Capacitive Sensor MCU

Capacitive Touch Software Filter Sample Program

Introduction

This application note describes software filters in for capacitive touch systems.

Target Device

RA2L1 Group (R7FA2L1AB2DFP)

RX130 Group (R5F51305ADFN)

RL78/G16 Group (R5F121BCAFP)

Contents

1. Overview

This application note describes the operations of the software filter sample program and the steps required for embedding it into an existing project.

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

1.1 Operation Confirmation Conditions

Table 1-1 to Table 1-3 list the operation confirmation conditions of the sample program in this application note.

Table 1-1 Operation Confirmation Conditions (RA2L1 Group)

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	e² studio Version 2024-04 (24.4.0)
C compiler	GCC Arm Embedded 13.2.1.arm-13-7
FSP	V5.3.0
Development Assistance Tool for Capacitive Touch Sensors	QE for Capacitive Touch V3.3.0
Emulator	Renesas E2 emulator Lite

Table 1-2 Operation Confirmation Conditions (RX130 Group)

Item	Description
Microcontroller used	RX130 (R5F51305ADFN)
Operating frequency	High-speed on-chip oscillator 32MHz
Operating voltage	5V
Board	Capacitive Touch Evaluation System with RX130 (Model: RTK0EG0003S02001BJ) ● RX130 CPU Board (Model: RTK0EG0004C01002BJ) ● Self-Capacitance Touch Button/Wheel/Slider Board (Model: RTK0EG0007B01002BJ)
Integrated development environment	e² studio Version 2024-04 (24.4.0)
C compiler	Renesas CC-RX V3.06.00
Development Assistance Tool for Capacitive Touch Sensors	QE for Capacitive Touch V3.3.0
Emulator	Renesas E2 emulator Lite

Table 1-3 Operation Confirmation Conditions (RL78/G16 Group)

Item	Description
Microcontroller used	RL78/G16 (R7F100GSN2DFB)
Operating frequency	High-speed on-chip oscillator 32MHz
Operating voltage	5V
Board	Capacitive Touch Evaluation System with RL78/G16 (Model: RTK0EG0047S01001BJ) • RL78/G16 CPU Board (Model: RTK0EG0046C01001BJ) • Self-Capacitance Touch Button/Wheel/Slider Board (Model: RTK0EG0019B01002BJ)
Integrated development environment	e² studio Version 2024-04 (24.4.0)
C compiler	Renesas CC-RL V1.13.00
Development Assistance Tool for Capacitive Touch Sensors	QE for Capacitive Touch V3.3.0
Emulator	Renesas E2 emulator Lite

Figure 1-1 shows the device connection diagram.

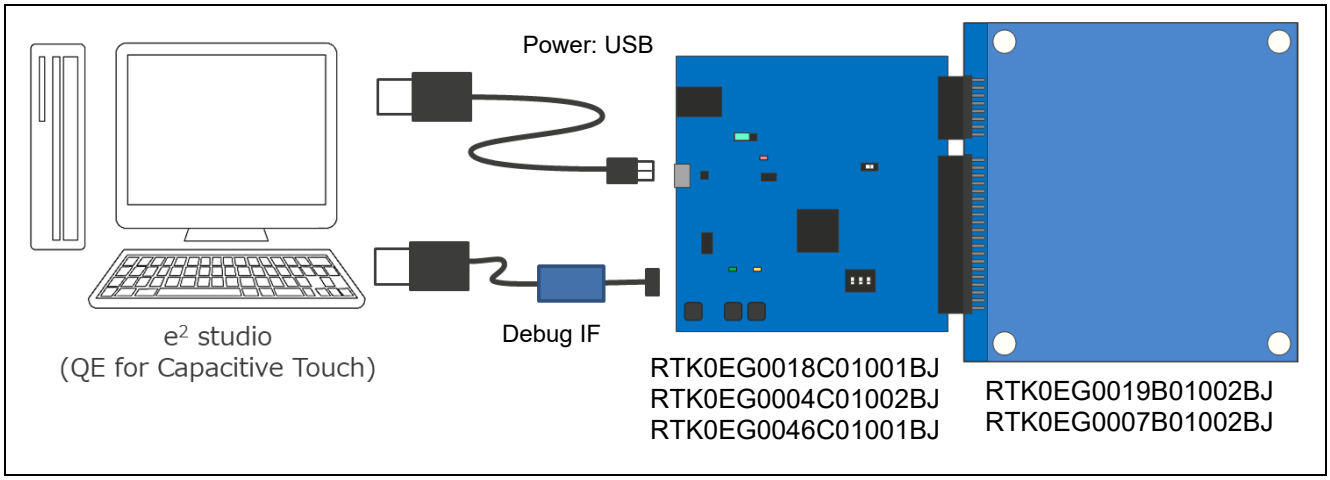


Figure 1-1 Device Connection Diagram

1.2 Correspondence Between Sample Code and Application Note

Please review sections 2.1 **Software Configuration Diagram** and 2.3 **File Structure** before using this sample project.

Also review 7.2 **How to Use the Filter Sample Code** for details on how to embed the filter module into an existing capacitive touch project and 7.1 **How to Use the Sample Project** for more details on how to use the sample project.

Filter specifications and parameter setting methods are described in 2 **Software Specifications**, 3 **FIR Filters**, 4 **IIR Filters**, 5 **Single-pole IIR Filters**, 6 **Median Filters**, and 7.2.3 **How to Adjust Filter Characteristics**

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. The software filters used in the program are managed by the filter API to be executed and the filter configuration definition data. The filter configuration described in the sample program consists of an FIR filter, IIR filter, and single-pole IIR filter, and Median filter.

The sample program example used in this document uses 1 button and touch detection using 5 types of filters. As an example of how to use the sample program, we provide a sample project that uses one button and performs touch detection applying five types of filters.

2.1 Software Configuration Diagram

Figure 2-1 shows the flow of data processing for this sample program.

Applies a software filter to the measurement data acquired by CTSU Driver (R_CTSU_DataGet()). To use the filtered measurement data with TOUCH Middleware, use the CTSU Driver (R_CTSU_DataInsert()) to replace the measurement data with the filtered measurement data. Use the filtered measurement data to perform touch judgment with TOUCH Middleware (RM_TOUCH_DataGet()).

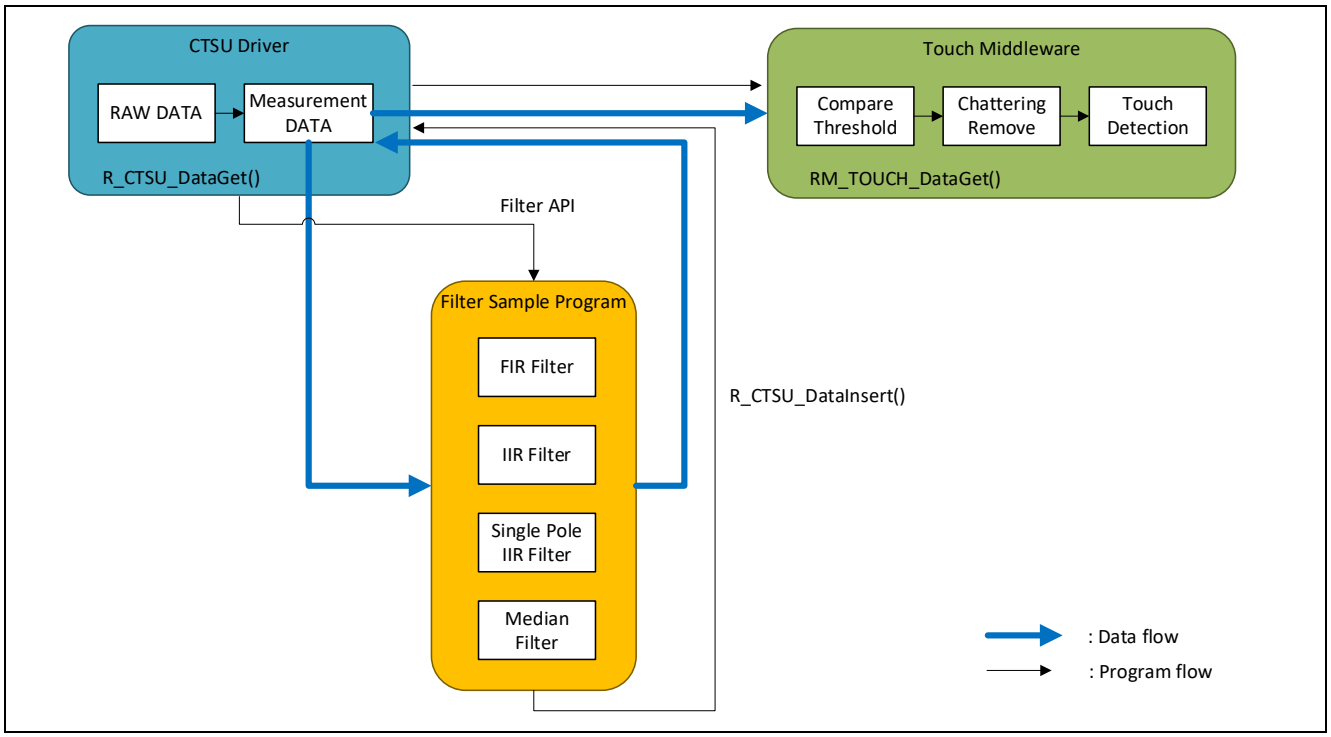


Figure 2-1 Data Processing Flow of Sample Program

Table 2-1 to **Table 2-3** list the components and version for each device group. Refer to FSP Configuration for component settings.

Table 2-1 List of Components (RA2L)

Component	Version
Board Support Packages Common Files	v5.3.0
I/O Port	v5.3.0
Arm CMSIS Version 5 – Core(M)	v5.9.0+renesas.1.fsp.5.3.0
RA2L1-RSSK Board Support Files	v5.3.0
Board support package for R7FA2L1AB2DFP	v5.3.0
Board support package for RA2L1	v5.3.0
Board support package for RA2L1 – FSP Data	v5.3.0
Asynchronous General Purpose Timer	v5.3.0
Capacitive Touch Sensing Unit	v5.3.0
Touch	v5.3.0

Table 2-2 List of Components (RX130)

Component	Version
Board Support Packages	v7.42
CMT driver	v5.60
CTSU QE API	v2.20
Touch QE API	v2.20

Table 2-3 List of Components (RL78/G16)

Component	Version
Board Support Packages	v1.62
Capacitive Touch Sensing Unit driver	v1.40
Touch middleware	v1.40
UART communication	v1.6.0
Interval timer	v1.4.0

2.2 Software Filter Types

Table 2-4 lists the types of filters used in the sample program as well as usage examples.

Table 2-4 Filter Types

Filter Type	Usage Example	Expected Noise
FIR filter	FIR moving-average filter	White noise, high-frequency noise, humming noise (50Hz)
IIR filter	IIR notch filter	Humming noise (60Hz)
	IIR low-pass filter	Humming noise (60Hz)
Single-pole IIR filter	Single-pole IIR low-pass filter	
Median filter	Median filter	

2.3 File Structure

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

```
an-r01an0427jj0400-capacitive-touch
├──filter_sample
│      ├──fir                                  ・・・FIR Filter Sample Module Storage Folder
│      │      fir_config_sample1.c     ・・・FIR Moving-average Filter Configuration Definition Sample Source
│      │      r_ctsu_fir_sample.c            ・・・FIR Filter Sample Program Source
│      │      r_ctsu_fir_sample.h            ・・・FIR Filter Sample Program Source Header
│      ├──iir                                  ・・・IIR Filter Sample Module Storage Folder
│      │      iir_config_sample1.c     ・・・IIR Notch/Low-pass Filter Configuration Definition Sample Source
│      │      r_ctsu_iir_sample.c            ・・・IIR Filter Sample Program Source
│      │      r_ctsu_iir_sample.h            ・・・IIR Filter Sample Program Source Header
│      ├──spiir                                ・・・Single-pole IIR Filter Sample Module Storage Folder
│      │      spiir_config_sample1.c・・・Single-pole IIR low-pass Filter Configuration Definition Sample Source
│      │      r_ctsu_spiir_sample.c          ・・・Single-pole IIR Filter Sample Program Source
│      │      r_ctsu_spiir_sample.h          ・・・Single-pole IIR Filter Sample Program Source Header
│      └──median                               ・・・Median Filter Sample Module Storage Folder
│             median_config_sample1.c         ・・・Median Filter Configuration Definition Sample Source
│             r_ctsu_median_sample.c          ・・・Median Filter Sample Program Source
│             r_ctsu_median_sample.h          ・・・Median Filter Sample Program Source Header
├──ra2l1_filter_sample                         ・・・RA2L1 Sample Project
├──rx130_filter_sample                         ・・・RX130 Sample Project
└──rl78g16_filter_sample                       ・・・RL78/G16 Sample Project
```

2.3.1 Application Data

This section describes the constants and global variables that are used in the sample project application.

Table 2-5 Sample Application Constants

Constant name	Value	Description
File name: qe_touch_sample.c		
TOUCH_SCAN_CYCLE	((20 * 1000) / 100)	Touch measurement cycle (20ms/100us)

Table 2-6 Sample Application Global Variables

Variable name	Type	Description
File name: qe_touch_sample.c		
touch_cycle_count	uint16_t	Touch measurement cycle counter
touch_cycle_flag	volatile uint8_t	Touch measurement cycle elapse flag

2.3.2 Application API

Table 2-7 shows the application APIs implemented in the sample project.

Table 2-7 Application API

Function name	Process description
File name: qe_touch_sample.c	
qe_touch_main	main processing
timer0_callback	AGT interrupt callback (RA2L1 group)
r_timer_callback	Timer interrupt callback (RX130 group, RL78/G16 group9

2.4 Size and Execution Time

2.4.1 RA2L1 Group

Table 2-8 shows the data sizes for the filtering process and execution times of each filter API for the sample program (five touch interface configurations: Button x 1 with shielded pins).

Table 2-8 Filter Processing Data Size, Differences and Execution Time (RA2L1)

Conditions	Memory Size [Bytes]			Execution Time [us]
	text	data	bss	
Before adding filters	13648	16	3280	
FIR filters	+396	+0	+24	6.60
IIR filters	+252	+0	+16	3.59
Single-pole IIR filters	+220	+0	+8	1.88
Median filters	+224	+0	+16	2.11

2.4.2 RX130 Group

Table 2-9 shows the data sizes for the filtering process and execution times of each filter API for the sample program (five touch interface configurations: Button x 1).

Table 2-9 Filter Processing Data Size, Differences and Execution Time (RX130)

Conditions	Memory Size [Bytes]			Execution Time [us]
	RAMDATA	ROMDATA	PROGRAM	
Before adding filters	4750	2684	9984	
FIR filters	+20	+18	+267	5.52
IIR filters	+10	+12	+221	4.09
Single-pole IIR filters	+4	+6	+179	1.03
Median filters	+10	+0	+238	4.63

2.4.3 RL78/G16 Group

Table 2-10 shows the data sizes for the filtering process and execution times of each filter API for the sample program (five touch interface configurations: Button x 1).

Table 2-10 Filter Processing Data Size, Differences and Execution Time (RL78/G16)

Conditions	Memory Size [Bytes]			Execution Time [us]
	RAMDATA	ROMDATA	PROGRAM	
Before adding filters	1338	1318	12201	
FIR filters	+20	+14	+461	54.33
IIR filters	+10	+12	+486	53.46
Single-pole IIR filters	+0	+6	+300	17.84
Median filters	+10	+0	+361	18.96

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

FIR filters sample input values, output the results of the multiply-and-accumulate operation on the sampled data and defined coefficients, and can handle different filter characteristics, such as low-pass filters and band-pass filters, depending on the defined coefficient.

In this sample program, the FIR filter is treated as a moving average filter.

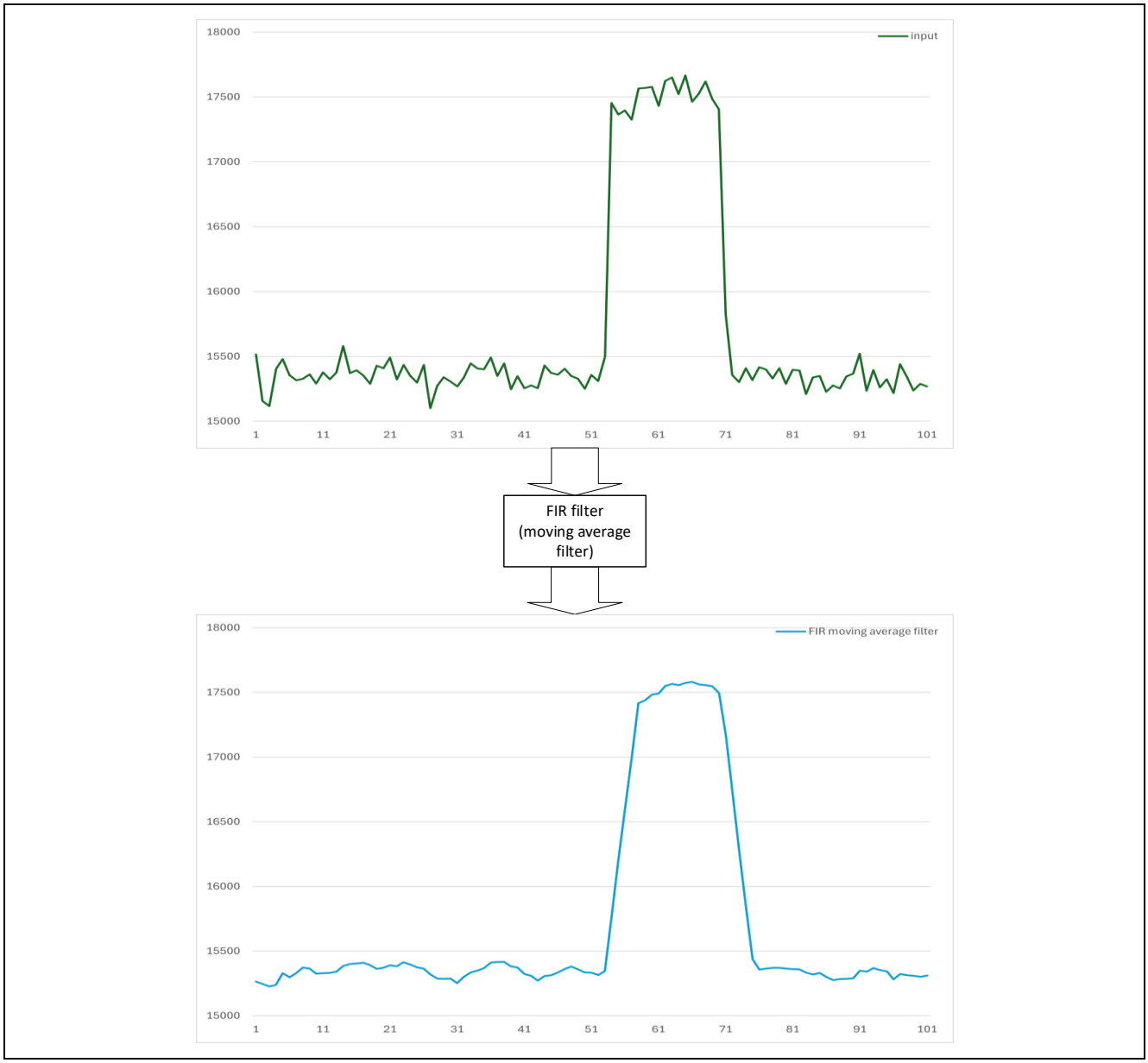


Figure 3-1 FIR Filter Operation (moving-average filter)

The FIR filter block diagram for the following FIR filter calculation is shown in Figure 3-2.

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

In the equation, *M* indicates the number of filter taps, n indicates the sample index, $h(\mathrm{m})$ indicates the coefficient, $x(n-\mathrm{m})$ indicates the input data of the m sample delay, and $y(n)$ indicates the output data.

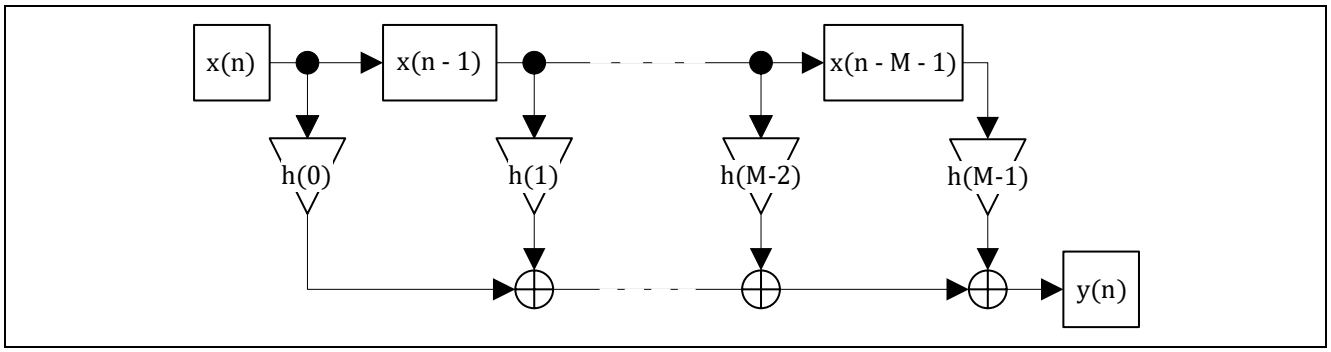


Figure 3-2 FIR Filter Block Diagram

3.1.1 Detection Delay

The FIR filter calculates the filter result from the sampled touch measurement values to remove noise signals, which causes a delay in normal touch detection.

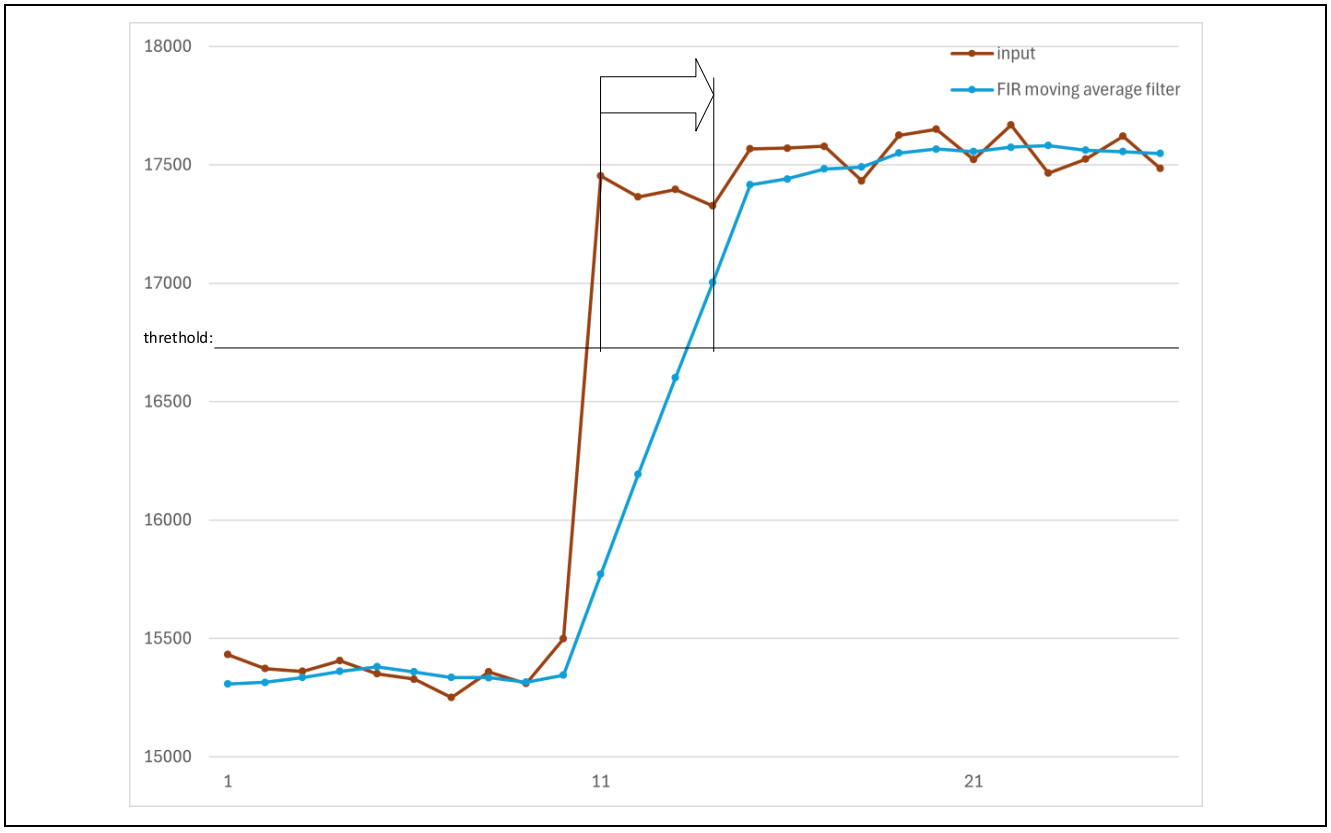


Figure 3-3 Detection Delay (FIR moving-average filter)

3.1.2 Filter Stabilization Time

After initialization, the FIR filter outputs calculation results that are lower than the input data until the filter processing is performed the number of times of sampled data.

If the filter processing result is used during this period for touch detection, the reference value for touch detection may be set lower than the original reference value, causing a continuous touch-ON state. Therefore, the filter calculation result during this period should be discarded (instead of calling RM_TOUCH_DataGet(), call RM_TOUCH_ScanStart() to start the next measurement).

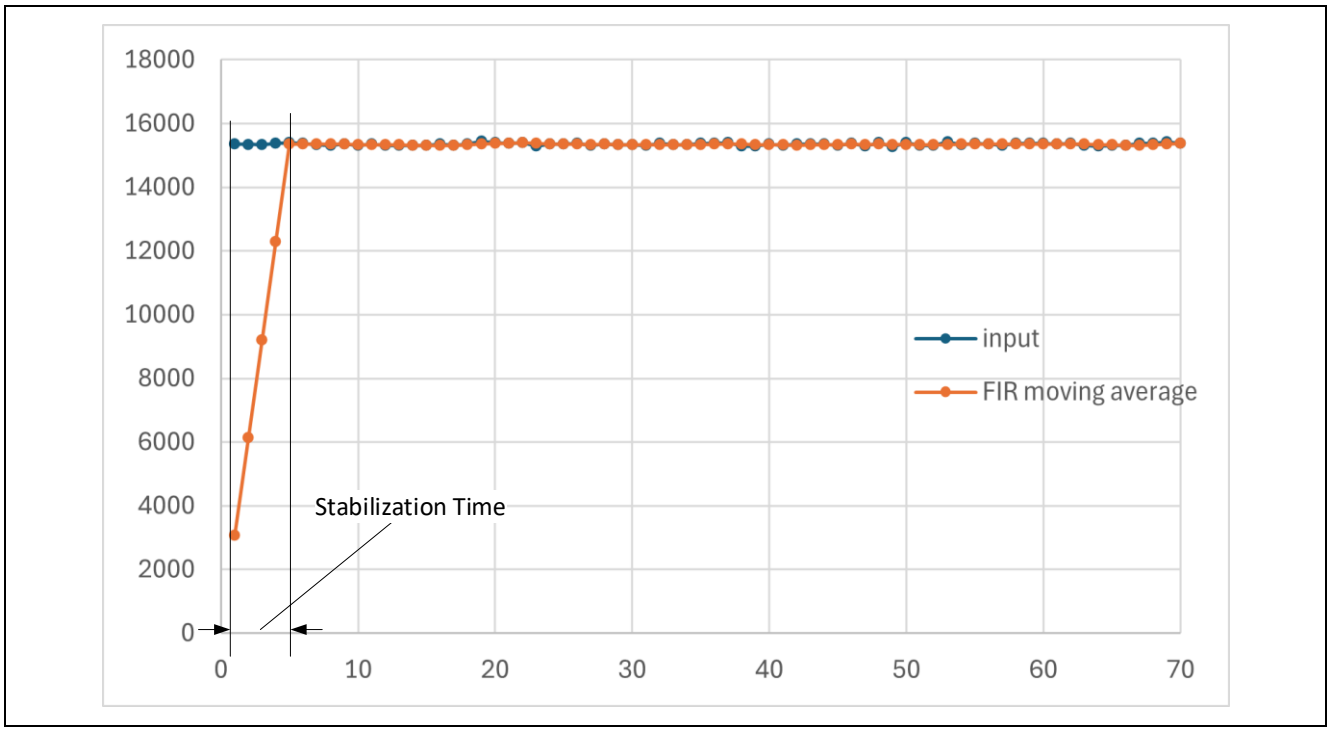


Figure 3-4 Filter Stabilization Time (FIR moving-average filter)

3.2 Filter Specifications

Table 3-1 lists the specifications of the FIR filters used in the sample program.

Table 3-1 FIR Filter Specifications

Item	Specification	Remarks
Input data type	Unsigned 16-bit integer type	
Output data type	Unsigned 16-bit integer type	
Coefficient data type	Signed 16-bit integer type	Fixed point with 14-bit decimal
Number of taps	1 < M ≦9	
Output results until filter stabilization time	Returns operation results during filter stabilization time and buffer unfilled response	Filter stabilization time = number of taps

Note: Coefficient: A set of constants to be applied to the constant multipliers that make up FIR filters.
Number of taps: Number of elements in the coefficient.

3.2.1 Filter Coefficient Usage Notes

FIR filter coefficient data must be defined within a value range of -1.0 to 1.0.

FIR filters must be defined so that the sum of the coefficient data is within a value range of -2.0 to 2.0.

3.3 List of Data for FIR Filters

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

3.3.1 Constants

Table 3-2 shows a list of constants for FIR filters.

Table 3-2 Constants for FIR Filters

Constant name	Setting value	Description
File name: r_ctsu_fir_sample.h		
FIR_TAP_SIZE_MAX	9	Maximum number of taps
FIR_DATA_SIZE	1	Data size to be filtered
File name: r_ctsu_fir_sample.c		
FIR_TAP_SIZE_MIN	2	Minimum number of taps
FIR_CFG_DECIMAL_POINT	14	Number of fixed-point digits
MAX_FIR_COEFFICIENT_SUM	0x00008000	Maximum value of sum of coefficients
MIN_FIR_COEFFICIENT_SUM	0xFFFF8000	Minimum value of sum of coefficients
MAX_FIR_COEFFICIENT	0x4000	Maximum value of coefficient
MIN_FIR_COEFFICIENT	0xC000	Minimum value of coefficient
FIR_RESULT_MAX	0x0000FFFF	Maximum value of filter result
FIR_RESULT_MIN	0x00000000	Minimum value of filter result

3.3.2 Structures

The following shows the management data structures for accessing the FIR filter APIs and the structures for defining the FIR filter configuration.

3.3.2.1 FIR Filter Configuration Definition (fir_config_t)

Table 3-3 FIR Filter Configuration Definition Structure (fir_config_t)

Member	Data type	Description
taps	uint16_t	Number of taps
p_coefficient	int16_t *	Pointer to FIR filter coefficient array Filter coefficients are stored in h0..hM-1 order

3.3.2.2 FIR filter management data (fir_ctrl_t)

Table 3-4 FIR Filter Management Data Structure (fir_ctrl_t)

Member	Data type	Description
count	uint16_t	Stabilization wait time counter
fir_data	uint16_t [FIR_DATA_SIZE][FIR_TAP_SIZE_MAX]	Delay buffer for FIR filter

3.4 FIR Filter API

3.4.1 r_ctsu_fir_initial

This function initializes the management data for FIR filter processing.

Make sure you execute this function before using r_ctsu_fir_filter.

Format

fsp_err_t r_ctsu_fir_initial(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_INVALID_ARGUMENT	/* Configuration definition parameters are invalid */

Properties

Protype is declared in r_ctsu_fir_sample.h.

Description

This function initializes the management data for FIR filter processing.

An error response is returned when the configuration definition parameters are not within valid range.

Example

```
err = r_ctsu_fir_initial(&g_ctsu_fir_control, &g_fir_cfg);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

3.4.2 r_ctsu_fir_filter

This function applies the FIR filter processing.

Format

fsp_err_t r_ctsu_fir_filter (fir_ctrl_t * const p_ctrl , fir_config_t const * const p_cfg , uint16_t *p_data);

Parameters

p_ctrl

FIR filter management data pointer

p_cfg

FIR filter configuration definition pointer

p_data

FIR filter measurement result data pointer

ReturnValues

FSP_SUCCESS /* Successfully completed */

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.

The result of the operation is limited to the range of unsigned 16-bit integers (65535 to 0); if it exceeds the range, it will be rounded to the upper or lower limit value.

After initialization, an error is returned until the processing for the number of taps indicated in the configuration definition is completed.

Example

```
/* FIR moving average filter sample */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config01.p_ctrl, filter_buffer);
if(err == FSP_SUCCESS)
{
 err=r_ctsu_fir_filter(&g_ctsu_fir_control,&g_fir_cfg, filter_buffer);
 if(err == FSP_SUCCESS)
 {
    R_CTSU_DataInsert(g_qe_ctsu_instance_config01.p_ctrl, filter_buffer);
  }
}
```

Special Notes:

Filter operations are performed even when an error response is received.

When configuring multiple filters in cascade, make sure that the filter processing in the next stage is executed even when an error response is received.

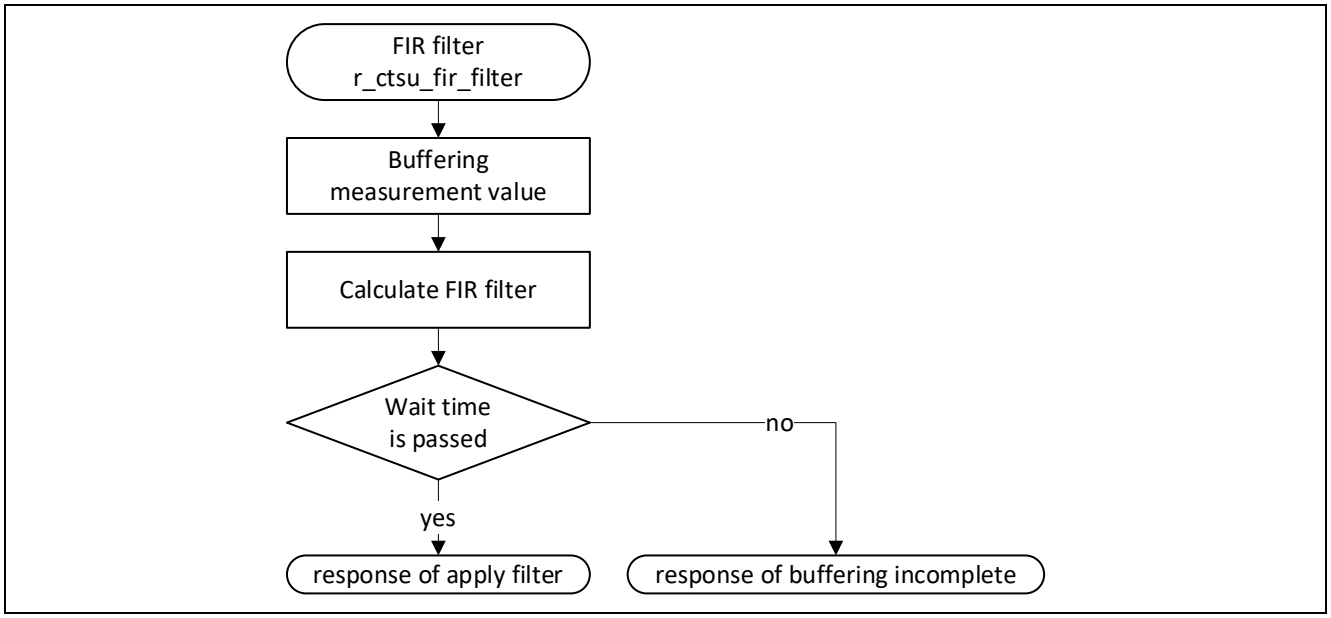


Figure 3-5 FIR Filter API Flowchart

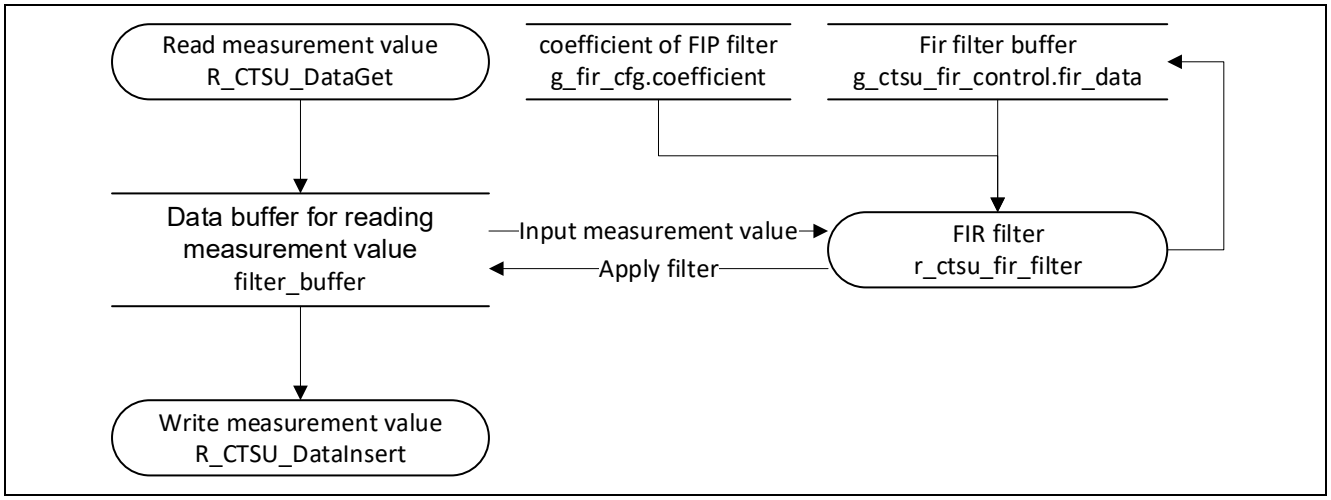


Figure 3-6 FIR Filter Data Flowchart

3.5 Usage Example

This sample program provides the FIR moving-average filter as an example of how to use FIR filters.

3.5.1 Filter Characteristics

Table 3-5 shows the coefficient definitions and characteristics of the sample FIR filter.

Table 3-5 Sample FIR Filter Coefficient Definition

	g_fir_cfg	
	FIR moving-average filter	
Number of taps	5	
Coefficient	3276 (0.199951171875)	
	3276 (0.199951171875)	
	3276 (0.199951171875)	
	3276 (0.199951171875)	
	3276 (0.199951171875)	

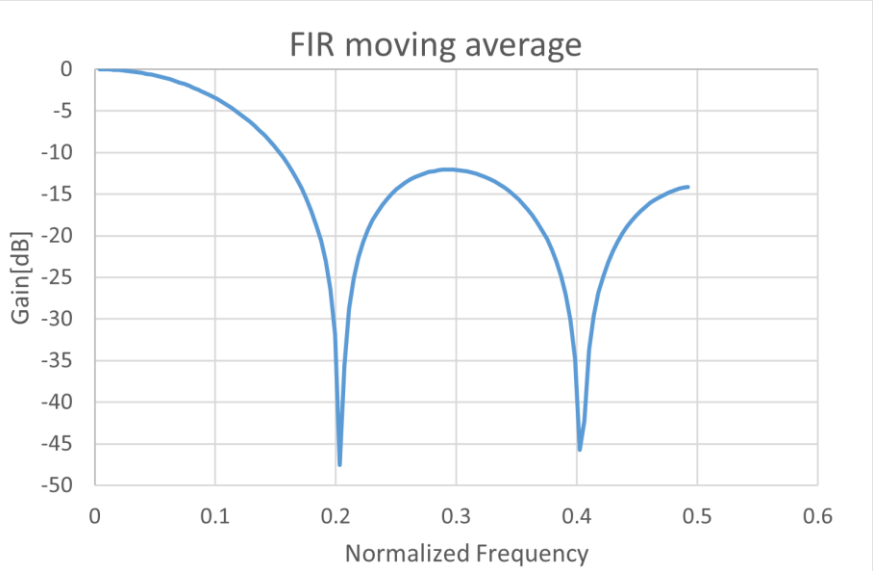


Figure 3-7 Sample FIR Filter Frequency Characteristics

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

IIR filters sample the input and output values, output the result of the multiply-and-accumulate operation on the sampled data and defined coefficients, and can handle different filter characteristics, such as low-pass filters and high-pass filters, depending on the defined coefficients.

This sample project implements notch filters and low-pass filters as IIR filters.

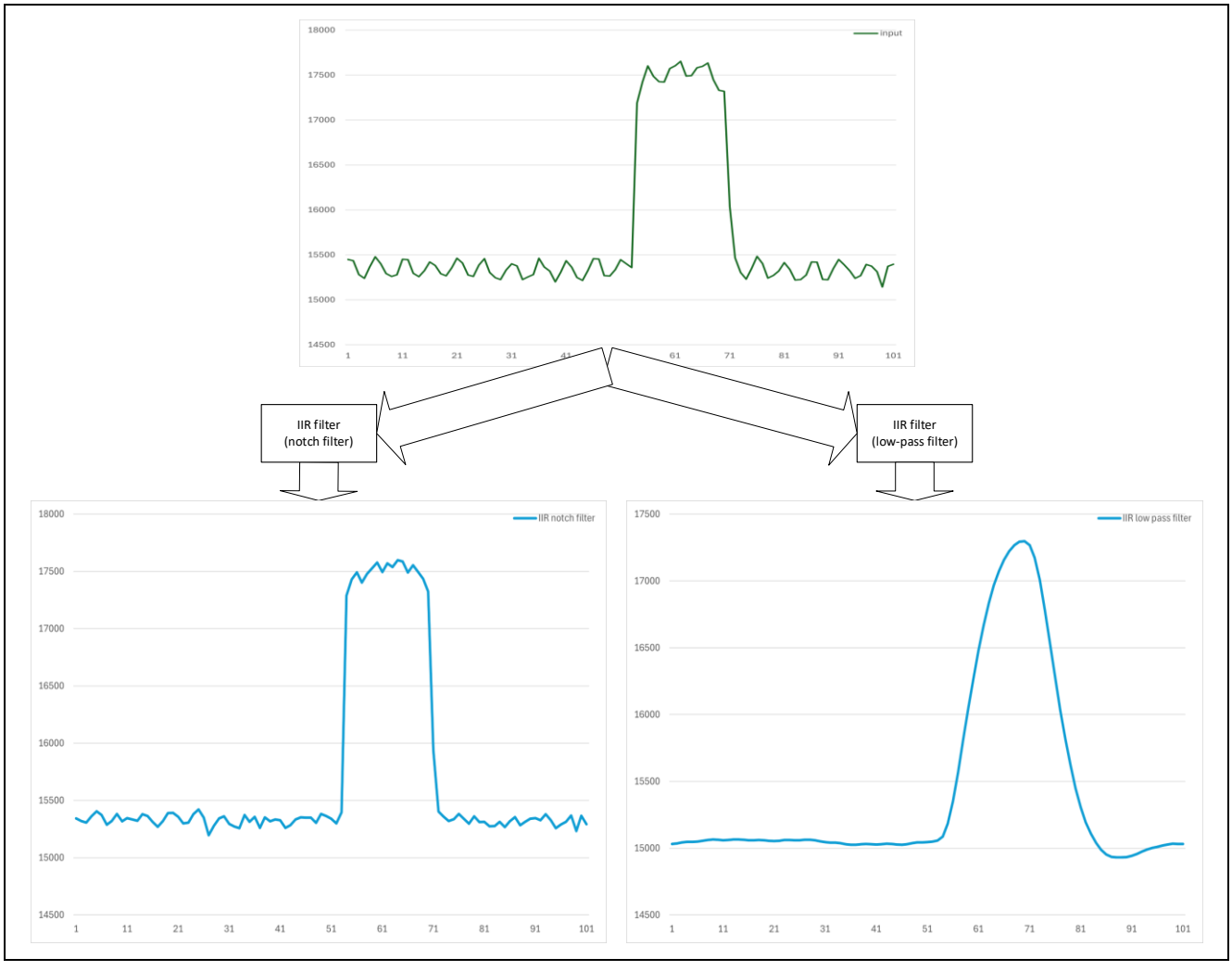


Figure 4-1 IIR Filter Processing (notch filter and low-pass filter)

The calculation formula for IIR filters is shown in the block diagram in **Figure 4-2**.

$$y(n) = \sum_{k=0}^{2} b(k) * x(n-k) + \sum_{m=1}^{2} a(m) * y(n-m)$$

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

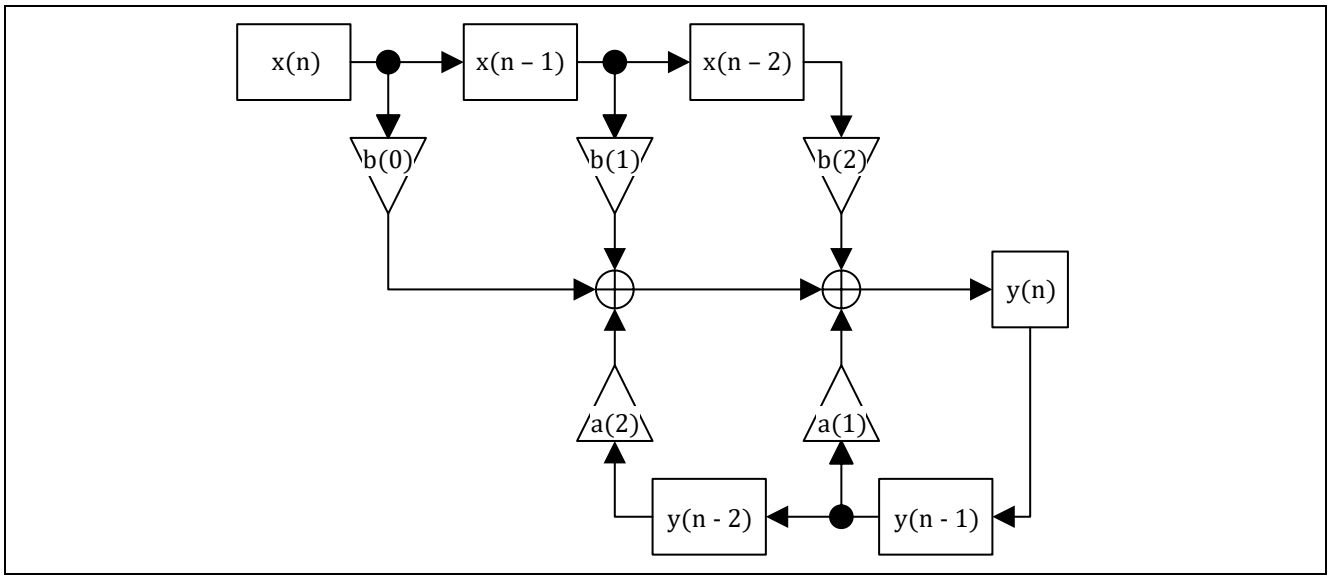


Figure 4-2 IIR Filter Block Diagram

4.1.1 Detection Delay

With the IIR filter, delays may occur in normal detection depending on filter characteristics.

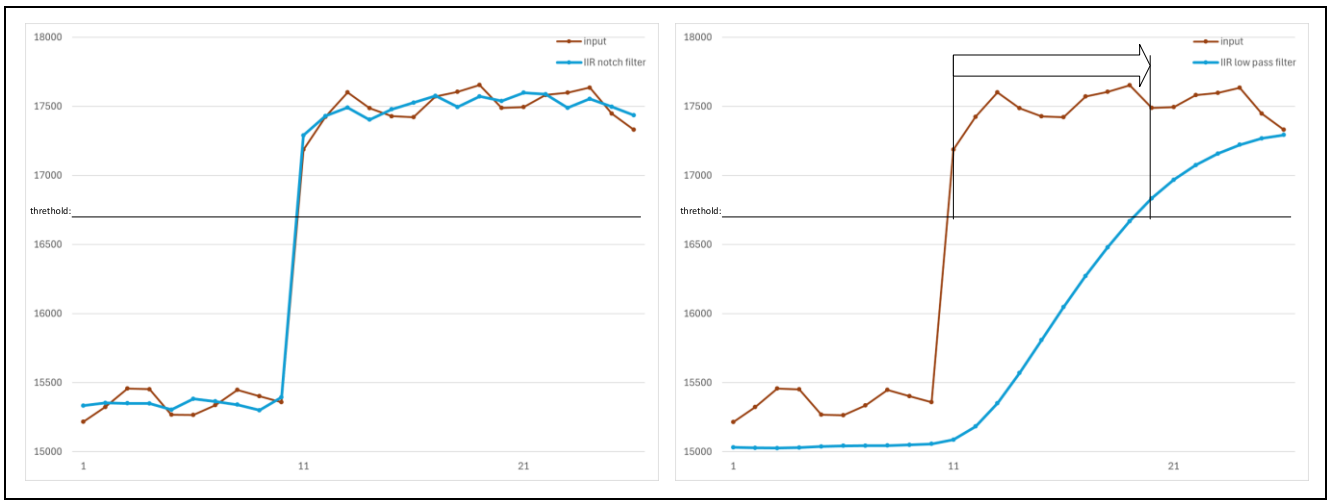


Figure 4-3 Detection Delay (IIR notch filter and low-pass filter)

4.1.2 Filter Stabilization Time

After initialization, the IIR filter outputs calculation results that deviate from the input data until a certain number of filter processes are performed.

This period differs depending on the filter characteristics. Make sure to determine how long it takes until a sufficiently stable calculation result is output, specify that time as the stabilization wait time, and then discard the filter calculation results during that period.

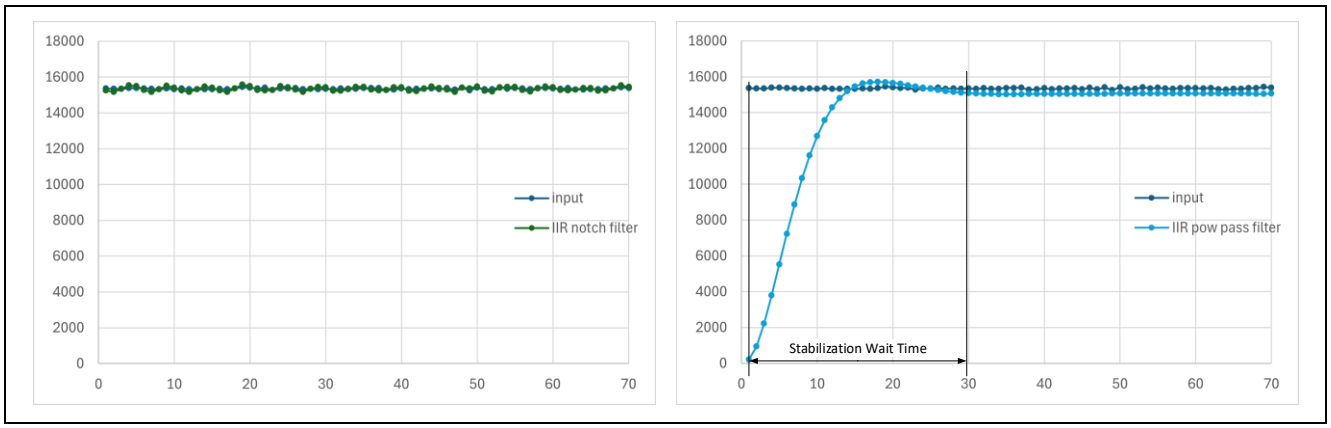


Figure 4-4 Filter Stabilization Wait Time (IIR notch filter, low-pass filter)

4.2 Filter Specifications

Table 4-1 lists the specifications of the IIR filters used in the sample program.

Table 4-1 IIR Filter Specifications

Item	Specification	Remarks
Input data type	Unsigned 16-bit integer type	
Output data type	Unsigned 16-bit integer type	
Coefficient data type	Signed 16-bit integer type	Fixed point with 11-bit decimal
Number of taps	2	
Output results until filter stabilization time	Returns operation results during filter stabilization time and buffer unfilled response	Filter stabilization time = number of samples specified in the configuration definition (stabilization time specification range = number of taps to 255)

Note: Coefficient: A set of constants to be applied to the constant multipliers that make up IIR filters.

4.2.1 Filter Processing Method

This sample program alternates between processing the multiply-and-accumulate operation and buffering the delay buffer.

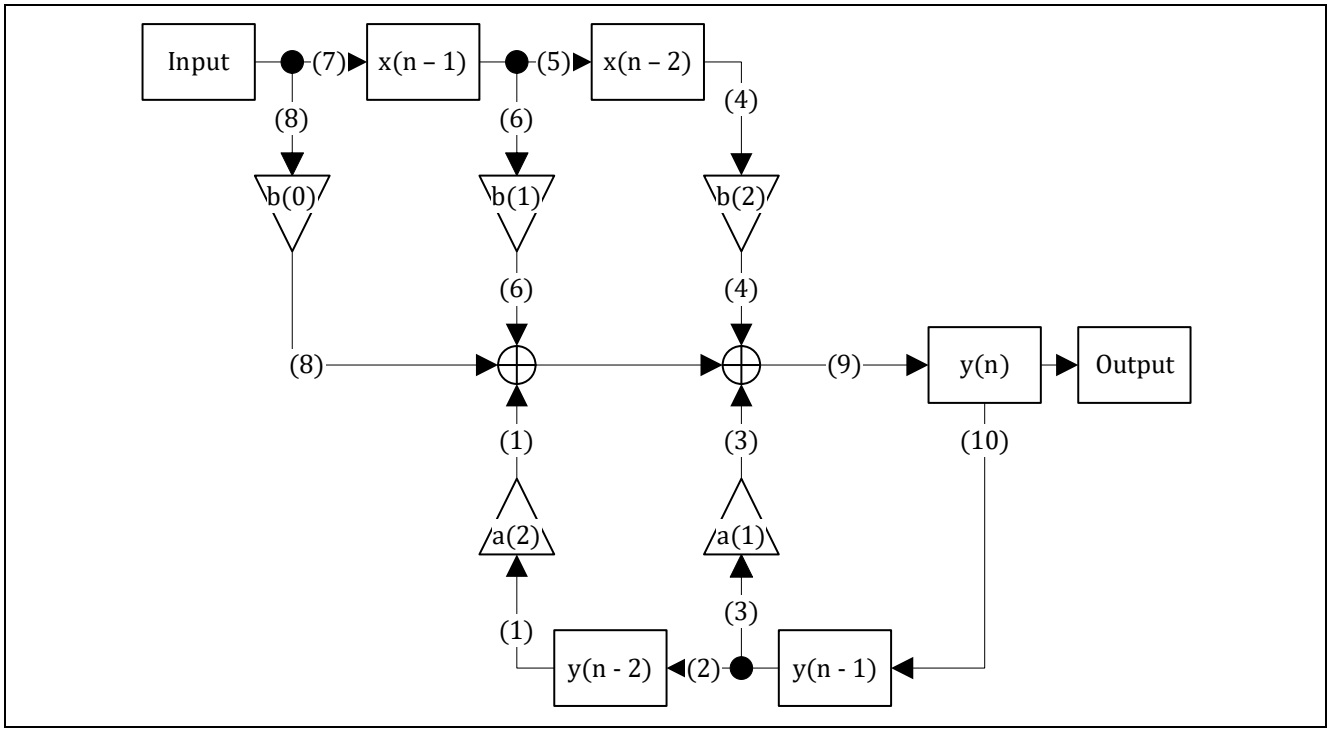


Figure 4-5 IIR Filter Processing

4.2.2 Usage Notes on Filter Coefficients

The coefficient data of the IIR filter must be defined within a value range of -2.0 to 2.0.

The signs in the calculation formulas for the IIR filters may differ depending on the document and tool used.

Note that in this case, the signs of coefficients a(1) and a(2) are reversed.

Table 4-2 IIR Filter Calculation Formulas and Coefficient Signs

For-mula	$y(n) = \sum_{k=0}^{2} b(k) * x(n-k) + \sum_{m=1}^{2} a(m) * y(n-m)$	$y(n) = \sum_{k=0}^{2} b(k) * x(n-k) - \sum_{m=1}^{2} a(m) * y(n-m)$
b(0)	0.9931640625	0.9931640625
b(1)	-0.61376953125	-0.61376953125
b(2)	0.9931640625	0.9931640625
a(1)	0.61376953125	-0.61376953125
a(2)	-0.98681640625	0.98681640625

4.3 List of Data for IIR Filters

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

4.3.1 Constants

Table 4-3 lists the constants IIR filters.

Table 4-3 Constants for IIR Filters

Constant name	Setting value	Description
File name: r_ctsu_iir_sample.h		
IIR_COEF_SIZE	5	
IIR_BUFFER_SIZE	4	
IIR_DATA_SIZE	1	Data size to be filtered
File name: r_ctsu_iir_sample.c		
IIR_SETTLINGS_MAX	255	Maximum wait time
IIR_SETTLINGS_MIN	3	Minimum wait time
IIR_CFG_DECIMAL_POINT	11	Number of fixed-point digits
MAX_IIR_COEFFICIENT	0x1000	Maximum value of the coefficient definition
MIN_IIR_COEFFICIENT	0xF000	Minimum value of the coefficient definition
IIR_RESULT_MAX	0x0000FFFF	Maximum value of filter result
IIR_RESULT_MIN	0x00000000	Minimum value of filter result

4.3.2 Structures

The following shows the management data structures for accessing the IIR filter APIs and the structures for defining the IIR filter configuration.

4.3.2.1 IIR Filter Configuration Definition (iir_config_t)

Table 4-4 IIR Filter Configuration Definition structure (iir_config_t)

Member	Data type	Description
settlings	uint16_t	Stabilization wait time
coefficient	int16_t [IIR_COEF_SIZE]	IIR filter coefficient Coefficient storage order is b0, b1, b2, a1, a2

4.3.2.2 IIR Filter Management Data (iir_ctrl_t)

Table 4-5 IIR Filter Management Data Structure (iir_ctrl_t)

Member	Data type	Description
count	uint16_t	Stabilization wait time counter
iir_data	uint16_t [IIR_DATA_SIZE][IIR_BUFFER_SIZE]	Delay buffer for IIR filter

4.4 IIR Filter API

4.4.1 r_ctsu_iir_initial

This function initializes the management data for IIR filter processing.

This function initializes management data for IIR filter processing.

Make sure you execute this function before using r_ctsu_iir_filter.

Format

fsp_err_t r_ctsu_iir_initial(iir_ctrl_t * 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_INVALID_ARGUMENT	/* Configuration definition parameters are invalid */

Properties

Protype is declared in r_ctsu_iir_sample.h.

Description

This function initializes the management data for IIR filter processing.

An error is returned if the configuration definition parameters are not within valid range.

Example

```
err = r_ctsu_iir_initial(&g_ctsu_iir_control1, &g_iir_cfg);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

4.4.2 r_ctsu_iir_filter

This function applies the IIR filter processing.

Format

fsp_err_t r_ctsu_iir_filter(iir_ctrl_t * const p_ctrl , iir_config_t const * const p_cfg , uint16_t *p_data);

Parameters

p_ctrl

IIR filter management data pointer

p_cfg

IIR filter configuration definition pointer

p_data

IIR filter measurement result data pointer

ReturnValues

FSP_SUCCESS /* Successfully completed */

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.

The result of the operation is limited to the range of unsigned 16-bit integers (65535 to 0); if it exceeds the range, it will be rounded to the upper or lower limit value.

After initialization, an error is returned while processing is performed for the stabilization wait time indicated in the configuration definition.

Example

```
/* IIR notch filter sample */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config02.p_ctrl, filter_buffer);
if(err == FSP_SUCCESS)
{
 err=r_ctsu_iir_filter(&g_ctsu_iir_control1, &g_iir_cfg1, filter_buffer);
 if(err == FSP_SUCCESS)
 {
    R_CTSU_DataInsert(g_qe_ctsu_instance_config02.p_ctrl, filter_buffer);
 }
}
```

Special Notes:

Filter operations are performed even when an error response is received.

When configuring multiple filters in cascade, make sure that the filter processing in the next stage is executed even when an error response is received.

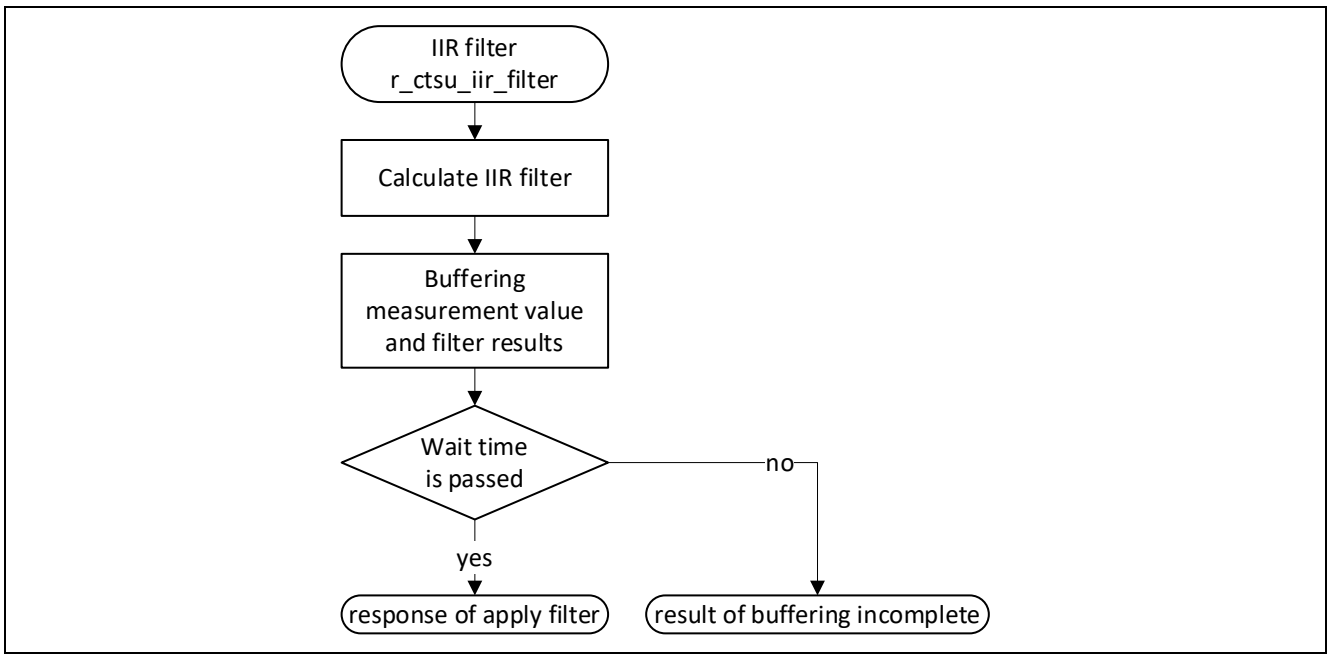


Figure 4-6 IIR Filter Execution API Flowchart

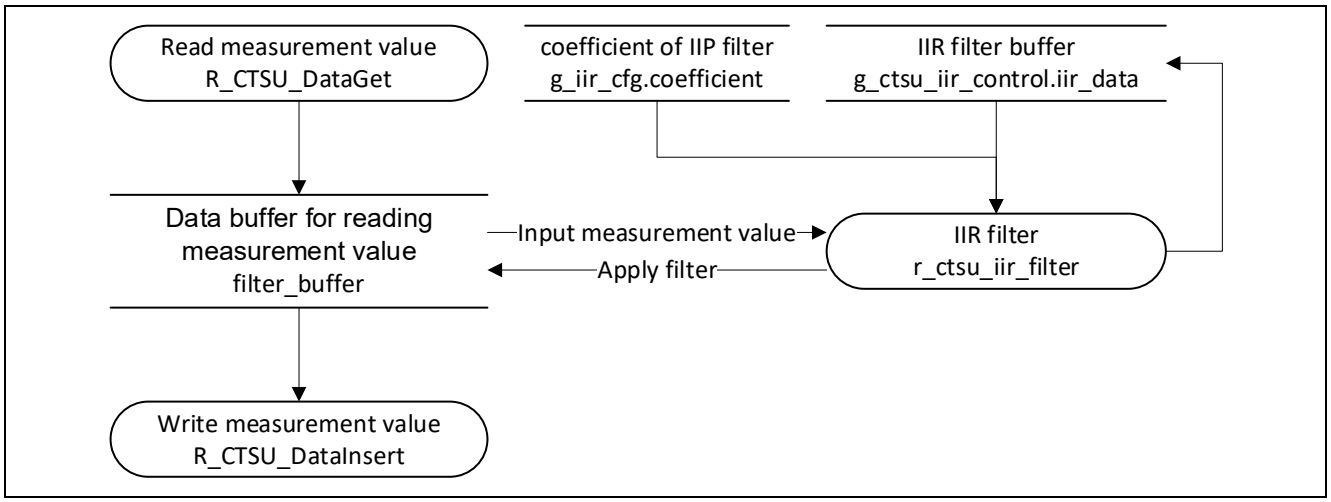


Figure 4-7 IIR Filter Data Flowchart

4.5 Usage Example

This sample program provides a notch filter and low-pass filter as an example of how to use IIR filters.

4.5.1 Filter Characteristics

Table 4-6 shows the coefficient definitions and characteristics of the sample IIR filters.

Table 4-6 Sample IIR Filter Coefficient Definitions

	g_iir_cfg1	g_iir_cfg2
	IIR notch filter	**IIR low-pass filter**
Coefficient B	2034 (0.9931640625)	27 (0.01318359375)
	-1257 (-0.61376953125)	54 (0.0263671875)
	2034 (0.9931640625)	27 (0.01318359375)
Coefficient A	1257 (0.61376953125)	3373 (1.64697265625)
	-2021 (-0.98681640625)	-1435 (-0.70068359375)

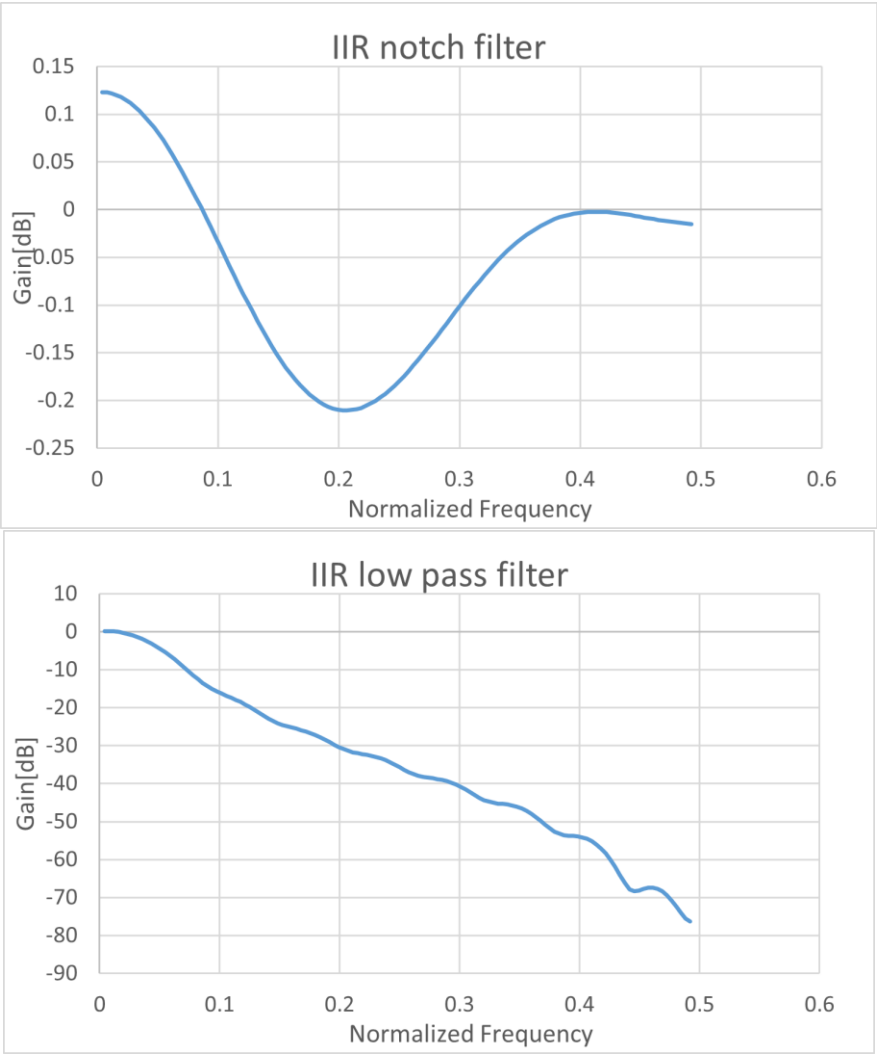


Figure 4-8 Sample IIR Filter Frequency Characteristics

5. Single-pole IIR Filters

Single-pole IIR (Single-pole Infinite Impulse Response) filters are the smallest IIR filter configuration.

5.1 Specifications

Single-pole IIR filters output the results of the multiply-and-accumulate operation result on the input value, the previous output value, and the defined coefficient, and can handle different filter characteristics, such as low-pass filters and high-pass filters, depending on the defined coefficient.

In this sample program, the single-pole IIR filter is treated as a low-pass filter.

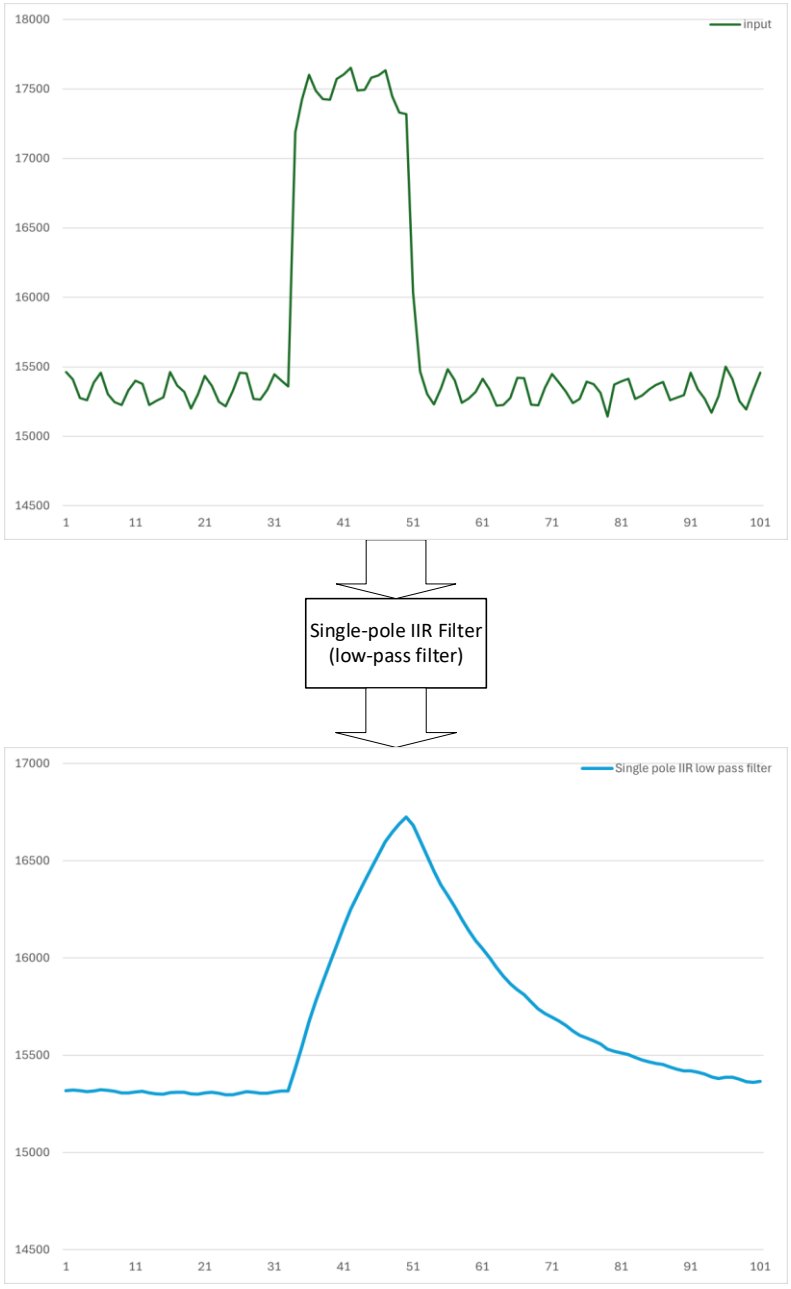


Figure 5-1 Single-pole IIR Filter Processing (low-pass filter)

The single-pole IIR filter block diagram for the following single-pole IIR filter calculation is shown in **Figure 5-2**.

$$y(n) = b * x(n) + a * y(n-1)$$

n indicates the sample index, *a* and *b* indicate the coefficients, $x(n)$ indicates the input data, $y(n-1)$ indicates the output data of sample delay 1, and $y(n)$ indicates the output data.

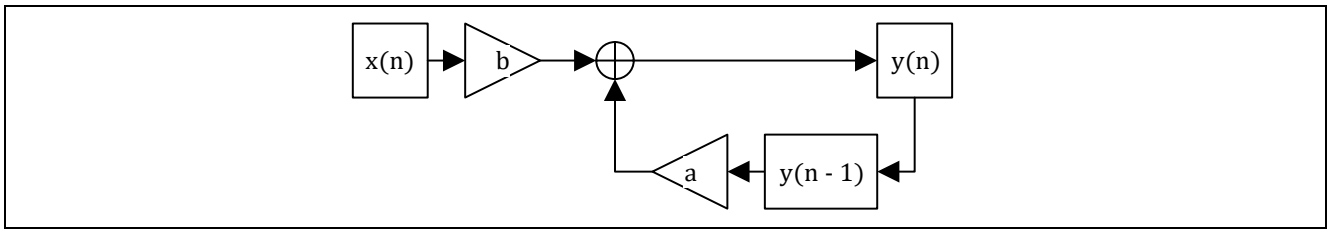


Figure 5-2 Single-pole IIR Filter Block Diagram

5.1.1 Detection Delay

With the single-pole IIR filter, delays may occur in normal touch detection.

Delay time varies depending on filter characteristics.

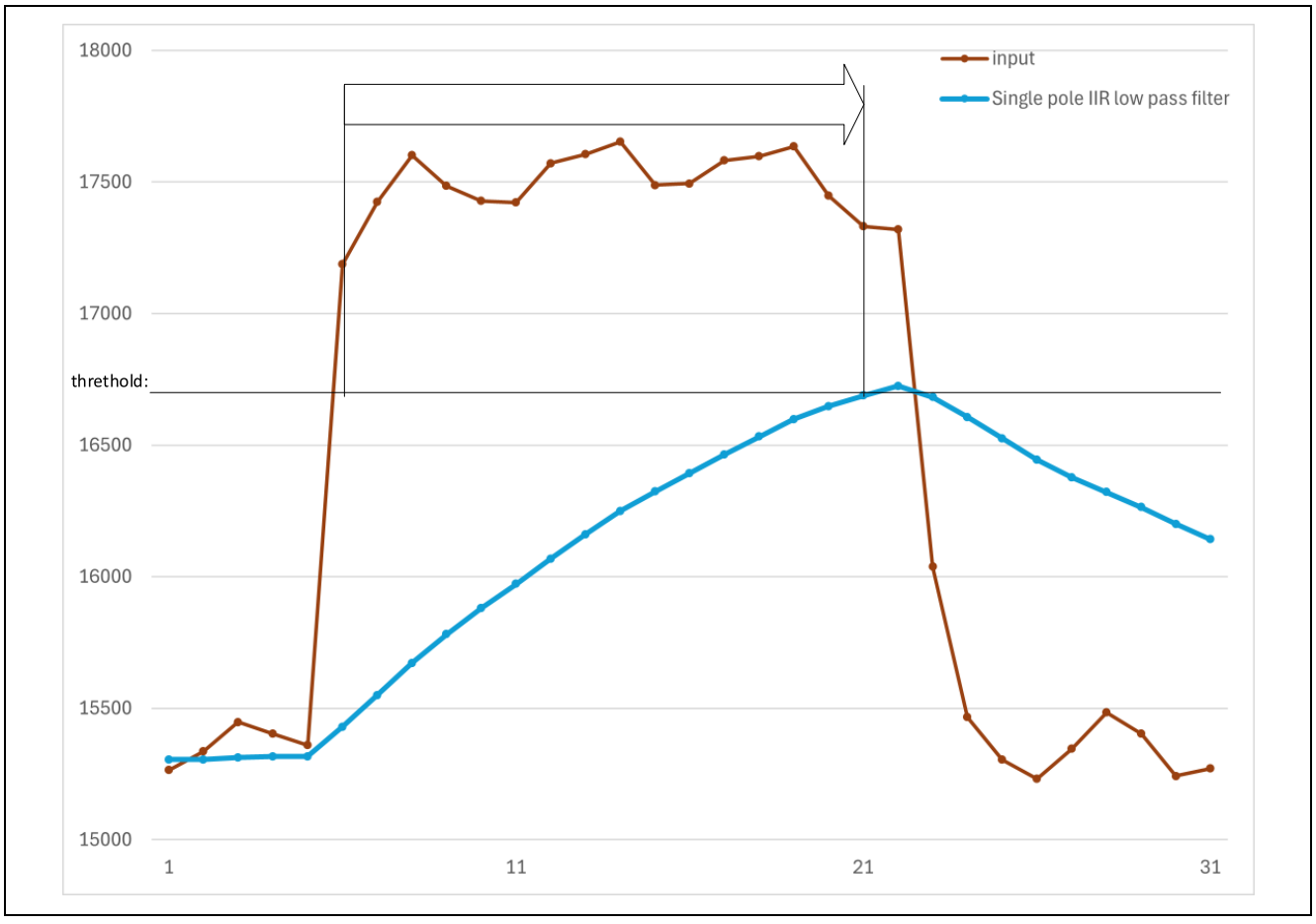


Figure 5-3 Detection Delay (single-pole IIR low-pass filter)

5.1.2 Filter Stabilization Time

After initialization, the single-pole IIR filter outputs calculation results that deviate from the input data until a certain number of filter processes are performed.

If the filter calculation result during this period is used for touch detection, the touch detection operation may not work properly, so we recommend discarding the filter calculation results during this period.

This period differs depending on the filter characteristics. Make sure to determine how long it takes until a sufficiently stable calculation result is output, and specify that time as the stabilization wait time.

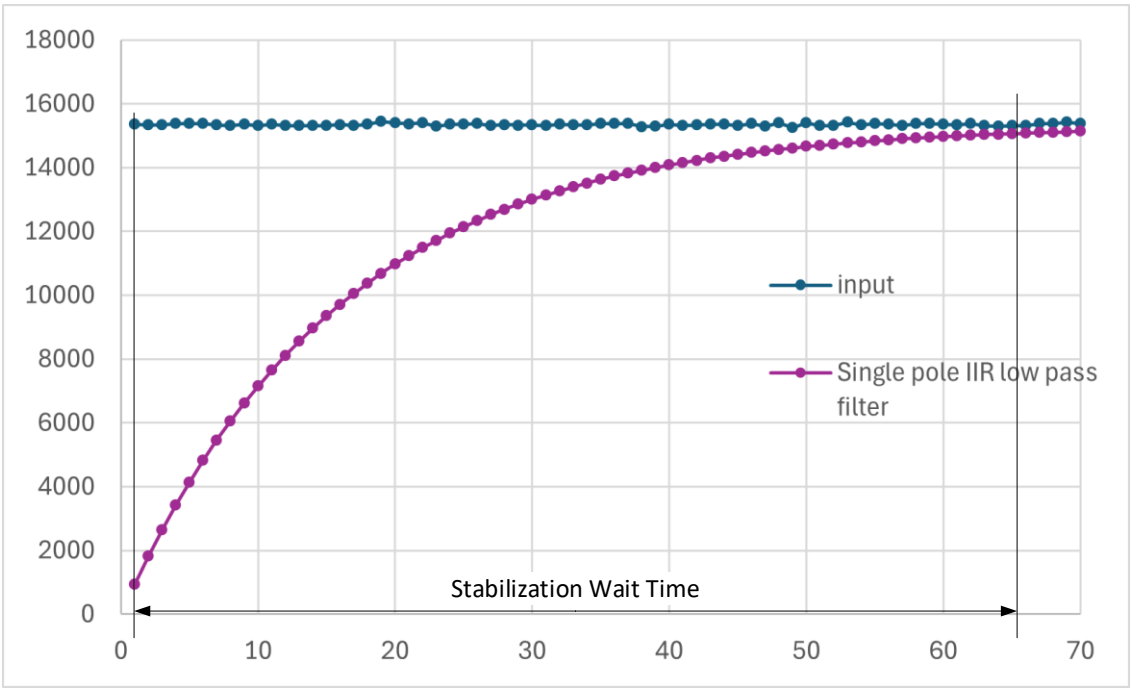


Figure 5-4 Filter Stabilization Wait Time (single-pole IIR low-pass filter)

5.2 Filter Specifications

Table 5-1 lists the specifications of the single-pole IIR filters used in the sample program.

Table 5-1 Single-pole IIR Filter Specifications

Item	Specification	Remarks
Input data type	Unsigned 16-bit integer type	
Output data type	Unsigned 16-bit integer type	
Coefficient data type	Signed 16-bit integer type	Fixed point with 14-bit decimal
Number of taps	Returns operation results during filter stabilization time and buffer unfilled response	Filter stabilization time = number of samples specified in the configuration definition (stabilization time specification range = 1 to 255)

Coefficient: A set of constants to be applied to the constant multipliers that make up single-pole IIR filters.

5.2.1 Filter Processing Method

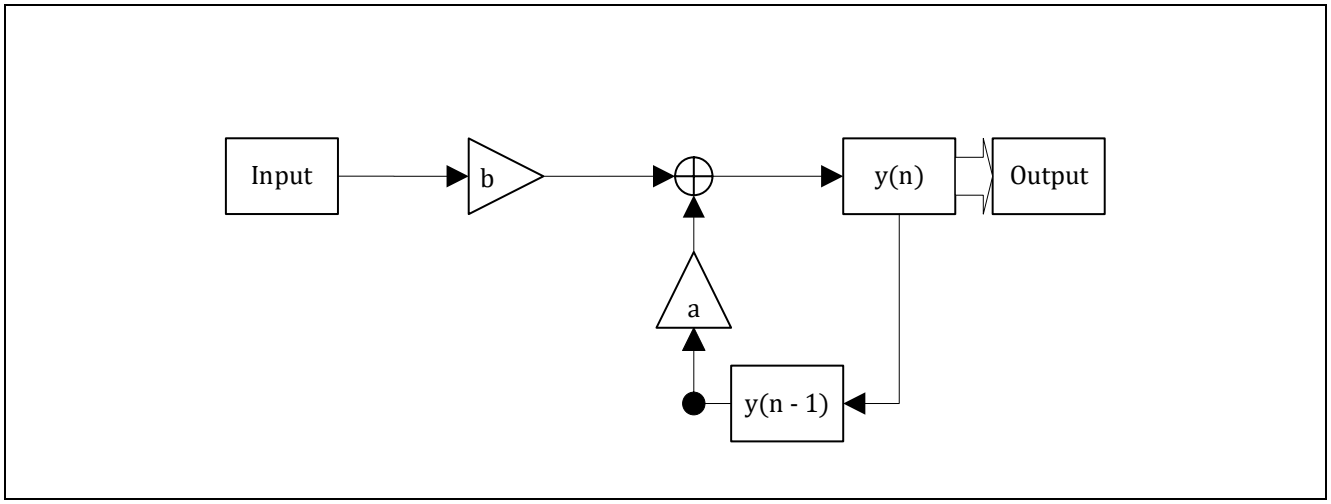


Figure 5-5 Single-pole IIR Filter Processing

5.2.2 Usage Notes on Filter Coefficients

The single-pole IIR filter coefficient data must be defined within a value range of -1.0 to 1.0.

5.3 List of Data for Single-pole IIR Filters

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

5.3.1 Constants

Table 5-2 lists the constants for single-pole IIR filters.

Table 5-2 Single-pole IIR Filter Constants

Constant name	Setting value	Description
File name: r_ctsu_spiir_sample.h		
SPIIR_COEF_SIZE	2	Array size for coefficients
SPIIR_DATA_SIZE	1	Data size to be filtered
File name: r_ctsu_spiir_sample.c		
SPIIR_SETTLINGS_MAX	255	Maximum wait time
SPIIR_SETTLINGS_MIN	1	Minimum wait time
SPIIR_CFG_DECIMAL_POINT	14	Number of fixed-point digits
SPIIR_COEF_MAX	0x4000	Maximum value of the coefficient definition
SPIIR_COEF_MIN	0xC000	Minimum value of the coefficient definition
SPIIR_RESULT_MAX	0x0000FFFF	Maximum value of filter result
SPIIR_RESULT_MIN	0x00000000	Minimum value of filter result

5.3.2 Structures

The following shows the management data structures for accessing the single-pole IIR filter APIs and the structures for defining the single-pole IIR filter configuration.

5.3.2.1 Single-pole IIR Filter Configuration Definition (spiir_config_t)

Table 5-3 Single-pole IIR Filter Configuration Definition Structure (spiir_config_t)

Member	Data type	Description
settlings	uint16_t	Stabilization wait time
coefficient	int16_t [SPIIR_COEF_SIZE]	Single-pole IIR filter coefficient

5.3.2.2 Single-pole IIR Filter Management Data (spiir_ctrl_t)

Table 5-4 Single-pole IIR Filter Management Data Structure (spiir_ctrl_t)

Member	Data type	Description
count	uint16_t	Stabilization wait time counter
spiir_data	uint16_t [SPIIR_DATA_SIZE]	Delay buffer for single-pole IIR filter

5.4 Singel-pole IIR Filter API

5.4.1 r_ctsu_spiir_initial

This function initializes the management data for single-pole IIR filter processing.

Make sure you execute this function before using r_ctsu_spiir_filter.

Format

fsp_err_t r_ctsu_spiir_initial(spiir_ctrl_t * p_ctrl , spiir_config_t const * const p_cfg);

Parameters

p_ctrl

Single-pole IIR filter management data pointer

p_cfg

Single-pole IIR filter configuration definition pointer

ReturnValues

FSP_SUCCESS	/* Successfully completed */
FSP_ERR_INVALID_ARGUMENT	/* Configuration definition parameters are invalid */

Properties

Protype is declared in r_ctsu_spiir_sample.h.

Description

This function initializes the management data for single-pole IIR filter processing.

An error is returned when the configuration definition parameters are not within valid range.

Example

```
err = r_ctsu_spiir_initial(&g_ctsu_spiir_control1, &g_spiir_cfg);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

5.4.2 r_ctsu_spiir_filter

This function applies the single-pole IIR filter processing.

Format

fsp_err_t r_ctsu_spiir_filter(spiir_ctrl_t * const p_ctrl, spiir_config_t const * const p_cfg,uint16_t *p_data);

Parameters

p_ctrl

Single-pole IIR filter management data pointer

p_cfg

Single-pole IIR filter configuration definition pointer

p_data

Single-pole IIR filter measurement result data pointer

ReturnValues

FSP_SUCCESS /* Successfully completed */

FSP_ERR_BUFFER_EMPTY /* Some filters are not yet applied because buffer is unfilled. */

Properties

Protype is declared in r_ctsu_spiir_sample.h.

Description

This function applies the single-pole IIR filter processing.

The result of the operation is limited to the range of unsigned 16-bit integers (65535 to 0); if it exceeds the range, it will be rounded to the upper or lower limit value.

After initialization, an error is returned while processing is performed for the stabilization wait time indicated in the configuration definition.

Example

```
/* Single Pole IIR low pass filter sample */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config04.p_ctrl, filter_buffer);
if(err == FSP_SUCCESS)
{
 err=r_ctsu_spiir_filter(&g_ctsu_spiir_control1, &g_spiir_cfg1,
filter_buffer);
  if(err == FSP_SUCCESS)
  {
     R_CTSU_DataInsert(g_qe_ctsu_instance_config04.p_ctrl, filter_buffer);
  }
}
```

Special Notes:

Filter operations are performed even when an error response is received.

When configuring multiple filters in cascade, make sure that the filter processing in the next stage is executed even when an error response is received.

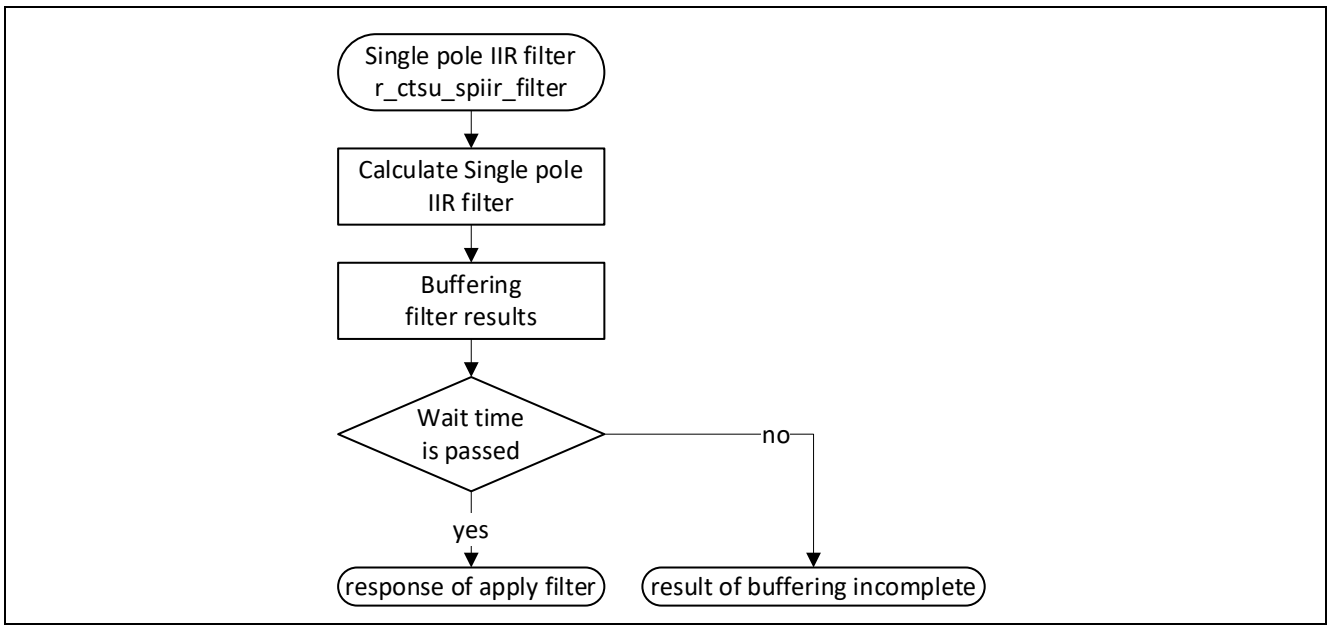


Figure 5-6 Single-pole IIR Filter Execution API Flowchart

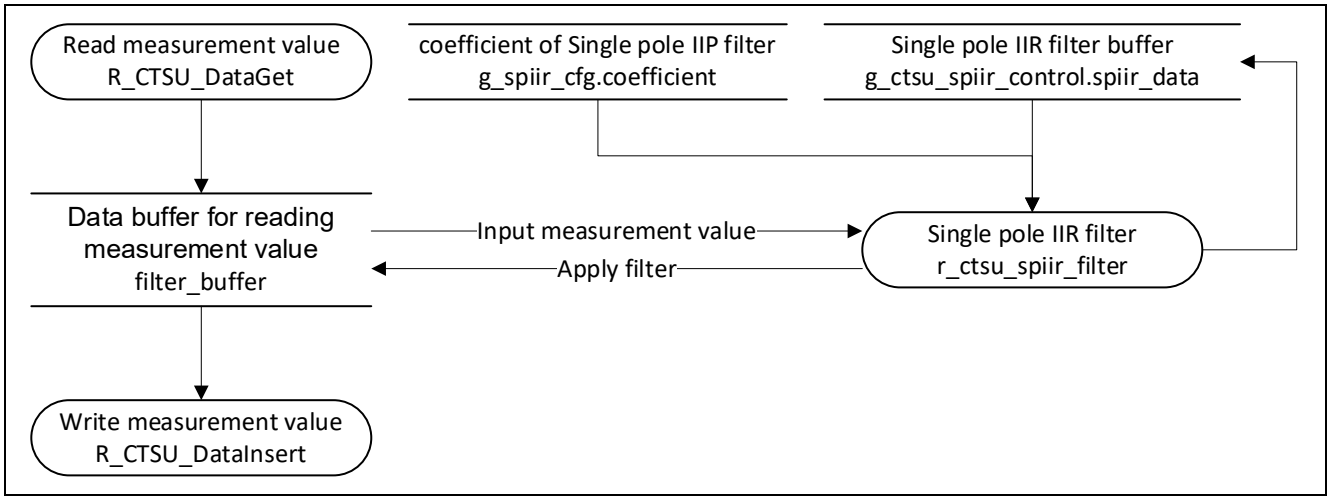


Figure 5-7 Single-pole IIR Filter Data Flowchart

5.5 Usage Example

This sample program provides a low-pass filter as an example of how to use single-pole IIR filters.

5.5.1 Filter Characteristics

Table 5-5 shows the coefficient definitions and characteristics of the sample single-pole IIR filters.

Table 5-5 Sample Single-pole IIR Filter Coefficient Definitions

	g_spiir_cfg
	Single-pole IIR low-pass filter
Coefficient A	15386 (0.9390869140625)
Coefficient B	997 (0.06085205078125)

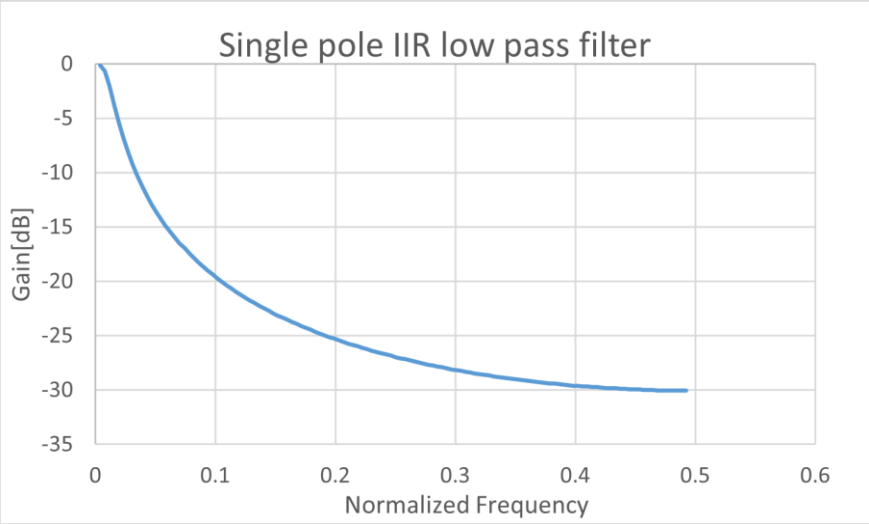


Figure 5-8 Sample Single-pole IIR Filter Frequency Characteristics

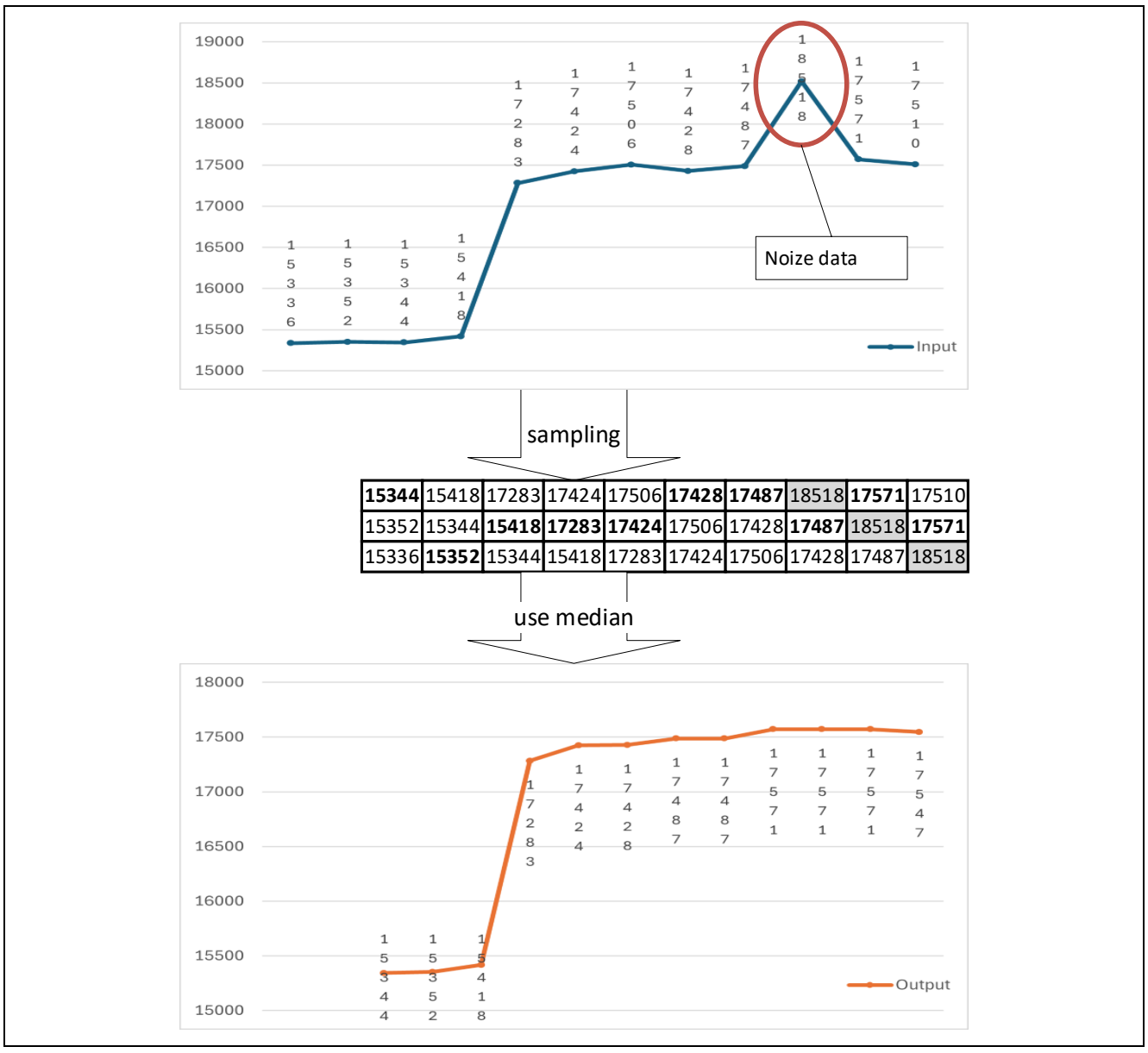
6. Median Filters

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.

6.1 Specifications

Median filters sample input values and use the median value of the sampled data as the output value.

Pulsed noise is input as data that deviates greatly from the median value and is therefore removed from the data by the median filter operation.



15344	15418	17283	17424	17506	**17428**	**17487**	18518	**17571**	17510
15352	15344	**15418**	**17283**	**17424**	17506	17428	**17487**	18518	**17571**
15336	**15352**	15344	15418	17283	17424	17506	17428	17487	18518

Figure 6-1 Median Filter Processing

6.1.1 Detection Delay

The median filter samples the touch measurement values to remove noise signals which therefore causes a delay in normal touch detection.

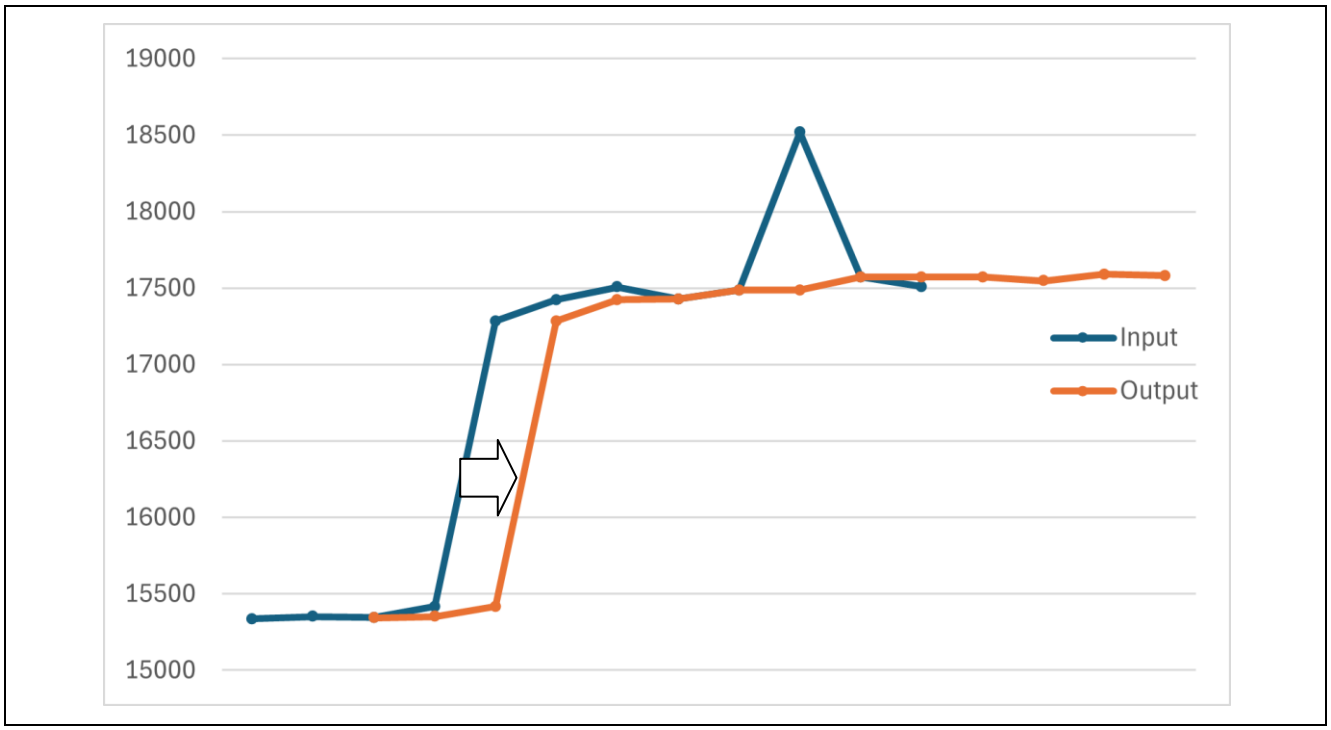


Figure 6-2 Median Filter Detection Delay

6.1.2 Filter Stabilization Time

After initialization, the median filter outputs calculation results that are lower than the input data until the filter processing has been performed the number of times of sampled data.

If the filter processing result is used during this period for touch detection, the touch detection operation may not work properly, so we recommend discarding the filter operation results output during this period.

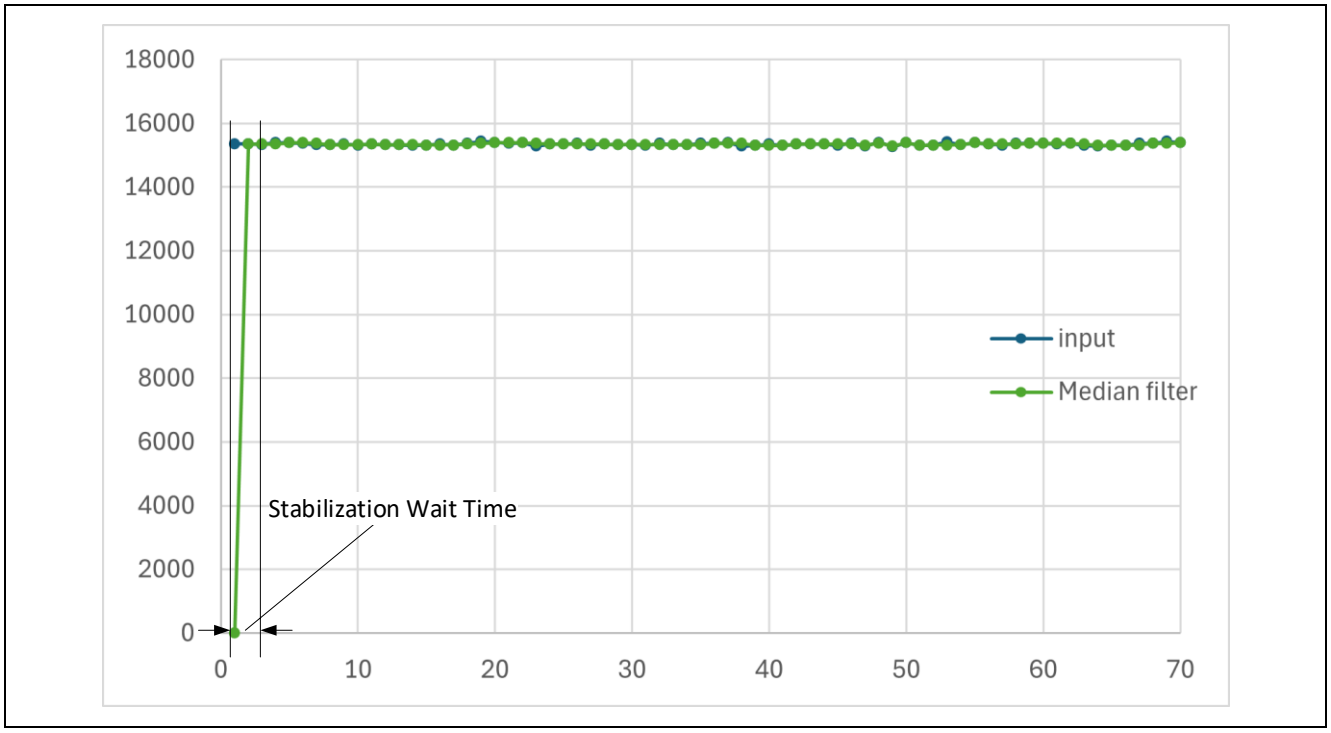


Figure 6-3 Filter Stabilization Wait Time (median)

6.2 Filter Specifications

Table 6-1 lists the specifications of the median filters used in the sample program.

Table 6-1 Median Filter Specifications

Item	Specification	Remarks
Input data type	Unsigned 16-bit integer type	
Output data type	Unsigned 16-bit integer type	
Sample reference range	3, 5, 7, 9	Total number of input values and past samples
Processing method	Median detection with insert sorting	
Output results after filter initialization	Returns operation results during filter stabilization time and buffer unfilled response	Filter stabilization time = sample reference period

6.2.1 Filter Processing Method

The median filter samples the input data, copies the sampled data into a temporary buffer, sorts it, and outputs the data in the middle position.

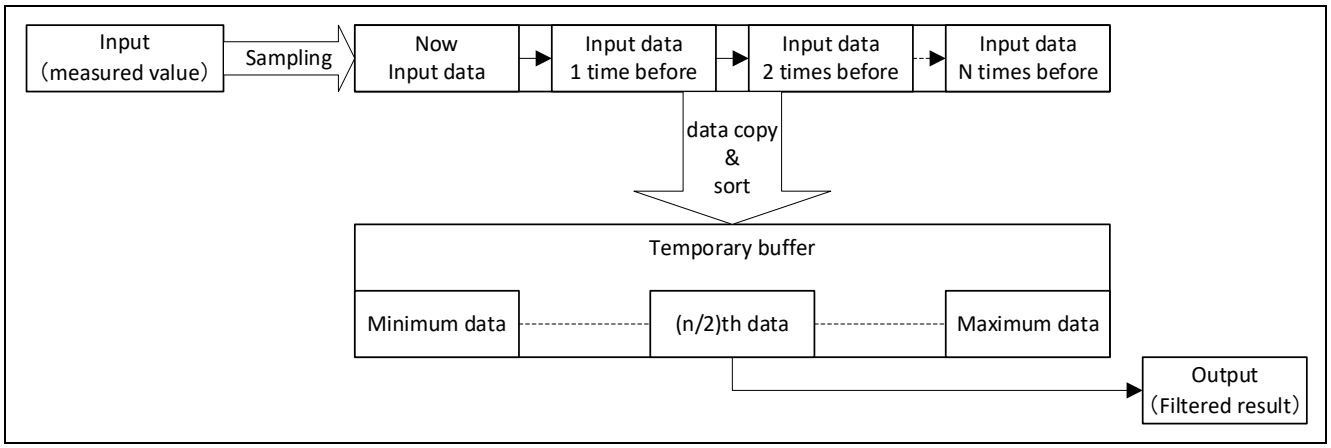


Figure 6-4 Median Filter Processing

6.3 List of Data for Median Filters

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

6.3.1 Constants

Table 6-2 lists the constants for median filters.

Table 6-2 Median Filter Constants

Constant name	Setting value	Description
File name: r_ctsu_median_sample.h		
MEDIAN_SAMPLE_SIZE	3	Sampling size
MEDIAN_DATA_SIZE	1	Data size to be filtered
File name: r_ctsu_median_sample.c		
MEDIAN_SAMPLE_SIZE_MIN	3	Minimum sample reference period
MEDIAN_SAMPLE_SIZE_MAX	9	Maximum sample reference period

6.3.2 Structures

The following shows the management data structures for accessing the median filter API.

6.3.2.1 Median Filter Management Data (median_ctrl_t)

Table 6-3 Median Filter Management Data Structure (median_ctrl_t)

Member	Data type	Description
count	uint16_t	Stabilization wait time counter
index	uint16_t	Median filter sampling buffer input data storage location
median_data	uint16_t [MEDIAN_DATA_SIZE][MEDIAN_SAMPLE_SIZE]	Median filter sampling buffer

6.4 Median Filter API

6.4.1 r_ctsu_median_initial

This function initializes management data for median filter processing.

Make sure you execute this function before using r_ctsu_median_filter.

Format

fsp_err_t r_ctsu_median_initial(median_ctrl_t * const p_ctrl);

Parameters

p_ctrl

Median filter management data pointer

ReturnValues

FSP_SUCCESS	/* Successfully completed */
FSP_ERR_INVALID_ARGUMENT	/* Configuration definition parameters are invalid */

Properties

Protype is declared in r_ctsu_median_sample.h.

Description

This function initializes the management data for median filter processing.

An error is returned if the sampling size definition exceeds the valid range.

Example

```
err = r_ctsu_median_initial(&g_ctsu_median_control);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

6.4.2 r_ctsu_median_filter

This function applies the median filter processing.

Format

fsp_err_t r_ctsu_median_filter(median_ctrl_t * const p_ctrl , uint16_t *p_data);

Parameters

p_ctrl

Median filter management data pointer

p_data

Median filter measurement result data pointer

ReturnValues

FSP_SUCCESS /* Successfully completed */

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.

After initialization, an error is returned while processing of sampling size is in progress.

Example

```
/* Median filter sample */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config05.p_ctrl, filter_buffer);
if(err == FSP_SUCCESS)
{
 err=r_ctsu_median_filter(&g_ctsu_median_control, filter_buffer);
 if(err == FSP_SUCCESS)
 {
    R_CTSU_DataInsert(g_qe_ctsu_instance_config05.p_ctrl, filter_buffer);
 }
}
```

Special Notes:

Filter operations are performed even when an error response is received.

When configuring multiple filters in cascade, make sure that the filter processing in the next stage is executed even when an error response is received.

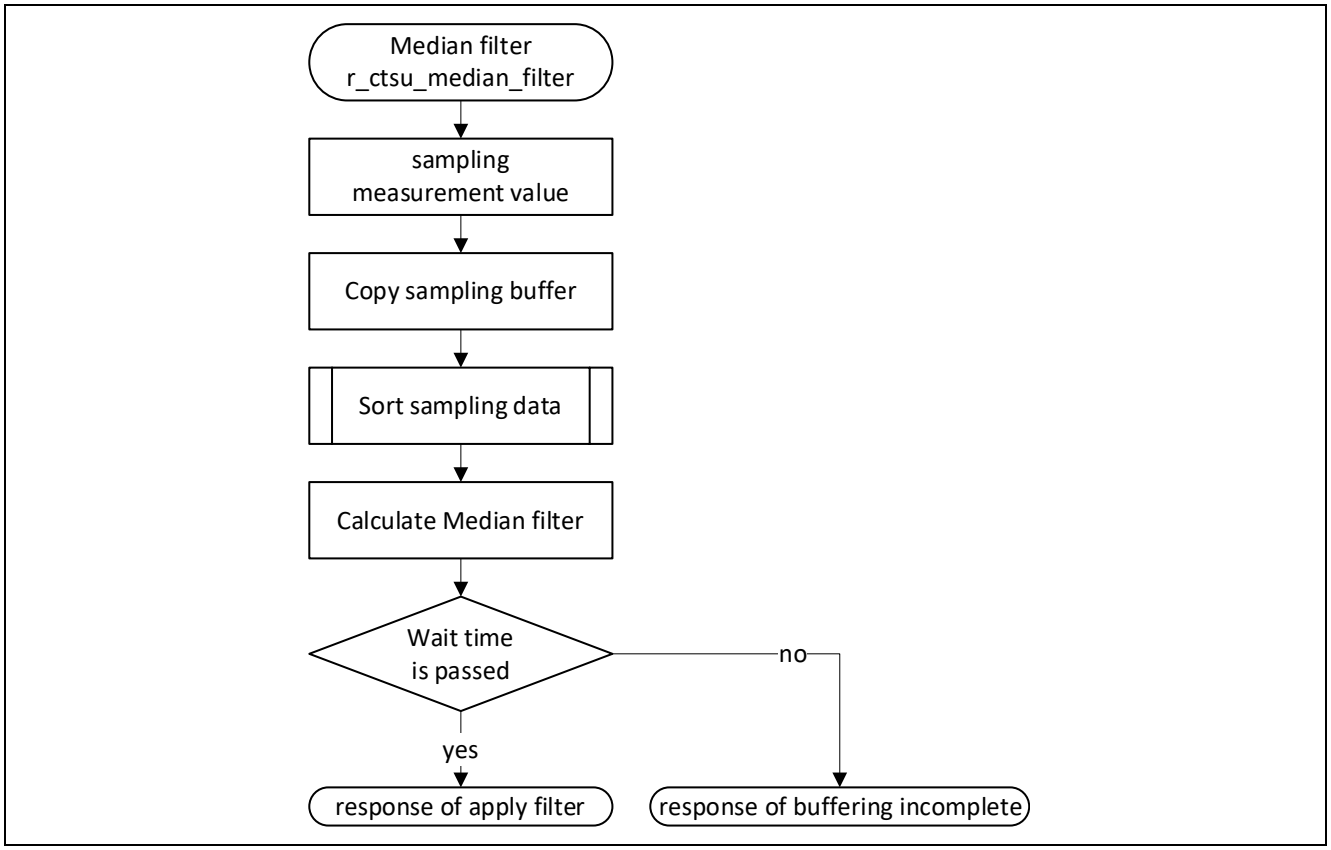


Figure 6-5 Median Filter Execution API Flowchart

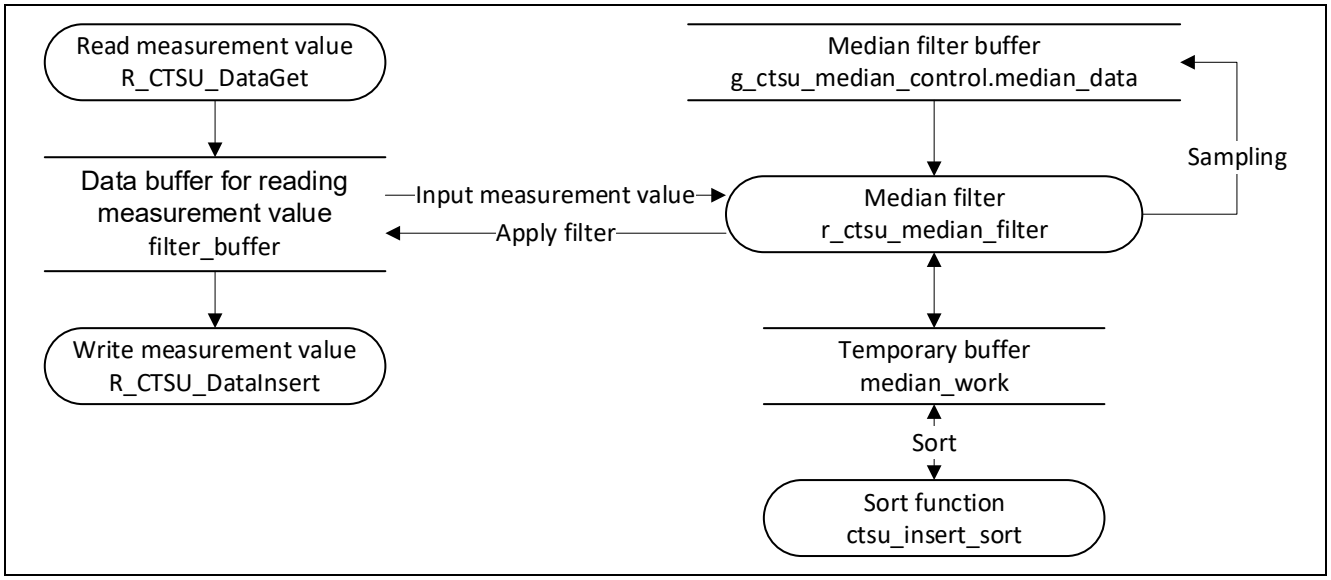


Figure 6-6 Median Filter Data Flowchart

6.4.1 ctsu_insert_sort

This function sorts the specified data by value.

Format

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

Parameters

p_list

Specified sorting data pointer

size

Number of data to be sorted

ReturnValues

None

Properties

Protype is declared in r_ctsu_median_sample.c.

Description

This function sorts the specified data in ascending order.

6.5 Usage Example

This sample program provides three median filters in the sample reference period as a usage example.

6.5.1 Filter Characteristics

Table 6-4 shows the coefficient definitions and characteristics of the sample median filters.

Table 6-4 Sample Median Filter Characteristics

Sample Reference period	3(60ms)
Noise removal width	1(20ms)
Detection delay	1(20ms)

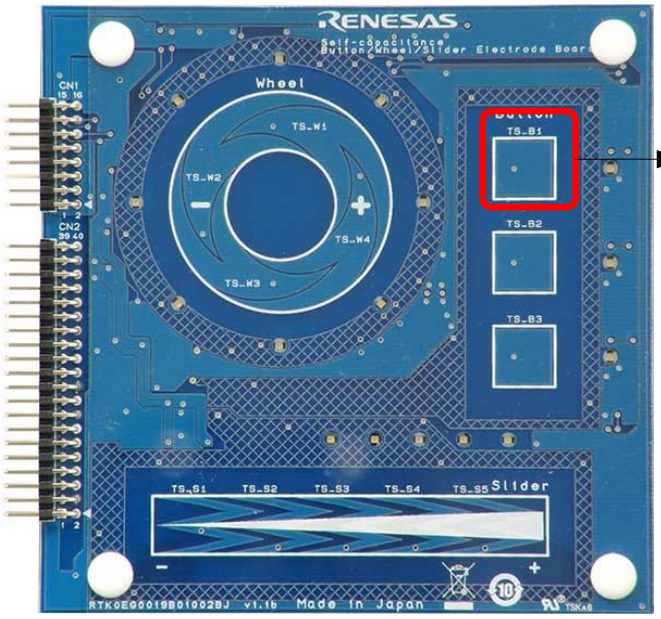
7. How to Use the Sample Project/Sample Code

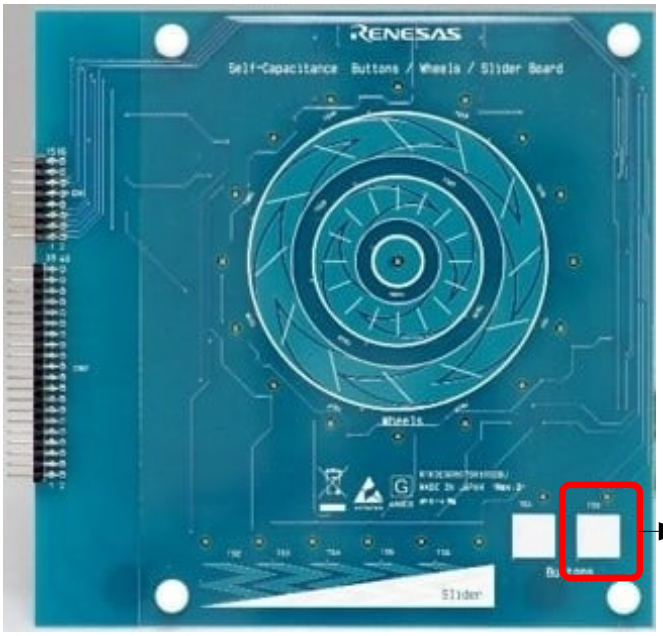
7.1 How to Use the Sample Project

This section explains how to use the sample project which applies the software filter sample program.

7.1.1 Sample Application

The application in the sample project uses only one button and applies five types of filters for touch detection.





FIR moving-average filter
IIR notch filter
IIR low-pass filter
Single-pole IIR low-pass filter
Median filter

FIR moving- average filter
IIR notch filter
IIR low-pass filter
Single-pole IIR low-pass filter
Median filter

7.1.2 Functions

The functions of the sample project are listed below.

- The only touch electrode used on the self-capacitance electrode board is Button 1 (TS-B1: for RA2L1 and RL78G16, TS0: for RX130).
- Five methods are defined for the touch electrode measurement results of Button 1, and a different software filter is applied to each method.
- You can use the serial monitoring function of QE for Capacitive Touch to check the measurement with the software filter applied.

Table 7-1 Method Definitions and Corresponding Filters

Method	Filter Description
CONFIG01	FIR moving-average filter
CONFIG02	IIR notch filter
CONFIG03	IIR low-pass filter
CONFIG04	Single-pole IIR low-pass filter
CONFIG05	Median filter

7.1.3 File Structure

This section explains the file structure of the sample project.

7.1.3.1 RA2L1Group

The project configuration file and Smart Configuration generation file of the development environment have been omitted.

```
ra2l1_filter_sample
 │
 ├──QE-Touch
 │      ├ ra2l1_filter_sample_log_tuning20240322151628.log   ・・・QE Tuning Log
 │      └ ra2l1_filter_sample.tifcfg                         ・・・Touch Interface Configuration File
 │
 ├──qe_gen
 │      ├ qe_touch_config.c                                  ・・・Touch Configuration Source
 │      ├ qe_touch_config.h                                  ・・・Touch Configuration Header
 │      ├ qe_touch_define.h                                  ・・・Touch Definition Header
 │      └ qe_touch_sample.c                                  ・・・Touch Sample Application
 │
 ├──src
 │      └ hal_entry.c                                        ・・・main Files
 │
 └──filter_sample
        ├ filter_define_sample.h                             ・・・MCU Related Headers
        ├ fir
        │      ├ fir_config_sample1.c                        ・・・FIR Moving-average Filter Configuration Definition Source
        │      ├ r_ctsu_fir_sample.c                         ・・・FIR Filter Processing Source
        │      └ r_ctsu_fir_sample.h                         ・・・FIR Filter Processing Header
        ├ iir
        │      ├ iir_config_sample1.c                        ・・・IIR Filter Configuration Definition Source
        │      ├ r_ctsu_iir_sample.c                         ・・・IIR Filter Processing Source
        │      └ r_ctsu_iir_sample.h                         ・・・IIR Filter Processing Header
        ├ median
        │      ├ median_config_sample1.c                     ・・・Median Filter Configuration Definition Source
        │      ├ r_ctsu_median_sample.c                      ・・・Median Filter Processing Source
        │      └ r_ctsu_median_sample.h                      ・・・Median Filter Processing Header
        └ spiir
               ├ spiir_config_sample1.c                      ・・・Single-pole IIR Low-pass Filter Configuration Definition Source
               ├ r_ctsu_spiir_sample.c                       ・・・Single-pole IIR Filter Processing Source
               └ r_ctsu_spiir_sample.h                       ・・・Single-pole IIR Filter Processing Header
```

7.1.3.2 RX130 Group

The project configuration file and Smart Configuration generation file of the development environment have been omitted.

```
rx130_filter_sample
 │
 ├──QE-Touch
 │      ├ rx130_filter_sample_log_tuning20240322164158.log  ・・・QE Tuning Log
 │      └ rx130_filter_sample.tifcfg                        ・・・Touch Interface Configuration File
 │
 ├──qe_gen
 │      ├ qe_touch_config.c                                 ・・・Touch Configuration Source
 │      ├ qe_touch_config.h                                 ・・・Touch Configuration Header
 │      ├ qe_touch_define.h                                 ・・・Touch Definition Header
 │      └ qe_touch_sample.c                                 ・・・Touch Sample Application
 │
 ├──src
 │      └ rx130_filter_sample.c                             ・・・main Files
 │
 └──filter_sample
        ├ filter_define_sample.h                            ・・・MCU Related Headers
        ├ fir
        │      ├ fir_config_sample1.c                       ・・・FIR Moving-average Filter Configuration Definition Source
        │      ├ r_ctsu_fir_sample.c                        ・・・FIR Filter Processing Source
        │      └ r_ctsu_fir_sample.h                        ・・・FIR Filter Processing Header
        ├ iir
        │      ├ iir_config_sample1.c                       ・・・IIR Filter Configuration Definition Source
        │      ├ r_ctsu_iir_sample.c                        ・・・IIR Filter Processing Source
        │      └ r_ctsu_iir_sample.h                        ・・・IIR Filter Processing Header
        ├ median
        │      ├ median_config_sample1.c                    ・・・Median Filter Configuration Definition Source
        │      ├ r_ctsu_median_sample.c                     ・・・Median Filter Processing Source
        │      └ r_ctsu_median_sample.h                     ・・・Median Filter Processing Header
        └ spiir
               ├ spiir_config_sample1.c                     ・・・Single-pole IIR Low-pass Filter Configuration Definition Source
               ├ r_ctsu_spiir_sample.c                      ・・・Single-pole IIR Filter Processing Source
               └ r_ctsu_spiir_sample.h                      ・・・Single-pole IIR Filter Processing Header
```

7.1.3.3 RL78/G16 Group

The project configuration file and Smart Configuration generation file of the development environment have been omitted.

```
rl78g16_filter_sample
 │
 ├──QE-Touch
 │      ├ rl78g16_filter_sample_log_tuning20240325091840.log      ・・・QE Tuning Log
 │      └ rl78g16_filter_sample.tifcfg                            ・・・Touch Interface Configuration File
 │
 ├──qe_gen
 │      ├ qe_touch_config.c                                       ・・・Touch Configuration Source
 │      ├ qe_touch_config.h                                       ・・・Touch Configuration Header
 │      ├ qe_touch_define.h                                       ・・・Touch Definition Header
 │      └ qe_touch_sample.c                                       ・・・Touch Sample Application
 │
 ├──src
 │      └ rl78g16_filter_sample.c                                 ・・・main Files
 │
 └──filter_sample
        ├ filter_define_sample.h                                  ・・・MCU Related Headers
        ├ fir
        │      ├ fir_config_sample1.c                             ・・・FIR Moving-average Filter Configuration Definition Source
        │      ├ r_ctsu_fir_sample.c                              ・・・FIR Filter Processing Source
        │      └ r_ctsu_fir_sample.h                              ・・・FIR Filter Processing Header
        ├ iir
        │      ├ iir_config_sample1.c                             ・・・IIR Filter Configuration Definition Source
        │      ├ r_ctsu_iir_sample.c                              ・・・IIR Filter Processing Source
        │      └ r_ctsu_iir_sample.h                              ・・・IIR Filter Processing Header
        ├ median
        │      ├ median_config_sample1.c                          ・・・Median Filter Configuration Definition Source
        │      ├ r_ctsu_median_sample.c                           ・・・Median Filter Processing Source
        │      └ r_ctsu_median_sample.h                           ・・・Median Filter Processing Header
        └ spiir
               ├ spiir_config_sample1.c                           ・・・Single-pole IIR Low-pass Filter Configuration Definition Source
               ├ r_ctsu_spiir_sample.c                            ・・・Single-pole IIR Filter Processing Source
               └ r_ctsu_spiir_sample.h                            ・・・Single-pole IIR Filter Processing Header
```

7.1.4 How to Import the Sample Program

Import the "xxxx_filter_sample.zip " folder (xxxx indicates the MCU Group name) included in this sample code into your workspace using the e2studio import function.

Figure 7-1 shows how to import a sample project.

For operations after importing the project, refer to the Quick Start Guide or related Application Note corresponding to the MCU Family you are using in your development.

- RA Family
  Renesas RA Family RA2L1 Group Capacitive Touch Evaluation System Quick Start Guide (Q12QS0040)
- RX Family
  Using QE and FIT to Develop Capacitive Touch Applications (R01AN4516)
- RL78 Family
  RL78 Family Using QE and SIS to Develop Capacitive Touch Applications (R01AN5512)

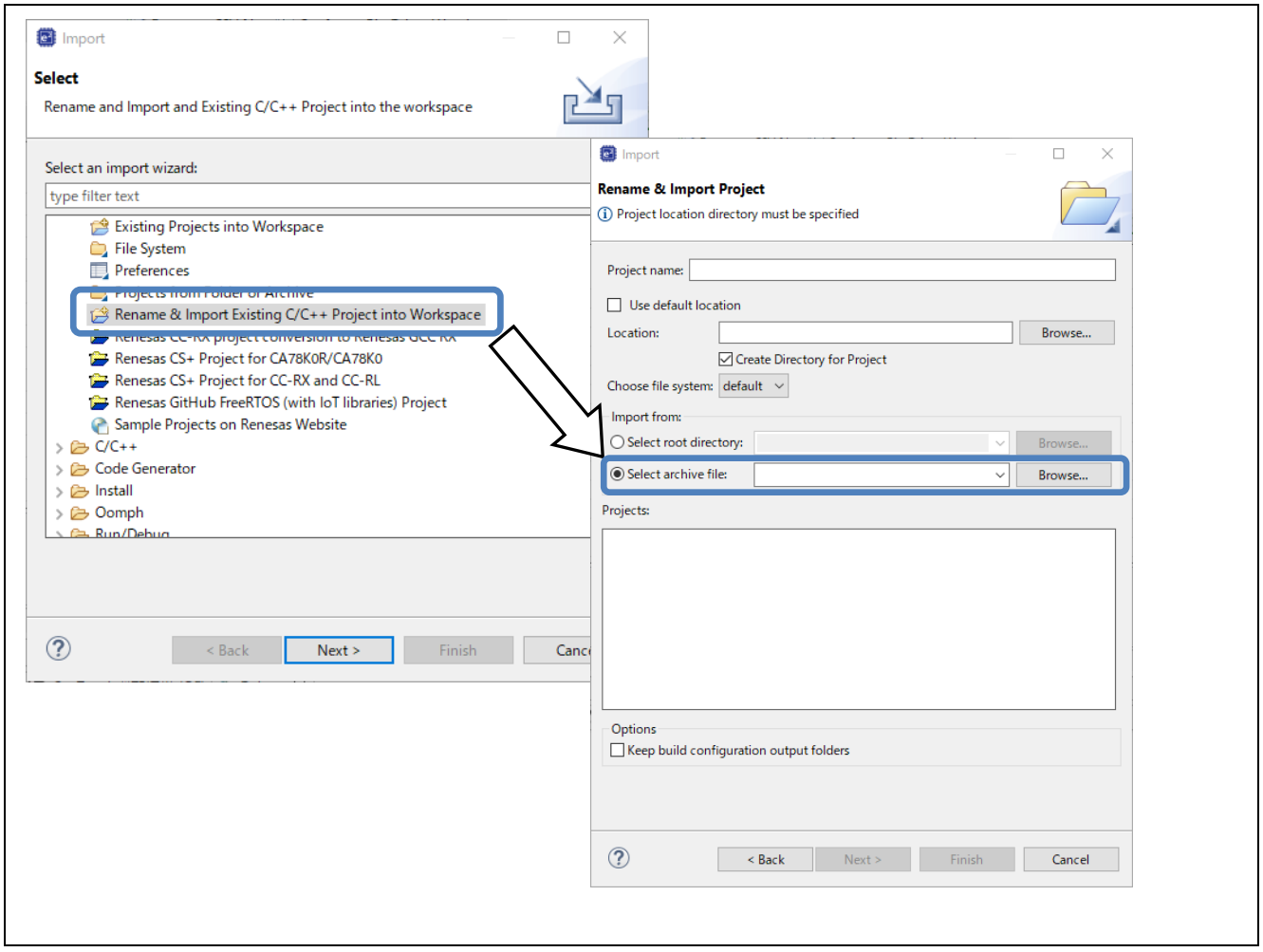


Figure 7-1 Importing the Sample Project

7.2 How to Use the Filter Sample Code

7.2.1 Procedure for Integration into an Existing Project

Use the following procedure to integrate software filters into an existing capacitive touch application.

1. Copy the filter_sample folder into the target project folder.
   You can delete folders from the filter_sample that do not indicate the name of the filter you are using.

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

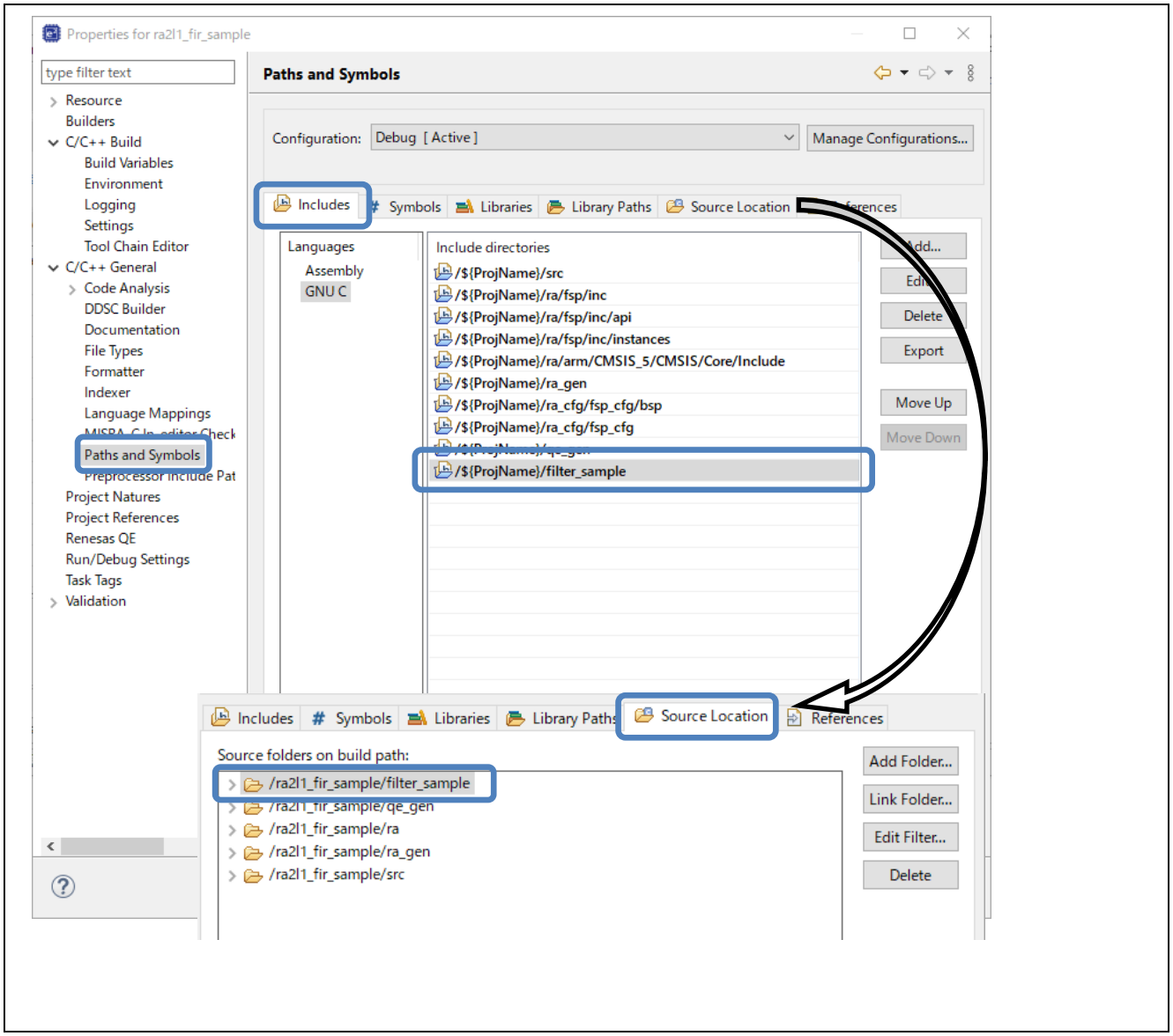


Figure 7-2 Embedding a Sample Program in an Existing Environment

3. Add the filter management data in the XXX_config_sample1.c (XXX indicates the name of the folder used) that corresponds to the method configuration of the touch interface.

Example: using the FIR filter for three methods

fir_config_sample1.c

```
fir_ctrl_t g_ctsu_fir_control;
fir_ctrl_t g_ctsu_fir_control2;
fir_ctrl_t g_ctsu_fir_control3;
```

Since the FIR filter is used for three methods, add three separate management data.

4. Change the XXX_DATA_SIZE definition in r_ctsu_XXX_sample.h (where XXX is the name of the filter used) to match the amount of touch measurement data in the method configuration of the touch interface that will use the software filter.
The number of touch measurement data for a method is twice the number of TS pins for the self-capacitance method and twice the number of matrices for the mutual capacitance method.
Define XXX_DATA_SIZE using the maximum value within the number of touch measurement data for the method using the filter.

Example: using the FIR filter for three methods with TS pin counts of 3/4/5

r_ctsu_fir_sample.h

```
#define FIR_DATA_SIZE    (5)
```

Change the number of pins to that of the FIR filter method with the largest number of TS pins

5. Add the include specification file that corresponds to the filter to be used in the qe_touch_sample.c file (or equivalent file) and add the filter processing implementation code (see Section 7.2.2).

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

Table 7-2 Include Files to be Added

Filter to be Used	Include Specification File
FIR filter	\fir\r_ctsu_fir_sample.h
IIR filter	\iir\r_ctsu_iir_sample.h
Single-pole IIR filter	\spiir\r_ctsu_spiir_sample.h
Median filter	\median\r_ctsu_median_sample.h

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 the sample code.
   If there are multiple touch interface configuration methods, change the CTSU driver configuration definition for all methods.

- Example definition of touch interface configuration

qe_touch_config.c (file generated by QE for Capacitive Touch)

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

ctsu_instance_ctrl_t g_qe_ctsu_ctrl_config01;

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


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

ctsu_instance_ctrl_t g_qe_ctsu_ctrl_config02;

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

Change to 1

Change to 1

7.2.2 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.

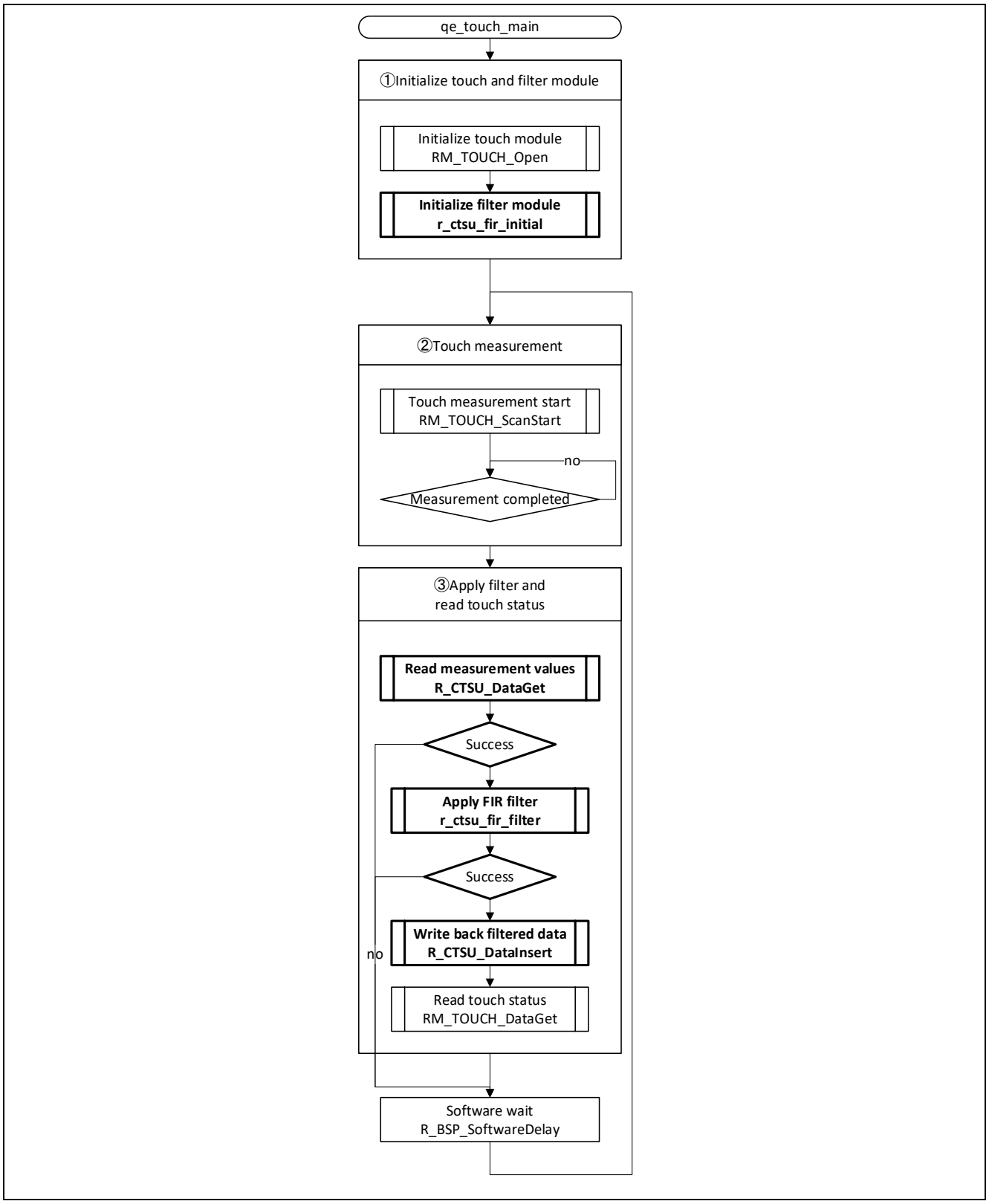


Figure 7-3 Sample Application Flowchart

This section describes the numbers indicated in **Figure 7-3**.

① Initialize filter

Check the valid range for the touch interface configuration definition and initialize the filter management data.

```
/* Check filter configuration & Initialize filter control data */
err = r_ctsu_fir_initial(&g_ctsu_fir_control, &g_fir_cfg);
if (FSP_SUCCESS != err)
{
 while (true) {}
}
```

② Touch measurement

Perform touch measurement and wait for measurement to be completed.

③ Apply filters and get touch input status

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.

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

```
/* FIR moving average filter sample */
err = R_CTSU_DataGet(g_qe_ctsu_instance_config01.p_ctrl, filter_buffer);
if(err == FSP_SUCCESS)
{
 err=r_ctsu_fir_filter(&g_ctsu_fir_control,&g_fir_cfg, filter_buffer);
 if(err == FSP_SUCCESS)
 {
    R_CTSU_DataInsert(g_qe_ctsu_instance_config01.p_ctrl, filter_buffer);
    err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl,
&button_status, NULL, NULL);
    if (FSP_SUCCESS == err)
    {
       /* TODO: Add your own code here. */
    }
 }
}
```

7.2.3 How to Adjust Filter Characteristics

7.2.3.1 Fixed Point Decimal Definition

The coefficients of the FIR/IIR/single-pole IIR filters are defined in the form of int16_t fixed point decimals.

The number of decimal places in the fixed point definition differs depending on the filter type.

Calculate the fixed point definition as "decimal value x 2 $^{\text{number of fixed-point digits}}$.

Table 7-3 shows examples of fixed point definitions.

Table 7-3 Fixed Point Definition Examples

Fractional Number	Number of Fixed-Point Digits	Decimal	Remarks
0.199951171875	14	3276=0.199951171875 x 16384	FIR、Single-pole IIR
-0.61376953125	11	-1257= -0.61376953125 x 2048	IIR

7.2.3.2 FIR Filters

Adjust the filter characteristics by changing the coefficient definition of the FIR filters.

Up to nine coefficients can be defined using a signed 16-bit fixed point with 14 fixed-point digits. They are treated as coefficients h(0) through h(8), in that order, as shown in **Figure 3-2**.

The number of taps indicates the number of defined coefficient values.

```
const int16_t g_fir_coefficient[] =
{
  3276,         /* h0 : 0.199951171875 */
  3276,         /* h1 : 0.199951171875 */
  3276,         /* h2 : 0.199951171875 */
  3276,         /* h3 : 0.199951171875 */
  3276,         /* h4 : 0.199951171875 */
};

const fir_config_t g_fir_cfg =
{
  .taps = 5,
  .p_coefficient = g_fir_coefficient,
};
```

Specify up to nine coefficient values using fixed decimal values with 14 fixed-point digits.

Specify the number of coefficient values

7.2.3.3 IIR Filters

Adjust the filter characteristics by changing the coefficient definition of the IIR filters.

Five coefficients can be defined using a signed 16-bit fixed point with 11 fixed-point digits. They are treated as coefficients b(0), b(1), b(2), a(1), and a(2), in that order, as shown in **Figure 4-2**.

After setting the coefficient, set a provisional value for the stabilization wait time and confirm the measurement value at the reset start.

If the stabilization wait time is insufficient, the reference value for touch measurement will be low, preventing touch detection from performing correctly. Adjust the stabilization wait time to ensure that touch measurement is performed correctly. Specify coefficient values in the order of coefficient b then coefficient a using fixed decimal values with 11 fixed-point digits.

```
const iir_config_t g_iir_cfg1 =
{
  .settlings = 3,
  .coefficient = {
   /* coefficient b */
   2034,                /* b0 : 0.9931640625    */
   -1257,               /* b1 : -0.61376953125  */
   2034,                /* b2 : 0.9931640625    */
   /* coefficient a */
   1257,                /* a1 : 0.61376953125   */
   -2021,               /* a2 : -0.98681640625  */
  }
};
```

Confirm the measurement value at reset start and adjust accordingly.

Specify coefficient values in the order of coefficient b then coefficient a using fixed decimal values with 11 fixed-point digits.

7.2.3.4 Cascaded IIR Filter Configuration

Create a third-order or higher IIR filter by cascading the IIR filters.

Execute IIR filter processing at least twice consecutively from the application.

Each successive filter processing requires separate management data and a separate configuration definition.

```
  err = R_CTSU_DataGet(g_qe_ctsu_instance_config02.p_ctrl, filter_buffer);
  if(err == FSP_SUCCESS)
  {
   err1=r_ctsu_iir_filter(&g_ctsu_iir_control1, &g_iir_cfg1, filter_buffer);
   err2=r_ctsu_iir_filter(&g_ctsu_iir_control2, &g_iir_cfg2, filter_buffer);
   if((err1 == FSP_SUCCESS) && (err1 == FSP_SUCCESS))
   {
      R_CTSU_DataInsert(g_                                          ffer);
      err = RM_TOUCH_DataG
&button_status, NULL, NULL
      if (FSP_SUCCESS == err)
      {
         /* TODO: Add your own code here. */
      }
   }
  }
```

Successive filter processes are implemented with separate management data and configuration definitions.

7.2.3.5 Single-pole IIR Filter

Adjust the filter characteristics by changing the coefficient definition of the single-pole IIR filters.

Two coefficients can be defined using a signed 16-bit fixed point with 14 fixed-point digits. They are treated as coefficients a and b, in that order, as shown in **Figure 5-2**.

After setting the coefficient, set a provisional value for the stabilization wait time and confirm the measurement value at the reset start.

If the stabilization wait time is insufficient, the reference value for touch measurement will be low, preventing touch detection from performing correctly. Adjust the stabilization wait time to ensure that touch measurement is performed correctly.

```
const spiir_config_t g_spiir_cfg =
{
  .settlings = 128,
  .coefficient = {
   /* coefficient a */
   15386,              /* a : 0.9390869140625   */
   /* coefficient b */
   997,                /* b : 0.06085205078125 */
  },
};
```

Confirm the measurement value at reset start and adjust accordingly.

Specify coefficient values in the order of coefficient b then coefficient a using fixed decimal values with 14 fixed-point digits.

7.2.3.6 Median Filters

Change the sampling rate of the median filter to adjust the width of noise that can be removed.

```
#define MEDIAN_SAMPLE_SIZE  (3)
```

Specify sampling size

8. 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)
- Using QE and FIT to Develop Capacitive Touch Applications (R01AN4516)
- RL78 Family Using QE and SIS to Develop Capacitive Touch Applications (R01AN4516)

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-touch

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 Added “Section 4. IIR Filters” Added IIR filter-related items to folder structure in Section 1.1 Corrected Figure 2.1 Added IIR filter-related items to file structure in Section 2.2 Corrected remarks regarding coefficient data type in Table 3.1 Corrected mistakes in Section 3.5.4 Added IIR filter-related items to Section
3.00	Nov.30.23	4 5 6 7 8 9 10 13 19 24 24 26 28 28 31 45 49 56 68	Added median filters related items to the folder structure in Section 1.1 Updated Table 1.1 (Operation Confirmation Conditions) Deleted description of planned functions from Section 2 Updated Table 2.1 (List of Components) Added items related to median filters to the file structure and added folder descriptions in Section 2.2 Added valid/invalid definitions to each filter in Table 2.2 Added median filters to Table 2.3 Added median filters to Table 2.12 Added median filters to Figure 2.5 Added median filters to Section 2.4.6 Added median filter initialization setting API to Section 2.4.6 Added median filter special note to Section 2.4.7 Added data size and incremental amount to Table 2.13 Updated filter processing execution time in Table 2.14 Corrected information in Section 3.3.1 Corrected information in Section 4.3.1 Corrected information in Table 4.4 Added Section 5 (Median Filters) Revised structure of Section 6
4.00	Jun.30.24	-	Complete revision

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

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/.