RA ファミリ

静電容量タッチ ソフトウェアフィルタ サンプルプログラム

要旨

本アプリケーションノートは、静電容量タッチシステム向けのソフトウェアフィルタについて説明します。

動作確認デバイス

RA2L1 グループ (R7FA2L1AB2DFP)

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合わせて変更し、十分評価を行ってください。

目次

1. 概要

本アプリケーションノートでは、ソフトウェアフィルタ サンプルプログラムの動作及び、既存プロジェクトへの組み込み手順について説明します。

ソフトウェアフィルタの詳細については静電容量センサマイコン静電容量タッチノイズイミュニティガイド(R30AN0426)を参照してください。

1.1 フォルダ構成

本サンプルプログラムのフォルダ構成を示します。

本サンプルプログラムは静電容量タッチソフトウェアフィルタ サンプルプログラムの格納フォルダ(Touch_filter_sample_source)と、RA2L1 静電容量タッチ評価システム Example Project (R20AN0595)にソフトウェアフィルタサンプルプログラムを適用したサンプルプロジェクト(ra2l1_rssk_filter_sample)で構成されます。

以降の文中では、RA2L1 静電容量タッチ評価システム Example Project を Example Project と表記します。

```
an-r01an0427jj0300-capacitive-touch
 │
 ├── Touch_filter_sample_source      ・・・フィルタサンプルモジュール格納フォルダ
 │     └──touch_filter_fir           ・・・FIR フィルタサンプルモジュール格納フォルダ
 │     │       └ filter_sample      ・・・FIR フィルタサンプルプログラム
 │     └──touch_filter_iir           ・・・IIR フィルタサンプルモジュール格納フォルダ
 │     │       └ filter_sample      ・・・IIR フィルタサンプルプログラム
 │     └──touch_filter_median        ・・・メディアンフィルタモジュールサンプル格納フォルダ
 │             └ filter_sample      ・・・メディアンフィルタサンプルプログラム
 │
 └── ra2l1_rssk_filter_sample        ・・・フィルタサンプルを実装した Example Project
                                          (RA2L1 静電容量タッチ評価システム用)
```

1.2 動作確認条件

表 1-1 に本アプリケーションノートにおけるサンプルプログラムの動作確認条件を示します

表 1-1 動作確認条件

項目	内容
使用マイコン	RA2L1 (R7FA2L1AB2DFP)
動作周波数	高速オンチップオシレータ 48MHz
動作電圧	5V
使用ボード	RA2L1 搭載静電容量タッチ評価システム (製品型名：RTK0EG0022S01001BJ) ● RA2L1 CPU ボード(型名：RTK0EG0018C01001BJ) ● 自己容量タッチボタン／ホイール／スライダボード (型名：RTK0EG0019B01002BJ)
統合開発環境	e² studio Version 2023-10 (23.10.0)
C コンパイラ	GCC Arm Embedded 10.3-2021.10
FSP	V5.0.0
静電容量式タッチセンサ対応開発支援ツール	QE for Capacitive Touch V3.3.0
エミュレータ	Renesas E2 エミュレータ Lite

図 1-1 に機器接続図を示します。

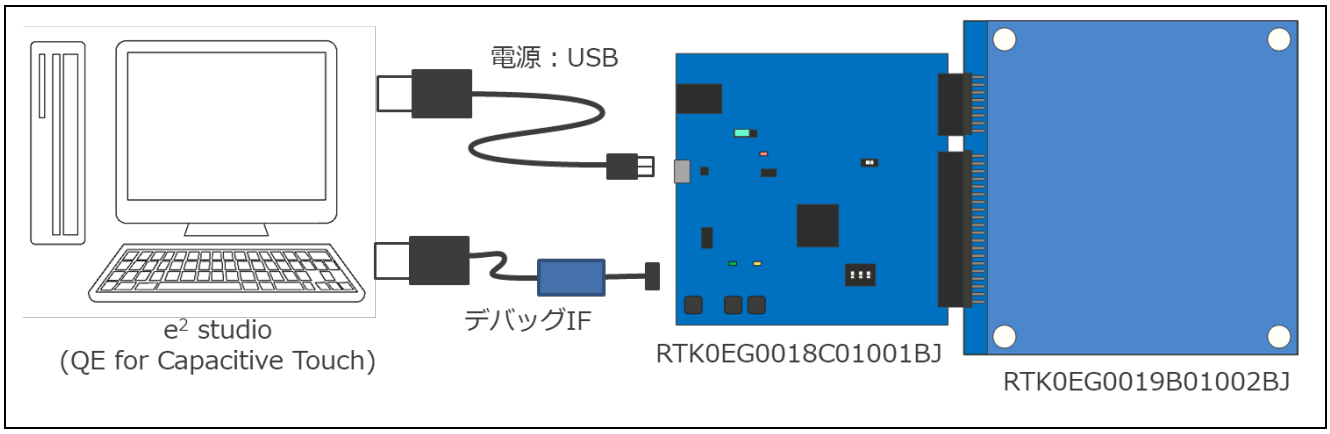


図 1-1 機器接続図

1.3 サンプルコードと本書の対応

本サンプルプロジェクトを使用する前に「2.1 ソフトウェア構成図」および「2.2 ファイル構成」を参照してください。

既存の静電容量タッチプロジェクトへのフィルタモジュールの組み込み方法は「6.1 フィルタサンプルプログラム 」、フィルタサンプルを実装した Example Project の使用方法は「6.2 フィルタサンプルを実装した Example Project」を参照してください。

各フィルタの仕様およびパラメータ設定方法は「2. ソフトウェア仕様」「3. FIR フィルタ」「4. IIR フィルタ」「5. メディアンフィルタ」を参照してください。

2. ソフトウェア仕様

本サンプルプログラムは Touch API と CTSU API で取得したデータにフィルタ API を適用することでソフトウェアフィルタとして動作します。使用するソフトウェアフィルタはフィルタ構成定義で管理します。本サンプルプログラムで定義しているフィルタ構成はフィルタ A(FIR フィルタ)、フィルタ B(IIR フィルタ)、フィルタ C(メディアンフィルタ)ですが、複数のソフトウェアフィルタを適用することも可能です。複数のソフトウェアフィルタを使用する場合の適用順序はフィルタ構成定義の定義順となります。

2.1 ソフトウェア構成図

図 2-1 に本サンプルプログラムの構成を示します。

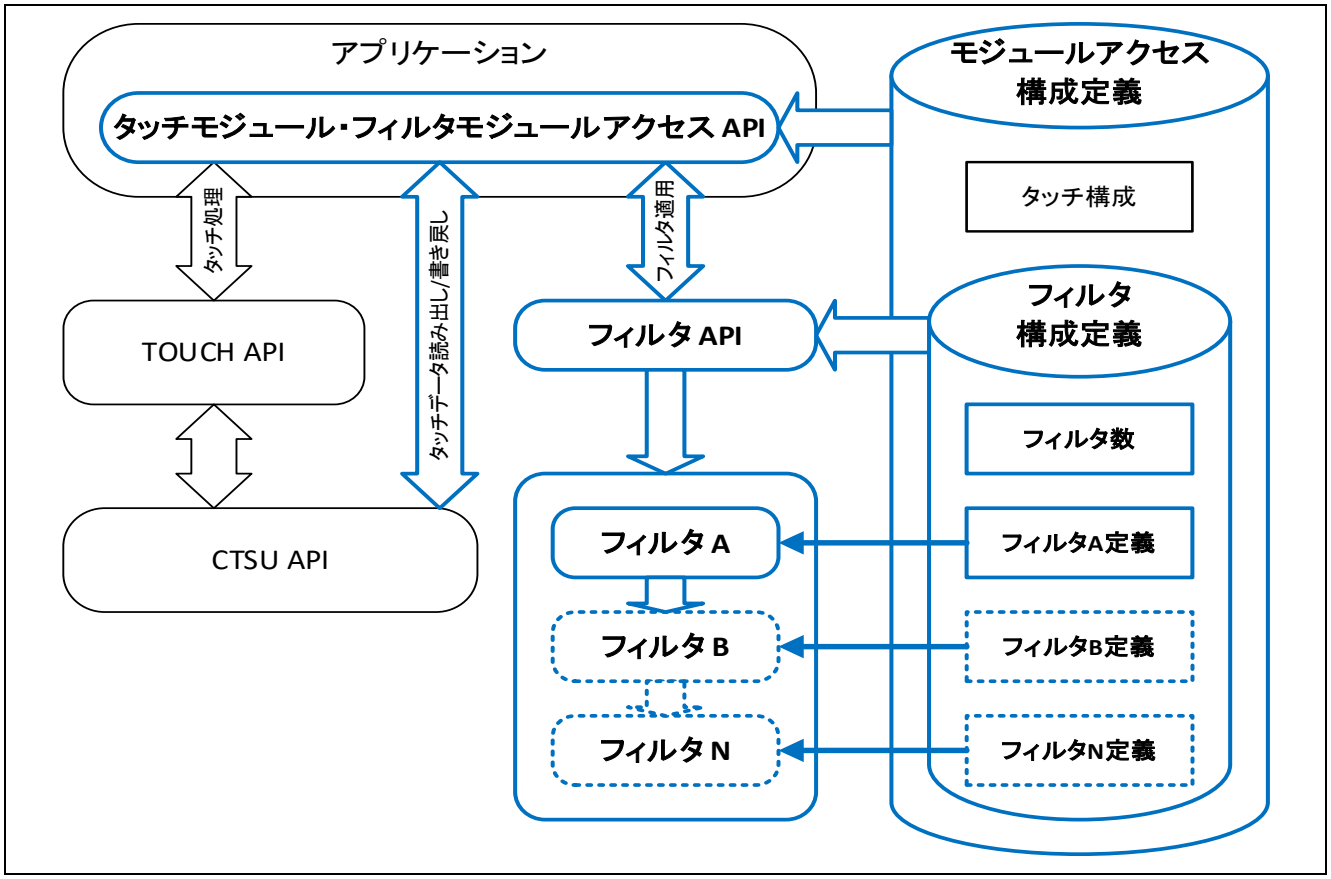


図 2-1 サンプルプログラムの構成

図 2-2 に本サンプルプログラムのデータ処理フローを示します

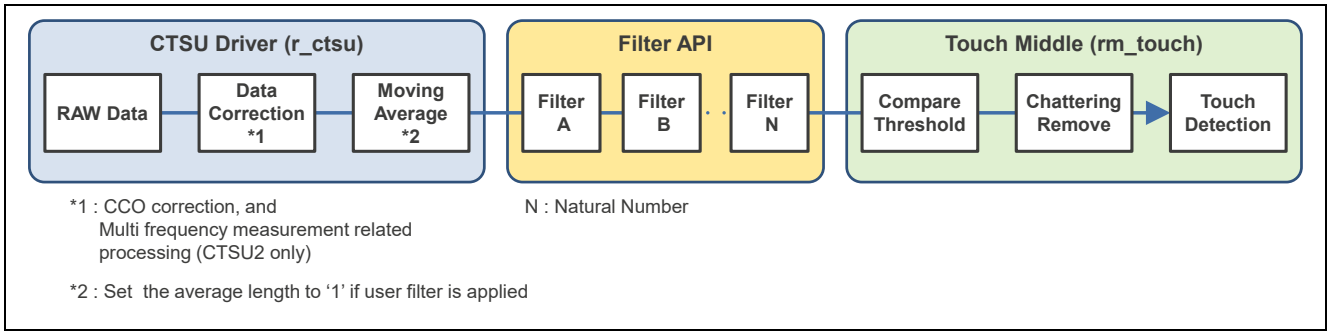


図 2-2 サンプルプログラムのデータ処理フロー

表 2-1 にコンポーネントとバージョンの一覧を示します。コンポーネントの設定は FSP Configuration で参照してください。

表 2-1 コンポーネント一覧

コンポーネント	バージョン
Board Support Packages Common Files	v5.0.0
I/O Port	v5.0.0
Arm CMSIS Version 5 – Core(M)	v5.9.0+renesas.0.fsp.5.0.0
RA2L1-RSSK Board Support Files	v5.0.0
Board support package for R7FA2L1AB2DFP	v5.0.0
Board support package for RA2L1	v5.0.0
Board support package for RA2L1 – FSP Data	v5.0.0
Asynchronous General Purpose Timer	v5.0.0
Capacitive Touch Sensing Unit	v5.0.0
SCI UART	v5.0.0
Touch	v5.0.0

2.2 ファイル構成

本サンプルコードのファイル構成を示します。

```
ra2l1_rssk_filter_sample
├──Touch_filter_sample_source
│      ├──touch_filter_fir                ・・・FIR フィルタサンプルモジュール格納フォルダ
│      │      └──filter_sample
│      │            filter_config_sample.c      ・・・FIR フィルタ構成定義サンプルソース
│      │            filter_config_sample.h      ・・・FIR フィルタ構成定義サンプルヘッダ
│      │            fir_config_sample1.c        ・・・FIR フィルタ用サンプルプリセット 1 ソース
│      │            fir_config_sample2.c        ・・・FIR フィルタ用サンプルプリセット 2 ソース
│      │            fir_config_sample3.c        ・・・FIR フィルタ用サンプルプリセット 3 ソース
│      │            fir_config_sample4.c        ・・・FIR フィルタ用サンプルプリセット 4 ソース
│      │            r_ctsu_filter_sample.c      ・・・フィルタ API サンプルプログラムソース
│      │            r_ctsu_filter_sample.h      ・・・フィルタ API サンプルプログラムヘッダ
│      │            r_ctsu_fir_sample.c         ・・・FIR フィルタサンプルプログラムソース
│      │            r_ctsu_fir_sample.h         ・・・FIR フィルタサンプルプログラムヘッダ
│      ├──touch_filter_iir                ・・・IIR フィルタサンプルモジュール格納フォルダ
│      │      └──filter_sample
│      │            filter_config_sample.c      ・・・IIR フィルタ構成定義サンプルソース
│      │            filter_config_sample.h      ・・・IIR フィルタ構成定義サンプルヘッダ
│      │            iir_config_sample1.c        ・・・IIR フィルタ用サンプルプリセット 1 ソース
│      │            iir_config_sample2.c        ・・・IIR フィルタ用サンプルプリセット 2 ソース
│      │            iir_config_sample3.c        ・・・IIR フィルタ用サンプルプリセット 3 ソース
│      │            iir_config_sample4.c        ・・・IIR フィルタ用サンプルプリセット 4 ソース
│      │            iir_config_sample5.c        ・・・IIR フィルタ用サンプルプリセット 5 ソース
│      │            iir_config_sample6.c        ・・・IIR フィルタ用サンプルプリセット 6 ソース
│      │            r_ctsu_filter_sample.c      ・・・フィルタ API サンプルプログラムソース
│      │            r_ctsu_filter_sample.h      ・・・フィルタ API サンプルプログラムヘッダ
│      │            r_ctsu_iir_sample.c         ・・・IIR フィルタサンプルプログラムソース
│      │            r_ctsu_iir_sample.h         ・・・IIR フィルタサンプルプログラムヘッダ
│      └──touch_filter_median             ・・・メディアンフィルタサンプルモジュール格納フォルダ
│             └──filter_sample
│                   filter_config_sample.c      ・・・メディアンフィルタ構成定義サンプルソース
│                   filter_config_sample.h      ・・・メディアンフィルタ構成定義サンプルヘッダ
│                   median_config_sample1.c     ・・・メディアンフィルタ用サンプルプリセット 1 ソース
│                   median_config_sample2.c     ・・・メディアンフィルタ用サンプルプリセット 2 ソース
│                   r_ctsu_filter_sample.c      ・・・フィルタ API サンプルプログラムソース
│                   r_ctsu_filter_sample.h      ・・・フィルタ API サンプルプログラムヘッダ
│                   r_ctsu_median_sample.c      ・・・メディアンフィルタサンプルプログラムソース
│                   r_ctsu_median_sample.h      ・・・メディアンフィルタサンプルプログラムヘッダ
└──ra2l1_rssk_filter_sample             ・・・フィルタサンプルを実装した Example Project
```

2.3 フィルタ構成定義用データー覧

ソフトウェアフィルタサンプルプログラムでフィルタ構成定義用に用意している定数・グローバル変数・構造体について説明します。

2.3.1 定数

表 2-2 に定数の一覧を示します。

表 2-2 フィルタ構成定義用定数

定数名	設定値	内容
ファイル名：filter_config_sample.h		
CTSU_FILTER_NUM	1～3	使用するフィルタの直列接続数 (フィルタ種類、プリセット定義によって値が異なります)
FILTER_ELEMENT_SIZE	CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS × 2)	CTSU API で取得する計測結果個数 (タッチインターフェース構成の定義から計算されます)
ファイル名：r_ctsu_filter_sample.c		
CTSU_IF_MAX	8	タッチインターフェース構成の最大定義数
FILTER_SIZE	CTSU_FILTER_NUM × CTSU_IF_MAX	フィルタ管理データ用の個数
FILTER_RESULT_MIN	0	フィルタ適用結果の最小値
FILTER_RESULT_MAX	65535	フィルタ適用結果の最大値
CTSU_FILTER_OPEN	0x464C5452	初期化済み管理用定義値
ファイル名：r_ctsu_fir_sample.h		
FIR_FILTER_ENABLE	0～1	FIR フィルタ有効、無効定義（0=無効、1=有効）
ファイル名：r_ctsu_iir_sample.h		
IIR_FILTER_ENABLE	0～1	IIR フィルタ有効、無効定義（0=無効、1=有効）
ファイル名：r_ctsu_median_sample.h		
MEDIAN_FILTER_ENABLE	0～1	メディアンフィルタ有効、無効定義（0=無効、1=有効）

2.3.2 列挙型

表 2-3 に filter_type_t の列挙型を示します。

表 2-3 filter_type_t

メンバ	値	説明
FILTER_TYPE_NONE	0	フィルタ種類：フィルタ無し
FILTER_TYPE_FIR	1	フィルタ種類：FIR フィルタ
FILTER_TYPE_IIR	2	フィルタ種類：IIR フィルタ
FILTER_TYPE_MEDIAN	3	フィルタ種類：メディアンフィルタ

2.3.3 グローバル変数

表 2-4 にグローバル変数の一覧を示します。

表 2-4 フィルタ構成定義用グローバル変数

変数名	型	説明
ファイル名：r_ctsu_filter_sample.c		
g_ctsu_filter_element_index	uint16_t	管理データ割り当て用インデクス
g_ctsu_filter_element_control	filter_element_ctrl_t	フィルタ管理データ (使用するフィルタ総数×計測結果個数分のデータサイズが定義されます)

2.3.4 構造体

qe_touch_sample.c 内での API アクセス用構造体と filter_config_sample.c 内でフィルタ個数と各フィルタの種類を定義している構造体を示します。

表 2-5 フィルタ構成定義

定義内容	データ型	備考
ファイル名：qe_touch_sample.c		
タッチモジュール、フィルタモジュールアクセス用定義	filtering_instance_t	r_rssk_filter_initialize()、 r_rssk_filter_dataget()用定義
ファイル名：filter_config_sample.c		
フィルタ管理用定義	ctsu_filter_instance_t	タッチインターフェース構成のメソッド毎に用意してください。
フィルタ管理データ	filter_instance_ctrl_t	
フィルタ構成定義	filter_config_t	タッチインターフェース構成のメソッド毎に使用するフィルタ内容を変更する場合はフィルタ内容毎に定義を用意してください。
フィルタ内容定義	filter_element_config_t	

2.3.4.1 タッチモジュール、フィルタモジュールアクセス用定義(filtering_instance_t)

表 2-6 タッチモジュール、フィルタモジュールアクセス用定義構造体(filtering_instance_t)

メンバ	データ型	内容
p_touch_instance	touch_instance_t const *	タッチモジュール管理用定義ポインタ
p_filter_instance	ctsu_filter_instance_t const *	フィルタ管理用定義ポインタ

- タッチモジュール、フィルタモジュールアクセス用定義構造体(filtering_instance_t)の記述例

```
filtering_instance_t g_qe_filtering_instance_config[] =
{
  {
    .p_touch_instance = &g_qe_touch_instance_config01,
    .p_filter_instance = &g_ctsu_filter_instance01,
  },
};
```

2.3.4.2 フィルタ管理用定義(ctsu_filter_instance_t)

表 2-7 フィルタ管理用定義構造体(ctsu_filter_instance_t)

メンバ	データ型	内容
p_ctrl	filter_ctrl_t *	フィルタ管理データポインタ (このデータポインタは void 型ポインタとして定義されています) フィルタ管理データ (filter_instance_ctrl_t)で定義した変数を割り当てて使用してください)
p_cfg	filter_config_t const *	フィルタ構成定義ポインタ
p_api	filter_api_t const *	フィルタ API ポインタ

- フィルタ管理用定義(ctsu_filter_instance_t)の記述例

```
filter_instance_ctrl_t g_ctsu_filter_control01;
const ctsu_filter_instance_t g_ctsu_filter_instance01 =
{
  .p_ctrl = &g_ctsu_filter_control01,
  .p_cfg  = &g_ctsu_filter_config,
  .p_api  = &g_filter_on_ctsu,
};
```

タッチインターフェース構成のメソッド毎に管理用データと使用するフィルタ構成を定義します。

2.3.4.3 フィルタ管理データ(filter_instance_ctrl_t)

表 2-8 フィルタ管理データ用構造体(filter_instance_ctrl_t)

メンバ	データ型	内容
open	uint32_t	初期化状態
p_cfg	filter_config_t const *	フィルタ構成定義ポインタ
p_element_ctrl	filter_element_ctrl_t *	フィルタ個別管理データポインタ

2.3.4.4 フィルタ個別管理データ(filter_element_ctrl_t)

表 2-9 フィルタ個別管理データ用構造体(filter_element_ctrl_t)

メンバ	データ型	内容
element_num	uint16_t	計測結果個数
p_filter_ctrl	filter_ctrl_t *	フィルタ毎の管理データポインタ

2.3.4.5 フィルタ構成定義(filter_config_t)

表 2-10 フィルタ構成定義用構造体(filter_config_t)

メンバ	データ型	内容
filter_num	uint8_t	使用するフィルタの直列接続数
p_filter_cfg	filter_element_config_t	フィルタ内容定義ポインタ

- フィルタ構成定義(filter_config_t)の記述例
  適用するフィルタパターン毎にフィルタ個数とフィルタ種類を定義します。

```
const filter_config_t g_ctsu_filter_config =
{
  .filter_num      = CTSU_FILTER_NUM,
  .p_filter_cfg = g_ctsu_filter_element_config,
};
```

2.3.4.6 フィルタ内容定義(filter_element_config_t)

表 2-11 フィルタ内容定義用構造体(filter_element_config_t)

メンバ	データ型	内容
type	filter_type_t	フィルタ種類
filter_element_cfg	filter_ctrl_t *	フィルタ毎の構成定義ポインタ

- フィルタ内容定義(filter_element_config_t)の記述例

```
const filter_element_config_t g_ctsu_filter_element_config[] =
{
  {
    .type  = FILTER_TYPE_FIR,
    .filter_element_cfg = &fir_cfg01,
  },
};
```

使用するフィルタの種類を適用する順に定義します

2.4 ソフトウェアフィルタ API

表 2-12 に本サンプルプログラムで実装されているソフトウェアフィルタ API の一覧を示します。

表 2-12 ソフトウェアフィルタ API 一覧

関数名	処理概要
ファイル名：qe_touch_sample.c	
qe_touch_main	メイン処理
r_rssk_filter_initialize	タッチモジュール、フィルタ初期化処理
r_rssk_filter_dataget	フィルタ実施済みタッチ結果取得処理
timer0_callback	AGT 割り込みコールバック
r_rssk_initialize	静電容量タッチ評価システムの LED 初期化処理
r_rssk_led_test	静電容量タッチ評価システムの LED テスト処理
ファイル名：r_ctsu_filter_sample.c	
r_ctsu_filter_open	フィルタ初期化処理
ctsu_fir_filter_open	FIR フィルタ用初期化処理
ctsu_iir_filter_open	IIR フィルタ用初期化処理
ctsu_median_filter_open	メディアンフィルタ用初期化処理
r_ctsu_filter_exec	フィルタ実行処理

2.4.1 r_rssk_filter_initialize

この関数は、タッチモジュールとソフトウェアフィルタの初期化をする関数です。この関数は他のタッチモジュール、ソフトウェアフィルタ API 関数を使用する前に実行する必要があります。タッチインターフェース毎に実行してください。

Format

fsp_err_t r_rssk_filter_initialize (filtering_instance_t * const p_ctrl);

Parameters

p_ctrl

タッチミドルウェア、フィルタ管理用定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_ALREADY_OPEN	/* 初期化実施済みです */
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

qe_touch_sample.c にプロトタイプ宣言されています。

Description

この関数は、RM_TOUCH_Open()、r_ctsu_filter_open()をコールしてタッチモジュールとソフトウェアフィルタを初期化します。

Example

```
/* Open Touch middleware and filter sample */
err = r_rssk_filter_initialize(&g_qe_filtering_instance_config[0]);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

Special Notes:

本関数は QE 生成コードの RM_TOUCH_Open()のコール箇所と差し替えての使用を想定しています。

2.4.2 r_rssk_filter_dataget

この関数は、タッチ計測結果にソフトウェアフィルタを適用し、フィルタ適用後のタッチ状態を取得します。

Format

fsp_err_t r_rssk_filter_dataget (filtering_instance_t * const p_ctrl, uint64_t * p_button_status, uint16_t * p_slider_position, uint16_t * p_wheel_position);

Parameters

p_ctrl

タッチミドルウェア、フィルタ管理用定義へのポインタ

p_button_status

ボタン状態を格納するバッファへのポインタ

p_sliderbutton_status

スライダ位置を格納するバッファへのポインタ

p_button_status

ホイール位置を格納するバッファへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_NOT_OPEN	/* Open のコールなしにコールされました */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

qe_touch_sample.c にプロトタイプ宣言されています。

Description

この関数は、R_CTSU_DataGet()、r_ctsu_filter_exec()、R_CTSU_DataInsert()をコールして計測値にソフトウェアフィルタを適用し、正常に適用できた場合は RM_TOUCH_DataGet()をコールしてタッチ判定および位置検出をします。

Example

```
  /* Use filter sample software */
  err =
r_rssk_filter_dataget(&g_qe_filtering_instance_config[0],&button_status, NULL,
NULL);
  if (FSP_SUCCESS == err)
  {
    /* TODO: Add your own code here. */
  }
```

Special Notes:

本関数は QE 生成コードの RM_TOUCH_DataGet()のコール箇所と差し替えての使用を想定しています。

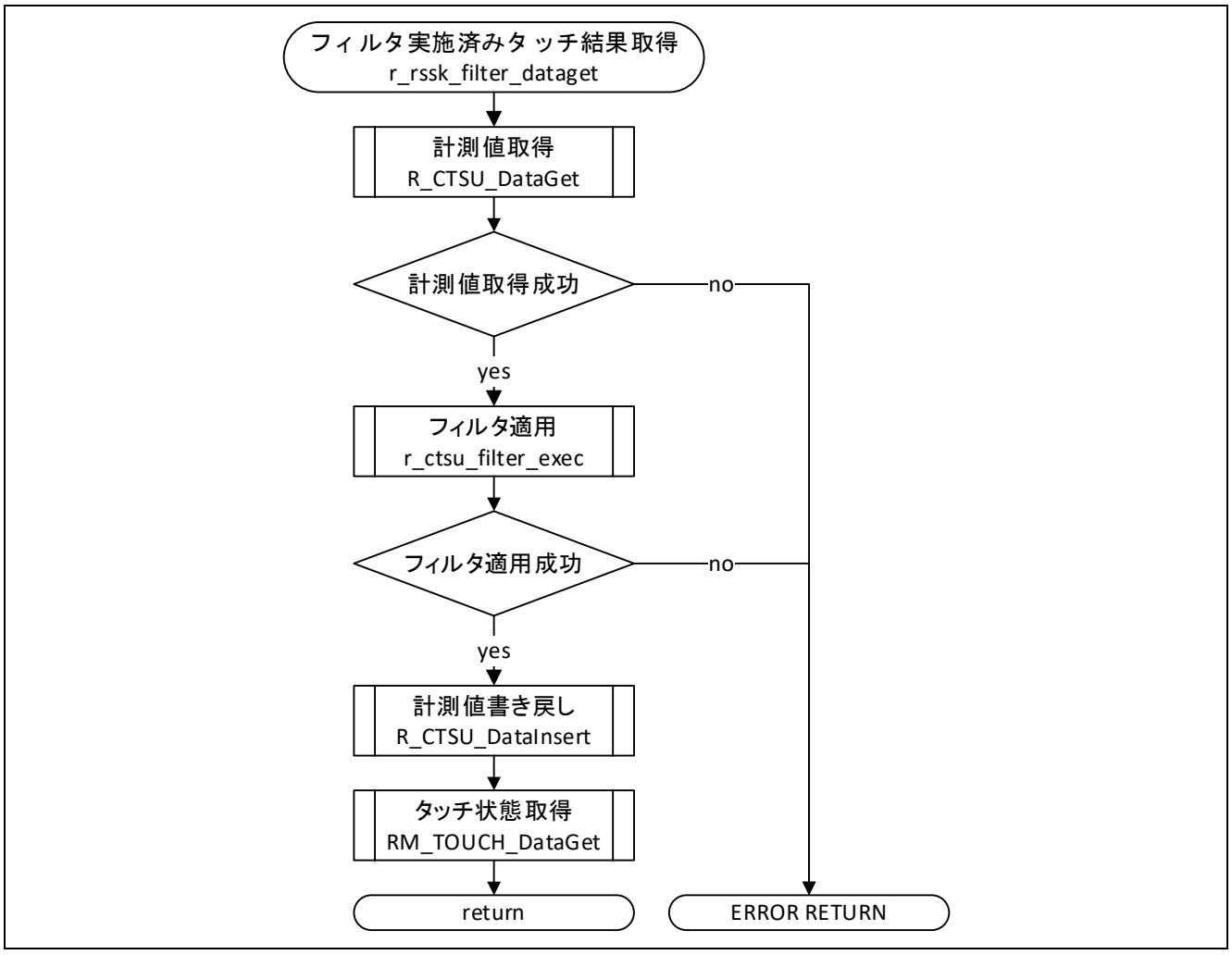


図 2-3 フィルタ実施済みタッチ結果取得 API フロー

2.4.3 r_ctsu_filter_open

この関数は、ソフトウェアフィルタを初期化する関数です。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。使用するタッチインターフェース分のフィルタ管理用データとコンフィグレーション定義を用意し、タッチインターフェース毎に実行してください。

Format

fsp_err_t r_ctsu_filter_open(filter_ctrl_t * const p_ctrl , filter_config_t const * const p_cfg , ctsu_cfg_t const * const p_ctsu_cfg);

Parameters

p_ctrl

フィルタ管理用データへのポインタ

p_cfg

ソフトウェアフィルタのコンフィグレーション定義へのポインタ

p_ctsu_cfg

CTSU ドライバのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_ALREADY_OPEN	/* 初期化実施済みです */
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

r_ctsu_filter_sample.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、引数 p_ctsu_cfg に従ってフィルタ管理用データの初期設定を行います。

Example

```
  /* Open filter sample software */
  err = r_ctsu_filter_open(g_ctsu_filter_instance01.p_ctrl,
g_ctsu_filter_instance01.p_cfg, g_qe_ctsu_instance_config01.p_cfg);
  if (FSP_SUCCESS != err)
  {
   while (true) {}
  }
```

Special Notes:

本関数では CTSU ドライバのコンフィグレーション定義を参照して計測モードが自己容量の場合は「端子数」回、相互容量の場合は「送信端子数×受信端子数×2」回、各フィルタモジュールのフィルタ初期化 API をコールしています。以下の API 説明も参照してください。

- FIR フィルタ初期化 API：r_ctsu_fir_open
- IIR フィルタ初期化 API：r_ctsu_iir_open
- メディアンフィルタ初期化 API：r_ctsu_median_open

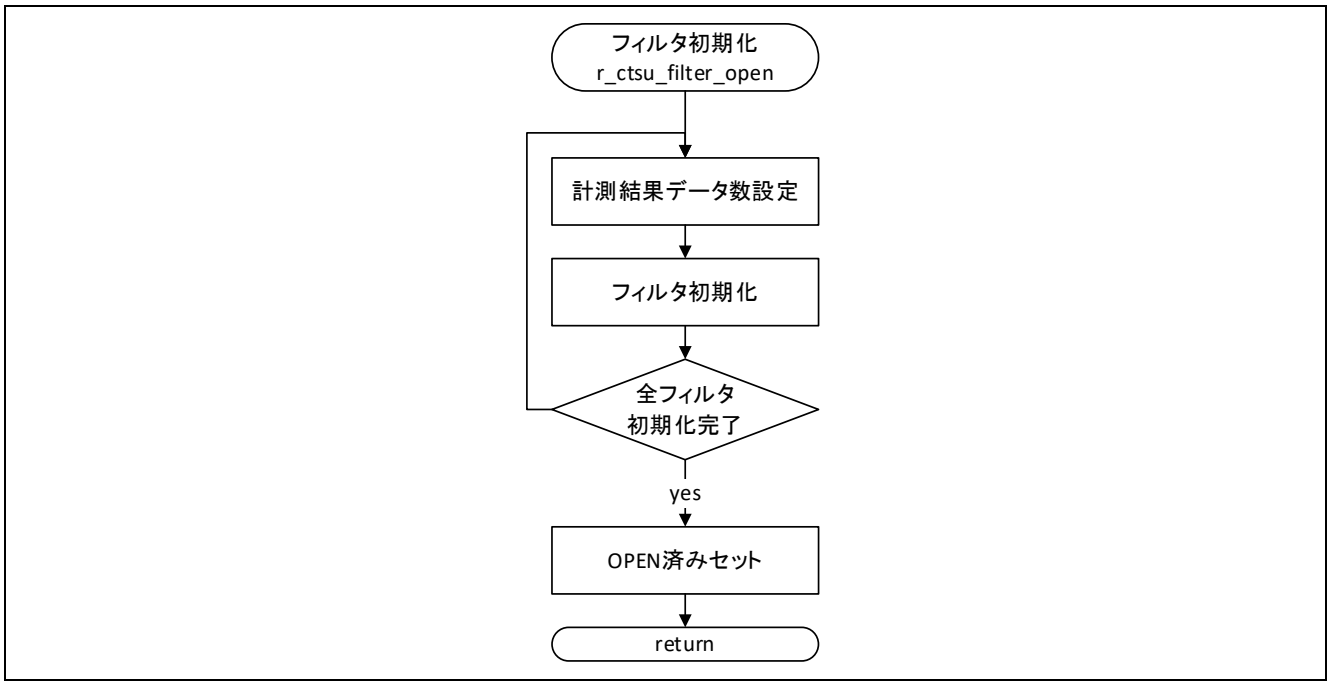


図 2-4 フィルタ初期化 API フロー

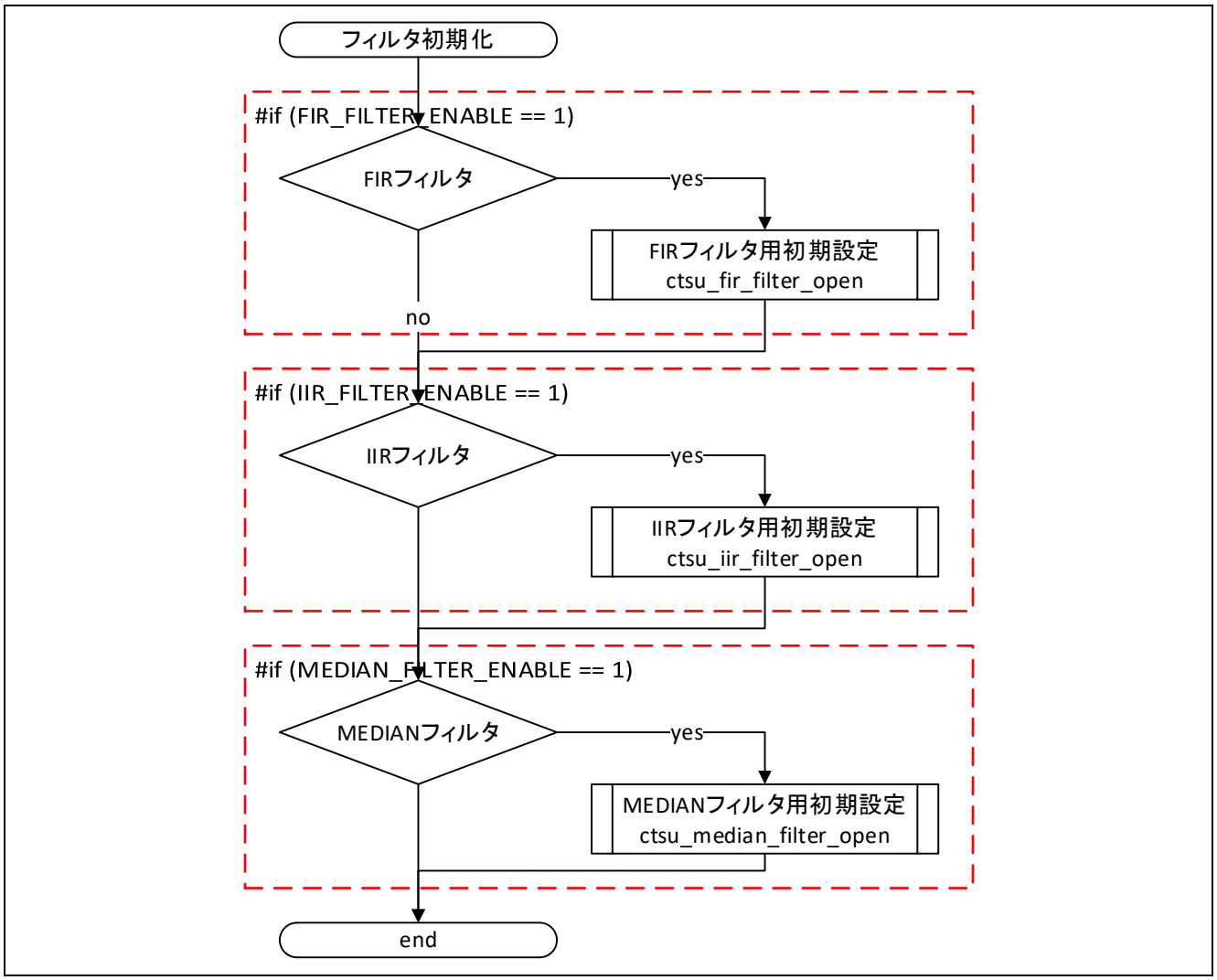


図 2-5 フィルタ初期化 API 内フィルタ初期化処理フロー

2.4.4 ctsu_fir_filter_open

この関数は、FIR フィルタ使用時に r_ctsu_filter_open()からコールされる関数です。この関数は FIR フィルタ用の管理データ割り当てと r_ctsu_fir_open()のコールを行います。

Format

static fsp_err_t ctsu_fir_filter_open (filter_element_ctrl_t * p_ctrl, filter_element_config_t * p_cfg);

Parameters

p_ctrl

フィルタ個別管理データへのポインタ

p_cfg

FIR フィルタのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_filter_sample.c にプロトタイプ宣言されています。

Description

この関数は、引数 p_ctrl、引数 p_cfg に従って FIR フィルタ管理用データの割り当てと初期設定を行います。

Example

```
if(p_filter_cfgs->type == FILTER_TYPE_FIR)
{
  ret = ctsu_fir_filter_open(&p_instance_ctrl->p_element_ctrl[filter_id],
p_filter_cfgs->filter_element_cfg);
}
```

Special Notes:

本関数では r_ctsu_filter_open()で初期設定した計測結果個数を参照して、FIR フィルタモジュールのフィルタ初期化 API をコールしています。以下の API 説明も参照してください。

- FIR フィルタ初期化 API：r_ctsu_fir_open

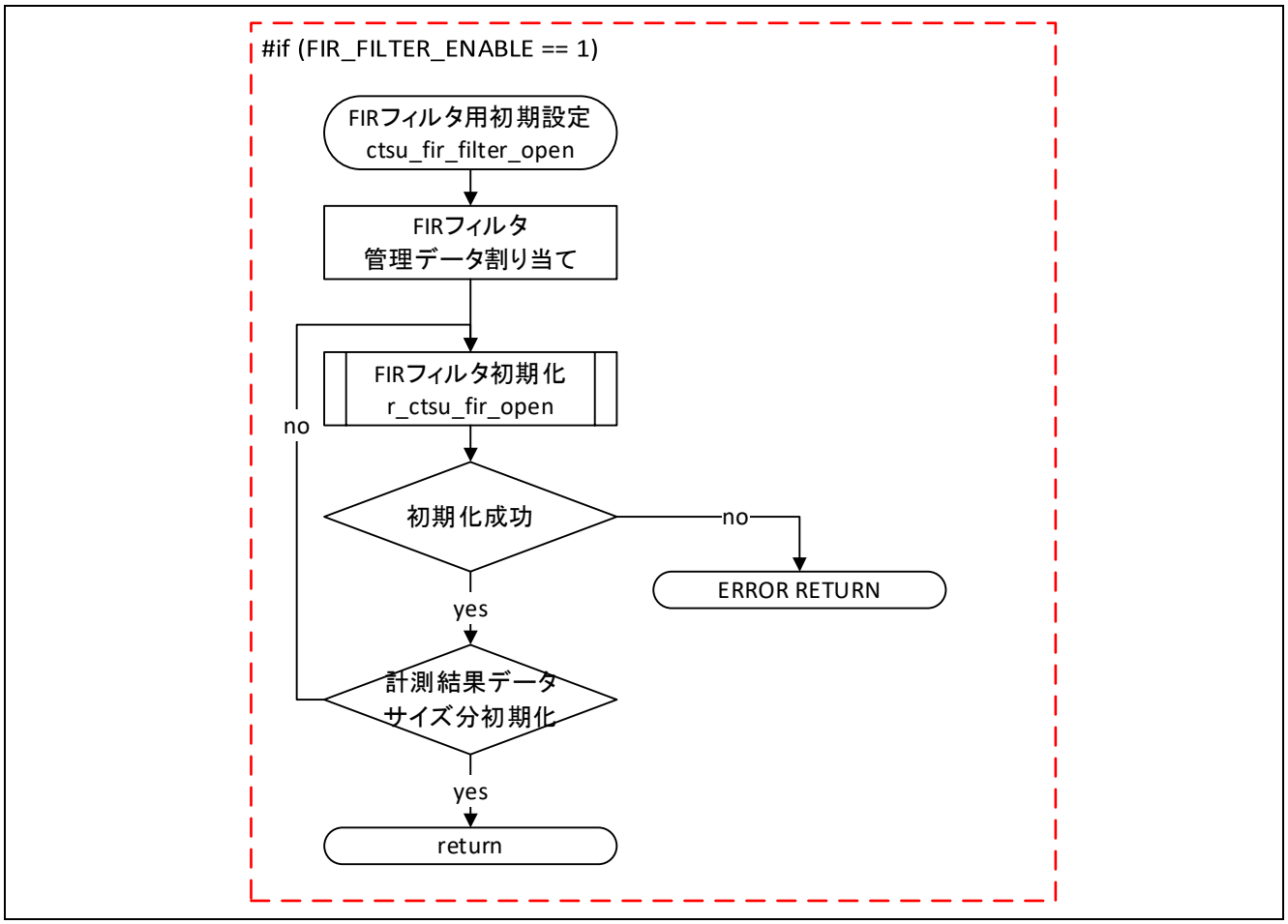


図 2-6 FIR フィルタ用初期設定 API フロー

2.4.5 ctsu_iir_filter_open

この関数は、IIR フィルタ使用時に r_ctsu_filter_open()からコールされる関数です。この関数は IIR フィルタ用の管理データ割り当てと r_ctsu_iir_open()のコールを行います。

Format

static fsp_err_t ctsu_iir_filter_open (filter_element_ctrl_t * p_ctrl, filter_element_config_t * p_cfg);

Parameters

p_ctrl

フィルタ個別管理データへのポインタ

p_cfg

IIR フィルタのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_filter_sample.c にプロトタイプ宣言されています。

Description

この関数は、引数 p_ctrl、引数 p_cfg に従って IIR フィルタ管理用データの割り当てと初期設定を行います。

Example

```
if(p_filter_cfgs->type == FILTER_TYPE_IIR)
{
 ret = ctsu_iir_filter_open(&p_instance_ctrl->p_element_ctrl[filter_id],
p_filter_cfgs->filter_element_cfg);
}
```

Special Notes:

本関数では r_ctsu_filter_open()で初期設定した計測結果個数を参照して、IIR フィルタモジュールのフィルタ初期化 API をコールしています。以下の API 説明も参照してください。

- IIR フィルタ初期化 API：r_ctsu_iir_open

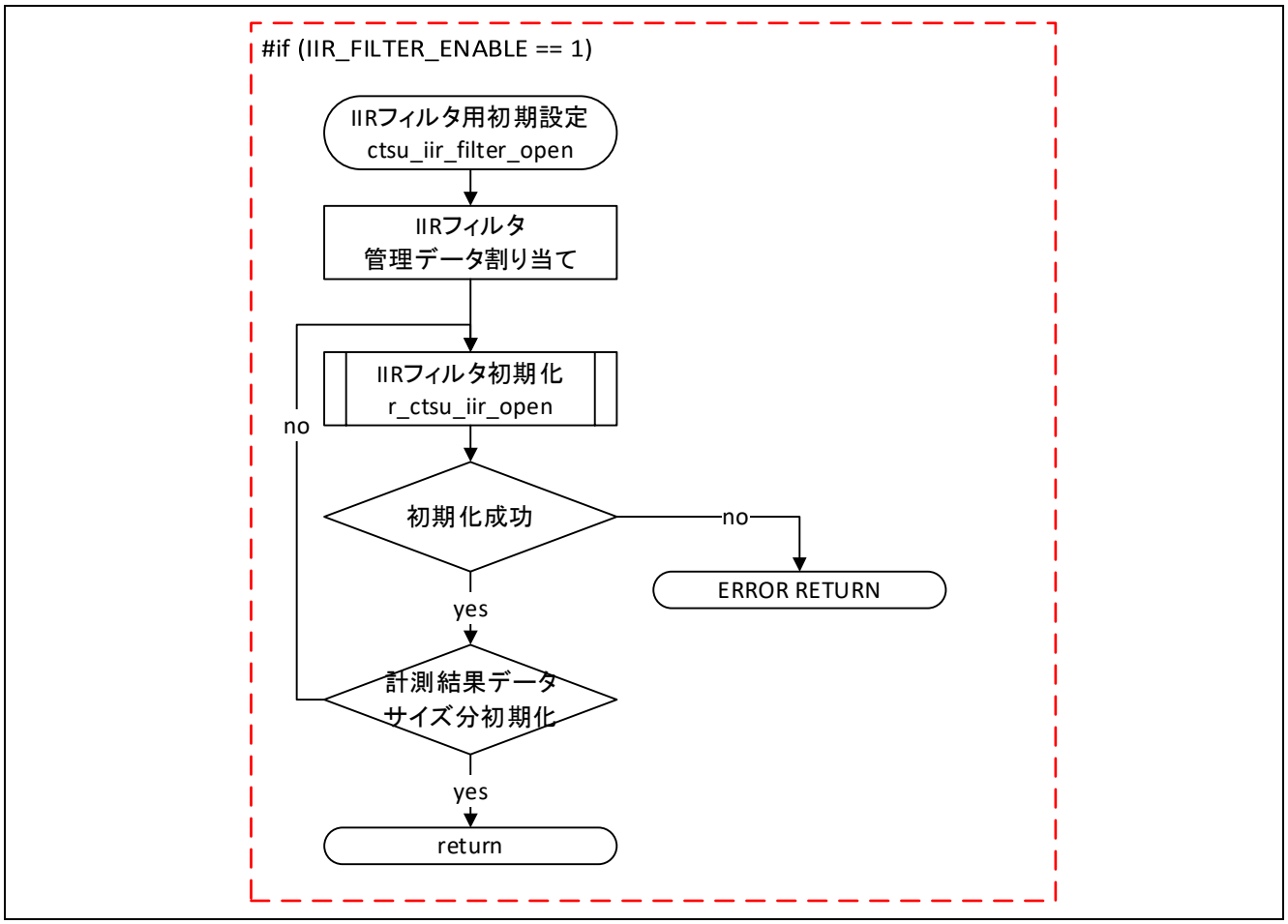


図 2-7 IIR フィルタ用初期設定 API フロー

2.4.6 ctsu_median_filter_open

この関数は、メディアンフィルタ使用時に r_ctsu_filter_open()からコールされる関数です。この関数はメディアンフィルタ用の管理データ割り当てと r_ctsu_median_open()のコールを行います。

Format

static fsp_err_t ctsu_median_filter_open (filter_element_ctrl_t * p_ctrl, filter_element_config_t * p_cfg);

Parameters

p_ctrl

フィルタ個別管理データへのポインタ

p_cfg

メディアンフィルタのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_filter_sample.c にプロトタイプ宣言されています。

Description

この関数は、引数 p_ctrl、引数 p_cfg に従ってメディアンフィルタ管理用データの割り当てと初期設定を行います。

Example

```
if(p_filter_cfgs->type == FILTER_TYPE_MEDIAN)
{
 ret = ctsu_median_filter_open(&p_instance_ctrl->p_element_ctrl[filter_id],
p_filter_cfgs->filter_element_cfg);
}
```

Special Notes:

本関数では r_ctsu_filter_open()で初期設定した計測結果個数を参照して、メディアンフィルタモジュールのフィルタ初期化 API をコールしています。以下の API 説明も参照してください。

- メディアンフィルタ初期化 API：r_ctsu_median_open

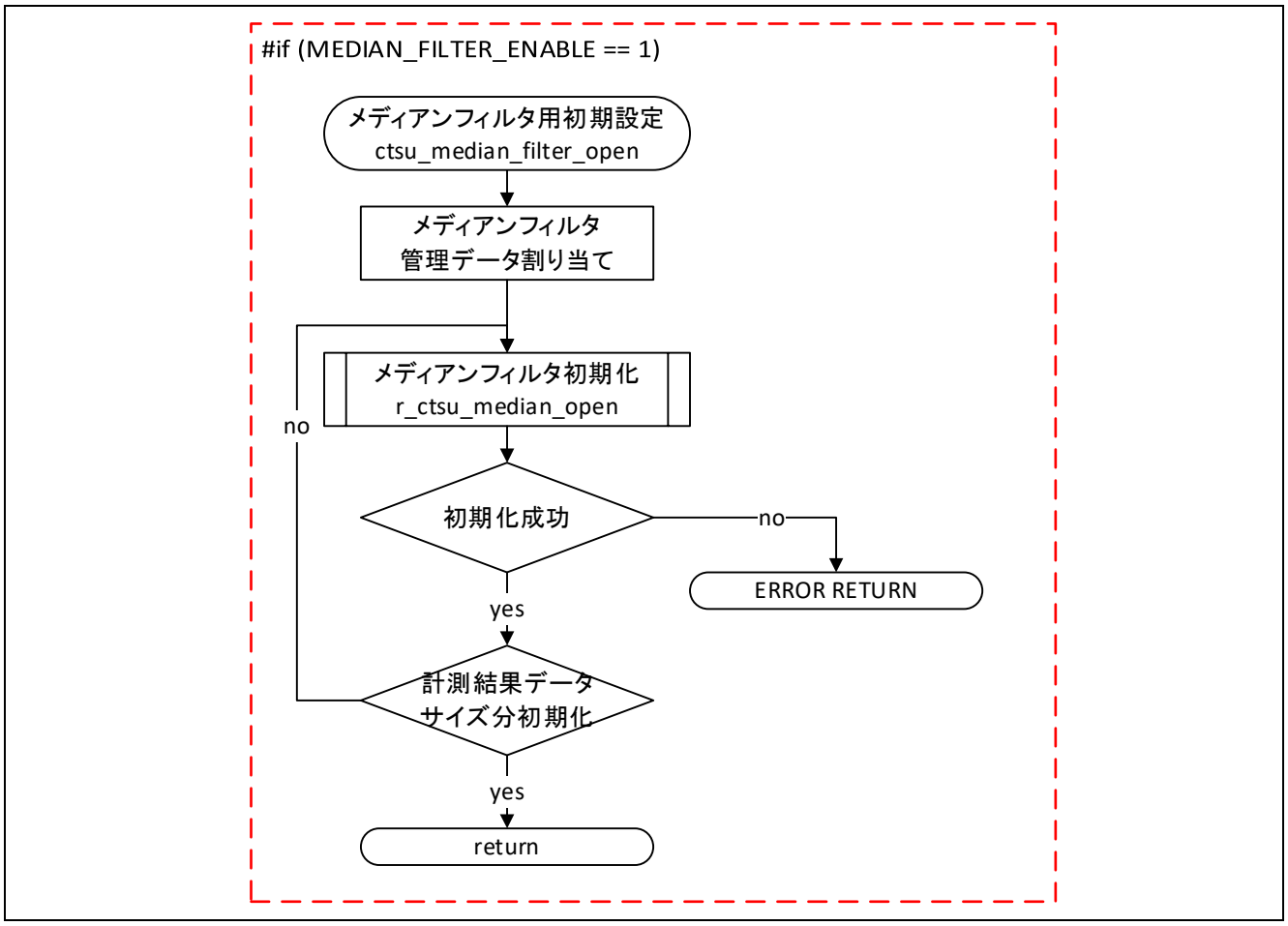


図 2-8 メディアンフィルタ用初期設定 API フロー

2.4.7 r_ctsu_filter_exec

この関数は、計測結果データにソフトウェアフィルタを適用します。

Format

fsp_err_t r_ctsu_filter_exec(filter_ctrl_t * const p_ctrl , uint16_t *p_data);

Parameters

p_ctrl

フィルタ管理用データへのポインタ

p_data

入出力データバッファへのポインタ。

指定した計測値データにフィルタを適用し、フィルタ適用結果で上書きします。

ReturnValues

FSP_SUCCESS	/* 正常終了 */
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()のコールなしに実行されました */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_filter_sample.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、R_CTSU_DataInsert 関数と組み合わせて使用し、タッチ計測データにフィルタ構成で定義したフィルタを適用します。

Example

```
/* Use filter sample software */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config01.p_ctrl, g_filter_buffer);
if (FSP_SUCCESS == err)
{
 r_ctsu_filter_exec(g_ctsu_filter_instance01.p_ctrl, g_filter_buffer);
 R_CTSU_DataInsert(g_qe_ctsu_instance_config01.p_ctrl, g_filter_buffer);
}
```

Special Notes:

この関数で各フィルタモジュールのフィルタ実行 API をコールしています。以下の API 説明も参照してください。

- FIR フィルタ実行 API：r_ctsu_fir_filter
- IIR フィルタ実行 API：r_ctsu_iir_filter
- メディアンフィルタ実行 API：r_ctsu_median_filter

この関数は引数 p_data で指定した計測値データにフィルタを適用して上書きします。

フィルタ適用前の計測結果データを他の用途で使用する場合は API 実行前にデータを退避してください。

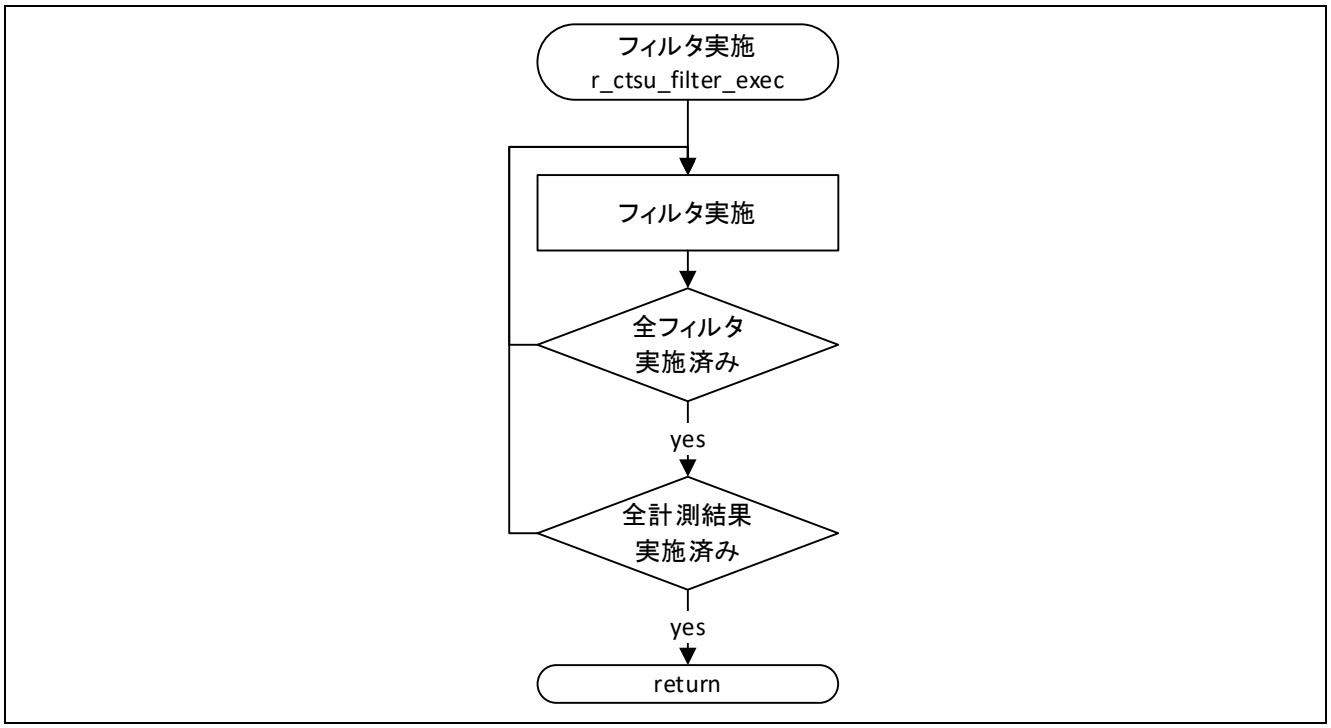


図 2-9 フィルタ実行 API フロー

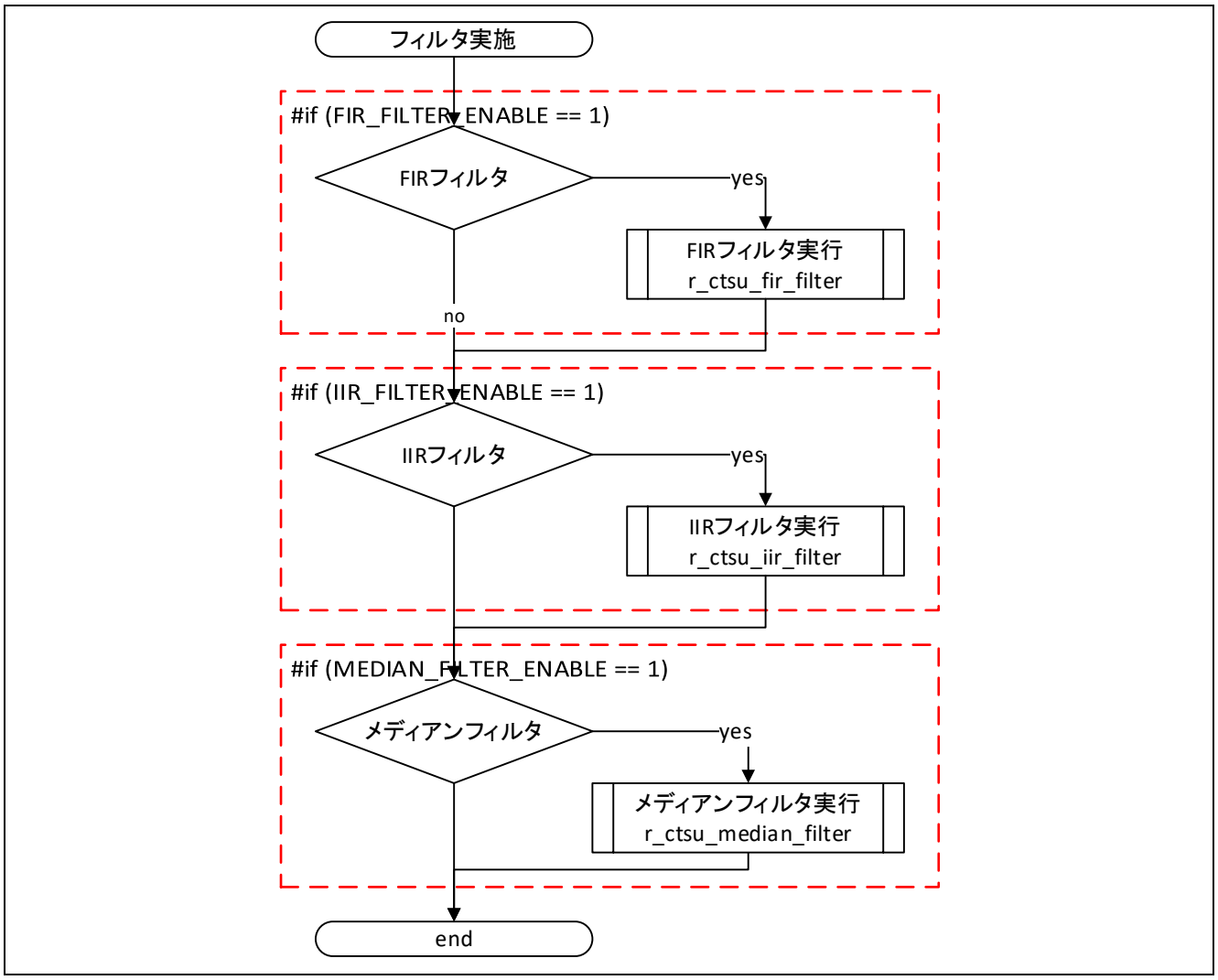


図 2-10 フィルタ実行 API 内フィルタ実施処理フロー

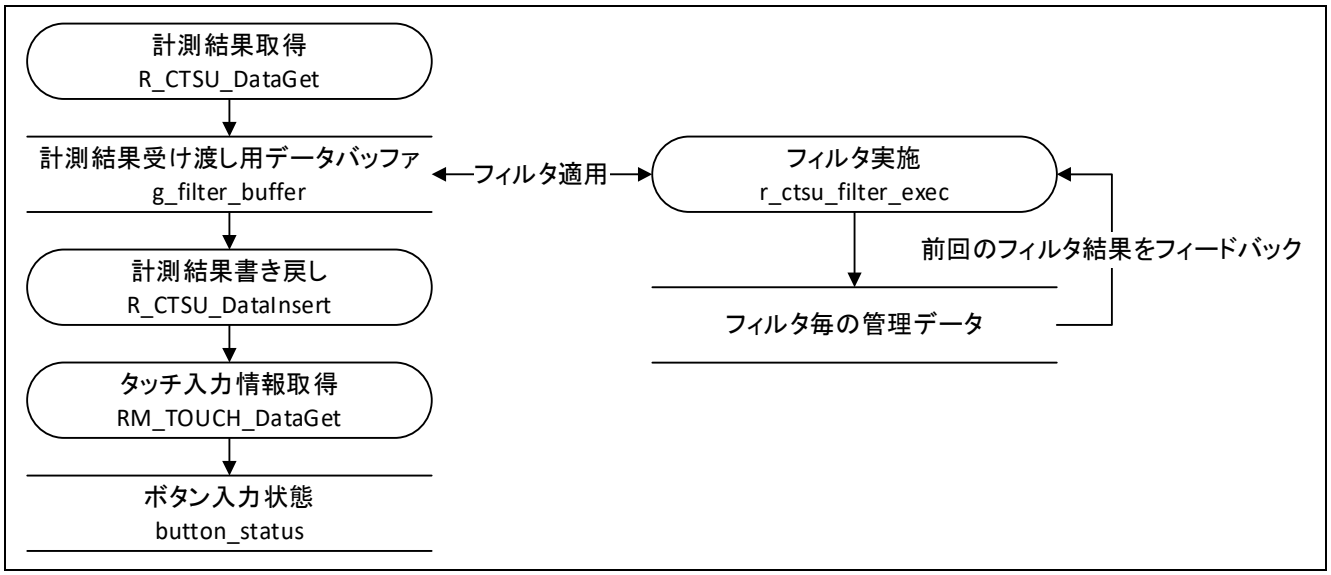


図 2-11 ソフトウェアフィルタデータフロー

2.5 サイズと実行時間

表 2-13~表 2-14 に本サンプルプログラム(タッチインターフェース構成が 3 つ、ボタン×3、スライダ×1、ホイール×1、シールド端子有り)の場合のフィルタ処理のデータサイズおよび実行時間を示します。

表 2-13 フィルタ処理のデータサイズ及び増加分

条件	サイズ [Bytes]		
	text	data	bss
フィルタ追加前	25156	24	3360
フィルタ管理	+140	+24	+104
FIR フィルタ(直接型)(FIR_PRESET_TYPE_4)(注)	+820	+4	+580
FIR フィルタ(置換型)(FIR_PRESET_TYPE_4)(注)	+788	+4	+628
IIR フィルタ(IIR_PRESET_TYPE_4)(注)	+952	+4	+388
メディアンフィルタ(MEDIAN_PRESET_TYPE_2)(注)	+712	+4	+612

【注】 サイズはフィルタのフィルタ次数により変化します。

表 2-14 フィルタ処理の実行時間

条件	実行時間(1ch)
FIR フィルタ(直接型)(FIR_PRESET_TYPE_4)	13.67us
FIR フィルタ(置換型)(FIR_PRESET_TYPE_4)	9.37us
IIR フィルタ(IIR_PRESET_TYPE_4)	17.01us
メディアンフィルタ(MEDIAN_PRESET_TYPE_2)	8.69us（注）

【注】 実行時間は自己容量方式の場合を示しています。相互容量方式の場合は 2 回計測のため実行時間は約 2 倍となります。

【注】 記載の実行時間は平均実行時間となります、メディアンフィルタの実行時間は約 5 倍まで変動する場合があります。

3. FIR フィルタ

FIR(Finite Impulse Response)フィルタはランダムノイズおよび周期ノイズ低減用途として定常的に使用するフィルタです。

詳細については静電容量センサマイコン静電容量タッチノイズイミュニティガイド(R30AN0426)を参照してください。

3.1 仕様

FIR フィルタの演算式を以下に示します。

$$y(n) = \sum_{m=0}^{M} h(m) * x(n-m)$$

nはサンプルのインデックス、$h(\mathrm{m})$は係数、$x(n-\mathrm{m})$は m サンプル遅延の入力データ、$y(n)$は出力データを示します。

表 3-1 に本サンプルプログラムの FIR フィルタの仕様を示します。

表 3-1 FIR フィルタ仕様

項目	仕様	備考
入力データ型	符号付き 32bit 整数型	
出力データ型	符号付き 32bit 整数型	
係数データ型	符号付き 15bit 固定小数点	内部演算は符号付き 32bit 固定小数 (整数部 17bit、小数部 14bit)
最大次数	8	タップ数は「次数+1」で示します。
フィルタ処理方法	• 直接型 • 置換型	条件付きコンパイルにより切り替え可能 (3.5.1 章を参照してください。)
フィルタ安定時間までの出力結果	安定時間中の演算結果と バッファ充填中応答を返す	フィルタ安定時間は タップ数(次数+1)×サンプル数

【注】 係数：FIR フィルタを構成する定数乗算器に与える一連の定数。
次数：係数の要素数。
タップ数：0 次も含めた次数の数。(次数 + 1 の値を示す)

3.2 本サンプルプログラムにおける FIR フィルタの使用方法

本サンプルプログラムでは条件付きコンパイルによりフィルタの処理方法とフィルタ特性を指定することができます。

表 3-2 に FIR フィルタ処理の指定方法を示します。

直接型処理は使用するデータサイズが小さく、置換型処理では処理時間が短くなります。

データサイズと処理時間については表 2-13 及び表 2-14 を参照してください。

表 3-2 サンプル FIR フィルタ処理方法指定

ファイル	定義名	内容
r_ctsu_fir_sample.h	FIR_FILTER_TYPE	フィルタ処理方法 FIR_FILTER_TYPE_DIRECT＝直接型 FIR_FILTER_TYPE_TRANSPOSE＝置換型

3.3 FIR フィルタ API

表 3-3 に本サンプルプログラムで実装されている FIR フィルタ API の一覧を示します。

表 3-3 FIR フィルタ API 一覧

関数名	処理概要
ファイル名：r_ctsu_fir_sample.c	
r_ctsu_fir_open	FIR フィルタ初期化処理
r_ctsu_fir_filter	FIR フィルタ実行処理
r_ctsu_fir_direct_filter	直接型 FIR フィルタ処理
r_ctsu_fir_transpose_filter	置換型 FIR フィルタ処理

3.3.1 r_ctsu_fir_open

この関数は、FIR フィルタ処理用のバッファ割り当てと初期化を行う関数です。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。

Format

fsp_err_t r_ctsu_fir_open(fir_ctrl_t * const p_ctrl , fir_config_t const * const p_cfg);

Parameters

p_ctrl

FIR フィルタ管理用データへのポインタ

p_cfg

FIR フィルタのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_fir_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分の FIR フィルタ処理用のバッファを割り当てて初期化を行います。

Example

```
p_fir_cfg = (fir_config_t *)p_cfg;
p_element_ctrl->p_filter_ctrl = gp_ctsu_fir_ctrl;
for(element_id = 0; element_id < p_element_ctrl->element_num; element_id++)
{
 p_fir_ctrl = (fir_ctrl_t *)p_element_ctrl->p_filter_ctrl;
 ret = r_ctsu_fir_open(&p_fir_ctrl[element_id], p_fir_cfg);
 if(ret != FSP_SUCCESS)
 {
    return ret;
 }
}
```

Special Notes:

本館数の実行前に FIR フィルタ管理データ割り当て時位置ポインタを参照して FIR フィルタ管理用データへのポインタを設定する必要があります。

本関数はタッチインターフェース毎に CTSU ドライバで読み込みを行う計測結果データの個数回実行する必要があります。（自己容量の場合は「端子数」回、相互容量の場合は「送信端子数×受信端子数×2」回）

フィルタ初期化 API(r_ctsu_filter_open)の説明も参照してください。

3.3.2 r_ctsu_fir_filter

この関数は、1 計測結果分の FIR フィルタ動作を実施します。

Format

fsp_err_t r_ctsu_fir_filter (fir_ctrl_t * const p_ctrl , int32_t *p_data) ;

Parameters

p_ctrl

FIR フィルタ管理用データへのポインタ

p_data

FIR フィルタを適用する計測値データへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_fir_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分の FIR フィルタ処理を行います。

Example

```
/* Apply FIR filter */
if(p_instance_ctrl->p_cfg->p_filter_cfg[filter_id].type == FILTER_TYPE_FIR)
{
 p_fir_ctrl = (fir_ctrl_t *)p_instance_ctrl-
>p_element_ctrl[filter_id].p_filter_ctrl;
  fir_err = r_ctsu_fir_filter(&p_fir_ctrl[element_id], &filter_data);
  if( FSP_SUCCESS != fir_err )
  {
     ret = fir_err;
  }
}
```

Special Notes:

本関数は条件付きコンパイル FIR_FILTER_TYPE により実施する処理が異なります。

直接型 FIR フィルタ実行 API(r_ctsu_fir_direct_filter)、置換型 FIR フィルタ実行 API(r_ctsu_fir_transport_filter) も参照してください。

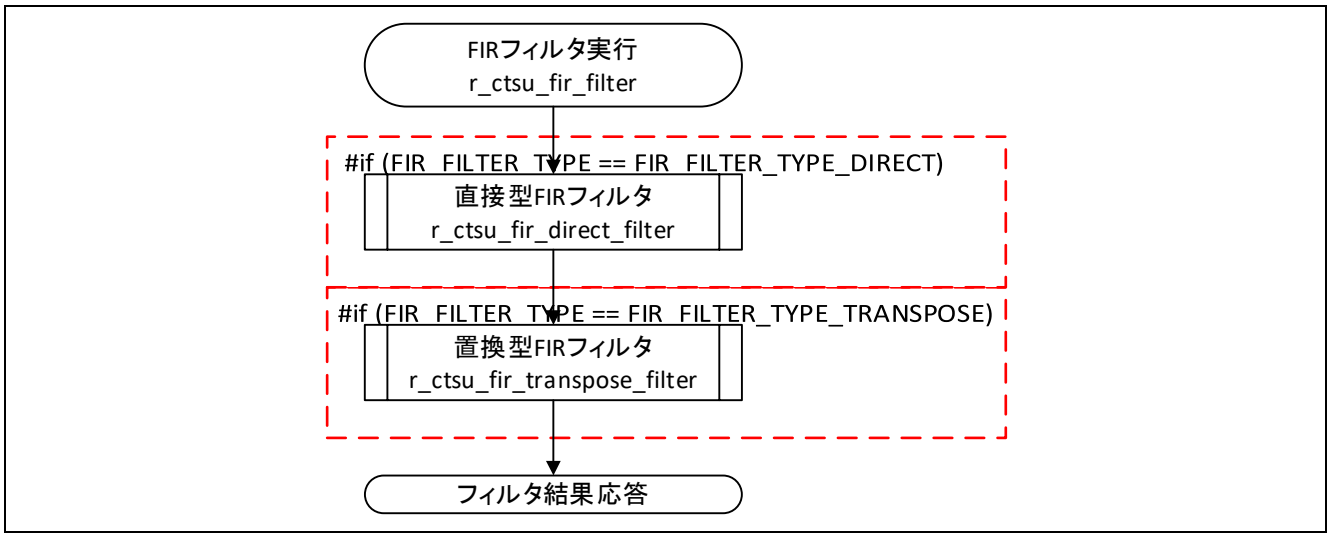


図 3-1　FIR フィルタ実行 API フロー

3.3.3 r_ctsu_fir_direct_filter

この関数は、1 計測結果分の FIR フィルタ動作を直接型処理で実施します。

Format

fsp_err_t r_ctsu_fir_direct_filter(fir_ctrl_t * const p_ctrl , int32_t *p_data);

Parameters

p_ctrl

FIR フィルタ管理用データへのポインタ

p_data

FIR フィルタを適用する計測値データへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_fir_sample.c にプロトタイプ宣言されています。

Description

この関数は、条件付きコンパイル FIR_FILTER_TYPE = FIR_FILTER_TYPE_DIRECT の場合に FIR フィルタ実行処理(r_ctsu_fir_filter)からコールされます。

計測値データとして符号付き 18bit 整数の範囲(131071～-131072)以上のデータが渡された場合は、上限値もしくは下限値が入力されたものとして演算を行います。

演算結果は符号付き 18bit 整数の範囲(131071～-131072)までとなり、それを超える場合は上限値もしくは下限値に丸められます。

Special Notes:

フィルタ安定時間が経過するまでの間はバッファ充填中応答を返します。

フィルタ安定時間が経過するまでのフィルタ適用結果は未充填範囲が初期化状態(0)での演算結果となります。

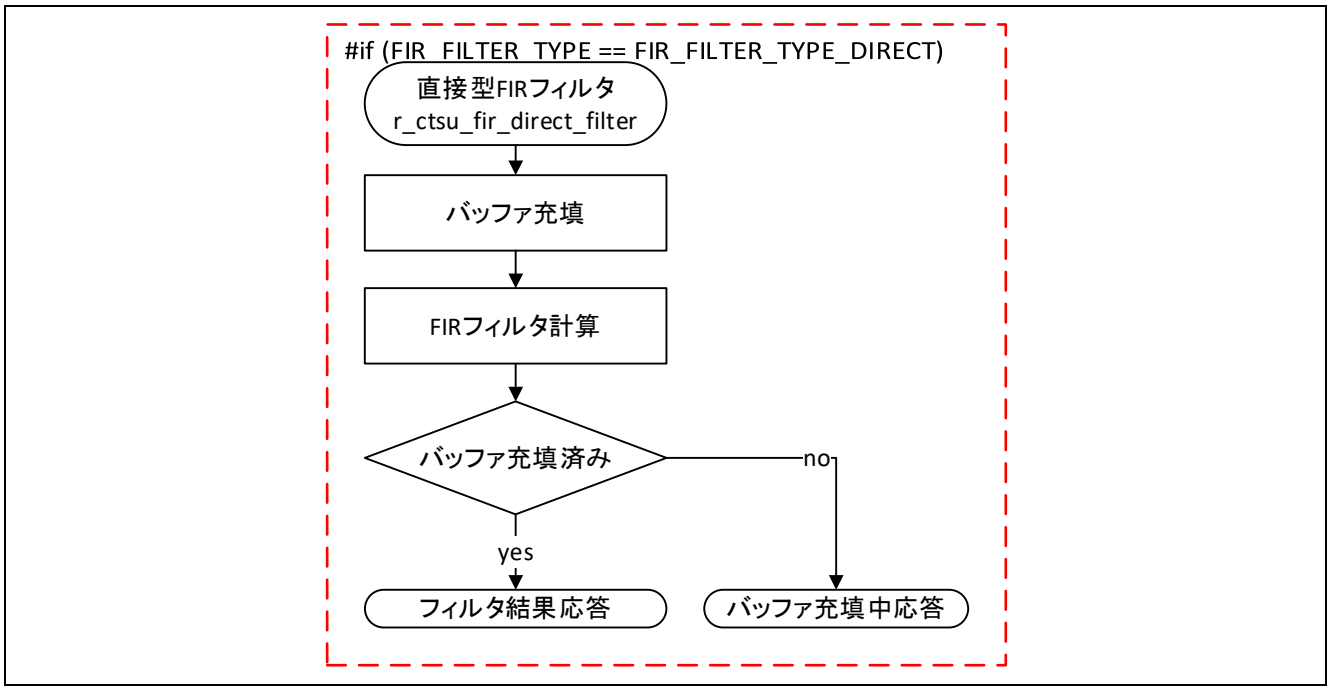


図 3-2 直接型 FIR フィルタ API フロー

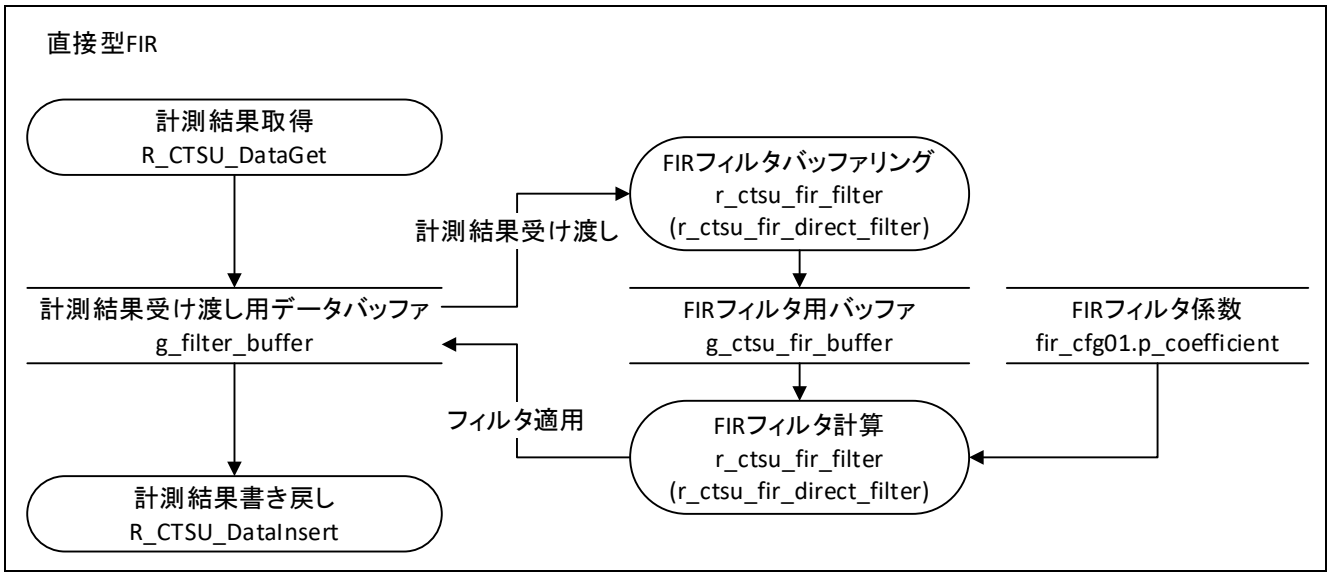


図 3-3 直接型 FIR フィルタデータフロー

3.3.4 r_ctsu_fir_transpose_filter

この関数は、1 計測結果分の FIR フィルタ動作を置換型処理で実施します。

Format

fsp_err_t r_ctsu_fir_transpose_filter (fir_ctrl_t * const p_ctrl , int32_t *p_data);

Parameters

p_ctrl

FIR フィルタ管理用データへのポインタ

p_data

FIR フィルタを適用する計測値データへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_fir_sample.c にプロトタイプ宣言されています。

Description

この関数は、条件付きコンパイル FIR_FILTER_TYPE = FIR_FILTER_TYPE_TRANSPOSE の場合に FIR フィルタ実行処理(r_ctsu_fir_filter)からコールされます。

計測値データとして符号付き 18bit 整数の範囲(131071～-131072)以上のデータが渡された場合は、上限値もしくは下限値が入力されたものとして演算を行います。

演算結果は符号付き 18bit 整数の範囲(131071～-131072)までとなり、それを超える場合は上限値もしくは下限値に丸められます。

Special Notes:

フィルタ安定時間が経過するまでの間はバッファ充填中応答を返します。

フィルタ安定時間が経過するまでのフィルタ適用結果は未充填範囲が初期化状態(0)での演算結果となります。

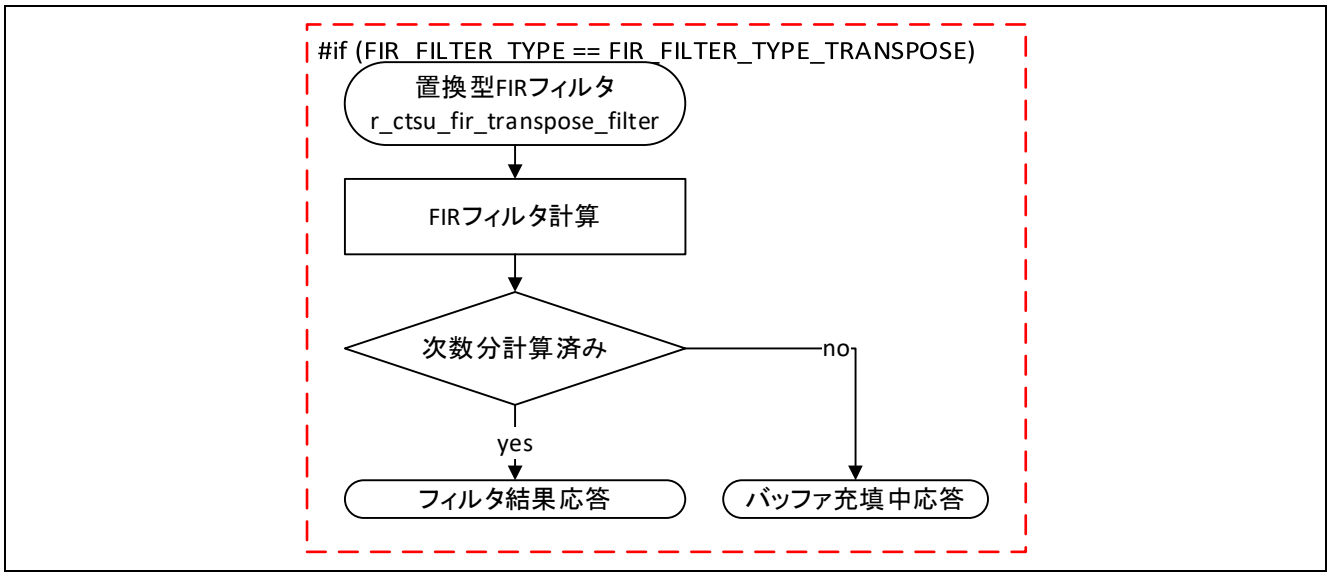


図 3-4 置換型 FIR フィルタ API フロー

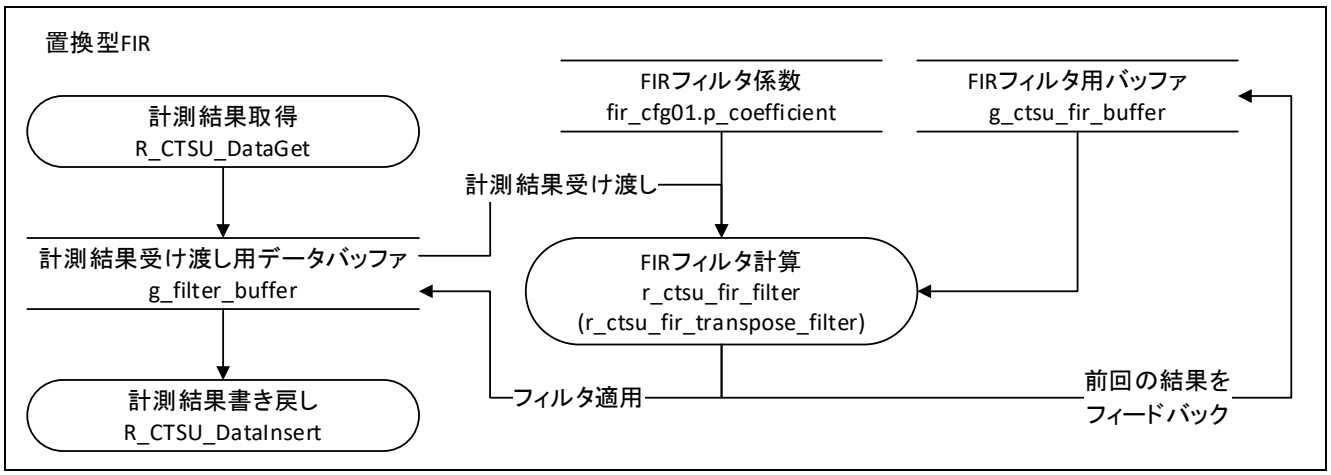


図 3-5 置換型 FIR フィルタデータフロー

3.4 FIR フィルタ用データー覧

FIR フィルタ用に用意している定数・グローバル変数について説明します

3.4.1 定数

表 3-4 に定数の一覧を示します。

表 3-4 FIR フィルタ用定数

定数名	設定値	内容
ファイル名：r_ctsu_fir_sample.h		
FIR_FILTER_NUM	1	FIR フィルタ段数
ファイル名：r_ctsu_fir_sample.c		
FIR_TAP_SIZE_MIN	2	最小タップ数
FIR_TAP_SIZE_MAX	9	最大タップ数
FIR_CFG_DECIMAL_POINT	14	固定小数点桁数
FIR_FILTER_SIZE	FIR_FILTER_NUM × FILTER_ELEMENT_SIZE	FIR フィルタ用バッファサイズ (FIR フィルタ段数と計測結果個数から計算されます)
MAX_FIR_COEFFICIENT_SUM	0x00008000	係数合計の最大値
MIN_FIR_COEFFICIENT_SUM	0xFFFF7FFF	係数合計の最小値
MAX_FIR_COEFFICIENT_PLUS	0x3FFF	係数値の最大値
MIN_FIR_COEFFICIENT_MINUS	0xC000	係数値の最小値
FIR_RESULT_MAX	0x0001FFFF	フィルタ結果最大値
FIR_RESULT_MIN	0xFFFE0000	フィルタ結果最小値

3.4.2 グローバル変数

表 3-5 にグローバル変数の一覧を示します。

表 3-5 FIR フィルタ用グローバル変数

変数名	型	説明
ファイル名：r_ctsu_fir_sample.c		
g_ctsu_fir_element_index	uint16_t	バッファ割り当て管理用インデクス
g_ctsu_fir_ctrl[FIR_FILTER_SIZE]	fir_ctrl_t	FIR フィルタ管理データ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×FIR フィルタ段数 ※相互容量電極数：送信電極数×受信電極数
gp_ctsu_fir_ctrl	fir_ctrl_t *	FIR フィルタ管理データ割り当て時位置ポインタ
g_ctsu_fir_buffer[FIR_FILTER_SIZE][FIR_TAP_SIZE_MAX]	int32_t	FIR フィルタ用バッファ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×最大タップ数(9) ※相互容量電極数：送信電極数×受信電極数 ※置換型の場合はバッファサイズが端子数(自己容量電極数＋相互容量電極数×2)×(最大タップ数(9)+1)になります

3.5　フィルタ調整手順

FIR フィルタの係数定義を変更し、フィルタ特性を調整することができます。

3.5.1　フィルタ処理方法

条件付きコンパイルにより FIR フィルタの処理方法を指定することができます。直接型処理は使用するデータサイズが小さく、置換型処理では処理時間が短くなります。

条件付きコンパイルの設定方法は表 3-2 を参照してください。

データサイズと処理時間については表 2-13 及び表 2-14 を参照してください。

図 3-6 に FIR フィルタのブロック図を示します。

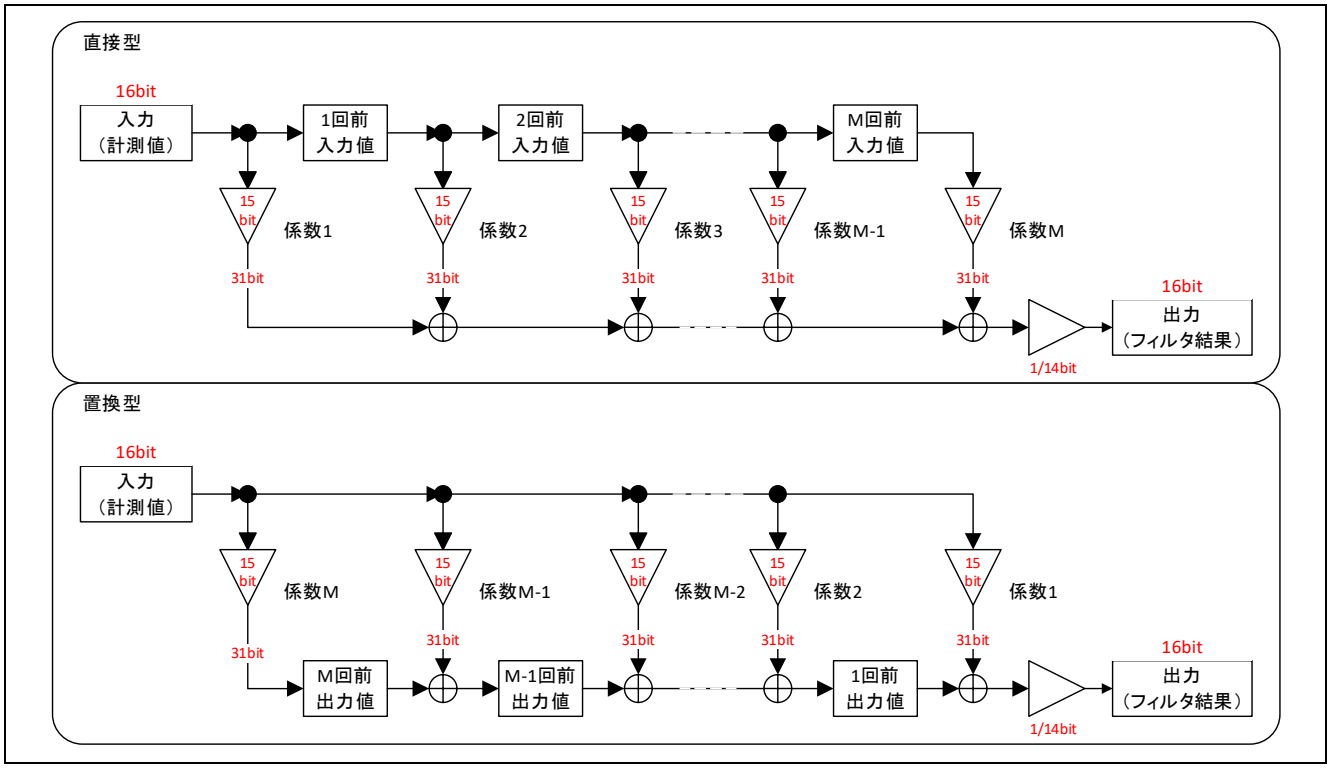


図 3-6 FIR フィルタブロック図

3.5.2　フィルタ特性

本サンプルプログラムでは次数が 8 までのフィルタを扱うことができます。

表 3-6 にサンプル FIR フィルタの特性指定の定義を示します。表 3-7 に示す係数定義およびフィルタ構成定義を指定することでフィルタ特性を変更することができます。

表 3-6 サンプル FIR フィルタ特性指定

ファイル	定義名	内容
r_ctsu_fir_sample.h	FIR_PRESET_TYPE	FIR フィルタで使用するサンプルプリセット指定

表 3-7 サンプル FIR フィルタ係数定義

	FIR_PRESET_TYPE_1	FIR_PRESET_TYPE_2	FIR_PRESET_TYPE_3	FIR_PRESET_TYPE_4
	FIR 移動平均フィルタ		FIR ローパスフィルタ	
次数	2	5	3	8
係数	0.33331298828125	0.1666259765625	0.1636962890625	-0.00604248046875
	0.33331298828125	0.1666259765625	0.3363037109375	-0.01336669921875
	0.33331298828125	0.1666259765625	0.3363037109375	0.05047607421875
		0.1666259765625	0.1636962890625	0.26800537109375
		0.1666259765625		0.40185546875000
		0.1666259765625		0.26800537109375
				0.05047607421875
				-0.01336669921875
				-0.00604248046875

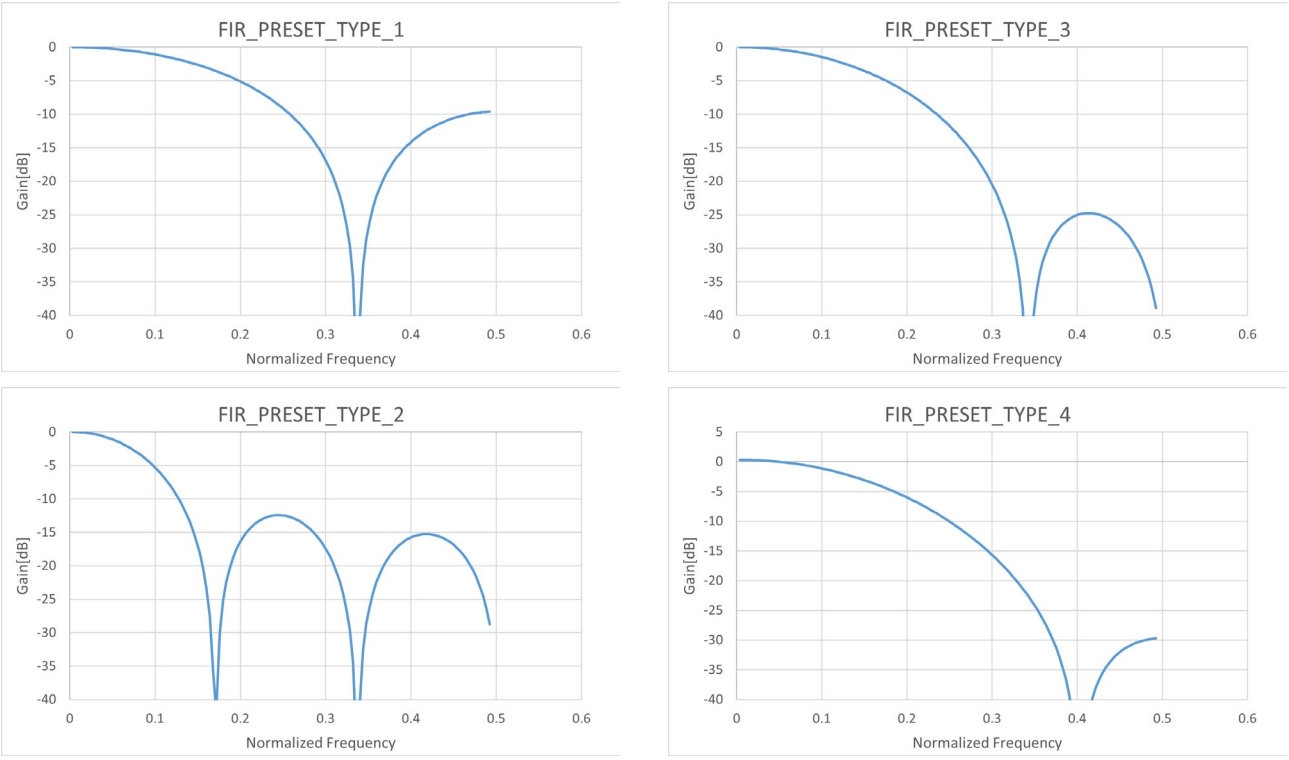


図 3-7 サンプルプリセットのフィルタ特性

3.5.3　係数定義

FIR フィルタ構成の係数は整数部なし、下位 14bit を小数部とした符号付き固定小数の形で定義し、係数値÷16384 として扱われます。

サンプルプログラムの係数は値域が-1.0 から 1.0 の範囲で設計し、小数の係数に 16384(0x4000)を乗じた値を係数定義に設定してください。1LSB 未満の小数値は表現できないため演算誤差が発生します。

表 3-8 に 10 進数、16 進数、小数の対応例を示します。

表 3-8 固定小数点定義例

小数	16 進数	10 進数
-0.00604248046875	-0.00604248046875 ×0x4000 = FF9D	-0.00604248046875 ×16384 = -99
-0.01336669921875	-0.01336669921875 ×0x4000 = FF25	-0.01336669921875 ×16384 = -219
0.05047607421875	0.05047607421875 ×0x4000 = 033B	0.05047607421875 ×16384 = 827
0.26800537109375	0.26800537109375 ×0x4000 = 1127	0.26800537109375 ×16384 = 4391
0.40185546875000	0.40185546875000 ×0x4000 = 19B8	0.40185546875000 ×16384 = 6584
0.26800537109375	0.26800537109375 ×0x4000 = 1127	0.26800537109375 ×16384 = 4391
0.05047607421875	0.05047607421875 ×0x4000 = 033B	0.05047607421875 ×16384 = 827
-0.01336669921875	-0.01336669921875 ×0x4000 = FF25	-0.01336669921875 ×16384 = -219
-0.00604248046875	-0.00604248046875 ×0x4000 = FF9D	-0.00604248046875 ×16384 = -99

3.5.4 FIR フィルタ構成定義

fir_config_t 型のデータテーブルで FIR フィルタのタップ数、係数を定義します。

タップ数には FIR フィルタの次数+1 を指定し、係数テーブルには FIR フィルタの係数値を 0 次から順に 15bit の符号付き固定小数点として記載します。

タップ数は 2~9、係数テーブルは係数定義の合計が-2.0 から 2.0 の範囲内でのみ定義可能です。

【注】 係数テーブルは係数定義の合計が 1.0 に近づくように定義してください。
係数テーブルの合計が 1.0 を超える場合は計測結果が増幅、1.0 未満の場合は計測結果が減衰します。

```
const fir_config_t fir_cfg04 =
{
  .taps = 9,
  .p_coefficient =
  {
    -99,
    -219,
    827,
    4391,
    6584,
    4391,
    827,
    -219,
    -99,
  },
};
```

FIR フィルタのタップ数として
フィルタの次数+1 を指定します

FIR フィルタのタップ数分の係数を定義します

4. IIR フィルタ

IIR(Infinite Impulse Response)フィルタは省メモリ、小演算負荷で高周波成分低減用途として定常的に使用するフィルタです。

詳細については静電容量センサマイコン静電容量タッチノイズイミュニティガイド(R30AN0426)を参照してください。

4.1 仕様

IIR フィルタの演算式を以下に示します。

$$y(n) = \sum_{k=0}^{K} b(k) * x(n-k) - \sum_{m=1}^{M} a(m) * y(n-m)$$

nはサンプルのインデックス、a(m)、a(k)は係数、$x(n-k)$は k サンプル遅延の入力データ、$y(n-m)$は m サンプル遅延の出力データ、$y(n)$は出力データを示します。

表 4-1 に本サンプルプログラムの IIR フィルタの仕様を示します。

表 4-1 IIR フィルタ仕様

項目	仕様	備考
入力データ型	符号付き 32bit 整数型	
出力データ型	符号付き 32bit 整数型	
係数データ型	符号付き 16bit 固定小数点	内部演算は符号付き 32bit (整数部 19bit、小数部 12bit)
最大次数	4	タップ数は「次数+1」で示します
フィルタ処理方法	標準型	IIR フィルタを複数段の接続することで縦続型 IIR フィルタとして使用可能 (4.5.4.1 章を参照してください。)
フィルタ安定時間までの出力結果	安定時間中の演算結果とバッファ充填中応答を返す	フィルタ安定時間は構成定義で指定したサンプル数 （安定時間の指定範囲はタップ数～254）

【注】 係数：IIR フィルタを構成する定数乗算器に与える一連の定数。
次数：係数の要素数。
タップ数：0 次も含めた次数の数。(次数 + 1 の値を示す。)

4.2 本サンプルプログラムにおける IIR フィルタの使用方法

本サンプルプログラムでは条件付きコンパイルによりフィルタ特性を指定することができます。

4.3 IIR フィルタ API

表 4-2 に本サンプルプログラムで実装されている IIR フィルタ API の一覧を示します。

表 4-2 IIR フィルタ API 一覧

関数名	処理概要
ファイル名：r_ctsu_iir_sample.c	
r_ctsu_iir_open	IIR フィルタ初期化処理
r_ctsu_iir_filter	IIR フィルタ実行処理

4.3.1 r_ctsu_iir_open

この関数は、IIR フィルタ処理用のバッファ割り当てと初期化を行う関数です。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。

Format

fsp_err_t r_ctsu_iir_open(iir_ctrl_t * const p_ctrl , iir_config_t const * const p_cfg);

Parameters

p_ctrl

IIR フィルタ管理用データへのポインタ

p_cfg

IIR フィルタのコンフィグレーション定義へのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_iir_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分の IIR フィルタ処理用のバッファを割り当てて初期化を行います。

Example

```
p_iir_cfg = (iir_config_t *)p_cfg;
p_element_ctrl->p_filter_ctrl = gp_ctsu_iir_ctrl;
for(element_id = 0; element_id < p_element_ctrl->element_num; element_id++)
{
 p_iir_ctrl = (iir_ctrl_t *)p_element_ctrl->p_filter_ctrl;
 ret = r_ctsu_iir_open(&p_iir_ctrl[element_id], p_iir_cfg);
 if(ret != FSP_SUCCESS)
 {
    return ret;
 }
}
```

Special Notes:

本関数の実行前に IIR フィルタ管理データ割り当て時位置ポインタを参照して IIR フィルタ管理用データへのポインタを設定する必要があります。

本関数はタッチインターフェース毎に CTSU ドライバで読み込みを行う計測結果データの個数回実行する必要があります。（自己容量の場合は「端子数」回、相互容量の場合は「送信端子数×受信端子数×2」回）

フィルタ初期化 API(r_ctsu_filter_open)の説明も参照してください。

4.3.2 r_ctsu_iir_filter

この関数は、1 計測結果分の IIR フィルタ動作を実施します。

Format

fsp_err_t r_ctsu_iir_filter(iir_ctrl_t * const p_ctrl , int32_t *p_data);

Parameters

p_ctrl

IIR フィルタ管理用データへのポインタ

p_data

IIR フィルタを適用する計測値データへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_iir_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分の IIR フィルタ処理を行います。

計測値データとして符号付き 19bit 整数の範囲(262143～-262144)以上のデータが渡された場合は、上限値もしくは下限値が入力されたものとして演算を行います。

演算結果は符号付き 18bit 整数の範囲(131071～-131072)までとなり、それを超える場合は上限値もしくは下限値に丸められます。

Example

```
/* Apply IIR filter */
if(p_instance_ctrl->p_cfg->p_filter_cfg[filter_id].type == FILTER_TYPE_IIR)
{
 p_iir_ctrl = (iir_ctrl_t *)p_instance_ctrl-
>p_element_ctrl[filter_id].p_filter_ctrl;
  filter_err = r_ctsu_iir_filter(&p_iir_ctrl[element_id], &filter_data);
  if( FSP_SUCCESS != filter_err )
  {
     ret = filter_err;
  }
}
```

Special Notes:

フィルタ安定時間が経過するまでの間はバッファ充填中応答を返します。

フィルタ安定時間が経過するまでのフィルタ適用結果は未充填範囲が初期化状態(0)での演算結果となります。

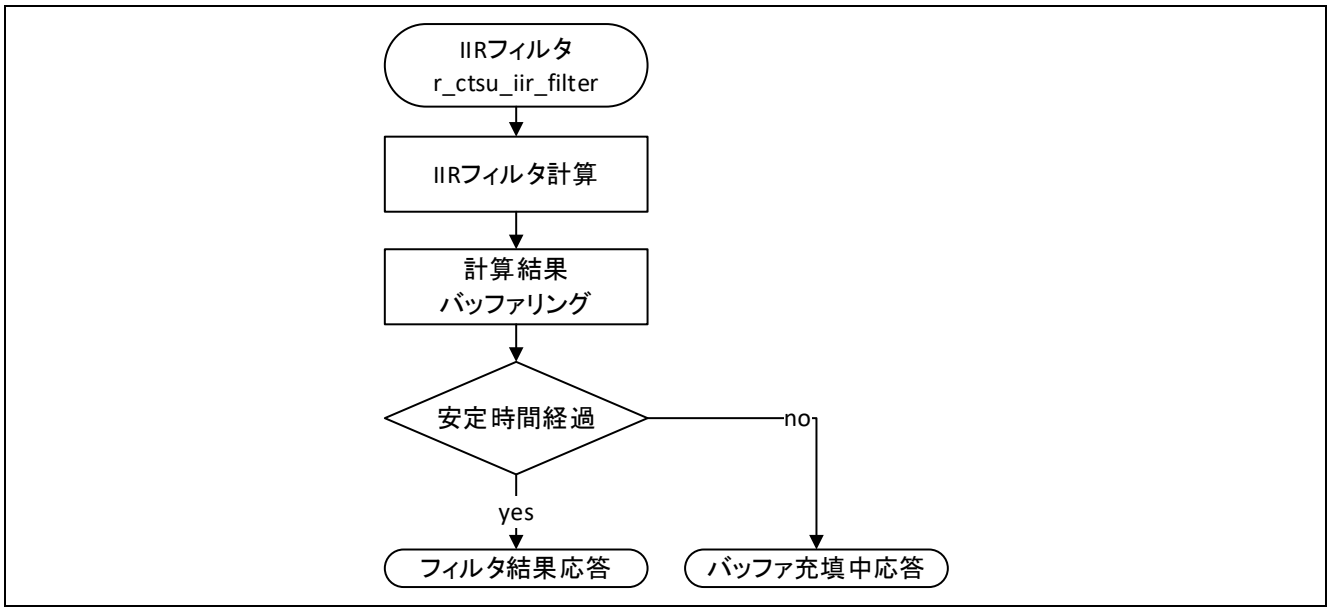


図 4-1 IIR フィルタ実行 API フロー

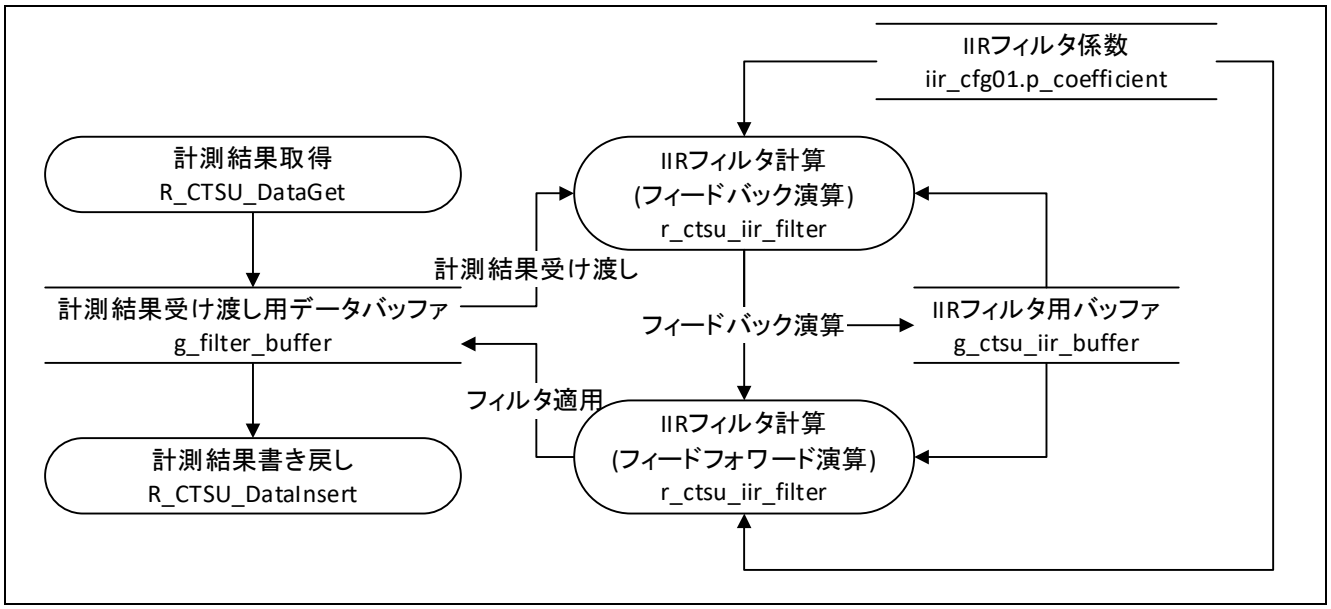


図 4-2 IIR フィルタデータフロー

4.4 IIR フィルタ用データー覧

IIR フィルタ用に用意している定数・グローバル変数について説明します

4.4.1 定数

表 4-3 に定数の一覧を示します。

表 4-3 IIR フィルタ用定数

定数名	設定値	内容
ファイル名：filter_config_sample.h		
CTSU_FILTER_NUM	1～8	縦続型 IIR フィルタを構成する場合に 2 以上を指定します
ファイル名：r_ctsu_iir_sample.h		
IIR_FILTER_NUM	1～8	IIR フィルタ段数 縦続型 IIR フィルタを構成する場合に 2 以上を指定します
ファイル名：r_ctsu_iir_sample.c		
IIR_TAP_SIZE_MIN	2	最小タップ数(次数 1+1)
IIR_TAP_SIZE_MAX	5	最大タップ数(次数 4+1)
IIR_SETTLINGS_MAX	255	最大安定待ち時間
IIR_CFG_DECIMAL_POINT	14	固定小数点桁数
IIR_CFG_POINT_OFFSET	2	演算用固定小数点桁数補正値
IIR_FILTER_SIZE	IIR_FILTER_NUM × FILTER_ELEMENT_SIZE	IIR フィルタ用バッファサイズ (IIR フィルタ段数と計測結果個数から計算されます)
IIR_DECIMAL_MAX	0x0003FFFF	フィルタデータ整数部上限値
IIR_DECIMAL_MIN	0xFFFC0000	フィルタデータ整数部下限値
IIR_CALC_MAX	0x7FFFFFFF	演算用上限値
IIR_CALC_MIN	0x80000000	演算用下限値
IIR_RESULT_MAX	0x0001FFFF	フィルタ結果最大値
IIR_RESULT_MIN	0xFFFE0000	フィルタ結果最小値

4.4.2 グローバル変数

表 4-4 にグローバル変数の一覧を示します。

表 4-4 IIR フィルタ用グローバル変数

変数名	型	説明
ファイル名：r_ctsu_fir_sample.c		
g_ctsu_iir_element_index	uint16_t	バッファ割り当て管理用インデクス
g_ctsu_iir_ctrl[IIR_FILTER_SIZE]	iir_ctrl_t	IIR フィルタ管理データ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×IIR フィルタ段数 ※相互容量電極数：送信電極数×受信電極数
gp_ctsu_iir_ctrl	iir_ctrl_t *	IIR フィルタ管理データ割り当て時位置ポインタ
g_ctsu_iir_buffer[IIR_FILTER_SIZE][IIR_TAP_SIZE_MAX]	int32_t	IIR フィルタ用バッファ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×最大タップ数(5) ※相互容量電極数：送信電極数×受信電極数

4.5　フィルタ調整手順

IIR フィルタの係数定義を変更、もしくは IIR フィルタを複数段接続してフィルタ特性を調整することができます。

4.5.1　フィルタ処理方法

データサイズと処理時間については表 2-13 及び表 2-14 を参照してください。

図 4-3 に IIR フィルタのブロック図を示します。

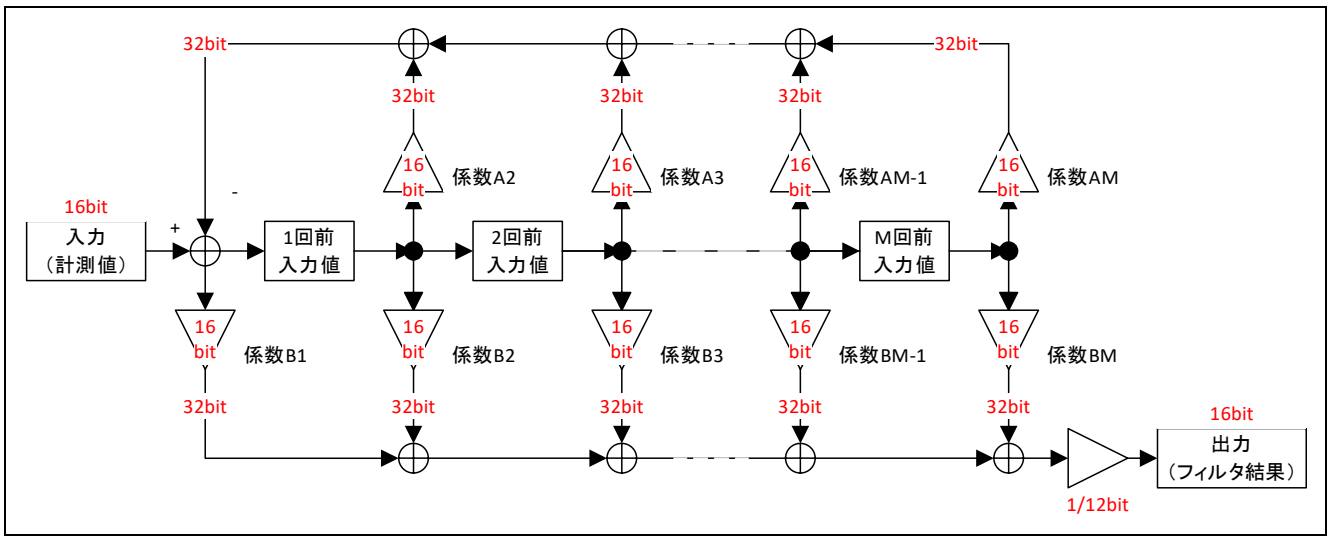


図 4-3 IIR フィルタブロック図

4.5.2　フィルタ特性

本サンプルプログラムでは次数が 4 までのフィルタを扱うことができます。

フィルタを複数個定義する事で 5 次以上のフィルタを構成することもできます。

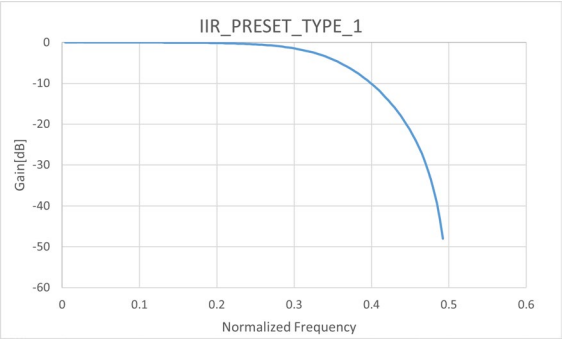
表 4-5 にサンプル IIR フィルタの特性指定の定義を示します。表 4-6、表 4-7 に示す係数定義およびフィルタ構成定義を指定することでフィルタ特性を変更することができます。

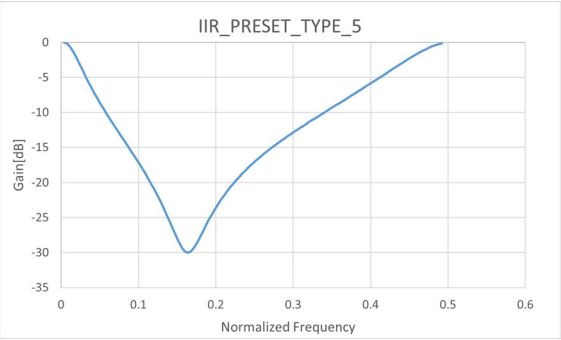
表 4-5 サンプル IIR フィルタ特性指定

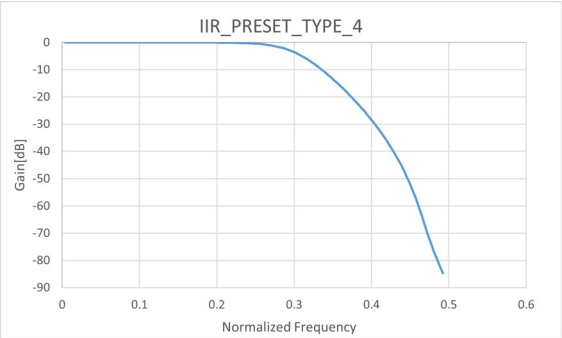
ファイル	定義名	内容
r_ctsu_iir_sample.h	IIR_PRESET_TYPE	IIR フィルタで使用するサンプルプリセット指定

表 4-6 サンプル IIR フィルタ係数定義(1/2)

	IIR_PRESET_TYPE_1 IIR ローパスフィルタ	IIR_PRESET_TYPE_4 IIR ローパスフィルタ	IIR_PRESET_TYPE_5 IIR ピーキングフィルタ	IIR_PRESET_TYPE_6 IIR 移動平均フィルタ
次数	2	4	2	1
係数 A	0	0	0	0
	0.595458984375	0.6468505859375	-0.2279052734375	-0.75
	0.23492431640625	0.6185302734375	-0.57470703125	
		0.14617919921875		
		0.0260009765625		
係数 B	0.45758056640625	0.15234375	0.237548828125	0.25
	0.9151611328125	0.609375	-0.2279052734375	0
	0.45758056640625	0.91412353515625	0.187744140625	
		0.609375		
		0.15234375		







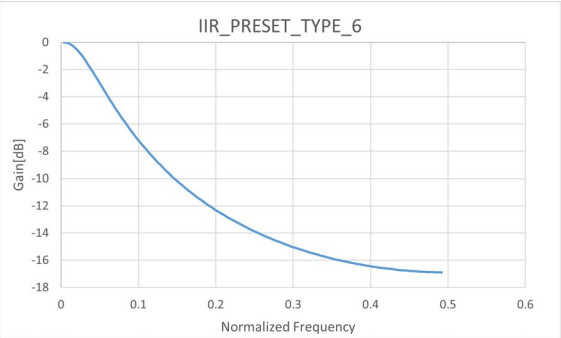
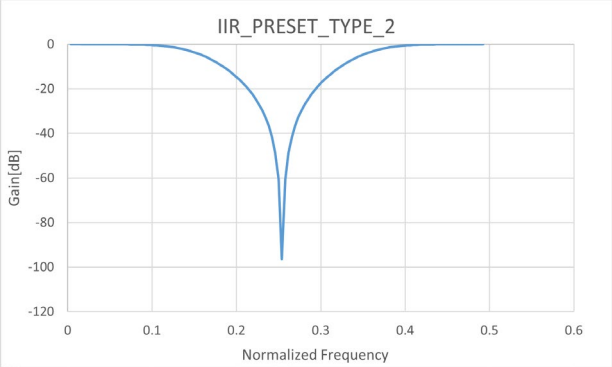
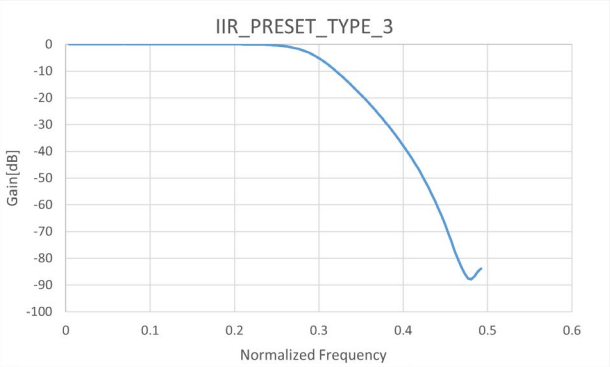


表 4-7 サンプル IIR フィルタ係数定義(2/2)

	IIR_PRESET_TYPE_2 縦続型双二次 IIR 帯域禁止フィルタ		IIR_PRESET_TYPE_3 縦続型双二次 IIR ローパスフィルタ		
次数	4		5		
係数A	0	0	0	0	0
	-0.78240966796875	0.78240966796875	0.104736328125	0.23138427734375	0.318359375
	0.4263916015625	0.4263916015625	0	0.11639404296875	0.53570556640625
係数B	0.3555908203125	1	0.5523681640625	0.3369140625	0.4635009765625
	0	0	0.5523681640625	0.67388916015625	0.927001953125
	0.3555908203125	1	0	0.3369140625	0.4635009765625





4.5.3　係数定義

IIR フィルタ構成の係数は整数部 1bit、下位 14bit を小数部とした符号付き固定小数の形で定義し、係数値÷16384 として扱われます。

サンプルプログラムの係数は値域が-2.0 から 2.0 の範囲で設計し、小数の係数に 16384(0x4000)を乗じた値を係数定義に設定してください。1LSB 未満の小数値は表現できないため演算誤差が発生します。

表 4-8 に 10 進数、16 進数、小数の対応例を示します。

表 4-8　固定小数点定義例

小数	16 進数	10 進数
0.6468505859375	0.6468505859375 × 0x4000 = 0x2966	0.6468505859375 × 16384 = 10598
0.6185302734375	0.6185302734375 × 0x4000 = 0x2796	0.6185302734375 × 16384 = 10134
0.14617919921875	0.14617919921875 × 0x4000 = 0x095B	0.14617919921875 × 16384 = 2395
0.0260009765625	0.0260009765625 × 0x4000 = 0x01AA	0.0260009765625 × 16384 = 426
0.15234375	0.15234375 × 0x4000 = 0x09C0	0.15234375 × 16384 = 2496
0.609375	0.609375 × 0x4000 = 0x2700	0.609375 × 16384 = 9984
0.91412353515625	0.91412353515625 × 0x4000 = 0x3A81	0.91412353515625 × 16384 = 14977
0.609375	0.609375 × 0x4000 = 0x2700	0.609375 × 16384 = 9984
0.15234375	0.15234375 × 0x4000 = 0x09C0	0.15234375 × 16384 = 2496

4.5.4　IIR フィルタ構成定義

iir_config_t 型のデータテーブルで IIR フィルタのタップ数、係数を定義します。

タップ数には IIR フィルタの次数+1 を指定し、係数テーブルには IIR フィルタの係数値を係数 B、係数 A の順に係数 A と B をセットにして 0 次から順に 16bit の符号付き固定小数点として記載します。

係数 A0 の定義は 0 固定となります、0 以外を定義しないでください。

タップ数は 2~5 の範囲内でのみ定義可能です。

安定時間設定はタップ数を指定した状態でフィルタ動作を確認し、フィルタの安定に時間がかかる場合は設定値を増やしてください。

安定時間設定はタップ数から 255 までの範囲で設定可能です。

【注】 係数 A は IIR フィルタのフィードバック演算に使用するため定義値によっては演算結果が発散し、正常な演算結果が出力されなくなります。
係数 A の定義は IIR フィルタが安定となるよう十分な注意が必要となります。

【注】 係数テーブルは「係数 B の合計÷係数 A の合計」が 1.0 に近づくように定義してください。
「係数 B 定義の合計÷(係数 A 定義の合計+1)」が 1.0 を超える場合は計測結果が増幅、1.0 未満の場合は計測結果が減衰します。

```
const iir_config_t iir_cfg07 =
{
  .taps = 5,
  .settlings = 5,
  .p_coefficient =
  {
   /* coefficient b,a */
   2496, 0,        /* b0 : 0.15234375, a0 : fixed 0     */
   9984, 10598,    /* b1 : 0.609375   , a1 : 0.646850586*/
   14977,10134,    /* b2 : 0.914123535  , a2 : 0.618530273*/
   9984, 2395,     /* b3 : 0.609375   , a3 : 0.146179199*/
   2496, 426,      /* b4 : 0.15234375, a4 : 0.026000977*/
  },
};
```

IIR フィルタのタップ数として
フィルタの次数+1 を指定します

安定時間設定は通常はタップ数と同じにし、安定
までに時間がかかる場合は値を増やしてください

係数 A0 は必ず 0 にします

IIR フィルタのタップ数分の係数を係数 B、係数 A の
順に係数 AB をセットで定義します

4.5.4.1 縦続型 IIR フィルタ構成

IIR フィルタを複数定義することで縦続型の IIR フィルタを構成することができます。

フィルタ次数が 2 の IIR フィルタを複数定義した場合は一般的な縦続型双二次 IIR フィルタとなります。

filter_config_sample.h

```
#define CTSU_FILTER_NUM (2)
```

フィルタ管理と IIR フィルタのフィルタ段数を指定します

r_ctsu_iir_sample.h

```
#define IIR_FILTER_NUM  (2)
```

filter_config_sample.c

```
const filter_element_config_t g_ctsu_filter_element_config[] =
{
  {
    .type  = FILTER_TYPE_IIR,
    .filter_element_cfg = &iir_cfg02,
  },
  {
    .type  = FILTER_TYPE_IIR,
    .filter_element_cfg = &iir_cfg03,
  },
};
```

縦続型にする IIR フィルタの構成定義を続けて指定します

上側に記載したフィルタが前、下側に記載したフィルタが後に適用されます

iir_config_sample2.c

```
const iir_config_t iir_cfg02 =
{
  .taps = 3,
  .settlings = 3,
  .p_coefficient =
  {
    /* coefficient b,a */
    5826, 0,       /* b0 : 0.3555908203125 , a0 : fixed 0          */
    0,    -12819,  /* b1 : 0          , a1 : -0.78240966796875*/
    5826, 6986,    /* b2 : 0.3555908203125 , a2 : 0.4263916015625     */
  },
};

const iir_config_t iir_cfg03 =
{
  .taps = 3,
  .settlings = 3,
  .p_coefficient =
  {
    /* coefficient b,a */
    16384,0,       /* b0 : 1, a0 : fixed 0       */
    0,    12819,   /* b1 : 0, a1 : 0.78240966796875 */
    16384,6986,    /* b2 : 1, a2 : 0.4263916015625  */
  },
};
```

縦続型にする個々の IIR フィルタを定義します。

5. メディアンフィルタ

メディアンフィルタはパルス性ノイズの除去に使用できるフィルタです。ランダムノイズや低周期ノイズに対しては効果が限定的となるため、そのような場合は FIR や IIR フィルタと組み合わせて使用することもできます。

5.1 動作説明

メディアンフィルタは入力値と過去の数サンプルを参照期間とし、演算により中央値を求めフィルタの出力値とします。過去サンプルを参照するため、過去サンプルのバッファが未充填であるフィルタ初期化直後は初期化状態(0)のバッファと入力値の中央値を出力し、バッファに過去データが充填されると中央値を出力します。フィルタ動作状態はバッファ充填中応答により判断できます。

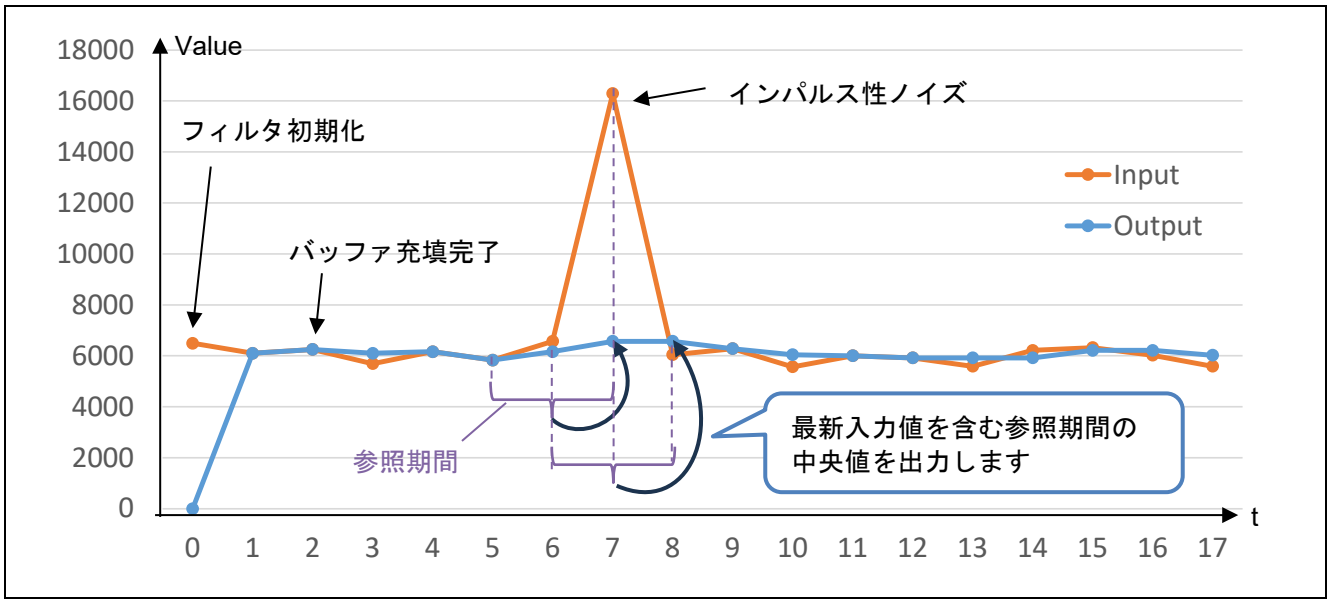


図 5-1　メディアンフィルタの動作例（参照期間 = 3）

表 5-1　メディアンフィルタの動作例（参照期間 = 3）

t	入力値	参照期間 (t)	参照期間データ (太字は中央値)	出力値	バッファ 充填中応答（注）
0	6490	0	0, **0**, 6490	0	FSP_ERR_BUFFER_EMPTY
1	6100	0, 1	0, 6490, **6100**	6100	
2	6250	0, 1, 2	6490, 6100, **6250**	6250	FSP_SUCCESS
3	5690	1, 2, 3	**6100**, 6250, 5690	6100	
4	6160	2, 3, 4	6250, 5690, **6160**	6160	
5	5830	3, 4, 5	5690, 6160, **5830**	5830	
6	6570	4, 5, 6	**6160**, 5830, 6570	6160	
7	16294	5, 6, 7	5830, **6570**, 16294	6570	
8	6040	6, 7, 8	**6570**, 16294, 6040	6570	
9	6280	7, 8, 9	16294, 6040, **6280**	6280	
10	5570	8, 9, 10	**6040**, 6280, 5570	6040	
11	6000	9, 10, 11	6280, 5570, **6000**	6000	
12	5920	10, 11, 12	5570, 6000, **5920**	5920	
13	5590	11, 12, 13	6000, **5920**, 5590	5920	
14	6210	12, 13, 14	**5920**, 5590, 6210	5920	
15	6320	13, 14, 15	5590, **6210**, 6320	6210	
16	6020	14, 15, 16	**6210**, 6320, 6020	6210	
17	5590	15, 16, 17	6320, **6020**, 5590	6020	

注：メディアンフィルタ処理 API の応答。詳細は「5.4.2 r_ctsu_median_filter」を参照してください。

5.2 仕様

表 5-2 に本サンプルプログラムのメディアンフィルタの仕様を示します。

表 5-2 メディアンフィルタ仕様

項目	仕様	備考
入力データ型	符号付き 32bit 整数型	CTSU ドライバの計測データは符号なし 16bit 整数型のためデータ型の変換が必要となります 本サンプルソフトではソフトウェアフィルタ API でデータ型変換を行っています
出力データ型	符号付き 32bit 整数型	
サンプル参照範囲	3,5,7,9	入力値と過去サンプルの合計個数
処理方法	挿入ソートによる中央値検出	
フィルタ初期化後の出力結果	フィルタ安定時間内の演算結果およびバッファ充填中応答を返す	フィルタ安定時間は構成定義で任意サンプル参照期間を指定可能

5.3 データー覧

本節ではメディアンフィルタ用に用意している定数・グローバル変数について説明します。メディアンフィルタを使用するにはソフトウェアフィルタサンプルコード共通の定数およびグローバル変数等のデータ定義が必要です。ソフトウェアフィルタサンプルコード共通のデータ定義の詳細は「2.3 フィルタ構成定義用データー覧」を参照してください。

5.3.1 定数

表 5-3 にメディアンフィルタ用定数の一覧を示します。

表 5-3 メディアンフィルタ用定数

定数名	設定値	内容
ファイル名：r_ctsu_median_sample.h		
MEDIAN_FILTER_NUM	1	メディアンフィルタ構成数 メディアンフィルタ構成を複数定義してタッチ構成に応じて使い分ける場合に設定値を変更してください。
ファイル名：r_ctsu_median_sample.c		
MEDIAN_SAMPLE_SIZE_MAX	9	最大サンプル参照期間 タッチ計測のサンプリング周期が本サンプルプログラムより短い周期のシステムで使用し、11 以上のサンプル参照期間が必要な場合はこの定義を変更してください。

MEDIAN_FILTER_SIZE	MEDIAN_FILTER_NUM × FILTER_ELEMENT_SIZE	メディアンフィルタ用バッファサイズ (メディアンフィルタ段数と計測結果個数から計算されます) “FILTER_ELEMENT_SIZE”は「表 2-2 フィルタ構成定義用定数」を参照してください

5.3.2 グローバル変数

にグローバル変数の一覧を示します。

これらの変数はメディアンフィルタ初期化処理を使用して初期化、更新を行います。

表 5-4 メディアンフィルタ用グローバル変数

変数名	型	説明
ファイル名：r_ctsu_median_sample.c		
g_ctsu_median_element_index	uint16_t	バッファ割り当て管理用インデクス
g_ctsu_median_ctrl[MEDIAN_FILTER_SIZE]	median_ctrl_t	メディアンフィルタ管理データ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×メディアンフィルタ段数 ※相互容量電極数：送信電極数×受信電極数
gp_ctsu_median_ctrl	median_ctrl_t *	メディアンフィルタ管理データ割り当て時位置ポインタ
g_ctsu_median_buffer[MEDIAN_FILTER_SIZE][MEDIAN_SAMPLE_SIZE_MAX]	int32_t	メディアンフィルタ用サンプリングバッファ バッファサイズは端子数(自己容量電極数＋相互容量電極数×2)×最大サンプル参照期間(9) ※相互容量電極数：送信電極数×受信電極数
g_ctsu_median_work[MEDIAN_SAMPLE_SIZE_MAX]	int32_t	ソート処理用一時バッファ バッファサイズは最大サンプル参照期間(9)

5.3.3 構造体

メディアンフィルタ API アクセス用構造体とメディアンフィルタ構成定義用の構造体を示します。

表 5-5 フィルタ構成定義

定義内容	データ型	備考
メディアンフィルタ構成定義	median_config_t	
メディアンフィルタ管理データ	median_ctrl_t	

5.3.3.1 メディアンフィルタ構成定義(median_config_t)

表 5-6 メディアンフィルタ構成定義構造体(median_config_t)

メンバ	データ型	内容
samples	uint16_t	サンプル参照期間 最大サンプル参照期間(9)サンプル以内の奇数のみ指定可能です

- メディアンフィルタ構成定義(median_config_t)の記述例

```
const median_config_t median_cfg02 =
{
  .samples = 5,
};
```

5.3.3.2 メディアンフィルタ管理データ(median_ctrl_t)

表 5-7 メディアンフィルタ管理データ構造体(median_ctrl_t)

メンバ	データ型	内容
index	uint16_t	メディアンフィルタ用サンプリングバッファ入力データ格納位置
count	uint16_t	バッファ充填数カウンタ
p_buffer	int32_t *	メディアンフィルタ用サンプリングバッファポインタ
p_cfg	median_config_t *	メディアンフィルタ構成定義ポインタ

5.4 メディアンフィルタ API

本節ではメディアンフィルタ用に用意している API について説明します。メディアンフィルタを使用するにはソフトウェアフィルタサンプルコード共通の API が必要です。ソフトウェアフィルタサンプルコード共通の API の詳細は「2.4 ソフトウェアフィルタ API」を参照してください。

表 5-8 に本サンプルプログラムで実装されているメディアンフィルタ API の一覧を示します。

表 5-8 メディアンフィルタ API 一覧

関数名	処理概要
ファイル名：r_ctsu_median_sample.c	
r_ctsu_median_open	メディアンフィルタ初期化処理
r_ctsu_median_filter	メディアンフィルタ実行処理
ctsu_insert_sort	挿入ソート処理

5.4.1 r_ctsu_median_open

この関数は、メディアンフィルタ処理用のバッファ割り当てと初期化を行う関数です。この関数は他のメディアンフィルタ API 関数を使用する前に実行する必要があります。

本関数の実行前にメディアンフィルタ管理データ割り当て時位置ポインタを参照してメディアンフィルタ管理用データへのポインタを設定する必要があります。

本関数はタッチインターフェース毎に CTSU ドライバで読み込みを行う計測結果データの個数回実行する必要があります。（自己容量の場合は「端子数」回、相互容量の場合は「送信端子数×受信端子数×2」回）

フィルタ初期化 API(r_ctsu_filter_open)の説明も参照してください。

Format

```
fsp_err_t r_ctsu_median_open(median_ctrl_t * const p_ctrl , median_config_t const * const p_cfg);
```

Parameters

p_ctrl

メディアンフィルタ管理用データへのポインタ

メディアンフィルタ管理データ割り当て時位置ポインタ(gp_ctsu_median_ctrl)を指定します。

メディアンフィルタ管理データ割り当て時位置ポインタ(gp_ctsu_median_ctrl)は本 API 実行毎に更新されるため、タッチ構成毎に先頭のポインタ位置を保持しておく必要があります。

p_cfg

メディアンフィルタの構成定義へのポインタ

本サンプルソフトではメディアンフィルタサンプルプリセット用ソースのメディアンフィルタ構成定義を指定します。

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
FSP_ERR_INVALID_ARGUMENT	/* コンフィグレーションのパラメータが不正です */

Properties

r_ctsu_median_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分のメディアンフィルタ処理用のバッファを割り当てて初期化を行います。

5.4.2 r_ctsu_median_filter

この関数は、1 計測結果分のメディアンフィルタ動作を実施します。

フィルタ安定時間が経過するまでの間はバッファ充填中応答を返します。

フィルタ安定時間が経過するまでのフィルタ適用結果は未充填範囲が初期化状態(0)での演算結果となります。

Format

```
fsp_err_t r_ctsu_median_filter(median_ctrl_t * const p_ctrl , int32_t *p_data);
```

Parameters

p_ctrl

メディアンフィルタ管理用データへのポインタ

p_data

メディアンフィルタを適用する計測値データへのポインタ

ReturnValues

FSP_SUCCESS	/* 正常終了 */
FSP_ERR_ASSERTION	/* 引数のポインタが指定されていません */
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 */
FSP_ERR_BUFFER_EMPTY	/* バッファ充填中のため未適用のフィルタがあります */

Properties

r_ctsu_median_sample.h にプロトタイプ宣言されています。

Description

この関数は、1 計測結果分のメディアンフィルタ処理を行います。

計測値データとして符号付き 31bit 整数の範囲(1073741823～- 1073741824)以上のデータが渡された場合は、上限値もしくは下限値が入力されたものとして演算を行います。

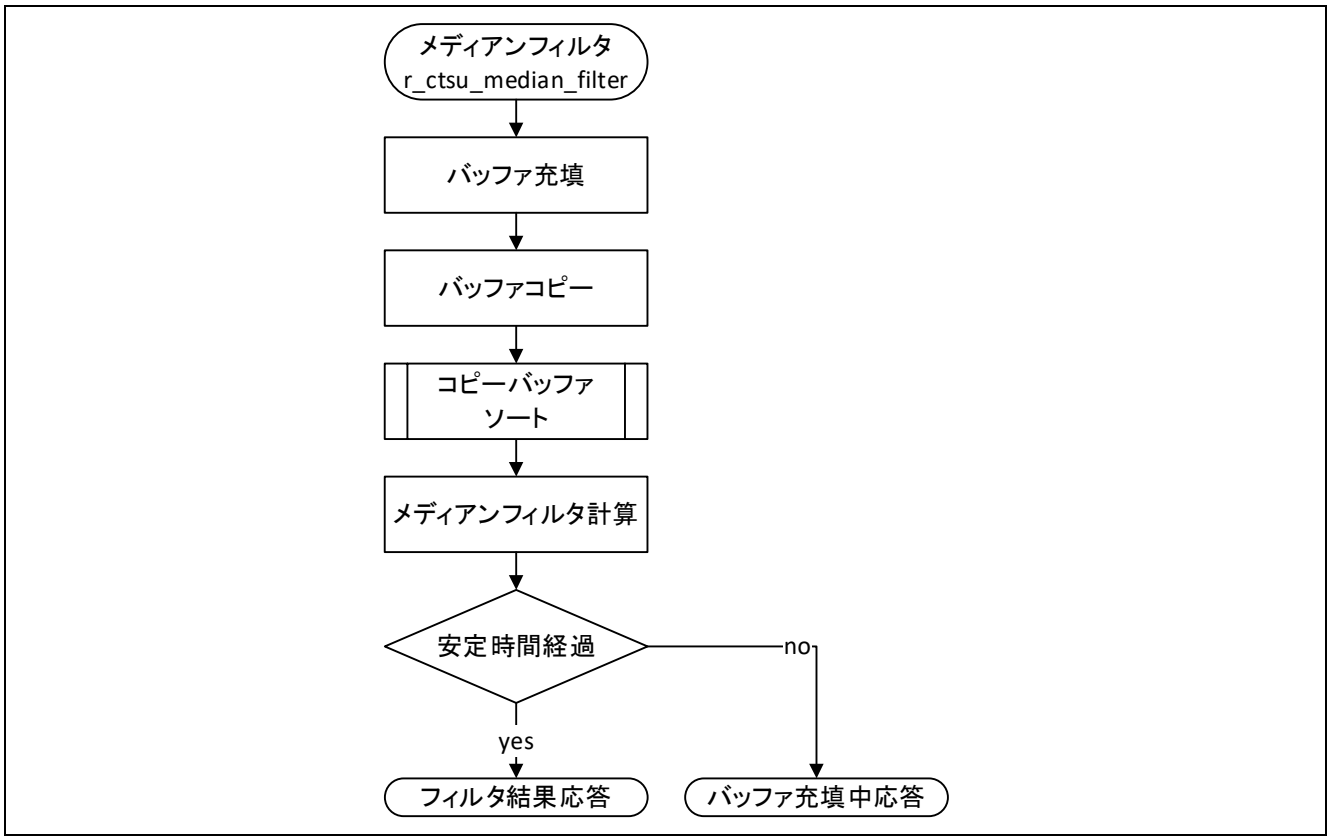


図 5-2 メディアンフィルタ実行 API フロー

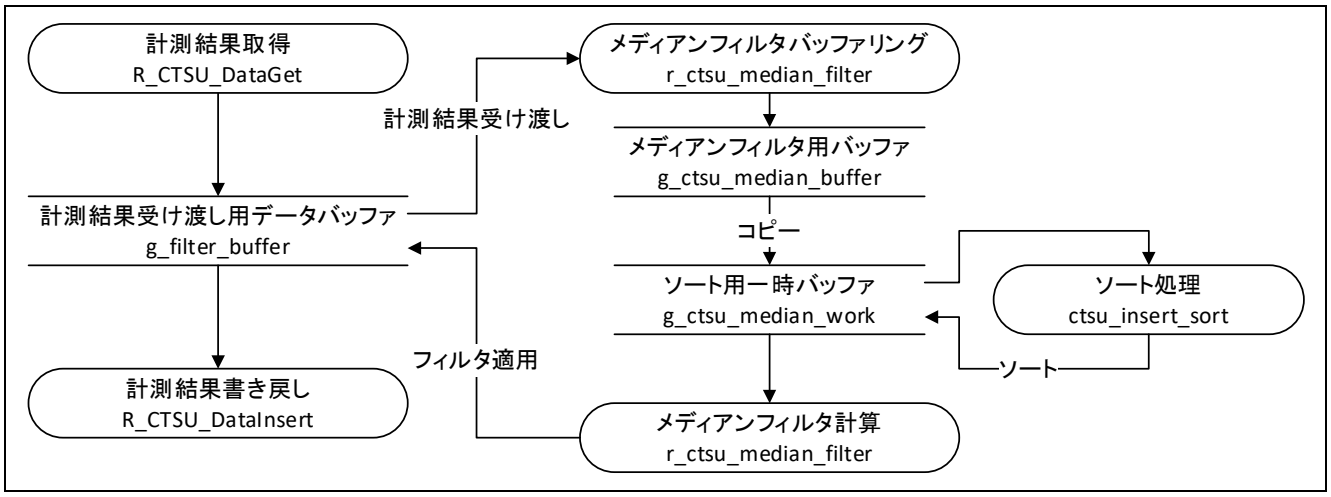


図 5-3 メディアンフィルタデータフロー

5.4.1 ctsu_insert_sort

この関数は、指定したデータを値順に並べ替えます。

Format

static void ctsu_insert_sort(int32_t * list , uint16_t size);

Parameters

p_list

並び替えを行うデータへのポインタ

size

並び替えを行うデータの個数

ReturnValues

なし

Properties

r_ctsu_median_sample.c にプロトタイプ宣言されています。

Description

この関数は、指定したデータを昇順に並び替えます。

5.5 使用例

5.5.1 プログラム実装例

Touch_filter_sample_source ¥ touch_filter_median ¥ filter_sample ¥r_ctsu_filter_sample.c

```
#include "r_ctsu_filter_sample.h"
#include "r_ctsu_fir_sample.h"
#include "r_ctsu_iir_sample.h"
#include "filter_config_sample.h"
#include "r_ctsu.h"

～～

#if (MEDIAN_FILTER_ENABLE == 1)
static fsp_err_t ctsu_median_filter_open (filter_element_ctrl_t * p_ctrl,
filter_ctrl_t const * const p_cfg)
{
    filter_element_ctrl_t * p_element_ctrl = (filter_element_ctrl_t *) p_ctrl;
    fsp_err_t ret = FSP_SUCCESS;
    median_ctrl_t * p_median_ctrl;
    median_config_t * p_median_cfg;
    uint16_t element_id = 0;

    p_median_cfg = (median_config_t *)p_cfg;
    p_element_ctrl->p_filter_ctrl = gp_ctsu_median_ctrl;
    for (element_id = 0; element_id < p_element_ctrl->element_num;
element_id++)
    {
        p_median_ctrl = (median_ctrl_t *)p_element_ctrl->p_filter_ctrl;
        ret = r_ctsu_median_open(&p_median_ctrl[element_id], p_median_cfg);
        if (ret != FSP_SUCCESS)
        {
            return ret;
        }
    }

    return ret;
}
#endif

～～

#if (MEDIAN_FILTER_ENABLE == 1)
          /* Apply MEDIAN filter */
          if (p_instance_ctrl->p_cfg->p_filter_cfg[filter_id].type ==
FILTER_TYPE_MEDIAN)
          {
              p_median_ctrl = (median_ctrl_t *)p_instance_ctrl-
>p_element_ctrl[filter_id].p_filter_ctrl;
              filter_err = r_ctsu_median_filter(&p_median_ctrl[element_id],
&filter_data);
              if (FSP_SUCCESS != filter_err)
              {
                  ret = filter_err;
              }
          }
#endif
```

5.5.2　フィルタ調整手順

本サンプルプログラムでは条件付きコンパイルによりフィルタ特性を指定することができます。

メディアンフィルタ構成のサンプル参照期間指定を変更してフィルタ特性を調整することもできます。

5.5.2.1　フィルタ処理方法

メディアンフィルタでは計測値をサンプリング、ソートして中央値を演算するためサンプリング数が大きくなるほど処理時間が長くなります。

データサイズと処理時間については表 2-13 及び表 2-14 を参照してください。

図 5-4 にメディアンフィルタのブロック図を示します。

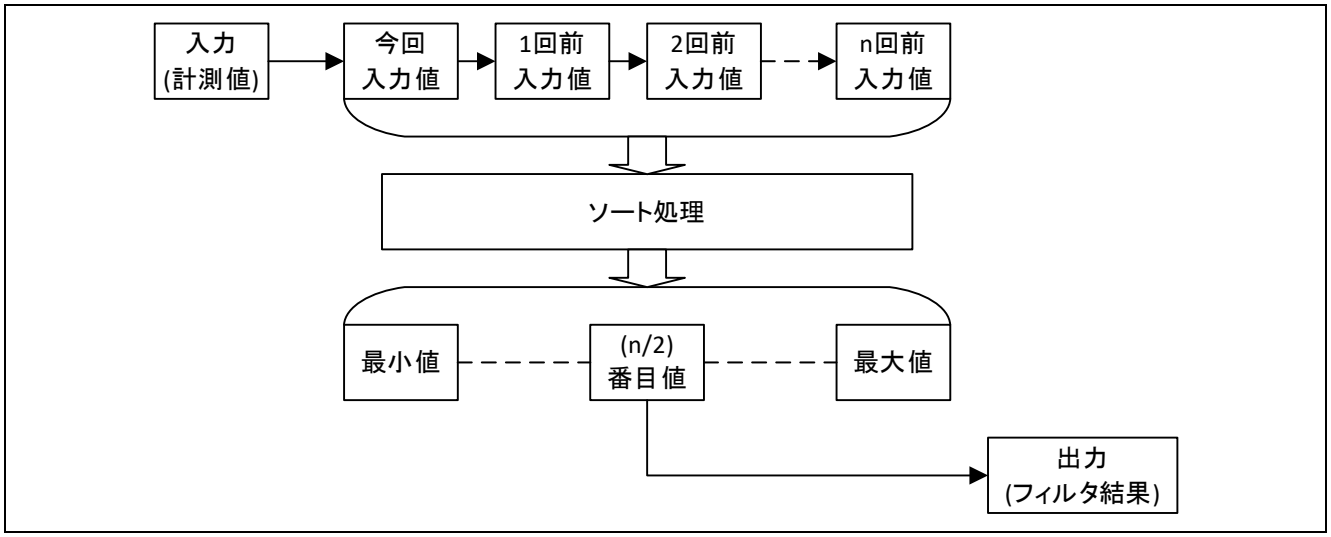


図 5-4 メディアンフィルタブロック図

5.5.3 フィルタ特性

本サンプルプログラムではサンプル参照期間が 3 から 9 のフィルタを扱うことができます。

表 5-9 にサンプルメディアンフィルタの特性指定の定義を示します。表 5-10 にフィルタ構成定義を示します。

表 5-9 サンプルメディアンフィルタ特性指定

ファイル	定義名	内容
r_ctsu_median_sample.h	MEDIAN_PRESET_TYPE	メディアンフィルタで使用するサンプルプリセット指定

表 5-10 サンプルメディアンフィルタ構成定義

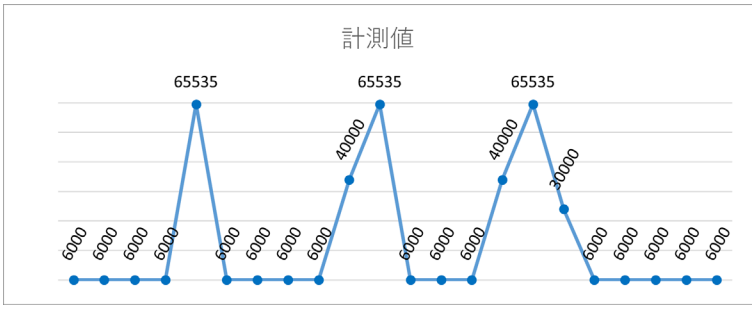
	MEDIAN_PRESET_TYPE_1	MEDIAN_PRESET_TYPE_2
サンプル参照期間	3	5
除去ノイズ幅	1(20ms)	2(40ms)
検出遅延	1(20ms)	2(40ms)

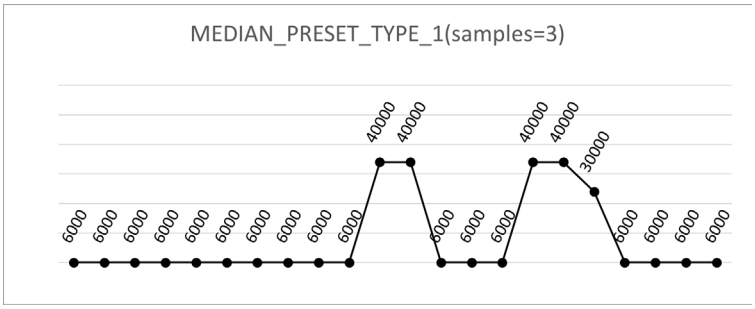
5.5.3.1 除去可能ノイズ幅について

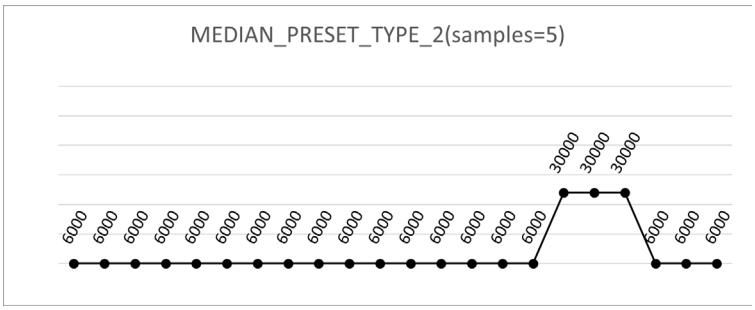
メディアンフィルタではサンプリング周期×除去ノイズ幅以下のノイズ信号は除去されます。

除去ノイズ幅は(サンプル参照期間-1)÷2 で計算され、サンプリング周期×除去ノイズ幅を超えるノイズ信号は除去ノイズ幅分の信号を除去した形状となります。

サンプルプログラムのプリセット指定では除去ノイズ幅 1~2 を対象としていますので、それ以上の幅のノイズ信号を除去する場合は 5.3.3.1 章を参照してサンプル参照期間を 7 以上に設定してください。







- 信号の幅が 20ms(ノイズ幅 1)のノイズ信号は PRESET1(サンプル参照期間=3)、PRESET2(サンプルサンプル参照範囲=5)のどちらでも除去されます。

- 信号の幅が 40ms(ノイズ幅 2)のノイズ信号は PRESET1(サンプル参照期間=3)では信号の先端部分 1 点(65535)が削除された形状になり、PRESET2(サンプル参照期間=5)では除去されます

- 信号の幅が 60ms(ノイズ幅 3)のノイズ信号は PRESET1(サンプル参照期間=3)では信号の先端部分 1 点(65535)が削除された形状になり、PRESET2(サンプル参照期間=5)では信号の先端部分 2 点(65535,40000)が削除された形状になります

ノイズ幅 1(20ms)の場合

計測値	フィルタ結果	
	PRESET1	PRESET2
6000	6000	6000
6000	6000	6000
6000	6000	6000
6000	6000	6000
65536	6000	6000
6000	**6000**	6000
6000	6000	**6000**
6000	6000	6000

ノイズ幅 2(40ms)の場合

計測値	フィルタ結果	
	PRESET1	PRESET2
6000	6000	6000
6000	6000	6000
6000	6000	6000
6000	6000	6000
40000	6000	6000
65536	**40000**	6000
6000	**40000**	**6000**
6000	6000	**6000**
6000	6000	6000

ノイズ幅 3(60ms)の場合

計測値	フィルタ結果	
	PRESET1	PRESET2
6000	6000	6000
6000	6000	6000
6000	6000	6000
6000	6000	6000
40000	6000	6000
65536	**40000**	6000
30000	**40000**	**30000**
6000	**30000**	**30000**
6000	6000	**30000**
6000	6000	6000

5.5.3.2　検出遅延について

メディアンフィルタではタッチ計測値のサンプリングによってノイズ信号の除去を行うため通常のタッチ検出に遅延が発生します。

タッチ検出の遅延時間は除去可能ノイズ幅と同じ時間(サンプリング周期×除去ノイズ幅）となります。

6. 本サンプルプロジェクトの使用方法

6.1 フィルタサンプルプログラム

6.1.1 既存プロジェクトへのフィルタ組み込み手順

既存の静電容量タッチアプリケーションに FIR フィルタを組み込む場合は以下の手順で行います。

IIR フィルタを組み込む場合はフォルダ名、ファイル名等を IIR フィルタのものに読み替えて手順を実施してください。

1.Touch_filter_sample_source¥touch_filter_fir フォルダ内にある filter_sample フォルダを対象のプロジェクトにコピーします。

2.メニューのプロジェクト⇒C/C++Project Settings を開き、C/C++一般⇒パスおよびシンボルのインクルードとソース・ロケーションに filter_sample フォルダを追加します。

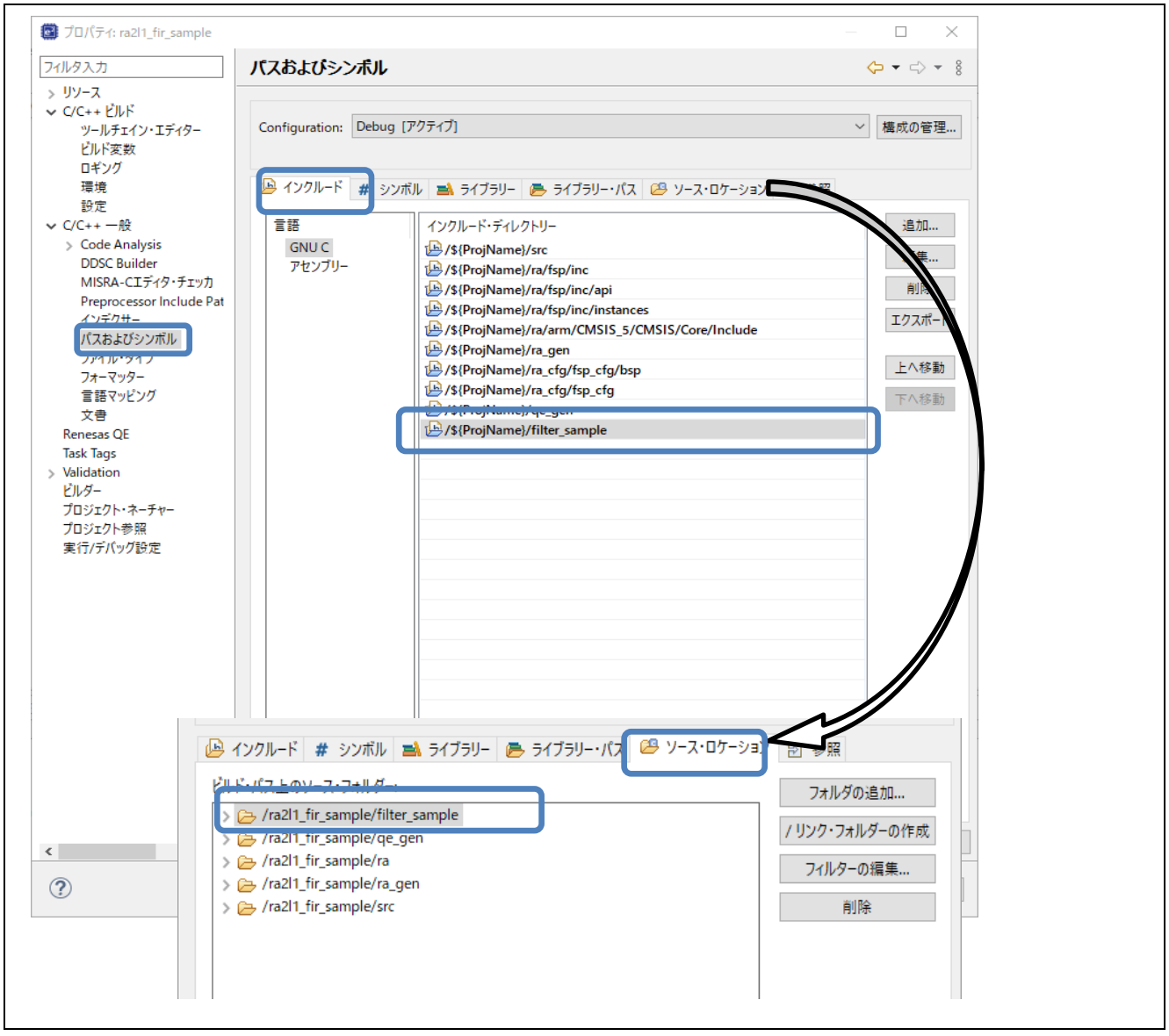


図 6-1 既存環境へのサンプルプログラムの組み込み

3. 組み込み環境のタッチインターフェース構成のメソッド（構成）数に合わせてフィルタ構成定義を追加します。
   qe_touch_config.c ファイルを確認し、touch_instance_t 型のデータ定義と同数になるように filter_config_sample.c ファイルの ctsu_filter_instance_t 型のデータ定義と filter_instance_ctrl_t 型のデータを追加してください。

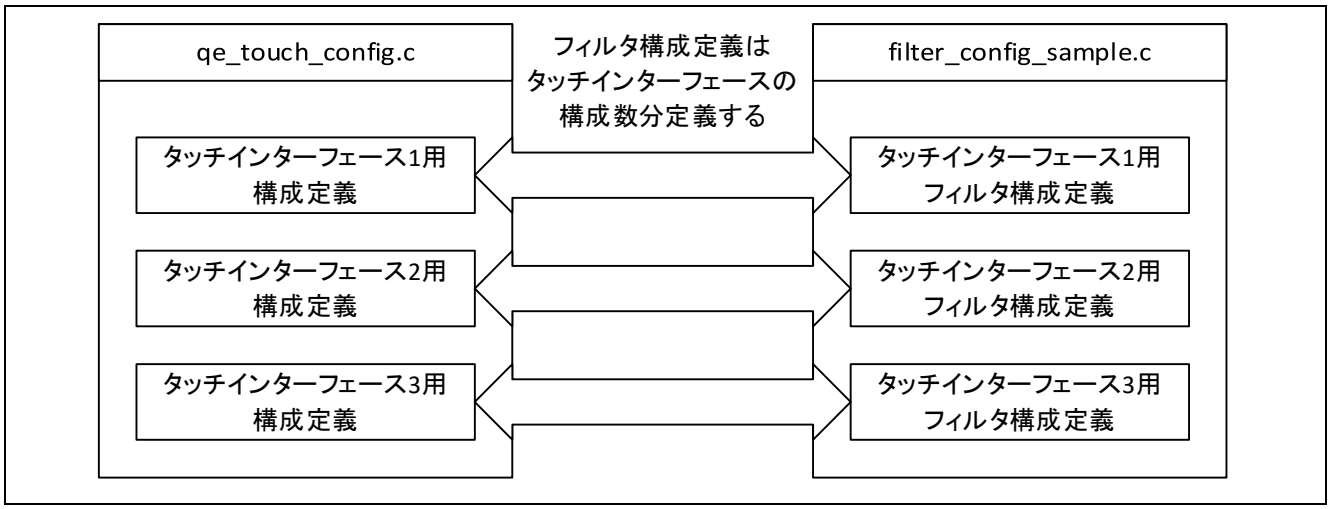


図 6-2 フィルタ構成定義の追加

- タッチインターフェース構成定義の記述例

qe_touch_config.c（QE for Capacitive Touch で生成されるファイル）

```
touch_instance_ctrl_t g_qe_touch_ctrl_config01;
const touch_instance_t g_qe_touch_instance_config01 =

（省略）

touch_instance_ctrl_t g_qe_touch_ctrl_config02;
const touch_instance_t g_qe_touch_instance_config02 =

（省略）

touch_instance_ctrl_t g_qe_touch_ctrl_config03;
const touch_instance_t g_qe_touch_instance_config03 =
```

- フィルタ構成定義の記述例

filter_config_sample.c

```
filter_instance_ctrl_t g_ctsu_filter_control01;
const ctsu_filter_instance_t g_ctsu_filter_instance01 =

（省略）

filter_instance_ctrl_t g_ctsu_filter_control02;
const ctsu_filter_instance_t g_ctsu_filter_instance02 =

（省略）

filter_instance_ctrl_t g_ctsu_filter_control03;
const ctsu_filter_instance_t g_ctsu_filter_instance03 =
```

構成定義の個数を合わせる

4. 使用する環境に合わせて filter_config_sample.c のフィルタ構成定義を変更し、適用するフィルタを指定してください。(2.3.4 章を参照してください。)
   FIR フィルタの場合は条件付きコンパイル FIR_PRESET_TYPE で 4 パターンのサンプルプリセットからフィルタ特性を指定する事ができます。

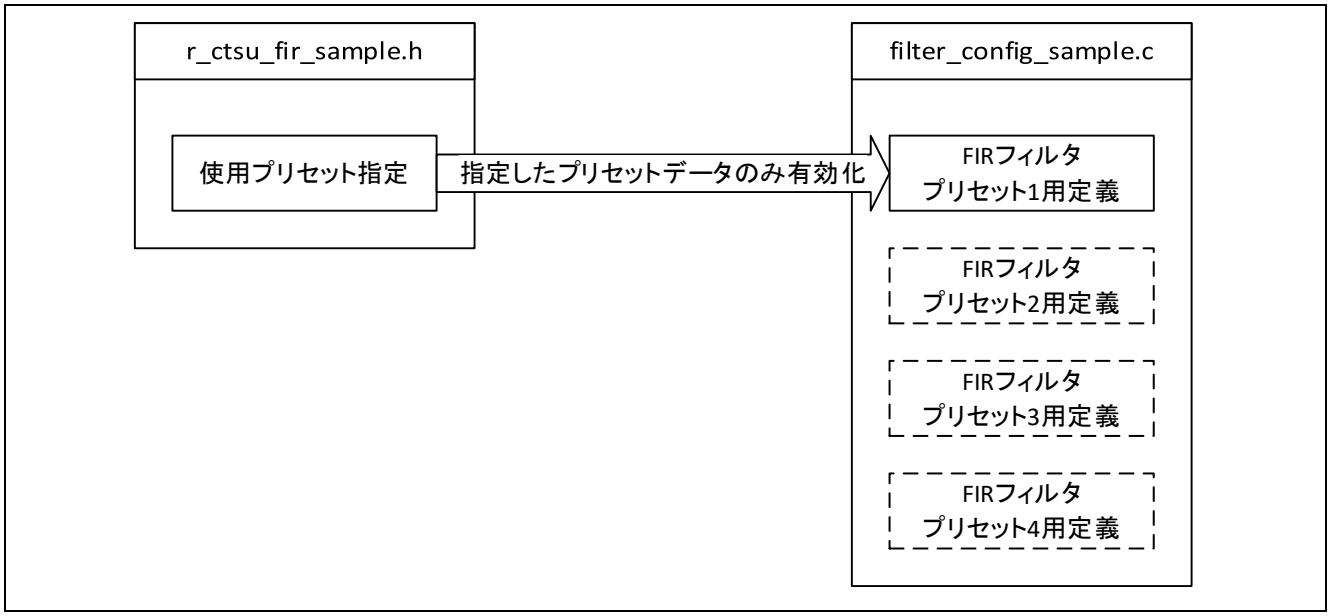


図 6-3 使用するフィルタ指定(FIR)

- FIR フィルタのプリセット指定記述例

r_ctsu_fir_sample.h

```
#define  FIR_FILTER_ENABLE (1)

#define  FIR_FILTER_TYPE_DIRECT      (0)
#define  FIR_FILTER_TYPE_TRANSPOSE   (1)
#define  FIR_FILTER_TYPEFIR_FILTER_TYPE_DIRECT

#if (FIR_FILTER_ENABLE == 1)
#define FIR_PRESET_TYPE_1   (1)
#define FIR_PRESET_TYPE_2   (2)
#define FIR_PRESET_TYPE_3   (3)
#define FIR_PRESET_TYPE_4   (4)
#define FIR_PRESET_TYPE     FIR_PRESET_TYPE_1
#define FIR_FILTER_NUM      (1)
#else
#define FIR_PRESET_TYPE     (0)
#endif
```

- フィルタ構成定義の記述例

filter_config_sample.c

```
const filter_element_config_t g_ctsu_filter_element_config[] =
{
#if (FIR_PRESET_TYPE == FIR_PRESET_TYPE_1)
  {
    .type  = FILTER_TYPE_FIR,
    . filter_element_cfg = &fir_cfg01,
  },
#endif
#if (FIR_PRESET_TYPE == FIR_PRESET_TYPE_2)
  {
    .type  = FILTER_TYPE_FIR,
    . filter_element_cfg = &fir_cfg02,
  },
#endif

(省略)

};
```

5. qe_touch_sample.c ファイル(もしくはそれに準ずるファイル)に filter_config_sample.h ファイルのインクルード指定を追加し、フィルタ処理実施コードを追加します(6.1.2 章を参照してください)。

【注】 1. フィルタ処理用のデータ読み出し、データ書き戻しは Touch API ではなく CTSU API であることに注意してください。
2. フィルタ処理の実行記載はタッチインターフェース構成のメソッド毎に必要であることに注意してください。

6. qe_touch_config.c ファイル(もしくはそれに準ずるファイル)内の CTSU ドライバ構成定義(QE for Capacitive Touch 生成では g_qe_ctsu_ctrl_XXX)の num_moving_average 設定を 1 に変更しデフォルトの移動平均処理を無効にします。デフォルトの移動平均処理と FIR フィルタを併用する場合は変更は不要です。
タッチインターフェース構成のメソッドが複数ある場合は全てのメソッドの CTSU ドライバ構成定義を変更してください。

- タッチインターフェース構成定義の記述例

qe_touch_config.c（QE for Capacitive Touch で生成されるファイル）

```
const ctsu_cfg_t g_qe_ctsu_cfg_config01 =
{
(省略)
    .num_moving_average = 1,
    .tunning_enable     = true,
    .p_callback    = &qe_touch_callback,
(省略)
};

ctsu_instance_ctrl_t g_qe_ctsu_ctrl_config01;

const ctsu_instance_t g_qe_ctsu_instance_config01 =
{
    .p_ctrl = &g_qe_ctsu_ctrl_config01,
    .p_cfg  = &g_qe_ctsu_cfg_config01,
    .p_api  = &g_ctsu_on_ctsu,
};


const ctsu_cfg_t g_qe_ctsu_cfg_config02 =
{
(省略)
    .num_moving_average = 1,
    .tunning_enable     = true,
    .p_callback    = &qe_touch_callback,
(省略)
};

ctsu_instance_ctrl_t g_qe_ctsu_ctrl_config02;

const ctsu_instance_t g_qe_ctsu_instance_config02 =
{
    .p_ctrl = &g_qe_ctsu_ctrl_config02,
    .p_cfg  = &g_qe_ctsu_cfg_config02,
    .p_api  = &g_ctsu_on_ctsu,
};
```

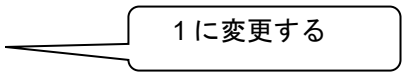
1 に変更する

1 に変更する

```
const ctsu_cfg_t g_qe_ctsu_cfg_config03 =
{
(省略)
    .num_moving_average = 1,
    .tunning_enable     = true,
    .p_callback    = &qe_touch_callback,
(省略)
};

ctsu_instance_ctrl_t g_qe_ctsu_ctrl_config03;

const ctsu_instance_t g_qe_ctsu_instance_config03 =
{
    .p_ctrl = &g_qe_ctsu_ctrl_config03,
    .p_cfg  = &g_qe_ctsu_cfg_config03,
    .p_api  = &g_ctsu_on_ctsu,
};
```



6.1.2　サンプルアプリケーション構成と動作

QE for Capacitive Touch で出力したサンプルコード(qe_touch_sample.c)にフィルタサンプルプログラムを組み込む場合のフローチャートを示します。本例はタッチインターフェース構成(メソッド)が 3 個ある場合を示します。

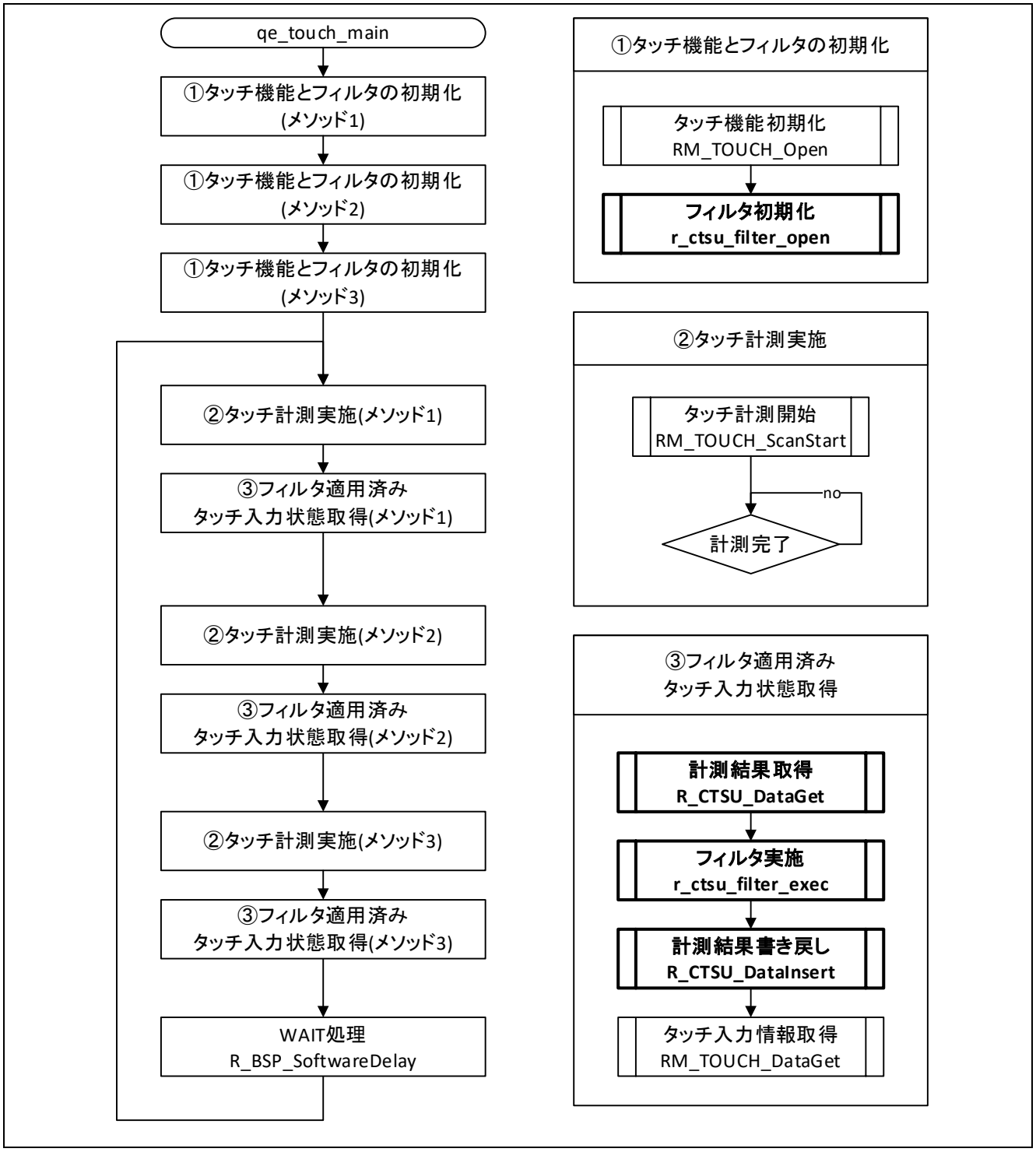


図 6-4 サンプルアプリケーションフロー

図 6-4 の図中番号の説明を示します。

① タッチ機能とフィルタの初期化

タッチ機能の初期化とフィルタの初期化を実施します。

フィルタの初期化にはタッチインターフェース構成を確認して各メソッドの対応する CTSU ドライバ構成定義を指定してください。

```
/* Open Touch middleware */
err = RM_TOUCH_Open(p_touch_instance->p_ctrl, p_touch_instance->p_cfg);
if (FSP_SUCCESS == err)
{
 /* Open filter sample software */
 err = r_ctsu_filter_open(p_filter_instance->p_ctrl, p_filter_instance->p_cfg,
p_ctsc_instance->p_cfg);
}
```

② タッチ計測実施

タッチ計測を行い、計測完了を待ちます。

②~③はタッチインターフェース構成のメソッド単位で連続して実施されるようにしてください。

③ フィルタ適用済みタッチ入力状態取得

CTSU ドライバ API を使用して計測結果を取得し、フィルタ適用後に CTSU ドライバに書き戻し、フィルタ適用後のデータを使用してタッチ入力情報を取得します。

②~③はタッチインターフェース構成のメソッド単位で連続して実施されるようにしてください。

CTSU ドライバ API の詳細は Renesas Flexible Software Package (FSP) User's Manual (R11UM0155)の v4.3.0 以降を参照してください。

```
/* Use filter sample software */
err = R_CTSU_DataGet(p_ctsc_instance->p_ctrl, g_filter_buffer);
if (FSP_SUCCESS == err)
{
 err = r_ctsu_filter_exec(p_filter_instance->p_ctrl, g_filter_buffer);
 if (FSP_SUCCESS == err)
 {
    R_CTSU_DataInsert(p_ctsc_instance->p_ctrl, g_filter_buffer);
    err = RM_TOUCH_DataGet(p_touch_ctrl, p_button_status, p_slider_position,
p_wheel_position);
 }
}
```

6.2　フィルタサンプルを実装した Example Project

RA2L1 静電容量タッチ評価システム Example Project に対しソフトウェアフィルタサンプルプログラムを適用したサンプルプロジェクト(ra2l1_rssk_filter_sample)の使用方法を説明します。

6.2.1　機能

以下に機能を示します。

- 自己容量電極ボードのすべてのタッチ電極の計測結果に対し、ソフトウェアフィルタを適用します。
- 自己容量電極ボードのタッチ電極をタッチすると対応した箇所の LED が点灯します。
- ソフトウェアフィルタを適用した計測結果は QE for Capacitive Touch のシリアルモニター機能で確認できます。

6.2.2　ファイル構成

サンプルプロジェクトのファイル構成を示します。

開発環境のプロジェクト構成ファイルと FSP Configuration 生成ファイルは省略しています。

Example Project との差分は太字で示しています。変更のないファイル内容の詳細は、RA2L1 グループ RA2L1 搭載静電容量タッチ評価システム Example Project (R20AN0595)を参照してください。

```
ra2l1_rssk_filter_sample
 |
 ├──QE-Touch
 |     ├ qe_tuning20230221103059.log      ・・・QE チューニングログ
 |     └ quickstart_rssk_ra2l1_ep.tifcfg  ・・・タッチインターフェース構成ファイル
 |
 ├──qe_gen
 |     ├ qe_touch_config.c                ・・・タッチコンフィグレーションソース
 |     ├ qe_touch_config.h                ・・・タッチコンフィグレーションヘッダ
 |     ├ qe_touch_define.h                ・・・タッチ定義ヘッダ
 |     └ qe_touch_sample.c                ・・・タッチサンプルアプリケーション
 |
 ├──src
 |     ├ hal_entry.c                      ・・・main ファイル
 |     ├ r_rssk_switch_led.c              ・・・スイッチと LED 処理用ソース
 |     ├ r_rssk_switch_led.h              ・・・スイッチと LED 処理用ヘッダ
 |     ├ r_rssk_touch_led.c               ・・・タッチ電極 LED 処理用ソース
 |     └ r_rssk_touch_led.h               ・・・タッチ電極 LED 処理用ヘッダ
 |
 └──filter_sample
       ├ filter_config_sample.c           ・・・フィルタ構成定義用ソース
       ├ filter_config_sample.h           ・・・フィルタ構成定義用ヘッダ
       ├ fir_config_sample1.c             ・・・FIR フィルタサンプルプリセット 1 用ソース
```

- fir_config_sample2.c ・・・FIR フィルタサンプルプリセット 2 用ソース
- fir_config_sample3.c ・・・FIR フィルタサンプルプリセット 3 用ソース
- fir_config_sample4.c ・・・FIR フィルタサンプルプリセット 4 用ソース
- iir_config_sample1.c ・・・IIR フィルタサンプルプリセット 1 用ソース
- iir_config_sample2.c ・・・IIR フィルタサンプルプリセット 2 用ソース
- iir_config_sample3.c ・・・IIR フィルタサンプルプリセット 3 用ソース
- iir_config_sample4.c ・・・IIR フィルタサンプルプリセット 4 用ソース
- iir_config_sample5.c ・・・IIR フィルタサンプルプリセット 5 用ソース
- iir_config_sample6.c ・・・IIR フィルタサンプルプリセット 6 用ソース
- median_config_sample1.c ・・・メディアンフィルタサンプルプリセット 1 用ソース
- median_config_sample2.c ・・・メディアンフィルタサンプルプリセット 2 用ソース
- r_ctsu_filter_sample.c ・・・フィルタ処理用ソース
- r_ctsu_filter_sample.h ・・・フィルタ処理用ヘッダ
- r_ctsu_fir_sample.c ・・・FIR フィルタ処理用ソース
- r_ctsu_fir_sample.h ・・・FIR フィルタ処理用ヘッダ
- r_ctsu_iir_sample.c ・・・IIR フィルタ処理用ソース
- r_ctsu_iir_sample.h ・・・IIR フィルタ処理用ヘッダ
- r_ctsu_median_sample.c ・・・メディアンフィルタ処理用ソース
- r_ctsu_median_sample.h ・・・メディアンフィルタ処理用ヘッダ

6.2.3　インポート方法

本サンプルコードに付属している「ra2l1_rssk_filter_sample」フォルダを e²studio のインポート機能によりワークスペースへインポートしてください。

図 6-5 にサンプルプロジェクトのインポート方法を示します。

インポート後の操作については Renesas RA ファミリ RA2L1 グループ 静電容量タッチ評価システム クイックスタートガイド(Q12QS0040)を参照してください。

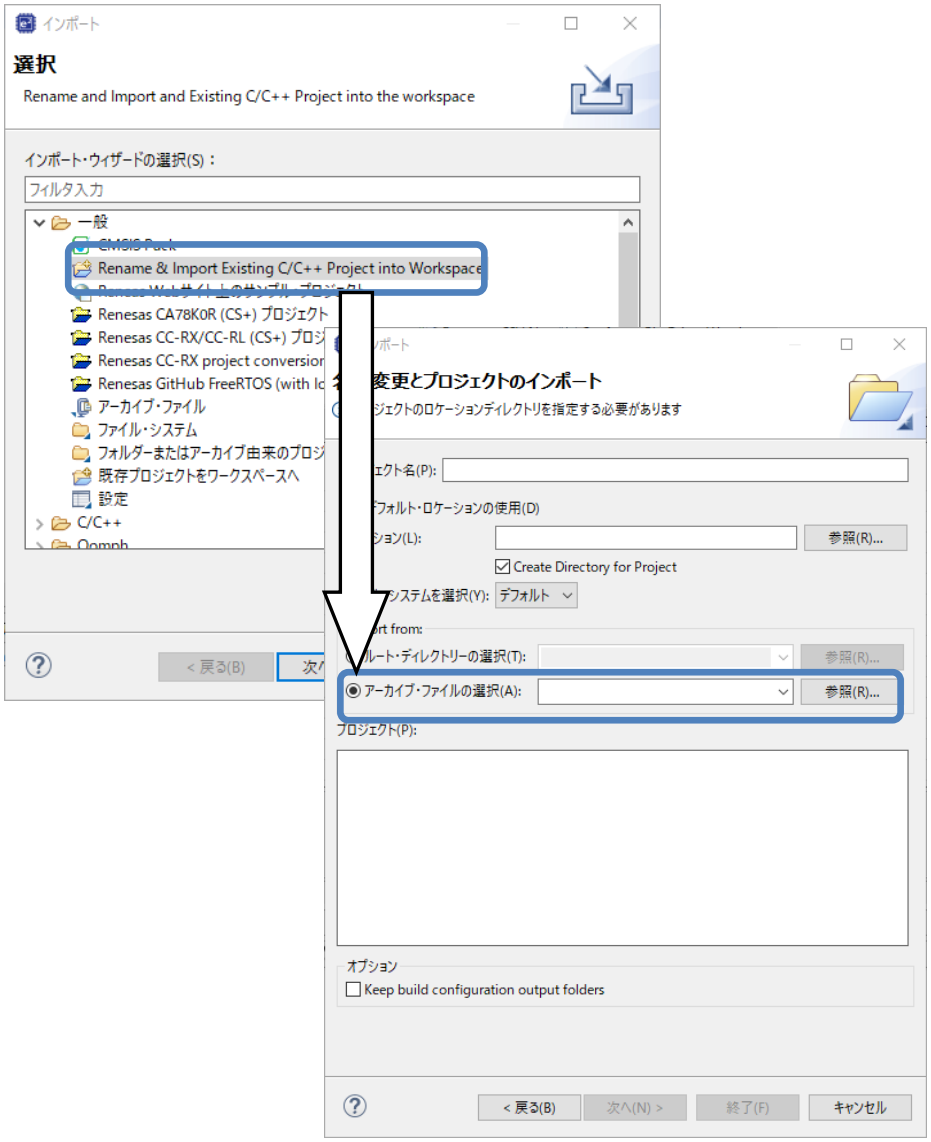


図 6-5 サンプルプロジェクトのインポート

6.2.4　フィルタ構成およびプリセット変更方法

Example Project で適用するフィルタ構成（種類）を変更するには「表 2-2 フィルタ構成定義用定数」を参照してください。

FIR フィルタのプリセットを変更するには「表 3-6 サンプル FIR フィルタ特性指定」を参照してください。

IIR フィルタのプリセットを変更するには「表 4-5 サンプル IIR フィルタ特性指定」を変更してください。

メディアンフィルタのプリセットを変更するには、表 5-9 サンプルメディアンフィルタ特性指定を変更してください。

7. 参考資料

- 静電容量センサマイコン 静電容量タッチ ノイズイミュニティガイド(R30AN0426)
- Renesas RA ファミリ RA2L1 グループ 静電容量タッチ評価システム クイックスタートガイド (Q12QS0040)
- RA ファミリ QE と FSP を使用した静電容量タッチアプリケーションの開発(R01AN4934)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

https://www.renesas.com/

静電容量センサユニット関連ページ

https://www.renesas.com/rssk-touch-ra2l1

https://www.renesas.com/qe-capacitive-touch

お問い合わせ

https://www.renesas.com/support

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jun.12.23	-	新規発行
2.00	Aug.31.23		文書全体の再構成 4. IIR フィルタを追加 1.1 のフォルダ構造に IIR フィルタ関連の項目を追加。 図 2-1 を修正 2.2 のファイル構造に IIR フィルタ関連の項目を追加 表 3-1 の係数データ型に関する注記を修正 3.5.4 誤記修正 IIR フィルタ関連項目をセクションに追加
3.00	Nov.30.23	4 5 6 7 8 9 10 13 19 24 24 26 28 28 31 45 49 56 68	1.1 のフォルダ構造にメディアンフィルタ関連の項目を追加 表 1-1 動作確認条件のバージョンを更新 2. 追加予定機能についての記載を削除 表 2-1 コンポーネント一覧のバージョンを更新 2.2 のファイル構造にメディアンフィルタ関連の項目を追加、フォルダの説明記載を追加 表 2-2 各フィルタの有効、無効定義を追加 表 2-3 メディアンフィルタを追加 表 2-12 メディアンフィルタを追加 図 2-5 メディアンフィルタを追加 2.4.6 メディアンフィルタを追加 2.4.6 メディアンフィルタ用初期設定 API を追加 2.4.7 メディアンフィルタについて記載を追加 表 2-13 データサイズ及び増分量を更新 表 2-14 フィルタ処理の実行時間を更新 3.3.1 誤記修正 4.3.1 誤記修正 表 4.4 誤記修正 5. メディアンフィルタを追加 6. 章の構成を変更

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。

© 2023 Renesas Electronics Corporation. All rights reserved.