User's Manual

RI850MP

Real-Time Operating System

User's Manual: Coding

Target Tool

RI850MP

Renesas Electronics
www.renesas.com

Rev.1.00 Apr 2011

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 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.
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 exporting 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 product's 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" or 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 sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
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 damages or losses 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 product(s)" means any product developed or manufactured by or for Renesas Electronics.

How to Use This Manual

Readers

This manual is intended for users who design and develop application systems using V850 microcontroller products.

Purpose

This manual is intended for users to understand the functions of real-time OS "RI850MP " manufactured by Renesas Electronics, described the organization listed below.

Organization

This manual consists of the following major sections.

How to read this manual

It is assumed that the readers of this manual have general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, C language, and assemblers.

To understand the hardware functions of the V850 microcontroller
→ Refer to the **User's Manual** of each product.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numerical representation:	Binary...XXXX or XXXXB
	Decimal...XXXX
	Hexadecimal...0xXXXX

Prefixes indicating power of 2 (address space and memory capacity):

K (kilo)	$2^{10} = 1024$
M (mega)	$2^{20} = 1024^2$

Related Documents Refer to the documents listed below when using this manual.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to development tools (User's Manuals)

Document Name		Document No.
RI Series	Start	R20UT0509E
	Message	R20UT0510E
RI78V4	Coding	R20UT0511E
	Debug	R20UT0520E
	Analysis	R20UT0513E
	Internal Structure	R20UT0514E
RI850V4	Coding	R20UT0515E
	Debug	R20UT0516E
	Analysis	R20UT0517E
	Internal Structure	R20UT0518E
RI850MP	Coding	This document
CubeSuite+ Integrated Development Environment	Start	R20UT0545E
	78K0 Design	R20UT0546E
	78K0R Design	R20UT0547E
	RL78 Design	R20UT0548E
	V850 Design	R20UT0549E
	R8C Design	R20UT0550E
	78K0 Coding	R20UT0551E
	RL78,78K0R Coding	R20UT0552E
	V850 Coding	R20UT0553E
	Coding for CX Compiler	R20UT0554E
	R8C Coding	R20UT0576E
	78K0 Build	R20UT0555E
	RL78,78K0R Build	R20UT0556E
	V850 Build	R20UT0557E
	Build for CX Compiler	R20UT0558E
	R8C Build	R20UT0575E
	78K0 Debug	R20UT0559E
	78K0R Debug	R20UT0560E
	RL78 Debug	R20UT0561E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest edition of each document when designing.

[MEMO]

[MEMO]

[MEMO]

TABLE OF CONTENTS

CHAPTER 1 GENERAL

1.1 Outline

The RI850MP is a built-in real-time, multi-task OS that provides a highly efficient real-time, multi-task environment to increases the application range of processor control units.

The RI850MP is a high-speed, compact OS capable of being stored in and run from the ROM of a target system.

CHAPTER 2 TASK MANAGEMENT FUNCTIONS

This chapter describes the task management functions provided by the RI850MP.

2.1 Outline

The task management functions provided by the RI850MP include functions to reference task states and functions to manipulate task states.

Remark For details about service called provided by the RI850MP for task management functions, refer to "13.4.1 Task management functions".

2.2 Tasks

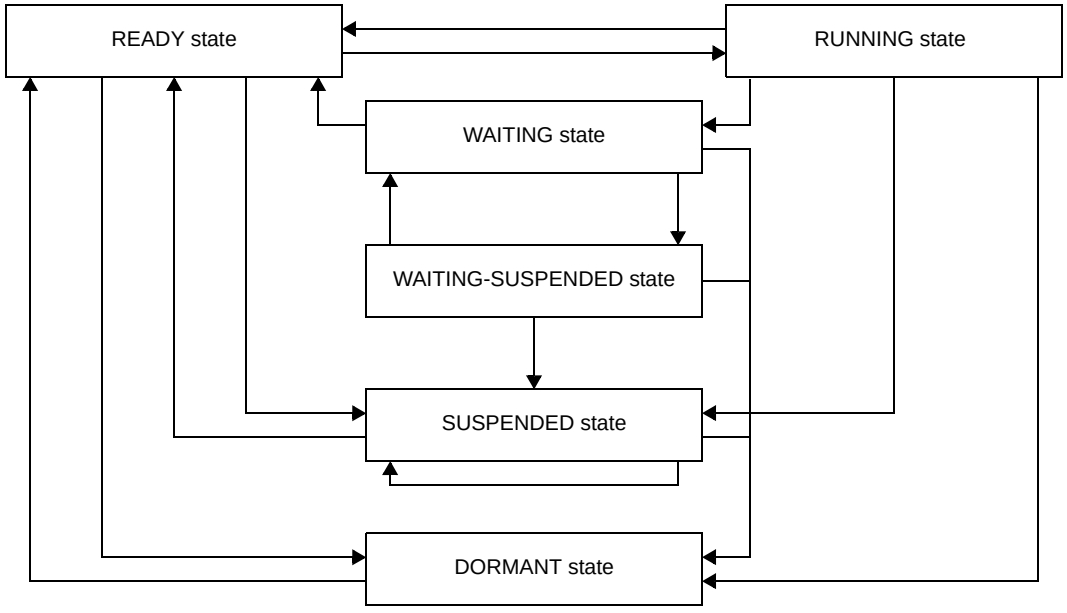
A task is processing program that is not executed unless it is explicitly manipulated via service calls provided by the RI850MP, unlike other processing programs (such as cyclic handlers, interrupt handlers, and CPU exception handlers).

2.2.1 Task states

Tasks enter various states according to whether the OS resources required for task execution have been acquired, and the occurrence/non-occurrence of various events.

The RI850MP classifies task states into the following six types.

Figure 2-1. Task State Transitions



- DORMANT state
  State of a task that is not active, or the state entered by a task whose processing has ended.
  A task in the waiting state, while being under management of the RI850MP, is not subject to RI850MP scheduling.

- READY state
  State of a task for which the preparations required for processing execution have been completed, but which is currently waiting to acquire the right to use the CPU, because the processing of another task with a higher priority level is currently being executed.

- RUNNING state
  State of a task that has acquired the right to use the CPU and is currently running.

- WAITING state
  State in which processing execution has been suspended because conditions required for execution are not satisfied.
  Resumption of processing from the WAITING state starts from the point where the processing execution was suspended.
  On the RI850MP, the WAITING state is classified into the following nine types according to the type of required conditions.

Table 2-1. Waiting State Types

State type	Outline
Sleeping state	A task enters this state if the counter for the task (registering the number of times the wakeup request has been issued) indicates 0x0 upon the issuance of a slp_tsk or tslp_tsk.
Delayed state	A task enters this state upon the issuance of a dly_tsk.
WAITING state for a semaphore resource	A task enters this state if it cannot acquire a resource from the relevant semaphore upon the issuance of a wai_sem or twai_sem.
WAITING state for an eventflag	A task enters this state if a relevant eventflag does not satisfy a predetermined condition upon the issuance of a wai_flg or twai_flg.
Sending waiting state for a data queue	A task enters this state if cannot receive a data from the relevant data queue upon the issuance of a snd_dtq or tsnd_dtq.
Receiving waiting state for a data queue	A task enters this state if cannot receive data from the relevant data queue upon the issuance of a rcv_dtq or trcv_dtq.
Receiving waiting state for a mailbox	A task enters this state if cannot receive a message from the relevant mailbox upon the issuance of a rcv_mbx or trcv_mbx.
WAITING state for a mutex	A task enters this state if it cannot lock the relevant mutex upon the issuance of a loc_mtx or tloc_mtx.
WAITING state for a fixed-sized memory block	A task enters this state if it cannot acquire a fixed-sized memory block from the targetr fixed-sized memory pool upon the issuance of a get_mpf or tget_mpf.

- SUSPENDED state
  State in which processing execution has been suspended forcibly.
  Resumption of processing from the SUSPENDED state starts from the point where the processing execution was suspended.

- WAITING-SUSPENDED state
  State in which the WAITING and SUSPENDED states are combined.
  A task enters the SUSPENDED state when the WAITING state is cancelled, or enters the WAITING state when the SUSPENDED state is cancelled.

2.2.2 Task priorities

A priority level that determines the order in which that task will be processed in relation to other tasks is assigned to each task. As a result, on the RI850MP, the task that has the highest priority level of all the tasks that have entered an executable state (RUNNING state or READY state) is selected and given the right to use the CPU.

On the RI850MP, the following two types of priorities are used for management purposes.

- Initial priority
  This is a value defined by using the static API "CRE_TSK" in the system configuration file.

- Current priority
  This is the priority of a task that has been activated, to which the RI850MP refers when executing various operations.

Remarks 1. On the RI850MP, a task having a smaller priority number is given a higher priority.

2. The range of priorities that can be used on the system is the range of values defined by using the static API "MAX_PRI" in the system configuration file.
For details about the static API "MAX_PRI", refer to "B.3.5 Maximum priority information".

2.2.3 Basic format of tasks

When coding a task, code it as a void function with one VP_INT argument.

The argument *exinf* contains extended information specified with "Task information".

The following shows the basic format of tasks coded in C.

[CX version]

```
#include    <kernel.h>
#pragma     rtos_task   task
void
task(VP_INT exinf){
        .........
        .........
        ext_tsk();
}
```

[CCV850E version]

```
#include    <kernel.h>
void
task(VP_INT exinf){
        .........
        .........
        ext_tsk();
}
```

2.2.4 Task creation

The RI850MP supports only static task creation. Therefore, tasks cannot be created dynamically by issuing a service call from a processing program.

Static task creation means defining of tasks using the static API "CRE_TSK" in the system configuration file.

Remark For details about the static API "CRE_TSK", refer to "B.5.1 Task information".

CHAPTER 3 TASK DEPENDENT SYNCHRONIZATION FUNCTIONS

This chapter describes the task dependent synchronization functions provided by the RI850MP.

3.1 Outline

The task dependent synchronization functions provided by the RI850MP enable synchronization between tasks to be performed concurrently with task state manipulations.

Remark For details about service calls provided by the RI850MP for task dependent synchronization functions, refer to "13.4.2 Task dependent synchronization functions".

CHAPTER 4 SYNCHRONIZATION AND COMMUNICATION FUNCTIONS

This chapter describes the synchronization and communication functions provided by the RI850MP.

4.1 Outline

The synchronization and communication functions of the RI850MP consist of Semaphores, Eventflags, Data Queues, and Mailboxes.

Remark For details about service calls provided by the RI850MP for synchronization and communication, refer to "13.4.3 Synchronization and communication functions (semaphores)", "13.4.4 Synchronization and communication functions (eventflags)", "13.4.5 Synchronization and communication functions (data queues)", and "13.4.6 Synchronization and communication functions (mailboxes)".

4.2 Semaphores

Semaphores are provided to enable synchronization between tasks.

Remark The RI850MP provides non-negative counter-type semaphores.

4.2.1 Semaphore creation

The RI850MP supports only static creation of semaphores. Therefore, semaphores cannot be created dynamically by issuing a service call from a processing program.

Static semaphore creation means defining of semaphores by using the static API "CRE_SEM" in the system configuration file.

Remark For details about the static API "CRE_SEM", refer to "B.5.2 Semaphore information".

4.3 Eventflags

Eventflags are provided to enable synchronization between tasks.

Remark On the RI850MP, the eventflag width is 32 bits.

4.3.1 Eventflag creation

The RI850MP supports only static creation of eventflags. Therefore, eventflags cannot be created dynamically by issuing a service call from a processing program.

Static eventflag creation means defining of eventflags by using the static API "CRE_FLG" in the system configuration file.

Remark For details about the static API "CRE_FLG", refer to "B.5.3 Eventflag information".

4.4 Data Queues

Data queues are provided to enable synchronization and communication between tasks.

Remark The RI850MP provides data queues that enable transmission and reception of data in the specified size (4 bytes).

4.4.1 Data queue creation

The RI850MP supports only static data queue creation. Therefore, data queues cannot be created dynamically by issuing a service call from a processing program.

Static data queue creation means defining of data queues by using the static API "CRE_DTQ" in the system configuration file.

Remark For details about the static API "CRE_DTQ", refer to "B.5.4 Data queue information".

4.5 Mailboxes

Mailboxes are provided to enable synchronization and communication between tasks.

Remark The RI850MP provides mailboxes that enable transmission and reception of data in any size.

4.5.1 Mailbox creation

The RI850MP supports only static mailbox creation. Therefore, mailboxes cannot be created dynamically by issuing a service call from a processing program.

Static mailbox creation means defining of mailboxes by using the static API "CRE_MBX" in the system configuration file.

Remark For details about the static API "CRE_MBX", refer to "B.5.5 Mailbox information".

CHAPTER 5 EXTENDED SYNCHRONIZATION AND COMMUNICATION FUNCTIONS

This chapter describes the extended synchronization and communication functions provided by the RI850MP.

5.1 Outline

The RI850MP provides mutexes as extended synchronization and communication functions to enable synchronization between tasks.

Remark For details about service calls provided by the RI850MP for extended synchronization and communication functions, refer to "13.4.7 Extended synchronization and communication functions".

5.2 Mutexes

Mutexes are provided to enable synchronization between tasks.

5.2.1 Mutex creation

The RI850MP supports only static mutex creation. Therefore, mutexes cannot be created dynamically by issuing a service call from a processing program.

Static mutex creation means defining of mutexes by using the static API "CRE_MTX" in the system configuration file.

Remark For details about the static API "CRE_MTX", refer to "B.5.6 Mutex information".

CHAPTER 6 MEMORY POOL MANAGEMENT FUNCTIONS

This chapter describes the memory pool management functions provided by the RI850MP.

6.1 Outline

The memory pool management functions of the RI850MP provide a fixed-size memory pool to enable dynamic acquisition and release of memory areas by processing programs.

Remark For details about service calls provided by the RI850MP for memory pool management, refer to "13.4.8 Memory pool management functions".

6.2 Fixed-Sized Memory Pool

The fixed-size memory pool is a memory area where processing programs can dynamically acquire and release memory blocks in fixed-size units.

6.2.1 Fixed-sized memory pool creation

The RI850MP supports only static creation of the fixed-sized memory pool. Therefore, the fixed-sized memory pool cannot be created dynamically by issuing a service call from a processing program.

Static fixed-size memory pool creation means defining of a fixed-size memory pool by using the static API "CRE_MPF" in the system configuration file.

Remark For details about the static API "CRE_MPF", refer to "B.5.7 Fixed-sized memory pool information".

CHAPTER 7 TIME MANAGEMENT FUNCTIONS

This chapter describes the time management functions provided by the RI850MP.

7.1 Outline

The time management functions of the RI850MP utilize timer interrupts that occur at constant intervals. These functions are provided to enable time-dependent processing.

Remark For details about service calls provided by the RI850MP for time management, refer to "13.4.9 Time management functions".

7.2 Timer Interrupts

The time management functions of the RI850MP utilize timer interrupts that occur at constant intervals to enable time-dependent processing such as updating of the system time, task timeouts, and activation of cyclic handlers.

7.2.1 Registration of timer interrupts

The RI850MP supports only static registration of timer interrupts. Therefore, timer interrupts cannot be registered dynamically by issuing a service call from a processing program.

Static timer interrupt registration means defining of timer interrupt information by using the static APIs "DEF_TIM" and "CLK_INTNO" in the system configuration file.

Remarks 1. For details about the static API "DEF_TIM", refer to "B.3.2 Base clock cycle information".

2. For details about the static API "CLK_INTNO", refer to "B.3.3 Timer interrupt information".

7.3 Cyclic Handler

A cyclic handler is a routine dedicated to cyclic processing that is activated periodically at a constant interval (activation cycle).

The RI850MP handles cyclic handlers as "non-tasks", separately from tasks. Therefore, even if the task with the highest priority in the system is currently being executed, the processing is suspended when a specified activation cycle has come, and control is passed to the cyclic handler.

7.3.1 Cyclic handler states

The RI850MP classifies the states that cyclic handler can enter into the following two types.

- Non-operational state
  A state in which the cyclic handler is not activated, even when the time specified for the cycle has passed.

- Operational state
  A state in which the cyclic handler is activated when the time specified for the cycle has passed.
  The interval up to activation for the first transition from the non-operational state to the operational state depends on whether the TA-PHS property (activation phase saved: whether the activation phase is saved) is set for the target cyclic handler.

7.3.2 Basic format of cyclic handlers

When coding a cyclic handler, use a void function with one VP_INT argument.

The argument *exinf* contains extended information specified with "Cyclic handler information".

The following shows the basic format of a cyclic handler coded in C.

```
#include    <kernel.h>
void
cychdr(VP_INT exinf){
        .........
        .........
        return;
}
```

7.3.3 Cyclic handler registration

The RI850MP supports only static registration of cyclic handlers. Therefore, timer interrupts cannot be registered dynamically by issuing a service call from a processing program.

Static cyclic handler registration means defining of cyclic handlers by using the static API "CRE_CYC" in the system configuration file.

Remark For details about the static API "CRE_CYC", refer to "B.5.8 Cyclic handler information".

CHAPTER 8 SYSTEM STATE MANAGEMENT FUNCTIONS

This chapter describes the system state management functions provided by the RI850MP.

8.1 Outline

The system state management functions provided by the RI850MP include functions to reference and manipulate system states.

Remark For details about service calls provided by the RI850MP for system state management, refer to "13.4.10 System state management functions".

CHAPTER 9 INTERRUPT MANAGEMENT FUNCTIONS

This chapter describes the interrupt management functions provided by the RI850MP.

9.1 Outline

The RI850MP provides interrupt management functions related to the interrupt handlers that are activated when an interrupt occurs.

Remark For details about service calls provided by the RI850MP for interrupt management, refer to "13.4.11 Interrupt management functions".

9.2 User-Own Coding Modules

To enable the RI850MP to support various execution environments, interrupt management processing that depends on the user execution environment is extracted as user-own coding modules and provided as sample source files.

This enhances portability for various execution environments and facilitates customization as well.

9.2.1 Interrupt mask logical OR routine

This is a routine dedicated to interrupt mask pattern processing. It is extracted in a target-dependent module, for ORing the interrupt mask pattern specified by the parameter of this user-own function and the CPU interrupt mask pattern (the value of interrupt control register EIC*n*, or interrupt mask flag EIMKn of the interrupt mask register IMR*m*) and storing the result to the interrupt mask flag EIMK*n* of the target register.

It is called when service call loc_cpu or iloc_cpu is issued from the processing program.

- Basic format

  Code an interrupt mask logical OR routine as a void type function that has one VP type argument (function name: _kernel_usr_msk_intmsk).

  The argument *p_intms* contains a pointer to an area where the interrupt mask pattern to be set is stored.

  The following shows the basic format of an interrupt mask logical OR routine coded in C.

```
#include    <kernel.h>
void
_kernel_usr_msk_intmsk(VP p_intms){
        .........
        .........
        return;
}
```

9.2.2 Interrupt mask acquisition routine

This is a routine dedicated to interrupt mask pattern acquisition. It is extracted in a target-dependent module, for storing the CPU interrupt mask pattern (the value of interrupt control register EIC*n* or interrupt mask flag EIMK*n* of the interrupt mask register IMR*m*) into the area specified by the parameter of this user-own function. It is called when the service call loc_cpu or iloc_cpu is issued from the processing program.

- Basic format

  Code an interrupt mask acquisition routine as a void type function that has one VP type argument (function name: _kernel_usr_get_intmsk).

  The argument *p_intms* contains a pointer to an area where the acquired interrupt mask pattern is to be stored.

  The following shows the basic format of an interrupt mask acquisition routine coded in C.

```
#include    <kernel.h>
void
_kernel_usr_get_intmsk(VP p_intms){
        .........
        .........
        return;
}
```

9.2.3 Interrupt mask overwrite routine

This is a routine dedicated to interrupt mask pattern writing. It is extracted in a target-dependent module, for storing the interrupt mask pattern specified by the parameter of this user-own function into the CPU interrupt mask pattern (the value of interrupt control register EIC*n* or interrupt mask flag EIMK*n* of the interrupt mask register IMR*m*). It is called when the service call unl_cpu or iunl_cpu is issued from the processing program.

- Basic format

  Code an interrupt mask overwrite routine as a void type function that has one VP type argument (function name: _kernel_usr_set_intmsk).

  The argument *p_intms* contains a pointer to an area where the interrupt mask pattern to be set is stored.

  The following shows the basic format of an interrupt mask overwrite routine coded in C.

```
#include    <kernel.h>
void
_kernel_usr_set_intmsk(VP p_intms){
        .........
        .........
        return;
}
```

9.2.4 Disable interrupt routine

This is a routine dedicated to disabling maskable interrupts. It is extracted in a target-dependent module, for changing the acknowledgment status of a specified maskable interrupt from enabled to disabled. It is called when the service call dis_int is issued from the processing program.

- Basic format

  Code a disable interrupt routine as a void type function that has one INTNO type argument (function name:_kernel_usr_dis_int).

  The argument *intno* contains the exception code that corresponds to the maskable interrupt for which acknowledgment is to be disabled.

  The following shows the basic format of a disable interrupt routine coded in C.

```
#include    <kernel.h>
void
_kernel_usr_set_dis_int(INTNO intno){
        .........
        .........
        return;
}
```

9.2.5 Enable interrupt routine

This is a routine dedicated to enabling maskable interrupts. It is extracted in a target-dependent module, for changing the acknowledgment status of a specified maskable interrupt from disabled to enabled. It is called when the service call ena_int is issued from the processing program.

- Basic format

  Code an enable interrupt routine as a void type function that has one INTNO type argument (function name:_kernel_usr_ena_int).

  The argument *intno* contains the exception code that corresponds to the maskable interrupt for which acknowledgment is to be enabled.

  The following shows the basic format of an enable interrupt routine coded in C.

```
#include    <kernel.h>
void
_kernel_usr_set_ena_int(INTNO intno){
        .........
        .........
        return;
}
```

9.2.6 Interrupt entry routine

This is a routine dedicated to entry processing. It is extracted in a target-dependent module, for assignment of instructions to branch to interrupt preprocessing, for a handler address to which the CPU forcibly passes control when an interrupt occurs.

However, when an interrupt entry routine is defined with "Interrupt handler information", corresponding to an exception code, the interrupt entry is included in the entry file created by executing the configurator.

If customization of interrupt entry processing is unnecessary, use of the relevant entry file therefore makes coding of interrupt entry processing unnecessary.

- Basic format

  When coding an interrupt entry routine, assign instructions to branch to preprocessing code, for a handler address to which the CPU forcibly passes control when an interrupt occurs.

  The following shows the basic format of an interrupt entry routine in assembly.

  [CX version]

```
inhno       .cseg   text        -- inhno : Exception name
jr          __kernel_int_entry  -- Branch to interrupt preprocessing
```

  [CCV850E version]

```
.org        inthdr              -- inthdr : Activation address
jr          __kernel_int_entry  -- Branch to interrupt preprocessing
```

9.3 Interrupt Handlers

An interrupt handler is a routine dedicated to interrupt servicing that is activated when an interrupt occurs.

The RI850MP handles interrupt handlers as "non-tasks", separately from tasks. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when an interrupt occurs, and the control is passed to the interrupt handler.

9.3.1 Basic format of interrupt handlers

Code an interrupt handler as a void type function that has no arguments.

The following shows the basic format of an interrupt handler coded in C.

```
#include    <kernel.h>
void
inthdr(void){
        .........
        .........
        return;
}
```

9.3.2 Interrupt handler registration

The RI850MP supports only static registration of interrupt handlers. Therefore, interrupt handlers cannot be registered dynamically by issuing a service call from a processing program.

Static interrupt handler registration means defining of interrupt handlers by using the static API "DEF_INH" in the system configuration file.

Remarks 1. For details about the static API "DEF_INH", refer to "B.5.9 Interrupt handler information".

2. The RI850MP provides TIME MANAGEMENT FUNCTIONS by utilizing timer interrupts that occur at constant intervals. For this reason, it is not possible to register an interrupt handler for an exception code defined with "Timer interrupt information".

CHAPTER 10 SYSTEM CONFIGURATION MANAGEMENT FUNCTIONS

This chapter describes the system configuration management functions provided by the RI850MP.

10.1 Outline

The system configuration management functions provided by the RI850MP include CPU exception handlers, which are activated when a CPU exception occurs, and initialization routines called from SYSTEM INITIALIZATION ROUTINE.

10.2 User-Own Coding Modules

To enable the RI850MP to support various execution environments, system configuration management functions that depend on the user execution environment are extracted in a user-own coding module and provided as sample source files.

This enhances portability for various execution environments and facilitates customization as well.

10.2.1 CPU exception entry routine

This is a routine dedicated to entry processing. It is extracted in a target-dependent module, for assignment of instructions to branch to interrupt preprocessing, for a handler address to which the CPU forcibly passes control when a CPU exception occurs.

However, when a CPU exception entry routine is defined with CPU exception handler information, corresponding to an exception code, the CPU exception entry is included in the entry file created by executing the configurator.

If customization of CPU exception entry processing is unnecessary, use of the relevant entry file therefore makes coding of CPU exception entry processing unnecessary.

- Basic format

  When coding a CPU exception entry routine, assign instructions to branch to preprocessing code, for a handler address to which the CPU forcibly passes control when a CPU exception occurs.

  The following shows the basic format of a CPU exception entry routine in assembly.

  [CX version]

```
excno       .cseg   text        -- excno : Exception name
jr          __kernel_int_entry  -- Branch to interrupt preprocessing
```

  [CCV850E version]

```
.org        exchdr              -- exchdr : Activation address
jr          __kernel_int_entry  -- Branch to interrupt preprocessing
```

10.3 CPU Exception Handlers

A CPU exception handler is a routine dedicated to handling a CPU exception, which is called when a CPU exception occurs.

The RI850MP handles a CPU exception handler as a "non-task", separately from tasks. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when a CPU exception occurs, and control is passed to the CPU exception handler.

10.3.1 Basic format of CPU exception handlers

Code a CPU exception handler as a void type function that has no arguments.

The following shows the basic format of a CPU exception handler coded in C.

```
#include    <kernel.h>
void
exchdr(void){
        .........
        .........
        return;
}
```

10.3.2 CPU exception handler registration

The RI850MP supports only static registration of CPU exception handlers. Therefore, CPU exception handlers cannot be registered dynamically by issuing a service call from a processing program.

Static CPU exception handler registration means defining of CPU exception handlers by using the static API "DEF_EXC" in the system configuration file.

Remark For details about the static API "DEF_EXC", refer to "B.5.10 CPU exception handler information".

10.4 Initialization Routine

The initialization routine is a routine dedicated to initialization processing that is extracted as a user-own coding module to initialize hardware and software that depends on the user execution environment. It is called from the SYSTEM INITIALIZATION ROUTINE.

10.4.1 Basic format of initialization routines

Code an initialization routine as a void type function that has one VP_INT type argument.
The argument *exinf* contains extended information defined with "Initialization routine information".
The following shows the basic format of an initialization routine coded in C.

```
#include    <kernel.h>
void
inirtn(VP_INT exinf){
        .........
        .........
        return;
}
```

10.4.2 Initialization routine registration

The RI850MP supports only static registration of initialization routines. Therefore, CPU exception handlers cannot be registered dynamically by issuing a service call from a processing program.

Static initialization routine registration means defining of initialization routines by using the static API "ATT_INI" in the system configuration file.

Remark For details about the static API "ATT_INI", refer to "B.5.11 Initialization routine information".

CHAPTER 11 SCHEDULER

This chapter describes the scheduler of the RI850MP.

11.1 Outline

The scheduling functions provided by the RI850MP consist of functions to manage and decide the order in which tasks are executed by monitoring. This is done by monitoring the transition states of dynamically changing tasks, so that the right to use the CPU is given to the optimum task.

11.2 Drive Method

The RI850MP employs an event-driven system in which the scheduler is activated when an event (trigger) occurs. The following lists the events that cause the RI850MP to activate the scheduler.

- Issuance of a service call that may cause a task state transition
- Issuance of an instruction for returning from a non-task (cyclic handler, interrupt handler, etc.)
- Occurrence of a timer interrupt used to implement time management functions

11.3 Scheduling Methods

As task scheduling methods, the RI850MP employs the priority level method, which uses the priority level (current priority) defined for each task, and the FCFS method, which uses the time elapsed from the point when a task becomes subject to RI850MP scheduling.

- Priority level method
  The task that has the highest priority level (current priority) of all the tasks that have entered an executable state (RUNNING state or READY state) is selected and given the right to use the CPU.

- FCFS method
  On a first come first served (FCFS) basis, the task for which the longest interval of time has elapsed since it entered an executable state is selected as the task to which the right to use the CPU is granted.
  FCFS task scheduling is executed when multiple tasks with the highest priority level (current priority) according to the selection criteria of the priority level method exist simultaneously.

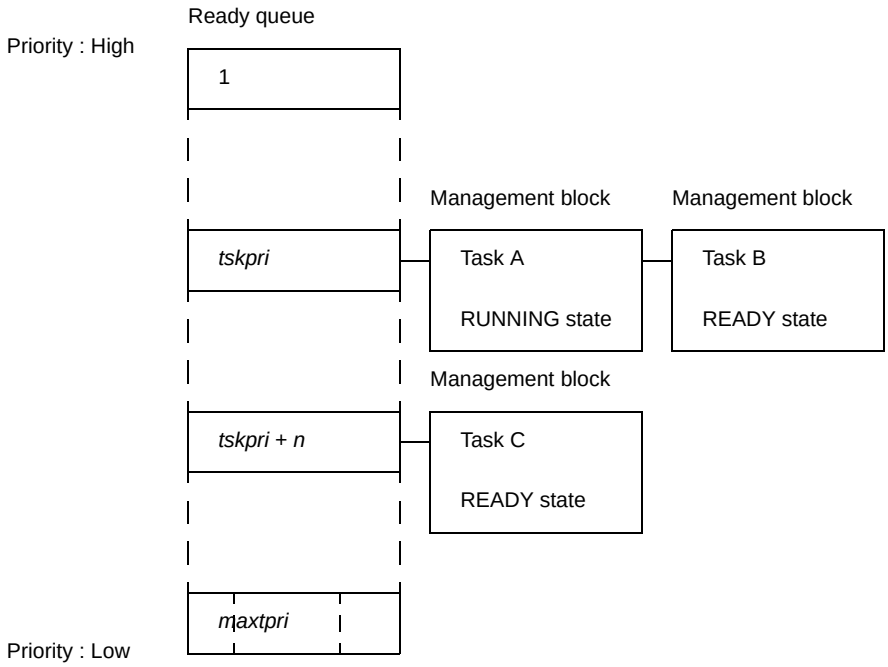
11.4 Ready Queue

The RI850MP uses a "ready queue" to implement task scheduling.

The RI850MP ready queue is a hash table that uses priority as the key. The management blocks of tasks that have entered an executable state (READY state or RUNNING state) are queued in FIFO order at the respective priority position.

Therefore, the RI850MP scheduler executes task detection processing from the highest priority level of the ready queue. When it detects tasks in the ready queue, it gives the right to use the CPU the first task of the proper priority level.

Figure 11-1. Ready Queue



11.4.1 Ready queue creation

The RI850MP supports only static ready queue creation. Therefore, the ready queue cannot be created dynamically by issuing a service call from a processing program.

Static ready queue creation means defining information related to priority by using the static API "MAX_PRI" in the system configuration file.

Remark For details about the static API "MAX_PRI", refer to "B.3.5 Maximum priority information".

11.5 Scheduling Lock Function

On the RI850MP, service calls provided for SYSTEM STATE MANAGEMENT FUNCTIONS (loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, dis_dsp, ena_dsp) can be used to manipulate scheduler states explicitly from the processing program, to disable and enable dispatch processing.

11.6 Idle Routine

The idle routine is a routine dedicated to idle processing that is extracted as a user-own coding module to utilize the standby function provided by the CPU (to enable low power consumption systems). It is called from the scheduler when there is no longer any task subject to scheduling by the RI850MP remaining in the system.

11.6.1 Basic format of idle routine

Code the idle routine as a void type function that has no arguments.

The following shows the basic format of an idle routine coded in C.

```
#include    <kernel.h>
void
idlrtn(void){
        .........
        .........
        return;
}
```

11.6.2 Idle routine registration

The RI850MP supports only static registration of idle routines. Therefore, an idle routine cannot be registered dynamically by issuing a service call from a processing program.

Static idle routine registration means defining of an idle routine by using the static API "VATT_IDL" in the system configuration file.

Remark For details about the static API "VATT_IDL", refer to "B.5.12 Idle routine information".

CHAPTER 12 SYSTEM INITIALIZATION ROUTINE

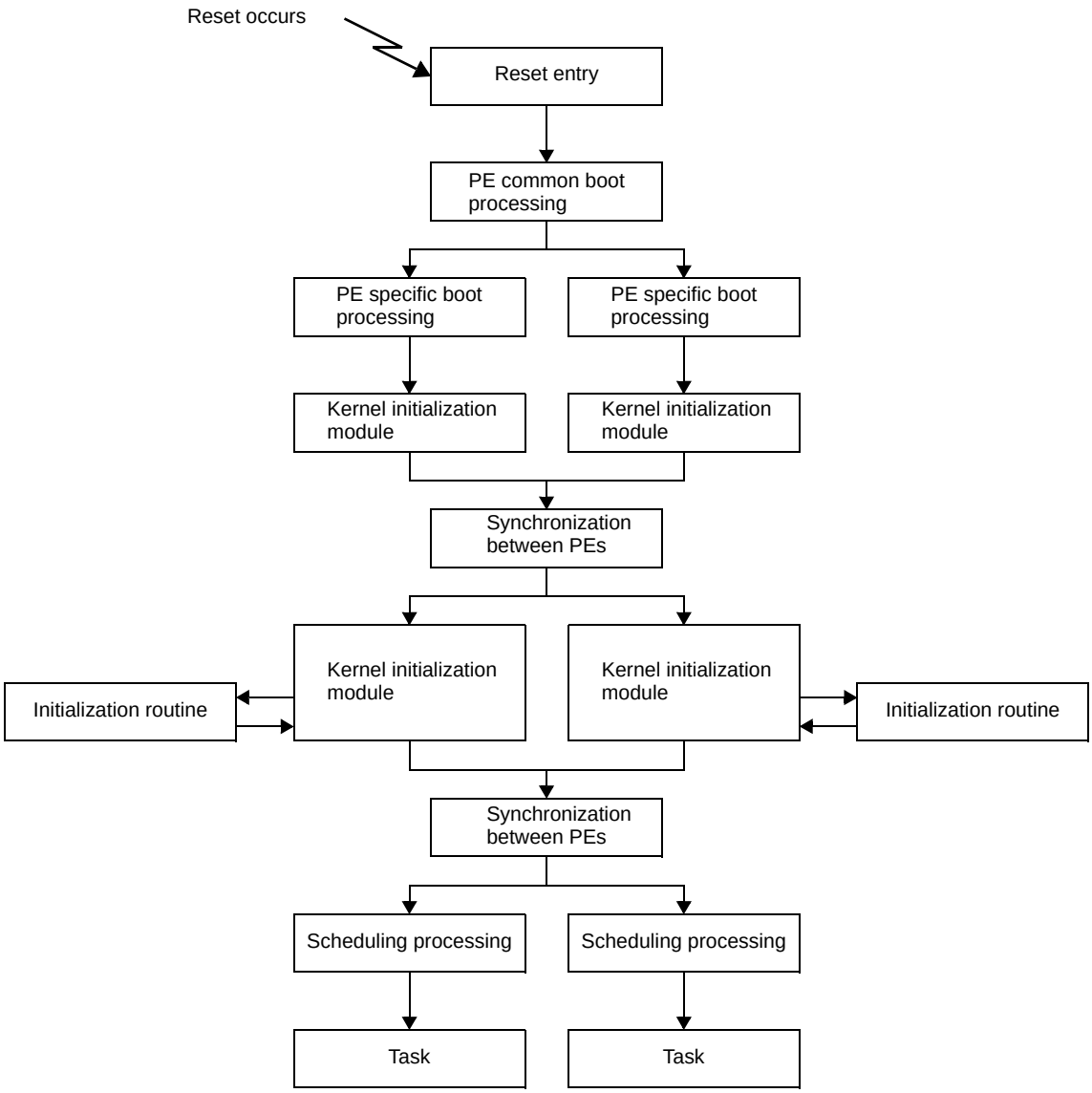
This chapter describes the system initialization routine provided by the RI850MP.

12.1 Outline

The system initialization routine of the RI850MP provides initialization of the hardware and software resources required for processing by the RI850MP from the time when a reset interrupt occurs until control is passed to a processing program (task).

The following shows the processing flow from when a reset interrupt occurs until control is passed to the processing program.

Figure 12-1. Flow of System Initialization Processing



12.2 User-Own Coding Modules

To enable the RI850MP to support various execution environments, processing that depends on the user execution environment is extracted as user-own coding modules and provided as sample source files.

This enhances portability for various execution environments and facilitates customization as well.

12.2.1 Reset entry routines

A reset entry routine is a routine dedicated to entry processing that is extracted for assignment of instructions to branch to PE common boot processing, for a handler address to which the CPU forcibly passes control when a reset occurs.

- Basic format

  When coding a reset entry routine, assign processing to branch to PE common boot processing, for a handler address to which the CPU forcibly passes control when a reset occurs.

  However, when PE common boot processing is defined as a CPU exception handler with "CPU exception handler information", the reset entry is included in the entry file created by executing the configurator. Therefore, if customization of the reset entry processing is unnecessary, use of the relevant entry file makes coding of reset entry processing unnecessary.

  The following shows the basic format of reset entry processing in assembly.

  [CX version]

```
RESET       .cseg   text
jr          __boot              -- Branch to PE common boot processing
```

  [CCV850E version]

```
.org        0x00000000
jr          __boot              -- Branch to PE common boot processing
```

12.2.2 Boot processing

Boot processing is a routine dedicated to initialization processing that is extracted as a user-own coding module to initialize the minimum required hardware for the RI850MP to perform processing. It is comprised of PE common processing and PE specific boot processing.

- PE common boot processing

  PE common boot processing is a routine dedicated to initialization processing that is extracted as a user-own coding module to initialize the minimum hardware required in common by PEs. It is called from reset entry processing. The following shows the basic format of PE common boot processing in assembly.

```
        .text
        .align      0x4
        .global     __boot
__boot:
        mov         __boot_PE1, r10
        mov         __boot_PE2, r20
        mov         PEID, r1
        ld.h        0[r1], r2
        cmp         1, r2
        cmove       r10, r20, r21
        jmp         [r21]               -- Branch to PE specific boot processing
```

- PE specific boot processing

  PE specific boot processing is a routine dedicated to initialization processing that is extracted as a user-own coding module to initialize the minimum hardware and software required by a specific PE. It is called from PE common boot processing.

  The following shows the basic format of PR specific boot processing in assembly.

  [CX version]

```
        .text
        .align      0x4
        .global     __boot_PEn
__boot_PEn:
        .extern     __kernel_sit
        .........
        .........
        mov         #__kernel_sit, r6
        jarl        __kernel_start, lp  -- Branch to kernel initialization module
```

[CCV850E version]

```
        .text
        .align      0x4
        .global     __boot_PEn
__boot_PEn:
        .extern     __kernel_sit
        .........
        .........
        mov         __kernel_sit, r6
        jarl        __kernel_start, lp  -- Branch to kernel initialization module
```

12.3 Kernel Initialization Module

The kernel initialization module is a dedicated initialization processing routine provided for initializing the minimum required hardware for the RI850MP to perform processing. It is called from PE specific boot processing.

The following processing is executed in the kernel initialization module.

- Floating decimal point settings/Status register initialization
- System time initialization
- Task creation and activation
- Semaphore creation
- Eventflag creation
- Data queue creation
- Mailbox creation
- Mutex creation
- Fixed-sized memory pool creation
- Cyclic handler registration and activation
- Interrupt handler registration
- CPU exception handler registration
- Initialization routine registration
- Idle routine registration
- Initialization routine calling
- Scheduler activation

Remark The kernel initialization module is included in system initialization processing provided by the RI850MP. The user is therefore not required to code the kernel initialization module.

12.4 Initialization Routine

The initialization routine is a routine dedicated to initialization processing that is extracted as a user-own coding module to initialize hardware and software that depends on the user execution environment. It is called from the Kernel Initialization Module.

Remark For details about the initialization routine, refer to "10.4 Initialization Routine".

CHAPTER 13 SERVICE CALLS

This chapter describes the service calls supported by the RI850MP.

13.1 Outline

Service calls are provided by the RI850MP to manipulate the resources (tasks, semaphores, data queues, etc.) managed by the RI850MP from a processing program written by the user.
The service calls provided by the RI850MP are listed below by function.

- Task management functions
  act_tsk, iact_tsk, can_act, ican_act, ext_tsk, ter_tsk, chg_pri, ichg_pri, get_pri, iget_pri, ref_tsk, iref_tsk

- Task dependent synchronization functions
  slp_tsk, tslp_tsk, wup_tsk, iwup_tsk, can_wup, ican_wup, rel_wai, irel_wai, sus_tsk, isus_tsk, rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk, dly_tsk

- Synchronization and communication functions (semaphores)
  sig_sem, isig_sem, wai_sem, pol_sem, ipol_sem, twai_sem, ref_sem, iref_sem

- Synchronization and communication functions (eventflags)
  set_flg, iset_flg, clr_flg, iclr_flg, wai_flg, pol_flg, ipol_flg, twai_flg, ref_flg, iref_flg

- Synchronization and communication functions (data queues)
  snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq, rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq, ref_dtq, iref_dtq

- Synchronization and communication functions (mailboxes)
  snd_mbx, isnd_mbx, rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx, ref_mbx, iref_mbx

- Extended synchronization and communication functions
  loc_mtx, ploc_mtx, tloc_mtx, unl_mtx, ref_mtx, iref_mtx

- Memory pool management functions
  get_mpf, pget_mpf, ipget_mpf, tget_mpf, rel_mpf, irel_mpf, ref_mpf, iref_mpf

- Time management functions
  set_tim, iset_tim, get_tim, iget_tim, sta_cyc, ista_cyc, stp_cyc, istp_cyc, ref_cyc, iref_cyc

- System state management functions
  rot_rdq, irot_rdq, get_tid, iget_tid, loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, dis_dsp, ena_dsp, sns_ctx, sns_loc, sns_dsp, sns_dpn

- Interrupt management functions
  dis_int, ena_int, chg_ipm, ichg_ipm, get_ipm, iget_ipm

13.1.1 Calling a service call

The method for calling service calls from processing programs coded either in C or assembly language is described below.

- Processing programs coded in the C language
  The service call's parameters are passed to the RI850MP using the same calling convention as an ordinary C function, and the corresponding process is executed.

- Processing programs coded in assembly language
  After setting the parameters and return address in accordance with the function calling convention of the C compiler package being used, make the call via a jarl instruction to pass the service-call parameters to the RI850MP, and execute the corresponding process.

Remark To call the service calls provided by the RI850MP from a processing program, the header files listed below must be coded (include processing).

kernel.h : Standard header file
kernel_id.h : System information header file

13.2 Data Macros

Below are shown the data macros used when processing programs issue service calls provided by the RI850MP.

13.2.1 Data types

The data types are defined in the header file "types.h" that is called from standard header file "kernel.h".

Table 13-1. Data Types

Macro	Type	Description
B	signed char	Signed 8-bit integer
H	signed short	Signed 16-bit integer
W	signed long	Signed 32-bit integer
UB	unsigned char	Unsigned 8-bit integer
UH	unsigned short	Unsigned 16-bit integer
UW	unsigned long	Unsigned 32-bit integer
VB	signed char	8-bit value with undetermined data type
VH	signed short	16-bit value with undetermined data type
VW	signed long	32-bit value with undetermined data type
VP	void *	Pointer to value with undetermined data type
FP	void (*)	Startup address of processing program (pointer)
INT	signed int	Signed 32-bit integer
UINT	unsigned int	Unsigned 32-bit integer
BOOL	signed int	Boolean value (TRUE or FALSE)
FN	signed short	Function code of service call
ER	signed long	Return value of service call
ID	signed short	Object ID
ATR	unsigned short	Object attribute
STAT	unsigned short	Object status
MODE	unsigned short	Operation mode of service call
PRI	signed short	Object priority
SIZE	unsigned long	Size of the memory area
TMO	signed long	Wait time
RELTIM	unsigned long	Relative time
SYSTIM	-	See "13.3.11 System time SYSTIM" for details about SYSTIM
VP_INT	signed int	Pointer to undetermined type or signed 32-bit integer
ER_BOOL	signed long	Return value of service call or Boolean value (TRUE or FALSE)
ER_ID	signed long	Return value of service call or object ID
ER_UINT	unsigned int	Return value of service call or unsigned 32-bit integer
FLGPTN	unsigned int	Bit pattern
T_MSG	-	See "13.3.9 Message (no priority) T_MSG" for details about T_MSG

Macro	Type	Description
T_MSG_PRI	-	See "13.3.10 Message (with priority) T_MSG_PRI" for details about T_MSG_PRI
INTNO	unsigned short	Exception cause code
INTPMR	unsigned short	Register value
PE_ID	unsigned char	PE number

13.2.2 Return values

The return values are defined in the header files "errcd.h" and "options.h" that are called from standard header file "kernel.h".

Table 13-2. Return Values

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function
E_PAR	-17	Parameter is invalid
E_ID	-18	ID is invalid
E_CTX	-25	Context error
E_ILUSE	-28	Invalid service call use
E_OBJ	-41	Object status error
E_QOVR	-43	Queuing overflow
E_RLWAI	-49	Forced cancellation of WAITING state
E_TMOUT	-50	Polling failure or timeout
TRUE	1	True
FALSE	0	False
NULL	0	Invalid pointer

13.2.3 Object attributes

The object attributes are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-3. Object Attributes

Macro	Num.	Description
TA_HLNG	0x0	C language
TA_ASM	0x1	Assembly language
TA_TFIFO	0x0	Tasks queued in FIFO order
TA_TPRI	0x1	Tasks queued in priority order
TA_MFIFO	0x0	Messages queued in FIFO order
TA_MPRI	0x2	Messages queued in priority order
TA_ACT	0x2	READY state
TA_WSGL	0x0	1 task
TA_WMUL	0x2	Multiple tasks
TA_CLR	0x4	Bit pattern cleared if the request conditions are met
TA_STA	0x2	Operating status
TA_PHS	0x4	Stores an activation phase
TA_ENAINT	0x0	Enables interrupts
TA_DISINT	0x8000	Disables interrupts

13.2.4 Task wait time

The task wait times are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-4. Task Wait Time

Macro	Num.	Description
TMO_POL	0	Polling
TMO_FEVR	-1	Wait forever

13.2.5 Task request conditions

The task request conditions are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-5. Task Request Conditions

Macro	Num.	Description
TWF_ANDW	0x0	AND wait
TWF_ORW	0x1	OR wait

13.2.6 Current task status

The current task statuses are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-6. Current Task Status

Macro	Num.	Description
TTS_RUN	0x1	RUNNING state
TTS_RDY	0x2	READY state
TTS_WAI	0x4	WAITING state
TTS_SUS	0x8	SUSPENDED state
TTS_WAS	0xC	WAITING-SUSPENDED state
TTS_DMT	0x10	DORMANT state

13.2.7 Task wait causes

The task wait causes (wait state types) are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-7. Task Wait Causes

Macro	Num.	Description
TTW_NONE	0x0	Not waiting
TTW_SLP	0x1	Sleeping state
TTW_DLY	0x2	Delayed state
TTW_SEM	0x4	WAITING state for a semaphore resource
TTW_FLG	0x8	WAITING state for an eventflag
TTW_SDTQ	0x10	Sending WAITING state for a data queue
TTW_RDTQ	0x20	Receiving WAITING state for a data queue
TTW_MBX	0x40	WAITING state for a mailbox
TTW_MTX	0x80	WAITING state for a mutex
TTW_MPF	0x2000	WAITING state for a fixed-sized memory block

13.2.8 Current state of cyclic handler

The current cyclic handler states are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-8. Current State of Cyclic Handler

Macro	Num.	Description
TCYC_STP	0x0	Non-operational state
TCYC_STA	0x1	Operational state

13.2.9 Other constants

Other constants are defined in the header file "options.h" that is called from standard header file "kernel.h".

Table 13-9. Other Constants

Macro	Num.	Description
TSK_SELF	0	Current task
TSK_NONE	0	Task is not queued
TPRI_SELF	0	Current priority of task
TPRI_INI	0	Initial priority of task

13.2.10 Conditional compilation macros

The RI850MP provides the following macros for conditional compilation.

Table 13-10. Conditional Compilation Macros

Macro	Description
__cx__	Use CX as C compiler package
__ccv850e__	Use CCV850E as C compiler package
__v850e2m__	Use V850E2M as target device

13.3 Data Structures

Below are shown the data structures used when processing programs issue service calls provided by the RI850MP.

13.3.1 Task information T_RTSK

Task information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rtsk {
        STAT    tskstat;    /* Current state */
        PRI     tskpri;     /* Current priority */
        PRI     tskbpri;    /* Reserved for future use */
        STAT    tskwait;    /* Wait cause */
        ID      wobjid;     /* Object ID */
        TMO     lefttmo;    /* Remaining time until timeout */
        UINT    actcnt;     /* Activation request nesting count */
        UINT    wupcnt;     /* Wake-up request nesting count */
        UINT    suscnt;     /*Forced wait request nesting count */
        ATR     tskatr;     /* Attribute */
        PRI     itskpri;    /* Initial priority */
        PE_ID   peid;       /* PE number */
} T_RTSK;
```

Below are details about the task information.

- tskstat

  This stores the task's current status.

Macro	Num.	Description
TTS_RUN	0x1	RUNNING state
TTS_RDY	0x2	READY state
TTS_WAI	0x4	WAITING state
TTS_SUS	0x8	SUSPENDED state
TTS_WAS	0xC	WAITING-SUSPENDED state
TTS_DMT	0x10	DORMANT state

- tskpri

  This stores the task's current priority.

  If the task is DORMANT state, the initial priority is stored.

- tskbpri

  This area is reserved by the system.

- tskwait

This stores the tasks wait cause (wait state type).

If the task is in WAITING state or WAITING-SUSPENDED state, TTW_NONE (=0x0) is stored.

Macro	Num.	Description
TTW_NONE	0x0	Not waiting
TTW_SLP	0x1	Sleeping state
TTW_DLY	0x2	Delayed state
TTW_SEM	0x4	WAITING state for a semaphore resource
TTW_FLG	0x8	WAITING state for an eventflag
TTW_SDTQ	0x10	Sending WAITING state for a data queue
TTW_RDTQ	0x20	Receiving WAITING state for a data queue
TTW_MBX	0x40	WAITING state for a mailbox
TTW_MTX	0x80	WAITING state for a mutex
TTW_MPF	0x2000	WAITING state for a fixed-sized memory block

Remark When a tasks transitions to timeout wait status due to the issuance of tslp_tsk, twai_sem, twai_flg, or the like, the logical sum of the value indicating the WAITING state (e.g. TTW_SLP, TTW_SEM, or TTW_FLG) and TTW_DLY is stored.

- wobjid

This stores the ID of the object (e.g. semaphore, eventflag, or data queue) that caused the task to go into the wait state.

A value of 0 will be stored if the state of the task is other than WAITING state for semaphore resource, WAITING state for a eventflag, Sending WAITING state for a data queue, Receiving WAITING state for a data queue, WAITING state for a mailbox, WAITING state for a mutex, or WAITING state for a fixed-size memory block.

- lefttmo

This stores the time remaining (in milliseconds) until the Delayed state (transition consequent to issuance of tslp_tsk, dly_tsk, twai_sem, or the like) will be cleared.

A value of 0 will be stored if the state of the task is other than Delayed state.

Remark A value of TMO_FEVR (= -1) will be stored if the state of the task is wait forever.

- actcnt

This stores the activation request nesting level count (nesting level count of act_tsk/iact_tsk) maintained by the task.

If the task is DORMANT state, a value of 0 will be stored.

- wupcnt

This stores the wake-up request nesting level count (nesting level count of wup_tsk/iwup_tsk) maintained by the task.

If the task is DORMANT state, a value of 0 will be stored.

- suscnt

This stores the force wait request nesting level count (nesting level count of sus_tsk/isus_tsk) maintained by the task.

If the task is DORMANT state, a value of 0 will be stored.

- tskatr

This stores the task attributes (language in which task is coded, initial state, and initial interrupt state).

- Language in which task is coded (bit 0)

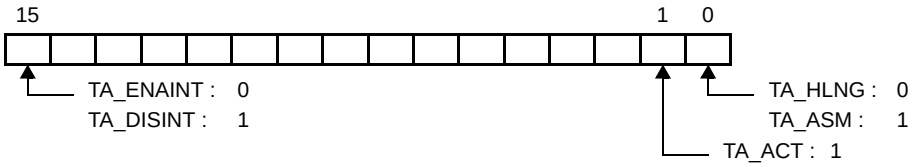
TA_HLNG : C language
TA_ASM : Assembly language

- Initial task state (bit 1)

TA_ACT : READY state

- Initial task interrupt state (bit 15)

TA_ENAINT : Interrupts enabled
TA_DISINT : Interrupts disabled



Remark If the initial state of the task is DORMANT state, 0 will be stored in bit 1.

- itskpri

This stores the task's initial priority.

- peid

This stores the PE number of the processor element to which the task belongs.

13.3.2 Semaphore information T_RSEM

Semaphore information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rsem {
        ID      wtskid;   /* Existence of waiting task */
        UINT    semcnt;   /* Current resource count */
        ATR     sematr;   /* Attribute */
        UINT    maxsem;   /* Maximum resource count */
} T_RSEM;
```

Details of the semaphore information are described below.

- wtskid

  This stores whether there are any tasks queued in the semaphore wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks are queued in the wait queue
-	Other than 0	The ID of the first task queued in the wait queue

- semcnt

  This stores the current number of semaphore resources.

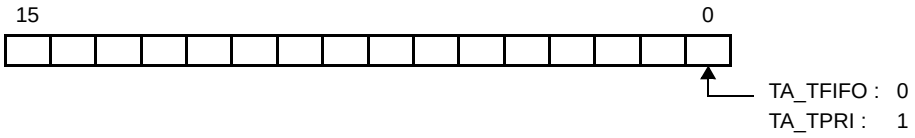
- sematr

  This stores the semaphore attribute (queuing method).

  - Task queuing method (bit 0)

    TA_TFIFO : Order in which resource acquisitions were requested

    TA_TPRI : Order of task priority



- maxsem

  This stores the maximum number of semaphore resources.

13.3.3 Eventflag information T_RFLG

Eventflag information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rflg {
        ID      wtskid;   /* Existence of waiting task */
        FLGPTN  flgptn;   /* Current bit pattern */
        ATR     flgatr;   /* Attribute */
} T_RFLG;
```

Details of the eventflag information are described below.

- wtskid

  This stores whether there are any tasks queued in the eventflag wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks are queued in the wait queue
-	Other than 0	The ID of the first task queued in the wait queue

- flgptn

  This stores the current bit pattern of the eventflag.

- flgatr

  This stores the eventflag attributes (queuing method, maximum number of tasks, and clearing method).

  - Task queuing method (bit 0)

    TA_TFIFO : Order in which determination of bit pattern was requested

    TA_TPRI : Order of task priority

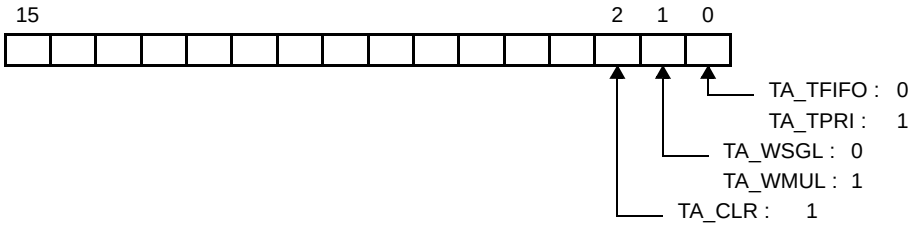
  - Maximum number of tasks that can be queued (bit 1)

    TA_WSGL : 1 task

    TA_WMUL : Multiple tasks

  - Method of clearing bit pattern (bit 2)

    TA_CLR : Bit pattern cleared if the request conditions are met



Remark If the bit-pattern clearing method is "Do not clear bit pattern when request conditions are met", then 0 will be stored in bit 2.

13.3.4 Data queue information T_RDTQ

Data queue information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rdtq {
        ID      stskid;     /* Existence of tasks awaiting sending */
        ID      rtskid;     /* Existence of tasks awaiting receiving */
        UINT    sdtqcnt;    /* Data count */
        ATR     dtqatr;     /* Attribute */
        UINT    dtqcnt;     /* Max data count */
} T_RDTQ;
```

Below are details about the data queue information.

- stskid

  This stores whether there are any tasks awaiting sending in the data wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks awaiting sending are queued in the wait queue
-	Other than 0	The ID of the first task awaiting sending queued in the wait queue

- rtskid

  This stores whether there are any tasks awaiting reception in the data wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks awaiting reception are queued in the wait queue
-	Other than 0	The ID of the first task awaiting reception queued in the wait queue

- sdtqcnt

  This stores the number of data entries queued in the data wait queue.

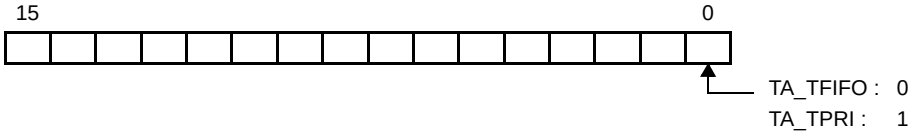
- dtqatr

  This stores the data queue attribute (queuing method).

  - Task queuing method (bit 0)

    TA_TFIFO : Order in which data-send requests were made

    TA_TPRI : Order of task priority



- dtqcnt

  This stores the maximum number of data entries that can be queued in the data wait queue.

13.3.5 Mailbox information T_RMBX

Mailbox information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rmbx {
        ID      wtskid;     /* Existence of waiting task */
        T_MSG   *pk_msg;    /* Existence of waiting message */
        ATR     mbxatr;     /* Attribute */
} T_RMBX;
```

Below are details about the mailbox information.

- wtskid

This stores whether there are any tasks queued in the mailbox wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks are queued in the wait queue
-	Other than 0	The ID of the first task queued in the wait queue

- pk_msg

This stores whether there are any messages queued in the mailbox wait queue.

Macro	Num.	Description
NULL	0	No messages are queued in the wait queue
-	Other than 0	The start address of the first message queued in the wait queue

- mbxatr

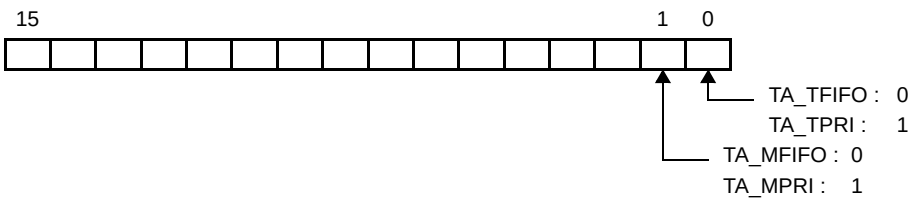
This stores the mailbox attribute (queuing method).

- Task queuing method (bit 0)

TA_TFIFO : Order in which message-receipt requests were made
TA_TPRI : Order of task priority

- Message queuing method (bit 1)

TA_MFIFO : Order in which message-send requests were made
TA_MPRI : Order of message priority



13.3.6 Mutex information T_RMTX

Mutex information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rmtx {
        ID      htskid;     /* Existence of tasks to acquire */
        ID      wtskid;     /* Existence of waiting task */
        ATR     mtxatr;     /* Attribute */
        PRI     ceilpri;    /* Reserved for future use */
} T_RMTX;
```

Below are details about the mutex information.

- htskid

  This stores whether any tasks have acquired the mutex.

Macro	Num.	Description
TSK_NONE	0	No tasks have acquired it
-	Other than 0	ID of task that has acquired it

- wtskid

  This stores whether there are any tasks queued in the mutex wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks are queued in the wait queue
-	Other than 0	The ID of the first task queued in the wait queue

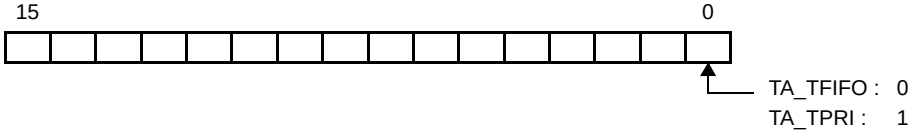
- mtxatr

  This stores the mutex attribute (queuing method).

  - Task queuing method (bit 0)

    TA_TFIFO : Order in which mutex acquisition was requested

    TA_TPRI : Order of task priority



- ceilpri

  This area is reserved by the system.

13.3.7 Fixed-sized memory pool information T_RMPF

Fixed-sized memory pool information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rmpf {
        ID      wtskid;     /* Existence of waiting task */
        UINT    fblkcnt;    /* Number of available blocks remaining */
        ATR     mpfatr;     /* Attribute */
} T_RMPF;
```

Below are details about the fixed-sized memory pool information.

- wtskid

  This stores whether there are any tasks queued in the fixed-sized memory pool wait queue.

Macro	Num.	Description
TSK_NONE	0	No tasks are queued in the wait queue
-	Other than 0	The ID of the first task queued in the wait queue

- fblkcnt

  This stores the number of remaining blocks that can be acquired.

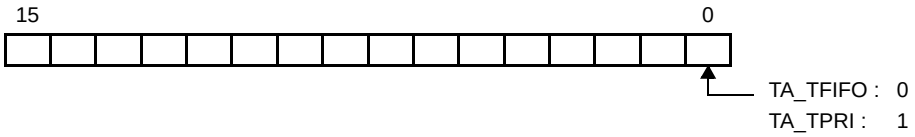
- mpfatr

  This stores the fixed-sized memory pool attribute (queuing method).

  - Task queuing method (bit 0)

    TA_TFIFO : Order in which fixed-size memory block acquisition was requested

    TA_TPRI : Order of task priority



13.3.8 Cyclic handler information T_RCYC

Cyclic handler information is defined in the header file "packet.h" that is called from standard header file "kernel.h".

```
typedef struct  t_rcyc {
        STAT    cycstat;    /* Current state */
        RELTIM  lefttim;    /* Remaining time */
        ATR     cycatr;     /* Attribute */
        RELTIM  cyctim;     /* Activation cycle */
        RELTIM  cycphs;     /* Starting phase */
        PE_ID   peid;       /* PE number */
} T_RCYC;
```

The details of cyclic handler information are described below.

- cycstat

  This stores the cyclic handler's current status.

Macro	Num.	Description
TCYC_STP	0x0	Non-operational state
TCYC_STA	0x1	Operational state

- lefttim

  This stores the remaining time (in milliseconds) until the cyclic handler will be activated next.

- cycatr

  This stores the cyclic handler attributes (language in which task is coded, initial state, and storage flag).

  - Language in which cyclic handler is coded (bit 0)

    TA_HLNG : C language

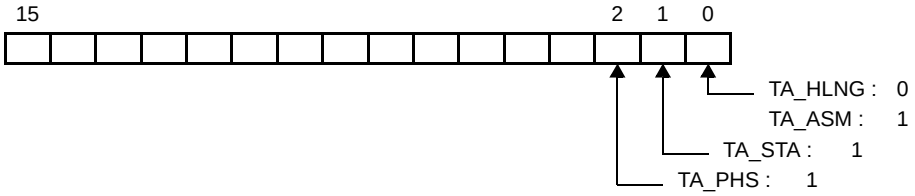
    TA_ASM : Assembly language

  - Initial state of cyclic handler (bit 1)

    TA_STA : Operational state

  - Whether activation phase has been stored (bit 2)

    TA_PHS : Activation phase stored



Remarks 1. If the initial state of the cyclic handler is stopped, 0 will be stored in bit 1.

2. If the activation-phase storage flag of the cyclic handler is "activation phase not stored", then 0 will be stored in bit 2.

- cyctim
  This stores the activation cycle (in milliseconds) of the cyclic handler.

- cycphs
  This stores the activation phase (in milliseconds) of the cyclic handler.

- peid
  This stores the PE number of the processor element to which the cyclic handler belongs.

13.3.9 Message (no priority) T_MSG

Messages (no priority) are defined in the header file "types.h" that is called from standard header file "kernel.h".

```
typedef struct  t_msg {
        struct  t_msg   *msgque;    /* Reserved for future use */
        .........                   /* Message body */
        .........
} T_MSG;
```

Details of messages (no priority) are described below.

- msgque
  This area is reserved by the system.

- .....
  The message body is stored here.
  The structure, data types, and the like of the message body are specified between the sending and receiving programs.

13.3.10 Message (with priority) T_MSG_PRI

Messages (with priority) are defined in the header file "types.h" that is called from standard header file "kernel.h".

```
typedef struct  t_msg_pri {
        T_MSG   msgque;             /* Reserved for future use */
        PRI     msgpri;             /* Priority */
        .........                   /* Message body */
        .........
} T_MSG_PRI;
```

Details of messages (with priority) are described below.

- msgque
  This area is reserved by the system.

- msgpri
  The message priority is stored here.

  Remark In the RI850MP, messages with lower priority numbers have higher priority.

- .....
  The message body is stored here.
  The structure, data types, and the like of the message body are specified between the sending and receiving programs.

13.3.11 System time SYSTIM

System times are defined in the header file "types.h" that is called from standard header file "kernel.h".

```
typedef struct  t_systim {
        UW      ltime;    /* System time (lower 32 bits) */
        UH      utime;    /* System time (higher 16 bits) */
} SYSTIM;
```

Details about system time are described below.

- ltime
  This stores the lower 32 bits of the system time (in milliseconds).

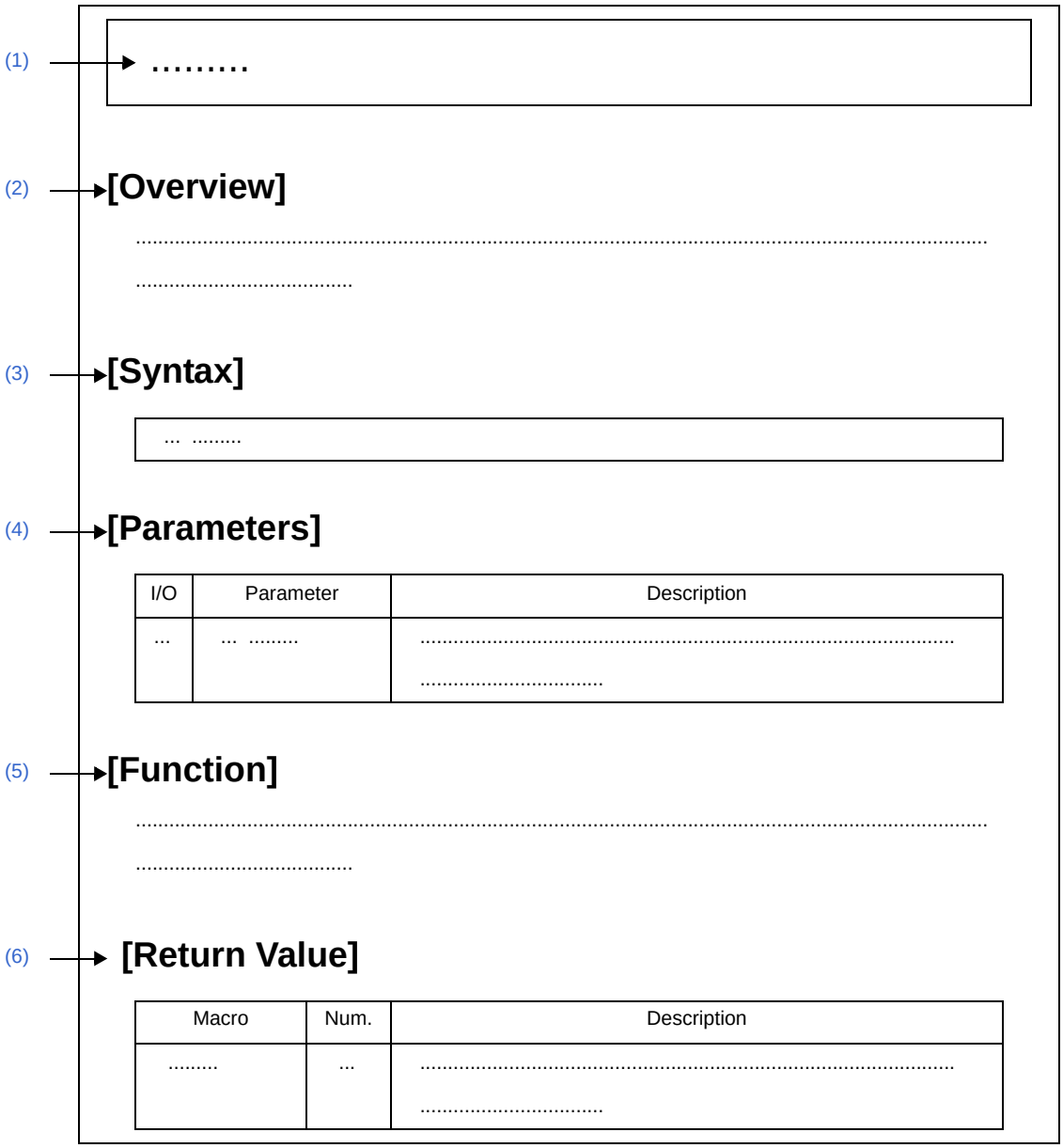
- utime
  This stores the upper 16 bits of the system time (in milliseconds).

Remark System time is relative time; it is the elapsed time since the system time was initialized in the Kernel Initialization Module, or it was modified via issuance of set_tim/iset_tim. When a timer interrupt specified by Exception cause code tintno occurs, it is updated by the value specified by Base clock cycle tbase.

13.4 Service Call Reference

This section describes the service calls provided by the RI850MP, using the following format.

Figure 13-1. Description Format of Service Calls



(1) Name

Indicates the name of the service call.

(2) [Overview]

Outlines the functions of the service call.

(3) [Syntax]

Indicates the format to be used when coding a service call to be issued in the C language.

(4) [Parameters]

Indicates the parameters of the service call, in the following format.

I/O	Parameter	Description
(a)	(b)	(c)

(a) I/O

Parameter type

I ... Input parameter

O ... Output parameter

(b) Parameter

Data type of parameter

(c) Description

Description of the parameter

(5) [Function]

Explains the function of a service call.

(6) [Return Value]

Indicates a service call's return value using the following format.

Macro	Num.	Description
(a)	(b)	(c)

(a) Macro

Macro return value

(b) Num.

Macro definition value

(c) Description

Description of return value

13.4.1 Task management functions

The following shows the service calls provided by the RI850MP as task management functions.

Table 13-11. Task Management Functions

Service Call	Function Overview
act_tsk/iact_tsk	Activate task
can_act/ican_act	Cancel activation request
ext_tsk	End this task
ter_tsk	Forcibly terminate task
chg_pri/ichg_pri	Change the current priority
get_pri/iget_pri	Get the current priority
ref_tsk/iref_tsk	Get task information

act_tsk

iact_tsk

[Overview]

Activate task.

[Syntax]

```
ER  act_tsk(ID tskid);
ER  iact_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task

[Function]

This issues an activation request to the task specified by *tskid*, and moves the task from DORMANT state to READY state.

If the target task is in a state other than the DORMANT state when this service call is issued, this service call does not change the task's state, and instead increments the activation request counter (by adding 1 to the activation request counter).

Remarks 1. Tasks moved to READY state are placed on the end of the ready queue corresponding to Initial priority itskpri.

2. If issuing this service call would cause the activation-request nesting count to exceed 127, then the activation request is not performed, and "E_QOVR (= -43)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status

Macro	Num.	Description
E_QOVR	-43	Queuing overflow - The activation request nesting count exceeds the maximum activation request nesting level of 127

can_act

ican_act

[Overview]

Cancel activation request.

[Syntax]

```
ER_UINT can_act(ID tskid);
ER_UINT ican_act(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value :I D of task

[Function]

This cancels the activation requests stored in the task specified by *tskid* (sets the activation request nest counter to 0) and returns the cancelled activation request nesting count (nesting count of act_tsk/iact_tsk) as the return value.

Remarks 1. If the target task is in DORMANT state, then a value of "0" is returned.

2. This service call does not manipulate the state of the task (e.g. moving it from READY to DORMANT state).

[Return Value]

Macro	Num.	Description
-	-	Normal termination - Number of activation request nesting levels cancelled
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status

ext_tsk

[Overview]

End this task.

[Syntax]

```
void  ext_tsk(void);
```

[Parameters]

None.

[Function]

Moves this task from RUNNING to DORMANT state.

If this task retains activation requests (its count value is other than 0) when this service call is issued, then after the task state is changed, processing equivalent to act_tsk/iact_tsk is performed.

Remarks 1. If this service call changes the task to DORMANT state, it also performs the following operations.
- Releases the acquired mutex
- Changes the current priority to the same value as Initial priority itskpri
- Sets the wake-up request nesting counter to 0
- Sets the forced wait request nesting counter to 0
- Changes the interrupt state to the same state as the initial interrupt state of Attribute tskatr

2. When the return instruction is called in a task, the RI850MP performs the same processing this service call.

3. If this service call is not defined for use in SCT Information, then subsequent behavior is not guaranteed.

4. If this service call is issued from a non-task, or from CPU locked state or dispatching disabled state, subsequent behavior is not guaranteed.

[Return Value]

None.

ter_tsk

[Overview]

Forcibly terminate task.

[Syntax]

```
ER  ter_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task

[Function]

This shifts the task specified in parameter *tskid* to DORMANT state.

If the target task retains activation requests (its count value is other than 0) when this service call is issued, then after the task state is changed, processing equivalent to act_tsk/iact_tsk is performed.

Remark If this service call changes the target task to DORMANT state, it also performs the following operations.

- Releases the acquired mutex
- Changes the current priority to the same value as Initial priority itskpri
- Sets the wake-up request nesting counter to 0
- Sets the forced wait request nesting counter to 0
- Changes the interrupt state to the same state as the initial interrupt state of Attribute tskatr

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - The target task belongs to a different PE - The invoking task was specified in *tskid*
E_OBJ	-41	Object status error - The target task is in DORMANT state

chg_pri

ichg_pri

[Overview]

Change the current priority

[Syntax]

```
ER  chg_pri(ID tskid, PRI tskpri);
ER  ichg_pri(ID tskid, PRI tskpri);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task
I	`PRI tskpri;`	Current priority after change TPRI_INI : Initial priority Numerical value : Current priority after change

[Function]

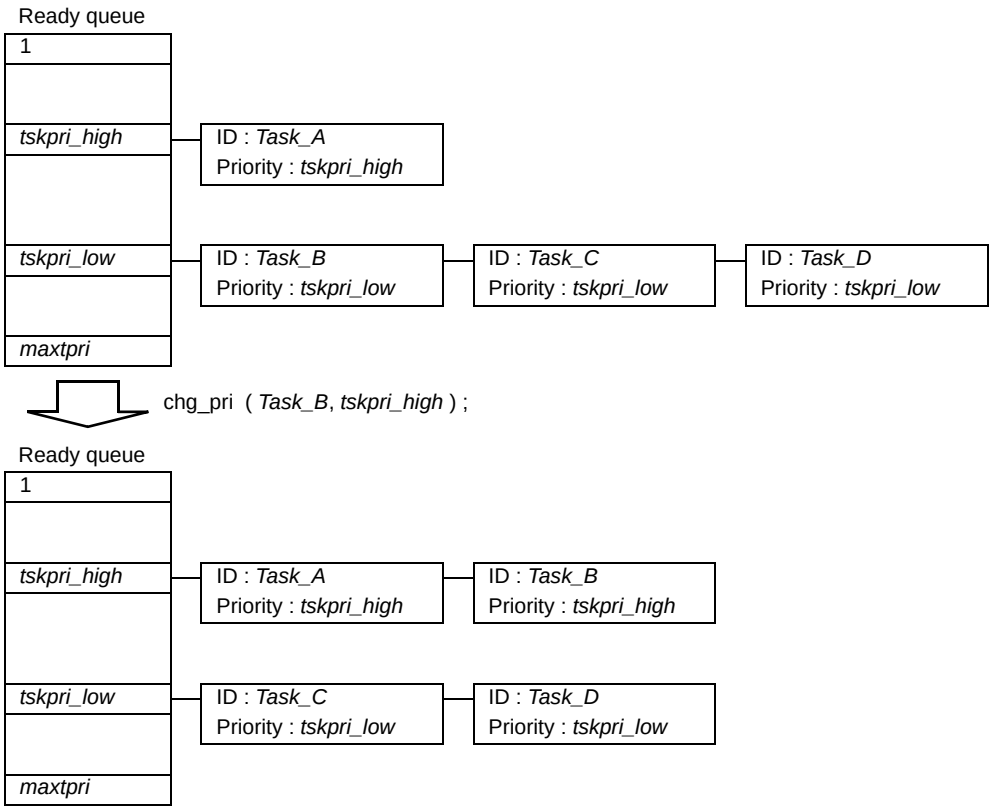
These service calls change the current priority of the task specified by *tskid* to the value specified by *tskpri*.

If the target task is in RUNNING or READY state when this service call is issued, then after the current priority is changed, the tasks will be relinked to the end of the ready queue corresponding to the priority specified by *tskpri*.

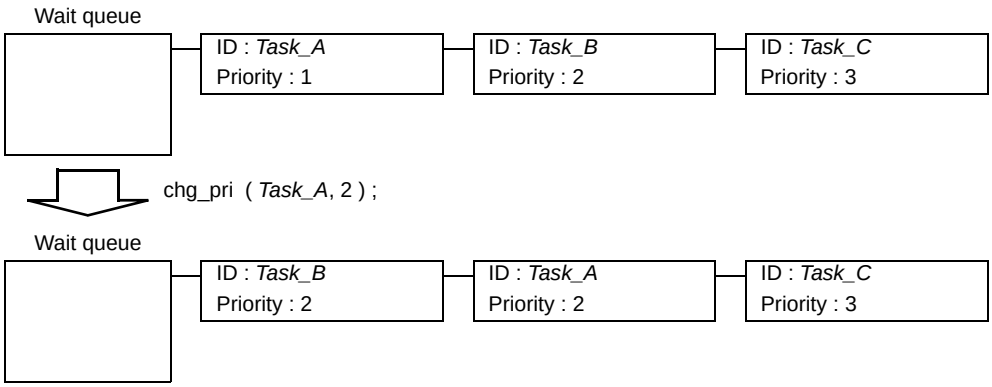
If the target task is not queued in a task-priority wait queue when this service call is issued, then after the priority is changed, the task will be relinked to an appropriate location on the wait queue.

Remarks 1. In the RI850MP, tasks with lower priority numbers have higher priority.

2. Below are described the changes that issuing this service call causes to the state of the ready queue.



3. Below are described the changes that issuing this service call causes to the state of the wait queue.



[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*tskpri* is invalid - *tskpri* < 0 - *tskpri* > Maximum priority maxtpri

Macro	Num.	Description
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - The target task belongs to a different PE
E_OBJ	-41	Object status error - The target task is in DORMANT state

get_pri

iget_pri

[Overview]

Get the current priority.

[Syntax]

```
ER  get_pri(ID tskid, PRI *p_tskpri);
ER  iget_pri(ID tskid, PRI *p_tskpri);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task
O	`PRI *p_tskpri;`	Pointer to area in which current priority is stored

[Function]

This acquires the current priority of the task specified by *tskid*, and stores it in the area specified by *p_tskpri*.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_tskpri* is invalid - *p_tskpri* = 0
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - The target task is in DORMANT state

ref_tsk

iref_tsk

[Overview]

Get task information.

[Syntax]

```
ER  ref_tsk(ID tskid, T_RTSK *pk_rtsk);
ER  iref_tsk(ID tskid, T_RTSK *pk_rtsk);
```

[Parameters]

I/O	Parameter	Description
I	ID *tskid*;	ID of task TSK_SELF : Current task Numerical value :I D of task
O	T_RTSK *pk_rtsk;	Pointer to the area storing the task information

[Task Information T_RTSK]

```
typedef struct  t_rtsk {
        STAT    tskstat;    /* Current state */
        PRI     tskpri;     /* Current priority */
        PRI     tskbpri;    /* Reserved for future use */
        STAT    tskwait;    /* Wait cause */
        ID      wobjid;     /* Object ID */
        TMO     lefttmo;    /* Remaining time until timeout */
        UINT    actcnt;     /* Activation request nesting count */
        UINT    wupcnt;     /* Wake-up request nesting count */
        UINT    sus_cnt;    /* Forced wait request nesting count */
        ATR     tskatr;     /* Attribute */
        PRI     itskpri;    /* Initial priority */
        PE_ID   peid;       /* PE number */
} T_RTSK;
```

[Function]

Stores task information (current state, etc.) of the task specified by *tskid* in the area specified by *pk_rtsk*.

Remark See "13.3.1 Task information T_RTSK" for details about task information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rtsk* is invalid - *pk_rtsk* = 0
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status

13.4.2 Task dependent synchronization functions

The following shows the service calls provided by the RI850MP as task dependent synchronization functions.

Table 13-12. Task Dependent Synchronization Functions

Service Call	Function Overview
slp_tsk	Put task in Sleeping state.
tslp_tsk	Put task in Sleeping state (with timeout).
wup_tsk/iwup_tsk	Resume task after sleep.
can_wup/ican_wup	Cancel wake-up request.
rel_wai/irel_wai	Forced cancellation of WAITING state.
sus_tsk/isus_tsk	Put task in SUSPENDED state.
rsm_tsk/irsm_tsk	Resume task from SUSPENDED state.
frsm_tsk/ifrsm_tsk	Forcibly resume task from SUSPENDED state.
dly_tsk	Put task in Delayed state.

slp_tsk

[Overview]

Put task in Sleeping state.

[Syntax]

```
ER  slp_tsk(void);
```

[Parameters]

None.

[Function]

Moves this task from RUNNING to Sleeping state.
In the following cases, the Sleeping state is cancelled, and then the task is moved to the READY state.

Cancellation of Sleeping State	Return Value
A wakeup request was issued as a result of issuing wup_tsk/iwup_tsk.	E_OK
Forced release from Sleeping state as a result of issuing rel_wai/irel_wai.	E_RLWAI

Remark If the task holds a wake-up request (counter value other than 0) when this service call is issued, then no changes are made to the task's state, and 1 is subtracted from the wake-up request nesting counter.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Sleeping state - Issuance of rel_wai/irel_wai

tslp_tsk

[Overview]

Put task in Sleeping state (with timeout).

[Syntax]

```
ER  tslp_tsk(TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`TMO tmout;`	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait tim

[Function]

Moves this task from RUNNING to Sleeping state.
In the following cases, the Sleeping state is cancelled, and then the task is moved to the READY state.

Cancellation of Sleeping State	Return Value
A wakeup request was issued as a result of issuing wup_tsk/iwup_tsk.	E_OK
Forced release from Sleeping state as a result of issuing rel_wai/irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout).	E_TMOUT

Remarks 1. If the task holds a wake-up request (counter value other than 0) when this service call is issued, then no changes are made to the task's state, and 1 is subtracted from the wake-up request nesting counter.

2. If TMO_FEVR is specified in *tmout*, then processing equivalent to slp_tsk will be performed.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*tmout* is invalid - *tmout* < TMO_FEVR
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status

Macro	Num.	Description
E_RLWAI	-49	Forced cancellation of Sleeping state - Issuance of rel_wai/irel_waii
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

wup_tsk

iwup_tsk

[Overview]

Resume task after sleep.

[Syntax]

```
ER  wup_tsk(ID tskid);
ER  iwup_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task

[Function]

This issues a wake-up request to the task specified by *tskid*, and moves the task from Sleeping state to READY state.

If the target task is in a state other than the Sleeping state when this service call is issued, this service call does not change the task's state, and instead increments the activation request counter (by adding 1 to it).

Remark If issuing this service call would cause the wake-up request nesting count to exceed 127, then the wake-up request is not performed, and "E_QOVR (= -43)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - The target task is in DORMANT state
E_QOVR	-43	Queuing overflow - The wake-up request nesting count exceeds the maximum wake-up request nesting level of 127

can_wup

ican_wup

[Overview]

Cancel wake-up request.

[Syntax]

```
ER_UINT can_wup(ID tskid);
ER_UINT ican_wup(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task

[Function]

This cancels the wake-up requests stored in the task specified by *tskid* (sets the wake-up request nest counter to 0) and returns the cancelled wake-up request nesting count (nesting count of wup_tsk/iwup_tsk) as the return value.

Remark This service call does not manipulate the state of the task (e.g. moving it from WAITING state to READY state).

[Return Value]

Macro	Num.	Description
-	-	Normal termination - Number of wake-up request nesting levels cancelled
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - The target task is in DORMANT state

rel_wai

irel_wai

[Overview]

Forced cancellation of WAITING state.

[Syntax]

```
ER  rel_wai(ID tskid);
ER  irel_wai(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task

[Function]

This forcibly cancels the WAITING state of the task specified by *tskid*. This removes the target task from the wait queue, and changes its status from WAITING state to READY, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. This service call does not cancel SUSPENDED state.

2. "E_RLWAI (= -49)" is returned from the service call that triggered the move to the WAITING state (slp_tsk, dly_tsk, wai_sem, or the like) to the task whose WAITING state is forcibly cancelled by this service call.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - Target task state is other than WAITING state or WAITING-SUSPENDED state

sus_tsk

isus_tsk

[Overview]

Put task in SUSPENDED state

[Syntax]

```
ER  sus_tsk(ID tskid);
ER  isus_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task TSK_SELF : Current task Numerical value : ID of task

[Function]

This issues a forced wait request to the task specified by *tskid*, and moves the task to SUSPENDED state or WAITING-SUSPENDED state.

If the target task is SUSPENDED state or WAITING-SUSPENDED state when this service call is issued, this service call does not change the task's state, and instead increments the forced wait request counter (by adding 1 to it).

Remark If issuing this service call would cause the FORCED WAIT request nesting count to exceed 127, then the forced wait request is not performed, and "E_QOVR (= -43)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status - Invoking task specified when issued from dispatching disabled status
E_OBJ	-41	Object status error - The target task is in DORMANT state

Macro	Num.	Description
E_QOVR	-43	Queuing overflow - The forced wait request nesting count exceeds the maximum forced wait request nesting level of 127

rsm_tsk

irsm_tsk

[Overview]

Resume task from SUSPENDED state.

[Syntax]

```
ER  rsm_tsk(ID tskid);
ER  irsm_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task

[Function]

This issues a cancel forced wait request to the task specified by *tskid*, and moves the task to READY or WAITING state.

If the task holds a forced wait request (counter value other than 0) when this service call is issued, then no changes are made to the task's state, and 1 is subtracted from the forced wait request nesting counter.

Remark This service call does not perform queuing of forced wait cancellation requests. For this reason, if the target task is in other than SUSPENDED state or WAITING-SUSPENDED state, the value "E_OBJ (= -41)" will be returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid* is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - Target task state is other than SUSPENDED state or WAITING-SUSPENDED state

frsm_tsk

ifrsm_tsk

[Overview]

Forcibly resume task from SUSPENDED state.

[Syntax]

```
ER  frsm_tsk(ID tskid);
ER  ifrsm_tsk(ID tskid);
```

[Parameters]

I/O	Parameter	Description
I	`ID tskid;`	ID of task

[Function]

This forcibly cancels the SUSPENDED state of the task specified by *tskid* (sets forced wait request nesting counter to 0). This changes the state of the target task from SUSPENDED state to READY state, or from WAITING-SUSPENDED state to WAITING state.

Remarks 1. This service call does not cancel WAITING state.

2. This service call does not perform queuing of forced wait cancellation requests. For this reason, if the target task is in other than SUSPENDED state or WAITING-SUSPENDED state, the value "E_OBJ (= -41)" will be returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*tskid*is invalid - ID is not defined in Task information - TSK_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status
E_OBJ	-41	Object status error - Target task state is other than SUSPENDED state or WAITING-SUSPENDED state

dly_tsk

[Overview]

Put task in Delayed state.

[Syntax]

```
ER  dly_tsk(RELTIM dlytim);
```

[Parameters]

I/O	Parameter	Description
I	`RELTIM dlytim;`	Specified timeout (in millisecond)

[Function]

This moves the task from RUNNING to Delayed state.
In the following cases, the Delayed state is cancelled and the task is moved to the READY state.

Cancellation of Delayed State	Return Value
Wait time specified by *dlytim* has elapsed.	E_OK
Forced release from Delayed state as a result of issuing rel_wai/irel_wai.	E_RLWAI

Remark Even if *dlytim* is specified as 0, the task state will be changed.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination - Wait time specified by *dlytim* has elapsed
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Delayed state - Issuance of rel_wai/irel_wai

13.4.3 Synchronization and communication functions (semaphores)

Below is a list of the service calls provided by the RI850MP as synchronization and communication functions (semaphores).

Table 13-13. Synchronization and Communication Functions (Semaphores)

Service Call	Function Overview
sig_sem/isig_sem	Release semaphore resource.
wai_sem	Acquire semaphore resource.
pol_sem/ipol_sem	Acquire semaphore resource (polling).
twai_sem	Acquire semaphore resource (with timeout).
ref_sem/iref_sem	Reference semaphore information.

sig_sem

isig_sem

[Overview]

Release semaphore resource.

[Syntax]

```
ER  sig_sem(ID semid);
ER  isig_sem(ID semid);
```

[Parameters]

I/O	Parameter	Description
I	`ID semid;`	ID of semaphore

[Function]

These service calls return the resource to the semaphore specified by *semid* (adds 1 to the semaphore counter).

If a task is queued in the wait queue of the target semaphore when this service call is issued, resource is not released, and instead it is passed to the relevant task (first task of wait queue). This removes the target task from the wait queue, and changes its status from WAITING state for a semaphore resource to READY, or from WAITING-SUSPENDED state to SUSPENDED state.

Remark If the number of resources issued by this service call exceeds Maximum resource count maxsem, then the resource will not be released, and "E_QOVR (= -43)" will be returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*semid* is invalid - ID is not identified in Semaphore information
E_CTX	-25	Context error - Issued from CPU lock status
E_QOVR	-43	Queuing overflow - Number of resources exceeds Maximum resource count maxsem

wai_sem

[Overview]

Acquire semaphore resource.

[Syntax]

```
ER  wai_sem(ID semid);
```

[Parameters]

I/O	Parameter	Description
I	`ID semid;`	ID of semaphore

[Function]

This service call acquires a resource from the semaphore specified by *semid* (subtracts 1 from the semaphore counter).

If the target semaphore resource cannot be acquired (the count is already 0) when this service call is issued, this service call queues the invoking task in the semaphore wait queue and moves it from the RUNNING state to WAITING state for a semaphore resource.

The WAITING state for a semaphore resource is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Semaphore Resource	Return Value
The resource was returned to the target semaphore as a result of issuing sig_sem/isig_sem.	E_OK
Forced release from WAITING state for a semaphore resource as a result of issuing rel_wai/ irel_wai.	E_RLWAI

Remark The queuing order of tasks in the semaphore wait queue is determined by the queuing method specified by Attribute sematr (order in which resource requests were made, or order of task priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*semid* is invalid - ID is not identified in Semaphore information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status

Macro	Num.	Description
E_RLWAI	-49	Forced cancellation of WAITING state for a semaphore resource - Issuance of rel_wai/irel_wai

pol_sem

ipol_sem

[Overview]

Acquire semaphore resource (polling).

[Syntax]

```
ER  pol_sem(ID semid);
ER  ipol_sem(ID semid);
```

[Parameters]

I/O	Parameter	Description
I	`ID semid;`	ID of semaphore

[Function]

This service call acquires a resource from the semaphore specified by *semid* (subtracts 1 from the semaphore counter).

If the target semaphore resource cannot be acquired (the count is already 0) when this service call is issued, "E_TMOUT (= -50)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*semid* is invalid - ID is not identified in Semaphore information
E_CTX	-25	Context error - Issued from CPU lock status
E_TMOUT	-50	Polling failure - Number of target semaphore resources is 0

twai_sem

[Overview]

Acquire semaphore resource (with timeout).

[Syntax]

```
ER  twai_sem(ID semid, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	ID *semid*;	ID of semaphore
I	TMO *tmout*;	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait tim

[Function]

This service call acquires a resource from the semaphore specified by *semid* (subtracts 1 from the semaphore counter).

If the target semaphore resource cannot be acquired (the count is already 0) when this service call is issued, this service call queues the invoking task in the semaphore wait queue and moves it from the RUNNING state to WAITING state for a semaphore resource.

The WAITING state for a semaphore resource is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Semaphore Resource	Return Value
The resource was returned to the target semaphore as a result of issuing sig_sem/isig_sem.	E_OK
Forced release from WAITING state for a semaphore resource as a result of issuing rel_wai/ irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout).	E_TMOUT

Remarks 1. The queuing order of tasks in the semaphore wait queue is determined by the queuing method specified by Attribute sematr (order in which resource requests were made, or order of task priority).

2. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to wai_sem. If it is specified as TMO_POL, then the processing is equivalent to pol_sem /ipol_sem.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

Macro	Num.	Description
E_PAR	-17	*tmout* is invalid - *tmout* < TMO_FEVR
E_ID	-18	*semid* is invalid - ID is not identified in Semaphore information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of WAITING state for a semaphore resource - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

ref_sem

iref_sem

[Overview]

Reference semaphore information.

[Syntax]

```
ER  ref_sem(ID semid, T_RSEM *pk_rsem);
ER  iref_sem(ID semid, T_RSEM *pk_rsem);
```

[Parameters]

I/O	Parameter	Description
I	`ID semid;`	ID of semaphore
O	`T_RSEM *pk_rsem;`	Pointer to area storing semaphore information

[Semaphore Information T_RSEM]

```
typedef struct  t_rsem {
        ID      wtskid;   /* Existence of waiting task */
        UINT    semcnt;   /* Current resource count */
        ATR     sematr;   /* Attribute */
        UINT    maxsem;   /* Maximum resource count */
} T_RSEM;
```

[Function]

This stores the information (e.g. existence of waiting tasks) for the semaphore specified by *semid* in the area specified by *pk_rsem*.

Remark See "13.3.2 Semaphore information T_RSEM" for details about semaphore information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rsem* is invalid - *pk_rsem* = 0
E_ID	-18	*semid* is invalid - ID is not identified in Semaphore information

Macro	Num.	Description
E_CTX	-25	Context error - Issued from CPU lock status

13.4.4 Synchronization and communication functions (eventflags)

Below is a list of the service calls provided by the RI850MP as synchronization and communication functions (eventflags).

Table 13-14. Synchronization and Communication Functions (Eventflags)

Service Call	Function Overview
set_flg/iset_flg	Bit pattern changed.
clr_flg/iclr_flg	Bit pattern cleared.
wai_flg	Bit pattern determined.
pol_flg/ipol_flg	Bit pattern determined (polling).
twai_flg	Bit pattern determined (with timeout).
ref_flg/iref_flg	Eventflag information referenced.

set_flg

iset_flg

[Overview]

Bit pattern changed.

[Syntax]

```
ER  set_flg(ID flgid, FLGPTN setptn);
ER  iset_flg(ID flgid, FLGPTN setptn);
```

[Parameters]

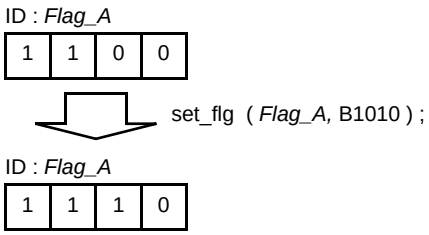
I/O	Parameter	Description
I	`ID` *`flgid`*;	ID of eventflag
I	`FLGPTN` *`setptn`*;	Bit pattern to set

[Function]

This performs a logical OR on the current bit pattern specified by *flgid*, and the bit pattern specified by *setptn*, and sets the results as the current bit pattern of the target eventflag.

If the required condition of the task queued to the target eventflag wait queue is satisfied when this service call is issued, then after the bit pattern is set, the target task will be removed from the wait queue. This changes the state of the target task from WAITING state for an eventflag to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. If the current bit pattern of the target eventflag is B1100, and the bit pattern specified by *setptn* is B1010, then B1110 will be set as the current bit pattern.



2. If the TA_WMUL attribute is set in the target eventflag (maximum number of tasks that can be queued: multiple tasks), then whether the tasks is subject to the determination of whether the issuance of this service call met the request conditions depends on whether the "2. TA_CLR attribute" (clear method: when request conditions are met, clear bit pattern).

- If the TA_CLR attribute has been assigned
  All tasks queued in the wait queue from the first task until the task satisfying the request conditions are subject to determination.

- If the TA_CLR attribute has not been assigned
  All tasks queued in the wait queue are subject to determination.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*flgid* is invalid - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from CPU lock status

clr_flg

iclr_flg

[Overview]

Bit pattern cleared.

[Syntax]

```
ER  clr_flg(ID flgid, FLGPTN clrptn);
ER  iclr_flg(ID flgid, FLGPTN clrptn);
```

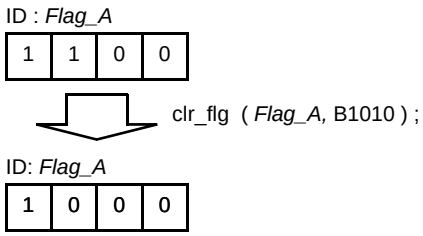
[Parameters]

I/O	Parameter	Description
I	`ID flgid;`	ID of eventflag
I	`FLGPTN clrptn;`	Bit pattern to clear

[Function]

This performs a logical AND on the current bit pattern specified by *flgid*, and the clear pattern specified by *clrptn*, and sets the results as the current bit pattern of the target eventflag.

Remark If the current bit pattern of the target eventflag is B1100, and the bit pattern specified by *clrptn* is B1010, then B1000 will be set as the current bit pattern.



[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*flgid* is invalid - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from CPU lock status

wai_flg

[Overview]

Bit pattern determined.

[Syntax]

```
ER  wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
```

[Parameters]

I/O	Parameter	Description
I	`ID flgid;`	ID of eventflag
I	`FLGPTN waiptn;`	Request bit pattern
I	`MODE wfmode;`	Request conditions TWF_ANDW : AND wait TWF_ORW : OR wait
O	`FLGPTN *p_flgptn;`	Pointer to the area in which bit pattern is stored

[Function]

This determines whether the current bit pattern of the eventflag specified by *flgid* meets the conditions of the request bit pattern specified by *waiptn* and the request conditions specified by *wfmode*. If the bit pattern satisfied the requirements, the current bit pattern in question is stored in the area specified by *p_flgptn*.

If the current bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue, after which the task transitions from the RUNNING state to the WAITING state (WAITING state for an eventflag).

The WAITING state for an eventflag is cancelled in the following cases, and then moved to the READY state.

Cancelation of WAITING State for an Eventflag	Return Value
A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing set_flg/iset_flg.	E_OK
Forced release from WAITING state for an eventflag as a result of issuing rel_wai/irel_wai.	E_RLWAI

Below are shown the *wfmode* request conditions, which are the criteria for judging bit patterns.

- TWF_ANDW
  Determine if all bits set to "1" in *waiptn* are set in the target eventflag.

- TWF_ORW
  Determine if any bits set to "1" in *waiptn* are set in the target eventflag.

Remarks 1. On the RI850MP, if this service call is issued for the eventflag of the TA_WSGL attribute (maximum number of tasks that can be queued: 1) for tasks already queued in the wait queue, a return value of "E_ILUSE (= -28) will be returned, regardless of whether the condition can be immediately met or not.

2. The wait queue for the eventflag for tasks with the TA_WMUL attribute (maximum number of tasks that can be queued: multiple) is ordered using the queuing method specified by the Attribute flgatr (order of bit-pattern determination requests or order of task priority).
3. If the WAITING state for an eventflag is forcibly released by issuing rel_wai/irel_wai, the contents of the area specified by *p_flgptn* will be undefined.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*waiptn*, *wfmode*, or *p_flgptn* is invalid - *waiptn* = 0 - *wfmode* is invalid - *p_flgptn* = 0
E_ID	-18	*flgid* is invalid - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_ILUSE	-28	Invalid service call use - Issued for an eventflag with a TA_WSGL attribute, and a task is already queued
E_RLWAI	-49	Forced cancelation of WAITING state for an eventflag - Issuance of rel_wai/irel_wai

pol_flg

ipol_flg

[Overview]

Bit pattern determined (polling).

[Syntax]

```
ER  pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER  ipol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
```

[Parameters]

I/O	Parameter	Description
I	`ID flgid;`	ID of eventflag
I	`FLGPTN waiptn;`	Request bit pattern
I	`MODE wfmode;`	Request conditions TWF_ANDW : AND wait TWF_ORW : OR wait
O	`FLGPTN *p_flgptn;`	Pointer to the area in which bit pattern is stored

[Function]

This determines whether the current bit pattern of the eventflag specified by *flgid* meets the conditions of the request bit pattern specified by *waiptn* and the request conditions specified by *wfmode*. If the bit pattern satisfied the requirements, the current bit pattern in question is stored in the area specified by *p_flgptn*.

If the current bit pattern of the target eventflag does not satisfy the condition when this service call is issued, "E_TMOUT (= -50)" is returned.

Below are shown the wfmode request conditions, which are the criteria for judging bit patterns.

- TWF_ANDW
  Determine if all bits set to "1" in *waiptn* are set in the target eventflag.

- TWF_ORW
  Determine if any bits set to "1" in *waiptn* are set in the target eventflag.

Remark On the RI850MP, if this service call is issued for the eventflag of the TA_WSGL attribute (maximum number of tasks that can be queued: 1) for tasks already queued in the wait queue, a return value of "E_ILUSE (= -28) will be returned, regardless of whether the condition can be immediately met or not.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination

Macro	Num.	Description
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*waiptn*, *wfmode*, or *p_flgptn* is invalid - *waiptn* = 0 - *wfmode* is invalid - *p_flgptn* = 0
E_ID	-18	*flgid* is invalid - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - Issued for an eventflag with a TA_WSGL attribute, and a task is already queued
E_TMOUT	-50	Polling failure - The bit pattern of the target eventflag does not satisfy the wait condition

twai_flg

[Overview]

Bit pattern determined (with timeout).

[Syntax]

```
ER  twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`ID flgid;`	ID of eventflag
I	`FLGPTN waiptn;`	Request bit pattern
I	`MODE wfmode;`	Request conditions TWF_ANDW : AND wait TWF_ORW : OR wait
O	`FLGPTN *p_flgptn;`	Pointer to the area in which bit pattern is stored
I	`TMO tmout;`	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Function]

This determines whether the current bit pattern of the eventflag specified by *flgid* meets the conditions of the request bit pattern specified by *waiptn* and the request conditions specified by *wfmode*. If the bit pattern satisfied the requirements, the current bit pattern in question is stored in the area specified by *p_flgptn*.

If the current bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue, after which the task transitions from the RUNNING state to the WAITING state (WAITING state for an eventflag).

The WAITING state for an eventflag is cancelled in the following cases, and then moved to the READY state.

Cancelation of WAITING State for an Eventflag	Return Value
A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing set_flg/iset_flg.	E_OK
Forced release from WAITING state for an eventflag as a result of issuing rel_wai/irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout).	E_TMOUT

Below are shown the *wfmode* request conditions, which are the criteria for judging bit patterns.

- TWF_ANDW

  Determine if all bits set to "1" in *waiptn* are set in the target eventflag.

- TWF_ORW
  Determine if any bits set to "1" in *waiptn* are set in the target eventflag.

Remarks 1. On the RI850MP, if this service call is issued for the eventflag of the TA_WSGL attribute (maximum number of tasks that can be queued: 1) for tasks already queued in the wait queue, a return value of "E_ILUSE (= -28) will be returned, regardless of whether the condition can be immediately met or not.

2. The wait queue for the eventflag for tasks with the TA_WMUL attribute (maximum number of tasks that can be queued: multiple) is ordered using the queuing method specified by the Attribute flgatr (order of bit-pattern determination requests or order of task priority).

3. If the WAITING state for an eventflag is forcibly released by issuing rel_wai/irel_wai, the contents of the area specified by *p_flgptn* will be undefined.

4. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to wai_flg. If it is specified as TMO_POL, then the processing is equivalent to pol_flg/ipol_flg.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*waiptn*, *wfmode*, *p_flgptn*, or *tmout* is invalid - *waiptn* = 0 - *wfmode* is invalid - *p_flgptn* = 0 - *tmout* < TMO_FEVR
E_ID	-18	*flgid* is invalid - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_ILUSE	-28	Invalid service call use - Issued for an eventflag with a TA_WSGL attribute, and a task is already
E_RLWAI	-49	Forced cancelation of WAITING state for an eventflag - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

ref_flg

iref_flg

[Overview]

Eventflag information referenced.

[Syntax]

```
ER  ref_flg(ID flgid, T_RFLG *pk_rflg);
ER  iref_flg(ID flgid, T_RFLG *pk_rflg);
```

[Parameters]

I/O	Parameter	Description
I	`ID flgid;`	ID of eventflag
O	`T_RFLG *pk_rflg;`	Pointer to the area storing the eventflag information

[Eventflag Information T_RFLG]

```
typedef struct  t_rflg {
        ID      wtskid;   /* Existence of waiting task */
        FLGPTN  flgptn;   /* Current bit pattern */
        ATR     flgatr;   /* Attribute */
} T_RFLG;
```

[Function]

Stores eventflag information (e.g. waiting task flag) of the eventflag specified by *flgid* in the area specified by *pk_rtsk*.

Remark See "13.3.3 Eventflag information T_RFLG" for details about eventflag information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rflg is invalid* - *pk_rflg* = 0
E_ID	-18	*flgid is invalid* - ID is not defined in Eventflag information
E_CTX	-25	Context error - Issued from CPU lock status

13.4.5 Synchronization and communication functions (data queues)

Below is a list of the service calls provided by the RI850MP as synchronization and communication functions (data queues).

Table 13-15. Synchronization and Communication Functions (Data Queues)

Service Call	Function Overview
snd_dtq	Send to data queue.
psnd_dtq/ipsnd_dtq	Send to data queue (polling).
tsnd_dtq	Send to data queue (with timeout).
fsnd_dtq/ifsnd_dtq	Force send to data queue.
rcv_dtq	Receive from data queue.
prcv_dtq/iprcv_dtq	Receive from data queue (polling).
trcv_dtq	Receive from data queue (with timeout).
ref_dtq/iref_dtq	Reference data queue information.

snd_dtq

[Overview]

Send to data queue.

[Syntax]

```
ER  snd_dtq(ID dtqid, VP_INT data);
```

[Parameters]

I/O	Parameter	Description
I	ID *dtqid*;	ID of data queue
I	VP_INT *data*;	Data element to be sent

[Function]

This sends the data specified by data to the data queue specified by *dtqid*.

Note that when this service call is issued, if the amount of data written to the target data queue's buffer area is the same as the value specified by Maximum data count dtqcnt, then the data-send operation is not performed.

Instead, the data is queued to the data wait queue for the current task, and the state changes from RUNNING state to Sending WAITING state for a data queue.

The Sending WAITING state for a queue is cancelled in the following cases, and then the state transitions from the Sending WAITING state for a data queue to the READY state.

Cancellation of Sending WAITING State for a data queue	Return Value
Available space was secured in the buffer area of the target data queue as a result of issuing rcv_dtq.	E_OK
Available space was secured in the buffer area of the target data queue as a result of issuing prcv_dtq/iprcv_dtq.	E_OK
Available space was secured in the buffer area of the target data queue as a result of issuing trcv_dtq.	E_OK
Forced release from Sending WAITING state for a data queue as a result of issuing rel_wai/ irel_wai.	E_RLWAI

If a task has been queued to the reception wait queue of the target data queue when this service call is issued, this service call does not send data but transfers the data to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from Receiving WAITING state for a data queue to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. On the RI850MP, the synchronization and communication functions (data queues) write data to the buffer area of the data queue as data send processes, and read data from these areas as data reception processes.

2. The queuing order of tasks in the data waiting queue is determined by the queuing method specified by Attribute dtqatr (order in which data send requests were made, or order of task priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*dtqid*is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Sending WAITING state for a data queue - Issuance of rel_wai/irel_wai

psnd_dtq

ipsnd_dtq

[Overview]

Send to data queue (polling).

[Syntax]

```
ER  psnd_dtq(ID dtqid, VP_INT data);
ER  ipsnd_dtq(ID dtqid, VP_INT data);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
I	`VP_INT data;`	Data element to be sent

[Function]

This sends the data specified by data to the data queue specified by *dtqid*.

Note that when this service call is issued, if the amount of data written to the target data queue's buffer area is the same as the value specified by Maximum data count dtqcnt, a return value of "E_TMOUT (= -50)" is returned.

If a task has been queued to the reception wait queue of the target data queue when this service call is issued, this service call does not send data but transfers the data to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from Receiving WAITING state for a data queue to READY, or from WAITING-SUSPENDED state to SUSPENDED state.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from CPU lock status
E_TMOUT	-50	Polling failure - Number of data elements exceeds Maximum data count dtqcnt

tsnd_dtq

[Overview]

Send to data queue (with timeout).

[Syntax]

```
ER  tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
I	`VP_INT data;`	Data element to be sent
I	`TMO tmout;`	Specified timeout (in milliseconds) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Function]

This sends the data specified by data to the data queue specified by *dtqid*.

Note that when this service call is issued, if the amount of data written to the target data queue's buffer area is the same as the value specified by Maximum data count dtqcnt, then the data-send operation is not performed.

Instead, the data is queued to the data wait queue for the current task, and the state changes from execution state to wait to sending WAITING state.

The Sending WAITING state for a data queue is cancelled in the following cases, and then the state transitions to the READY state.

Cancellation of Sending WAITING State for a data queue	Return Value
Available space was secured in the buffer area of the target data queue as a result of issuing rcv_dtq.	E_OK
Available space was secured in the buffer area of the target data queue as a result of issuing prcv_dtq/iprcv_dtq.	E_OK
Available space was secured in the buffer area of the target data queue as a result of issuing trcv_dtq.	E_OK
Forced release from Sending WAITING state for a data queue as a result of issuing rel_wai/ irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout)	E_TMOUT

If a task has been queued to the reception wait queue of the target data queue when this service call is issued, this service call does not send data but transfers the data to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from Receiving WAITING state for a data queue to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. The queuing order of tasks in the data waiting queue is determined by the queuing method specified by Attribute dtqatr (order in which data send requests were made, or order of task priority).

2. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to snd_dtq. If it is specified as TMO_POL, then the processing is equivalent to psnd_dtq/ipsnd_dtq.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*tmout* is invalid - *tmout* < TMO_FEVR
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Sending WAITING state for a data queue - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

fsnd_dtq

ifsnd_dtq

[Overview]

Force send to data queue.

[Syntax]

```
ER  fsnd_dtq(ID dtqid, VP_INT data);
ER  ifsnd_dtq(ID dtqid, VP_INT data);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
I	`VP_INT data;`	Data element to be sent

[Function]

This sends the data specified by *data* to the data queue specified by *dtqid*.

Note that when this service call is issued, if the amount of data written to the target data queue's buffer area is the same as the value specified by Maximum data count dtqcnt, then after the data is written to the buffer area, the oldest data is deleted, and then the data specified by *data* is sent.

If a task has been queued to the reception wait queue of the target data queue when this service call is issued, this service call does not send data but transfers the data to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from Receiving WAITING state for a data queue to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - Maximum data count dtqcnt of target data queue is 0

rcv_dtq

[Overview]

Receive from data queue.

[Syntax]

```
ER  rcv_dtq(ID dtqid, VP_INT *p_data);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
O	`VP_INT *p_data;`	Pointer to area in which data is stored

[Function]

This stores the data received from the data queue specified by *dtqid* into the area specified by *p_data*.

If no data could be received from the target data queue (no data has been written to the buffer area) when this service call is issued, the task is queued onto the data wait queue, and then transitions from RUNNING state to Receiving WAITING state for a data queue.

The Receiving WAITING state for a data queue is cancelled in the following cases, and then the state transitions from Receiving WAITING state for a data queue to the READY state.

Cancellation of Receiving WAITING State for a data queue	Return Value
Data was sent to the target data queue as a result of issuing snd_dtq.	E_OK
Data was sent to the target data queue as a result of issuing prcv_dtq/iprcv_dtq.	E_OK
Data was sent to the target data queue as a result of issuing tsnd_dtq.	E_OK
Data was sent to the target data queue as a result of issuing fsnd_dtq/ifsnd_dtq.	E_OK
Forced release from Receiving WAITING state for a data queue as a result of issuing rel_wai/ irel_wai.	E_RLWAI

Remarks 1. Tasks are queued in the data wait queue in the order in which data reception requests were made.

2. If the receiving WAITING state for a data queue is forcibly released by issuing rel_wai/irel_wai, the contents of the area specified by *p_data* will be undefined.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_data*is invalid - *p_data* = 0

Macro	Num.	Description
E_ID	-18	*dtqid*is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Receiving WAITING state for a data queue - Issuance of rel_wai/irel_wai

prcv_dtq

iprcv_dtq

[Overview]

Receive from data queue (polling)

[Syntax]

```
ER  prcv_dtq(ID dtqid, VP_INT *p_data);
ER  iprcv_dtq(ID dtqid, VP_INT *p_data);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
O	`VP_INT *p_data;`	Pointer to area in which data is stored

[Function]

This stores the data received from the data queue specified by *dtqid* into the area specified by *p_data*.

Note that when this service call is issued, if data could not be received from the target data queue (no data has been written to the buffer area), a return value of "E_TMOUT (= -50)" will be returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_data* is invalid - *p_data* = 0
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from CPU lock status
E_TMOUT	-50	Polling failure - No data has been written to the buffer area of the target data queue

trcv_dtq

[Overview]

Receive from data queue (with timeout).

[Syntax]

```
ER  trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
O	`VP_INT *p_data;`	Pointer to area in which data is stored
I	`TMO tmout;`	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Function]

This stores the data received from the data queue specified by *dtqid* into the area specified by *p_data*.

If no data could be received from the target data queue (no data has been written to the buffer area) when this service call is issued, the task is queued onto the data wait queue, and then transitions from execution state to "waiting to receive data" state.

The Receiving WAITING state for a data queue is cancelled in the following cases, and then the state transitions from Receiving WAITING state for a data queue to the READY state.

Cancellation of Receiving WAITING State for a data queue	Return Value
Data was sent to the target data queue as a result of issuing snd_dtq.	E_OK
Data was sent to the target data queue as a result of issuing prcv_dtq/iprcv_dtq.	E_OK
Data was sent to the target data queue as a result of issuing tsnd_dtq.	E_OK
Data was sent to the target data queue as a result of issuing fsnd_dtq/ifsnd_dtq.	E_OK
Forced release from Receiving WAITING state for a data queue as a result of issuing rel_wai/ irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout)	E_TMOUT

Remarks 1. Tasks are queued in the data wait queue in the order in which data reception requests were made.

2. If the receiving WAITING state for a data queue is forcibly released by issuing rel_wai/irel_wai, or the time specified by *tmout* has elapsed (timeout), the contents of the area specified by *p_data* will be undefined.

3. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to rcv_dtq. If it is specified as TMO_POL, then the processing is equivalent to prcv_dtq/iprcv_dtq.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_data* or *tmout* is invalid - *p_data* = 0 - *tmout* < TMO_FEVR
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of Receiving WAITING state for a data queue - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

ref_dtq

iref_dtq

[Overview]

Reference data queue information.

[Syntax]

```
ER  ref_dtq(ID dtqid, T_RDTQ *pk_rdtq);
ER  iref_dtq(ID dtqid, T_RDTQ *pk_rdtq);
```

[Parameters]

I/O	Parameter	Description
I	`ID dtqid;`	ID of data queue
O	`T_RDTQ *pk_rdtq;`	Pointer to the area storing detailed information about the data queue

[Data Queue Information T_RDTQ]

```
typedef struct  t_rdtq {
        ID      stskid;     /* Existence of tasks awaiting sending */
        ID      rtskid;     /* Existence of tasks awaiting receiving */
        UINT    sdtqcnt;    /* Data count */
        ATR     dtqatr;     /* Attribute */
        UINT    dtqcnt;     /* Max data count */
} T_RDTQ;
```

[Function]

These service calls store the information about the data queue (e.g. existence of tasks awaiting sending) specified by *dtqid* into the area specified by *pk_rdtq*.

Remark See "13.3.4 Data queue information T_RDTQ" for details about data queue information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rdtq* is invalid - *pk_rdtq* = 0

Macro	Num.	Description
E_ID	-18	*dtqid* is invalid - ID is not defined in Data queue information
E_CTX	-25	Context error - Issued from CPU lock status

13.4.6 Synchronization and communication functions (mailboxes)

Below is a list of the service calls provided by the RI850MP as synchronization and communication functions (mailboxes).

Table 13-16. Synchronization and Communication Functions (Mailboxes)

Service Call	Function Overview
snd_mbx/isnd_mbx	Send to a mailbox.
rcv_mbx	Receive from a mailbox.
prcv_mbx/iprcv_mbx	Receive from a mailbox (polling).
trcv_mbx	Receive from a mailbox (with timeout).
ref_mbx/iref_mbx	Reference mailbox information.

snd_mbx

isnd_mbx

[Overview]

Send to a mailbox.

[Syntax]

```
ER  snd_mbx(ID mbxid, T_MSG *pk_msg);
ER  isnd_mbx(ID mbxid, T_MSG *pk_msg);
```

[Parameters]

I/O	Parameter	Description
I	`ID mbxid;`	ID of mailbox
I	`T_MSG *pk_msg;`	Pointer to area in which message is stored

[Message (No Priority) T_MSG]

```
typedef struct  t_msg {
        struct  t_msg   *msgque;    /* Reserved for future use */
        .........                   /* Message body */
        .........
} T_MSG;
```

[Message (with Priority) T_MSG_PRI]

```
typedef struct  t_msg_pri {
        T_MSG   msgque;             /* Reserved for future use */
        PRI     msgpri;             /* Priority */
        .........                   /* Message body */
        .........
} T_MSG_PRI;
```

[Function]

This service call transmits the message specified by *pk_msg* to the mailbox specified by *mbxid* (queues the message in the wait queue).

If a task has been queued to the reception wait queue of the target mailbox when this service call is issued, this service call does not send the message, and instead transfers the message to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from WAITING state for a mailbox to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. The RI850MP's synchronization and communication functions (mailboxes) only receive and pass the start address of the message as their message send and receive processing. They do not copy the contents of the message in question into a separate area.

2. Messages are queued in the mailbox wait queue according to the queuing method specified by Attribute mbxatr (order in which message-send requests were made or order of message priority).

3. See "13.3.9 Message (no priority) T_MSG" and "13.3.10 Message (with priority) T_MSG_PRI" for details about messages (no priority) and messages (with priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_msg or* msgpri is invalid - *pk_msg* = 0 - msgpri ≤ 0 - msgpri > Maximum priority maxmpri
E_ID	-18	*mbxid* is invalid - ID is not identified in Mailbox information
E_CTX	-25	Context error - Issued from CPU lock status

rcv_mbx

[Overview]

Receive from a mailbox.

[Syntax]

```
ER  rcv_mbx(ID mbxid, T_MSG **ppk_msg);
```

[Parameters]

I/O	Parameter	Description
I	`ID mbxid;`	ID of mailbox
O	`T_MSG **ppk_msg;`	Start address of the message packet received from the mailbox

[Message (No Priority) T_MSG]

```
typedef struct  t_msg {
        struct  t_msg   *msgque;    /* Reserved for future use */
        .........                   /* Message body */
        .........
} T_MSG;
```

[Message (with Priority) T_MSG_PRI]

```
typedef struct  t_msg_pri {
        T_MSG   msgque;             /* Reserved for future use */
        PRI     msgpri;             /* Priority */
        .........                   /* Message body */
        .........
} T_MSG_PRI;
```

[Function]

Receive a message from the mailbox specified by *mbxid*, and store the start address of the message to the area specified by *ppk_msg*.

If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service queues the invoking task to the target mailbox wait queue and moves it from the RUNNING state to the WAITING state for a mailbox.

The WAITING state for a mailbox is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Mailbox	Return Value
A message was transmitted to the target mailbox as a result of issuing snd_mbx/isnd_mbx.	E_OK
Forced release from WAITING state for a mailbox as a result of issuing rel_wai/irel_wai.	E_RLWAI

Remarks 1. Tasks are queued in the mailbox wait queue according to the queuing method specified by Attribute mbxatr (order in which message-receipt requests were made or order of task priority).

2. If the WAITING state for a mailbox is forcibly released by issuing rel_wai/irel_wai, the contents of the area specified by *ppk_msg* will be undefined.

3. See "13.3.9 Message (no priority) T_MSG" and "13.3.10 Message (with priority) T_MSG_PRI" for details about messages (no priority) and messages (with priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*ppk_msg* is invalid - *ppk_msg* = 0
E_ID	-18	*mbxid* is invalid - ID is not identified in Mailbox information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of WAITING state for a mailbox - Issuance of rel_wai/irel_wai

prcv_mbx

iprcv_mbx

[Overview]

Receive from a mailbox (polling).

[Syntax]

```
ER  prcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER  iprcv_mbx(ID mbxid, T_MSG **ppk_msg);
```

[Parameters]

I/O	Parameter	Description
I	`ID mbxid;`	ID of mailbox
O	`T_MSG **ppk_msg;`	Start address of the message packet received from the mailbox

[Message (No Priority) T_MSG]

```
typedef struct  t_msg {
        struct  t_msg   *msgque;    /* Reserved for future use */
        .........                   /* Message body */
        .........
} T_MSG;
```

[Message (with Priority) T_MSG_PRI]

```
typedef struct  t_msg_pri {
        T_MSG   msgque;             /* Reserved for future use */
        PRI     msgpri;             /* Priority */
        .........                   /* Message body */
        .........
} T_MSG_PRI;
```

[Function]

Receive a message from the mailbox specified by *mbxid*, and store the start address of the message to the area specified by *ppk_msg*.

If the message could not be received from the target mailbox (no messages were queued in the wait queue) when this service call is issued, "E_TMOUT (= -50)" is returned.

Remark See "13.3.9 Message (no priority) T_MSG" and "13.3.10 Message (with priority) T_MSG_PRI" for details about messages (no priority) and messages (with priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*ppk_msg* is invalid - *ppk_msg* = 0
E_ID	-18	*mbxid* is invalid - ID is not identified in Mailbox information
E_CTX	-25	Context error - Issued from CPU lock status
E_TMOUT	-50	Polling failure - No messages are in the wait queue of the target mailbox

trcv_mbx

[Overview]

Receive from a mailbox (with timeout).

[Syntax]

```
ER  trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	ID *mbxid*;	ID of mailbox
O	T_MSG ***ppk_msg*;	Start address of the message packet received from the mailbox
I	TMO *tmout*;	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Message (No Priority) T_MSG]

```
typedef struct  t_msg {
        struct  t_msg   *msgque;    /* Reserved for future use */
        .........                   /* Message body */
        .........
} T_MSG;
```

[Message (with Priority) T_MSG_PRI]

```
typedef struct  t_msg_pri {
        T_MSG   msgque;             /* Reserved for future use */
        PRI     msgpri;             /* Priority */
        .........                   /* Message body */
        .........
} T_MSG_PRI;
```

[Function]

Receive a message from the mailbox specified by *mbxid*, and store the start address of the message to the area specified by *ppk_msg*.

If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service queues the invoking task to the target mailbox wait queue and moves it from RUNNING state to WAITING state for a mailbox.

The WAITING state for a mailbox is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Mailbox	Return Value
A message was transmitted to the target mailbox as a result of issuing snd_mbx/isnd_mbx.	E_OK
Forced release from WAITING state for a mailbox as a result of issuing rel_wai/irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout).	E_TMOUT

Remarks 1. Tasks are queued in the mailbox wait queue according to the queuing method specified by Attribute mbxatr (order in which message-receipt requests were made or order of task priority).

2. If the WAITING state for a mailbox is forcibly released by issuing rel_wai/irel_wai, or the time specified by *tmout* has elapsed (timeout), the contents of the area specified by parameter *ppk_msg* will be undefined.

3. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to rcv_mbx. If it is specified as TMO_POL, then the processing is equivalent to prcv_mbx/iprcv_mbx.

4. See "13.3.9 Message (no priority) T_MSG" and "13.3.10 Message (with priority) T_MSG_PRI" for details about messages (no priority) and messages (with priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*ppk_msg* or *tmout* is invalid - *ppk_msg* = 0 - *tmout* < TMO_FEVR
E_ID	-18	*mbxid* is invalid - ID is not identified in Mailbox information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of WAITING state for a mailbox - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

ref_mbx

iref_mbx

[Overview]

Reference mailbox information.

[Syntax]

```
ER  ref_mbx(ID mbxid, T_RMBX *pk_rmbx);
ER  iref_mbx(ID mbxid, T_RMBX *pk_rmbx);
```

[Parameters]

I/O	Parameter	Description
I	`ID mbxid;`	ID of mailbox
O	`T_RMBX *pk_rmbx;`	Pointer to area storing mailbox information

[Mailbox Information T_RMBX]

```
typedef struct  t_rmbx {
        ID      wtskid;     /* Existence of waiting task */
        T_MSG   *pk_msg;    /* Existence of waiting message */
        ATR     mbxatr;     /* Attribute */
} T_RMBX;
```

[Function]

The service calls store the information for the mailbox specified by *mbxid* (e.g. existence of waiting tasks) into the area specified by *pk_rmbx*.

Remark See "13.3.5 Mailbox information T_RMBX" for details about mailbox information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rmbx* is invalid - *pk_rmbx* = 0
E_ID	-18	*mbxid* is invalid - ID is not identified in Mailbox information

Macro	Num.	Description
E_CTX	-25	Context error - Issued from CPU lock status

13.4.7 Extended synchronization and communication functions

Below is a list of the service calls provided by the RI850MP as extended synchronization and communication functions.

Table 13-17. Extended Synchronization and Communication Functions

Service Call	Function Overview
loc_mtx	Acquire mutex.
ploc_mtx	Acquire mutex (polling).
tloc_mtx	Acquire mutex (with timeout).
unl_mtx	Release mutex.
ref_mtx/iref_mtx	Reference mutex state.

loc_mtx

[Overview]

Acquire mutex.

[Syntax]

```
ER  loc_mtx(ID mtxid);
```

[Parameters]

I/O	Parameter	Description
I	`ID mtxid;`	ID of mutex

[Function]

This service call acquires the mutex specified by *mtxid*.

If the target mutex cannot be acquired (another task has already acquired it) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to WAITING state for a mutex.

The WAITING state for a mutex is cancelled in the following cases, and then the state is moved to the READY state.

Cancellation of WAITING State for a Mutex	Return Value
The target mutex was released as a result of issuing unl_mtx.	E_OK
The target mutex was released as a result of issuing ext_tsk.	E_OK
The target mutex was released as a result of issuing ter_tsk.	E_OK
The WAITING state for a mutex was forcibly released as a result of issuing rel_wai/irel_wai.	E_RLWAI

Remark The queuing order of tasks in the mutex wait queue is determined by the queuing method specified by Attribute mtxatr (order in which acquire mutex requests were made, or order of task priority).

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*mtxid* is invalid - ID is not defined in Mutex information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status

Macro	Num.	Description
E_ILUSE	-28	Invalid service call use - Issued for a mutex that this task has already acquired
E_RLWAI	-49	Forced cancellation of WAITING state for a mutex - Issuance of rel_wai/irel_wai

ploc_mtx

[Overview]

Acquire mutex (polling).

[Syntax]

```
ER  ploc_mtx(ID mtxid);
```

[Parameters]

I/O	Parameter	Description
I	`ID mtxid;`	ID of mutex

[Function]

This service call acquires the mutex specified by *mtxid*.

If the target mutex cannot be acquired (another task already acquired it) when this service call is issued, "E_TMOUT (= -50)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*mtxid* is invalid - ID is not defined in Mutex information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - Issued for a mutex that this task has already acquired
E_TMOUT	-50	Polling failure - Another task has acquired the target mutex

tloc_mtx

[Overview]

Acquire mutex (with timeout).

[Syntax]

```
ER  tloc_mtx(ID mtxid, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`ID mtxid;`	ID of mutex
I	`TMO tmout;`	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Function]

This service call acquires the mutex specified by *mtxid.*

If the target mutex cannot be acquired (another task has already acquired it) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to WAITING state for a mutex.

The WAITING state for a mutex is cancelled in the following cases, and then the state is moved to the READY state.

Cancellation of WAITING State for a Mutex	Return value
The target mutex was released as a result of issuing unl_mtx.	E_OK
The target mutex was released as a result of issuing ext_tsk.	E_OK
The target mutex was released as a result of issuing ter_tsk.	E_OK
The WAITING state for a mutex was forcibly released as a result of issuing rel_wai/irel_wai.	E_RLWAI
Time specified by *tmout* has elapsed (timeout).	E_TMOUT

Remarks 1. The queuing order of tasks in the mutex wait queue is determined by the queuing method specified by Attribute mtxatr (order in which acquire mutex requests were made, or order of task priority).

2. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to lloc_mtx. If it is specified as TMO_POL, then the processing is equivalent to ploc_mtx.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

Macro	Num.	Description
E_PAR	-17	*tmout* is invalid - *tmout* < TMO_FEVR
E_ID	-18	*mtxid* is invalid - ID is not defined in Mutex information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_ILUSE	-28	Invalid service call use - Issued for a mutex that this task has already acquired
E_RLWAI	-49	Forced cancellation of WAITING state for a mutex - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

unl_mtx

[Overview]

Release mutex.

[Syntax]

```
ER  unl_mtx(ID mtxid);
```

[Parameters]

I/O	Parameter	Description
I	`ID mtxid;`	ID of mutex

[Function]

This service call releases the mutex specified by *mtxid*.

If a task has been queued to the target mutex wait queue when this service call is issued, the mutex release operation is not performed, and the mutex is passed to the task (first task in wait queue). This removes the target task from the wait queue, and changes its status from WAITING state for a mutex to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*mtxid* is invalid - ID is not defined in Mutex information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status
E_ILUSE	-28	Invalid service call use - Issued for a mutex that this task has already released - Issued for a mutex that another task has acquired

ref_mtx

iref_mtx

[Overview]

Reference mutex state.

[Syntax]

```
ER  ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
ER  iref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

[Parameters]

I/O	Parameter	Description
I	`ID mtxid;`	ID of mutex
O	`T_RMTX *pk_rmtx;`	Pointer to the area storing the mutex information

[Mutex Information T_RMTX]

```
typedef struct  t_rmtx {
        ID      htskid;     /* Existence of tasks to acquire */
        ID      wtskid;     /* Existence of waiting task */
        ATR     mtxatr;     /* Attribute */
        PRI     ceilpri;    /* Reserved for future use */
} T_RMTX;
```

[Function]

The service calls store the information for the mutex specified by *mtxid* (e.g. existence of acquired tasks) into the area specified by *pk_rmtx*.

Remark See "13.3.6 Mutex information T_RMTX" for details about mutex information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rmtx is invalid* - *pk_rmtx* = 0
E_ID	-18	*mtxid is invalid* - ID is not defined in Mutex information

Macro	Num.	Description
E_CTX	-25	Context error - Issued from CPU lock status

13.4.8 Memory pool management functions

The following shows the service calls provided by the RI850MP as memory pool management functions.

Table 13-18. Memory Pool Management Functions

Service Call	Function Overview
get_mpf	Acquire fixed-sized memory block.
pget_mpf/ipget_mpf	Acquire fixed-sized memory block (polling).
tget_mpf	Acquire fixed-sized memory block (with timeout).
rel_mpf/irel_mpf	Release fixed-sized memory block.
ref_mpf/iref_mpf	Reference fixed-sized memory pool information.

get_mpf

[Overview]

Acquire fixed-sized memory block.

[Syntax]

```
ER  get_mpf(ID mpfid, VP *p_blk);
```

[Parameters]

I/O	Parameter	Description
I	`ID mpfid;`	ID of fixed-sized memory pool
O	`VP *p_blk;`	Pointer to area storing start address of fixed-size memory block

[Function]

This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by *mpfid* and stores the start address of the block in the area specified by *p_blk*.

If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (the number of remaining blocks was already 0) when this service call is issued, this service queues the invoking task to the fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state for a fixed-size memory block.

The WAITING state for a fixed-sized memory block is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Fixed-sized Memory Block	Return Value
A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing rel_mpf/irel_mpf.	E_OK
The WAITING state for fixed-size memory block acquisition was forcibly released as a result of issuing rel_wai/irel_mpf.	E_RLWAI

Remarks 1. The queuing order of tasks in the fixed-sized memory pool wait queue is determined by the queuing method specified by Attribute mpfatr (order in which acquire fixed-size memory block requests were made, or order of task priority).

2. If the receiving WAITING state for fixed-size memory block acquisition is forcibly released by issuing rel_wai/irel_wai, the contents of the area specified by *p_blk* will be undefined.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_blk* is invalid - *p_blk* = 0

Macro	Num.	Description
E_ID	-18	*mpfid* is invalid - ID is not identified in Fixed-sized memory pool information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of WAITING state for a fixed-size memory block - Issuance of rel_wai/irel_wai

pget_mpf

ipget_mpf

[Overview]

Acquire fixed-sized memory block (polling).

[Syntax]

```
ER  pget_mpf(ID mpfid, VP *p_blk);
ER  ipget_mpf(ID mpfid, VP *p_blk);
```

[Parameters]

I/O	Parameter	Description
I	`ID mpfid;`	ID of fixed-sized memory pool
O	`VP *p_blk;`	Pointer to area storing start address of fixed-size memory block

[Function]

This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by *mpfid* and stores the start address of the block in the area specified by *p_blk*.

If a fixed-size memory block cannot be acquired from the fixed-sized memory pool (the number of remaining blocks is already 0) when this service call is issued, "E_TMOUT (= -50)" is returned.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_blk is invalid* - *p_blk* = 0
E_ID	-18	*mpfid* is invalid - ID is not identified in Fixed-sized memory pool information
E_CTX	-25	Context error - Issued from CPU lock status
E_TMOUT	-50	Polling failure - The number of blocks that can be acquired from the fixed-sized memory pool is 0

tget_mpf

[Overview]

Acquire fixed-sized memory block (with timeout).

[Syntax]

```
ER  tget_mpf(ID mpfid, VP *p_blk, TMO tmout);
```

[Parameters]

I/O	Parameter	Description
I	`ID mpfid;`	ID of fixed-sized memory pool
O	`VP *p_blk;`	Pointer to area storing start address of fixed-size memory block
I	`TMO tmout;`	Specified timeout (in millisecond) TMO_FEVR : Wait forever TMO_POL : Polling Numerical value : Wait time

[Function]

This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by *mpfid* and stores the start address of the block in the area specified by *p_blk*.

If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (the number of remaining blocks was already 0) when this service call is issued, this service queues the invoking task to the fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state for a fixed-size memory block.

The WAITING state for a fixed-sized memory block is cancelled in the following cases, and then moved to the READY state.

Cancellation of WAITING State for a Fixed-sized Memory Block	Return Value
A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing rel_mpf/irel_mpf.	E_OK
The WAITING state for fixed-size memory block acquisition was forcibly released as a result of issuing rel_wai/irel_mpf.	E_RLWAI
Time specified by *tmout* has elapsed (timeout)	E_TMOUT

Remarks 1. The queuing order of tasks in the fixed-sized memory pool wait queue is determined by the queuing method specified by Attribute mpfatr (order in which acquire fixed-size memory block requests were made, or order of task priority).

2. If the WAITING state for a fixed-size memory block is forcibly released by issuing rel_wai/irel_wai, or the time specified by tmout has elapsed (timeout), the contents of the area specified by *p_blk* will be undefined.

3. If *tmout* is specified as TMO_FEVR, then the processing is equivalent to get_mpf. If it is specified as TMO_POL, then the processing is equivalent to pget_mpf/ipget_mpf.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_blk* or *tmout* is invalid - *p_blk* = 0 - tmout < TMO_FEVR
E_ID	-18	*mpfid* is invalid - ID is not identified in Fixed-sized memory pool information
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status - Issued from dispatching disabled status
E_RLWAI	-49	Forced cancellation of WAITING state for a fixed-size memory block - Issuance of rel_wai/irel_wai
E_TMOUT	-50	Timeout - Time specified by *tmout* has elapsed (timeout)

rel_mpf

irel_mpf

[Overview]

Release fixed-sized memory block.

[Syntax]

```
ER  rel_mpf(ID mpfid, VP blk);
ER  irel_mpf(ID mpfid, VP blk);
```

[Parameters]

I/O	Parameter	Description
I	`ID mpfid;`	ID of fixed-sized memory pool
I	`VP blk;`	Start address of the memory block to be released

[Function]

This service call returns the fixed-sized memory block specified by *blk* to the fixed-sized memory pool specified by *mpfid*.

If a task is queued to the target fixed-sized memory pool wait queue when this service call is issued, block return processing is not performed, and instead the fixed-sized memory block is returned to the relevant task (first task of wait queue). This removes the target task from the wait queue, and changes its status from WAITING state for a fixed-size memory block to READY state, or from WAITING-SUSPENDED state to SUSPENDED state.

Remarks 1. When this service call returns a fixed-size memory block, it does not perform a clear. The contents of the returned fixed-size memory block are therefore undefined.

2. If a fixed-size memory block is returned to a fixed-sized memory pool that is different from the one it was acquired from, subsequent behavior is not guaranteed.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*blk* is invalid - *blk* = 0
E_ID	-18	*mpfid* is invalid - ID is not identified in Fixed-sized memory pool information
E_CTX	-25	Context error - Issued from CPU lock status

ref_mpf

iref_mpf

[Overview]

Reference fixed-sized memory pool information.

[Syntax]

```
ER  ref_mpf(ID mpfid, T_RMFP *pk_rmpf);
ER  iref_mpf(ID mpfid, T_RMPF *pk_rmpf);
```

[Parameters]

I/O	Parameter	Description
I	`ID mpfid;`	ID of fixed-sized memory pool
O	`T_RMPF *pk_rmpf;`	Pointer to area storing fixed-sized memory pool information

[Fixed-Sized Memory Pool Information T_RMPF]

```
typedef struct  t_rmpf {
        ID      wtskid;     /* Existence of waiting task */
        UINT    fblkcnt;    /* Number of available blocks remaining */
        ATR     mpfatr;     /* Attribute */
} T_RMPF;
```

[Function]

Stores fixed-sized memory pool state packet (ID number of the task at the head of the wait queue, number of free memory blocks, etc.) of the fixed-sized memory pool specified by *mpfid* in the area specified by *pk_rmpf*.

Remark See "13.3.7 Fixed-sized memory pool information T_RMPF" for details about fixed-sized memory pool information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rmpf* is invalid - *pk_rmpf* = 0
E_ID	-18	*mpfid* is invalid - ID is not identified in Fixed-sized memory pool information

Macro	Num.	Description
E_CTX	-25	Context error - Issued from CPU lock status

13.4.9 Time management functions

The following shows the service calls provided by the RI850MP as time management functions.

Table 13-19. Time Management Functions

Service Call	Function Overview
set_tim/iset_tim	Change system time.
get_tim/iget_tim	Reference system time.
sta_cyc/ista_cyc	Start cyclic handler.
stp_cyc/istp_cyc	End cyclic handler.
ref_cyc/iref_cyc	Reference cyclic handler information.

set_tim

iset_tim

[Overview]

Change system time.

[Syntax]

```
ER  set_tim(SYSTEM *p_systim);
ER  iset_tim(SYSTEM *p_systim);
```

[Parameters]

I/O	Parameter	Description
I	SYSTIM *p_systim;	Pointer to area storing system time (in milliseconds)

[System Time SYSTIM]

```
typedef struct  t_systim {
        UW      ltime;    /* System time (lower 32 bits) */
        UH      utime;    /* System time (higher 16 bits) */
} SYSTIM;
```

[Function]

These service calls change the system time (in milliseconds) to the time specified by *p_systim*.

Remarks 1. Issuing this service call will not affect the wait times of tslp_tsk, dly_tsk, twai_sem, or the like, or the cyclic handler activation phase or activation frequency.

2. See "13.3.11 System time SYSTIM" for details about system time.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_systim* is invalid - *p_systim* = 0
E_CTX	-25	Context error - Issued from CPU lock status

get_tim

iget_tim

[Overview]

Reference system time.

[Syntax]

```
ER  get_tim(SYSTIM *p_systim);
ER  iget_tim(SYSTIM *p_systim);
```

[Parameters]

I/O	Parameter	Description
O	`SYSTIM *p_systim;`	Pointer to area storing the system time

[System Time SYSTIM]

```
typedef struct  t_systim {
        UW      ltime;    /* System time (lower 32 bits) */
        UH      utime;    /* System time (higher 16 bits) */
} SYSTIM;
```

[Function]

This acquires the system time, and stores it in the area specified by *p_systim*.

Remark See "13.3.11 System time SYSTIM" for details about system time.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_systim* is invalid - *p_systim* = 0
E_CTX	-25	Context error - Issued from CPU lock status

sta_cyc

ista_cyc

[Overview]

Start cyclic handler.

[Syntax]

```
ER  sta_cyc(ID cycid);
ER  ista_cyc(ID cycid);
```

[Parameters]

I/O	Parameter	Description
I	`ID cycid;`	ID of cyclic handler

[Function]

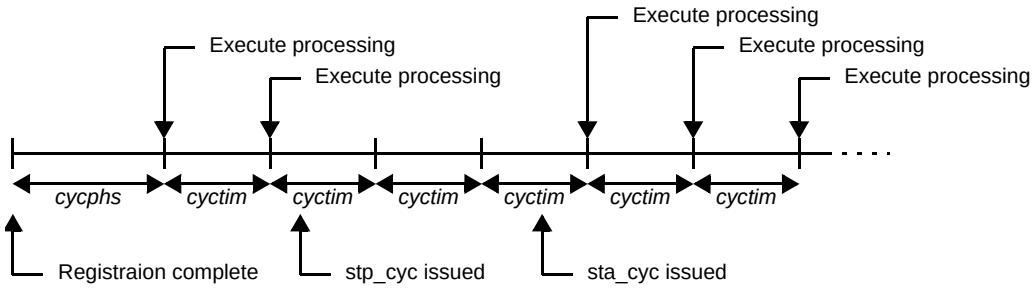
This issues an activation request to the cyclic handler specified by *cycid*, and moves the task from non-operational state to operating state.

The behavior from the issuance of this service call until the first time that the target cyclic handler performs processing will depend on whether the TA_PHS attribute (save flag: save activation phase) is assigned to the target cyclic handler.

- If the TA_PHS attribute has been assigned

  The target cyclic handler performs processing with timing adhering to the Activation phase cycphs and Cycle start cyctim, using the completion of registration of the target cyclic handler in the Kernel Initialization Module as a reference point.

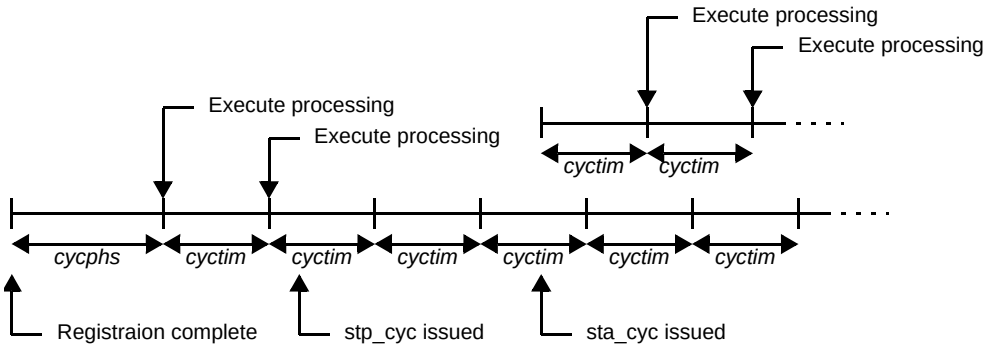
  The timing of processing when the TA_STA attribute (initial state: operating state) is assigned to the target cyclic handler is shown below.



- If the TA_PHS attribute has not been assigned

  The target cyclic handler performs processing with the timing of Cycle start cyctim, using the issuance of this service call as the reference point.

  The timing of processing when the TA_STA attribute (initial state: operating state) is assigned to the target cyclic handler is shown below.



Remark This service call does not perform queuing of activation requests. If the target cyclic handler is in other than non-operational state, then no actions will be performed, and it will also not be treated as an error.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*cycid* is invalid - ID is not defined in Cyclic handler information
E_CTX	-25	Context error - Issued from CPU lock status

stp_cyc

istp_cyc

[Overview]

End cyclic handler.

[Syntax]

```
ER  stp_cyc(ID cycid);
ER  istp_cyc(ID cycid);
```

[Parameters]

I/O	Parameter	Description
I	`ID cycid;`	ID of cyclic handler

[Function]

This issues a termination request to the cyclic handler specified by *cycid,* and moves the task from operating state to non-operational state.

Remark This service call does not perform queuing of termination requests. If the target cyclic handler is in other than operational state, then no actions will be performed, and it will also not be treated as an error.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_ID	-18	*cycid* is invalid - ID is not defined in Cyclic handler information
E_CTX	-25	Context error - Issued from CPU lock status

ref_cyc

iref_cyc

[Overview]

Reference cyclic handler information.

[Syntax]

```
ER  ref_cyc(ID cycid, T_RCYC *pk_rcyc);
ER  iref_cyc(ID cycid, T_RCYC *pk_rcyc);
```

[Parameters]

I/O	Parameter	Description
I	`ID cycid;`	ID of cyclic handler
O	`T_RCYC *pk_rcyc;`	Pointer to the area storing the cyclic handler information

[Cyclic handler information T_RCYC]

```
typedef struct  t_rcyc {
        STAT    cycstat;    /* Current state */
        RELTIM  lefttim;    /* Remaining time */
        ATR     cycatr;     /* Attribute */
        RELTIM  cyctim;     /* Activation cycle */
        RELTIM  cycphs;     /* Starting phase */
        PE_ID   peid;       /* PE number */
} T_RCYC;
```

[Function]

This stores the cyclic handler information (e.g. current state) of the cyclic handler specified by *cycid* in the area specified by *pk_rcyc*.

Remark See "13.3.8 Cyclic handler information T_RCYC" for details about cyclic handler information.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*pk_rcyc* is invalid - *pk_rcyc* = 0

Macro	Num.	Description
E_ID	-18	*cycid* is invalid - ID is not defined in Cyclic handler information
E_CTX	-25	Context error - Issued from CPU lock status

13.4.10 System state management functions

The following shows the service calls provided by the RI850MP as system state management functions.

Table 13-20. System State Management Functions

Service Call	Function Overview
rot_rdq/irot_rdq	Rotate priority.
get_tid/iget_tid	Get ID.
loc_cpu/iloc_cpu	Transition to locked CPU state.
unl_cpu/iunl_cpu	Transition to unlocked CPU state.
dis_dsp	Transition to dispatching disabled state.
ena_dsp	Transition to dispatching enabled state.
sns_ctx	Get context type (non-task context or task context).
sns_loc	Get system state type (CPU locked state or CPU unlocked state).
sns_dsp	Get system state type (dispatching disabled state or dispatching enabled state).
sns_dpn	Get system state type (dispatching hold state or non-dispatching hold state).

rot_rdq

irot_rdq

[Overview]

Rotate priority.

[Syntax]

```
ER  rot_rdq(PRI tskpri);
ER  irot_rdq(PRI tskpri);
```

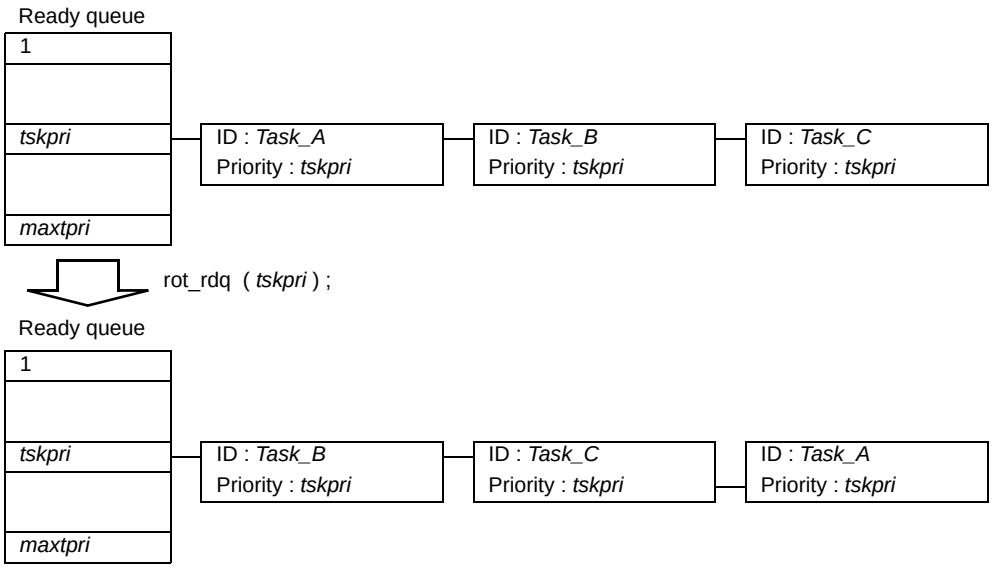
[Parameters]

I/O	Parameter	Description
I	`PRI tskpri;`	Task priority TPRI_SELF : Current priority of invoking task Numerical value : Task priority

[Function]

This issues a rotate request to the ready queue, and explicity changes the task execution order. This reverses the order of the task in the ready queue, with tasks having higher priority specified by *tskpri* (at the front of the queue) last, and the tasks with lowest priority first.

Remarks 1. Below are described the changes that issuing this service call causes to the state of the ready queue.



2. This service call does not perform queuing of rotation requests. If no tasks in the ready queue have the target priority, then no action will be performed, and it will not be handled as an error either.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*tskpri* is invalid - *tskpri* < 0 - *tskpri* > Maximum priority maxtpri - TPRI_SELF specified when issued from non-task
E_CTX	-25	Context error - Issued from CPU lock status

get_tid

iget_tid

[Overview]

Get ID.

[Syntax]

```
ER  get_tid(ID *p_tskid);
ER  iget_tid(ID *p_tskid);
```

[Parameters]

I/O	Parameter	Description
O	`ID *p_tskid;`	Pionter to area storing ID

[Function]

This acquires the ID of the task that has transitioned to RUNNING state, and stores it in the area specified by *p_tskid*.

Remark If no task has gone to the RUNNING state, TSK_NONE (= 0) will be stored in the area specified by *p_tskid*.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_tskid* is invalid - *p_tskid* = 0
E_CTX	-25	Context error - Issued from CPU lock status

loc_cpu

iloc_cpu

[Overview]

Transition to locked CPU state.

[Syntax]

```
ER  loc_cpu(void);
ER  iloc_cpu(void);
```

[Parameters]

None.

[Function]

These service calls change the system status type from the CPU unlocked state to the CPU locked state.

This will prevent dispatcher task switching, and disable the acknowledgement of all maskable interrupts, from the time this service call is issued, until unl_cpu/iunl_cpu is issued.

Remarks 1. CPU locked state (transition via this service call) is defined as follows.
- Dispatcher execution is prevented
- Acceptance of maskable interrupts is disabled

2. This service call does not perform queuing of lock requests. As a result, if the system status type cannot be changed when this service call is issued (the system status type was already CPU locked state), then no actions will be performed, and it will also not be treated as an error.

3. If a maskable interrupt is raised when in CPU locked state, transition to the corresponding interrupt handler will be delayed until unl_cpu/iunl_cpu is issued.

4. The RI850MP uses a timer interrupt, which is a type of maskable interrupt, as its time management function (e.g. updating the system time, task timeouts, and activation of cyclic handlers). As a result, if the CPU is continuously in locked state for longer than the interval specified by Base clock cycle tbase, the time management function may not operate correctly.

5. The RI850MP only allows the following 8 service calls to be issued while in CPU locked state.
loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, sns_ctx, sns_loc, sns_dsp, sns_dpn

6. The internal processing of this service call (Interrupt mask logical OR routine : _kernel_usr_msk_intmsk; Interrupt mask acquisition routine : _kernel_usr_get_intmsk) is dependent on the user's execution environment. For this reason, the RI850MP separates this processing from the "User-own" coding module, and provides it as a sample source file. See "9.2.1 Interrupt mask logical OR routine" and "9.2.2 Interrupt mask acquisition routine" for details about _kernel_usr_msk_intmsk and _kernel_usr_get_intmsk.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination

Macro	Num.	Description
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

unl_cpu

iunl_cpu

[Overview]

Transition to unlocked CPU state.

[Syntax]

```
ER  unl_cpu(void);
ER  iunl_cpu(void);
```

[Parameters]

None.

[Function]

These service calls change the system status type from the CPU locked state to the CPU unlocked state.

This performs dispatcher task-switching processing; this processing was disabled by the issuance of loc_cpu/iloc_cpu, and enables the acknowledgment of maskable interrupts, which had been disabled.

Remarks 1. This service call does not enable dispatching if it was disabled.

2. This service call does not perform queuing of cancellation requests. As a result, if the system status type cannot be changed when this service call is issued (the system status type was already CPU unlocked state), then no actions will be performed, and it will also not be treated as an error.

3. The internal processing of this service call (Interrupt mask overwrite routine : _kernel_usr_set_intmsk) is dependent on the user's execution environment. For this reason, the RI850MP separates this processing from the "User-own" coding module, and provides it as a sample source file. See "9.2.3 Interrupt mask overwrite routine" for details about "_kernel_usr_set_intmsk".

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

dis_dsp

[Overview]

Transition to dispatching disabled state.

[Syntax]

```
ER  dis_dsp(void);
```

[Parameters]

None.

[Function]

This service call moves the system status type to the dispatch disabled state.
As a result, task switching by the dispatcher is disabled from the time this service call is issued, until ena_dsp is issued.

Remarks 1. Dispatch disabled state (transition via this service call) is defined as follows.
- Dispatcher execution is prevented

2. This service call does not perform queuing of disable requests. As a result, if the system status type cannot be changed when this service call is issued (the system status type was already dispatch disabled state), then no actions will be performed, and it will also not be treated as an error.

3. Service calls issued when dispatching is disabled (e.g. act_tsk, chg_pri, and wup_tsk) only perform queue operations, counter operations, and other processing. Actual task switching is delayed until ena_dsp is issued.

4. If a service call that could cause the invoking task's state to change (e.g. slp_tsk, wai_sem, and wai_flg) while dispatching is disabled, the RI850MP will return "E_CTX (= -25)", regardless of the request conditions are instantly met.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status

ena_dsp

[Overview]

Transition to dispatching enabled state.

[Syntax]

```
ER  ena_dsp(void);
```

[Parameters]

None.

[Function]

This service call moves the system status to the dispatch enabled state.
This enables task switching by the dispatcher, which had been disabled by issuing dis_dsp.

Remark This service call does not perform queuing of enable requests. As a result, if the system status type cannot be changed when this service call is issued (the system status type was already dispatch enabled state), then no actions will be performed, and it will also not be treated as an error.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_CTX	-25	Context error - Issued from non-task - Issued from CPU lock status

sns_ctx

[Overview]

Get context type (non-task context or task context).

[Syntax]

```
BOOL  sns_ctx(void);
```

[Parameters]

None.

[Function]

This service call acquires the context type of the processing program that issued this service call (non-task context or task context), and returns it as the return value.

Remark For the purposes of this service call, a non-task context processing program is defined as one of the following:

- Cyclic handlers
- Interrupt handlers
- CPU exception handlers
- Initialization routine

[Return Value]

Macro	Num.	Description
TRUE	1	Normal termination - Non-task context
FALSE	0	Normal termination - Task context
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

sns_loc

[Overview]

Get system state type (CPU locked state or CPU unlocked state).

[Syntax]

```
BOOL  sns_loc(void);
```

[Parameters]

None.

[Function]

This service call acquires the system status type as of the time it is issued (CPU locked state or non-CPU locked state), and returns it as the return value.

Remark CPU locked state (transition via issuance of loc_cpu/iloc_cpu) is defined as follows.
- Dispatcher execution is prevented
- Acceptance of maskable interrupts is disabled

[Return Value]

Macro	Num.	Description
TRUE	1	Normal termination - CPU locked state
FALSE	0	Normal termination - CPU unlocked state
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

sns_dsp

[Overview]

Get system state type (dispatching disabled state or dispatching enabled state).

[Syntax]

```
BOOL  sns_dsp(void);
```

[Parameters]

None.

[Function]

This service call acquires the system status type as of the time it is issued (dispatching disabled or enabled), and returns it as the return value.

Remark Dispatch disabled state (transition via issuance of dis_dsp) is defined as follows.
- Dispatcher execution is prevented

[Return Value]

Macro	Num.	Description
TRUE	1	Normal termination - Dispatching disabled state
FALSE	0	Normal termination - Dispatching enabled state
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

sns_dpn

[Overview]

Get system state type (dispatching hold state or non-dispatching hold state).

[Syntax]

```
BOOL  sns_dpn(void);
```

[Parameters]

None.

[Function]

This service call acquires the system status type as of the time it is issued (dispatching hold state or non-dispatching hold state), and returns it as the return value.

Remark Dispatch hold state is defined as follows.
- A process with higher priority than the dispatcher is running
- Dispatcher execution is prevented
- Acceptance of maskable interrupts is disabled

[Return Value]

Macro	Num.	Description
TRUE	1	Normal termination - Dispatch hold state
FALSE	0	Normal termination - Non-dispatch hold state
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call

13.4.11 Interrupt management functions

The following shows the service calls provided by the RI850MP as interrupt management functions.

Table 13-21. Interrupt Management Functions

Service Call	Function Overview
dis_int	Disable maskable interrupt acknowledgement.
ena_int	Enable maskable interrupt acknowledgement.
chg_ipm/ichg_ipm	Change priority mask register.
get_ipm/iget_ipm	Reference priority mask register.

dis_int

[Overview]

Disable maskable interrupt acknowledgement.

[Syntax]

```
ER  dis_int(INTNO intno);
```

[Parameters]

I/O	Parameter	Description
I	`INTNO intno;`	Exception cause code

[Function]

This service call disables acknowledgment of maskable interrupts corresponding to the exception cause code specified by *intno*.

This will disable acknowledgement of the specified maskable interrupt from the time this service call is issued until ena_int is issued.

Remarks 1. This service call does not perform queuing of disable requests. As a result, if the acceptance state of maskable interrupts could not be changed when this service call was issued (acceptance of the maskable interrupt had already been disabled), then no operation will be performed, and it will not be treated as an error.

2. If a maskable interrupt is raised when the acceptance of maskable interrupts is disabled, the transition to the corresponding interrupt handler is delayed until ena_int is issued.

3. The RI850MP uses a timer interrupt, which is a type of maskable interrupt, as its time management function (e.g. updating the system time, task timeouts, and activation of cyclic handlers). As a result, if the acceptance of timer interrupts is disabled continuously for longer than the interval specified by Base clock cycle tbase , the time management function may not operate correctly.

4. The internal processing of this service call (Disable interrupt routine : _kernel_usr_dis_int) is dependent on the user's execution environment. For this reason, the RI850MP separates this processing from the "User-own" coding module, and provides it as a sample source file. See "9.2.4 Disable interrupt routine" for details about "_kernel_usr_dis_int".

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*intno* is invalid - 0x0 ≤ *intno* ≤ 0x70 - Not supported on target device exception cause code

Macro	Num.	Description
E_CTX	-25	Context error - Issued from CPU lock status

ena_int

[Overview]

Enable maskable interrupt acknowledgement.

[Syntax]

```
ER  ena_int(INTNO intno);
```

[Parameters]

I/O	Parameter	Description
I	`INTNO intno;`	Exception cause code

[Function]

This service call enables acknowledgment of maskable interrupts corresponding to the exception cause code specified by *intno*.

Remarks 1. This service call does not perform queuing of enable requests. As a result, if the acceptance state of maskable interrupts could not be changed when this service call was issued (acceptance of the maskable interrupt had already been enabled), then no operation will be performed, and it will not be treated as an error.

2. The internal processing of this service call (Enable interrupt routine : _kernel_usr_ena_int) is dependent on the user's execution environment. For this reason, the RI850MP separates this processing from the "User-own" coding module, and provides it as a sample source file. See "9.2.5 Enable interrupt routine" for details about "_kernel_usr_ena_int".

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*intno* is invalid - 0x0 ≤ *intno* ≤ 0x70 - Not supported on target device exception cause code
E_CTX	-25	Context error - Issued from CPU lock status

chg_ipm

ichg_ipm

[Overview]

Change priority mask register.

[Syntax]

```
ER  chg_ipm(INTPMR ipmptn);
ER  ichg_ipm(INTPMR ipmptn);
```

[Parameters]

I/O	Parameter	Description
I	`INTPMT ipmptn;`	Register value after change

[Function]

This changes the value of the Priority Mask Register (PMR) to the value specified by *ipmptn*.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_CTX	-25	Context error - Issued from CPU lock status

get_ipm

iget_ipm

[Overview]

Reference priority mask register.

[Syntax]

```
ER  get_ipm(INTPMR *p_ipmptn);
ER  iget_ipm(INTPMR *p_ipmptn);
```

[Parameters]

I/O	Parameter	Description
O	`INTPMR *p_ipmptn;`	Pointer to the area in which register value is stored

[Function]

This acquires the value of the Priority Mask Register (PMR), and stores it in the area specified by *p_ipmptn*.

[Return Value]

Macro	Num.	Description
E_OK	0	Normal termination
E_NOSPT	-9	Unsupported function - In the SCT Information, there is no definition for the use of this service call
E_PAR	-17	*p_ipmptn is invalid* - *p_ipmptn* = 0
E_CTX	-25	Context error - Issued from CPU lock status

APPENDIX A CONFIGURATOR

This appendix describes the configurator.

A.1 Outline

The configurator is a utility tool that accepts configuration files as input and outputs files containing configuration data to be provided to the RI850MP (system information table file, entry file, system information header file, service call table file).

The information files output from the configurator are explained below.

- System information table file
  An information file that contains data required by the RI850MP to operate.

- Entry file
  An information file that contains assignments to branch processing (to time management functions, interrupt handlers, and CPU exception handlers) for handler addresses to which the CPU forcibly passes control when a timer interrupt, other interrupt, or CPU exception occurs.

- System information header file
  An information file that contains the correspondence between object names (task names, semaphore names, or the like) described in the system configuration file and IDs.

- Service call table file
  An information file that contains information about use of the service calls provided by the RI850MP.

Remark ".NET Framework 2.0" is required to activate the configurator.

A.2 Activation Method

A.2.1 Activating from command line

The following describes how to activate the configurator from the command line.

In the description of activation options, "C:\>" indicates the command line prompt, "Δ" indicates input with the Space key, and "<Enter>" indicates input with the Enter key. Options enclosed in "[ ]" can be abbreviated.

```
C:\> cf850mp.exe [Δ@<cmd_file>] Δ-cpuΔ<name> [Δ-devpath=<path>] [Δ-iΔ<sitfile>] [Δ-eΔ<entryfile>] [Δ-dΔ<includefile>] [Δ-cΔ<sctfile>] [Δ-ni] [Δ-ne] [Δ-nd] [Δ-nc] [Δ-tΔ<tool>] [Δ-TΔ<compiler_path>] [Δ-IΔ<include_path>] [Δ-np] [Δ-cnvΔ<cnvfile>] [Δ-V] [Δ-help] [Δ<cffile>] <Enter>
```

The details of each activation option are explained below:

Activation Options	Meaning
@<cmd_file>	Specifies the name of a file input to the configurator (command file name). - If this activation option is not specified, the configurator does not load the command file. - For details about the command file, refer to "A.3 Command File".
-cpuΔ*<name>*	Specifies the type specification name of the target device. The keyword that can be specified as *<name>* is the device file name, minus the initial character "D" and the extension ".800". - When the device file name is "DF3507.800", the keyword specified in *<name>* is "F3507".

Activation Options	Meaning
-iΔ<*sitfile*>	Specifies the name of a file output from the configurator (system information table file name). - If this activation option is not specified, processing is carried out as if "-iΔsit.c" had been specified.
-eΔ<*entryfile*>	Specify the name of a file output from the configurator (entry file name). - If this activation option is not specified, processing is carried out as if "-eΔentry.s" had been specified (CX version) or as if "-eΔentry.850" had been specified (CCV850E version).
-dΔ<*includefile*>	Specifies the name of a file output from the configurator (system information header file name). - If this activation option is not specified, processing is carried out as if "-dΔkernel_id.h" had been specified.
-cΔ<*sctfile*>	Specifies the name of a file output from the configurator (service call table file name). - If this activation option is not specified, processing is carried out as if "-cΔsct.c" had been specified.
-ni	Disables output of the system information table file. - If this activation option is specified together with the "-iΔ<*sitfile*>" option, the configurator handles this activation option as the valid option.
-ne	Disables output of the entry file. - If this activation option is specified together with the "-eΔ<*entryfile*>" option, the configurator handles this activation option as the valid option.
-nd	Disables output of the system information header file. - If this activation option is specified together with the "-dΔ<*includefile*>" option, the configurator handles this activation option as the valid option.
-nc	Disables output of the service call table file. - If this activation option is specified together with the "-cΔ<*sctfile*>" option, the configurator handles this activation option as the valid option.
-tΔ<*tool*>	Specifies the type of the C compiler package used. The only keywords that can be specified as <*tool*> are "CX" and "CCV850E". - If this activation option is not specified, processing is carried out as if "-tΔCX" had been specified.
-TΔ<*compiler_path*>	Searches for the C preprocessor of the C compiler package specified with the "-tΔ<*tool*>" option from the <compiler_path> folder. - If this activation option is not specified, the search is carried out only in the current folder and folders specifed with Windows environmental variables (PATH and so on).
-IΔ<*include_path*>	Seaches the <*include_path*> folder for header files defined in the Header file declaration. - If this activation option is not specified, the search is carried out only in the default search folder of the C compiler package specified with the he "-tΔ<*tool*>" option.
-np	Suppresses activation of the C preprocessor. - When this activation option is specified, lines in the configuration file that start with # are treated as comment lines.
-cnvΔ<*cnvfile*>	Ouputs the configuration specified with <*cffile*> as a configuration file for the RI850MP.
-V	Outputs version information for the configurator to the standard output.
-help	Outputs information about the activation options of the configurator (types, usage, and so on) to the standard output.
<*cffile*>	Specifies the name of a file input to the configurator (configuration file name). - If this activation option is not specified, the configurator does not load the configuration file. This activation option can be omitted only when the "-V" or "-help" option is specified. - For details about the configuration file, refer to "APPENDIX B CONFIGURATION FILE".

A.2.2 Activating from CubeSuite+

The configurator is activated when CubeSuite+ performs a build, in accordance with the setting on the Property panel, on the [System Configuration File Related Information] tab.

A.3 Command File

The configurator supports a command file in order to avoid the limitation on the number of characters that can be specified for activation options on the command line.

The specification formats of the command file are described below.

- Character code
  Characters must be in the ASCII character code.
  Japanese can be written in comments only, using Shift-JIS, EUC-JP, or UTF-8 character codes.

- Comments
  Lines that start with # are treated as comment lines, up to the end of the line.

APPENDIX B CONFIGURATION FILE

This appendix explains the coding method for the configuration files.

B.1 Outline

Configuration files are required for creating files (system information table file, entry file, system information header file, service call table file) that contain configuration information to be provided to the RI850MP. These files are to be coded by the user with a text editor.

The following shows the notation method for configuration files.

- Character code

  Create the file using ASCII code.

  For Japanese language coding, Shift-JIS codes, EUC-JP codes, and UTF-8 codes can be used only for comments.

- Comments

  Parts between the start-of-comment "/*" and the end-of-comment "*/", and parts from the start-of-comment "//" to the line end are regarded as comments.

  Remark When "-np" is specified as the configurator activation option, parts from the line head "#" to the line end are also regarded as comments.

- Numeric values

  Words starting with a numeric value "0 to 9" are regarded as numeric values.

  The RI850MP distinguishes numeric values as follows.

  Decimal : Words starting with "1 to 9"

  Hexadecimal : Words starting with 0x or 0X

- Names

  Words starting with an alphabetic character, "a to z, A to Z", or underscore "_" are regarded as names.

  Up to 255 characters can be specified for names.

  Remarks 1. The RI850MP distinguishes between symbol names and other names based on the context in the configuration file.

  2. The RI850MP distinguishes between lower case "a to z" and upper case "A to Z" in the configuration file.

- Symbol names

  Words starting with an alphabetic character, "a to z, A to Z", or underscore "_" are regarded as symbol names.

  Up to 4095 characters can be specified for symbol names.

  Remarks 1. The RI850MP distinguishes between symbol names and other names based on the context in the configuration file.

  2. The RI850MP distinguishes between lower case "a to z" and upper case "A to Z" in the configuration file.

  3. Symbol names can also be written in the format "symbol name + offset", but the offset values that can be specified must be a "constant expression".

- Keywords

  The words shown below are reserved as keywords for configuration files. Using these words for any other purpose specified is therefore prohibited.

  ATT_INI, CLK_INTNO, CRE_CYC, CRE_DTQ, CRE_FLG, CRE_MBX, CRE_MPF, CRE_MTX, CRE_SEM, CRE_TSK, DEF_EXC, DEF_FPSR, DEF_INH, DEF_SCT, DEF_TIM, DOMAIN, DOMAIN_ALLOCATION, INCLUDE, MAX_PRI, MEM_AREA, NULL, RI850MP, RI_SERIES, SIZE_AUTO, SYS_STK, TA_ACT, TA_ASM, TA_CLR, TA_DISINT, TA_ENAINT, TA_HLNG, TA_MFIFO, TA_MPRI, TA_PHS, TA_STA, TA_TFIFO, TA_TPRI, TA_WMUL, TA_WSGL, V100 to V199, VATT_IDL, and service call names

B.1.1 Configuration Information

The configuration information that is coded in the RI850MP is divided into the following five main types.

- Declarative Information
- System Information
- Domain Information
- Static API Information
- SCT Information

The following illustrates how the configuration file is written.

Figure B-1. Configuration File Writing Format

```
// Declarative Information
INCLUDE("h_file");                  // Header file declaration

// System Information
RI_SERIES(osnam, osver);            // RI series information
DEF_TIM(tbase);                     // Base clock interval information
CLK_INTNO(tintno);                  // Timer interrupt information
SYS_STK(sysstksz, peno);            // System stack information
MAX_PRI(maxtpri);                   // Maximum priority information
DEF_FPSR(fpsr);                     // Floating-point setting/status register information
MEM_AREA(secnam, secsz);            //Section information
DOMAIN_ALLOCATION(domnam, peno);    // Processor element information

// Domain Information
DOMAIN (domnam) {
    // Static API Information
    CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz[:secnam], stk}); // Task information
    CRE_SEM(semid, {sematr, isemcnt, maxsem});       // Semaphore information
    CRE_FLG(flgid, {flgatr, iflgptn});               // Eventflag information
    CRE_DTQ(dtqid, {dtqatr, dtqcnt[:secnam], dtq});  // Data queue information
    CRE_MBX(mbxid, {mbxatr, maxmpri, mprihd});       // Mailbox information
    CRE_MTX(mtxid, {mtxatr, ceilpri});               // Mutex information
    CRE_MPF(mpfid, {mpfatr, blkcnt, blksz[:secnam], mpf});   // Fixed-sized memory pool
information
    CRE_CYC(cycid, {cycatr, exinf, cychdr, cyctim, cycphs}); // Cyclic handler information
    DEF_INH(inhno, {inhatr, inthdr});                // Interrupt handler information
    DEF_EXC(excno, {excatr, exchdr});                // CPU exception handler information
    ATT_INI({iniatr, exinf, inirtn});                // Initialization routine information
    VATT_IDL({idlatr, idlrtn});                      // Idle routine information
}

// SCT Information
DEF_SCT(svc_nam);
```

B.2 Declarative Information

Defines the following information as the declarative information.

- Header file declaration

The following illustrates how the declarative information is written.

```
INCLUDE("h_file");                  // Header file declaration
```

B.2.1 Header file declaration

Defines the following item as the header file declaration.

- Header file name h_file

The number of definitions for header file declaration is not restricted.
The following shows the header file declaration format.

```
INCLUDE("h_file");
```

(1) Header file name *h_file*

Specify the header file name output to the system information header file.
Values that can be specified for *h_file* are limited to names.

Remarks 1. If "INCLUDE (" <itron.h> ");" is specified, the following header file definition (include processing) is output to the system information header file.

```
#include    <itron.h>
```

2. If "INCLUDE (" <itron.h> ");" is specified, the following header file definition (include processing) is output to the system information header file.

```
#include    "itron.h"
```

3. Search folder for *h_file* is the folder specified in the configurator activation option "-I∆<*include_path*>".

B.3 System Information

Defines the following information as the system information.

- RI series information
- Base clock cycle information
- Timer interrupt information
- System stack information
- Maximum priority information
- Floating-point setting/status register information
- Section information
- Processor element information

The following illustrates how the system information is written.

```
RI_SERIES(osnam, osver);            // RI series information
DEF_TIM(tbase);                     // Base clock interval information
CLK_INTNO(tintno);                  // Timer interrupt information
SYS_STK(sysstksz, peno);            // System stack information
MAX_PRI(maxtpri);                   // Maximum priority information
DEF_FPSR(fpsr);                     // Floating-point setting/status register information
MEM_AREA(secnam, secsz);            // Section information
DOMAIN_ALLOCATION(domnam, peno);    // Processor element information
```

B.3.1 RI series information

Defines the following item as the information about real-time OS.

- Real-time OS name osnam
- Version information osver

Only one information item can be defined as RI series information.
The following shows the RI series information format.

```
RI_SERIES(osnam, osver);
```

(1) Real-time OS name *osnam*

Specifies the real-time OS name.
RI850MP is the only name that can be specified for *osnam*.

(2) Version information *osver*

Specifies the version number of real-time OS.
Values that can be specified for *osver* are limited to V100 to V199.

B.3.2 Base clock cycle information

This defines the following item as the information related to timer interrupts required for realizing the time management functions (system time update, task timeout, cyclic handler activation, etc.) provided by the RI850MP.

- Base clock cycle tbase

The number of items that can be defined as base clock cycle information is defined as being within the range of 0 to 1. The following shows the basic clock cycle information format.

```
DEF_TIM(tbase);
```

(1) Base clock cycle *tbase*

Specifies the occurrence interval of base clock timer interrupts (in millisecond).
Values that can be specified for *tbase* are limited to 1 to 65535.

Remark If the definition of this item is omitted, the RI850MP handles the occurrence interval of base clock timer interrupts as follows.

```
DEF_TIM(1);
```

B.3.3 Timer interrupt information

Defines the following item as the information related to timer interrupts required for realizing the time management functions (system time update, task timeout, cyclic handler activation, etc.) provided by the RI850MP.

- Exception cause code tintno

Only one information item can be defined as timer interrupt information.
The following shows the timer interrupt information format.

```
CLK_INTNO(tintno);
```

(1) Exception cause code *tintno*

Specifies the exception cause code for a timer interrupt.
Values that can be specified for *tintno* are limited to 16-byte boundary value past 0x90 corresponding to timer interrupt, or exception cause code name.

Remarks 1. It is not possible to assign an exception cause code specified by Exception cause code inhno or Exception cause code excno.

2. Only exception cause code names prescribed in the device file can be specified in this item.

B.3.4 System stack information

Defines the following item as the system stack information to be allocated to each processor element by the RI850MP.

- Stack size sysstksz
- PE number peno

Only two information items can be defined as system stack information.
The following shows the system stack information format.

```
SYS_STK(sysstksz, peno);
```

(1) Stack size *sysstksz*

Specifies the system stack size (in bytes).
A value from 0x0 to 0x10000000 (aligned to a 4-byte boundary) can be specified for *sysstksz*.

(2) PE number *peno*

This specifies the PE number of the processor element that allocates the system stack.
Values that can be specified for *peno* are limited to 1 to 2.

Remark The system stack is allocated to the .kernel_stack_pe*peno* section.

B.3.5 Maximum priority information

This defines the following item as task-priority information.

- Maximum priority maxtpri

Only 0 - 1 items can be defined as maximum priority information.

The following shows the maximum priority information format.

```
MAX_PRI(maxtpri);
```

(1) Maximum priority *maxtpri*

Specifies the range of task priority specified with Initial priority itskpri, and chg_pri/ichg_pri.

Values that can be specified for *maxtpri* are limited to 1 to 32.

Remark If the definition of this item is omitted, the RI850MP handles the maximum priority as follows.

```
MAX_PRI(32);
```

B.3.6 Floating-point setting/status register information

The following item is defined as floating-point setting/status register (FPSR) information.

- Floating-point setting/status register informationfpsr

The number of items that can be defined as floating-point setting/status register information is defined as being within the range of 0 to 1.

The format for coding the floating-point setting/status register information is as follows:

```
DEF_FPSR(fpsr);
```

(1) Floating-point setting/status register information*fpsr*

This specifies the initial value of the floating-point setting/status register.

Note that the allowable range of the *fpsr* setting is limited to 0x0 to 0xffffffff.

Remark If the definition of this item is omitted, the RI850MP handles the initial value of the floating-point setting/status register as follows.

```
DEF_FPSR(0x20000);
```

B.3.7 Section information

The following items are defined as information relating to sections allocating buffer areas for the system stack, task stack, and data queue, and for the fixed-size memory pool.

- Section namesecnam
- Section size secsz

0 to 255 items can be specified as section information.
The description format of section information is as follows:

```
MEM_AREA(secnam, secsz);
```

(1) Section name*secnam*

This specifies the section name allocating buffer areas for the system stack, task stack, and data queue, and for the fixed-size memory pool.
Only section names specified in the link directive file can be specified as *secnam*.

(2) Section size *secsz*

This specifies the maximum section size (in bytes).
A value from 0x0 to 0x10000000 (aligned to a 4-byte boundary) or SIZE_AUTO can be specified for *secsz*.

Remark If SIZE_AUTO is specified in this item, the appropriate value will be calculated from the sizes specified in System stack information, Task information, Data queue information, and Fixed-sized memory pool information, and this value will be assumed to have been specified.

Remark If the definition of this item is omitted, the RI850MP handles the section size as follows.

```
MEM_AREA(.kernel_work_pe1, SIZE_AUTO);
MEM_AREA(.kernel_work_pe2, SIZE_AUTO);
```

B.3.8 Processor element information

The following items are defined as processor element (PE) information.

- Domain name domnam
- PE number peno

1 to 31 items can be specified as processor element information.

The description format of processor element information is as follows:

```
DOMAIN_ALLOCATION(domnam, peno);
```

(1) Domain name *domnam*

This specifies the domain name.

Only domain names specified in Domain name domnam can be specified in Domain name domnam.

(2) PE number *peno*

This specifies the PE number of the processor element that assigns the domain.

Values that can be specified for *peno* are limited to 1 to 2.

B.4 Domain Information

The following item is defined as domain information.

- Domain name domnam

1 to 31 items can be specified as domain information.

The description format of domain information is as follows:

```
DOMAIN (domnam) {
    // Static API Information
}
```

(1) Domain name *domnam*

This specifies the domain name.

Only names can be specified as *domnam*.

Remark See "B.5 Static API Information" for details about static API information.

B.5 Static API Information

The following are defined as static API information.

- Task information
- Semaphore information
- Eventflag information
- Data queue information
- Mailbox information
- Mutex information
- Fixed-sized memory pool information
- Cyclic handler information
- Interrupt handler information
- CPU exception handler information
- Initialization routine information
- Idle routine information

The following illustrates how static API information is written.

```
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz[:secnam], stk});  // Task information
CRE_SEM(semid, {sematr, isemcnt, maxsem});            // Semaphore information
CRE_FLG(flgid, {flgatr, iflgptn});                    // Eventflag information
CRE_DTQ(dtqid, {dtqatr, dtqcnt[:secnam], dtq});       // Data queue information
CRE_MBX(mbxid, {mbxatr, maxmpri, mprihd});            // Mailbox information
CRE_MTX(mtxid, {mtxatr, ceilpri});                    // Mutex information
CRE_MPF(mpfid, {mpfatr, blkcnt, blksz[:secnam], mpf});   // Fixed-sized memory pool
information
CRE_CYC(cycid, {cycatr, exinf, cychdr, cyctim, cycphs}); // Cyclic handler information
DEF_INH(inhno, {inhatr, inthdr});                     // Interrupt handler information
DEF_EXC(excno, {excatr, exchdr});                     // CPU exception handler information
ATT_INI({iniatr, exinf, inirtn});                     // Initialization routine information
VATT_IDL({idlatr, idlrtn});                           // Idle routine information
```

B.5.1 Task information

The following items are defined as task information.

- ID tskid
- Attribute tskatr
- Extended information exinf
- Startup address task
- Initial priority itskpri
- Stack size stksz
- Section name secnam
- System reserved area stk

1 to 1,023 tasks can be specified as task information.
The description format of task information is as follows:

```
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz[:secnam], stk});
```

(1) ID *tskid*

This specifies the task ID.
Only names can be specified as *tskid*.

Remark The correspondence between *tskid* and numerical values is output to the system information header file in the following format.

```
#define tskid   number
```

(2) Attribute *tskatr*

This specifies the task attributes (language in which task is coded, initial state, and initial interrupt state).
Only the following keywords can be specified as *tskatr*.

- Language in which task is coded
  TA_HLNG : C language
  TA_ASM : Assembly language
- Initial task state (optional)
  TA_ACT : READY state
- Initial task interrupt state (optional)
  TA_ENAINT : Interrupts enabled
  TA_DISINT : Interrupts disabled

Remarks 1. If the specification of TA_ACT is omitted, then the initial task state will be DORMANT state.
2. If the specification of TA_ENAINT and TA_DISINT are omitted, then the initial task interrupt state will be interrupts enabled.

(3) Extended information *exinf*

Specifies the extended information for the task.
The value of *exinf* must be between 0x0 and 0xffffffff, or a symbol name.

Remark Extended information is set as a parameter in the task when it transitions from DORMANT state to READY state.

(4) Startup address *task*

Specifies the startup address of the task.

The value of *task* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

(5) Initial priority *itskpri*

Specifies the initial priority level of the task.

The only values that can be specified as *itskpri* are 1 to Maximum priority maxtpri.

Remark In the RI850MP, tasks with lower priority numbers have higher priority.

(6) Stack size *stksz*

This specifies the task stack size (in bytes).

A value from 0x0 to 0x10000000 (aligned to a 4-byte boundary) can be specified for *stksz*.

(7) Section name *secnam*

This specifies the section name assigned to the task stack.

The only values that can be specified in *secnam* are section names specified in Section information.

Remark If the definition of this item is omitted, the RI850MP assumes that ".kernel_work_pe*peno*" has been specified.

The value of *peno* in ".kernel_stack_pe*peno*" is the PE number of the processor element assigned to the domain of the task information (PE number specified in Processor element information).

(8) System reserved area *stk*

This area is reserved by the system.

Only 0 or NULL can be specified as *stk*.

B.5.2 Semaphore information

The following items are defined as semaphore information.

- ID semid
- Attribute sematr
- Initial resource count isemcnt
- Maximum resource count maxsem

0 to 1,023 items can be specified as semaphore information.

The description format of semaphore information is as follows:

```
CRE_SEM(semid, {sematr, isemcnt, maxsem});
```

(1) ID *semid*

Specifies the semaphore ID.

Only names can be specified as *semid*.

Remark The correspondence between *semid* and numerical values is output to the system information header file in the following format.

```
#define semid   number
```

(2) Attribute *sematr*

This specifies the semaphore attribute (queuing method).

Only the following keywords can be specified as *sematr*.

- Task queuing method

  TA_TFIFO : Order in which resource acquisitions were requested

  TA_TPRI : Order of task priority

(3) Initial resource count *isemcnt*

Specifies the initial number of semaphore resources.

The only values that can be specifies as *isemcnt* are 0 to Maximum resource count maxsem.

(4) Maximum resource count *maxsem*

Specifies the maximum number of semaphore resources.

The only values that can be specified for *maxsem* are from 1 to 65535.

B.5.3 Eventflag information

The following items are defined as eventflag information.

- ID flgid
- Attribute flgatr
- Initial bit pattern iflgptn

0 to 1,023 items can be specified as eventflag information.
The description format of eventflag information is as follows:

```
CRE_FLG(flgid, {flgatr, iflgptn});
```

(1) ID *flgid*

Specifies the ID of the eventflag.
Only names can be specified as *flgid*.

Remark The correspondence between *flgid* and numerical values is output to the system information header file in the following format.

```
#define flgid   number
```

(2) Attribute *flgatr*

This specifies the eventflag attributes (queuing method, maximum number of tasks, and clear flag).
Only the following keywords can be specified as *flgatr*.

- Task queuing method
  - TA_TFIFO : Order in which determination of bit pattern was requested
  - TA_TPRI : Order of task priority
- Maximum number of tasks that can be queued
  - TA_WSGL : 1 task
  - TA_WMUL : Multiple tasks
- Clear bit pattern flag (optional)
  - TA_CLR : Bit pattern cleared if the request conditions are met

Remark If the specification of TA_CLR is omitted, then the bit pattern will not be cleared when the request conditions are met.

(3) Initial bit pattern *iflgptn*

This specifies the initial bit pattern of the eventflag.
Note that the allowable range of the *iflgptn* setting is limited to 0x0 to 0xffffffff.

B.5.4 Data queue information

The following items are defined as data queue information.

- ID dtqid
- Attribute dtqatr
- Maximum data count dtqcnt
- Section name secnam
- System reserved area dtq

0 to 1,023 items can be specified as data queue information.
Below is the format for coding data queue information.

```
CRE_DTQ(dtqid, {dtqatr, dtqcnt[:secnam], dtq});
```

(1) ID *dtqid*

This specifies the data queue ID.
Only names can be specified as *dtqid*.

Remark The correspondence between *dtqid* and numerical values is output to the system information header file in the following format.

```
#define dtqid   number
```

(2) Attribute *dtqatr*

This specifies the data queue attribute (queuing method).
Only the following keywords can be specified as *dtqatr*.

- Task queuing method

TA_TFIFO : Order in which data-send requests were made
TA_TPRI : Order of task priority

Remark If the data cannot be received immediately when the task makes a data reception request, the task is added to the data queue's wait queue, in the order that the data reception request was made.

(3) Maximum data count *dtqcnt*

This specifies the maximum number of data elements that can be written to the buffer area of the data queue.
Values that can be specified for *dtqcnt* are limited to 0 to 1023.

(4) Section name *secnam*

This specifies the section name assigned to the buffer area of the data queue.
The only values that can be specified in *secnam* are section names specified in Section information.

Remark If the definition of this item is omitted, the RI850MP assumes that ".kernel_work_pe*peno*" has been specified.
The value of *peno* in ".kernel_work_pe*peno*" is the PE number of the processor element assigned to the domain of the data queue information (PE number specified in Processor element information).

(5) System reserved area *dtq*

This area is reserved by the system.
Only 0 or NULL can be specified as *dtq*.

B.5.5 Mailbox information

The following items are defined as mailbox information.

- ID mbxid
- Attribute mbxatr
- Maximum priority maxmpri
- System reserved area mprihd

0 to 1,023 items can be specified as mailbox information.
Below is the format for coding the mailbox information.

```
CRE_MBX(mbxid, {mbxatr, maxmpri, mprihd});
```

(1) ID *mbxid*

This specifies the ID of the mailbox.
Only names can be specified as *mbxid*.

Remark The correspondence between *mbxid* and numerical values is output to the system information header file in the following format.

```
#define mbxid   number
```

(2) Attribute *mbxatr*

This specifies the mailbox attribute (queuing method).
Only the following keywords can be specified as *mbxatr*.

- Task queuing method
  - TA_TFIFO : Order in which message-receipt requests were made
  - TA_TPRI : Order of task priority
- Message queuing method
  - TA_MFIFO : Order in which message-send requests were made
  - TA_MPRI : Order of message priority

(3) Maximum priority *maxmpri*

This specifies the maximum priority of messages that can be sent to the mailbox.
Values that can be specified for *maxmpri* are limited to 1 to 32,767.

Remark In the RI850MP, messages with lower priority numbers have higher priority.

(4) System reserved area *mprihd*

This area is reserved by the system.
Only 0 or NULL can be specified as *mprihd*.

B.5.6 Mutex information

The following items are defined as mutex information.

- ID mtxid
- Attribute mtxatr
- System reserved area ceilpri

0 to 1,023 items can be specified as mutex information.
Below is the format for coding the mutex information.

```
CRE_MTX(mtxid, {mtxatr, ceilpri});
```

(1) ID *mtxid*

This specifies the ID of the mutex.
Only names can be specified as *mtxid*.

Remark The correspondence between *mtxid* and numerical values is output to the system information header file in the following format.

```
#define mtxid   number
```

(2) Attribute *mtxatr*

This specifies the mutex attribute (queuing method).
Only the following keywords can be specified as *mtxatr*.

- Task queuing method

TA_TFIFO : Order in which mutex acquisition was requested
TA_TPRI : Order of task priority

(3) System reserved area *ceilpri*

This area is reserved by the system.
Only 0 or NULL can be specified as *ceilpri*.

B.5.7 Fixed-sized memory pool information

The following items are defined as fixed-size memory pool information.

- ID mpfid
- Attribute mpfatr
- Block count blkcnt
- Block size blksz
- Section name secnam
- System reserved area mpf

0 to 1023 resources can be specified as fixed-size memory pool information.
Below is the format for coding fixed-sized memory pool information.

```
CRE_MPF(mpfid, {mpfatr, blkcnt, blksz[:secnam], mpf});
```

(1) ID *mpfid*

This specifies the ID of the fixed-size memory pool.
Only names can be specified as *mpfid*.

Remark The correspondence between *mpfid* and numerical values is output to the system information header file in the following format.

```
#define mpfid   number
```

(2) Attribute *mpfatr*

This specifies the fixed-sized memory pool attribute (queuing method).
Only the following keywords can be specified as *mpfatr*.

- Task queuing method

TA_TFIFO : Order in which fixed-size memory block acquisition was requested
TA_TPRI : Order of task priority

(3) Block count *blkcnt*

This specifies the total count of the fixed-size memory blocks.
Values that can be specified for *blkcnt* are limited to 1 to 32,767.

(4) Block size *blksz*

This specifies the size per block (in bytes).
A value from 0x1 to 0x100000000 (aligned to a 4-byte boundary) can be specified for *blksz*.

(5) Section name *secnam*

This specifies the section name assigned to the fixed-size memory pool.
The only values that can be specified in *secnam* are section names specified in Section information.

Remark If the definition of this item is omitted, the RI850MP assumes that ".kernel_work_pe*peno*" has been specified.
The value of *peno* in ".kernel_work_pe*peno*" is the PE number of the processor element assigned to the domain of the fixed-size memory pool information (PE number specified in Processor element information).

(6) System reserved area *mpf*

This area is reserved by the system.

Only 0 or NULL can be specified as *mpf*.

B.5.8 Cyclic handler information

The following items are defined as cyclic handler information.

- ID cycid
- Attribute cycatr
- Extended information exinf
- Startup address cychdr
- Cycle start cyctim
- Activation phase cycphs

0 to 1,023 items can be defined as cyclic handler information.
The format for coding cyclic handler information is shown below.

```
CRE_CYC(cycid, {cycatr, exinf, cychdr, cyctim, cycphs});
```

(1) ID *cycid*

This specifies the ID of the cyclic handler.
Only names can be specified as *cycid*.

Remark The correspondence between *cycid* and numerical values is output to the system information header file in the following format.

```
#define cycid   number
```

(2) Attribute *cycatr*

This specifies the cyclic handler attributes (language used, initial state, and storage flag).
Only the following keywords can be specified as *cycatr*.

- Language in which cyclic handler is coded
  TA_HLNG : C language
  TA_ASM : Assembly language
- Initial state of cyclic handler (optional)
  TA_STA : Operating state
- Whether activation phase has been stored (optional)
  TA_PHS : Activation phase stored

Remarks 1. If the specification of TA_STA is omitted, then the initial cyclic handler state will be non-operational state.

2. If the specification of TA_PHS is omitted, then the activation phase storage flag will be "Do not store activation phase".

(3) Extended information *exinf*

This specifies extended information for the cyclic handler.
The value of *exinf* must be between 0x0 and 0xffffffff, or a symbol name.

Remark When a cyclic handler transitions from non-operational state to operating state, the extended information is set in the cyclic handler as a parameter.

(4) Startup address *cychdr*

This specifies the start address of the cyclic handler.
The value of *cychdr* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

(5) Cycle start *cyctim*

This specifies the startup interval of the cyclic handler (in milliseconds).

Only 0x1 to 0x7FFFFFFF can be specified as *cyctim*.

(6) Activation phase *cycphs*

This specifies the initial startup phase of the cyclic handler (in milliseconds).

Only 0x1 to 0x7FFFFFFF can be specified as *cycphs*.

B.5.9 Interrupt handler information

The following items are defined as interrupt handler information.

- Exception cause code inhno
- Attribute inhatr
- Startup address inthdr

0 to 1 items per PE can be defined as interrupt handler information for each exception cause.
The format for coding interrupt handler information is shown below.

```
DEF_INH(inhno, {inhatr, inthdr});
```

(1) Exception cause code *inhno*

This specifies the exception cause code corresponding to the interrupt that triggers the activation of the interrupt handler.
Values that can be specified for *inhno* are limited to 16-byte boundary value past 0x90 corresponding to the interrupt, or exception cause name.

Remarks 1. It is not possible to assign an exception cause code specified by Exception cause code tintno or Exception cause code excno.

2. Only exception cause code names prescribed in the device file can be specified in this item.

(2) Attribute *inhatr*

This specifies the interrupt handler's attribute (language it is coded in).
Only the following keywords can be specified as *inhatr*.

- Interrupt handler language

TA_HLNG : C language
TA_ASM : Assembly language

(3) Startup address *inthdr*

Specifies the start address for an interrupt handler.
The value of *inthdr* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

B.5.10 CPU exception handler information

The following items are defined as CPU exception handler information.

- Exception cause code excno
- Attribute excatr
- Startup address exchdr

0 to 1 items per PE can be defined as CPU exception handler information for each exception cause.
The format for coding CPU exception handler information is shown below.

```
DEF_EXC(excno, {excatr, exchdr});
```

(1) Exception cause code *excno*

This specifies the exception cause code corresponding to the CPU exception (EI level software exception or floating-point operation) that triggered the activation of the CPU exception handler.
Values that can be specified for *excno* are limited to a value aligned on a 16-byte boundary corresponding to the CPU exception, or exception cause name.

Remarks 1. It is not possible to assign an exception cause code specified by Exception cause code tintno or Exception cause code inhno.

2. Only exception cause code names prescribed in the device file can be specified in this item.

(2) Attribute *excatr*

This specifies the CPU exception handler's attribute (language it is coded in).
Only the following keywords can be specified as *excatr*.

- Language in which CPU exception handler is coded

  TA_HLNG : C language
  TA_ASM : Assembly language

(3) Startup address *exchdr*

Specifies the start address of the CPU exception handler.
The value of *exchdr* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

B.5.11 Initialization routine information

The following items are defined as initialization routine information.

- Attribute iniatr
- Extended information exinf
- Startup address inirtn

0 to 1,023 items can be defined as initialization routine information.
The format for coding initialization routine information is as follows:

```
ATT_INI({iniatr, exinf, inirtn});
```

(1) Attribute *iniatr*

This specifies the attribute of the initialization routine (language it is coded in).
Only the following keywords can be specified as *iniatr*.

- Language in which initialization routine is coded

  TA_HLNG : C language
  TA_ASM : Assembly language

(2) Extended information *exinf*

This specifies extended information for the initialization routine.
The value of *exinf* must be between 0x0 and 0xffffffff, or a symbol name.

Remark When the initialization routine is called from Kernel Initialization Module, the extended information is set in the initialization routine as a parameter.

(3) Startup address *inirtn*

Specifies the start address for an initialization routine.
The value of *inirtn* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

B.5.12 Idle routine information

The following items are defined as idle routine information.

- Attribute idlatr
- Startup address idlrtn

0 to 1 items per PE can be defined as idle routine information.
The format for coding idle routine information is as follows:

```
VATT_IDL({idlatr, idlrtn});
```

(1) Attribute *idlatr*

This specifies the attribute of the idle routine (language it is coded in).
Only the following keywords can be specified as *idlatr*.

- Language in which idle routine is coded

TA_HLNG : C language
TA_ASM : Assembly language

(2) Startup address *idlrtn*

This specifies the start address of the idle routine.
The value of *idlrtn* must be between 0x0 and 0xFFFFFFFE (aligned to a 2-byte boundary), or a symbol name.

Remark If the definition of this item is omitted, the RI850MP handles the start address of the idle routine as follows.

```
VATT_IDL(TA_HLNG, _default_idlrtn);
```

B.6 SCT Information

The following item is defined as information relating to flags for using service calls provided by the RI850MP.

- Service call name svc_nam

0 to 69 items can be defined as SCT information.

The format for coding SCT information is as follows:

```
DEF_SCT(svc_nam);
```

(1) Service call name *svc_nam*

This specifies the name of the service call to use in the processing program.

Only the following keywords can be specified as *svc_nam*.

- Task management functions
  act_tsk, iact_tsk, can_act, ican_act, ext_tsk, ter_tsk, chg_pri, ichg_pri, get_pri, iget_pri, ref_tsk, iref_tsk
- Task dependent synchronization functions
  slp_tsk, tslp_tsk, wup_tsk, iwup_tsk, can_wup, ican_wup, rel_wai, irel_wai, sus_tsk, isus_tsk, rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk, dly_tsk
- Synchronization and communication functions (semaphores)
  sig_sem, isig_sem, wai_sem, pol_sem, ipol_sem, twai_sem, ref_sem, iref_sem
- Synchronization and communication functions (eventflags)
  set_flg, iset_flg, clr_flg, iclr_flg, wai_flg, pol_flg, ipol_flg, twai_flg, ref_flg, iref_flg
- Synchronization and communication functions (data queues)
  snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq, rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq, ref_dtq, iref_dtq
- Synchronization and communication functions (mailboxes)
  snd_mbx, isnd_mbx, rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx, ref_mbx, iref_mbx
- Extended synchronization and communication functions
  loc_mtx, ploc_mtx, tloc_mtx, unl_mtx, ref_mtx, iref_mtx
- Memory pool management functions
  get_mpf, pget_mpf, ipget_mpf, tget_mpf, rel_mpf, irel_mpf, ref_mpf, iref_mpf
- Time management functions
  set_tim, iset_tim, get_tim, iget_tim, sta_cyc, ista_cyc, stp_cyc, istp_cyc, ref_cyc, iref_cyc
- System state management functions
  rot_rdq, irot_rdq, get_tid, iget_tid, loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, dis_dsp, ena_dsp, sns_ctx, sns_loc, sns_dsp, sns_dpn
- Interrupt management functions
  dis_int, ena_int, chg_ipm, ichg_ipm, get_ipm, iget_ipm

APPENDIX C INDEX

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Apr 01, 2011	-	First Edition issued

RI850MP
User's Manual: Coding

Publication Date: Rev.1.00 Apr 01, 2011

Published by: Renesas Electronics Corporation

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: +49-211-65030, 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
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

RI850MP

Renesas Electronics Corporation

R20UT0519EJ0100