To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

User's Manual

M3T-MR308/4 V.4.00

Reference Manual

Real-time OS for M16C/70,80,M32C/80 Series

Renesas Electronics
www.renesas.com

Rev.2.00 2005.11

Keep safety first in your circuit designs!

- Renesas Technology Corporation and Renesas Solutions Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation, Renesas Solutions Corporation or a third party.
- Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor for the latest product information before purchasing a product listed herein. The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors. Please also pay attention to information published by Renesas Technology Corporation and Renesas Solutions Corporation by various means, including the Renesas home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

\SUPPORT\Product-name\SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools

Preface

M3T-MR308 (hereafter abbreviated MR308) is the realtime operating system[1] for the M16C/70, 80 and M32C/80 series of Renesas microcomputers. MR308 is compliant with µITRON Specification. [2]
This manual describes how to create a program that uses MR308 and the precautions to be observed when creating a program. For details on how to use each service call of MR308, refer to the "MR308 Reference Manual."

Things to be prepared before MR308 can be used

Before a program can be created that uses MR308, the following products available from Renesas need to be purchased separately.

- C-compiler package M3T-NC308WA(abbreviated as NC308) for M16C/70,80 M32C/80 series microcomputers

Document List

The following sets of documents are supplied with the MR308.

- Release Note

  Presents a software overview and describes the corrections to the Users Manual and Reference Manual.

- Users Manual (PDF file)

  Describes the procedures and precautions to observe when using the MR308 for programming purposes.

- Reference Manual (PDF file)

  Describes the MR308 service call procedures and typical usage examples. Before reading the Users Manual, be sure to read the Release Note.
  Please read the release note before reading this manual.

Right of Software Use

The right of software use conforms to the software license agreement. You can use the MR308 for your product development purposes only, and are not allowed to use it for the other purposes. You should also note that this manual does not guarantee or permit the exercise of the right of software use.

---

[1] Hereinafter abbreviated "real-time OS"

[2] The µITRON Specification is originated by Dr.Ken Sakamura and his laboratory members at the Faculty Science of University of Tokyo. Therefore,Dr.Ken Sakamura holds the copyright on the µITRON Specification. By his consent,the MR308 is produced in compliance with the µITRON Specification.

Contents

Chapter 1 Service Call Reference

1.1. Task Management Function

Specifications of the task management function of MR308 are listed in Table 1. Specifications of the Task Management Function below. The task description languages in item No. 4 are those specified in the GUI configurator. They are not output to a configuration file, nor are the MR308 kernel concerned with them.

The task stack permits a section name to be specified for each task individually.

Item No.	Item	Content
1	Task ID	1-255
2	Task Priority	1-255
3	Maximum number of queued task startup requests	255
4	Task attribute	TA_HLNG: Tasks written in high-level language TA_ASM: Tasks written in assembly language TA_ACT: Startup attribute
5	Task stack	Sections specifiable

Table 1. Specifications of the Task Management Function

act_tsk　　Activate task
iact_tsk　　Activate task (handler only)

[[C Language API]]

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

● Parameters

ID	tskid	ID number of the task to be started

● Return parameters

ER	ercd	Terminated normally (E_OK) or error code

■ Assembly language API

```
.include mr308.inc
act_tsk   TSKID
iact_tsk TSKID
```

● Parameters

TSKID	ID number of the task to be started

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	Task ID

[[Error Code]]

E_QOVR	Queuing overflow

[[Functional description]]

This service call starts the task indicated by tskid. The started task goes from DORMANT state to READY state or RUNNING state.

The following lists the processing performed on startup.

1. Initializes the current priority of the task.
2. Clears the number of queued wakeup requests.
3. Clears the number of suspension requests.

Specifying tskid=TSK_SELF(0) specifies the issuing task itself. The task has passed to it as parameter the extended information of it that was specified when the task was created. If TSK_SELF is specified for tskid in non-task context, operation of this service call cannot be guaranteed.

If the target task is not in DORMANT state, a task activation request by this service call is enqueued. In other words, the activation request count is incremented by 1. The maximum value of the task activation request is 255. If this limit is exceeded, the error code E_QOVR is returned.

If TSK_SELF is specified for tskid, the issuing task itself is made the target task.

If this service call is to be issued from task context, use act_tsk; if issued from non-task context, use iact_tsk.

[[Example program statement]]

<<Example statement in C language>>

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1( VP_INT stacd )
{
   ER ercd;
        :
     ercd = act_tsk( ID_task2 );
        :
}
void task2( VP_INT stacd )
{
        :
     ext_tsk();
}
```

<<Example statement in assembly language>>

```
     .INCLUDE    mr308.inc
     .GLB        task
task:
                 :
     pushm               A0
     act_tsk             #ID_TASK3
                 :
```

can_act　　Cancel task activation request
ican_act　　Cancel task activation request (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the task to cancel

- **Return Parameters**

ER_UINT	actcnt > 0	Canceled activation request count
	actcnt <0	Error code

[[Assembly language API]]

```
.include mr308.inc
can_act  TSKID
ican_act TSKID
```

- **Parameters**

TSKID	ID number of the task to cancel

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Canceled startup request count or error code
A0	ID number of the target task

[[Error code]]

None

[[Functional description]]

This service call finds the number of task activation requests enqueued for the task indicated by tskid, returns the result as a return parameter, and at the same time invalidates all of the task's activation requests.

Specifying tskid=TSK_SELF(0) specifies the issuing task itself. If TSK_SELF is specified for tskid in non-task context, operation of this service call cannot be guaranteed.

This service call can be invoked for a task in DORMANT state as the target task. In that case, the return parameter is 0.

If this service call is to be issued from task context, use can_act; if issued from non-task context, use ican_act.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1()
  {
    ER_UINT actcnt;
        :
    actcnt = can_act( ID_task2 );
        :
  }
  void task2()
  {
        :
    ext_tsk();
  }
```

<<Example statement in assembly language>>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0
    can_act     #ID_TASK2
                :
```

sta_tsk　　Activate task with a start code
ista_tsk　　Activate task with a start code (handler only)

[[C Language API]]

```
ER ercd = sta_tsk( ID tskid,VP_INT stacd );
ER ercd = ista_tsk ( ID tskid,VP_INT stacd );
```

- **Parameters**

ID	tskid	ID number of the target task
VP_INT	stacd	Task start code

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
sta_tsk  TSKID,STACD
ista_tsk TSKID,STACD
```

- **Parameters**

TSKID	ID number of the target task
STATCD	Task start code

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R1	Task start code (16 low-order bits)
R3	Task start code (16 high-order bits)
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid (task indicated by tskid is not DOMANT state)

[[Functional description]]

This service call starts the task indicated by tskid. In other words, it places the specified task from DORMANT state into READY state or RUNNING state. This service call does not enqueue task activation requests. Therefore, if a task activation request is issued while the target task is not DORMANT state, the error code E_OBJ is returned to the service call issuing task. This service call is effective only when the specified task is in DORMANT state. The task start code stacd is 32 bits long. This task start code is passed as parameter to the activated task.

If a task is restarted that was once terminated by ter_tsk or ext_tsk, the task performs the following as it starts up.

1. Initializes the current priority of the task.
2. Clears the number of queued wakeup requests.
3. Clears the number of nested forcible wait requests.

If this service call is to be issued from task context, use sta_tsk; if issued from non-task context, use ista_tsk.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     ER ercd;
     VP_INT stacd = 0;
     ercd = sta_tsk( ID_task2, stacd );
           :
  }
  void task2(VP_INT msg)
  {
     if(msg == 0)
           :
  }
```

<<Example statement in assembly language>>

```
     .INCLUDE    mr308.inc
     .GLB        task
task:
                 :
     PUSHM       A0,R1,R3
     sta_tsk     #ID_TASK4,#012345678H
                 :
```

ext_tsk Terminates invoking task

[[C Language API]]

```
ER ercd = ext_tsk();
```

- **Parameters**

  None

- **Return Parameters**

  Not return from this service call

[[Assembly language API]]

```
.include mr308.inc
ext_tsk
```

- **Parameters**

  None

- **Register contents after service call is issued**

  Not return from this service call

[[Error code]]

Not return from this service call

[[Functional description]]

This service call terminates the invoking task. In other words, it places the issuing task from RUNNING state into DORMANT state. However, if the activation request count for the issuing task is 1, the activation request count is decremented by 1, and processing similar to that of act_tsk or iact_tsk is performed. In that case, the task is placed from DORMANT state into READY state. The task has its extended information passed to it as parameter when the task starts up.

This service call is designed to be issued automatically at return from a task.

In the invocation of this service call, the resources the issuing task had acquired previously (e.g., semaphore) are not released.

This service call can only be used in task context. This service call can be used even in a CPU locked state, but cannot be used in non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task(void)
  {
       :
    ext_tsk();
  }
```

<<Example statement in assembly language>>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    ext_tsk
```

ter_tsk Terminate task

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the forcibly terminated task

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
ter_tsk  TSKID
```

- **Parameters**

TSKID	ID number of the forcibly terminated task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)
E_ILUSE	Service call improperly used task indicated by tskid is the issuing task itself)

[[Functional description]]

This service call terminates the task indicated by tskid. If the activation request count of the target task is equal to or greater than 1, the activation request count is decremented by 1, and processing similar to that of act_tsk or iact_tsk is performed. In that case, the task is placed from DORMANT state into READY state. The task has its extended information passed to it as parameter when the task starts up.

If a task specifies its own task ID or TSK_SELF, an E_ILUSE error is returned.

If the specified task was placed into WAITING state and has been enqueued in some waiting queue, the task is dequeued from it by execution of this service call. However, the semaphore and other resources the specified task had acquired previously are not released.

If the task indicated by tskid is in DORMANT state, it returns the error code E_OBJ as a return value for the service call.

This service call can only be used in task context, and cannot be used in non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
          :
      ter_tsk( ID_main );
          :
  }
```

<<Example statement in assembly language>>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0
    ter_tsk     #ID_TASK3
                :
```

chg_pri　　Change task priority
ichg_pri　　Change task priority(handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task
PRI	tskpri	Priority of the target task

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
chg_pri  TSKID,TSKPRI
ichg_pri  TSKID,TSKPRI
```

- **Parameters**

TSKID	ID number of the target task
TSKPRI	Priority of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R3	Priority of the target task
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)

[[Functional description]]

This service call changes the priority of the task indicated by tskid to the value indicated by tskpri, and performs rescheduling based on the result of that priority change. Therefore, if this service call is executed on a task enqueued in a ready queue (including one that is in an executing state) or a task in a waiting queue in which tasks are enqueued in order of priority, the target task is moved to behind the tail of a relevant priority part of the queue. Even when the same priority as the previous one is specified, the task is moved to behind the tail of the queue.

The smaller the number, the higher the task priority, with 1 assigned the highest priority. The minimum value specifiable as priority is 1. The specifiable maximum value of priority is the maximum value of priority specified in a configuration file, providing that it is within the range 1 to 255. For example, if system specification in a configuration file is as follows,

```
system{       stack_size    = 0x100;
              priority      = 13;
};
```

then priority can be specified in the range 1 to 13.

If TSK_SELF is specified, the priority of the issuing task is changed. If TSK_SELF is specified for tskid in non-task context, operation of the service call cannot be guaranteed. If TPRI_INI is specified, the task has its priority changed to the initial priority that was specified when the task was created. The changed task priority remains effective until the task is terminated or this service call is executed again.

If the task indicated by tskid is in DORMANT state, it returns the error code E_OBJ as a return value for the service call. Since the M3T-MR308 does not support the mutex function, in no case will the error code E_ILUSE be returned.

If this service call is to be issued from task context, use chg_pri; if issued from non-task context, use ichg_pri.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
          :
       chg_pri( ID_task2, 2 );
          :
  }
```

<<Example statement in assembly language>>

```
     .INCLUDE    mr308.inc
     .GLB        task
task:
                 :
     pushm       A0,R3
     chg_pri     #ID_TASK3,#1
                 :
```

get_pri Reference task priority
iget_pri Reference task priority(handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task
PRI	*p_tskpri	Pointer to the area to which task priority is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
get_pri  TSKID
iget_pri  TSKID
```

- **Parameters**

TSKID	ID number of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Acquired task priority

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)

[[Functional description]]

This service call returns the priority of the task indicated by tskid to the area indicated by p-tskpri. If TSK_SELF is specified, the priority of the issuing task itself is acquired. If TSK_SELF is specified for tskid in non-task context, operation of the service call cannot be guaranteed.
If the task indicated by tskid is in DORMANT state, it returns the error code E_OBJ as a return value for the service call.
If this service call is to be issued from task context, use get_pri; if issued from non-task context, use iget_pri.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     PRI p_tskpri;
     ER ercd;
           :
     ercd = get_pri( ID_task2, &p_tskpri );
           :
  }
```

<<Example statement in assembly language>>

```
     .INCLUDE    mr308.inc
     .GLB        task
task:
                 :
     PUSHM       A0
     get_pri     #ID_TASK2
                 :
```

ref_tsk Reference task status
iref_tsk Reference task status (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task
T_RTSK	*pk_rtsk	Pointer to the packet to which task status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

Contents of pk_rtsk

```
typedef   struct t_rtsk{
```

STAT	tskstat	+0	2	Task status
PRI	tskpri	+2	2	Current priority of task
PRI	tskbpri	+4	2	Base priority of task
STAT	tskwait	+6	2	Cause of wait
ID	wobjid	+8	2	Waiting object ID
TMO	lefttmo	+10	4	Left time before timeout
UINT	actcnt	+14	2	Number of queued activation request counts
UINT	wupcnt	+16	2	Number of queued wakeup request counts
UINT	suscnt	+18	2	Number of nested suspension request counts

```
} T_RTSK;
```

[[Assembly language API]]

```
.include mr308.inc
ref_tsk  TSKID, PK_RTSK
iref_tsk  TSKID, PK_RTSK
```

- **Parameters**

TSKID	ID number of the target task
PK_RTSK	Pointer to the packet to which task status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task
A1	Pointer to the packet to which task status is returned

[[Error code]]

None

[[Functional description]]

This service call inspects the status of the task indicated by tskid and returns the current information on that task to the area pointed to by pk_rtsk as a return parameter. If TSK_SELF is specified, the status of the issuing task itself is inspected. If TSK_SELF is specified for tskid in non-task context, operation of the service call cannot be guaranteed.

◆ **tskstat (task status)**

tskstat has one of the following values returned to it depending on the status of the specified task.

- TTS_RUN(0x0001) RUNNING state
- TTS_RDY(0x0002) READY state
- TTS_WAI(0x0004) WAITING state
- TTS_SUS(0x0008) SUSPENDED state
- TTS_WAS(0x000C) WAITING-SUSPENDED state
- TTS_DMT(0x0010) DORMANT state

◆ **tskpri (current priority of task)**

tskpri has the current priority of the specified task returned to it. If the task is in DOMANT state, tskpri is indeterminate.

◆ **tskbpri (base priority of task)**

tskbpri has the base priority of the specified task returned to it. Since the M3T-MR308 does not support the mutex function, tskpri and tskbpri assume the same value. If the task is in DOMANT state, tskbpri is indeterminate.

◆ **tskwait (cause of wait)**

If the target task is in a wait state, one of the following causes of wait is returned. The values of the respective causes of wait are listed below. If the task status is other than a wait state (TTS_WAI or TTS_WAS), tskwait is indeterminate.

- TTW_SLP (0x0001) Kept waiting by slp_tsk or tslp_tsk
- TTW_DLY (0x0002) Kept waiting by dly_tsk
- TTW_SEM (0x0004) Kept waiting by wai_sem or twai_sem
- TTW_FLG (0x0008) Kept waiting by wai_flg or twai_flg
- TTW_SDTQ(0x0010) Kept waiting by snd_dtq or tsnd_dtq
- TTW_RDTQ(0x0020) Kept waiting by rcv_dtq or trcv_dtq
- TTW_MBX (0x0040) Kept waiting by rcv_mbx or trcv_mbx
- TTW_MPF (0x2000) Kept waiting by get_mpf or tget_mpf
- TTW_VSDTQ (0x4000) Kept waiting by vsnd_dtq or vtsnd_dtq[3]
- TTW_VRDTQ(0x8000) Kept waiting by vrcv_dtq or vtrcv_dtq

◆ **wobjid (waiting object ID)**

If the target task is in a wait state (TTS_WAI or TTS_WAS), the ID of the waiting target object is returned. Otherwise, wobjid is indeterminate.

◆ **lefttmo(left time before timeout)**

If the target task has been placed in WAITING state (TTS_WAI or TTS_WAS) by other than dly_tsk, the left time before it times out is returned. If the task is kept waiting perpetually, TMO_FEVR is returned. Otherwise, lefttmo is indeterminate.

◆ **actcnt(task activation request)**

The number of currently queued task activation request is returned.

◆ **wupcnt (wakeup request count)**

The number of currently queued wakeup requests is returned. If the task is in DORMANT state, wupcnt is indeterminate.

◆ **suscnt (suspension request count)**

The number of currently nested suspension requests is returned. If the task is in DORMANT state, suscnt is indeterminate.

If this service call is to be issued from task context, use ref_tsk; if issued from non-task context, use iref_tsk.

[3] TTW_VSDTQ and TTW_VRDTQ are the causes of wait outside the scope of μITRON 4.0 Specification.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RTSK rtsk;
       ER ercd;
            :
       ercd = ref_tsk( ID_main, &rtsk );
            :
  }
```

<<Example statement in assembly language>>

```
_refdata:       .blkb   20
     .include mr308.inc
     .GLB       task
task:
            :
     PUSHM      A0,A1
     ref_tsk    #TSK_SELF,#_refdata
            :
```

ref_tst Reference task status (simplified version)
iref_tst Reference task status (simplified version, handler only)

[[C Language API]]

```
ER ercd = ref_tst( ID tskid, T_RTST *pk_rtst );
ER ercd = iref_tst( ID tskid, T_RTST *pk_rtst );
```

- **Parameters**

ID	tskid	ID number of the target task
T_RTST	*pk_rtst	Pointer to the packet to which task status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

```
Contents of pk_rtsk
typedef   struct t_rtst{
    STAT    tskstat     +0    2     Task status
    STAT    tskwait     +2    2     Cause of wait
} T_RTST;
```

[[Assembly language API]]

```
.include mr308.inc
ref_tst  TSKID, PK_RTST
iref_tst  TSKID, PK_RTST
```

- **Parameters**

TSKID	ID number of the target task
PK_RTST	Pointer to the packet to which task status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task
A1	Pointer to the packet to which task status is returned

[[Error code]]

None

[[Functional description]]

This service call inspects the status of the task indicated by tskid and returns the current information on that task to the area pointed to by pk_rtst as a return value. If TSK_SELF is specified, the status of the issuing task itself is inspected. If TSK_SELF is specified for tskid in non-task context, operation of the service call cannot be guaranteed.

◆ **tskstat (task status)**

tskstat has one of the following values returned to it depending on the status of the specified task.

- TTS_RUN(0x0001) RUNNING state
- TTS_RDY(0x0002) READY state
- TTS_WAI(0x0004) WAITING state
- TTS_SUS(0x0008) SUSPENDED state
- TTS_WAS(0x000C) WAITING-SUSPENDED state
- TTS_DMT(0x0010) DORMANT state

◆ **tskwait (cause of wait)**

If the target task is in a wait state, one of the following causes of wait is returned. The values of the respective causes of wait are listed below. If the task status is other than a wait state (TTS_WAI or TTS_WAS), tskwait is indeterminate.

- TTW_SLP (0x0001) Kept waiting by slp_tsk or tslp_tsk
- TTW_DLY (0x0002) Kept waiting by dly_tsk
- TTW_SEM (0x0004) Kept waiting by wai_sem or twai_sem
- TTW_FLG (0x0008) Kept waiting by wai_flg or twai_flg
- TTW_SDTQ(0x0010) Kept waiting by snd_dtq or tsnd_dtq
- TTW_RDTQ(0x0020) Kept waiting by rcv_dtq or trcv_dtq
- TTW_MBX (0x0040) Kept waiting by rcv_mbx or trcv_mbx
- TTW_MPF (0x2000) Kept waiting by get_mpf or tget_mpf
- TTW_VSDTQ (0x4000) Kept waiting by vsnd_dtq or vtsnd_dtq[4]
- TTW_VRDTQ(0x8000) Kept waiting by vrcv_dtq or vtrcv_dtq

If this service call is to be issued from task context, use ref_tst; if issued from non-task context, use iref_tst.

[4] TTW_VSDTQ and TTW_VRDTQ are the causes of wait outside the scope of μITRON 4.0 Specification.

[[Example program statement]]

<<Example statement in C language>>

```
    #include <itron.h>
    #include <kernel.h>
    #include "kernel_id.h"
    void task()
    {
        T_RTST rtst;
        ER ercd;
                :
        ercd = ref_tst( ID_main, &rtst );
                :
    }
```

<<Example statement in assembly language>>

```
_refdata:       .blkb   4
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,A1
    ref_tst     #ID_TASK2,#_refdata
          :
```

1.2. Task Dependent Synchronization Function

Specifications of the task-dependent synchronization function are listed in Table 2 below.

Item No.	Item	Content
1	Maximum value of task wakeup request count	255
2	Maximum number of nested forcible task wait requests count	1

Table 2 . Specifications of the Task Dependent Synchronization Function

slp_tsk　　Put task to sleep
tslp_tsk　　Put task to sleep (with timeout)

[[C Language API]]

```
ER ercd = slp_tsk();
ER ercd = tslp_tsk( TMO tmout );
```

- **Parameters**
  - slp_tsk
    None
  - tslp_tsk

TMO	tmout	Timeout value

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
slp_tsk
tslp_tsk TMO
```

- **Parameters**

TMO	Timeout value

- **Register contents after service call is issued**

tslp_tsk

Register name	Content after service call is issued
R0	Error code
R1	Timeout value (16 low-order bits)
R3	Timeout value (16 high-order bits)

slp_tsk

Register name	Content after service call is issued
R0	Error code

[[Error code]]

E_TMOUT	Timeout value
E_RLWAI	Forced release from waiting

[[Functional description]]

This service call places the issuing task itself from RUNNING state into sleeping wait state. The task placed into WAITING state by execution of this service call is released from the wait state in the following cases:

◆ **When a task wakeup service call is issued from another task or an interrupt**
The error code returned in this case is E_OK.

◆ **When a forcible awaking service call is issued from another task or an interrupt**
The error code returned in this case is E_RLWAI.

◆ **When the first time tick occurred after tmout elapsed (for tslp_tsk)**
The error code returned in this case is E_TMOUT.

If the task receives sus_tsk issued from another task while it has been placed into WAITING state by this service call, it goes to WAITING-SUSPENDED state. In this case, even when the task is released from WAITING state by a task wakeup service call, it still remains in SUSPENDED state, and its execution cannot be resumed until rsm_tsk is issued.

The service call tslp_tsk may be used to place the issuing task into sleeping state for a given length of time by specifying tmout in a parameter to it. The parameter tmout is expressed in ms units. For example, if this service call is written as tslp_tsk(10);, then the issuing task is placed from RUNNING state into WAITING state for a period of 10 ms. If specified as tmout =TMO_FEVR(–1), the task will be kept waiting perpetually, with the service call operating the same way as slp_tsk.

The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, operation of the service call cannot be guaranteed.

This service call can only be issued from task context, and cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
       if( slp_tsk() != E_OK )
           error("Forced wakeup\n");
                :
       if( tslp_tsk( 10 ) == E_TMOUT )
           error("time out\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     slp_tsk
           :
     PUSHM       R1,R3
     tslp_tsk    #TMO_FEVR
           :
     PUSHM       R1,R3
     tslp_tsk    #100
           :
```

wup_tsk Wakeup task
iwup_tsk Wakeup task (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
wup_tsk  TSKID
iwup_tsk  TSKID
```

- **Parameters**

TSKID	ID number of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)
E_QOVR	Queuing overflow

[[Functional description]]

If the task specified by tskid has been placed into WAITING state by slp_tsk or tslp_tsk, this service call wakes up the task from the sleeping state to place it into READY or RUNNING state. Or if the task specified by tskid is in WAITING-SUSPENDED state, this service call awakes the task from only the sleeping state so that the task goes to SUSPEND state.

If a wakeup request is issued while the target task remains in DORMANT state, the error code E_OBJ is returned to the service call issuing task. If TSK_SELF is specified for tskid, it means specifying the issuing task itself. If TSK_SELF is specified for tskid in non-task context, operation of the service call cannot be guaranteed.

If this service call is issued to a task that has not been placed in WAITING state or in WAITING-SUSPENDED state by execution of slp_tsk or tslp_tsk, the wakeup request is accumulated. More specifically, the wakeup request count for the target task to be awakened is incremented by 1, in which way wakeup requests are accumulated.

The maximum value of the wakeup request count is 255. If while the wakeup request count = 255 a new wakeup request is generated exceeding this limit, the error code E_QOVR is returned to the task that issued the service call, with the wakeup request count left intact.

If this service call is to be issued from task context, use wup_tsk; if issued from non-task context, use iwup_tsk.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
    if( wup_tsk( ID_main ) != E_OK )
        printf("Can't wakeup main()\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     wup_tsk     #ID_TASK1
           :
```

can_wup　　Cancel wakeup request
ican_wup　　Cancel wakeup request (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task

- **Return Parameters**

ER_UINT	wupcnt > 0	Canceled wakeup request count
	wupcnt <0	Error code

[[Assembly language API]]

```
.include mr308.inc
can_wup  TSKID
ican_wup  TSKID
```

- **Parameters**

TSKID	ID number of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code,Canceled wakeup request count
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)

[[Functional description]]

This service call clears the wakeup request count of the target task indicated by tskid to 0. This means that because the target task was in either WAITING state nor WAITING-SUSPENDED state when an attempt was made to wake it up by wup_tsk or iwup_tsk before this service call was issued, the attempt resulted in only accumulating wakeup requests and this service call clears all of those accumulated wakeup requests.

Furthermore, the wakeup request count before being cleared to 0 by this service call, i.e., the number of wakeup requests that were issued in vain (wupcnt) is returned to the issuing task. If a wakeup request is issued while the target task is in DORMANT state, the error code E_OBJ is returned. If TSK_SELF is specified for tskid, it means specifying the issuing task itself.

If this service call is to be issued from task context, use can_wup; if issued from non-task context, use ican_wup.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
      ER_UINT wupcnt;
                :
      wupcnt = can_wup(ID_main);
      if( wup_cnt  > 0 )
          printf("wupcnt = %d\n",wupcnt);
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    can_wup     #ID_TASK3
          :
```

rel_wai Release task from waiting
irel_wai Release task from waiting (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
rel_wai  TSKID
irel_wai  TSKID
```

- **Parameters**

TSKID	ID number of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is not an wait state)

[[Functional description]]

This service call forcibly release the task indicated by tskid from waiting (except SUSPENDED state) to place it into READY or RUNNING state. The forcibly released task returns the error code E_RLWAI. If the target task has been enqueued in some waiting queue, the task is dequeued from it by execution of this service call.

If this service call is issued to a task in WAITING-SUSPENDED state, the target task is released from WAITING state and goes to SUSPENDED state.[5]

If the target task is not in a wait state, the error code E_OBJ is returned to the service call issuing task. This service call forbids specifying the issuing task itself for tskid.

If this service call is to be issued from task context, use rel_wai; if issued from non-task context, use irel_wai.

[5] This means that tasks cannot be resumed from SUSPENDED state by this service call. Only the rsm_tsk, irsm_tsk, frsm_tsk, and ifrsm_tsk service calls can release them from SUSPENDED state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
              :
       if( rel_wai( ID_main ) != E_OK )
           error("Can't rel_wai main()\n");
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
            :
     PUSHM       A0
     rel_wai     #ID_TASK2
            :
```

sus_tsk Suspend task
isus_tsk Suspend task (handler only)

[[C Language API]]

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

- **Parameters**

ID	tskid	ID number of the target task

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
sus_tsk  TSKID
isus_tsk  TSKID
```

- **Parameters**

TSKID	ID number of the target task

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is an inactive state)
E_QOVR	Queuing overflow

[[Functional description]]

This service call aborts execution of the task indicated by tskid and places it into SUSPENDED state. Tasks are resumed from this SUSPENDED state by the rsm_tsk, irsm_tsk, frsm_tsk, or ifrsm_tsk service call. If the task indicated by tskid is in DORMANT state, it returns the error code E_OBJ as a return value for the service call.

The maximum number of suspension requests by this service call that can be nested is 1. If this service call is issued to a task which is already in SUSPENDED state, the error code E_QOVR is returned.

This service call forbids specifying the issuing task itself for tskid.

If this service call is to be issued from task context, use sus_tsk; if issued from non-task context, use isus_tsk.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
       if( sus_tsk( ID_main ) != E_OK )
           printf("Can't suspend task main()\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     sus_tsk     #ID_TASK2
           :
```

rsm_tsk	**Resume suspended task**
irsm_tsk	**Resume suspended task(handler only)**
frsm_tsk	**Forcibly resume suspended task**
ifrsm_tsk	**Forcibly resume suspended task(handler only)**

[[C Language API]]

```
ER ercd = rsm_tsk( ID tskid );
ER ercd = irsm_tsk( ID tskid );
ER ercd = frsm_tsk( ID tskid );
ER ercd = ifrsm_tsk( ID tskid );
```

● Parameters

ID	tskid	ID number of the target task

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
rsm_tsk  TSKID
irsm_tsk  TSKID
frsm_tsk TSKID
ifrsm_tsk TSKID
```

● Parameters

TSKID	ID number of the target task

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target task

[[Error code]]

E_OBJ	Object status invalid(task indicated by tskid is not a forcible wait state)

[[Functional description]]

If the task indicated by tskid has been aborted by sus_tsk, this service call resumes the target task from SUSPENDED state. In this case, the target task is linked to behind the tail of the ready queue. In the case of frsm_tsk and ifrsm_tsk, the task is forcibly resumed from SUSPENDED state.

If a request is issued while the target task is not in SUSPENDED state (including DORMANT state), the error code E_OBJ is returned to the service call issuing task.

The rsm_tsk, irsm_tsk, frsm_tsk, and ifrsm_tsk service calls each operate the same way, because the maximum number of forcible wait requests that can be nested is 1.

If this service call is to be issued from task context, use rsm_tsk/frsm_tsk; if issued from non-task context, use irsm_tsk/ifrsm_tsk.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1()
  {
                :
       if( rsm_tsk( ID_main ) != E_OK )
           printf("Can't resume main()\n");
                :
                :
       if(frsm_tsk( ID_task2 ) != E_OK )
           printf("Can't forced resume task2()\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     rsm_tsk     #ID_TASK2
           :
     PUSHM       A0
     frsm_tsk    #ID_TASK1
           :
```

dly_tsk　　Delay task

[[C Language API]]

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

- **Parameters**

RELTIM	dlytim	Delay time

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
dly_tsk  RELTIM
```

- **Parameters**

RELTIM	Delay time

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R1	Delay time (16 low-order bits)
R3	Delay time (16 high-order bits)

[[Error code]]

E_RLWAI	Forced release from waiting

[[Functional description]]

This service call temporarily stops execution of the issuing task itself for a duration of time specified by dlytim to place the task from RUNNING state into WAITING state. In this case, the task is released from the WAITING state at the first time tick after the time specified by dlytim has elapsed. Therefore, if specified dlytim = 0, the task is placed into WAITING state briefly and then released from the WAITING state at the first time tick.

The task placed into WAITING state by invocation of this service call is released from the WAITING state in the following cases. Note that when released from WAITING state, the task that issued the service call is removed from the timeout waiting queue and linked to a ready queue.

◆ **When the first time tick occurred after dlytim elapsed**

The error code returned in this case is E_OK.

◆ **When the rel_wai or irel_wai service call is issued before dlytim elapses**

The error code returned in this case is E_RLWAI.

Note that even when the wup_tsk or iwup_tsk service call is issued during the delay time, the task is not released from WAITNG state.

The delay time dlytim is expressed in ms units. Therefore, if specified as dly_tsk(50);, the issuing task is placed from RUNNING state into a delayed wait state for a period of 50 ms.

The values specified for dlytim must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly.

This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
       if( dly_tsk() != E_OK )
           error("Forced wakeup\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       R1,R3
     dly_tsk     #500
           :
```

1.3. Synchronization & Communication Function (Semaphore)

Specifications of the semaphore function of MR308 are listed in Table 3. Specifications of the Semaphore Function.

Item No.	Item	Content
1	Semaphore ID	1-255
2	Maximum number of resources	1-65535
3	Semaphore attribute	TA_FIFO: Tasks enqueued in order of FIFO TA_TPRI: Tasks enqueued in order of priority

Table 3. Specifications of the Semaphore Function

sig_sem Release semaphore resource
isig_sem Release semaphore resource (handler only)

[[C Language API]]

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

- **Parameters**

ID	semid	Semaphore ID number to which returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
sig_sem SEMID
isig_sem SEMID
```

- **Parameters**

SEMID	Semaphore ID number to which returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Semaphore ID number to which returned

[[Error code]]

E_QOVR	Queuing overflow

[[Functional description]]

This service call releases one resource to the semaphore indicated by semid.
If tasks are enqueued in a waiting queue for the target semaphore, the task at the top of the queue is placed into READY state. Conversely, if no tasks are enqueued in that waiting queue, the semaphore resource count is incremented by 1. If an attempt is made to return resources (sig_sem or isig_sem service call) causing the semaphore resource count value to exceed the maximum value specified in a configuration file (maxsem), the error code E_QOVR is returned to the service call issuing task, with the semaphore count value left intact.
If this service call is to be issued from task context, use sig_sem; if issued from non-task context, use isig_sem.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
       if( sig_sem( ID_sem ) == E_QOVR )
         error("Overflow\n");
                :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     sig_sem     #ID_SEM2
           :
```

wai_sem	**Acquire semaphore resource**
pol_sem	**Acquire semaphore resource (polling)**
ipol_sem	**Acquire semaphore resource (polling, handler only)**
twai_sem	**Acquire semaphore resource(with timeout)**

[[C Language API]]

```
ER ercd = wai_sem( ID semid );
ER ercd = pol_sem( ID semid );
ER ercd = ipol_sem( ID semid );
ER ercd = twai_sem( ID semid, TMO tmout );
```

● Parameters

ID	semid	Semaphore ID number to be acquired
TMO	tmout	Timeout value (for twai_sem)

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
wai_sem SEMID
pol_sem SEMID
ipol_sem SEMID
twai_sem SEMID,TMO
```

● Parameters

SEMID	Semaphore ID number to be acquired
TMO	Timeout value(twai_sem)

● Register contents after service call is issued

`wai_sem,pol_sem,ipol_sem`

Register name	Content after service call is issued
R0	Error code
A0	Semaphore ID number to be acquired

`twai_sem`

Register name	Content after service call is issued
R0	Error code
R1	Timeout value(16 low-order bits)
R3	Timeout value(16 high-order bits)
A0	Semaphore ID number to be acquired

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout

[[Functional description]]

This service call acquires one semaphore resource from the semaphore indicated by semid.

If the semaphore resource count is equal to or greater than 1, the semaphore resource count is decremented by 1, and the service call issuing task continues execution. On the other hand, if the semaphore count value is 0, the wai_sem or twai_sem service call invoking task is enqueued in a waiting queue for that semaphore. If the attribute of the semaphore semid is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. For the pol_sem and ipol_sem service calls, the task returns immediately and responds to the call with the error code E_TMOUT.

For the twai_sem service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, operation of the service call cannot be guaranteed. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as pol_sem. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as wai_sem.

The task placed into WAITING state by execution of the wai_sem or twai_sem service call is released from the WAITING state in the following cases:

- **When the sig_sem or isig_sem service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.

If this service call is to be issued from task context, use wai_sem, twai_sem, or pol_sem; ; if issued from non-task context, use ipol_sem.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
                :
    if( wai_sem( ID_sem ) != E_OK )
        printf("Forced wakeup\n");
                :
    if( pol_sem( ID_sem ) != E_OK )
        printf("Timeout\n");
                :
      if( twai_sem( ID_sem, 10 ) != E_OK )
          printf("Forced wakeup or Timeout"n");
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    pol_sem     #ID_SEM1
          :
    PUSHM       A0
    wai_sem     #ID_SEM2
          :
    PUSHM       A0,R1,R3
    twai_sem    #ID_SEM3,300
          :
```

ref_sem Reference semaphore status
iref_sem Reference semaphore status (handler only)

[[C Language API]]

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

- **Parameters**

ID	semid	ID number of the target semaphore
T_RSEM	*pk_rsem	Pointer to the packet to which semaphore status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
T_RSEM	*pk_rsem	Pointer to the packet to which semaphore status is returned

Contents of pk_rsem

```
typedef   struct t_rsem{
```

ID	wtskid	+0	2	ID number of the task at the head of the semaphore's wait queue
UINT	semcnt	+2	2	Current semaphore resource count

```
} T_RSEM;
```

[[Assembly language API]]

```
.include mr308.inc
ref_sem SEMID, PK_RSEM
iref_sem SEMID, PK_RSEM
```

- **Parameters**

SEMID	ID number of the target semaphore
PK_RSEM	Pointer to the packet to which semaphore status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target semaphore
A1	Pointer to the packet to which semaphore status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the semaphore indicated by semid.

- **wtskid**
  Returned to wtskid is the ID number of the task at the head of the semaphore's wait queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **semcnt**
  Returned to semcnt is the current semaphore resource count.

If this service call is to be issued from task context, use ref_sem; if issued from non-task context, use iref_sem.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RSEM rsem;
       ER ercd;
            :
       ercd = ref_sem( ID_sem1, &rsem );
            :
  }
```

<<Example statement in assembly language>>

```
_ refsem:       .blkb   4
     .include mr308.inc
     .GLB       task
task:
            :
         PUSHM   A0,A1
         ref_sem #ID_SEM1,#_refsem
            :
```

1.4. Synchronization & Communication Function (Eventflag)

Specifications of the eventflag function of MR308 are listed in Table 4. .

Item No.	Item	
1	Event0flag ID	1–255
2	Number of bits comprising eventflag	16 bits
3	Eventflag attribute	TA_TFIFO: Waiting tasks enqueued in order of FIFO TA_TPRI: Waiting tasks enqueued in order of priority TA_WSGL: Multiple tasks cannot be kept waiting TA_WMUL: Multiple tasks can be kept waiting TA_CLR: Bit pattern cleared when waiting task is released

Table 4. Specifications of the Eventflag Function

set_flg Set eventflag
iset_flg Set eventflag (handler only)

[[C Language API]]

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

- **Parameters**

ID	flgid	ID number of the eventflag to be set
FLGPTN	setptn	Bit pattern to be set

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
set_flg FLGID,SETPTN
iset_flg FLGID,SETPTN
```

- **Parameters**

FLGID	ID number of the eventflag to be set
SETPTN	Bit pattern to be set

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R3	Bit pattern to be set
A0	Eventflag ID number

[[Error code]]

None

[[Functional description]]

Of the 16-bit eventflag indicated by flgid, this service call sets the bits indicated by setptn. In other words, the value of the eventflag indicated by flgid is OR'd with setptn. If the alteration of the eventflag value results in task-awaking conditions for a task that has been kept waiting for the eventflag by the wai_flg or twai_flg service call becoming satisfied, the task is released from WAITING state and placed into READY or RUNNING state.

Task-awaking conditions are evaluated sequentially beginning with the top of the waiting queue. If TA_WMUL is specified as an eventflag attribute, multiple tasks kept waiting for the eventflag can be released from WAITING state at the same time by one set_flg or iset_flg service call issued. Furthermore, if TA_CLR is specified for the attribute of the target eventflag, all bit patterns of the eventflag are cleared, with which processing of the service call is terminated.

If all bits specified in setptn are 0, no operation will be performed for the target eventflag, in which case no errors are assumed, however.

If this service call is to be issued from task context, use set_flg; if issued from non-task context, use iset_flg.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task(void)
  {
                :
    set_flg( ID_flg,(FLGPTN)0xff00 );
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0, R3
    set_flg     #ID_FLG3,#0ff00H
          :
```

clr_flg Clear eventflag
iclr_flg Clear eventflag (handler only)

[[C Language API]]

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

- **Parameters**

ID	flgid	ID number of the eventflag to be cleared
FLGPTN	clrptn	Bit pattern to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
clr_flg FLGID,CLRPTN
iclr_flg FLGID,CLRPTN
```

- **Parameters**

FLGID	ID number of the eventflag to be cleared
CLRPTN	Bit pattern to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the eventflag to be cleared
R3	Bit pattern to be cleared

[[Error code]]

None

[[Functional description]]

Of the 16-bit eventflag indicated by flgid, this service call clears the bits whose corresponding values in clrptn are 0. In other words, the eventflag bit pattern indicated by flgid is updated by AND'ing it with clrptn. If all bits specified in clrptn are 1, no operation will be performed for the target eventflag, in which case no errors are assumed, however.

If this service call is to be issued from task context, use clr_flg; if issued from non-task context, use iclr_flg.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task(void)
  {
                :
    clr_flg( ID_flg,(FLGPTN) 0xf0f0);
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0, R3
    clr_flg     #ID_FLG1,#0f0f0H
          :
```

wai_flg	**Wait for eventflag**
pol_flg	**Wait for eventflag(polling)**
ipol_flg	**Wait for eventflag(polling, handler only)**
twai_flg	**Wait for eventflag(with timeout)**

[[C Language API]]

```
ER ercd = wai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = pol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = ipol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = twai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn,
                    TMO tmout );
```

● Parameters

ID	flgid	ID number of the eventflag waited for
FLGPTN	waiptn	Wait bit pattern
MODE	wfmode	Wait mode
FLGPTN	*p_flgptn	Pointer to the area to which bit pattern is returned when released from wait
TMO	tmout	Timeout value (for twai_flg)

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code
FLGPTN	*p_flgptn	Pointer to the area to which bit pattern is returned when released from wait

[[Assembly language API]]

```
.include mr308.inc
wai_flg   FLGID, WAIPTN, WFMODE
pol_flg   FLGID, WAIPTN, WFMODE
ipol_flg  FLGID, WAIPTN, WFMODE
twai_flg  FLGID, WAIPTN, WFMODE, TMO
```

● Parameters

FLGID	ID number of the eventflag waited for
WAIPTN	Wait bit pattern
WFMODE	Wait mode
TMO	Timeout value (for twai_flg)

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
R1	Wait mode
R2	bit pattern is returned when released from wait
R3	Wait bit pattern
A0	ID number of the eventflag waited for

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out
E_ILUSE	Service call improperly used (Tasks present waiting for TA_WSGL attribute eventflag)

[[Functional description]]

This service call waits until the eventflag indicated by flgid has its bits specified by waiptn set according to task-awaking conditions indicated by wfmode. Returned to the area pointed to by p_flgptn is the eventflag bit pattern at the time the task is released from WAITING state.
If the target eventflag has the TA_WSGL attribute, or there are already other tasks waiting for the eventflag, the error code E_ILUSE is returned.
If task-awaking conditions have already been met when this service call is invoked, the task returns immediately and responds to the call with E_OK. If task-awaking conditions are not met and the invoked service call is wai_flg or twai_flg, the task is enqueued in an eventflag waiting queue. In that case, if the attribute of the specified eventflag is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. For the pol_flg and ipol_flg service calls, the task returns immediately and responds to the call with the error code E_TMOUT.
For the twai_flg service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as pol_flg. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as wai_flg.

The task placed into a wait state by execution of the wai_flg or twai_flg service call is released from WAITING state in the following cases:

- **When task-awaking conditions are met before the tmout time elapses**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.

The following shows how wfmode is specified and the meaning of each mode.

wfmdoe (wait mode)	Meaning
TWF_ANDW	Wait until all bits specified by waiptn are set (wait for the bits AND'ed)
TWF_ORW	Wait until one of the bits specified by waiptn is set (wait for the bits OR'ed)

If this service call is to be issued from task context, use wai_flg,twai_flg,pol_flg; if issued from non-task context, use ipol_flg.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
    UINT flgptn;
                :
      if(wai_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ANDW, &flgptn)!=E_OK)
        error("Wait Released\n");
                :
                :
      if(pol_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ORW, &flgptn)!=E_OK)
        printf("Not set EventFlag\n");
                :
                :
      if( twai_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ANDW, &flgptn, 5) != E_OK )
        error("Wait Released\n");
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,R1,R3
    wai_flg     #ID_FLG1,#0003H,#TWF_ANDW
          :
    PUSHM       A0,R1,R3
    pol_flg     #ID_FLG2,#0008H,#TWF_ORW
          :
    PUSHM       A0,R1,R3
    wai_flg     #ID_FLG3,#0003H,#TWF_ANDW,20
          :
```

ref_flg Reference eventflag status
iref_flg Reference eventflag status (handler only)

[[C Language API]]

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

● Parameters

ID	flgid	ID number of the target eventflag
T_RFLG	*pk_rflg	Pointer to the packet to which eventflag status is returned

● Return Parameters

ER	ercd	Terminated normally (E_OK)
T_RFLG	*pk_rflg	Pointer to the packet to which eventflag status is returned

```
Contents of pk_rflg
typedef   struct t_rflg{
    ID        wtskid      +0    2      Reception waiting task ID
    FLGPTN    flgptn      +2    2      Current eventflag bit pattern
} T_RFLG;
```

[[Assembly language API]]

```
.include mr308.inc
ref_flg   FLGID, PK_RFLG
iref_flg  FLGID, PK_RFLG
```

● Parameters

FLGID	ID number of the target eventflag
PK_RFLG	Pointer to the packet to which eventflag status is returned

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target eventflag
A1	Pointer to the packet to which eventflag status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the eventflag indicated by flgid.

- **wtskid**
  Returned to wtskid is the ID number of the task at the top of a waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **flgptn**
  Returned to flgptn is the current eventflag bit pattern.

If this service call is to be issued from task context, use ref_flg; if issued from non-task context, use iref_flg.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RFLG rflg;
       ER ercd;
            :
       ercd = ref_flg( ID_FLG1, &rflg );
            :
  }
```

<<Example statement in assembly language>>

```
_ refflg:       .blkb   4
     .include mr308.inc
     .GLB       task
task:
            :
         PUSHM   A0,A1
         ref_flg #ID_FLG1,#_refflg
            :
```

1.5. Synchronization & Communication Function (Data Queue)

Specifications of the data queue function of MR308 are listed in Table 5. Specifications of the Data Queue Function.

Item No.	Item	Content
1	Data queue ID	1-255
2	Capacity (data bytes) in data queue area	0-65535
3	Data size	32 bits
4	Data queue attribute	TA_TFIFO: Waiting tasks enqueued in order of FIFO TA_TPRI: Waiting tasks enqueued in order of priority

Table 5. Specifications of the Data Queue Function

snd_dtq	**Send to data queue**
psnd_dtq	**Send to data queue (polling)**
ipsnd_dtq	**Send to data queue (polling, handler only)**
tsnd_dtq	**Send to data queue (with timeout)**
fsnd_dtq	**Forced send to data queue**
ifsnd_dtq	**Forced send to data queue (handler only)**

[[C Language API]]

```
ER ercd = snd_dtq( ID dtqid, VP_INT data );
ER ercd = psnd_dtq( ID dtqid, VP_INT data );
ER ercd = ipsnd_dtq( ID dtqid, VP_INT data );
ER ercd = tsnd_dtq( ID dtqid, VP_INT data, TMO tmout );
ER ercd = fsnd_dtq( ID dtqid, VP_INT data );
ER ercd = ifsnd_dtq( ID dtqid, VP_INT data );
```

- Parameters

ID	dtqid	ID number of the data queue to which transmitted
TMO	tmout	Timeout value(tsnd_dtq)
VP_INT	data	Data to be transmitted

- Return Parameters

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
snd_dtq   DTQID, DTQDATA
isnd_dtq  DTQID, DTQDATA  ipsnd_dtq DTQID, DTQDATA
psnd_dtq  DTQID, DTQDATA
ipsnd_dtq DTQID, DTQDATA
tsnd_dtq  DTQID, DTQDATA,TMO
fsnd_dtq  DTQID, DTQDATA
ifsnd_dtq DTQID, DTQDATA
```

● Parameters

DTQID	ID number of the data queue to which transmitted
DTQDATA	Data to be transmitted
TMO	Timeout value (tsnd_dtq)

● Register contents after service call is issued

snd_dtq,psnd_dtq,ipsnd_dtq,fsnd_dtq,ifsnd_dtq

Register name	Content after service call is issued
R0	Error code
R1	Data to be transmitted (16 low-order bits)
R3	Data to be transmitted (16 high-order bits)
A0	ID number of the data queue to which transmitted

tsnd_dtq

Register name	Content after service call is issued
R0	Error code
R1	Data to be transmitted (16 low-order bits)
R2	Timeout value (16 high-order bits)
R3	Data to be transmitted (16 high-order bits)
A0	ID number of the data queue to which transmitted

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out
E_ILUSE	Service call improperly used (fsnd_dtq or ifsnd_dtq is issued for a data queue whose dtqcnt = 0)
EV_RST	Released from WAITING state by clearing of the data queue area

[[Functional description]]

This service call sends the 4-byte data indicated by data to the data queue indicated by dtqid. If any task is kept waiting for reception in the target data queue, the data is not stored in the data queue and instead sent to the task at the top of the reception waiting queue, with which the task is released from the reception wait state.

On the other hand, if snd_dtq or tsnd_dtq is issued for a data queue that is full of data, the task that issued the service call goes from RUNNING state to a data transmission wait state, and is enqueued in transmission waiting queue, kept waiting for the data queue to become available. In that case, if the attribute of the specified data queue is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. For psnd_dtq and ipsnd_dtq, the task returns immediately and responds to the call with the error code E_TMOUT.

For the tsnd_dtq service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as psnd_dtq. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as snd_dtq.

If there are no tasks waiting for reception, nor is the data queue area filled, the transmitted data is stored in the data queue.

The task placed into WAITING state by execution of the snd_dtq or tsnd_dtq service call is released from WAITING state in the following cases:

- **When the rcv_dtq, trcv_dtq, prcv_dtq, or iprcv_dtq service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.
- **When the target data queue being waited for is removed by the vrst_dtq service call issued from another task**
  The error code returned in this case is EV_RST.

For fsnd_dtq and ifsnd_dtq, the data at the top of the data queue or the oldest data is removed, and the transmitted data is stored at the tail of the data queue. If the data queue area is not filled with data, fsnd_dtq and ifsnd_dtq operate the same way as snd_dtq.
If this service call is to be issued from task context, use snd_dtq,tsnd_dtq,psnd_dtq,fsnd_dtq; if issued from non-task context, use ipsnd_dtq,ifsnd_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  VP_INT data[10];
  void task(void)
  {
                :
      if( snd_dtq( ID_dtq, data[0]) == E_RLWAI ){
        error("Forced released\n");
    }
                :
      if( psnd_dtq( ID_dtq, data[1])== E_TMOUT ){
        error("Timeout\n");
    }
                :
      if( tsnd_dtq( ID_dtq, data[2], 10 ) != E_ TMOUT ){
        error("Timeout \n");
    }
                :
      if( fsnd_dtq( ID_dtq, data[3]) != E_OK ){
        error("error\n");
    }
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
_g_dtq: .LWORD  12345678H
task:
          :
    PUSHM       R1,R2,R3,A0
    tsnd_dtq    #ID_DTQ1,_g_dtq,#100
          :
    PUSHM       R1,R3,A0
    psnd_dtq    #ID_DTQ2,#0FFFFH
          :
    PUSHM       R1,R3,A0
    fsnd_dtq    #ID_DTQ3,#0ABCDH
          :
```

rcv_dtq	**Receive from data queue**
prcv_dtq	**Receive from data queue (polling)**
iprcv_dtq	**Receive from data queue (polling, handler only)**
trcv_dtq	**Receive from data queue (with timeout)**

[[C Language API]]

```
ER ercd = rcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = prcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = iprcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = trcv_dtq( ID dtqid, VP_INT *p_data, TMO tmout );
```

● Parameters

ID	dtqid	ID number of the data queue from which to receive
TMO	tmout	Timeout value (trcv_dtq)
VP_INT	*p_data	Pointer to the beginning of the area in which received data is stored

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code
VP_INT	*p_data	Pointer to the beginning of the area in which received data is stored

[[Assembly language API]]

```
.include mr308.inc
rcv_dtq   DTQID
prcv_dtq  DTQID
iprcv_dtq DTQID
trcv_dtq  DTQID,TMO
```

● Parameters

DTQID	ID number of the data queue from which to receive
TMO	Timeout value (trcv_dtq)

● Register contents after service call is issued

`rcv_dtq,prcv_dtq,iprcv_dtq`

Register name	Content after service call is issued
R0	Error code
R1	Received data (16 low-order bits)
R3	Received data (16 high-order bits)
A0	Data queue ID number

`trcv_dtq`

Register name	Content after service call is issued
R0	Error code
R1	Received data (16 low-order bits)
R2	Timeout value(16 high-order bits)
R3	Received data (16 high-order bits)
A0	ID number of the data queue from which to receive

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out

[[Functional description]]

This service call receives data from the data queue indicated by dtqid and stores the received data in the area pointed to by p_data. If data is present in the target data queue, the data at the top of the queue or the oldest data is received. This results in creating a free space in the data queue area, so that a task enqueued in a transmission waiting queue is released from WAITING state, and starts sending data to the data queue area.

If no data exist in the data queue and there is any task waiting to send data (i.e., data bytes in the data queue area = 0), data for the task at the top of the data transmission waiting queue is received. As a result, the task kept waiting to send that data is released from WAITING state.

On the other hand, if rcv_dtq or trcv_dtq is issued for the data queue which has no data stored in it, the task that issued the service call goes from RUNNING state to a data reception wait state, and is enqueued in a data reception waiting queue. At this time, the task is enqueued in order of FIFO. For the prcv_dtq and iprcv_dtq service calls, the task returns immediately and responds to the call with the error code E_TMOUT.

For the trcv_dtq service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as prcv_dtq. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as rcv_dtq.

The task placed into a wait state by execution of the rcv_dtq or trcv_dtq service call is released from the wait state in the following cases:

- **When the rcv_dtq, trcv_dtq, prcv_dtq, or iprcv_dtq service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.

If this service call is to be issued from task context, use rcv_dtq,trcv_dtq,prcv_dtq; if issued from non-task context, use iprcv_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
                                          void task()
  {
    VP_INT data;
                :
      if( rcv_dtq( ID_dtq, &data ) != E_RLWAI )
          error("forced wakeup\n");
                :
      if( prcv_dtq( ID_dtq, &data ) != E_TMOUT )
          error("Timeout\n");
                :
      if( trcv_dtq( ID_dtq, &data, 10 ) != E_TMOUT )
          error("Timeout \n");
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    trcv_dtq    #ID_DTQ1,#TMO_POL
                  :
    PUSHM       A0
    prcv_dtq    #ID_DTQ2
          :
    PUSHM       A0
    rcv_dtq     #ID_DTQ2
          :
```

ref_dtq Reference data queue status
iref_dtq Reference data queue status (handler only)

[[C Language API]]

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

- **Parameters**

ID	dtqid	ID number of the target data queue
T_RDTQ	*pk_rdtq	Pointer to the packet to which data queue status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
T_RDTQ	*pk_rdtq	Pointer to the packet to which data queue status is returned

Contents of pk_rdtq

```
typedef   struct t_rdtq{
    ID      stskid    +0   2    Transmission waiting task ID
    ID      wtskid    +2   2    Reception waiting task ID
    UINT    sdtqcnt   +4   2    Data bytes contained in data queue
} T_RDTQ;
```

[[Assembly language API]]

```
.include mr308.inc
ref_dtq   DTQID, PK_RDTQ
iref_dtq  DTQID, PK_RDTQ
```

- **Parameters**

DTQID	ID number of the target data queue
PK_RDTQ	Pointer to the packet to which data queue status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target data queue
A1	Pointer to the packet to which data queue status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the data queue indicated by dtqid.

- **stskid**
  Returned to stskid is the ID number of the task at the top of a transmission waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **wtskid**
  Returned to wtskid is the ID number of the task at the top of a reception waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **sdtqcnt**
  Returned to sdtqcnt is the number of data bytes stored in the data queue area.

If this service call is to be issued from task context, use ref_dtq; if issued from non-task context, use iref_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RDTQ rdtq;
       ER ercd;
            :
       ercd = ref_dtq( ID_DTQ1, &rdtq );
            :
  }
```

<<Example statement in assembly language>>

```
_ refdtq:       .blkb   6
     .include mr308.inc
     .GLB       task
task:
            :
         PUSHM   A0,A1
         ref_dtq #ID_DTQ1,#_refdtq
            :
```

1.6. Synchronization & Communication Function (Mailbox)

Specifications of the mailbox function of MR308 are listed in Table 6. Specifications of the Mailbox Function.

Item No.	Item	Content
1	Mailbox ID	1-255
2	Mailbox priority	1-255
3	Mailbox attribute	TA_TFIFO: Waiting tasks enqueued in order of FIFO TA_TPRI: Waiting tasks enqueued in order of priority TA_MFIFO: Messages enqueued in order of FIFO TA_MPRI: Messages enqueued in order of priority

Table 6. Specifications of the Mailbox Function

snd_mbx Send to mailbox
isnd_mbx Send to mailbox (handler only)

[[C Language API]]

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

- Parameters

ID	mbxid	ID number of the mailbox to which transmitted
T_MSG	*pk_msg	Message to be transmitted

- Return Parameters

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
snd_mbx   MBXID,PK_MBX
isnd_mbx  MBXID,PK_MBX
```

- Parameters

MBXID	ID number of the mailbox to which transmitted
PK_MBX	Message to be transmitted (address)

- Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	ID number of the mailbox to which transmitted
A1	Message to be transmitted (address)

[[Structure of the message packet]]

```
<<Mailbox message header>>
typedef   struct t_msg{
    VP      msghead   +0    4     Kernel managed area
} T_MSG;
<<Mailbox message header with priority included>>
typedef   struct t_msg{
    T_MSG   msgque    +0    4     Message header
    PRI     msgpri    +4    2     Message priority
} T_MSG;
```

[[Error code]]

None

[[Functional description]]

This service call sends the message indicated by pk_msg to the mailbox indicated by mbxid. T_MSG* should be specified with a 32-bit address. If there is any task waiting to receive a message in the target mailbox, the transmitted message is passed to the task at the top of the waiting queue, and the task is released from WAITING state.
To send a message to a mailbox whose attribute is TA_MFIFO, add a T_MSG structure at the beginning of the message when creating it, as shown in the example below.
To send a message to a mailbox whose attribute is TA_MPRI, add a T_MSG_PRI structure at the beginning of the message when creating it, as shown in the example below.
Messages should always be created in a RAM area regardless of whether its attribute is TA_MFIFO or TA_MPRI.
The T_MSG area is used by the kernel, so that it cannot be rewritten after a message has been sent. If this area is rewritten before the message is received after it was sent, operation of the service call cannot be guaranteed.
If this service call is to be issued from task context, use snd_mbx; if issued from non-task context, use isnd_mbx.

<<Example format of a message>>

```
typedef struct user_msg{
        T_MSG  t_msg;        /* T_MSG structure */
        B      data[16];     /* User message data */
} USER_MSG;
```

<<Example format of a message with priority included>>

```
typedef struct user_msg{
        T_MSG_PRI     t_msg;        /* T_MSG_PRI structure */
        B             data[16];     /* User message data */
} USER_MSG;
```

[[Example program statement]]

<<Example statement in C language>>

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
typedef struct pri_message
{
       T_MSG_PRI      msgheader;
       char   body[12];
} PRI_MSG;

void task(void)
{
   PRI_MSG     msg;
               :
   msg.msgpri = 5;
   snd_mbx( ID_msg,(T_MSG)&msg);
       :
}
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
_g_userMsg:     .blkb   4       ; Header
                .blkb   12      ; Body
task:
          :
    PUSHM       A0,A1
    snd_mbx     #ID_MBX1,#_g_userMsg              :
          :
```

rcv_mbx Receive from mailbox
prcv_mbx Receive from mailbox (polling)
iprcv_mbx Receive from mailbox (polling, handler only)
trcv_mbx Receive from mailbox (with timeout)

[[C Language API]]

```
ER ercd = rcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = prcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = iprcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = trcv_mbx( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

- **Parameters**

ID	mbxid	ID number of the mailbox from which to receive
TMO	tmout	Timeout value (for trcv_mbx)
T_MSG	**ppk_msg	Pointer to the beginning of the area in which received message is stored

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code
T_MSG	**ppk_msg	Pointer to the beginning of the area in which received message is stored

[[Assembly language API]]

```
.include mr308.inc
rcv_mbx   MBXID
prcv_mbx  MBXID
iprcv_mbx MBXID
trcv_mbx  MBXID, TMO
```

- **Parameters**

MBXID	ID number of the mailbox from which to receive
TMO	Timeout value (for trcv_mbx)

- **Register contents after service call is issued**

rcv_mbx,prcv_mbx,iprcv_mbx

Register name	Content after service call is issued
R0	Error code
R1	Received message (upper address)
R2	Received message (lower address)
A0	ID number of the mailbox from which to receive

trcv_mbx

Register name	Content after service call is issued
R0	Error code
R1	Received message (upper address)
R2	Received message (lower address)
R3	Timeout value(16 high-order bits)
A0	ID number of the mailbox from which to receive

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out

[[Functional description]]

This service call receives a message from the mailbox indicated by mbxid and stores the beginning address of the received message in the area pointed to by ppk_msg. T_MSG* should be specified with a 32-bit address. If data is present in the target mailbox, the data at the top of the mailbox is received.

On the other hand, if rcv_mbx or trcv_mbx is issued for a mailbox that has no messages in it, the task that issued the service call goes from RUNNING state to a message reception wait state, and is enqueued in a message reception waiting queue. In that case, if the attribute of the specified mailbox is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. For prcv_mbx and iprcv_mbx, the task returns immediately and responds to the call with the error code E_TMOUT.

For the trcv_mbx service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as prcv_mbx. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as rcv_mbx.

The task placed into WAITING state by execution of the rcv_mbx or trcv_mbx service call is released from WAITING state in the following cases:

- **When the rcv_mbx, trcv_mbx, prcv_mbx, or iprcv_mbx service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.

If this service call is to be issued from task context, use rcv_mbx,trcv_mbx,prcv_mbx; if issued from non-task context, use iprcv_mbx.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"

  typedef struct fifo_message
  {
        T_MSG   head;
        char    body[12];
  } FIFO_MSG;
  void task()
  {
    FIFO_MSG *msg;
                :
      if( rcv_mbx((T_MSG *)&msg, ID_mbx) == E_RLWAI )
          error("forced wakeup\n");
                :
                :
      if( prcv_mbx((T_MSG *)&msg, ID_mbx) != E_TMOUT )
          error("Timeout\n");
                :
                :
      if( trcv_mbx((T_MSG *)&msg, ID_mbx,10) != E_TMOUT )
          error("Timeout\n");
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       R3,A0
    trcv_mbx    #ID_MBX1,#100
          :
    PUSHM       R3,A0
    rcv_mbx     #ID_MBX1
          :
    PUSHM       R3,A0
    prcv_mbx    #ID_MBX1
          :
```

ref_mbx Reference mailbox status
iref_mbx Reference mailbox status (handler only)

[[C Language API]]

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

- **Parameters**

ID	Mbxid	ID number of the target mailbox
T_RMBX	*pk_rmbx	Pointer to the packet to which mailbox status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
T_RMBX	*pk_rmbx	Pointer to the packet to which mailbox status is returned

```
Contents of pk_rmbx
typedef   struct t_rmbx{
    ID      wtskid    +0    2    Reception waiting task ID
    T_MSG   *pk_msg   +4    4    Next message packet to be received
} T_RMBX;
```

[[Assembly language API]]

```
.include mr308.inc
ref_mbx   MBXID, PK_RMBX
iref_mbx  MBXID, PK_RMBX
```

- **Parameters**

MBXID	ID number of the target mailbox
PK_RMBX	Pointer to the packet to which mailbox status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target mailbox
A1	Pointer to the packet to which mailbox status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the mailbox indicated by mbxid.

- **wtskid**
  Returned to wtskid is the ID number of the task at the top of a reception waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- ***pk_msg**
  Returned to *pk_msg is the beginning address of the next message to be received. If there are no messages to be received next, NULL is returned. T_MSG* should be specified with a 32-bit address.

If this service call is to be issued from task context, use ref_mbx; if issued from non-task context, use iref_mbx.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RMBX rmbx;
       ER ercd;
            :
       ercd = ref_mbx( ID_MBX1, &rmbx );
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_ refmbx:        .blkb   6
task:
            :
         PUSHM   A0,A1
         ref_mbx #ID_MBX1,#_refmbx
            :
```

1.7. Memory Pool Management Function (Fixed-size Memory Pool)

Specifications of the fixed-size memory pool function of MR308 are listed in Table 7. Specifications of the Fixed-size memory pool Function.

The memory pool area to be acquired can be specified by a section name for each memory pool during configuration.

Item No.	Item	Content
1	Fixed-size memory pool ID	1-255
2	Number of fixed-size memory pools	1-65535
3	Size of fixed-size memory pool	4-65535
4	Supported attributes	TA_TFIFO: Waiting tasks enqueued in order of FIFO TA_TPRI: Waiting tasks enqueued in order of priority
5	Specification of memory pool area	Area to be acquired specifiable by a section

Table 7. Specifications of the Fixed-size memory pool Function

get_mpf	**Aquire fixed-size memory block**
pget_mpf	**Aquire fixed-size memory block (polling)**
ipget_mpf	**Aquire fixed-size memory block (polling, handler only)**
tget_mpf	**Aquire fixed-size memory block (with timeout)**

[[C Language API]]

```
ER ercd = get_mpf( ID mpfid, VP *p_blk );
ER ercd = pget_mpf( ID mpfid, VP *p_blk );
ER ercd = ipget_mpf( ID mpfid, VP *p_blk );
ER ercd = tget_mpf( ID mpfid, VP *p_blk,TMO tmout );
```

- **Parameters**

ID	mpfid	ID number of the target fixed-size memory pool to be acquired
VP	*p_blk	Pointer to the beginning address of the acquired memory block
TMO	tmout	Timeout value(tget_mpf)

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code
VP	*p_blk	Pointer to the beginning address of the acquired memory block

[[Assembly language API]]

```
.include mr308.inc
get_mpf   MPFID
pget_mpf  MPFID
ipget_mpf MPFID
tget_mpf  MPFID,TMO
```

- **Parameters**

MPFID	ID number of the target fixed-size memory pool to be acquired
TMO	Timeout value(tget_mpf)

● Register contents after service call is issued

`get_mpf,pget_mpf,ipget_mpf`

Register name	Content after service call is issued
R0	Error code
R1	Beginning address of the acquired memory block (16 low-order bits)
R3	Beginning address of the acquired memory block (16 high-order bits)
A0	ID number of the target fixed-size memory pool to be acquired

`tget_mpf`

Register name	Content after service call is issued
R0	Error code
R1	Beginning address of the acquired memory block (16 low-order bits)
R2	Timeout value(16 high-order bits)
R3	Beginning address of the acquired memory block (16 high-order bits)
A0	ID number of the target fixed-size memory pool to be acquired

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out
EV_RST	Released from WAITING state by clearing of the memory pool area

[[Functional description]]

This service call acquires a memory block from the fixed-size memory pool indicated by mpfid and stores the beginning address of the acquired memory block in the variable p_blk. The content of the acquired memory block is indeterminate.

If the fixed-size memory pool indicated by mpfid has no memory blocks in it and the used service call is tget_mpf or get_mpf, the task that issued it goes to a memory block wait state and is enqueued in a memory block waiting queue. In that case, if the attribute of the specified fixed-size memory pool is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. If the issued service call was pget_mpf or ipget_mpf, the task returns immediately and responds to the call with the error code E_TMOUT.

For the tget_mpf service call, specify a wait time for tmout in ms units. The values specified for tmout must be within (0x7FFFFFFF – time tick). If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as pget_mpf. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as get_mpf.

The task placed into WAITING state by execution of the get_mpf or tget_mpf service call is released from WAITING state in the following cases:

- **When the rel_mpf or irel_mpf service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.
- **When the target memory pool being waited for is removed by the vrst_mpf service call issued from another task**
  The error code returned in this case is EV_RST.

The value of the memory block acquired by this service call is indeterminate because it is not initialized.

If this service call is to be issued from task context, use get_mpf,pget_mpf,tget_mpf; if issued from non-task context, use ipget_mpf.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  VP      p_blk;
  void task()
  {
    if( get_mpf(ID_mpf ,&p_blk) != E_OK ){
          error("Not enough memory\n");
    }
                :
    if( pget_mpf(ID_mpf ,&p_blk) != E_OK ){
          error("Not enough memory\n");
    }
                :
    if( tget_mpf(ID_mpf ,&p_blk, 10) != E_OK ){
          error("Not enough memory\n");
    }

  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    get_mpf     #ID_MPF1
          :
    PUSHM       A0
    pget_mpf    #ID_MPF1
          :
    PUSHM       A0
    tget_mpf    #ID_MPF1,#200
          :
```

rel_mpf Release fixed-size memory block
irel_mpf Release fixed-size memory block (handler only)

[[C Language API]]

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

- **Parameters**

ID	mpfid	ID number of the fixed-size memory pool to be released
VP	blk	Beginning address of the memory block to be returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
rel_mpf   MPFID,BLK
irel_mpf  MPFID,BLK
```

- **Parameters**

MPFID	ID number of the fixed-size memory pool to be released
BLK	Beginning address of the memory block to be returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R1	Beginning address of the memory block to be returned (16 low-order bits)
R3	Beginning address of the memory block to be returned (16 high-order bits)
A0	ID number of the fixed-size memory pool to be released

[[Error code]]

None

[[Functional description]]

This service call releases a memory block whose beginning address is indicated by blk. The beginning address of the memory block to be released that is specified here should always be that of the memory block acquired by get_mpf, tget_mpf, pget_mpf, or ipget_mpf.
If tasks are enqueued in a waiting queue for the target memory pool, the task at the top of the waiting queue is dequeued and linked to a ready queue, and is assigned a memory block. At this time, the task changes state from a memory block wait state to RUNNING or READY state. This service call does not check the content of blk, so that if the address stored in blk is incorrect, the service call may not operate correctly.
If this service call is to be issued from task context, use rel_mpf; if issued from non-task context, use irel_mpf.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
      VP   p_blf;
      if( get_mpf(ID_mpf1,&p_blf) != E_OK )
          error("Not enough memory \n");
                :
      rel_mpf(ID_mpf1,p_blf);
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_g_blk: .blkb 4
task:
           :
     PUSHM       A0
     get_mpf     #ID_MPF1
           :
     MOV.L       R3R1,_g_blk
     PUSHM       A0
     rel_mpf     #ID_MPF1,_g_blk
           :
```

ref_mpf Reference fixed-size memory pool status
iref_mpf Reference fixed-size memory pool status (handler only)

[[C Language API]]

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

- **Parameters**

ID	mpfid	Task ID waiting for memory block to be acquired
T_RMPF	*pk_rmpf	Pointer to the packet to which fixed-size memory pool status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
T_RMPF	*pk_rmpf	Pointer to the packet to which fixed-size memory pool status is returned

```
Contents of pk_rmpf
typedef   struct t_rmpf{
    ID      wtskid     +0    2     Task ID waiting for memory block to be acquired
    UINT    fblkcnt    +2    2     Number of free memory blocks
} T_RMPF;
```

[[Assembly language API]]

```
.include mr308.inc
ref_mpf   MPFID,PK_RMPF
iref_mpf  MPFID,PK_RMPF
```

- **Parameters**

MPFID	Task ID waiting for memory block to be acquired
PK_RMPF	Pointer to the packet to which fixed-size memory pool status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Task ID waiting for memory block to be acquired
A1	Pointer to the packet to which fixed-size memory pool status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the message buffer indicated by mpfid.

- **wtskid**
  Returned to wtskid is the ID number of the task at the top of a memory block waiting queue (the first queued task). If no tasks are kept waiting, TSK_NONE is returned.
- **fblkcnt**
  The number of free memory blocks in the specified memory pool is returned.

If this service call is to be issued from task context, use rel_mpf; if issued from non-task context, use irel_mpf.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RMPF rmpf;
       ER ercd;
            :
       ercd = ref_mpf( ID_MPF1, &rmpf );
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_ refmpf:        .blkb   4
task:
            :
         PUSHM   A0,A1
         ref_mpf #ID_MPF1,#_refmpf
            :
```

1.8. Memory Pool Management Function (Variable-size Memory Pool)

Specifications of the Variable-size Memory pool function of MR308 are listed in Table 8. Specifications of the variable-size memory Pool Function.

The memory pool area to be acquired can be specified by a section name for each memory pool during configuration.

Item No.	Item	Content
1	Variable-size memory pool ID	1-255
2	Size of Variable-size Memory pool	16-524288
3	Maximum number of memory blocks to be acquired	1-65520
4	Supported attributes	When memory is insufficient, task-waiting APIs are not supported.
5	Specification of memory pool area	Area to be acquired specifiable by a section

Table 8. Specifications of the variable-size memory Pool Function

pget_mpl Aquire variable-size memory block (polling)

[[C Language API]]

```
ER ercd = pget_mpl( ID mplid, UINT blksz, VP *p_blk );
```

- Parameters

ID	mplid	ID number of the target Variable-size Memory pool to be acquired
UINT	blksz	Memory size to be acquired (in bytes)
VP	*p_blk	Pointer to the beginning address of the acquired variable memory

- Return Parameters

ER	ercd	Terminated normally (E_OK) or error code
VP	*p_blk	Pointer to the beginning address of the acquired variable memory

[[Assembly language API]]

```
.include mr308.inc
pget_mpl MPLID,BLKSZ
```

- Parameters

MPLID	ID number of the target Variable-size Memory pool to be acquired
BLKSZ	Memory size to be acquired (in bytes)

- Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
R1	Memory size to be acquired (16 low-order bits)
R3	Memory size to be acquired (16 high-order bits)
A0	ID number of the target Variable-size Memory pool to be acquired

[[Error code]]

E_TMOUT	No memory block

[[Functional description]]

This service call acquires a memory block from the variable-size memory pool indicated by mplid and stores the beginning address of the acquired memory block in the variable p_blk. The content of the acquired memory block is indeterminate.

If the specified variable-size memory pool has no memory blocks in it, the task returns immediately and responds to the call with the error code E_TMOUT.

The value of the memory block acquired by this service call is indeterminate because it is not initialized.

This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  VP      p_blk;
  void task()
  {
     if( pget_mpl(ID_mpl , 200, &p_blk) != E_OK ){
          error("Not enough memory\n");
     }
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
          :
     PUSHM       A0
     pget_mpl    #ID_MPL1,#200
          :
```

rel_mpl Release variable-size memory block

[[C Language API]]

```
ER ercd = rel_mpl( ID mplid, VP blk );
```

- **Parameters**

ID	mplid	ID number of Variable-size Memory pool of the memory block to be released
VP	Blk	Beginning address of the memory block to be returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
rel_mpl   MPLID,BLK
```

- **Parameters**

MPLID	ID number of Variable-size Memory pool of the memory block to be released
BLK	Beginning address of the memory block to be returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R1	Beginning address of the memory block to be returned (16 low-order bits)
R3	Beginning address of the memory block to be returned (16 high-order bits)
A0	ID number of Variable-size Memory pool of the memory block to be released

[[Error code]]

None

[[Functional description]]

This service call releases a memory block whose beginning address is indicated by blk. The beginning address of the memory block to be released that is specified here should always be that of the memory block acquired by pget_mpl.
This service call does not check the content of blk, so that if the address stored in blk is incorrect, the service call may not operate correctly.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
      VP   p_blk;
      if( get_mpl(ID_mpl1, 200, &p_blk) != E_OK )
          error("Not enough memory \n");
              :
      rel_mpl(ID_mp1,p_blk);
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_g_blk: .blkb 4
task:
           :
     PUSHM       A0
     get_mpl     #ID_MPL1,#200
           :
     MOV.L       R3R1,_g_blk
     PUSHM       A0
     rel_mpf     #ID_MPL1,_g_blk
           :
```

ref_mpl Reference variable-size memory pool status
iref_mpl Reference variable-size memory pool status (handler only)

[[C Language API]]

```
ER ercd = ref_mpl( ID mplid, T_RMPL *pk_rmpl );
ER ercd = iref_mpl( ID mplid, T_RMPL *pk_rmpl );
```

- **Parameters**

ID	mplid	ID number of the target variable-size memory pool
T_RMPL	*pk_rmpl	Pointer to the packet to which variable-size memory pool status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
T_RMPL	*pk_rmpl	Pointer to the packet to which variable-size memory pool status is returned

```
Contents of pk_rmpl
typedef   struct t_rmpl{
```

ID	wtskid	+0	2	Task ID waiting for memory block to be acquired (unused)
SIZE	fmplsz	+4	4	Free memory size (in bytes)
UINT	fblksz	+8	2	Maximum size of memory that can be acquired immediately (in bytes)

```
} T_RMPL;
```

[[Assembly language API]]

```
.include mr308.inc
ref_mpl   MPLID,PK_RMPL
iref_mpl  MPLID,PK_RMPL
```

- **Parameters**

MPLID	ID number of the target variable-size memory pool
PK_RMPL	Pointer to the packet to which variable-size memory pool status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target variable-size memory pool
A1	Pointer to the packet to which variable-size memory pool status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the message buffer indicated by mplid.

- **wtskid**
  Unused.
- **fmplsz**
  A free memory size is returned.
- **fblksz**
  The maximum size of memory that can be acquired immediately is returned.

If this service call is to be issued from task context, use ref_mpl; if issued from non-task context, use iref_mpl.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RMPL rmpl;
       ER ercd;
            :
       ercd = ref_mpl( ID_MPL1, &rmpl );
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_ refmpl:        .blkb   8
task:
            :
         PUSHM   A0,A1
         ref_mpl #ID_MPL1,_refmpl
            :
```

1.9. Time Management Function

Specifications of the time management function of MR308 are listed in Table 9. .

Item No.	Item	Content
1	System time value	Unsigned 48 bits
2	Unit of system time value	1[ms]
3	System time updating cycle	User-specified time tick updating time [ms]
4	Initial value of system time (at initial startup)	000000000000H

Table 9. Specifications of the Time Management Function

set_tim Set system time
iset_tim Set system time (handler only)

[[C Language API]]

```
ER ercd = set_tim( SYSTIM *p_systim );
ER ercd = iset_tim( SYSTIM *p_systim );
```

- **Parameters**

SYSTIM	*p_systim	Pointer to the packet that indicates the system time to be set

```
Contents of p_systim
typedef   struct t_systim {
    UH      utime       0     2     (16 high-order bits)
    UW      ltime       +4    4     (32 low-order bits)
} SYSTIM;
```

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
set_tim   PK_TIM
iset_tim  PK_TIM
```

- **Parameters**

PK_TIM	Pointer to the packet that indicates the system time to be set

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Pointer to the packet that indicates the system time to be set

[[Error code]]

None

[[Functional description]]

This service call updates the current value of the system time to the value indicated by p_systim. The time specified in p_systim is expressed in ms units, and not by the number of time ticks.
The values specified for p_systim must be within 0x7FFF: FFFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly.
If this service call is to be issued from task context, use set_tim; if issued from non-task context, use iset_tim.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     SYSTIME time;         /* Time data storing variable */
     time.utime = 0;       /* Sets upper time data */
     time.ltime = 0;       /* Sets lower time data */
     set_tim( &time );     /* Sets the system time */
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB       task
_g_systim:
        .WORD   1111H
        .LWORD  22223333H
task:
           :
        PUSHM   A0
        set_tim #_g_systim
           :
```

get_tim Reference system time
iget_tim Reference system time (handler only)

[[C Language API]]

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

- **Parameters**

SYSTIM	*p_systim	Pointer to the packet to which current system time is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
SYSTIM	*p_systim	Pointer to the packet to which current system time is returned

Contents of p_systim

```
typedef   struct t_systim {
```

UH	utime	0	2	(16 high-order bits)
UW	ltime	+4	4	(32 low-order bits)

```
} SYSTIM;
```

[[Assembly language API]]

```
.include mr308.inc
get_tim   PK_TIM
iget_tim  PK_TIM
```

- **Parameters**

PK_TIM	Pointer to the packet to which current system time is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Pointer to the packet to which current system time is returned

[[Error code]]

None

[[Functional description]]

This service call stores the current value of the system time in p_systim.

If this service call is to be issued from task context, use get_tim; if issued from non-task context, use iget_tim.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
    SYSTIME time;         /* Time data storing variable */
    get_tim( &time );     /* Refers to the system time */
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
_g_systim:      .blkb   6
task:
          :
        PUSHM   A0
        get_tim #_g_systim
          :
```

isig_tim Supply a time tick

[[Functional description]]

This service call updates the system time.

The isig_tim is automatically started every tick_time interval(ms) if the system clock is defined by the configuration file. The application cannot call this function because it is not implementing as service call.

When a time tick is supplied, the kernel is processed as follows:

(1) Updates the system time
(2) Starts an alarm handler
(3) Starts a cyclic handler
(4) Processes the timeout processing of the task put on WAITING state by service call with timeout such as tslp_tsk.

1.10. Time Management Function (Cyclic Handler)

Specifications of the cyclic handler function of MR308 are listed in Table 10. Specifications of the Cyclic Handler Function. The cyclic handler description languages in item No. 4 are those specified in the GUI configurator. They are not output to a configuration file, nor are the MR308 kernel concerned with them.

Item No.	Item	Content
1	Cyclic handler ID	1-255
2	Activation cycle	0-7fffffff[ms]
3	Activation phase	0-7fffffff[ms]
4	Extended information	32 bits
4	Cyclic handler attribute	TA_HLNG: Handlers written in high-level language TA_ASM: Handlers written in assembly language TA_STA: Starts operation of cyclic handler TA_PHS: Saves activation phase

Table 10. Specifications of the Cyclic Handler Function

sta_cyc　Start cyclic handler operation
ista_cyc　Start cyclic handler operation (handler only)

[[C Language API]]

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

- **Parameters**

ID	cycid	ID number of the cyclic handler to be operated

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
sta_cyc   CYCNO
ista_cyc  CYCNO
```

- **Parameters**

CYCNO	ID number of the cyclic handler to be operated

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the cyclic handler to be operated

[[Error code]]

None

[[Functional description]]

This service call places the cyclic handler indicated by cycid into an operational state. If the cyclic handler attribute of TA_PHS is not specified, the cyclic handler is started every time the activate cycle elapses, beginning with the time at which this service call was invoked.

If while TA_PHS is not specified this service call is issued to a cyclic handler already in an operational state, it sets the time at which the cyclic handler is to start next.

If while TA_PHS is specified this service call is issued to a cyclic handler already in an operational state, it does not set the startup time.

If this service call is to be issued from task context, use sta_cyc; if issued from non-task context, use ista_cyc.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
     sta_cyc ( ID_cyc1 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB       task
task:
          :
        PUSHM   A0
        sta_cyc #ID_CYC1
          :
```

stp_cyc　　Stops cyclic handler operation
istp_cyc　　Stops cyclic handler operation (handler only)

[[C Language API]]

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

- **Parameters**

ID	cycid	ID number of the cyclic handler to be stopped

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
stp_cyc   CYCNO
istp_cyc  CYCNO
```

- **Parameters**

CYCNO	ID number of the cyclic handler to be stopped

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the cyclic handler to be stopped

[[Error code]]

None

[[Functional description]]

This service call places the cyclic handler indicated by cycid into a non-operational state.
If this service call is to be issued from task context, use stp_cyc; if issued from non-task context, use istp_cyc.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
    stp_cyc ( ID_cyc1 );
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        stp_cyc #ID_CYC1
          :
```

ref_cyc Reference cyclic handler status
iref_cyc Reference cyclic handler status (handler only)

[[C Language API]]

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

● Parameters

ID	cycid	ID number of the target cyclic handler
T_RCYC	*pk_rcyc	Pointer to the packet to which cyclic handler status is returned

● Return Parameters

ER	ercd	Terminated normally (E_OK)
T_RCYC	*pk_rcyc	Pointer to the packet to which cyclic handler status is returned

Contents of pk_rcyc

```
typedef   struct t_rcyc{
```

STAT	cycstat	+0	2	Operating status of cyclic handler
RELTIM	lefttim	+2	4	Left time before cyclic handler starts up

```
} T_RCYC;
```

[[Assembly language API]]

```
.include mr308.inc
ref_cyc   ID,PK_RCYC
iref_cyc  ID,PK_RCYC
```

● Parameters

CYCNO	ID number of the target cyclic handler
PK_RCYC	Pointer to the packet to which cyclic handler status is returned

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target cyclic handler
A1	Pointer to the packet to which cyclic handler status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the cyclic handler indicated by cycid.

- **cycstat**
  The status of the target cyclic handler is returned.
  - *TCYC_STA Cyclic handler is an operational state.
  - *TCYC_STP Cyclic handler is a non-operational state.
- **lefttim**
  The remaining time before the target cyclic handler will start next is returned. This time is expressed in ms units. If the target cyclic handler is non-operational state, the returned value is indeterminate.

If this service call is to be issued from task context, use ref_cyc; if issued from non-task context, use iref_cyc.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RCYC rcyc;
       ER ercd;
            :
       ercd = ref_cyc( ID_CYC1, &rcyc );
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_ refcyc:        .blkb   6
task:
            :
         PUSHM   A0,A1
         ref_cyc #ID_CYC1,#_refcyc
            :
```

1.11. Time Management Function (Alarm Handler)

Specifications of the alarm handler function of MR308 are listed in Table 11. Specifications of the Alarm Handler Function. The alarm handler description languages in item No. 4 are those specified in the GUI configurator. They are not output to a configuration file, nor are the MR308 kernel concerned with them.

Item No.	Item	Content
1	Alarm handler ID	1-255
2	Activation time	0-7fffffff [ms]
3	Extended information	32 bits
4	Alarm handler attribute	TA_HLNG: Handlers written in high-level language TA_ASM: Handlers written in assembly language

Table 11. Specifications of the Alarm Handler Function

sta_alm　　Start alarm handler operation
ista_alm　　Start alarm handler operation (handler only)

[[C Language API]]

```
ER ercd = sta_alm( ID almid, RELTIM almtim );
ER ercd = ista_alm( ID almid, RELTIM almtim );
```

- **Parameters**

ID	almid	ID number of the alarm handler to be operated
RELTIM	almtim	Alarm handler startup time (relative time)

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
sta_alm   ALMID,ALMTIM
ista_alm  ALMID,ALMTIM
```

- **Parameters**

ALMID	ID number of the alarm handler to be operated
ALMTIM	Alarm handler startup time (relative time)

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
R1	Alarm handler startup time (relative time) (16 low-order bits)
R3	Alarm handler startup time (relative time) (16 high-order bits)
A0	ID number of the alarm handler to be operated

[[Error code]]

None

[[Functional description]]

This service call sets the activation time of the alarm handler indicated by almid as a relative time of day after the lapse of the time specified by almtim from the time at which it is invoked, and places the alarm handler into an operational state.

If an already operating alarm handler is specified, the previously set activation time is cleared and updated to a new activation time. If almtim = 0 is specified, the alarm handler starts at the next time tick. The values specified for almtim must be within (0x7FFFFFFF – time tick). If any value exceeding this limit is specified, the service call may not operate correctly. If 0 is specified for almtim , the alarm handler is started at the next time tick.

If this service call is to be issued from task context, use sta_alm; if issued from non-task context, use ista_alm.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
     sta_alm ( ID_alm1,100 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB       task
task:
          :
        PUSHM   A0
        sta_alm #ID_ALM1,#100
          :
```

stp_alm　　Stop alarm handler operation
istp_alm　　Stop alarm handler operation (handler only)

[[C Language API]]

```
ER ercd = stp_alm( ID almid );
ER ercd = istp_alm( ID almid );
```

- **Parameters**

ID	almid	ID number of the alarm handler to be stopped

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
stp_alm   ALMID
istp_alm  ALMID
```

- **Parameters**

ALMID	ID number of the alarm handler to be stopped

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the alarm handler to be stopped

[[Error code]]

None

[[Functional description]]

This service call places the alarm handler indicated by almid into a non-operational state.
If this service call is to be issued from task context, use stp_alm; if issued from non-task context, use istp_alm.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
    stp_alm ( ID_alm1 );
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
        PUSHM   A0
        stp_alm #ID_ALM1
          :
```

ref_alm Reference alarm handler status
iref_alm Reference alarm handler status (handler only)

[[C Language API]]

```
ER ercd = ref_alm( ID almid, T_RALM *pk_ralm );
ER ercd = iref_alm( ID almid, T_RALM *pk_ralm );
```

● Parameters

ID	almid	ID number of the target alarm handler
T_RALM	*pk_ralm	Pointer to the packet to which alarm handler status is returned

● Return Parameters

ER	ercd	Terminated normally (E_OK)
T_RALM	*pk_ralm	Pointer to the packet to which alarm handler status is returned

```
Contents of pk_ralm
typedef   struct t_ralm{
```

STAT	almstat	+0	2	Operating status of alarm handler
RELTIM	lefttim	+2	4	This service call returns various statuses of the alarm handler indicat

```
} T_RALM;
```

[[Assembly language API]]

```
.include mr308.inc
ref_alm   ALMID,PK_RALM
iref_alm  ALMID,PK_RALM
```

● Parameters

ALMID	ID number of the target alarm handler
PK_RALM	Pointer to the packet to which alarm handler status is returned

● Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target alarm handler
A1	Pointer to the packet to which alarm handler status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the alarm handler indicated by almid.

- **almstat**
  The status of the target alarm handler is returned.

*TALM_STA	Alarm handler is an operational state.
*TALM_STP	Alarm handler is a non-operational state.

- **lefttim**
  The remaining time before the target alarm handler will start next is returned. This time is expressed in ms units. If the target alarm handler is a non-operational state, the returned value is indeterminate.

If this service call is to be issued from task context, use ref_alm; if issued from non-task context, use iref_alm.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RALM ralm;
       ER ercd;
            :
       ercd = ref_alm( ID_ALM1, &ralm );
            :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
_ refalm:        .blkb   6
task:
            :
         PUSHM   A0,A1
         ref_alm #ID_ALM1,#_refalm
            :
```

1.12. System Status Management Function

rot_rdq	**Rotate task precedence**
irot_rdq	**Rotate task precedence (handler only)**

[[C Language API]]

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

- Parameters

PRI	tskpri	Task priority to be rotated

- Return Parameters

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
rot_rdq   TSKPRI
irot_rdq  TSKPRI
```

- Parameters

TSKPRI	Task priority to be rotated

- Register contents after service call is issued

Register name	Content after service call is issued
R0	Error code
R3	Task priority to be rotated

[[Error code]]

None

[[Functional description]]

This service call rotates the ready queue whose priority is indicated by tskpri. In other words, it relocates the task enqueued at the top of the ready queue of the specified priority by linking it to behind the tail of the ready queue, thereby switching over the executed tasks that have the same priority. Figure 1-1 depicts the manner of how this is performed.

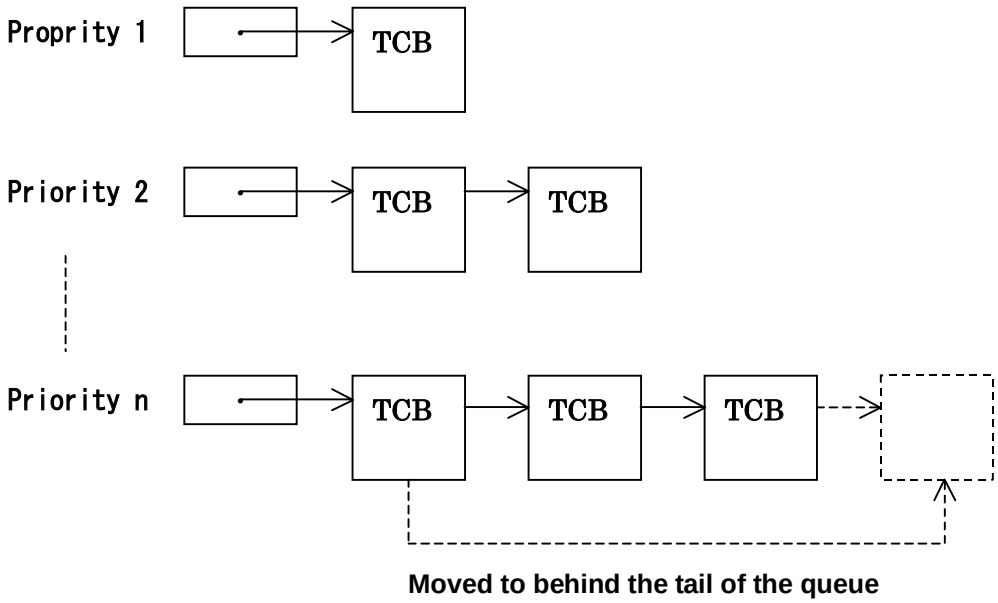


Figure 1-1. Manipulation of the ready queue by the rot_rdq service call

By issuing this service call at given intervals, it is possible to perform round robin scheduling. If tskpri=TPRI_SELF is specified when using the rot_rdq service call, the ready queue whose priority is that of the issuing task is rotated. TPRI_SELF cannot be specified in the irot_rdq service call. TPRI_SELF cannot be specified by irot_rdq service call. However, an error is not returned even if it is specified.

If the priority of the issuing task itself is specified in this service call, the issuing task is relocated to behind the tail of the ready queue in which it is enqueued. Note that if the ready queue of the specified priority has no tasks in it, no operation is performed.

If this service call is to be issued from task context, use rot_rdq; if issued from non-task context, use irot_rdq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
          :
      rot_rdq( 2 );
          :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   R3
        rot_rdq #2
          :
```

get_tid Reference task ID in the RUNNING state
iget_tid Reference task ID in the RUNNING state (handler only)

[[C Language API]]

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

- **Parameters**

ID	*p_tskid	Pointer to task ID

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)
ID	*p_tskid	Pointer to task ID

[[Assembly language API]]

```
.include mr308.inc
get_tid
iget_tid
```

- **Parameters**

None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Acquired task ID

[[Error code]]

None

[[Functional description]]

This service call returns the task ID currently in RUNNING state to the area pointed to by p_tskid. If this service call is issued from a task, the ID number of the issuing task is returned. If this service call is issued from non-task context, the task ID being executed at that point in time is returned. If there are no tasks currently in an executing state, TSK_NONE is returned.
If this service call is to be issued from task context, use get_tid; if issued from non-task context, use iget_tid.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     ID tskid;
        :
      get_tid(&tskid);
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        get_tid
           :
```

loc_cpu　　Lock the CPU
iloc_cpu　　Lock the CPU (handler only)

[[C Language API]]

```
ER ercd = loc_cpu();
ER ercd = iloc_cpu();
```

- **Parameters**

None

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
loc_cpu
iloc_cpu
```

- **Parameters**

None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code

[[Error code]]

None

[[Functional description]]

This service call places the system into a CPU locked state, thereby disabling interrupts and task dispatches. The features of a CPU locked state are outlined below.

(1) No task scheduling is performed during a CPU locked state.
(2) No external interrupts are accepted unless their priority levels are higher than the kernel interrupt mask level defined in the configurator.
(3) Only the following service calls can be invoked from a CPU locked state. If any other service calls are invoked, operation of the service call cannot be guaranteed.
  * ext_tsk
  * loc_cpu, iloc_cpu
  * unl_cpu, iunl_cpu
  * sns_ctx
  * sns_loc
  * sns_dsp
  * sns_dpn

The system is freed from a CPU locked state by one of the following operations.

(a) Invocation of the unl_cpu or iunl_cpu service call
(b) Invocation of the ext_tsk service call

Transitions between CPU locked and CPU unlocked states occur only when the loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, or ext_tsk service call is invoked. The system must always be in a CPU unlocked state when the interrupt handler or the time event handler is terminated. If either handler terminates while the system is in a CPU locked state, handler operation cannot be guaranteed. Note that the system is always in a CPU unlocked state when these handlers start.
Invoking this service call again while the system is already in a CPU locked state does not cause an error, in which case task queuing is not performed, however.
If this service call is to be issued from task context, use loc_cpu; if issued from non-task context, use iloc_cpu.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
       loc_cpu();
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
         loc_cpu
           :
```

unl_cpu　　Unlock the CPU
iunl_cpu　　Unlock the CPU (handler only)

[[C Language API]]

```
ER ercd = unl_cpu();
ER ercd = iunl_cpu();
```

- **Parameters**

None

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
unl_cpu
iunl_cpu
```

- **Parameters**

None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code

[[Error code]]

None

[[Functional description]]

This service call frees the system from a CPU locked state that was set by the loc_cpu or iloc_cpu service call. If the unl_cpu service call is issued from a dispatching enabled state, task scheduling is performed. If the system was put into a CPU locked state by invoking iloc_cpu within an interrupt handler, the system must always be placed out of a CPU locked state by invoking iunl_cpu before it returns from the interrupt handler.

The CPU locked state and the dispatching disabled state are managed independently of each other. Therefore, the system cannot be freed from a dispatching disabled state by the unl_cpu or iunl_cpu service call unless the ena_dsp service call is used.

If this service call is to be issued from task context, use unl_cpu; if issued from non-task context, use iunl_cpu.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
      unl_cpu();
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
        unl_cpu
          :
```

dis_dsp Disable dispatching

[[C Language API]]

```
ER ercd = dis_dsp();
```

- **Parameters**

  None

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
dis_dsp
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code

[[Error code]]

None

[[Functional description]]

This service call places the system into a dispatching disabled state. The features of a dispatching disabled state are outlined below.

(1) Since task scheduling is not performed anymore, no tasks other than the issuing task itself will be placed into RUNNING state.
(2) Interrupts are accepted.
(3) No service calls can be invoked that will place tasks into WAITING state.

If one of the following operations is performed during a dispatching disabled state, the system status returns to a task execution state.

(a) Invocation of the ena_dsp service call
(b) Invocation of the ext_tsk service call

Transitions between dispatching disabled and dispatching enabled states occur only when the dis_dsp, ena_dsp, or ext_tsk service call is invoked.
Invoking this service call again while the system is already in a dispatching disabled state does not cause an error, in which case task queuing is not performed, however.
This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
       dis_dsp();
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
         dis_dsp
           :
```

ena_dsp Enables dispatching

[[C Language API]]

```
ER ercd = ena_dsp();
```

- **Parameters**

  None

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
ena_dsp
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code

[[Error code]]

None

[[Functional description]]

This service call frees the system from a dispatching disabled state that was set by the dis_dsp service call. As a result, task scheduling is resumed when the system has entered a task execution state.

Invoking this service call from a task execution state does not cause an error, in which case task queuing is not performed, however.

This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

```
<<Example statement in C language>>
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
        :
       ena_dsp();
        :
  }
<<Example statement in assembly language>>
     .include mr308.inc
     .GLB       task
task:
          :
        ena_dsp
          :
```

sns_ctx Reference context

[[C Language API]]

```
BOOL state = sns_ctx();
```

- **Parameters**

  None

- **Return Parameters**

BOOL	state	TRUE: Non-task context FALSE: Task context

[[Assembly language API]]

```
.include mr308.inc
sns_ctx
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	TRUE:Non-Task context FALSE: Task context

[[Error code]]

None

[[Functional description]]

This service call returns TRUE when it is invoked from non-task context, or returns FALSE when invoked from task context. This service call can also be invoked from a CPU locked state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
        :
      stat = sns_ctx();
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
        sns_ctx
          :
```

sns_loc Reference CPU state

[[C Language API]]

```
BOOL state = sns_loc();
```

- **Parameters**

  None

- **Return Parameters**

BOOL	state	TRUE: CPU locked state FALSE: CPU unlocked state

[[Assembly language API]]

```
.include mr308.inc
sns_loc
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	TRUE: CPU locked state FALSE:CPUCPU unlocked state

[[Error code]]

None

[[Functional description]]

This service call returns TRUE when the system is in a CPU locked state, or returns FALSE when the system is in a CPU unlocked state. This service call can also be invoked from a CPU locked state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
        :
       stat = sns_loc();
         :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
        sns_loc
           :
```

sns_dsp Reference dispatching state

[[C Language API]]

```
BOOL state = sns_dsp();
```

- **Parameters**

  None

- **Return Parameters**

BOOL	state	TRUE: Dispatching disabled state FALSE: Dispatching enabled state

[[Assembly language API]]

```
.include mr308.inc
sns_dsp
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	TRUE: Dispatching disabled state FALSE: Dispatching enabled state

[[Error code]]

None

[[Functional description]]

This service call returns TRUE when the system is in a dispatching disabled state, or returns FALSE when the system is in a dispatching enabled state. This service call can also be invoked from a CPU locked state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
        :
      stat = sns_dsp();
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
        sns_dsp
          :
```

sns_dpn Reference dispatching pending state

[[C Language API]]

```
BOOL state = sns_dpn();
```

- **Parameters**

  None

- **Return Parameters**

BOOL	state	TRUE: Dispatching pending state FALSE: Not dispatching pending state

[[Assembly language API]]

```
.include mr308.inc
sns_dpn
```

- **Parameters**

  None

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	TRUE: Dispatching pending state FALSE: Not dispatching pending state

[[Error code]]

None

[[Functional description]]

This service call returns TRUE when the system is in a dispatching pending state, or returns FALSE when the system is not in a dispatching pending state. More specifically, FALSE is returned when all of the following conditions are met; otherwise, TRUE is returned.

(1) The system is not in a dispatching pending state.
(2) The system is not in a CPU locked state.
(3) The object made pending is a task.

This service call can also be invoked from a CPU locked state. It returns TRUE when the system is in a dispatching disabled state, or returns FALSE when the system is in a dispatching enabled state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
         :
       stat = sns_dpn();
         :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
        sns_dpn
          :
```

1.13. Interrupt Management Function

ret_int — Returns from an interrupt handler (when written in assembly language)

[[C Language API]]

This service call cannot be written in C language.[6]

[[Assembly language API]]

```
.include mr308.inc
ret_int
```

■ **Parameters**

None

[[Error code]]

Not return to the interrupt handler that issued this service call.

■ **Functional description**

This service call performs the processing necessary to return from an interrupt handler. Depending on return processing, it activates the scheduler to switch tasks from one to another.

If this service call is executed in an interrupt handler, task switching does not occur, and task switching is postponed until the interrupt handler terminates.

However, if the ret_int service call is issued from an interrupt handler that was invoked from an interrupt that occurred within another interrupt, the scheduler is not activated. The scheduler is activated for interrupts from a task only.

When writing this service call in assembly language, be aware that the service call cannot be issued from a subroutine that is invoked from an interrupt handler entry routine. Always make sure this service call is executed in the entry routine or entry function of an interrupt handler. For example, a program like the one shown below may not operate normally.

```
.include mr308.inc
/* NG */
.GLB intr
intr:
          jsr.b func
:
func:
          ret_int
```

Therefore, write the program as shown below.

```
.include mr308.inc
/* OK */
.GLB intr
intr:
          jsr.b func
          ret_int
func:
:
          rts
```

Make sure this service call is issued from only an interrupt handler. If issued from a cyclic handler, alarm handler, or a task, this service call may not operate normally.

[6] If the starting function of an interrupt handler is declared by #pragma INTHANDLER, the ret_int service call is automatically issued at the exit of the function.

1.14. System Configuration Management Function

ref_ver	**Reference version information**
iref_ver	**Reference version information (handler only)**

[[C Language API]]

```
ER ercd = ref_ver( T_RVER *pk_rver );
ER ercd = iref_ver( T_RVER *pk_rver );
```

- **Parameters**

T_RVER	*pk_rver	Pointer to the packet to which version information is returned

```
Contents of pk_rver
typedef   struct t_rver {
```

UH	maker	0	2	Kernel manufacturer code
UH	prid	+2	2	Kernel identification number
UH	spver	+4	2	ITRON specification version number
UH	prver	+6	2	Kernel version number
UH	prno[4]	+8	2	Kernel product management information

```
} T_RVER;
```

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
ref_ver   PK_VER
iref_ver  PK_VER
```

- **Parameters**

PK_VER	Pointer to the packet to which version information is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Pointer to the packet to which version information is returned

[[Error code]]

None

[[Functional description]]

This service call reads out information about the version of the currently executing kernel and returns the result to the area pointed to by pk_rver.

The following information is returned to the packet pointed to by pk_rver.

- **maker**
  The code H'115 denoting Renesas Technology Corporation is returned.
- **prid**
  The internal identification code IDH'150 of the M3T-MR308 is returned.
- **spver**
  The code H'5402 denoting that the kernel is compliant with µITRON Specification Ver 4.02.00 is returned.
- **prver**
  The code H'0401 denoting the version of the M3T-MR308/4 is returned.
- **prno**
  - **prno[0]**
    The product release number '01' is acquired.
  - **prno[1]**
    The 2 low-order digits of the product release year (calendar) and the month H'0510 are acquired.
  - **prno[2]**
    Reserved for future extension.
  - **prno[3]**
    Reserved for future extension.

If this service call is to be issued from task context, use ref_ver; if issued from non-task context, use iref_ver.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
      T_RVER   pk_rver;
      ref_ver( &pk_rver );
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB       task
_ refver:       .blkb   6
task:
          :
        PUSHM   A0
        ref_ver #_refver
          :
```

1.15. Extended Function (Short Data Queue)

Specifications of the short data queue function of MR308 are listed in Table 12. Specifications of the Short Data Queue Function. This function is outside the scope of µITRON 4.0 Specification.

Item No.	Item	Content
1	Short data queue ID	1-255
2	Capacity (data bytes) in short data queue area	0-65535
3	Short data size	16 bits
4	Short data queue attribute	TA_TFIFO: Waiting tasks enqueued in order of FIFO TA_TPRI: Waiting tasks enqueued in order of priority

Table 12. Specifications of the Short Data Queue Function

vsnd_dtq	**Send to short data queue**
vpsnd_dtq	**Send to short data queue (polling)**
vipsnd_dtq	**Send to short data queue (polling, handler only)**
vtsnd_dtq	**Send to short data queue (with timeout)**
vfsnd_dtq	**Forced send to short data queue**
vifsnd_dtq	**Forced send to short data queue (handler only)**

[[C Language API]]

```
ER ercd = vsnd_dtq( ID vdtqid, H data );
ER ercd = vpsnd_dtq( ID vdtqid, H data );
ER ercd = vipsnd_dtq( ID vdtqid, H data );
ER ercd = vtsnd_dtq( ID vdtqid, H data, TMO tmout );
ER ercd = vfsnd_dtq( ID vdtqid, H data );
ER ercd = vifsnd_dtq( ID vdtqid, H data );
```

● Parameters

ID	vdtqid	ID number of the short data queue to which transmitted
TMO	tmout	Timeout value(tsnd_dtq)
H	data	Data to be transmitted

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code

[[Assembly language API]]

```
.include mr308.inc
vsnd_dtq        VDTQID, DTQDATA
visnd_dtq       VDTQID, DTQDATA
vpsnd_dtq       VDTQID, DTQDATA
vipsnd_dtq      VDTQID, DTQDATA
vtsnd_dtq       VDTQID, DTQDATA,TMO
vfsnd_dtq       VDTQID, DTQDATA
vifsnd_dtq      VDTQID, DTQDATA
```

● Parameters

VDTQID	ID number of the short data queue to which transmitted
DTQDATA	Data to be transmitted
TMO	Timeout value(tsnd_dtq)

● Register contents after service call is issued

vsnd_dtq,vpsnd_dtq,vipsnd_dtq,vfsnd_dtq,vifsnd_dtq

Register name	Content after service call is issued
R0	Error code
R1	Data to be transmitted
A0	ID number of the short data queue to which transmitted

vtsnd_dtq

Register name	Content after service call is issued
R0	Error code
R1	Data to be transmitted
R2	Timeout value(16 high-order bits)
A0	ID number of the short data queue to which transmitted

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out
E_ILUSE	Service call improperly used (vfsnd_dtq or vifsnd_dtq is issued for a short data queue whose dtqcnt = 0)
EV_RST	Released from a wait state by clearing of the short data queue area

[[Functional description]]

This service call sends the signed 2-byte data indicated by data to the short data queue indicated by vdtqid. If any task is kept waiting for reception in the target short data queue, the data is not stored in the short data queue and instead sent to the task at the top of the reception waiting queue, with which the task is released from the reception wait state.
On the other hand, if vsnd_dtq or vtsnd_dtq is issued for a short data queue that is full of data, the task that issued the service call goes from RUNNING state to a data transmission wait state, and is enqueued in a transmission waiting queue, kept waiting for the short data queue to become available. In that case, if the attribute of the specified short data queue is TA_TFIFO, the task is enqueued in order of FIFO; if TA_TPRI, the task is enqueued in order of priority. For vpsnd_dtq and vipsnd_dtq, the task returns immediately and responds to the call with the error code E_TMOUT.
For the vtsnd_dtq service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as vpsnd_dtq. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as vsnd_dtq.
If there are no tasks waiting for reception, nor is the short data queue area filled, the transmitted data is stored in the short data queue.
The task placed into a wait state by execution of the vsnd_dtq or vtsnd_dtq service call is released from WAITING state in the following cases:

- **When the vrcv_dtq, vtrcv_dtq, vprcv_dtq, or viprcv_dtq service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.
- **When the target short data queue being waited for is removed by the vrst_vdtq service call issued from another task**
  The error code returned in this case is EV_RST.

For vfsnd_dtq and vifsnd_dtq, the data at the top of the short data queue or the oldest data is removed, and the transmitted data is stored at the tail of the short data queue. If the short data queue area is not filled with data, vfsnd_dtq and vifsnd_dtq operate the same way as vsnd_dtq.
If this service call is to be issued from task context, use vsnd_dtq,vtsnd_dtq,vpsnd_dtq,vfsnd_dtq; if issued from non-task context, use vipsnd_dtq,vifsnd_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  H data[10];
  void task(void)
  {
                :
      if( vsnd_dtq( ID_dtq, data[0]) == E_RLWAI ){
        error("Forced released\n");
    }
                :
      if( vpsnd_dtq( ID_dtq, data[1])== E_TMOUT ){
        error("Timeout\n");
    }
                :
      if( vtsnd_dtq( ID_dtq, data[2], 10 ) != E_ TMOUT ){
        error("Timeout \n");
    }
                :
      if( vfsnd_dtq( ID_dtq, data[3]) != E_OK ){
        error("error\n");
    }
                :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
_g_dtq: .WORD   1234H
task:
          :
    PUSHM       R1,R2,A0
    vtsnd_dtq   #ID_DTQ1,_g_dtq,#100
          :
    PUSHM       R1,A0
    vpsnd_dtq   #ID_DTQ2,#0FFFFH
          :
    PUSHM       R1,A0
    vfsnd_dtq   #ID_DTQ3,#0ABCDH
          :
```

vrcv_dtq	**Receive from short data queue**
vprcv_dtq	**Receive from short data queue (polling)**
viprcv_dtq	**Receive from short data queue (polling,handler only)**
vtrcv_dtq	**Receive from short data queue (with timeout)**

[[C Language API]]

```
ER ercd = vrcv_dtq( ID dtqid, H *p_data );
ER ercd = vprcv_dtq( ID dtqid, H *p_data );
ER ercd = viprcv_dtq( ID dtqid, H *p_data );
ER ercd = vtrcv_dtq( ID dtqid, H *p_data, TMO tmout );
```

● Parameters

ID	vdtqid	ID number of the short data queue from which to receive
TMO	tmout	Timeout value(vtrcv_dtq)
H	*p_data	Pointer to the beginning of the area in which received data is stored

● Return Parameters

ER	ercd	Terminated normally (E_OK) or error code
H	*p_data	Pointer to the beginning of the area in which received data is stored

[[Assembly language API]]

```
.include mr308.inc
vrcv_dtq        VDTQID
vprcv_dtq       VDTQID
viprcv_dtq      VDTQID
vtrcv_dtq       VDTQID,TMO
```

● Parameters

VDTQID	ID number of the short data queue from which to receive
TMO	Timeout value(trcv_dtq)

● Register contents after service call is issued

vrcv_dtq,vprcv_dtq,viprcv_dtq

Register name	Content after service call is issued
R0	Error code
R1	Received data
A0	ID number of the short data queue from which to receive

vtrcv_dtq

Register name	Content after service call is issued
R0	Error code
R1	Received data
R2	Timeout value(16 high-order bits)
A0	ID number of the short data queue from which to receive

[[Error code]]

E_RLWAI	Forced release from waiting
E_TMOUT	Polling failure or timeout or timed out

[[Functional description]]

This service call receives data from the short data queue indicated by vdtqid and stores the received data in the area pointed to by p_data. If data is present in the target short data queue, the data at the top of the queue or the oldest data is received. This results in creating a free space in the short data queue area, so that a task enqueued in a transmission waiting queue is released from WAITING state, and starts sending data to the short data queue area.

If no data exist in the short data queue and there is any task waiting to send data (i.e., data bytes in the short data queue area = 0), data for the task at the top of the data transmission waiting queue is received. As a result, the task kept waiting to send that data is released from WAITING state.

On the other hand, if vrcv_dtq or vtrcv_dtq is issued for the short data queue which has no data stored in it, the task that issued the service call goes from RUNNING state to a data reception wait state, and is enqueued in a data reception waiting queue. At this time, the task is enqueued in order of FIFO. For the vprcv_dtq and viprcv_dtq service calls, the task returns immediately and responds to the call with the error code E_TMOUT.

For the vtrcv_dtq service call, specify a wait time for tmout in ms units. The values specified for tmout must be within 0x7FFFFFFF. If any value exceeding this limit is specified, the service call may not operate correctly. If TMO_POL=0 is specified for tmout, it means specifying 0 as a timeout value, in which case the service call operates the same way as vprcv_dtq. Furthermore, if specified as tmout=TMO_FEVR(–1), it means specifying an infinite wait, in which case the service call operates the same way as vrcv_dtq.

The task placed into a wait state by execution of the vrcv_dtq or vtrcv_dtq service call is released from the wait state in the following cases:

- **When the vrcv_dtq, vtrcv_dtq, vprcv_dtq, or viprcv_dtq service call is issued before the tmout time elapses, with task-awaking conditions thereby satisfied**
  The error code returned in this case is E_OK.
- **When the first time tick occurred after tmout elapsed while task-awaking conditions remain unsatisfied**
  The error code returned in this case is E_TMOUT.
- **When the task is forcibly released from WAITING state by the rel_wai or irel_wai service call issued from another task or a handler**
  The error code returned in this case is E_RLWAI.

If this service call is to be issued from task context, use vrcv_dtq,vtrcv_dtq,vprcv_dtq; if issued from non-task context, use viprcv_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
    H data;
              :
      if( vrcv_dtq( ID_dtq, &data ) != E_RLWAI )
          error("forced wakeup\n");
              :
      if( vprcv_dtq( ID_dtq, &data ) != E_TMOUT )
          error("Timeout\n");
              :
      if( vtrcv_dtq( ID_dtq, &data, 10 ) != E_TMOUT )
          error("Timeout\n");
              :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,R3
    vtrcv_dtq   #ID_DTQ1,#TMO_POL
                  :
    PUSHM       A0
    vprcv_dtq   #ID_DTQ2
          :
    PUSHM       A0
    vrcv_dtq    #ID_DTQ2
          :
```

vref_dtq Reference short data queue status
viref_dtq Reference short data queue status (handler only)

[[C Language API]]

```
ER ercd = vref_dtq( ID vdtqid, T_RDTQ *pk_rdtq );
ER ercd = viref_dtq( ID vdtqid, T_RDTQ *pk_rdtq );
```

- **Parameters**

ID	vdtqid	ID number of the target short data queue
T_RDTQ	*pk_rdtq	Pointer to the packet to which short data queue status is returned

- **Return Parameters**

ER	ercd	Terminated normally (E_OK) or error code
T_RDTQ	*pk_rdtq	Pointer to the packet to which short data queue status is returned

```
Contents of pk_rdtq
typedef   struct t_rdtq{
```

ID	stskid	+0	2	Transmission waiting task ID
ID	wtskid	+2	2	Reception waiting task ID
UINT	sdtqcnt	+4	2	Data bytes contained in short data queue

```
} T_RDTQ;
```

[[Assembly language API]]

```
.include mr308.inc
vref_dtq  VDTQID, PK_RDTQ
viref_dtq VDTQID, PK_RDTQ
```

- **Parameters**

VDTQID	ID number of the target short data queue
PK_RDTQ	Pointer to the packet to which short data queue status is returned

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	ID number of the target short data queue
A1	Pointer to the packet to which short data queue status is returned

[[Error code]]

None

[[Functional description]]

This service call returns various statuses of the short data queue indicated by vdtqid.

- **stskid**
  Returned to stskid is the ID number of the task at the top of a transmission waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **wtskid**
  Returned to wtskid is the ID number of the task at the top of a reception waiting queue (the next task to be dequeued). If no tasks are kept waiting, TSK_NONE is returned.
- **sdtqcnt**
  Returned to sdtqcnt is the number of data bytes stored in the short data queue area.

If this service call is to be issued from task context, use ref_dtq; if issued from non-task context, use iref_dtq.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
       T_RDTQ rdtq;
       ER ercd;
            :
       ercd = vref_dtq( ID_DTQ1, &rdtq );
            :
  }
```

<<Example statement in assembly language>>

```
_ refdtq:       .blkb   6
     .include mr308.inc
     .GLB       task
task:
            :
         PUSHM   A0,A1
         vref_dtq        #ID_DTQ1,#_refdtq
            :
```

1.16. Extended Function (Reset Function)

This function initializes the content of an object. This function is outside the scope of µITRON 4.0 Specification.

vrst_dtq Clear data queue area

[[C Language API]]

```
ER ercd = vrst_dtq( ID dtqid );
```

- **Parameters**

ID	dtqid	Data queue ID to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
vrst_dtq DTQID
```

- **Parameters**

DTQID	Data queue ID to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Data queue ID to be cleared

[[Error code]]

None

[[Functional description]]

This service call clears the data stored in the data queue indicated by dtqid. If the data queue area has no more areas to be added and tasks are enqueued in a data transmission waiting queue, all of the tasks enqueued in the data transmission waiting queue are released from WAITING state. Furthermore, the error code EV_RST is returned to the tasks that have been released from WAITING state.

Even when the number of data queues defined is 0, all of the tasks enqueued in a data transmission waiting queue are released from WAITING state.

This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1(void)
  {
        :
      vrst_dtq( ID_dtq1 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     vrst_dtq    #ID_DTQ1
           :
```

vrst_vdtq Clear short data queue area

[[C Language API]]

```
ER ercd = vrst_vdtq( ID vdtqid );
```

- **Parameters**

ID	vdtqid	Short data queue ID to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
vrst_vdtq VDTQID
```

- **Parameters**

VDTQID	Short data queue ID to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Short data queue ID to be cleared

[[Error code]]

None

[[Functional description]]

This service call clears the data stored in the short data queue indicated by vdtqid. If the short data queue area has no more areas to be added and tasks are enqueued in a data transmission waiting queue, all of the tasks enqueued in the data transmission waiting queue are released from WAITING state. Furthermore, the error code EV_RST is returned to the tasks that have been released from WAITING state.

Even when the number of short data queues defined is 0, all of the tasks enqueued in a data transmission waiting queue are released from WAITING state.

This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1(void)
  {
        :
      vrst_vdtq( ID_vdtq1 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     vrst_vdtq   #ID_VDTQ1
           :
```

vrst_mbx Clear mailbox area

[[C Language API]]

```
ER ercd = vrst_mbx( ID mbxid );
```

- **Parameters**

ID	mbxid	Mailbox ID to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
vrst_mbx  MBXID
```

- **Parameters**

MBXID	Mailbox ID to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Mailbox ID to be cleared

[[Error code]]

None

[[Functional description]]

This service call clears the messages stored in the mailbox indicated by mbxid.
This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1(void)
  {
        :
      vrst_mbx( ID_mbx1 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     vrst_mbx    #ID_MBX1
           :
```

vrst_mpf Clear fixed-size memory pool area

[[C Language API]]

```
ER ercd = vrst_mpf( ID mpfid );
```

- **Parameters**

ID	mpfid	Fixed-size memory pool ID to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
vrst_mpf  MPFID
```

- **Parameters**

MPFID	Fixed-size memory pool ID to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Fixed-size memory pool ID to be cleared

[[Error code]]

None

[[Functional description]]

This service call initializes the fixed-size memory pool indicated by mpfid. If tasks are enqueued in a memory block waiting queue, all of the tasks enqueued in the memory block waiting queue are released from WAITING state. Furthermore, the error code EV_RST is returned to the tasks that have been released from WAITING state.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1(void)
  {
        :
       vrst_mpf( ID_mpf1 );
        :
  }
```

<<Example statement in assembly language>>

```
    .include mr308.inc
    .GLB       task
task:
          :
    PUSHM      A0
    vrst_mpf   #ID_MPF1
          :
```

vrst_mpl Clear variable-size memory pool area

[[C Language API]]

```
ER ercd = vrst_mpl( ID mplid );
```

- **Parameters**

ID	mplid	Variable-size memory pool ID to be cleared

- **Return Parameters**

ER	ercd	Terminated normally (E_OK)

[[Assembly language API]]

```
.include mr308.inc
vrst_mpl MPLID
```

- **Parameters**

MPLID	Variable-size memory pool ID to be cleared

- **Register contents after service call is issued**

Register name	Content after service call is issued
R0	Error code
A0	Variable-size memory pool ID to be cleared

[[Error code]]

None

[[Functional description]]

This service call initializes the variable-size memory pool indicated by mplid.
This service call can be issued only from task context. It cannot be issued from non-task context.

[[Example program statement]]

<<Example statement in C language>>

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task1(void)
  {
        :
      vrst_mpl( ID_mpl1 );
        :
  }
```

<<Example statement in assembly language>>

```
     .include mr308.inc
     .GLB        task
task:
           :
     PUSHM       A0
     vrst_mpl    #ID_MPL1
           :
```

Chapter 2 Stack Size Calculation Method

2.1. Stack Size Calculation Method

The MR308 provides two kinds of stacks: the system stack and the user stack. The stack size calculation method differ between the stacks.

- User stack

  This stack is provided for each task. Therefore, writing an application by using the MR308 requires to allocate the stack area for each stack.

- System stack

  This stack is used inside the MR308 or during the execution of the handler.
  When a task issues a service call, the MR308 switches the user stack to the system stack. (See Figure 2.1:System Stack and User Stack)
  The system stack uses interrupt stack(ISP).

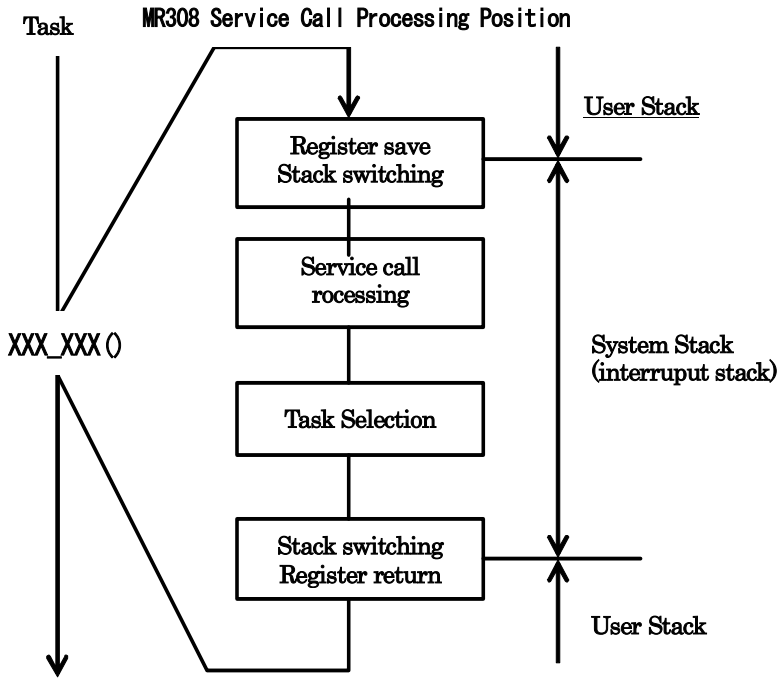


Figure 2.1:System Stack and User Stack

The sections of the system stack and user stack each are located in the manner shown below. However, the diagram shown below applies to the case where the stack areas for all tasks are located in the stack section during configuration.

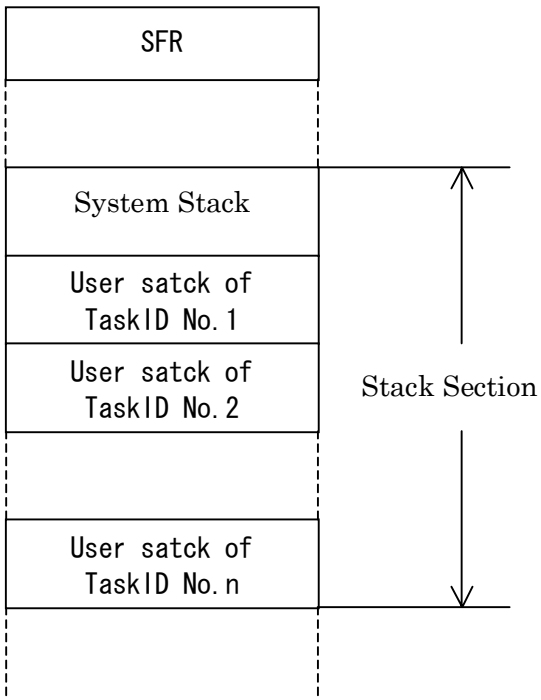


Figure 2.2: Layout of Stacks

2.1.1. User Stack Calculation Method

User stacks must be calculated for each task. The following shows an example for calculating user stacks in cases when an application is written in the C language and when an application is written in the assembly language.

- When an application is written in the C language

  Using the stack size calculation utility STK Viewer[7], calculate the stack size of each task. The necessary stack size of a task is the sum of the stack size output by STK Viewer plus a context storage area of 30 bytes[8] The following shows how to calculate a stack size using

- When an application is written in the assembly language

  - Sections used in user program
    The necessary stack size of a task is the sum of the stack size used by the task in subroutine call plus the size used to save registers to a stack in that task.
  - **Sections used in MR308**
    The sections used in MR308 refer to a stack size that is used for the service calls issued. MR308 requires that if you issue only the service calls that can be issued from tasks, 6 bytes of area be allocated for storing the PC and FLG registers. Also, if you issue the service calls that can be issued from both tasks and handlers, see the stack sizes listed in Table 2.2 to ensure that the necessary stack area is allocated.
    Furthermore, when issuing multiple service calls, include the maximum value of the stack sizes used by those service calls as the sections used by MR308 as you calculate the necessary stack size.

Therefore,
User stack size =
Sections used in user program + **registers used** + **Sections used in MR308**
(registers used is total size of used registers. If you used R0,R1,R2 and R3, add by 2bytes. If you used A0,A1,SB and FB, add by 4bytes.)
Figure 2.3:Example of Use Stack Size Calculation shows an example for calculating a user stack. In the example below, the registers used by the task are R0, R1, and A0.

[7] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA.
[8] If written in the C language, this size is fixed.

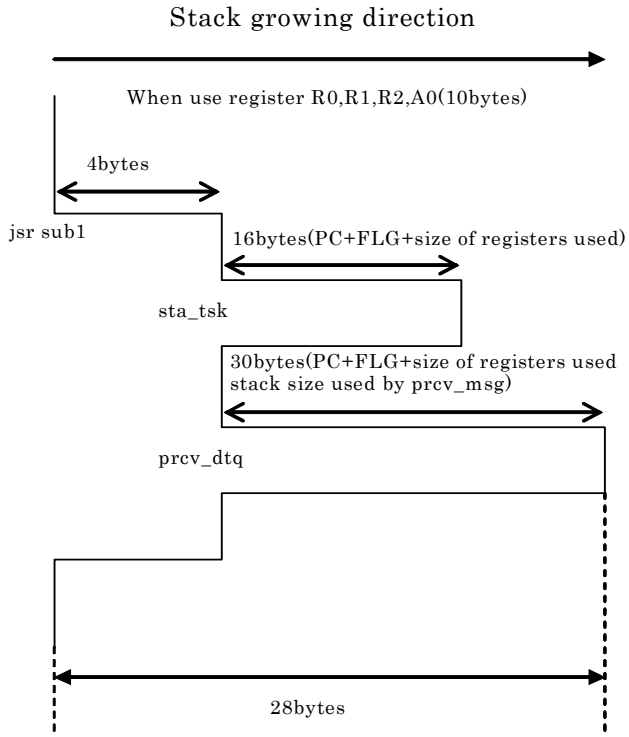


Figure 2.3:Example of Use Stack Size Calculation

2.1.2. System Stack Calculation Method

The system stack is most often consumed when an interrupt occurs during service call processing followed by the occurrence of multiple interrupts.[9] The necessary size (the maximum size) of the system stack can be obtained from the following relation:

$$\text{Necessary size of the system stack} = \alpha\Sigma\beta i(\ \gamma)$$

- α

  The maximum system stack size among the service calls to be used.[10].
  When sta_tsk, ext_tsk, and dly_tsk are used for example, according to the Table 2.1,each of system stack size is the following.

Service Call name	System Stack Size
sta_tsk	4bytes
ext_tsk	0bytes
slp_tsk	4bytes
dly_tsk	8bytes

  Therefore,the maximum system stack size among the service calls to be used is the 8 bytes of dly_tsk.

- βi

  The stack size to be used by the interrupt handler.[11] The details will be described later.

- γ

  Stack size used by the system clock interrupt handler. This is detailed later.

[9] After switchover from user stack to system stack
[10] Refer from Table 2.1 to Table 2.3 for the system stack size used for each individual system call.
[11] OS-dependent interrupt handler (not including the system clock interrupt handler here) and OS-independent interrupt handler.

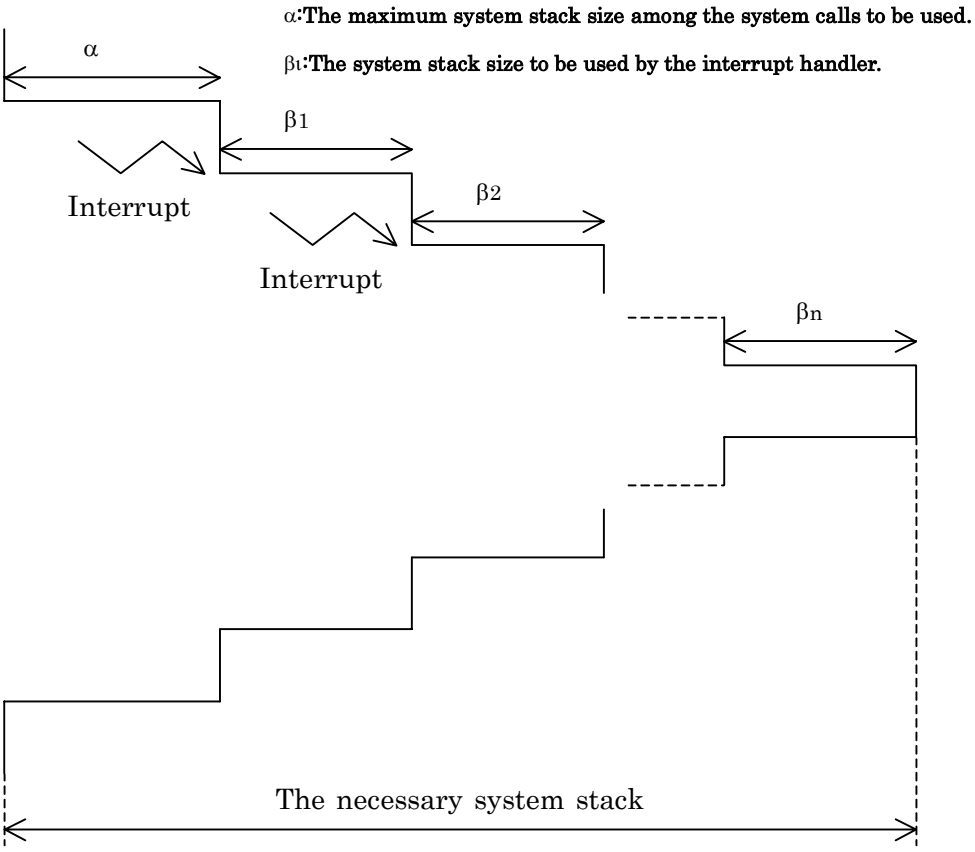


Figure 2.4: System Stack Calculation Method

[(Stack size βi used by interrupt handlers)]

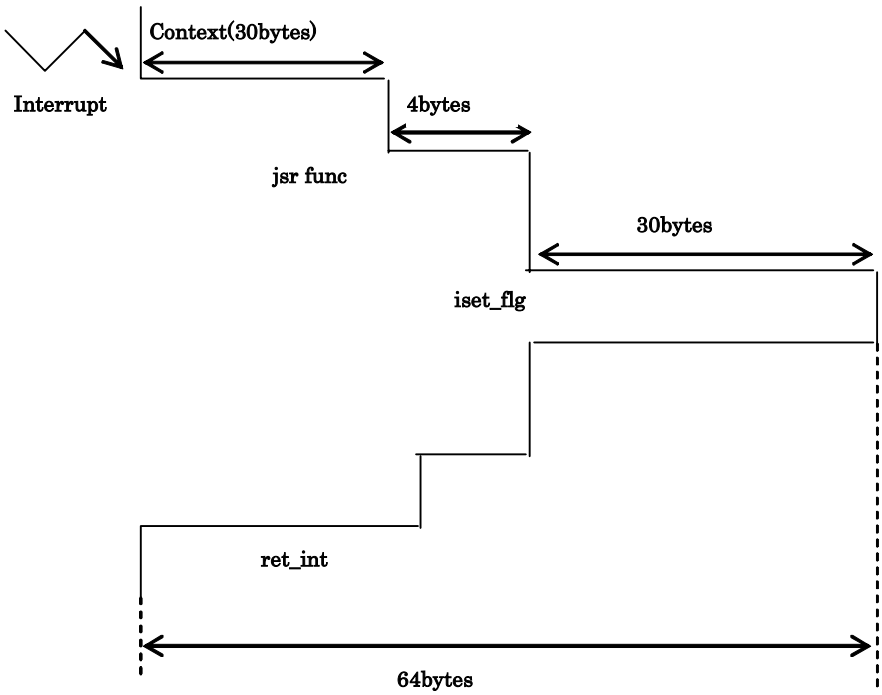
The stack size used by an interrupt handler that is invoked during a service call can be calculated by the equation below.
The stack size βi used by an interrupt handler is shown below.

C language
Using the stack size calculation utility STK Viewer[12], calculate the stack size of each interrupt handler.
Refer to the manual of STK Viewer for detailed use of STK Viewer.

Assembly language
The stack size to be used by OS-dependent interrupt handler
= **register to be used** + **user size** + **stack size to be used by service call**

The stack size to be used by OS-independent interrupt handler
= **register to be used** + **user size**

User size is the stack size of the area written by user.



Context: 30 bytes when written in C language.
When written in assembly language,
Context = size of registers used + 6(PC+FLG)bytes

Figure 2.5: Stack size to be used by Kernel Interrupt Handler

[12] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA..

[(System stack size γ used by system clock interrupt handler)]

When you do not use a system timer, there is no need to add a system stack used by the system clock interrupt handler.

The system stack size γ used by the system clock interrupt handler is whichever larger of the two cases below:

42 **+** maximum size used by cyclic handler
42 **+** maximum size used by alarm handler

C language

Using the stack size calculation utility STK Viewer [13], calculate the stack size of each Alarm or Cyclic handler.

Refer to the manual of STK Viewer for detailed use of STK Viewer.

Assembly language

The stack size to be used by Alarm or Cyclic handler
= register to be used + **user size** + **stack size to be used by service call**

If neither cyclic handler nor alarm handler is used, then

γ = 30bytes

When using the interrupt handler and system clock interrupt handler in combination, add the stack sizes used by both.

[13] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA.

2.2. Necessary Stack Size

Table 2.1 lists the stack sizes (system stack) used by service calls that can be issued from tasks.

Table 2.1 Stack Sizes Used by System Calls Issued from Tasks (in bytes)

Service Call	Stack Size		Service Call	Stack Size	
	User Stack	System Stack		User Stack	System Stack
act_tsk	0	4	rcv_mbx	0(8*)	34
can_act	14	0	prcv_mbx	26(8*)	0
sta_tsk	0	4	trcv_mbx	0(8*)	38
ext_tsk	0	4	ref_mbx	14	0
ter_tsk	0	8	get_mpf	0(8*)	34
chg_pri	0	36	pget_mpf	26(8*)	0
get_pri	14(8*)	0	tget_mpf	0(8*)	38
ref_tsk	30	0	rel_mpf	0	8
ref_tst	14	0	ref_mpf	14	0
slp_tsk	0	4	pget_mpl	0(8*)	40
tslp_tsk	0	8	rel_mpl	0	76
wup_tsk	0	4	ref_mpl	18	0
can_wup	14	0	set_tim	14	0
rel_wai	0	8	get_tim	14	0
sus_tsk	0	4	sta_cyc	14	0
rsm_tsk	0	4	stp_cyc	14	0
frsm_tsk	0	4	ref_cyc	14	0
dly_tsk	0	8	sta_alm	14	0
sig_sem	0	4	stp_alm	14	0
wai_sem	0	34	ref_alm	14	0
pol_sem	14	0	rot_rdq	0	0
twai_sem	0	36	get_tid	14(8*)	0
ref_sem	14	0	loc_cpu	6	0
set_flg	0	8	unl_cpu	0	0
clr_flg	14	0	ref_ver	18	0
wai_flg	0(6*)	34	vsnd_dtq	0	34
pol_flg	14(6*)	0	vpsnd_dtq	0	8
twai_flg	0(10*)	34	vtsnd_dtq	0(8*)	38
ref_flg	14	0	vfsnd_dtq	0	8
snd_dtq	0	34	vrcv_dtq	0(6*)	8
psnd_dtq	0	8	vprcv_dtq	0(6*)	8
tsnd_dtq	0(8*)	38	vtrcv_dtq	0(6*)	8
fsnd_dtq	0	8	vref_dtq	14	0
rcv_dtq	0(8*)	8	vrst_dtq	0	30
prcv_dtq	0(8*)	8	vrst_vdtq	0	30
trcv_dtq	0(8*)	8	vrst_mbx	14	0
ref_dtq	14	0	vrst_mpf	0	30
snd_mbx	0	28	vrst_mpl	58	0
dis_dsp	0	0	ena_dsp	0	0

*: Stack sizes used by service call in C programs.

Table 2.2 lists the stack sizes (system stack) used by service calls that can be issued from handlers.

Table 2.2 Stack Sizes Used by System Calls Issued from Handlers (in bytes)

Service Call	Stack Size	Service Call	Stack Size
iact_tsk	20	iprcv_mbx	26(34*)
ican_act	14	iref_mbx	14
ista_tsk	20	ipget_mpf	26(34*)
ichg_pri	50	irel_mpf	28
iget_pri	14(22*)	iref_mpf	14
iref_tsk	30	iset_tim	14
iref_tst	14	iget_tim	14
iwup_tsk	22	ista_cyc	14
ican_wup	14	istp_cyc	14
irel_wai	22	iref_cyc	14
isus_tsk	18	ista_alm	14
irsm_tsk	18	istp_alm	14
ifrsm_tsk	18	iref_alm	14
isig_sem	26	irot_rdq	18
ipol_sem	14	iget_tid	14(22*)
iref_sem	14	iloc_cpu	6
iset_flg	30	iunl_cpu	14
iclr_flg	14	ret_int	0
ipol_flg	14(20*)	iref_ver	18
iref_flg	14	vipsnd_dtq	28
ipsnd_dtq	28	vifsnd_dtq	28
ifsnd_dtq	28	viprcv_dtq	28(34*)
iprcv_dtq	30(34*)	viref_dtq	14
iref_dtq	14	isnd_mbx	48
iref_mpl	18		

*: Stack sizes used by service call in C programs.

Table 2.3 lists the stack sizes (system stack) used by service calls that can be issued from both tasks and handlers. If the service call issued from task, system uses user stack. If the service call issued from handler, system uses system stack.

Table 2.3 Stack Sizes Used by System Calls Issued from Tasks and Handlers (in bytes)

Service Call	Stack Size	Service Call	Stack Size
sns_ctx	14	sns_loc	14
sns_dsp	14	sns_dpn	14

Chapter 3 Appendix

3.1. List of Service Call

Task Management Function Service Call

Service Call name		Function
act_tsk	[S]	Activate task
iact_tsk	[S]	Activate task
can_act	[S]	Cancel task activation request
ican_act	[S]	Cancel task activation request (handler only)
sta_tsk		Activate task with a start code
ista_tsk		Activate task with a start code (handler only)
ext_tsk	[S]	Terminates invoking task
ter_tsk	[S]	Terminate task
chg_pri	[S]	Change task priority
ichg_pri		Change task priority (handler only)
get_pri	[S]	Reference task priority
iget_pri		Reference task priority (handler only)
ref_tsk		Reference task status
iref_tsk		Reference task status (handler only)
ref_tst		Reference task status (simplified version)
iref_tst		Reference task status (simplified version, handler only)

Task Dependent Synchronization Function Service Call

Service Call name		Function
slp_tsk	[S]	Put task to sleep
tslp_tsk	[S]	Put task to sleep(with timeout)
wup_tsk	[S]	Wakeup task
iwup_tsk	[S]	Wakeup task (handler only)
can_wup	[S]	Cancel wakeup request
ican_wup		Cancel wakeup request (handler only)
rel_wai	[S]	Release task from waiting
irel_wai	[S]	Release task from waiting (handler only)
sus_tsk	[S]	Suspend task
isus_tsk		Suspend task (handler only)
rsm_tsk	[S]	Resume suspended task
irsm_tsk		Resume suspended task (handler only)
frsm_tsk	[S]	Forcibly resume suspended task
ifrsm_tsk		Forcibly resume suspended task (handler only)
dly_tsk	[S]	Delay task

Semaphore

Service Call name		Function
sig_sem	[S]	Release semaphore resource
isig_sem	[S]	Release semaphore resource (handler only)
wai_sem	[S]	Acquire semaphore resource
twai_sem	[S]	Acquire semaphore resource (with timeout)
pol_sem	[S]	Acquire semaphore resource (polling)
ipol_sem		Acquire semaphore resource (polling, handler only)
ref_sem		Reference semaphore status
iref_sem		Reference semaphore status (handler only)

Eventflag

Service Call name		Function
set_flg	[S]	Set eventflag
iset_flg	[S]	Set eventflag (handler only)
clr_flg	[S]	Clear eventflag
iclr_flg		Clear eventflag (handler only)
wai_flg	[S]	Wait for eventflag
twai_flg	[S]	Wait for eventflag (with timeout)
pol_flg	[S]	Wait for eventflag (polling)
ipol_flg		Wait for eventflag (polling, handler only)
ref_flg		Reference eventflag status
iref_flg		Reference eventflag status (handler only)

Data Queue

Service Call name		Function
snd_dtq	[S]	Send to data queue
psnd_dtq	[S]	Send to data queue (polling)
ipsnd_dtq	[S]	Send to data queue (polling, handler only)
tsnd_dtq	[S]	Send to data queue (with timeout)
fsnd_dtq	[S]	Forced send to data queue
ifsnd_dtq	[S]	Forced send to data queue (handler only)
rcv_dtq	[S]	Receive from data queue
prcv_dtq	[S]	Receive from data queue (polling)
iprcv_dtq		Receive from data queue (polling, handler only)
trcv_dtq	[S]	Receive from data queue (with timeout)
ref_dtq		Reference data queue status
iref_dtq		Reference data queue status (handler only)

Mailbox

Service Call name		Function
snd_mbx	[S]	Send to mailbox
isnd_mbx		Send to mailbox (handler only)
rcv_ mbx	[S]	Receive from mailbox
trcv_ mbx	[S]	Receive from mailbox (with timeout)
prcv_ mbx	[S]	Receive from mailbox (polling)
iprcv_mbx		Receive from mailbox (polling, handler only)
ref_mbx		Reference mailbox status
iref_mbx		Reference mailbox status (handler only)

Fixed-size memory pool

Service Call name		Function
get_mpf	[S]	Aquire fixed-size memory block
tget_mpf	[S]	Aquire fixed-size memory block (with timeout)
pget_mpf	[S]	Aquire fixed-size memory block (polling)
ipget_mpf		Aquire fixed-size memory block (polling, handler only)
rel_mpf	[S]	Release fixed-size memory block
irel_mpf		Release fixed-size memory block (handler only)
ref_mpf		Reference fixed-size memory pool status
ief_mpf		Reference fixed-size memory pool status (handler only)

Variable-size Memory Pool

Service Call name	Function
pget_npl	Aquire variable-size memory block (polling)
rel_mpl	Release variable-size memory block
ref_mpl	Reference variable-size memory pool status
iref_mpl	Reference variable-size memory pool status (handler only)

Time Management Function

Service Call name		Function
set_tim	[S]	Set system time
iset_tim		Set system time (handler only)
get_tim	[S]	Reference system time
iget_tim		Reference system time (handler only)
isig_tim	[S]	Supply a time tick (handler only) (This function is built into the kernel)

Cyclic Handler

Service Call name		Function
sta_cyc	[S]	Start cyclic handler operation
ista_cyc		Start cyclic handler operation (handler only)
stp_cyc	[S]	Stops cyclic handler operation
stp_cyc		Stops cyclic handler operation (handler only)
ref_cyc		Reference cyclic handler status
iref_cyc		Reference cyclic handler status (handler only)

Alarm Handler

Service Call name	Function
sta_ alm	Start alarm handler operation
ista_ alm	Start alarm handler operation (handler only)
stp_ alm	Stop alarm handler operation
stp_ alm	Stop alarm handler operation (handler only)
ref_ alm	Reference alarm handler status
iref_ alm	Reference alarm handler status (handler only)

System Status Management Function

Service Call name		Function
rot_rdq	[S]	Rotate task precedence
irot_rdq	[S]	Rotate task precedence (handler only)
get_tid	[S]	Reference task ID in the RUNNING state
iget_tid	[S]	Reference task ID in the RUNNING state (handler only)
loc_cpu	[S]	Lock the CPU
iloc_cpu	[S]	Lock the CPU (handler only)
unl_cpu	[S]	Unlock the CPU
iunl_cpu	[S]	Unlock the CPU (handler only)
dis_dsp	[S]	Disable dispatching
ena_dsp	[S]	Enable dispatching
sns_ctx	[S]	Reference context
sns_loc	[S]	Reference CPU state
sns_dsp	[S]	Reference dispatching state
sns_dpn	[S]	Reference dispatching pending state

Interrupt Management Function

Service Call name	Function
ret_int	Returns from an interrupt handler (when written in assembly language)

System Configuration Management Function

Service Call name	Function
ref_ver	Reference version information
iref_ver	Reference version information (handler only)

Short Data Queue

Service Call name	Function
vsnd_dtq	Send to short data queue
vpsnd_dtq	Send to short data queue (polling)
vipsnd_dtq	Send to short data queue (polling, handler only)
vtsnd_dtq	Send to short data queue (with timeout)
vfsnd_dtq	Forced send to short data queue
vifsnd_dtq	Forced send to short data queue (handler only)
vrcv_dtq	Receive from short data queue
vprcv_dtq	Receive from short data queue (polling)
viprcv_dtq	Receive from short data queue (polling, handler only)
vtrcv_dtq	Receive from short data queue (with timeout)
vref_dtq	Reference short data queue status
viref_dtq	Reference short data queue status (handler only)

Extended Function

Service Call name	Function
vrst_dtq	Clear data queue area
vrst_vdtq	Clear short data queue area
vrst_mbx	Clear mailbox area
vrst_mpf	Clear fixed-size memory pool area
vrst_mpl	Clear variable-size memory pool area

3.2. List of Error code

Error code	Value	Description
E_OK	0	Terminated normally
E_ILUSE	-28	Service call improperly used
E_OBJ	-41	Object status invalid
E_QOVR	-43	Queuing or nest overflow
E_TMOUT	-50	Polling failed or timeout
E_RLWAI	-49	Forced release from waiting
EV_RST	-254	Released from WAITING state by clearing

3.3. Data type

```
typedef    signed char      B;          /* Signed 8-bit integer */
typedef    signed short     H;          /* Signed 16-bit integer */
typedef    signed long      W;          /* Signed 32-bit integer */
typedef    unsigned char    UB;         /* Unsigned 8-bit integer */
typedef    unsigned short   UH;         /* Unsigned 16-bit integer */
typedef    unsigned long    UW;         /* Unsigned 32-bit integer */
typedef    char             VB          /* One whose data type does not match, signed
                                           (8 bits in size) */
typedef    short            VH;         /* One whose data type does not match, signed
                                           (16 bits in size) */
typedef    long             VW;         /* One whose data type does not match, signed
                                           (32 bits in size) */
typedef    void             *VP;        /* Pointer to one whose data type does not match */
typedef    void             (*FP)();    /* Start address of program, general */
typedef    W                INT         /* Signed 32-bit integer */
typedef    UW               UINT;       /* Unsigned 32-bit integer */
typedef    W                FN          /* Function code */
typedef    H                ID;         /* Object ID number */
typedef    H                PRI;        /* Task priority */
typedef    W                TMO;        /* Timeout */
typedef    W                ER;         /* Error code (signed integer) */
typedef    UW               ATR;        /* Object attribute (unsigned integer) */
typedef    UW               STAT;       /* Task status */
typedef    UW               MODE;       /* Operation mode of service call */
typedef    UW               SIZE;       /* Size of memory area */
typedef    UW               RELTIM      /* Relative time */
typedef    W                VP_INT;     /* Pointer to one whose data type is indeterminate
                                           or signed integer in sizes natural to processor */
typedef    struct           systim{     /* System time */
           UH               utime;      /* 16 high-order bits of time */
           UW               ltimer;     /* 16 low-order bits of time */
} SYSTIM;
typedef    W                ER_ID;      /* Error code or ID */
typedef    W                ER_UINT;    /* Error code or unsigned integer */
```

3.4. Common Constants and Packet Format of Structure

```
----Common formats----
TRUE            1               /* True */
FALSE           0               /* False */
----Formats related to task management----
TSK_SELF        0               /* Specifies the issuing task itself */
TPRI_RUN        0               /* Specifies priority of task being executed then */
typedef struct t_rtsk {
        STAT    tskstat;        /* Task status */
        PRI     tskpri;         /* Current priority of task */
        PRI     tskbpri;        /* Base priority of task */
        STAT    tskwait;        /* Reason for which task is kept waiting */
        ID      wid;            /* Object ID for which task is kept waiting */
        TMO     tskatr;         /* Remaining time before task times out */
        UINT    actcnt;         /* Number of activation requests */
        UINT    wupcnt;         /* Number of wakeup requests */
        UINT    suscnt;         /* Number of suspension requests */
} T_RTSK;
typedef struct t_rtst {
        STAT    tskstat;        /* Task status */
        STAT    tskwait;        /* Reason for which task is kept waiting */
} T_RTST;
----Formats related to semaphore----
typedef struct t_rsem {
        ID      wtskid;         /* ID number of task at the top of waiting queue */
        INT     semcnt;         /* Current semaphore count value */
} T_RSEM;
----Formats related to eventflag----
wfmod:
        TWF_ANDW    H'0000      /* AND wait */
        TWF_ORW     H'0002      /* OR wait */
typedef struct t_rflg {
        ID      wtskid;         /* ID number of task at the top of waiting queue */
        UINT    flgptn;         /* Current bit pattern of eventflag */
} T_RFLG;
----Formats related to data queue and short data queue----
typedef struct t_rdtq {
         ID     stskid;         /* ID number of task at the top of transmission waiting queue */
         ID     rtskid;         /* ID number of task at the top of reception waiting queue */
         UINT   sdtqcnt;        /* Number of data bytes contained in data queue */
} T_RDTQ;
----Formats related to mailbox----
typedef struct   t_msg   {
        VP      msghead;        /* Message header */
} T_MSG;
typedef struct t_msg_pri {
        T_MSG   msgque;         /* Message header */
        PRI     msgpri;         /* Message priority */
} T_MSG_PRI;
typedef struct t_mbx {
        ID      wtskid;         /* ID number of task at the top of waiting queue */
        T_MSG   *pk_msg;        /* Next message to be received */
} T_RMBX;
----Formats related to fixed-size memory pool----
typedef struct t_rmpf {
        ID      wtskid;         /* ID number of task at the top of memory acquisition waiting queue */
        UINT    frbcnt;         /* Number of memory blocks */
} T_RMPF;
```

```
----Formats related to Variable-size Memory pool----
typedef struct t_rmpl {
        ID          wtskid;         /* ID number of task at the top of memory acquisition waiting queue */
        SIZE        fmplsz;         /* Total size of free areas */
        UINT        fblksz;         /* Maximum memory block size that can be acquired immediately */
} T_RMPL;
----Formats related to cyclic handler----
typedef struct t_rcyc {
        STAT        cycstat;        /* Operating status of cyclic handler */
        RELTIM      lefttim;        /* Remaining time before cyclic handler starts */
} T_RCYC;
----Formats related to alarm handler----
typedef struct t_ralm {
        STAT        almstat;        /* Operating status of alarm handler */
        RELTIM      lefttim;        /* Remaining time before alarm handler starts */
} T_RALM;
----Formats related to system management----
typedef struct t_rver {
        UH          maker;          /* Maker */
        UH          prid;           /* Type number */
        UH          spver;          /* Specification version */
        UH          prver;          /* Product version */
        UH          prno[4];        /* Product management information */
} T_RVER;
```

3.5. Assembly Language Interface

When issuing a service call in the assembly language, you need to use macros prepared for invoking service calls.

Processing in a service call invocation macro involves setting each parameter to registers and starting execution of a service call routine by a software interrupt. If you issue service calls directly without using a service call invocation macro, your program may not be guaranteed of compatibility with future versions of MR308.

The table below lists the assembly language interface parameters. The values set forth in μITRON specifications are not used for the function code.

Task Management Function

ServiceCall	INTNo.	Parameter					ReturnParameter	
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0	A0
ista_tsk	62	10	stacd	stacd	tskid	-	ercd	-
sta_tsk	63	8	stacd	stacd	tskid	-	ercd	-
act_tsk	63	0	-	-	tskid	-	ercd	-
iact_tsk	62	2	-	-	tskid	-	ercd	-
ter_tsk	63	14	-	-	tskid	-	ercd	-
can_act	62	4	-	-	tskid	-	actcnt	-
ican_act	62	6	-	-	tskid	-	actcnt	-
chg_pri	63	16	-	tskpri	tskid	-	ercd	-
ichg_pri	62	18	-	tskpri	tskid	-	ercd	-
rel_wai	63	44	-	-	tskid	-	ercd	-
irel_wai	62	46	-	-	tskid	-	ercd	-
ref_tst	62	28	-	-	tskid	pk_rtst	ercd	-
iref_tst	62	30	-	-	tskid	pk_rtst	ercd	-
ref_tsk	62	24	-	-	tskid	pk_rtsk	ercd	-
iref_tsk	62	26	-	-	tskid	pk_rtsk	ercd	-
ext_tsk	58	106	-	-	-	-	-	-
get_pri	62	20	-	-	tskid	-	ercd	tskpri
iget_pri	62	22	-	-	tskid	-	ercd	tskpri

Task Dependent Synchronization Function

ServiceCall	INTNo.	Parameter					ReturnParameter
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0
slp_tsk	63	32	-	-	-	-	ercd
wup_tsk	63	36	-	-	tskid	-	ercd
iwup_tsk	62	38	-	-	tskid	-	ercd
can_wup	62	40	-	-	tskid	-	wupcnt
ican_wup	62	42	-	-	tskid	-	wupcnt
tslp_tsk	63	34	tmout	tmout	-	-	ercd
sus_tsk	63	48	-	-	tskid	-	ercd
isus_tsk	62	50	-	-	tskid	-	ercd
rsm_tsk	63	52	-	-	tskid	-	ercd
irsm_tsk	62	54	-	-	tskid	-	ercd
frsm_tsk	63	56	-	-	tskid	-	ercd
ifrsm_tsk	62	58	-	-	tskid	-	ercd
dly_tsk	63	60	tmout	tmout	-	-	ercd

Synchronization & Communication Function

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
wai_sem	63	66	-	-	-	semid	-	ercd	-	-	-
pol_sem	62	68	-	-	-	semid	-	ercd	-	-	-
ipol_sem	62	70	-	-	-	semid	-	ercd	-	-	-
sig_sem	63	62	-	-	-	semid	-	ercd	-	-	-
isig_sem	62	64	-	-	-	semid	-	ercd	-	-	-
twai_sem	63	72	tmout	-	tmout	semid	-	ercd	-	-	-
ref_sem	62	74	-	-	-	semid	pk_rsem	ercd	-	-	-
iref_sem	62	76	-	-	-	semid	pk_rsem	ercd	-	-	-
wai_flg	63	86	wfmode	-	waiptn	flgid	-	ercd	-	flgptn	-
twai_flg	55	tmout	wfmode	tmout	waiptn	flgid	92	ercd	-	flgptn	-
pol_flg	62	88	wfmode	-	waiptn	flgid	-	ercd	-	flgptn	-
ipol_flg	62	90	wfmode	-	waiptn	flgid	-	ercd	-	flgptn	-
set_flg	63	78	-	-	setptn	flgid	-	ercd	-	-	-
iset_flg	62	80	-	-	setptn	flgid	-	ercd	-	-	-
ref_flg	62	94	-	-	-	flgid	pk_rflg	ercd	-	-	-
iref_flg	62	96	-	-	-	flgid	pk_rflg	ercd	-	-	-
clr_flg	62	82	-	-	clrptn	flgid	-	ercd	-	-	-
iclr_flg	62	84	-	-	clrptn	flgid	-	ercd	-	-	-
snd_dtq	63	98	data	-	data	dtqid	-	ercd	-	-	-
psnd_dtq	63	100	data	-	data	dtqid	-	ercd	-	-	-
ipsnd_dtq	62	102	data	-	data	dtqid	-	ercd	-	-	-
fsnd_dtq	63	106	data	-	data	dtqid	-	ercd	-	-	-
ifsnd_dtq	62	108	data	-	data	dtqid	-	ercd	-	-	-
tsnd_dtq	55	tmout	data	tmout	data	dtqid	104	ercd	-	-	-

Synchronization & Communication Function

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
rcv_dtq	63	110	-	-	-	dtqid	-	ercd	data	-	data
prcv_dtq	63	112	-	-	-	dtqid	-	ercd	data	-	data
iprcv_dtq	62	114	-	-	-	dtqid	-	ercd	data	-	data
trcv_dtq	63	116	tmout	-	tmout	dtqid	-	ercd	data	-	data
ref_dtq	62	118	-	-	-	dtqid	pk_rdtq	ercd	-	-	-
iref_dtq	62	120	-	-	-	dtqid	pk_rdtq	ercd	-	-	-
snd_mbx	63	122	-	-	-	mbxid	pk_msg	ercd	-	-	-
isnd_mbx	62	124	-	-	-	mbxid	pk_msg	ercd	-	-	-
rcv_mbx	63	126	-	-	-	mbxid	-	ercd	pk_msg	pk_msg	-
prcv_mbx	62	128	-	-	-	mbxid	-	ercd	pk_msg	pk_msg	-
iprcv_mbx	62	130	-	-	-	mbxid	-	ercd	pk_msg	pk_msg	-
trcv_mbx	63	132	tmout	-	tmout	mbxid	-	ercd	pk_msg	pk_msg	-
ref_mbx	62	134	-	-	-	mbxid	pk_rmbx	ercd	-	-	-
iref_mbx	62	136	-	-	-	mbxid	pk_rmbx	ercd	-	-	-

System Management Functions

ServiceCall	INTNo.	Parameter		ReturnParameter	
		FuncCode R0	R3	R0	A0
rot_rdq	63	190	tskpri	ercd	--
irot_rdq	62	192	tskpri	ercd	--
get_tid	62	194	--	ercd	tskid
iget_tid	62	196	--	ercd	tskid
loc_cpu	59	198	--	ercd	--
iloc_cpu	59	200	--	ercd	--
dis_dsp	60	206	--	ercd	--
ena_dsp	63	208	--	ercd	--
unl_cpu	63	202	--	ercd	--
iunl_cpu	62	204	--	ercd	--
sns_ctx	62	210	--	ercd	--
sns_loc	62	212	--	ercd	--
sns_dsp	62	214	--	ercd	--
sns_dpn	62	216	--	ercd	--

Interrupt Management Functions

ServiceCall	INTNo.	Parameter	ReturnParameter
		FuncCode R0	R0
ret_int	61	-	-

Memorypool Management Functions

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
get_mpf	63	140	-	-	-	mpfid	-	ercd	p_blk	-	p_blk
pget_mpf	62	138	-	-	-	mpfid	-	ercd	p_blk	-	p_blk
ipget_mpf	62	246	-	-	-	mpfid	-	ercd	p_blk	-	p_blk
tget_mpf	63	142	tmout	-	tmout	mpfid	-	ercd	p_blk	-	p_blk
rel_mpf	63	144	blk	-	blk	mpfid	-	ercd	-	-	-
irel_mpf	62	146	blk	-	blk	mpfid	-	ercd	-	-	-
ref_mpf	62	148	-	-	-	mpfid	pk_rmpf	ercd	-	-	-
iref_mpf	62	150	-	-	-	mpfid	pk_rmpf	ercd	-	-	-
pget_mpl	63	152	blksz	-	-	mplid	-	ercd	p_blk	-	p_blk
rel_mpl	63	154	blk	-	blk	mplid		ercd	-	-	-
ref_mpl	62	156	-	-	-	mplid	pk_rmpl	ercd	-	-	-
iref_mpl	62	262	-	-	-	mplid	pk_rmpl	ercd	-	-	-

Time Management Functions

ServiceCall	INTNo.	Parameter					ReturnParameter
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0
set_tim	62	158	-	-	p_systim	-	ercd
iset_tim	62	160	-	-	p_systim	-	ercd
get_tim	62	162	-	-	p_systim	-	ercd
iget_tim	62	164	-	-	p_systim	-	ercd
sta_cyc	62	166	-	-	cycid	-	ercd
ista_cyc	62	168	-	-	cycid	-	ercd
stp_cyc	62	170	-	-	cycid	-	ercd
istp_cyc	62	172	-	-	cycid	-	ercd
ref_cyc	62	174	-	-	cycid	pk_rcyc	ercd
iref_cyc	62	176	-	-	cycid	pk_rcyc	ercd
sta_alm	62	178	almtim	almtim	almid	-	ercd
ista_alm	62	180	almtim	almtim	almid	-	ercd
stp_alm	62	182	-	-	almid	-	ercd
istp_alm	62	184	-	-	almid	-	ercd
ref_alm	62	186	-	-	almid	pk_ralm	ercd
iref_alm	62	188	-	-	almid	pk_ralm	ercd

System Configuration Management Function

ServiceCall	INTNo.	Parameter		ReturnParameter
		FuncCode R0	A0	R0
ref_ver	62	218	pk_rver	ercd
iref_ver	62	220	pk_rver	ercd

Extended Function(Reset functions)

ServiceCall	INTNo.	Parameter		ReturnParameter
		FuncCode R0	A0	R0
vrst_vdtq	63	256	vdtqid	ercd
vrst_dtq	63	248	dtqid	ercd
vrst_mbx	62	250	mbxid	ercd
vrst_mpf	63	252	mpfid	ercd
vrst_mpl	62	254	mplid	ercd

Extended Function(Short data queue functions)

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
vsnd_dtq	63	222	data	-	-	vdtqid	-	ercd	-	-	-
vpsnd_dtq	63	224	data	-	-	vdtqid	-	ercd	-	-	-
vipsnd_dtq	62	226	data	-	-	vdtqid	-	ercd	-	-	-
vfsnd_dtq	63	230	data	-	-	vdtqid	-	ercd	-	-	-
vifsnd_dtq	62	232	data	-	-	vdtqid	-	ercd	-	-	-
vtsnd_dtq	55	tmout	data	tmout		vdtqid	228	ercd	-	-	-
vrcv_dtq	63	234	-	-	-	vdtqid	-	ercd	data	-	-
vprcv_dtq	63	236	-	-	-	vdtqid	-	ercd	data	-	-
viprcv_dtq	62	238	-	-	-	vdtqid	-	ercd	data	-	-
vtrcv_dtq	63	240	tmout	-	tmout	vdtqid	-	ercd	data	-	-
vref_dtq	62	242	-	-	-	vdtqid	pk_rdtq	ercd	-	-	-
viref_dtq	62	244	-	-	-	vdtqid	pk_rdtq	ercd	-	-	-

Real-time OS for M16C/70,80,M32C/80 Series
M3T-MR308/4 Reference Manual

Publication Date: Nov. 1, 2005 Rev.2.00

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.

Edited by: Application Engineering Department 1
Renesas Solutions Corp.

M3T-MR308/4 V.4.00
Reference Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J1026-0200