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).

Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426)

https://www.renesas.com/node/25428131

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) to which the Software Filter Sample Program is applied 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
│
└─ 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) • RA2L1 CPU (Model: RTK0EG0018C01001BJ) • 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	V4.3.0
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.

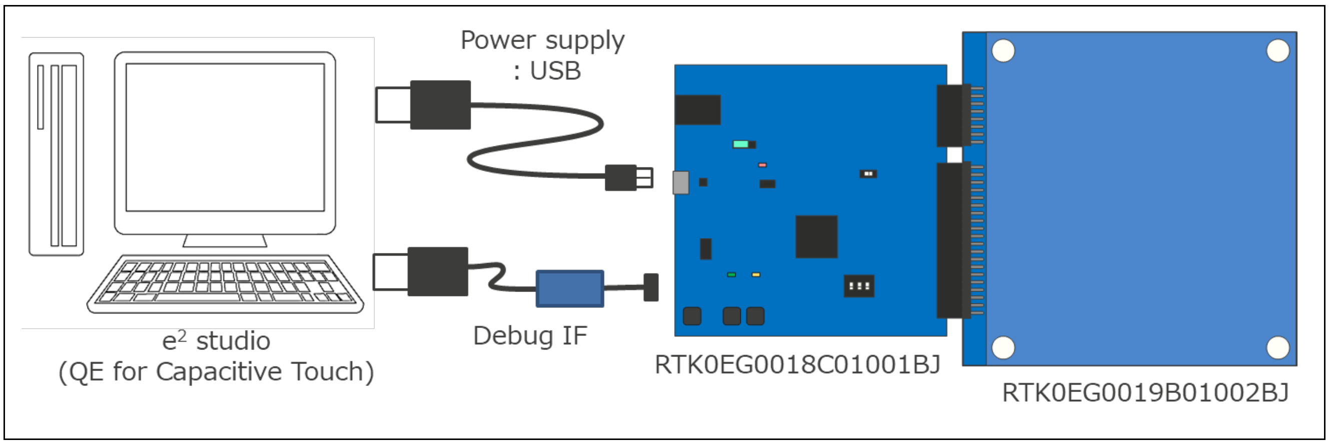


Figure 1.1 Device Connection Diagram

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. You manage the software filters you use in the filter configuration definition. Although the filter configuration defined in this sample program is only filter A (FIR filter), several software filters can also be applied. When multiple software filters are used, the application order is the order in which the filter configuration definitions are defined.

In the future, plan to release multiple software filters, and plan to update them one by one.

2.1 Software Configuration Diagram

Figure 2.1 shows the configuration of this sample program.

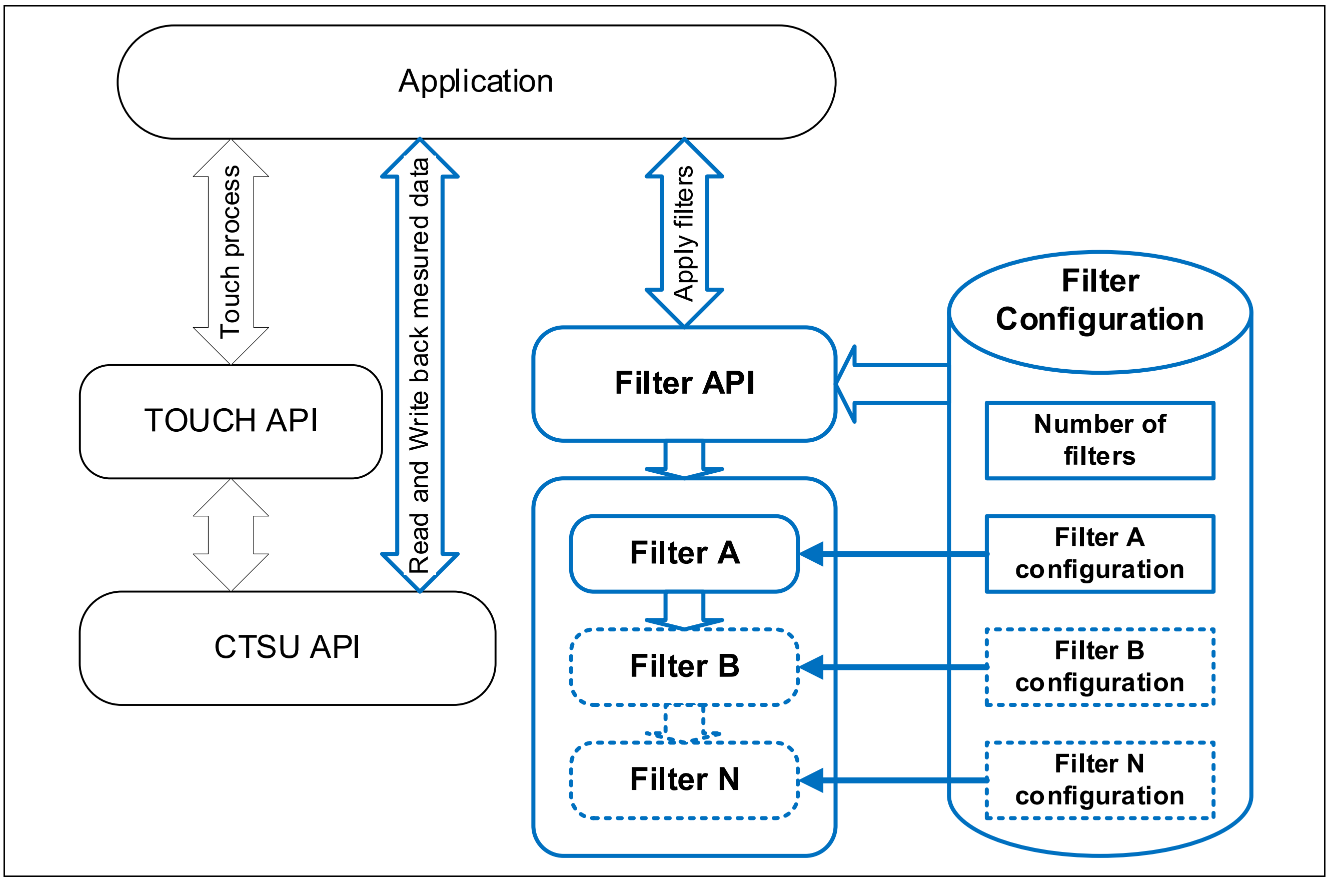


Figure 2.1 Configuration of Sample Program

Figure 2.2 shows the flow of data processing for this sample program.

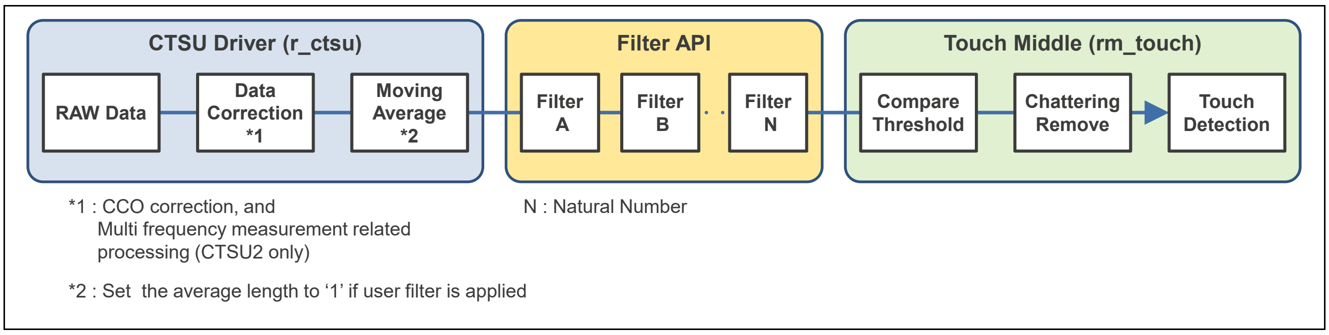


Figure 2.2 Data Processing Flow of Sample Program

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	v4.3.0
I/O Port	v4.3.0
Arm CMSIS Version 5 – Core(M)	v5.9.0+renesas.0.fsp.4.3.0
RA2L1-RSSK Board Support Files	v4.3.0
Board support package for R7FA2L1AB2DFP	v4.3.0
Board support package for RA2L1	v4.3.0
Board support package for RA2L1 – FSP Data	v4.3.0
Asynchronous General Purpose Timer	v4.3.0
Capacitive Touch Sensing Unit	v4.3.0
SCI UART	v4.3.0
Touch	v4.3.0

2.2 File Structures

Table 2.2 to Table 2.4 show the filter_sample file structure.

Table 2.2 Configuration of Filter API Files

File name	Description
r_ctsu_filter_sample.c	Filtering API Sample Program
r_ctsu_filter_sample.h	

Table 2.3 Filter Configuration Definition File Configuration

File name	Description
filter_config_sample.c	Filter configuration definition sample
filter_config_sample.h	
fir_config_sample1.c	Sample Presets for FIR Filters
fir_config_sample2.c	
fir_config_sample3.c	
fir_config_sample4.c	

Table 2.4 FIR Filter File Configuration

File name	Description
r_ctsu_fir_sample.c	FIR filter sample program
r_ctsu_fir_sample.h	

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 2.5 lists the constants.

Table 2.5 Constants for Filter Configuration Definition

Data	Value	Description
CTSU_FILTER_NUM	1	Number of series connections of filters used
FILTER_ELEMENT_SIZE	CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS × 2)	Number of CTSU driver measurement results (Calculated from the touch interface configuration definition.)
FILTER_SIZE	CTSU_FILTER_NUM × FILTER_ELEMENT_SIZE	Number for filter management data

2.3.2 Global Variables

Table 2.6 lists the global variables.

Table 2.6 Global Variables for Filter Configuration Definitions

Data	Data type	Description
g_ctsu_filter_element_index	uint16_t	Index for assigning management data
g_ctsu_fir_ctrl[]	fir_ctrl_t	Management data for FIR filters (The data size of the total number of filters to be used x the number of measurement results is defined.)
g_filter_buffer	uint16_t	Data buffer for reading measurement result data from R_CTSU_DataGet API and passing it to the r_ctsu_filter_exec API

2.3.3 Structures

Table 2.7 to Table 2.11 show the number of filters and the types of filters defined in the filter_config_sample.c.

Table 2.7 Filter Configuration Definition

Definition content	Data type	Remarks
Definition for filter management	ctsu_filter_instance_t	Prepare for each method of touch interface configuration.
Filter management data	filter_ctrl_t	
Filter configuration definition	filter_config_t	To change the filter contents to be used for each method in the Touch Interface configuration, prepare a definition for each filter content.
Filter content definition	filter_element_config_t	

Table 2.8 Structures for Defining Filter Management

Member	Data type	Description
p_ctrl	filter_ctrl_t *	Filter management data pointer
p_cfg	filter_config_t const *	Filter configuration definition pointer
p_pia	filter_api_t const *	Filtering API pointer

Table 2.9 Structures for Filter Management Data

Member	Data type	Description
element_num	uint16_t	Number of measurement results
p_cfg	filter_config_t const *	Filter configuration definition pointer
p_fir_ctrl	fir_ctrl_t *	FIR filter configuration definition pointer

Table 2.10 Structures for Defining Filter Configuration

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

Table 2.11 Structures for Defining Filter Contents

Member	Data type	Description
type	filter_type_t	Filter type
fir_cfg	fir_config_t const *	FIR filter configuration definition pointer

The following shows a sample description of a filter configuration definition.

- Description example of filter content definition (filter_element_config_t)

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

Define the filter types you want to use in the order in which you want them to apply.

- Description example of filter configuration definition (filter_config_t)

  Define the number of filters and filter type 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,
};
```

- Description example of filter management definition (ctsu_filter_instance_t)

```
filter_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 in the Touch Interface configuration.

2.4 Software Filter API

Table 2.12 and Table 2.13 show the software filter API implemented in this sample program. The software filter API executes the software filter specified in the file configuration definition.

Table 2.12 Filter Initialization API

API	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)	
Argument	filter_ctrl_t * const p_ctrl	Filter management data pointer
	filter_config_t const * const p_cfg	Filter configuration definition pointer
	ctsu_cfg_t const * const p_ctsu_cfg	CTSU configuration-definition pointers
Description	Assigns management data for the specified filter configuration definition and initializes the assigned management data. When using multiple touch interface configurations, it is necessary to prepare the same number of filter management data and configuration definitions and call this API. (When executing the FIR filter initialization API (r_ctsu_fir_open) from this API during mutual-capacitance measurement, the structure is such that the filter is initialized "number of transmission terminals x number of reception terminals x 2" times.)	

Table 2.13 Filter Execution API

API	r_ctsu_filter_exec(filter_ctrl_t * const p_ctrl, uint16_t *p_data)	
Argument	filter_ctrl_t * const p_ctrl	Filter management data pointer
	uint16_t *p_data	Measurement result data pointer
Description	Applies the filter defined in the filter configuration to the measurement result data. Since the result of applying the filter overwrites the contents of the measurement result data pointer, if the measurement result data before applying the filter is used for other purposes, it must be saved before executing this API.	

2.5 Size and Execution Time

Table 2.14 and Table 2.15 show the data sizes and execution times of filtering for one touch interface configuration (Button × 3, Slider × 1, Wheel × 1, and without shielded terminals) on a RA2L1 mounted capacitive touch evaluation system.

Table 2.14 Filter Processing Data Size and Increments

Conditions	Size [Bytes]		
	text	data	bss
Before adding filters	14728	16	2424
Filter management	+240	+0	+16
FIR filters (Direct Type) (Note)	+556	+0	+368
FIR filters (Transpose type) (Note)	+548	+0	+584

Note: It varies depending on the filter order of FIR filter.
The values shown are when the maximum order is defined.

Table 2.15 Filter Processing Execution Time

Conditions	Execution time (1ch)
FIR filters (Direct type)	13.206us
FIR filters (Transpose Type)	6.692us

Note: The execution time is shown in the case of the self-capacitance method. In the case of the mutual-capacitance method, the execution time is approximately doubled because of two measurements.

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.

$$y(n) = \sum_{i=0}^{T-1} h(i) * x(n-i)$$

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

Table 3.1 shows the specifications of FIR filters of this sample program.

Table 3.1 FIR Filters Specifications

Item	Specifications	Remarks
Input data type	Unsigned 16bit Integer Type	
Output Data Type	Unsigned 16bit Integer Type	
Coefficient data type	Signed 15bit fixed point	Internal operations are signed 32bit (Integer part 16bit, decimal part 14bit)
Maximum coefficient	8	The number of taps is indicated by "order + 1"
Filter processing method	• Direct type • Transpose type	Can be switched by conditional compilation (Refer to chapter 3.5.1)
Output results up to filter stabilization time	Output Zero	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 2.14 and Table 2.15.

Table 3.2 Sample FIR filtering specification

File	Definition name	Description

r_ctsu_fir_sample.h	FIR_FILTER_TYPE	Filter processing method
		FIR_FILTER_TYPE_DIRECT = Direct type
		FIR_FILTER_TYPE_TRANSPOSE = Transpose Type

3.3 FIR Filter API

Table 3.3 to Table 3.6 show FIR filter API implemented by this sample program.

Table 3.3 FIR Filter Initialization API

API	r_ctsu_fir_open	
Argument	fir_ctrl_t * const p_ctrl	FIR filter management data pointer
	fir_config_t const * const p_cfg	FIR filter configuration definition pointer
Description	Buffer assignment for one measurement result is performed. Buffer allocation is performed in units of maximum order (8) + 1 regardless of the order of the configuration definition. When using this API for mutual-capacitance measurement, perform this API twice as there are two measurements per measurement.	

Table 3.4 FIR Filter API

API	r_ctsu_fir_filter	
Argument	fir_ctrl_t * const p_ctrl	FIR filter management data pointer
	uint16_t *p_data	Measurement result data pointer
Description	Perform FIR filter operation. The process to be performed depends on the conditional compilation FIR_FILTER_TYPE.	

Table 3.5 Direct Type FIR Filter API

API	r_ctsu_fir_direct_filter	
Argument	fir_ctrl_t * const p_ctrl	FIR filter management data pointer
	uint16_t *p_data	Measurement result data pointer ※If the status is excessive (before inputting the order number), 0 is returned.
Description	Perform direct type FIR filter operation. Before inputting the data for the order, the filter operation result is attenuated with respect to the input. Therefore, the result is returned as 0 to prevent drift correction from malfunctioning. [Note] Calculations are performed using 16-bit unsigned integers for measurement values and 15-bit signed fixed-point numbers for coefficients.	

Table 3.6 Transpose Type FIR Filter API

API	r_ctsu_fir_transpose_filter	
Argument	fir_ctrl_t * const p_ctrl	FIR filter management data pointers
	uint16_t *p_data	Measurement result data pointer ※If the status is excessive (before inputting the order number), 0 is returned.
Description	Perform transpose type FIR filter operation. Before inputting the data for the order, the filter operation result is attenuated with respect to the input. Therefore, the result is returned as 0 to prevent drift correction from malfunctioning. [Note] Calculations are performed using 16-bit unsigned integers for measurement values and 15-bit signed fixed-point numbers for coefficients.	

Figure 3.1 shows FIR filter process.

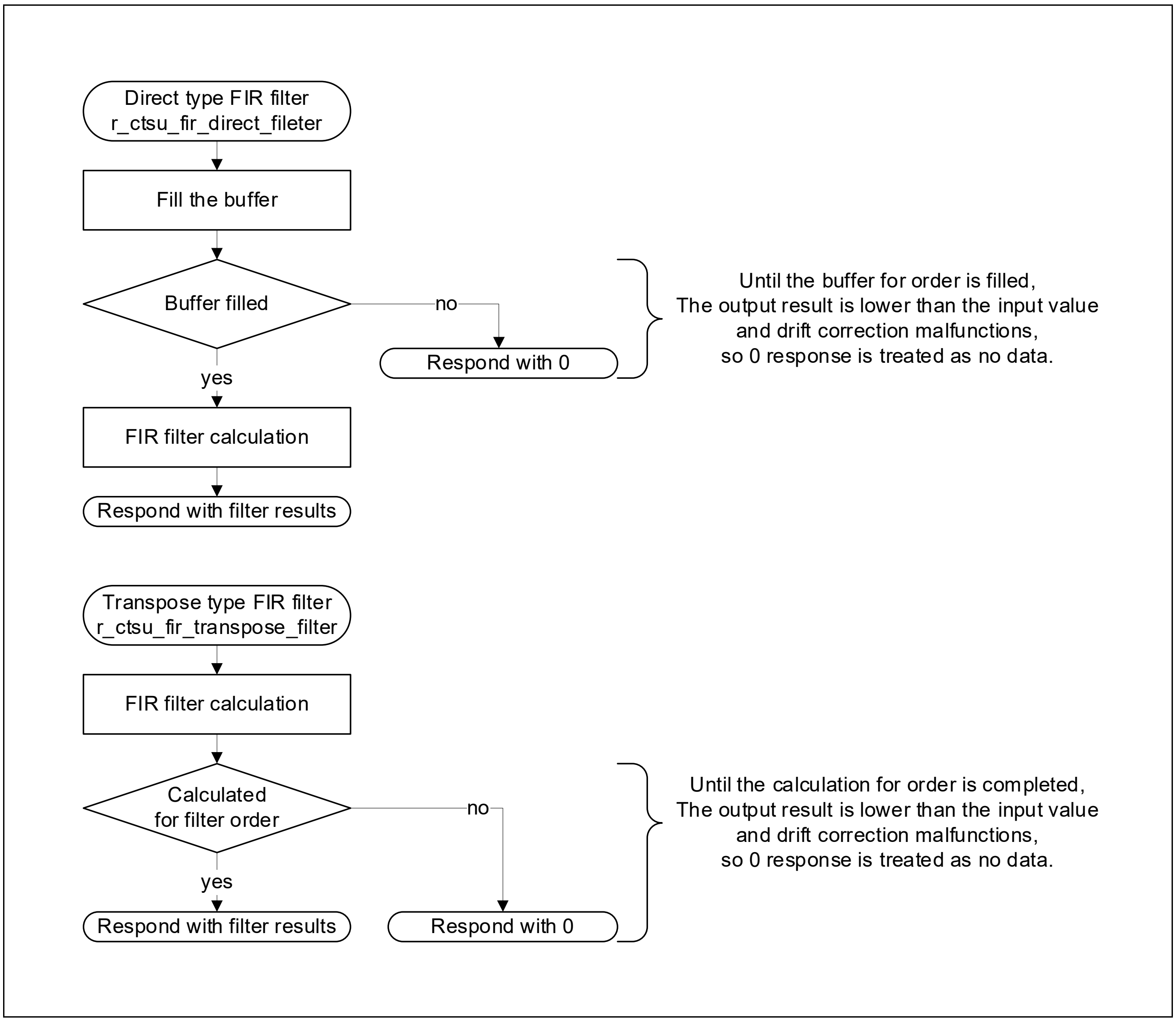

Direct type FIR filter
r_ctsu_fir_direct_fileter
Fill the buffer
Buffer filled
no
yes
Respond with 0
Until the buffer for order is filled,
The output result is lower than the input value
and drift correction malfunctions,
so 0 response is treated as no data.
FIR filter calculation
Respond with filter results
Transpose type FIR filter
r_ctsu_fir_transpose_filter
FIR filter calculation
Calculated
for filter order
no
yes
Respond with filter results
Respond with 0
Until the calculation for order is completed,
The output result is lower than the input value
and drift correction malfunctions,
so 0 response is treated as no data.

Figure 3.1 Flowchart of FIR Filter Process

3.4 List of Data for FIR Filters

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

3.4.1 Constants

Table 3.7 lists the constants.

Table 3.7 Constants for FIR Filter

Data	Value	Description
TAP_SIZE_MAX	9	Maximum number of taps
FIR_CFG_DECIMAL_POINT	14	Fixed-point number of digits
FIR_FILTER_SIZE	FIR_FILTER_NUM × FILTER_ELEMENT_SIZE	Buffer size for FIR filters (Calculated from the number of taps and the number of measurement results.)
MAX_COEFFICIENT_SUM	0x8000	Maximum value of the coefficient sum
MAX_COEFFICIENT_PLUS	0x3FFF	Maximum value of the coefficient value
MIN_COEFFICIENT_MINUS	0xC000	Minimum value of the coefficient value

3.4.2 Global Variables

Table 3.8 lists the global variables.

Table 3.8 Global Variables for FIR Filters

Data	Data type	Description
g_ctsu_fir_element_index	uint16_t	Buffer allocation management index
g_ctsu_fir_buffer[]	uint16_t	FIR filter buffers 16bit length for direct type, 32bit length 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) ※Number of mutual-capacitance electrodes: Number of transmitting electrodes × Number of receiving electrodes
	uint32_t	

3.5 Filter Adjustment Procedure

You can change the coefficient definition of FIR filters and adjust the filter properties.

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 2.14 and Table 2.15.

Figure 3.2 shows a block diagram of FIR filter.

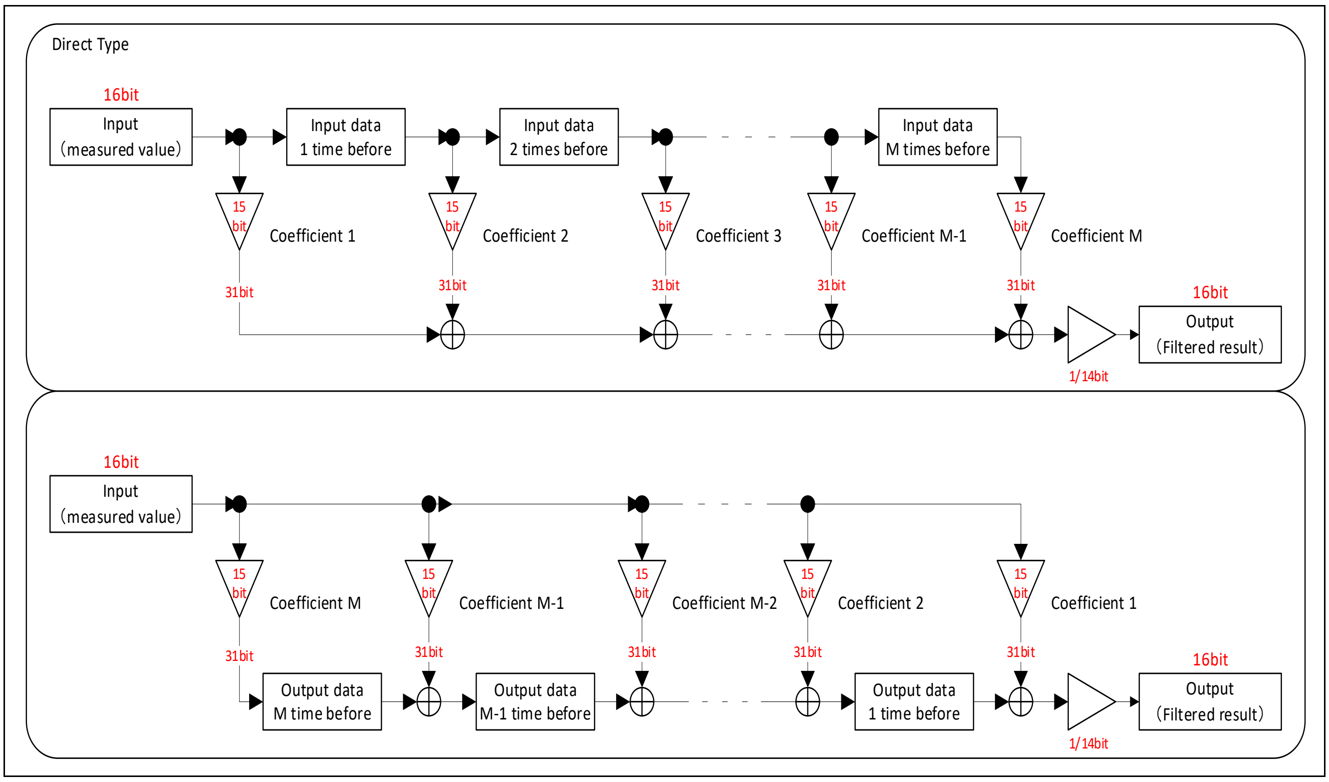


Figure 3.2 Block Diagram of FIR Filter

3.5.2 Filter Characteristics

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

Table 3.9 defines the characteristics of sample FIR filters. The filter characteristics can be changed by specifying the coefficient and filter configuration definitions shown in Table 3.10.

Table 3.9 Sample FIR Filters Specification

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

Table 3.10 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

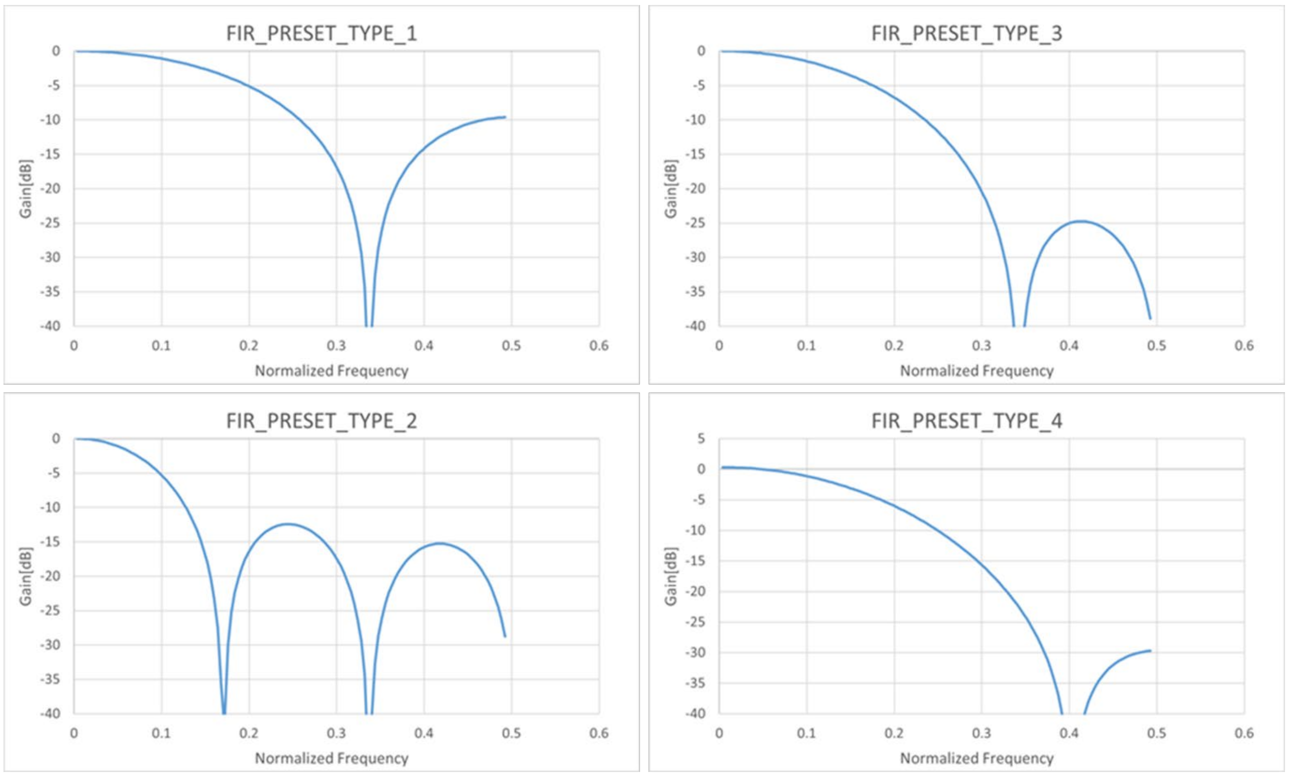


Figure 3.3 Sample Preset Filters Characteristics

3.5.3 Coefficient Definition

The coefficients of FIR filter configuration are defined in the form of signed fixed decimal with no integral part and the lower 14bit as the decimal part and are treated as the coefficient value/16384.

The coefficient 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, so operation errors occur.

Table 3.11 shows examples of decimal, hexadecimal, and decimal correspondence.

Table 3.11 Fixed Point Definition Example

Fractional number	Hex	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 0th order.

The number of taps is 1 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 tables 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. Operation explanation of this sample project

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

4.1 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.

4.2 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
|
├─QE-Touch
|        ├ qe_tuning20230221103059.log       ・・・QE Tuning Log
|        └ quickstart_rssk_ra2l1_ep.tifcfg   ・・・Touch Interface Configuration File
|
├─qe_gen
|        ├ qe_touch_config.c                 ・・・Source for Touch Configuration
|        ├ qe_touch_config.h                 ・・・Header for Touch Configuration
|        ├ qe_touch_define.h                 ・・・Header for Touch Definition
|        └ qe_touch_sample.c                 ・・・Touch Sample Application
|
├─src
|        ├ hal_entry.c                       ・・・main Files
|        ├ r_rssk_switch_led.c               ・・・Source for Switch and LED processing
|        ├ r_rssk_switch_led.h               ・・・Header for Switch and LED processing
|        ├ r_rssk_touch_led.c                ・・・Source for Touch electrode LED processing
|        └ r_rssk_touch_led.h                ・・・Header for Touch electrode LED processing
|
└─filter_sample
         ├ filter_config_sample.c            ・・・Source for Filter Configuration Definition
         ├ filter_config_sample.h            ・・・Header for Filter Configuration Definition
         ├ fir_config_sample1.c              ・・・Source for FIR Filter Sample Preset 1
         ├ fir_config_sample2.c              ・・・Source for FIR Filter Sample Preset 2
         ├ fir_config_sample3.c              ・・・Source for FIR Filter Sample Preset 3
         ├ fir_config_sample4.c              ・・・Source for FIR Filter Sample Preset 4
         ├ r_ctsu_filter_sample.c            ・・・Source for Filter processing
         ├ r_ctsu_filter_sample.h            ・・・Header for Filter processing
         ├ r_ctsu_fir_sample.c               ・・・Source for FIR filter processing
         └ r_ctsu_fir_sample.h               ・・・Header for FIR Filter processing
```

4.3 How to use This Sample Project

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

Figure 4.1 shows how to import a sample project.

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

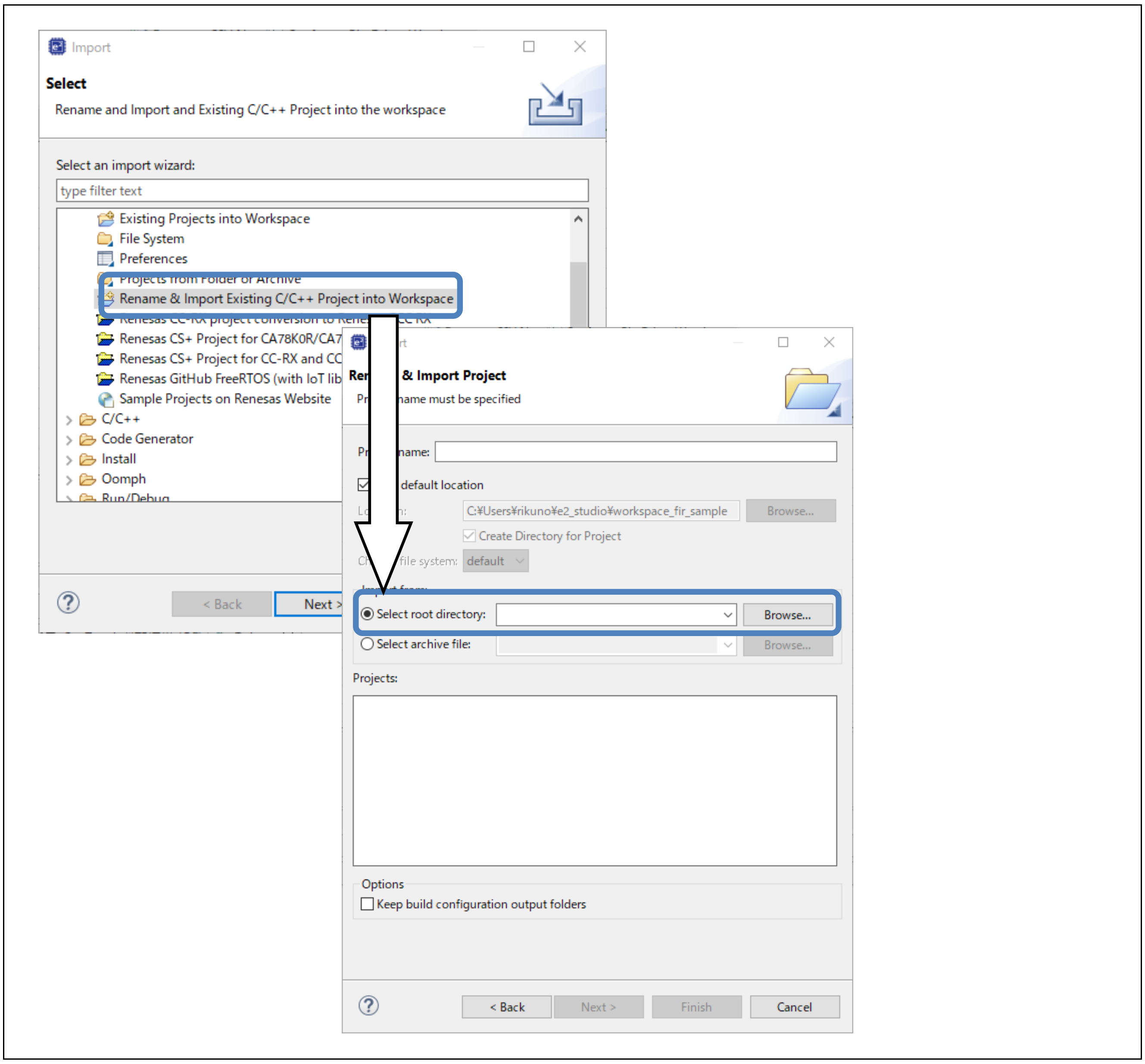

Import
Select
Rename and Import and Existing C/C++ Project into the workspace
Select an import wizard:
type filter text
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
Rename & Import Existing C/C++ Project into Workspace
Renesas CC-RX project conversion to Renesas CC-RX
Renesas CS+ Project for CA78K0R/CA7
Renesas CS+ Project for CC-RX and CC
Renesas GitHub FreeRTOS (with IoT lib
Sample Projects on Renesas Website
C/C++
Code Generator
Install
Oomph
Run/Debug
< Back
Next >
& Import Project
name must be specified
name:
default location
C:¥Users¥rikuno¥e2_studio¥workspace_fir_sample
Browse...
Create Directory for Project
file system:
default
Select root directory:
Browse...
Select archive file:
Browse...
Projects:
Options
Keep build configuration output folders
< Back
Next >
Finish
Cancel

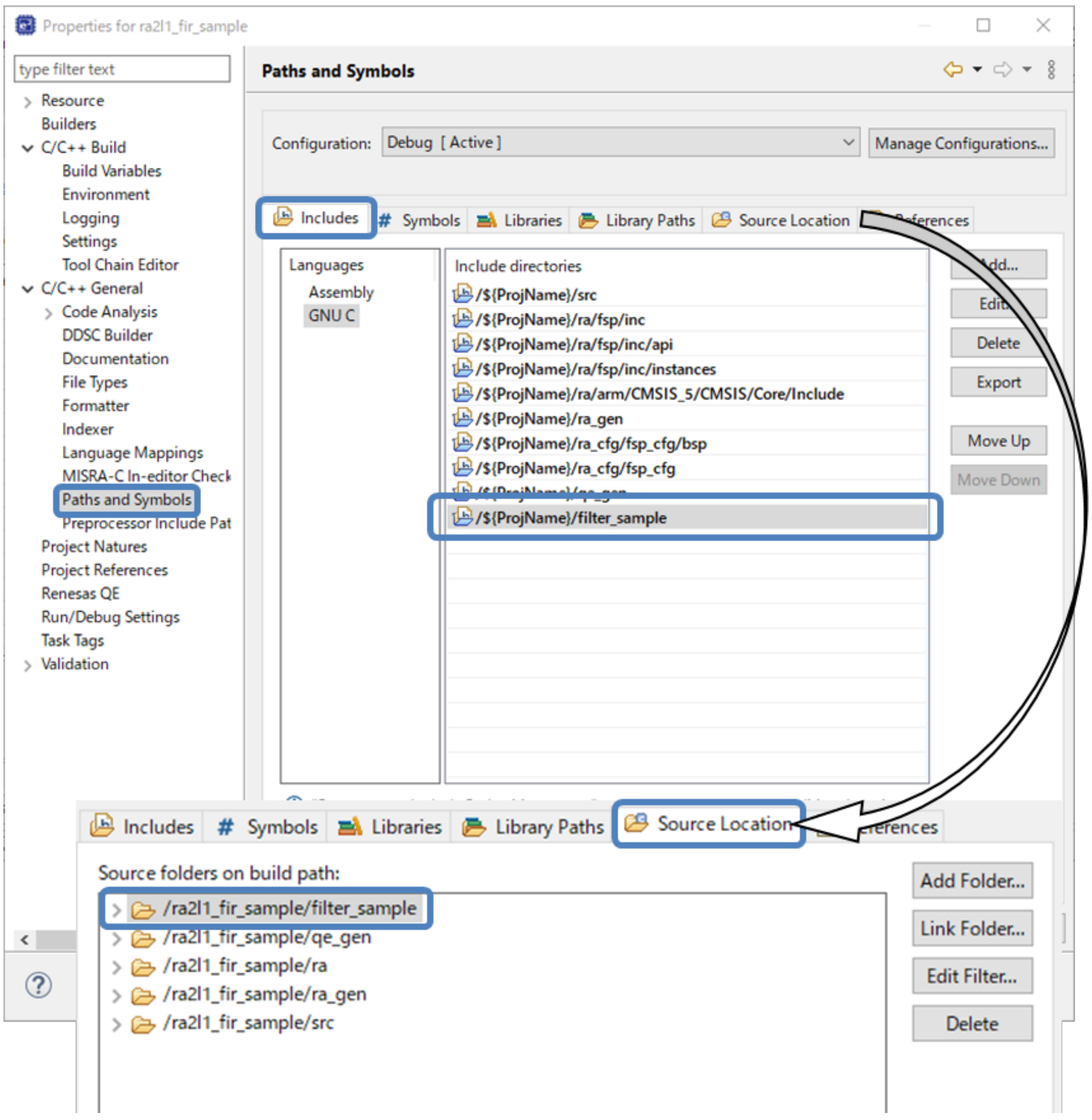
Figure 4.1 Importing the Sample Project

4.4 Procedure for Integration into an Existing Project

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

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 add the filter_sample folder to "Paths and Symbol Include and Source Locations" in C/C++ General.


Properties for ra2l1_fir_sample

Paths and Symbols

Configuration: Debug [Active]

Include directories:
- /${ProjName}/src
- /${ProjName}/ra/fsp/inc
- /${ProjName}/ra/fsp/inc/api
- /${ProjName}/ra/fsp/inc/instances
- /${ProjName}/ra/arm/CMSIS_5/CMSIS/Core/Include
- /${ProjName}/ra_gen
- /${ProjName}/ra_cfg/fsp_cfg/bsp
- /${ProjName}/ra_cfg/fsp_cfg
- /${ProjName}/filter_sample

Source folders on build path:
- /ra2l1_fir_sample/filter_sample
- /ra2l1_fir_sample/qe_gen
- /ra2l1_fir_sample/ra
- /ra2l1_fir_sample/ra_gen
- /ra2l1_fir_sample/src

Figure 4.2 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.

- 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 configurations definitions

- Filter_config_sample.c

```
filter_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,
};

filter_ctrl_t g_ctsu_filter_control02;
const ctsu_filter_instance_t g_ctsu_filter_instance02 =
{
  .p_ctrl= &g_ctsu_filter_control02,
  .p_cfg  = &g_ctsu_filter_config,
  .p_api  = &g_filter_on_ctsu,
};

filter_ctrl_t g_ctsu_filter_control03;
const ctsu_filter_instance_t g_ctsu_filter_instance03 =
{
  .p_ctrl= &g_ctsu_filter_control03,
  .p_cfg  = &g_ctsu_filter_config,
  .p_api  = &g_filter_on_ctsu,
```

4. Modify the filter configuration definition in filter_config_sample.c according to your environment and specify the filter to be applied. (See chapter 2.3.3.)
   For FIR filter, you can specify filter characteristics from a 4-pattern sample preset in the conditional compilation FIR_PRESET_TYPE.

- 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,
   .fir_cfg = &fir_cfg01,
  },
#endif
#if (FIR_PRESET_TYPE == FIR_PRESET_TYPE_2)
  {
   .type  = FILTER_TYPE_FIR,
   .fir_cfg = &fir_cfg02,
  },
#endif
#if (FIR_PRESET_TYPE == FIR_PRESET_TYPE_3)
  {
   .type  = FILTER_TYPE_FIR,
   .fir_cfg = &fir_cfg03,
  },
#endif
#if (FIR_PRESET_TYPE == FIR_PRESET_TYPE_4)
  {
   .type  = FILTER_TYPE_FIR,
   .fir_cfg = &fir_cfg04,
  },
#endif
};
```

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 4.5).

[Note] 1. Note that data reading and data writing back for filtering are not touch API, but CTSU drivers.
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. When FIR filters are applied, they do not work properly unless the averaging process is disabled.
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,
};
```

4.5 Sample Application Configuration and Operation

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

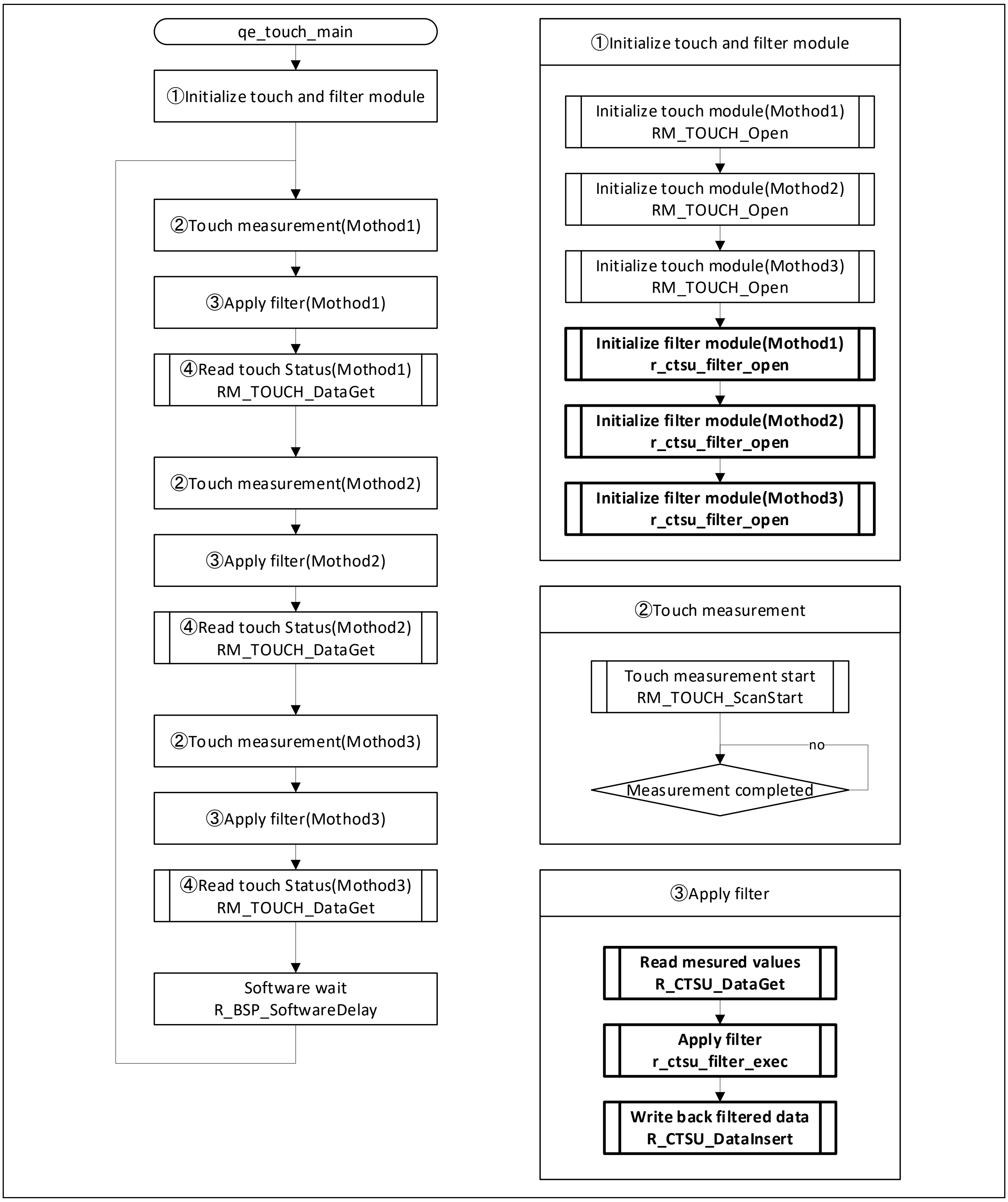

qe_touch_main
①Initialize touch and filter module
②Touch measurement(Mothod1)
③Apply filter(Mothod1)
④Read touch Status(Mothod1) RM_TOUCH_DataGet
②Touch measurement(Mothod2)
③Apply filter(Mothod2)
④Read touch Status(Mothod2) RM_TOUCH_DataGet
②Touch measurement(Mothod3)
③Apply filter(Mothod3)
④Read touch Status(Mothod3) RM_TOUCH_DataGet
Software wait R_BSP_SoftwareDelay
①Initialize touch and filter module
Initialize touch module(Mothod1) RM_TOUCH_Open
Initialize touch module(Mothod2) RM_TOUCH_Open
Initialize touch module(Mothod3) RM_TOUCH_Open
Initialize filter module(Mothod1) r_ctsu_filter_open
Initialize filter module(Mothod2) r_ctsu_filter_open
Initialize filter module(Mothod3) r_ctsu_filter_open
②Touch measurement
Touch measurement start RM_TOUCH_ScanStart
no
Measurement completed
③Apply filter
Read mesured values R_CTSU_DataGet
Apply filter r_ctsu_filter_exec
Write back filtered data R_CTSU_DataInsert

Figure 4.3 Sample Application Flow

This section describes the numbers in the figure in Figure 4.3.

The code that you add to the qe_touch_main in “qe_touch_sample.c” are shown in bold.

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(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    If (FSP_SUCCESS != err)
    {
        While (true) {}
    }
    Err = RM_TOUCH_Open(g_qe_touch_instance_config02.p_ctrl, g_qe_touch_instance_config02.p_cfg);
    If (FSP_SUCCESS != err)
    {
        While (true) {}
    }
    Err = RM_TOUCH_Open(g_qe_touch_instance_config03.p_ctrl, g_qe_touch_instance_config03.p_cfg);
    If (FSP_SUCCESS != err)
    {
        While (true) {}
    }

  /* 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) {}
    }
  Err = r_ctsu_filter_open(g_ctsu_filter_instance02.p_ctrl, g_ctsu_filter_instance02.p_cfg,
g_qe_ctsu_instance_config02.p_cfg);
    If (FSP_SUCCESS != err)
    {
        While (true) {}
    }
  Err = r_ctsu_filter_open(g_ctsu_filter_instance03.p_ctrl, g_ctsu_filter_instance03.p_cfg,
g_qe_ctsu_instance_config03.p_cfg);
    If (FSP_SUCCESS != err)
    {
        While (true) {}
    }
```

② Touch measurement

Perform touch measurement and wait for measurement to be completed.

② to ④ should be executed consecutively for each method of the touch interface configuration.

③ Filter application

After completing the touch measurement, use the CTSU driver API to get the measurement result and write it back to the CTSU driver after applying the filter.

To use the CTSU driver API, it is necessary to specify the CTSU driver configuration definition and set management data specified at filter initialization. (The content is changed from .p_cfg in the CTSU driver configuration definition to .p_ctrl)

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

② to ④ 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).

```
Static uint16_t g_filter_buffer[CTSU_CFG_NUM_SELF_ELEMENTS];
Void qe_touch_main(void)
{

  (Omitted)

  /* 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);

  (Omitted)

        /* 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);
        }
```

④ Touch status acquisition

Get touch input information using filtered data.

② to ④ should be executed consecutively for each method of the touch interface configuration.

5. Supports

See the following website for information on capacitive touch, downloading tools and documentation, and technical support.

Capacitive Sensor MCU Capacitive Touch Noise Immunity Guide (R30AN0426)

https://www.renesas.com/node/25428131

Capacitive Touch Evaluation System for RA2L1 (RTK0EG0022S01001BJ)

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

Renesas RA Family RA2L1 Group Capacitive Touch Evaluation System Quick Start Guide (Q12QS0040)

https://www.renesas.com/node/1403601

RA Family Using QE and FSP to Develop Capacitive Touch Applications (R01AN4934)

https://www.renesas.com/node/1289806

QE for Capacitive Touch: Development Assistance Tool for Capacitive Touch Sensors

renesas.com/qe-capacitive-touch

Renesas support

renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun.12.23	-	First edition issued

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

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/.