# RA family

## Capacitive Touch Software Filter Sample Program

### Introduction

This application note describes software filters for capacitive touch systems.

### Target Device

RA2L1 Group (R7FA2L1AB2DFP)

When applying the contents of this application note to other MCUs, please change them according to the specifications of the MCUs and perform a thorough evaluation.

### Contents

# 1. Overview

This application note describes the operation of the software filter sample program and how to incorporate it into an existing project.

For more information on software filters, refer to the Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426).

## 1.1 Folder Structure

The following shows the folder structure of this sample program.

This sample program consists of a storage folder (Touch_filter_sample_source) for the Capacitive Touch Software Filter Sample Program and a sample project (ra2l1_rssk_filter_sample) which applies the Software Filter Sample Program to RA2L1 Capacitive Touch Evaluation System Example Project (R20AN0595).

The RA2L1 Capacitive Touch Evaluation System Example Project is referred to as Example Project in the following.

```
an-r01an0427ej0100-capacitive-touch
|
├─ Touch_filter_sample_source          ・・・Filter Sample Storage Folder
|        └─touch_filter_fir            ・・・FIR Filter Sample Storage Folder
|        |      └ filter_sample        ・・・FIR Filter Sample Program
|        └─touch_filter_iir            ・・・IIR Filter Sample Storage Folder
|               └ filter_sample        ・・・IIR Filter Sample Program
|        └─touch_filter_median         ・・・Median Filter Module Sample Storage Folder
|               └ filter_sample        ・・・Median Filter Sample Program

|
└─ ra2l1_rssk_filter_sample            ・・・Sample Project
                                          (RA2L1 Capacitive Touch Evaluation System)
```

## 1.2 Operation Confirmation Conditions

Table 1.1 shows the operation confirmation conditions of the sample program in this application note.

**Table 1.1 Operation Confirmation Conditions**

| Item | Description |
|---|---|
| Microcontroller used | RA2L1 (R7FA2L1AB2DFP) |
| Operating frequency | High-speed on-chip oscillator 48MHz |
| Operating voltage | 5V |
| Board | Capacitive Touch Evaluation System with RA2L1 (Model: RTK0EG0022S01001BJ)<br>• RA2L1 CPU (Model: RTK0EG0018C01001BJ)<br>• Self-Capacitance Touch Button/Wheel/Slider Board (Model: RTK0EG0019B01002BJ) |
| Integrated development environment | e$^2$ studio Version 2023-01 (23.1.0) |
| C compiler | GCC Arm Embedded 10.3-2021.10 |
| FSP | V5.0.0 |
| Development Assistance Tool for Capacitive Touch Sensors | QE for Capacitive Touch V3.2.0 |
| Emulator | Renesas E2 emulator Lite |

Figure 1-1 shows the device connection diagram.

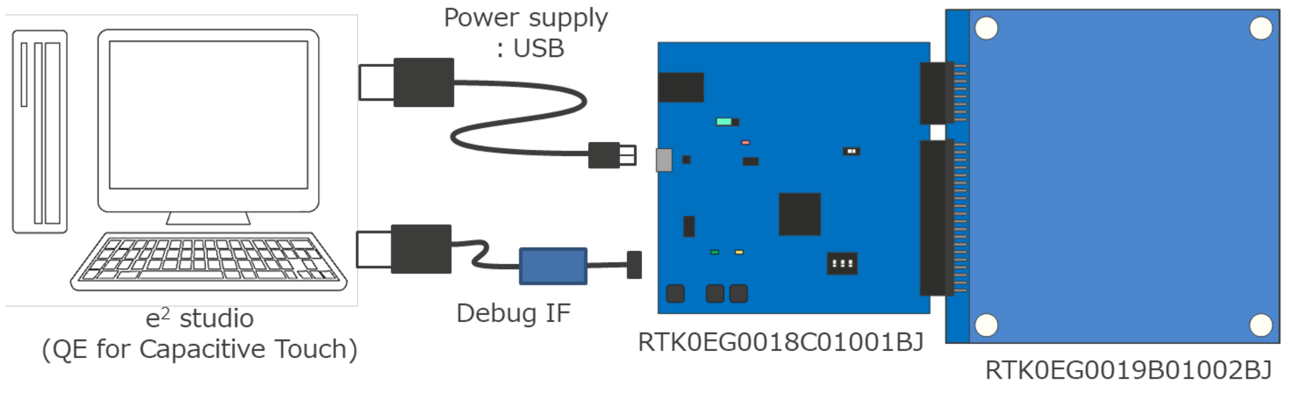



**Figure 1-1 Device Connection Diagram**

## 1.3 Correspondence Between Sample Code and Application Note

Please review Figure 2-1 Configuration of Sample Program and Figure 2-2 Data Processing Flow of Sample Program before using this sample project.

Also review 6.1 Sample Filter Program for details on how to embed the filter module into an existing capacitive touch project and 6.2 Example Project Integrating Filter Sample Program for details on how to use the Example Project programmed with the filter sample.

Filter specifications and parameter setting methods are described in 2. Software Specifications, 3. FIR Filters, 4. IIR Filters, and 5. Median Filters.

## 2. Software Specifications

This sample program operates as a software filter by applying a filter API to the data acquired by Touch API and CTSU API. Manage the software filters you use in the filter configuration definition. The filter configuration described in the sample program consists of Filter A (FIR filter), Filter B (IIR filter), and Filter C (Median filter), but multiple software filters can also be used. When multiple software filters are used, the application order is the order in which the filter configuration definitions are defined.

### 2.1 Software Configuration Diagram

Figure 2-1 shows the configuration of this sample program.

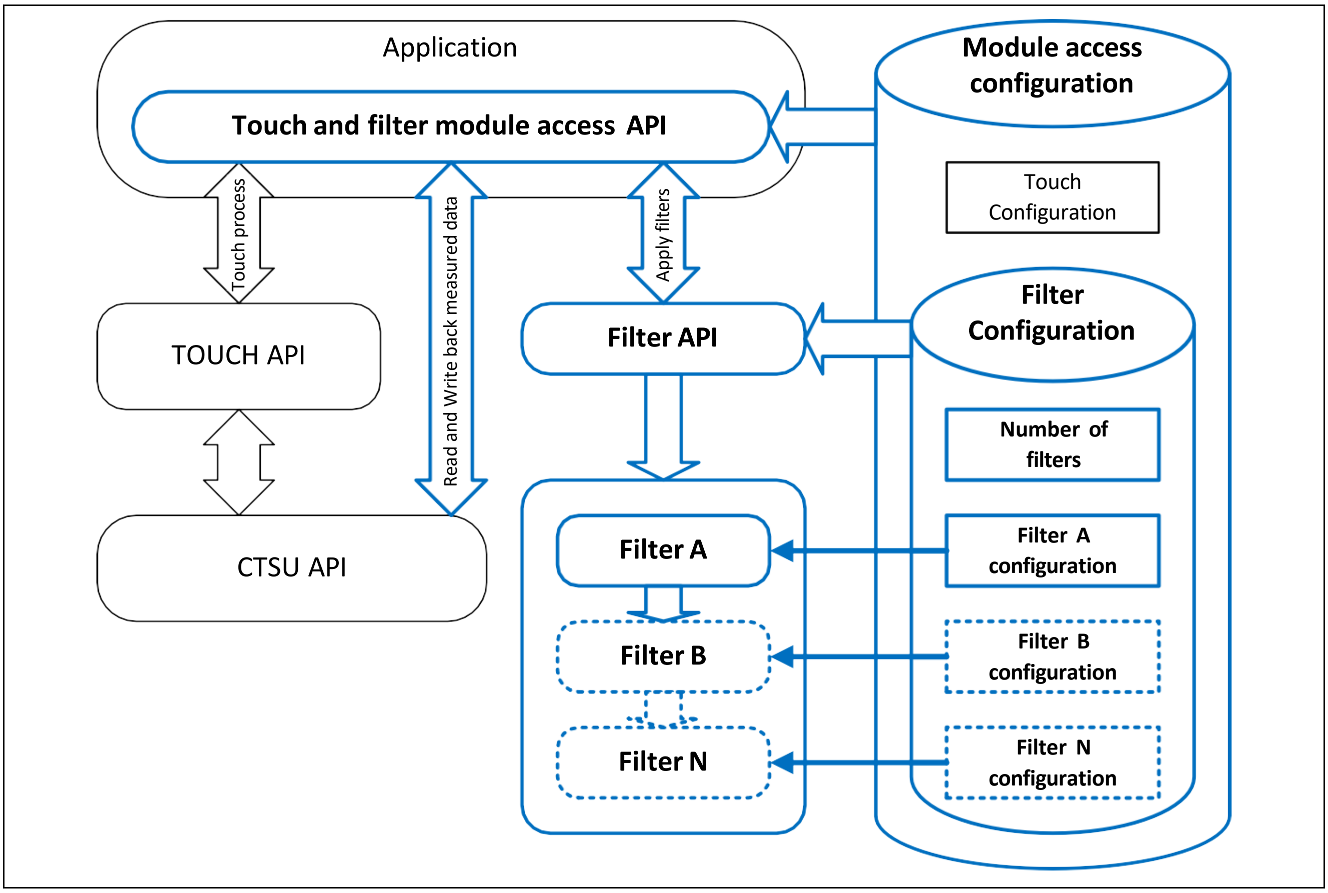



**Figure 2-1 Configuration of Sample Program**

Figure 2-2 shows the flow of data processing for this sample program.

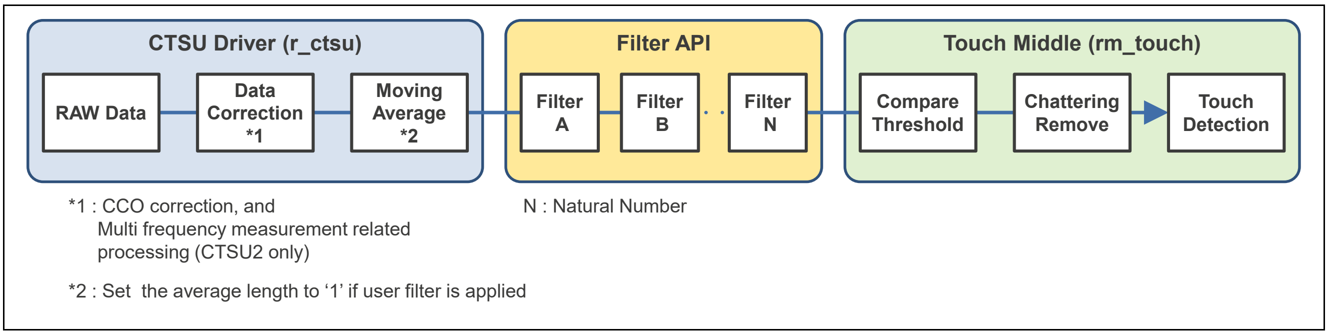



**Figure 2-2 Data Processing Flow of Sample Program**

Table 2.1 lists the components and versions. Refer to FSP Configuration for component settings.

**Table 2.1 List of Components**

| Component | Version |
|---|---|
| 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 File Structures

The following tree shows the file structure for the sample code.

```
ra2l1_rssk_filter_sample
├─Touch_filter_sample_source
│        ├─touch_filter_fir                    ・・・FIR Filter Sample Module Storage Folder
│        │        └─filter_sample
│        │                filter_config_sample.c   ・・・Filter Configuration Definition Sample Source
│        │                filter_config_sample.h   ・・・Filter Configuration Definition Sample Header
│        │                fir_config_sample1.c     ・・・FIR Filter Sample Preset 1 Source
│        │                fir_config_sample2.c     ・・・FIR Filter Sample Preset 2 Source
│        │                fir_config_sample3.c     ・・・FIR Filter Sample Preset 3 Source
│        │                fir_config_sample4.c     ・・・FIR Filter Sample Preset 4 Source
│        │                r_ctsu_filter_sample.c   ・・・Filter API Sample Program Source
│        │                r_ctsu_filter_sample.h   ・・・Filter API Sample Program Header
│        │                r_ctsu_fir_sample.c      ・・・FIR Filter Sample Program Source
│        │                r_ctsu_fir_sample.h      ・・・FIR Filter Sample Program Header
│        └─touch_filter_iir                    ・・・IIR Filter Sample Module Storage Folder
│                 └─filter_sample
│                         filter_config_sample.c   ・・・Filter Configuration Definition Sample Source
│                         filter_config_sample.h   ・・・Filter Configuration Definition Sample Header
│                         iir_config_sample1.c     ・・・IIR Filter Sample Preset 1 Source
│                         iir_config_sample2.c     ・・・IIR Filter Sample Preset 2 Source
│                         iir_config_sample3.c     ・・・IIR Filter Sample Preset 3 Source
│                         iir_config_sample4.c     ・・・IIR Filter Sample Preset 4 Source
│                         iir_config_sample5.c     ・・・IIR Filter Sample Preset 5 Source
│                         iir_config_sample6.c     ・・・IIR Filter Sample Preset 6 Source
│                         r_ctsu_filter_sample.c   ・・・Filter API Sample Program Source
│                         r_ctsu_filter_sample.h   ・・・Filter API Sample Program Header
│                         r_ctsu_iir_sample.c      ・・・IIR Filter Sample Program Source
│                         r_ctsu_iir_sample.h      ・・・IIR Filter Sample Program Header
│        └─touch_filter_median                 ・・・Median Filter Sample Module Storage Folder
│                 └─filter_sample
│                         filter_config_sample.c ・・・Median Filter Configuration Definition Sample Source
│                         filter_config_sample.h ・・・Median Filter Configuration Definition Sample Header
│                         median_config_sample1.c・・・Median Filter Sample Preset 1 Source
│                         median_config_sample2.c・・・Median Filter Sample Preset 2 Source
│                         r_ctsu_filter_sample.c・・・Filter API Sample Program Source
│                         r_ctsu_filter_sample.h・・・Filter API Sample Program Header
│                         r_ctsu_median_sample.c・・・Median Filter Sample Program Source
│                         r_ctsu_median_sample.h・・・Median Filter Sample Program Header
└─ra2l1_rssk_filter_sample・・・Example Project Implementing Filter Sample
```

## 2.3 Data List for Filter Configuration Definition

This section describes the constants, global variables, and structures that are provided in the software filter sample program for defining the filter configuration.

### 2.3.1 Constants

Table 2.2 lists the constants.

**Table 2.2 Constants for Filter Configuration Definitions**

| Constant name | Value | Description |
|---|---|---|
| **File name: filter_config_sample.h** | | |
| CTSU_FILTER_NUM | 1 to 3 | Number of filters connected in series<br>(Value varies according to filter type and preset definition.) |
| FILTER_ELEMENT_SIZE | CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS × 2) | Number of measurement results obtained using the CTSU API (Calculated from the touch interface configuration definition.) |
| File name: r_ctsu_filter_sample.c | | |
| CTSU_IF_MAX | 8 | Maximum number of definitions in touch interface configuration |
| FILTER_SIZE | CTSU_FILTER_NUM × CTSU_IF_MAX | Number for filter management data |
| FILTER_RESULT_MIN | 0 | Minimum value of filtered result |
| FILTER_RESULT_MAX | 65535 | Maximum value of filtered result |
| CTSU_FILTER_OPEN | 0x464C5452 | Initialized management definition value |
| File name: r_ctsu_fir_sample.h | | |
| FIR_FILTER_ENABLE | 0 to 1 | FIR filter enable/disable definition (0 = disable, 1 = enable) |
| File name: r_ctsu_iir_sample.h | | |
| IIR_FILTER_ENABLE | 0 to 1 | IIR filter enable/disable definition (0 = disable, 1 = enable) |
| File name: r_ctsu_median_sample.h | | |
| MEDIAN_FILTER_ENABLE | 0 to 1 | Median filter enable/disable definition (0 = disable, 1 = enable) |

### 2.3.2 Enumerations

Table 2.3 lists the enumerations for filter_type_t.

**Table 2.3 filter_type_t**

| Member | Value | Description |
|---|---|---|
| FILTER_TYPE_NONE | 0 | Filter type: no filter |
| FILTER_TYPE_FIR | 1 | Filter type: FIR filter |
| FILTER_TYPE_IIR | 2 | Filter type: IIR filter |
| FILTER_TYPE_MEDIAN | 3 | Filter type: Median filter |

### 2.3.3 Global Variables

Table 2.4 lists the global variables.

**Table 2.4 Global Variables for Filter Configuration Definitions**

| Data | Data type | Description |
|---|---|---|
| **File name: r_ctsu_filter_sample.c** | | |
| g_ctsu_filter_element_index | uint16_t | Index for assigning management data |
| g_ctsu_filter_element_control | filter_element_ctrl_t | Management data for filters (Defines the data size as the total<br>number of filters used x number of measurement results.) |

### 2.3.4 Structures

This section describes the structures used for API access in qe_touch_sample.c and the number and types of filters defined in filter_config_sample.c

**Table 2.5 Filter Structure Definitions**

| Definition content | Data type | Remarks |
|---|---|---|
| Filter name: qe_touch_sample.c | | |
| Definition for touch module and filter module access | filtering_instance_t | Definition of r_rssk_filter_initialize() and r_rssk_filter_dataget() |
| File name: filter_config_sample.c | | |
| Filter management definition | ctsu_filter_instance_t | Prepare for each method of touch interface configuration. |
| Filter management data | filter_instance_ctrl_t | |
| Filter configuration definition | filter_config_t | To change the filter contents to be used for each method of touch interface configuration, prepare a definition for each filter content. |
| Filter content definition | filter_element_config_t | |

#### 2.3.4.1 Definition for touch module and filter module access (filtering_instance_t)

**Table 2.6 Structures for Defining Touch Module and Filter Module Access (filtering_instance_t)**

| Member | Data type | Description |
|---|---|---|
| p_touch_instance | touch_instance_t const * | Touch module management definition pointer |
| p_filter_instance | ctsu_filter_instance_t const * | Filter management definition pointer |

- Example description of structures for defining touch module and filter module access (filtering_instance_t)

```
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 Filter management definition (ctsu_filter_instance_t)

**Table 2.7 Structures for Defining Filter Management (ctsu_filter_instance_t)**

| Member | Data type | Description |
|---|---|---|
| p_ctrl | filter_ctrl_t * | Filter management data pointer<br>(This data pointer is defined as the void type pointer.)<br>Filter management pointer<br>(Use this to assign the variables defined in filter_instance_ctrl_t) |
| p_cfg | filter_config_t const * | Filter configuration definition pointer |
| p_api | filter_api_t const * | Filtering API pointer |

- Description example of filter management definition (ctsu_filter_instance_t)

```
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,
};
```

Define the management data and filter configuration to be used for each method of touch interface configuration.

#### 2.3.4.3 Filter management data (filter_instance_ctrl_t)

**Table 2.8 Structures for Filter Management Data (filter_instance_ctrl_t)**

| Member | Data type | Description |
|---|---|---|
| open | uint32_t | Initialized state |
| p_cfg | filter_config_t const * | Filter configuration definition pointer |
| p_element_ctrl | filter_element_ctrl_t * | Filter individual management pointer |

#### 2.3.4.4 Filter individual management data (filter_element_ctrl_t)

**Table 2.9 Structures for Filter Individual Management Data (filter_element_ctrl_t)**

| Member | Data type | Description |
|---|---|---|
| element_num | uint16_t | Number of measurement results |
| p_filter_ctrl | filter_ctrl_t * | Management data pointer for each filter |

#### 2.3.4.5 Filter configuration definition (filter_config_t)

**Table 2.10 Structures for Defining Filter Configuration (filter_config_t)**

| Member | Data type | Description |
|---|---|---|
| filter_num | uint8_t | Number of filters connected in series |
| p_filter_cfg | filter_element_config_t | Filter content definition pointer |

- Description example of filter configuration definition (filter_config_t)
  Defines the number and type of filters for each filter pattern to be applied.

```
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 Filter content definition (filter_element_config_t)

**Table 2.11 Structures for Defining Filter Content (filter_element_config_t)**

| Member | Data type | Description |
|---|---|---|
| type | filter_type_t | Filter type |
| filter_element_cfg | filter_ctrl_t * | Configuration definition pointer for each filter |

- Description example of filter content definition (filter_element_config_t)
-

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

Define the filter types to be used in the order in which they will be applied.

## 2.4 Software Filter APIs

Table 2.12 shows the software filter APIs implemented in this sample program.

**Table 2.12 Filter Initialization APIs**

| Function name | Process description |
|---|---|
| File name: qe_touch_sample.c | |
| qe_touch_main | main processing |
| r_rssk_filter_initialize | Touch module and filter initialization |
| r_rssk_filter_dataget | Acquisition of filtered touch result |
| timer0_callback | AGT interrupt callback |
| r_rssk_initialize | Initialization of CTSU LEDs |
| r_rssk_led_test | Test processing for CTSU LEDs |
| File name: r_ctsu_filter_sample.c | |
| r_ctsu_filter_open | Filter initialization |
| ctsu_fir_filter_open | FIR filter initialization |
| ctsu_iir_filter_open | IIR filter initialization |
| ctsu_median_filter_open | Median filter initialization |
| r_ctsu_filter_exec | Filter execution |

### 2.4.1 r_rssk_filter_initialize

This function initializes the touch module and software filter. Make sure to implement this function before using any other touch module or software filter API functions. This function must be implemented for each touch interface.

Format

fsp_err_t r_rssk_filter_initialize (filtering_instance_t * const p_ctrl);

Parameters

p_ctrl

Dynamic software and filter management definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_ALREADY_OPEN | /* Initialization completed. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in qe_touch_sample.c.

Description

This function calls RM_TOUCH_Open() and r_ctsu_filter_open() to initialize the touch module and software filter.

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:

This function is intended to be used in place of the RM_TOUCH_Open() call in the QE generated code.

### 2.4.2 r_rssk_filter_dataget

This function applies a software filter to the touch measurement result and acquires the filtered touch state.

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

Touch middleware and filter management definition pointer

p_button_status

Button status storage buffer pointer

p_sliderbutton_status

Slider position storage buffer pointer

p_button_status

Wheel position storage buffer pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_NOT_OPEN | /* Called without calling Open API. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in qe_touch_sample.c.

Description

This function calls R_CTSU_DataGet(), r_ctsu_filter_exec(), and R_CTSU_DataInsert() to apply the software filter on the measured value. If the filter is successfully applied, the function calls RM_TOUCH_DataGet() to determine touch and detect position.

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:

This function is intended to be used in place of the RM_TOUCH_DataGet() call in the QE generated code.

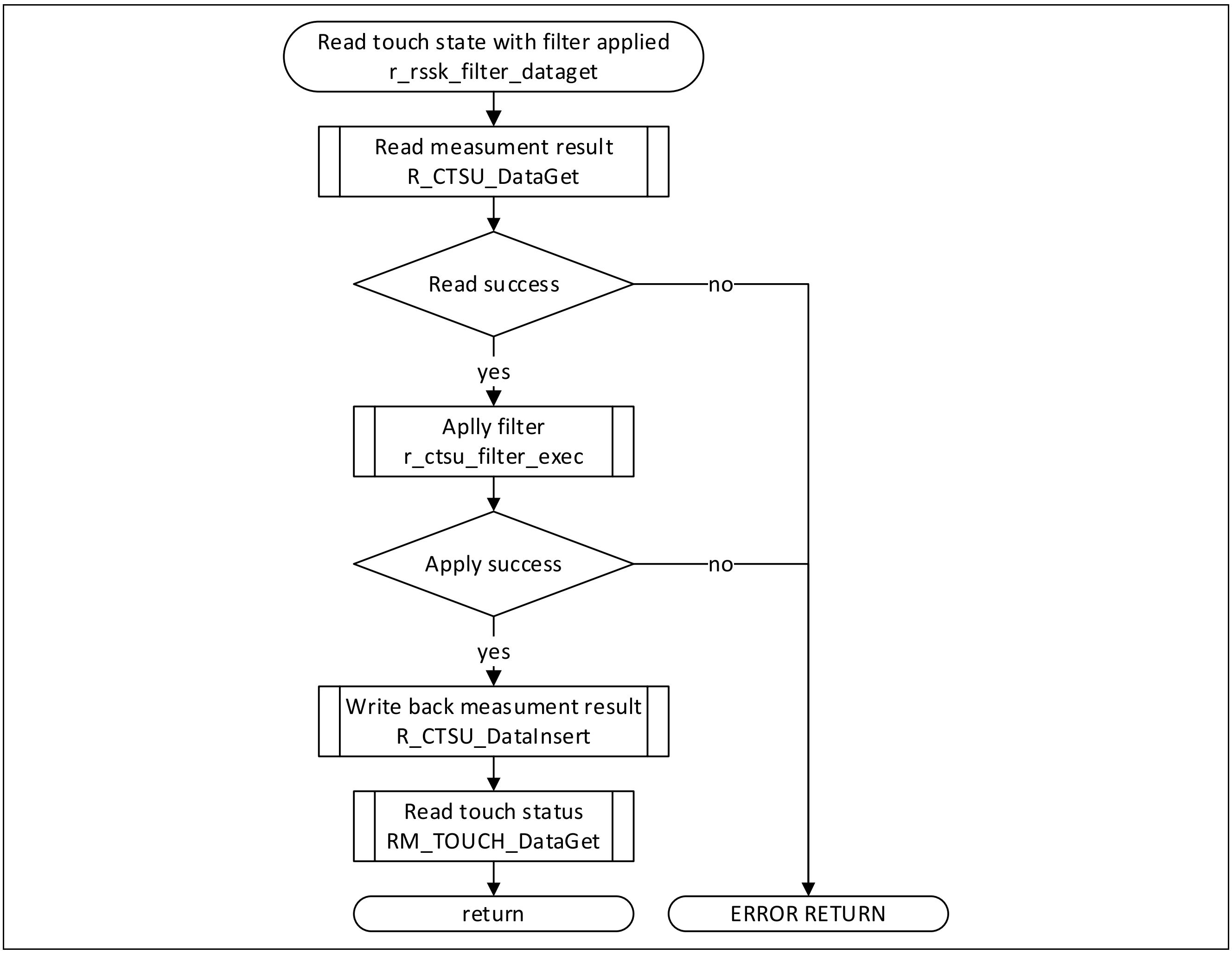



**Figure 2-3 Filtered Touch Result Acquisition API Flowchart**

### 2.4.3 r_ctsu_filter_open

This function initializes the software filter. Make sure you implement this function before using any other API functions. You will need to prepare filter management data and configuration definitions for the number of touch interfaces (methods) to be used and implement them function for each touch interface.

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

Filter management data pointer

p_cfg

Software filter configuration definition pointer

p_ctsu_cfg

CTSU driver configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_ALREADY_OPEN | /* Initialization completed. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_filter_sample.h.

Description

This function initializes the filter management data according to arguments p_cfg and p_ctsu_cfg.

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:

This function references the configuration definition of the CTSU driver to determine how many times it must call the filter initialization API for each filter module. For the self-capacitance measurement mode, the number of calls equals the number of pins. For the mutual capacitance measurement mode, the number calls is "the number of transmitting pins x the number of receiving pins x 2". Refer to the API descriptions below for more details.

- FIR filter initialization API: r_ctsu_fir_open
- IIR filter initialization API: r_ctsu_iir_open
- Median filter initialization API： r_ctsu_median_open

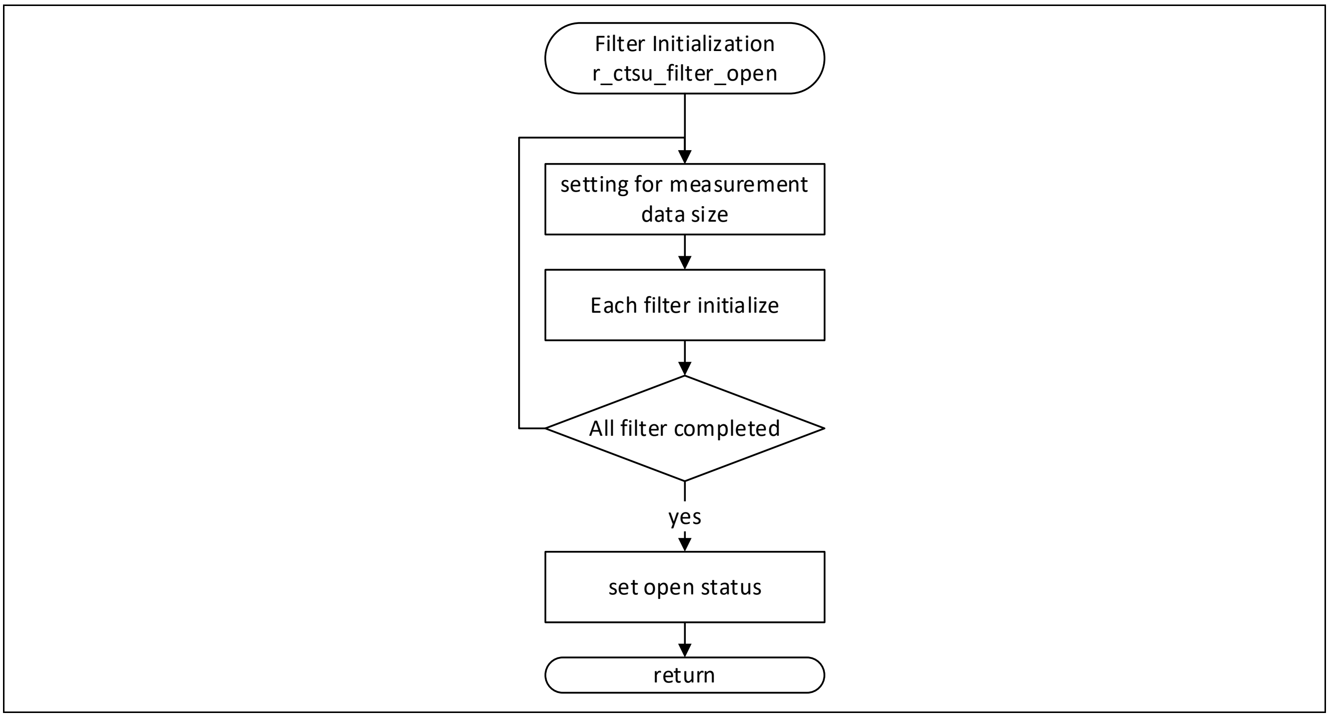



**Figure 2-4 Filter Initialization API Flowchart**

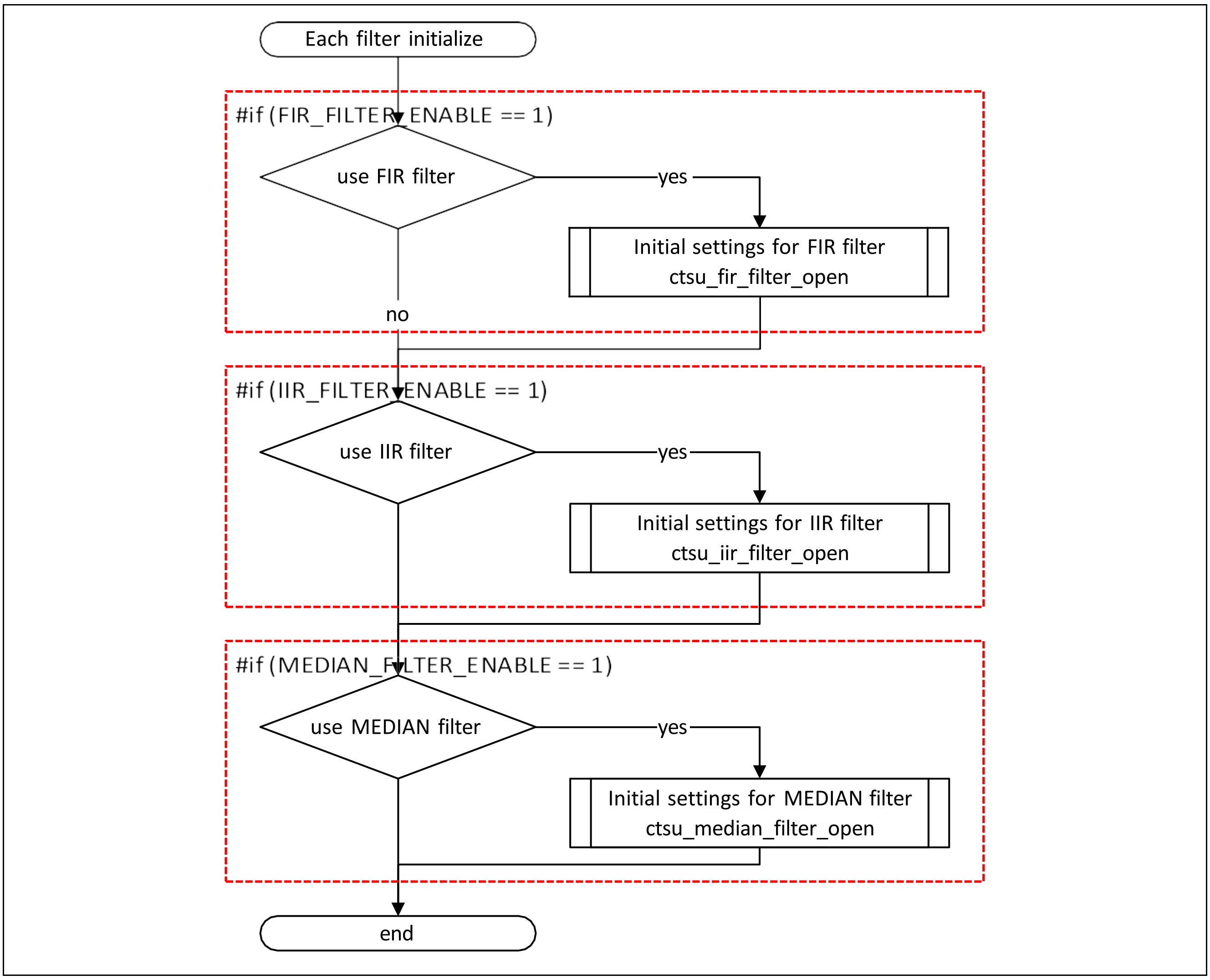



**Figure 2-5 Filter Initialization API: Filter Initialization Processing Flowchart**

### 2.4.4 ctsu_fir_filter_open

This function is called from r_ctsu_filter_open() when using the FIR filter. This function allocates management data for the FIR filter and calls r_ctsu_fir_open().

Format

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

Parameters

p_ctrl

Filter individual management data pointer

p_cfg

FIR filter configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_filter_sample.c.

Description

This function assigns and initializes the FIR filter management data according to arguments p_ctrl and p_cfg.

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:

This function references the number of measurement results initially set in r_ctsu_filter_open(), and calls the filter initialization API for the FIR filter module. Refer to the API description below for more details.

- FIR filter initialization API: r_ctsu_fir_open

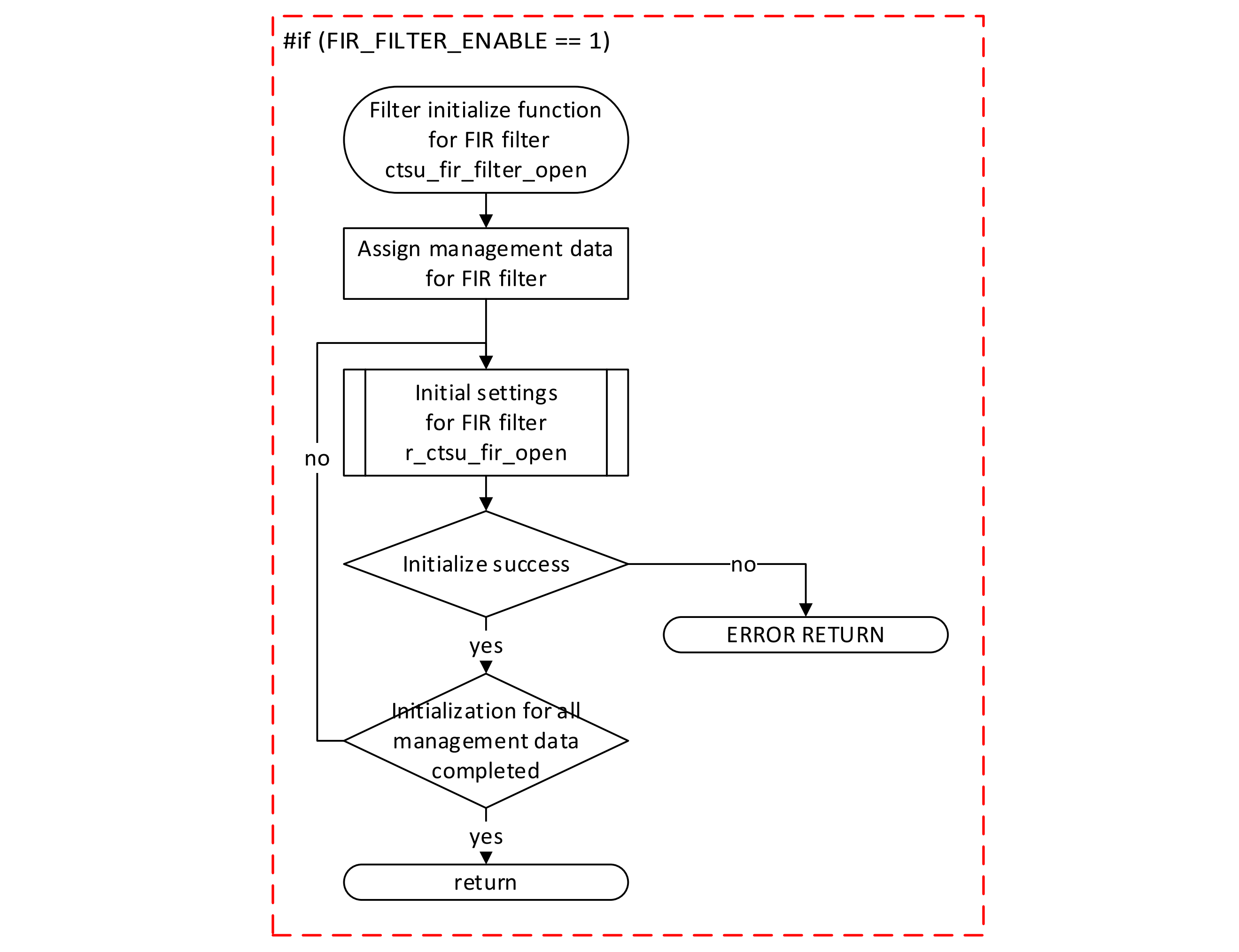



**Figure 2-6 FIR Filter Initialization API Flowchart**

### 2.4.5 ctsu_iir_filter_open

This function is called from r_ctsu_filter_open() when using the IIR filter. This function allocates management data for the IIR filter and calls r_ctsu_iir_open().

Format

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

Parameters

p_ctrl

Filter individual management data pointer

p_cfg

IIR filter configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_filter_sample.c.

Description

This function assigns and initializes the IIR filter management data according to arguments p_ctrl and p_cfg.

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:

This function references the number of measurement results initially set in r_ctsu_filter_open(), and calls the filter initialization API for the IIR filter module. Refer to the API description below for more details.

- IIR filter initialization API: r_ctsu_iir_open

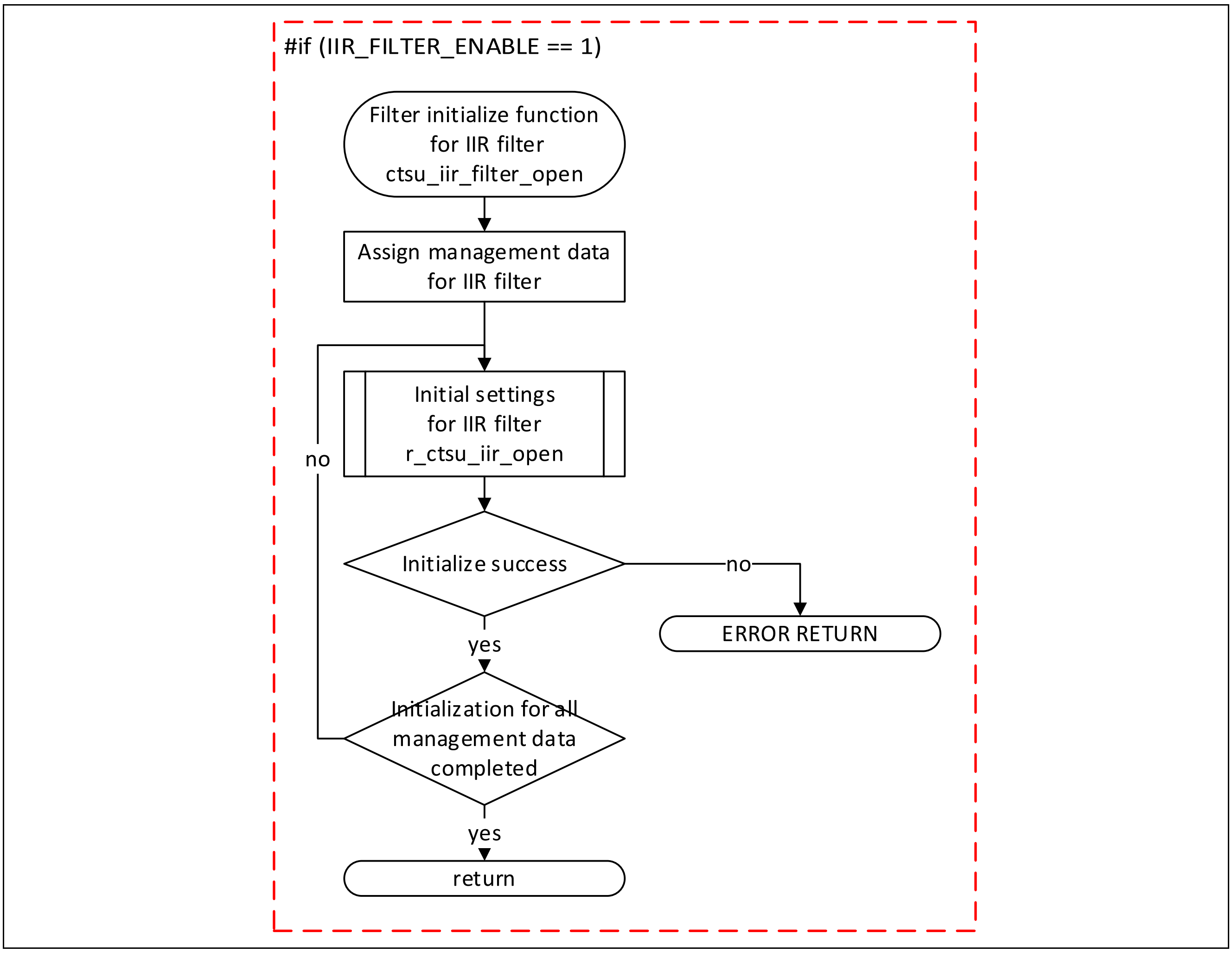



**Figure 2-7 IIR Filter Initialization API Flowchart**

### 2.4.6 ctsu_median_filter_open

This function is called from r_ctsu_filter_open() when using a median filter. The function assigns the management data for the median filter and calls r_ctsu_median_open()

Format

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

Parameters

p_ctrl

Filter management data pointer

p_cfg

Median filter configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Prototype is declared in r_ctsu_filter_sample.c.

Description

This function assigns and initializes the median filter management data according to arguments p_ctrl and p_cfg.

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:

This function references the number of measurement results initialized in r_ctsu_filter_open() and calls the filter initialization API for the median filter module. Refer to the following API explanations for details.

- Median filter initialization API: r_ctsu_median_open

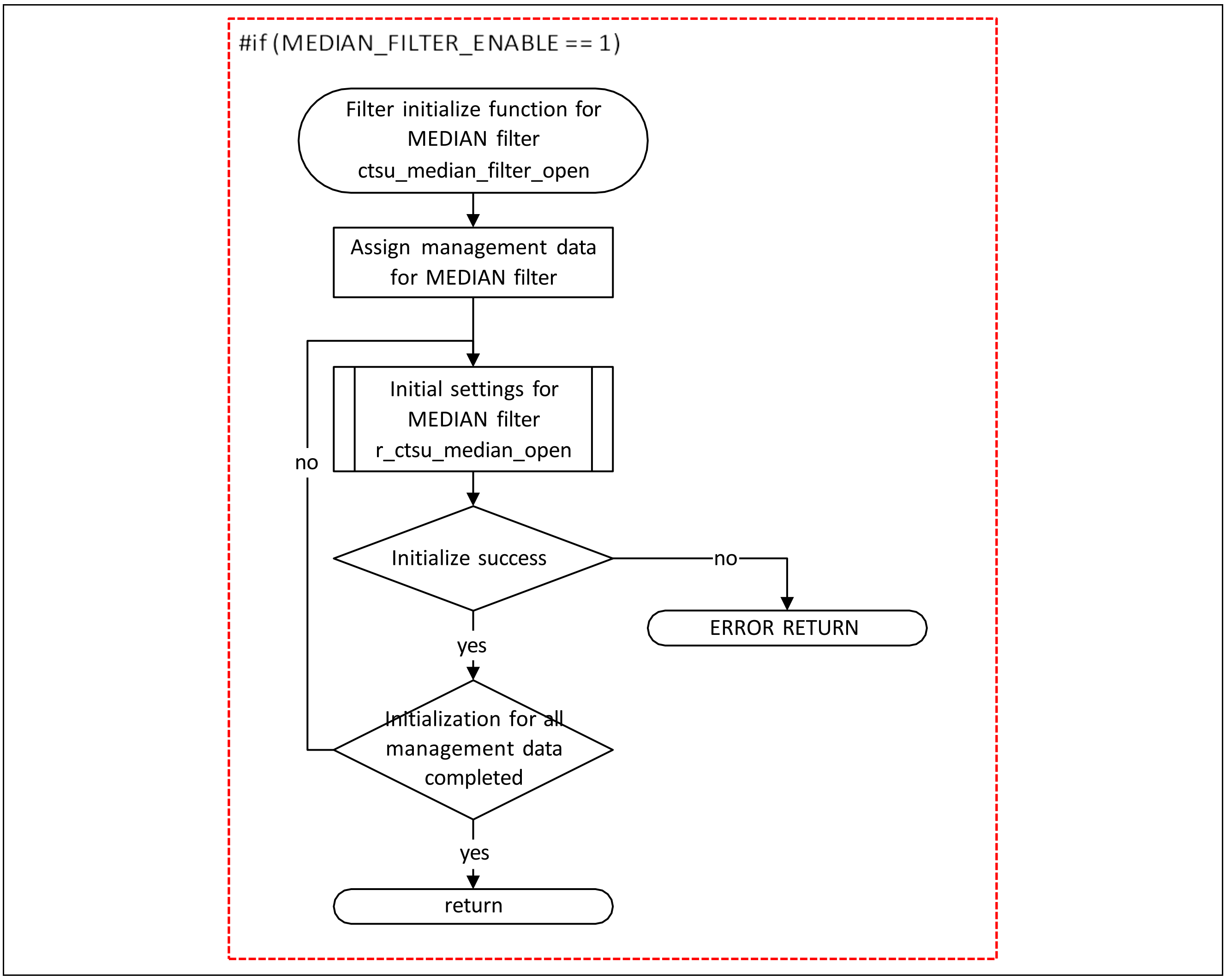



**Figure 2-8 Median Filter Initialization API Flowchart**

### 2.4.7 r_ctsu_filter_exec

This function applies the software filter to the measurement result data.

Format

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

Parameters

p_ctrl

Filter management data pointer

p_data

Pointer to input/output data buffer. Applies a filter to the data in the buffer specified by this pointer and overwrites and stores the result after filtering.

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_NOT_OPEN | /* Executed without calling Open().*/ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters not applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_filter_sample.h.

Description

This function is used in combination with R_CTSU_DataGet() and R_CTSU_DataInsert() to apply the filters defined by the filter configuration on the measured touch data.

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:

This function calls the filter execution API of each filter module. Refer to the API descriptions below for more details.

- FIR filter initialization API: r_ctsu_fir_filter
- IIR filter initialization API: r_ctsu_iir_filter
- Median filter execution API: r_ctsu_median_filter

This function applies the filter to overwrite the measurement value data specified in the argument.

To use the measurement result data for other purposes before filtering, make sure you store the unfiltered data before executing the API.

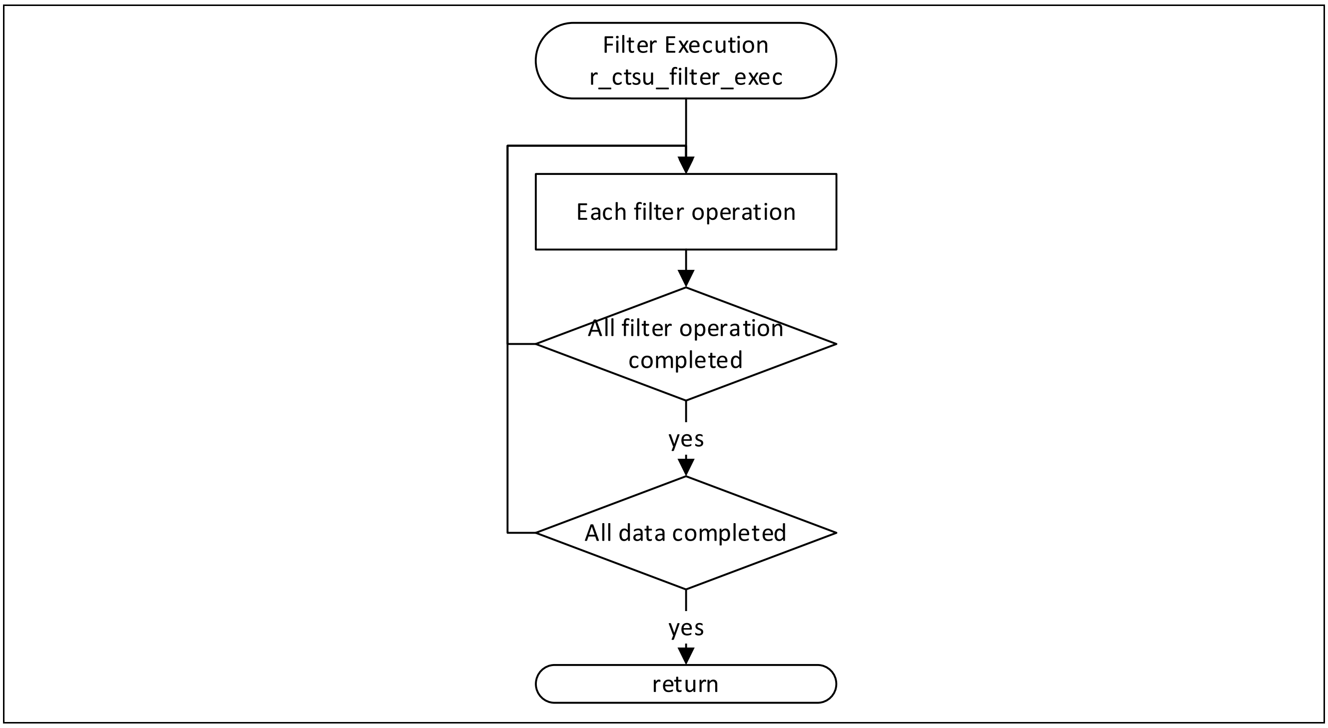



**Figure 2-9 Filter Execution API Flowchart**

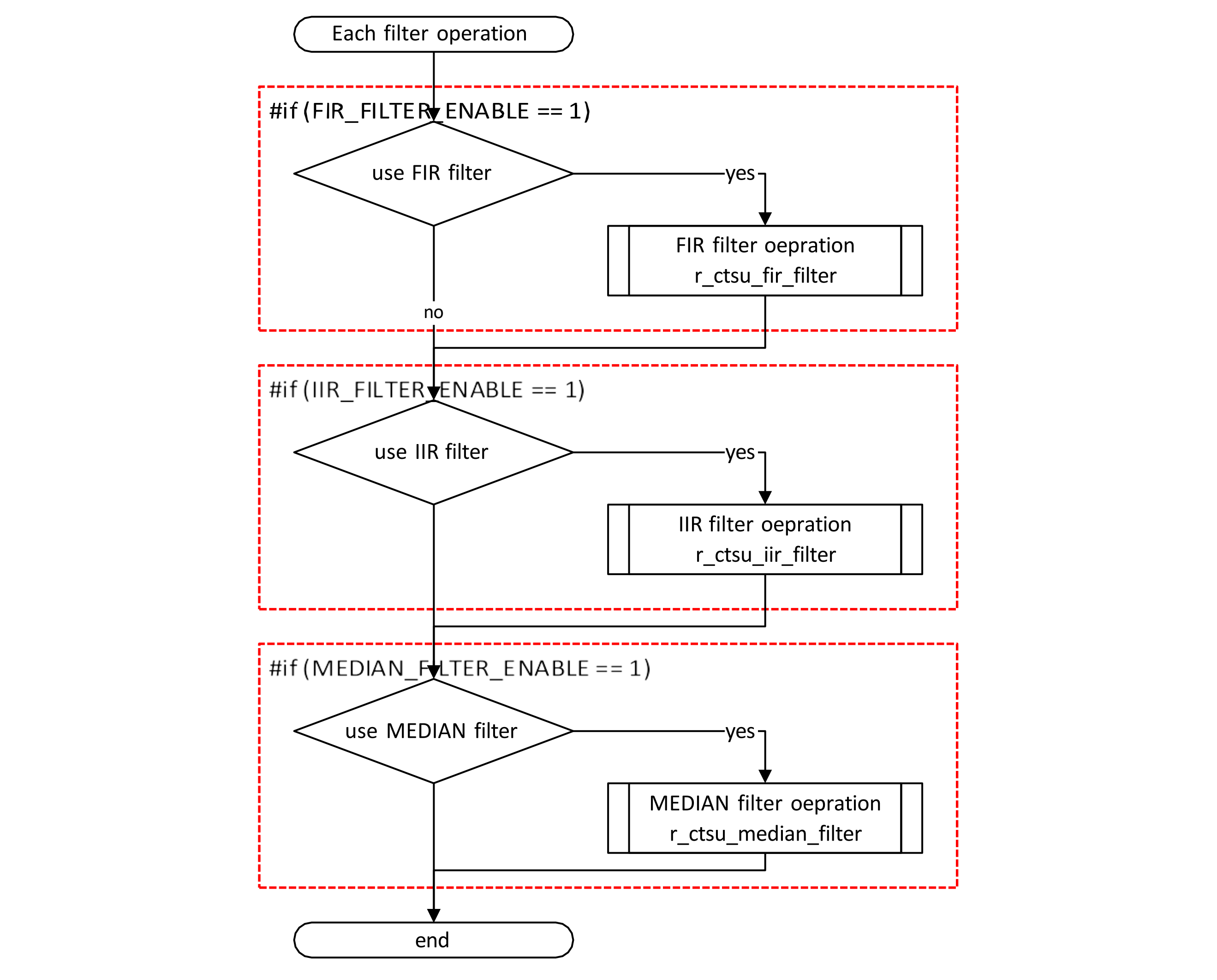



**Figure 2-10 Filter Execution API: Filter Execution Processing Flowchart**

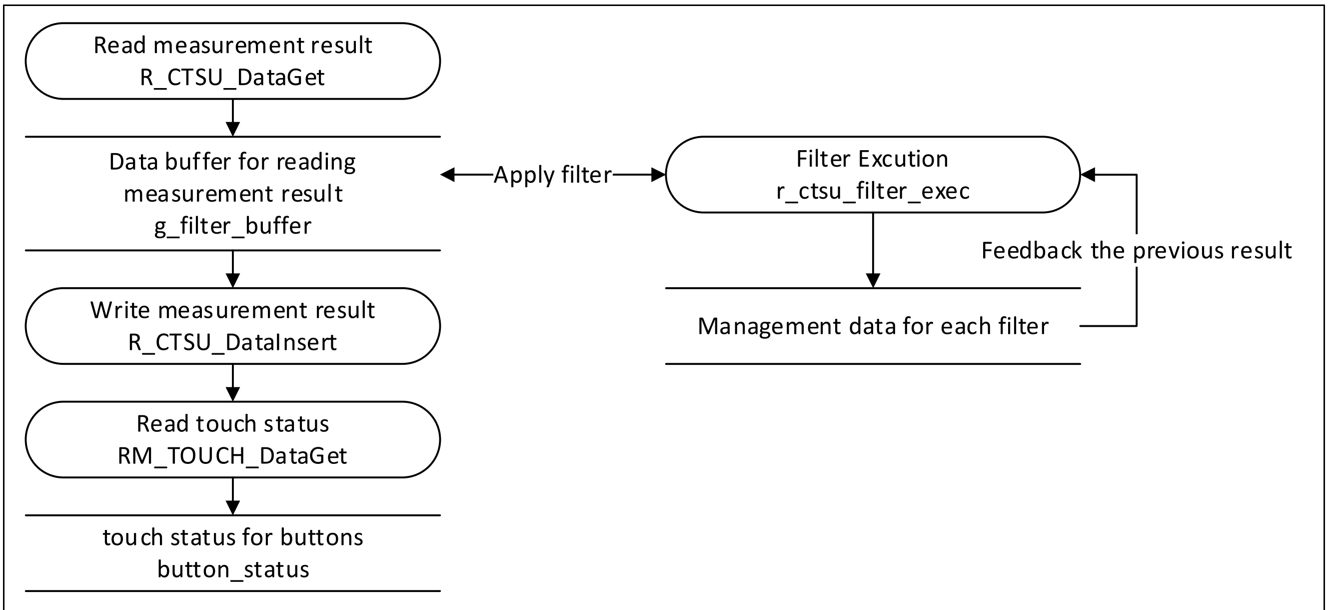



**Figure 2-11 Software Filter Data Flowchart**

## 2.5 Size and Execution Time

Table 2.13 and Table 2.14 show the data sizes and execution times of filtering for the sample program (three touch interface configurations: Button × 3, Slider × 1, Wheel × 1, and with shielded pins).

**Table 2.13 Filter Processing Data Size and Differences**

| Conditions | Size [Bytes] | | |
|---|---|---|---|
| | text | data | bss |
| Before adding filters | 25008 | 24 | 3360 |
| Filter management | +220 | +24 | +104 |
| FIR filters (Direct type) (Note) | +820 | +4 | +580 |
| FIR filters (Transpose type) (Note) | +788 | +4 | +628 |
| IIR filters (Note) | +948 | +4 | +388 |
| Median filters (MEDIAN_PRESET_TYPE_2)( Note) | +712 | +4 | +612 |

Note: This varies depending on the order of filters.
The values shown reflect values when the maximum order is defined.

**Table 2.14 Filter Processing Execution Time**

| Conditions | Execution time (1ch) |
|---|---|
| FIR filters (Direct type) (FIR_PRESET_TYPE_4) | 13.84us |
| FIR filters (Transpose type) (FIR_PRESET_TYPE_4) | 8.69us |
| IIR filters (IIR PRESET TYPE 4) | 16.51us |
| Median filters (MEDIAN_PRESET_TYPE_2) | 8.69us (Note) |

Note: The execution time shown reflects the self-capacitance method. In the mutual-capacitance method, the execution time is approximately doubled because two measurements are taken per measurement.

Note: The time noted is the average execution time. The median filter execution time can vary by 5 times the average.

# 3. FIR Filters

FIR (Finite Impulse Response) filters are regularly used to reduce random and periodic noise.

For more information, refer to "Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426).

## 3.1 Specifications

The calculation formulas for FIR filters are shown below.

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

$n$ indicates the sample index, $h$ (m) indicates the coefficient, $x(n - \mathrm{m})$ indicates the input data of the m sample delay, and $y(n)$ indicates the output data.

Table 3.1 shows the specifications of the sample program's FIR filters.

**Table 3.1 FIR Filters Specifications**

| Item | Specifications | Remarks |
|---|---|---|
| Input data type | Unsigned 32-bit integer type | |
| Output data type | Unsigned 32-bit integer type | |
| Coefficient data type | Signed 15-bit fixed point | Internal operations are signed 32-bit decimal<br>(Integer part 17-bit, decimal part 14-bit) |
| Maximum coefficient | 8 | The number of taps is indicated by "order + 1" |
| Filter processing method | • Direct type<br>• Transpose type | Can be switched by conditional compilation (Refer to chapter 3.5.1) |
| Output results up to filter stabilization time | Output Zero<br>Returns operation results during stabilization time and buffer unfilled response | Filter stabilization time is number of taps (order + 1) x number of samples |

Note: Coefficient: A set of constants to be applied to the constant multipliers that make up FIR filters.
Order: Number of elements in the coefficient.
Number of taps: Number of orders including zero order. (Indicates the order + 1 value)

## 3.2 How to Use the Filter in This Sample Program

This sample program allows you to specify filtering methods and filter characteristics by conditional compilation.

Table 3.2 shows how to specify FIR filtering.

Direct type processing uses a smaller data size, and transpose type processing requires a shorter processing time.

For details on the data size and processing time, see Table 2.13 and Table 2.13.

**Table 3.2 Sample FIR Filtering Specification**

| File | Definition name | Description |
|---|---|---|
| r_ctsu_fir_sample.h | FIR_FILTER_TYPE | Filter processing method<br>FIR_FILTER_TYPE_DIRECT = Direct type<br>FIR_FILTER_TYPE_TRANSPOSE = Transpose Type |

## 3.3 FIR Filter API

Table 3.3 shows the FIR filter API implemented by this sample program.

**Table 3.3 FIR Filter API**

| Function name | Process description |
|---|---|
| File name: r_ctsu_fir_sample.c | |
| r_ctsu_fir_open | FIR filter initialization processing |
| r_ctsu_fir_filter | FIR filter execution processing |
| r_ctsu_fir_direct_filter | Direct-type FIR filter processing |
| r_ctsu_fir_transpose_filter | Transpose-type FIR filter processing |

### 3.3.1 r_ctsu_fir_open

This function assigns and initializes the buffer for FIR filter processing. Make sure you execute this function before using any other API.

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 filter management data pointer

p_cfg

FIR filter configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_fir_sample.h.

Description

This function assigns and initializes the FIR filter processing buffer for one measurement result.

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:

Before executing this function, it is necessary to set a pointer to the FIR filter management data by referring to the position pointer at the time the FIR filter management data is assigned.

This function must be executed the number of times the measurement result data is read by the CTSU driver for each touch interface. (For self-capacitance method, this is the number of pins; for mutual capacitance method, this is "the number of transmitting pins x the number of receiving pins x 2.”

Refer to the filter initialization API (r_ctsu_filter_open) description for more details.

### 3.3.2 r_ctsu_fir_filter

This function applies the FIR filter processing on one measurement result.

Format

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

Parameters

p_ctrl

FIR filter management data pointer

p_data

FIR filter measurement result data pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_fir_sample.h.

Description

This function applies the FIR filter processing on one measurement result.

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:

The processing executed by this function varies according to the conditional compilation (FIR_FILTER_TYPE).

Also refer to the direct-type FIR filter execution API (r_ctsu_fir_direct_filter) and the transpose-type FIR filter execution API (r_ctsu_fir_transport_filter).

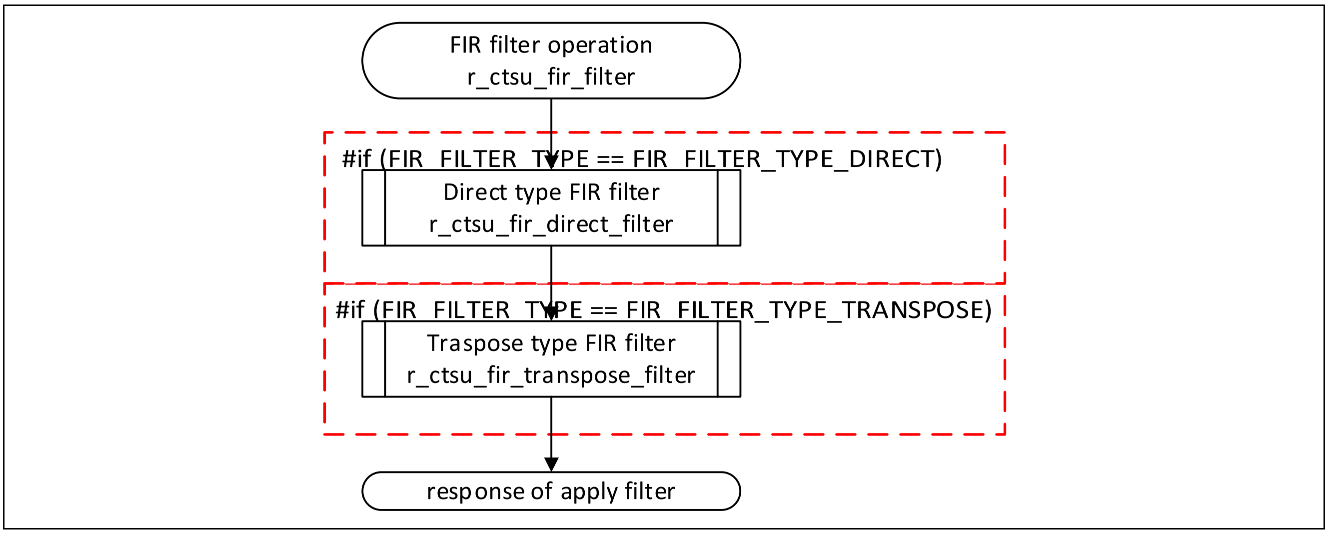



**Figure 3-1 FIR Filter Execution API Flowchart**

### 3.3.3 r_ctsu_fir_direct_filter

This function applies the direct-type FIR filter processing on one measurement result.

Format

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

Parameters

p_ctrl

FIR filter management data pointer

p_data

FIR filter measurement result data pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_fir_sample.c.

Description

This function is called from the FIR filter execution process (r_ctsu_fir_filter) when the conditional compilation FIR_FILTER_TYPE = FIR_FILTER_TYPE_DIRECT.

When data in the signed 18-bit integer range (131071 to -131072) or higher is passed as measurement value data, the operation is performed as if the upper or lower limit value was entered.

The result of the operation is limited to the range of signed 18-bit integers (131071 to -131072); if it exceeds the range, it will be rounded to the upper or lower limit.

Special Notes:

Returns buffer unfilled response during filter stabilization time.

Until the filter stabilization time elapses, the filtered result is the operation result when the unfilled range is in the initialized state (0).

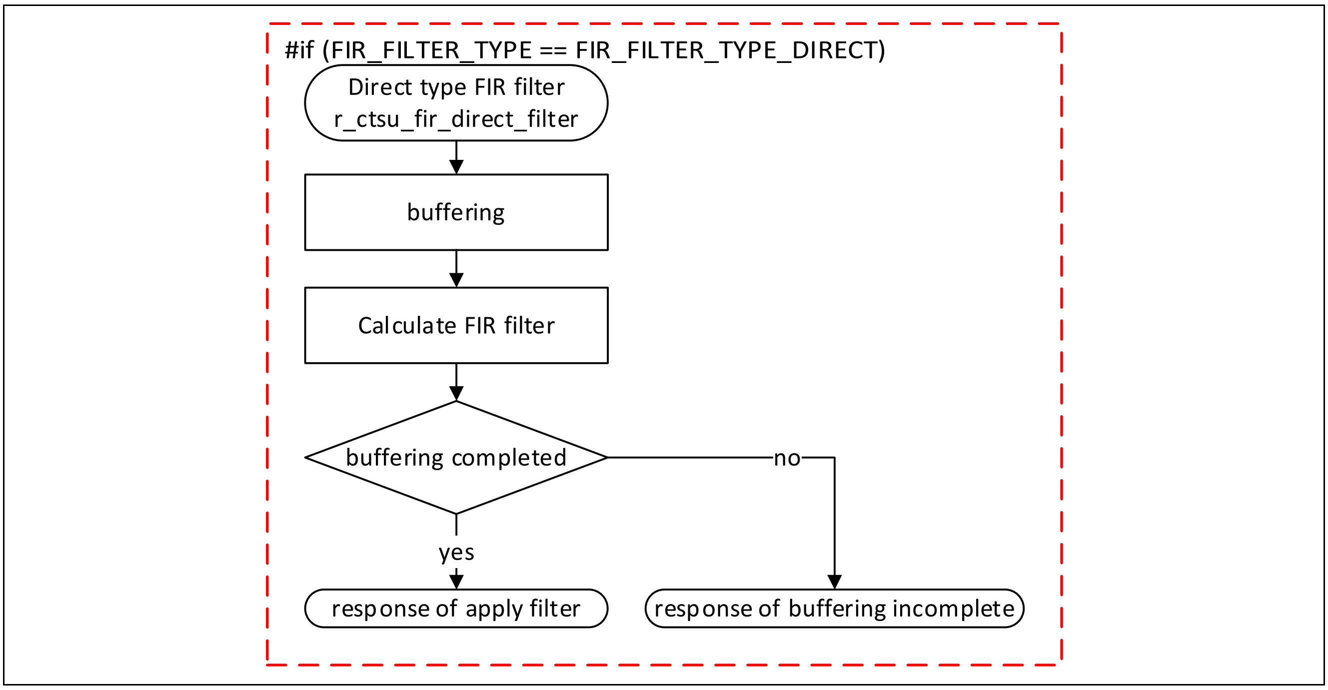



**Figure 3-2 Direct-type FIR Filter API Flowchart**

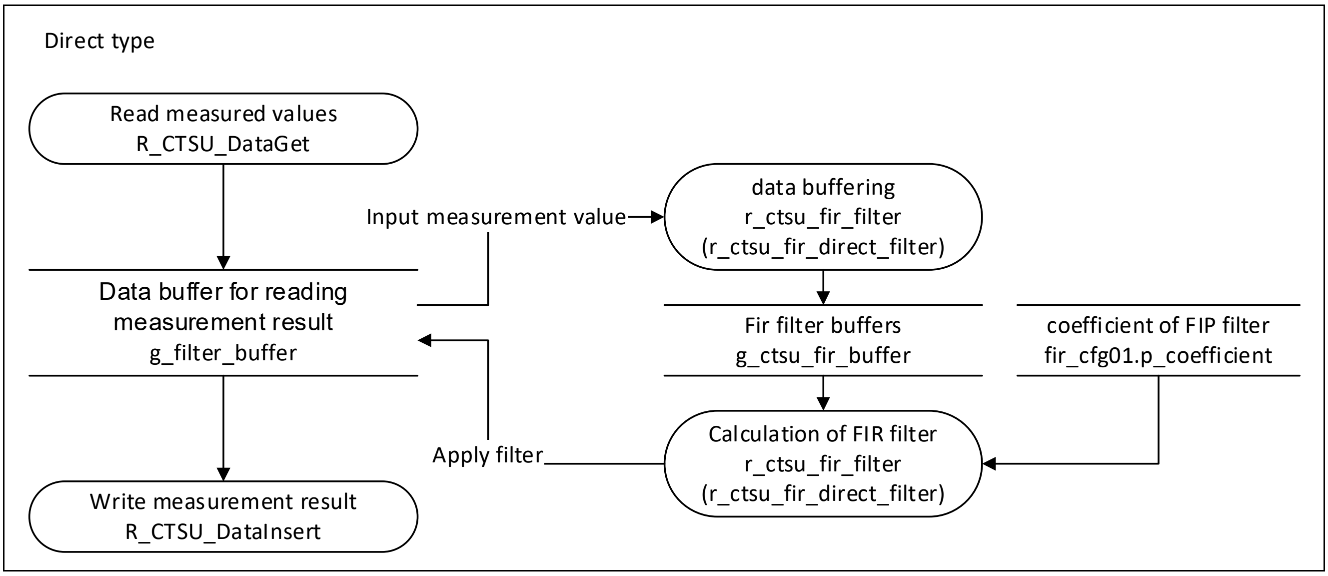



**Figure 3-3 Direct-type FIR Filter Data Flowchart**

### 3.3.4 r_ctsu_fir_transpose_filter

This function applies the transpose filter processing on one measurement result.

Format

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

Parameters

p_ctrl

FIR filter management data pointer

p_data

FIR filter measurement result data pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_fir_sample.c.

Description

This function is called from the FIR filter execution process (r_ctsu_fir_filter) when the conditional compilation FIR_FILTER_TYPE = FIR_FILTER_TYPE_TRANSPOSE.

When data in the signed 18-bit integer range (131071 to -131072) or higher is passed as measurement value data, the operation is performed as if the upper or lower limit value was entered.

The result of the operation is limited to the range of signed 18-bit integers (131071 to -131072); if it exceeds the range, it will be rounded to the upper or lower limit.

Special Notes:

Returns buffer unfilled response during filter stabilization time.

Until the filter stabilization time elapses, the filtered result is the operation result when the unfilled range is in the initialized state (0).

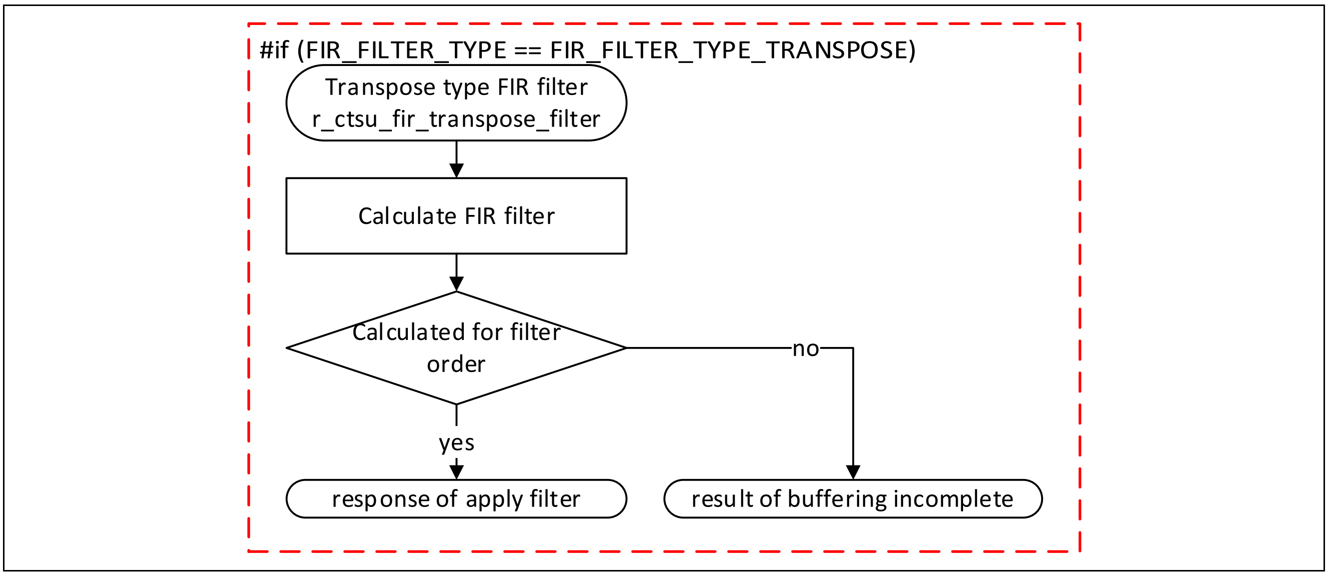



**Figure 3-4 Transpose-type FIR Filter API Flowchart**

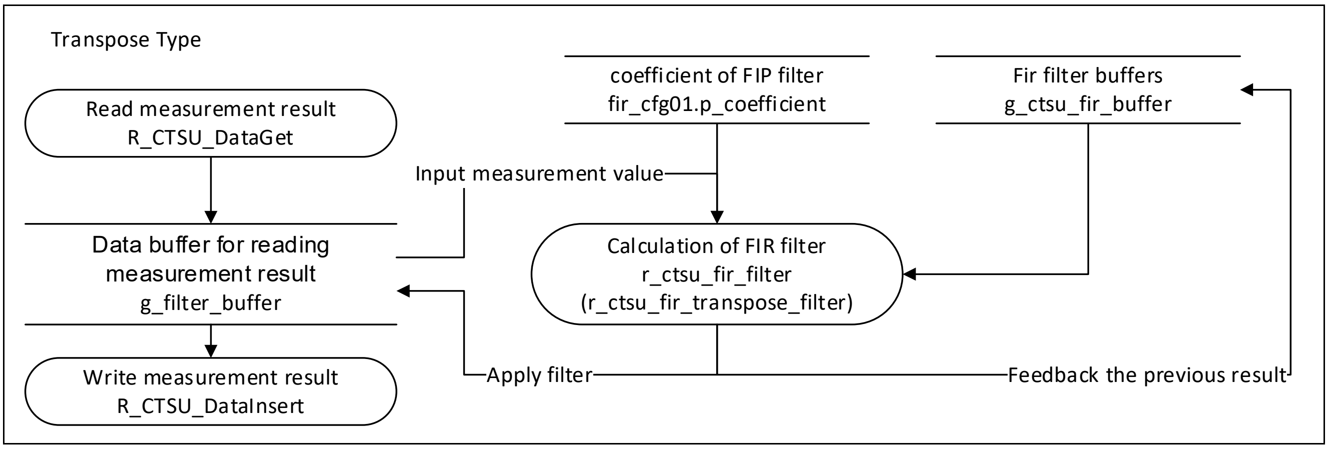



**Figure 3-5 Transpose-type FIR Filter Data Flowchart**

## 3.4 List of Data for FIR Filters

This section explains the constants and global variables provided for FIR filters.

### 3.4.1 Constants

Table 3.4 lists the constants.

**Table 3.4 Constants for FIR Filters**

| Constant name | Setting value | Description |
|---|---|---|
| File name: r_ctsu_fir_sample.h | | |
| FIR_FILTER_NUM | 1 | Number of filter stages |
| File name: r_ctsu_fir_sample.c | | |
| FIR_TAP_SIZE_MIN | 2 | Minimum number of taps |
| FIR_TAP_SIZE_MAX | 9 | Maximum number of taps |
| FIR_CFG_DECIMAL_POINT | 14 | Fixed point number of digits |
| FIR_FILTER_SIZE | FIR_FILTER_NUM x FILTER_ELEMENT_SIZE | Buffer size for FIR filters (Calculated from the number of filter stages and the number of measurement results.) |
| MAX_FIR_COEFFICIENT_SUM | 0x00008000 | Maximum value of the coefficient sum |
| MIN_FIR_COEFFICIENT_SUM | 0xFFFF7FFF | Minimum value of the coefficient sum |
| MAX_FIR_COEFFICIENT_PLUS | 0x3FFF | Maximum value of the coefficient value |
| MIN_FIR_COEFFICIENT_MINUS | 0xC000 | Minimum value of the coefficient value |
| FIR_RESULT_MAX | 0x0001FFFF | Maximum value of the filter result |
| FIR_RESULT_MIN | 0xFFFE0000 | Minimum value of the filter result |

### 3.4.2 Global Variables

Table 3.5 lists the global variables.

**Table 3.5 Global Variables for FIR Filters**

| Variable name | Data type | Description |
|---|---|---|
| File name: r_ctsu_fir_sample.c | | |
| g_ctsu_fir_element_index | uint16_t | Buffer allocation management index |
| g_ctsu_fir_ctrl[FIR_FILTER_SIZE] | fir_ctrl_t | FIR filter management data<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x number of FIR filter stages<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins |
| gp_ctsu_fir_ctrl | fir_ctrl_t * | Position pointer at time of FIR filter management data allocation |
| g_ctsu_fir_buffer[FIR_FILTER_SIZE][FIR_TAP_SIZE_MAX] | int32_t | FIR filter buffer<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x maximum number of taps (9)<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins<br>※For transpose-type: buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x (maximum number of taps (9) + 1) |

## 3.5 Filter Adjustment Procedure

You can change the coefficient definition of FIR filters and adjust the filter properties, as described below.

### 3.5.1 Filter Processing Method

Conditional compilation allows you to specify how FIR filters are handled. Direct-type processing uses a smaller data size, and transpose-type processing requires a shorter processing execution time.

See Table 3.2 for how to set up conditional compilation.

For details on the data size and execution processing time, see Table 2.13 and Table 2.14.

Figure 3-6 shows a block diagram of the FIR filter.

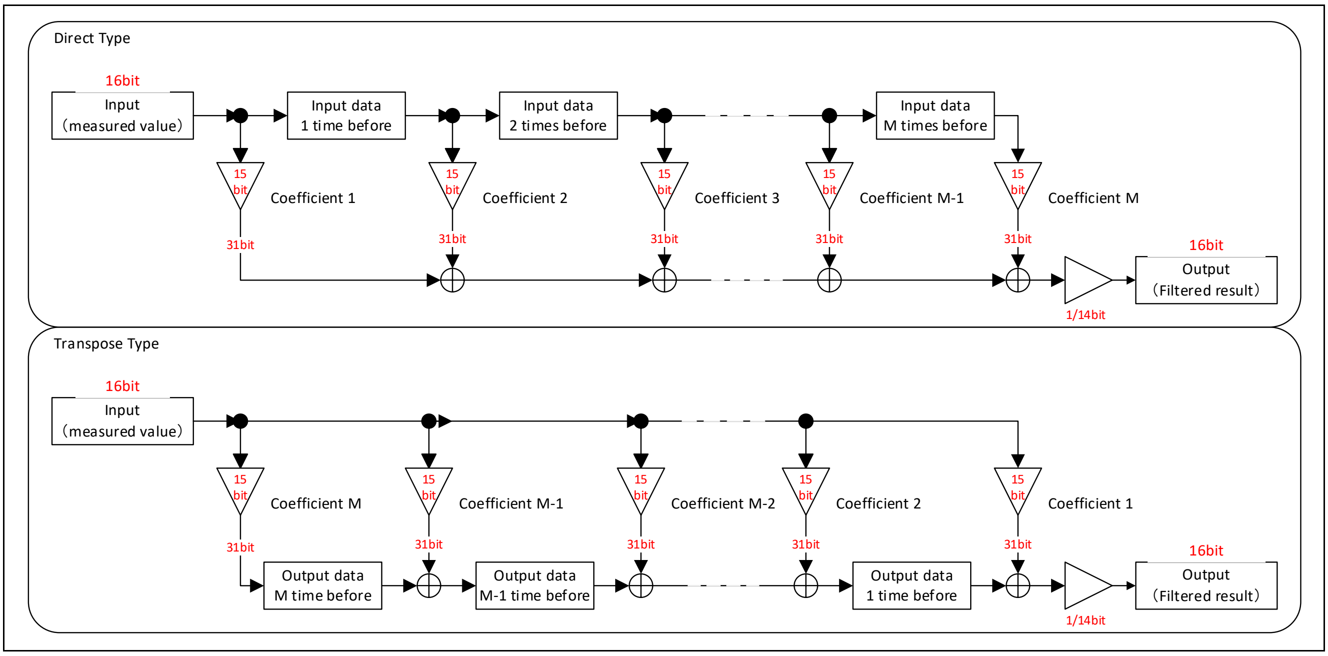



**Figure 3-6 FIR Filter Block Diagram**

### 3.5.2 Filter Characteristics

This sample program can handle filters of up to eight orders.

Table 3.6 defines the characteristics of sample FIR filters. The filter characteristics can be changed by specifying the coefficient and filter configuration definitions shown in Table 3.7.

**Table 3.6 Sample FIR Filters Specification**

| File | Definition name | Description |
|---|---|---|
| filter_config_sample.h | FIR_PRESET_TYPE | Sample preset specification for use with FIR filter |

**Table 3.7 Sample FIR Filters Coefficient Definition**

| | FIR_PRESET_TYPE_1 | FIR_PRESET_TYPE_2 | FIR_PRESET_TYPE_3 | FIR_PRESET_TYPE_4 |
|---|---|---|---|---|
| | FIR moving-average filter | | FIR low-pass filter | |
| **Order** | 2 | 5 | 3 | 8 |
| **Coefficient** | 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 |

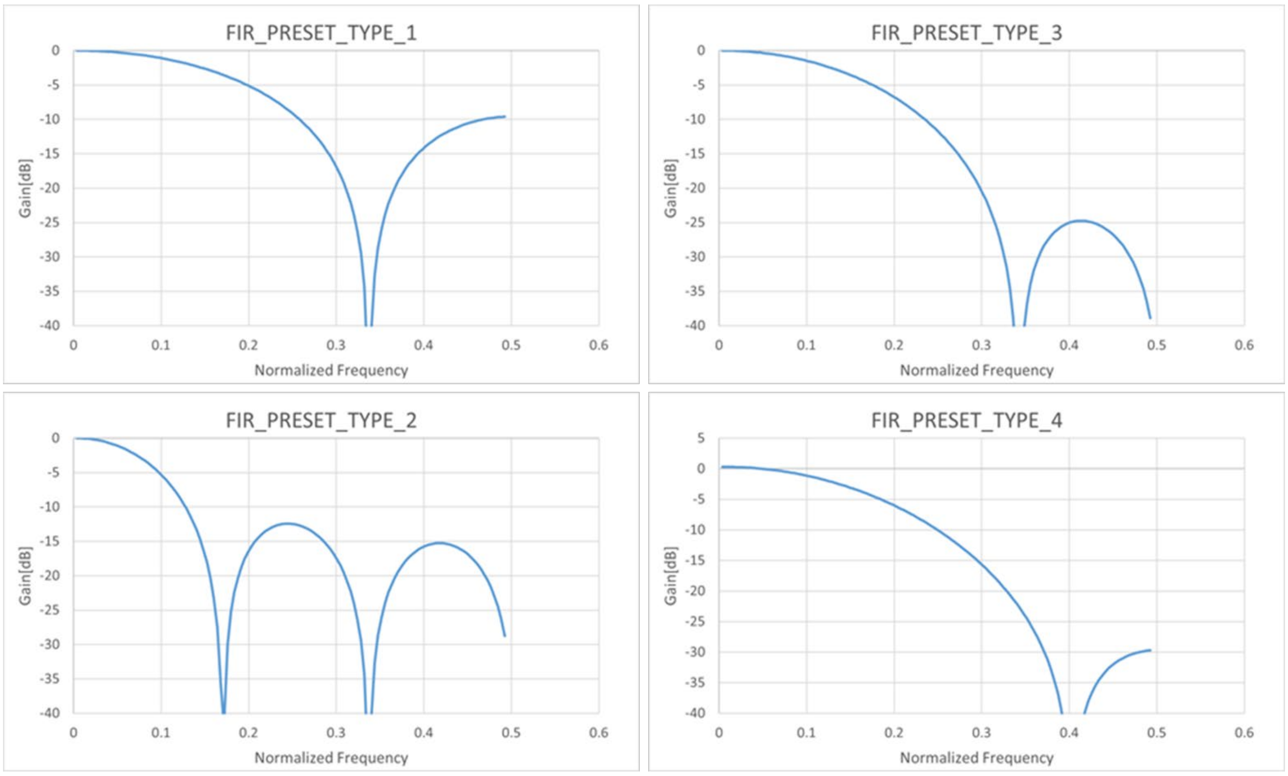



**Figure 3-7 Sample Preset Filter Characteristics**

### 3.5.3 Coefficient Definitions

The coefficients of the FIR filter configuration are defined in the form of signed fixed points (decimal numbers) with no integral part and the lower 14 bits are the decimal part; they are treated as the coefficient value divided by 16384.

The coefficients of the sample program should be designed with a value range of -1.0 to 1.0, and the value obtained by multiplying the fractional coefficient by 16384 (0x4000) should be set as the coefficient definition. Small numbers less than 1LSB cannot be expressed and will cause operation errors.

Table 3.8 shows examples of decimal, hexadecimal, and decimal correspondence.

**Table 3.8 Fixed Point Definition Examples**

| Fractional number | Hexadecimal | Decimal |
|---|---|---|
| -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 Filter Configuration Definition

Define the number of taps/coefficients for FIR filters in the fir_config_t type data table.

The number of taps specifies the order of the FIR filter + 1, and the coefficient table describes the coefficient values of the FIR filter in 15-bit signed fixed point in order from the zero order.

The number of taps is 2 to 9, and the coefficient table can only be defined within the range of -2.0 to 2.0 for the sum of the coefficient definitions.

Note: Define the coefficient table so that the sum of the coefficient definitions approaches 1.0.
If the sum of the coefficient table exceeds 1.0, the measurement result is amplified. If it is less than 1.0, the measurement result is attenuated.

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

Specifies filter order + 1 as the number of FIR filter taps.

Defines the coefficient for the number of taps in FIR filters.

# 4. IIR Filters

IIR (Infinite Impulse response) filters are used regularly to reduce high-frequency components with limited memory and small calculation load.

For more information, refer to Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426).

## 4.1 Specifications

The calculation formulas for IIR filters are shown below.

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

$n$ indicates the sample index, a(m) and a(k) indicate the coefficients, $x(n - \mathrm{k})$ indicates the input data of the k sample delay, $y(n - \mathrm{m})$ indicates the output data of the m sample delay, and $y(n)$ indicates the output data.

Table 4.1 shows the specifications of sample program's IIR filters.

**Table 4.1 IIR Filter Specifications**

| Item | Specifications | Remarks |
|---|---|---|
| Input data type | Signed 32-bit integer | |
| Output data type | Signed 32-bit integer | |
| Coefficient data type | Signed 16-bit fixed point | Internal operations are signed 32-bit (Integer part 19-bit, decimal part 12-bit) |
| Maximum coefficient | 4 | The number of taps is indicated by "order + 1" |
| Filter processing method | Standard type | Can be used as a cascaded IIR filter by connecting several stages of IIR filters. (Refer to section 4.5.4.1 for details.) |
| Output results up to filter stabilization time | Returns operation results during stabilization time and buffer unfilled response. | Filter stabilization time is the number of samples indicated in the configuration definition.<br>(The specified range for stabilization time is from the number of taps to 254.) |

Note: Coefficient: Set of constants to be applied to the constant multipliers that make up IIR filters.
Order: Number of elements in the coefficient.
Number of taps: Number of orders including zero order. (Indicates the order + 1 value)

## 4.2 How to Use the IIR Filter in This Sample Program

This sample program allows you to specify filter characteristics by conditional compilation.

## 4.3 IIR Filter API

Table 4.2 shows a list of IIR filter API included in this sample program.

**Table 4.2 IIR Filter API List**

| Function name | Processing Description |
|---|---|
| File name: r_ctsu_iir_sample.c | |
| r_ctsu_iir_open | IIR filter initialization processing |
| r_ctsu_iir_filter | IIR filter execution processing |

### 4.3.1 r_ctsu_iir_open

This function assigns and initializes the buffer for IIR filter processing. Make sure you execute this function before using any other API.

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 filter management data pointer

p_cfg

IIR filter configuration definition pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_iir_sample.h.

Description

This function assigns and initializes the IIR filter processing buffer for one measurement result.

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:

Before executing this function, it is necessary to set a pointer to the IIR filter management data by referring to the position pointer at the time the IIR filter management data is assigned.

This function must be executed the number of times the measurement result data is read by the CTSU driver for each touch interface. (For self-capacitance method, this is the number of pins, for mutual capacitance method, this is "the number of transmitting pins x the number of receiving pins x 2.")

Refer to the filter initialization API (r_ctsu_filter_open) description for more details.

### 4.3.2 r_ctsu_iir_filter

This function applies the IIR filter processing on one measurement result.

Format

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

Parameters

p_ctrl

IIR filter management data pointer

p_data

IIR filter measurement result data pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_iir_sample.h.

Description

This function applies the IIR filter processing on one measurement result.

When data in the signed 19-bit integer range (262143 to -262144) or higher is passed as measurement value data, the operation is performed as if the upper or lower limit value was entered.

The result of the operation is limited to the range of signed 18-bit integers (131071 to -131072); if it exceeds the range, it will be rounded to the upper or lower limit.

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:

Returns buffer unfilled response during filter stabilization time.

Until the filter stabilization time elapses, the filtered result is the operation result when the unfilled range is in the initialized state (0).

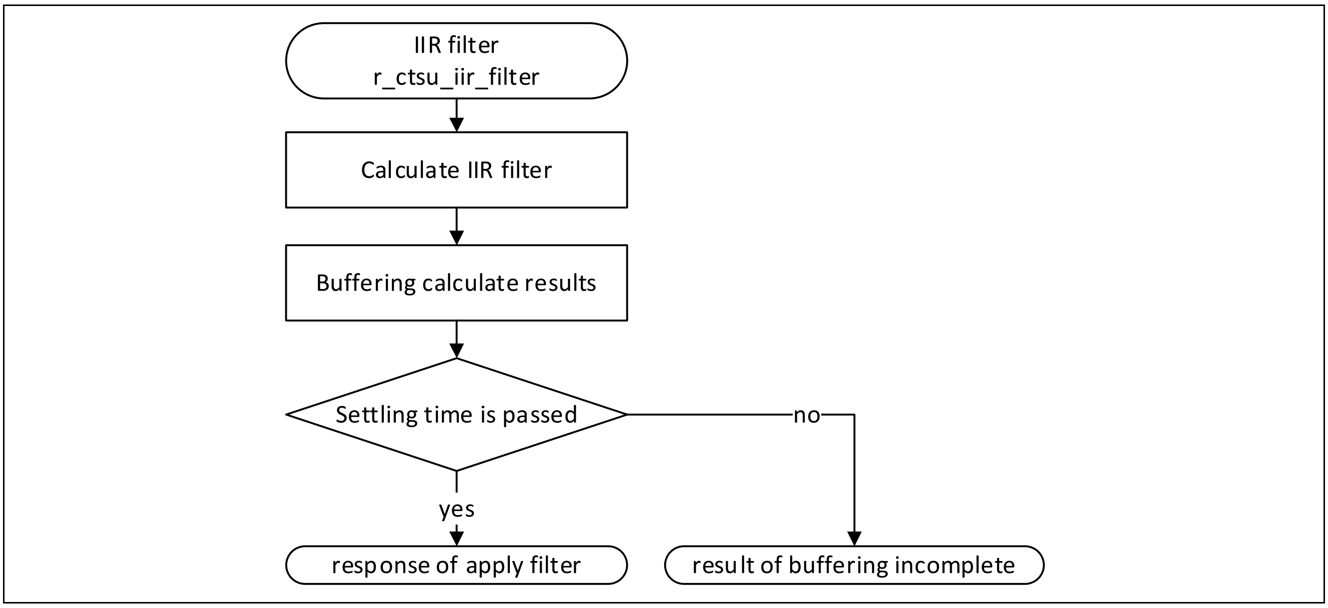



**Figure 4-1 IIR Filter Execution API Flowchart**

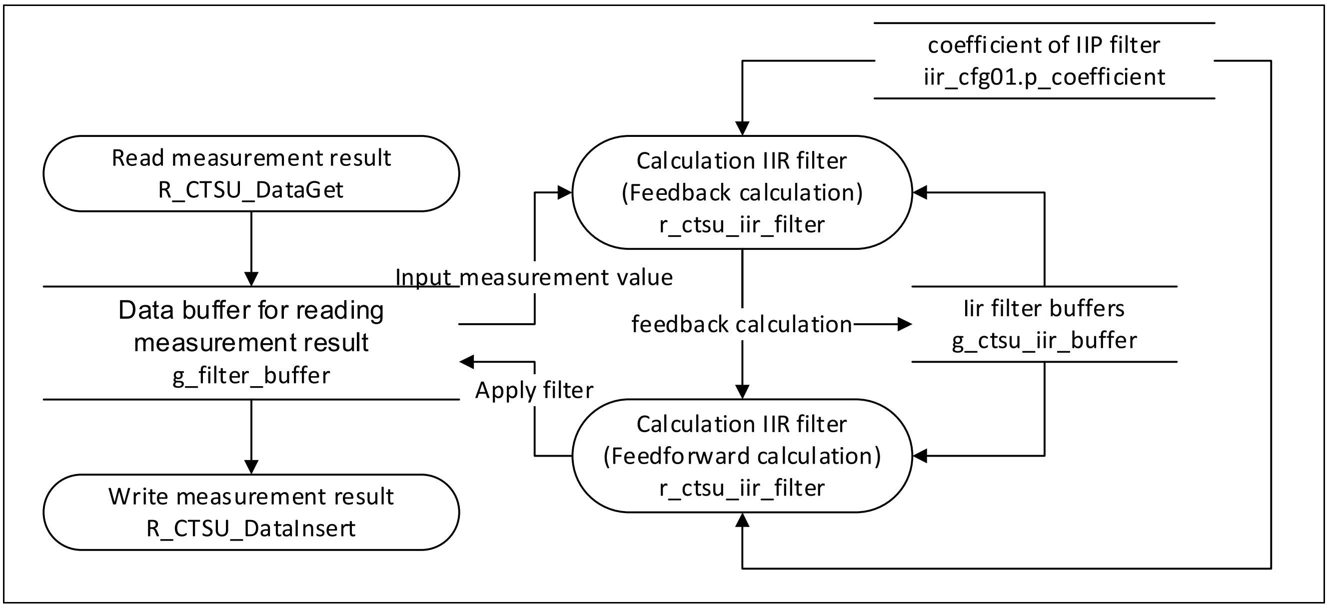



**Figure 4-2 IIR Filter Data Flowchart**

## 4.4 List of Data for IIR Filters

This section explains the constants and global variables provided for IIR filters.

### 4.4.1 Constants

Table 4.3 shows a list of constants for IIR filters.

**Table 4.3 Constants for IIR Filters**

| Constant name | Setting value | Description |
|---|---|---|
| File name: filter_config_sample.h | | |
| CTSU_FILTER_NUM | 1 to 8 | To configure a cascaded IIR filter, specify 2 or more filters. |
| File name: r_ctsu_iir_sample.h | | |
| IIR_FILTER_NUM | 1 to 8 | Number of IIR filter stages<br>To configure a cascaded IIR filter, specify 2 or more filters. |
| File name: r_ctsu_iir_sample.c | | |
| IIR_TAP_SIZE_MIN | 2 | Minimum number of taps (order 1 + 1) |
| IIR_TAP_SIZE_MAX | 5 | Maximum number of taps (order 4 + 1) |
| IIR_SETTLINGS_MAX | 255 | Maximum stabilization wait time |
| IIR_CFG_DECIMAL_POINT | 14 | Fixed-point number of digits |
| IIR_CFG_POINT_OFFSET | 2 | Operational correction value for fixed-point number of digits |
| IIR_FILTER_SIZE | IIR_FILTER_NUM x FILTER_ELEMENT_SIZE | Buffer size for IIR filters<br>(Calculated from the number of filter stages and the number of measurement results.) |
| IIR_DECIMAL_MAX | 0x0003FFFF | Upper limit value of filter data integer part |
| IIR_DECIMAL_MIN | 0xFFFC0000 | Lower limit value of filter data integer part |
| IIR_CALC_MAX | 0x7FFFFFFF | Upper limit value of calculation |
| IIR_CALC_MIN | 0x80000000 | Lower limit value of calculation |
| IIR_RESULT_MAX | 0x0001FFFF | Maximum value of filter result |
| IIR_RESULT_MIN | 0xFFFE0000 | Minimum value of filter result |

### 4.4.2 Global Variables

Table 4.4 lists the global variable

**Table 4.4 Global Variables for IIR Filters**

| Variable name | Data type | Description |
|---|---|---|
| File name: r_ctsu_fir_sample.c | | |
| g_ctsu_iir_element_index | uint16_t | Buffer allocation management index |
| g_ctsu_iir_ctrl[FIR_FILTER_SIZE] | iir_ctrl_t | IIR filter management data<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x number of IIR filter stages.<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins |
| gp_ctsu_iir_ctrl | iir_ctrl_t * | Position pointer at time of IIR filter management data allocation |
| g_ctsu_fir_buffer[IIR_FILTER_SIZE][IIR_TAP_SIZE] | int32_t | IIR filter buffer<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x maximum number of taps (5)<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins |

## 4.5 Filter Adjustment Procedure

You can change the coefficient definition of IIR filters and adjust the filter properties using multiple stages of IIR filters, as described below.

### 4.5.1 Filter Processing Method

For details on the data size and execution processing time, see Table 2.13 and Table 2.14.

Figure 4.3 shows a block diagram of IIR filter.

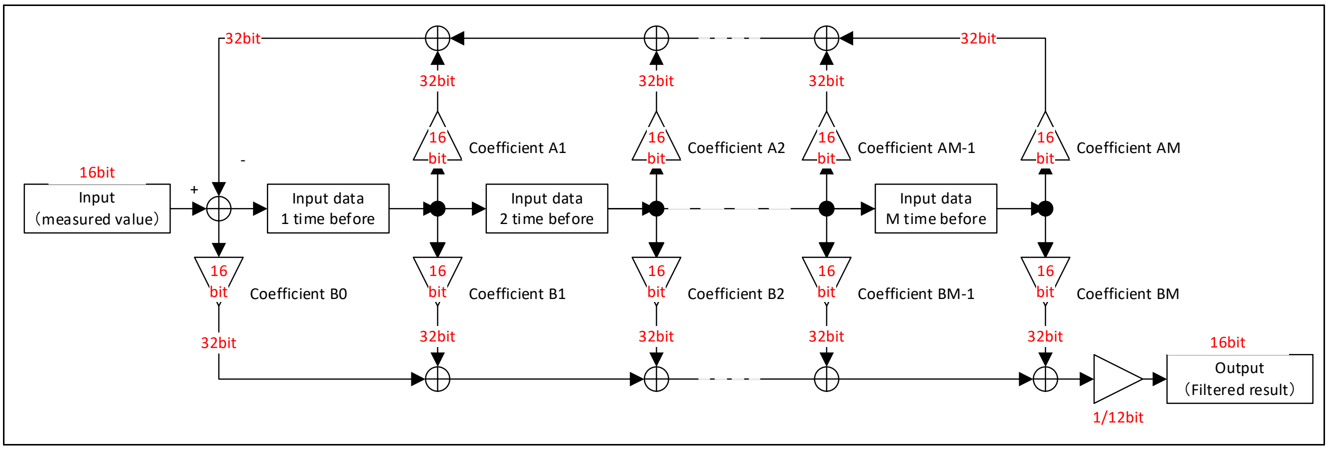



**Figure 4-3 IIR Block Diagram**

### 4.5.2 Filter Characteristics

This sample program can handle filters of up to four orders.

You can also configure five or more orders by defining multiple filters.

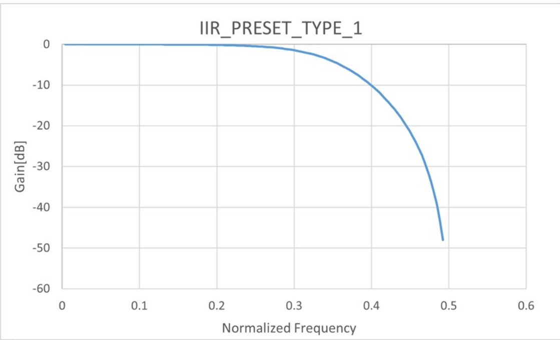
Table 4.5 defines the characteristics of sample FIR filters. The filter characteristics can be changed by specifying the coefficient and filter configuration definitions shown in Table 4.6 and Table 4.7.

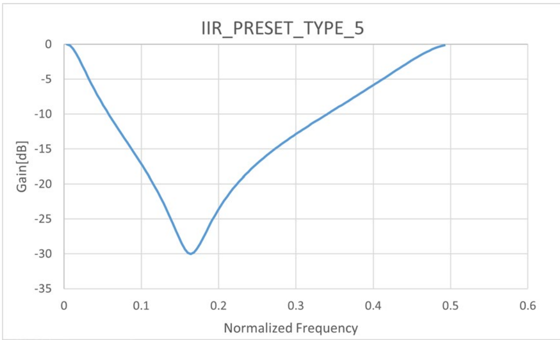
**Table 4.5 Sample IIR Filter Specification**

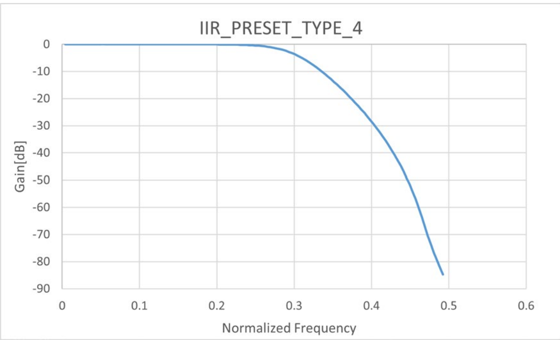
| File | Definition name | Description |
|---|---|---|
| r_ctsu_iir_sample.h | IIR_PRESET_TYPE | Sample preset specification for use with IIR filter |

**Table 4.6 Sample IIR Filters Coefficient Definition (1/2)**

| | IIR_PRESET_TYPE_1 | IIR_PRESET_TYPE_4 | IIR_PRESET_TYPE_5 | IIR_PRESET_TYPE_6 |
|---|---|---|---|---|
| | IIR lowpass filter | IIR lowpass filter | IIR peaking filter | IIR moving-average filter |
| Order | 2 | 4 | 2 | 1 |
| Coefficient A | 0 | 0 | 0 | 0 |
| | 0.595458984375 | 0.6468505859375 | -0.2279052734375 | -0.75 |
| | 0.23492431640625 | 0.6185302734375 | -0.57470703125 | |
| | | 0.14617919921875 | | |
| | | 0.0260009765625 | | |
| Coefficient 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 | | |



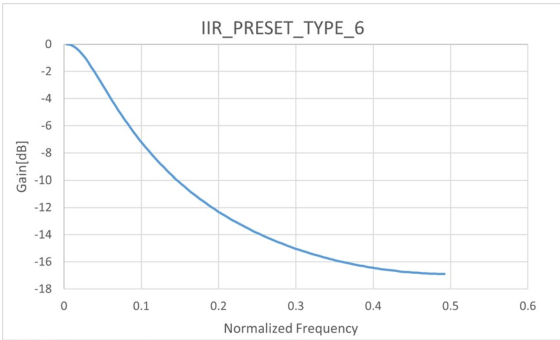
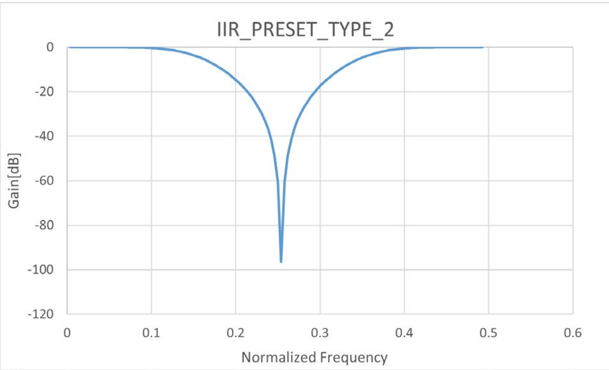
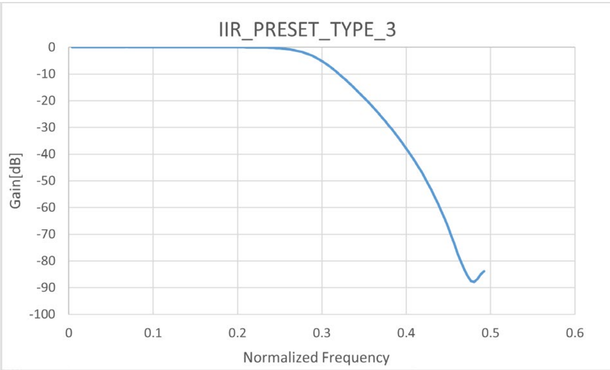



**Table 4.7 Sample IIR Filters Coefficient Definition (2/2)**

| | IIR_PRESET_TYPE_2 | | IIR_PRESET_TYPE_3 | | |
|---|---|---|---|---|---|
| | Cascaded second-order (biquad) IIR band stop filter | | Cascaded second-order (biquad) IIR lowpass filter | | |
| **Order** | 4 | | 5 | | |
| **Coefficient 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 |
| **Coefficient 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 Coefficient Definition

The coefficients of the IIR filter configuration are defined in the form of signed fixed points (decimal numbers) with a 1-bit integral part, and the lower 14 bits are the decimal part; they are treated as the coefficient value divided by 16384.

The coefficient of the sample program should be designed with a value range of -2.0 to 2.0, and the value obtained by multiplying the fractional coefficient by 16384 (0x4000) should be set as the coefficient definition. Small numbers less than 1LSB cannot be expressed and will cause operation errors.

Table 4.8 shows examples of decimal, hexadecimal, and decimal correspondence.

**Table 4.8 Fixed Point Definition Examples**

| Fractional number | Hexadecimal | Decimal |
|---|---|---|
| 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 Filter Configuration Definition

Define the number of taps and coefficients for IIR filters in the iir_config_t type data table.

The number of taps specifies the order of the IIR filter + 1, and the coefficient values of the IIR filter are listed in the coefficient table as 16-bit signed fixed-point numbers in a coefficient AB set, in the order of coefficient B then coefficient A, starting from the zero order.

The definition of coefficient A0 is fixed as 0, so only define it as 0.

The number of taps can only be defined within the range of 2 to 5.

When setting the stabilization time, confirm the filter operation within the specified number of taps. If it takes longer for the filter to stabilize, increase the setting value.

The stabilization time can be set within the range of the number of taps and 255.

Note: Coefficient A is used in the feedback operation of the IIR filter, so depending on the defined value, the operation result may diverge and prevent normal operation results from being output.
Pay careful attention to the definition of coefficient A to ensure that the IIR filter stabilizes.

Note: Define the coefficient table so that the “sum of coefficient B definitions divided by the sum of coefficient A definitions” approaches 1.0.
If the “sum of coefficient B definitions divided by the sum of ‘coefficient A definitions +1’” exceeds 1.0, the measurement result is amplified. If it is less than 1.0, the measurement result is attenuated.

```
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*/
  },
};
```

Specify filter order + 1 as the number of IIR filter taps.

The stabilization time setting is usually the same as the number of taps, but if it takes longer for the filter to stabilize, increase the setting value.

Coefficient A0 must always be 0.

Define the coefficients for the number of taps in IIR filters as coefficient AB sets, in the order of coefficient B then coefficient A.

#### 4.5.4.1 Cascaded IIR Filter Configuration

A cascaded IIR filter can be configured by defining several IIR filters.

When defining multiple IIR filters with a filter order of 2, this becomes a common cascaded biquad IIR filter.

filter_config_sample.h

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

Specify the filter management and number of filter stages of the IIR filter.

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_cfg02,
  },
};
```

Continue to specify the configuration definition of the IIR filter as cascaded.

The filter listed at the top is applied first, the filter listed at the bottom is applied later.

iir_config_sample2.c

Define each IIR filter that will be cascaded.

```
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  */
  },
};
```

# 5. Median Filters

Median filters can be used to remove pulse noise. The effectiveness of median filters against random noise and low-period noise is limited, but they can be used in combination with FIR or IIR filters for such cases.

## 5.1 Operation Explanation

Median filters use the input value and several past samples as a reference period, calculate the median value, and use the result as the output value of the filter. To refer to past samples, immediately after filter initialization when the past sample buffer is not filled, the buffer is initialized (0) and the median input value is output; when the buffer is filled with past data, the median value is output. The filter operation status is determined by the buffer unfilled response.

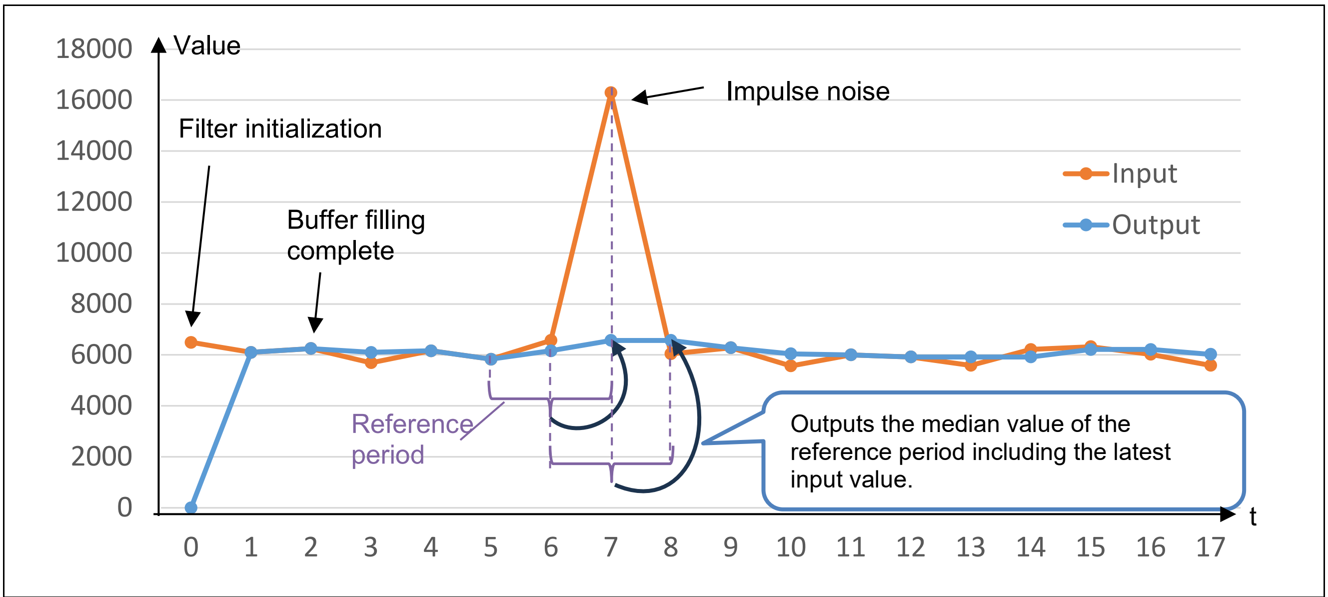



**Figure 5-1 Median Filter Operation Example (Reference Period = 3)**

**Table5.1 Median Filter Operation Example (Reference Period = 3**

| **t** | **Input value** | **Reference period (t)** | **Reference period data (bold number = median value)** | **Output value** | Buffer unfilled response (note) |
|---|---|---|---|---|---|
| 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 | |

Note: Median filter processing API response. For details, refer to 5.4.2 r_ctsu_median_filter.

## 5.2 Specifications

Table 5.2 lists the specifications for median filters used in the sample program.

**Table 5.2 Median Filter Specifications**

| Item | Specification | Remarks |
|---|---|---|
| Input data type | Signed 32-bit integer type | CTSU driver measurement data is unsigned 16-bit integer type, so data type conversion is required. This sample software performs data type conversion in the software filter API. |
| Output data type | Signed 32-bit integer type | |
| Sample reference range | 3, 5, 7, 9 | Total number of input values and past samples |
| Processing method | Median detection with insertion sort | |
| Output results after filter initialization | Returns operation results during filter stabilization time and buffer unfilled response | Any sample reference period can be specified as the filter stabilization time in the configuration definition. |

## 5.3 List of Data for Median Filters

This section explains the constants and global variables prepared for use with median filters. Data definitions such as constants and global variables that are common to the software filter sample code are required when using median filters. For details regarding data definitions common to the software filter sample code, refer to 2.3 Data List for Filter Configuration Definition.

### 5.3.1 Constants

Table 5.3 lists the median filter constants used in the sample program.

**Table 5.3 Constants for Median Filters**

| Constant name | Setting value | Description |
|---|---|---|
| File name: r_ctsu_median_sample.h | | |
| MEDIAN_FILTER_NUM | 1 | Number of median filter configurations<br>Change the setting value when defining multiple median filter configurations and using them based on the touch configuration. |
| File name: r_ctsu_median_sample.c | | |
| MEDIAN_SAMPLE_SIZE_MAX | 9 | Maximum sample reference time<br>Change this setting if using a system where the touch measurement sampling period is shorter than this sample program and 11 or more sample reference periods are necessary. |
| MEDIAN_FILTER_SIZE | MEDIAN_FILTER_NUM × FILTER_ELEMENT_SIZE | Median filter buffer size<br>(calculated from the number of median filter stages and the number of measurement results)<br>“FILTER_ELEMENT_SIZE”: refer to Table 2.2 Constants for Filter Configuration Definitions for details. |

### 5.3.2 Global Variables

Table 5.4 lists the global variables used in the sample program.

The variables are initialized and updated using the median filter initialization process.

**Table 5.4 Global Variables Median Filters**

| Variable name | Type | Description |
|---|---|---|
| File name: r_ctsu_median_sample.c | | |
| g_ctsu_median_element_index | uint16_t | Buffer allocation management index |
| g_ctsu_median_ctrl[MEDIAN_FILTER_SIZE] | median_ctrl_t | Median filter management data<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x number of median filter stages.<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins |
| gp_ctsu_median_ctrl | median_ctrl_t * | Position pointer at time of median filter management data allocation |
| g_ctsu_median_buffer[MEDIAN_FILTER_SIZE][MEDIAN_SAMPLE_SIZE_MAX] | int32_t | Median filter sample buffer<br>Buffer size is number of pins (number of self-capacitance electrodes + number of mutual-capacitance electrodes x 2) x maximum sample reference period (9)<br>※Number of mutual-capacitance electrodes = number of transmitting pins × number of receiving pins |
| g_ctsu_median_work[MEDIAN_SAMPLE_SIZE_MAX] | int32_t | Sorting processing temporary buffer<br>Buffer size is maximum sample reference period (9) |

### 5.3.3 Structures

The following shows the structure for accessing the median filter APIs and the structures for defining the median filter configuration.

**Table 5.5 Filter Structure Definitions**

| Definition content | Data type | Remarks |
|---|---|---|
| Median filter configuration definition | median_config_t | |
| Median filter management data | median_ctrl_t | |

#### 5.3.3.1 Median filter configuration definition (median_config_t)

**Table 5.6 Median Filter Configuration Definition Structure (median_config_t)**

| Member | Data type | Description |
|---|---|---|
| samples | uint16_t | Sample reference period<br>Only odd numbers within the maximum sample reference period (9) samples can be specified. |

- Example description of median filter configuration definition (median_config_t)

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

#### 5.3.3.2 Median filter management data (median_ctrl_t)

**Table 5.7 Median Filter Management Data (median_ctrl_t)**

| Member | Data type | Description |
|---|---|---|
| index | uint16_t | Median filter sampling buffer input data storage location |
| count | uint16_t | Buffer filling counter |
| p_buffer | int32_t * | Median filter sampling buffer pointer |
| p_cfg | median_config_t * | Median filter configuration definition pointer |

## 5.4 Median Filter APIs

This section explains the APIs prepared for use with median filters. APIs that are common to the sample software filter sample code are required when using median filters. For details regarding APIs common to the software filter sample code, refer to 2.4 Software Filter APIs.

Table5.8 lists the median filter APIs included in the sample program.

**Table5.8 Median Filter APIs**

| Function name | Process description |
|---|---|
| Filter name: r_ctsu_median_sample.c | |
| r_ctsu_median_open | Median filter initialization process |
| r_ctsu_median_filter | Median filter execution process |
| ctsu_insert_sort | Insert sorting process |

### 5.4.1 r_ctsu_median_open

This function allocates and initializes the buffer for median filter processing. This function must be executed before using any other median filter API functions.

Before executing this function, make sure to set a pointer to the median filter management data by referencing the median filter management data allocation position pointer.

This function must be executed the number of times the measurement result data is read by the CTSU driver for each touch interface. (For self-capacitance method, this is the number of pins, for mutual capacitance method, this is "the number of transmitting pins x the number of receiving pins x 2.")

Refer to the filter initialization API (r_ctsu_filter_open) description for more details.

Format

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

Parameters

p_ctrl

Median filter management data pointer

Sets the median filter management data allocation position pointer (gp_ctsu_median_ctrl).

The first pointer position for each touch configuration must be retained because the median filter management data allocation position pointer (gp_ctsu_median_ctrl) is updated each time this API is executed.

p_cfg

Median filter configuration definition pointer

In this software, the function specifies the source median filter configuration definition for the median filter sample preset.

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |

Properties

Protype is declared in r_ctsu_median_sample.h.

Description

This function allocates and initializes the buffer for median filter processing of 1 measurement result.

### 5.4.2 r_ctsu_median_filter

This function applies the median filter operations on one measurement result.

It returns the buffer unfilled response until the filter stabilization time elapses.

Until the filter stabilization time elapses, the filter application result is the calculation result when the unfilled range is in the initialized state (0).

Format

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

Parameters

p_ctrl

Median filter management data pointer

p_data

Median filter application measurement result data pointer

ReturnValues

| | |
|---|---|
| FSP_SUCCESS | /* Successfully completed. */ |
| FSP_ERR_ASSERTION | /* Argument pointer not specified. */ |
| FSP_ERR_INVALID_ARGUMENT | /* Configuration parameters are invalid. */ |
| FSP_ERR_BUFFER_EMPTY | /* Some filters are not yet applied because buffer is unfilled. */ |

Properties

Protype is declared in r_ctsu_median_sample.h。

Description

This function applies the median filter processing on one measurement result.

When data in the signed 13-bit integer range (1073741823 to -1073741824) or higher is passed as measurement value data, the operation is performed as if the upper or lower limit value was entered.

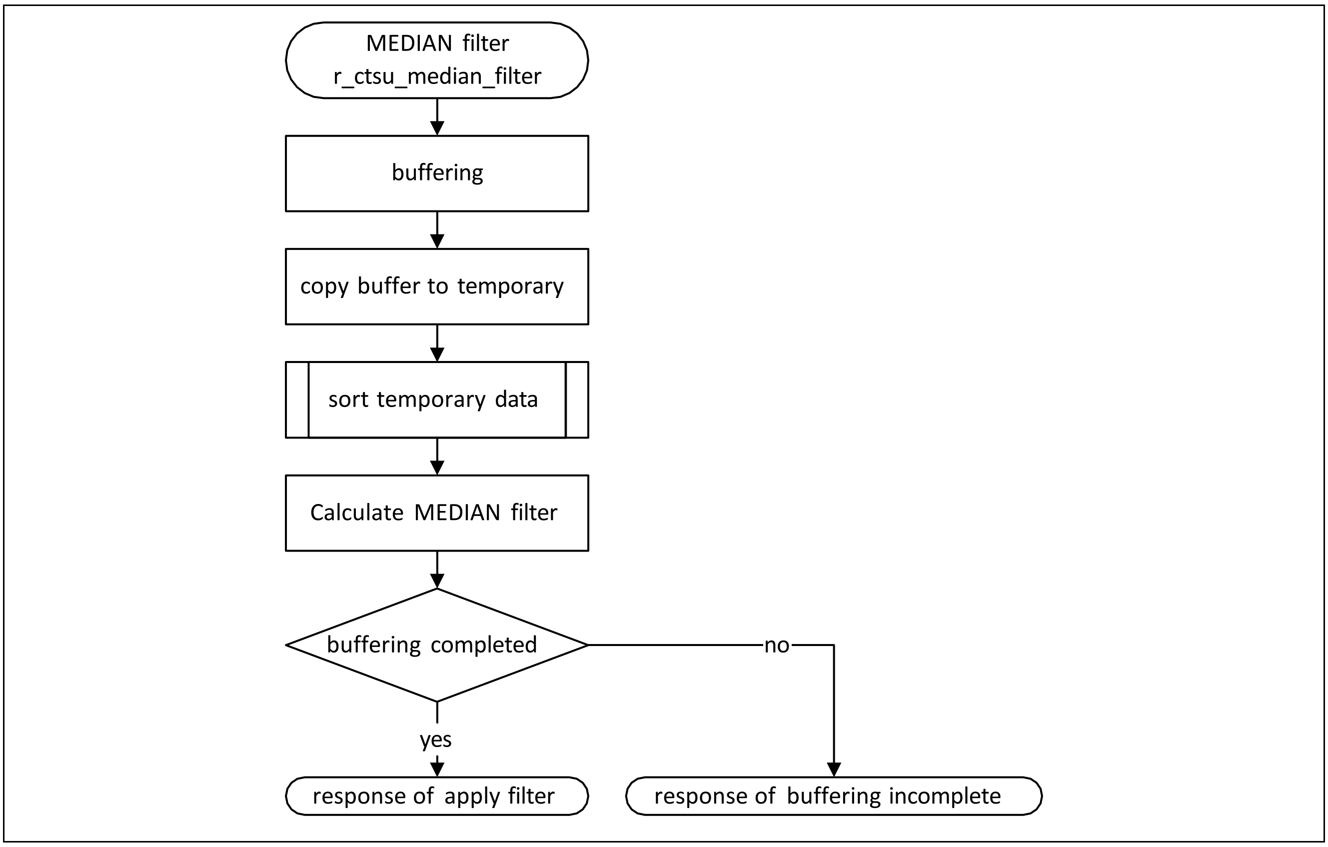



**Figure 5-2 Median Filter Execution API Flowchart**

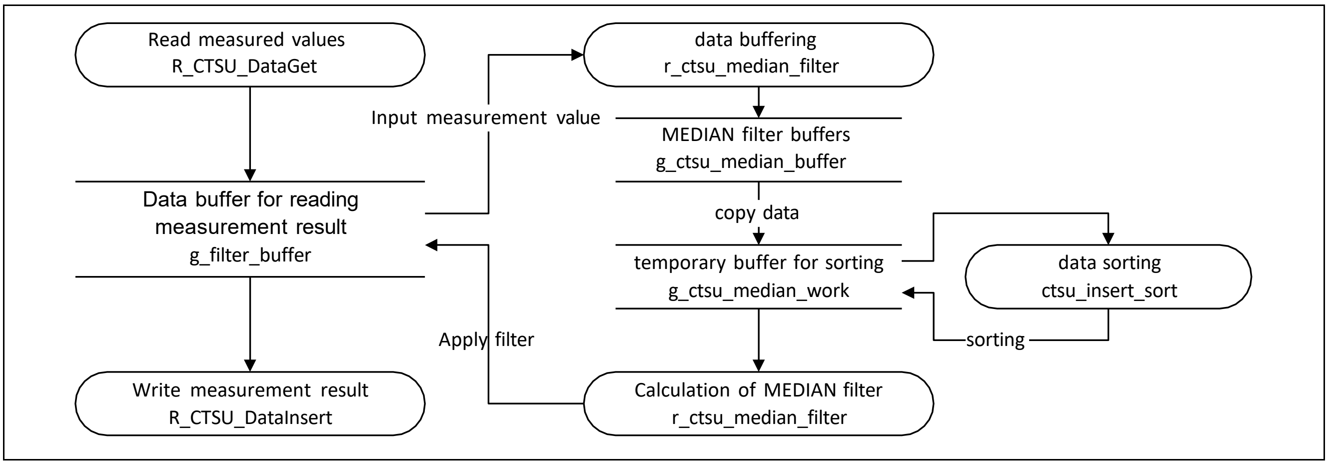



**Figure 5-3 Median Filter Data Flowchart**

### 5.4.1 ctsu_insert_sort

This function sorts the specified data by value.

Format

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

Parameters

p_list

Specified sorting data pointer

size

Number of data to be sorted

ReturnValues

Note

Properties

Protype is declared in r_ctsu_median_sample.c.

Description

This function sorts the specified data in ascending order.

## 5.5 Usage Example

### 5.5.1 Program Implementation Example

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 Filter Adjustment Procedure

In this sample program, you can specify the filter characteristics with a conditional compilation.

To do so, change the sample reference period specification of the media filter configuration and then adjust the filter characteristics.

#### 5.5.2.1 Filter Processing Method

The median filter processing method samples and sorts the measurement results, and then calculates the median value, so the larger number of samplings the long processing time.

For details regarding the data size and processing time, refer to Table 2.13 and Table 2.14.

Figure 5-4 Figure 5-4 shows the block diagram of the median filter.

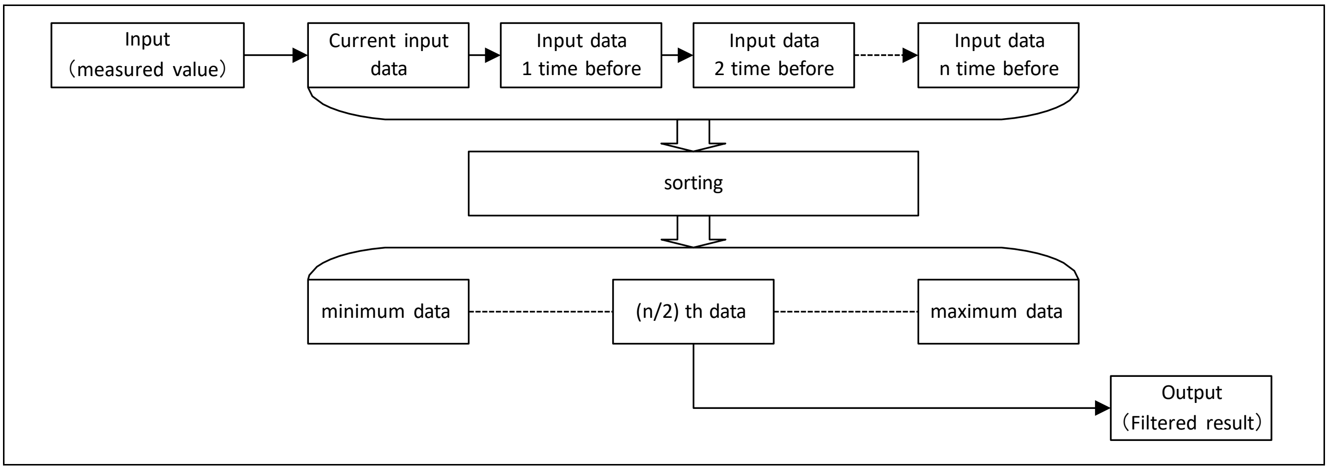



**Figure 5-4 Median Filter Block Diagram**

### 5.5.3 Filter Characteristics

This sample program can handle from 3 to 9 orders in the sample reference period.

Table 5.9 lists the definition of the specified characteristics of the sample median filter. Table 5.10 lists the filter configuration definitions.

**Table 5.9 Sample Median Filter Specification**

| File | Definition name | Description |
|---|---|---|
| r_ctsu_median_sample.h | MEDIAN_PRESET_TYPE | Sample preset specification for use with median filters |

**Table 5.10 Sample Median Filter Configuration Definitions**

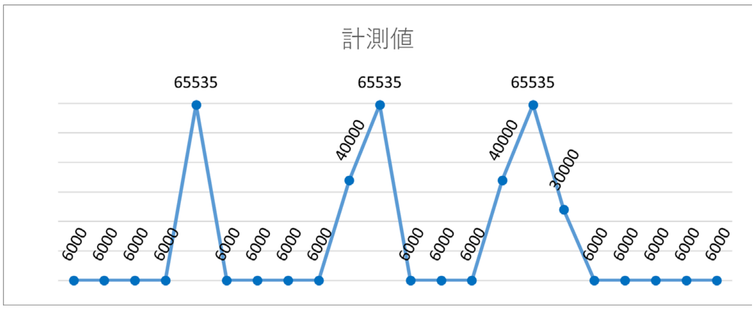
| | MEDIAN_PRESET_TYPE_1 | MEDIAN_PRESET_TYPE_2 |
|---|---|---|
| **Sample reference period** | 3 | 5 |
| **Noise removal width** | 1(20ms) | 2(40ms) |
| **Detection delay** | 1(20ms) | 2(40ms) |

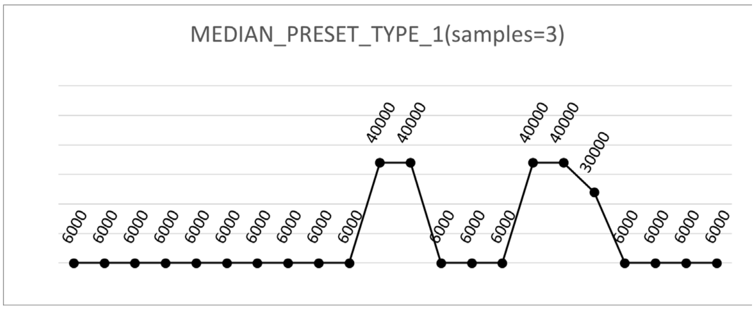
#### 5.5.3.1 Removeable noise width

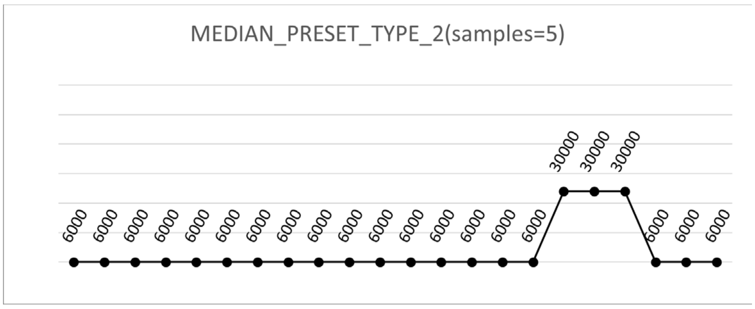
A median filter removes noise signals that are equal to or less than the “sampling period” × “noise removal width”.

The “noise removal width” is calculated as ((sample reference period - 1) ÷ 2）, and a noise signal that exceeds the “sampling period” × “noise removal width” will take a signal shape in which the signal corresponding to the “noise removal width” has been removed.

The sample program’s preset specification targets a noise removal width 1 or 2, so if you want to remove a noise signal with a wider width, refer to 5.3.3.1 Median filter configuration definition (median_config_t) and set the sample reference period to 7 or more.




- Noise signals with a signal width of 20ms (noise width 1) are removed by either Preset 1 (sample reference period = 3) or Preset 2 (sample reference range = 5).

- A noise signal with a signal width of 40ms (noise width 2) has a shape in which one point (65535) at the peak of the signall is removed in Preset 1 (sample reference period = 3), and is removed in Preset 2 (sample reference period = 5).

- A noise signal with a signal width of 60ms (noise width 3) has a shape in which one point (65535) at the peak of the signal is removed in Preset 1(sample reference period = 3), and a shape in which 2 points (65535, 40000) in the peak of the signal are removed in Preset 2 (sample reference period = 5).

**Noise width 1 (20ms):**

| Measure-ment value | Filter result | |
|---|---|---|
| | 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 |

**Noise width 2 (40ms):**

| Meausre-ment value | Filter result | |
|---|---|---|
| | PRESET 1 | PRESET 2 |
| 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 |

**Noise width 3 (60ms):**

| Measure-ment value | Filter result | |
|---|---|---|
| | PRESET 1 | PRESET 2 |
| 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 Detection delay

The median filter removes noise signals by sampling touch measurement values, causing a delay in normal touch detection.

The touch detection delay time is the same as the removable noise width (sampling period x noise removal width).

# 6. How to use This Sample Project

## 6.1 Sample Filter Program

### 6.1.1 Procedure for Integration into an Existing Project

To incorporate FIR filters into an existing capacitive touch application, proceed as follows:

To incorporate IIR filters, replace the folder names, filter names etc. with those belonging to the IIR folder before executing the procedure.

1. Copy the filter_sample folder in Touch_filter_sample_source/touch_filter_fir folder to the target project.

2. Open "C/C++Project Settings" in the menu project and go to Paths and Symbols under C/C++ General. Add the filter_sample folder to “Include” and “Source Locations."

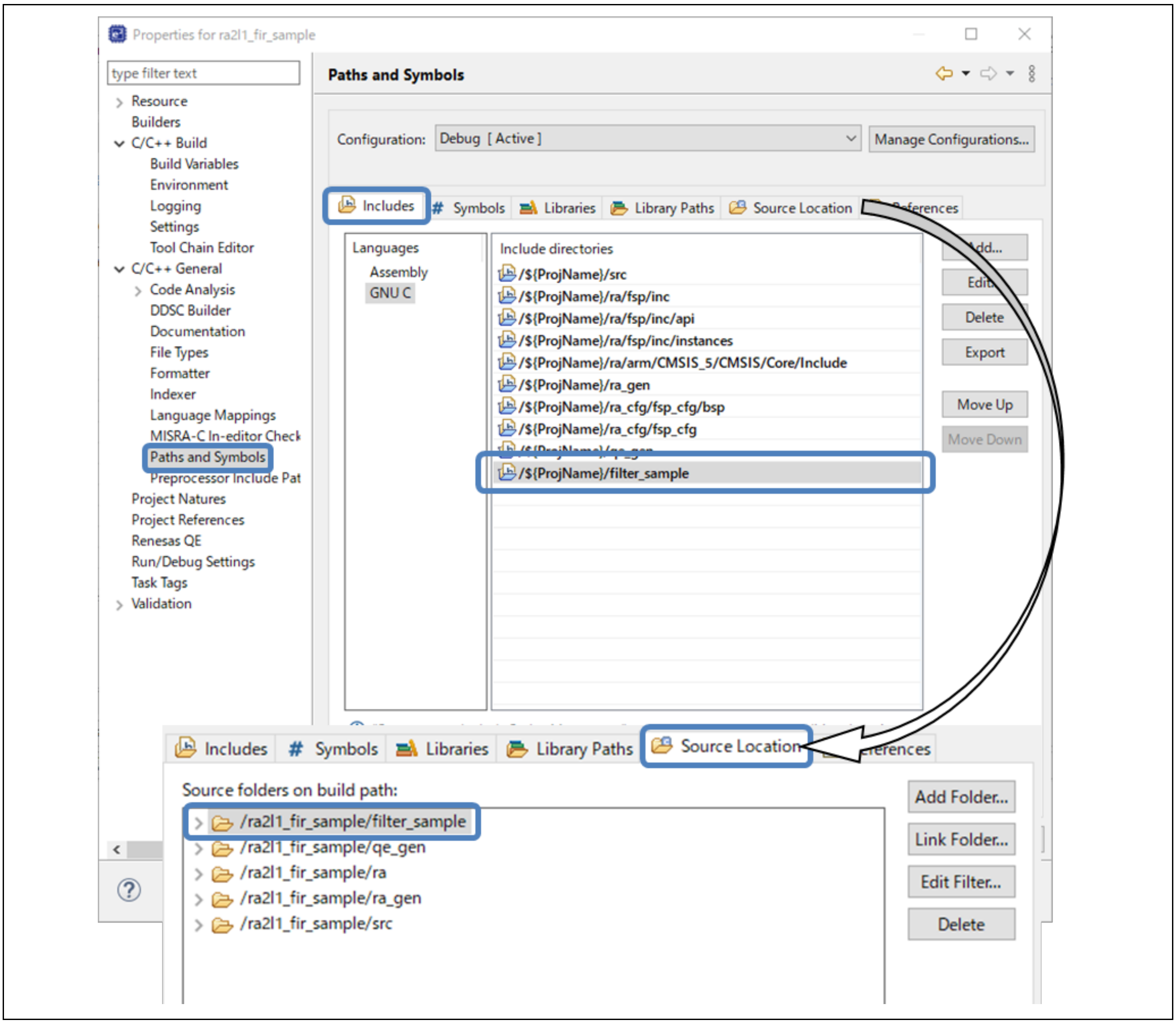



**Figure 6-1 Embedding a Sample Program in an Existing Environment**

3. Add filter configuration definitions to match the number of methods in the touch interface configuration of the embedded environment.
   Check the qe_touch_config.c file, and add the data definition of the ctsu_filter_instance_t type and the data of the filter_ctrl_t type of the filter_config_sample.c file so that the number is equal to the data definition of the touch_instance_t type.

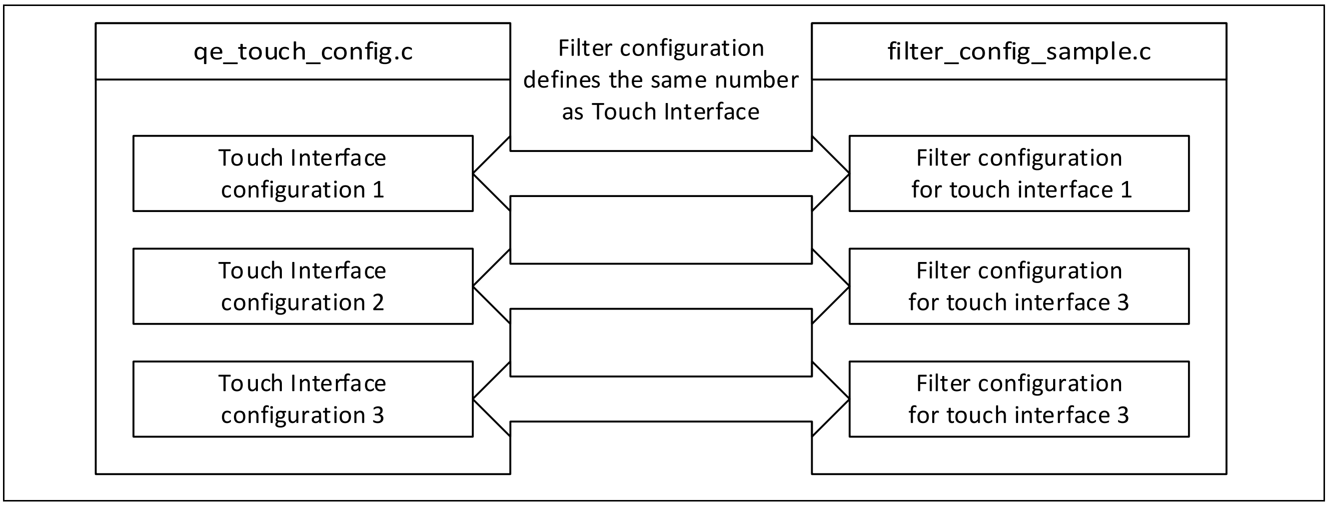



**Figure 6-2 Adding Filter Configuration Definitions**

- Description example of filter configuration definitions

```
qe_touch_config.c

touch_instance_ctrl_t g_qe_touch_ctrl_config01;
const touch_instance_t g_qe_touch_instance_config01 =

(Omitted)

touch_instance_ctrl_t g_qe_touch_ctrl_config02;
const touch_instance_t g_qe_touch_instance_config02 =

(Omitted)

touch_instance_ctrl_t g_qe_touch_ctrl_config03;
const touch_instance_t g_qe_touch_instance_config03 =
```

Match the number of configuration definitions

```
filter_config_sample.c

filter_instance_ctrl_t g_ctsu_filter_control01;
const ctsu_filter_instance_t g_ctsu_filter_instance01 =

(Omitted)

filter_instance_ctrl_t g_ctsu_filter_control02;
const ctsu_filter_instance_t g_ctsu_filter_instance02 =

(Omitted)

filter_instance_ctrl_t g_ctsu_filter_control03;
const ctsu_filter_instance_t g_ctsu_filter_instance03 =
```

4. Modify the filter configuration definition in filter_config_sample.c according to your environment and specify the filter to be applied. (See section 2.3.4).)
   For FIR filters, you can specify filter characteristics from a 4-pattern sample preset in the conditional compilation FIR_PRESET_TYPE.

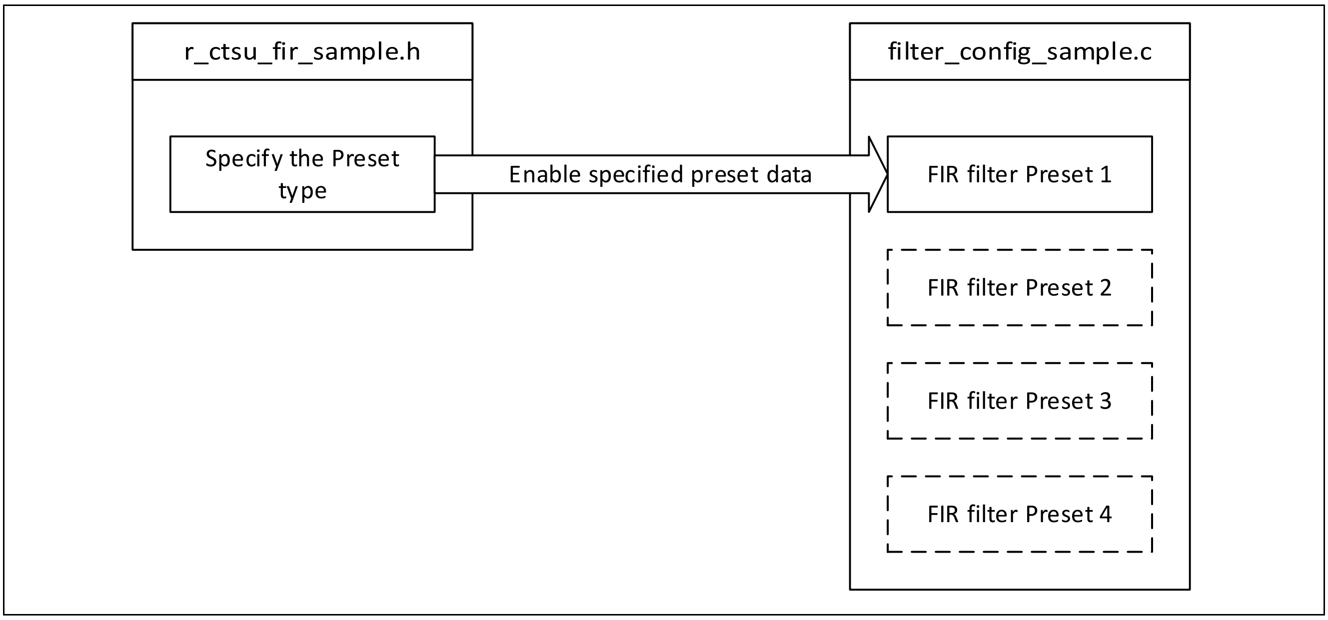



**Figure 6-3 FIR Filter Specification**

Example definition of FIR filter preset specification

```
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


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

(Omittted)

};
```

Use the specified sample presets.

5. Include the filter_config_sample.h file in the qe_touch_sample.c file (or equivalent file) and add a description of how to perform filtering (see Section 5.5).

[Note] 1. Note that data reading and data writing back for filter processing occur in the CTSU drivers, not in the touch API.
2. Note that the description of performing the filtering is required for each method of the Touch Interface configuration.

6. Change the num_moving_average setting of CTSU driver configuration definition (g_qe_ctsu_ctrl_XXX for QE for Capacitive Touch generation) in the qe_touch_config.c file (or equivalent file) to 1 to disable the default moving averaging. No changes are required when using the default moving averaging with FIR filters.
If there are multiple touch interface configuration methods, change the CTSU driver configuration definition for all methods.

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

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,
};
```

Change to 1

```
Const ctsu_cfg_t g_qe_ctsu_cfg_config02 =
{
(Omitted)
    .num_moving_average = 1,        Change to 1
    .tunning_enable = true,
    .p_callback = &qe_touch_callback,
(Omitted)
};

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,
};
Const ctsu_cfg_t g_qe_ctsu_cfg_config03 =
{
(Omitted)
    .num_moving_average = 1,        Change to 1
    .tunning_enable = true,
    .p_callback = &qe_touch_callback,
(Omitted)
};

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 Sample Application Configuration and Operation

The flowchart for incorporating a filter sample program into the sample code (qe_touch_sample.c) outputted by QE for Capacitive Touch is shown below. This sample program shows three touch interface configurations (methods).

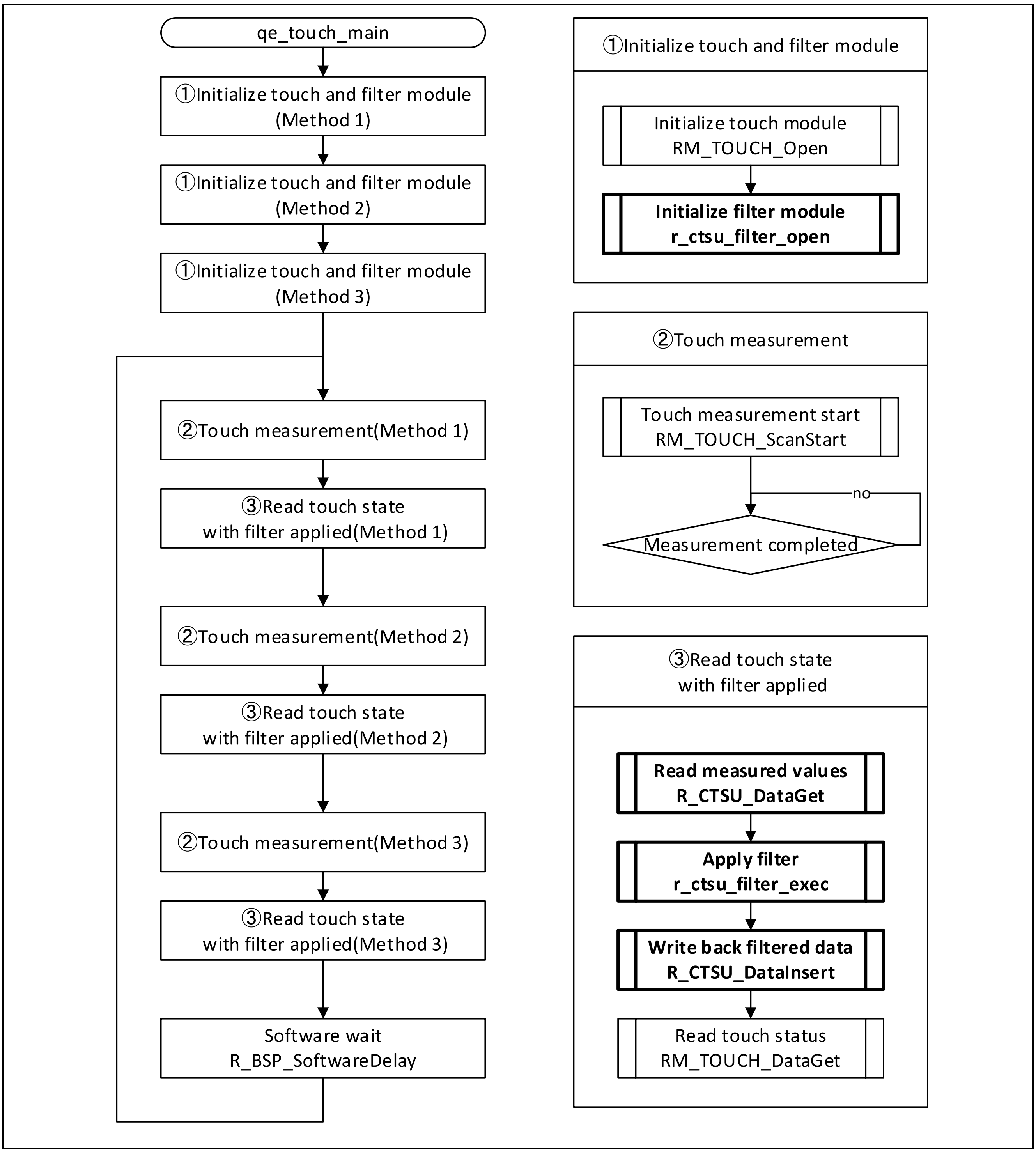



**Figure 6-4 Sample Application Flowchart**

This section describes the numbers indicated in Figure 5.5

① Initialize the touch functions and filter

Initializes the touch function and initializes the filter.

To initialize the filter, check the touch interface configuration and specify the corresponding CTSU driver configuration definition for the respective method.

```
/* 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 Touch measurement

Perform touch measurement and wait for measurement to be completed.

② to ③ should be executed consecutively for each method of the touch interface configuration.

③ Touch input stats after filter application

Use the CTSU driver API to get the measurement result, write it back to the CTSU driver after applying the filter, and then get the touch input information using the filtered data.

The data buffer is required for data transfer between the CTSU driver and the filter function.

② to ③ should be executed consecutively for each method of the touch interface configuration.

For details of the CTSU driver API, refer to v4.3.0 or later of Renesas Flexible Software Package (FSP) User's Manual (R11UM0155).

```
/* 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 Example Project Integrating Filter Sample Program

This section explains the operation of the sample project (ra2l1_rssk_filter_sample) that applies the software filter sample program to the RA2L1 Capacitive Touch Evaluation System Example Project.

### 6.2.1 Function

The functions are shown below.

- Applies a software filter to the measurement results of all touch electrodes on the self-capacitance electrode board.
- When the touch electrodes of the self-capacitance electrode board are touched, the corresponding LED lights.
- You can use the serial monitoring function of QE for Capacitive Touch to check the measurement with the software filter applied.

### 6.2.2 File Structure

This section explains the file structure of the sample project.

The project configuration file and FSP Configuration generation file of the development environment are omitted.

Differences from Example Project are shown in bold. For more information on unchanged files, refer to "RA2L1 Group Capacitive Touch Evaluation System Example Project" (R20AN0595).

ra2l1_rssk_filter_sample<br>
│<br>
├─QE-Touch<br>
│　　├ qe_tuning20230221103059.log ・・・QE Tuning Log<br>
│　　└ quickstart_rssk_ra2l1_ep.tifcfg ・・・Touch Interface Configuration File<br>
│<br>
├─qe_gen<br>
│　　├ qe_touch_config.c ・・・Touch Configuration Source<br>
│　　├ qe_touch_config.h ・・・Touch Configuration Header<br>
│　　├ qe_touch_define.h ・・・Touch Definition Header<br>
│　　└ **qe_touch_sample.c ・・・Touch Sample Application**<br>
│<br>
├─src<br>
│　　├ hal_entry.c ・・・main Files<br>
│　　├ r_rssk_switch_led.c ・・・Switch and LED Processing Header<br>
│　　├ r_rssk_switch_led.h ・・・Switch and LED Processing Header<br>
│　　├ r_rssk_touch_led.c ・・・Touch Electrode LED Processing Header<br>
│　　└ r_rssk_touch_led.h ・・・Touch Electrode LED Processing Header<br>
│<br>
└─**filter_sample**<br>
　　├ **filter_config_sample.c ・・・Filter Configuration Definition Source**<br>
　　├ **filter_config_sample.h ・・・Filter Configuration Definition Header**<br>
　　├ **fir_config_sample1.c ・・・FIR Filter Sample Preset 1 Source**<br>
　　├ **fir_config_sample2.c ・・・FIR Filter Sample Preset 2 Source**<br>
　　├ **fir_config_sample3.c ・・・FIR Filter Sample Preset 3 Source**<br>
　　├ **fir_config_sample4.c ・・・FIR Filter Sample Preset 4 Source**<br>
　　├ **iir_config_sample1.c ・・・IIR Filter Sample Preset 1 Source**<br>
　　├ **iir_config_sample2.c ・・・IIR Filter Sample Preset 2 Source**<br>
　　├ **iir_config_sample3.c ・・・IIR Filter Sample Preset 3 Source**<br>
　　├ **iir_config_sample4.c ・・・IIR Filter Sample Preset 4 Source**<br>
　　├ **iir_config_sample5.c ・・・IIR Filter Sample Preset 5 Source**<br>
　　├ **iir_config_sample6.c ・・・IIR Filter Sample Preset 6 Source**<br>
　　├ **median_config_sample1.c ・・・Median Filter Sample Preset 1 Source**<br>
　　├ **median_config_sample2.c ・・・Median Filter Sample Preset 2 Source**<br>
　　├ **r_ctsu_filter_sample.c ・・・Filter Processing Source**<br>
　　├ **r_ctsu_filter_sample.h ・・・Filter Processing Header**<br>
　　├ **r_ctsu_fir_sample.c ・・・FIR filter Processing Source**

└ **r_ctsu_fir_sample.h** ・・・**FIR filter Processing Header**
├ **r_ctsu_iir_sample.c** ・・・**IIR filter Processing Source**
└ **r_ctsu_iir_sample.h** ・・・**IIR Filter Processing Header**
├ **r_ctsu_median_sample.c** ・・・**Median Filter Processing Source**
└ **r_ctsu_median_sample.h** ・・・**Median Filter Processing Header**

### 6.2.3 How to Import the Sample Project

Import the "ra2l1_rssk_filter_sample" folder attached to this sample code into your workspace using the e2studio import function.

Figure 6-5 shows how to import a sample project.

For operations after import, refer to “RA2L1 Group Capacitive Touch Evaluation System Quick Start Guide (Q12QS0040).”

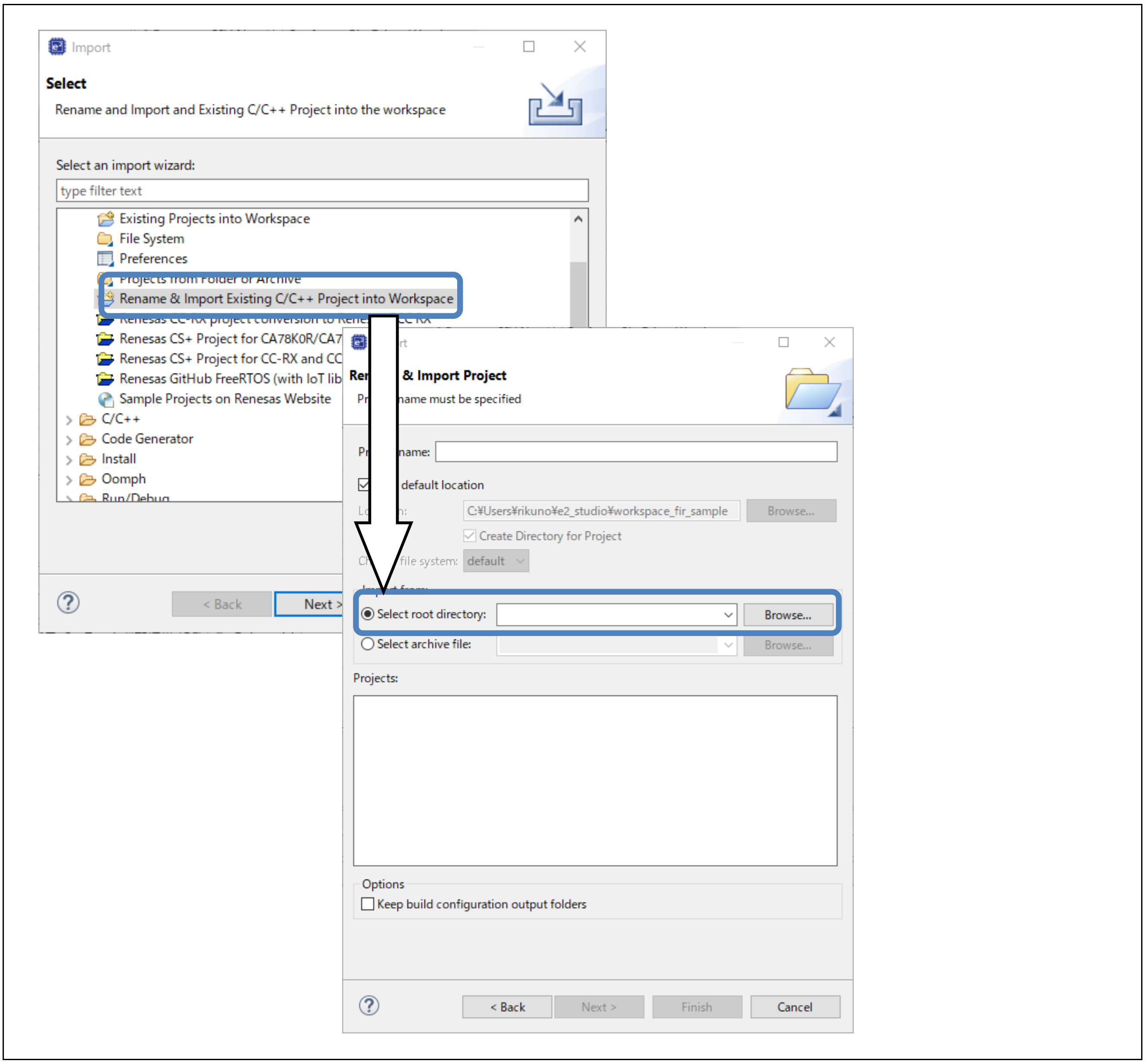



**Figure 6-5 Importing the Sample Project**

### 6.2.4 How to Change the Filter Configuration and Preset

For details on how to change the appropriate filter configuration (type) in the sample project, refer to Table 2.2 Constants for Filter Configuration Definitions.

To change the FIR Filter preset, refer to Table 3.6 Sample FIR Filters Specification.

To change the IIR filter preset, refer to Table 4.5 Sample IIR Filter Specification.

To change the median filter preset, refer to Table 5.9 Sample Median Filter Specification.

## 7. Supporting Documentation

- Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426)
- Renesas RA Family RA2L1 Group Capacitive Touch Evaluation System Quick Start Guide (Q12QS0040)
- RA Family Using QE and FSP to Develop Capacitive Touch Applications (R01AN4934)

Renesas Website and Support Desk

Renesas Electronics Website

https://www.renesas.com/

Capacitive Touch Sensor Unit (CTSU) related links

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

https://www.renesas.com/qe-capacitive-tou ch

Renesas Support Desk

https://www.renesas.com/support

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Jun.12.23 | - | First edition issued |
| 2.00 | Aug.31.23 | | Overall restructure of document<br>Added “Section 4. IIR Filters”<br>Added IIR filter-related items to folder structure in Section 1.1<br>Corrected Figure 2.1<br>Added IIR filter-related items to file structure in Section 2.2<br>Corrected remarks regarding coefficient data type in Table 3.1<br>Corrected mistakes in Section 3.5.4<br>Added IIR filter-related items to Section |
| 3.00 | Nov.30.23 | 4 | Added median filters related items to the folder structure in Section 1.1 |
| | | 5 | Updated Table 1.1 (Operation Confirmation Conditions) |
| | | 6 | Deleted description of planned functions from Section 2 |
| | | 7 | Updated Table 2.1 (List of Components) |
| | | 8 | Added items related to median filters to the file structure and added folder descriptions in Section 2.2 |
| | | 9 | Added valid/invalid definitions to each filter in Table 2.2 |
| | | 10 | Added median filters to Table 2.3 |
| | | 13 | Added median filters to Table 2.12 |
| | | 19 | Added median filters to Figure 2.5 |
| | | 24 | Added median filters to Section 2.4.6 |
| | | 24 | Added median filter initialization setting API to Section 2.4.6 |
| | | 26 | Added median filter special note to Section 2.4.7 |
| | | 28 | Added data size and incremental amount to Table 2.13 |
| | | 28 | Updated filter processing execution time in Table 2.14 |
| | | 31 | Corrected information in Section 3.3.1 |
| | | 45 | Corrected information in Section 4.3.1 |
| | | 49 | Corrected information in Table 4.4 |
| | | 57 | Added Section 5 (Median Filters) |
| | | 70 | Revised structure of Section 6 |

# General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com




## Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.