RX Family

RTC Module Using Firmware Integration Technology

Introduction

This Realtime Clock (RTC) driver supports 24-hr and calendar count operation. Functions include setting of date/time, alarms, periodic interrupts, and clock output, and start/stop setting of counters. For the RX230, RX231, RX23W, RX64M, RX65N, RX71M, and RX72M, the time capture function is supported as well. Recovery from low power consumption states can be performed by an alarm interrupt or periodic interrupt.

Target Device

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

- **RX110, RX111, RX113, RX130 Groups**
- **RX230, RX231,RX23W Groups**
- **RX64M Group**
- **RX65N Group**
- **RX71M Group**
- **RX72M Group**

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

Target Compilers

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

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

Related Documents

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

Contents

1. Overview

This Realtime Clock (RTC) driver supports the 24-hour calendar count mode operation on the RX MCUs. The hardware functionality is detailed in the User's Manual: Hardware.

This driver supports the common RTC functions such as:

- Setting date/time
- Starting/stopping counting
- Setting alarms
- Periodic interrupts
- Clock output

For the RX230, RX231,RX64M, RX65N, RX71M, and RX72M three time capture event input pins are supported:

- RTCIC0
- RTCIC1
- RTCIC2

For the RX23W two time capture event input pins are supported:

- RTCIC0
- RTCIC1

Features not supported by this driver are:

- 12-Hour mode
- Binary count mode
- 30 seconds adjustment function
- Clock error correction function
- Carry interrupt
- Main clock as RTC count source (RX64M, RX65N, RX71M, and RX72M)

2. API Information

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

2.1 Hardware Requirements

This driver requires that your MCU support the following features:

- RTCc, RTCd, RTCe or RTCA peripherals

2.2 Hardware Resource Requirements

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver and the user cannot use them.

2.2.1 RTC

This driver makes use of the RTC peripheral.

2.2.2 I/O Port, MPC

Clock output and the time capture function are available with this driver. When using these functions, corresponding pins need to be configured.

2.2.3 Sub-Clock Oscillator

The RTC peripheral operates on the sub-clock. Before calling this driver's API functions, start the sub-clock oscillator and wait for oscillation to stabilize. Refer to the User's Manual: Hardware for details.

2.3 Software Requirements

This driver is dependent upon the following packages:

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

2.4 Supported Toolchains

This driver is tested and working with the toolchains listed in 6.1 Operation Confirmation Environment.

2.5 Interrupt Vector

The periodic interrupt and the alarm interrupt are enabled with the value specified in the argument for the R_RTC_Open function or the R_RTC_Control function.

Table 2.1 lists the Interrupt Vector Used in the RTC FIT Module

Table 2.1 Interrupt Vector Used in the RTC FIT Module

Device	Interrupt Vector
All target devices	ALM interrupt (vector no.: 92) PRD interrupt (vector no.: 93)

2.6 Header Files

All API function declarations and their supporting interface definitions are located in r_rtc_rx_if.h.

2.7 Integer Types

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

2.8 Configuration Overview

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

<table>
<tr><th colspan="2">Configuration options in r_rtc_rx_config.h</th></tr>
<tr><td>#define RTC_CFG_PARAM_CHECKING_ENABLE
The default value is 1</td><td>If this macro is set to 1, parameter checking is included in the build. If the macro is set to 0, the parameter checking is omitted from the build. Setting this macro to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting.</td></tr>
<tr><td>#define RTC_CFG_CALCULATE_YDAY
The default value is 0</td><td>If this macro is set to 1, when the R_RTC_Read function is called, the number of days from January 1 is calculated and stored in the “tm_yday” member of the “tm_t” structure variable, which is specified in the argument. If this macro is set to 0, calculation of day will be skipped.</td></tr>
<tr><td>Default enable:
#define RTC_CFG_DRIVE_CAPACITY_STD

Different definition:
//#define RTC_CFG_DRIVE_CAPACITY_LO
//#define RTC_CFG_DRIVE_CAPACITY_MD
//#define RTC_CFG_DRIVE_CAPACITY_HI</td><td>This macro specifies the sub-clock oscillator drive capacity. The setting specified when starting the sub-clock oscillator must be set here.

<table>
<tr><th rowspan="2">MCU</th><th colspan="4">Drive Capacity</th></tr>
<tr><th>Low (LO)</th><th>Middle (MD)</th><th>High (HI)</th><th>Standard (STD)</th></tr>
<tr><td>RX11x</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>RX130
RX230
RX231
RX23W
RX64M
RX71M
RX72M
RX65N</td><td>X</td><td>-</td><td>-</td><td>X</td></tr>
</table>
X: Available, -: Not available</td></tr>
</table>

2.9 Code Size

Typical code sizes associated with this module are listed below. Information is listed for a single representative device of the RX100 Series, RX200 Series, and RX600 Series, respectively.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.8, Configuration Overview. The table lists reference values when compile options of the C compiler (described in 2.4, Supported Toolchains) are set to their default values. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

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

Module Revision: r_rtc_rx rev2.77

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

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

GCC for Renesas RX 4.8.4.201902

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

IAR C/C++ Compiler for Renesas RX version 4.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	2,321 bytes	1,995 bytes	4,356 bytes	3,692 bytes	3,275 bytes	2,797 bytes
	RAM	8 bytes		0 bytes		14 bytes	
	STACK*1	112 bytes		-		164 bytes	
RX231	ROM	2,707 bytes	2,282 bytes	5,020 bytes	4,292 bytes	3,856 bytes	3,240 bytes
	RAM	16 bytes		0 bytes		22 bytes	
	STACK*1	112 bytes		-		164 bytes	
RX65N	ROM	2,769 bytes	2,389 bytes	5,236 bytes	4,484 bytes	3,938 bytes	3,366 bytes
	RAM	16 bytes		0 bytes		20 bytes	
	STACK*1	124 bytes		-		192 bytes	

Note1. The sizes of maxmum usage stack of Interrupts functions is included.

2.10 Arguments

The API data structures are located in the file “r_rtc_rx_if.h” and discussed in 3, API Functions.

2.11 Callback Function

In this module, the callback function specified by the user is called in the periodic interrupt handler or the alarm interrupt handler.

The callback function is specified by storing the address of the user function in the “p_callback” structure member (see 2.10, Arguments). When the callback function is called, the variable which stores the constant listed in Table 2.2 is passed as the argument.

The argument is passed as void type. Thus the argument of the callback function is cast to a void pointer. See examples below as reference.

When using a value in the callback function, type cast the value.
Set FIT_NO_FUNC to “p_callback” when not using the callback function.

Table 2.2 Arguments of the Callback Function (enum rtc_cb_evt_t)

Constant	Description
RTC_EVT_ALARM	Callback function called from the alarm interrupt handler.
RTC_EVT_PERIODIC	Callback function called from the periodic interrupt handler.

```
/* Callback function usage example */
      :
rtc_init.p_callback = rtc_callback;       //Set the callback function name.
err = R_RTC_Open(&rtc_init, &init_time);  //RTC initialization

void rtc_callback(void *p_args)
{
      rtc_cb_evt_t event;
      event = *(rtc_cb_evt_t *)p_args;
      if (event == RTC_EVT_PERIODIC)      //Periodic interrupt
      {
            do_something_prd();
      }
      else if (event == RTC_EVT_ALARM)    //Alarm interrupt
      {
            do_something_alm();
      }
}
```

```
/* When not using the callback function. */
      :
rtc_init.p_callback = FIT_NO_FUNC;        //Set ‘FIT_NO_FUNC’.
err = R_RTC_Open(&rtc_init, &init_time);  //RTC initialization
```

2.12 Adding the FIT Module to Your Project

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

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

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

(3) Adding the FIT module to your project using “Smart Configurator” on CS+
By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

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

.
.

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

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

The following shows example of description.

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

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

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

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_RTC_Open()	This function initializes the RTC, sets the current date/time, and configures the periodic interrupt and clock output, and starts counting.
R_RTC_Close()	This function stops counting, and disables the periodic interrupt and the alarm interrupt.
R_RTC_Control()	This function updates the current date/time and the alarm date/time, and configures the time capture function (only when available in the MCU) and other settings.
R_RTC_Read()	This function returns the current date/time and the alarm date/time
R_RTC_GetVersion ()	This function returns the driver version number.

3.2 Return Values

The following enumeration lists the possible error codes that can be returned by the API functions:

```
typedef enum                // RTC API return codes
{
   RTC_SUCCESS,
   RTC_ERR_ALREADY_OPEN,    // R_RTC_Open has already been called.
   RTC_ERR_NOT_OPENED,      // R_RTC_Open is not called.
   RTC_ERR_BAD_PARAM,       // Missing or invalid parameter specified
   RTC_ERR_MISSING_CALLBACK, // Callback function has not been specified.
   RTC_ERR_TIME_FORMAT,     // Improper time format (field out of range)
   RTC_ERR_NO_CAPTURE       // Time capture event is not detected.
} rtc_err_t;
```

3.3 R_RTC_Open ()

This function initializes the RTC, sets the current date/time, configures the relevant interrupt, and starts counting.
The function initializes the RTC FIT module. This function must be called before calling any other API functions.

Format

```
rtc_err_t   R_RTC_Open (rtc_init_t  *p_init,
                        tm_t        *p_current_time);
```

Parameters

p_init
Pointer to initialization structure (see below).

p_current_time
Pointer to date/time structure (see below) to set current time.

Initialization structure used for p_init:

```
typedef struct
{
    rtc_cb_func_t   p_callback;     // Specifies the pointer to the callback
                                    // function.
    rtc_output_t    output_freq;    // Specifies frequency of clock output.
                                    //  (The setting value is invalid when
                                     //   set_time = false.)
    rtc_periodic_t  periodic_freq;  // Specifies the period of the periodic
                                    // interrupt.
    uint8_t         periodic_priority;  // Specifies the periodic interrupt
                                        // priority level.
                                        // INT priority; 0 to 15 (0=disable)
    bool            set_time;       // Executes/skips the RTC initialization and
                                    // date/time setting.
                                    // (true: Execute, false: Skip)
} rtc_init_t;

typedef void (*rtc_cb_func_t)(void *p_args);

typedef enum e_rtc_output
{
    RTC_OUTPUT_OFF,
    RTC_OUTPUT_1_HZ,
    RTC_OUTPUT_64_HZ,
} rtc_output_t;


typedef enum e_rtc_periodic
{
    RTC_PERIODIC_OFF    = 0,
    RTC_PERIODIC_256_HZ = 6,
    RTC_PERIODIC_128_HZ = 7,
    RTC_PERIODIC_64_HZ  = 8,
    RTC_PERIODIC_32_HZ  = 9,
    RTC_PERIODIC_16_HZ  = 10,
    RTC_PERIODIC_8_HZ   = 11,
    RTC_PERIODIC_4_HZ   = 12,
    RTC_PERIODIC_2_HZ   = 13,
    RTC_PERIODIC_1_HZ   = 14,
    RTC_PERIODIC_2_SEC  = 15,
} rtc_periodic_t;
```

Structure used for p_current_time:

```
Typedef struct
{
    int tm_sec;         // Seconds (0-59)
    int tm_min;         // Minutes (0-59)
    int tm_hour;        // Hour (0-23)
    int tm_mday;        // Day of the month (1-31)
    int tm_mon;         // Month (0-11, 0=January)
    int tm_year;        // Year (100-199, 100=Year 2000)
    int tm_wday;        // Day of the week (0-6, 0=Sunday)
    int tm_yday;        // Day of the year (0-365); Setting invalid
                        // (This used when RTC_CFG_CALCULATE_YDAY is 1.)
    int tm_isdst;       // Daylight Savings Time; unused here
                        // (“-1” is set.)
} tm_t;
```

Return Values

RTC_SUCCESS	
RTC_ERR_ALREADY_OPEN	*R_RTC_Open has already been called.*
RTC_ERR_BAD_PARAM	*Missing or invalid parameter specified*
RTC_ERR_MISSING_CALLBACK	*Callback function has not been set.*
RTC_ERR_TIME_FORMAT	*Improper time format (field out of range)*

Properties

Prototyped in file “r_rtc_rx_if.h”.

Description

This function initializes the RTC and starts the RTC counter. The function returns RTC_SUCCESS after the RTC has been initialized and started counting successfully.

When the “set_time” member of the “rtc_init_t” structure is set to ‘true’, the RTC is initialized and date/time is specified with the “p_current_time” argument. When the “set_time” member is false, the “p_current_time” argument is ignored. Normally, “true” is set at cold start and “false” at warm start (such as reset).

The “tm_t” structure which is used for “p_current_time” is defined in the C standard library. If the compiler does not support it, the “tm_t” structure defined in the “r_rtc_rx_if.h” file is used.

Example

```
rtc_err_t  err;
rtc_init_t rtc_init;

/* Set the current date & time to be Aug 31, 2015 (Monday) 11:59:20pm */
tm_t init_time =
{
      20, //Seconds (0-59)
      59, //Minutes (0-59)
      23, //Hour (0-23)
      31, //Day of the month (1-31)
       7, //Month (0-11, 0=January)
     115, //Year (100-199, 100=Year 2000)
       1, //Day of the week (0-6, 0=Sunday)
       0, //Day of the year (0-365); disabled
       0, //Daylight savings; disabled
};

rtc_init.output_freq = RTC_OUTPUT_1_HZ;     // Generate 1 Hz output clock
rtc_init.periodic_freq = RTC_PERIODIC_2_HZ; // Gen periodic int every .5sec
rtc_init.periodic_priority = 7;             // Set the periodic interrupt
                                            // priority level to 7.
rtc_init.set_time = true;                   // Perform RTC initialization
                                            // and date/time setting.
rtc_init.p_callback = rtc_callback;         // Set the callback function.

err = R_RTC_Open(&rtc_init, &init_time);
```

Special Notes:

Before calling this function, start the sub-clock oscillator and wait for oscillation to stabilize. For details on oscillating the sub-clock and specifying the oscillation stabilization wait time, refer to the User's Manual: Hardware for the MCU used.

This function must be called regardless of cold start or warm start.

And keep following notes when using clock output.

- Configure the RTCOUT pin with the application software after initializing clock output with the R_RTC_Open function or R_RTC_Control function. Refer to 4. Pin Setting for details.
- In warm start mode (rtc_init_t->set_time = false), configuration of clock output by R_RTC_Open function are invalid. To use clock output at warm start, configure clock output with the R_RTC_Control function after calling the R_RTC_Open function.

3.4 R_RTC_Close ()

This function stops counting, resets the RTC, and disables all RTC interrupts.

Format
```
void   R_RTC_Close (void);
```

Parameters
None.

Return Values
None.

Properties
Prototyped in file “r_rtc_rx_if.h”.

Description
This function stops counting, resets the RTC, and disables all RTC interrupts.

Example
```
rtc_err_t   err;
rtc_init_t  rtc_init;
tm_t        init_time;

     :
    err = R_RTC_Open(&rtc_init, &init_time);
     :
    R_RTC_Close();
```

Special Notes:
None.

3.5 R_RTC_Control ()

This function updates the current date/time and the alarm date/time, and configures the time capture function (only when available in the MCU) and other settings.

Format

```
rtc_err_t   R_RTC_Control(rtc_cmd_t    cmd,
                          void   *p_args);
```

Parameters

cmd

Command to process (see enum below)

p_args

Pointer to optional argument structure (refer to the Description for each command setting.)

Commands available:

```
typedef enum
{
    /*   All MCUs   */
    RTC_CMD_SET_OUTPUT,
    RTC_CMD_SET_PERIODIC,
    RTC_CMD_SET_CURRENT_TIME,
    RTC_CMD_SET_ALARM_TIME,
    RTC_CMD_ENABLE_ALARM,
    RTC_CMD_STOP_COUNTERS,
    RTC_CMD_START_COUNTERS,
    RTC_CMD_PARTIAL_RESET,

    /*  RX230, RX231, RX23W, RX64M, RX65N, RX71M, RX72M only  */
    RTC_CMD_CONFIG_CAPTURE,
    RTC_CMD_CHECK_PIN0_CAPTURE,
    RTC_CMD_CHECK_PIN1_CAPTURE,
    RTC_CMD_CHECK_PIN2_CAPTURE,
    RTC_CMD_DISABLE_CAPTURE
} rtc_cmd_t;
```

Return Values

RTC_SUCCESS	
RTC_ERR_NOT_OPENED	*R_RTC_Open is not called.*
RTC_ERR_BAD_PARAM	*Missing or invalid parameter specified*
RTC_ERR_MISSING_CALLBACK	*Callback function has not been specified.*
RTC_ERR_TIME_FORMAT	*Improper time format (field out of range)*
RTC_ERR_NO_CAPTURE	*Time capture event is not detected.*

Properties

Prototyped in file “r_rtc_rx_if.h”.

Description

This function updates the current date/time and the alarm date/time, and configures the time capture function (only when available in the MCU) and other settings. A brief summary for each command follows.

RTC_CMD_SET_OUTPUT:
This command is to change the setting for clock output using the “rtc_output_t” structure. And it stops counting while the setting. The following shows a sample call:

```
rtc_output_t  out_freq=RTC_OUTPUT_OFF;

err = R_RTC_Control(RTC_CMD_SET_OUTPUT, &out_freq);
```

RTC_CMD_SET_PERIODIC:
This command is to change the periodic interrupt generation interval using the “rtc_periodic_cfg_t” structure. The following shows a sample call:

```
rtc_periodic_cfg_t  periodic;

periodic.frequency = RTC_PERIODIC_2_HZ;     // Get INT every 1/2 second
periodic.int_priority = 9;
err = R_RTC_Control(RTC_CMD_SET_PERIODIC, &periodic);
```

RTC_CMD_SET_CURRENT_TIME:
This command is to change the current date/time setting using the “tm_t” structure. And it stops counting while the setting. The following shows a sample call:

```
tm_t    time;
 :
err = R_RTC_Control(RTC_CMD_SET_CURRENT_TIME, &time);
```

RTC_CMD_SET_ALARM_TIME:
This command is to set the alarm date/time using the “tm_t” structure. When setting the alarm date/time, disable the alarm function with RTC_CMD_ENABLE_ALARM certainly before the setting. The following shows a sample call:

```
tm_t    time;
 :
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &time);
```

RTC_CMD_ENABLE_ALARM:
This command is to specify fields (year, month, day of the month/week, etc.) to compare the current date/time with the alarm date/time, and enable the alarm interrupt. The following shows a sample call:

```
tm_t              time;
rtc_alarm_ctrl_t  alarm;

/* CREATE ALARM FOR 9:00AM ON THE 1st OF EVERY MONTH */
time.tm_sec = 0;    // Seconds (0-59)
time.tm_min = 0;    // Minutes (0-59)
time.tm_hour = 9;   // Hour (0-23)
time.tm_mday = 1;   // Day of the month (1-31)
time.tm_mon = 0;    // Month (0-11, 0=January)
time.tm_year = 100; // Year (100-199, 100=Year 2000)
time.tm_wday = 0;   // Day of the week (0-6, 0=Sunday)
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &time);

alarm.int_priority = 4;   // Set the alarm interrupt priority level to 4.
alarm.sec = false;        // Seconds
alarm.min = false;        // Minutes
alarm.hour = true;  // Hour (true = compare with current time)
alarm.mday = true;  // Day of the month (true = compare with current date)
alarm.mon = false;        // Month
alarm.year = false;       // Year
alarm.wday = false;       // Day of the week
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
```

RTC_CMD_STOP_COUNTERS:
This command is to stop counting. Set the second argument to NULL or FIT_NO_PTR.
The following shows a sample call:

```
R_RTC_Control(RTC_CMD_STOP_COUNTERS, NULL);
```

RTC_CMD_START_COUNTERS:
This command is to resume counting after it is halted by RTC_CMD_STOP_COUNTERS. Set the second argument to NULL or FIT_NO_PTR. The following shows a sample call:

```
R_RTC_Control(RTC_CMD_START_COUNTERS, NULL);
```

RTC_CMD_PARTIAL_RESET:
This command is to reset registers for clock output, alarm, and time capture (see the RCR2.RESET register bit description in the User's Manual: Hardware for a complete list of affected registers). Set the second argument to NULL or FIT_NO_PTR. The following shows a sample call:

```
R_RTC_Control(RTC_CMD_PARTIAL_RESET, NULL);
```

RTC_CMD_CONFIG_CAPTURE:
This command is to specify the event detection condition for RTCIC0, RTCIC1, or RTCIC2 pin using the "rtc_capture_cfg_t" structure. The following shows a sample call:

```
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);
```

RTC_CMD_CHECK_PIN0_CAPTURE:
RTC_CMD_CHECK_PIN1_CAPTURE:
RTC_CMD_CHECK_PIN2_CAPTURE:
After the capture pin is configured, it must be polled to determine if an event has occurred. When a capture was made, the captured date and time are stored in the argument specified as the second parameter and RTC_SUCCESS is returned. When a capture was not made, RTC_ERR_NO_CAPTURE is returned. The following shows a sample call:

```
tm_t                time;
rtc_err_t           err;
rtc_capture_cfg_t   capture;

  :
err = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);

while(1)
{
    /* main processing */
      :
    /* check if an event was detected on the RTCIC0 pin. */
    if (R_RTC_Control(RTC_CMD_CHECK_PIN0_CAPTURE, &time) == RTC_SUCCESS)
    {
        /* If event was detected outside of 9-5 business hours */
        if ((time.tm_hour < 9) || (time.tm_hour > 17))
        {
            RED_LED = ON;
            write_flash(log_addr, sizeof(tm_t), &time);
            log_addr += sizeof(tm_t);
        }
    }
}
```

RTC_CMD_DISABLE_CAPTURE:
This command is to disable the capture pin setting. Use RTC_CMD_CONFIG_CAPTURE to enable again. The following shows a sample call:

```
    rtc_pin_t      pin=RTC_PIN_0;

    err = R_RTC_Control(RTC_CMD_DISABLE_CAPTURE, &pin)
```

Example

```
/* CREATE ALARM INTERRUPT TO OCCUR EVERY 30 SECONDS */
rtc_err_t err;
rtc_init_t rtc_init;
tm_t g_init_time={0, 0, 0, 1, 0, 100, 0, 0, 0};
rtc_alarm_ctrl_t alarm={4, false, false, false, false, false, false, false};
tm_t alm_time;

rtc_init.output_freq = RTC_OUTPUT_OFF;    // Clock is not output.
rtc_init.periodic_freq = RTC_PERIODIC_OFF;// Disables the periodic interrupt.
rtc_init.periodic_priority = 0;           // Sets the periodic interrupt
                                          // priority level to 0.
rtc_init.set_time = true;                 // Performs RTC initialization and
                                          // date/time setting.
rtc_init.p_callback = rtc_callback;       // Specifies the callback function.

err = R_RTC_Open(&rtc_init, &g_init_time);

/* Issues the alarm interrupt request when the value of second becomes 30. */
alm_time = g_init_time;
alm_time.tm_sec = 30;
alm_time.int_priority = 7;                // Sets the alarm interrupt priority
                                          // level to 7.
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &alm_time);

/* Enables the second field for alarm. */
alarm.sec = true;
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
      :
/* Callback function */
void rtc_callback(void *p_args)
{
      rtc_err_t err;
  // Resets the current time to 0. When the value of second becomes 30,
  // the alarm interrupt again occurs.
  err = R_RTC_Control(RTC_CMD_SET_CURRENT_TIME, &g_init_time);

  // Processing to be executed here.
}
```

Special Notes:

When using time capture function, the pins to be used must be configured by the application software before executing the RTC_CMD_CONFIG_CAPTURE command in the R_RTC_Control function after calling the R_RTC_Open function. Refer to 4. Pin Setting for details.
And executing the RTC_CMD_SET_OUTPUT command or the RTC_CMD_SET_CURRENT_TIME command stops RTC counting while processing.

3.6 R_RTC_Read ()

This function returns the current date/time and the alarm date/time set in the RTC.

Format

```
rtc_err_t    R_RTC_Read (tm_t * p_current_time,
                         tm_t * p_alarm_time);
```

Parameters

p_current

Pointer for loading the current date/time from the RTC. Specify NULL or FIT_NO_PTR to skip reading the current date/time.

p_alarm

Pointer for loading the alarm date/time from the RTC. Specify NULL or FIT_NO_PTR to skip reading the alarm date/time.

Return Values

RTC_SUCCESS
RTC_ERR_NOT_OPENED *R_RTC_Open is not called.*

Properties

Prototyped in file “r_rtc_rx_if.h”.

Description

This function reads the current date/time and the alarm date/time.

Example

```
    tm_t       cur_time;
    tm_t       alm_time;
    rtc_err_t  err;

    err = R_RTC_Read(&cur_time, NULL);          // Read current date/time only
    err = R_RTC_Read(NULL, &alm_time);          // Read alarm date/time only
    err = R_RTC_Read(&cur_time, &alm_time);     // Read both date/times
```

Special Notes:

To read the current date/time using this function after return from a reset, deep software standby mode, software standby mode, or the battery backup state, wait for 1/128 second while counting has been started with the condition of (RCR2.START bit = 1).
When a carry of the RTC counter occurs while reading the current time, this function reads the current time again. For checking the carry, the function uses carry interrupt status flag (IR bit)
For that, it enables carry interrupt (RCR1.CIE bit = 1). So, do not clear this status flag in the application software.

3.7 R_RTC_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_RTC_GetVersion(void);

Parameters
None.

Return Values
Version number.

Properties
Prototyped in file “r_rtc_rx_if.h”

Description
Returns the version of this module. The top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Example
```
uint32_t   version;
version = R_RTC_GetVersion();
```

Special Notes:
None.

4. Pin Setting

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

Set the RTCOUT pin according to the following.

- At cold start mode (rtc_init_t->set_time = true), configure the RTCOUT pin after setting clock output with the R_RTC_Open function or R_RTC_Control function.
- At warm start mode (rtc_init_t->set_time = false), setting of clock output by R_RTC_Open function is invalid. After calling the R_RTC_Open function, configure the RTCOUT pin after setting clock output with the R_RTC_Control function.

Perform the RTCICn (n = 0 to 2) pin setting before executing the RTC_CMD_CONFIG_CAPTURE command in the R_RTC_Control function after calling the R_RTC_Open function.

When performing the Pin Setting in the e^2 studio, the Pin Setting feature of the FIT Configurator or the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT Configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.

Table 4.1 Function Output by the FIT Configurator

MCU Used	Function to be Output	Remarks
All MCUs	R_RTC_PinSet()	For the RX100 Series, the setting for RTCICn (n = 0 to 2) is not output.

5. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g.. r_bsp).

5.1 rtc_demo_rskrx130

Description

A simple demo of the RX130 Realtime Clock (RTCc) for the RSKRX130 starter kit (FIT module “r_rtc_rx”). The demo uses the RTC API from r_rtc_rx_if.h to initialize the realtime clock to an arbitrary date/time and start a 2 sec periodic interrupt. The interrupt handler reads the current date/time into global variables for printing to the debug console by main(). LED 0 is also toggled when the periodic timer expires.

Setup and Execution

1. Compile and download the sample code.
2. Click ‘Reset Go’ to start the software. If PC stops at Main, press F8 to resume.
3. Set breakpoints and watch global variables

Boards Supported

RSKRX130

5.2 rtc_demo_rskrx231

Description

A simple demo of the RX231 Realtime Clock (RTCe) for the RSKRX231 starter kit (FIT module “r_rtc_rx”). This demo is identical to the RX130 demo above.

Boards Supported

RSKRX231

5.3 rtc_demo_rskrx64m

Description

A simple demo of the RX64M Realtime Clock (RTCd) for the RSKRX64M starter kit (FIT module “r_rtc_rx”). This demo is identical to the RX130 demo above.

Boards Supported

RSKRX64M

5.4 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”.

6. Appendices

6.1 Operation Confirmation Environment

This section describes operation confirmation environment for the RTC FIT module.

Table 6.1 Operation Confirmation Environment (Rev. 2.41)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 4.2.0.012
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.04.01
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.41
Board used	Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE)

Table 6.2 Operation Confirmation Environment (Rev. 2.50)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 5.0.1.005
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.05.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.50
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE)

Table 6.3 Operation Confirmation Environment (Rev. 2.70)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0.XXX
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.70
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE)

Table 6.4 Operation Confirmation Environment (Rev. 2.71)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0.XXX
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.71
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX113 (product No.: R0K505113SxxxBE)

Table 6.5 Operation Confirmation Environment (Rev. 2.72)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.1.0.XXX
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.72
Board used	Renesas Starter Kit+ for RX64M (product No.: R0K50564MSxxxBE) Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE) Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE)

Table 6.6 Operation Confirmation Environment (Rev. 2.73)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.73

Table 6.7 Operation Confirmation Environment (Rev. 2.74)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.74

Table 6.8 Operation Confirmation Environment (Rev. 2.75)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201803 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 4.10.01 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.75
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxx)

Table 6.9 Operation Confirmation Environment (Rev. 2.76)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00
	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.2.76
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxx)

Table 6.10 Operation Confirmation Environment (Rev. 2.77)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment
Endian	Big endian/little endian
Revision of the module	Rev.2.77
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

6.2 Troubleshooting

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

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

- When using CS+:
  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"
- When using e^2 studio:
  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

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

(2) Q: I have added the FIT module to the project and built it. Then I got the following errors:
ERROR - A drive capacity #define must be uncommented in r_rtc_rx_config.h
ERROR - Only one drive capacity #define may be uncommented in r_rtc_rx_config.h
ERROR - RTC_CFG_DRIVE_CAPACITY_MD in r_rtc_rx_config.h is invalid selection for MCU.

A: The setting value in the "r_rtc_rx_config.h" file may be wrong. Check the file "r_rtc_rx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.8, Configuration Overview for details.

(3) Q: A clock is not output from the RTCOUT pin.

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

(4) Q: An event is not detected though an edge is input to the RTCICn (n = 0 to 2) pin.

A: The event detection condition or the pin setting may not be configured properly. Check the setting for the RTC_CMD_CONFIG_CAPTURE command in 3.5, R_RTC_Control (). Also, confirm that the RTCICn (n = 0 to 2) pin is set to general I/O port.

(5) Q: Even if the R_RTC_Open function is called, an infinite loop is entered within the function and counting is not started.

A: The sub-clock may not oscillate correctly. Check whether the sub-clock starts oscillating before calling the R_RTC_Open function. Then, follow the setting procedure in the User's Manual: Hardware.

(6) Q: The counter is always initialized at warm start.

A: When calling the R_RTC_Open function at warm start, check whether 'false' is set to "set_time" in the "rtc_init_t" structure. Also, when processing is branched using the cold/warm start determination flag (RSTSR1.CWSF), check whether the flag is set to 1 (warm start) in the cold start processing.

7. 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 Record

Rev.	Date	Description	
		Page	Summary
1.00	Nov.22.13	—	First edition issued
2.00	Apr.16.14	all	Modified for new API. Added support for RX110, RX210, and RX63N/631 Added support for Capture feature
2.10	Sep.03.14	1,3-4, 6-7,11	Added support for RX64M.
2.20	Dec.03.14	1,3-5	Added support for RX113.
2.30	Jan.26.15	1,3,5,7,11, 18	Added support for RX71M.
2.40	Jul.20.15	1,2,5,18	Added support for RX231, added RX231 demo.
2.41	Mar.1.16	1,3,5,6,8,9 ,13	Added support for RX130, 230.
		program	Added definition for sub-clock drive capacity. RTC_CFG_DRIVE_CAPACITY_MD Added the rtc_enable_ints function in order to enable the interrupt regardless of the cold start or warm start. Fixed the issue of initial setting procedure for the time capture.
2.50	Oct.1.16	1,3,5,6,12, 19	Added support for RX65N.
		6	Changed a description of code size in section 2.9.
		12	Modified a setting example for the RTCOUT pin. Added a description on how to set up a callback function in section 3.3.
		17	Deleted a setting example for the RTCOUT pin. Modified a setting example for the timestamp capture event input pins.
		20	Added “4. Pin Setting”.
		program	Change the range of values that can be set in the interrupt priority level. (Can set value of 0) Change the specification for the registration of a callback function. Changed the setting of the carry interrupt enable bit (RCR1.CIE) specified by the R_RTC_Open function from “enabled” to “disabled”. (This FIT module does not support the carry interrupt, therefore the specification has been improved to disable an unused interrupt.)
2.60	Mar.31.17	4	2.2 Hardware Resource Requirements: - 2.2.2 I/O Port, MPC: Modified the description regarding the MPC. - 2.2.3 Sub-Clock Oscillator: Modified. 2.4 Supported Toolchains: Now the detailed information of toolchains is listed in Section 5.1. 2.5 Interrupt Vector: Added. 2.6 Header Files: Deleted unnecessary information. 2.7 Integer Types: Deleted unnecessary information.

Rev.	Date	Description	
		Page	Summary
2.60	Mar.31.17	5	2.8 Configuration Overview: Modified some descriptions.
		7	2.11 Callback Function: Added.
		8	2.12 Adding the FIT Module to Your Project: Revised.
		9	3.1 Summary: Modified some descriptions. 3.2 Return Values: Modified some descriptions.
		10	3.3 R_RTC_Open (): - Parameters: Modified the descriptions for structures.
		11	- Description: Modified.
		12	- Example: Modified. - Special Notes: Modified the description regarding the sub-clock and moved the description regarding the callback function to 2.11, Callback Function.
		14	3.5 R_RTC_Control (): - Parameters: Modified the description regarding the structure. - Description: Modified the description for each command.
		17	- Example: Modified. - Special Notes: Modified the description regarding the capture pin.
		18	3.6 R_RTC_Read(): - Parameters: Deleted the description regarding the structure. - Description: Modified.
		20	4. Pin Setting: Modified.
2.70	Jul.31.17	—	Added support for RX130-512KB and RX65N-2MB.
		—	Removed RX210, RX631, and RX63N from the target device in this FIT module since the release of the CGC FIT module has been canceled for RX210, RX631, and RX63N.
		1	Related Documents: Added the following document: “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)”
		4	2.1 Hardware Requirements: Deleted RTCa (RX210) and RTCb (RX631, RX63N).
		6	2.9 Code Size: Revised the description above the table and updated the ROM sizes in the table.
		21	5. Demo Projects: Deleted.
		21	5. Appendices: Added.
		23	6. Reference Documents: Added.
		Program	Removed the definitions for RX210, RX631, and RX63N from the conditional expression of the preprocessor.
			Fixed the following issues in the rtc_init function: - The RCR3.RTCEN bit is set to 0 in the beginning of processing. - When the R_RTC_Open function is called while the sub-clock oscillator does not operate, an infinite loop is entered in the verification processing after setting registers.
			Added the dummy read processing after setting registers in the rtc_set_current_time function and the rtc_set_alarm_time function.

Rev.	Date	Description	
		Page	**Summary**
2.71	Sep.20.17		3.1 R_RTC_Open()
		10	- Parameters Added descriptions for output_freq member.
		12	- Special Notes Added descriptions. 3.5 R_RTC_Control()
		15	- Description RTC_CMD_SET_OUTPUT command Added descriptions. RTC_CMD_SET_CURRENT_TIME command Added descriptions. RTC_CMD_SET_ALARM_TIME command Added descriptions.
		17	- Special Notes Added descriptions. 3.6 R_RTC_Read()
		18	- Special Notes Added descriptions.
		20	4.Pin Setting Added descriptions. 5.1 Operation Confirmation Environment
		22	Added Table 5.4.
		Program	Modified the following issues. **Processing for reading the current time** [Description] When a carry of the RTC counter occurs while reading the current time using the R_RTC_Read function, an incorrect time is read. (According to the specification, the software reads again the current time when a carry occurs. But the carry is not detected and the current time is not read again.) e.g) A carry occurs after just reading seconds at 0:00:59. Although the time should be 0:01:00, the time readout is 0:01:59 (59 seconds difference) [Conditions] When a carry of the RTC counter occurs while reading the RTC counter. [Workaround] Use rev. 2.71 or a later version of the RTC FIT module. The definition is changed as follows. rev.2.70) RTC_INT_ENABLE (0x05) rev.2.71) RTC_INT_ENABLE (0x07) By this way, carry interrupt enable bit (RCR1.CIE) becomes enabled and the process detects a carry during reading of the current time and reads again.

Rev.	Date	Description Page	Summary
2.71	Sep.20.17	Program	**Stop counting at warm start** [Description] If "false" is set to the member "set_time" of the argument "rtc_init_t" of the R_RTC_Open function, the count is stopped during the processing of the R_RTC_Open function without specific operation.(Above is occurred at warm start) [Conditions] When "false" is set to the member "set_time" of the argument "rtc_init_t" of the R_RTC_Open function. [Workaround] Use rev. 2.71 or a later version of the RTC FIT module. In rev. 2.71, the rtc_set_output function called in the R_RTC_Open function is moved into the routine that executes only at cold start. This change will prevent the count from temporarily stopping at warm start. Added limitation Clock output setting by the R_RTC_Open function is invalid at warm start.
2.72	Dec.14.17	4	2.4 Supported Toolchains The following is changed. 5.1 -> 6.1 5. Demo Projects
		21	Added descriptions. 6.1 Operation Confirmation Environment
		23	Added Table 6.5
2.73	Dec.03.18	23	6.1 Operation Confirmation Environment: Added Table 6.6 Confirmed Operation Environment (Rev. 2.73).
		Program	Added document number of the application note accompanying the sample program of the FIT module to xml file.
2.74	Feb.01.19	24	6.1 Operation Confirmation Environment: Added Table 6.7 Confirmed Operation Environment (Rev. 2.74).
		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.

Rev.	Date	Description	
		Page	Summary
2.75	May.20.19	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Added Target Compilers.
		1	Deleted R01AN1723, R01AN1826, R20AN0451 from Related Documents.
		4	Added revision of dependent r_bsp module in 2.3 Software Requirements.
		6	2.9 Code Size, amended.
		19	3.7 R_RTC_GetVersion function, deleted special notes.
		24	Added Table 6.8 in Operation Confirmation Environment.
2.76	Jun.20.19	1	Added the RX23W group to the Target Device. Deleted R01AN1833 from Related Documents.
		3	In overview, Added time capture event input pins for RX23W.
		5	Added RX23W to the subclock oscillator drive capability table.
		6	2.9 Code Size, amended.
		8	Added Section 2.13 “for”, “while” and “do while” statements.
		24	Added Table 6.9 Confirmed Operation Environment (Rev. 2.76).
2.77	Jul.30.19	1	Added the RX72M group to the Target Device.
		3	In overview, Added time capture event input pins for RX72M.
		5	2.8 Configuration Overview, Added RX72M to the subclock oscillator drive capability table.
		6	2.9 Code Size, amended.
		9-19	Deleted the Reentrant for each API in API Functions.
		25	Added Table 6.10 Confirmed Operation Environment (Rev. 2.77).

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

All trademarks and registered trademarks are the property of their respective owners.

Contact information

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