Introduction

In embedded systems, scheduling of activities to be performed on a periodic basis or after a specific time has elapsed are almost indispensable. For example, a memory refresh mechanism is required to refresh dynamic memory periodically to prevent lost of data. Time management function is one of the modules provided by MR8C/4 that fulfills such needs.

The time management functions include system time management and soft-timer facilities; cyclic handlers and alarm handlers. This application note discusses on the roles, setup procedures and behavior of the time event handlers; cyclic handlers and alarm handlers in MR8C/4.

Target Device

Applicable MCU: R8C Family

Contents

1. Guide in using this Document .............................................................. 2
2. Introduction to MR8C/4 Time Event Handlers .................................................... 3
3. Understanding Cyclic Handlers in MR8C/4 .......................................................... 5
4. Understanding Alarm Handlers in MR8C/4 ......................................................... 15
5. Reference Documents ..................................................................................... 20
1. Guide in using this Document

This document aims to address users’ concerns in the implementation of cyclic handlers and alarm handlers in MR8C/4. With due coverage on the characteristics and narration on the steps of their implementation, users will be able to utilize the time event handlers in MR8C/4 for the R8C Family devices.

Table 1 Explanation of Document Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Objective</th>
<th>Pre-requisite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to MR8C/4 Time Event Handlers</td>
<td>Introduction to MR8C/4 cyclic handlers and alarm handlers</td>
<td>Knowledge in MR8C/4</td>
</tr>
<tr>
<td>Understanding Cyclic Handlers in MR8C/4</td>
<td>Explanation on the components and implementation methods of cyclic handlers</td>
<td>Knowledge in MR8C/4</td>
</tr>
<tr>
<td>Understanding Alarm Handlers in MR8C/4</td>
<td>Explanation on the components and implementation methods of alarm handlers</td>
<td>Knowledge in MR8C/4</td>
</tr>
<tr>
<td>Reference Documents</td>
<td>Listing of documents that equip users with knowledge in the pre-requisite requirements</td>
<td>None</td>
</tr>
</tbody>
</table>
2. Introduction to MR8C/4 Time Event Handlers

Most RTOSs offer an array of timing services entirely based on the system time tick. The timing services may include:

1. Timeout features to prevent task hogging (e.g. waiting for semaphores)
2. Cyclical responses to perform a particular execution repetitively at a specified interval
3. A one short response that perform an execution only once at a specified time (absolute date) or duration (relative time)

MR8C/4 provides the above-mentioned timing services ‘2’ and ‘3’ through its time event handlers; cyclic handlers and alarm handlers.

2.1 Roles of Time Event Handlers

2.1.1 Cyclic Handlers

A cyclical event is one that occurs repeatedly at a predefined interval. A cyclic timer performs the cyclical event as defined by users. It runs for the number of ticks specified, trigger the event and repeats until the timer is explicitly stopped. The cyclic handler is a time event handler in MR8C/4 that functions as a cyclic timer.

The cyclic handler is commonly used as a periodic source of events to “kick” a task into active state to perform some processing and then return to inactive state until the next event. Cyclic handler is useful in eliminating the hassle and inaccurate triggering of events if task is required to re-program the timer for every event. Cyclic handler circumvents these problems by requiring that the task set the timer once.

2.1.2 Alarm Handlers

An alarm handler functions as a one-shot timer that enables users to trigger an event that occurs once at a predefined time relative to its startup instance. An alarm handler will wait for the number of ticks specified, and trigger when the number of ticks is reached. After the trigger, the alarm handler will then move to the non-operational mode.

Alarm handlers have a variety of functions. They can be implemented as a delay timer that places a task in a holding state for a stipulated duration. The handlers can also double as a timeout feature that pull a task out of wait state (e.g. for semaphore) after a period of time.

2.2 Characteristics of Time Event Handlers

The time event handlers in MR8C/4 exhibit the following characteristics:

2.2.1 Time event handler is statistically created.

To reduce complications and increases predictability, MR8C/4 only provides ability to statically create time event handler objects. Users are required to define the time event handler objects in the configuration file (e.g. template.cfg).
When a time event handler object is created during system initialization, it is in non-operational mode. For time event handler to be triggered, users are required to place the time event handler object in operational mode by issuing “sta_cyc” (for cyclic handlers) or “sta_alm” (for alarm handlers) service calls. Once the event handler object is in operational mode, its predefined interval/time value will start counting down and triggered once it reaches zero count.

2.2.2 Time event handlers execute in their own independent contexts (non-task contexts)

Time event handlers are not executed within the tasks in which the handlers are triggered or started (i.e. task in which “sta_cyc” or “sta_alarm” is issued).

Time event handlers are executed within its own context, termed non-task contexts. Thus, only non-task context service calls (e.g. “iset_flg”) can be invoked within the handlers. Task scheduling is also not possible when executing the handlers. In addition, time event handlers operate within the system stack.

2.2.3 Time event handlers execute at lower precedence than the interrupt, but at higher precedence than the dispatcher

The precedence for execution of each processing unit in MR8C/4 is as follow:

1. Interrupt Handler
2. Time event handler (i.e. cyclic handlers and alarm handlers)
3. Dispatcher
4. Task
2.2.4 For time event handlers to be triggered, the system must be in the CPU unlocked state

A system’s CPU can either be in locked or unlocked state. When the CPU is in the locked state, kernel interrupts handlers; time event handlers and dispatching will be blocked and does not occur. Therefore, to ensure time event handlers can be triggered, system must be kept in CPU unlocked state.

2.3 Areas of Concern

There are two main areas of concern when using time event handlers.

The first concern is in regards to the accuracy of time event handlers. A user will want a time event handler to be triggered according to the timing he/she had specified. For example, if a cyclic handler is scheduled to be triggered 10 seconds after a particular event, it should be activated exactly to the scheduled time. However, in a generic preemptive RTOS, a higher-priority process might have been scheduled to run and preempted long enough that the specified timeout will have expired. In contrast to RTOS that does not have cyclic handler, MR8C/4 provides cyclic handlers that has higher precedence over task dispatcher that avoids the occurrence of such a problem.

The second concern involves the resolution on time setting of the handlers. Certain circumstances might arise that require users to set a time event handler to occur within a time interval ranging from few milliseconds to a couple of hours. The time setting of the handler is dependent on the resolution of the system timer that in turn depends on the MCU device selected. For example, if the system timer provide ticks only at a resolution in seconds, it will not be possible to set the cyclic handler to occur at an interval of less than a second. For R8C family devices, this concern is less critical as the devices timers can provide timing resolution up to 1 millisecond.

3. Understanding Cyclic Handlers in MR8C/4

Cyclic handler function belongs to the time management module in MR8C/4. Users are able to create up to 255 cyclic handlers statically and selectively trigger the handlers by specifying their activation time.

To use cyclic handler function, MR8C/4 kernel requires one timer to be used as the system clock. The system clock provides the time tick for the operation of the cyclic handler. Therefore, timing accuracy for triggering of cyclic handlers is dependent on the resolution of time tick.

If the resolution of system timer is larger than activation cycle of cyclic handler, the cyclic handler will not be triggered at the start of every activation cycle. Figure 2 illustrates such a scenario.

![Figure 2 Operating Cycle of Cyclic Handler with Coarser Time Tick](image-url)
### 3.1 Components of Cyclic Handler

In MR8C/4, cyclic handler functions include the ability to create a cyclic handler, to start and stop a cyclic handler’s operation. A cyclic handler object comprises of the following components:

- ID number
- Activation cycle
- Activation phase
- Extended information
- Attribute

#### 3.1.1 ID Number

In MR8C/4, users are able to specify up to 255 cyclic handlers. An individual cyclic handler is an object identified by an ID number. Each cyclic handler has a unique ID number that is also called the cyclic handler ID.

```c
    cyclic_hand[1]{
        entry_address = WatchUpdate_Cyc1();
        name = ID_Cycl_WatchUpdate;
        exinf = 0x0;
        start = OFF;
        phsatr = OFF;
        interval_counter = 0x3e8; //1sec;
        phs_counter = 0x0;
    }
```

Figure 3 Defining a Cyclic Handler Object in Configuration File (ID Number)

The MR8C/4 configurator is capable of automatically assign ID number to the cyclic handler object if user did not specify its ID number. Assignment of ID numbers by the configuration is based on the declaration sequence in the configuration file (refer to Figure 4).
cyclic_hand[1][
    entry_address = AlarmUpdate_Cyc2();
    name = ID_Cyc2_AlarmUpdate;
    exinf = 0x0;
    start = OFF;
    phsstr = OFF;
    interval_counter = 0x1F4; //500msec;
    phs_counter = 0x0; //0msec;
];

cyclic_hand[3][
    entry_address = StopWatchCount_Cyc3();
    name = ID_Cyc3_StopWatchCount;
    exinf = 0x0;
    start = OFF;
    phsstr = ON;
    interval_counter = 0x0A; //10msec;
    phs_counter = 0x0; //0msec;
];

cyclic_hand[2][
    entry_address = WatchUpdate_Cyc1();
    name = ID_Cyc1_WatchUpdate;
    exinf = 0x0;
    start = OFF;
    phsstr = ON;
    interval_counter = 4000; //1sec;
    phs_counter = 6000;
];

cyclic_hand[4][
    entry_address = LEDFlicker_Cyc4();
    name = ID_Cyc4_LEDFlicker;
    exinf = 0x0;
    start = OFF;
    phsstr = ON;
    interval_counter = 0x3e8; //1sec;
    phs_counter = 0x0; //0msec;
];

Notes: 1. ID_Cyc2_AlarmUpdate is the first cyclic handler defined and ID number '1' is not defined. Thus ID number ‘1’ is assigned to this cyclic handler.
2. ID number ‘3’ is defined for ID_Cyc3_StopWatchCount. Thus ID number ‘3’ is assigned to this cyclic handler.
3. ID_Cyc1_WatchUpdate is the third cyclic handler defined but ID number ‘3’ is already defined. Thus ID number ‘2’ is assigned to this cyclic handler.
4. ID number ‘4’ is defined for ID_Cyc4_LEDFlicker. Thus ID number ‘4’ is assigned to this cyclic handler.

Figure 4 Defining Cyclic Handler Objects in Configuration File with No ID Number
3.1.2 Activation cycle

Activation cycle (Figure 5) defines the periodic interval at which the cyclic handler will be activated. The numeric value for this parameter range from 1 to 0x7FFFFFFF (which is equivalent to 1millisecond to ~24 days). To define a cyclic handler that activates at 2-second intervals, activation cycle parameter must be set to 0x7D0(hex) or 2000(decimal).

```c
    cyclic_hand[1][
        entry_address = WatchUpdate_Cyc10;
        name = ID_Cyc1_WatchUpdate;
        extinf = 0x0;
        start = OFF;
        phsatr = OFF;
        interval_counter = 0x3e8; //1sec;
        phs_counter = 0x0;
    ];
```

**Figure 5 Defining a Cyclic Handler Object in Configuration File (Activation Cycle)**

Figure 6 shows how the parameter “activation cycle” plays a part in the operation of a cyclic handler object.

![Figure 6 Cyclic Handler Operations](image)

3.1.3 Activation Phase

Activation phase indicates the startup time of a cyclic handler. In MR8C/4, cyclic handler is created statically (when system initializes). Therefore, activation phase is the relative duration from the time the system initializes to the startup of the first cyclic handler activation cycle. Activation phase determines the first countdown of activation cycle and thereby effectively dictates the startup time of the cyclic handlers.

Figure 7 depicts the scenario on the creation to the activation of a cyclic handler. With reference to Figure 7, cyclic handler is in non-operational state from the instance it is created to the time before “sta_cyc” is issued.
Notes: 1. User defines cyclic handler in configuration file.
    Cyclic handler is created upon system initialization.
2. Activation phase starts upon creation of cyclic handler.
3. “sta_cyc” API is required to be issued to activate the cyclic handler.
   Cyclic handler changes from non-operational to operational state.
   Cyclic handler will be triggered at the start of the next activation cycle.
4. The 1st activation cycle always starts counting immediately after the activation phase.
5. Cyclic handler is called at startpoint of next activation cycle immediately after “sta_cyc” is issued.

Figure 7 Cyclic Handler Creation and Activation

It is not necessary for “sta_cyc” service call to be issued after the commencement of the 1st activation cycle. Sometimes, cyclic handler can be placed into operational mode during the activation phase. Figure 8 shows the behavior of a cyclic handler when the activation phase defined is greater than activation cycle.

Figure 8 Cyclic Handler Operation when Activation Phase Greater than Activation Cycle

3.1.4 Extended Information

Extended information (16 bits) is added in uTRON 4.0 specifications to allow users to include additional information about the cyclic handler attributes. If user wishes to change the contents of this information, the user should allocate memory area and set the address of the memory packet to exinf, as it cannot be read by any MR8C/4 service calls.
In MR8C/4 (uITRON 4.0 specification), it is optional to provide extended information. Extended information is specified in the configuration file and passed as a parameter when the cyclic handler object starts to execute. It does not have any effects on the operation of the kernel or software component.

```c
    cyclic_handler[1] {
        entry_address = WatchUpdate_Cyc1();
        name = ID_Cyc1_WatchUpdate;
        exinf = 0x53;
        start = OFF;
        phsatr = OFF;
        interval_counter = 0x3e8; //1sec;
        phs_counter = 0x0;
    };
```

**Figure 9 Defining a Cyclic Handler Object in Configuration File (Extended Information)**

### 3.1.5 Attribute

Each cyclic handler object has four object attributes that define its operational mode, initiate state and the language the handler is written in. Following are four attributes of a cyclic handler:

- **TA_HLNG (=0x00):** Starts cyclic handler through a high-level language interface
- **TA_ASM (=0x01):** Starts cyclic Handler through an assembly language interface
- **TA_STA (=0x02):** Place cyclic handler in operational mode
- **TA_PHS (=0x03):** Preserve activation phase

Notes:
1. "entry_address" defined with open-close brackets represents TA_HLNG is specified. "entry_address" not defined with open-close brackets represents TA_ASM is specified.
2. "ON" represents TA_STA attribute is specified.
   Cyclic handler is placed in operational mode (cyclic counter starts counting) when it is created upon system initialization.
   "OFF" represents TA_STA attribute is not specified.
   Cyclic handler remains in non-operational mode (cyclic counter does not count) when it is created upon system initialization.
3. "ON" represents TA_PHS attribute is specified.
   Cyclic handler is placed in operational mode immediately when "sta_cyc" is issued.
   "OFF" represents TA_PHS attribute is not specified.
   Cyclic handler is triggered immediately when "sta_cyc" is issued.

**Figure 10 Defining a Cyclic Handler Object in Configuration File (Attribute)**
Figure 11 illustrates the difference on the trigger time of cyclic handler when TA_PHS attribute is specified and not specified. As can be seen in the illustration, when TA_PHS is specified, activation phase is taken into calculation and cycle handler is triggered at the start of 3rd activation cycle without resetting 2nd activation cycle. When TA_PHS is not specified, activation phase does not determine the 1st trigger of the cyclic handler.

Figure 11 Cyclic Handler Operation with/without TA_PHS Specified
3.2 Cyclic Handler Service Calls

MR8C/4 provides two simple cyclic handler function service calls, namely, “sta_cyc” and “stp_cyc”. Both service calls can only be invoked in the task context and CPU-unlocked system states.

“sta_cyc” service call places a cyclic handler from non-operational to operational state. “sta_cyc” has the function characteristics as shown in Figure 13.

<table>
<thead>
<tr>
<th>TA_PHI</th>
<th>TA_STA</th>
<th>Cyclic Handler State before “sta_cyc” invoked</th>
<th>&quot;sta_cyc” Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>Non-operational</td>
<td>Place cyclic handler in operational mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cyclic handler will be triggered one activation cycle after “sta_cyc” invoked</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>Non-operational</td>
<td>Place cyclic handler in operational mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“sta_cyc” has no effect on cyclic handler’s trigger time</td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td>Operational</td>
<td>Cyclic handler trigger time recalculate. Cyclic handler function will be triggered one activation cycle after “sta_cyc” is invoked</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>Operational</td>
<td>No effect</td>
</tr>
</tbody>
</table>

Figure 13 Service call “sta_cyc” Functional Characteristics
“stp_cyc” service call performs the inverse operation of “sta_cyc” (refer to Figure 14. It places a cyclic handler from operational to operational state. For “stp_cyc” to be effective, users are required to ensure system CPU is in CPU-unlocked state when invoking the service call.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sta_cyc</td>
<td>Place cyclic handler in operational state. The instance the cyclic handler starts is dependent on the TA_PHS setting.</td>
</tr>
<tr>
<td>stp_cyc</td>
<td>Place cyclic handler in non-operational state</td>
</tr>
</tbody>
</table>

**Figure 14 Cyclic Handler Service Calls**

### 3.3 Implementing Cyclic Handler

There are three main steps to setup a cyclic handler in MR8C/4. They are:

1. Define cyclic handler object in configuration file (refer Figure 15)
2. Define cyclic handler routine function (refer Figure 16)
3. Implement cyclic handler service calls (refer Figure 17)

**Figure 15 Defining Cyclic Handler Object in Configuration File (e.g. Template.cfg)**
R8C Family Implementing Time Event Handlers in MR8C/4

Figure 16 Defining Cyclic Handler Routine Function

Figure 17 Implement Cyclic Handler Service Calls
4. Understanding Alarm Handlers in MR8C/4

4.1 Components of Alarm Handler

In MR8C/4, alarm handler functions include the ability to create an alarm handler, to start and stop an alarm handler’s operation. An alarm handler object comprises of the following components:

- ID number
- Activation time
- Extended information
- Extended Attribute

4.1.1 ID Number

MR8C/4 supports up to 255 alarm handlers. An individual alarm handler is identified by its ID number. Shows the definition of an alarm handler ID number.

```c
alarm_handler[11] =
entry_address = AlarmSetup_Alm1();
name = ID_Alm1;
exinf = 0x0;
};
```

Figure 18 Defining an Alarm Handler Object in Configuration File (ID Number)

The MR8C/4 configurator is capable of automatically assign ID number to the alarm handler object if user did not specify its ID number. Assignment of ID numbers by the configurator is based on the declaration sequence in the configuration file as explained for cyclic handlers in sections 3.1.1.

4.1.2 Activation time

This parameter is defined dynamically during program runtime through the service call “sta_alm”. Activation time is a relative duration between the invoke of “sta_alm” and triggering of the alarm handler function. It is measured in milliseconds. The numeric value for this parameter range from 1 to 0x7FFFFFFF (which is equivalent to 1 millisecond to ~24 days). To define an alarm handler to trigger in 3 seconds, activation time must be set to 0xBB8(hex) or 3000(decimal).

```c
ercd = sta_alm(ID_Alm1,3000);
```

Figure 19 Defining Activation Time of Alarm Handler Object

Figure 20 illustrates the effect of “activation time” on the operation of an alarm handler object.
4.1.3 Extended Information
A 16-bit extended information is also provided for users to include additional information about the alarm handler attributes. Likewise for cyclic handler, it is optional for users to provide extended information for alarm handler.

![Figure 21 Defining a Alarm Handler Object in Configuration File (Extended Information)]

4.1.4 Attribute
Contrary to a cyclic handler, each alarm handler object has only two object attributes that define the language the handler is written in. Following are two attributes of an alarm handler:

- **TA_HLNG (=0x00)**: Starts alarm handler through a high-level language interface
- **TA_ASM (=0x01)**: Starts alarm handler through an assembly language interface

```c
alarm_hand[1]{
    entry_address = Alm1();
    name = ID_Alm1;
    exinf = 0x0;
}
```

**Notes:** 1. “entry_address” defined with open-close brackets represents TA_HLNG is specified.
   “entry_address” not defined with open-close brackets represents TA_ASM is specified.

![Figure 22 Defining an Alarm Handler Object in Configuration File (Attribute)]
4.2 Alarm Handler Service Calls

MR8C/4 provides two service calls “sta_alm” and “stp_alm” to start and stop alarm handler respectively. Both service calls can only be invoked in the task context and CPU-unlocked system states. A summary of both service calls and their respective features are provided in Figure 24.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sta_alm</td>
<td>Place alarm handler in operational state and set activation time</td>
</tr>
<tr>
<td>stp_alm</td>
<td>Place alarm handler in non-operational state and releases its activation time</td>
</tr>
</tbody>
</table>

4.3 Implementing Alarm Handler in MR8C/4

An alarm handler can be setup as follows:

1. Define alarm handler object in configuration file (refer Figure 25)
2. Define alarm handler routine function (refer Figure 26)
3. Implement alarm handler service calls (refer Figure 27)
Figure 25 Defining Alarm Handler Object in Configuration File (e.g. Template.cfg)

```c
alarm_handler[]{
    entry_address = AlarmSetup_A11();
    name = "ID_A11"
    exinf = 0x00;
};
```

Figure 26 Defining Alarm Handler Routine Function

```c
/Header***********************************************************************************************
Name: AlarmSetup_A11
Description: Alarm Handler for ID_A11 (Activated based on user setting)
Parameters: none
Returns: none
*************************************************************************/
/void AlarmSetup_A11(WP_INT std)
  
  ercd = imap_task(ID_Task6,StartAlarm); /* Change Task8 from WAITING to RUNNING state */
```
Figure 27 Implement Alarm Handler Service Calls

```c
errno = ststm(ID_Alarm1,calltime+10000); // Activate ID_Alarm1 alarm handler 10sec #/ 1
errno = ststm(ID_Alarm1); // Do not activate alarm
```

Service calls to place alarm handler “ID_Alarm1” in operational/non-operational state.
5. Reference Documents

User’s Manual

- MR8C/4 V1.00 User's Manual
- R8C Family Hardware Manual

The latest version can be downloaded from the Renesas Technology website
Website and Support

Renesas Technology Website

• http://www.renesas.com/

Inquiries

• http://www.renesas.com/inquiry

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>March.01.10</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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   - Handle unused pins in accord 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 one with a different type number, confirm that the change will not lead to problems.
     - The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to
Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties or by 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.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software, and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When aspiring the 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. You should not use Renesas Electronics products or the 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. 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.

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the products quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" and for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard", unless otherwise expressly specified in a Renesas Electronics data sheet or data books, etc.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation, and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. 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 system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. 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 damage or loss occurring as a result of your noncompliance with applicable laws and regulations.

11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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