Abstract
This document describes an example of the real time clock (hereinafter called "RTC") for the RZ/A1H.

Products
RZ/A1H

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

1. Specifications ........................................................................................................................................... 4
2. Operation Confirmation Conditions ........................................................................................................... 5
3. Reference Application Notes .................................................................................................................. 5
4. Peripheral Functions ................................................................................................................................. 6
5. Hardware .................................................................................................................................................. 7
   5.1 Pins Used ........................................................................................................................................... 7
6. Software .................................................................................................................................................... 8
   6.1 Operation Overview ............................................................................................................................... 8
       6.1.1 RTC Initial Setting (Command 1) ................................................................................................. 9
       6.1.2 RTC Time Setting (Command 2) ............................................................................................... 10
       6.1.3 RTC Time Display (Command 3) ............................................................................................. 11
       6.1.4 Starting RTC Time Count Operation (Command 4) .................................................................. 11
       6.1.5 Stopping RTC Time Count Operation (Command 5) ................................................................. 11
       6.1.6 RTC Alarm Time Setting and Transition to Deep Standby Mode (Command 6) ...................... 12
   6.2 Peripheral Functions and Memory Allocation in Sample Code ....................................................... 13
       6.2.1 Setting for Peripheral Functions ............................................................................................... 13
       6.2.2 Section Assignment in Sample Code ....................................................................................... 14
   6.3 Fixed-Width Integers ......................................................................................................................... 17
   6.4 Constants ........................................................................................................................................... 18
   6.5 Structure List ..................................................................................................................................... 19
   6.6 Functions .......................................................................................................................................... 20
   6.7 Function Specifications ....................................................................................................................... 21
   6.8 Flowcharts ....................................................................................................................................... 34
       6.8.1 Initialization of Peripheral Functions ($Sub$main Function) .................................................... 34
       6.8.2 Processing Corresponding to Deep Standby Mode Cancel Source ........................................ 36
       6.8.3 Acquisition Processing for Deep Standby Mode Cancel Time .............................................. 37
       6.8.4 Main Processing ....................................................................................................................... 37
       6.8.5 Processing for Deep Standby Mode Cancel Time Display ....................................................... 38
       6.8.6 Sample Code Main Processing ............................................................................................... 39
       6.8.7 RTC Sample Code Main Processing ....................................................................................... 40
       6.8.8 RTC Initial Setting ..................................................................................................................... 41
       6.8.9 RTC Time Setting ..................................................................................................................... 41
       6.8.10 RTC Time Display .................................................................................................................. 42
       6.8.11 Starting RTC Time Count Operation ....................................................................................... 43
       6.8.12 Stopping RTC Time Count Operation ..................................................................................... 44
       6.8.13 RTC Alarm Time Setting and Transition to Deep Standby Mode ......................................... 45
       6.8.14 RTC Initial Setting .................................................................................................................. 48
       6.8.15 Starting RTC Time Count Operation ....................................................................................... 48
       6.8.16 Stopping RTC Time Count Operation ..................................................................................... 48
       6.8.17 Setting Values to RTC Time Counter ....................................................................................... 49
       6.8.18 Obtaining Values from RTC Time Counter ............................................................................. 51
6.8.19 Setting Values to RTC Alarm Registers ................................................................. 53
6.8.20 Obtaining Values from RTC Alarm Registers ....................................................... 56
6.8.21 RTC Initial Setting ................................................................................................. 59
6.9 Running Sample Code .............................................................................................. 60

7. Sample Code................................................................................................................. 61

8. Reference Documents................................................................................................. 61
1. Specifications

The RTC time (Second, minute, hour, day of the week, day, month, and year) should be set to start the time count operation using RTC. After starting the time count operation, the time is read from RTC and displayed on the terminal on the host PC. Then the RZ/A1H transits to the power-down mode (deep standby mode) and returns from the mode when the RTC alarm interrupt is generated. These basic operations with RTC can be executed by inputting commands from the terminal. Note that the time does not need to be reset after returning from deep standby mode because RTC continues the time count operation during deep standby mode.

In this application note, the serial communication interface with FIFO and the power-down mode are abbreviated as SCIF and STB respectively.

Table 1.1 lists the Peripheral Functions and Their Applications and Figure 1.1 shows the Operation Overview.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC</td>
<td>Used to set and display time using clock/calendar functions.</td>
</tr>
<tr>
<td></td>
<td>Uses alarm interrupt as deep standby mode cancel source.</td>
</tr>
<tr>
<td>SCIF</td>
<td>For communication between SCIF channel 2 and the host PC.</td>
</tr>
<tr>
<td>STB</td>
<td>Used for clock supply to RTC and for transition to and release from deep</td>
</tr>
<tr>
<td></td>
<td>standby mode.</td>
</tr>
</tbody>
</table>

Table 1.1 Peripheral Functions and Their Applications

Figure 1.1 Operation Overview
2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>RZ/A1H</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>CPU clock (Iφ): 400MHz</td>
</tr>
<tr>
<td></td>
<td>Image processing clock (Gφ): 266.67MHz</td>
</tr>
<tr>
<td></td>
<td>Internal bus clock (Bφ): 133.33MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 1 (P1φ): 66.67MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 0 (P0φ): 33.33MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>Power supply voltage (I/O): 3.3V</td>
</tr>
<tr>
<td></td>
<td>Power supply voltage (Internal): 1.18V</td>
</tr>
<tr>
<td>Integrated development</td>
<td>ARM® integrated development environment</td>
</tr>
<tr>
<td>environment</td>
<td>ARM Development Studio 5 (DS-5™) Version 5.16</td>
</tr>
<tr>
<td>C compiler</td>
<td>ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]</td>
</tr>
<tr>
<td></td>
<td>Compiler options (excluding addition of directory path)</td>
</tr>
<tr>
<td></td>
<td>-O3 -Ospace --cpu=Cortex-A9 --littleend --arm --apcs=/interwork</td>
</tr>
<tr>
<td></td>
<td>--no_unaligned_access --fpu=vfpv3_fp16 --g --asm</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Boot mode 0 (CS0-space 16-bit booting)</td>
</tr>
<tr>
<td>Terminal software</td>
<td>Communication speed: 115200bps</td>
</tr>
<tr>
<td>communication setting</td>
<td>Data length: 8 bits</td>
</tr>
<tr>
<td></td>
<td>Parity: None</td>
</tr>
<tr>
<td></td>
<td>Stop bit length: 1 bit</td>
</tr>
<tr>
<td></td>
<td>Flow control: None</td>
</tr>
<tr>
<td>Board used</td>
<td>GENMAI board</td>
</tr>
<tr>
<td></td>
<td>RTK772100BC00000BR (R7S72100 CPU board)</td>
</tr>
<tr>
<td>Device used</td>
<td>NOR flash memory (Connected to CS0 and CS1 spaces)</td>
</tr>
<tr>
<td></td>
<td>Manufacturer: Spansion Inc.</td>
</tr>
<tr>
<td></td>
<td>Part No.: S29GL512S10TFI01</td>
</tr>
<tr>
<td></td>
<td>Serial interface (D-sub 9-pin connector)</td>
</tr>
<tr>
<td></td>
<td>LED1</td>
</tr>
</tbody>
</table>

Note: * The operating frequency used in clock mode 0 (Clock input 13.33MHz from EXTAL pin).

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RZ/A1H Group I/O definition header file <iodefine.h> (R01AN1860EJ)
- RZ/A1H Group Example of Initialization (R01AN1864EJ)
4. Peripheral Functions

This chapter provides supplementary information on RTC. Refer to the "RZ/A1H Group User's Manual: Hardware" for basic information.

RTC has registers (hereinafter collectively called "time counter") which perform BCD coding for second, minute, hour, day of week, day, month, and year to count time. After the time counter has been set and the time count operation starts, the time counter can be used as the information of current time (second, minute, hour, day of week, day, month, and year). Note that the time count operation continues after transition to deep standby mode because the on-chip crystal oscillator circuit does not stop during power-down mode (deep standby mode).

RTC supports the alarm function and generates alarm interrupt with any of or combination of second, minute, hour, day of week, day, month, and year. The interrupt using alarm function can be used as the deep standby mode cancel source. After the alarm time has been set, the RZ/A1H transits to deep standby mode and then returns from it at the alarm time. Note that if the alarm interrupt is used as the deep standby mode cancel source, it runs as the cancel source regardless of the setting value of the running priority register for the interrupt controller. Therefore it is not necessary to enable the alarm interrupt using the alarm interrupt enable flag.
5. Hardware

5.1 Pins Used

Table 5.1 lists the Pins Used and Their Functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A25 to A1</td>
<td>Output</td>
<td>Address signal output to NOR flash memory</td>
</tr>
<tr>
<td>D15 to D0</td>
<td>Input/output</td>
<td>Data signal input/output of NOR flash memory</td>
</tr>
<tr>
<td>CS0#</td>
<td>Output</td>
<td>Device select signal output to NOR flash memory connected to CS0 space</td>
</tr>
<tr>
<td>RD#</td>
<td>Output</td>
<td>Read control signal output to NOR flash memory</td>
</tr>
<tr>
<td>WE0#</td>
<td>Output</td>
<td>Write enable control signal output to NOR flash memory</td>
</tr>
<tr>
<td>MD_BOOT1</td>
<td>Input</td>
<td>Selects boot mode</td>
</tr>
<tr>
<td>MD_BOOT0</td>
<td>Input</td>
<td>MD_BOOT1: &quot;L&quot;, MD_BOOT0: &quot;L&quot; (Set to boot mode 0)</td>
</tr>
<tr>
<td>P4_10</td>
<td>Output</td>
<td>LED on/off</td>
</tr>
<tr>
<td>RxD2</td>
<td>Input</td>
<td>Serial receive data signal</td>
</tr>
<tr>
<td>TxD2</td>
<td>Output</td>
<td>Serial transmit data signal</td>
</tr>
<tr>
<td>RTC_X1</td>
<td>Input</td>
<td>Used to connect a 32.768kHz crystal resonator for RTC</td>
</tr>
<tr>
<td>RTC_X2</td>
<td>Output</td>
<td></td>
</tr>
</tbody>
</table>

Note: The symbol # indicates negative logic (or active low).
6. Software

6.1 Operation Overview

This sample code executes six types of sample operations using RTC. The sample operations can be performed by inputting commands from the terminal. Table 6.1 lists the Sample Operations.

The basic command input procedure to operate sample code is listed as follows.

1. Turn ON the GENMAI board under the operation environment shown in Figure 1.1.
2. Input "1"+"Enter" from the terminal and initialize RTC.
3. Input "2"+"Enter" from the terminal and set the RTC time counter using the time input according to the message displayed on the terminal.
4. Input "4"+"Enter" from the terminal and start the RTC time count operation.
5. Input "3"+"Enter" from the terminal and display the current time on the terminal.
6. Input "6"+"Enter" from the terminal and set the RTC alarm time using the time input according to the message displayed on the terminal. Then transit to deep standby mode.
7. Return from deep standby mode when the value of time counter indicates the alarm time set in step (6) by using the time count operation in deep standby mode.

<table>
<thead>
<tr>
<th>Sample operation</th>
<th>Outline</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC initial setting</td>
<td>Initializes RTC. The RTC time count operation is stopped.</td>
<td>1</td>
</tr>
<tr>
<td>RTC time setting</td>
<td>Sets current time to time counter.</td>
<td>2</td>
</tr>
<tr>
<td>RTC time display</td>
<td>Displays current time using time counter.</td>
<td>3</td>
</tr>
<tr>
<td>Starting RTC time count operation</td>
<td>Starts the RTC time count operation.</td>
<td>4</td>
</tr>
<tr>
<td>Stopping RTC time count operation</td>
<td>Stops the RTC time count operation.</td>
<td>5</td>
</tr>
<tr>
<td>RTC alarm time setting and transition to deep standby mode</td>
<td>Sets RTC alarm time and transits to deep standby mode. Returns from deep standby mode at the alarm time.</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: Command 1 should be executed before executing Commands 2 to 6.
6.1.1 RTC Initial Setting (Command 1)
Set STB to supply a clock to RTC, and initialize RTC after the time count operation is stopped.
When this command has been executed, the RTC time count operation is in stopped state. Note that this command should be executed before executing other commands.
Figure 6.1 shows the Terminal Display for Initial Setting.

```
RTC SAMPLE> 1
RTC initialize complete.
```

Figure 6.1 Terminal Display for Initial Setting
6.1.2 RTC Time Setting (Command 2)
Sets the RTC time counter (Second, minute, hour, day of week, day, month, and year). Input the time by using decimal digit in the order of day of week, month, day, year, hour, minute, and second. Note that no value is set to the RTC time counter if "-1" is input. The time count operation starts with the initial value listed in "RZ/A1H group User's Manual: Hardware" when starting the time count operation without setting time using this command after the GENMAI board turned ON.

Table 6.2 Table of Time Input

<table>
<thead>
<tr>
<th>Time</th>
<th>Value can be Input (Decimal digit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day of week</td>
<td>0 to 6</td>
</tr>
<tr>
<td></td>
<td>0: Sunday, 1: Monday, 2: Tuesday,</td>
</tr>
<tr>
<td></td>
<td>3: Wednesday, 4: Thursday, 5:</td>
</tr>
<tr>
<td></td>
<td>Friday, 6: Saturday</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Month</td>
<td>1 to 12</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Day</td>
<td>1 to 31</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Year</td>
<td>0 to 9999</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Hour</td>
<td>0 to 23</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Minute</td>
<td>0 to 59</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
<tr>
<td>Second</td>
<td>0 to 59</td>
</tr>
<tr>
<td></td>
<td>-1: No change</td>
</tr>
</tbody>
</table>

Figure 6.2 shows the example to set "Tuesday April 1st, 2014 at 10:15" ("Seconds" is not used.)

```
RTC SAMPLE> 2
Enter time (decimal).
Enter "-1" to the item where you do not want to change the time.
Day of week (Sun=0/Mon=1/Tue=2/Wed=3/Thu=4/Fri=5/Sat=6) : 2
Month (1 - 12) : 4
Day (1 - 31) : 1
Year (0 - 9999) : 2014
Hour (0 - 23) : 10
Minute (0 - 59) : 15
Second (0 - 59) : -1
RTC time counter setting complete.
```

Figure 6.2 Example of Time Setting
6.1.3 RTC Time Display (Command 3)
Obtain BCD-coded time information from the RTC time counter and display it on the terminal in the form of "Day of
week month day, year at hour: minute: second".
Figure 6.3 shows the example of "Tuesday April 1st, 2014 at 10:15:45".

```
RTC SAMPLE> 3
---------------------
Tue. Apr. 1, 2014 at 10:15:45
---------------------
```

Figure 6.3 Example of Time Display

6.1.4 Starting RTC Time Count Operation (Command 4)
Start the RTC time count operation. Display the time to start it.
Figure 6.4 shows the Example of Starting of Time Count Operation on "Tuesday April 1st, 2014 at 10:15:10".

```
RTC SAMPLE> 4
---------------------
Tue. Apr. 1, 2014 at 10:15:10
---------------------
RTC time count start.
```

Figure 6.4 Example of Starting of Time Count Operation

6.1.5 Stopping RTC Time Count Operation (Command 5)
Stop the RTC time count operation. Display the time to stop it.
Figure 6.5 shows the Example of Stopping of Time Count Operation on "Tuesday April 1st, 2014 at 10:15:45".

```
RTC SAMPLE> 5
RTC time count stop.
---------------------
Tue. Apr. 1, 2014 at 10:15:45
---------------------
```

Figure 6.5 Example of Stopping of Time Count Operation
6.1.6 RTC Alarm Time Setting and Transition to Deep Standby Mode (Command 6)
The following is performed when executing this command.
(1) Input time to be set to the RTC alarm time from the terminal.
(2) Set the time input in step (1) to the alarm time.
(3) Obtain current time from the RTC time counter and display it on the terminal as transition time to deep standby mode.
(4) Set the deep standby mode cancel source to the alarm interrupt and transit to deep standby mode.
(5) Return from deep standby mode when the value of time counter indicates the alarm time set in step (2) by using time count operation in deep standby mode.
(6) Check the deep standby mode cancel source. When the cancel source is the RTC alarm interrupt, obtain time from the time counter and display it on the terminal as the deep standby mode cancel time.

Figure 6.6 shows the example when the deep standby mode cancel time is "10:20" and transition to deep standby mode on "Tuesday April 1st, 2014 at 10:15:45".

```
RTC SAMPLE> 6
Transit to deep standby mode.
Enter time to cancel deep standby mode (decimal).
Enter "-1" to the item where you set current time.

Hour (0 - 23) : 10
Minute (0 - 59) : 20

Transit to deep standby mode at..
-------------------------------
Tue. Apr.  1, 2014 at 10:15:45
-------------------------------

Deep standby mode will be canceled at 10:20:00.

RZ/A1H CPU Board Sample Program. Ver.X.XX
Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved.

Deep standby mode canceled at..
-------------------------------
Tue. Apr.  1, 2014 at 10:20:00
-------------------------------

select sample program.
```

Figure 6.6 Example of Alarm Time Setting and Transition to Deep Standby Mode

The deep standby mode cancel time is stored in the sample function STB_CancelDeepStandby called by the initial setting function for peripheral function ($Sub$main). The stored time is displayed on the terminal using the sample function RTC_DispTimeCanDeepStb called by the main function.
### 6.2 Peripheral Functions and Memory Allocation in Sample Code

#### 6.2.1 Setting for Peripheral Functions

<table>
<thead>
<tr>
<th>Module</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RTC</strong></td>
<td>Operating clock: Select 32.768kHz from RTC_X1&lt;br&gt;Periodic interrupt generation: Disabled&lt;br&gt;Carry interrupt generation: Disabled&lt;br&gt;Alarm interrupt generation: Disabled</td>
</tr>
<tr>
<td><strong>SCIF</strong></td>
<td>Sets the channel 2 in asynchronous communication mode.&lt;br&gt;- Data length: 8 bits&lt;br&gt;- Stop bit length: 1 bit&lt;br&gt;- Parity: None&lt;br&gt;Sets the clock source without frequency dividing and the bit rate value at 17.&lt;br&gt;Sets the bit rate to be 115200bps when P1φ is 66.67MHz.&lt;br&gt;Difference is 0.46%.</td>
</tr>
<tr>
<td><strong>STB</strong></td>
<td>Clock supply to RTC.&lt;br&gt;Transition to deep standby mode.&lt;br&gt;Deep standby mode is cancelled when the RTC alarm interrupt is generated.&lt;br&gt;Startup from the external memory after deep standby mode is cancelled.</td>
</tr>
</tbody>
</table>
6.2.2 Section Assignment in Sample Code

Table 6.4 and Table 6.5 list the Sections Used in this sample code. Figure 6.7 shows the Section Assignment for the initial condition of the sample code and the condition after using the scatter loading function.

Refer to "Image structure and generation" in "ARM Compiler toolchain Using the Linker" provided by the ARM for more information about the section and the scatter-loading function.

<table>
<thead>
<tr>
<th>Area Name</th>
<th>Description</th>
<th>Type</th>
<th>Loading Area</th>
<th>Execution Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_TABLE</td>
<td>Exception processing vector table</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>RESET_HANDLER</td>
<td>Program code area of reset handler processing</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• INITCA9CACHE (L1 cache setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• INIT_TTB (MMU setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• RESET_HANDLER (Reset handler)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_BASIC_SETUP</td>
<td>Program code area to optimize operating frequency and flash memory</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>InRoot</td>
<td>This area consists of the sections located in the root area such as C standard library.</td>
<td>Code and RO Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_FPU_INIT</td>
<td>Program code area for NEON and VFP initializations</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CODE_FPU_INIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• FPU_INIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_RESET</td>
<td>Program code area for hardware initializations</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CODE_RESET (Startup processing)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• INIT_VBAR (Vector base setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_IO_REGRW</td>
<td>Program code area for read/write function of I/O register</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>CODE</td>
<td>Program code area for defaults</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>All the Code type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONST</td>
<td>Constant data area for defaults</td>
<td>RO Data</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>All the RO Data type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 6.5 Sections to be Used (2/2)

<table>
<thead>
<tr>
<th>Area Name</th>
<th>Description</th>
<th>Type</th>
<th>Load Area</th>
<th>Execution Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_MIRROR_TABLE</td>
<td>Exception processing vector table (Section to transfer data to large-capacity on-chip RAM)</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER_JMPTBL</td>
<td>Program code area for user-defined functions of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER</td>
<td>Program code area of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA_HANDLER_JMPTBL</td>
<td>Registration table data area for user-defined functions of IRQ interrupt handler</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA_RESET</td>
<td>Data area with initial value for hardware initializations</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>BSS_RESET</td>
<td>Data area without initial value for hardware initializations</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_STACK</td>
<td>Application stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>IRQ_STACK</td>
<td>IRQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>FIQ_STACK</td>
<td>FIQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>SVC_STACK</td>
<td>Supervisor (SVC) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ABT_STACK</td>
<td>Abort (ABT) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>TTB</td>
<td>MMU translation table area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_HEAP</td>
<td>Application heap area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA</td>
<td>Data area with initial value for defaults</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>BSS</td>
<td>Data area without initial value for defaults</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
</tbody>
</table>

Notes:
1. "FLASH" and "LRAM" shown in Loading Area and Execution Area indicate the NOR flash memory area and the large-capacity on-chip RAM area respectively.
2. Basically the section name is set to be the same as the region's, however it consists of some sections in the areas of RESET_HANDLER, InRoot, CODE_FPU_INIT, CODE_RESET, CODE, CONST, CODE_HANDLER, DATA, and BSS. Refer to the ARM compiler toolchain manual about the region and the section.
Figure 6.7 Section Assignment
### 6.3 Fixed-Width Integers

Table 6.6 lists the Fixed-Width Integers Used in Sample Code.

**Table 6.6 Fixed-Width Integers Used in Sample Code**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char_t</code></td>
<td>8-bit character</td>
</tr>
<tr>
<td><code>bool_t</code></td>
<td>Boolean type, value: true (1) or false (0)</td>
</tr>
<tr>
<td><code>int_t</code></td>
<td>High-speed integer, signed</td>
</tr>
<tr>
<td></td>
<td>32-bit integer in this sample code</td>
</tr>
<tr>
<td><code>int8_t</code></td>
<td>8-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td><code>int16_t</code></td>
<td>16-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td><code>int32_t</code></td>
<td>32-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td><code>int64_t</code></td>
<td>64-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td>8-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td>16-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>32-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td><code>uint64_t</code></td>
<td>64-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td><code>float32_t</code></td>
<td>32-bit floating point</td>
</tr>
<tr>
<td></td>
<td>(Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td><code>float64_t</code></td>
<td>64-bit floating point</td>
</tr>
<tr>
<td></td>
<td>(Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td><code>float128_t</code></td>
<td>128-bit floating point</td>
</tr>
</tbody>
</table>
## 6.4 Constants

Table 6.7 lists the Constants Used in Sample Code.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_ENABLE</td>
<td>(1)</td>
<td>Enable setting and reading time counter (Second, minute, hour, day of week, day, month, and year)</td>
</tr>
<tr>
<td>RTC_DISABLE</td>
<td>(0)</td>
<td>Disable setting and reading time counter (Second, minute, hour, day of week, day, month, and year)</td>
</tr>
<tr>
<td>RTC_WK_SUNDAY</td>
<td>(0)</td>
<td>Day of week definitions</td>
</tr>
<tr>
<td>RTC_WK_MONDAY</td>
<td>(1)</td>
<td>Sunday</td>
</tr>
<tr>
<td>RTC_WK_TUESDAY</td>
<td>(2)</td>
<td>Monday</td>
</tr>
<tr>
<td>RTC_WK_WEDNESDAY</td>
<td>(3)</td>
<td>Wednesday</td>
</tr>
<tr>
<td>RTC_WK_THURSDAY</td>
<td>(4)</td>
<td>Thursday</td>
</tr>
<tr>
<td>RTC_WK_FRIDAY</td>
<td>(5)</td>
<td>Friday</td>
</tr>
<tr>
<td>RTC_WK_SATURDAY</td>
<td>(6)</td>
<td>Saturday</td>
</tr>
<tr>
<td>RTC_WK_TOTAL</td>
<td>(7)</td>
<td>Total days of week</td>
</tr>
<tr>
<td>STB_GENERATE_ALARM_INT</td>
<td>(1)</td>
<td>The definition in the state where deep standby mode was canceled by RTC alarm interrupt</td>
</tr>
<tr>
<td>STB_NO_GENERATE_ALARM_INT</td>
<td>(0)</td>
<td>The definition in the state where deep standby mode was not canceled by RTC alarm interrupt (Startup from power-on reset also uses this definition)</td>
</tr>
</tbody>
</table>
6.5 Structure List

Figure 6.8 shows the Structure Used in Sample Code.

```c
/* === Type declaration for time setting === */
/* ---- Type declaration for registers with 8-bit width ---- */
typedef struct rtc_8
{
    uint8_t value; /* Setting values for time counter or alarm register */
    uint8_t enable; /* Specification for setting or acquisition */
                /* (RTC_DISABLE : Disable setting or acquisition, */
                /*  RTC_ENABLE  : Enable setting or acquisition) */
} rtc_8_t;

/* ---- Type declaration for registers with 16-bit width */
/* (For year counter and year alarm registers) ---- */
typedef struct rtc_16
{
    uint16_t value; /* Setting values for year counter or year alarm register */
    uint8_t  enable; /* Specification for setting or acquisition */
                /* (RTC_DISABLE : Disable setting or acquisition, */
                /*  RTC_ENABLE  : Enable setting or acquisition) */
} rtc_16_t;

/* ==== Structure declaration for time setting ==== */
typedef struct rtc_time
{
    rtc_8_t  second; /* Second   (0 to 59)  (RTC_DISABLE or RTC_ENABLE)  */
    rtc_8_t  minute; /* Minute   (0 to 59)  (RTC_DISABLE or RTC_ENABLE)  */
    rtc_8_t  hour;  /* Hour     (0 to 23)  (RTC_DISABLE or RTC_ENABLE)  */
    rtc_8_t  week;  /* Day of week (0 to 6)  (RTC_DISABLE or RTC_ENABLE) */
    rtc_8_t  day;   /* Day      (1 to 31)  (RTC_DISABLE or RTC_ENABLE)  */
    rtc_8_t  month; /* Month    (1 to 12)  (RTC_DISABLE or RTC_ENABLE) */
    rtc_16_t year;  /* Year     (0 to 9999) (RTC_DISABLE or RTC_ENABLE) */
} rtc_time_t;

/* ==== Structure declaration for alarm-specified information (ENB bit) ==== */
typedef struct rtc_alarm_enb
{
    uint8_t second; /* Second   (0 or 1) */
    uint8_t minute; /* Minute   (0 or 1) */
    uint8_t hour;  /* Hour     (0 or 1) */
    uint8_t week;  /* Day of week (0 or 1) */
    uint8_t day;   /* Day      (0 or 1) */
    uint8_t month; /* Month    (0 or 1) */
    uint8_t year;  /* Year     (0 or 1) */
} rtc_alarm_enb_t;
```

Figure 6.8  Structure Used in Sample Code
6.6 Functions

The sample code consists of the interface functions to control RTC (API functions), the user-defined functions which need to be prepared by the user for the system applications (functions called from the API functions), and the sample function required to operate the sample code.

Table 6.8, Table 6.9, and Table 6.10 list the Sample Functions, API Functions, and User-Defined Function.

Table 6.8 Sample Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Sub$main</td>
<td>Initialization of on-chip peripheral functions</td>
</tr>
<tr>
<td>(Call $Super$main function, and branch to main function)</td>
<td></td>
</tr>
<tr>
<td>STB_CancelDeepStandby</td>
<td>Processing corresponding to deep standby mode cancel source</td>
</tr>
<tr>
<td>RTC_GetTimeCanDeepStb</td>
<td>Acquisition processing for deep standby mode cancel time</td>
</tr>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>RTC_DispTimeCanDeepStb</td>
<td>Processing for deep standby mode cancel time display</td>
</tr>
<tr>
<td>Sample_Main</td>
<td>Sample code main processing</td>
</tr>
<tr>
<td>Sample_RTC_Main</td>
<td>RTC sample code main processing</td>
</tr>
<tr>
<td>Sample_RTC_Init</td>
<td>RTC initial setting</td>
</tr>
<tr>
<td>Sample_RTC_SetTime</td>
<td>RTC time setting</td>
</tr>
<tr>
<td>Sample_RTC_GetTime</td>
<td>RTC time display</td>
</tr>
<tr>
<td>Sample_RTC_Start</td>
<td>Starting RTC time count operation</td>
</tr>
<tr>
<td>Sample_RTC_Stop</td>
<td>Stopping RTC time count operation</td>
</tr>
<tr>
<td>Sample_RTC_DeepStandby</td>
<td>RTC alarm time setting and transition to deep standby mode</td>
</tr>
</tbody>
</table>

Table 6.9 API Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_RTC_Init</td>
<td>RTC initial setting</td>
</tr>
<tr>
<td>R_RTC_Open</td>
<td>Starting RTC time count operation</td>
</tr>
<tr>
<td>R_RTC_Close</td>
<td>Stopping RTC time count operation</td>
</tr>
<tr>
<td>R_RTC_SetCnt</td>
<td>Setting values to RTC time counter</td>
</tr>
<tr>
<td>R_RTC_GetCnt</td>
<td>Obtaining values from RTC time counter</td>
</tr>
<tr>
<td>R_RTC_SetAlarm</td>
<td>Setting values to RTC alarm registers</td>
</tr>
<tr>
<td>R_RTC_GetAlarm</td>
<td>Obtaining values from RTC alarm registers</td>
</tr>
</tbody>
</table>

Table 6.10 User-Defined Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userdef_RTC_Init</td>
<td>RTC initial setting</td>
</tr>
</tbody>
</table>
### 6.7 Function Specifications

The following tables list the sample code function specifications.

#### $Sub$$main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Initialization of on-chip peripheral functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void $Sub$$main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Executes initial setting for the peripheral functions, and jumps to the main function by calling the $Super$$main of the library function. In the sample code, the processing corresponding to the deep standby mode cancel source is executed by calling the sample function STB_CancelDeepStandby, and the initial settings for STB, PORT, INTC, and L1 cache are performed. The vector address is also set in the on-chip RAM area. The IRQ and the FIQ interrupts are enabled by calling the __enable_irq and __enable_fiq of the library function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

#### STB_CancelDeepStandby

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing corresponding to deep standby mode cancel source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void STB_CancelDeepStandby(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Determine if this function has been called after returning from deep standby mode with reference to the cancel source confirmation flag for the deep standby cancel source flag register (DSFR). If this function has been called after returning from deep standby mode, the processing corresponding to the deep standby mode cancel source should be executed to release the retention of pin state. In the sample code, if the deep standby mode cancel source is the alarm interrupt of the RTC, the RTCARF bit in the deep standby cancel source flag register (DSFR) is cleared to &quot;0&quot;, and each counter value of the RTC is stored in the area retained in the large-capacity on-chip RAM as the time when the deep standby mode is cancelled. The stored time can be obtained from the sample function RTC_GetTimeCanDeepStb. The flag should be set to notify that this function has been called after returning from deep standby mode. Also, when the IOKEEP bit in the deep standby cancel source flag register (DSFR) is &quot;1&quot;, the retention of the pin state can be released by setting the IOKEEP bit to &quot;0&quot;.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>
RTC_GetTimeCanDeepStb

Outline
Acquisition processing for deep standby mode cancel time

Declaration
uint32_t RTC_GetTimeCanDeepStb(rtc_time_t *time)

Description
Stores the time of deep standby mode cancellation in the area specified by the argument time.

Arguments
rtc_time_t *time
  : Time of deep standby mode cancellation
    time->second.value  : Second (0 to 59)
    time->minute.value  : Minute (0 to 59)
    time->hour.value    : Hour (0 to 23)
    time->week.value    : Day of week (0 to 6)
      0: Sunday
      1: Monday
      2: Tuesday
      3: Wednesday
      4: Thursday
      5: Friday
      6: Saturday
    time->day.value     : Day (1 to 31)
    time->month.value   : Month (1 to 12)
    time->year.value    : Year (0 to 9999)

Return Value
STB_GENERATE_ALARM_INT: The state where deep standby mode was canceled by RTC alarm interrupt
STB_NO_GENERATE_ALARM_INT: The state where deep standby mode was not canceled by RTC alarm interrupt or the state started from power-on reset

Note
The sample function STB_CancelDeepStandby is used to store the time when the deep standby mode is cancelled. Therefore, this function should be called after the function STB_CancelDeepStandby is executed.
If this function return STB_NO_GENERATE_ALARM_INT, time of deep standby mode cancellation is not stored to the argument time.
**main**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int_t main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Displays the sample code information on the terminal of the host PC which is connected to the GENMAI board by the serial interface. If this function is called after the deep standby mode is cancelled, displays the time when the deep standby mode is cancelled. Also, executes initial setting for the PORT connected with the LEDs on the board. Executes initial setting for the OSTM channel 0, and sets the timer counter so that the OSTM0 interrupt occurs every 500ms.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>0</td>
</tr>
</tbody>
</table>

**RTC_DispTimeCanDeepStb**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing for deep standby mode cancel time display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void RTC_DispTimeCanDeepStb(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This is a sample function to obtain the time when the deep standby mode is cancelled. In the sample code, the value of the RTC time counter is obtained by using the sample function RTC_GetTimeCanDeepStb. If the time when the deep standby mode is cancelled is stored, the value of the BCD-coded time counter is converted into integer value and display to the terminal.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Note</td>
<td>The RTC time counter values (BCD) are stored as the time when deep standby mode is cancelled by using the sample function STB_CancelDeepStandby.</td>
</tr>
</tbody>
</table>

**Sample_Main**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Sample code main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void Sample_Main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Waits for character input from the terminal running on the host PC connected to the GENMAI board via the serial interface. Activates the RTC sample code when &quot;RTC&quot; + &quot;Enter&quot; key is input.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>
### Sample_RTC_Main

<table>
<thead>
<tr>
<th>Outline</th>
<th>RTC sample code main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>int32_t Sample_RTC_Main(int32_t argc, char_t ** argv)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Waits for character input from the terminal on the host PC connected to GENMAI board by the serial interface. Runs each sample according to the input characters input. When &quot;1&quot;+&quot;Enter&quot; keys are input, execute RTC initial settings. When &quot;2&quot;+&quot;Enter&quot; keys are input, execute RTC time setting. When &quot;3&quot;+&quot;Enter&quot; keys are input, execute RTC time display. When &quot;4&quot;+&quot;Enter&quot; keys are input, execute starting RTC time count operation. When &quot;5&quot;+&quot;Enter&quot; keys are input, execute stopping RTC time count operation. When &quot;6&quot;+&quot;Enter&quot; keys are input, execute RTC alarm time setting and transition to deep standby mode.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>int32_t argc : The number of command arguments input from the terminal. char_t **argv : Pointer to the command input from the terminal.</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>COMMAND_EXIT : Termination of RTC sample code processing</td>
</tr>
</tbody>
</table>

### Sample_RTC_Init

<table>
<thead>
<tr>
<th>Outline</th>
<th>RTC initial setting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>int32_t Sample_RTC_Init(int32_t argc, char_t ** argv)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a sample function to initialize RTC. Called when &quot;1&quot;+&quot;Enter&quot; keys are input during the wait processing for character input using the sample function Sample_RTC_Main. In the sample code, RTC is initialized by using the API function R_RTC_Init.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>int32_t argc : The number of command arguments input from the terminal. Not used in this function. char_t **argv : Pointer to the command input from the terminal. Not used in this function.</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>COMMAND_SUCCESS : Success of RTC sample code processing</td>
</tr>
</tbody>
</table>
### Sample_RTC_SetTime

<table>
<thead>
<tr>
<th>Outline</th>
<th>RTC time setting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>int32_t Sample_RTC_SetTime(int32_t argc, char_t ** argv)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a sample function to set the time to the RTC time counter. Called when &quot;2&quot;+&quot;Enter&quot; keys are input during the wait processing for character input using the sample function Sample_RTC_Main. In the sample code, the time (decimal digit) input from the terminal is set to the RTC time counter using the API function R_RTC_SetCnt. The items for the time should be input in order day of week, month, day, year, hour, minute, and second. Inputs the values of the specified range (Refer to Table 6.2 for effective range for input values). If a value outside the specified range is input, waits until appropriate value is input. When &quot;-1&quot; is input, the item should not be set to the RTC time counter.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>int32_t argc</td>
</tr>
<tr>
<td></td>
<td>char_t **argv</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>COMMAND_SUCCESS</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This sample function runs on the condition that initial settings have been made by using the sample function Sample_RTC_Init.</td>
</tr>
</tbody>
</table>

### Sample_RTC_GetTime

<table>
<thead>
<tr>
<th>Outline</th>
<th>RTC time display</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>int32_t Sample_RTC_GetTime(int32_t argc, char_t ** argv)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a sample function to obtain time from the RTC time counter. Called when &quot;3&quot;+&quot;Enter&quot; keys are input during wait processing for character input using the sample function Sample_RTC_Main. In the sample code, this function displays the time obtained from the RTC time counter to the terminal using the API function R_RTC_GetCnt.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>int32_t argc</td>
</tr>
<tr>
<td></td>
<td>char_t **argv</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>COMMAND_SUCCESS</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This sample function runs on the condition that initial settings have been made by using the sample function Sample_RTC_Init.</td>
</tr>
</tbody>
</table>
### Sample_RTC_Start

**Outline**
Starting RTC time count operation

**Declaration**
```c
int32_t Sample_RTC_Start(int32_t argc, char_t ** argv)
```

**Description**
This is a sample function to start the RTC time count operation. Called when "4"+"Enter" keys are input during wait processing for character input using the sample function Sample_RTC_Main.

In the sample code, this function displays the time obtained from the RTC time counter to the terminal and starts the RTC time count operation using the API function R_RTC_Open.

**Arguments**
- `int32_t argc`: The number of command arguments input from the terminal. Not used in this function.
- `char_t **argv`: Pointer to the command input from the terminal. Not used in this function.

**Return Value**
- `COMMAND_SUCCESS`: Success of RTC sample code processing

**Note**
This sample function runs on the condition that initial settings have been made by using the sample function Sample_RTC_Init.

### Sample_RTC_Stop

**Outline**
Stopping RTC time count operation

**Declaration**
```c
int32_t Sample_RTC_Stop(int32_t argc, char_t ** argv)
```

**Description**
This is a sample function to stop the RTC time count operation. Called when "5"+"Enter" keys are input during wait processing for character input using the sample function Sample_RTC_Main.

In the sample code, this function stops the RTC time count operation using the API function R_RTC_Close, and displays the time obtained from the RTC time counter to the terminal.

**Arguments**
- `int32_t argc`: The number of command arguments input from the terminal. Not used in this function.
- `char_t **argv`: Pointer to the command input from the terminal. Not used in this function.

**Return Value**
- `COMMAND_SUCCESS`: Success of RTC sample code processing

**Note**
This sample function runs on the condition that initial settings have been made by using the sample function Sample_RTC_Init.
Sample_RTC_DeepStandby

Outline
RTC alarm time setting and transition to deep standby mode

Declaration
int32_t Sample_RTC_DeepStandby(int32_t argc, char_t ** argv)

Description
This is a sample processing to transit to deep standby mode after selecting alarm interrupt as deep standby mode cancel source. Called when "6"+"Enter" keys are input during wait processing for character input using the sample function Sample_RTC_Main.

In the sample code, the time values input from the terminal are specified for the hour alarm register (RHRAR) and the minute alarm register (RMINAR), and "0" for the second alarm register (RSECAR) to generate alarm interrupt at the time (hour and minute) input from the terminal. Then the ENB bits of each register are set. Reads the time from the RTC time counter and displays it to the terminal as the time to transit to deep standby mode. Sets the STB so that the cancel deep standby mode is cancelled when the RTC alarm interrupt is generated, and transits to deep standby mode. The STB settings are made to activation through the external memory (NOR flash memory connected to the CS0 space) after the deep standby mode is cancelled.

When both the time (Hour, Minute) input from the terminal is "-1", or when reading time from the RTC time counter and an error occurs, COMMAND_ERROR is returned by this function.

Arguments
int32_t argc : The number of command arguments input from the terminal.
              Not used in this function.
char_t **argv : Pointer to the command input from the terminal.
                Not used in this function.

Return Value
COMMAND_SUCCESS : Success of RTC sample code processing
COMMAND_ERROR  : Failure of RTC sample code processing

Note
This sample function performs initial setting by using the sample function Sample_RTC_Init and runs on the condition that RTC is in the time count operation.
### R_RTC_Init

**Outline**
RTC initial setting

**Declaration**
void R_RTC_Init(void)

**Description**
Initializes the RTC.
Calls the user-defined function Userdef_RTC_Init and initializes the RTC by Userdef_RTC_Init.

**Arguments**
None

**Return Value**
None

### R_RTC_Open

**Outline**
Starting RTC time count operation

**Declaration**
void R_RTC_Open(void)

**Description**
Starts the RTC time count operation.
RTC starts the time count according to the current time of time counter.

**Arguments**
None

**Return Value**
None

**Note**
Set available time to RTC using the API function R_RTC_SetCnt before calling this API function.

### R_RTC_Close

**Outline**
Stopping RTC time count operation

**Declaration**
void R_RTC_Close(void)

**Description**
Stops the RTC time count operation.

**Arguments**
None

**Return Value**
None
R_RTC_SetCnt

### Outline
Setting values to RTC time counter

### Declaration
```c
int32_t R_RTC_SetCnt(rtc_time_t * time)
```

### Description
Sets the time specified by the argument time to RTC time counter. Sets the RESET bit in the control register 2 (RCR2). If RTC_ENABLE is specified for the argument time->second.enable, performs BCD-coding for the time of the argument time->second.value and writes it to the RTC second counter (RSECCNT). If RTC_DISABLE is specified for the argument time->second.enable, writing is not performed. Otherwise, any other member of the time should also be written to the respective RTC time counters.

Because writing to the time counter is disabled while the RTC is in time count operation, suspends the time count operation at the beginning of this function and restarts at the end of this function.

### Arguments
- `rtc_time_t * time`: Time
  - `time->second.value`: Second (0 to 59)
  - `time->minute.value`: Minute (0 to 59)
  - `time->hour.value`: Hour (0 to 23)
  - `time->week.value`: Day of week (0 to 6)
    - 0: Sunday
    - 1: Monday
    - 2: Tuesday
    - 3: Wednesday
    - 4: Thursday
    - 5: Friday
    - 6: Saturday
  - `time->day.value`: Day (1 to 31)
  - `time->month.value`: Month (1 to 12)
  - `time->year.value`: Year (0 to 9999)

### Specification for setting object
(RTC_ENABLE: Do set, RTC_DISABLE: Do NOT set)

- `time->second.enable`: Second counter setting
- `time->minute.enable`: Minute counter setting
- `time->hour.enable`: Hour counter setting
- `time->week.enable`: Day of week counter setting
- `time->day.enable`: Day counter setting
- `time->month.enable`: Month counter setting
- `time->year.enable`: Year counter setting

### Return Value
- **DEVDRV_SUCCESS**: Success in setting value to RTC time counter
- **DEVDRV_ERROR**: Failure in setting value to RTC time counter

### Note
Call the API function `R_RTC_Close` before calling this API function to stop the time count operation.
<table>
<thead>
<tr>
<th>Outline</th>
<th>Obtaining values from RTC time counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int32_t R_RTC_GetCnt(rtc_time_t * time)</td>
</tr>
<tr>
<td>Description</td>
<td>Obtains the time from the RTC time counter and stores it in the area specified by the argument time. If RTC_ENABLE is specified for the argument time-&gt;second.enable, converts the BCD-coded time read from the RTC second counter (RSECCNT) into integer value and stores it in the argument time-&gt;second.value. If RTC_DISABLE is specified for the argument time-&gt;second.enable, readout is not performed. Otherwise, any other member of the time should also be readout from the respective RTC time counters. During the readout processing from the RTC time counter, the carry flag (CF) of the control register 1 (RCR1) is cleared to &quot;0&quot;, the count value from the time counter is read, and the carry flag is verified. If the carry flag is set when reading out from the time counter, the readout is determined to be invalid and the readout processing from the time counter is re-executed. If the carry flag is not set even after the readout processing has been executed twice, this function returns DEVDRV_ERROR.</td>
</tr>
<tr>
<td>Arguments</td>
<td>rtc_time_t * time : Storage area for obtained time</td>
</tr>
<tr>
<td></td>
<td>time-&gt;second.value : Second (0 to 59)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;minute.value : Minute (0 to 59)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;hour.value : Hour (0 to 23)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;week.value : Day of week (0 to 6)</td>
</tr>
<tr>
<td></td>
<td>0: Sunday</td>
</tr>
<tr>
<td></td>
<td>1: Monday</td>
</tr>
<tr>
<td></td>
<td>2: Tuesday</td>
</tr>
<tr>
<td></td>
<td>3: Wednesday</td>
</tr>
<tr>
<td></td>
<td>4: Thursday</td>
</tr>
<tr>
<td></td>
<td>5: Friday</td>
</tr>
<tr>
<td></td>
<td>6: Saturday</td>
</tr>
<tr>
<td></td>
<td>time-&gt;day.value : Day (1 to 31)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;month.value : Month (1 to 12)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;year.value : Year (0 to 9999)</td>
</tr>
<tr>
<td>Specification for obtaining object</td>
<td>(RTC_ENABLE: Do obtain, RTC_DISABLE: Do NOT obtain)</td>
</tr>
<tr>
<td></td>
<td>time-&gt;second.enable : Second counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;minute.enable : Minute counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;hour.enable : Hour counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;week.enable : Day of week counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;day.enable : Day counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;month.enable : Month counter obtaining</td>
</tr>
<tr>
<td></td>
<td>time-&gt;year.enable : Year counter obtaining</td>
</tr>
<tr>
<td>Return Value</td>
<td>DEVDRV_SUCCESS : Success in obtaining value from RTC time counter</td>
</tr>
<tr>
<td></td>
<td>DEVDRV_ERROR : Failure in obtaining value from RTC time counter</td>
</tr>
</tbody>
</table>
R_RTC_SetAlarm

Outline  Setting values to RTC alarm registers

Declaration  int32_t R_RTC_SetAlarm(rtc_time_t * time, rtc_alarm_enb_t * alarm_enb)

Description  Sets the alarm time specified by the argument time to the RTC alarm register. Also, specifies the time (second, minute, hour, day of week, day, month, and year) for alarm to become active using member of the argument alarm_enb. If this function is called in the state of the alarm interrupt enable, disables the alarm interrupt. If RTC_ENABLE is specified for the argument time->second.enable, performs BCD-coding for the alarm time of the argument time->second.value and writes it to the RTC second alarm register (RSECAR). If RTC_DISABLE is specified for the argument time->second.enable, writing is not performed. Otherwise, any other member of the alarm timer should also be written to the respective RTC alarm registers.

Writes the value of the argument alarm_enb->second to the ENB bit in the RTC second alarm register (RSECAR). Other member of the setting information of activating of the alarm time is also written to the ENB bits in the RTC alarm registers. Clears the alarm flag (AF bit in the Control Register 1 (RCR1)) to "0". If this function is called in the state of the alarm interrupt enable, enables the alarm interrupt. When the time of the alarm register in which "1" has been set to the ENB bit matches the time counter, "1" is set to the alarm flag. That means the alarm flag informs that the current time matches the alarm time.

Arguments  rtc_time_t * time : Alarm time
  time->second.value : Second (0 to 59)
  time->minute.value : Minute (0 to 59)
  time->hour.value : Hour (0 to 23)
  time->week.value : Day of week (0 to 6)
    0: Sunday, 1: Monday, 2: Tuesday, 3: Wednesday,
    4: Thursday, 5: Friday, 6: Saturday
  time->day.value : Day (1 to 31)
  time->month.value : Month (1 to 12)
  time->year.value : Year (0 to 9999)

  Specification for setting object
  (RTC_ENABLE: Do set, RTC_DISABLE: Do NOT set)
  time->second.enable : Second alarm setting
  time->minute.enable : Minute alarm setting
  time->hour.enable : Hour alarm setting
  time->week.enable : Day of week alarm setting
  time->day.enable : Day alarm setting
  time->month.enable : Month alarm setting
  time->year.enable : Year alarm setting

rtc_alarm_enb_t * alarm_enb : Setting information of activating of the alarm time

  (0: Activate alarm time, 1: Deactivate alarm time)
  alarm_enb->second : Second (0 or 1)
  alarm_enb->minute : Minute (0 or 1)
  alarm_enb->hour : Hour (0 or 1)
  alarm_enb->week : Day of week (0 or 1)
  alarm_enb->day : Day (0 or 1)
  alarm_enb->month : Month (0 or 1)
  alarm_enb->year : Year (0 or 1)

Return Value  DEVDRV_SUCCESS : Success in setting value to RTC alarm register
DEVDRV_ERROR : Failure in setting value to RTC alarm register
Outline
Obtaining values from RTC alarm registers

Declaration
int32_t R_RTC_GetAlarm(rtc_time_t * time, rtc_alarm_enb_t * alarm_enb)

Description
Obtains the alarm time from the RTC alarm register and stores it in the area specified by the argument time.

If RTC_ENABLE is specified for the argument time->second.enable, converts the BCD-coded alarm time read from the RTC second alarm register (RSECAR) into integer value and stores it in the argument time->second.value. If RTC_DISABLE is specified for the argument time->second.enable, any readout is not performed. Any other member of the alarm time is read out from the respective RTC alarm registers. Reads the setting information of activating of the alarm time from the ENB bit in the second alarm register (RSECAR) and stores it in the argument alarm_enb->second.

Other member of the setting information of activating of the alarm time is also read from the ENB bits in the RTC alarm registers.

Arguments
rtc_time_t * time : Storage area for obtained alarm time
time->second.value : Second (0 to 59)
time->minute.value : Minute (0 to 59)
time->hour.value : Hour (0 to 23)
time->week.value : Day of week (0 to 6)
  0: Sunday
  1: Monday
  2: Tuesday
  3: Wednesday
  4: Thursday
  5: Friday
  6: Saturday
time->day.value : Day (1 to 31)
time->month.value : Month (1 to 12)
time->year.value : Year (0 to 9999)

Specification for obtaining object
(RTC_ENABLE: Do obtain, RTC_DISABLE: Do NOT obtain)
time->second.enable : Second alarm obtaining
time->minute.enable : Minute alarm obtaining
time->hour.enable : Hour alarm obtaining
time->week.enable : Day of week alarm obtaining
time->day.enable : Day alarm obtaining
time->month.enable : Month alarm obtaining
time->year.enable : Year alarm obtaining

rtc_alarm_enb_t * alarm_enb : Storage area for obtained setting information of activating of the alarm time
(alarm_enb->second : Second (0 or 1))
(alarm_enb->minute : Minute (0 or 1))
(alarm_enb->hour : Hour (0 or 1))
(alarm_enb->week : Day of week (0 or 1))
(alarm_enb->day : Day (0 or 1))
(alarm_enb->month : Month (0 or 1))
(alarm_enb->year : Year (0 or 1))

Return Value
DEVDRV_SUCCESS : Success in obtaining value from RTC alarm register
DEVDRV_ERROR : Failure in obtaining value from RTC alarm register
<table>
<thead>
<tr>
<th>Userdef_RTC_Init</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>RTC initial setting</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RTC_Init(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. RTC should be initialized. In the sample code, this function stops the time count operation and disables the carry interrupt, alarm interrupt, and periodic interrupt after the RTC module standby has been cancelled. Selects 32.768kHz from RTC_X1 as an operation clock and sets RTC to operate the on-chip crystal oscillator. The RESET bit in the RTC control register 2 (RCR2) is also set.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
6.8 Flowcharts

6.8.1 Initialization of Peripheral Functions ($Sub$$main Function)

Figure 6.9 and Figure 6.10 show flowchart of Initialization of Peripheral Functions ($Sub$$main Function).

![Flowchart of Initialization of Peripheral Functions](image)

- Initialize STB
  - STB_Init()

- Initialize PORT
  - PORT_Init()

- Initialize BSC in CS1 space
  - R_BSC_Init()

- Initialize BSC in CS3 space
  - R_BSC_Init()

- Initialize BSC in CS2 space
  - R_BSC_Init()

- Initialize INTC
  - R_INTC_Init()

If deep standby mode has been cancelled, the processing corresponding to the deep standby mode cancel source is executed and the retention of pin state is released.

Initialize the STB.
The sample program sets STB2 to STB12 and supplies clocks. Refer to the RZ/A1H Group Example of Initialization application note for details.

Initialize the PORT.
The sample program initializes the PORT to use the NOR flash memory in the CS1, and the SDRAM in CS2 and CS3. Refer to the RZ/A1H Group Example of Initialization application note for details.

Initialize the BSC to use the CS1, CS2, and CS3 spaces. The sample program executes initialization to use the NOR flash memory in area 1, and the SDRAM in area 2 and 3. When using the SDRAM in both these areas, the CS2 space should be initialized after initializing the BSC in the CS3. Refer to the RZ/A1H Group Example of Initialization application note for details.

Initialize the INTC. Refer to the RZ/A1H Group Example of Initialization application note for details.

Figure 6.9 Initialization of Peripheral Functions ($Sub$$main Function) (1/2)
Initialize L1 cache
L1CacheInit()

Set level 1 cache, and enable the instruction cache and the data cache.

Initialize vector base address (VBAR)
VbarInit()

Set the vector base address, and place the exception processing vector table to the on-chip RAM. The exception processing vector table is assigned to the 32 bytes area from the address H'2002 0000.

Call library function__enable_irq()

Enable the IRQ interrupt.

Call library function__enable_fiq()

Enable the FIQ interrupt.

Initialize SCIF channel 2 to connect with the terminal by serial communication
IoInitScif2()

Initialize the SCIF channel 2 to connect with the terminal by the UART communication. Set the baud rate to be 115200bps when P1f is 66.67MHz. Refer to the RZ/A1H Group Example of Initialization application note for details.

Call standard library function$Super$$main()

Call the standard library function $Super$$main. Branch from $Super$$main to the main function.
6.8.2 Processing Corresponding to Deep Standby Mode Cancel Source

Figure 6.11 shows the flowchart of Processing Corresponding to Deep Standby Mode Cancel Source.

```
Figure 6.11 Processing Corresponding to Deep Standby Mode Cancel Source
```

The time structure variable `stb_rtc_time` and 32bit variable `stb_rtc_state` which can be accessed from this function and the sample function `RTC_GetTimeCanDeepStb` is allocated to the on-chip RAM.

1. **Initialize variable which indicates deep standby mode cancel time has been stored**
   - `stb_rtc_state ← STB_NO_GENERATE_ALARM_INT`

2. **Check deep standby mode cancel source flag**
   - **RTCARF bit:**
     - 0 : No realtime clock alarm interrupt generated
     - 1 : Realtime clock alarm interrupt generated
     (Deep standby mode is cancelled by the alarm interrupt.)

3. **Deep standby mode cancel by alarm interrupt?**
   - Deep standby mode is cancelled by alarm interrupt

4. **Clear alarm flag**
   - RCR1 register: AF bit ← 0
   - After setting the AF bit, perform a readout (dummy read) of the RCR1 register.

5. **Clear deep standby mode cancel source flag**
   - DSFR register: RTCARF bit ← 0
   - After setting the RTCARF bit, perform a readout (dummy read) of the DSFR register.

6. **Store the values of RTC time counters in variables which indicate deep standby mode cancel time**
   - `stb_rtc_time.second.value ← Read the RSECCNT register`
   - `stb_rtc_time.minute.value ← Read the RMINCNT register`
   - `stb_rtc_time.hour.value ← Read the RHRCNT register`
   - `stb_rtc_time.day.value ← Read the RDAYCNT register`
   - `stb_rtc_time.month.value ← Read the RMONCNT register`
   - `stb_rtc_time.year.value ← Read the RYRCNT register`

7. **Set variable which indicates deep standby mode cancel time has been stored**
   - `stb_rtc_state ← STB_GENERATE_ALARM_INT`

8. **Check retention of pin state due to transition to deep standby mode**
   - **IOKEEP bit:**
     - 0 : Pin state NOT retained
     - 1 : Pin state retained
     (State when returning from deep standby mode)

9. **Pin state is retained? (IOKEEP = 1?)**
   - Pin state retained

10. **Release retention of pin state due to transition to deep standby mode**
   - DSFR register: IOKEEP bit ← 0
   - After setting the IOKEEP bit, perform a readout (dummy read) of the DSFR register.

11. **return**
6.8.3 Acquisition Processing for Deep Standby Mode Cancel Time

Figure 6.12 shows the flowchart of Acquisition Processing for Deep Standby Mode Cancel Time.

The time when deep standby mode has been cancelled is stored in the variable stb_rtc_state using the sample function STB_CancelDeepStandby, and this function obtain value of stb_rtc_state.

```
return (stb_rtc_state)
```

This function returns with a variable stb_rtc_state stored by the sample function STB_CancelDeepStandby. That stb_rtc_state is "STB_GENERATE_ALARM_INT" means being in the state where deep standby mode was cancelled by alarm interrupt of RTC.

6.8.4 Main Processing

Figure 6.13 shows the flowchart of Main Processing.

Output the sample code version information on the terminal of the host PC which is connected by the serial interface.

```
Output to terminal
printf()
```

When deep standby mode has been cancelled, display the cancel time on the terminal.

```
Processing for deep standby mode cancel time display
RTC_DispTimeCanDeepStb() 
```

Blinks The LED at 500ms intervals using the OSTM channel 0 interrupt. Refer to the RZ/A1H Group Example of Initialization application note for details.

```
OSTM0-related settings
```

Branches to processing that waits to receive a command from the terminal. When "RTC" + "Enter" keys are input, execute RTC sample code.

```
Peripheral function sample code startup function
Sample_Main() 
```

Figure 6.13 Main Processing
6.8.5 Processing for Deep Standby Mode Cancel Time Display

Figure 6.14 shows the flowchart of Processing for Deep Standby Mode Cancel Time Display.

The time when deep standby mode has been cancelled is obtained using the sample function \texttt{RTC\_GetTimeCanDeepStb}, the BCD-coded time counter values are converted into integer values and then displayed on the terminal.

By using the sample function \texttt{STB\_CancelDeepStandby}, the value of the RTC time counter (BCD) are stored as the time when deep standby mode has been cancelled.

```
RTC_DispTimeCanDeepStb

Acquisition processing for deep standby mode cancel time
\texttt{RTC\_GetTimeCanDeepStb()}

Deep standby mode cancel time has been stored?

\begin{itemize}
  \item Stored (Return value of the function is \texttt{STB\_GENERATE\_ALARM\_INT})
  \item Not stored (Return value of the function is \texttt{STB\_NO\_GENERATE\_ALARM\_INT})
\end{itemize}

Convert BCD-coded values for time counter into integer values

Display time on terminal
Display "Day of week Month Day, Year at Hour : Minute : Second" using the time stored in \texttt{time}.
```

Figure 6.14 Processing for Deep Standby Mode Cancel Time Display
6.8.6 Sample Code Main Processing

Figure 6.15 shows the flowchart of Sample Code Main Processing. This function waits for the character input from the terminal running on the host PC.

The RTC sample code is executed when "RTC" + "Enter" keys is input.

Outputs a prompt from the sample code to the terminal of the host PC.

Obtains the main processing command list. The menu list for launching the RTC sample code is obtained in this sample code.

Registers the main processing command list. The menu list for launching the RTC sample code is registered in this sample code.

Wait for command input from the terminal and store it in the command buffer.

Analyzes and executes the contents of the command buffer. In the sample code, this function registers the menu list for RTC sample code processing and branches to the Sample_RTC_Main function, which performs the sample code processing.

Figure 6.15 Sample Code Main Processing
6.8.7_RTC Sample Code Main Processing

Figure 6.16 shows the flowchart of RTC Sample Code Main Processing. This function waits for the character input from the terminal running on the host PC and branches to the RTC sample code processing according to the input command.

- When "1" + "Enter" key is input, execute RTC initial setting.
- When "2" + "Enter" key is input, execute RTC time setting.
- When "3" + "Enter" key is input, execute RTC time display.
- When "4" + "Enter" key is input, execute starting RTC time count operation.
- When "5" + "Enter" key is input, execute stopping RTC time count operation.
- When "6" + "Enter" key is input, execute RTC alarm time setting and transition to deep standby mode.

Sample_RTC_Main

Output to terminal
printf()

Acquisition of command list
of RTC sample processing
Sample_RTC_GetCmdList()

Registration of command list
of RTC sample processing
CommandSetCmdList()

Wait for command input
gets()

Analyze and run command
CommandExe()

"EXIT" was NOT input
"EXIT" input?

"EXIT" was NOT input

"EXIT" was input

return (COMMAND_EXIT)

Output the RTC sample code version information on the terminal on the host PC.

Obtains the RTC sample processing command list. The menu list for launching the RTC sample code is obtained in this sample code.

Registers the RTC sample processing command list. The menu list for launching the RTC sample code is registered in this sample code.

Wait for command input from the terminal and store it in the command buffer.

Analyze and execute the contents of the command buffer. In the sample code, this function branches to the following RTC sample code processing routines, according to the input command.
- "1" : Branch to the Sample_RTC_Init function
- "2" : Branch to the Sample_RTC_SetTime function
- "3" : Branch to the Sample_RTC_GetTime function
- "4" : Branch to the Sample_RTC_Start function
- "5" : Branch to the Sample_RTC_Stop function
- "6" : Branch to the Sample_RTC_DeepStandby function
- "HELP" : Displays a list of available commands

Figure 6.16_RTC Sample Code Main Processing
6.8.8 RTC Initial Setting

Figure 6.17 shows the flowchart of RTC Initial Setting.

This function runs when Command 1 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

```c
Sample_RTC_Init
RTC initial setting
R_RTC_Init()
return (COMMAND_SUCCESS)
```

After cancelling the RTC module standby mode, stop the time count operation and disable the carry interrupt, alarm interrupt, and periodic interrupt.

Select 32.768kHz from RTC_X1 as operating clock and set RTC to operate the on-chip crystal oscillator. The RESET bit in the RTC control register 2 (RCR2) is set.

Figure 6.17 RTC Initial Setting

6.8.9 RTC Time Setting

Figure 6.18 shows the flowchart of RTC Time Setting.

This function runs when Command 2 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

```c
Sample_RTC_SetTime
Obtain time input from terminal
Setting values to RTC time counter
R_RTC_SetCnt()
return (COMMAND_SUCCESS)
```

Wait for the time input from the terminal and store the input values in the time structure variable time declared in this function. Upper and lower limit is given to the values for the time. Wait for re-enter when the values exceed the limit. Store the values in time.xxxx.value and RTC_ENABLE in time.xxxx.enable when the input values are within the limit. If "-1" is input, store RTC_DISABLE in time.xxxx.enable.

<table>
<thead>
<tr>
<th>Day of week</th>
<th>time.week.value</th>
<th>0 to 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>time.month.value</td>
<td>1 to 12</td>
</tr>
<tr>
<td>Day</td>
<td>time.day.value</td>
<td>1 to 31</td>
</tr>
<tr>
<td>Year</td>
<td>time.year.value</td>
<td>0 to 9999</td>
</tr>
<tr>
<td>Hour</td>
<td>time.hour.value</td>
<td>0 to 23</td>
</tr>
<tr>
<td>Minute</td>
<td>time.minute.value</td>
<td>0 to 59</td>
</tr>
<tr>
<td>Second</td>
<td>time.second.value</td>
<td>0 to 59</td>
</tr>
</tbody>
</table>

Set the time store in `time` to the RTC time counter. Note that only the time which has RTC_ENABLE for time.xxxx.enable can be set.

Figure 6.18 RTC Time Setting
6.8.10 RTC Time Display

Figure 6.19 shows the flowchart of RTC Time Display.

This function runs when Command 3 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

Figure 6.19 RTC Time Display

- **Sample_RTC_GetTime**
- **Preparation to obtain values for RTC time counter**
  - Declare time structure variable *time* in this function and store RTC_ENABLE in *time.xxxx.enable* to obtain all values for the RTC time counter.
  - | Day of week | time.week.value | RTC_ENABLE |
  - | Month | time.month.value | RTC_ENABLE |
  - | Day | time.day.value | RTC_ENABLE |
  - | Year | time.year.value | RTC_ENABLE |
  - | Hour | time.hour.value | RTC_ENABLE |
  - | Minute | time.minute.value | RTC_ENABLE |
  - | Second | time.second.value | RTC_ENABLE |

- **Obtaining values from RTC time counter**
  - Store the time obtained from the RTC time counter in *time*.
- **Display obtained time to terminal**
  - Display "Day of week Month Day, Year at Hour : Minute : Second" using the time stored in *time*. 

**return (COMMAND_SUCCESS)**
6.8.11 Starting RTC Time Count Operation

Figure 6.20 shows the flowchart of Starting RTC Time Count Operation.

This function runs when Command 4 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

![Flowchart of Starting RTC Time Count Operation]

- **Sample_RTC_Start**
  - Preparation to obtain values for RTC time counter
  - Obtain values from RTC time counter
    - `R_RTC_Open()`
    - Return (COMMAND_SUCCESS)
  - Display obtained time to terminal
  - Starting RTC time count operation
    - `R_RTC_GetCnt()`
  - Store the time obtained from the RTC time counter in `time`.
  - Display "Day of week Month Day, Year at Hour : Minute : Second" using the time stored in `time`.

<table>
<thead>
<tr>
<th>Day of week</th>
<th><code>time.week.value</code></th>
<th><code>RTC_ENABLE</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td><code>time.month.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.month.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td>Day</td>
<td><code>time.day.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.day.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td>Year</td>
<td><code>time.year.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.year.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td>Hour</td>
<td><code>time.hour.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.hour.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td>Minute</td>
<td><code>time.minute.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.minute.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td>Second</td>
<td><code>time.second.value</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
<tr>
<td></td>
<td><code>time.second.enable</code></td>
<td><code>RTC_ENABLE</code></td>
</tr>
</tbody>
</table>

*Figure 6.20 Starting RTC Time Count Operation*
### 6.8.12 Stopping RTC Time Count Operation

Figure 6.21 shows the flowchart of Stopping RTC Time Count Operation.

This function runs when Command 5 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

- **Sample_RTC_Stop**
  - Stopping RTC time count operation
    - R_RTC_Close()
  - Return (COMMAND_SUCCESS)

- Preparation to obtain values for RTC time counter
  - Declare time structure variable `time` in this function and store RTC_ENABLE in `time.xxxx.enable` to obtain all values for the RTC time counter.

- Obtaining values from RTC time counter
  - R_RTC_GetCnt()

- Display obtained time to terminal
  - Display "Day of week Month Day, Year at Hour : Minute : Second" using the time stored in `time`.

#### Figure 6.21 Stopping RTC Time Count Operation
6.8.13 RTC Alarm Time Setting and Transition to Deep Standby Mode

Figure 6.22 to Figure 6.24 show the flowcharts of RTC Alarm Time Setting and Transition to Deep Standby Mode. This function runs when Command 6 is input during the RTC command wait processing of the sample function Sample_RTC_Main.

RTC has been set to generate an alarm interrupt at the time input from the terminal. It sets STB so that deep standby mode may be cancelled by the alarm interrupt and transits to deep standby mode.

This function is executed on the condition that the RTC time counter is in operation.

![Flowchart](image-url)

Figure 6.22 RTC Alarm Time Setting and Transition to Deep Standby Mode (1/3)
Display obtained time to terminal

Store obtained deep standby mode cancel time in variable for alarm setting

Store ENB bit of the RTC alarm register in variable for setting

Setting values to RTC alarm registers
R_RTC_SetAlarm()

Display time set in alarm register to terminal

Obtaining values from RTC alarm registers
R_RTC_GetAlarm()

Call library function
__disable_irq()

Call library function
__disable_fiq()

Set standby_mode_en bit

Set for operation after deep standby mode cancellation

Display "Day of week Month Day, Year at Hour : Minute : Second" as the time to transit to deep standby mode using the time stored in time.

Store deep standby mode cancel time in variable to set alarm time. In this sample code, store obtained time in time.hour.value and time.minute.value, and store "0" in time.second.value.

Set member of structure alarm_enb for setting information of activating of the alarm time. In this sample code, activate alarm time of hour, minute, and second by storing "1" in alarm_enb.hour, alarm_enb.minute, and alarm_enb.second.

Set the alarm setting time for time and set the setting information of activating of the alarm time for alarm_enb to the RTC alarm register.

Display "Hour : Minute : Second" as the time for deep standby mode cancellation using the time stored in time.

Perform a dummy read of the RTC alarm registers.

Disable the IRQ interrupt.

Disable the FIQ interrupt.

Power Control Register of PL310 ← H'0000 0001, standby_mode_en bit = 1
After setting the Power Control Register of PL310, perform a readout (dummy read) of the register.

DSCTR register ← H'00
EBUSKEEPE bit = 0 : The state of the external memory control pins is not retained when returning from deep standby mode.
RAMBOOT bit = 0 : Method after returning from deep standby mode depends on the boot mode.
After setting the DSCTR register, perform a readout (dummy read) of the register.

Display obtained time to terminal

Display "Day of week Month Day, Year at Hour : Minute : Second" as the time to transit to deep standby mode using the time stored in time.

Figure 6.23  RTC Alarm Time Setting and Transition to Deep Standby Mode (2/3)
Set deep standby mode
cancel source

DSSSR register
RTCAR bit ← 1 : Deep standby mode is canceled by a realtime clock alarm interrupt.
After setting the RTCAR bit, perform a readout (dummy read) of the DSSSR register.

Retain state of output pin in deep standby mode

FRQCR register
CKOEN bit ← 1 : CKIO pin is fixed at low level during deep standby mode.
After setting the CKOEN bit, perform a readout (dummy read) of the FRQCR register.

Clear deep standby mode cancel source flag

DSFR register
RTCARF bit ← 0 : Clear the RTCAR flag
After setting the RTCARF bit, perform a readout (dummy read) of the DSFR register.

Setting for transition to deep standby mode

STBCR1 register ← H'C0
STBY bit = 1
DEEP bit = 1 : Executing WFI instruction puts RZ/A1H into deep standby mode.
After setting the STBCR1 register, perform a readout (dummy read) of the STBCR1 register.

Cancel source flag (RTCARF) has been cleared?

CMNCR register
HIZMEM bit ← 1 : Set the pins of A25 to A0, BS#, RD/WR#, WEn#/DQMxx/AH#, and RD# to drive state in deep standby mode.
HIZCNT bit ← 1 : Set the pins of CKE, RAS#, and CAS# to drive state in deep standby mode.

Flag cleared (RTCARF = 0)

Clear deep standby mode cancel source flag

Setting for interrupt other NMI is not notified to the CPU

ICCICR register ← H'00000000
After setting the ICCICR register, perform a readout (dummy read) of the register.

Call library function __wfi()
6.8.14 RTC Initial Setting
Figure 6.25 shows the flowchart of RTC Initial Setting.

In the sample code, RTC is initialized by using the user-defined function Userdef_RTC_Init.

![Diagram of RTC Initial Setting](image)

6.8.15 Starting RTC Time Count Operation
Figure 6.26 shows the flowchart of Starting RTC Time Count Operation.

Start time count operation
RCR2 register
START bit ← 1 : Second, minute, hour, day, day of week, month, and year counters run normally.

![Diagram of Starting RTC Time Count Operation](image)

6.8.16 Stopping RTC Time Count Operation
Figure 6.27 shows the flowchart of Stopping RTC Time Count Operation.

Stop time count operation
RCR2 register
START bit ← 0 : Second, minute hour, day, day of week, month, and year counters halt.

![Diagram of Stopping RTC Time Count Operation](image)
### Setting Values to RTC Time Counter

Figure 6.28 and Figure 6.29 show the flowcharts of Setting Values to RTC Time Counter.

---

**Figure 6.28 Setting Values to RTC Time Counter (1/2)**

1. **R_RTC_SetCnt**
   - Error in specification by time? → return (DEVDRV_ERROR)

2. **BCD-coding for time**
   - Perform BCD-coding for the time specified by the argument time to write to the time counter and store it in the time structure variable `bcd_value` declared in this function.

3. **Time counter is in operation?**
   - Operation stopped
     - **Stop time count operation**
       - RCR2 register
         - START bit ← 0 : Stop time counter

4. **RTC reset**
   - RCR2 register
     - RESET bit ← 1 : Initializes divider circuit, the R64CNT register, alarm registers, the RCR3 register, bits CF and AF in RCR1, and bit PEF in RCR2.

5. **Setting instruction for Second counter?**
   - With setting instruction (time->second.enable = RTC_ENABLE)
     - **Write to Second counter**
       - RSECCNT register ← `bcd_value.second.value`

6. **Setting instruction for Minute counter?**
   - With setting instruction (time->minute.enable = RTC_ENABLE)
     - **Write to Minute counter**
       - RMINCNT register ← `bcd_value.minute.value`

7. **Setting instruction for Hour counter?**
   - With setting instruction (time->hour.enable = RTC_ENABLE)
     - **Write to Hour counter**
       - RHRCNT register ← `bcd_value.hour.value`
Start time count operation

RCR2 register
START bit ← 1

: The counter is under normal operation

return (DEVDRV_SUCCESS)

Figure 6.29 Setting Values to RTC Time Counter (2/2)
6.8.18 Obtaining Values from RTC Time Counter

Figure 6.30 and Figure 6.31 show the flowcharts of Obtaining Values from RTC Time Counter.

```
R_RTC_GetCnt

Argument error in function?

Error in specification by time

return (DEVDRV_ERROR)

Initialize variable for return value

RCR1 register
CF bit ← 0 : Clear carry flag
AF bit ← 1 : Holds previous value
After setting the RCR1 register, perform a readout (dummy read) for the register.

Clear carry flag

Acquisition instruction for Second counter?

With acquisition instruction (time->second.enable = RTC_ENABLE)

Read from Second counter
Read the RSECCNT register
Store it in bcd_value.second.value

Acquisition instruction for Minute counter?

With acquisition instruction (time->minute.enable = RTC_ENABLE)

Read from Minute counter
Read the RMINCNT register
Store it in bcd_value.minute.value

Acquisition instruction for Hour counter?

With acquisition instruction (time->hour.enable = RTC_ENABLE)

Read from Hour counter
Read the RHRCNT register
Store it in bcd_value.hour.value

Acquisition instruction for Day of week counter?

With acquisition instruction (time->week.enable = RTC_ENABLE)

Read from Day of week counter
Read the RWKCNT register
Store it in bcd_value.week.value

Initialize variable for return value

Set the variable ret which can be used as return value in this function to "DEVDRV_ERROR"

R
A
B
```

Figure 6.30 Obtaining Values from RTC Time Counter (1/2)
A

Acquisition instruction for Day counter?

With acquisition instruction (time->day.enable = RTC_ENABLE)

Read from Day counter
Read the RDAYCNT register
Store it in bcd_value.day.value

Acquisition instruction for Month counter?

With acquisition instruction (time->month.enable = RTC_ENABLE)

Read from Month counter
Read the RMONCNT register
Store it in bcd_value.month.value

Acquisition instruction for Year counter?

With acquisition instruction (time->year.enable = RTC_ENABLE)

Read from Year counter
Read the RYRCNT register
Store it in bcd_value.year.value

Read carry flag

Read the RCR1 register
CF bit :
0: NO carry
1: With carry

Carry occurred?

Carry NOT occurred

First time (Read time again)

Second time (Time obtained from the time counter is invalid)

Second verification of carry?

Set value to variable for return value

Set the variable ret which is used as return value in this function to "DEVDRV_SUCCESS"

Convert BCD-coded values for time counter into integer values

Convert the BCD-coded values for time counter stored in the time structure variable bcd_value into integer values and store them in the area specified by the argument time.

return (ret)

B

Figure 6.31 Obtaining Values from RTC Time Counter (2/2)
6.8.19 Setting Values to RTC Alarm Registers

Figure 6.32 to Figure 6.34 show the flowchart of Setting Values to RTC Alarm Registers.

Figure 6.32 Setting Values to RTC Alarm Registers (1/3)
A

Setting instruction for Day of week alarm register?

With setting instruction (time->week.enable = RTC_ENABLE)

Write to Day of week alarm register

RWKAR register ← bcd_value.week.value

Setting instruction for Day alarm register?

With setting instruction (time->day.enable = RTC_ENABLE)

Write to Day alarm register

RDAYAR register ← bcd_value.day.value

Setting instruction for Month alarm register?

With setting instruction (time->month.enable = RTC_ENABLE)

Write to Month alarm register

RMONAR register ← bcd_value.month.value

Setting instruction for Year alarm register?

With setting instruction (time->year.enable = RTC_ENABLE)

Write to Year alarm register

RYRAR register ← bcd_value.year.value

B

Figure 6.33 Setting Values to RTC Alarm Registers (2/3)
B

Write to ENB bit of Second alarm register
RSECAR register
ENB bit ← alarm_enb->second

Write to ENB bit of Minute alarm register
RMINAR register
ENB bit ← alarm_enb->minute

Write to ENB bit of Hour alarm register
RHRAR register
ENB bit ← alarm_enb->hour

Write to ENB bit of Day of week alarm register
RWKAR register
ENB bit ← alarm_enb->week

Write to ENB bit of Day alarm register
RDAYAR register
ENB bit ← alarm_enb->day

Write to ENB bit of Month alarm register
RMONAR register
ENB bit ← alarm_enb->month

Write to ENB bit of Year alarm register
RCR3 register
ENB bit ← alarm_enb->year

Clear alarm flag
RCR1 register
AF bit ← 0
After setting the AF bit, perform a readout (dummy read) for the RCR1 register.

The alarm interrupt enable when this function called?

Disables the alarm interrupt

Enables the alarm interrupt

Enable alarm interrupt
RCR1 register
AIE bit ← 1 : Enable the alarm interrupt

return (DEVDRV_SUCCESS)

Figure 6.34 Setting Values to RTC Alarm Registers (3/3)
6.8.20 Obtaining Values from RTC Alarm Registers

Figure 6.35 to Figure 6.37 show the flowcharts of Obtaining Values from RTC Alarm Registers.

Figure 6.35 Obtaining Values from RTC Alarm Registers (1/3)
A

Acquisition instruction for
Day of week alarm register?

With acquisition instruction (time->week.enable = RTC_ENABLE)

Read from Day of week
alarm register

Read the RWKAR register
Store it in bcd_value.week.value

Acquisition instruction for
Day alarm register?

With acquisition instruction (time->day.enable = RTC_ENABLE)

Read from Day alarm register

Read the RDAYAR register
Store it in bcd_value.day.value

Acquisition instruction for
Month alarm register?

With acquisition instruction (time->month.enable = RTC_ENABLE)

Read from Month alarm register

Read the RMONAR register
Store it in bcd_value.month.value

Acquisition instruction for
Year alarm register?

With acquisition instruction (time->year.enable = RTC_ENABLE)

Read from Year alarm register

Read the RYRAR register
Store it in bcd_value.year.value

Convert BCD-coded values for
alarm registers into integer values

Convert BCD-coded values for the alarm registers stored in
the time structure variable bcd_value into integer values and
store them in the area specified by the argument time.

B

Figure 6.36 Obtaining Values from RTC Alarm Registers (2/3)
Read ENB bit in Second alarm register

Read the RSECAR register.
Store the value of the ENB bit in alarm_enb->second.

Read ENB bit in Minute alarm register

Read the RMINAR register.
Store the value of the ENB bit in alarm_enb->minute.

Read ENB bit in Hour alarm register

Read the RHRAR register.
Store the value of the ENB bit in alarm_enb->hour.

Read ENB bit in Day of week alarm register

Read the RWKAR register.
Store the value of the ENB bit in alarm_enb->week.

Read ENB bit in Day alarm register

Read the RDAYAR register.
Store the value of the ENB bit in alarm_enb->day.

Read ENB bit in Month alarm register

Read the RMONAR register.
Store the value of the ENB bit in alarm_enb->month.

Read ENB bit in Year alarm register

Read the RCR3 register.
Store the value of the ENB bit in alarm_enb->year.

return (DEVDRV_SUCCESS)

Figure 6.37  Obtaining Values from RTC Alarm Registers (3/3)
6.8.21 RTC Initial Setting

Figure 6.38 shows the flowchart of RTC Initial Setting.

```
Userdef_RTC_Init

Cancel RTC module standby mode
STBCR6 register
MSTP60 bit ← 0 : Supply a clock to RTC
After setting the MSTP60 bit, perform a readout (dummy read) for the bit.

Stop time count operation
RCR2 register
START bit ← 0 : Stop time counter

Disable carry interrupt, alarm interrupt, and periodic interrupt, and clear interrupt flags
RCR1 register
CIE bit ← 0 : Disable the carry interrupt generation
CF bit ← 0 : Clear the carry flag
AIE bit ← 0 : Disable the alarm interrupt generation
AF bit ← 0 : Clear the alarm flag
RCR2 register
PES bit ← 0 : Disable the periodic interrupt generation
PEF bit ← 0 : Clear the periodic interrupt

Select operating clock of time counter
RCR5 register
RCKSEL bit ← 0 : Select 32.768kHz from RTC_X1

Run on-chip crystal oscillator and enable operating clock input
RCR2 register
RTCEN bit ← 1 : Runs the on-chip crystal oscillator / Enables the RTC_X1

RTC Reset
RCR2 register
RESET bit ← 1 : Initializes the divider circuit, the R64CNT register, the alarm register, the RCR3 register, bits CF and AF in RCR1, and bit REF in RCR2.

return
```

Figure 6.38 RTC Initial Setting
6.9 Running Sample Code

The sample code is operated by entering commands in the terminal program running on the host PC connected to the GENMAI board via the serial interface.

After supplying power to the GENMAI board, the message (1) in Figure 6.39 is output. To run the RTC sample code, input "RTC" + "Enter" key subsequent to the "SAMPLE>" prompt. When the message (2) in Figure 6.39 is output, Input "1" to "6" + "Enter" key subsequent to the "RTC SAMPLE>" prompt to run the RTC sample code.

By inputting "HELP" + "Enter" key, the sample code information (3) is displayed. "EXIT" + "Enter" key terminates the RTC sample code operation.

Ver.X.XX and Ver.Y.YY shows in Figure 6.39 indicates the main processing version of the sample code and the RTC sample code version respectively.

<table>
<thead>
<tr>
<th>Display messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>RZ/A1H CPU Board Sample Program. Ver.X.XX (1)</td>
</tr>
<tr>
<td>Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved.</td>
</tr>
<tr>
<td>select sample program.</td>
</tr>
<tr>
<td>SAMPLE&gt;</td>
</tr>
</tbody>
</table>

| RZ/A1H Realtime Clock(RTC) Sample Program. Ver.Y.YY (2) |
| Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved. |
| select sample program. |
| RTC SAMPLE> |

| RTC SAMPLE> help (3) |
| 1 : Initialize RTC |
| 2 : Set time |
| 3 : Get time |
| 4 : Start RTC |
| 5 : Stop RTC |
| 6 : Transition to Deep Standby Mode |
| : -> Canceled by RTC alarm interrupt |
| EXIT : Exit RTC sample |
| RTC SAMPLE> |

Figure 6.39 Terminal Display at RTC Sample Code Startup
7. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

8. Reference Documents

User's Manual: Hardware
RZ/A1H Group User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

The latest version can be downloaded from the Renesas Electronics website.

The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0
The latest version can be downloaded from the ARM website.

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

ARM software development tools (ARM Compiler toolchain, ARM DS-5 etc.) are available on the ARM website.
The latest version can be downloaded from the ARM website.
Website and Support

Renesas Electronics Website
http://www.renesas.com/

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

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev.0.81</td>
<td>Sep. 05, 2014</td>
<td>-</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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 an MPU or MCU 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 this document, 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 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 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 or bodily injury (artificial life support devices or systems, surgical implantation 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 range.

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.

(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.