RX Family

RTC Module Using Firmware Integration Technology

R01AN1817EJ0250
Rev. 2.50
Oct 1, 2016

Introduction

This Real-Time Clock (RTC) driver supports 24-hr (versus 12-hour am/pm) and calendar count (versus binary count) operation. Functions include setting of the date/time, setting and enabling alarms, counters, periodic interrupts, and an output clock. For the RX210, RX230, RX231, RX63N/631, RX64M, RX65N, and RX71M the time Capture facility is supported as well. Recovery from software standby mode 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**
- **RX210 Group**
- **RX230, RX231 Groups**
- **RX631, RX63N Groups**
- **RX64M Group**
- **RX65N Group**
- **RX71M Group**

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

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- Board Support Package Firmware Integration Technology Module (R01AN1685)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)

Contents

1. Overview

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

This driver supports the common RTC functions such as:

- setting date/time
- starting/stopping counting
- setting alarms
- periodic interrupts
- output clock

For the RX210, RX230, RX231, RX63N/631, RX64M, RX65N, and RX71M three timestamp capture event input pins are supported:

- RTCIC0
- RTCIC1
- RTCIC2

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 clock source (RX63N/631, RX64M,RX65N, and RX71M; not battery-backed)

2. API Information

The sample code provided with this application note has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

This driver requires that your MCU support the following features:

- RTCa, RTCb, 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 GPIO,MPC

This driver utilizes RTCOUT pin for clock output and must be configured as a peripheral for this usage. The Capture pins RTCIC0, RTCIC1, and RTCIC2 should be configured as input GPIO.

2.2.3 Sub-Clock Oscillator

The RTC peripheral operates on sub-clock. The sub-clock is enabled by this driver. After starting sub-clock oscillator, it must wait for oscillation to stabilize. The sub-clock may be started prior to opening this driver if the accuracy is required.

2.3 Software Requirements

This driver is dependent upon the following packages:

- Renesas Board Support Package (r_bsp)

2.4 Limitations

No software limitations.

2.5 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v2.02.00
- Renesas RX Toolchain v.2.04.01 (RX130)
- Renesas RX Toolchain v.2.05.00 (RX65N)

2.6 Header Files

All API function declarations and their supporting interface definitions are located in r_rtc_rx_if.h. This file should be included by any file which makes an API call. Compile time configurable options are located in r_rtc_rx\ref\ r_rtc_rx_config_reference.h. This file should be copied into the r_config subdirectory and renamed to t_rtc_rx_config.h then modified there. The original file serves as a reference.

2.7 Integer Types

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

2.8 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 <i>r_rtc_rx_config.h</i></th></tr>
<tr><td><code>#define RTC_CFG_PARAM_CHECKING_ENABLE</code>
<code>Note:The default value is 1</code></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><code>#define RTC_CFG_CALCULATE_YDAY</code>
<code>Note:The default value is 0</code></td><td>If this macro is set to 1, R_RTC_Read() will calculate the day of the year by software. If this macro is set to 0, R_RTC_Read() will skip calculation of day of the year.</td></tr>
<tr><td><code>Default enable:</code>
<code>#define RTC_CFG_DRIVE_CAPACITY_STD</code>

<code>Different definition:</code>
<code>//#define RTC_CFG_DRIVE_CAPACITY_LO</code>
<code>//#define RTC_CFG_DRIVE_CAPACITY_MD</code>
<code>//#define RTC_CFG_DRIVE_CAPACITY_HI</code></td><td>This macro specifies the sub-clock oscillator drive capacity. Uncomment the line which applies to the circuit (no need to assign a value).
<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
RX210
RX230
RX231
RX631
RX63N
RX64M
RX71M
RX65N</td><td>X</td><td>-</td><td>-</td><td>X</td></tr>
</table>
</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 the C compiler’s compile options are set to their default values, as described in 2.5, Supported Toolchains. 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.

ROM, RAM and Stack Code Sizes				
Device	Category	Memory Used		Remarks
		With Parameter Checking	Without Parameter Checking	
RX130	ROM	1,800 bytes	1,429 bytes	
	RAM	8 bytes		
	Maximum stack usage	116 bytes		
RX231	ROM	2,247 bytes	1,815 bytes	
	RAM	16 bytes		
	Maximum stack usage	116 bytes		
RX65N	ROM	2,249 bytes	1817 bytes	
	RAM	16 bytes		
	Maximum stack usage	128 bytes		

2.10 API Data Structures

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

2.11 Fit Module Addition Methods

This module must be added to each project used in e^2 studio.

There are two methods for adding to a project: using the FIT plug-in and adding manually.

When the FIT plug-in is used, FIT modules can be added to projects easily and the include file path will be updated automatically. Therefore we recommend using the FIT plug-in when adding FIT modules to a project.

For details on the method for addition FIT modules using the FIT plug-in, see section 2., Adding FIT Modules to e^2 studio Projects Using FIT Plug-In, in the “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” application note.

See section 3, Adding FIT Modules to e^2 studio Projects Manually, for adding the FIT module by hand without using the FIT plugin.

When this FIT module is used, the Board Support Package FIT module (BSP module) must also be added to the project. See the Board Support Package Module Using Firmware Integration Technology (R01AN1685) application note for the methods for adding the BSP module.

.

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 current time, and configures the periodic interrupt and output clock, and starts the counters.
R_RTC_Close()	This function stops the RTC counter and disables the relevant interrupts.
R_RTC_Control()	This function updates the time & alarm settings, handles capture operation (RX210/RX230/RX231/RX631/RX63N/RX64M/RX65N/RX71M), and provides other configuration and control commands.
R_RTC_Read()	This function returns the current date/time and 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,        // Multiple calls to Open
   RTC_ERR_NOT_OPENED,          // Open not called
   RTC_ERR_BAD_PARAM,           // Missing or invalid parameter specified
   RTC_ERR_MISSING_CALLBACK,    // Callback function has not been set.
                                // Periodic or Alarm INTs requested
   RTC_ERR_TIME_FORMAT,         // Improper time format (field out of range)
   RTC_ERR_NO_CAPTURE           // Capture event did not occur
} rtc_err_t;
```

3.3 R_RTC_Open ()

This function initializes the RTC, sets the current time, and configures the relevant interrupts and starts the RTC counter.
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 time/date structure (see below) to set current time.

Initialization structure for p_init:

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


typedef struct
{
    rtc_cb_func_t   p_callback;          // For alarm and periodic interrupts
    rtc_output_t    output_freq;
    rtc_periodic_t  periodic_freq;
    uint8_t         periodic_priority;   // INT priority; 0=disable, 1=low,
15=high
    bool            set_time;            // true if want time argument applied
} rtc_init_t;


typedef enum e_rtc_output
{
    RTC_OUTPUT_OFF,
    RTC_OUTPUT_1_HZ,
    RTC_OUTPUT_64_HZ,
    //RX110,RX111,RX113,RX130,RX230,RX231,RX64M,RX65N,RX71M only
} 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;
```

The structure tm_t for p_current_time is found in the standard C header file “time.h”. If the compiler does not have this file, it is automatically reproduced in the r_rtc_rx_if.h interface file. Note that time is always in 24-hour format.

```
Typedef struct
{
    int tm_sec;         // Seconds (0-59)
    int tm_min;         // Minute (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 since 1900
    int tm_wday;        // Day of the week (0-6, 0=Sunday)
    int tm_yday;        // Day of the year (0-365); unused here
    int tm_isdst;       // Daylight Savings Time; unused here
} tm_t;
```

Return Values

RTC_SUCCESS	
RTC_ERR_ALREADY_OPEN	*Multiple calls to Open*
RTC_ERR_BAD_PARAM	*Missing or invalid parameter specified*
RTC_ERR_MISSING_CALLBACK	*Callback function has not been set.* *Periodic or Alarm INTs requested*
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. The function returns *RTC_SUCCESS* after the RTC is successfully initialized and started. The field “set_time” in `rtc_init_t` will apply the “current_time” argument when set to “true”. Generally, this field should be true when a cold start (POR) is detected and “false” when a warm start is detected (reset signal). The “current_time” field is ignored when “set_time” is false.

This driver does not automatically support Daylight Savings Time. The application must adjust the time when appropriate. This driver does automatically support leap year adjustments.

Reentrant

No.

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, //Second
      59, //Minutes
      23, //Hours
      31, //Day of month
       7, //Month
     115, //Years since 1900
       1, //Day of week
       0, //
       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;
rtc_init.set_time = true;
rtc_init.p_callback = rtc_callback;

err = R_RTC_Open(&rtc_init, &init_time);    // Initialize the RTC
```

Special Notes:

The sub-clock may take up to 2 seconds to stabilize after starting. For greatest accuracy, use the CGC FIT module call R_CGC_ClockStart( ) to start the sub-clock 2 seconds prior to calling Open().

If using the output clock, the application must complete MPC and PORT initialization after to calling Open(). A sample initialization is provided here:

```
    /* RTCOUT on pin PA1 */
    PORTA.PMR.BIT.B1 = 0;          // gpio
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);  // unlock
    MPC.PA1PFS.BIT.PSEL = 0x07; // RTCOUT
    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);  // lock
    PORTA.PMR.BIT.B1 = 1;          // peripheral, not gpio
```

If using the periodic or alarm interrupts, a callback function must be specified. The only argument passed in is what event (interrupt) occurred. However, if you do not use a callback function, please set the FIT_NO_FUNC definition to p_callback. A template callback function is provided here:

Using a callback function:

```
typedef enum            // defined in r_rtc_rx_if.h
{
    RTC_EVT_ALARM,      // alarm interrupt occurred
    RTC_EVT_PERIODIC    // periodic interrupt occurred
} rtc_cb_evt_t;

                     :
    rtc_init.p_callback = rtc_callback;  // Set a call back function name

    err = R_RTC_Open(&rtc_init, &init_time);    // Initialize the RTC
                     :

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();
    }
}
```

No using a callback function:

```
                     :
    rtc_init.p_callback = FIT_NO_FUNC;  // Set the FIT_NO_FUNC definition
                     :
    err = R_RTC_Open(&rtc_init, &init_time);    // Initialize the RTC
                     :
```

3.4 R_RTC_Close ()

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

Format

void R_RTC_Close (void);

Parameters

No.

Return Values

No.

Properties

Prototyped in file “r_rtc_rx_if.h”

Description

This function stops the RTC counter and disables all RTC interrupts.

Reentrant

No.

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:

No.

3.5 R_RTC_Control ()

This function updates the time & alarm settings, handles capture operation (RX210/RX230/RX231/RX631/RX63N/RX64M/RX65N/RX71M), and provides other configuration and control commands.

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 (command dependent)

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,  // primarily output clock, alarm & capture registers

    /*  RX210, RX230, RX231, RX63N/631, RX64M, RX65N, RX71M only  */
    RTC_CMD_CONFIG_CAPTURE,       // configure capture pin
    RTC_CMD_CHECK_PIN0_CAPTURE,   // if capture event occurred load timestamp
    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	*Open not previously 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)*
RTC_ERR_NO_CAPTURE	*A Capture event did not occur*

Properties

Prototyped in file “r_rtc_rx_if.h”

Description

This function is used primarily for updating the current date/time, setting and clearing alarms, or setting up and processing Capture timestamp events (RX210, RX230, RX231, RX63N/631, RX64M, RX65N, and RX71M). Other operational commands are also available. A brief summary for each command follows.

RTC_CMD_SET_OUTPUT:
The output clock may be disabled or set to an MCU-dependent frequency. This command overrides the setting used in the Open() call. Sample call:

```
rtc_output_t  out_freq=RTC_OUTPUT_OFF;

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

RTC_CMD_SET_PERIODIC:
The periodic interrupt is also typically set during Open(), but can be changed here. It has its own unique configuration structure (see r_rtc_rx_if.h). 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:
The current time is always specified in Open(), but may be changed later (for example, Daylight Savings Time change). The standard tm_t time structure is used here and is detailed in the R_RTC_Open() section. Sample call:

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

RTC_CMD_SET_ALARM_TIME:
This command initializes the RTC registers for use by the alarm facility when it is enabled. The structure for setting alarm time is identical to that used for current time (tm_t). Sample call:

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

RTC_CMD_ENABLE_ALARM:
This command specifies which fields of the tm_t structure specified in RTC_CMD_SET_ALARM_TIME must match the current time for an alarm interrupt to occur. The structure is defined in r_rtc_rx_if.h. 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;
time.tm_min = 0;
time.tm_hour = 9;
time.tm_mday = 1;
time.tm_mon = 0;
time.tm_year = 100;  // minimum legal value
time.tm_wday = 0;
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &time);

alarm.int_priority = 4;
alarm.sec = false;
alarm.min = false;
alarm.hour = true;
alarm.mday = true;
alarm.mon = false;
alarm.year = false;
alarm.wday = false;
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
```

RTC_CMD_STOP_COUNTERS:
Counters are automatically started in Open(). The second argument may be NULL or FIT_NO_PTR. Issuing this command stops the count operation. Sample call:

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

RTC_CMD_START_COUNTERS:
This command is used to resume counting after it is halted by RTC_CMD_STOP_COUNTERS. The second argument may be NULL or FIT_NO_PTR. Sample call:

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

RTC_CMD_PARTIAL_RESET:
This command is used primarily for resetting the Output Clock, and the Alarm and Capture registers (see the RCR2.RESET register bit description in the Hardware User's Manual for a complete list of affected registers). The second argument may be NULL or FIT_NO_PTR. Sample call:

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

RTC_CMD_CONFIG_CAPTURE:
The RTC can be configured such that when a change is detected on the RTCIC0, RTCIC1, or RTCIC2 pins, a snapshot of the date/time is saved to a set of registers. The structure used for configuring these pins is found in r_rtc_rx_if.h. 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 a Capture pin is configured, it must be polled to determine if an event has occurred. These commands return RTC_SUCCESS when a capture was made, RTC_ERR_NO_CAPTURE if not. 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 tamper event occurred on pin 0 */
        if (R_RTC_Control(RTC_CMD_CHECK_PIN0_CAPTURE, &time) == RTC_SUCCESS)
        {
            /* if event occurred outside of 9-5 business hours, show breach */
            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:
After configuring a pin for capture, it may be disabled at a later time. Use RTC_CMD_CONFIG_CAPTURE to enable again. Sample call:

```
    rtc_pin_t  pin=RTC_PIN_0;

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

Reentrant

No.

Example

```
tm_t  g_init_time={0, 0, 0, 1, 0, 100, 0, 0, 0};   // minimum legal values
 :

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

 :
/* "Zero" current time */
err = R_RTC_Open(&rtc_init, &g_init_time);

/* Set alarm for when seconds = 30 */
alm_time = g_init_time;
alm_time.tm_sec = 30;
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &alm_time);

/* Enable alarms for seconds field */
alarm.sec = true;
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
      :
/* Callback function */
void rtc_callback(void *p_args)
{
    rtc_err_t  err;

    // because no periodic interrupts, do not have to check what event occurred

    // reset current time to 0; INT will occur again when seconds = 30 */
    err = R_RTC_Control(RTC_CMD_SET_CURRENT_TIME, &g_init_time);

    // do processing here
}
```

Special Notes:

When using the Capture facility, the corresponding pin must be set for GPIO. Example:

```
    /* Capture 0 on pin P30 */
    PORT3.PDR.BIT.B0 = 0;       // input
    PORT3.PMR.BIT.B0 = 0;       // gpio
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);  // unlock
    MPC.P30PFS.BIT.PSEL = 0;
    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);   // lock
```

3.6 R_RTC_Read ()

This function returns the current time and alarm 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 time.

P_alarm

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

The time structure, tm_t is identical to that found in the standard C header file “time.h” and used in Open() and Control(). The tm_yday and tm_isdst fields are unused for alarm time.

```
Typedef struct
{
    int tm_sec;          // Seconds (0-59)
    int tm_min;          // Minute (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 since 1900
    int tm_wday;         // Day of the week (0-6, 0=Sunday)
    int tm_yday;         // Day of the year (0-365)if specified in config.h
    int tm_isdst;        // Daylight Savings Time; always -1 (unknown)
} tm_t;
```

Return Values

RTC_SUCCESS	
RTC_ERR_NOT_OPENED	*Open not called*

Properties

Prototyped in file “r_rtc_rx_if.h”

Description

This function reads the current time counter and alarm registers. Should a carry occur while reading the counters, an additional read is performed.

Reentrant

Yes.

Example

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

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

Special Notes:

To read the value from the timer counter after return from a reset, deep software standby mode, software standby mode, or the battery backup state, wait for 1/128 second while the clock is operating (RCR2.START bit = 1).

3.7 R_RTC_GetVersion()

This function returns the driver version number at runtime.

Format

uint32_t R_RTC_GetVersion(void);

Parameters

No.

Return Values

Version number.

Properties

Prototyped in file “r_rtc_rx_if.h”

Description

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

Reentrant

Yes

Example

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

Special Notes:

This function is inlined using the “#pragma inline” directive

4. Pin Setting

Regarding the pin setting used for this module, it is strictly recommended to perform the pin setting after calling the function R_RTC_Open.

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_rskrx113

Description

A simple demo of the RX113 Realtime Clock (RTCA) for the RSKRX113 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

RSKRX113

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 RX113 demo above.

Boards Supported

RSKRX231

5.3 rtc_demo_rskrx71m

Description

A simple demo of the RX71M Realtime Clock (RTCd) for the RSKRX71M starter kit (FIT module "r_rtc_rx"). This demo is identical to the RX113 demo above.

Boards Supported

RSKRX71M

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".

Related Technical Updates

This module reflects the content of the following technical updates.

None

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/contact/

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.)

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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0