイト	+0 バイト
RAM	21 バイト	+16 バイト	+5 バイト	+5 バイト	21 バイト	+16 バイト

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_qe.h、rm_touch_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 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_touch_instance_ctrl
{
    uint32_t                open;            ///< Whether or not driver is open.
    touch_button_info_t     binfo;           ///< Information of button.
    touch_slider_info_t     sinfo;           ///< Information of slider.
    touch_wheel_info_t      winfo;           ///< Information of wheel.
    touch_cfg_t const     * p_touch_cfg;     ///< Pointer to initial configurations.
    ctsu_instance_t const * p_ctsu_instance; ///< Pointer to CTSU instance.
} touch_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 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、RM_TOUCH_Open()の第二引数に設定してください。

```
typedef struct st_touch_cfg
{
    touch_button_cfg_t const * p_buttons;    ///< Pointer to array of button configuration.
    touch_slider_cfg_t const * p_sliders;    ///< Pointer to array of slider configuration.
    touch_wheel_cfg_t const  * p_wheels;     ///< Pointer to array of wheel configuration.
    uint8_t                  num_buttons;    ///< Number of buttons.
    uint8_t                  num_sliders;    ///< Number of sliders.
    uint8_t                  num_wheels;     ///< Number of wheels.
    uint8_t                  on_freq;        ///< The cumulative number of determinations of ON.
    uint8_t                  off_freq;       ///< The cumulative number of determinations of OFF.
    uint16_t                 drift_freq;     ///< Base value drift frequency. [0 : no use]
```

```
    uint16_t                cancel_freq;     ///< Maximum continuous ON. [0 : no use]
    uint8_t                 number;          ///< Configuration number for QE monitor.
    ctsu_instance_t const * p_ctsu_instance; ///< Pointer to CTSU instance.
    void const            * p_context;       ///< User defined context passed into callback function.
    void const            * p_extend;        ///< Pointer to extended configuration by instance of interface.
} touch_cfg_t;
```

上記構造体で使用している列挙型を示します。

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

/** Configuration of each slider */
typedef struct st_touch_slider_cfg
{
    uint8_t const * p_elem_index;     ///< Element number array used by this slider.
    uint8_t         num_elements;     ///< Number of elements used by this slider.
    uint16_t        threshold;        ///< Position calculation start threshold value.
} touch_slider_cfg_t;

/** Configuration of each wheel */
typedef struct st_touch_wheel_cfg_t
{
    uint8_t const * p_elem_index;     ///< Element number array used by this wheel.
    uint8_t         num_elements;     ///< Number of elements used by this wheel.
    uint16_t        threshold;        ///< Position calculation start threshold value.
} touch_wheel_cfg_t;

/** Callback function parameter data */
typedef struct st_ctsu_callback_args touch_callback_args_t;/** CTSU Events for callback function */
```

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_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


      /* 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

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

2.12 API 互換モード

Rev.2.00 で API 関数を全面改訂しています。

Rev.1.11 の API を使用するために API 互換モードを用意しています。

QE でのコード出力時に API 互換モードのコードを出力してください。QE でコード出力オプションの「API 互換モードを使用する」にチェックして、「ファイル出力」ボタンを押すことで、API 互換モード用のコードが出力されます。

Rev.v1.11 の API については QE Touch モジュール Firmware Integration Technology Rev.1.11 (R01AN4470JU0111)を参照してください。

API 互換モードでは、R_TOUCH_Open ()で検出していた QE_ERR_OT_WINDOW_SIZE のエラー検出をサポートしていません。

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(通常は QE 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 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

rm_touch_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(通常は QE 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

rm_touch_qe.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(通常は QE 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

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 */*

Properties

rm_touch_qe.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(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

rm_touch_qe.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_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(通常は QE 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 */*

Properties

rm_touch_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 = 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。

3.6　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(通常は QE 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

rm_touch_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 = 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。

3.7　RM_TOUCH_SensitivityRatioGet

この関数は、QE 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返します。

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(通常は QE によって生成)

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 */*

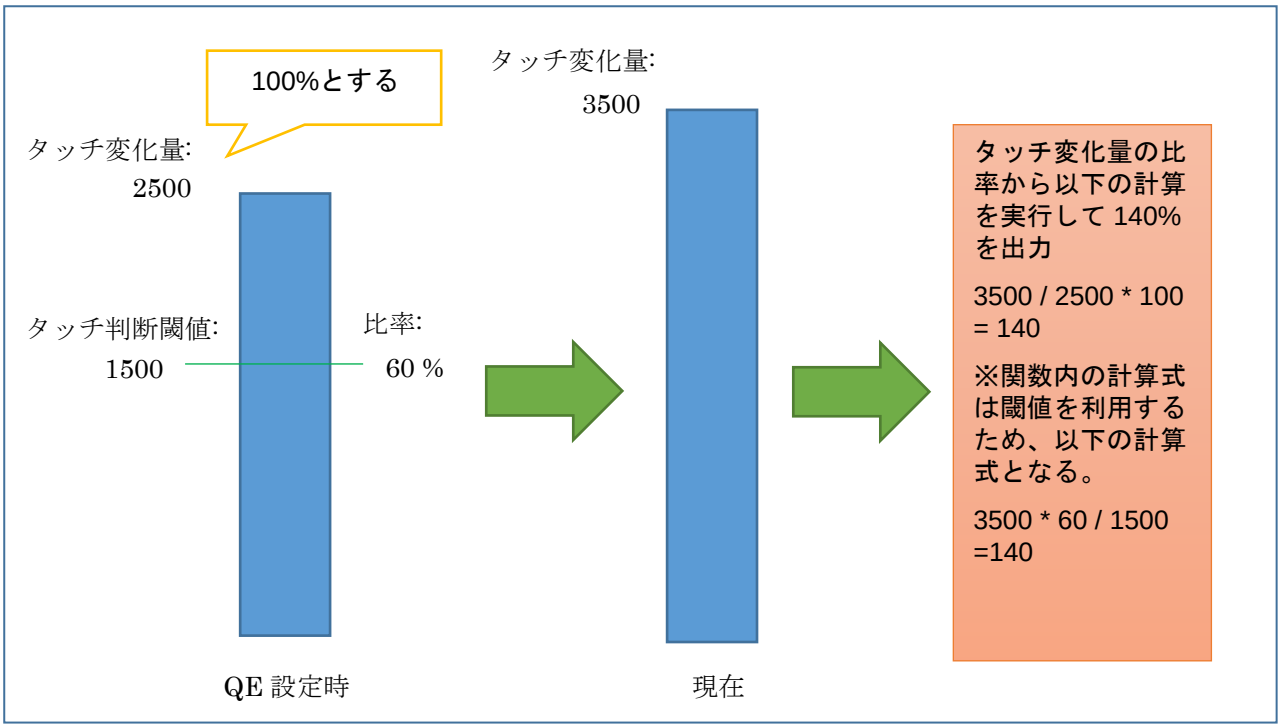
FSP_ERR_CTSU_INCOMPLETE_TUNING　　*/* 初期オフセット調整中 */*

Properties

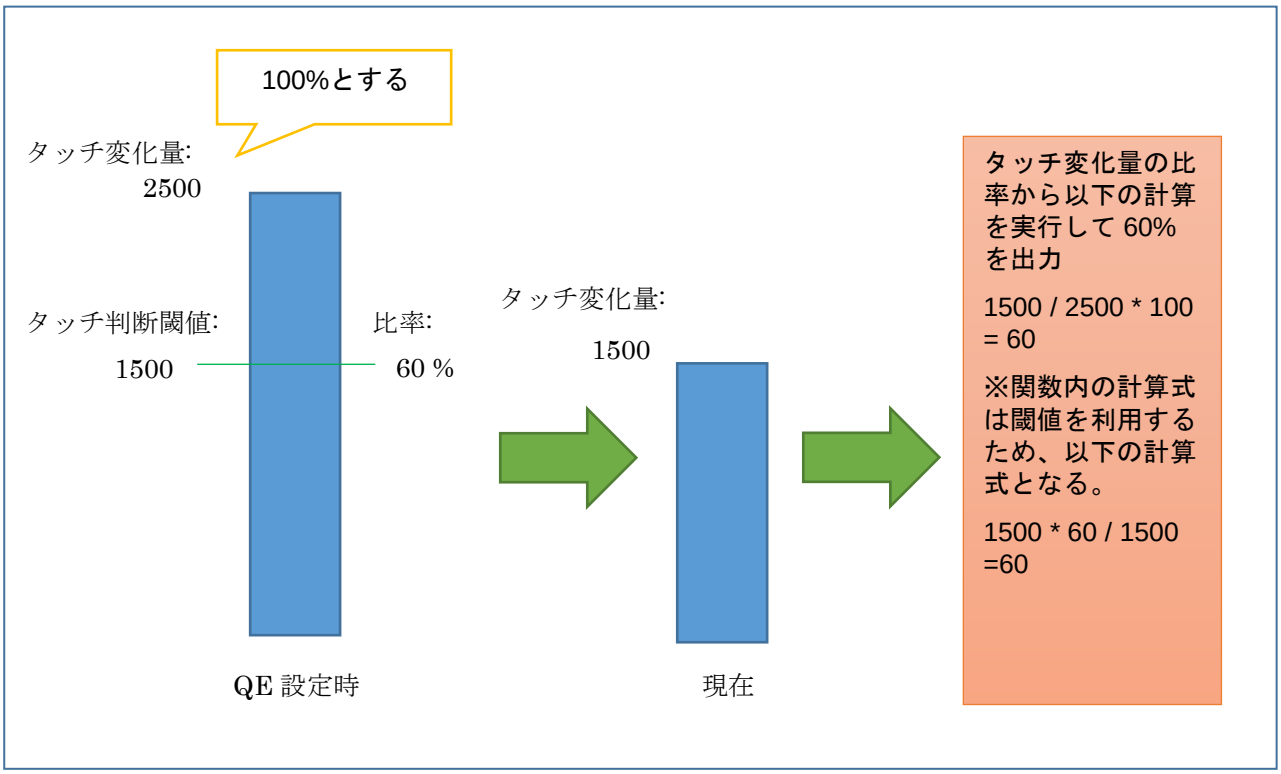
rm_touch_qe.h にプロトタイプ宣言されています。

Description

QE 設定時のタッチ変化量を 100%とし、現在のタッチ変化量の比率を出力します。
オーバレイが薄くなり、タッチ変化量が増えた場合、以下のようになります。



オーバレイが厚くなり、タッチ変化量が減った場合、以下のようになります。



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 = 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]);
```

3.8　RM_TOUCH_ThresholdAdjust

この関数は、タッチ変化量に対するタッチ判断閾値の比率、ヒステリシス値の比率を変更し、現在のタッチ変化量に対応するタッチ判断閾値に変更します。

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(通常は QE によって生成)

p_modifier

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

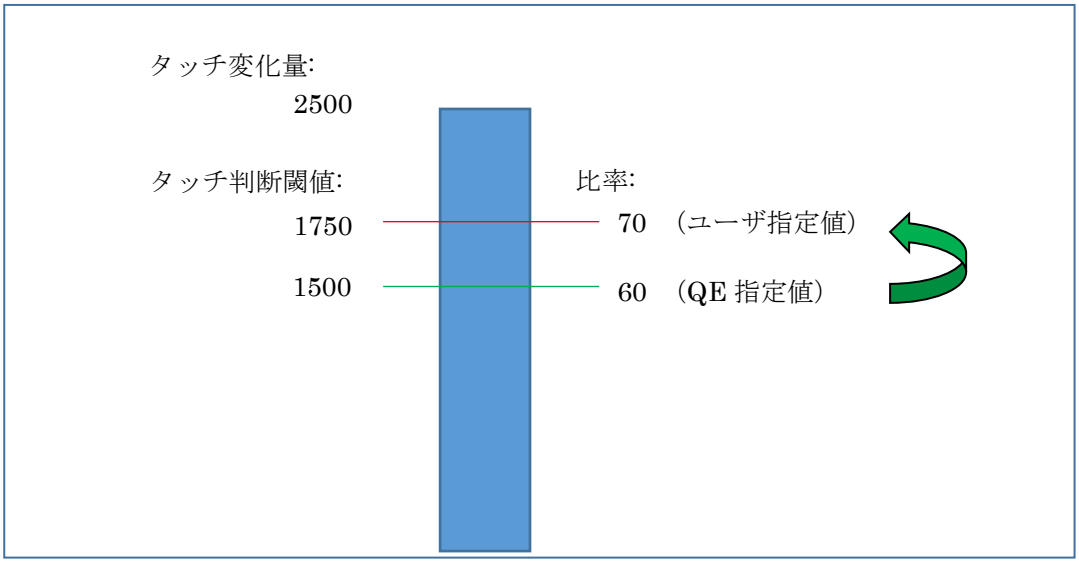
FSP_SUCCESS	*/* タッチ判断閾値の変更に成功しました */*
FSP_ERR_INVALID_POINTER	*/* ポインタが無効なメモリ位置を指しています */*
FSP_ERR_INVALID_ARGUMENT	*/* コンフィグレーションのパラメータが不正です */*

Properties

rm_touch_qe.h にプロトタイプ宣言されています。

Description

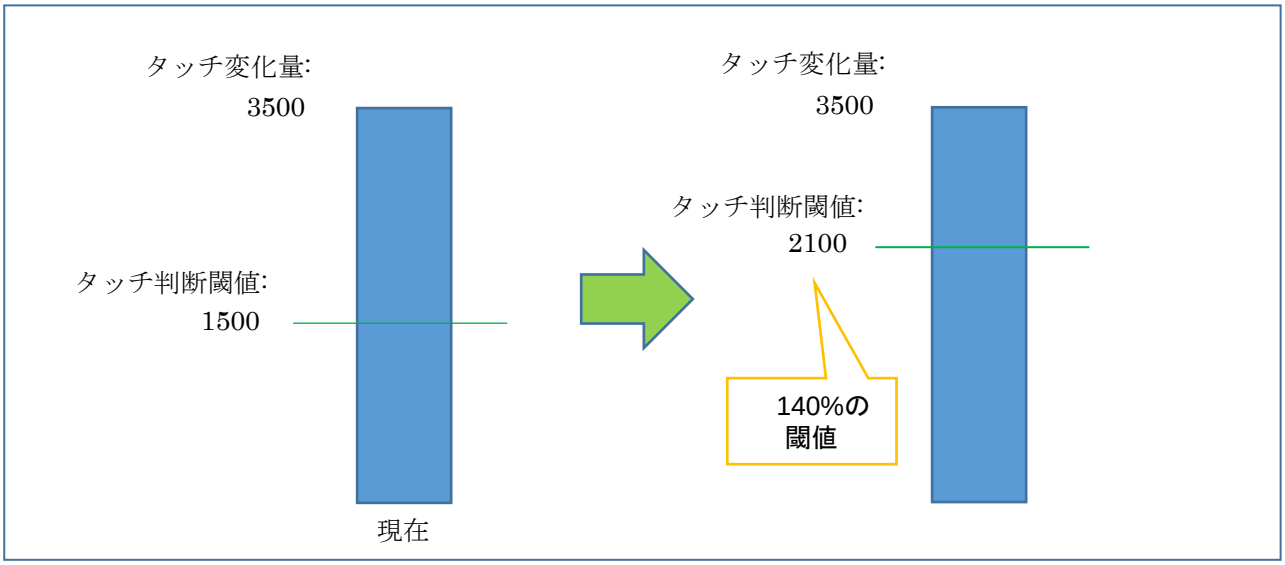
QE のタッチ判断閾値の比率 60% からユーザが指定する閾値の比率 70%とする場合、タッチ判断閾値は以下のようになります。



この設定をしたい場合は第 2 引数のメンバを下記のように設定してください。タッチ変化量の比率およびヒステリシス値の設定も必要です。
*p_touch_sensitivity_ratio = 100
old_threshold_ratio = 60

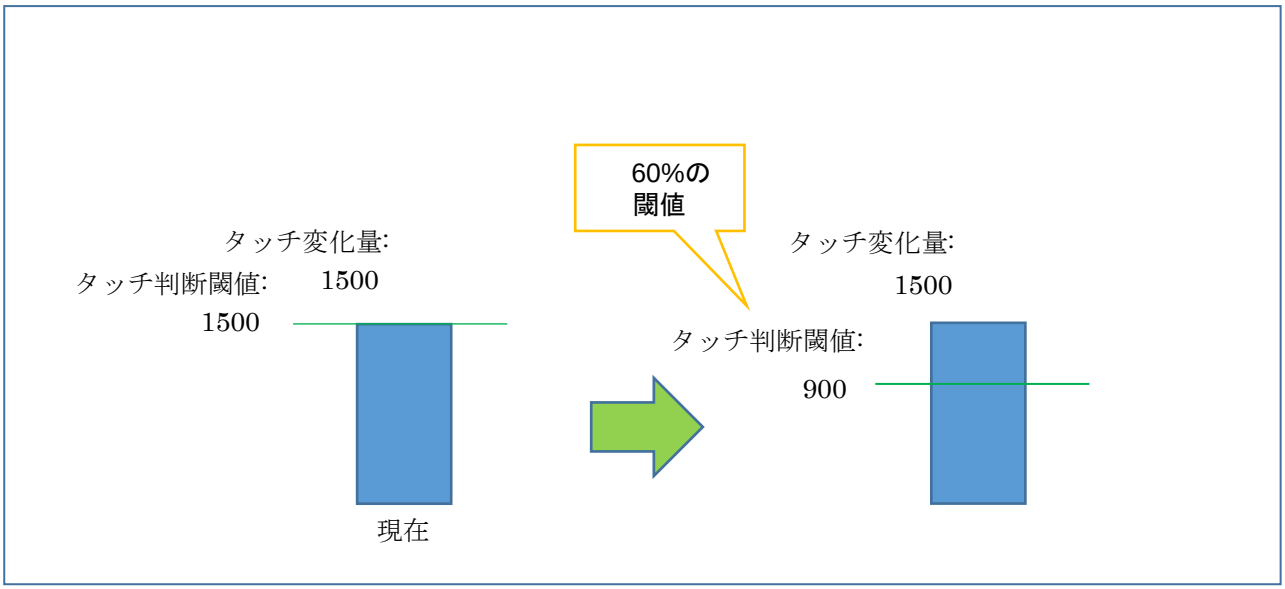
new_threshold_ratio = 70
new_hysteresis_ratio = 5

また、RM_TOUCH_SensitivityRatioGet()を使用して取得したタッチ変化量の比率を引数として新しいタッチ判断閾値を設定する例を示します。
[計算例 1]
タッチ変化量の比率 140%、QE 設定閾値 1500 の場合
140 * 1500 / 100 = 2100



*p_touch_sensitivity_ratio = 140
old_threshold_ratio = 60
new_threshold_ratio = 60
new_hysteresis_ratio = 5

[計算例 2]
タッチ変化量の比率 60%、QE 設定閾値 1500 の場合
60 * 1500 / 100 = 900



*p_touch_sensitivity_ratio = 60
old_threshold_ratio = 60
new_threshold_ratio = 60
new_hysteresis_ratio = 5

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:
QE チューニング時のタッチ変化量と閾値の比率を変更せずにタッチ変化量を変えたい場合は、
RM_TOUCH_ThresholdAdjust()の第 2 引数のメンバを下記のように設定してください。

old_threshold_ratio= 60

new_threshold_ratio = 60

new_hysteresis_ratio = 5

3.9　RM_TOUCH_DriftControl

この関数は、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(通常は QE によって生成)

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*/*

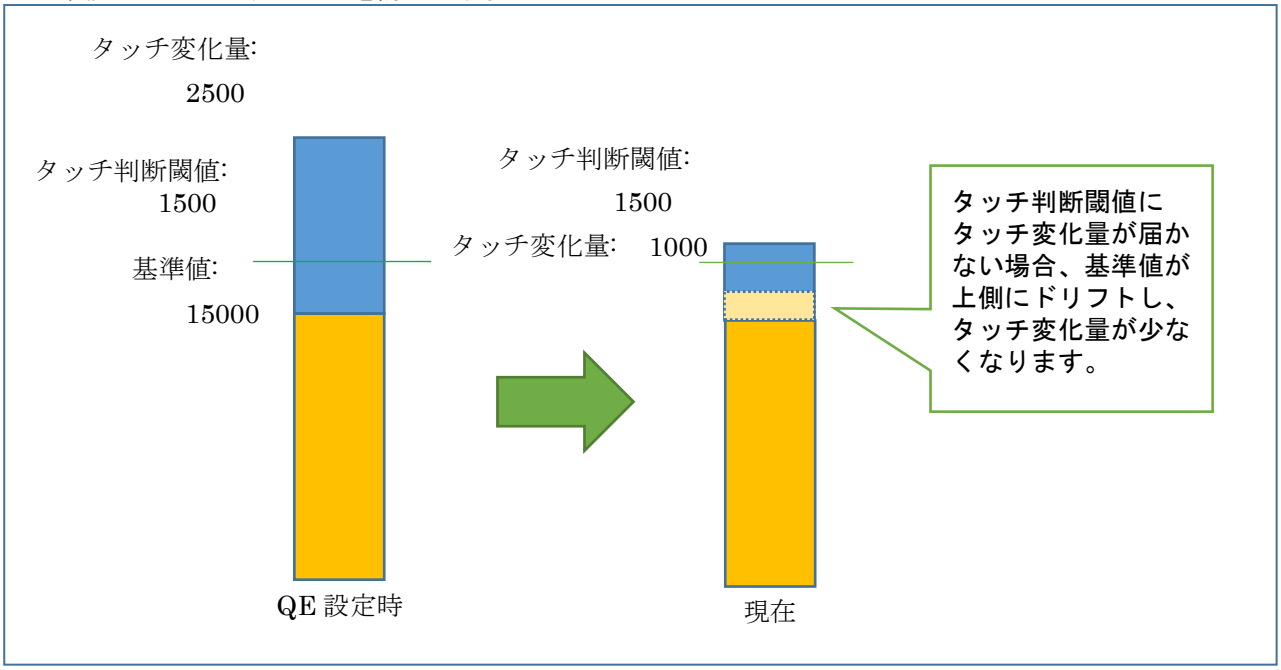
Properties

rm_touch_qe.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。0 にするとドリフト補正機能の停止を設定します。

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);
```

3.10 RM_TOUCH_MonitorAddressGet

この関数は、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,
                                      uint32_t * p_monitor_buf,
                                      uint32_t * p_monitor_id,
                                      uint32_t * p_monitor_size)
```

Parameters

p_ctrl

コントロール構造体へのポインタ(通常は QE によって生成)

p_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

p_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

p_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_qe.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;
uint32_t monitor_buf_address;
uint32_t monitor_id_address;
uint32_t monitor_size_address;

err =  RM_TOUCH_MonitorAddressGet(g_qe_touch_instance_config01.p_ctrl,
                                  &monitor_buf_address,
                                  &monitor_id_address,
                                  &monitor_size_address);
```

Special Notes:

通常は QE が QE モニタ機能のためにサンプルアプリケーション出力で使用する以外は使用しません。

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、47	対象デバイスに RX 23W を追加
		4～6	補正とオフセット調整の定義を追加
		17、24、25、29、31、41～43	API の戻り値を更新
		34、36	R_TOUCH_Control()関数に TOUCH_CMD_GET_FAILED_SENSOR および TOUCH_CMD_GET_LAST_SCAN_METHOD コマンドを追加 オフセット調整処理を R_TOUCH_Open()に移動
		16、18～22	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、22	IEC 60730 準拠に関する内容を追加
1.11	2019/12/11	35～36 34、37 4、6、23、42～43	サンプルコードを更新しました。 低電力アプリケーション用に TOUCH_CMD_CLEAR_TOUCH_STATES を追加しました。 API 関数 R_TOUCH_GetBtnBaselines()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。
2.00	2021/7/30	-	全面改訂
2.01	2021/12/17	10	2.8 コードサイズの内容を修正
2.10	2022/4/20	7	1.2 API 概要に RM_TOUCH_MonitorAddressGet を追加
		29,30	3.10 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。