R8C Family

Implementing Time Event Handlers in MR8C/4

R20AN0089ES0100
Rev.1.00
Mar 01, 2010

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

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

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

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.

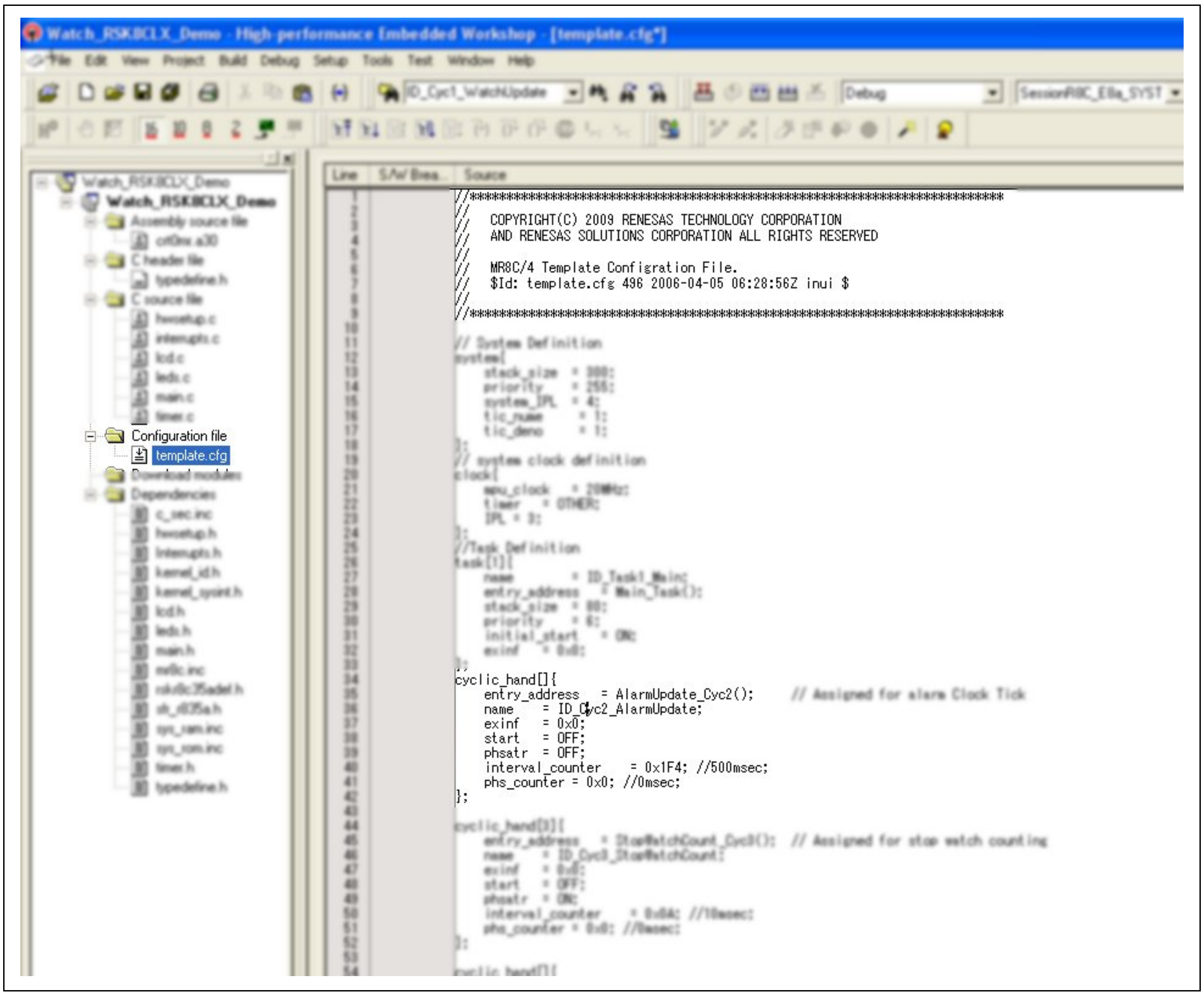
Alarm handlers have a variety of functions. They can be implemented as a delay timer that places a task in a holding state for a stipulated duration. The handlers can also double as a timeout feature that pull a task out of wait state (e.g. for semaphore) after a period of time.

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).



```
//****************************************************************************
//
//   COPYRIGHT(C) 2009 RENESAS TECHNOLOGY CORPORATION
//   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
//
//   MR8C/4 Template Configration File.
//   $Id: template.cfg 496 2006-04-05 06:28:56Z inui $
//
//****************************************************************************

// System Definition
system{
    stack_size  = 300;
    priority    = 255;
    system_IPL  = 4;
    tic_nume     = 1;
    tic_deno     = 1;
};
// system clock definition
clock{
    mpu_clock   = 20MHz;
    timer   = OTHER;
    IPL = 3;
};
//Task Definition
task[1]{
    name        = ID_Task1_Main;
    entry_address   = Main_Task();
    stack_size  = 80;
    priority    = 6;
    initial_start   = ON;
    exinf   = 0x0;
};
cyclic_hand[]{
    entry_address   = AlarmUpdate_Cyc2();     // Assigned for alarm Clock Tick
    name    = ID_Cyc2_AlarmUpdate;
    exinf   = 0x0;
    start   = OFF;
    phsatr  = OFF;
    interval_counter    = 0x1F4; //500msec;
    phs_counter = 0x0; //0msec;
};

cyclic_hand[3]{
    entry_address   = StopWatchCount_Cyc3();  // Assigned for stop watch counting
    name    = ID_Cyc3_StopWatchCount;
    exinf   = 0x0;
    start   = OFF;
    phsatr  = ON;
    interval_counter    = 0x0A; //10msec;
    phs_counter = 0x0; //0msec;
};

cyclic_hand[]{
```

Figure 1 Defining Cyclic Handler Objects in Configuration File

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 starts 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.

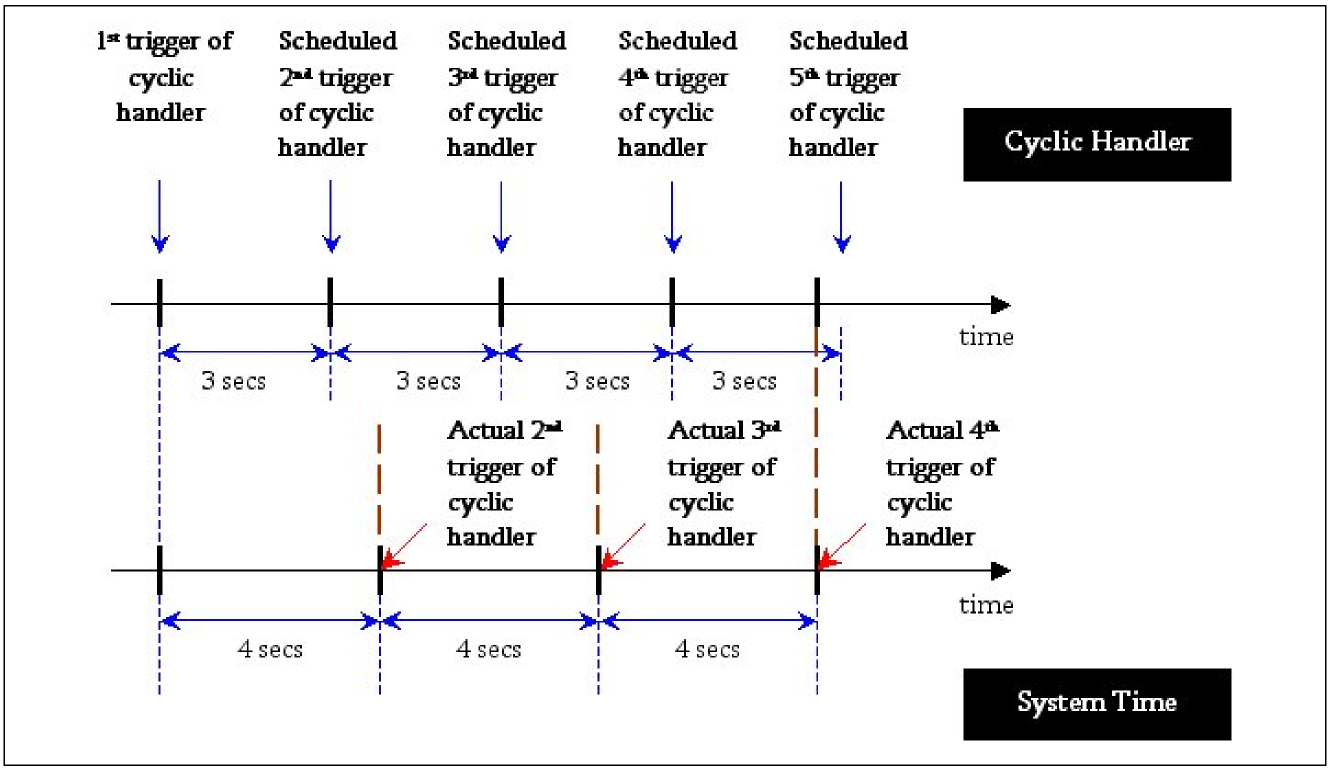

1st trigger of cyclic handler
Scheduled 2nd trigger of cyclic handler
Scheduled 3rd trigger of cyclic handler
Scheduled 4th trigger of cyclic handler
Scheduled 5th trigger of cyclic handler
Cyclic Handler
time
3 secs
3 secs
3 secs
3 secs
Actual 2nd trigger of cyclic handler
Actual 3rd trigger of cyclic handler
Actual 4th trigger of cyclic handler
time
4 secs
4 secs
4 secs
System Time

Figure 2 Operating Cycle of Cyclic Handler with Coarser Time Tick

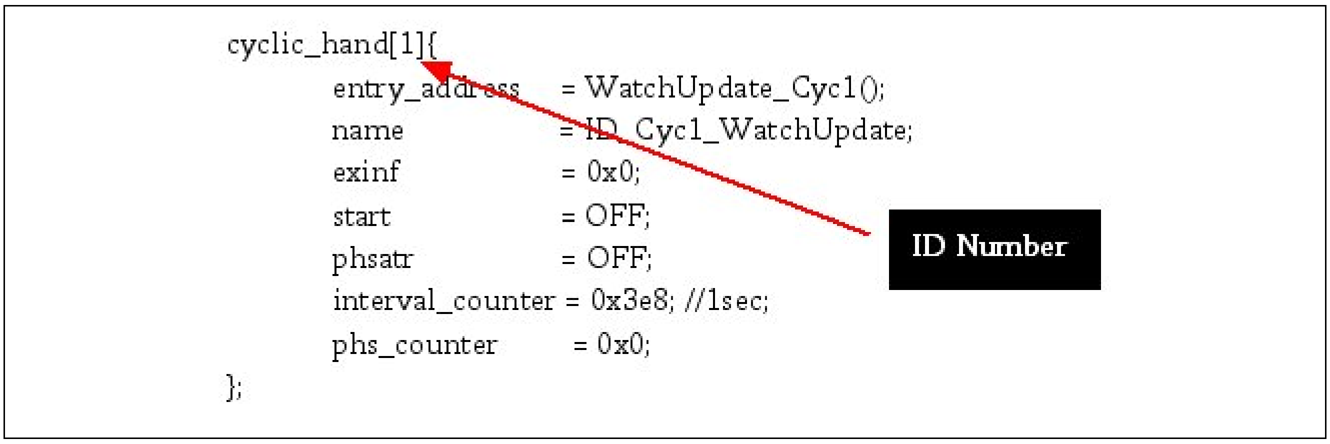
3.1 Components of Cyclic Handler

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

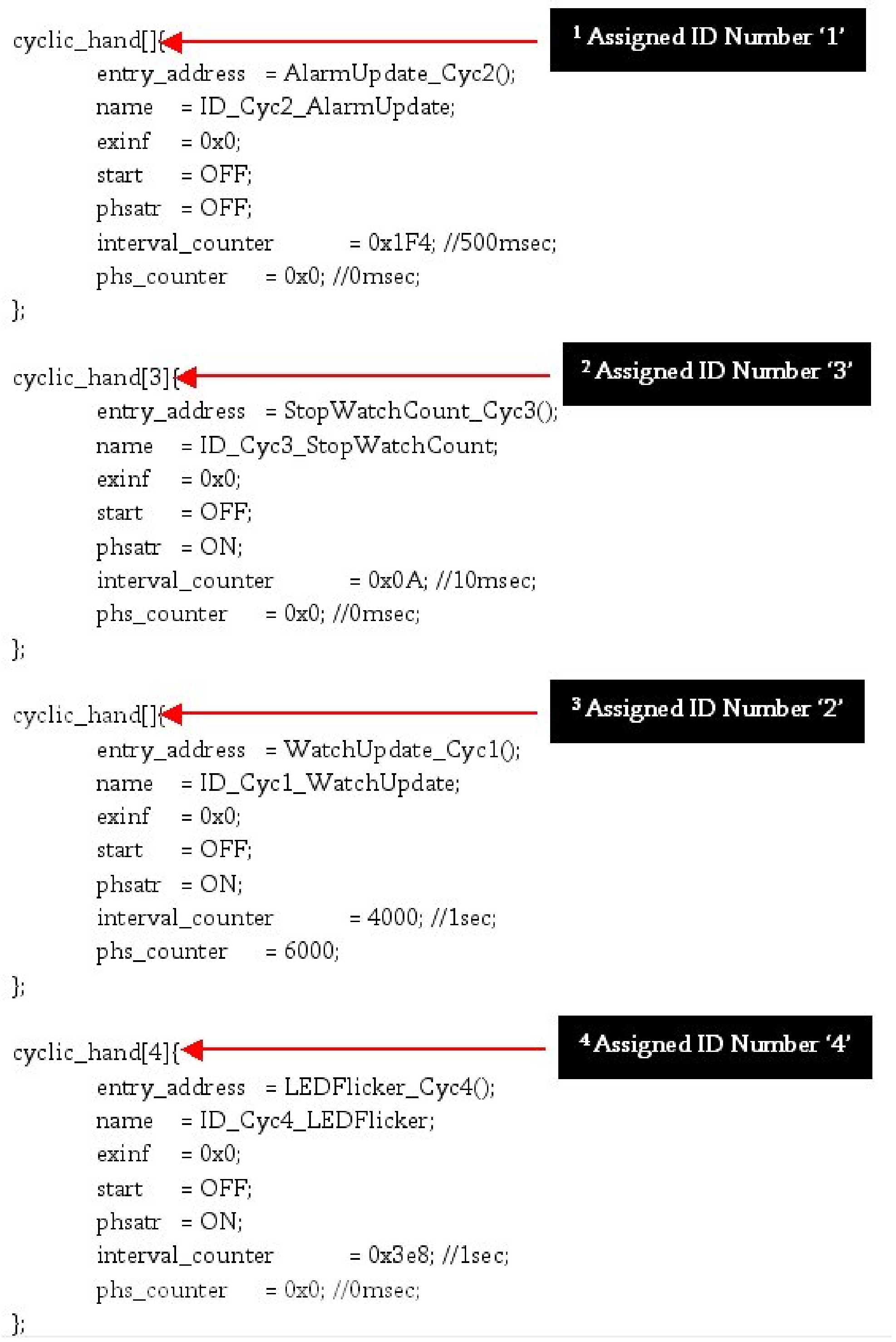
In MR8C/4, users are able to specify up to 255 cyclic handlers. An individual cyclic handler is an object identified by an ID number. Each cyclic handler has a unique ID number that is also called the cyclic handler ID.



```
cyclic_hand[1]{
        entry_address   = WatchUpdate_Cyc1();
        name            = ID_Cyc1_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).



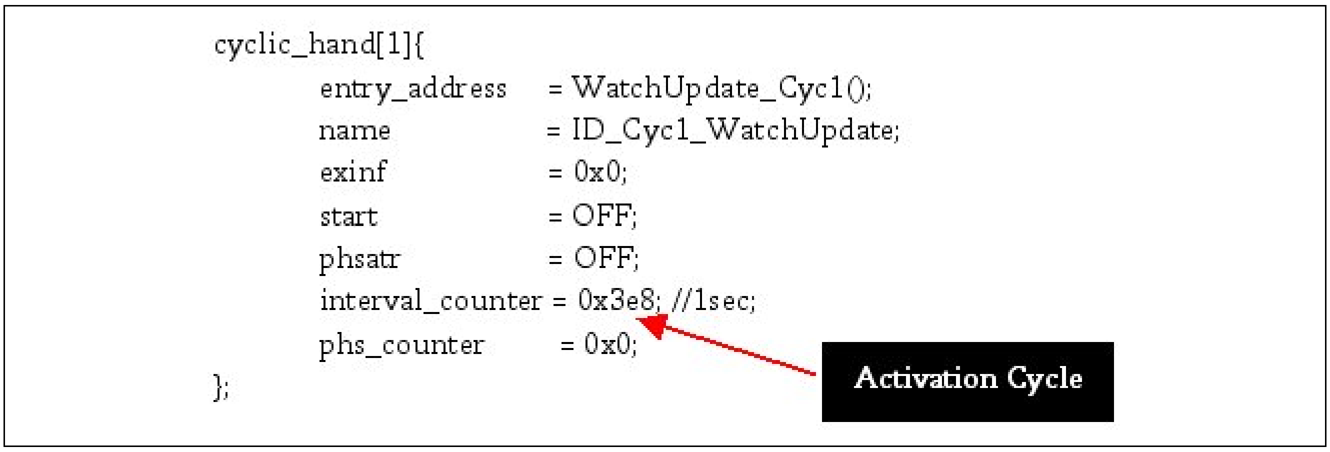
```
cyclic_hand[]{
        entry_address   = AlarmUpdate_Cyc2();
        name    = ID_Cyc2_AlarmUpdate;
        exinf   = 0x0;
        start   = OFF;
        phsatr  = OFF;
        interval_counter          = 0x1F4; //500msec;
        phs_counter     = 0x0; //0msec;
};

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

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

cyclic_hand[4]{
        entry_address   = LEDFlicker_Cyc4();
        name    = ID_Cyc4_LEDFlicker;
        exinf   = 0x0;
        start   = OFF;
        phsatr  = 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).



```
cyclic_hand[1]{
        entry_address    = WatchUpdate_Cyc1();
        name             = ID_Cyc1_WatchUpdate;
        exinf            = 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.

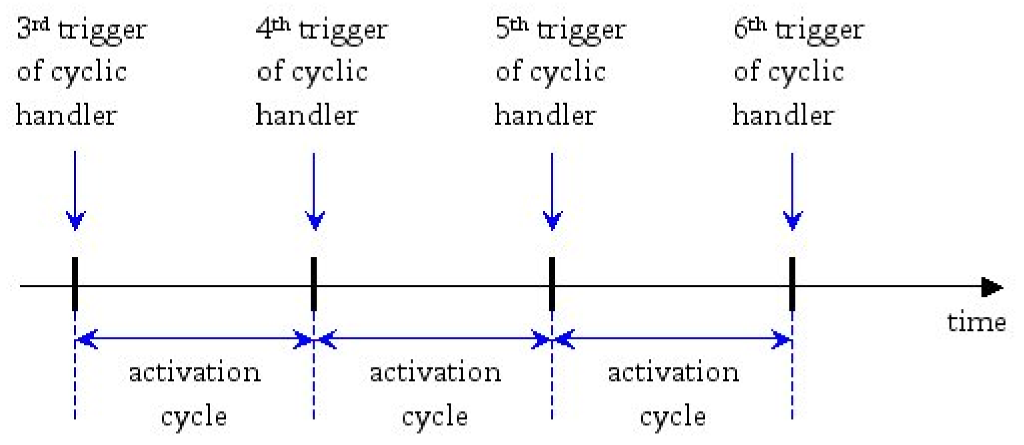
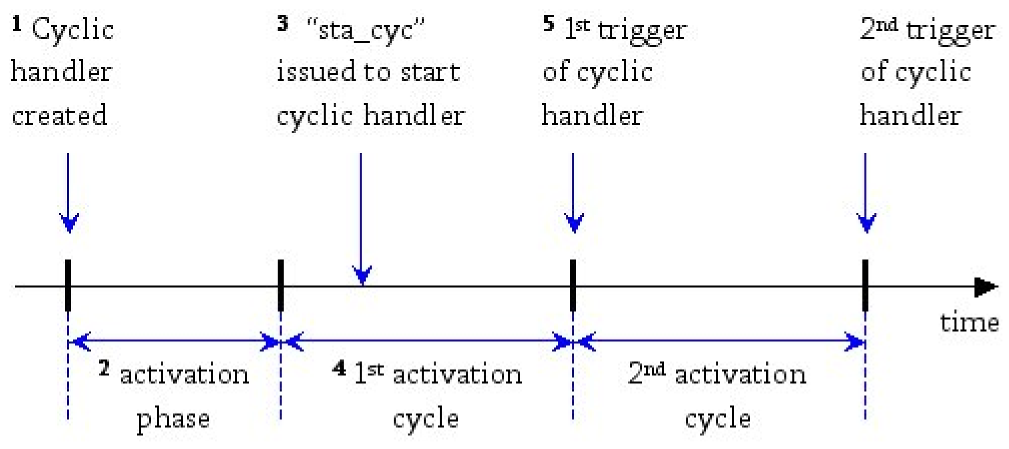

3rd trigger of cyclic handler
4th trigger of cyclic handler
5th trigger of cyclic handler
6th trigger of cyclic handler
time
activation cycle
activation cycle
activation cycle

Figure 6 Cyclic Handler Operations

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.


1 Cyclic handler created
3 "sta_cyc" issued to start cyclic handler
5 1st trigger of cyclic handler
2nd trigger of cyclic handler
time
2 activation phase
4 1st activation cycle
2nd activation cycle

Notes: 1. User defines cyclic handler in configuration file.
Cylic 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.

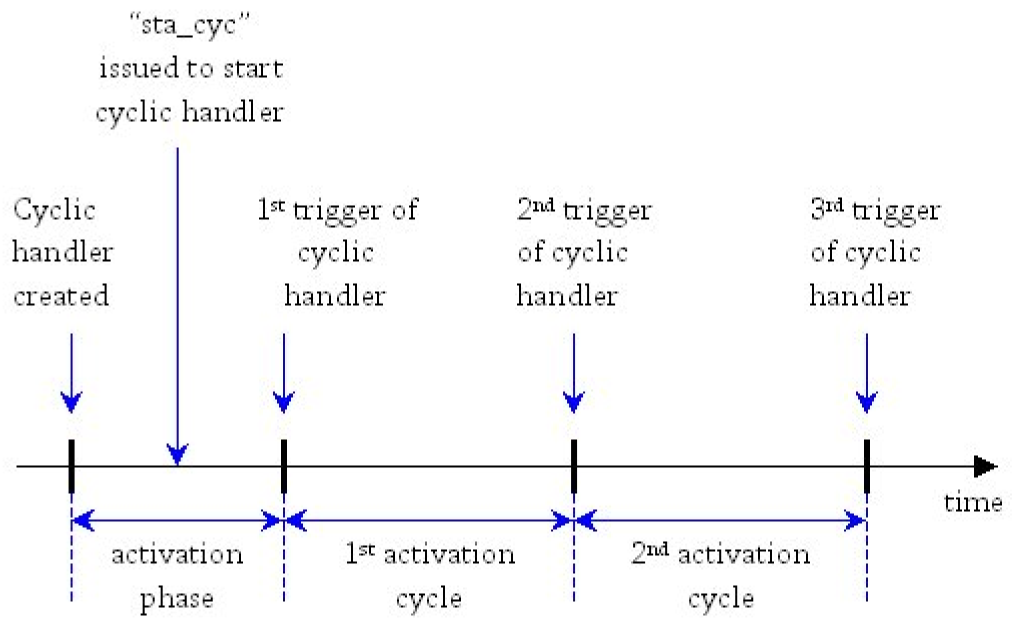
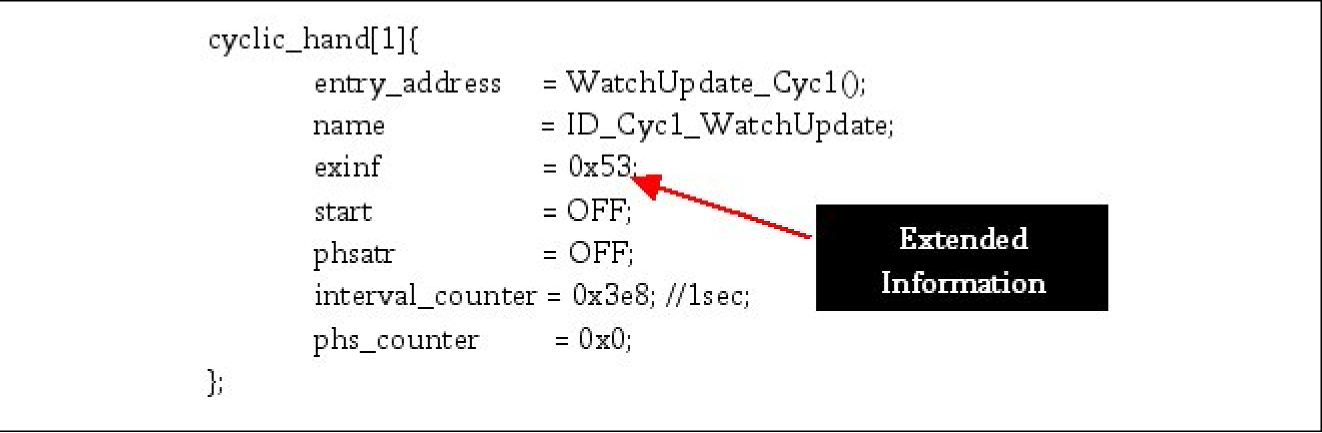

"sta_cyc" issued to start cyclic handler
Cyclic handler created
1st trigger of cyclic handler
2nd trigger of cyclic handler
3rd trigger of cyclic handler
time
activation phase
1st activation cycle
2nd activation cycle

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

3.1.4 Extended Information

Extended information (16 bits) is added in uITRON 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 provided extended information. Extended information is specified in 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 kernel or software component.



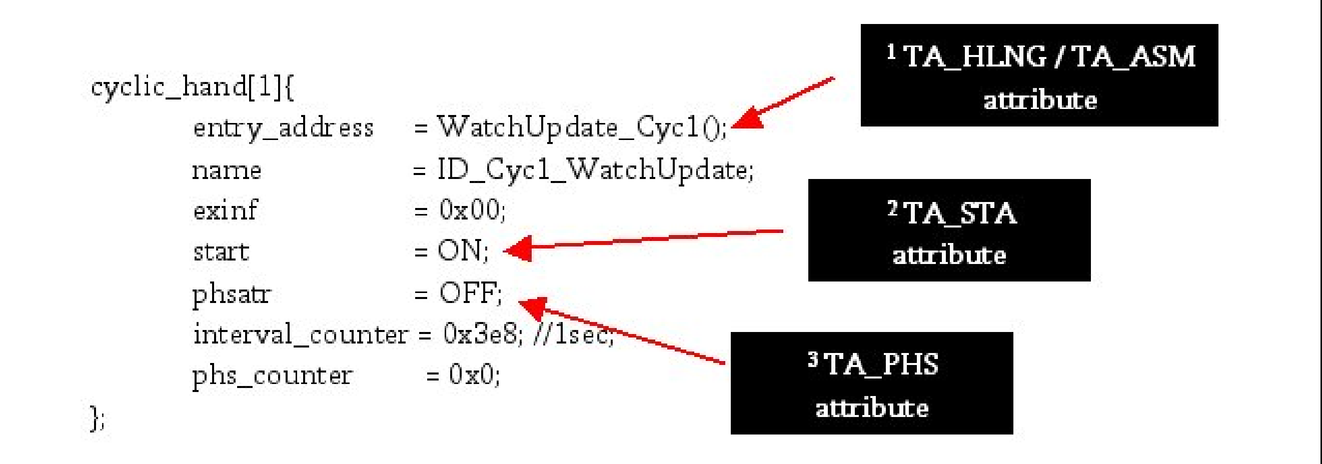
```
cyclic_hand[1]{
        entry_address    = WatchUpdate_Cyc1();
        name             = ID_Cyc1_WatchUpdate;
        exinf            = 0x53;
        start            = OFF;
        phsatr           = OFF;
        interval_counter = 0x3e8; //1sec;
        phs_counter      = 0x0;
};
```

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



```
cyclic_hand[1]{
        entry_address    = WatchUpdate_Cyc1();
        name             = ID_Cyc1_WatchUpdate;
        exinf            = 0x00;
        start            = ON;
        phsatr           = OFF;
        interval_counter = 0x3e8; //1sec;
        phs_counter      = 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.
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.

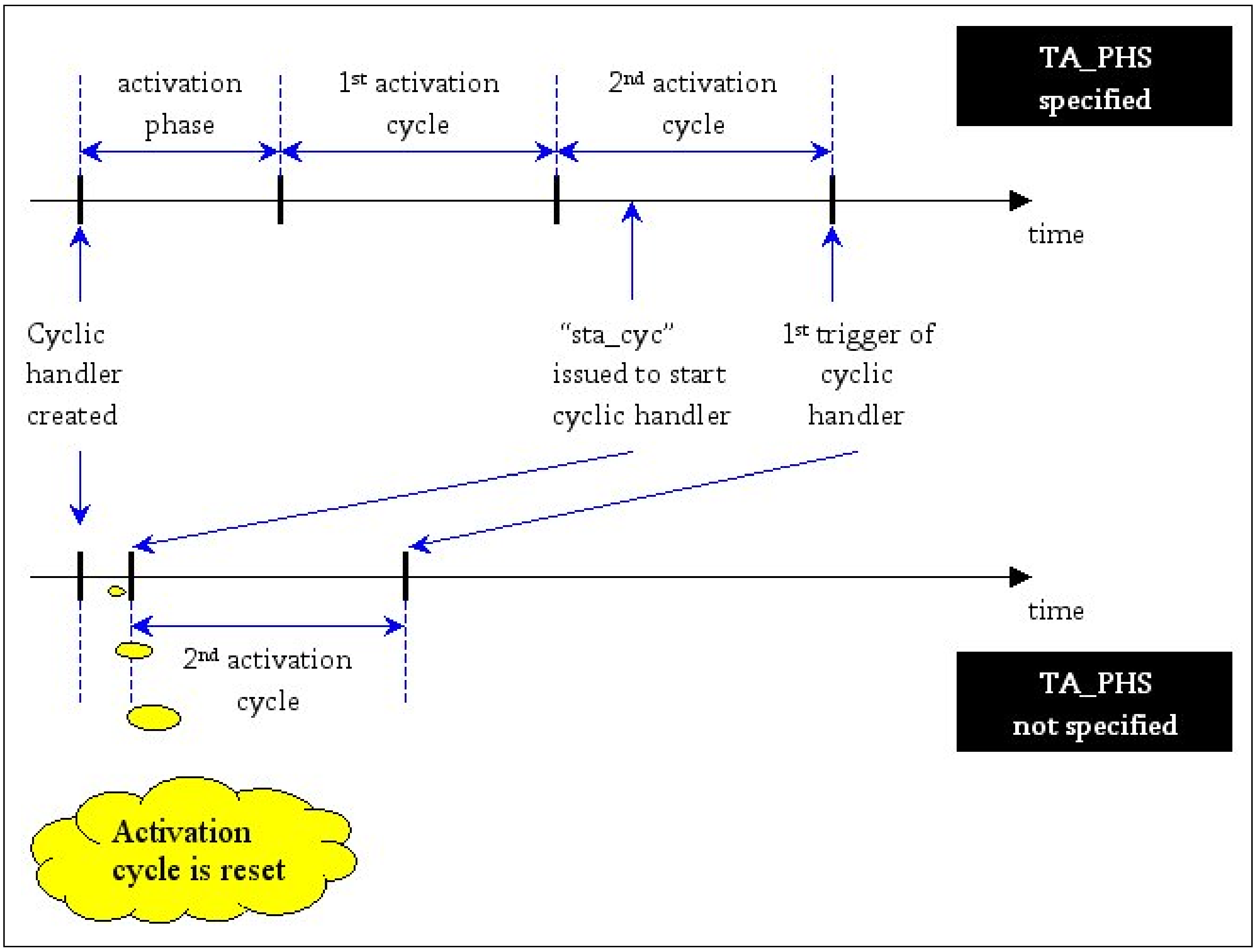
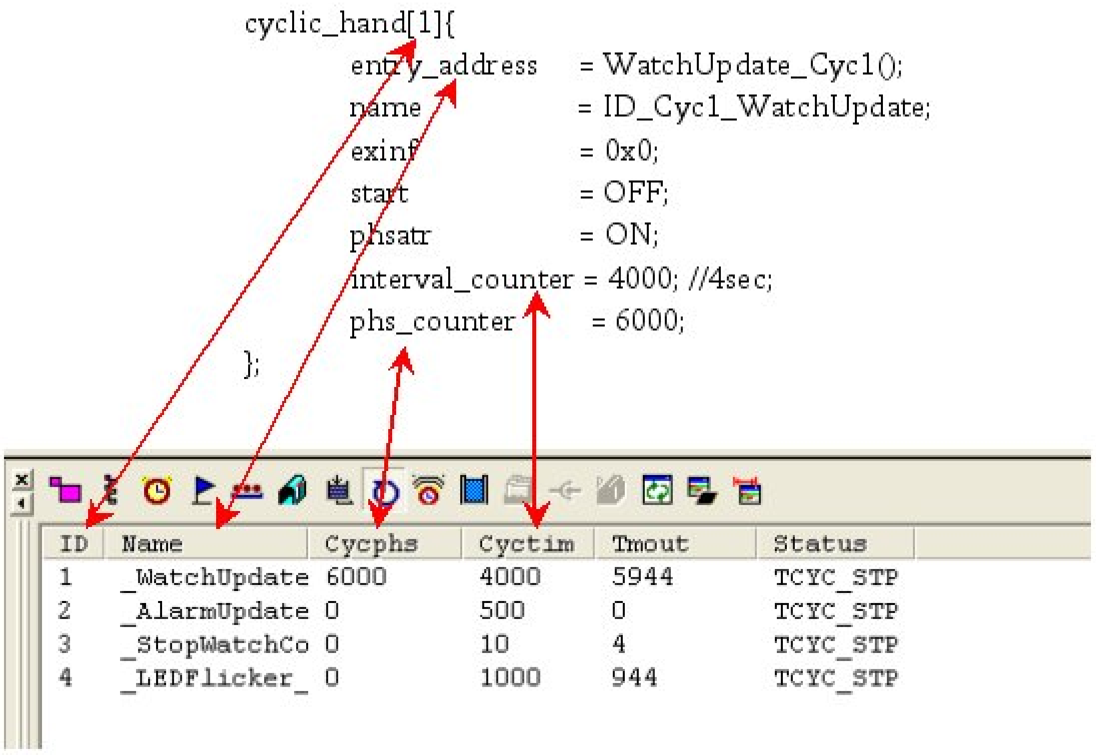


Figure 11 Cyclic Handler Operation with/without TA_PHS Specified



```
cyclic_hand[1]{
        entry_address    = WatchUpdate_Cyc1();
        name             = ID_Cyc1_WatchUpdate;
        exinf            = 0x0;
        start            = OFF;
        phsatr           = ON;
        interval_counter = 4000; //4sec;
        phs_counter        = 6000;
};
```

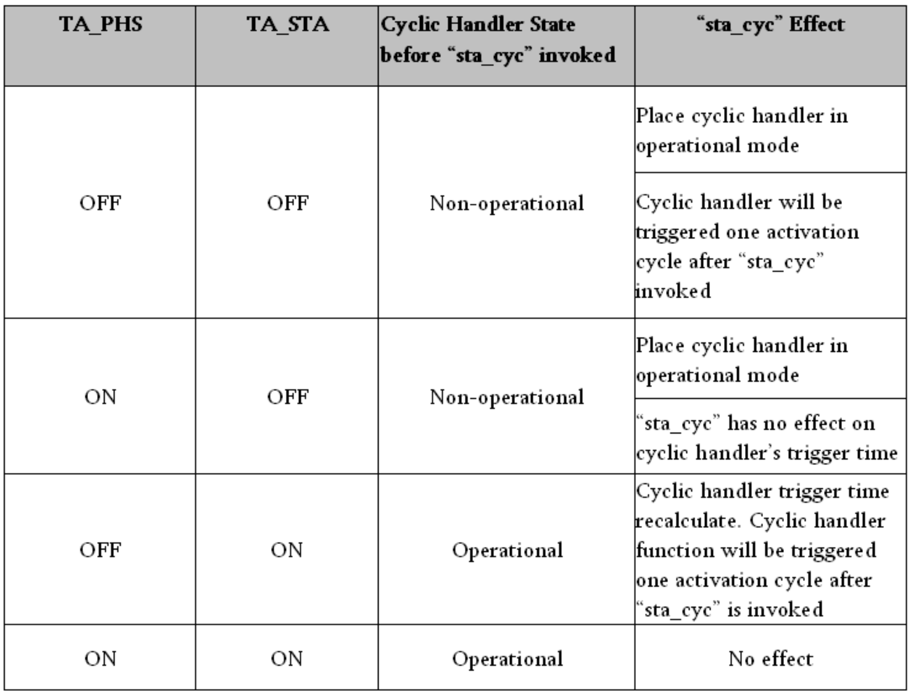
```
ID  Name         Cycphs  Cyctim  Tmout  Status
1   _WatchUpdate 6000    4000    5944   TCYC_STP
2   _AlarmUpdate 0       500     0      TCYC_STP
3   _StopWatchCo 0       10      4      TCYC_STP
4   _LEDFlicker_ 0       1000    944    TCYC_STP
```

Figure 12 Cyclic Handler Components Display in Renesas HEW MR Window

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.

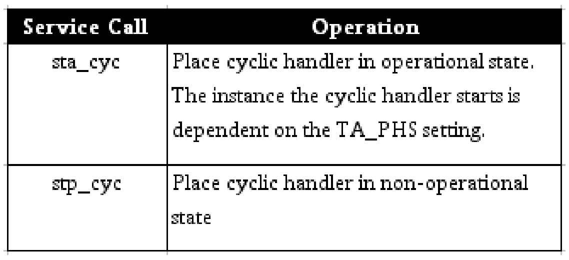
“sta_cyc” service call places a cyclic handler from non-operational to operational state. “sta_cyc” has the function characteristics as shown in Figure 13.



TA_PHS	TA_STA	Cyclic Handler State before “sta_cyc” invoked	“sta_cyc” Effect
OFF	OFF	Non-operational	Place cyclic handler in operational mode
			Cyclic handler will be triggered one activation cycle after “sta_cyc” invoked
ON	OFF	Non-operational	Place cyclic handler in operational mode
			“sta_cyc” has no effect on cyclic handler’s trigger time
OFF	ON	Operational	Cyclic handler trigger time recalculate. Cyclic handler function will be triggered one activation cycle after “sta_cyc” is invoked
ON	ON	Operational	No effect

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.



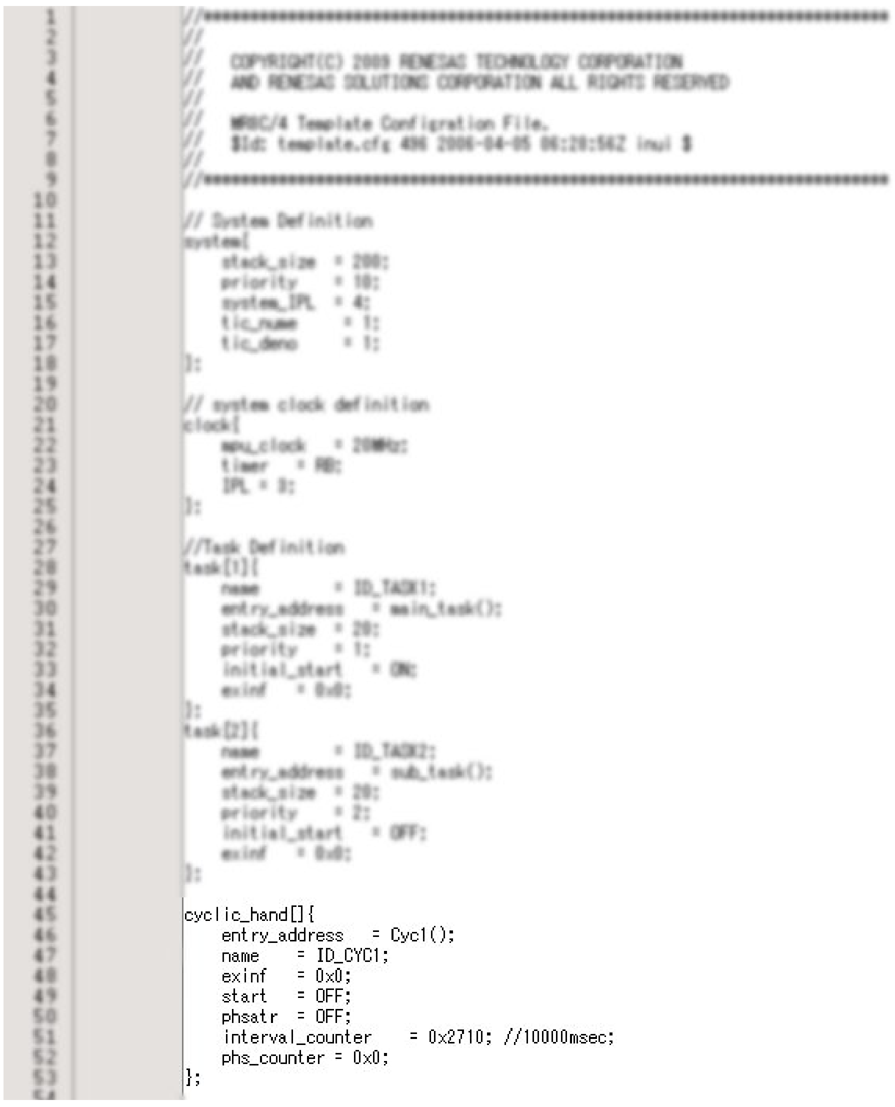
Service Call	Operation
sta_cyc	Place cyclic handler in operational state. The instance the cyclic handler starts is dependent on the TA_PHS setting.
stp_cyc	Place cyclic handler in non-operational state

Figure 14 Cyclic Handler Service Calls

3.3 Implementing Cyclic Handler

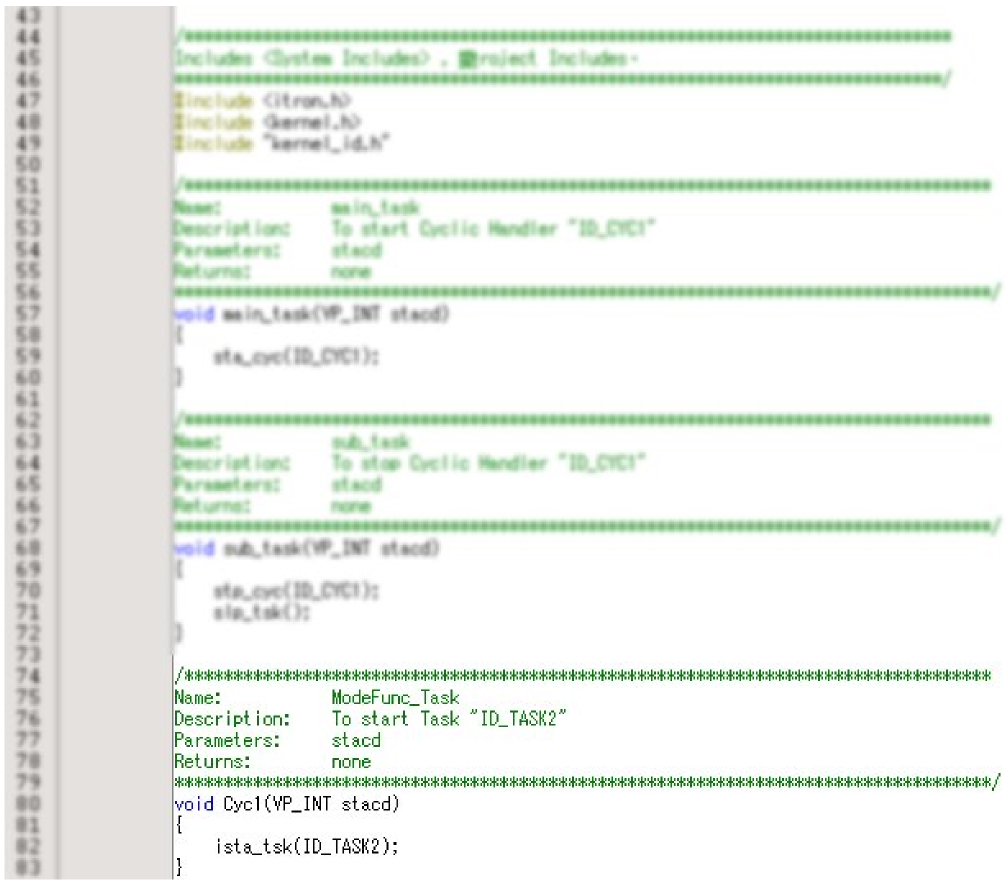
There are three main steps to setup a cyclic handler in MR8C/4. They are:

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)



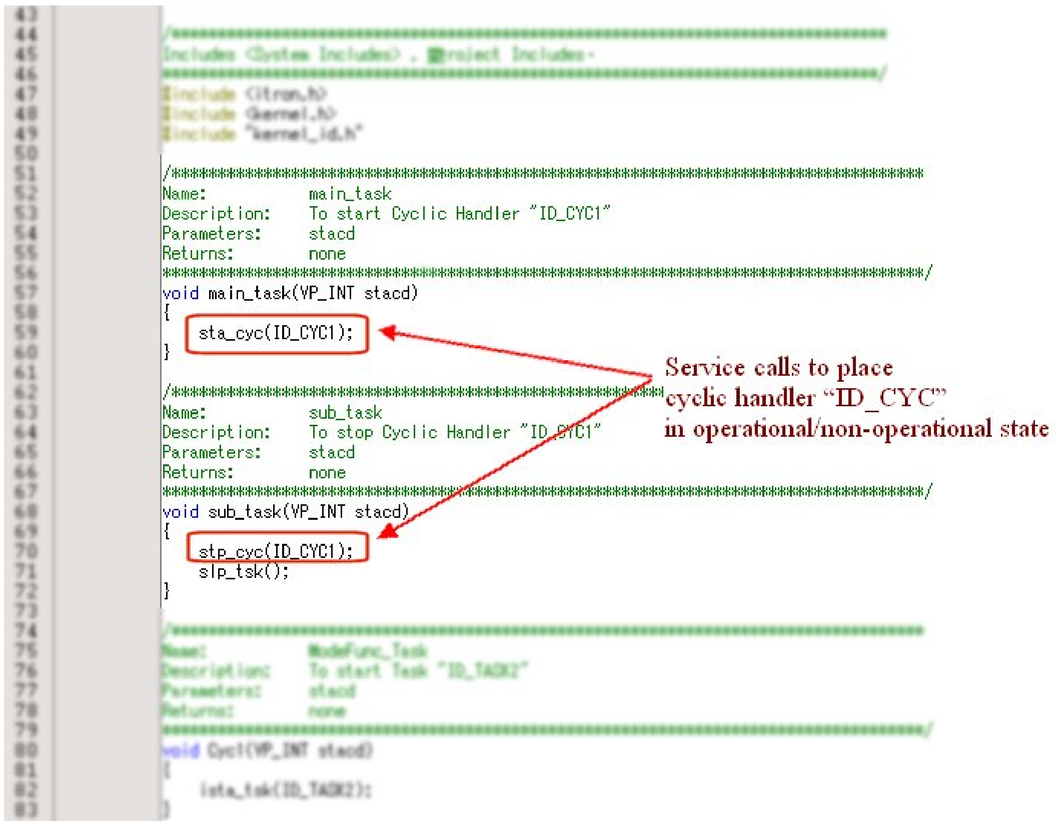
```
//****************************************************************************
//
//    COPYRIGHT(C) 2009 RENESAS TECHNOLOGY CORPORATION
//    AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
//
//    MR8C/4 Template Configration File.
//    $Id: template.cfg 496 2006-04-05 06:20:56Z inui $
//
//****************************************************************************

// System Definition
system{
    stack_size   = 200;
    priority     = 10;
    system_IPL   = 4;
    tic_nume      = 1;
    tic_deno      = 1;
};

// system clock definition
clock{
    mpu_clock    = 20MHz;
    timer    = RB;
    IPL = 3;
};

//Task Definition
task[1]{
    name          = ID_TASK1;
    entry_address    = main_task();
    stack_size   = 20;
    priority     = 1;
    initial_start    = ON;
    exinf     = 0x0;
};
task[2]{
    name          = ID_TASK2;
    entry_address    = sub_task();
    stack_size   = 20;
    priority     = 2;
    initial_start    = OFF;
    exinf     = 0x0;
};

cyclic_hand[]{
    entry_address    = Cyc1();
    name    = ID_CYC1;
    exinf   = 0x0;
    start   = OFF;
    phsatr  = OFF;
    interval_counter    = 0x2710; //10000msec;
    phs_counter = 0x0;
};
```

Figure 15 Defining Cyclic Handler Object in Configuration File (e.g. Template.cfg)



```
/***************************************************************************
Includes <System Includes> , "Project Includes"
***************************************************************************/
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

/***************************************************************************
Name:           main_task
Description:    To start Cyclic Handler "ID_CYC1"
Parameters:     stacd
Returns:        none
***************************************************************************/
void main_task(VP_INT stacd)
{
    sta_cyc(ID_CYC1);
}

/***************************************************************************
Name:           sub_task
Description:    To stop Cyclic Handler "ID_CYC1"
Parameters:     stacd
Returns:        none
***************************************************************************/
void sub_task(VP_INT stacd)
{
    stp_cyc(ID_CYC1);
    slp_tsk();
}

/***************************************************************************
Name:           ModeFunc_Task
Description:    To start Task "ID_TASK2"
Parameters:     stacd
Returns:        none
***************************************************************************/
void Cyc1(VP_INT stacd)
{
    ista_tsk(ID_TASK2);
}
```

Figure 16 Defining Cyclic Handler Routine Function



```
/***************************************************************************
Includes <System Includes> , "Project Includes"
***************************************************************************/
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

/***************************************************************************
Name:           main_task
Description:    To start Cyclic Handler "ID_CYC1"
Parameters:     stacd
Returns:        none
***************************************************************************/
void main_task(VP_INT stacd)
{
    sta_cyc(ID_CYC1);
}

/***************************************************************************
Name:           sub_task
Description:    To stop Cyclic Handler "ID_CYC1"
Parameters:     stacd
Returns:        none
***************************************************************************/
void sub_task(VP_INT stacd)
{
    stp_cyc(ID_CYC1);
    slp_tsk();
}

/***************************************************************************
Name:           ModeFunc_Task
Description:    To start Task "ID_TASK2"
Parameters:     stacd
Returns:        none
***************************************************************************/
void Cyc1(VP_INT stacd)
{
    ista_tsk(ID_TASK2);
}
```

Figure 17 Implement Cyclic Handler Service Calls

4. Understanding Alarm Handlers in MR8C/4

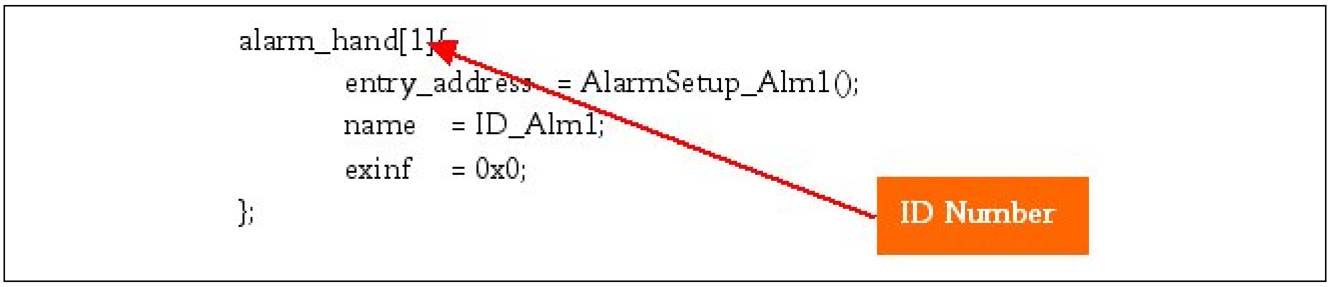
4.1 Components of Alarm Handler

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.



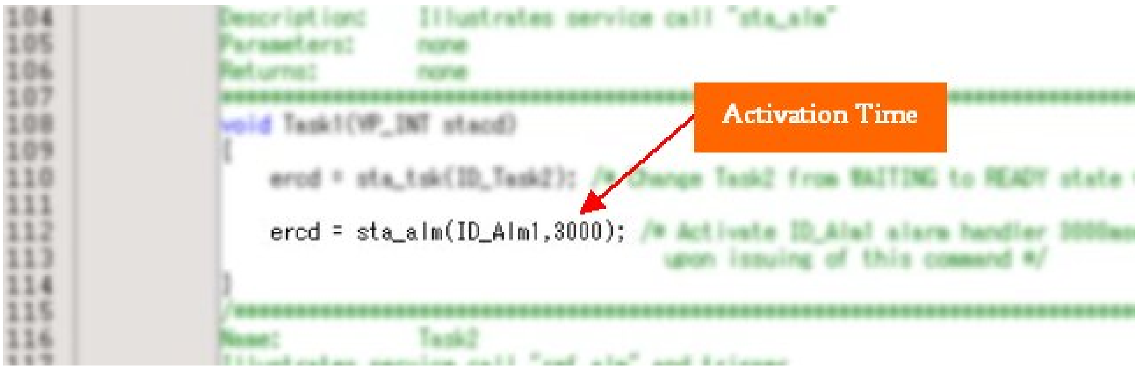
```
alarm_hand[1]{
        entry_address  = AlarmSetup_Alm1();
        name    = ID_Alm1;
        exinf   = 0x0;
};
```

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 1millisecond to ~24 days). To define an alarm handler to trigger in 3 seconds, activation time must be set to 0xBB8(hex) or 3000(decimal).



```
Description:    Illustrates service call "sta_alm"
Parameters:     none
Returns:        none
************************************************************
void Task1(VP_INT stacd)
{
    ercd = sta_tsk(ID_Task2); /* Change Task2 from WAITING to READY state
    
    ercd = sta_alm(ID_Alm1,3000); /* Activate ID_Alm1 alarm handler 3000ms
                                     upon issuing of this command */
}
/************************************************************
Name:           Task2
```

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.

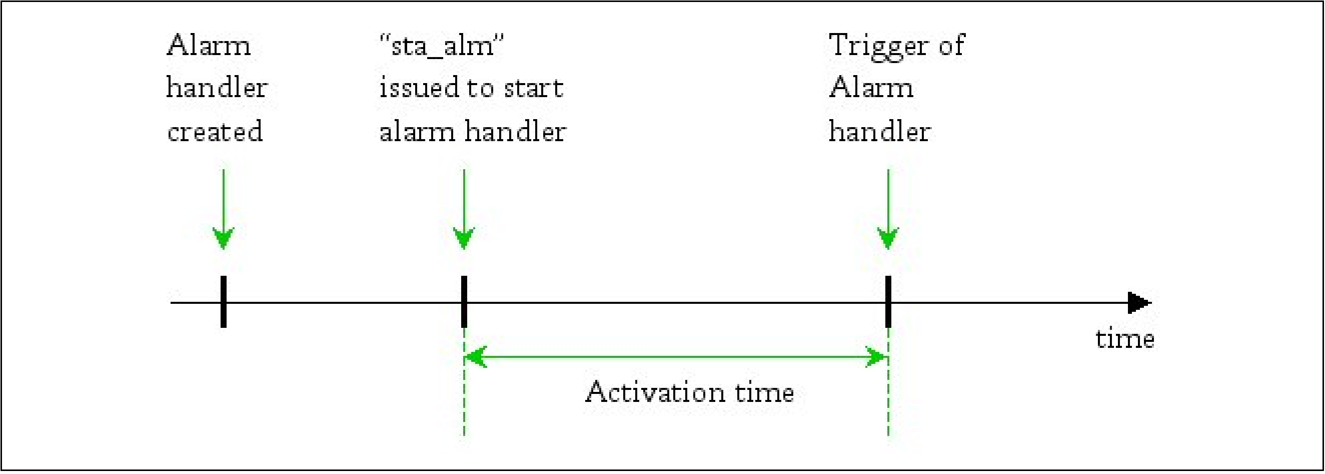
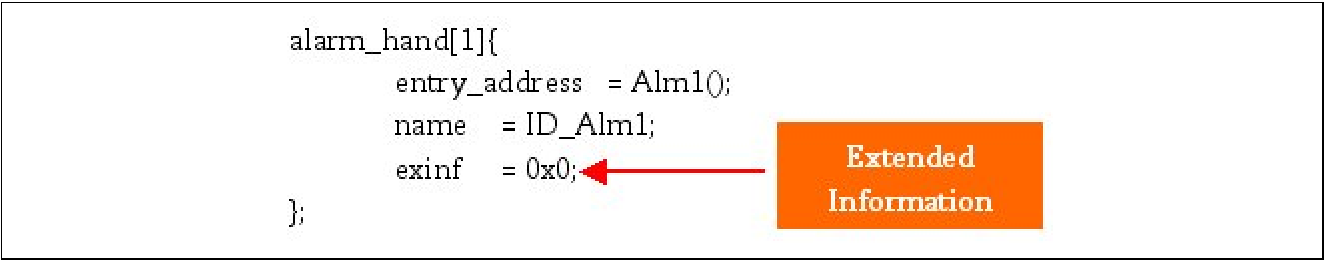


Figure 20 Alarm Handler Operations

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.



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

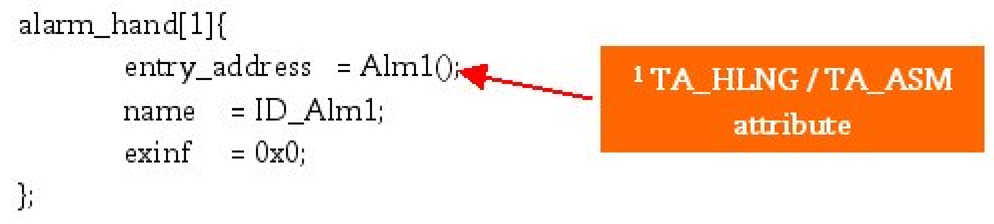
Extended Information

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

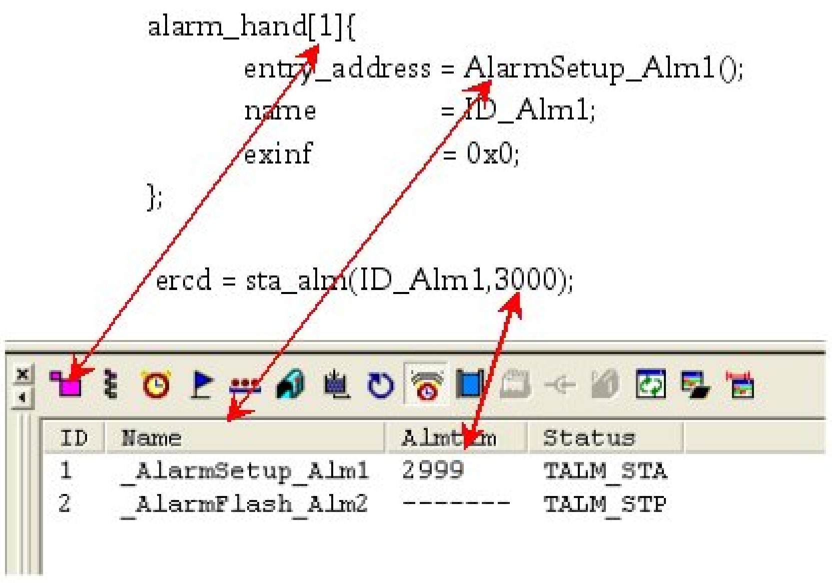


```
alarm_hand[1]{
        entry_address  = Alm1();
        name   = ID_Alm1;
        exinf  = 0x0;
};
```

1 TA_HLNG / TA_ASM attribute

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)



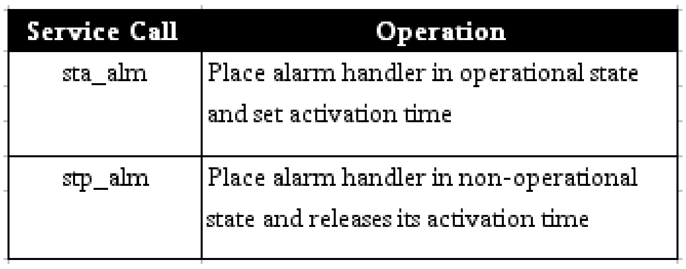
```
alarm_hand[1]{
        entry_address = AlarmSetup_Alm1();
        name          = ID_Alm1;
        exinf         = 0x0;
};

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

```
ID  Name              Almtim   Status
1   _AlarmSetup_Alm1  2999     TALM_STA
2   _AlarmFlash_Alm2  -------  TALM_STP
```

Figure 23 Alarm Handler Components Display in Renesas HEW MR Window

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.



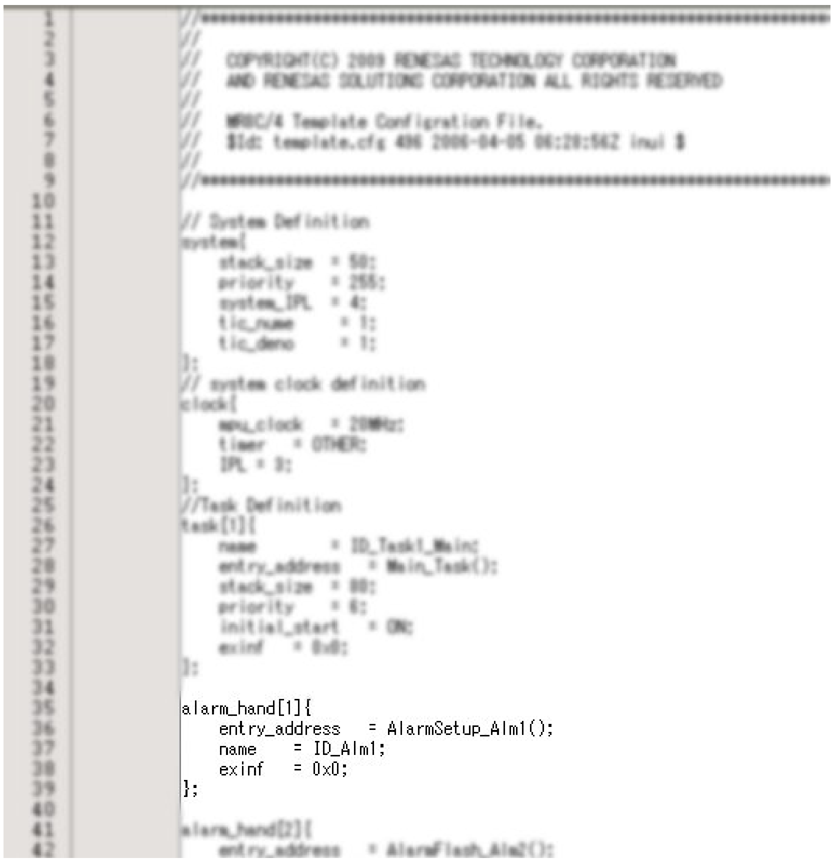
Service Call	Operation
sta_alm	Place alarm handler in operational state and set activation time
stp_alm	Place alarm handler in non-operational state and releases its activation time

Figure 24 Alarm Handler Service Calls

4.3 Implementing Alarm Handler in MR8C/4

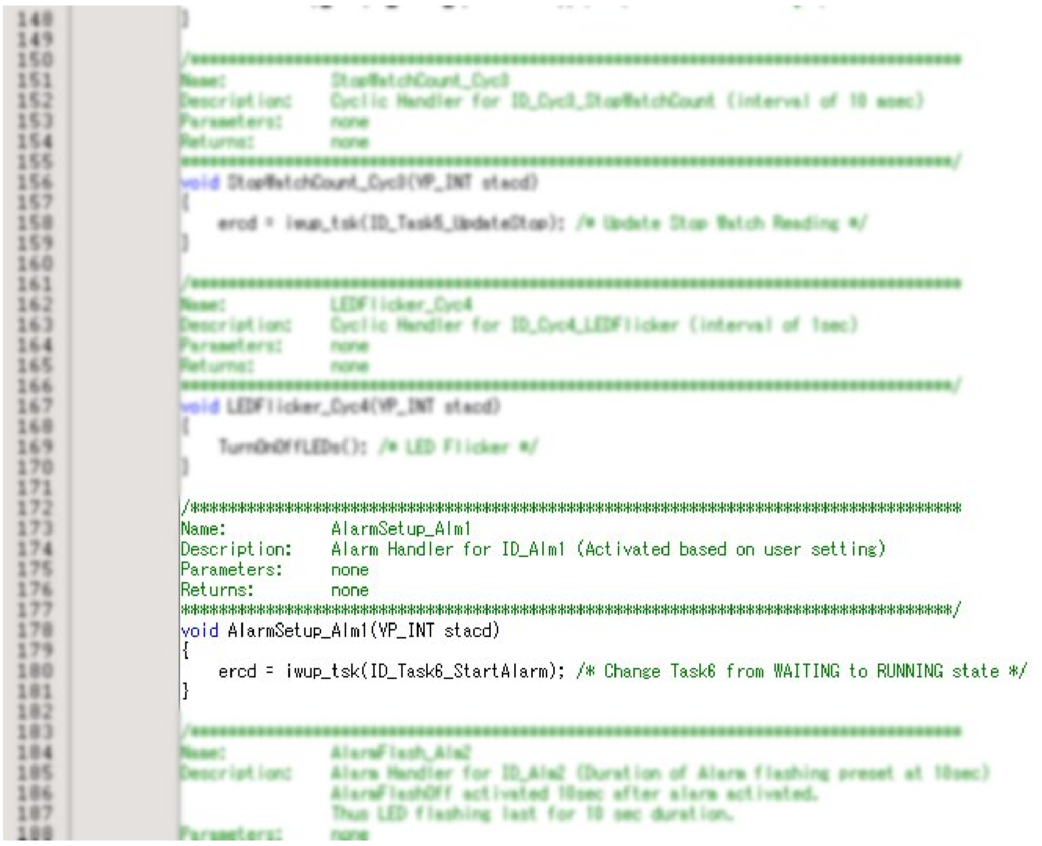
An alarm handler can be setup as follows:

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)



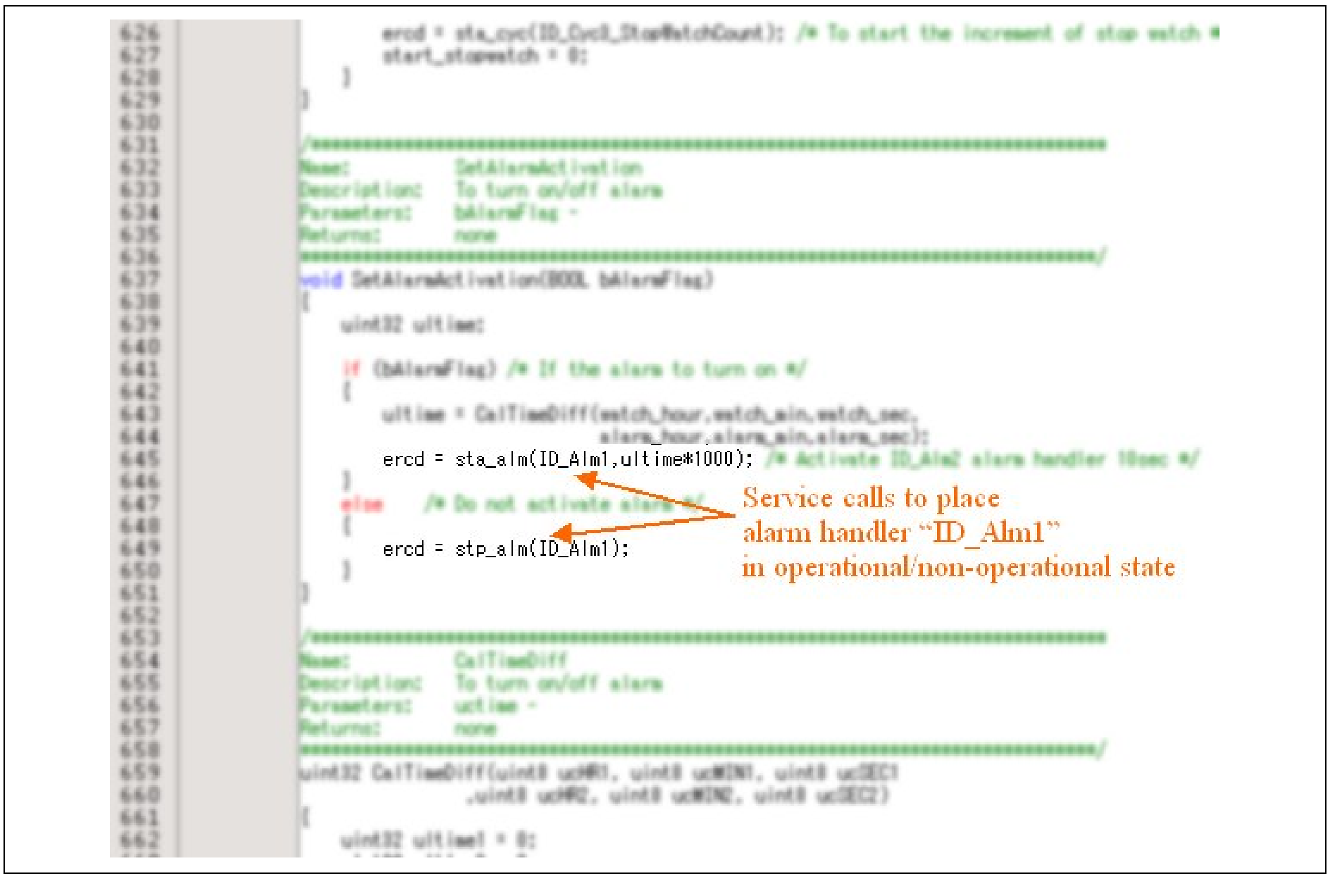
```
//******************************************************************
//
//    COPYRIGHT(C) 2009 RENESAS TECHNOLOGY CORPORATION
//    AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
//
//    MR8C/4 Template Configration File.
//    $Id: template.cfg 496 2006-04-05 06:28:56Z inui $
//
//******************************************************************

// System Definition
system{
    stack_size  = 50;
    priority    = 255;
    system_IPL  = 4;
    tic_nume     = 1;
    tic_deno     = 1;
};
// system clock definition
clock{
    mpu_clock   = 20MHz;
    timer    = OTHER;
    IPL = 3;
};
//Task Definition
task[1]{
    name          = ID_Task1_Main;
    entry_address    = Main_Task();
    stack_size  = 80;
    priority     = 6;
    initial_start    = ON;
    exinf    = 0x0;
};

alarm_hand[1]{
    entry_address   = AlarmSetup_Alm1();
    name    = ID_Alm1;
    exinf   = 0x0;
};

alarm_hand[2]{
    entry_address   = AlarmFlash_Alm2();
```

Figure 25 Defining Alarm Handler Object in Configuration File (e.g. Template.cfg)



```
}

/******************************************************************************
Name:           StopWatchCount_Cyc3
Description:    Cyclic Handler for ID_Cyc3_StopWatchCount (interval of 10 msec)
Parameters:     none
Returns:        none
******************************************************************************/
void StopWatchCount_Cyc3(VP_INT stacd)
{
    ercd = iwup_tsk(ID_Task5_UpdateStop); /* Update Stop Watch Reading */
}

/******************************************************************************
Name:           LEDFlicker_Cyc4
Description:    Cyclic Handler for ID_Cyc4_LEDFlicker (interval of 1sec)
Parameters:     none
Returns:        none
******************************************************************************/
void LEDFlicker_Cyc4(VP_INT stacd)
{
    TurnOnOffLEDs(); /* LED Flicker */
}

/******************************************************************************
Name:           AlarmSetup_Alm1
Description:    Alarm Handler for ID_Alm1 (Activated based on user setting)
Parameters:     none
Returns:        none
******************************************************************************/
void AlarmSetup_Alm1(VP_INT stacd)
{
    ercd = iwup_tsk(ID_Task6_StartAlarm); /* Change Task6 from WAITING to RUNNING state */
}

/******************************************************************************
Name:           AlarmFlash_Alm2
Description:    Alarm Handler for ID_Alm2 (Duration of Alarm flashing preset at 10sec)
                AlarmFlashOff activated 10sec after alarm activated.
                Thus LED flashing last for 10 sec duration.
Parameters:     none
```

Figure 26 Defining Alarm Handler Routine Function



```
        ercd = sta_cyc(ID_Cyc3_StopWatchCount); /* To start the increment of stop watch *
        start_stopwatch = 0;
    }
}

/****************************************************************************
Name:           SetAlarmActivation
Description:    To turn on/off alarm
Parameters:     bAlarmFlag -
Returns:        none
****************************************************************************/
void SetAlarmActivation(BOOL bAlarmFlag)
{
    uint32 ultime;

    if (bAlarmFlag) /* If the alarm to turn on */
    {
        ultime = CalTimeDiff(watch_hour,watch_min,watch_sec,
                             alarm_hour,alarm_min,alarm_sec);
        ercd = sta_alm(ID_Alm1,ultime*1000); /* Activate ID_Alm2 alarm handler 10sec */
    }
    else    /* Do not activate alarm */
    {
        ercd = stp_alm(ID_Alm1);
    }
}

/****************************************************************************
Name:           CalTimeDiff
Description:    To turn on/off alarm
Parameters:     uctime -
Returns:        none
****************************************************************************/
uint32 CalTimeDiff(uint8 ucHR1, uint8 ucMIN1, uint8 ucSEC1
                  ,uint8 ucHR2, uint8 ucMIN2, uint8 ucSEC2)
{
    uint32 ultime1 = 0;
```

Figure 27 Implement Alarm Handler Service Calls

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	March.01.10	—	First edition issued

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

RENESAS

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

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

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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

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

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0