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.

Application Note

µITRON Specification OS Transition Manual Transition Guide from MR32R to MR32R/4

Application Note

Renesas Microcomputer Development Environment System

Renesas Electronics
www.renesas.com

Rev.1.00 2005.09

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts 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

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. assumes 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.
3. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. assumes 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 Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. 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.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

1. TRON is an acronym of “The Realtime Operating system Nucleus”, ITRON is an acronym of the “Industrial TRON”, and µITRON is an acronym of the “Micro Industrial TRON”.
2. TRON, ITRON, and µITRON are the names of computer specifications and do not indicate a specific group of the commodity or the commodity.
3. The µITRON4.0 specification and µITRON4.0 protection function extension are open realtime-kernel specifications defined by the TRON association. The specifications of µITRON4.0 and µITRON4.0 protection function extension can be downloaded from the TRON association homepage (http://www.ertl.jp/ITRON/home-e.html).
4. The copyright of the µITRON specification belongs to the TRON association.
5. Microsoft® Windows® 98, Microsoft® Windows® Millennium Edition (Windows® Me) operating system, Microsoft® Windows NT® operating system, Microsoft® Windows® 2000 operating system, and Microsoft® Windows® XP operating system are registered trademarks of Microsoft Corporation in the United States and/or other countries.
6. SuperHTM is a trademark of Renesas Technology Corp..
7 All other product names are trademarks or registered trademarks of the respective holders.

Contents

Section 1 Preface

This guide describes how to shift from the MR32R based on the µITRON3.0 specifications to the MR32R/4 based on the µITRON4.0 specifications.

This guide does not guarantee correct system transition from the MR32R to the MR32R/4.

For detailed specifications, be sure to refer to the manuals of both products.

Section 2 Overview

This section gives an overview of the functions that are deleted, added, or modified with regard to the MR32R/4.

For the details of each function, refer to the respective manual.

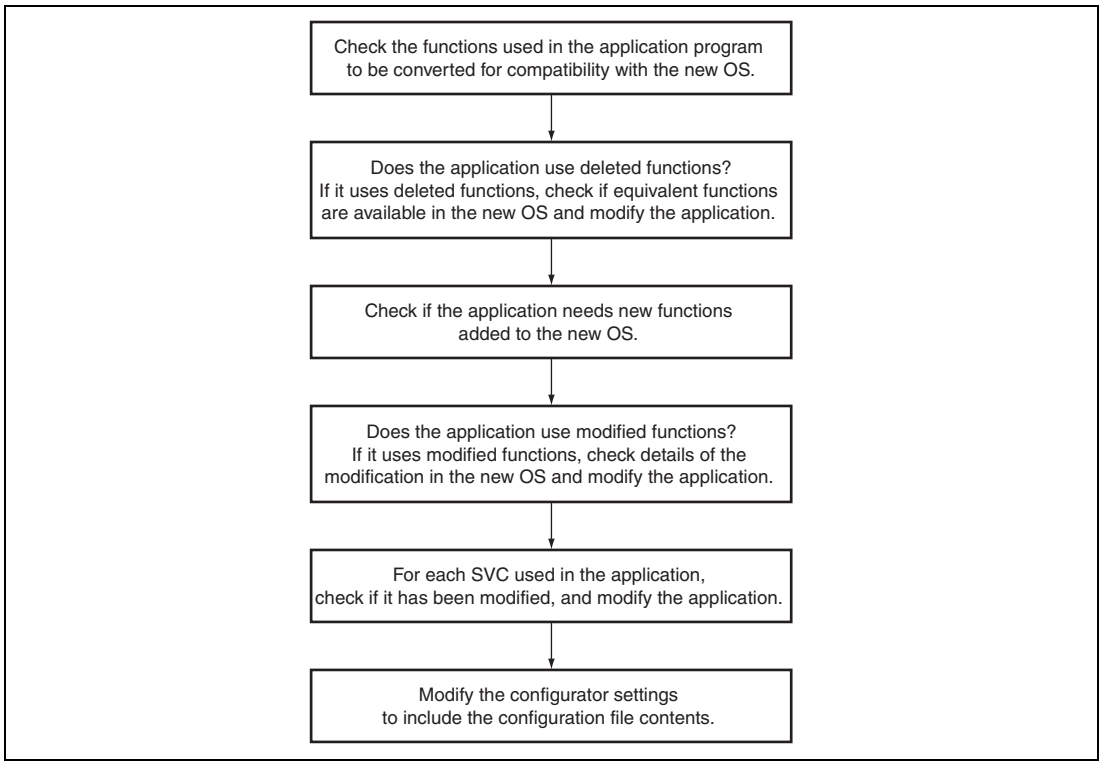

Figure 2.1 Transition Steps

Table 2.1 gives a comparative overview of the MR32R and MR32R/4.

Table 2.1 Comparison between MR32R and MR32R/4

No.	MR32R Function	Modification in MR32R/4	Section in This Guide
1	Extended function (mailbox with priority)	Integrated into ordinary mailbox function	3.1.1
2	Data queue	New function	3.2.1
3	Task exception	New function	3.2.2
4	Multiple task start requests	New function	3.2.3
5	Service calls starting with "i"	Added	3.2.4
6	Dispatch and CPU lock	Meanings of the words are modified	3.3.1
7	Object attribute	New attributes are added to each object	3.3.2
8	wup_tsk, sus_tsk	Current task can be specified	3.3.3
9	Event flag	Clearing specification is modified	3.3.4
10	Mailbox	New message header type is added	3.3.5
11	Rendezvous	Timeout specification is modified	3.3.6
12	Time management	Time unit and management are modified	3.3.7
13	Cyclic handler	Modification made throughout the function	3.3.8
14	Alarm handler	Modification made throughout the function	3.3.9

Section 3 Added, Deleted, and Modified Functions

3.1 Deleted Functions

3.1.1 Extended Function (Mailbox with Priority)

The extended function (mailbox with priority) is deleted from the MR32R/4.

This function is now included in the synchronous and communication function (mailbox) and equivalent operation is available.

3.1.2 Other Functions

Table 3.1 shows the MR32R/4 substitute means for the functions deleted from the MR32R.

Table 3.1 Correspondence to MR32R/4 Functions

No.	SVC	Function	Substitute Means
1	def_exc	Define an exception handler	Equivalent function is available through the task exception processing function
2	vclr_ems	Clear an exception mask	Equivalent function is available through the task exception processing function
3	vset_ems	Set an exception mask	Equivalent function is available through the task exception processing function
4	vras_fex	Forcibly generate an exception	Equivalent function is available through the task exception processing function

3.2 Added Functions

3.2.1 Data Queue

The MR32R/4 has a new function called a data queue.

This function provides synchronization and communication through 1-word message (data) transfer.

Table 3.2 gives an overview of the data queue function.

Table 3.2 Data Queue Function

No.	Function	SVC	API
1	Create a data queue	cre_dtq	ER ercd = cre_dtq(ID dtqid, T_CDTQ *pk_cdtq);
2	Create a data queue (automatically assign ID)	acre_dtq	ER_ID dtqid = acre_dtq(T_CDTQ *pk_cdtq);
3	Delete a data queue	del_dtq	ER ercd = del_dtq(ID dtqid);
4	Send data to a data queue	snd_dtq	ER ercd = snd_dtq(ID dtqid, VP_INT data);
5	Send data to a data queue (polling)	psnd_dtq	ER ercd = psnd_dtq(ID dtqid, VP_INT data);
		ipsnd_dtq	ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);
6	Send data to a data queue (with timeout)	tsnd_dtq	ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
7	Forcibly send data to a data queue	fsnd_dtq	ER ercd = fsnd_dtq(ID dtqid, VP_INT data);
		ifsnd_dtq	ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);
8	Receive data from a data queue	rcv_dtq	ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data);
9	Receive data from a data queue (polling)	prcv_dtq	ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data);
		iprcv_dtq	ER ercd = iprcv_dtq(ID dtqid, VP_INT *p_data);
10	Receive data from a data queue (with timeout)	trcv_dtq	ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);
11	Refer to the data queue state	ref_dtq	ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq);
		iref_dtq	ER ercd = iref_dtq(ID dtqid, T_RDTQ *pk_rdtq);

3.2.2 Task Exception Processing

The MR32R/4 has a new function called task exception processing.

This function processes exceptions generated in tasks.

Table 3.3 gives an overview of the task exception processing.

Table 3.3 Task Exception Processing

No.	Function	SVC	API
1	Define a task exception processing routine	def_tex	ER ercd = def_tex (ID tskid, T_DTEX *pk_dtex);
2	Request task exception processing	ras_tex	ER ercd = ras_tex(ID tskid, TEXPTN rasptn);
		iras_tex	ER ercd = iras_tex(ID tskid, TEXPTN rasptn);
3	Disable task exception processing	dis_tex	ER ercd = dis_tex();
4	Enable task exception processing	ena_tex	ER ercd = ena_tex();
5	Refer to the task exception disable state	sns_tex	BOOL state =sns_tex();
6	Refer to the task exception processing state	ref_tex	ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex);
		iref_tex	ER ercd = iref_tex(ID tskid, T_RTEX *pk_rtex);

3.2.3 Multiple Start Requests for a Task

The MR32R/4 enables multiple start requests to be issued for a task.

By using this function, when act_tsk is issued to a task that has already been started, the attempted request is recorded and the task is automatically started again when the task is terminated.

This function is implemented by the act_tsk service call.

3.2.4 Service Call Names Starting with "i"

The MR32R/4 provides new service calls whose names start with "i". These service calls are dedicated to a non-task context. In a non-task context, only these dedicated service calls should be used. Table 3.4 is a list of these service calls.

Table 3.4 List of Service Calls Dedicated to Non-Task Context

No.	Service Call	Function	No.	Service Call	Function
1	iact_tsk	Start a task	2	ista_tsk	Start a task (specifying start code)
3	ican_act	Cancel start task request	4	ichg_pri	Change the task priority
5	iget_pri	Refer to the task priority	6	iref_tsk	Refer to the task state
7	iref_tst	Refer to the task state (simple version)	8	iwup_tsk	Wake up a task
9	ican_wup	Cancel wakeup task	10	irel_wai	Release WAITING state forcibly
11	isus_tsk	Suspend a task	12	irsm_tsk	Resume a suspended task
13	ifrsm_tsk	Resume a suspended task forcibly	14	iras_tex	Request task exception processing
15	iref_tex	Refer to the task exception processing	16	isig_sem	Release a semaphore resource
17	ipol_sem	Get a semaphore resource (polling)	18	iref_sem	Refer to the semaphore state
19	iset_flg	Set an event flag	20	iclr_flg	Clear an event flag
21	ipol_flg	Wait for an event flag (polling)	22	iref_flg	Refer to the event flag state
23	ipsnd_dtq	Send data to a data queue (polling)	24	ifsnd_dtq	Send data to a data queue forcibly
25	iprcv_dtq	Receive data from a data queue (polling)	26	iref_dtq	Refer to the data queue state
27	isnd_mbx	Send data to a mailbox	28	iprcv_mbx	Receive data from a mailbox (polling)
29	iref_mbx	Refer to the mailbox state	30	iref_drv	Refer to the rendezvous state
31	iref_por	Refer to the rendezvous port state	32	ipget_mpf	Get a fixed-size memory block (polling)
33	irel_mpf	Release a fixed-size memory block	34	iref_mpf	Refer to the fixed-size memory pool state

No.	Service Call	Function	No.	Service Call	Function
35	iref_mpl	Refer to the variable-size memory pool state	36	iset_tim	Set the system clock
37	iget_tim	Refer to the system clock	38	ista_cyc	Start a cyclic handler
39	iref_cyc	Refer to the cyclic handler state	40	ista_alm	Start an alarm handler
41	istp_alm	Stop an alarm handler	42	iref_alm	Refer to the alarm handler state
43	irot_rdq	Rotate the ready queue	44	iget_tid	Refer to the task ID in RUNNING state
45	iloc_cpu	Lock the CPU	46	iunl_cpu	Unlock the CPU
47	idef_int	Define an interrupt handler	48	iref_ver	Refer to the version information

3.3 Modified Functions

3.3.1 Meanings of Dispatch-Disabled State and CPU-Locked State

In the MR32R, the CPU-locked state indicates the state where the dispatch and interrupts are disabled. In the MR32R/4, the CPU-locked state and dispatch-disabled state are managed as independent states.

For example, in the MR32R/4, when the system is in the dispatch-disabled and CPU-locked states, if the CPU-locked state is canceled, the system remains in the dispatch-disabled state.

As another example, in the MR32R/4, after the system enters the CPU-locked state from the dispatch-disabled state, if the CPU-locked state is canceled, the system remains in the dispatch-disabled state. On the other hand, after the system enters the CPU-locked state from the dispatch-enabled state, if the CPU-locked state is canceled, the system enters the dispatch-enabled state.

When shifting to the MR32R/4, note the dispatch-disabled state and CPU-locked state after unl_cpu, loc_cpu, dis_dsp, or ena_dsp execution.

Table 3.5 State Transition by dis_dsp and loc_cpu Service Calls in MR32R

State No.	System State	Current State		Service Call Issued and State No. Entered			
		Interrupt	Dispatch	dis_dsp	ena_dsp	loc_cpu	unl_cpu
1	RUNNING	Enabled	Enabled	→2	→1	→3	→1
2	Dispatch disabled	Enabled	Disabled	→2	→1	→3	→1
3	CPU locked	Disabled	Disabled	E_CTX	E_CTX	→3	→1

Table 3.6 State Transition by dis_dsp and loc_cpu Service Calls in MR32R/4

State No.	System State	Current State		Service Call Issued and State No. Entered			
		CPU Lock	Dispatch	dis_dsp	ena_dsp	loc_cpu	unl_cpu
1	Dispatch enabled	—	Enabled	→2	→1	→4	→3
2	Dispatch disabled	—	Disabled	→2	→1	→4	→3
3	CPU unlocked	Unlocked	—	→2	→1	→4	→3
4	CPU locked	Locked	—	E_CTX	E_CTX	→4	→3

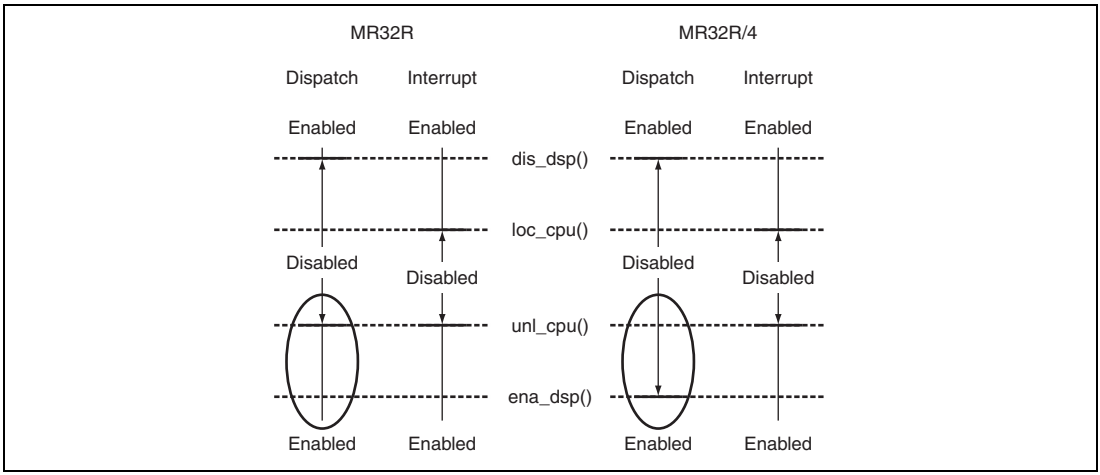

Figure 3.1 Modification of Dispatch-Disabled State and CPU-Locked State

3.3.2 New Attributes for Objects

Attributes are added or deleted for the objects to be created in the service calls shown in the following table.

These attributes are ignored in the MR32R even if they are specified, but in the MR32R/4 the new attributes must be specified.

The following table shows the new attribute names and descriptions together with the related service calls and objects.

Table 3.7 New Attributes and Related Service Calls

Service Call	Object	Attribute	Description
cre_tsk	Task	TA_ACT	After the task has been created, the same processing as act_tsk is performed and the task makes a transition to the READY state
cre_sem	Semaphore	TA_TFIFO (*)	Task wait queue is managed on a FIFO basis
		TA_TPRI	Task wait queue is managed on priority
cre_flg	Event flag	TA_TFIFO (*)	Task wait queue is managed on a FIFO basis
		TA_TPRI	Task wait queue is managed on priority
		TA_WSGL	Does not permit multiple tasks to wait for the event flag
		TA_WMUL (*)	Permits multiple tasks to wait for the event flag
		TA_CLR	Clearing specification
cre_mbx	Mailbox	TA_TFIFO (*)	Task wait queue is managed on a FIFO basis
		TA_TPRI	Task wait queue is managed on priority
		TA_MPRI	Message queue is managed on a FIFO basis
		TA_MFIFO (*)	Message queue is managed on priority
cre_mbf	Message buffer	TA_TFIFO (*)	Task queue waiting for sending is managed on a FIFO basis
		TA_TPRI	Task queue waiting for sending is managed on priority
cre_por	Rendezvous port	TA_TFIFO (*)	Task queue waiting for calling is managed on a FIFO basis
		TA_TPRI	Task queue waiting for calling is managed on priority
cre_mpf	Fixed-size memory pool	TA_TFIFO (*)	Task wait queue is managed on a FIFO basis
		TA_TPRI	Task wait queue is managed on priority
cre_mpl	Variable-size memory pool	TA_TFIFO (*)	Task wait queue is managed on a FIFO basis

Note: * This attribute has the same function as the default attribute in the MR32R.

The following table shows the new attributes that are added in the MR32R/4 but ignored by the kernel and their related service calls.

Table 3.8 Attributes Ignored by Kernel

Service Call	Object	Attribute	Description
cre_tsk	Task	TA_HLNG	Indicates that the application is written in a high-level language
cre_cyc	Cyclic handler		
cre_alm	Alarm handler	TA_ASM	Indicates that the application is written in assembly language
def_inh	Interrupt handler		

The following table shows the attributes deleted from the MR32R/4.

Table 3.9 Deleted Attributes in MR32R/4

Service Call	Object	Attribute	Substitute Means
cre_tsk	Task	__MR_USER	Specify the start address of the stack allocated by the user through the packet variable that indicates the start address of the stack. When the attribute value is __MR_INT or __MR_EXT, NULL must be specified in that variable.
cre_mbx	Mailbox	__MR_USER	No substitute means available.
cre_mbf	Message buffer	__MR_USER	Specify the start address of the message buffer area allocated by the user through the packet variable that indicates the start address of the message buffer area. When the attribute value is __MR_INT or __MR_EXT, NULL must be specified in that variable.
cre_mpf	Fixed-size memory pool	__MR_USER	Specify the start address of the fixed-size memory pool area allocated by the user through the packet variable that indicates the start address of the fixed-size memory pool area. When the attribute value is __MR_INT or __MR_EXT, NULL must be specified in that variable.
cre_mpl	Variable-size memory pool	__MR_USER	Specify the start address of the variable-size memory pool area allocated by the user through the packet variable that indicates the start address of the variable-size memory pool area. When the attribute value is __MR_INT or __MR_EXT, NULL must be specified in that variable.

3.3.3 wup_tsk and sus_tsk

Parameter tskid = TSK_SELF (0) specifies the current task in the MR32R/4.

Table 3.10 Difference in wup_tsk and sus_tsk

Parameter tskid	MR32R	MR32R/4
TSK_SELF(0)	Must not be specified. If specified, the system does not operate correctly.	Current task specification

3.3.4 Clearing Specification for Event Flags

In the MR32R, clearing of an event flag is specified through the wait mode when an event-waiting system call is issued. In the MR32R/4, each event flag has a clearing specification attribute. Figure 3.2 shows the how the clearing specification differs.

When shifting to the MR32R/4, check if one event flag is waited for both with and without clearing specifications; if both waiting modes are used for one event flag, consider the order of the clearing specifications.

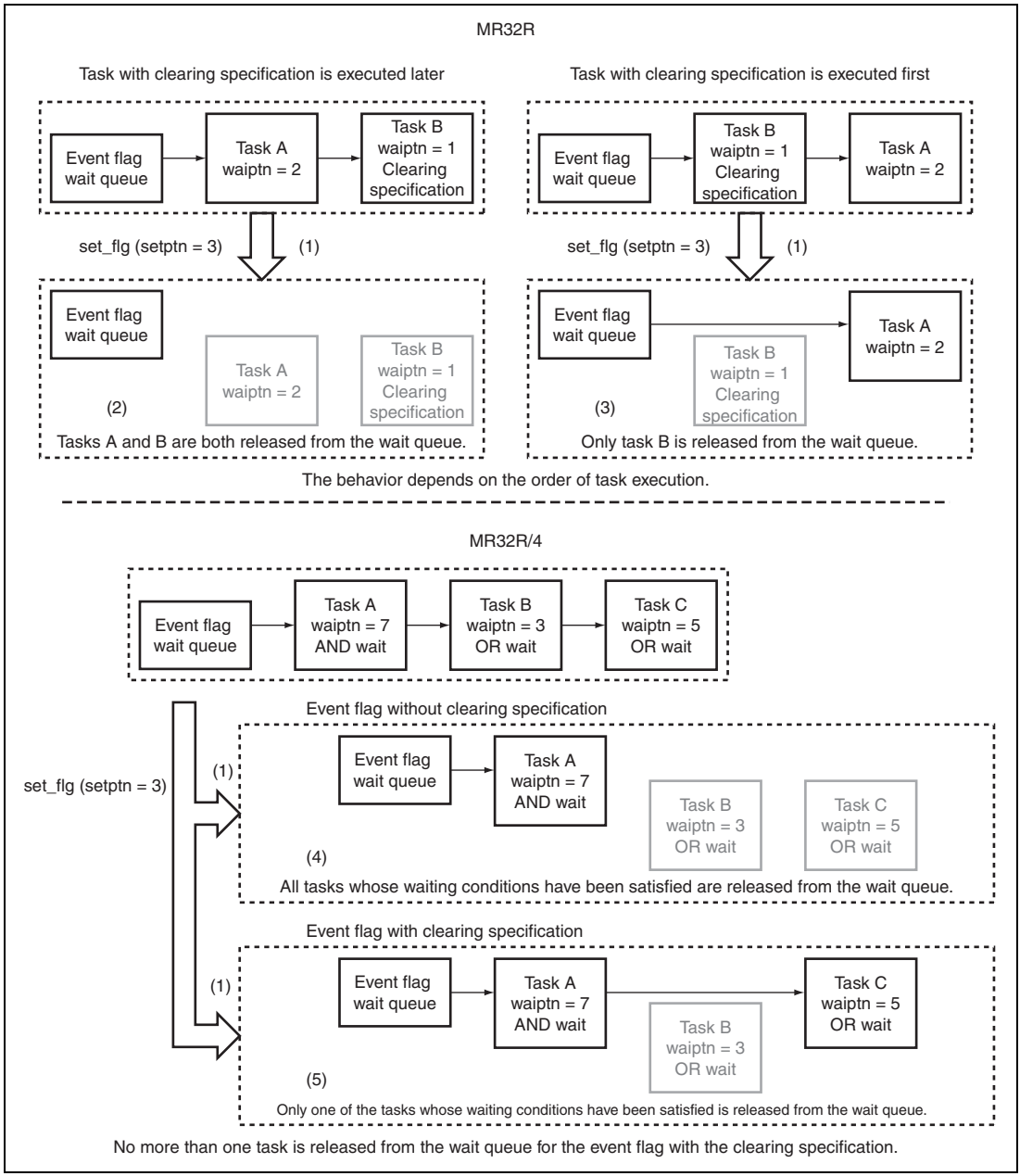

Figure 3.2 Differences in Event Flag Clearing Specification

1. Set bit pattern (3) in the event flag.
2. Task A is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 2. As flag clearing is not specified, the next wait task is also checked.

   Task B is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 1. As flag clearing is specified, processing ends.
3. Task B is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 1. As flag clearing is specified in task B, the tasks behind task B in the wait queue are not checked and processing ends.
4. Task A remains in the wait queue because it waits for bit pattern 7 with the AND condition, which does not match the pattern set in step 1.

   Task B is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 3 with the OR condition. As flag clearing is not specified for the event flag, all tasks in the wait queue are checked.

   Task C is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 5 with the OR condition.
5. Task A remains in the wait queue because it waits for bit pattern 7 with the AND condition, which does not match the pattern set in step 1.

   Task B is released from the wait queue by the bit pattern set in step 1 because it waits for bit pattern 3 with the OR condition. As flag clearing is specified for the event flag, the tasks behind task B in the wait queue are not checked and processing ends.

3.3.5 Modification Related to Mailbox

In the MR32R, 32-bit data can be sent and received as a message through a mailbox, but in the MR32R/4, only the start address of a message can be sent and received through a mailbox.

32-bit data transfer is available through a data queue.

The extended function (mailbox with priority) in the MR32R is integrated into the ordinary mailbox function in the MR32R/4.

Table 3.11 Modification Related to Mailbox

Function	MR32R	MR32R/4
32-bit data transfer	Mailbox	Data queue
Mailbox with priority	Extended function (mailbox with priority)	Ordinary mailbox function

3.3.6 Modification Related to Rendezvous

The timeout time and behavior in a rendezvous call differ between the MR32R and MR32R/4. Refer to table 3.12 and figure 3.3 for details.

With this timeout modification, pcal_por is deleted from the MR32R/4; the pcal_por function is done through the timeout specification in tcal_por.

Table 3.12 Modification Related to Rendezvous

Function	MR32R	MR32R/4
Timeout time specified in a rendezvous call	Time until the rendezvous call is canceled	Time until the rendezvous operation ends
Behavior when timeout occurs in a rendezvous call	The task issuing a rendezvous call is placed in the call-waiting queue.	The processing is canceled.

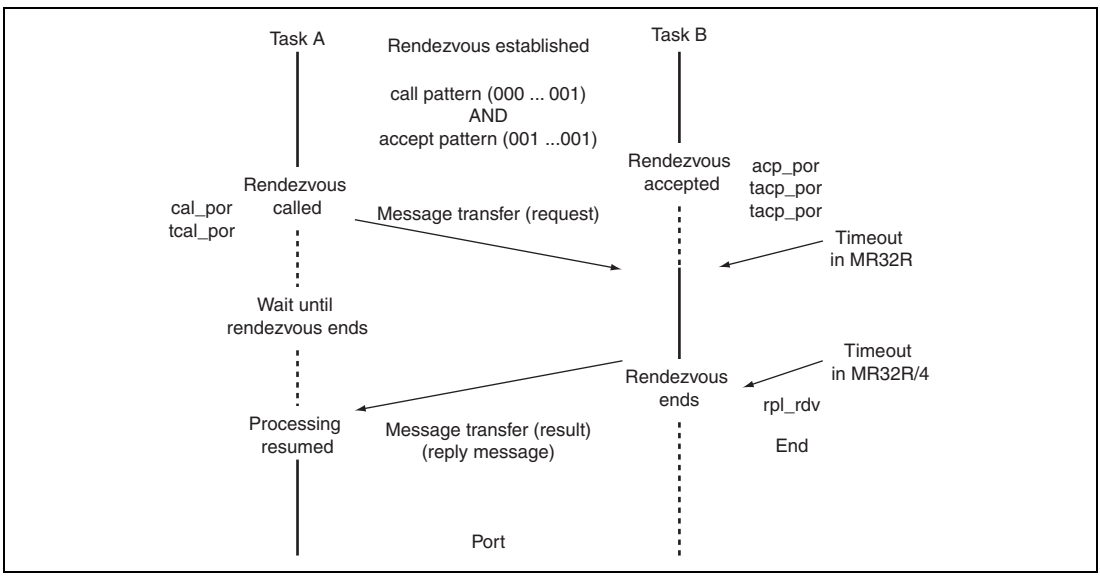

Figure 3.3 Modification of Timeout in Rendezvous

3.3.7 Time Management

The clock and time management is modified in the MR32R/4.

The clock and time are managed in time tick units (the number of interrupts generated in the hardware timer used by the kernel) in the MR32R; in the MR32R/4, the management unit is fixed to 1 ms.

Table 3.13 Time Unit for Time Parameters

No.	Modified Item	MR32R	MR32R/4
1	Time unit for time parameters	Number of time ticks	milliseconds

In the MR32R, a timeout occurs when the specified number of time ticks has been passed. The MR32R/4 conforms to the µITRON4.0 specifications; that is, the system must guarantee that the specified time has elapsed.

See figure 3.4 for the difference in time management between the MR32R and MR32R/4.

For example, if the hardware timer cycle is the same as the time tick cycle, specify (timeout value set in MR32R - 1) as the timeout value in the MR32R/4 to achieve the same actual time as in the MR32R. Calculate the timeout time in this way and modify the application.

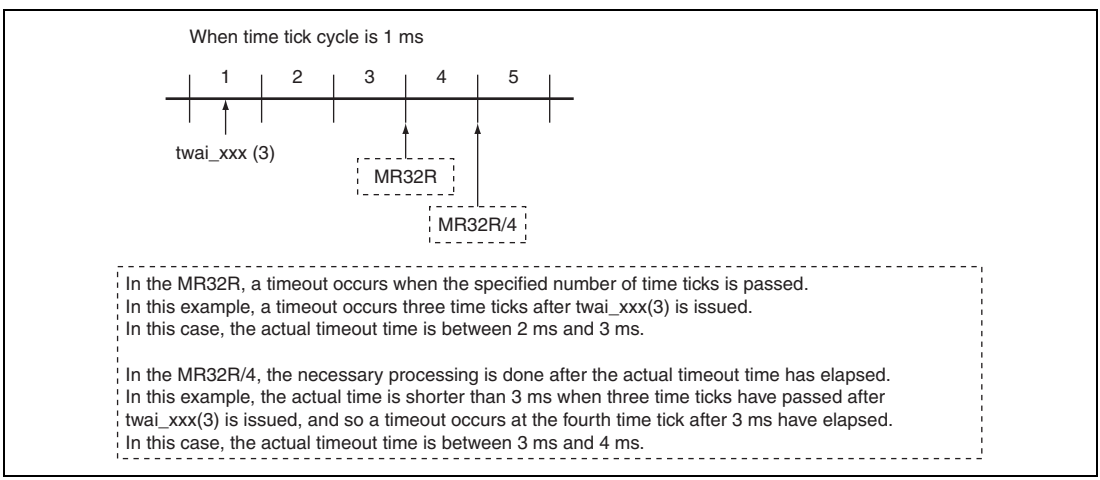

Figure 3.4 Difference in Time Management between MR32R and MR32R/4

3.3.8 Cyclic Handler

Modifications have been made throughout the cyclic handler function in the MR32R/4.

Modify the application by referring to the respective manuals.

Table 3.14 Comparison of Cyclic Handler between MR32R and MR32R/4

Cyclic Handler in MR32R		Cyclic Handler in MR32R/4	
def_cyc	Define a cyclic handler	cre_cyc	Create a cyclic handler
		acre_cyc	Create a cyclic handler (automatically assign ID)
act_cyc	Activate a cyclic handler	sta_cyc	Start a cyclic handler
		ista_cyc	
		stp_cyc	Stop a cyclic handler
		istp_cyc	
ref_cyc	Refer to the cyclic handler state	ref_cyc	Refer to the cyclic handler state
		iref_cyc	
—	—	del_cyc	Delete a cyclic handler

Table 3.15 Settings for Cyclic Handler

Operation	Settings in MR32R	Settings in MR32R/4
Start a cyclic handler (storing initiation phase)	act_cyc: TCY_ON	sta_cyc (specifying TA_PHS in cre_cyc)
Start a cyclic handler (clearing initiation phase)	act_cyc: TCY_INI_ON	sta_cyc (not specifying TA_PHS in cre_cyc)
Stop a cyclic handler	act_cyc: TCY_OFF	stp_cyc

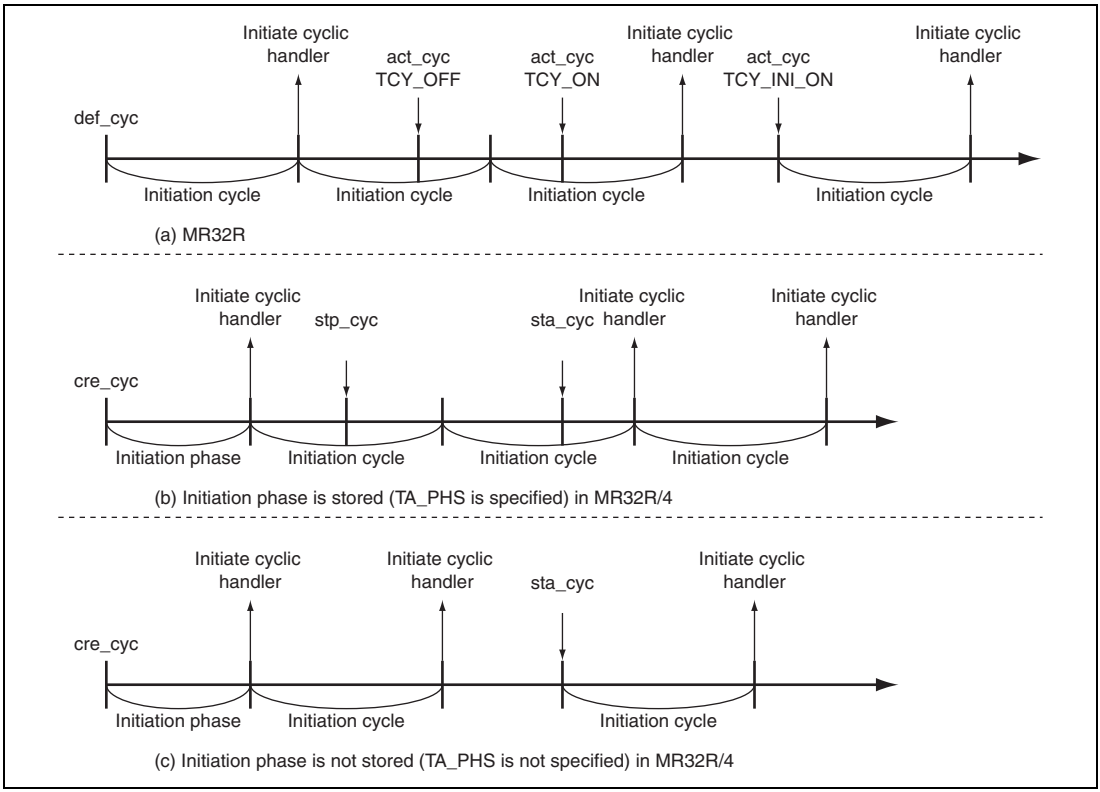

Figure 3.5 Differences in Cyclic Handler Operation

3.3.9 Alarm Handler

Modifications have been made throughout the alarm handler function in the MR32R/4.

Modify the application by referring to the respective manuals.

Table 3.16 Comparison of Alarm Handler between MR32R and MR32R/4

Alarm Handler in MR32R		Alarm Handler in MR32R/4	
—	—	cre_alm	Create an alarm handler
		acre_alm	Create an alarm handler (automatically assign ID)
		sta_alm	Start an alarm handler
		ista_alm	
		stp_alm	Stop an alarm handler
		istp_alm	
ref_alm	Refer to the alarm handler state	ref_alm	Refer to the alarm handler state
		iref_alm	
—	—	del_alm	Delete an alarm handler

3.3.10 Coding Examples

(1) Task

Figures 3.6 and 3.7 show examples of task coding.

```
#include <mr32r.h>                  ← At the beginning of the file, be sure to include "mr32r.h"
#include "id.h"                         stored in the system directory and "id.h" stored in the current
                                        directory.
void task(void)                     ← No value is returned from the task start function.
{                                       Therefore, declare the function as a void-type function.
    /* Processing */
    ext_tsk();                      ← Be sure to issue ext_tsk() at the end of the task start function.
}
```

Figure 3.6 Example of Task Coding in MR32R

```
#include <itron.h>                  ← At the beginning of the file, be sure to include "itron.h" and
#include <kernel.h>                     "kernel.h" stored in the system directory and "kernel_id.h"
#include "kernel_id.h"                  stored in the current directory.

void task(VP_INT stacd)             ← No value is returned from the task start function.
{                                       Therefore, declare the function as a void-type function.
   /* Processing */
}                                   ← There is no need to issue ext_tsk() at the end of the task start
                                        function; the kernel automatically issues ext_tsk.
```

Figure 3.7 Example of Task Coding in MR32R/4

(2) Interrupt Handler

Figures 3.8 and 3.9 show examples of interrupt handler coding.

```
#include <mr32r.h>                  ← At the beginning of the file, be sure to include "mr32r.h"
#include "id.h"                         stored in the system directory and "id.h" stored in the current
                                        directory.

void int_handler(void)              ← Be sure to use the void type to declare the return value
{                                       and parameter for the interrupt handler start function.
   /* Processing */
   iwup_tsk(ID_main);
}
```

Figure 3.8 Example of Interrupt Handler Coding in MR32R

```
#include <itron.h>                  ← At the beginning of the file, be sure to include "itron.h" and
#include <kernel.h>                     "kernel.h" stored in the system directory and "kernel_id.h"
#include “kernel_id.h”                 stored in the current directory.

void int_handler(void)              ← Be sure to use the void type to declare the return value and
                                        parameter for the interrupt handler start function.
{
    /* Processing */
    iwup_tsk(ID_main);              ← Be sure to issue ext_tsk() at the end of the task start function.
}
```

Figure 3.9 Example of Interrupt Handler Coding in MR32R/4

Section 4 Service Call Differences

4.1 SVC Differences

In the following tables, the shaded rows indicate the changed functions.
The "Change" column shows the modifications in the APIs.
The following terms are used in these tables.
Prm: Parameter R_Prm: Return parameter struct: Structure

4.1.1 Task Management

	MR32R (µITRON 3.0) Function				MR32R/4 (µITRON 4.0) Function				
	SVC	API			SVC	API			Change
1	Create task				Create task				Packet struct changed
	cre_tsk	ER ercd = cre_tsk (ID tskid, T_CTSK *pk_ctsk);			cre_tsk	ER ercd = cre_tsk (ID tskid, T_CTSK *pk_ctsk);			
		Packet struct typedef struct t_ctsk {				Packet struct typedef struct t_ctsk {			
		VP	exinf;	Extended information		ATR	tskatr	Task attribute	
		ATR	tskatr;	Task attribute		VP_INT	exinf	Task extended information	
		FP	task;	Task start address		FP	task	Task start address	
		PRI	itskpri;	Priority at task initiation		PRI	itskpri	Task priority at initiation	
		INT	stksz;	Stack Size		SIZE	stksz	Task stack area size (bytes)	
		VP	stk;	Start address of area allocated by user		VP	stk	Start address of task stack area	
		} T_CTSK;				} T_CTSK;			
—	—				Create task (assign task ID automatically)				New function
	Not available	—			acre_tsk	ER_ID tskid = acre_tsk (T_CTSK *pk_ctsk);			
2	Delete task				Delete task				
	del_tsk	ER ercd = del_tsk (ID tskid);			del_tsk	ER ercd = del_tsk (ID tskid);			
3	Start task				Start task (with start code specified)				Prm type changed.
	sta_tsk	ER ercd = sta_tsk (ID tskid, INT stacd);			sta_tsk	ER ercd = sta_tsk (ID tskid, VP_INT stcd);			
		Prm	INT	stacd; Task start code		Prm	VP_INT	stcd	Task start code
4	Start task (dedicated to handlers)				Start task (with start code specified) (dedicated to handlers)				Prm type changed.
	ista_tsk	ER ercd = ista_tsk (ID tskid, INT stacd);			ista_tsk	ER ercd = ista_tsk (ID tskid, VP_INT stcd);			
		Prm	INT	stacd; Task start code		Prm	VP_INT	stcd	Task start code
—	—				Initiate task				New function
	Not available	—			act_tsk iact_tsk	ER ercd = act_tsk (ID tskid); ER ercd = iact_tsk (ID tskid);			
—	—				Cancel task initiation request				New function
	Not available	—			can_act ican_act	ER_UINT actcnt = can_act (ID tskid); ER_UINT actcnt = ican_act (ID tskid);			

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
5	Exit current task normally		Exit current task		
	ext_tsk	void ext_tsk();	ext_tsk	ext_tsk();	
6	Forcibly exit and delete current task		Exit and delete current task		
	exd_tsk	void exd_tsk();	exd_tsk	exd_tsk();	
7	Forcibly terminate task		Forcibly terminate task		
	ter_tsk	ER ercd = ter_tsk (ID tskid);	ter_tsk	ER ercd = ter_tsk (ID tskid);	
8	Change task priority		Change task priority		
	chg_pri	ER ercd = chg_pri (ID tskid, PRI tskpri);	chg_pri	ER ercd = chg_pri (ID tskid, PRI tskpri);	
9	Change task priority (dedicated to handlers)		Change task priority (dedicated to handlers)		
	ichg_pri	ER ercd = ichg_pri (ID tskid, PRI tskpri);	ichg_pri	ER ercd = ichg_pri (ID tskid, PRI tskpri);	
—	—		Refer to task priority		New function
	Not available	—	get_pri iget_pri	ER ercd = get_pri (ID tskid, PRI *p_tskpri); ER ercd = iget_pri (ID tskid, PRI *p_tskpri);	
10	Disable task dispatch		Disable dispatch		Function changed. See 3.3.1.
	dis_dsp	ER ercd = dis_dsp();	dis_dsp	ER ercd = dis_dsp();	
11	Enable task dispatch		Enable dispatch		Function changed. See 3.3.1.
	ena_dsp	ER ercd = ena_dsp();	ena_dsp	ER ercd = ena_dsp();	
12	Rotate ready queue		Rotate task priority		
	rot_rdq	ER ercd = rot_rdq (PRI tskpri);	rot_rdq	ER ercd = rot_rdq (PRI tskpri);	
13	Rotate ready queue (dedicated to handlers)		Rotate task priority (dedicated to handlers)		
	irot_rdq	ER ercd = irot_rdq (PRI tskpri);	irot_rdq	ER ercd = irot_rdq (PRI tskpri);	
14	Cancel WAITING state forcibly		Cancel WAITING state forcibly		
	rel_wai	ER ercd = rel_wai (ID tskid);	rel_wai	ER ercd = rel_wai (ID tskid);	
15	Cancel WAITING state forcibly (dedicated to handlers)		Cancel WAITING state forcibly (dedicated to handlers)		
	irel_wai	ER ercd = irel_wai (ID tskid);	irel_wai	ER ercd = irel_wai (ID tskid);	
16	Get current task ID		Get task ID in RUNNING state		Operation changed.
	get_tid	ER ercd = get_tid (ID *p_tskid);	get_tid iget_tid	ER ercd = get_tid (ID *p_tskid); ER ercd = iget_tid (ID *p_tskid);	
	Operation	Returns FALSE(0) as the task ID if the SVC is issued in a task-independent portion.	Operation	Returns the ID of the task that is being executed if the SVC is issued in a non-task context.	

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
17	Refer to task state				Refer to task state				Prm sequence and packet struct changed.
	ref_tsk	ER ercd = ref_tsk (T_RTSK *pk_rtsk, ID tskid);			ref_tsk iref_tsk	ER ercd = ref_tsk (ID tskid, T_RTSK *pk_rtsk); ER ercd = iref_tsk (ID tskid, T_RTSK *pk_rtsk);			
		Packet struct	typedef struct t_rtsk {			Packet struct	typedef struct t_rtsk {		
			VP	exinf;	Extended information		STAT	tskstat	Task state
			PR	tskpri	Current task priority		PRI	tskpri	Current priority of task
			UH	tskstat	Task state		PRI	tskbpri	Base priority of task
			UINT	tskwait	Wait cause		STAT	tskwait	Wait cause
			ID	wid	Wait object ID		ID	wobjid	Wait object ID
			INT	wupcnt	Number of wakeup requests		TMO	lefttmo	Time to timeout
			ATR	tskatr	Task attribute		UINT	actcnt	Number of queued initiation requests
			FP	task	Task start address		UINT	wupcnt	Number of queued wakeup requests
			PRI	itskpri	Task priority at initiation		UINT	suscnt	Suspend request nest count
			INT	stksz	Stack size		} T_RTSK;		
			UW	epndptn	Exception pending pattern				
			} T_RTSK;						
—	—				Refer to task state (simple version)				New function
	Not available	—			ref_tst iref_tst	ER_ercd = ref_tst (ID tskid, T_RTST *pk_rtst); ER_ercd = iref_tst (ID tskid, T_RTST *pk_rtst);			

4.1.2 Task Synchronization

	MR32R (µITRON 3.0) Function					MR32R/4 (µITRON 4.0) Function					Change
	SVC	API				SVC	API				
1	Suspend task					Suspend task					Operation modified. See 3.3.3.
	sus_tsk isus_tsk	ER ercd = sus_tsk (ID tskid); ER ercd = isus_tsk (ID tskid);				sus_tsk isus_tsk	ER ercd = sus_tsk (ID tskid); ER ercd = isus_tsk (ID tskid);				
		Operation	Current task cannot be specified.				Operation	Current task can be specified through tskid = 0.			
2	Resume task					Resume task					
	rsm_tsk irsm_tsk	ER ercd = rsm_tsk (ID tskid); ER ercd = irsm_tsk (ID tskid);				rsm_tsk irsm_tsk	ER ercd = rsm_tsk (ID tskid); ER ercd = irsm_tsk (ID tskid);				
—		—				Resume task forcibly					New function
	Not available	—				frsm_tsk ifrsm_tsk	ER ercd = frsm_tsk (ID tskid); ER ercd = ifrsm_tsk (ID tskid);				
3	Sleep task					Sleep task					
	slp_tsk	ER ercd = slp_tsk();				slp_tsk	ER ercd = slp_tsk();				
4	Sleep task for a specified time					Sleep task (with timeout)					
	tslp_tsk	ER ercd = tslp_tsk (TMO tmout);				tslp_tsk	ER ercd = tslp_tsk (TMO tmout);				
5	Wake up task					Wake up task					Operation modified. See 3.3.3.
	wup_tsk iwup_tsk	ER ercd = wup_tsk (ID tskid); ER ercd = iwup_tsk (ID tskid);				wup_tsk iwup_tsk	ER ercd = wup_tsk (ID tskid); ER ercd = iwup_tsk (ID tskid);				
		Operation	Current task cannot be specified.				Operation	Current task can be specified through tskid = 0.			
6	Cancel wakeup request					Cancel wakeup request					Prm and R_Prm changed.
	can_wup	ER ercd = can_wup (INT *p_wupcnt, ID tskid);				can_wup ican_wup	ER UINT wupcnt = can_wup (ID tskid); ER UINT wupcnt = ican_wup (ID tskid);				
		Prm	INT	*p_wupcnt;	Start address of area to store the number of canceled wakeup requests		Prm	ID	tskid;	Task ID	
			ID	tskid;	Task ID						
		R_Prm	ER	ercd;	Error code		R_Prm	ER_UINT	wupcnt;	wupcnt > 0: Number of canceled wakeup requests wupcnt < 0: Error code	
			INT	*p_wupcnt	Number of canceled wakeup requests						

4.1.3 Task Exception Processing

MR32R (µITRON 3.0)			MR32R/4 (µITRON 4.0)		
Function			Function		
	SVC	API	SVC	API	Change
—	—	—	Define task exception processing routine		New function. See 3.2.2.
	Not available	—	def_tex	ER ercd = def_tex (ID tskid, T_DTEX *pk_dtex);	
—	—	—	Request task exception processing		New function. See 3.2.2.
	Not available	—	ras_tex	ER ercd = ras_tex (ID tskid, TEXPTN rasptn);	
—	—	—	Request task exception processing (dedicated to handlers)		New function. See 3.2.2.
	Not available	—	iras_tex	ER ercd = iras_tex (ID tskid, TEXPTN rasptn);	
—	—	—	Disable task exception processing		New function. See 3.2.2.
	Not available	—	dis_tex	ER ercd = dis_tex();	
—	—	—	Enable task exception processing		New function. See 3.2.2.
	Not available	—	ena_tex	ER ercd = ena_tex();	
—	—	—	Refer to task exception processing disabled state		New function. See 3.2.2.
	Not available	—	sns_tex	ER ercd = sns_tex();	
—	—	—	Refer to task exception processing state		New function. See 3.2.2.
	Not available	—	ref_tex	ER ercd = ref_tex (ID tskid, T_RTEX *pk_rtex);	
—	—	—	Refer to task exception processing state (dedicated to handlers)		New function. See 3.2.2.
	Not available	—	iref_tex	ER ercd = iref_tex (ID tskid, T_RTEX *pk_rtex);	

4.1.4 Synchronization and Communication (Semaphore)

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
1	Create semaphore		Create semaphore		Packet struct changed.
	cre_sem	ER ercd = cre_sem (ID semid, T_CSEM *pk_csem);	cre_sem	ER ercd = cre_sem (ID semid, T_CSEM *pk_csem);	
		Packet struct typedef struct t_csem { VP exinf; Extended information ATR sematr; Semaphore attribute INT isemcnt; Initial value of semaphore INT maxsem; Maximum semaphore value } T_CSEM;		Packet struct typedef struct t_csem { ATR sematr; Semaphore attribute UINT isemcnt; Initial value of semaphore resource count UINT maxsem; Maximum number of semaphore resources } T_CSEM;	
—	—		Create semaphore (assign ID automatically)		New function
	Not available	—	acre_sem	ER_ID semid = acre_sem (T_CSEM *pk_csem);	
2	Delete semaphore		Delete semaphore		
	del_sem	ER ercd = del_sem (ID semid);	del_sem	ER ercd = del_sem (ID semid);	
3	Return semaphore resource		Return semaphore resource		
	sig_sem	ER ercd = sig_sem (ID semid);	sig_sem	ER ercd = sig_sem (ID semid);	
4	Return semaphore resource (dedicated to handlers)		Return semaphore resource (dedicated to handlers)		
	isig_sem	ER ercd = isig_sem (ID semid);	isig_sem	ER ercd = isig_sem (ID semid);	
5	Wait for semaphore resource		Wait for semaphore resource		
	wai_sem	ER ercd = wai_sem (ID semid);	wai_sem	ER ercd = wai_sem (ID semid);	
6	Wait for semaphore resource (with timeout)		Wait for semaphore resource (with timeout)		
	twai_sem	ER ercd = twai_sem (ID semid, TMO tmout);	twai_sem	ER ercd = twai_sem (ID semid, TMO tmout);	
7	Wait for semaphore resource (polling)		Wait for semaphore resource (polling)		SVC name changed.
	preq_sem	ER ercd = preq_sem (ID semid);	pol_sem ipol_sem	ER ercd = pol_sem (ID semid); ER ercd = ipol_sem (ID semid);	
8	Refer to semaphore state		Refer to semaphore state		Packet struct and Prm sequence changed.
	ref_sem	ER ercd = ref_sem (T_RSEM *pk_rsem, ID semid);	ref_sem iref_sem	ER ercd = ref_sem (ID semid, T_RSEM *pk_rsem); ER ercd = iref_sem (ID semid, T_RSEM *pk_rsem);	
		Packet struct typedef struct t_rsem { VP exinf; Extended information BOOL_ID wtsk; Waiting task information INT semcnt; Current semaphore count } T_RSEM;		Packet struct typedef struct t_rsem { ID wtskid; Waiting task ID UINT semcnt; Current semaphore counter value } T_RSEM;	

4.1.5 Synchronization and Communication (Event Flag)

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
1	Create event flag					Create event flag					Packet struct changed.
	cre_flg	ER ercd = cre_flg (ID flgid, T_CFLG *pk_cflg);				cre_flg	ER ercd = cre_flg (ID flgid, T_CFLG *pk_cflg);				
		Packet struct	typedef struct t_cflg {				Packet struct	typedef struct t_cflg {			
			VP	exinf;	Extended information			ATR	flgatr;	Event flag attribute	
			ATR	flgatr;	Event flag attribute			FLGPTN	iflgptn;	Initial value of event flag	
			UINT	iflgptn;	Initial value of event flag			} T_CFLG;			
			} T_CFLG;								
—	—					Create event flag (assign ID automatically)					New function
	Not available	—				acre_flg	ER_ID flgid = acre_flg (T_CFLG *pk_cflg);				
2	Delete event flag					Delete event flag					
	del_flg	ER ercd = del_flg (ID flgid);				del_flg	ER ercd = del_flg (ID flgid);				
3	Set event flag					Set event flag					Prm type and operation changed. See 3.3.4.
	set_flg	ER ercd = set_flg (ID flgid, UINT setptn);				set_flg	ER ercd = set_flg (ID flgid, FLGPTN setptn);				
		Prm	UINT	setptn;	Bit pattern to set		Prm	FLGPTN	setptn;	Bit pattern to set	
4	Set event flag (dedicated to handlers)					Set event flag (dedicated to handlers)					Prm type and operation changed. See 3.3.4.
	iset_flg	ER ercd = iset_flg (ID flgid, UINT setptn);				iset_flg	ER ercd = iset_flg (ID flgid, FLGPTN setptn);				
		Prm	UINT	setptn;	Bit pattern to set		Prm	UINT	setptn;	Bit pattern to set	
5	Clear event flag					Clear event flag					Prm type changed.
	clr_flg	ER ercd = clr_flg (ID flgid, UINT clrptn);				clr_flg iclr_flg	ER ercd = clr_flg (ID flgid, FLGPTN clrptn); ER ercd = iclr_flg (ID flgid, FLGPTN clrptn);				
		Prm	UINT	setptn;	Bit pattern to be cleared		Prm	UINT	setptn;	Bit pattern to be cleared	
6	Wait for event flag					Wait for event flag					Prm type, Prm sequence, and operation changed. See 3.3.4.
	wai_flg	ER ercd = wai_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);				wai_flg	ER ercd = wai_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);				
		Prm	UINT	*p_flgptn;	Start address of area where the bit pattern at waiting release is to be returned		Prm	FLGPTN	*p_flgptn;	Pointer to area where the bit pattern at waiting release is to be returned	
			UINT	waiptn;	Wait bit pattern			MODE	waiptn;	Wait bit pattern	
			UINT	wfmode;	Wait mode			FLGPTN	wfmode;	Wait mode	
7	Wait for event flag (with timeout)					Wait for event flag (with timeout)					Prm type, Prm sequence, and operation changed. See 3.3.4.
	twai_flg	ER ercd = twai_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO tmout);				twai_flg	ER ercd = twai_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);				
		Prm	UINT	*p_flgptn;	Start address of area where the bit pattern at waiting release is to be returned		Prm	FLGPTN	*p_flgptn;	Pointer to area where the bit pattern at waiting release is to be returned	
			UINT	waiptn;	Wait bit pattern			MODE	waiptn;	Wait bit pattern	
			UINT	wfmode;	Wait mode			FLGPTN	wfmode;	Wait mode	

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
8	Acquire event flag (without wait)					Wait for event flag (polling)					Prm type, Prm sequence, and operation changed. See 3.3.4.
	pol_flg	ER ercd = pol_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);				pol_flg ipol_flg	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);				
		Prm	UINT	*p_flgptn;	Start address of area where the bit pattern at waiting release is to be returned		Prm	FLGPTN	*p_flgptn;	Pointer to area where the bit pattern at waiting release is to be returned	
			UINT	waiptn;	Wait bit pattern			MODE	waiptn;	Wait bit pattern	
			UINT	wfmode;	Wait mode			FLGPTN	wfmode;	Wait mode	
9	Refer to event flag state					Refer to event flag state					Prm type and Prm sequence changed.
	ref_flg	ER ercd = ref_flg (T_RFLG *pk_rflg, ID flgid);				ref_flg iref_flg	ER ercd = ref_flg (ID flgid, T_RFLG *pk_rflg); ER ercd = iref_flg (ID flgid, T_RFLG *pk_rflg);				
		Prm	typedef struct t_rflg{				Prm	typedef struct t_rflg{			
			VP	exinf;	Extended information			ID	wtskid	Wait task ID	
			BOOL_ID	wtsk;	Waiting task information			FLGPTN	flgptn	Current event flag bit pattern	
			UINT	flgptn;	Event flag bit pattern			} T_RFLG;			
			} T_RFLG;								

4.1.6 Synchronization and Communication (Data Queue)

MR32R (µITRON 3.0)			MR32R/4 (µITRON 4.0)		
Function			Function		
	SVC	API	SVC	API	Change
—	—	—	Create data queue		New function. See 3.2.1
	Not available	—	cre_dtq	ER ercd = cre_dtq (ID dtqid, T_CDTQ *pk_cdtq);	
—	—	—	Create data queue (assign ID automatically)		New function. See 3.2.1
	Not available	—	acre_dtq	ER_ID dtqid = acre_dtq (T_CDTQ *pk_cdtq);	
—	—	—	Delete data queue		New function. See 3.2.1
	Not available	—	del_dtq	ER ercd = del_dtq (ID dtqid);	
—	—	—	Send data to data queue		New function. See 3.2.1
	Not available	—	snd_dtq	ER ercd = snd_dtq (ID dtqid, VP_INT data);	
—	—	—	Send data to data queue (polling)		New function. See 3.2.1
	Not available	—	psnd_dtq	ER ercd = psnd_dtq (ID dtqid, VP_INT data);	
—	—	—	Send data to data queue (dedicated to handlers)		New function. See 3.2.1
	Not available	—	ipsnd_dtq	ER ercd = ipsnd_dtq (ID dtqid, VP_INT data);	
—	—	—	Send data to data queue (with timeout)		New function. See 3.2.1
	Not available	—	tsnd_dtq	ER ercd = tsnd_dtq (ID dtqid, VP_INT data, TMO tmout);	
—	—	—	Forcibly send data to data queue		New function. See 3.2.1
	Not available	—	fsnd_dtq	ER ercd = fsnd_dtq (ID dtqid, VP_INT data);	
—	—	—	Forcibly send data to data queue (dedicated to handlers)		New function. See 3.2.1
	Not available	—	ifsnd_dtq	ER ercd = ifsnd_dtq (ID dtqid, VP_INT data);	
—	—	—	Receive data from data queue		New function. See 3.2.1
	Not available	—	rcv_dtq	ER ercd = rcv_dtq (ID dtqid, VP_INT *p_data);	
—	—	—	Receive data from data queue (polling)		New function. See 3.2.1
	Not available	—	prcv_dtq	ER ercd = prcv_dtq (ID dtqid, VP_INT *p_data);	
—	—	—	Receive data from data queue (dedicated to handlers)		New function. See 3.2.1
	Not available	—	iprcv_dtq	ER ercd = iprcv_dtq (ID dtqid, VP_INT *p_data);	
—	—	—	Receive data from data queue (with timeout)		New function. See 3.2.1
	Not available	—	trcv_dtq	ER ercd = trcv_dtq (ID dtqid, VP_INT *p_data, TMO tmout);	
—	—	—	Refer to data queue state		New function. See 3.2.1
	Not available	—	ref_dtq	ER ercd = ref_dtq (ID dtqid, T_RDTQ *pk_rdtq);	
—	—	—	Refer to data queue state (dedicated to handlers)		New function. See 3.2.1
	Not available	—	iref_dtq	ER ercd = iref_dtq (ID dtqid, T_RDTQ *pk_rdtq);	

4.1.7 Synchronization and Communication (Mailbox)

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
1	Create mailbox					Create mailbox					Packet struct changed.
	cre_mbx	ER ercd = cre_mbx (ID mbxid, T_CMBX *pk_cmbx);				cre_mbx	ER ercd = cre_mbx (ID mbxid, T_CMBX *pk_cmbx);				
		Packet struct	typedef struct t_cmbx {				Packet struct	typedef struct t_cmbx {			
			VP	exinf;	Extended information			ATR	mbxatr;	Mailbox attribute	
			ATR	mbxatr;	Mailbox attribute			PRI	maxmpri;	Maximum value of message priority	
			INT	bufcnt;	Ring buffer size			VP	mprihd;	Start address of message queue header with priority	
			VP	mbx;	Start address of mailbox area allocated by user			} T_CMBX;			
			} T_CMBX;								
—	—					Create mailbox (assign ID automatically)					New function
	Not available	—				acre_mbx	ER_ID mbxid = acre_mbx (T_CMBX *pk_cmbx);				
2	Delete mailbox					Delete mailbox					
	del_mbx	ER ercd = del_mbx (ID mbxid);				del_mbx	ER ercd = del_mbx (ID mbxid);				
3	Send data to mailbox					Send data to mailbox					SVC name and message header changed. See 3.3.5.
	snd_msg	ER ercd = snd_msg (ID mbxid, T_MSG *pk_msg);				snd_mbx	ER ercd = snd_mbx (ID mbxid, T_MSG *pk_msg);				
		Packet struct	typedef UW	T_MSG;	Start address of packet is sent as a message		Packet struct	typedef struct t_msg {			
								VP	msghead;	Kernel management area	
								} T_MSG;		(Message header in mailbox)	
			typedef void	*PT_MSG;	32-bit data is sent as a message			typedef struct t_msg_pri {			
								T_MSG	msgque;	Message header	
								PRI	msgpri;	Message priority	
								} T_MSG_PRI;		(Mailbox message header with priority)	

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
4	Send data to mailbox (dedicated to handlers)				Send data to mailbox (dedicated to handlers)				SVC name and message header changed. See 3.3.5.
	isnd_msg	ER ercd = isnd_msg (ID mbxid, T_MSG *pk_msg);			isnd_mbx	ER ercd = isnd_mbx (ID mbxid, T_MSG *pk_msg);			
		Packet struct	typedef UW T_MSG;	Start address of packet is sent as a message		Packet struct	typedef struct t_msg {		
							VP msghead;	Kernel management area	
							} T_MSG;	(Message header in mailbox)	
		typedef void *PT_MSG;		32-bit data is sent as a message			typedef struct t_msg_pri {		
							T_MSG msgque;	Message header	
							PRI msgpri;	Message priority	
							} T_MSG_PRI;	(Mailbox message header with priority)	
5	Receive data from mailbox				Receive data from mailbox				SVC name, Prm sequence, and message header changed. See 3.3.5.
	rcv_msg	ER ercd = rcv_msg (T_MSG **ppk_msg, ID mbxid);			rcv_mbx	ER ercd = rcv_mbx (ID mbxid, T_MSG **ppk_msg);			
		Packet struct	typedef UW T_MSG;	Start address of packet is received as a message		Packet struct	typedef struct t_msg {		
							VP msghead;	Kernel management area	
							} T_MSG;	(Message header in mailbox)	
		typedef void *PT_MSG;		32-bit data is received as a message			typedef struct t_msg_pri {		
							T_MSG msgque;	Message header	
							PRI msgpri;	Message priority	
							} T_MSG_PRI;	(Mailbox message header with priority)	
6	Receive data from mailbox (with timeout)				Receive data from mailbox (with timeout)				SVC name, Prm sequence, and message header changed. See 3.3.5.
	trcv_msg	ER ercd = trcv_msg (T_MSG **ppk_msg, ID mbxid, TMO tmout);			trcv_mbx	ER ercd = trcv_mbx (ID mbxid, T_MSG **ppk_msg, TMO tmout);			
		Packet struct	typedef UW T_MSG;	Start address of packet is received as a message		Packet struct	typedef struct t_msg {		
							VP msghead;	Kernel management area	
							} T_MSG;	(Message header in mailbox)	
		typedef void *PT_MSG;		32-bit data is received as a message			typedef struct t_msg_pri {		
							T_MSG msgque;	Message header	
							PRI msgpri;	Message priority	
							} T_MSG_PRI;	(Mailbox message header with priority)	

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
7	Receive data (without wait)				Receive data from mailbox (polling)				SVC name, Prm sequence, and message header changed. See 3.3.5.
	prcv_msg	ER ercd = prcv_msg (T_MSG **ppk_msg, ID mbxid);			prcv_mbx iprcv_mbx	ER ercd = prcv_mbx (ID mbxid, T_MSG **ppk_msg); ER ercd = iprcv_mbx (ID mbxid, T_MSG **ppk_msg);			
		Packet struct	typedef UW T_MSG;	Start address of packet is received as a message		Packet struct	typedef struct t_msg {		
							VP msghead;	Kernel management area	
							} T_MSG		(Message header in mailbox)
			typedef void *PT_MSG;	32-bit data is received as a message			typedef struct t_msg_pri {		
							T_MSG msgque;	Message header	
							PRI msgpri;	Message priority	
							} T_MSG_PRI;		(Mailbox message header with priority)
8	Refer to mailbox state				Refer to mailbox state				Packet struct and Prm sequence changed.
	ref_mbx	ER ercd = ref_mbx (T_RMBX *pk_rmbx, ID mbxid);			ref_mbx iref_mbx	ER ercd = ref_mbx (ID mbxid, T_RMBX *pk_rmbx); ER ercd = iref_mbx (ID mbxid, T_RMBX *pk_rmbx);			
		Packet struct	typedef struct t_rmbx {			Packet struct	typedef struct t_rmbx {		
			VP exinf;	Extended information			ID wtskid	Wait task ID	
			BOOL_ID wtsk;	Waiting task information			T_MSG *pk_msg;	Next message packet to be received	
			T_MSG *pk_msg;	Start address of the next message packet to be received			} T_RMBX;		
			INT msgcnt;	Number of messages stored in mailbox					
			} T_RMBX;						

4.1.8 Extended Synchronization and Communication (Message Buffer)

	MR32R (µITRON 3.0) Function					MR32R/4 (µITRON 4.0) Function					Change
	SVC	API				SVC	API				
1	Create message buffer					Create message buffer					Packet struct changed.
	cre_mbf	ER ercd = cre_mbf (ID mbfid, T_CMBF *pk_rmbf);				cre_mbf	ER ercd = cre_mbf (ID mbfid, T_CMBF *pk_cmbf);				
		Packet struct	typedef struct t_cmbf {				Packet struct	typedef struct t_cmbf {			
			VP	exinf;	Extended information			ATR	mbfatr;	Message buffer attribute	
			ATR	mbfatr;	Message buffer attribute			UINT	maxmsz;	Maximum message size (bytes)	
			INT	bufsz;	Message buffer size			SIZE	mbfsz;	Message buffer size (bytes)	
			INT	maxmsz;	Maximum message size			VP	mbf;	Start address of message buffer area	
			VP	mbf;	Start address of message buffer area allocated by user			} T_CMBF;			
			} T_CMBF;								
—	—					Create message buffer (assign ID automatically)					New function
	Not available	—				acre_mbf	ER_ID mbfid = acre_mbf (T_CMBF *pk_cmbf);				
2	Delete message buffer					Delete message buffer					
	del_mbf	ER ercd = del_mbf (ID mbfid);				del_mbf	ER ercd = del_mbf (ID mbfid);				
3	Send data to message buffer					Send data to message buffer					Prm type changed.
	snd_mbf	ER ercd = snd_mbf (ID mbfid, VP msg, INT msgsz);				snd_mbf	ER ercd = snd_mbf (ID mbfid, VP msg, UINT msgsz);				
		Prm	INT	msgsz;	Size of the message to send			Prm	UINT	msgsz;	Size of the message to send
4	Send data to message buffer (with timeout)					Send data to message buffer (with timeout)					Prm type changed.
	tsnd_mbf	ER ercd = tsnd_mbf (ID mbfid, VP msg, INT msgsz, TMO tmout);				tsnd_mbf	ER ercd = tsnd_mbf (ID mbfid, VP msg, UINT msgsz, TMO tmout);				
		Prm	INT	msgsz;	Size of the message to send			Prm	UINT	msgsz;	Size of the message to send
5	Send data to message buffer (without wait)					Send data to message buffer (polling)					Prm type changed.
	psnd_mbf	ER ercd = psnd_mbf (ID mbfid, VP msg, INT msgsz);				psnd_mbf	ER ercd = psnd_mbf (ID mbfid, VP msg, UINT msgsz);				
		Prm	INT	msgsz;	Size of the message to send			Prm	UINT	msgsz;	Size of the message to send
6	Receive data from message buffer					Receive data from message buffer					Prm and R_Prm changed
	rcv_mbf	ER ercd = rcv_mbf (VP msg, INT *p_msgsz, ID mbfid);				rcv_mbf	ER_UINT msgsz = rcv_mbf (ID mbfid, VP msg);				
		Prm	VP	msg;	Address to store received message			Prm	ID	mbfid;	Message buffer ID for receiving data
			INT	*p_msgsz;	Received message size				VP	msg;	Pointer to area where received message is to be stored
			ID	mbfid;	Message buffer ID						
		R_Prm	ER	ercd;	Error code			R_Prm	ER_UINT	msgsz;	Received message size (positive value) or error code (negative value)
			INT	*p_msgsz;	Received message size				VP	msg;	Pointer to area storing received message
			VP	msg;	Address storing received message						

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
7	Receive data from message buffer (with timeout)				Receive data from message buffer (with timeout)				Prm and R_Prm changed
	trcv_mbf	ER ercd = trcv_mbf (VP msg, INT *p_msgsz, ID mbfid, TMO tmout);			trcv_mbf	ER_UINT msgsz = trcv_mbf (ID mbfid, VP msg, TMO tmout);			
		Prm	VP	msg;	Address to store received message		Prm	ID	mbfid;	Message buffer ID for receiving data
			INT	*p_msgsz;	Received message size			TMO	tmout;	Timeout value
			ID	mbfid;	Message buffer ID			VP	msg;	Pointer to area where received message is to be stored
			TMO	tmout	Timeout value					
		R_Prm	ER	ercd;	Error code		R_Prm	ER_UINT	msgsz;	Received message size (positive value) or error code (negative value)
			INT	*p_msgsz;	Received message size			VP	msg;	Pointer to area storing received message
			VP	msg;	Address storing received message					
8	Receive data from message buffer (without wait)				Receive data from message buffer (polling)				Prm and R_Prm changed
	prcv_mbf	ER ercd = prcv_mbf (VP msg, INT *p_msgsz, ID mbfid);			prcv_mbf	ER_UINT msgsz = prcv_mbf (ID mbfid, VP msg);			
		Prm	VP	msg;	Address to store received message		Prm	ID	mbfid;	Message buffer ID for receiving data
			INT	*p_msgsz;	Received message size			VP	msg;	Pointer to area where received message is to be stored
			ID	mbfid;	Message buffer ID					
		R_Prm	ER	ercd;	Error code		R_Prm	ER_UINT	msgsz;	Received message size (positive value) or error code (negative value)
			INT	*p_msgsz;	Received message size			VP	msg;	Pointer to area storing received message
			VP	msg;	Address storing received message					
9	Refer to message buffer state				Refer to message buffer state				Packet struct and Prm sequence changed.
	ref_mbf	ER ercd = ref_mbf (T_RMBF *pk_rmbf, ID mbfid);			ref_mbf iref_mbf	ER ercd = ref_mbf (ID mbfid, T_RMBF *pk_rmbf); ER ercd = iref_mbf (ID mbfid, T_RMBF *pk_rmbf);			
		Packet struct	typedef struct t_rmbf {			Packet struct	typedef struct t_rmbf {		
			VP	exinf;	Extended information			ID	stskid;	Send-waiting task ID
			BOOL_ID	wtsk;	Receive-waiting task information			ID	rtskid;	Receive-waiting task ID
			BOOL_ID	stsk;	Send-waiting task information			UINT	smsgcnt;	Number of messages in message buffer
			INT	msgsz;	Size of next message to be received (bytes)			SIZE	fmbfsz;	Size of free buffer (bytes)
			INT	frbufsz;	Size of free buffer (bytes)			} T_RMBF;		
			} T_RMBF;							

4.1.9 Extended Synchronization and Communication (Rendezvous)

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
1	Create port for rendezvous					Create port for rendezvous					Packet struct changed.
	cre_por	ER ercd = cre_por (ID porid, T_CPOR *pk_cpor);				cre_por	ER ercd = cre_por (ID porid, T_CPOR *pk_cpor);				
		Packet struct	typedef struct t_cpor {				Packet struct	typedef struct t_cpor {			
			VP	exinf;	Extended information			ATR	poratr;	Rendezvous port attribute	
			ATR	poratr;	Port attribute			UINT	maxcmsz;	Maximum call message size (bytes)	
			INT	maxcmsz;	Maximum number of messages for call			UINT	maxrmsz;	Maximum replay message size (bytes)	
			INT	maxrmsz;	Maximum number of messages for reply			} T_CPOR;			
			} T_CPOR;								
—	—					Create port for rendezvous (assign ID automatically)					New function
	Not available	—				acre_por	ER_ID porid = acre_por (T_CPOR *pk_cpor);				
2	Delete port for rendezvous					Delete port for rendezvous					
	del_por	ER ercd = del_por (ID porid);				del_por	ER ercd = del_por (ID porid);				
3	Call port for rendezvous					Call rendezvous port					Prm and R_Prm changed.
	cal_por	ER ercd = cal_por (VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz);				cal_por	ER_UINT rmsgsz = cal_por (ID porid, RDVPTN calptn, VP msg, UINT cmsgsz);				
		Prm	VP	msg;	Address storing call message		Prm	ID	porid;	Rendezvous port ID for call	
			INT	*p_rmsgsz;	Address of area to store size of reply message			RDVPTN	calptn;	Bit pattern of call condition	
			ID	porid;	Rendezvous port ID			VP	msg;	Start address of call message (start address to store reply message)	
			UINT	calptn;	Bit pattern indicating call condition			UINT	cmsgsz;	Call message size (bytes)	
			INT	cmsgsz;	Call message size						
		R_Prm	ER	ecrd;	Error code		R_Prm	ER_UINT	rmsgsz;	Error code	

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
4	Call port for rendezvous (with timeout)				Call rendezvous port (with timeout)				Prm, R_Prm, and timeout specification changed. See 3.3.6.
	tcal_por	ER ercd = tcal_por (VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz, TMO tmout);			tcal_por	ER_UINT rmsgsz = tcal_por (ID porid, RDVPTN calptn, VP msg, UINT cmsgsz, TMO tmout);			
		Prm	VP	msg;	Address storing call message	Prm	ID	porid;	Rendezvous port ID for call
			INT	*p_rmsgsz;	Address of area to store size of reply message		RDVPTN	calptn;	Bit pattern of call condition
			ID	porid;	Rendezvous port ID		VP	msg;	Start address of call message (start address to store reply message)
			UINT	calptn;	Bit pattern indicating call condition		UINT	cmsgsz;	Call message size (bytes)
			INT	cmsgsz;	Call message size		TMO	tmout;	Timeout value (for tcal_por)
			TMO	tmout;	Timeout value				
		R_Prm	ER	ecrd;	Error code	R_Prm	ER_UINT	rmsgsz;	Error code
5	Call port for rendezvous (without wait)				—				Deleted. See 3.1.2
	pcal_por	ER ercd = pcal_por (VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz);			Not available	—			
6	Accept port for rendezvous				Accept rendezvous port				Prm and R_Prm changed.
	acp_por	ER ercd = acp_por (RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn);			acp_por	ER_UINT cmsgsz = acp_por (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, VP msg);			
		Prm	RNO	*p_rdvno;	Rendezvous number	Prm	ID	porid;	Rendezvous port ID for accept
			VP	msg;	Start address of area to store call message		RDVPTN	acpptn;	Bit pattern of accept condition
			INT	*p_cmsgsz;	Call message size		VP	msg;	Start address of area to store call message
			ID	porid;	Rendezvous port ID		RDVNO	*p_rdvno;	Pointer to established rendezvous number
			UINT	acpptn;	Bit pattern indicating accept condition				
		R_Prm	ER	ecrd;	Error code	R_Prm	ER_UINT	cmsgsz;	Error code
							RDVNO	*p_rdvno;	Pointer to established rendezvous number

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
7	Accept port for rendezvous (with timeout)				Accept port for rendezvous (with timeout)				Prm and R_Prm changed.
	tacp_por	ER ercd = tacp_por (RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn, TMO tmout);			tacp_por	ER_UINT cmsgsz = tacp_por (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, VP msg, TMO tmout);			
		Prm	RNO	*p_rdvno; Rendezvous number		Prm	ID	porid; Rendezvous port ID for accept	
			VP	msg; Start address of area to store call message			RDVPTN	acpptn; Bit pattern of accept condition	
			INT	*p_cmsgsz; Call message size			VP	msg; Start address of area to store call message	
			ID	porid; Rendezvous port ID			RDVNO	*p_rdvno; Pointer to established rendezvous number	
			UINT	acpptn; Bit pattern indicating accept condition			TMO	tmout; Timeout value	
			TMO	tmout; Timeout value					
		R_Prm	ER	ecrd; Error code		R_Prm	ER_UINT	cmsgsz; Error code	
							RDVNO	*p_rdvno; Pointer to established rendezvous number	
8	Accept port for rendezvous (without wait)				Accept port for rendezvous (polling)				Prm and R_Prm changed.
	pacp_por	ER ercd = pacp_por (RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn);			pacp_por	ER_UINT cmsgsz = pacp_por (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, VP msg);			
		Prm	RNO	*p_rdvno; Rendezvous number		Prm	ID	porid; Rendezvous port ID for accept	
			VP	msg; Start address of area to store call message			RDVPTN	acpptn; Bit pattern of accept condition	
			INT	*p_cmsgsz; Call message size			VP	msg; Start address of area to store call message	
			ID	porid; Rendezvous port ID			RDVNO	*p_rdvno; Pointer to established rendezvous number	
			UINT	acpptn; Bit pattern indicating accept condition					
		R_Prm	ER	ecrd; Error code		R_Prm	ER_UINT	cmsgsz; Error code	
							RDVNO	*p_rdvno; Pointer to established rendezvous number	
9	Forward rendezvous to other port				Forward rendezvous to other port				Prm changed. See 3.3.6.
	fwd_por	ER ercd = fwd_por (ID porid, UINT calptn, RNO rdvno, VP msg, INT cmsgsz);			fwd_por	ER ercd = fwd_por (ID porid, RDVPTN calptn, RDVNO rdvno, VP msg, UINT cmsgsz);			
		Prm	ID	porid; ID of port where rendezvous is forwarded		Prm	ID	porid; ID of port where rendezvous is forwarded	
			UINT	calptn; Bit pattern indicating call condition			RDVPTN	calptn; Bit pattern of call condition	
			RNO	rdvno; Rendezvous number			RDVNO	rdvno; Rendezvous number to be forwarded	
			VP	msg; Address to store call message			VP	msg; Start address of call message (start address to store reply message)	
			INT	cmsgsz; Call message size			UINT	cmsgsz; Call message size (bytes)	

	MR32R (µITRON 3.0) Function					MR32R/4 (µITRON 4.0) Function					Change
	SVC	API				SVC	API				
10	Reply rendezvous					Reply rendezvous					Prm changed.
	rpl_rdv	ER ercd = rpl_rdv (RNO rdvno, VP msg, INT rmsgsz);				rpl_rdv	ER ercd = rpl_rdv (RDVNO rdvno, VP msg, UINT rmsgsz);				
		Prm	RNO	rdvno	Rendezvous number		Prm	RDVNO	rdvno;	Rendezvous number to be terminated	
			VP	msg	Address to store reply message			VP	msg;	Start address of reply message	
			INT	rmsgsz	Reply message size			UINT	rmsgsz;	Reply message size (bytes)	
11	Refer to rendezvous port state					Refer to rendezvous port state					Packet struct and Prm changed.
	ref_por	ER ercd = ref_por (T_RPOR *pk_rpor, ID porid);				ref_por iref_por	ER ercd = ref_por (ID porid, T_RPOR *pk_rpor); ER ercd = iref_por (ID porid, T_RPOR *pk_rpor);				
		Prm	T_RPOR	*pk_rpor;	Start address of struct where port state is to be returned		Prm	ID	porid;	Target rendezvous port ID	
			ID	porid;	Rendezvous port number			T_RPOR	*pk_rpor;	Pointer to packet where rendezvous port state is to be returned	
		Packet struct	typedef struct t_rpor {				Packet struct	typedef struct t_rpor {			
			VP	exinf;	Extended information			ID	ctskid;	Call-waiting task ID	
			BOOL_ID	wtsk;	Accept-waiting task information			ID	atskid;	Accept-waiting task ID	
			BOOL_ID	atsk;	Call-waiting task information			} T_RPOR;			
		} T_RPOR;									
—	—					Refer to rendezvous state					New function
	Not available	—				ref_rdv	ER ercd = ref_rdv (RDVNO rdvno, T_RRDV *pk_rrdv);				
—	—					Refer to rendezvous state (dedicated to handlers)					New function
	Not available	—				iref_rdv	ER ercd = iref_rdv (RDVNO rdvno, T_RRDV *pk_rrdv);				

4.1.10 Interrupt Management

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
1	Define interrupt handler				Define interrupt handler				Prm and packet struct changed
	def_int	ER ercd = def_int (UINT dintno, T_DINT *pk_dint);			def_inh idef_inh	ER ercd = def_inh (INHNO inhno, T_DINH *pk_dinh); ER ercd = idef_inh (INHNO inhno, T_DINH *pk_dinh);			
		Prm	UINT	dintno; Vector number of interrupt handler to be defined		Prm	INHNO	inhno; Vector number of interrupt handler to be defined	
			T_DINT	*pk_dint; Interrupt definition information			T_DINH	*pk_dinh; Pointer to packet storing interrupt handler definition information	
		Packet struct	typedef struct t_dint {			Packet struct	typedef struct t_dinh {		
			ATR	intatr; Interrupt handler attribute			ATR	intatr; Interrupt handler attribute	
			FP	inthdr; Interrupt handler address			FP	inthdr; Interrupt handler entry address	
			} T_DINT;				} T_DINH;		
2	Return from interrupt handler				—				Deleted. (ret_int is necessary only when the application is written in assembly language)
	ret_int	void ret_int();			Not available	—			
3	Disable interrupt and task dispatch				Lock CPU				Meaning of CPU lock changed. See 3.3.1.
	loc_cpu	ER ercd = loc_cpu();			loc_cpu iloc_cpu	ER ercd = loc_cpu(); ER ercd = iloc_cpu();			
4	Enable interrupt and task dispatch				Unlock CPU				Meaning of CPU lock changed. See 3.3.1.
	unl_cpu	ER ercd = unl_cpu();			unl_cpu iunl_cpu	ER ercd = unl_cpu(); ER ercd = iunl_cpu();			

4.1.11 Fixed-Size Memory Pool Management

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
1	Create fixed-size memory pool					Create fixed-size memory pool					Packet struct changed.
	cre_mpf	ER ercd = cre_mpf (ID mpfid, T_CMPF *pk_cmpf);				cre_mpf	ER ercd = cre_mpf (ID mpfid, T_CMPF *pk_cmpf);				
		Packet struct	typedef struct t_cmpf {				Packet struct	typedef struct t_cmpf {			
			VP	exinf;	Extended information			ATR	mpfatr;	Fixed-size memory pool attribute	
			ATR	mpfatr;	Memory pool attribute			UINT	blkcnt;	Number of blocks that can be acquired	
			INT	mpfcnt;	Number of blocks in memory pool			UINT	blksz;	Memory block size (bytes)	
			INT	blfsz;	Block size of fixed-size memory pool			VP	mpf;	Start address of fixed-size memory pool area	
			VP	mpf;	Start address of memory pool area allocated by user			} T_CMPF;			
			} T_CMPF;								
—	—					Create fixed-size memory pool (assign ID automatically)					New function
	Not available	—				acre_mpf	ER_ID mpfid = acre_mpf (T_CMPF *pk_cmpf);				
2	Delete fixed-size memory pool					Delete fixed-size memory pool					
	del_mpf	ER ercd = del_mpf (ID mpfid);				del_mpf	ER ercd = del_mpf (ID mpfid);				
3	Get memory block					Get fixed-size memory block					SVC name and Prm changed
	get_blf	ER ercd = get_blf (VP *p_blf, ID mpfid);				get_mpf	ER ercd = get_mpf (ID mpfid, VP *p_blk);				
		Prm	VP	*p_blf;	Address of area to store the start address of the acquired memory block		Prm	ID	mpfid;	ID of fixed-size memory pool where a memory block is to be acquired	
			ID	mpfid;	Memory pool ID			VP	*p_blk;	Pointer to start address of acquired memory block	
4	Get memory block (with timeout)					Get fixed-size memory block (with timeout)					SVC name and Prm changed
	tget_blf	ER ercd = tget_blf (VP *p_blf, ID mpfid, TMO tmout);				tget_mpf	ER ercd = tget_mpf (ID mpfid, VP *p_blk, TMO tmout);				
		Prm	VP	*p_blf;	Address of area to store the start address of the acquired memory block		Prm	ID	mpfid;	ID of fixed-size memory pool where a memory block is to be acquired	
			ID	mpfid;	Memory pool ID			VP	*p_blk;	Pointer to start address of acquired memory block	
			TMO	tmout;	Timeout value			TMO	tmout;	Timeout value	
5	Get memory block (without wait)					Get fixed-size memory block (polling)					SVC name and Prm changed
	pget_blf	ER ercd = pget_blf (VP *p_blf, ID mpfid);				pget_mpf ipget_mpf	ER ercd = pget_mpf (ID mpfid, VP *p_blk); ER ercd = ipget_mpf (ID mpfid, VP *p_blk);				
		Prm	ID	mpfid;	Memory pool ID		Prm	ID	mpfid;	ID of fixed-size memory pool where a memory block is to be acquired	
			VP	*p_blf;	Address of area to store the start address of the acquired memory block			VP	*p_blk;	Pointer to start address of acquired memory block	

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
6	Release memory block					Release fixed-size memory block					SVC name and Prm changed
	rel_blf	ER ercd = rel_blf (ID mpfid, VP p_blf);				rel_mpf	ER ercd = rel_mpf (ID mpfid, VP blk);				
		Prm	ID	mpfid;	Memory pool ID		Prm	ID	mpfid;	ID of fixed-size memory pool where a memory block is to be returned	
			VP	p_blf;	Start address of memory block to be released			VP	blk;	Start address of memory block to be returned	
7	Release memory block (dedicated to handlers)					Release fixed-size memory block (dedicated to handlers)					SVC name and Prm changed
	irel_blf	ER ercd = irel_blf (ID mpfid, VP p_blf);				irel_mpf	ER ercd = irel_mpf (ID mpfid, VP blk);				
		Prm	ID	mpfid;	Memory pool ID		Prm	ID	mpfid;	ID of fixed-size memory pool where a memory block is to be returned	
			VP	p_blf;	Start address of memory block to be released			VP	blk;	Start address of memory block to be returned	
8	Refer to fixed-size memory pool state					Refer to fixed-size memory pool state					Prm sequence and packet struct changed.
	ref_mpf	ER ercd = ref_mpf (T_RMPF *pk_rmpf, ID mpfid);				ref_mpf iref_mpf	ER ercd = ref_mpf (ID mpfid, T_RMPF *pk_rmpf); ER ercd = iref_mpf (ID mpfid, T_RMPF *pk_rmpf);				
		Prm	typedef struct t_rmpf {				Prm	typedef struct t_rmpf {			
			VP	exinf;	Extended information			ID	wtskid;	ID of task waiting for memory block acquisition	
			BOOL_ID	wtsk;	Waiting task information			UINT	fblkcnt;	Number of available memory blocks	
			INT	frbcnt;	Number of blocks in free space			} T_RMPF;			
			INT	blfsz;	Block size						
			} T_RMPF;								

4.1.12 Variable-Size Memory Pool Management

	MR32R (µITRON 3.0)					MR32R/4 (µITRON 4.0)					
	Function					Function					
	SVC	API				SVC	API				Change
1	Create variable-size memory pool					Create variable-size memory pool					Packet struct changed.
	cre_mpl	ER ercd = cre_mpl (ID mplid, T_CMPL *pk_cmpl);				cre_mpl	ER ercd = cre_mpl (ID mplid, T_CMPL *pk_cmpl);				
		Packet struct	typedef struct t_cmpl {				Packet struct	typedef struct t_cmpl {			
			VP	exinf;	Extended information			ATR	mplatr	Variable-size memory pool attribute	
			ATR	mplatr;	Memory pool attribute			SIZE	mplsz	Size of memory pool to be created	
			INT	mplsz;	Memory pool size			UINT	maxblksz	Maximum size of memory block to be acquired	
			INT	maxblksz;	Maximum size of memory block to be acquired			VP	mpl	Start address of variable-size memory pool area	
			VP	mpf;	Start address of memory pool area allocated by user			} T_CMPL;			
			} T_CMPL;								
—	—					Create variable-size memory pool (assign ID automatically)					New function
	Not available	—				acre_mpl	ER_ID mplid = acre_mpl (T_CMPL *pk_cmpl);				
2	Delete variable-size memory pool					Delete variable-size memory pool					
	del_mpl	ER ercd = del_mpl (ID mplid);				del_mpl	ER ercd = del_mpl (ID mplid);				
3	Get memory block					Get variable-size memory block					SVC name, Prm sequence, and Prm type changed.
	get_blk	ER ercd = get_blk (VP *p_blk, ID mplid, INT blksz);				get_mpl	ER ercd = get_mpl (ID mplid, UINT blksz, VP *p_blk);				
		Prm	INT	blksz;	Size of memory block to be acquired		Prm	UINT	blksz;	Size of memory block to be acquired (bytes)	
4	Get memory block (with timeout)					Get variable-size memory block (with timeout)					SVC name, Prm sequence, and Prm type changed.
	tget_blk	ER ercd = tget_blk (VP *p_blk, ID mplid, INT blksz, TMO tmout);				tget_mpl	ER ercd = tget_mpl (ID mplid, UINT blksz, VP *p_blk, TMO tmout);				
		Prm	INT	blksz;	Size of memory block to be acquired		Prm	UINT	blksz;	Size of memory block to be acquired (bytes)	
5	Get memory block (without wait)					Get variable-size memory block (polling)					SVC name, Prm sequence, and Prm type changed.
	pget_blk	ER ercd = pget_blk (VP *p_blk, ID mplid, INT blksz);				pget_mpl	ER ercd = pget_mpl (ID mplid, UINT blksz, VP *p_blk);				
		Prm	INT	blksz;	Size of memory block to be acquired		Prm	UINT	blksz;	Size of memory block to be acquired (bytes)	
6	Release memory block					Release variable-size memory block					SVC name changed.
	rel_blk	ER ercd = rel_blk (ID mplid, VP blk);				rel_mpl	ER ercd = rel_mpl (ID mplid, VP blk);				

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
7	Refer to variable-size memory pool state				Refer to variable-size memory pool state				Prm sequence and packet struct changed.
	ref_mpl	ER ercd = ref_mpl (T_RMPL *pk_rmpl, ID mplid);			ref_mpl iref_mpl	ER ercd = ref_mpl (ID mplid, T_RMPL *pk_rmpl); ER ercd = iref_mpl (ID mplid, T_RMPL *pk_rmpl);			
		Packet struct	typedef struct t_rmpl {			Packet struct	typedef struct t_rmpl {		
			VP	exinf;	Extended information		ID	wtskid	ID of task waiting for memory block acquisition
			BOOL_ID	wtsk;	Waiting task information		SIZE	fmplsz	Size of free space (bytes)
			INT	frsz;	Total size of free space		UINT	fblksz	Size of maximum available memory block (bytes)
			INT	maxsz;	Size of maximum free area		} T_RMPL;		
			} T_RMPL;						

4.1.13 Time Management

	MR32R (µITRON 3.0) Function				MR32R/4 (µITRON 4.0) Function				Change
	SVC	API			SVC	API			
1	Set system clock				Set system clock				Prm and packet struct changed. See 3.3.7.
	set_tim	ER ercd = set_tim (SYSTIME *pk_tim);			set_tim iset_tim	ER ercd = set_tim (SYSTIM *p_systim); ER ercd = iset_tim (SYSTIM *p_systim);			
		Prm	SYSTIME	*pk_tim; Start address of system clock data to be set		Prm	SYSTIM	*p_systim Pointer to packet indicating system clock to be set	
		Packet struct	typedef struct t_systime {			Packet struct	typedef struct t_systim {		
			H	utime; Upper 16 bits			UH	utime Upper 16 bits	
			UW	ltime; Lower 32 bits			UW	ltime Lower 32 bits	
			} SYSTIME;				} SYSTIM;		
2	Get system clock				Get system clock				Prm and packet struct changed. See 3.3.7.
	get_tim	ER ercd = get_tim (SYSTIME *pk_tim);			get_tim iget_tim	ER ercd = get_tim (SYSTIM *p_systim); ER ercd = iget_tim (SYSTIM *p_systim);			
		Prm	SYSTIME	*pk_tim; Start address of system clock data to be set		Prm	SYSTIM	*p_systim Pointer to packet indicating system clock to be set	
		Packet struct	typedef struct t_systime {			Packet struct	typedef struct t_systim {		
			H	utime; Upper 16 bits			UH	utime Upper 16 bits	
			UW	ltime; Lower 32 bits			UW	ltime Lower 32 bits	
			} SYSTIME;				} SYSTIM;		
—	—				Supply time tick				New function
	Not available	—			isig_tim				
3	Delay task execution for a specified period				Delay task				Prm changed. See 3.3.7.
	dly_tsk	ER ercd = dly_tsk (DLYTIME dlytim);			dly_tsk	ER ercd = dly_tsk (TMO tmout);			
		Prm	DLYTIME	dlytim; Delay time		Prm	TMO	tmout Delay time	
4	Define cyclic handler				Create cyclic handler				Modification made throughout the function. See 3.3.8.
	def_cyc	ER ercd = def_cyc (HNO cycno, T_DCYC pk_dcyc);			cre_cyc	ER ercd = cre_cyc (ID cycid, T_CCYC *pk_ccyc);			
—	—				Create cyclic handler (assign ID automatically)				
	Not available	—			acre_cyc	ER_ID cycid = acre_cyc (T_CCYC *pk_ccyc);			
—	—				Delete cyclic handler				
	Not available	—			del_cyc	ER ercd = del_cyc (ID cycid);			
5	Activate cyclic handler				Start cyclic handler				
	act_cyc	ER ercd = act_cyc (HNO cycno, UINT cycact);			sta_cyc ista_cyc	ER ercd = sta_cyc (ID cycid); ER ercd = ista_cyc (ID cycid);			
					Stop cyclic handler				
					stp_cyc istp_cyc	ER ercd = stp_cyc (ID cycid); ER ercd = istp_cyc (ID cycid);			
6	Refer to cyclic handler state				Refer to cyclic handler state				
	ref_cyc	ER ercd = ref_cyc (T_RCYC *pk_rcyc, HNO cycno);			ref_cyc iref_cyc	ER ercd = ref_cyc (ID cycid, T_RCYC *pk_rcyc); ER ercd = iref_cyc (ID cycid, T_RCYC *pk_rcyc);			

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
—	—	—	Create alarm handler		Modification made throughout the function. See 3.3.9.
	Not available	—	cre_alm	ER ercd = cre_alm (ID almid, T_CALM *pk_calm);	
—	—	—	Create alarm handler (assign ID automatically)		
	Not available	—	acre_alm	ER_ID almid = acre_alm (T_CALM *pk_calm);	
—	—	—	Delete alarm handler		
	Not available	—	del_alm	ER ercd = del_alm (ID almid);	
—	—	—	Start alarm handler		
	Not available	—	sta_alm ista_alm	ER ercd = sta_alm (ID almid, RELTIM almtim); ER ercd = ista_alm (ID almid, RELTIM almtim);	
—	—	—	Stop alarm handler		
	Not available	—	stp_alm istp_alm	ER ercd = stp_alm (ID almid); ER ercd = istp_alm (ID almid);	
7	Refer to alarm handler state		Refer to alarm handler state		
	ref_alm	ER ercd = ref_alm (T_RALM *pk_ralm, HNO almno);	ref_alm iref_alm	ER ercd = ref_alm (ID almid, T_RALM *pk_ralm); ER ercd = iref_alm (ID almid, T_RALM *pk_ralm);	

4.1.14 System Management

	MR32R (µITRON 3.0)				MR32R/4 (µITRON 4.0)				
	Function				Function				
	SVC	API			SVC	API			Change
1	Refer to OS version				Refer to version information				Prm and packet struct changed.
	get_ver	ER ercd = get_ver (T_VER *pk_ver);			ref_ver iref_ver	ER ercd = ref_ver (T_RVER *pk_rver); ER ercd = iref_ver (T_RVER *pk_rver);			
		Prm	T_VER	*pk_ver; Start address of struct where version management information is to be returned		Prm	T_RVER	*pk_rver; Pointer to packet where version management information is to be returned	
		Packet struct	typedef struct t_ver {			Packet struct	typedef struct t_rver {		
			UH	maker; Manufacturer			UH	maker Kernel manufacturer code	
			UH	id; Type number			UH	prid Kernel ID number	
			UH	spver; Specification version			UH	spver ITRON specification version	
			UH	prver; Product version			UH	prver Kernel version	
			UH	prno[4]; Product management information			UH	prno[4] Kernel product management information	
			UH	cpu; CPU information			} T_RVER;		
			UH	var; Variation descriptor					
			} T_VER;						
2	Refer to system state				—				Deleted. See 3.1.2.
	ref_sys	ER ercd = ref_sys (T_RSYS *pk_rsys);			Not available	—			
3	Define exception handler				—				Deleted. See 3.1.2.
	def_exc	ER ercd = def_exc (UINT exckind, T_DEXC *pk_dexc);			Not available	—			

4.1.15 System State Management

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
—	—	—	Refer to context		New function
	Not available	—	sns_ctx	ER ercd = sns_ctx();	
—	—	—	Refer to CPU-locked state		New function
	Not available	—	sns_loc	ER ercd = sns_loc();	
—	—	—	Refer to dispatch-disabled state		New function
	Not available	—	sns_dsp	ER ercd = sns_dsp();	
—	—	—	Refer to dispatch-pended state		New function
	Not available	—	sns_dpn	ER ercd = sns_dpn();	

4.1.16 Extended Functions

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
1	Clear exception mask		—		Deleted. See 3.1.2.
	vclr_ems	ER ercd = vclr_ems (ID tskid);	Not available	—	
2	Set exception mask		—		Deleted. See 3.1.2.
	vset_ems	ER ercd = vset_ems (ID tskid);	Not available	—	
3	Initiate forced exception handler		—		Deleted. See 3.1.2.
	vras_fex	ER ercd = vras_fex (ID tskid, UW exccd);	Not available	—	
—	—		Initialize data queue area		New function
	Not available	—	vrst_dtq	ER ercd = vrst_dtq (ID dtqid);	
4	Clear mailbox		Initialize mailbox area		SVC name changed.
	vrst_msg	ER ercd = vrst_msg (ID mbxid);	vrst_mbx	ER ercd = vrst_mbx (ID mbxid);	
5	Clear message buffer		Initialize message buffer area		
	vrst_mbf	ER ercd = vrst_mbf (ID mbfid);	vrst_mbf	ER ercd = vrst_mbf (ID mbfid);	
6	Release fixed-size memory pool		Initialize fixed-size memory pool area		SVC name changed.
	vrst_blf	ER ercd = vrst_blf (ID blfid);		ER ercd = vrst_mpf (ID mpfid);	
7	Release variable-size memory pool		Initialize variable-size memory pool area		SVC name changed.
	vrst_blk	ER ercd = vrst_blk (ID blkid);		ER ercd = vrst_mpl (ID mplid);	

4.1.17 Extended Functions (Mailbox with Priority)

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
	Function		Function		
	SVC	API	SVC	API	Change
1	Create mailbox with priority			—	Deleted. Ordinary mailbox can achieve equivalent function. See 3.1.1.
	vcre_mbx	ER ercd = vcre_mbx (ID vmbxid, T_CVMBX *pk_rmbf);	Not available	—	
2	Delete mailbox with priority			—	
	vdel_mbx	ER ercd = vdel_mbx (ID vmbxid);	Not available	—	
3	Send message with priority			—	
	vsnd_mbx	ER ercd = vsnd_mbx (ID vmbxid, T_MSG pk_msg);	Not available	—	
4	Send message with priority (dedicated to handlers)			—	
	visnd_mbx	ER ercd = visnd_mbx (ID vmbxid, T_MSG pk_msg);	Not available	—	
5	Receive message with priority (without timeout)			—	
	vrcv_mbx	ER ercd = vrcv_mbx (T_MSG *ppk_msg, ID vmbxid);	Not available	—	
6	Receive message with priority (with timeout)			—	
	vtrcv_mbx	ER ercd = vtrcv_mbx (T_MSG *ppk_msg, ID vmbxid, TMO tmout);	Not available	—	
7	Receive message with priority (without wait)			—	
	vprcv_mbx	ER ercd = vprcv_mbx (T_MSG *ppk_msg, ID vmbxid);	Not available	—	
8	Clear mailbox with priority			—	
	vrst_mbx	ER ercd = vref_mbx (T_RVMBF *pk_rvmbf, ID vmbxid);	Not available	—	
9	Refer to state of mailbox with priority			—	
	vref_mbx	ER ercd = vrst_mbx (ID vmbxid);	Not available	—	

4.2 Error Code Differences

The following list shows the comparison between the error codes for the system calls (service calls) between the MR32R and the MR32R/4.

Table 4.1 Error Code Comparison

	MR32R (µITRON 3.0)		MR32R/4 (µITRON 4.0)		
No.	Error Code (Mnemonic)	Error Code Value	Error Code (Mnemonic)	Error Code Value	Description
1	E_OK	00000000H	E_OK	00000000H	Normal termination
2	E_OBJ	0FFFFFFC1H	E_OBJ	0FFFFFFD7H	Object state is invalid
3	E_QOVR	0FFFFFFB7H	E_QOVR	0FFFFFFD5H	Queuing or nest overflow
4	E_TMOUT	0FFFFFFABH	E_TMOUT	0FFFFFFCEH	Polling failed or timeout
5	E_RLWAI	0FFFFFFAAH	E_RLWAI	0FFFFFFCFH	WAITING state is forcibly cancelled
6	E_NOEXS	0FFFFFFCCH	E_NOEXS	0FFFFFFD6H	Object does not exist
7	E_DLT	0FFFFFFAFH	E_DLT	0FFFFFFCDH	Waiting object deleted
8	E_NOMEM	0FFFFFFF6H	E_NOMEM	0FFFFFFDFH	Memory insufficient
9	EV_RST	0FFFFFF02H	EV_RST	0FFFFFF81H	WAITING state is cancelled by a reset
10	—	—	E_ILUSE	0FFFFFFE4H	Illegal use of service call
11	—	—	E_NOID	0FFFFFFDEH	No ID available

The following shows the service calls for which a new error code is added, an error code is deleted, or an error code is replaced with another code.

Table 4.2 Added or Deleted Error Codes

No.	Service Call	Error Code in MR32R	Error Code in MR32R/4	Description
1	ter_tsk	—	E_ILUSE	Current task ID or TSK_SELF is specified for the target task
2	wai_flg	—	E_ILUSE	Illegal use of service call (a task is waiting for an event flag with the TA_WSGL attribute)
3	twai_flg	—	E_ILUSE	Illegal use of service call (a task is waiting for an event flag with the TA_WSGL attribute)
4	pol_flg	—	E_ILUSE	Illegal use of service call (a task is waiting for an event flag with the TA_WSGL attribute)
5	cre_mbx	E_NOMEM	—	Memory insufficient
6	snd_msg	E_QOVR	—	Queuing or nest overflow
7	isnd_msg	E_QOVR	—	Queuing or nest overflow

Section 5 Difference in Configuration Operation

In the MR32R, configuration files must be created with a text editor. For the MR32R/4, a GUI tool is provided to facilitate the creation of configuration files.

For details, refer to the help file for the MR32R/4 GUI configurator.

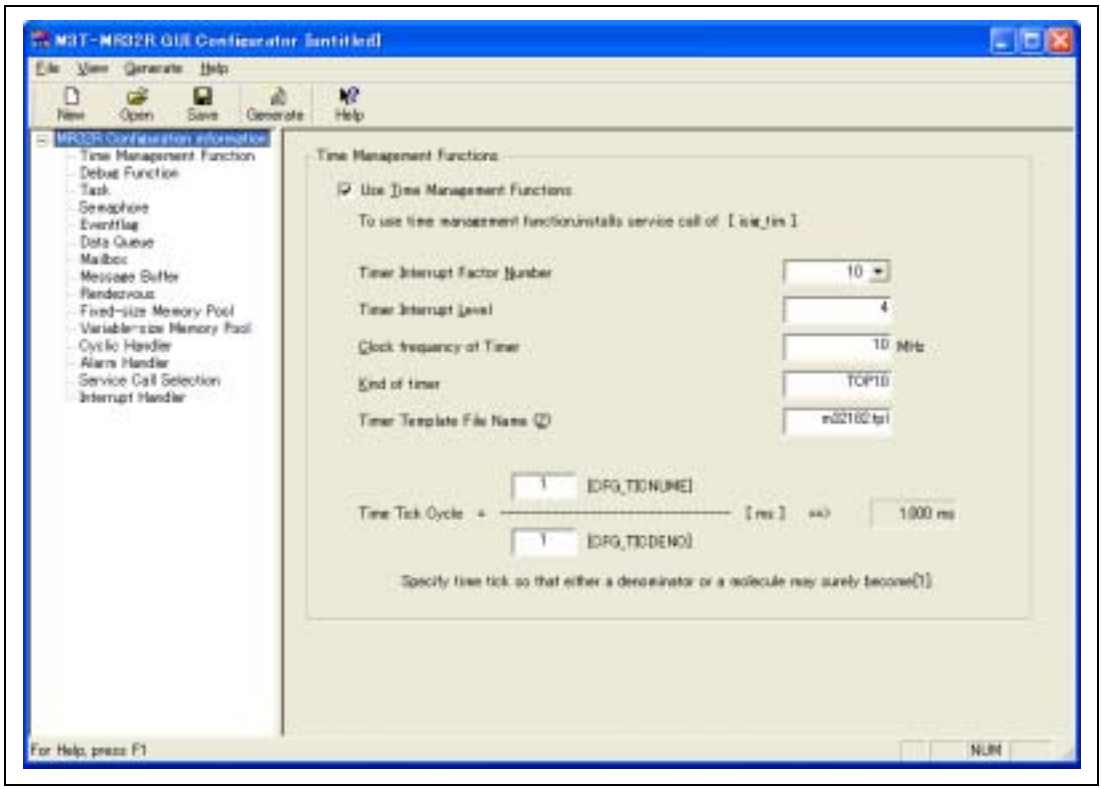

Figure 5.1 MR32R/4 Configurator Window

μITRON Specification OS Transition Manual
Transition Guide from MR32R to MR32R/4

Publication Date: Rev.1.00, Sep. 14, 2005
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Customer Support Department
Global Strategic Communication Div.
Renesas Solutions Corp.

© 2005. Renesas Technology Corp., All rights reserved. Printed in Japan.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS SALES OFFICES

http://www.renesas.com

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

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, United Kingdom
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology Hong Kong Ltd.
7th Floor, North Tower, World Finance Centre, Harbour City, 1 Canton Road, Tsimshatsui, Kowloon, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2730-6071

Renesas Technology Taiwan Co., Ltd.
10th Floor, No.99, Fushing North Road, Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology (Shanghai) Co., Ltd.
Unit2607 Ruijing Building, No.205 Maoming Road (S), Shanghai 200020, China
Tel: <86> (21) 6472-1001, Fax: <86> (21) 6415-2952

Renesas Technology Singapore Pte. Ltd.
1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

Renesas Technology Korea Co., Ltd.
Kukje Center Bldg. 18th Fl., 191, 2-ka, Hangang-ro, Yongsan-ku, Seoul 140-702, Korea
Tel: <82> 2-796-3115, Fax: <82> 2-796-2145

Renesas Technology Malaysia Sdn. Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No.18, Jalan Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: <603> 7955-9390, Fax: <603> 7955-9510

µITRON Specification OS Transition Manual
Transition Guide from MR32R to MR32R/4
Application Note

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ05B0765-0100