RX Family

ADC Module Using Firmware Integration Technology

Introduction

This application note describes the ADC module using Firmware Integration Technology. This module supports the functions of the 12-bit A/D converter. It is referred to below as the ADC FIT module.

Target Devices

The following is a list of devices that are currently supported by this API:

- RX110 Group,
- RX111 Group
- RX113 Group
- RX130 Group,
- RX231, RX230 Groups
- RX23W Group
- RX64M Group
- RX65N, RX651 Groups
- RX66T Group
- RX71M Group
- RX72M Group
- RX72T Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Confirmed Operation Environment".

Contents

1. Overview

1.1 ADC FIT Module

This module can be incorporated into projects in the form of APIs. Refer to 2.12, Adding the FIT Module to Your Project, for instructions for incorporating the ADC FIT module into projects.

1.2 ADC FIT Module Overview

The ADC FIT module supports the operating modes and functions listed below. The available functions differ depending on the MCU.

Table 1.1 lists the operating modes, and Table 1.2 the functions, supported by the ADC FIT module.

Table 1.1 Operating Modes Supported by ADC FIT Module

	RX110, RX111, RX113	RX130, RX230, RX231, RX23W, RX64M, RX65x, RX66T, RX71M, RX72M, RX72T
Single scan mode	○	○
Continuous scan mode	○	○
Group scan mode	○	○
Group scan mode (group priority control)	—	○

Table 1.2 Functions Supported by ADC FIT Module

	RX110, RX111, RX113	RX130, RX230, RX231, RX23W	RX64M, RX65x, RX71M, RX72M	RX66T, RX72T
Channel-dedicated sample-and-hold function	—	—	○	○
Variable sampling state count function	○	○	○	○
Self-diagnostic function	—	○	○	○
A/D-converted value addition mode	○	○	○	○
A/D-converted value average mode	—	○	○	○
Analog input disconnection detection assist function	—	○	○	○
Double trigger mode	○	○	○	○
12-/10-/8-bit conversion switching function	—	—	○	—
A/D data register automatic clear function	○	○	○	○
Extended analog input function	—	—	○	—
Comparison function	—	○	○	○
Channel conversion order setting function	—	—	—	○
Input signal amplification function (programmable gain amplifier)	—	—	—	○

The S12AD begins conversion when it receives a trigger. When the conversion is complete, a flag is set and an interrupt issued if enabled. If the S12AD is operating in a single scan mode, only one scan takes place per trigger. If the S12AD is operating in a continuous mode, scans continue indefinitely after the initial trigger occurs.

The majority of the driver serves to initialize the A/D peripheral and provide functions to read conversion results. With the ADC FIT module, settings which are common to all channels such as conversion alignment or addition count are set in the R_ADC_Open() call. Specific channel enabling is done via the R_ADC_Control() function. To retrieve conversion results, use the R_ADC_Read() function which retrieves a single conversion value or the R_ADC_ReadAll() function which retrieves all conversion registers.

The ADC FIT module supports the following 12-bit A/D Converter (S12AD) for each RX MCU.

Table 1.3 S12AD Supported by Each MCU

	S12ADb	S12ADC	S12ADE	S12ADFa	S12ADH
RX110	○				
RX111	○				
RX113	○				
RX130			○		
RX210	○				
RX230			○		
RX231			○		
RX23W			○		
RX64M		○			
RX65x				○	
RX66T					○
RX71M		○			
RX72M				○	
RX72T					○

1.3 API Overview

Table 1.4 lists the API functions contained in the ADC FIT module.

Table 1.4 API Functions

Functions	Description
R_ADC_Open	Initializes the 12-bit A/D converter.
R_ADC_Control	Makes function settings to the 12-bit A/D converter, performs interrupt control, and obtains the A/D conversion start/stop status.
R_ADC_Read	Reads the conversion result from the register for a single channel, sensor, double trigger, or self-diagnostic test.
R_ADC_ReadAll	Reads all registers in which conversion results are stored.
R_ADC_Close	Completes the A/D conversion being processed, disables interrupts, and ends A/D converter operation.
R_ADC_GetVersion	Returns the version number of the ADC FIT module.

1.4 Processing Examples

Figure 1.1 to Figure 1.4 show an initialization example for the ADC FIT module. Figure 1.5 and Figure 1.6 show examples of API function calls using the ADC FIT module.

The examples shown in Figure 1.1 to Figure 1.6 show all the relevant processing, with no distinction by MCU. In your projects, it is only necessary to execute the processing required by your MCU. Also, make sure to check the return value after calling an API function.

There are restrictions on the order in which commands are issued using the R_ADC_Control function. For details on issuing commands using the R_ADC_Control function, refer to 3.2, R_ADC_Control().

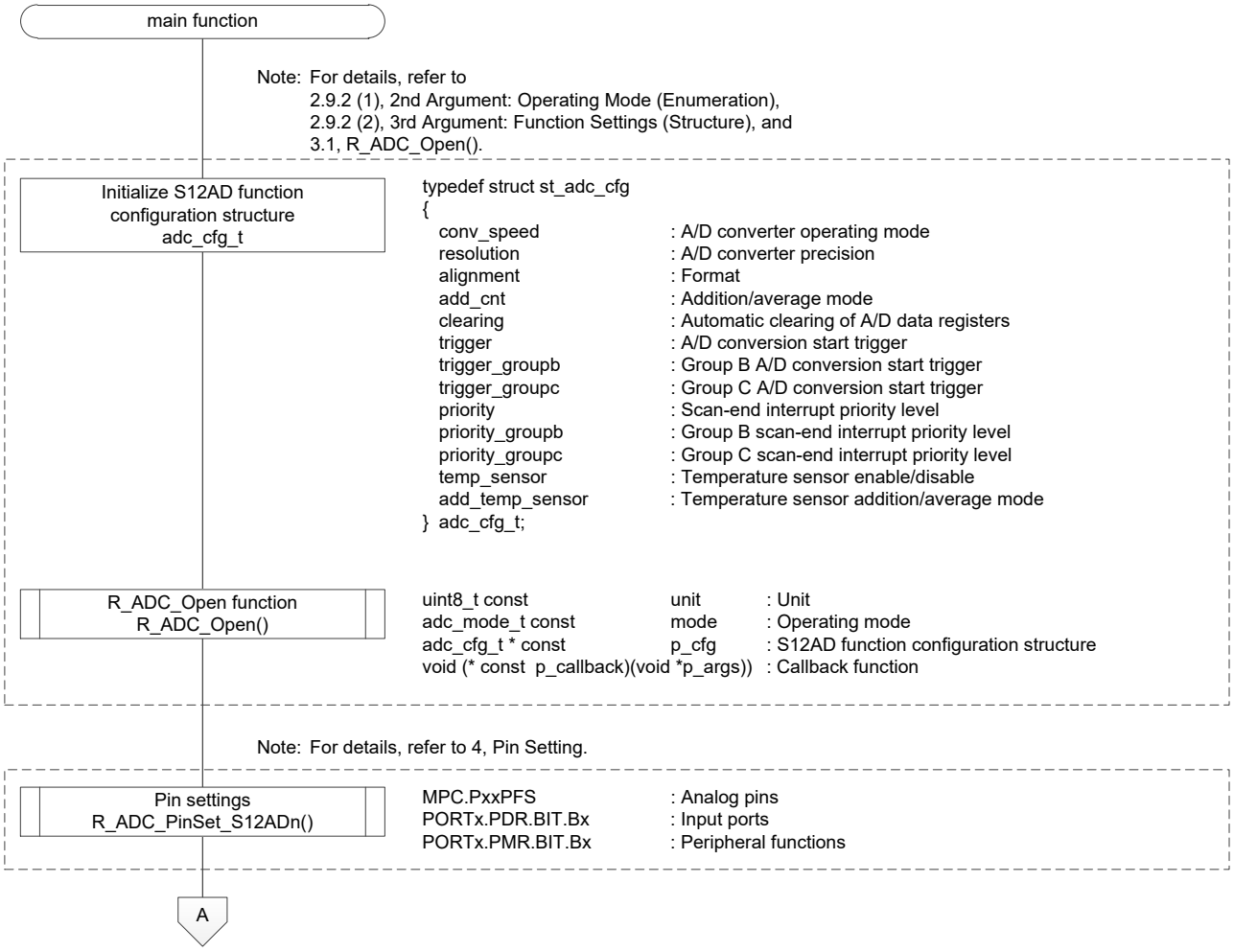

Figure 1.1 ADC FIT Module Initialization Example (1/4)

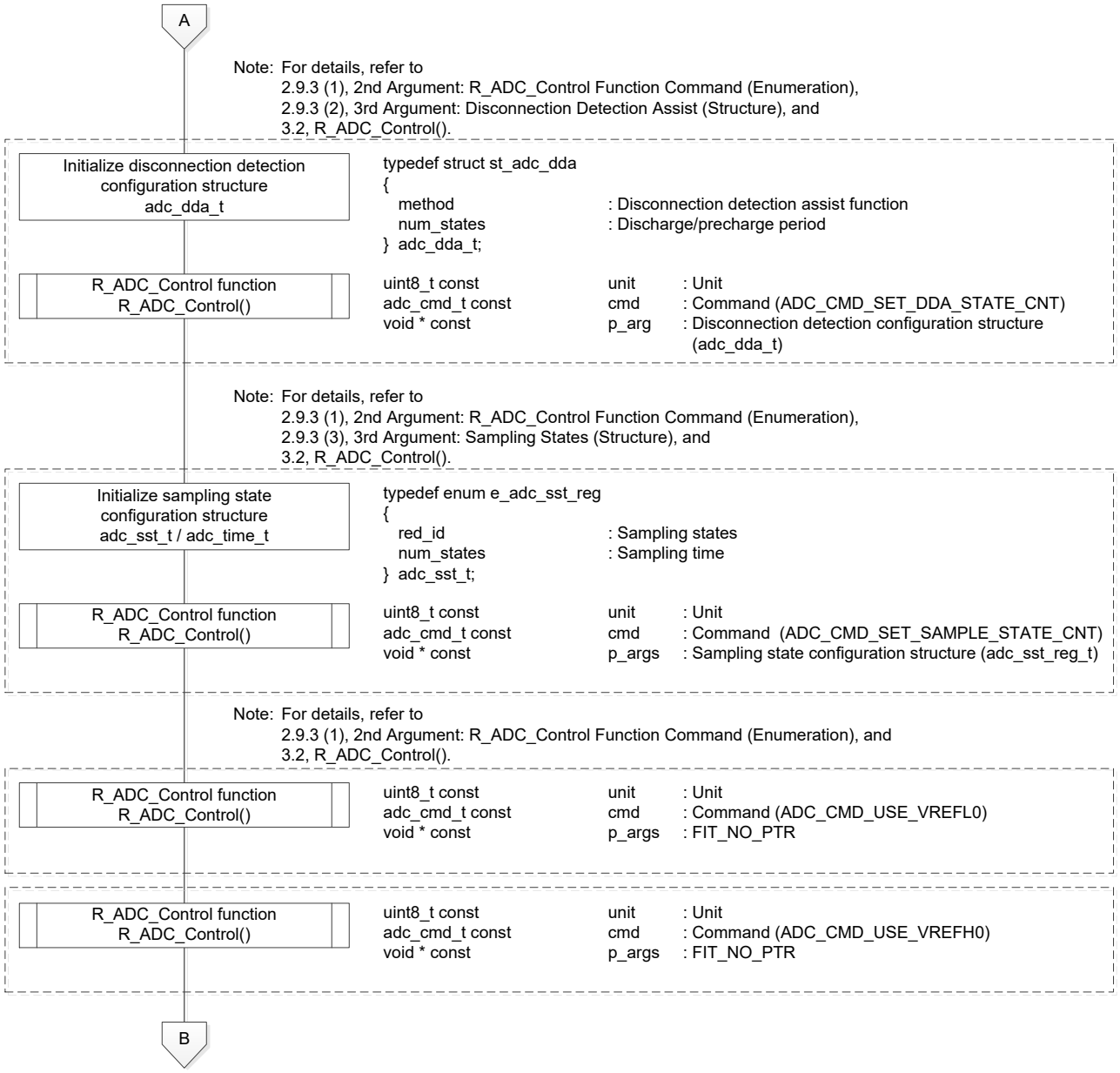

Figure 1.2 ADC FIT Module Initialization Example (2/4)

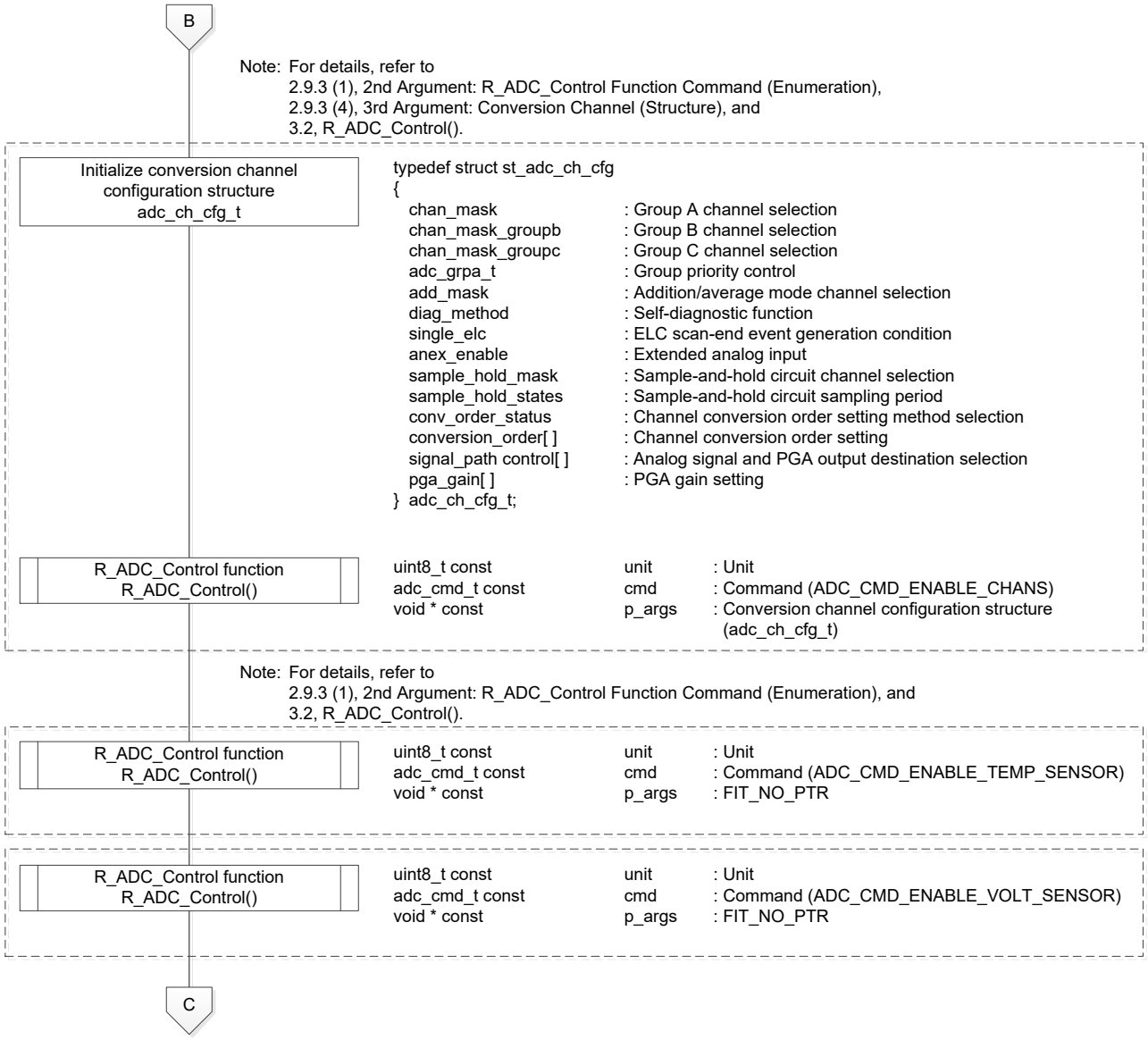

Figure 1.3 ADC FIT Module Initialization Example (3/4)

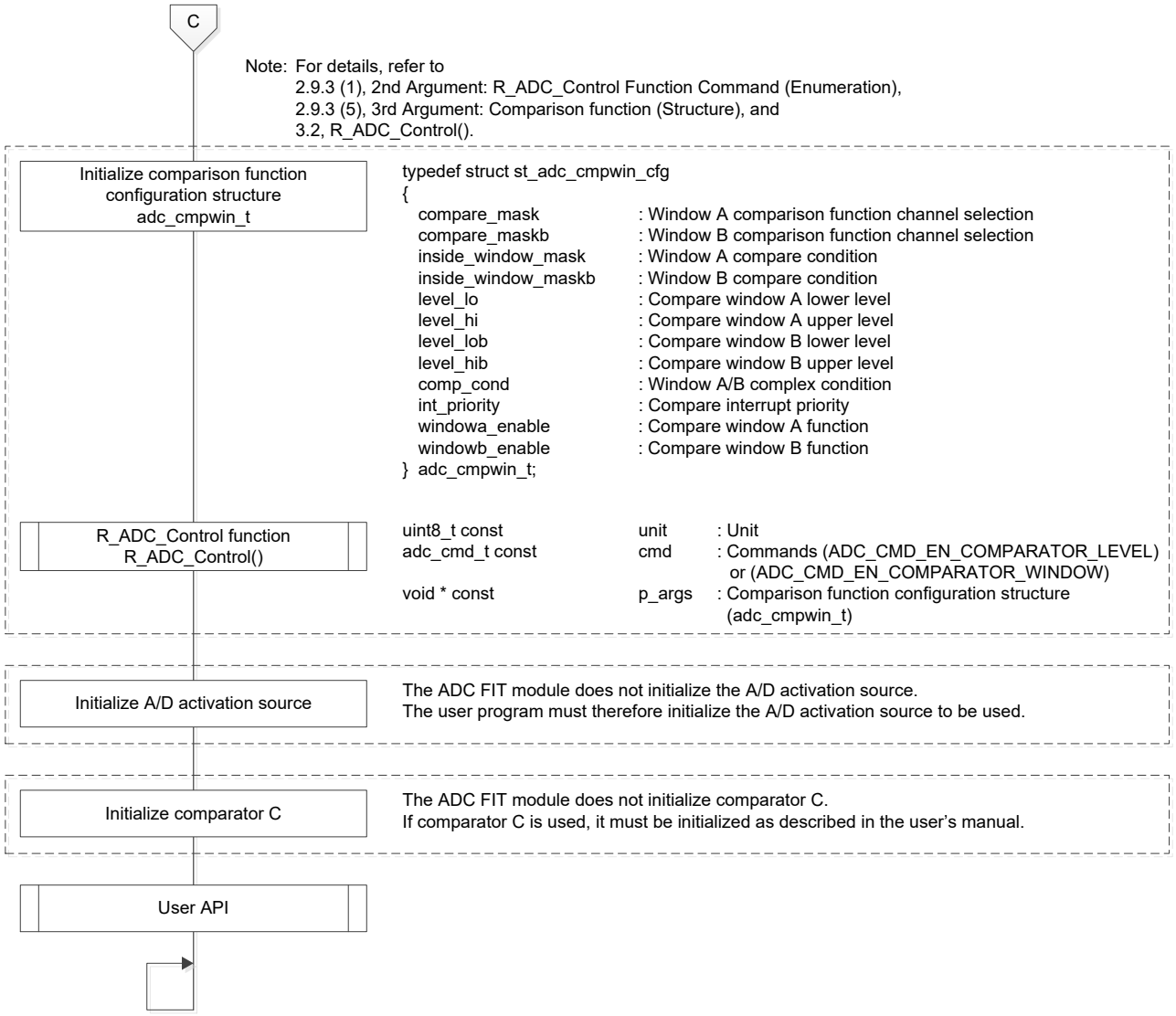

Figure 1.4 ADC FIT Module Initialization Example (4/4)

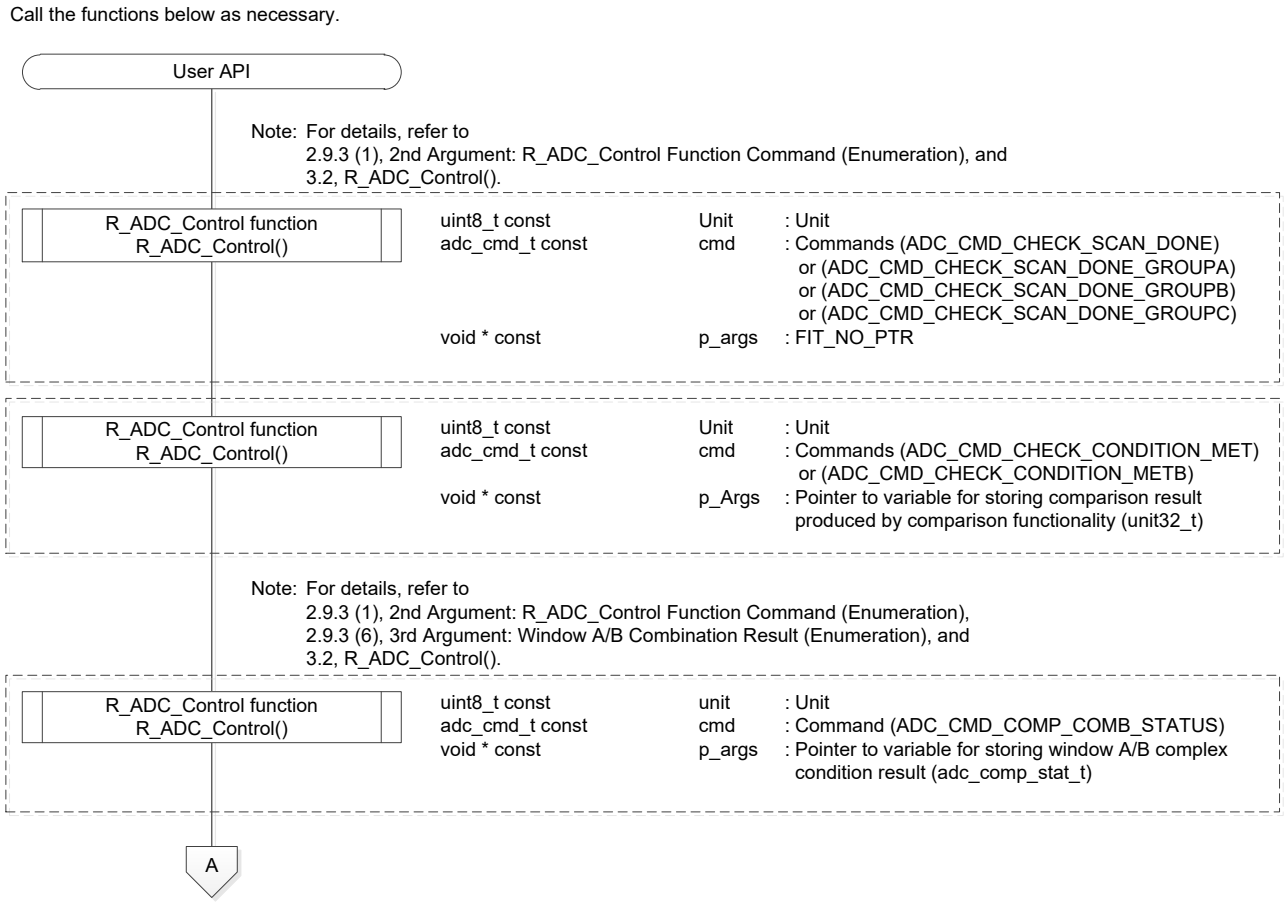

Figure 1.5 ADC FIT Module API Function Call Examples (1/2)

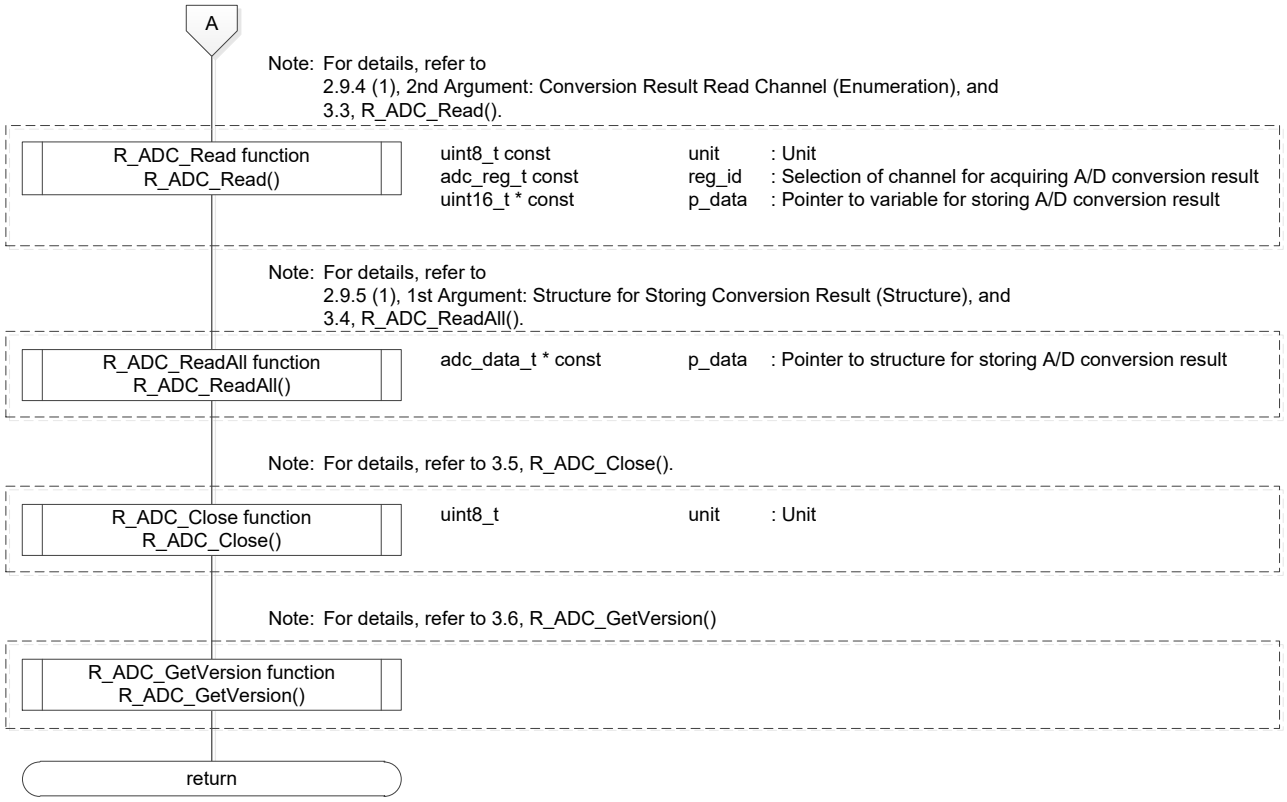

Figure 1.6 ADC FIT Module API Function Call Examples (2/2)

1.5 Restrictions

The registers, settings, and usage precautions differ depending on the operating mode of the 12-bit A/D converter. Use the APIs described in this application note in accordance with the description in the 12-bit A/D converter section of the hardware manual of your specific MCU.

Check to make sure that the state of the MCU and board match the settings in r_bsp_config.h before using the APIs. In particular, make sure that the package and power supply voltage settings are correct, as permanent damage could result if they are in error.

Use the latest version of the Renesas Board Support Package (r_bsp).

2. API Information

The operation of the ADC FIT module has been confirmed under the following conditions.

2.1 Hardware Requirements

This driver requires your MCU support the following features:

S12AD

2.2 Software Requirements

This driver is dependent on the following FIT module:

Renesas Board Support Package (r_bsp) Rev.5.20 or higher

2.3 Supported Toolchain

The operation of the ADC FIT module has been confirmed with the toolchain listed in 6.1, Confirmed Operation Environment.

2.4 Interrupt Vector

When the interrupt priority level is set to a value other than 0 in the R_ADC_Open() function, the interrupt (S12ADIn, S12GBADIn or GCADIn) for the interrupt source will be enabled.

Table 2.1 lists the interrupt vector used in the ADC FIT Module.

Table 2.1 Interrupt Vector Used in the ADC FIT Module

Device	Interrupt Vector
RX110, RX111, RX113, RX130, RX230, RX231, RX23W	S12ADI0 interrupt (vector No.: 102) GBADI interrupt (vector No.: 103)
RX64M, RX71M	S12ADI0 interrupt (vector No.: 190)*1 S12ADI1 interrupt (vector No.: 192)*1 S12GBADI0 interrupt (vector No.: 191)*1 S12GBADI1 interrupt (vector No.: 193)*1 GROUPBL1 interrupt (vector No.: 111) • S12CMPI0 interrupt (group interrupt source No.: 20) • S12CMPI1 interrupt (group interrupt source No.: 22)
RX65x, RX72M	S12ADI0 interrupt (vector No.: 186)*1 S12ADI1 interrupt (vector No.: 189)*1 S12GBADI0 interrupt (vector No.: 187)*1 S12GBADI1 interrupt (vector No.: 190)*1 S12GCADI0 interrupt (vector No.: 188)*1 S12GCADI1 interrupt (vector No.: 191)*1 GROUPBL1 interrupt (vector No.: 111) • S12CMPAI interrupt (group interrupt source No.: 20) • S12CMPBI interrupt (group interrupt source No.: 21) • S12CMPAI1 interrupt (group interrupt source No.: 22) • S12CMPBI1 interrupt (group interrupt source No.: 23)
RX66T, RX72T	S12ADI interrupt (vector No.: 128) S12ADI1 interrupt (vector No.: 132) S12ADI2 interrupt (vector No.: 136) S12GBADI interrupt (vector No.: 129) S12GBADI1 interrupt (vector No.: 133) S12GBADI2 interrupt (vector No.: 137) S12GCADI interrupt (vector No.: 130) S12GCADI1 interrupt (vector No.: 134) S12GCADI2 interrupt (vector No.: 138) GROUPBL1 interrupt (vector No.: 111) • S12CMPAI interrupt (group interrupt source No.: 20) • S12CMPBI interrupt (group interrupt source No.: 21) • S12CMPAI1 interrupt (group interrupt source No.: 22) • S12CMPBI1 interrupt (group interrupt source No.: 23) • S12CMPAI2 interrupt (group interrupt source No.: 18) • S12CMPBI2 interrupt (group interrupt source No.: 19)

Note 1. The interrupt vector numbers for software configurable interrupt B shown here are the default values specified in the board support package FIT module (BSP module).

2.5 Header Files

All API calls and their supporting interface definitions are located in the file “r_s12ad_rx_if.h” and this file should be included by the User’s application.

Build-time configuration options are selected or defined in the file “r_s12ad_rx_config.h”.

2.6 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in *stdint.h*.

2.7 Configuration Overview

All configurable options that can be set at build time are located in the file “r_s12ad_rx_config.h”. A summary of these settings are provided in the following table:

Configuration options in *r_s12ad_rx_config.h*	
ADC_CFG_PARAM_CHECKING_ENABLE Note: The default value is “BSP_CFG_PARAM_CHECKING_ENABLE.”	Selects whether or not processing for parameter checking is included in the object code. When 0 is selected, processing for parameter checking is omitted from the object code, resulting in a smaller code size. 0 = Omit parameter checking from object code. 1 = Include parameter checking in object code. The default value of “BSP_CFG_PARAM_CHECKING_ENABLE” is a setting in the BSP configuration options.

2.8 Code Sizes

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the RX100 Series, RX200 Series, and RX600 Series, respectively.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview.

The values in the table below are confirmed under the following conditions.

Module Revision: r_s12ad_rx rev.4.00

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of "lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201803

(The option of "lang = c99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.11.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	2,634 bytes	2,077 bytes	4,372 bytes	3,316 bytes	3,763 bytes	2,935 bytes
	RAM	12 bytes		12 bytes		8 bytes	
	STACK *1	164 bytes		-		124 bytes	
RX231	ROM	2,636 bytes	2,079 bytes	4,460 bytes	3,396 bytes	3,760 bytes	2,932 bytes
	RAM	12 bytes		12 bytes		8 bytes	
	STACK *1	164 bytes		-		124 bytes	
RX65N	ROM	5,595 bytes	4,424 bytes	9,588 bytes	7,324 bytes	7,777 bytes	6,147 bytes
	RAM	40 bytes		40 bytes		32 bytes	
	STACK *1	188 bytes		-		148 bytes	

Note 1. The sizes of maximum usage stack of Interrupts functions is included.

2.9 Arguments

The structures and enumerations used as arguments by the API functions are listed below. Most of the parameters used by the API functions are defined as enumerations. This is in order to reduce the number of errors when type checking is performed.

These structures and enumerations are defined in prototype declarations and also in r_s12ad_rx_if.h and r_s12ad_rx[MCU]_if.h.

The structures and enumerations available for use differ depending on the MCU.

2.9.1 Structures and Enumerations Used as Arguments for Callback Functions

(1) 1st Argument: Callback Function Status (Structure)

```
typedef struct st_adc_cb_args
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_cb_args_t;
```

Member	Description
abc_cb_evt_t event	Indicates the type of event.
uint32_t compare_flags	Stores the comparison result for each channel of window A. The comparison result for channel n corresponds to bit n. 0: Comparison condition not met. 1: Comparison condition met.
uint32_t compare_flagsb	Stores the comparison result for each channel of window B. The comparison result for channel n corresponds to bit n. 0: Comparison condition not met. 1: Comparison condition met.
uint8_t unit	Indicates the unit generating the event.

(a) 1st Argument Structure Members: Callback Function Events (Enumeration)

```
typedef enum e_adc_cb_evt
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_cb_evt_t;
```

Member	Description
ADC_EVT_SCAN_COMPLETE	Indicates completion of single scan A/D conversion or A/D conversion of group A.
ADC_EVT_SCAN_COMPLETE_GROUPB	Indicates completion of A/D conversion of group B.
ADC_EVT_SCAN_COMPLETE_GROUPC	Indicates completion of A/D conversion of group C.
ADC_EVT_CONDITION_MET	Indicates that the window A comparison condition was met.
ADC_EVT_CONDITION_METB	Indicates that the window B comparison condition was met.

2.9.2 Structures and Enumerations Used as Arguments for R_ADC_Open Function

(1) 2nd Argument: Operating Mode (Enumeration)

```
typedef enum e_adc_mode
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_mode_t;
```

Member	Description
ADC_MODE_SS_TEMPERATURE	A/D conversion is performed on the temperature sensor output in single scan mode. For channel, select the temperature sensor.
ADC_MODE_SS_INT_REF_VOLT	A/D conversion is performed on the internal reference voltage in single scan mode. For channel, select the internal reference voltage.
ADC_MODE_SS_ONE_CH	A/D conversion is performed on one channel in single scan mode. For channel, select a single channel.*1
ADC_MODE_SS_MULTI_CH	A/D conversion is performed on multiple channels in single scan mode.*1*3
ADC_MODE_CONT_ONE_CH	A/D conversion is performed on one channel in continuous scan mode. For channel, select a single channel.*1*2
ADC_MODE_CONT_MULTI_CH	A/D conversion is performed on multiple channels in continuous scan mode.*1*2
ADC_MODE_SS_ONE_CH_DBLTRIG	A/D conversion is performed on one channel in double trigger mode. For channel, select a single channel.*1*2
ADC_MODE_SS_MULTI_CH_GROUPED	A/D conversion is performed on multiple channels using two groups (group A and group B). Select different channels for group A and group B.*1*3
ADC_MODE_SS_MULTI_CH_GROUPED_GROUPC	A/D conversion is performed on multiple channels using three groups (group A, group B, and group C). Select different channels for each group.*3
ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A	A/D conversion is performed on multiple channels using two groups (group A and group B). Operation is in double trigger mode for group A. For group A, select a single channel only, and for group B, select channels other than that selected for group A.*1*2
ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A_GROUPC	A/D conversion is performed on multiple channels using three groups (group A, group B, and group C). Operation is in double trigger mode for group A. For group A, select a single channel only. Also, select different channels for each group.*2

Note 1. On the S12ADb and S12ADE it is not possible to select the internal reference voltage and temperature sensor.

Note 2. On the S12ADH it is not possible to select the internal reference voltage and temperature sensor.

Note 3. On the S12ADH it is not possible to select the internal reference voltage or the temperature sensor and a channel with analog input at the same time.

(2) 3rd Argument: Function Settings (Structure)

```
typedef struct st_adc_cfg
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_cfg_t;
```

Member	Description
adc_speed_t conv_speed	Specifies the operating mode for A/D conversion.
adc_res_t resolution	Specifies the precision of A/D conversion. The lower the resolution, the shorter the conversion time.
adc_align_t alignment	Specifies the format.*1
adc_add_t add_cnt	Specifies the addition/average mode.
adc_clear_t clearing	Enables/disables automatic clearing of A/D data registers.
adc_trig_t trigger	Specifies the start trigger for A/D conversion.
adc_trig_t trigger_groupb	Specifies the start trigger for A/D conversion of group B.
adc_trig_t trigger_groupc	Specifies the start trigger for A/D conversion of group C.
uint8_t priority	Sets the priority (0 to 15) of the S12ADIn interrupt. Specifying 0 disables the S12ADIn interrupt.
uint8_t priority_groupb	Sets the priority (0 to 15) of the S12GBADIn and GBADIn interrupts. Specifying 0 disables the S12GBADIn and GBADIn interrupts.
uint8_t priority_groupc	Sets the priority (0 to 15) of the S12GCADIn interrupt. Specifying 0 disables the S12GCADIn interrupt.
adc_temp_t temp_sensor	Specifies whether or not the temperature sensor is used.
adc_add_temp_t add_temp_sensor	Specifies whether the temperature sensor operates in addition/average mode.

Note 1. On the S12ADb the setting of this member has no effect when addition mode is enabled.

(a) 3rd Argument Structure Members: Conversion Operation (Enumeration)

```
typedef enum e_adc_speed
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_speed_t;
```

Member	Description
ADC_CONVERT_SPEED_PCLK_DIV8	Selects PCLK/8 as the A/D conversion clock.
ADC_CONVERT_SPEED_PCLK_DIV4	Selects PCLK/4 as the A/D conversion clock.
ADC_CONVERT_SPEED_PCLK_DIV2	Selects PCLK/2 as the A/D conversion clock.
ADC_CONVERT_SPEED_PCLK	Selects PCLK as the A/D conversion clock.
ADC_CONVERT_SPEED_DEFAULT	Selects the default setting.
ADC_CONVERT_SPEED_NORM	Selects normal conversion operation.
ADC_CONVERT_SPEED_HIGH	Selects high-speed conversion operation.
ADC_CONVERT_CURRENT_LOW	Selects low-current conversion operation.

(b) 3rd Argument Structure Members: Conversion Precision (Enumeration)

```
typedef enum e_adc_res
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_res_t;
```

Member	Description
ADC_RESOLUTION_12_BIT	A/D conversion is performed at 12-bit precision.
ADC_RESOLUTION_10_BIT	A/D conversion is performed at 10-bit precision.
ADC_RESOLUTION_8_BIT	A/D conversion is performed at 8-bit precision.

(c) 3rd Argument Structure Members: Data Register Format (Enumeration)

```
typedef enum e_adc_align
{
      /* Refer to the table below for the members. */
} adc_align_t;
```

Member	Description
ADC_ALIGN_RIGHT	A/D conversion results are stored in right-justified format
ADC_ALIGN_LEFT	A/D conversion results are stored in left-justified format

(d) 3rd Argument Structure Members: Converted Value Addition/Average Mode (Enumeration)

```
typedef enum e_adc_add
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_add_t;
```

Member	Description
ADC_ADD_OFF	Neither addition nor average mode is used.
ADC_ADD_TWO_SAMPLES	Conversion is performed twice. (Addition is performed once.)
ADC_ADD_THREE_SAMPLES	Conversion is performed three times. (Addition is performed twice.)
ADC_ADD_FOUR_SAMPLES	Conversion is performed four times. (Addition is performed three times.)
ADC_ADD_ SIXTEEN _SAMPLES	Conversion is performed 16 times. (Addition is performed 15 times.)
ADC_ADD_AVG_2_SAMPLES	The average of two conversion values is used.
ADC_ADD_AVG_4_SAMPLES	The average of four conversion values is used.

(e) 3rd Argument Structure Members: Data Register Automatic Clearing (Enumeration)

```
typedef enum e_adc_clear
{
      /* Refer to the table below for the members. */
} adc_clear_t;
```

Member	Description
ADC_CLEAR_AFTER_READ_OFF	A/D data registers are not cleared automatically.
ADC_CLEAR_AFTER_READ_ON	A/D data registers are cleared automatically.

(f) 3rd Argument Structure Members: Conversion Start Trigger (Enumeration)

```
typedef enum e_adc_trig
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_trig_t;
```

Member	Description
ADC_TRIG_ASYNC_ADTRG	External trigger (ADTRG#)
ADC_TRIG_SYNC_TRG0AN	MTU0 TGRA
ADC_TRIG_SYNC_TRG0BN	MTU0 TGRB
ADC_TRIG_SYNC_TRG1AN	MTU1 TGRA
ADC_TRIG_SYNC_TRG2AN	MTU2 TGRA
ADC_TRIG_SYNC_TRG3AN	MTU3 TGRA
ADC_TRIG_SYNC_TRGAN	MTUx TGRA
ADC_TRIG_SYNC_TRGAN_OR_UDF4N	MTUx TGRA or MTU4 underflow (complementary PWM)
ADC_TRIG_SYNC_TRG4AN_OR_UDF4N	MTU4 TGRA or MTU4 underflow (complementary PWM)
ADC_TRIG_SYNC_TRG6AN	MTU6 TGRA
ADC_TRIG_SYNC_TRG7AN_OR_UDF7N	MTU7 TGRA or MTU7 underflow (complementary PWM)
ADC_TRIG_SYNC_TRG0EN	MTU0 TGRE
ADC_TRIG_SYNC_TRG0FN	MTU0 TGRF
ADC_TRIG_SYNC_TRG4AN	MTU4 TADCORA
ADC_TRIG_SYNC_TRG4BN	MTU4 TADCORB
ADC_TRIG_SYNC_TRG4AN_OR_TRG4BN	MTU4 TADCORA or TADCORB
ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN	MTU4 TADCORA and TADCORB
ADC_TRIG_SYNC_TRG7AN	MTU7 TADCORA
ADC_TRIG_SYNC_TRG7BN	MTU7 TADCORB
ADC_TRIG_SYNC_TRG7AN_OR_TRG7BN	MTU7 TADCORA or TADCORB
ADC_TRIG_SYNC_TRG7AN_AND_TRG7BN	MTU7 TADCORA and TADCORB
ADC_TRIG_SYNC_TRG9AN	MTU9 TGRA
ADC_TRIG_SYNC_TRG9EN	MTU9 TGRE
ADC_TRIG_SYNC_TRG0AN_OR_TRG0EN	MTU0 TGRA or MTU0 TGRE
ADC_TRIG_SYNC_TRG9AN_OR_TRG9EN	MTU9 TGRA or MTU9 TGRE
ADC_TRIG_SYNC_TRG0AN_OR_TRG9AN	MTU0 TGRA or MTU9 TGRA
ADC_TRIG_SYNC_TRG0EN_OR_TRG9EN	MTU0 TGRE or MTU9 TGRE
ADC_TRIG_SYNC_TRG9AN_AND_TRG9EN	MTU9 TGRA and MTU9 TGRE
ADC_TRIG_SYNC_TRG0AN_AND_TRG0EN	MTU0 TGRA and MTU0 TGRE
ADC_TRIG_SYNC_TRG0AN_AND_TRG9AN	MTU0 TGRA and MTU9 TGRA
ADC_TRIG_SYNC_TRG0EN_AND_TRG9EN	MTU0 TGRE and MTU9 TGRE
ADC_TRIG_SYNC_GTADTR0AN	GPT0 GTADTRA
ADC_TRIG_SYNC_GTADTR0BN	GPT0 GTADTRB
ADC_TRIG_SYNC_GTADTR1AN	GPT1 GTADTRA
ADC_TRIG_SYNC_GTADTR1BN	GPT1 GTADTRB
ADC_TRIG_SYNC_GTADTR2AN	GPT2 GTADTRA
ADC_TRIG_SYNC_GTADTR2BN	GPT2 GTADTRB
ADC_TRIG_SYNC_GTADTR3AN	GPT3 GTADTRA
ADC_TRIG_SYNC_GTADTR3BN	GPT3 GTADTRB
ADC_TRIG_SYNC_GTADTR0AN_OR_GTADTR0BN	GPT0 GTADTRA or GTADTRB
ADC_TRIG_SYNC_GTADTR1AN_OR_GTADTR1BN	GPT1 GTADTRA or GTADTRB

Member	Description
ADC_TRIG_SYNC_GTADTR2AN_OR_GTADTR2BN	GPT2 GTADTRA or GTADTRB
ADC_TRIG_SYNC_GTADTR3AN_OR_GTADTR3BN	GPT3 GTADTRA or GTADTRB
ADC_TRIG_SYNC_TMRTRG0AN	TMR0 TCORA
ADC_TRIG_SYNC_TMRTRG2AN	TMR2 TCORA
ADC_TRIG_SYNC_TMRTRG4AN	TMR4 TCORA
ADC_TRIG_SYNC_TMRTRG6AN	TMR6 TCORA
ADC_TRIG_SYNC_TPUTRG0AN	TPU0 TRGA
ADC_TRIG_SYNC_TPUTRGAN	TPUx TRGA
ADC_TRIG_SYNC_TEMPS	Temperature sensor
ADC_TRIG_SYNC_ELC	ELC
ADC_TRIG_SYNC_ELCTRG0	ELCTRG0
ADC_TRIG_SYNC_ELCTRG1	ELCTRG1
ADC_TRIG_SYNC_ELCTRG0_OR_ELCTRG1	ELCTRG0 or ELCTRG1
ADC_TRIG_SOFTWARE	Software trigger
ADC_TRIG_NONE	No trigger source selected

(g) 3rd Argument Structure Members: Temperature Sensor (Enumeration)

```
typedef enum e_adc_temp
{
      /* Refer to the table below for the members. */
} adc_temp_t;
```

Member	Description
ADC_TEMP_SENSOR_NOT_AD_CONVERTED	No A/D conversion is performed on the temperature sensor output.
ADC_TEMP_SENSOR_AD_CONVERTED	A/D conversion is performed on the temperature sensor output in single scan mode and in group A in group scan mode.
ADC_TEMP_SENSOR_AD_CONVERTED_GROUPB	A/D conversion is performed on the temperature sensor output in group B in group scan mode.
ADC_TEMP_SENSOR_AD_CONVERTED_GROUPC	A/D conversion is performed on the temperature sensor output in group C in group scan mode.

(h) 3rd Argument Structure Members: Temperature Sensor Addition/Average Mode (Enumeration)

```
typedef enum e_adc_add_temp
{
      /* Refer to the table below for the members. */
} adc_add_temp_t;
```

Member	Description
ADC_TEMP_SENSOR_ADD_OFF	The temperature sensor’s addition/average mode is not used.
ADC_TEMP_SENSOR_ADD_ON	The temperature sensor’s addition/average mode is used.

2.9.3 Structures and Enumerations Used as Arguments for R_ADC_Control Function

(1) 2nd Argument: R_ADC_Control Function Command (Enumeration)

```
typedef enum e_adc_cmd
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_cmd_t;
```

What is specified using the 3rd argument (p_args) of the R_ADC_Control function will differ depending on the command used. A list of commands and the MCUs on which they can be used is presented below. For commands that do not use parameters, specify FIT_NO_PTR as the 3rd argument of the R_ADC_Control function.

Member	Description
ADC_CMD_USE_INT_VOLT_AS_HVREF	Uses the internal reference voltage as the high-side reference voltage. No parameters are used.
ADC_CMD_USE_VREFL0	Uses VREFL0 as the low-side reference voltage. No parameters are used.
ADC_CMD_USE_VREFH0	Uses VREFH0 as the high-side reference voltage. No parameters are used.
ADC_CMD_SET_DDA_STATE_CNT	Configures the A/D disconnection detection assist function. Specify the disconnection detection configuration structure (adc_dda_t) as the parameter.
ADC_CMD_SET_SAMPLE_STATE_CNT	Changes the number of A/D sampling states. Specify the sampling state configuration structure (adc_time_t or adc_sst_t) as the parameter.
ADC_CMD_ENABLE_CHANS	Specifies the A/D conversion channel. Specify the conversion channel configuration structure (adc_ch_cfg_t) as the parameter.
ADC_CMD_ENABLE_TEMP_SENSOR	Enables the temperature sensor. No parameters are used.
ADC_CMD_ENABLE_VOLT_SENSOR	Enables the internal reference voltage sensor. No parameters are used.
ADC_CMD_EN_COMPARATOR_LEVEL	Specifies that the comparison function is used with the window function disabled (threshold comparison). Specify the comparison function configuration structure (adc_cmpwin_t) as the parameter.
ADC_CMD_EN_COMPARATOR_WINDOW	Specifies that the comparison function is used with the window function enabled (range comparison). Specify the comparison function configuration structure (adc_cmpwin_t) as the parameter.
ADC_CMD_COMP_COMB_STATUS	Gets the window A/B complex condition result. Specify a pointer to the combination result monitor (adc_comp_stat_t) variable as the parameter.
ADC_CMD_ENABLE_TRIG	Enables A/D conversion start by synchronous or asynchronous trigger. No parameters are used.
ADC_CMD_SCAN_NOW	Enables A/D conversion start by software trigger. No parameters are used.
ADC_CMD_CHECK_SCAN_DONE	Checks whether A/D conversion is in progress in single scan mode. No parameters are used.

Member	Description
ADC_CMD_CHECK_SCAN_DONE_GROUPA	Checks whether A/D conversion of group A is in progress in group scan mode. No parameters are used.
ADC_CMD_CHECK_SCAN_DONE_GROUPB	Checks whether A/D conversion of group B is in progress in group scan mode. No parameters are used.
ADC_CMD_CHECK_SCAN_DONE_GROUPC	Checks whether A/D conversion of group C is in progress in group scan mode. No parameters are used.
ADC_CMD_CHECK_CONDITION_MET	Gets the comparison result produced by the comparison function.*1 Specify a pointer to the uint32_t variable storing the comparison result as the parameter.
ADC_CMD_CHECK_CONDITION_METB	Gets the comparison result produced by the group B comparison function.*1 Specify a pointer to the uint32_t variable storing the comparison result as the parameter.
ADC_CMD_DISABLE_TRIG	Disables A/D conversion start by synchronous or asynchronous trigger. No parameters are used.
ADC_CMD_DISABLE_INT	Disables the S12ADI interrupt. No parameters are used.
ADC_CMD_ENABLE_INT	Enables the S12ADI interrupt. No parameters are used.
ADC_CMD_DISABLE_INT_GROUPB	Disables the GBADI/S12GBADI interrupt. No parameters are used.
ADC_CMD_ENABLE_INT_GROUPB	Enables the GBADI/S12GBADI interrupt. No parameters are used.
ADC_CMD_DISABLE_INT_GROUPC	Disables the S12GCADI interrupt. No parameters are used.
ADC_CMD_ENABLE_INT_GROUPC	Enables the S12GCADI interrupt. No parameters are used.

Note 1. After execution of this command, the comparison result is initialized to 0 (comparison condition not met). Therefore, this command must be executed once only after A/D conversion completes.

(2) 3rd Argument: Disconnection Detection Assist (Structure)

```
typedef struct st_adc_dda
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_dda_t;
```

Member	Description
adc_charge_t method	Makes disconnection detection assist (discharge/precharge) settings.
uint8_t num_states	Sets the number of states in the discharge/precharge period. The lower limit value for the number of states in the discharge/precharge period differs depending on the MCU. Check the user's manual before making the setting. A setting of 0 disables the disconnection detection assist function.

(a) 3rd Argument Structure Members: Discharge/Precharge (Enumeration)

```
typedef enum e_adc_charge
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_charge_t;
```

Member	Description
ADC_DDA_DISCHARGE	Selects discharge.
ADC_DDA_PRECHARGE	Selects precharge.
ADC_DDA_OFF	The disconnection detection function is not used.

(3) 3rd Argument: Sampling States (Structure)

```
typedef struct st_adc_sst    /* Defined by st_adc_time for some MCUs. */
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_sst_t;    /* Defined by st_adc_time_t for some MCUs. */
```

Member	Description
adc_sst_reg_t reg_id	Selects the channel to which the sampling state setting is applied.
uint8_t num_states	Sets the number of sampling states. The lower limit value for the number of sampling states differs depending on the MCU. Check the user's manual before making the setting.

(a) 3rd Argument Structure Members: Sampling State Setting Channel (Enumeration)

```
typedef enum e_adc_sst_reg
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_sst_reg_t;
```

Member	Description
ADC_SST_CHn (n = channel number)	Selects channel n.*1 Only channels implemented on your MCU may be selected.
ADC_SST_CHi_TO_j (j and i = channel numbers)	Selects channels i to j.*1
ADC_SST_TEMPERATURE	Selects the temperature sensor.
ADC_SST_VOLTAGE	Selects the internal reference voltage.

Note 1. The available channels differ depending on the MCU and the pin count.

(4) 3rd Argument: Conversion Channel (Structure)

```
typedef struct st_adc_ch_cfg
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_ch_cfg_t;
```

Member	Description
uint32_t chan_mask	Selects the channels to be used.*1*2
uint32_t chan_mask_groupb	Selects the channels to be used by group B.*1*2 If group B is not used, specify ADC_MASK_GROUPB_OFF.
uint32_t chan_mask_groupc	Selects the channels to be used by group C.*1*2 If group C is not used, specify ADC_MASK_GROUPC_OFF.
adc_grpa_t priority_groupa	Configures group priority control operation.
uint32_t add_mask	Selects the channels to be used in addition mode.*1*3 If addition mode is not used, specify ADC_MASK_ADD_OFF. If addition mode is used, select from among the channels selected by chan_mask.
adc_diag_t diag_method	Configures the self-diagnostic mode.
adc_elc_t signal_elc	Sets the generation condition for the ELC scan-end event.
bool anex_enable	Specifies whether or not the extended analog input (ANEX1) is used.
uint8_t sample_hold_mask	Selects the channels to be used by the sample-and-hold circuit.*1 Select channels used by the channel-dedicated sample-and-hold from among channels 0 to 2.
uint8_t sample_hold_states	Sets the sampling time. The lower limit value for the sampling time differs depending on the MCU. Check the user's manual before making the setting.
adc_conv_order_stat_t conv_order_status	Selects the channel conversion order setting method.
uint32_t conversion_order[ ]	Sets the channel conversion order. For the channel conversion order setting, use ADC_MASK_CHn (n being the channel number) or ADC_MASK_CONV_ORDER_OFF.*4*5*6*7 Check the user's manual before setting the channel conversion order.
adc_path_ctrl_t signal_path control[ ]	Sets the analog signal and PGA output destination.
adc_pga_gain_t pga_gain[ ]	Sets the PGA gain.

Note 1. As the channel designation, use ADC_MASK_CHn (n being the channel number), ADC_MASK_TEMP (temperature sensor), ADC_MASK_VOLT (internal reference voltage sensor), or a combination.
Example: (ADC_MASK_CH1 | ADC_MASK_CH3 | ADC_MASK_CH5)

Note 2. If "A/D conversion is performed on the temperature sensor output" is selected for 2.9.2(2)(g), 3rd Argument Structure Members: Temperature Sensor (Enumeration), specify ADC_MASK_TEMP.

Note 3. If "The temperature sensor's addition/average mode is used" is selected for 2.9.2(2)(h) 3rd Argument Structure Members: Temperature Sensor Addition/Average Mode (Enumeration), specify ADC_MASK_TEMP.

Note 4. Specify all channels selecting as conversion targets by the A/D channel select register.

Note 5. Set the channel conversion order starting with conversion_order[0], and set any leftover variables to ADC_MASK_CONV_ORDER_OFF.

Note 6. Do not specify the same channel.

Note 7. The setting value has no effect when conv_order_status is set to ADC_CONV_ORDER_AUTO_SETTING.

(a) 3rd Argument Structure Members: Group Priority Control (Enumeration)

```
typedef enum e_adc_grpa
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_grpa_t;
```

Member	Description
ADC_GRPA_PRIORITY_OFF	No priority control operation is performed.
ADC_GRPA_GRPB_WAIT_TRIG	[Maximum group count of 2] Priority control is applied to group A, and no restart occurs after A/D conversion on group B is interrupted.
ADC_GRPA_GRPB_RESTART_SCAN	[Maximum group count of 2] Priority control is applied to group A, and a restart occurs after A/D conversion on group B is interrupted.*1
ADC_GRPA_GRPB_CONT_SCAN	[Maximum group count of 2] Continuous single scan operation is performed on group B. (Group A operation has priority when an A/D conversion request for group A is generated.)
ADC_GRPA_GRPB_GRPC_WAIT_TRIG	[Maximum group count of 3] Group priority control is applied, and no restart occurs after A/D conversion on a low-priority group is interrupted.
ADC_GRPA_GRPB_GRPC_TOP_RESTART_SCAN	[Maximum group count of 3] Group priority control is applied, and a restart occurs from the first channel after A/D conversion on a low-priority group is interrupted.
ADC_GRPA_GRPB_GRPC_RESTART_TOP_CONT_SCAN	[Maximum group count of 3] Continuous single scan operation is performed on the lowest-priority group. Group priority control is applied, and a restart occurs from the first channel after A/D conversion on a low-priority group is interrupted.*1
ADC_GRPA_GRPB_GRPC_RESTART_SCAN	[Maximum group count of 3] Group priority control is applied, and a restart occurs from the unfinished channel after A/D conversion on a low-priority group is interrupted.
ADC_GRPA_GRPB_GRPC_TOP_CONT_SCAN	[Maximum group count of 3] Continuous single scan operation is performed on the lowest-priority group. Group priority control is applied, and no restart occurs after A/D conversion on a low-priority group is interrupted.*1
ADC_GRPA_GRPB_GRPC_RESTART_CONT_SCAN	[Maximum group count of 3] Continuous single scan operation is performed on the lowest-priority group. Group priority control is applied, and a restart occurs from the unfinished channel after A/D conversion on a low-priority group is interrupted.*1

Note 1. When making this setting on the S12ADC, S12ADE, or S12ADFa, ensure that the frequency ratio of the peripheral module clock (PCLK) and A/D conversion clock (ADCLK) is 1:1. For details, see the user's manual.

(b) 3rd Argument Structure Members: Self-Diagnostic (Enumeration)

```
typedef enum e_adc_diag
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_diag_t;
```

Member	Description
ADC_DIAG_OFF	The self-diagnostic function is not used.
ADC_DIAG_0_VOLT	A self-diagnostic test using a voltage of 0 V is performed.
ADC_DIAG_HALF_VREFH0	A self-diagnostic test using a voltage of (reference voltage × 1/2) is performed.
ADC_DIAG_VREFH0	A self-diagnostic test using the reference voltage is performed.
ADC_DIAG_ROTATE_VOLTS	Self-diagnostic rotation mode is used.

(c) 3rd Argument Structure Members: ELC Event Output Condition (Enumeration)

```
typedef enum e_adc_elc
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_elc_t;
```

Member	Description
ADC_ELC_SCAN_DONE	Event output when scanning of group A completes
ADC_ELC_GROUPA_SCAN_DONE	
ADC_ELC_GROUPB_SCAN_DONE	Event output when scanning of group B completes
ADC_ELC_ALL_SCANS_DONE	Event output when all scanning completes
ADC_ELC_ANY_ONE_OF_SCAN_DONE	Event output when scanning of a group completes (event output when scanning of group A, group B, or group C completes)
ADC_ELC_GROUPC_SCAN_DONE	Event output when scanning of group C completes

(d) 3rd Argument Structure Members: Channel Conversion Order (Enumeration)

```
typedef enum e_adc_conv_order_stat
{
      /* Refer to the table below for the members. */
} adc_conv_order_stat_t;
```

Member	Description
ADC_CONV_ORDER_AUTO_SETTING	A/D conversion is performed on channels selected as conversion targets by the A/D channel select registers in order, starting from the lowest channel number. In this case, the conversion_order[ ] setting has no effect.
ADC_CONV_ORDER_MANUAL_SETTING	A/D conversion is performed in the order specified by the user (the order specified by conversion_order[ ]).

(e) 3rd Argument Structure Members: Signal Path Control (Enumeration)

```
typedef enum e_adc_path_ctrl
{
      /* Refer to the table below for the members. */
} adc_path_ctrl_t;
```

Member	Description
ADC_ANALOG_INPUT_1	Input analog pin signal to A/D converter.
ADC_ANALOG_INPUT_2	Input analog pin signal to CMPCm0.
ADC_ANALOG_INPUT_3	Input analog pin signal to A/D converter and CMPCm0.
ADC_PGA_SINGLE_END_INPUT_1	Input analog pin signal to CMPCm0, and input PGA signal to CMPCm1.
ADC_PGA_SINGLE_END_INPUT_2	Input analog pin signal to A/D converter and CMPCm0, and input single-end input setting PGA output to CMPCm1.
ADC_PGA_SINGLE_END_INPUT_3	Input analog pin signal to CMPCm0, and input single-end input setting PGA output to A/D converter and CMPCm1.
ADC_PGA_DIFFERENTIAL_INPUT_1	Input pseudo-differential input setting PGA output to CMPCm1.
ADC_PGA_DIFFERENTIAL_INPUT_2	Input analog pin signal to A/D converter, and input pseudo-differential input setting PGA output to CMPCm1.
ADC_PGA_DIFFERENTIAL_INPUT_3	Input pseudo-differential input setting PGA output to A/D converter and CMPCm1.
ADC_GENERAL_PORT_1	Use a general input port. (No signal input to A/D converter, CMPCm0, and CMPCm1.)

(f) 3rd Argument Structure Members: PGA Gain (Enumeration)

```
typedef enum e_adc_pga_gain
{
     /* Refer to the table below for the members. */
} adc_pga_gain_t;
```

Member	Description
ADC_PGA_GAIN_OFF	No PGA gain setting.*1
ADC_PGA_GAIN_2_000	PGA single-end input × 2.000*2
ADC_PGA_GAIN_2_500	PGA single-end input × 2.500*2
ADC_PGA_GAIN_3_077	PGA single-end input × 3.077*2
ADC_PGA_GAIN_3_636	PGA single-end input × 3.636*2
ADC_PGA_GAIN_4_000	PGA single-end input × 4.000*2
ADC_PGA_GAIN_4_444	PGA single-end input × 4.444*2
ADC_PGA_GAIN_5_000	PGA single-end input × 5.000*2
ADC_PGA_GAIN_6_667	PGA single-end input × 6.667*2
ADC_PGA_GAIN_8_000	PGA single-end input × 8.000*2
ADC_PGA_GAIN_10_000	PGA single-end input × 10.000*2
ADC_PGA_GAIN_13_333	PGA single-end input × 13.333*2
ADC_PGA_GAIN_20_000	PGA single-end input × 20.000*2
ADC_PGA_GAIN_1_500_DIFF	PGA pseudo-differential input × 1.500*3
ADC_PGA_GAIN_4_000_DIFF	PGA pseudo-differential input × 4.000*3
ADC_PGA_GAIN_7_000_DIFF	PGA pseudo-differential input × 7.000*3
ADC_PGA_GAIN_12_333_DIFF	PGA pseudo-differential input × 12.333*3

Note 1. Select this if the PGA will not be used (if ADC_ANALOG_INPUT_n or ADC_GENERAL_PORT_n are selected by signal path control).

Note 2. Select this if PGA single-end input will be used. (if ADC_PGA_SINGLE_END_INPUT_n is selected by signal path control)

Note 3. Select this if PGA pseudo-differential input will be used. (if ADC_PGA_DIFFERENTIAL_INPUT_n is selected by signal path control)

(5) 3rd Argument: Comparison Function (Structure)

```
typedef struct st_adc_cmpwin_cfg
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_cmpwin_t;
```

Member	Description
uint32_t compare_mask	Selects the channels used by the window A comparison function.*1
uint32_t compare_maskb	Selects the channels used by the window B comparison function.*2
uint32_t inside_window_mask	Selects a compare condition for each channel of window A. Bit n corresponds to channel n. Window function disabled (ADC_CMD_EN_COMPARATOR_LEVEL command) 0: Match when level_lo > A/D-converted value. 1: Match when level_lo < A/D-converted value. Window function enabled (ADC_CMD_EN_COMPARATOR_WINDOW command) 0: Match when A/D-converted value < level_lo or level_hi < A/D-converted value. 1: Match when level_lo < A/D-converted value < level_hi.
uint32_t inside_window_maskb	Selects compare conditions for window B. Window function disabled (ADC_CMD_EN_COMPARATOR_LEVEL command) ADC_COMP_WINB_COND_BELOW: Match when level_lo > A/D-converted value. ADC_COMP_WINB_COND_ABOVE: Match when level_lo < A/D-converted value. Window function enabled (ADC_CMD_EN_COMPARATOR_WINDOW command) ADC_COMP_WINB_COND_BELOW: Match when A/D-converted value < level_lo or level_hi < A/D-converted value. ADC_COMP_WINB_COND_ABOVE: Match when level_lo < A/D-converted value < level_hi.
uint16_t level_lo	Sets the lower level for compare window A.*3
uint16_t level_lob	Sets the lower level for compare window B.*3 This setting only has an effect when using the ADC_CMD_EN_COMPARATOR_WINDOW command.
uint16_t level_hi	Sets the upper level for compare window A.*3
uint16_t level_hib	Sets the upper level for compare window B.*3 This setting only has an effect when using the ADC_CMD_EN_COMPARATOR_WINDOW command.
adc_comp_cond_t comp_cond	Sets the window A/B complex condition.
unit8_t int_priority	Sets the priority (0 to 15) of the S12CMPAI interrupt and S12CMPBI interrupt. Specifying 0 disables the S12CMPAI interrupt and S12CMPBI interrupt.

Member	Description
bool windowa_enable	Enables/disables the compare window A function.
bool windowb_enable	Enables/disables the compare window B function.

Note 1. As the channel designation, use ADC_MASK_CHn (n being the channel number), ADC_MASK_TEMP (temperature sensor), ADC_MASK_VOLT (internal reference voltage sensor), or a combination.
Example: (ADC_MASK_CH1 | ADC_MASK_CH3 | ADC_MASK_CH5)

Note 2. As the window B channel designation, use ADC_COMP_WINB_CHn (n being the channel number), ADC_COMP_WINB_TEMP (temperature sensor), or ADC_COMP_WINB_VOLT (internal reference voltage sensor).

Note 3. The setting details will differ depending on the A/D data register format (right-justified/left-justified), A/D conversion precision, and A/D-converted value addition mode settings. For details, refer to the hardware manual of your MCU.

(a) 3rd Argument Structure Members: Window A/B Complex Condition (Enumeration)

```
typedef enum e_adc_comp_cond
{
      /* Refer to the table below for the members. */
} adc_comp_cond_t;
```

Member	Description
ADC_COND_OR	Window A comparison condition met OR window B comparison condition met.
ADC_COND_EXOR	Window A comparison condition met EXOR window B comparison condition met.
ADC_COND_AND	Window A comparison condition met AND window B comparison condition met.

(6) 3rd Argument: Window A/B Combination Result (Enumeration)

```
typedef enum e_adc_comp_stat
{
      /* Refer to the table below for the members. */
} adc_comp_stat_t;
```

Member	Description
ADC_COMP_COND_NOTMET	Window A/B complex condition not met.
ADC_COMP_COND_MET	Window A/B complex condition met.

2.9.4 Structures and Enumerations Used as Arguments for R_ADC_Read Function

(1) 2nd Argument: Conversion Result Read Channel (Enumeration)

```
typedef enum e_adc_reg
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_reg_t;
```

Member	Description
ADC_REG_CHn (n = channel number)	Specifies the channel n A/D-converted value.*1
ADC_REG_TEMP	Specifies the temperature sensor A/D-converted value.
ADC_REG_VOLT	Specifies the internal reference voltage sensor A/D-converted value.
ADC_REG_DBLTRIG	Specifies the double trigger A/D-converted value.
ADC_REG_DBLTRIGA	Specifies the A/D-converted value in double trigger extended mode (ADDBLDRA register).
ADC_REG_DBLTRIGB	Specifies the A/D-converted value in double trigger extended mode (ADDBLDRB register).
ADC_REG_SELF_DIAG	Specifies the self-diagnostic A/D-converted value.

Note 1. The available channels differ depending on the MCU and the pin count.

2.9.5 Structures and Enumerations Used as Arguments for R_ADC_ReadAll Function

(1) 1st Argument: Structure for Storing Conversion Result (Structure)

```
typedef struct st_adc_data
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_data_t;
```

Member	Description
uint16_t chan[ADC_n_REG_ARRAY_MAX] (n = channel number)	Stores the A/D conversion result of each channel.*1
uint16t temp	Stores the temperature sensor A/D conversion result.
uint16t volt	Stores the internal reference voltage A/D conversion result.
uint16t dbltrig	Stores the double trigger A/D conversion result.
uint16t self_diag	Stores the self-diagnostic A/D conversion result.
adc_unitM_data_t unitM (M = unit number)	Structure for storing A/D conversion results for each unit.*2

Note 1. For the channel specification, use ADC_REG_CHn (n being the channel number).

Note 2. If there are multiple units, provide a separate structure for storing A/D conversion results for each unit.

(a) 1st Argument Structure Members: Structure for Storing Conversion Results for Each Unit (Structure)

```
typedef struct st_adc_unitM_data  /* M = unit number */
{
      /* Refer to the table below for the members. The members available for
         use differ depending on the MCU. */
} adc_unitM_data_t; /* M = unit number */
```

Member	Description
uint16_t chan[ADC_n_REG_ARRAY_MAX] (n = channel number)	Stores the A/D conversion result of each channel.*1
uint16_t temp	Stores the temperature sensor A/D conversion result.
uint16_t volt	Stores the internal reference voltage A/D conversion result.
uint16_t dbltrig	Stores the double trigger A/D conversion result.
uint16_t dbltrigA	Stores the A/D conversion result in double trigger extended mode (ADDBLDRA register).
uint16_t dbltrigB	Stores the A/D conversion result in double trigger extended mode (ADDBLDRB register).
uint16_t self_diag	Stores the self-diagnostic A/D conversion result.

Note 1. For the channel specification, use ADC_REG_CHn (n being the channel number).

2.10 Return Values

These are the different error codes API functions can return. The enum is found in r_s12ad_rx_if.h along with the API function declarations.

```
typedef enum e_adc_err          // ADC API error codes
{
    ADC_SUCCESS = 0,
    ADC_ERR_AD_LOCKED,          // Open() call is in progress elsewhere
    ADC_ERR_AD_NOT_CLOSED,      // peripheral still running in another mode
    ADC_ERR_MISSING_PTR,        // missing required pointer argument
    ADC_ERR_INVALID_ARG,        // argument is not valid for parameter
    ADC_ERR_ILLEGAL_ARG,        // argument is illegal for mode
    ADC_ERR_SCAN_NOT_DONE,      // default, Group A, or Group B scan not done
    ADC_ERR_TRIG_ENABLED,       // scan running, cannot configure comparator
    ADC_ERR_CONDITION_NOT_MET, // no chans/sensors passed comparator condition
    ADC_ERR_UNKNOWN             // unknown hardware error
} adc_err_t;
```

2.11 Callback Functions

The ADC FIT module calls the user-specified callback function when a scan-end interrupt (S12ADIn, S12GBADIn, S12GCADIn, or GBADIn) or compare condition-met interrupt (S12CMPAIn or S12CMPBIn) occurs.

The callback function is specified using the R_ADC_Open function. For details, refer to 3.1, R_ADC_Open().

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends using “Smart Configurator” described in (1) or (3). However, “Smart Configurator” only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using “Smart Configurator” in e^2 studio
By using the “Smart Configurator” in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using “FIT Configurator” in e^2 studio
By using the “FIT Configurator” in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using “Smart Configurator” on CS+
By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User Guide: CS+ (R20AN0470)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.13 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API Functions

3.1 R_ADC_Open()

This function initializes the 12-bit A/D converter. This function must be run before calling any other API function.

Format

```
adc_err_t  R_ADC_Open(uint8_t           unit,
                      adc_mode_t const  mode,
                      adc_cfg_t * const p_cfg,
                      void              (* const  p_callback)(void *p_args));
```

Parameters

unit

Unit number. Set this to 0 if your MCU supports only one unit.

mode

Operating mode. For information on operating modes, refer to 2.9.2 (1), 2nd Argument: Operating Mode (Enumeration).

p_cfg

Pointer to the configuration structure of the 12-bit A/D converter function. For information on the function configuration structure, refer to 2.9.2 (2), 3rd Argument: Function Settings (Structure).

p_callback

Optional pointer to function called from interrupt when a scan completes or a comparator condition is met. When not using this parameter, set FIT_NO_PTR.

Return Values

ADC_SUCCESS:	*Successful*
ADC_ERR_AD_LOCKED:	*Open() call is in progress elsewhere*
ADC_ERR_AD_NOT_CLOSED:	*Peripheral is still running in another mode; Perform R_ADC_Close() first*
ADC_ERR_INVALID_ARG:	*Element of p_cfg structure has invalid value*
ADC_ERR_ILLEGAL_ARG:	*an argument is illegal based upon mode*
ADC_ERR_MISSING_PTR:	*p_cfg pointer is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. With a non-zero interrupt priority (interrupt usage), a callback function is called by the interrupts whenever a scan has completed or a comparator condition is met. When setting the interrupt priority to 0, a callback function is not called. In this case, poll for scan completion with the R_ADC_ Control() function when necessary.

To set values of parameters used in this function, first clear all members of parameters to 0, and then set values.

Example (S12ADb)

```
adc_cfg_t   config;

/* Clear all members of the adc_cfg_t structure */
memset(&config, 0, sizeof(config));

    /* INITIALIZE FOR SINGLE SCAN OF TEMPERATURE SENSOR
     * - use software trigger to start scan; poll for completion
     * - don't do any summing of conversion values
     * - keep the data registers aligned right, and clear after reading is off
     * - use normal speed conversion
     */
    config.trigger = ADC_TRIG_SOFTWARE;
    config.priority = 0;                // denotes polling!
    config.add_cnt = ADC_ADD_OFF;
    config.alignment = ADC_ALIGN_RIGHT;
    config.clearing = ADC_CLEAR_AFTER_READ_OFF;
    config.conv_speed = ADC_CONVERT_SPEED_NORM;

    R_ADC_Open(0, ADC_MODE_SS_TEMPERATURE, &config, FIT_NO_FUNC);
```

Special Notes (RX Family Common):

The application must complete MPC and PORT initialization prior to calling R_ADC_Open(). Refer to the User's Manual: Hardware about limitations of using output pins on the same port as analog pins. The following is a sample initialization for an RSKRX111 Rev 1 board:

```
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

      PORT4.PDR.BIT.B0 = 0;      // set direction of A/D conversion port to
                                    input
      PORT4.PMR.BIT.B0 = 0;      // set A/D conversion port to general I/O
      MPC.P40PFS.BYTE = 0x80;    // set P40 function to A/D conversion port
                                    (AN000)

      MPC.PB0PFS.BIT.PSEL = 0x09;// set PB0 function to ADTRIG0
                                 // (SW3 on RSKRX111)
      PORTB.PDR.BIT.B0 = 0;      // set ADTRIG0 pin direction to input
      PORTB.PMR.BIT.B0 = 1;      // set ADTRIG0 pin mode to peripheral

    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

The application must set the A/D conversion clock prior to calling R_ADC_Open() function.

To stop A/D conversion which is started in continuous scan mode, call the R_ADC_Close function.

If continuous scan mode is selected, it is recommended not to use the S12ADI interrupt since scan completion occurs continuously. That causes the majority of the processing time to be spent at the interrupt level.

If interrupts are in use, a callback function is required which takes a single argument. This is a pointer to a structure which is cast to a void pointer (provides consistency with other FIT module callback functions). Cast to the adc_cb_args_t pointer in the interrupt handling. See 2.9.1 (1) (a), 1st Argument Structure Members: Callback Function Events (Enumeration) for details on 'adc_cb_args_t'.

An example template for a callback function is provided here:

```
void MyCallback(void *p_args)
{
adc_cb_args_t    *args;


    args = (adc_cb_args_t *)p_args;

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        // Read results here
        nop();
    }
    else if (args->event == ADC_EVT_GROUPB_SCAN_COMPLETE)
    {
        // Read Group B results here
        nop();
    }
    else if (args->event == ADC_EVT_CONDITION_MET)
    {
        // Process chans/sensors indicated in args->compare_flags
        nop();
    }
}
```

Special Notes (S12ADb, A12ADC, S12ADE, S12ADFa, S12ADH):
After the R_ADC_Open() function is executed, wait at least 1 µs before executing A/D conversion.

3.2 R_ADC_Control()

This function makes 12-bit A/D converter function settings and acquires the interrupt control and A/D converter start/stop state.

Format

```
adc_err_t   R_ADC_Control(uint8_t            unit,
                          adc_cmd_t const    cmd,
                          void * const       p_args);
```

Parameters

unit

Unit number. Set this to 0 if your MCU supports only one unit.

cmd

Command to run. For commands and the arguments used with them, refer to 2.9.3, Structures and Enumerations Used as Arguments for R_ADC_Control Function.

p_args

Pointer to optional configuration structure. Clear all members of the argument to 0 before setting values to them. If the command requires no argument, set FIT_NO_PTR.

Return Values

ADC_SUCCESS:	*Successful*
ADC_ERR_MISSING_PTR:	*p_args pointer is FIT_NO_PTR/NULL when required as an argument*
ADC_ERR_INVALID_ARG:	*Invalid value is specified to p_args structure*
ADC_ERR_ILLEGAL_ARG:	*cmd is illegal based upon mode*
ADC_ERR_SCAN_NOT_DONE:	*The requested scan has not completed*
ADC_ERR_TRIG_ENABLED:	*Cannot configure comparator because scan still running*
ADC_ERR_CONDITION_NOT_MET:	*No channels/sensors met the comparison condition*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Provides commands for enabling channels and sensors and for runtime operations. These include enabling/disabling trigger sources and interrupts, initiating a software trigger, and checking for scan completion.

After the R_ADC_Open function is called, the commands listed below can be issued by using the R_ADC_Control function.

Only the necessary commands should be issued, and in the order indicated by the No. column below. For the arguments of the R_ADC_Control function, refer to 2.9.3, Structures and Enumerations Used as Arguments for R_ADC_Control Function.

No.	Command	Description
1	ADC_CMD_SET_DDA_STATE_CNT	Configures the A/D disconnection detection assist function. Issue this command if the disconnection detection assist function will be used.
2	ADC_CMD_SET_SAMPLE_STATE_CNT	Specifies the A/D sampling number of states. Issue this command to change the value of the ADSSTRn register from the default.
3	ADC_CMD_USE_VREFH0	Sets the high-side reference voltage to VREFH0. AVCC0 is selected after the R_ADC_Open function is called. Issue this command to select VREFH0 instead.
4	ADC_CMD_USE_VREFL0	Sets the low-side reference voltage to VREFL0. AVSS0 is selected after the R_ADC_Open function is called. Issue this command to select VREFL0 instead.
5	ADC_CMD_ENABLE_CHANS	Selects and configures the channels on which A/D conversion is performed. No A/D conversion channels are selected after a reset. Issue this command before A/D conversion starts.
6	ADC_CMD_ENABLE_TEMP_SENSOR	Enables the temperature sensor. Issue this command if the temperature sensor will be used.
7	ADC_CMD_ENABLE_VOLT_SENSOR	Enables the internal reference voltage sensor. Issue this command if the internal reference voltage sensor will be used.
8	ADC_CMD_EN_COMPARATOR_LEVEL	Specifies that the comparison function is used with the window function disabled (threshold comparison). Issue this command if the comparison function will be used.
9	ADC_CMD_EN_COMPARATOR_WINDOW	Specifies that the comparison function is used with the window function enabled (range comparison). Issue this command if the comparison function will be used.
10	ADC_CMD_ ENABLE _TRIG	Enables A/D conversion start by synchronous or asynchronous trigger. Issue this command to select synchronous or asynchronous trigger.
11	ADC_CMD_SCAN_NOW	Enables A/D conversion start by software trigger. Issue this command to select software trigger.
12	ADC_CMD_CHECK_SCAN_DONE	Checks whether A/D conversion is in progress in single scan mode. Used this command when completion of A/D conversion is checked by polling, without using a callback function.
13	ADC_CMD_CHECK_SCAN_DONE_GROUPA	Checks whether A/D conversion of group A is in progress in group scan mode. Used this command when the group A interrupt priority level is set to 0 and polling is used to confirm A/D conversion.

No.	Command	Description
14	ADC_CMD_CHECK_SCAN_DONE_GROUPB	Checks whether A/D conversion of group B is in progress in group scan mode. Used this command when the group B interrupt priority level is set to 0 and polling is used to confirm A/D conversion.
15	ADC_CMD_CHECK_SCAN_DONE_GROUPC	Checks whether A/D conversion of group C is in progress in group scan mode. Used this command when the group C interrupt priority level is set to 0 and polling is used to confirm A/D conversion.
16	ADC_CMD_CHECK_CONDITION_MET	Stores the result obtained by the comparison function in the variable specified by the argument. The comparison result for channel n is stored in bit n.*1 0: Comparison condition not met. 1: Comparison condition met.
17	ADC_CMD_CHECK_CONDITION_METB	Gets the comparison result produced by the group B comparison function. Stores the group B comparison result in the variable specified by the argument.*1 0x0000: Comparison condition not met. 0x0001: Comparison condition met.
18	ADC_CMD_COMP_COMB_STATUS	Acquires the window A/B complex condition result. The window A/B combination result is stored in the variable specified by the argument. ADC_COMP_COND_NOTMET: Window A/B complex condition not met. ADC_COMP_COND_MET: Window A/B complex condition met.

Note 1. After execution of this command, the comparison result is initialized to 0 (comparison condition not met). Therefore, this command must be executed once only after A/D conversion completes.

However, Yes only when the ADC_CMD_CHECK_SCAN_DONE_GROUPA, ADC_CMD_CHECK_SCAN_DONE_GROUPB, or ADC_CMD_CHECK_SCAN_DONE_GROUPC command is being executed.

Example 1: Single Channel Polling Unit 0 (RX64M, RX71M, RX65x only)

```
uint16_t       data;
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;
adc_err_t      err;

/* OPEN ADC */

/* Clear all members of the adc_cfg_t structure */
memset(&config, 0, sizeof(config));


/* Open ADC for software trigger, single scan of one channel, and polling */
config.resolution = ADC_RESOLUTION_12_BIT;
config.trigger    = ADC_TRIG_SOFTWARE;
config.priority   = 0;                          // denotes polling
config.add_cnt    = ADC_ADD_OFF;
config.alignment  = ADC_ALIGN_RIGHT;
config.clearing   = ADC_CLEAR_AFTER_READ_OFF;
config.temp_sensor   = ADC_TEMP_SENSOR_NOT_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
err = R_ADC_Open(0,ADC_MODE_SS_ONE_CH, &config, NULL);

/* ENABLE CHANNELS */

/* Clear all members of the adc_ch_cfg_t structure */
memset(&ch_cfg, 0, sizeof(ch_cfg));

/* Specify and enable potentiometer channel on RSKRX64M */
ch_cfg.chan_mask        = ADC_MASK_CH0;
ch_cfg.diag_method      = ADC_DIAG_OFF;
ch_cfg.anex_enable      = false;
ch_cfg.sample_hold_mask = 0;
err = R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &ch_cfg);

/* After open, wait 1 us or longer before A/D conversion starts */

/* Repeatedly trigger, poll for completion, and read result */
while(1)
{
  /* CAUSE SOFTWARE TRIGGER */
  err = R_ADC_Control(0, ADC_CMD_SCAN_NOW, NULL);

  /* WAIT FOR SCAN TO COMPLETE */
  while (R_ADC_Control(0,ADC_CMD_CHECK_SCAN_DONE,NULL) ==
ADC_ERR_SCAN_NOT_DONE)
  {
  }

  /* READ RESULT */
  err = R_ADC_Read(0, ADC_REG_CH0, &data);
}
```

Example 2: Temperature Sensor Polling and Set Sample State Count Unit 1 (RX64M, RX71M, RX65x)

```
uint16_t       data;
adc_cfg_t      config;
adc_sst_t      sst;           // sample state
adc_ch_cfg_t   ch_cfg;
adc_err_t      adc_err;

/* OPEN ADC */

/* Clear all members of the adc_cfg_t structure */
memset(&config, 0, sizeof(config));

/* Open ADC for software trigger, single scan temperature sensor, and polling
*/
config.resolution = ADC_RESOLUTION_10_BIT;
config.trigger  = ADC_TRIG_SOFTWARE;
config.priority = 0;                          // denotes polling
config.add_cnt  = ADC_ADD_OFF;
config.alignment = ADC_ALIGN_RIGHT;
config.clearing  = ADC_CLEAR_AFTER_READ_OFF;
config.temp_sensor   = ADC_TEMP_SENSOR_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
adc_err = R_ADC_Open(1, ADC_MODE_SS_ONE_CH, &config, NULL);

/* DO SPECIAL HARDWARE CONFIGURATION */

/* Clear all members of the adc_sst_t structure */
memset(&sst, 0, sizeof(sst));

/* Clear all members of the adc_ch_cfg_t structure */
memset(&ch_cfg, 0, sizeof(ch_cfg));

/* Set number of sampling states for 4us sample *
/* For PCLKD=60MHz, 1 state = 1/60MHz = 16.7ns, 4us/16.7ns = 240 states */
sst.reg_id = ADC_SST_TEMPERATURE;
sst.num_states = 240;
adc_err = R_ADC_Control(1, ADC_CMD_SET_SAMPLE_STATE_CNT, &sst);

/* CONFIGURE SCAN */
ch_cfg.chan_mask = ADC_MASK_TEMP;
ch_cfg.diag_method = ADC_DIAG_OFF;
ch_cfg.anex_enable = false;
ch_cfg.sample_hold_mask = 0;     // not available on unit 1
adc_err = R_ADC_Control(1, ADC_CMD_ENABLE_CHANS, &ch_cfg);

/* After open, wait 1 us or longer before A/D conversion starts */
/* CAUSE SOFTWARE TRIGGER */
adc_err = R_ADC_Control(1, ADC_CMD_SCAN_NOW, NULL);

/* WAIT FOR SCAN TO COMPLETE */
while (R_ADC_Control(1, ADC_CMD_CHECK_SCAN_DONE, NULL) ==
ADC_ERR_SCAN_NOT_DONE)
{
}

/* READ RESULT */
adc_err = R_ADC_Read(1, ADC_REG_TEMP, &data);
```

Example 3: Grouped Channels with Interrupt Triggers, Double Trigger on Group A, and Averaging Four Samples (RX64M, RX71M, RX65x)

```
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;

/* INITIALIZE MTU HERE (USED FOR TRIGGER SOURCES) */

/* OPEN ADC */

/* Clear all members of each structure */
memset(&config, 0, sizeof(config));
memset(&ch_cfg, 0, sizeof(ch_cfg));

/* INITIALIZE ADC FOR GROUP SCANNING WITH DOUBLE TRIGGER
 *  - use synchronous trigger TRGA0N to start Group A scan; int priority 4
 *  - use synchronous trigger TRG0N to start Group B scan; int priority 5
 *  - allow each channel to be scanned four times and averaged before
      continuing
 *  - do not clear registers after reading
 */
config.resolution = ADC_RESOLUTION_8_BIT;
config.trigger    = ADC_TRIG_SYNC_TRG0AN;
config.priority   = 4;
config.trigger_groupb = ADC_TRIG_SYNC_TRG0EN;
config.priority_groupb= 5;
config.add_cnt    = ADC_ADD_AVG_4_SAMPLES;
config.alignment  = ADC_ALIGN_RIGHT;
config.clearing   = ADC_CLEAR_AFTER_READ_OFF;
config.temp_sensor   = ADC_TEMP_SENSOR_NOT_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
R_ADC_Open(1, ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A, &config, MyCallback);

/* CONFIGURE SCAN */

/* Can only have 1 channel for double triggering, and is only channel in Group
   A
   Have channel 8 as Group A, have 2, 3, and 9 as Group B
   Perform addition/average on all channels except 9
*/
ch_cfg.chan_mask = ADC_MASK_CH8;
ch_cfg.chan_mask_groupb = ADC_MASK_CH2 | ADC_MASK_CH3 | ADC_MASK_CH9;
ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
ch_cfg.add_mask = ADC_MASK_CH8 | ADC_MASK_CH2 | ADC_MASK_CH3;
ch_cfg.diag_method = ADC_DIAG_OFF;
ch_cfg.anex_enable = false;
ch_cfg.sample_hold_mask = 0;
R_ADC_Control(1, ADC_CMD_ENABLE_CHANS, &ch_cfg);

/* After open, wait 1 us or longer before A/D conversion starts */

/* ENABLE TRIGGERS */
R_ADC_Control(1, ADC_CMD_ENABLE_TRIG, NULL);

/* INTERRUPT OCCURS UPON SCAN COMPLETION */

/* The callback is called twice from interrupt level- once after each
 *  group scan completes. The order depends upon the trigger order.
 */
```

```
void MyCallback(void *p_args)
{
adc_cb_args_t  *args;
uint16_t        dbltrg,data2,data3,data8,data9;


    args = (adc_cb_args_t *)p_args;

    /* READ RESULTS */

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        /* From S12ADIO interrupt, Group A scan complete, read registers */
        R_ADC_Read(1, ADC_REG_CH8, &data8);
        R_ADC_Read(1, ADC_REG_DBLTRIG, &dbltrg);
    }
    else if (args->event == ADC_EVT_SCAN_COMPLETE_GROUPB)
    {
        /* From GBADI interrupt, Group B scan complete, read registers */
        R_ADC_Read(1, ADC_REG_CH2, &data2);
        R_ADC_Read(1, ADC_REG_CH3, &data3);
        R_ADC_Read(1, ADC_REG_CH9, &data9);
    }

    /* process data, or set flag for application level to do so */
}
```

Example 4: Grouped Channels with Interrupt Triggers (RX65x)

```
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;

/* INITIALIZE MTU HERE (USED FOR TRIGGER SOURCES) */

/* OPEN ADC */

/* Clear all members of each structure */
memset(&config, 0, sizeof(config));

/* INITIALIZE ADC FOR GROUP SCANNING WITH DOUBLE TRIGGER
 *  - use synchronous trigger TRGA0N to start Group A scan; int priority 4
 *  - use synchronous trigger TRGA1N to start Group B scan; int priority 5
 *  - use synchronous trigger TRGA2N to start Group C scan; int priority 6
 *  - allow each channel to be scanned four times and averaged before
      continuing
 *  - do not clear registers after reading
 */
config.resolution = ADC_RESOLUTION_8_BIT;
config.trigger    = ADC_TRIG_SYNC_TRG0AN;
config.priority   = 4;
config.trigger_groupb = ADC_TRIG_SYNC_TRG1AN;
config.priority_groupb= 5;
config.trigger_groupc = ADC_TRIG_SYNC_TRG2AN;
config.priority_groupc= 6;
config.add_cnt    = ADC_ADD_OFF;
config.alignment  = ADC_ALIGN_RIGHT;
config.clearing   = ADC_CLEAR_AFTER_READ_OFF;
```

```
config.temp_sensor   = ADC_TEMP_SENSOR_NOT_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
R_ADC_Open(0, ADC_MODE_SS_MULTI_CH_GROUPED_GROUPC, &config, MyCallback);

/* CONFIGURE SCAN */

/* Clear all members of the adc_ch_cfg_t structure */
memset(&ch_cfg, 0, sizeof(ch_cfg));


/* Have channel 1 and 2 as Group A, have 3 and 4 as Group B,
   have 5 and 6 as Group C
   Perform addition/average on all channels except 9
*/
ch_cfg.chan_mask = ADC_MASK_CH1 | ADC_MASK_CH2;
ch_cfg.chan_mask_groupb = ADC_MASK_CH3 | ADC_MASK_CH4;
ch_cfg.chan_mask_groupc = ADC_MASK_CH5 | ADC_MASK_CH6;
ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
ch_cfg.add_mask = 0;
ch_cfg.diag_method = ADC_DIAG_OFF;
ch_cfg.anex_enable = false;
ch_cfg.sample_hold_mask = 0;
R_ADC_Control(0, ADC_CMD_CONFIGURE_SCAN, &ch_cfg);

/* After open, wait 1 us or longer before A/D conversion starts */

/* ENABLE TRIGGERS */
R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);

/* INTERRUPT OCCURS UPON SCAN COMPLETION */



/* The callback is called twice from interrupt level- once after each
 *  group scan completes. The order depends upon the trigger order.
 */
void MyCallback(void *p_args)
{
adc_cb_args_t  *args;
uint16_t        data1,data2,data3,data4,data5,data6;


    args = (adc_cb_args_t *)p_args;

    /* READ RESULTS */

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        /* From S12ADIO interrupt, Group A scan complete, read registers */
        R_ADC_Read(0, ADC_REG_CH1, &data1);
        R_ADC_Read(0, ADC_REG_CH2, &data2);
    }
    else if (args->event == ADC_EVT_SCAN_COMPLETE_GROUPB)
    {
        /* From GBADI interrupt, Group B scan complete, read registers */
        R_ADC_Read(0, ADC_REG_CH3, &data3);
        R_ADC_Read(0, ADC_REG_CH4, &data4);
    }
```

```
    else if (args->event == ADC_EVT_SCAN_COMPLETE_GROUPC)
    {
        /* From GCADI interrupt, Group C scan complete, read registers */
        R_ADC_Read(0, ADC_REG_CH5, &data5);
        R_ADC_Read(0, ADC_REG_CH6, &data6);
}
    /* process data, or set flag for application level to do so */
}
```

Example 5: Multiple Channels with Interrupt Trigger and Comparator Checking (RX64M, RX71M)

```
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;
adc_cmpwin_t   cmpwin;


/* INITIALIZE MTU HERE (USED FOR TRIGGER SOURCES) */

/* OPEN UNIT 0 */

/* Clear all members of the adc_cfg_t structure */
memset(&config, 0, sizeof(config));

config.resolution = ADC_RESOLUTION_12_BIT;
config.trigger = ADC_TRIG_SYNC_TRG0AN;
config.priority = 4;
config.add_cnt = ADC_ADD_OFF;
config.alignment = ADC_ALIGN_RIGHT;
config.clearing = ADC_CLEAR_AFTER_READ_OFF;
config.temp_sensor   = ADC_TEMP_SENSOR_NOT_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
R_ADC_Open(0, ADC_MODE_SS_MULTI_CH, &config, MyCallback);


/* CONFIGURE SCAN OF CHANNELS 3-5 */

/* Clear all members of the adc_ch_cfg_t structure */
memset(&ch_cfg, 0, sizeof(ch_cfg));

ch_cfg.chan_mask = ADC_MASK_CH3 | ADC_MASK_CH4 | ADC_MASK_CH5;
ch_cfg.diag_method = ADC_DIAG_OFF;
ch_cfg.anex_enable = false;
ch_cfg.sample_hold_mask = 0;
R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &ch_cfg);


/* HAVE COMPARATOR CHECK ON CHANNELS 3-4 FOR DROPPING BELOW 1.65V */

/* Clear all members of the adc_cmpwin_t structure */
memset(&cmpwin, 0, sizeof(cmpwin));

cmpwin.compare_mask = ADC_MASK_CH3 | ADC_MASK_CH4;
cmpwin.inside_window_mask = 0;          // condition met when below level
cmpwin.level_lo = 0x7FF;                // 12-bit 3.3V=0xFFF, 1.65V=0x7FF
cmpwin.int_priority = 3;
R_ADC_Control(0, ADC_CMD_EN_COMPARATOR_LEVEL, &cmpwin);
```

```
/* ENABLE TRIGGERS */
R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);


/* INTERRUPT OCCURS UPON SCAN COMPLETION */

  :

/* Callback called from interrupt level: */

void MyCallback(void *p_args)
{
adc_cb_args_t  *args;
uint16_t        data3,data4,data5;


    args = (adc_cb_args_t *)p_args;

    /* READ RESULTS */

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        R_ADC_Read(0, ADC_REG_CH3, &data3);
        R_ADC_Read(0, ADC_REG_CH4, &data4);
        R_ADC_Read(0, ADC_REG_CH5, &data5);
    }

    if (args->event == ADC_EVT_CONDITION_MET)
    {
        if (args->compare_flags & ADC_MASK_CH3)
        {
            // processing when channel 3 voltage is too low
        }
        else
        {
            // processing when channel 4 voltage is too low
        }
    }
}
```

Example 6: Multiple Channels with Interrupt Trigger and 2 Comparator Checking (RX65x)

```
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;
adc_cmpwin_t   cmpwin;

/* Clear all members of each structure */
memset(&config, 0, sizeof(config));
memset(&ch_cfg, 0, sizeof(ch_cfg));
memset(&cmpwin, 0, sizeof(cmpwin));

/* INITIALIZE MTU HERE (USED FOR TRIGGER SOURCES) */

/* OPEN UNIT 0 */

config.resolution = ADC_RESOLUTION_12_BIT;
config.trigger = ADC_TRIG_SYNC_TRG0AN;
config.priority = 4;
```

```
config.add_cnt = ADC_ADD_OFF;
config.alignment = ADC_ALIGN_RIGHT;
config.clearing = ADC_CLEAR_AFTER_READ_OFF;
config.temp_sensor   = ADC_TEMP_SENSOR_NOT_AD_CONVERTED;
config.add_temp_sensor   = ADC_TEMP_SENSOR_ADD_OFF;
R_ADC_Open(0, ADC_MODE_SS_MULTI_CH, &config, MyCallback);


/* CONFIGURE SCAN OF CHANNELS 3-4 */

ch_cfg.chan_mask = ADC_MASK_CH3 | ADC_MASK_CH4 | ADC_MASK_CH5;
ch_cfg.diag_method = ADC_DIAG_OFF;
ch_cfg.anex_enable = false;
ch_cfg.sample_hold_mask = 0;
R_ADC_Control(0, ADC_CMD_CONFIGURE_SCAN, &ch_cfg);


/* HAVE COMPARATOR CHECK ON CHANNELS 3-4 FOR DROPPING BELOW 1.65V */

cmpwin.compare_mask = ADC_MASK_CH3 | ADC_MASK_CH4;
cmpwin.compare_maskb = ADC_COMP_WINB_CH5;
cmpwin.inside_window_mask = 0;            // Condition met when below level
cmpwin.inside_window_maskb = ADC_COMP_WINB_COND_BELOW;
cmpwin.level_lo = 0x7FF;                  // 12-bit 3.3V=0xFFF, 1.65V=0x7FF
cmpwin.level_lob = 0x7FF;                 // 12-bit 3.3V=0xFFF, 1.65V=0x7FF
cmpwin.int_priority = 3;
cmpwin.windowa_enable = true;
cmpwin.windowb_enable = true;
R_ADC_Control(0, ADC_CMD_EN_COMPARATOR_LEVEL, &cmpwin);

/* After open, wait 1 us or longer before A/D conversion starts */

/* ENABLE TRIGGERS */
R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);

/* INTERRUPT OCCURS UPON SCAN COMPLETION */

  :

/* Callback called from interrupt level: */

void MyCallback(void *p_args)
{
adc_cb_args_t  *args;
uint16_t        data3,data4,data5;


    args = (adc_cb_args_t *)p_args;

    /* READ RESULTS */

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        R_ADC_Read(0, ADC_REG_CH3, &data3);
        R_ADC_Read(0, ADC_REG_CH4, &data4);
        R_ADC_Read(0, ADC_REG_CH5, &data5);
    }
```

```
    if (args->event == ADC_EVT_CONDITION_MET)
    {
        if (args->compare_flags & ADC_MASK_CH3)
        {
            // processing when channel 3 voltage is too low
        }
        else
        {
            // processing when channel 4 voltage is too low
        }
    }

    if (args->event == ADC_EVT_CONDITION_METB)
    {
        // processing when channel 5 voltage is too low
    }
}
```

Special Notes (RX Family Common):
When the A/D conversion start (ADST) bit is 1, settings such as mode must not be changed using this function. However, the conversion status or the comparison result can be obtained.

When switching channels used for A/D conversion or settings, call the R_ADC_Close() function once and then call the R_ADC_Open() function again to start.

When waiting completion of A/D conversion using the R_ADC_Control function, use the following commands.

A/D Conversion Channel Settings			Commands for the R_ADC_Control Function	
Mode	**A/D Conversion Start Trigger**	**Interrupt**	**Starts A/D Conversion**	**Waits Completion of A/D Conversion**
Single scan	Software trigger	—	ADC_CMD_SCAN_NOW	ADC_CMD_CHECK_SCAN_DONE
	Other than software trigger	Disabled	ADC_CMD_ENABLE_TRIG	ADC_CMD_CHECK_SCAN_DONE_GROUPA
Continuous scan	Software trigger	Disabled	ADC_CMD_SCAN_NOW	ADC_CMD_CHECK_SCAN_DONE_GROUPA
	Other than software trigger	Disabled	ADC_CMD_ENABLE_TRIG	ADC_CMD_CHECK_SCAN_DONE_GROUPA
Group scan	Other than software trigger	Disabled	ADC_CMD_ENABLE_TRIG	ADC_CMD_CHECK_SCAN_DONE_GROUPA ADC_CMD_CHECK_SCAN_DONE_GROUPB*1 ADC_CMD_CHECK_SCAN_DONE_GROUPC*2

Note 1. Use ADC_CMD_CHECK_SCAN_DONE_GROUPB when waiting completion of A/D conversion for Group B.
Note 2. ADC_CMD_CHECK_SCAN_DONE_GROUPC can be used with S12ADFa or S12ADH.

When A/D conversion interrupts are enabled, the R_ADC_Control() function cannot be used to wait completion of A/D conversion except when using single scan mode with software trigger. In this case, use the callback function for the A/D conversion interrupt to wait completion of A/D conversion.

Special Notes (S12ADC, S12ADFa):
Channels and sensors can be combined in the same unit.

ELC is only for S12ADI, not S12GBADI or S12CMPI. (S12ADC)

ELC is only for S12ADI, not GBADI, GCADI, S12CMPAI or S12CMPBI. (S12ADFa)

The application should wait 30 µs after configuring the scan before enabling the trigger for Temperature Sensor for best results.

If Group A Priority is selected such that Group B operates in continuous scan mode, it is recommended not to use the S12GBADI interrupt (S12ADC) and GBADI interrupt (S12ADFa) since the interrupt handling will be processed so often. That causes the majority of the processing time to be spent at the interrupt level.

Enabling the comparator should be done prior to enabling the triggers.Some features may not be used with others. The following table illustrates this.

	Dbl Trig	Group Scan	Self-Diag	Add/ Avg	ANEX	Sample & Hold	Priority Group A	Sensors	Comparator	DDA
Double trigger			X			B		X	X	
Group scan					X	S				
Self-diagnosis	X			X	X				X	X
Add/avg			X							
ANEX		X	X					X		X
Sample and hold	B	S					A			
Priority group A						A				
Sensors	X				X					X
Comparator	X		X							
Disconnect detection assist			X		X			X		

X: Combination may not be used. For example, ANEX may not be used with group scan modes, Self-Diagnosis, sensors or Disconnect Detection Assist.
A: Sample and Hold channels must be in Group A.
B: Sample and Hold channels must be in Group B or Group C.
S: Sample and Hold channels cannot be split across groups.

Special Notes (S12ADE):
This function does not support following features.

- Compare function window B
- Compare function window A/B composite condition setting

Special Notes (S12ADC, S12ADE, S12ADFa):
When using the comparison, configure the comparison after the channel configuration.

Special Notes (S12ADb, S12ADFa, and A12ADH):
For temperature sensor output and internal reference voltage, the number of sampling states must be set as indicated below, at a minimum:

S12ADb, S12ADFa: 5 μs

S12ADH: 4 μs

3.3 R_ADC_Read()

This function reads conversion results from a single channel, sensor, double trigger, or self-diagnosis register.

Format

```
adc_err_t  R_ADC_Read(uint8_t              unit,
                      adc_reg_t const      reg_id,
                      uint16_t * const     p_data);
```

Parameters

unit

Unit number. Set this to 0 if your MCU supports only one unit.

reg_id

ID of the register to read. For information on the register ID, refer to 2.9.4, Structures and Enumerations Used as Arguments for R_ADC_Read Function.

p_data

Pointer to variable to load value into.

Return Values

ADC_SUCCESS:	*Success*
ADC_ERR_INVALID_ARG:	*unit or reg_id contains an invalid value.*
ADC_ERR_MISSING _PTR:	*p_data is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Reads conversion results from a single channel, sensor, double trigger, or self-diagnosis register.

Example

```
uint16_t data;
     :
    /* Read channel 0 on unit 0 */
    R_ADC_Read(0, ADC_CH0_REG, &data); // conversion value placed in “data”
```

Special Notes (S12ADb):

For temperature sensor output and internal reference voltage, discard the first A/D conversion result after the open, and use the second and the subsequent A/D conversion results.

3.4 R_ADC_ReadAll()

This function reads conversion results from all storage registers supported by the MCU.

Format

```
adc_err_t  R_ADC_ReadAll(adc_data_t * const     p_data);
```

Parameters

p_data

Pointer to structure in which register values are loaded. For information on the structure, refer to 2.9.5, Structures and Enumerations Used as Arguments for R_ADC_ReadAll Function.

Return Values

ADC_SUCCESS:	*Success*
ADC_ERR_MISSING_PTR:	*p_data is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Reads conversion results from all potential sources, enabled or not.

Example

```
adc_data_t data;
     :
    /* Read all channel registers available on hardware */
    R_ADC_ReadAll(&data);  // "data" loaded with all conversion reg values
```

Special Notes:

None.

3.5 R_ADC_Close()

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Format

```
adc_err_t   R_ADC_Close(uint8_t unit);
```

Parameters

unit

Unit number. Set this to 0 if your MCU supports only one unit.

Return Values

ADC_SUCCESS:	*Success*
ADC_ERR_INVALID_ARG:	*unit contains an invalid value.*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Ends the A/D conversion in progress, disables interrupts, and ends A/D converter operation. This function can be called once per unit after the R_ADC_Open function is called.

When changing A/D conversion settings, call the R_ADC_Open() function again after this function is called.

Example

```
    :
err = R_ADC_Open(1, ADC_MODE_SS_MULTI_CH_GROUPED, &config, MyCallback);
    :
R_ADC_Close(1);
```

Special Notes:

This function will abort any scan that may be in progress.

3.6 R_ADC_GetVersion()

This function returns the driver version number at runtime.

Format

```
uint32_t  R_ADC_GetVersion(void)
```

Parameters

None

Return Values

Version number.

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Returns the version of this module. The version number is encoded such that the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Example

```
uint32_t   version;
  :
version = R_ADC_GetVersion();
```

Special Notes:

None.

4. Pin Setting

To use the ADC FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC). The pin assignment is referred to as the “Pin Setting” in this document. Please perform the pin setting after calling the R_ADC_Open function.

When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file.

The function name output from Smart Configurator is R_ADC_PinSet_S12ADx. “x” is the unit number of the selected option. For example, if unit 0 is selected as an option, the output function name will be R_ADC_PinSet_S12AD0.

5. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g. r_bsp). The standard naming convention for the demo project is <module>_demo_<board> where <module> is the peripheral acronym (e.g. s12ad, cmt, sci) and the <board> is the standard RSK (e.g. rskrx113). For example, s12ad FIT module demo project for RSKRX113 will be named as s12ad_demo_rskrx113. Similarly the exported .zip file will be <module>_demo_<board>.zip. For the same example, the zipped export/import file will be named as s12ad_demo_rskrx113.zip.

5.1 s12ad_int_demo_rskrx113

This demo uses periodic interrupts from MTU0 to trigger the ADC module to scan the potentiometer on the board. Each time a scan completes, the program reads the converted value at interrupt level in a callback function and places it into a global variable called “data”. This variable should be added to the Expressions window and made into a Realtime Watch (double-click to make realtime). As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.2 s12ad_poll_demo_rskrx113

This demo scans the potentiometer on the board via a software trigger in an endless loop. Each time a scan completes, the program reads the converted value at the application level and places it into a global variable called “data”. This variable should be added to the Expressions window and made into a Realtime Watch (double-click to make realtime). As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.3 s12ad_poll_demo_rskrx130

This demo scans the potentiometer on the board via a software trigger in an endless loop. Each time a scan completes, the program reads the converted value at the application level and places it into a global variable called “data”. This variable should be added to the Expressions window and made into a Realtime Watch (double-click to make realtime). As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.4 s12ad_demo_rskrx64m

This is a simple demo of the RX64M A/D Converter (S12AD) for the RSKRX64M starter kit (FIT module "r_s12ad_rx"). The demo uses the Multi-Function Timer Pulse Unit (MTU3a) to periodically trigger the ADC module to perform conversion on channel 0 which is connected to the on-board potentiometer. Each time a scan completes, the program reads the converted value at interrupt level in a callback function and places it into a global variable called “g_data”. This variable should be added to the Expressions window and made into a Realtime Watch (double-click to make realtime). As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.5 s12ad_demo_rskrx71m

This is a demo of the RX71M A/D Converter (S12AD) for the RSKRX71M starter kit (FIT module "r_s12ad_rx"). The demo uses the Multi-Function Timer Pulse Unit 3 (MTU3a) to periodically trigger the ADC module to perform conversion on channel 0 which is connected to the on-board potentiometer. Each time a scan completes, the program reads the converted value at interrupt level in a callback function and places it into a global variable called “g_data”. This variable should be added to the Expressions window and made into a Realtime Watch (double-click to make realtime). As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.6 s12ad_demo_rskrx231

This is a demo of the RX231 A/D Converter (S12ADE) for the RSKRX231 starter kit (FIT module “r_s12ad_rx”). The demo uses the Multi-Function Timer Pulse Unit 2 (MTU2a) to periodically trigger the ADC module to perform a conversion on channel 0, which is connected to the on-board potentiometer. Each time a scan completes, the program reads the converted value at interrupt level in a callback function and places it into a global variable called "g_data". This variable should be added to the Expressions window and made into a Realtime Watch. To do that, add it to the Expressions window then right-click it. From the drop-down menu click on "Realtime Refresh". Right click again and select "Realtime Refresh Interval" and set the refresh value to 200 ms. As the program runs, change the potentiometer position and observe the corresponding changes in the variable.

5.7 s12ad_demo_rskrx66t

This is a simple demo of the RX66T A/D converter (S12AD) for use with RSKRX66T (FIT module “r_s12ad_rx”). The demo uses multifunction timer pulse unit 3 (MTU3d) to perform A/D conversion at regular intervals on channel 0, which is connected to a variable resistor. When A/D conversion completes, the demo program reads the conversion value when a callback function interrupt occurs and stores the A/D conversion result in global variable “g_data.” After running the program, change the setting of the variable resistor to alter the voltage of the A/D input channel, and check the value of “g_data” in the emulator.

5.8 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the e^2 studio installation directory. To add a demo project to a workspace, select File>Import>General>Existing Projects into Workspace, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the demo subdirectory, select the desired demo zip file, then click “Finish”.

5.9 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on the required application note and select “Sample Code (download)” from the context menu in the *Smart Brower >> Application Notes* tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes confirmed operation environment for the ADC FIT module.

Table 6.1 Confirmed Operation Environment (Rev. 2.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 5.4.0 (WS Patch)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.30
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE)

Table 6.2 Confirmed Operation Environment (Rev. 3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.00
Board used	Renesas Starter Kit for RX66T (product No.: RTK500566T0SxxxxxBE)

Table 6.3 Confirmed Operation Environment (Rev. 3.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.01

Table 6.4 Confirmed Operation Environment (Rev. 3.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.10
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxx)

Table 6.5 Confirmed Operation Environment (Rev. 4.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.11.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201803 GCC for Renesas RX 4.8.4.201902 (RX66T only) Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.11.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.00
Board used	Renesas Starter Kit for RX113 (product No.: R0K505113xxxxxx) Renesas Starter Kit for RX130-512KB (product No.: RTK505130xxxxxxxxx) Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx) Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx) Renesas Starter Kit for RX66T (product No.: RTK500566Txxxxxxxxx) Renesas Starter Kit+ for RX71M (product No.: R0K50571Mxxxxxx)

Table 6.6 Confirmed Operation Environment (Rev. 4.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.4.10
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx)

Table 6.7 Confirmed Operation Environment (Rev. 4.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.20
Board used	Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx)

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

— When using CS+:
Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

— When using e^2 studio:
Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. For this, refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_s12ad_rx module.

A: The FIT module you added may not support the target device chosen in the user project. Check if the FIT module supports the target device for the project used.

(3) Q: The voltage input to the analog input pin and the A/D conversion result do not match.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4. Pin Setting for details.

Related Technical Updates

This module reflects the content of the following technical updates.

- TN-RX*-A124A/E
- TN-RX*-A117A/E

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.15.13	—	First edition issued
1.20	Apr.21.14	1,3 11,12	Added mention of support for RX110/63x. Added interface for RX210 Sample&Hold, Self-Diagnosis, and Disconnect Detection Assist (DDA)
1.30	Jun.05.14	—	Fixed bug in code that eliminated channels 8-15.
1.40	Nov.07.14	—	Added RX113 support.
2.00	Mar.30.15	—	Added RX64M/RX71M support. Modified interface to include a unit number.
2.10	Jun.15.15	—	Added RX231 support. Added an RX231 demo.
2.11	Mar.01.16	—	Added RX130 and RX230 support.
2.20	Dec.01.16	—	Added RX65N support.
		5	2.9 Code Size: • Changed code sizes for RX111. • Added code sizes for RX65N.
		53 to 64	3.2 R_ADC_Open(), 3.3 R_ADC_Control(): Added the following code in each Example section. • Code to clear all fields of each structure. • Comment regarding a wait time before A/D conversion starts after open.
		55	3.2 R_ADC_Open(): Added the Special Notes (RX 63x) and Special Notes (RX110/RX111/RX113/RX210/RX130/RX230/ RX231/RX65x).
		56	3.3 R_ADC_Control(): Added the sentence to clear all members of parameters in the Description.
		65	3.3 R_ADC_Control(): Added and modified the following items: • Special Notes (RX Family Common): Added four special notes. • Special Notes (RX64M/RX71M/RX65x): Added a special note regarding operation under Group A Priority Control and modified the table. • Special Notes (RX63x) and Special Notes (RX110/RX111/ RX113): Added.
		67	3.4 R_ADC_Read(): Added Special Notes (RX110/RX111/ RX113).
		71	4. Pin Setting: Added.
		72	5.3 s12ad_poll_demo_rskrx130: Added.
		Program	Fixed typo on comment lines.
			Revised the initialization in the R_ADC_Open function.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Fixed the following issue: Target Device: RX64M/RX71M/RX230/RX231 Description: There is an error in checking the range of the arguments. Thus, when the trigger source de-selection state is set as the trigger for group B, the R_ADC_Open function returns an error. Condition: The following combination of arguments for the R_ADC_Open function is set. Second parameter (mode) ADC_MODE_SS_MULTI_CH_GROUPED or ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A Third parameter (p_cfg->trigger_groupb) ADC_TRIG_NONE_GROUPB. Measure: Modified the code for checking the arguments of the adc_check_open_cfg function. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX230/RX231 Description: The compare window A operation enable bit is not set to be enabled. Thus comparison for levels and windows does not work. Condition: Comparison does not work under any condition. Measure: Modified the code to enable the CMPAE bit using the adc_control function when the compare function is selected. Use Rev. 2.20 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Fixed the following issue: Target Device: RX64M/RX71M/RX230/RX231 Description: After Disconnection Detection Assist (DDA) is set, the register is not reset. Thus the Disconnection Detection Assist (DDA) setting remains and this causes a combination error when setting self-diagnosis. Then the R_ADC_Control function returns an error. Condition: After Disconnection Detection Assist (DDA) is set, the FIT module is closed and re-opened, and then self-diagnosis is set. Measure: Added processing to reset all S12AD related registers in the adc_open function and deleted the check during Disconnection Detection Assist (DDA) operation from the check with self-diagnosis set in the adc_check_scan_config function. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX230/RX231 Description: The numbers of arguments (enum value) for an index of the register table do not match and the indexed value becomes out of range. Then the R_ADC_Read function cannot obtain the result of self-diagnosis. Condition: Occurs under any conditions. Measure: Deleted unnecessary definitions from the enum (abc_reg_t) for an index of the register table. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX210 Description: A parameter needed for compiling was deleted in rev. 2.10, thus a build error occurs when compiling with RX210. Condition: A project with Rev.2.10 or Rev.2.11 of the ADC FIT module is built. Measure: Added ADC_CFG_PGA_GAIN to r_s12ad_rx_config.h. Use Rev. 2.20 or later version of the ADC FIT module.
			Deleted unnecessary definitions.
			Deleted unnecessary members.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Modified the following procedures according to the User's Manual: Hardware: • Procedure for when A/D conversion stops • Procedure for when entering low power consumption modes • Procedure to rewrite the ADHSC bit
			Fixed the following issue: Target Device: RX64M/RX71M/RX230/RX231 Description: Since the operator is incorrect in processing to avoid the upper limit voltage becoming less than the lower limit voltage, the upper and lower limit voltages cannot be set to the same value in the comparison setting. Condition: The comparison (window comparison) is used. Measure: Use Rev. 2.20 or later version of the ADC FIT module.
			Modified the code to set the delay time properly when converting the temperature sensor in RX64M and RX71M.
			Modified processing for checking an invalid channel when using the extended analog input in RX64M and RX71M.
			Unify the name of definitions that have same meanings but have different names among MCU Groups. RX63x ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRG0AN_0 → ADC_TRIG_SYNC_TRG0AN ADC_TRIG_SYNC_TRG0BN_0 → ADC_TRIG_SYNC_TRG0BN ADC_TRIG_SYNC_TRGAN_0 → ADC_TRIG_SYNC_TRGAN ADC_TRIG_SYNC_TRGAN_1 → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TRG0EN_0 → ADC_TRIG_SYNC_TRG0EN ADC_TRIG_SYNC_TRG0FN_0 → ADC_TRIG_SYNC_TRG0FN ADC_TRIG_SYNC_TRG4ABN_0 → ADC_TRIG_SYNC_TRG4AN_OR_TRG4BN ADC_TRIG_SYNC_TRG4ABN_1 → ADC_TRIG_SYNC_TPUTRG0AN ADC_TRIG_SYNC_TMRTRG0AN_0 → ADC_TRIG_SYNC_TMRTRG0AN ADC_TRIG_SYNC_TMRTRG0AN_1 → ADC_TRIG_SYNC_TMRTRG2AN

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	RX110 ADC_CONVERT_SPEED_HI → ADC_CONVERT_SPEED_HIGH ADC_TRIG_NONE_GROUPB → Deleted ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG RX111 ADC_CONVERT_SPEED_HI → ADC_CONVERT_SPEED_HIGH ADC_TRIG_NONE_GROUPB → Deleted ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN RX113 ADC_CONVERT_SPEED_HI → ADC_CONVERT_SPEED_HIGH ADC_TRIG_NONE_GROUPB → Deleted ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN RX210 ADC_TRIG_NONE_GROUPB → Deleted ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_PLACEHOLDER → ADC_TRIG_SYNC_TEMPS ADC_TRIG_SYNC_TRGAN1 → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TRG4ABN1 → ADC_TRIG_SYNC_TPUTRG0AN

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	RX64M ADC_CMD_CONFIGURE_SCAN → ADC_CMD_ENABLE_CHANS ADC_TRIG_NONE_GROUPB → ADC_TRIG_NONE ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGA0N → ADC_TRIG_SYNC_TRG0AN ADC_TRIG_SYNC_TRGA1N → ADC_TRIG_SYNC_TRG1AN ADC_TRIG_SYNC_TRGA2N → ADC_TRIG_SYNC_TRG2AN ADC_TRIG_SYNC_TRGA3N → ADC_TRIG_SYNC_TRG3AN ADC_TRIG_SYNC_TRGA4N → ADC_TRIG_SYNC_TRG4AN_OR_UDF4N ADC_TRIG_SYNC_TRGA6N → ADC_TRIG_SYNC_TRG6AN ADC_TRIG_SYNC_TRGA7N → ADC_TRIG_SYNC_TRG7AN_OR_UDF7N ADC_TRIG_SYNC_TRG0N → ADC_TRIG_SYNC_TRG0EN ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_SYNC_TRG7ABN → ADC_TRIG_SYNC_TRG7AN_AND_TRG7BN ADC_TRIG_SYNC_GTADTRA0N → ADC_TRIG_SYNC_GTADTR0AN ADC_TRIG_SYNC_GTADTRB0N → ADC_TRIG_SYNC_GTADTR0BN ADC_TRIG_SYNC_GTADTRA1N → ADC_TRIG_SYNC_GTADTR1AN ADC_TRIG_SYNC_GTADTRB1N → ADC_TRIG_SYNC_GTADTR1BN ADC_TRIG_SYNC_GTADTRA2N → ADC_TRIG_SYNC_GTADTR2AN ADC_TRIG_SYNC_GTADTRB2N → ADC_TRIG_SYNC_GTADTR2BN ADC_TRIG_SYNC_GTADTRA3N → ADC_TRIG_SYNC_GTADTR3AN ADC_TRIG_SYNC_GTADTRB3N → ADC_TRIG_SYNC_GTADTR3BN ADC_TRIG_SYNC_GTADTRA0N_OR_GTADTRB0N → ADC_TRIG_SYNC_GTADTR0AN_OR_GTADTR0BN ADC_TRIG_SYNC_GTADTRA1N_OR_GTADTRB1N → ADC_TRIG_SYNC_GTADTR1AN_OR_GTADTR1BN ADC_TRIG_SYNC_GTADTRA2N_OR_GTADTRB2N → ADC_TRIG_SYNC_GTADTR2AN_OR_GTADTR2BN ADC_TRIG_SYNC_GTADTRA3N_OR_GTADTRB3N → ADC_TRIG_SYNC_GTADTR3AN_OR_GTADTR3BN ADC_TRIG_SYNC_TMTRG0AN_0 → ADC_TRIG_SYNC_TMRTRG0AN

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	ADC_TRIG_SYNC_TMTRG0AN_1 → ADC_TRIG_SYNC_TMRTRG2AN ADC_TRIG_SYNC_TPTRGAN → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TPTRG0AN → ADC_TRIG_SYNC_TPUTRG0AN ADC_TRIG_SYNC_ELCTRG → ADC_TRIG_SYNC_ELC RX71M ADC_CMD_CONFIGURE_SCAN → ADC_CMD_ENABLE_CHANS ADC_TRIG_NONE_GROUPB → ADC_TRIG_NONE ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGA0N → ADC_TRIG_SYNC_TRG0AN ADC_TRIG_SYNC_TRGA1N → ADC_TRIG_SYNC_TRG1AN ADC_TRIG_SYNC_TRGA2N → ADC_TRIG_SYNC_TRG2AN ADC_TRIG_SYNC_TRGA3N → ADC_TRIG_SYNC_TRG3AN ADC_TRIG_SYNC_TRGA4N → ADC_TRIG_SYNC_TRG4AN_OR_UDF4N ADC_TRIG_SYNC_TRGA6N → ADC_TRIG_SYNC_TRG6AN ADC_TRIG_SYNC_TRGA7N → ADC_TRIG_SYNC_TRG7AN_OR_UDF7N ADC_TRIG_SYNC_TRG0N → ADC_TRIG_SYNC_TRG0EN ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_SYNC_TRG7ABN → ADC_TRIG_SYNC_TRG7AN_AND_TRG7BN ADC_TRIG_SYNC_GTADTRA0N → ADC_TRIG_SYNC_GTADTR0AN ADC_TRIG_SYNC_GTADTRB0N → ADC_TRIG_SYNC_GTADTR0BN ADC_TRIG_SYNC_GTADTRA1N → ADC_TRIG_SYNC_GTADTR1AN ADC_TRIG_SYNC_GTADTRB1N → ADC_TRIG_SYNC_GTADTR1BN

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	ADC_TRIG_SYNC_GTADTRA2N → ADC_TRIG_SYNC_GTADTR2AN ADC_TRIG_SYNC_GTADTRB2N → ADC_TRIG_SYNC_GTADTR2BN ADC_TRIG_SYNC_GTADTRA3N → ADC_TRIG_SYNC_GTADTR3AN ADC_TRIG_SYNC_GTADTRB3N → ADC_TRIG_SYNC_GTADTR3BN ADC_TRIG_SYNC_GTADTRA0N_OR_GTADTRB0N → ADC_TRIG_SYNC_GTADTR0AN_OR_GTADTR0BN ADC_TRIG_SYNC_GTADTRA1N_OR_GTADTRB1N → ADC_TRIG_SYNC_GTADTR1AN_OR_GTADTR1BN ADC_TRIG_SYNC_GTADTRA2N_OR_GTADTRB2N → ADC_TRIG_SYNC_GTADTR2AN_OR_GTADTR2BN ADC_TRIG_SYNC_GTADTRA3N_OR_GTADTRB3N → ADC_TRIG_SYNC_GTADTR3AN_OR_GTADTR3BN ADC_TRIG_SYNC_TMTRG0AN_0 → ADC_TRIG_SYNC_TMRTRG0AN ADC_TRIG_SYNC_TMTRG0AN_1 → ADC_TRIG_SYNC_TMRTRG2AN ADC_TRIG_SYNC_TPTRGAN → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TPTRG0AN → ADC_TRIG_SYNC_TPUTRG0AN ADC_TRIG_SYNC_ELCTRG → ADC_TRIG_SYNC_ELC RX130 ADC_TRIG_NONE_GROUPB → ADC_TRIG_NONE ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_SYNC_ELCTRG0 → ADC_TRIG_SYNC_ELC RX230 ADC_TRIG_NONE_GROUPB → ADC_TRIG_NONE ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N → ADC_TRIG_SYNC_ELC ADC_TRIG_SYNC_TRGAN1 → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TRG4ABN1 → ADC_TRIG_SYNC_TPUTRG0AN

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	RX231 ADC_TRIG_NONE_GROUPB → ADC_TRIG_NONE ADC_TRIG_ASYNC_ADTRG0 → ADC_TRIG_ASYNC_ADTRG ADC_TRIG_SYNC_TRGAN → ADC_TRIG_SYNC_TRGAN_OR_UDF4N ADC_TRIG_SYNC_TRG4ABN → ADC_TRIG_SYNC_TRG4AN_AND_TRG4BN ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N → ADC_TRIG_SYNC_ELC ADC_TRIG_SYNC_TRGAN1 → ADC_TRIG_SYNC_TPUTRGAN ADC_TRIG_SYNC_TRG4ABN1 → ADC_TRIG_SYNC_TPUTRG0AN
			Unify the member names in the adc_ch_cfg_t structure that are different among MCU Groups. RX64M/RX71M scan_mask → chan_mask scan_mask_groupb → chan_mask_groupb
			Deleted processing for checking the range of enum value to simplify the processing. * See the warning on compiling to check the enum range.
			Fixed the following issue: Target Device: RX210 Description: In the processing for checking arguments, ADC_TRIG_SYNC_TEMPS is checked with “trigger” instead of "trigger_groupb". Then the R_ADC_Open function returns an error even if the ADC_TRIG_SYNC_TEMPS setting is valid. Condition: ADC_TRIG_SYNC_TEMPS is set as the trigger of A/D conversion. Measure: Deleted the code for checking ADC_TRIG_SYNC_TEMPS in the adc_open function. * “trigger_groupb” is ignored in modes other than group scan mode. In group scan mode, if ADC_TRIG_SYNC_TEMPS is set to trigger_groupb, an error is returned. Thus the checking process for ADC_TRIG_SYNC_TEMPS is unnecessary. Use Rev. 2.20 or later version of the ADC FIT module.
			Added the temperature sensor (temp) and internal reference voltage (volt) to the adc_data_t structure in the RX63x, RX110, RX111, RX113, and RX210 Groups to unify the behavior of the R_ADC_ReadAll function over all MCU groups.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Fixed the following issue: Target Device: RX64M/RX71M Description: In the processing for checking arguments, ADC_TRIG_NONE is checked with "trigger". Then the R_ADC_Open function returns an error even if the ADC_TRIG_NONE setting is valid. Condition: ADC_TRIG_NONE is set as the trigger of A/D conversion. Measure: Deleted the code for checking ADC_TRIG_NONE in the adc_open function since ADC_TRIG_NONE can be set to the TRSA register as well as the TRSB register. Use Rev. 2.20 or later version of the ADC FIT module.
			Modified the code to reset the ADGSPCR register when setting a mode other than group scan mode in the RX130, RX230, RX231, RX64M, and RX71M.
			Changed the structure for arguments of comparison in RX130/RX230/RX231 to similar to the structure in RX65N. The adc_cmplvl_t structure has been discarded, accordingly. Please use the adc_cmpwin_t structure when using the level comparison.
			Fixed the following issue: Target Device: RX130/RX230/RX231 Description: No processing is provided to set the compare window operation enable bit to "disabled". Thus once the compare function is enabled, only the way to disable it is reopening. However, please note that reopening does not work for RX230 and RX231. Condition: Always occurs when the compare function is used. Measure: Added "windowa_enable" to the structure for arguments of the compare function. Now the compare window operation enable bit can be set to "enabled" or "disabled" according to true/false setting of "windowa_enable", i.e. same processing as RX65N. Use Rev. 2.20 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Fixed the following issue: Target Device: RX64M/RX71M Description: No processing is provided to set the WCMPE bit to 0 (level comparison). Thus once window comparison is enabled, the comparison cannot be set to level comparison. Condition: The comparison is reset to level comparison after setting to window comparison. Measure: Modified the code to properly set the WCMPE bit according to the selection of window or level comparison. Use Rev. 2.20 or later version of the ADC FIT module.
			Modified the code to use the interface provided in the BSP (R_BSP_InterruptControl function) for specifying the interrupt enable bit and interrupt priority level when the compare interrupt is used in RX64M and RX71M.
			Fixed the following issue: Target Device: RX64M/RX71M Description: No processing is provided to set the compare interrupt enable bit to “disabled”. Thus once the comparison is enabled, the compare interrupt cannot be disabled. Condition: The interrupt priority level is set to 1 or greater while the comparison is enabled. Measure: Modified the code to disable the compare interrupt when executing the adc_close function and to disable group interrupts if no FIT module uses group interrupts. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX130/RX230/RX231/RX64M/RX71M Description: An unspecified callback function (NULL) is executed and improper interrupt occurs. Condition: After the R_ADC_Open function is executed with interrupts disabled, the interrupt priority level of the compare interrupt is set to 1 or greater. Measure: Modified the code to check the callback function before executing it. If the callback function is NULL, the interrupt handler is exited without performing any processing. Use Rev. 2.20 or later version of the ADC FIT module.
			Deleted unnecessary processing to reset the register when enabling an output of the temperature sensor in RX210 since the register is already reset to 0.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Replaced the RX113 provided wait function (adc_delay) with the BSP provided wait function (R_BSP_SoftwareDelay). * The RX113 provided wait function (adc_delay) has been deleted.
			Fixed the following issue: Target Device: RX210 Description: An unnecessary error determination is performed. Because of this, when specifying a setting with the channel-dedicated sample-and-hold function, the R_ADC_Control function returns an error. Condition: In group scan mode, A/D conversion channels for group A and group B are set with the channel-dedicated sample-and-hold function. Measure: Deleted an unnecessary error determination as no limitation regarding it is described in the User's Manual: Hardware. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX210 Description: Since an error determination processing is not provided, if self-diagnosis is enabled in a mode where self-diagnosis does not work, the R_ADC_Control function cannot return an error. Condition: Self-diagnosis is enabled when double trigger mode is selected in single scan mode or group scan mode. Measure: Added the error determination processing for when self-diagnosis is enabled. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX130/RX230/RX231 Description: An unnecessary error determination is performed. Because of this, when setting the disconnection detection assist function after self-diagnosis is enabled, the R_ADC_Control function returns an error. Condition: Discharge or precharge is selected for the disconnection detection assist function after self-diagnosis is enabled. Measure: Deleted unnecessary determination processing described in the Description above. Use Rev. 2.20 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	Fixed the following issue: Target Device: RX63x Description: The definition to determine a valid channel is incorrect and channel 20 cannot be selected. Condition: A chip with 177, 176, 145, or 144 pins is selected. Measure: Modified the definition to determine a valid channel. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX631 Description: There is no definition to determine a valid channel and this causes a compiling error. Condition: A chip with 64 pins or 48 pins is selected. Measure: Added the definition to determine a valid channel. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX64M/RX71M/RX65x Description: When obtaining the compare match result, the compare channel is cleared. Then, the subsequent compare match is not performed. Condition: When any of the unit 1 channel from channel 16 to channel 20 is specified as the compare channel, the condition is met and the compare match interrupt occurs, or the R_ADC_Control function is executed by setting ADC_CMD_CHECK_CONDITION_MET. Measure: Modified the register that was initialized when obtaining the compare match result. Use Rev. 2.20 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX64M/RX71M/RX65x/RX130/RX230/RX231 Description: When enabling self-diagnosis under a prohibited setting condition, the operation ends normally. Condition: Self-diagnosis is enabled in double trigger mode with single scan mode selected. Measure: Modified processing for checking the error condition when self-diagnosis is enabled. Use Rev. 2.20 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.20	Dec.01.16	Program	In RX63x and RX210, the TEMPS register is now modified only when the temperature sensor module is enabled.
			Added the definition "ADC_CONVERT_SPEED_DEFAULT" for conversion speed of A/D conversion in RX110, RX111, and RX113. "ADC_CONVERT_SPEED_DEFAULT" has the same value as "ADC_CONVERT_SPEED_NORM".
			Fixed the following issue: Target Device: RX110 Description: An error occurs when attempting to set the minimum value for the number of sampling states. Condition: An error occurs whenever the number of sampling states can be set. Measure: Modified the definition of the minimum value for the number of sampling states. Use Rev. 2.20 or later version of the ADC FIT module.
			In RX64M, RX71M, RX65x, RX130, RX230, and RX231, some function declarations differed from prototypes. These function declarations now correspond to the prototypes.
2.30	Jul.24.17	—	Applications of descriptions are now indicated by the S12AD peripherals (not MCUs).
			Added support for RX65N-2MB (177 pins and 176 pins).
			Added support for RX130-512KB (100 pins).
		1	Related Documents: Added the following document: “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)”
		3	1. Overview: Revised the descriptions.
		4	2.5 Supported Toolchains: The information of the toolchains are now described in 6.1.
		5	2.6 Interrupt Vector: Added
		6-7	2.10 Code Size: Updated the sizes according to changes in the program.
		7-43	2.11 API Data Structures: Revised. Now descriptions have given by each structure.
		44	2.13 Adding a FIT Module to Your Project: Revised.
		46-48	3.2 R_ADC_Open(): Revised.
		49-61	3.3 R_ADC_Control(): Revised.
		66	4. Pin Setting: Revised.
		68	5.8 Downloading Demo Projects: Added.
		69, 70	6. Appendices: Added.
		Program	In RX65N, deleted processing for checking the range of enum value to simplify the processing. * See the warning on compiling to check out-of-range for enum.

Rev.	Date	Description	
		Page	Summary
2.30	Jul.24.17	Program	Fixed the following issue: Target Device: RX130/RX230/RX231/RX64M/RX71M/RX65N Description: When a channel is opened in a mode other than group scan mode, even if the parameter only available for group scan mode is set for the channel, an error does not occur. Condition: When in a mode other than group scan mode, a channel for group B, channel for group C (RX65N only), and group priority control is set. Measure: Modified processing to check invalid combination in group scan mode and return an error. Use Rev. 2.30 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX130/RX230/RX231/RX64M/RX71M/RX65N Description: The procedure to specify the register for group priority control does not follow the procedure in the User's Manual: Hardware. Due to this, scanning operation and the result stored cannot be guaranteed. Condition: Group priority control is used. Measure: Modified the register setting procedure for group priority control. Use Rev. 2.30 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX65N Description: When the interrupt priority level is set (interrupt enabled) without specifying the callback function, an error does not occur. Condition: The interrupt priority level is set to 1 or greater. Measure: Modified the checking procedure at open to return an error. Use Rev. 2.30 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.30	Jul.24.17	Program	Fixed the following issue: Target Device: RX65N Description: Even if addition mode is specified with an invalid combination, an error does not occur. Condition: When "16 samples" is selected for addition mode, 10-bit accuracy or 8-bit accuracy is selected. Measure: Modified the checking procedure at open to return an error. Use Rev. 2.30 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX65N Description: The procedure to stop A/D conversion does not follow the procedure described in the User's Manual: Hardware. Due to this, an unexpected operation may be performed. Condition: Close processing is performed with group priority control enabled. Measure: Modified the register setting procedure at close. Use Rev. 2.30 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX65N Description: Window B comparison condition may not be specified correctly. Condition: With comparison function, window B comparison condition is set to 2 or greater. Measure: Window B comparison condition does not have range check. Thus, the code has been modified to return an error when an out-of-range error occurs. Use Rev. 2.30 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
2.30	Jul.24.17	Program	Fixed the following issue: Target Device: RX65N Description: The trigger for group A cannot be set to the external trigger. Condition: Double trigger is disabled in group scan mode. Measure: The external trigger was disabled in RX65N (same as the RX64M). For RX65N, the external trigger now can be set only for group A. Use Rev. 2.30 or later version of the ADC FIT module.
			Fixed the following issue: Target Device: RX65N Description: The result cannot be obtained when the window A/B complex condition is set. Condition: Occurs at any time. Measure: Added I/F for obtaining the comparison result with window A/B complex condition to the R_ADC_Control function. Use Rev. 2.30 or later version of the ADC FIT module.

Rev.	Date	Description	
		Page	Summary
3.00	Sep.03.18	—	Added RX66T support.
		—	Updated Demo projects.
		1	Introduction: Modified content.
			Target Devices: Added RX66T Group.
		4	1. Overview: Modified content.
			1.1 ADC FIT Module: Added section.
			1.2 ADC FIT Module Overview: Added section.
		6	1.3 API Overview: Added section.
		7	1.4 Processing Examples: Added section.
		13	1.5 Restrictions: Added section.
		14	2. API Information: Modified content.
			2.1 Hardware Requirements: Modified content.
			2.2 Hardware Resource Requirements: Deleted section.
			2.3 Supported Toolchain: Modified content.
		15	2.4 Limitations: Deleted section.
			2.4 Interrupt Vectors: Added content.
		16	2.7 Configuration Overview: Modified content.
		17	2.8 Code Sizes: Added content.
		18 to 36	2.9 Arguments: Modified content.
		37	2.11 Callback Functions: Added section.
		38	2.13 “for”, “while” and “do while” statements: Added section.
		39	3.1 Summary: Deleted section.
		39 to 41	3.1 R_ADC_Open(): Modified content.
		42 to 55	3.2 R_ADC_Control(): Modified content.
		56	3.3 R_ADC_Read(): Modified content.
		57	3.4 R_ADC_ReadAll(): Modified content.
		58	3.5 R_ADC_Close(): Modified content.
		59	3.6 R_ADC_GetVersion(): Modified content.
		60	4. Pin Setting: Added content.
		61, 62	5. Demo Projects: Added content.
		62	6.1 Operation Confirmation Environment: Added content.
		Program	Fixed the following issue: Target Device: RX64M/RX65N/RX71M Description: The temperature sensor setting does not follow the procedure in the User's Manual: Hardware.
			Fixed the following issue: Target Device: RX65N Description: The trigger setting for group priority control does not follow the procedure in the User's Manual: Hardware.
3.01	Dec.03.18	43	3.2 R_ADC_Control(): Modified command name of Description.
		58	3.5 R_ADC_Close(): Modified content of Return Values.

Rev.	Date	Description	
		Page	Summary
3.01	Dec.03.18	62	6.1 Operation Confirmation Environment: Corrected board used in Table 6.2 Confirmed Operation Environment (Rev. 3.00). Added Table 6.3 Confirmed Operation Environment (Rev. 3.01).
		Program	Added document number of the application note accompanying the sample program of the FIT module to xml file.
3.10	Feb.15.19	—	Added support for RX72T. Added support for RX651 with 64 pin package.
		1	Target Devices: Added RX72T Group.
		4	Table 1.1 Operating Modes Supported by ADC FIT Module: Updated.
		4	Table 1.2 Functions Supported by ADC FIT Module: Updated.
		5	Table 0.1 S12AD Supported by Each MCU: Updated.
		15	Table 0.2 Interrupt Vector Used in the ADC FIT Module: Updated.
		16	2.8 Code Sizes: Added content.
		60	Table 0.3 Function Output by the FIT Configurator: Updated.
		64	Table 0.4 Confirmed Operation Environment (Rev. 3.10) : Updated.
4.00	Apr.05.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Deleted the RX210, RX631, and RX63N in Target Devices for end of update these devices.
			Added the section of Target compilers.
			Deleted related documents.
		4, 5	1.2 ADC FIT Module Overview: Deleted the description of RX210, RX631, and RX63N.
		14	2.2 Software Requirements: Added the revision number of depending module.
		15	2.4 Interrupt Vector: Deleted the description of RX210, RX631, RX63N, and S12ADa.
		16	2.7 Configuration Overview: Deleted the macro definition of ADC_CFG_PGA_GAIN.
		17	Updated the section of 2.8 Code Size.
		19, 20	2.9.2 Structures and Enumerations Used as Arguments for R_ADC_Open Function: Deleted the note of S12ADa.
			Updated the section of 2.12 Adding the FIT module to Your Project.
		40	3.1 R_ADC_Open(): Deleted the description of RX210.
		41	3.1 R_ADC_Open(): Deleted the Special Note(S12ADa).
		54	3.2 R_ADC_Control(): Deleted the note of S12ADa.
		55	3.2 R_ADC_Control(): Deleted the Special Note(S12ADa).
		56	3.3 R_ADC_Read():Deleted the description of RX210.
		59	Updated the section of 3.6 R_ADC_GetVersion().
		60	4. Pin Setting: Deleted the description of RX210, RX631, and RX63N.

Rev.	Date	Description	
		Page	Summary
4.00	Apr.05.19	64	Table 6.5 Confirmed Operation Environment (Rev. 4.00) : Updated.
		66	Deleted the section of Website and Support.
		program	Deleted the process of RX210, RX631, and RX63N for end of update these devices.
			Changed bellow for support GCC and IAR compiler: 1. Deleted the inline expansion of the R_ADC_GetVersion function. 2. Replaced evenaccess with the macro definition of BSP. 3. Replaced nop with the intrinsic functions of BSP. 4. Replaced the declaration of interrupt functions with the macro definition of BSP.
			Changed the processing to prevent register access contention between peripheral functions that occurs when using RTOS or when multiple interrupts are enabled. 1. Changed the setting process of the Interrupt Request Enable Bits (IEN) [Description] Changed the setting process of the Interrupt Request Enable Bits (IEN) to use R_BSP_InterruptRequestDisable, and R_BSP_InterruptRequestEnable in the API functions of BSP. 2. Changed the setting process of the Group Interrupt Request Enable Register (GENBL1) (RX64M, RX65N, RX66T, RX71M, and RX72T). [Description] Changed to perform the setting process of the Group Interrupt Request Enable Register (GENBL1) while interrupts are disabled.
4.10	Jun.28.19	—	Added support for RX23W.
		program	Fixed the following issue: Target Device: RX230/RX231 Description: The value of ADC_SST_CNT_MIN changes, according to the clock setting. PCLKB < ADCLK #define ADC_SST_CNT_MIN (6) PCLKB >= ADCLK #define ADC_SST_CNT_MIN (5)
4.20	Jul.31.19	—	Added support for RX72M.
		1	Target Devices: Added RX72M Group.
		4	Table 1.1 Operating Modes Supported by ADC FIT Module: Updated.
		4	Table 1.2 Functions Supported by ADC FIT Module: Updated.
		5	Table 1.3 S12AD Supported by Each MCU: Updated.
		15	Table 2.1 Interrupt Vector Used in the ADC FIT Module: Updated.
		61	4. Pin Setting: Revised the descriptions.

Rev.	Date	Description	
		Page	Summary
4.20	Jul.31.19	65	Table 6.7 Confirmed Operation Environment (Rev. 4.20) : Updated.

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

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/.