RX Family
VBATT Module Firmware Integration Technology

Introduction

This application note describes a battery backup function module that uses Firmware Integration Technology (FIT).

This module reports to the user whether or not a voltage drop has occurred in either the battery backup supply voltage or the VBATT pin voltage. Based on the content reported, the user can determine whether or not the value of the real-time clock can be guaranteed, and whether or not the VBATT pin voltage has fallen.

This module is referred to as the battery backup function FIT module in the remainder of this document.

Target Device

- RX230 Group
- RX231 Group
- RX23W Group
- RX671 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 “4.1 Confirmed Operation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

1.1 The Battery Backup Function FIT Module

This module is embedded in projects as an API. See section 2.12, Adding the FIT Module to Your Project, for embedding this module.

1.2 Battery Backup Function FIT Module Overview

When this module is used, the VBATT pin voltage drop detection function can be set up and the state of the battery backup function can be read out. Furthermore, the following five states can be recognized using callback function arguments.

< Common >
(1) No battery backup supply voltage drop detected
(2) Battery backup supply voltage drop detected
< RX230, RX231 and RX23W >
(3) A nonmaskable interrupt due to VBATT pin voltage drop detection occurred
(4) A maskable interrupt due to VBATT pin voltage drop detection occurred
< RX671 >
(5) A maskable interrupt due to Tamper detection occurred

1.2.1 API Function Specifications

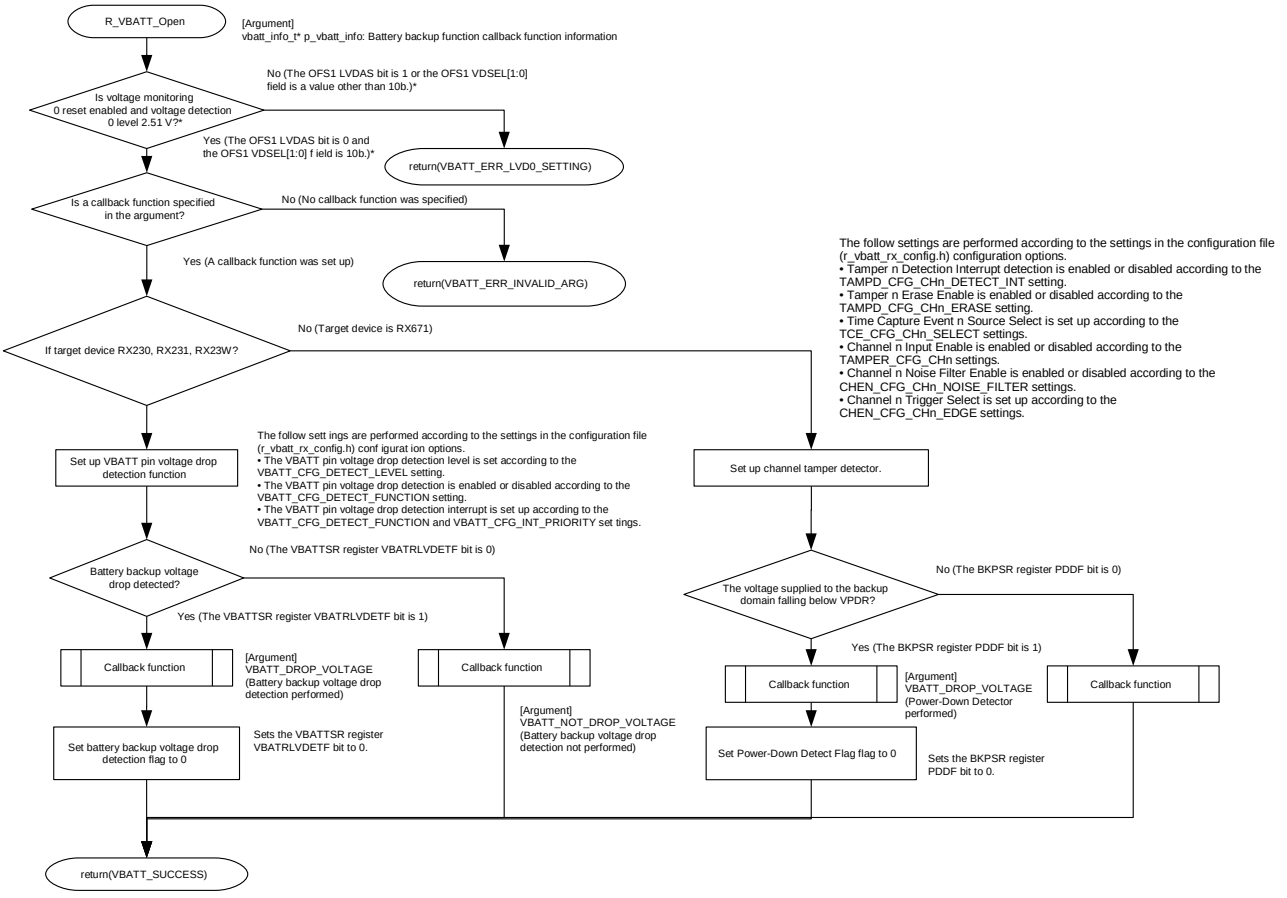
When the R_VBATT_Open() function is called, it checks whether or not the voltage monitoring 0 settings are correct and whether or not a callback function was specified and if there is a problem, it returns an error and terminates function execution. If the argument settings are valid, it sets up the VBATT pin voltage drop detection function (RX230, RX231 and RX23W) or sets Tamper detection (RX671) according to the settings in the configuration file (r_vbatt_rx_config.h) configuration options. After that it checks if the battery backup voltages (the VCC and VBATT pins) have fallen (the VBATTSR register VBATRLVDETF bits (RX230, RX231 and RX23W) or the BKPSR register PDDF bit (RX671)) and performs the following processing according to that result.

If a battery backup voltage drop has been detected, it calls the callback function with VBATT_DROP_VOLTAGE as the argument. After calling the callback function, it sets the battery backup voltage drop detection flag to 0 (battery backup voltage drop not detected).

If no battery backup voltage drop was detected, it calls the callback function with VBATT_NOT_DROP_VOLTAGE as the argument.

Figure 1.1 shows the flowchart of the R_VBATT_Open() function.

See section 2.7, Configuration Overview, and section 2.11, Callback Function, for details on the configuration file options and macro definitions used as callback function arguments.



Note: * For the RX231 microcontroller. For RX671, the voltage detection level (OFS1.VDSEL bit) can be selected from 2.94V (01b), 2.87V (10b), or 2.80V (11b).

Figure 1.1 R_VBATT_Open() Function Flowchart

When the R_VBATT_Control() function is called, it checks whether or not the argument values are correct and, and if there is a problem, it returns an error and terminates function execution. If the argument settings are valid, it sets the VBATT pin voltage drop detection function enabled/disabled state and the detection level (RX230, RX231 and RX23W) or sets Tamper detection (RX671), and it sets up the interrupt, according to the values set in the arguments.

Figure 1.2 shows the flowchart of the R_VBATT_Control() function.

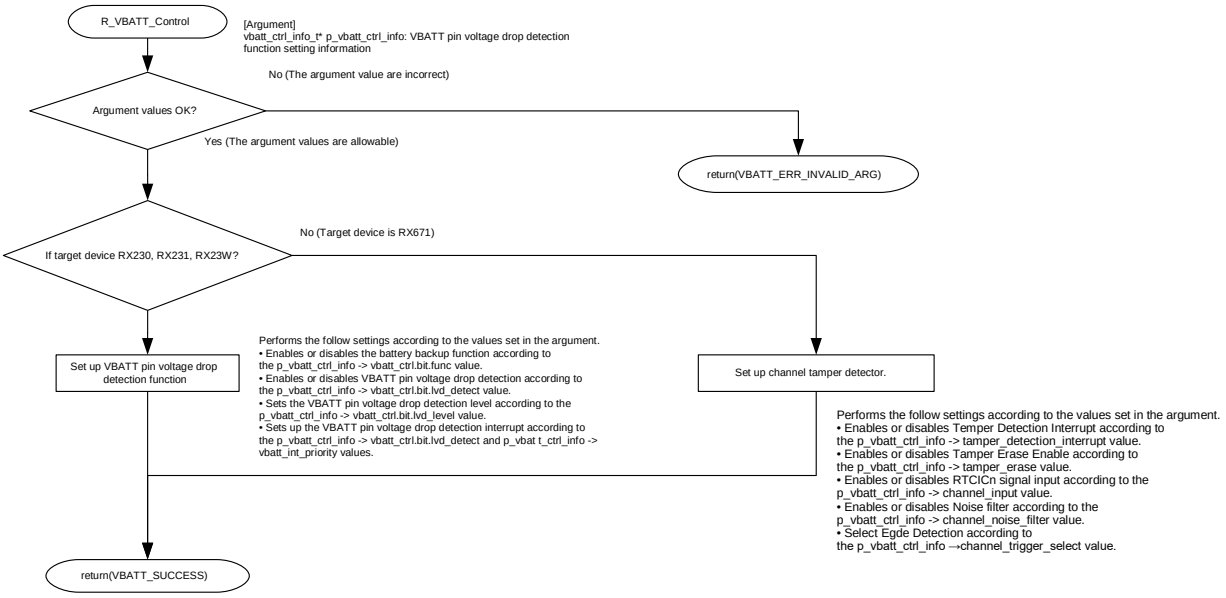


Figure 1.2 R_VBATT_Control() Function Flowchart

When the R_VBATT_GetStatus() function is called, it determines if VBATT pin voltage drop detection is enabled and if whether or not the argument values are correct and, and if there is a problem, it returns an error and terminates function execution. If the argument settings are valid, it reads out the value of the VBATT status register (VBATTSR) (RX230, RX231 and RX23W) or (TAMPIMR and TAMPSR) (RX671). The read out value is stored at the address received as an argument.

Figure 1.3 shows the flowchart of the R_VBATT_GetStatus() function.

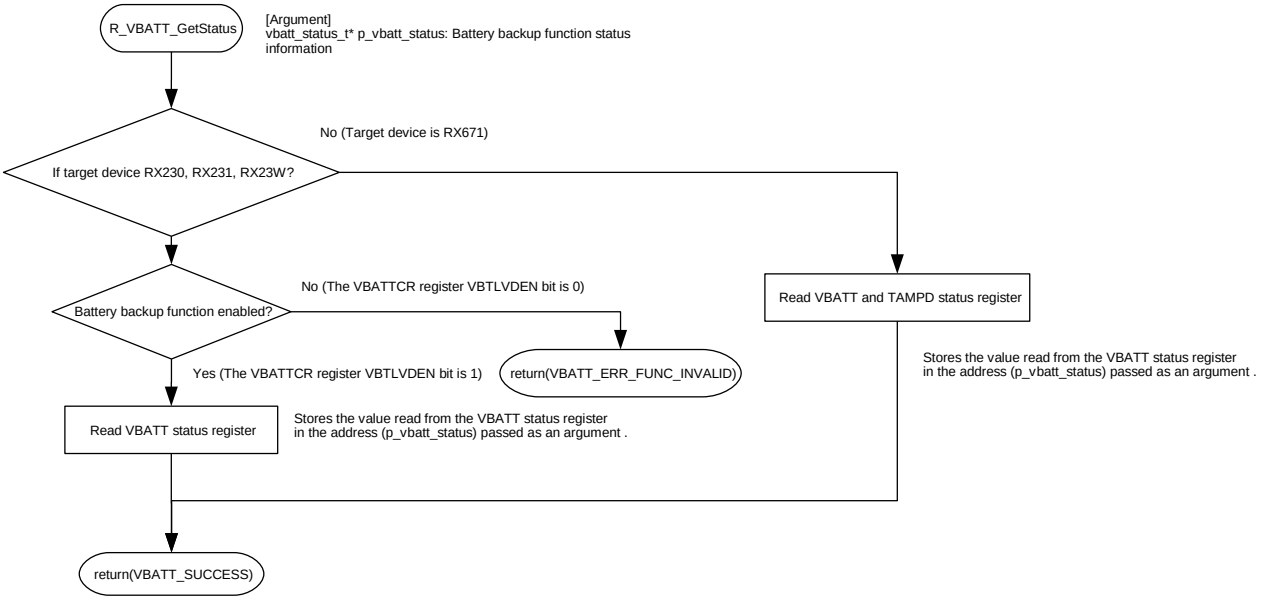


Figure 1.3 R_VBATT_GetStatus() Function Flowchart

1.2.2 Interrupt Specifications

For RX230, RX231 and RX23W, when the interrupt used a VBATT pin voltage drop is set to be a maskable interrupt, when this maskable interrupt occurs, the callback function is called with VBATT_MASKABLE_INTERRUPT as the argument.
For RX671, when the Tamper detection interrupt occurs, the callback function is called with VBATT_TAMPER_CH0_INTERRUPT, VBATT_TAMPER_CH1_INTERRUPT, VBATT_TAMPER_CH2_INTERRUPT (RX671) as the argument.

Figure 1.4 shows the flowchart of the r_vbatt_isr() function.

Figure 1.4 r_vbatt_isr() Function Flowchart

When the interrupt used when a VBATT pin voltage drop is set to be a nonmaskable interrupt, when this nonmaskable interrupt occurs, the callback function is called with VBATT_NON_MASKABLE_INTERRUPT (RX230, RX231 and RX23W) as the argument.

Figure 1.5 shows the flowchart of the r_vbatt_nmi_isr() function.

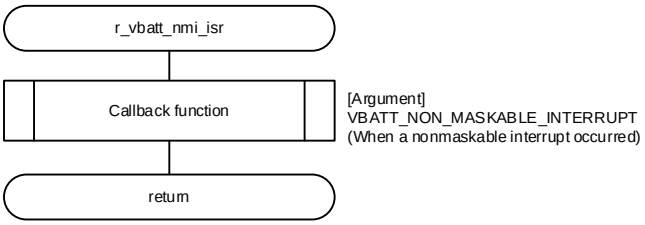


Figure 1.5 r_vbatt_nmi_isr() Function Flowchart

1.3 Using the FIT VBATT module

Using FIT VBATT module in C++ project

For C++ project, add FIT VBATT module interface header file within extern “C”{}:

```
Extern “C”
{
  #include “r_smc_entry.h”
  #include “r_vbatt_rx_if.h”
}
```

1.4 API Overview

Table 1.1 lists the API functions provided by this module.

Table 1.1 API Functions

Function	Function description
R_VBATT_Open()	In device RX230, RX231, RX23W this function sets the enabled or disabled state of the VBATT pin voltage drop detection function, the detection level, and the interrupts according to settings in the configuration options. After that, it determines whether or not the battery backup supply voltage has dropped and calls the callback function. In device RX671 this function sets the enabled or disabled state of the TAMPD function, the detection level, time capture event, channel input, channel noise filter, and the interrupts according to settings in the configuration options. After that, it determines whether or not the voltage for the backup domain has fallen below V_{PDR}. and calls the callback function .
R_VBATT_Control()	In device RX230, RX231, RX23W this function sets the enabled or disabled state of the battery backup function, the enabled or disabled state of the VBATT pin voltage drop detection function, the detection level, and the interrupts according to the settings of the arguments. In device RX671 this function sets the enabled or disabled state of the TAMPD function, the detection level, time capture event, channel input, channel noise filter, and the interrupts according to settings in the configuration to the settings of the arguments.
R_VBATT_GetStatus()	In device RX230, RX231, RX23W this function reads out the VBATT status register (VBATTSR) to acquire the status of the battery backup function. It then stores that information at the address passed as an argument. In device RX671 this function reads out the TAMPD status register to acquire the status of the tamper detector function. It then stores that information at the address passed as an argument.
R_VBATT_ReadBackupData ()	This function read Backup Register. This function is only available on device RX671
R_VBATT_WriteBackupData ()	This function write Backup Register. This function is only available on device RX671
R_VBATT_GetVersion()	Returns the version number of this module.

1.5 Usage Example

Call the R_VBATT_Open() function immediately after the reset start. The callback function should perform processing based on the value of its arguments.

Table 1.2 lists the callback function's arguments and processing that should be performed, and Figure 1.6 shows a usage example of the battery backup function FIT module.

Table 1.2 Callback Function's Arguments and Processing that Should be Performed

Argument	VBATT_NOT_DROP_VOLTAGE (Battery backup voltage drop detection not performed)	VBATT_DROP_VOLTAGE (Battery backup voltage drop detection performed)	VBATT_MASKABLE_INTERRUPT VBATT_NON_MASKABLE_INTERRUPT (VBATT pin voltage drop detection interrupt) TAMPER_INTERRUPT(Tamper detection interrupt)
Processing that Should be Performed	Reset the RAM and RTC registers as required.	The real-time clock must be initialized.	In device RX230, RX231, RX23W operations such as displaying a warning that the external battery level is low and backing up data should be performed. In device RX671 operations such as displaying a warning that the external intrusion to the system has been detected should be performed.
Reason	When the battery backup supply voltage has not dropped, the value of the RTC register will be saved, but RAM and the RTC interrupt registers will be set to their post-reset values. Therefore, the RAM and RTC interrupt register values must be set again.	When the battery backup supply voltage drops, the real-time clock registers and RAM go to their values after a reset. Therefore initialization is required.	In device RX230, RX231, RX23W this interrupt occurs when the supply voltage on the VBATT pin falls. Processing to handle the situation where the VBATT pin supply voltage has fallen can be performed. In device RX671 this interrupt occurs when the tamper detector has detected an intrusion to the system. Processing to handle the situation where the tamper detector has detected an intrusion can be performed.

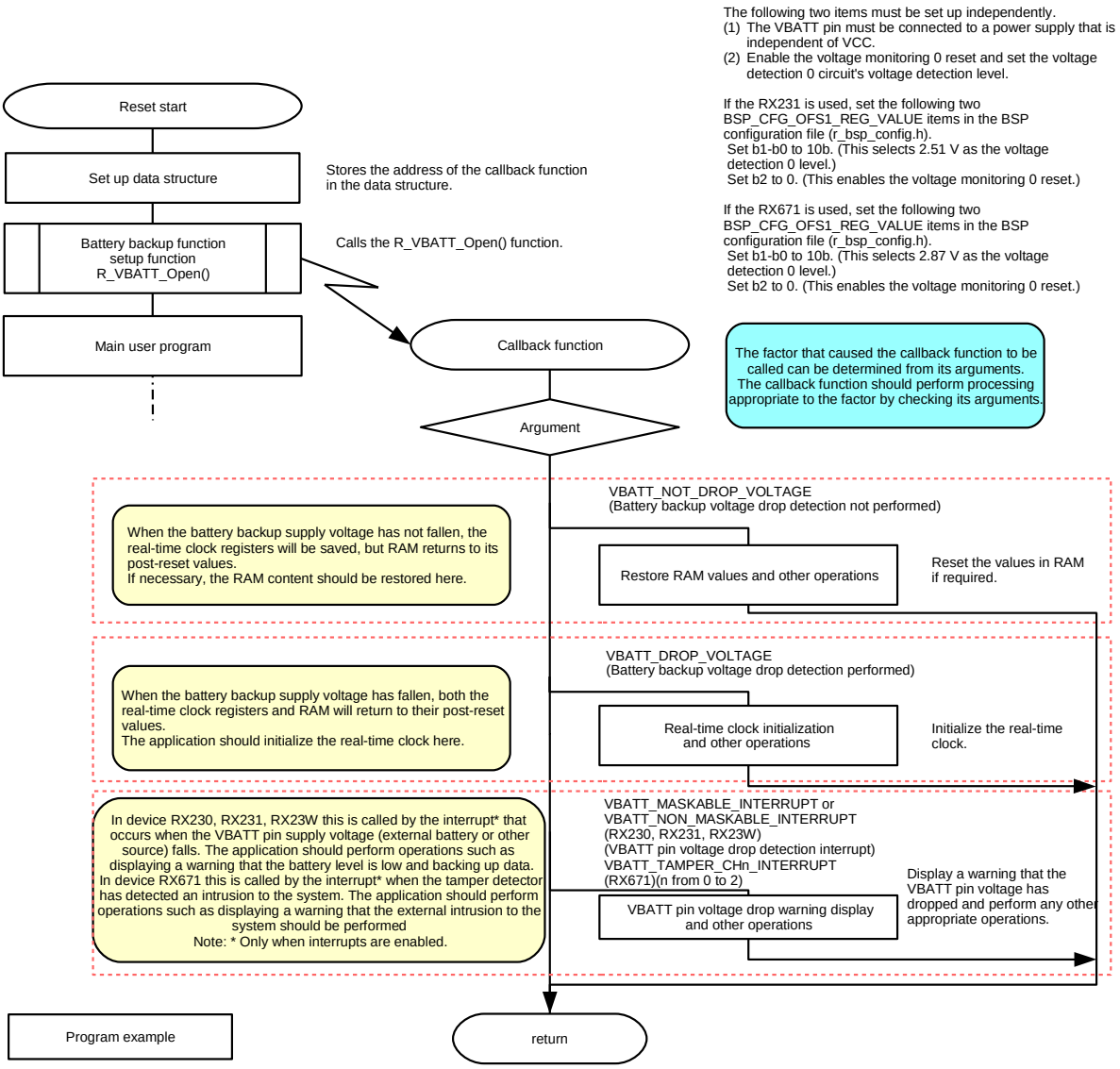


Figure 1.6 Usage Example of the Battery Backup Function FIT Module

1.6 Limitations

- The VBATT pin must be connected to a power supply independent from VCC.
- The voltage monitoring function 0 reset should be enabled and the voltage detection level for voltage detection 0 circuit should be set as follows

  RX231, RX230, RX23W Groups:
  Set the voltage detection level to 2.51V.

  RX671 Group:
  The voltage detection level can be set to 2.94V, 2.87V, or 2.80V.

- Limitations on the RX671 Group
  When using the tamper detector and the time capture function of the RTC together, use the time capture function of the RTC with the following settings, please set the noise filter setting (capture.filter) to OFF (RTC_FILTER_OFF) and set the edge detection (capture.edge) to RTC_EDGE_RISING.
  When VBATT_TAMPER_TCE_TAMPER_EVENT is selected, it is necessary to wait for the time required for time capture detection before clearing the detection flag. Please refer 4.3.1(2) Example for device RX671 for more details.

```
rtc_capture_cfg_t capture;

capture.pin = RTC_PIN_0;
capture.edge = RTC_EDGE_RISING;
capture.filter = RTC_FILTER_OFF;
err = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);
```

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The microcontroller used must support the following function.

- Battery backup function

2.2 Software Requirements

This driver is dependent upon the following FIT module:

- Renesas Board Support Package (r_bsp) v5.00 or higher

2.3 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 4.1, Confirmed Operation Environment.

2.4 Interrupt Vector

The VBATT pin voltage drop detection interrupt is enabled by executing the R_VBATT_Open function (while the macro definition VBATT_CFG_DETECT_FUNCTION is VBATT_DTCT_ENABLE_NMI_ENABLE or VBATT_DTCT_ENABLE_INT_ENABLE).

The TAMP detection interrupt is enabled by executing the R_VBATT_Open function (while the macro definition VBATT_CFG_TAMPER_CHn_DETECT_INT is VBATT_TAMPER_DETECT_INT_ENABLE.(n from 0 to 2)

Table 2.1 lists the interrupt vector used in the Battery Backup Function FIT Module.

Table 2.1 Interrupt Vector Used in the Battery Backup Function FIT Module

Device	Interrupt Vector
RX230 RX231 RX23W	VBATT pin voltage drop detection interrupt (vector no.: 91) (1)
RX671	Tamper detection interrupt (vector no : 91)

Note 1. A non-maskable interrupt is generated if the macro definition VBATT_CFG_DETECT_FUNCTION is VBATT_DTCT_ENABLE_NMI_ENABLE, and a maskable interrupt is generated if VBATT_DTCT_ENABLE_INT_ENABLE.

2.5 Header Files

All API calls and their supporting interface definitions are located in r_vbatt_rx_if.h.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

This module configuration options are set in r_vbatt_rx_config.h.

The table below lists the option names and set values.

Configuration options in r_vbatt_rx_config.h	
The following definition is used for RX231, RX230, RX23W Groups	
#define VBATT_CFG_DETECT_FUNCTION Note: The default value is "VBATT_DTCT_DISABLE"	Selects whether or not the VBATT pin voltage drop detection function is used. This also selects the interrupt generated when a voltage drop is detected. For VBATT_DTCT_DISABLE: The VBATT pin voltage drop detection function is set to invalid and the interrupt is disabled. For VBATT_DTCT_ENABLE_INT_DISABLE: The VBATT pin voltage drop detection function is enabled and the interrupt is disabled. For VBATT_DTCT_ENABLE_NMI_ENABLE: The VBATT pin voltage drop detection function is enabled and the nonmaskable interrupt is enabled as the interrupt. For VBATT_DTCT_ENABLE_INT_ENABLE: The VBATT pin voltage drop detection function is enabled and the maskable interrupt is enabled as the interrupt.
#define VBATT_CFG_DETECT_LEVEL Note: The default value is "VBATT_DTCT_LEVEL_2_20_V"	The VBATT pin voltage drop detection level can be selected. For VBATT_DTCT_LEVEL_2_20_V, the detection level is set to 2.20 V. For VBATT_DTCT_LEVEL_2_00_V, the detection level is set to 2.00 V.
The following definition is used for RX671 Group	
#define VBATT_CFG_TAMPER_CH0 Note: The default value is "VBATT_TAMPER_DISABLE"	Setting to Channel 0 Input Please choose from macro definition of the following. VBATT_TAMPER_DISABLE = The TAMPI0/RTCIC0 signal input is disabled. VBATT_TAMPER_ENABLE = The TAMPI0/RTCIC0 signal input is enabled.
#define VBATT_CFG_TAMPER_CH1 Note: The default value is "VBATT_TAMPER_DISABLE"	Setting to Channel 1 Input Please choose from macro definition of the following. VBATT_TAMPER_DISABLE = The TAMPI1/RTCIC1 signal input is disabled. VBATT_TAMPER_ENABLE = The TAMPI1/RTCIC1 signal input is enabled.
#define VBATT_CFG_TAMPER_CH2 Note: The default value is "VBATT_TAMPER_DISABLE"	Setting to Channel 2 Input Please choose from macro definition of the following. VBATT_TAMPER_DISABLE = The TAMPI2/RTCIC2 signal input is disabled. VBATT_TAMPER_ENABLE = The TAMPI2/RTCIC2 signal input is enabled.
#define VBATT_CFG_TAMPER_CH0_DETECT_INT Note: The default value is "VBATT_TAMPER_DETECT_INT_DISABLE"	Setting to Tamper 0 Detection Interrupt Enable Please choose from macro definition of the following. VBATT_TAMPER_DETECT_INT_DISABLE = Tamper 0 detection interrupt is disabled. VBATT_TAMPER_DETECT_INT_ENABLE = Tamper 0 detection interrupt is enabled.

Configuration options in r_vbatt_rx_config.h	
#define VBATT_CFG_TAMPER_CH0_ERASE Note: The default value is "VBATT_TAMPER_ERASE_DISABLE"	Setting to Tamper 0 Erase Enable Please choose from macro definition of the following. VBATT_TAMPER_ERASE_DISABLE = Backup registers are not erased in response to a tamper 0 event. VBATT_TAMPER_ERASE_ENABLE = Backup registers are erased in response to a tamper 0 event.
#define VBATT_CFG_TAMPER_TCE_CH0_SELECT Note: The default value is "VBATT_TAMPER_TCE_RTCIC_PIN"	Setting to Time Capture Event 0 Source Select Please choose from macro definition of the following. VBATT_TAMPER_TCE_RTCIC_PIN = Input signal from the RTCIC0 pin. VBATT_TAMPER_TCE_TAMPER_EVENT = Tamper 0 event.
#define VBATT_CFG_TAMPER_CHEN_CH0_NOISE_FILTER Note: The default value is "VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE"	Setting to Channel 0 Noise Filter Enable Please choose from macro definition of the following. VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE = Noise filter for the TAMPI0/RTCIC0 pin is disabled. VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE = Noise filter for the TAMPI0/RTCIC0 pin is enabled.
#define VBATT_CFG_TAMPER_CHEN_CH0_EDGE Note: The default value is "VBATT_TAMPER_CHEN_RISING_EDGE"	Setting to Channel 0 Trigger Select Please choose from macro definition of the following. VBATT_TAMPER_CHEN_FALLING_EDGE = A falling edge of the input on the TAMPI0 pin. VBATT_TAMPER_CHEN_RISING_EDGE = A rising edge of the input on the TAMPI0 pin.
#define VBATT_CFG_TAMPER_CH1_DETECT_INT Note: The default value is "VBATT_TAMPER_DETECT_INT_DISABLE"	Setting to Tamper 1 Detection Interrupt Enable Please choose from macro definition of the following. VBATT_TAMPER_DETECT_INT_DISABLE = Tamper 1 detection interrupt is disabled. VBATT_TAMPER_DETECT_INT_ENABLE = Tamper 1 detection interrupt is enabled.
#define VBATT_CFG_TAMPER_CH1_ERASE Note: The default value is "VBATT_TAMPER_ERASE_DISABLE"	Setting to Tamper 1 Erase Enable Please choose from macro definition of the following. VBATT_TAMPER_ERASE_DISABLE = Backup registers are not erased in response to a tamper 1 event. VBATT_TAMPER_ERASE_ENABLE = Backup registers are erased in response to a tamper 1 event.
#define VBATT_CFG_TAMPER_TCE_CH1_SELECT Note: The default value is "VBATT_TAMPER_TCE_RTCIC_PIN"	Setting to Time Capture Event 1 Source Select Please choose from macro definition of the following. VBATT_TAMPER_TCE_RTCIC_PIN = Input signal from the RTCIC1 pin. VBATT_TAMPER_TCE_TAMPER_EVENT = Tamper 1 event.
#define VBATT_CFG_TAMPER_CHEN_CH1_NOISE_FILTER Note: The default value is "VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE"	Setting to Channel 1 Noise Filter Enable Please choose from macro definition of the following. VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE = Noise filter for the TAMPI1/RTCIC1 pin is disabled. VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE = Noise filter for the TAMPI1/RTCIC1 pin is enabled.
#define VBATT_CFG_TAMPER_CHEN_CH1_EDGE Note: The default value is "VBATT_TAMPER_CHEN_RISING_EDGE"	Setting to Channel 1 Trigger Select Please choose from macro definition of the following. VBATT_TAMPER_CHEN_FALLING_EDGE = A falling edge

Configuration options in r_vbatt_rx_config.h	
	of the input on the TAMPI1 pin. VBATT_TAMPER_CHEN_RISING_EDGE = A rising edge of the input on the TAMPI1 pin.
#define VBATT_CFG_TAMPER_CH2_DETECT_INT Note: The default value is "VBATT_TAMPER_DETECT_INT_DISABLE"	Setting to Tamper 2 Detection Interrupt Enable Please choose from macro definition of the following. VBATT_TAMPER_DETECT_INT_DISABLE = Tamper 2 detection interrupt is disabled. VBATT_TAMPER_DETECT_INT_ENABLE = Tamper 2 detection interrupt is enabled.
#define VBATT_CFG_TAMPER_CH2_ERASE Note: The default value is "VBATT_TAMPER_ERASE_DISABLE"	Setting to Tamper 2 Erase Enable Please choose from macro definition of the following. VBATT_TAMPER_ERASE_DISABLE = Backup registers are not erased in response to a tamper 2 event. VBATT_TAMPER_ERASE_ENABLE = Backup registers are erased in response to a tamper 2 event.
#define VBATT_CFG_TAMPER_TCE_CH2_SELECT Note: The default value is "VBATT_TAMPER_TCE_RTCIC_PIN"	Setting to Time Capture Event 2 Source Select Please choose from macro definition of the following. VBATT_TAMPER_TCE_RTCIC_PIN = Input signal from the RTCIC2 pin. VBATT_TAMPER_TCE_TAMPER_EVENT = Tamper 2 event.
#define VBATT_CFG_TAMPER_CHEN_CH2_NOISE_FILTER Note: The default value is "VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE"	Setting to Channel 2 Noise Filter Enable Please choose from macro definition of the following. VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE = Noise filter for the TAMPI2/RTCIC2 pin is disabled. VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE = Noise filter for the TAMPI2/RTCIC2 pin is enabled.
#define VBATT_CFG_TAMPER_CHEN_CH2_EDGE Note: The default value is "VBATT_TAMPER_CHEN_RISING_EDGE"	Setting to Channel 2 Trigger Select Please choose from macro definition of the following. VBATT_TAMPER_CHEN_FALLING_EDGE = A falling edge of the input on the TAMPI2 pin. VBATT_TAMPER_CHEN_RISING_EDGE = A rising edge of the input on the TAMPI2 pin.
The following definition is used for all target devices	
#define VBATT_CFG_INT_PRIORITY Note: The default value is "5"	For RX231, RX230, RX23W, the interrupt priority level can be selected when a maskable interrupt is used as the VBATT pin voltage drop detection interrupt. For RX671, the interrupt priority level of the tamper interrupt can be selected. The value selected by a value from 1 to 15 is set as the interrupt level. Note: This setting is only valid when VBATT_DTCT_ENABLE_INT_ENABLE is selected by VBATT_CFG_DETECT_FUNCTION. (For RX231, RX230, RX23W)

2.8 Code Size

The sizes of ROM, RAM, and maximum stack usage associated with this module are listed below.

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

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

Module Revision: r_vbatt_rx rev2.00

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00

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

GCC for Renesas RX 8.3.0.202004

(The option of "-std = gnu99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.20.1

(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
RX230	ROM	**550 bytes**	**493 bytes**	**1192 bytes**	**1128 bytes**	**995 bytes**	**827 bytes**
	RAM	**4 bytes**		**4 bytes**		**4 bytes**	
	STACK	**44 bytes**		**-**		**124 bytes**	
RX231	ROM	**550 bytes**	**493 bytes**	**1192 bytes**	**1128 bytes**	**995 bytes**	**827 bytes**
	RAM	**4 bytes**		**4 bytes**		**4 bytes**	
	STACK	**44 bytes**		**-**		**124 bytes**	
RX671	ROM	**789 bytes**	**692 bytes**	**2064 bytes**	**1944 bytes**	**1549 bytes**	**1185 bytes**
	RAM	**4 bytes**		**4 bytes**		**4 bytes**	
	STACK	**44 bytes**		**-**		**116 bytes**	

Note 1. The size includes BSP.

2.9 Arguments

This section presents the structures used as arguments to the API functions. These structures are included in the file r_vbatt_rx_if.h along with the API function prototype declarations.

```
/* Battery backup function data structure */
typedef volatile struct
{
    vbatt_callback_t    callbackfunc;  /* Callback function */
} vbatt_info_t;
```

```
/* Structure used to set up the VBATT pin voltage drop detection function again */
typedef volatile struct
{
#if defined(BSP_MCU_RX230) || defined(BSP_MCU_RX231) || defined(BSP_MCU_RX23W)
   uint8_t       rsv2;               /* reserve                                    */
   uint8_t       rsv1;               /* reserve                                    */
   uint8_t       vbatt_int_priority;     /* Interrupt priority level of
                                    VBATT Pin Voltage Drop Detection maskable interrupt  */
   union
   {
      uint8_t     byte;
      R_BSP_ATTRIB_STRUCT_BIT_ORDER_LEFT_4
      (
         uint8_t rsv:3,              /* reserve                                    */
         uint8_t lvd_level:2,          /* VBATT Pin Voltage Drop Detection Level          */
         uint8_t lvd_detect:2,         /* VBATT Pin Voltage Drop Detection Function       */
         uint8_t func:1              /* Battery Backup Function                    */
      ) bit;
   } vbatt_ctrl;
#endif

#if defined(BSP_MCU_RX671)

   uint8_t tamper_channel;              /* Tamper channel is enabled. */

   /* VBATT_TAMPER_INT_ENABLE
   VBATT_TAMPER_INT_DISABLE */
   uint8_t tamper_detection_interrupt;     /* TAMPCR
                                 0: Tamper n detection interrupt is disabled.
                                 1: Tamper n detection interrupt is enabled. */
   /* VBATT_TAMPER_ERASE_ENABLE
     VBATT_TAMPER_ERASE_DISABLE */
   uint8_t tamper_erase;                /* TAMPCR
                                 0: Backup registers are not erased in response to a tamper n event.
                                 1: Backup registers are erased in response to a tamper n event. */
   /* VBATT_TAMPER_TCE_RTCIC_PIN
     VBATT_TAMPER_TCE_TEMPER_EVENT */
   uint8_t time_capture_source;         /* TCECR
                                 0: Input signal from the RTCICn pin
                                 1: Tamper n event */
   /* VBATT_TAMPER_CHEN_INPUT_DISABLE
     VBATT_TAMPER_CHEN_INPUT_ENABLE */
   uint8_t channel_input;               /* TAMPICR1
                                 0: The RTCICn signal input is disabled.
                                 1: The RTCICn signal input is enabled. */
   /* VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE
     VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE */
   uint8_t channel_noise_filter;        /* TAMPICR2
                                 0: Noise filter for the RTCICn pin is disabled.
                                 1: Noise filter for the RTCICn pin is enabled. */
   /* VBATT_TAMPER_CHEN_FALLING_EDGE
     VBATT_TAMPER_CHEN_RISING_EDGE */
   uint8_t channel_trigger_select;      /* TAMPICR2
                                 0: A falling edge of the input on the RTCICn pin.
                                 1: A rising edge of the input on the RTCICn pin. */

   uint8_t tamper_int_priority;         /* interrupt priority; 1=low, 15=high */
```

```
  #endif /* defined(BSP_MCU_RX671) */
  } vbatt_ctrl_info_t;

  /* Battery Backup Function status information structure */
  typedef volatile struct
  {
  #if defined(BSP_MCU_RX230) || defined(BSP_MCU_RX231) || defined(BSP_MCU_RX23W)
     union
     {
        uint8_t     byte;
        R_BSP_ATTRIB_STRUCT_BIT_ORDER_LEFT_3
        (
           uint8_t rsv:6,              /* reserve */
           uint8_t vbatt_mon:1,        /* VBATT Pin Voltage Monitor Flag */
           uint8_t rsv1:1              /* reserve */
        ) bit;
     } vbatt_status;
  #endif
  #if defined(BSP_MCU_RX671)
     uint8_t tamper_channel;  /* Tamper channel need get status. */
     uint8_t tamper_detection_flag;  /* Tamper Detection Flag */
     uint8_t tamper_level_monitoring_flag;  /* Channel Level Monitoring Flag */
     bool  action_clear; /* Action clear status register */
  #endif
} vbatt_status_t;
```

2.10 Return Values

This section presents the return values from the API functions. This enumeration type is defined in the file r_vbatt_rx_if.h along with the API function prototype declarations.

```
typedef enum                        /* Return value used for calls to battery backup
                                       API functions */
{
    VBATT_SUCCESS,                  /* Processing completed without problem */
    VBATT_ERR_LOCK_FUNC,            /* The function was called multiple times */
                                       (Not implemented)
    VBATT_ERR_LVD0_SETTING,  /* Illegal voltage monitoring 0 settings in
                                       option function selection register 1 (OFS1) */
    VBATT_ERR_INVALID_ARG,   /* Invalid argument */
    VBATT_ERR_FUNC_INVALID,  /* R_VBATT_GetStatus() was called when VBATT pin
                                       voltage drop detection was invalid */
    VBATT_ERR_OTHER                 /* Other error */
} vbatt_return_t;
```

2.11 Callback Function

In this module, a callback function is called when the R_VBATT_Open() function is called or when an interrupt occurs. The callback function takes an argument whose set value determines whether a battery backup voltage drop was detected, or whether it was called from an interrupt handler when the VBATT pin voltage fell.

Table 2.2 lists the constant definitions (enum vbatt_cb_evt_t) for the argument passed to the callback function.

For callback function setup, the address of the callback function to be registered should be stored in “callbackfunc” member in the structure described in section 2.9, Arguments.

Table 2.2 Definitions of Constants Passed as an Argument to the Callback Function (enum vbatt_cb_evt_t)

Defined Constant	Conditions for Passing as an Argument
VBATT_NOT_DROP_VOLTAGE	When R_VBATT_Open() was called in the state where a battery backup supply voltage drop was not detected (the VBATTSR register VBATRLVDETF bit was 0)
VBATT_DROP_VOLTAGE	When R_VBATT_Open() was called in the state where a battery backup supply voltage drop was detected (the VBATTSR register VBATRLVDETF bit was 1)
VBATT_MASKABLE_INTERRUPT	When a maskable interrupt due to a voltage drop on the VBATT pin occurred
VBATT_NON_MASKABLE_INTERRUPT	When a nonmaskable interrupt due to a voltage drop on the VBATT pin occurred
VBATT_TAMPER_CH0_INTERRUPT VBATT_TAMPER_CH1_INTERRUPT VBATT_TAMPER_CH2_INTERRUPT	When a maskable interrupt due to tamper detector has detected an intrusion to the system

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the 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’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the 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 “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in 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’s 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 “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” 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

R_VBATT_Open()

In device RX230, RX231, RX23W this function sets up the VBATT pin voltage drop detection function and determines whether or not the battery backup supply voltage has fallen.
This function is called once and only once after a reset start. The processing performed when a drop in the battery backup supply voltage is detected and when no drop is detected are performed by a callback function called by this function.

In device RX671 this function sets up the TAMPD detection function and determines whether or not the voltage for the backup domain has fallen below V_{PDR}.
This function is called once and only once after a reset start. The processing performed when a voltage for the backup domain has fallen below V_{PDR} is detected and when no drop is detected are performed by a callback function called by this function.

Format

```
vbatt_return_t    R_VBATT_Open (
   vbatt_info_t *       p_vbatt_info
)
```

Parameters

p_vbatt_info

Pointer to a data structure for the battery backup information.
The following members are used by this function. See section 2.9, Arguments, for details on this structure.

```
vbatt_callback_t      callbackfunc;          /* Address of the callback function */
```

Return Values

```
VBATT_SUCCESS                 /* Processing completed without problem */
VBATT_ERR_LVD0_SETTING        /* Illegal voltage monitoring 0 settings in
                                    option function selection register 1 (OFS1) */
VBATT_ERR_INVALID_ARG         /* Invalid argument */
```

Properties

A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description

In device RX230, RX231, RX23W this function sets the enabled or disabled state of the VBATT pin voltage drop detection function, the detection level, and the interrupts according to settings in the configuration options. After that, it determines whether or not the battery backup supply voltage has dropped and calls the callback function.

In device RX671 this function sets the enabled or disabled state of the TAMPD function, the detection level, time capture event, channel input, channel noise filter, and the interrupts according to settings in the configuration options. After that, it determines whether or not the voltage for the backup domain has fallen below V_{PDR}. and calls the callback function
When calling the callback function:

- If a battery backup supply voltage drop has not been detected:
  A pointer to a variable that has been set to VBATT_NOT_DROP_VOLTAGE is passed as an argument.
- If a battery backup supply voltage drop has been detected:
  A pointer to a variable that has been set to VBATT_DROP_VOLTAGE is passed as an argument.

Example

```
#include "r_vbatt_rx_if.h"

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);

void main(void)
```

```
{
	vbatt_return_t		ret;
	vbatt_info_t		vbatt_info;

	vbatt_info.callbackfunc = vbatt_callback;

	ret = R_VBATT_Open(&vbatt_info);
	if (VBATT_SUCCESS != ret)
	{
		/* Please do the processing at the time of the error */
	}

	while(1);

}

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
	switch(*vbatt_cb_event)
	{
		/* Battery backup power voltage drop not detected */
		case VBATT_NOT_DROP_VOLTAGE:

			/* Please set RAM again as needed */

		break;

		/* Battery backup power voltage drop detected */
		case VBATT_DROP_VOLTAGE:

			/* Please initialize the Realtime Clock */

		break;

		/* VBATT voltage drop detected interrupt */
		case VBATT_MASKABLE_INTERRUPT:
		case VBATT_NON_MASKABLE_INTERRUPT:

			/* Please process warning indication, backup, etc. */

		break;

		/* Tamper detection interrupt */
		case VBATT_TAMPER_CH0_INTERRUPT:

			/* Please process warning indication, etc. */

		break;

		/* Tamper detection interrupt */
		case VBATT_TAMPER_CH1_INTERRUPT:

			/* Please process warning indication, etc. */

		break;


		/* Tamper detection interrupt */
		case VBATT_TAMPER_CH2_INTERRUPT:

			/* Please process warning indication, etc. */
```

```
                break;

                default:

                        /* Do nothing */

                break;
        }
}
```

Special Notes

If the tamper was detected during the battery backup mode, executing this function clears the tamper detection status.

Therefore, if you want to check the tamper detection in battery backup mode, you can check it by using the R_VBATT_GetStatus function.

R_VBATT_Control()

In device RX230, RX231, RX23W this function sets the enabled or disabled state of the battery backup function and sets up the VBATT pin voltage drop detection function. This function is used when changing the content of these settings from those made with the R_VBATT_Open() function.

In device RX671 this function sets the enabled or disabled state of the TAMPD function and sets up the TAMPD function. This function is used when changing the content of these settings from those made with the R_VBATT_Open() function.

Format

```
vbatt_return_t          R_VBATT_Control (
    vbatt_ctrl_info_t *       p_vbatt_ctrl_info
)
```

Parameters

p_vbatt_ctrl_info

Pointer to the data structure used by the VBATT pin voltage drop detection function.

The following members are used by this function. See section 2.9, Arguments, for details on this structure.

```
typedef volatile struct
{
#if defined(BSP_MCU_RX230) || defined(BSP_MCU_RX231) || defined(BSP_MCU_RX23W)
uint8_t rsv2; /* Reserved area */
uint8_t rsv1; /* Reserved area */
uint8_t vbatt_int_priority; /* Interrupt priority level for
the VBATT pin voltage drop detection
interrupt (maskable interrupt) */
union
{
uint8_t byte;
struct
{
uint8_t rsv:3; /* Reserved area */
uint8_t lvd_level:2; /* VBATT pin voltage drop detection level */
uint8_t lvd_detect:2; /* VBATT pin voltage drop detection function */
uint8_t func:1; /* Enabled/disabled state of the battery backup
function */
} bit;
} vbatt_ctrl;
#endif
#if defined(BSP_MCU_RX671)

   uint8_t tamper_channel;              /* Tamper channel is enabled. */

   /* VBATT_TAMPER_INT_ENABLE
   VBATT_TAMPER_INT_DISABLE */
   uint8_t tamper_detection_interrupt;    /* TAMPCR
                              0: Tamper n detection interrupt is disabled.
                              1: Tamper n detection interrupt is enabled. */
   /* VBATT_TAMPER_ERASE_ENABLE
      VBATT_TAMPER_ERASE_DISABLE */
   uint8_t tamper_erase;               /* TAMPCR
                              0: Backup registers are not erased in response to a tamper n event.
                              1: Backup registers are erased in response to a tamper n event. */
   /* VBATT_TAMPER_TCE_RTCIC_PIN
      VBATT_TAMPER_TCE_TEMPER_EVENT */
   uint8_t time_capture_source;        /* TCECR
                              0: Input signal from the RTCICn pin
                              1: Tamper n event */
   /* VBATT_TAMPER_CHEN_INPUT_DISABLE
      VBATT_TAMPER_CHEN_INPUT_ENABLE */
   uint8_t channel_input;              /* TAMPICR1
                              0: The RTCICn signal input is disabled.
                              1: The RTCICn signal input is enabled. */
   /* VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE
      VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE */
   uint8_t channel_noise_filter;       /* TAMPICR2
                              0: Noise filter for the RTCICn pin is disabled.
                              1: Noise filter for the RTCICn pin is enabled. */
   /* VBATT_TAMPER_CHEN_FALLING_EDGE
      VBATT_TAMPER_CHEN_RISING_EDGE */
   uint8_t channel_trigger_select;     /* TAMPICR2
                              0: A falling edge of the input on the RTCICn pin.
                              1: A rising edge of the input on the RTCICn pin. */
   uint8_t tamper_int_priority;        /* interrupt priority; 1=low, 15=high */
#endif /* defined(BSP_MCU_RX671) */
} vbatt_ctrl_info_t;
```

Return Values

VBATT_SUCCESS	*/* Processing completed without problem */*
VBATT_ERR_INVALID_ARG	*/* Invalid argument */*

Properties

A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description

In device RX230, RX231, RX23W this function sets the enabled or disabled state of the battery backup function, the enabled or disabled state of the VBATT pin voltage drop detection function, the detection level, and the interrupts according to the settings of the arguments.

In device RX671 this function sets up the TAMPER detection function and the interrupts according to the settings of the arguments.

Example

Example for device RX230,RX231, RX23W

```
#include "r_vbatt_rx_if.h"

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);

void main(void)
{
        vbatt_return_t          ret;
        vbatt_info_t            vbatt_info;
        vbatt_ctrl_info_t       vbatt_ctrl_info;

        vbatt_info.callbackfunc = vbatt_callback;

        ret = R_VBATT_Open(&vbatt_info);
        if (VBATT_SUCCESS != ret)
        {
                /* Please do the processing at the time of the error */
        }

        /* Battery backup function enable */
        vbatt_ctrl_info.vbatt_ctrl.bit.func = 1;
        /* VBATT drop detect function enable and maskable interrupt enable */
        vbatt_ctrl_info.vbatt_ctrl.bit.lvd_detect = VBATT_DTCT_ENABLE_INT_ENABLE;
        /* VBATT drop detect level is 2.00V */
        vbatt_ctrl_info.vbatt_ctrl.bit.lvd_level = VBATT_DTCT_LEVEL_2_00_V;
        /* interrupt priority level is 7*/
        vbatt_ctrl_info.vbatt_int_priority = 7;

        ret = R_VBATT_Control(&vbatt_ctrl_info);
        if (VBATT_SUCCESS != ret)
        {
                        /* Please do the processing at the time of the error */
        }

        while(1);
}
void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
        /* Please process base on vbatt_cb_evt_t */
}
```

Example for device RX671

```
#include "r_vbatt_rx_if.h"

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);

void main(void)
{
        vbatt_return_t    ret;
        vbatt_info_t              vbatt_info;
        vbatt_ctrl_info_t         vbatt_ctrl_info;

    vbatt_info.callbackfunc = vbatt_callback;

        ret = R_VBATT_Open(&vbatt_info);
        if (VBATT_SUCCESS != ret)
        {
                /* Please do the processing at the time of the error */
        }

        vbatt_ctrl_info_t vbatt_ctrl_info;
        /* Tamper n channel is enabled. */
        vbatt_ctrl_info.tamper_channel = VBATT_TAMPER_CH0;
        /* Tamper detection interrupt is enabled. */
        vbatt_ctrl_info.tamper_detection_interrupt = VBATT_TAMPER_DETECT_INT_ENABLE;
        /* Backup registers are erased in response to a tamper n event. */
        vbatt_ctrl_info.tamper_erase = VBATT_TAMPER_ERASE_ENABLE;
        /* Input signal from the RTCICn pin */
        vbatt_ctrl_info.time_capture_source = VBATT_TAMPER_TCE_TAMPER_EVENT;
        /* The RTCICn signal input is enabled. */
        vbatt_ctrl_info.channel_input = VBATT_TAMPER_CHEN_INPUT_ENABLE;
        /* Noise filter for the RTCICn pin is enabled. */
        vbatt_ctrl_info.channel_noise_filter = VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE;
        /* A falling edge of the input on the RTCICn pin. */
        vbatt_ctrl_info.channel_trigger_select = VBATT_TAMPER_CHEN_FALLING_EDGE;
        /* Interrupt priority; 1=low, 15=high. */
        vbatt_ctrl_info.tamper_int_priority = 5;
        ret = R_VBATT_Control(&vbatt_ctrl_info);
        if (VBATT_SUCCESS != ret)
        {
                /* Please do the processing at the time of the error */
        }
        while(1);
}
void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
        /* Please process base on vbatt_cb_evt_t */
}
```

Special Notes

1. The table below lists and describes the range of values the arguments may be set to and the meanings of those arguments.

Structure (vbatt_ctrl_info_t) in device RX230, RX231, RX23W			
Member	**Bit**	**Allowable Range**	**Meaning**
vbatt_ctrl	func	0 to 1	The enabled or disabled state of the battery backup function can be changed. 0: The battery backup function is disabled. 1: The battery backup function is enabled.

	lvd_detect	Selected from macro definitions for the corresponding meaning	Selects whether or not the VBATT pin voltage drop detection function is used. This also selects the interrupt generated when a voltage drop is detected. For VBATT_DTCT_DISABLE: The VBATT pin voltage drop detection function is set to invalid and the interrupt is disabled. For VBATT_DTCT_ENABLE_INT_DISABLE: The VBATT pin voltage drop detection function is enabled and the interrupt is disabled. For VBATT_DTCT_ENABLE_NMI_ENABLE: The VBATT pin voltage drop detection function is enabled and the nonmaskable interrupt is enabled as the interrupt. For VBATT_DTCT_ENABLE_INT_ENABLE: The VBATT pin voltage drop detection function is enabled and the maskable interrupt is enabled as the interrupt.
	lvd_level	Selected from macro definitions for the corresponding meaning	The VBATT pin voltage drop detection level can be selected. For VBATT_DTCT_LEVEL_2_20_V, the detection level is set to 2.20 V. For VBATT_DTCT_LEVEL_2_00_V, the detection level is set to 2.00 V.
vbatt_int_priority	–	1 to 15	The interrupt priority level can be selected when a maskable interrupt is used as the VBATT pin voltage drop detection interrupt. The value selected by a value from 1 to 15 is set as the interrupt level. Note: This setting is only valid when VBATT_DTCT_ENABLE_INT_ENABLE is selected by lvd_detect.
Structure (vbatt_ctrl_info_t) in device RX671			
Member	**Bit**	**Allowable Range**	**Meaning**
tamper_channel	–	Selected from macro definitions for the corresponding meaning	The channel is selected to control. For VBATT_TAMPER_CH0 channel 0 is selected. For VBATT_TAMPER_CH1 channel 1 is selected. For VBATT_TAMPER_CH2 channel 2 is selected.
tamper_detection_interrupt	–	Selected from macro definitions for the corresponding meaning	Selects enables or disables the tamper_channel detection interrupt. For VBATT_TAMPER_DETECT_INT_ENABLE tamper_channel detection interrupt is enabled. For VBATT_TAMPER_DETECT_INT_DISABLE tamper_channel detection interrupt is disabled.
tamper_erase	–	Selected from macro definitions for the corresponding meaning	Selects whether the backup registers should be or should not be erased in response to a tamper_channel event. For VBATT_TAMPER_ERASE_ENABLE: Backup registers are erased in response to a tamper_channel event. For VBATT_TAMPER_ERASE_DISABLE: Backup registers are not erased in response to a tamper_channel event.
time_capture_source	–	Selected from macro definitions for the corresponding meaning	Selects the source of time capture event n for the RTC For VBATT_TAMPER_TCE_RTCIC_PIN: Input signal from the RTCICn pin. For VBATT_TAMPER_TCE_TAMPER_EVENT: tamper_channel event.
channel_input	–	Selected from macro definitions for the corresponding meaning	Selects enables or disables the input from the RTCICn pin. For VBATT_TAMPER_CHEN_INPUT_DISABLE: The tamper_channel signal input is disabled. For VBATT_TAMPER_CHEN_INPUT_ENABLE: The tamper_channel signal input is enabled.
channel_noise_filter	–	Selected from macro definitions for the corresponding meaning	Selects enables or disables the noise filter for the RTCICn pin. For VBATT_TAMPER_CHEN_NOISE_FILTER_DISABLE:

			Noise filter for the tamper_channel pin is disabled. For VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE: Noise filter for the tamper_channel pin is enabled.
channel_trigger_select	–	Selected from macro definitions for the corresponding meaning	Select the effective edge for the input signal from the RTCICn pin for use as a trigger of tamper event detection. For VBATT_TAMPER_CHEN_FALLING_EDGE: A falling edge of the input on the tamper_channel pin. For VBATT_TAMPER_CHEN_RISING_EDGE: A rising edge of the input on the tamper_channel pin.
tamper_int_priority	–	1 to 15	The interrupt priority level can be selected when tamper_channel detection interrupt is enabled. The value selected by a value from 1 to 15 is set as the interrupt level. Note: This setting is only valid when VBATT_TAMPER_DETECT_INT_ENABLE is selected by tamper_detection_interrupt.

2. In device RX671, when this function is executed, the tamper detection status is cleared.

R_VBATT_GetStatus()

This function acquires the status for the battery backup function. This function is used to check the status of the battery backup function..

Format

```
vbatt_return_t          R_VBATT_GetStatus (
    vbatt_status_t *                p_vbatt_status
)
```

Parameters

p_vbatt_status

Pointer to a variable to hold the battery backup function status.

The following members are used by this function. See section 2.9, Arguments, for details on this structure.

```
typedef volatile struct
{
#if defined(BSP_MCU_RX230) || defined(BSP_MCU_RX231) || defined(BSP_MCU_RX23W)
   union
   {
      uint8_t     byte;
      R_BSP_ATTRIB_STRUCT_BIT_ORDER_LEFT_3
      (
         uint8_t rsv:6,              /* reserve                                  */
         uint8_t vbatt_mon:1,        /* VBATT Pin Voltage Monitor Flag           */
         uint8_t rsv1:1              /* reserve     */
      ) bit;
   } vbatt_status;
#endif
#if defined(BSP_MCU_RX671)
   uint8_t tamper_channel;  /* Tamper channel need get status. */
   uint8_t tamper_detection_flag;   /* Tamper Detection Flag */
   uint8_t tamper_level_monitoring_flag;   /* Channel Level Monitoring Flag */
   bool  action_clear; /* Action clear status register */
#endif
  } vbatt_status_t;
```

Return Values

```
VBATT_SUCCESS               /* Processing completed without problem */
VBATT_ERR_INVALID_ARG           /* Invalid argument */
VBATT_ERR_FUNC_INVALID      /* R_VBATT_GetStatus() was called when VBATT pin
                               voltage drop detection was invalid */
```

Properties

A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description

In device RX230, RX231, RX23W this function reads out the VBATT status register (VBATTSR) to acquire the status of the battery backup function. It then stores that information at the address passed as an argument.

In device RX671 this function reads out the TAMPD status register to acquire the status of the tamper detector function. It then stores that information at the address passed as an argument.

Example

Example for device RX230, RX231, RX23W

```
#include "r_vbatt_rx_if.h"

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);

void main(void)
{
	vbatt_return_t		ret;
	vbatt_info_t		vbatt_info;
	vbatt_status_t		vbatt_status;
	vbatt_ctrl_info_t	vbatt_ctrl_info;

	vbatt_info.callbackfunc = vbatt_callback;

	ret = R_VBATT_Open(&vbatt_info);
	if (VBATT_SUCCESS != ret)
	{
		/* Please do the processing at the time of the error */
	}

	/* VBATT drop detect function enable */
	vbatt_ctrl_info.vbatt_ctrl.bit.func = 1;
	vbatt_ctrl_info.vbatt_ctrl.bit.lvd_detect = VBATT_DTCT_ENABLE_INT_DISABLE;
	vbatt_ctrl_info.vbatt_ctrl.bit.lvd_level = VBATT_DTCT_LEVEL_2_20_V;
	vbatt_ctrl_info.vbatt_int_priority = 5;
	ret = R_VBATT_Control(&vbatt_ctrl_info);
	if (VBATT_SUCCESS != ret)
	{
			/* Please do the processing at the time of the error */
	}

	/* gets the state of the battery backup function */
	ret = R_VBATT_GetStatus(&vbatt_status);
	if (VBATT_SUCCESS != ret)
	{
			/* Please do the processing at the time of the error */
	}

	while(1);
}
void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
	/* Please process base on vbatt_cb_evt_t */
}
```

Example for device RX671

```
#include "r_vbatt_rx_if.h"

void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);

void main(void)
{
	vbatt_return_t    ret;
	vbatt_info_t      vbatt_info;
	vbatt_status_t    vbatt_status;
	vbatt_ctrl_info_t vbatt_ctrl_info;

	vbatt_info.callbackfunc = vbatt_callback;

	ret = R_VBATT_Open(&vbatt_info);
	if (VBATT_SUCCESS != ret)
	{
		/* Please do the processing at the time of the error */
	}

	/* Tamper n channel is enabled. */
	vbatt_ctrl_info.tamper_channel = VBATT_TAMPER_CH0;
	/* Tamper detection interrupt is enabled. */
	vbatt_ctrl_info.tamper_detection_interrupt = VBATT_TAMPER_DETECT_INT_ENABLE;
	/* Backup registers are erased in response to a tamper n event. */
	vbatt_ctrl_info.tamper_erase = VBATT_TAMPER_ERASE_ENABLE;
	/* Input signal from the RTCICn pin */
	vbatt_ctrl_info.time_capture_source = VBATT_TAMPER_TCE_RTCIC_PIN;
	/* The RTCICn signal input is enabled. */
	vbatt_ctrl_info.channel_input = VBATT_TAMPER_CHEN_INPUT_ENABLE;
	/* Noise filter for the RTCICn pin is enabled. */
	vbatt_ctrl_info.channel_noise_filter = VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE;
	/* A falling edge of the input on the RTCICn pin. */
	vbatt_ctrl_info.channel_trigger_select = VBATT_TAMPER_CHEN_FALLING_EDGE;
	/* Interrupt priority; 1=low, 15=high. */
	vbatt_ctrl_info.tamper_int_priority = 5;
	ret = R_VBATT_Control(&vbatt_ctrl_info);
	if (VBATT_SUCCESS != ret)
	{
		/* Please do the processing at the time of the error */
	}
	/* gets the state of the battery backup function */
	vbatt_status.tamper_channel = VBATT_TAMPER_CH0;
	vbatt_status.action_clear = true;
	ret = R_VBATT_GetStatus(&vbatt_status);
	if (VBATT_SUCCESS != ret)
	{
		/* Please do the processing at the time of the error */
	}
	 while(1);
}
void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
	/* Please process base on vbatt_cb_evt_t */
}
```

Special Notes

1. The table below shows the layout of the status flags.

Structure (vbatt_ctrl_info_t) in device RX230, RX231, RX23W			
Bit	**b7 to b2**	**b1**	**b0**
Bit Name	Reserved area	VBATT pin voltage monitor flag	Reserved area
Symbol	rsv	vbatt_mon	rsv1
Function	Undefined	0: VBATT < Vdetvbt 1: VBATT ≥ Vdetvbt or the VBATT detection function was disabled.	Undefined
Structure (vbatt_ctrl_info_t) in device RX671			
Member	**Allowable Range**		**Meaning**
tamper_channel	VBATT_TAMPER_CH0 VBATT_TAMPER_CH1 VBATT_TAMPER_CH2		Tamper channel need get status.
tamper_detection_flag	0 to 1		0: Tamper event has not been detected. 1: Tamper event has been detected.
tamper_level_monitoring_flag	0 to 1		0: The low level is being input on the RTCIC pin. 1: The high level is being input on the RTCIC pin.
action_clear	0 to 1		0: Clear status register is disabled. 1: Clear status register is enabled.

2. In device RX671, when transitioning to battery backup mode with tamper detection enabled, you can use this function to check whether tamper detection has been detected during battery backup mode before executing the R_VBATT_Open function. Note that executing the R_VBATT_Open function clears the tamper detection status.

R_VBATT_ReadBackupData()

This function is only available on device RX671.

This function read data from Backup Registers.

Format

```
vbatt_return_t R_VBATT_ReadBackupData(
        uint8_t index,
        uint8_t * p_data
)
```

Parameters

uint8_t index,

Index of Backup register need to read.
Range: 0 to 127

*uint8_t * p_data,*

Pointer to a variable to store data read from Backup Registers.

Return Values

VBATT_SUCCESS /* Processing completed without problem */
VBATT_ERR_INVALID_ARG /* Invalid argument */

Properties

A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description

This function read data from Backup Register.

Example

```
#include "r_vbatt_rx_if.h"
void main(void)
{
   vbatt_return_t    ret;
   uint8_t index = 0;
   /* Variable store content read from Backup Registers
   uint8_t backup_register;
   ret = R_VBATT_ReadBackupData(index, &backup_register);
   while(ret != VBATT_SUCCESS);

}
```

Special Notes

None

R_VBATT_WriteBackupData()

This function is only available on device RX671.
This function write data to Backup Registers.

Format

```
vbatt_return_t R_VBATT_WriteBackupData(
        uint8_t index,
        uint8_t * p_data
)
```

Parameters

uint8_t index,
Index of Backup register need to write.
Range: 0 to 127
*uint8_t * p_data,*
Pointer to a variable to store data write to Backup Registers.

Return Values

VBATT_SUCCESS /* Processing completed without problem */
VBATT_ERR_INVALID_ARG /* Invalid argument */

Properties

A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description

This function write data to Backup Register.

Example

```
#include "r_vbatt_rx_if.h"
void main(void)
{
   vbatt_return_t    ret;
   uint8_t index = 0;
   /* Variable store content write to Backup Registers
   uint8_t backup_register = 0x5;
   ret = R_VBATT_WriteBackupData(index, &backup_register);
   while(ret != VBATT_SUCCESS);

}
```

Special Notes

None

R_VBATT_GetVersion()

This function returns the version number of the API.

Format
uint32_t R_VBATT_GetVersion(void)

Parameters
None

Return Values
Version number

Properties
A prototype declaration for this function appears in r_vbatt_rx_if.h.

Description
This function returns the version number of this API.

Example
```
uint32_t                version;

version = R_VBATT_GetVersion();
```

Special Notes
None

4. Appendices

4.1 Confirmed Operation Environment

This section describes confirmed operation environment for the battery backup function FIT module.

Table 4.1 Confirmed Operation Environment (Rev. 2.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2022-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202202 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev. 2.20
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxx)

Table 4.2 Confirmed Operation Environment (Rev. 2.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev. 2.10
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

Table 4.3 Confirmed Operation Environment (Rev. 2.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev. 2.00
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

Table 4.4 Confirmed Operation Environment (Rev. 1.04)

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 V3.01.00 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. 1.04
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx)

Table 4.5 Confirmed Operation Environment (Rev. 1.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.10.1
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 4.8.4.2018.01 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 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev. 1.03
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231S900BE)

4.2 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:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. 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_vbatt_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got the error: Parameter error in configures file

A: The setting in the file "r_vbatt_rx_config.h" may be wrong. Check the file "r_vbatt_rx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7 Configuration Overview for details.

4.3 Sample Code

4.3.1 Example of Use Combined with the RTC FIT Module

(1) **Example for device RX230, RX231, RX23W**

This sample code shows the use of the battery backup function FIT module in combination with the RTC FIT module, which is used for real-time clock settings.

The configuration options are set as follows.

- The BSP FIT module BSP_CFG_OFS1_REG_VALUE is set to 0xFFFFFFFA.
- The BSP FIT module BSP_CFG_RTC_ENABLE is set to 1
- The BSP FIT module BSP_CFG_SOSC_DRV_CAP is set to 0 or 1

- The default values are used for the RTC FIT module and the VBATT FIT module settings.

This sample code operates in the sequence (1) to (3) shown below.

(1) The R_VBATT_Open() function is called.
(The callback function is called when the R_VBATT_Open() function is called.)
(2) The callback function for the battery backup function sets up the RTC according to whether or not a battery backup supply voltage drop is detected. Whether or not there is a battery backup supply voltage drop is recognized from the arguments to the callback function.
(2-A) If the callback function argument is VBATT_NOT_DROP_VOLTAGE, the RTC FIT module is set up again using the R_RTC_Open() and R_RTC_Read() functions.
(2-B) If the callback function argument is VBATT_DROP_VOLTAGE, the R_RTC_Open() function is used to initialize the RTC.
(3) In the RTC periodic interrupt callback function, the R_RTC_Read() function is used to read out the current time. The read out time is displayed on the debugging console.

```
#include <stdio.h>                    /* This is required because printf() is used for debugging display. */
#include "r_rtc_rx_if.h"
#include "r_vbatt_rx_if.h"

static void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);
static void rtc_callback(void *event);

static tm_t rtc_curr_time;  /* Data structure used to store the current time from the RTC. */

void main(void)
{
   vbatt_return_t     ret;              /* For confirming the return value from the API function. */
   vbatt_info_t       vbatt_info;       /* Data structure for the battery backup function */

       SYSTEM.RSTSR1.BIT.CWSF = 1;          /* set Cold/Warm Start Determination Flag. */

   vbatt_info.callbackfunc = vbatt_callback;   /* Callback function setup */

   /* VBATT pin voltage detection function setup and battery backup supply voltage drop determination. */
   ret = R_VBATT_Open(&vbatt_info);
   if (VBATT_SUCCESS != ret)
   {
      while(1);
   }

   while(1);
}


/* Battery backup function callback function */
static void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
   rtc_err_t   ret;                  /* For confirming the return value from the API function. */
   rtc_init_t  rtc_info;             /* RTC data structure */
```

Executing this function results in the set up callback function (vbatt_callback()) being called.

Figure 4.1 Usage Example in Combination with the RTC FIT Module (1/3)

```
    /* Discriminate based on the argument */
    switch(*vbatt_cb_event)
    {
        /* If no battery backup supply voltage drop is detected. */
        case VBATT_NOT_DROP_VOLTAGE:
            /* Set up the RTC data structure again */
            rtc_info.output_freq = RTC_OUTPUT_OFF;       /* Stop RTCOUT output */
            rtc_info.periodic_freq = RTC_PERIODIC_1_HZ;  /* RTC periodic interrupt generation period: 1 second */
            rtc_info.periodic_priority = 8;              /* Interrupt priority level */
            rtc_info.set_time = false;                   /* Do not update the RTC clock counter I/O register. */
            rtc_info.p_callback = rtc_callback;          /* Callback function setup */

            /* RTC re-setup */
            ret = R_RTC_Open(&rtc_info, &rtc_curr_time);
            if (RTC_SUCCESS != ret)
            {
                while(1);
            }

            /* Read out the current time information from the RTC clock counter I/O register
               and store it in the data structure. */
            ret = R_RTC_Read(&rtc_curr_time, NULL);
            if (RTC_SUCCESS != ret)
            {
                while(1);
            }

        break;

        /*  When a battery backup supply voltage drop was detected */
        case VBATT_DROP_VOLTAGE:
            /* RTC data structure setup */
            rtc_info.output_freq = RTC_OUTPUT_OFF;       /* Stop RTCOUT output */
            rtc_info.periodic_freq = RTC_PERIODIC_1_HZ;  /* RTC periodic interrupt generation period: 1 second */
            rtc_info.periodic_priority = 8;              /* Interrupt priority level */
            rtc_info.set_time = true;                    /* Update the RTC clock counter I/O register. */
            rtc_info.p_callback = rtc_callback;          /* Callback function setup */

            /* Set the current time information structure time setting to "2015-06-30 12:34:56" */
            rtc_curr_time.tm_sec   = 56;                 /* Seconds (0 - 59) */
            rtc_curr_time.tm_min   = 34;                 /* Minutes (0 - 59) */
            rtc_curr_time.tm_hour  = 12;                 /* Hours (0 - 23) */
            rtc_curr_time.tm_mday  = 30;                 /* Day (1 - 31) */
            rtc_curr_time.tm_mon   = 6;                  /* Month (0 - 11, 0 = January) */
            rtc_curr_time.tm_year  = 115;                /* Year (referenced to 1900) */
            rtc_curr_time.tm_wday  = 0;                  /* Day of week (0 - 6, 0 = Sunday) */
            rtc_curr_time.tm_yday  = 0;                  /* Day in year (0 - 365) */
            rtc_curr_time.tm_isdst = 0;                  /* Daylight saving time in effect (> 0),
                                                            not in effect (= 0). */

            /* RTC initialization */
            ret = R_RTC_Open(&rtc_info, &rtc_curr_time);
            if (RTC_SUCCESS != ret)
            {
                while(1);
            }

        break;

        /* Interrupt due to VBATT pin voltage drop */
        case VBATT_MASKABLE_INTERRUPT:
        case VBATT_NON_MASKABLE_INTERRUPT:
            /* Unused in this sample code */
        break;

        default:
            /* No processing */
        break;
    }
}
```

Although the RTC I/O register is saved, RAM and other registers are reset. Therefore, the RTC FIT module is setup again.

Since the RTC I/O register is saved, the current time information is read from the RTC I/O register by the R_RTC_Read() function and stored again in the data structure.

If a battery backup supply voltage drop was detected, the microcontroller is in a state where RTC operation is not guaranteed. Therefore the RTC is reinitialized.

Figure 4.2 Usage Example in Combination with the RTC FIT Module (2/3)

```
/* RTC callback function */
static void rtc_callback(void *event)
{
   rtc_err_t ret;                       /* For confirming the return value from the API function. */

   /* For a periodic interrupt */
   if (*(rtc_cb_evt_t *)event == RTC_EVT_PERIODIC)
   {
      /* For a periodic interrupt */
      /* Read out the current time information from the RTC clock counter I/O register
         and store it in the data structure. */
      ret = R_RTC_Read(&rtc_curr_time, NULL);
      if (RTC_SUCCESS != ret)
      {
         while(1);
      }

      /* Display on the debugging console. */
      printf("%d/%d/%d %02d:%02d:%02d\n",   rtc_curr_time.tm_year + 1900,
                              rtc_curr_time.tm_mon,
                              rtc_curr_time.tm_mday,
                              rtc_curr_time.tm_hour,
                              rtc_curr_time.tm_min,
                              rtc_curr_time.tm_sec);
   }
}
```

Each time a period interrupt occurs (at 1-second intervals) the current time is read out.

Figure 4.3 Usage Example in Combination with the RTC FIT Module (3/3)

(2) Example for device RX671

This sample code shows the use of the battery backup function FIT module in combination with the RTC FIT module, which is used for real-time clock settings also shows how to use time capture and tamper function together.

The configuration options are set as follows.

- The BSP FIT module BSP_CFG_OFS1_REG_VALUE is set to 0xFFFFFFFA.
- The BSP FIT module BSP_CFG_RTC_ENABLE is set to 1
- The BSP FIT module BSP_CFG_SOSC_DRV_CAP is set to 0 or 1
- The VBATT FIT module VBATT_CFG_TAMPER_CH0 is set to VBATT_TAMPER_ENABLE.
- The VBATT FIT module VBATT_CFG_TAMPER_CH0_DETECT_INT is set to VBATT_TAMPER_DETECT_INT_ENABLE.
- The VBATT FIT module VBATT_CFG_TAMPER_CH0_ERASE is set to VBATT_TAMPER_ERASE_DISABLE.
- The VBATT FIT module VBATT_CFG_TAMPER_TCE_CH0_SELECT is set to VBATT_TAMPER_TCE_TAMPER_EVENT.
- The VBATT FIT module VBATT_CFG_TAMPER_CHEN_CH0_NOISE_FILTER is set to VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE.
- The VBATT FIT module VBATT_CFG_TAMPER_CHEN_CH0_EDGE is set to VBATT_TAMPER_CHEN_RISING_EDGE.
- The default values are used for others in the VBATT FIT module settings.
- The default values are used for the RTC FIT module settings.

This sample code operates in the sequence (1) to (3) shown below.

(1) The R_VBATT_Open() function is called.
(The callback function is called when the R_VBATT_Open() function is called.)
(2) The callback function for the battery backup function sets up the RTC according to whether or not a battery backup supply voltage drop is detected. Whether or not there is a battery backup supply voltage drop is recognized from the arguments to the callback function.
(2-A) If the callback function argument is VBATT_NOT_DROP_VOLTAGE, the RTC FIT module is set up again using the R_RTC_Open(), R_RTC_Read() and R_RTC_Control() functions .
(2-B) If the callback function argument is VBATT_DROP_VOLTAGE, the R_RTC_Open() and R_RTC_Control() function is used to initialize the RTC.
(2-C) If the callback function argument is VBATT_TAMPER_CH0_INTERRUPT, perform the necessary processing.

(3) In the RTC periodic interrupt callback function, the R_RTC_Read() function is used to read out the current time. The read out time is displayed on the debugging console.

```
#include <stdio.h>                      /* This is required because printf() is used for debugging display. */
#include "r_rtc_rx_if.h"
#include "r_vbatt_rx_if.h"

static void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event);
static void rtc_callback(void *event);

static tm_t rtc_curr_time;  /* Data structure used to store the current time from the RTC. */
static tm_t rtc_capture_time; /* RTC time capture */

void main(void)
{
   vbatt_return_t      vbatt_ret;          /* For confirming the return value from the API function. */
   vbatt_info_t        vbatt_info;         /* Data structure for the battery backup function */
    rtc_err_t          rtc_ret;            /* For confirming the return value from the API function. */

       SYSTEM.RSTSR1.BIT.CWSF = 1;              /* set Cold/Warm Start Determination Flag. */

   vbatt_info.callbackfunc = vbatt_callback;   /* Callback function setup */

   /* VBATT pin voltage detection function setup and battery backup supply voltage drop determination. */
   vbatt_ret = R_VBATT_Open(&vbatt_info);
   if (VBATT_SUCCESS != vbatt_ret)
   {
      while(1);
   }

   while(1)
     {
      /* Get time capture */
      rtc_ret = R_RTC_Control(RTC_CMD_CHECK_PIN0_CAPTURE, &rtc_capture_time);
      if (RTC_SUCCESS == rtc_ret)
      {
         printf("tamper 0 detect time: %d/%d %02d:%02d:%02d\n", rtc_capture_time.tm_mon,
                                    rtc_capture_time.tm_mday,
                                    rtc_capture_time.tm_hour,
                                    rtc_capture_time.tm_min,
                                    rtc_capture_time.tm_sec);
      }
   }
}


/* Battery backup function callback function */
static void vbatt_callback(vbatt_cb_evt_t * vbatt_cb_event)
{
   rtc_err_t   ret;                    /* For confirming the return value from the API function. */
   rtc_init_t  rtc_info;                /* RTC data structure */
   rtc_capture_cfg_t capture;              /* RTC capture config structure */
   vbatt_status_t    vbatt_status;
   vbatt_return_t  vbatt_ret;          /* For confirming the return value from the API function. */
```

Executing this function results in the set up callback function (vbatt_callback()) being called.

Figure 4.4 Usage Example in Combination with the RTC FIT Module (1/3)

```
 /* Discriminate based on the argument */
switch(*vbatt_cb_event)
{
   /* If no battery backup supply voltage drop is detected. */
   case VBATT_NOT_DROP_VOLTAGE:
      /* Set up the RTC data structure again */
      rtc_info.output_freq = RTC_OUTPUT_OFF;       /* Stop RTCOUT output */
      rtc_info.periodic_freq = RTC_PERIODIC_1_HZ;  /* RTC periodic interrupt generation period: 1 second */
      rtc_info.periodic_priority = 8;              /* Interrupt priority level */
      rtc_info.set_time = false;                   /* Do not update the RTC clock counter I/O register. */
      rtc_info.p_callback = rtc_callback;          /* Callback function setup */

      /* Set up the RTC capture config structure again */
      capture.pin = RTC_PIN_0;
      capture.edge = RTC_EDGE_RISING;              /* Always edge rising if using RX671 */
      capture.filter = RTC_FILTER_OFF;             /* Filter off if VBATT_CFG_TAMPER_CHEN_CH0_NOISE_FILTER
                                                      is VBATT_TAMPER_CHEN_NOISE_FILTER_ENABLE */

      /* RTC re-setup */
      ret = R_RTC_Open(&rtc_info, &rtc_curr_time);
      if (RTC_SUCCESS != ret)
      {
         while(1);
      }

      /* Read out the current time information from the RTC clock counter I/O register
         and store it in the data structure. */
      ret = R_RTC_Read(&rtc_curr_time, NULL);
      if (RTC_SUCCESS != ret)
      {
         while(1);
      }

      /* RTC config time capture */
      ret = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);
      if (RTC_SUCCESS != ret)
      {
         while(1);
      }

   break;

   /*  When a battery backup supply voltage drop was detected */
   case VBATT_DROP_VOLTAGE:
      /* RTC data structure setup */
      rtc_info.output_freq = RTC_OUTPUT_OFF;       /* Stop RTCOUT output */
      rtc_info.periodic_freq = RTC_PERIODIC_1_HZ;  /* RTC periodic interrupt generation period: 1 second */
      rtc_info.periodic_priority = 8;              /* Interrupt priority level */
      rtc_info.set_time = true;                    /* Update the RTC clock counter I/O register. */
      rtc_info.p_callback = rtc_callback;          /* Callback function setup */

      /* Set the current time information structure time setting to "2015-06-30 12:34:56" */
      rtc_curr_time.tm_sec   = 56;                 /* Seconds (0 - 59) */
      rtc_curr_time.tm_min   = 34;                 /* Minutes (0 - 59) */
      rtc_curr_time.tm_hour  = 12;                 /* Hours (0 - 23) */
      rtc_curr_time.tm_mday  = 30;                 /* Day (1 - 31) */
      rtc_curr_time.tm_mon   = 6;                  /* Month (0 - 11, 0 = January) */
      rtc_curr_time.tm_year  = 115;                /* Year (referenced to 1900) */
      rtc_curr_time.tm_wday  = 0;                  /* Day of week (0 - 6, 0 = Sunday) */
      rtc_curr_time.tm_yday  = 0;                  /* Day in year (0 - 365) */
      rtc_curr_time.tm_isdst = 0;                  /* Daylight saving time in effect (> 0),
                                                      not in effect (= 0). */

      /* Set up the RTC capture config structure again */
      capture.pin = RTC_PIN_0;
      capture.edge = RTC_EDGE_RISING;
      capture.filter = RTC_FILTER_OFF;


      /* RTC initialization */
      ret = R_RTC_Open(&rtc_info, &rtc_curr_time);
      if (RTC_SUCCESS != ret)
      {
         while(1);
      }

      /* RTC config time capture */
      ret = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);
```

Although the RTC I/O register is saved, RAM and other registers are reset. Therefore, the RTC FIT module is setup again.

Since the RTC I/O register is saved, the current time information is read from the RTC I/O register by the R_RTC_Read() function and stored again in the data structure.

If a battery backup supply voltage drop was detected, the microcontroller is in a state where RTC operation is not guaranteed. Therefore the RTC is reinitialized.

```
            if (RTC_SUCCESS != ret)
            {
                while(1);
            }

        break;

        /* Interrupt due to VBATT pin voltage drop */
        case VBATT_MASKABLE_INTERRUPT:
        case VBATT_NON_MASKABLE_INTERRUPT:
            /* Unused in this sample code */
        break;

       /* Tamper detection interrupt */
        case VBATT_TAMPER_CH0_INTERRUPT:
            /* Tamper detection notification */
            printf("tamper 0 detect!!\n");

            /* Wait 1 cycle of sub-clock */
            R_BSP_SoftwareDelay(33, BSP_DELAY_MICROSECS);

            /* Tamper detection clear */
            vbatt_status.tamper_channel = VBATT_TAMPER_CH0;
            vbatt_status.action_clear = true;
            vbatt_ret = R_VBATT_GetStatus(&vbatt_status);
            if (VBATT_SUCCESS != vbatt_ret)
            {
                /* Please do the processing at the time of the error */
            }
        break;
       /* Tamper detection interrupt */
        case VBATT_TAMPER_CH1_INTERRUPT:
            /* Unused in this sample code */
        break;

       /* Tamper detection interrupt */
        case VBATT_TAMPER_CH2_INTERRUPT:
            /* Unused in this sample code */
        break;

        default:
            /* No processing */
        break;
    }
}
```

Figure 4.5 Usage Example in Combination with the RTC FIT Module (2/3)

```
/* RTC callback function */
static void rtc_callback(void *event)
{
    rtc_err_t ret;                      /* For confirming the return value from the API function. */

    /* For a periodic interrupt */
    if (*(rtc_cb_evt_t *)event == RTC_EVT_PERIODIC)
    {
        /* For a periodic interrupt */
        /* Read out the current time information from the RTC clock counter I/O register
           and store it in the data structure. */
        ret = R_RTC_Read(&rtc_curr_time, NULL);
        if (RTC_SUCCESS != ret)
        {
            while(1);
        }

        /* Display on the debugging console. */
        printf("%d/%d/%d %02d:%02d:%02d\n",   rtc_curr_time.tm_year + 1900,
                                    rtc_curr_time.tm_mon,
                                    rtc_curr_time.tm_mday,
                                    rtc_curr_time.tm_hour,
                                    rtc_curr_time.tm_min,
                                    rtc_curr_time.tm_sec);
    }
}
```

Each time a period interrupt occurs (at 1-second intervals) the current time is read out.

Figure 4.6 Usage Example in Combination with the RTC FIT Module (3/3)

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

5.1 vbatt_demo_rskrx671, vbatt_demo_rskrx671_gcc

This is a simple demo of the RX671 Battery Backup Function (VBATTB) for the RSKRX671 starter kit (FIT module "r_vbatt_rx"). This sample demo shows the use of the battery backup function FIT module in combination with the RTC FIT module, which is used for real-time clock settings; it also shows how to use time capture and tamper function.

5.2 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. 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 FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.3 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 this application note and select “Sample Code (download)” from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Reference Documents

User’s Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 24.15	—	First edition issued
1.01	Aug. 31.15	Program	Modified the battery backup function FIT module due to the iodefine.h (V1.0C) is updated. [Description] Compilations error occurs when the iodefine.h (V0.9E) is used. [Workaround] Please use rev.1.01 or a later version of the battery backup function FIT module.
1.02	Feb. 01.19	Program	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
1.03	May 20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Added the section of Target Compilers.
		3	Updated the section of 1.2 Battery Backup Function FIT Module Overview.
		8	Added the section of 1.5 Limitations.
		9	Added the section of 2.4 Interrupt Vector.
		11	Added the section of 2.8 Code Size.
		15	Added the section of 2.13 “for”, “while” and “do while” statements.
		23	Updated the section of R_VBATT_GetVersion().
		24	Added the section of 4.1 Confirmed Operation Environment.
		Program	Deleted the inline expansion of the R_VBATT_GetVersion function.
1.04	Jun.30.19	1	Target Device: Added the RX23W support.
		24	Added Table 4.1 Confirmed Operation Environment (Rev. 1.04) to the section of 4.1 Confirmed Operation Environment.
		26	Updated the section of 4.3 Sample Code.
1.05	Jun.10.20	—	Changed API function comments to Doxygen style.
		1	Deleted Related Documents R01AN1833.
		14	Changed Section 2.12 Adding the FIT Module to Your Project.
		16 to 21	Deleted “Reentrant” item on the API description page.

2.00	Mar.31.21	1	Added support for RX671
		3-10	Section 1.2 Battery Backup Function FIT Module Overview Section 1.3 API Overview Section 1.4 Usage Example Section 1.5 Limitations Updated description for RX671
		12	Added Interrupt vector number for RX671
		13-16	Section 2.7 Configuration Overview Updated description for RX671
		17	Added the section of 2.8 Code Size.
		18-19	Updated structures for RX671 Section 2.9 Arguments
		24-36	Section 3. API Functions Updated example for device RX671
		37	Updated the section of R_VBATT_ReadBackupData()
		38	Updated the section of R_VBATT_WriteBackupData ()
		40	4.1 Confirmed Operation Environment: Added Table for Rev.2.00
		Program	Added support for RX671
2.10	Sep.13.21	44	4.1 Confirmed Operation Environment: Added Table for Rev.2.10.
		51	Added section 5. Demo Projects.
2.20	Jul.29.22	41	4.1 Confirmed Operation Environment: Added Table for Rev.2.20.
		Program	Updated demo projects

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

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/.

Trademarks

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