User's Manual

RENESAS

R-IN32M4-CL3

Programming Manual (OS edition)

All information of mention is things at the time of this document publication, and Renesas Electronics may change the product or specifications that are listed in this document without a notice. Please confirm the latest information such as shown by website of Renesas

Document number : R18UZ0072EJ0100
Issue date : Oct 30, 2019

Renesas Electronics
www.renesas.com

ARM

Notice

1. 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.
2. 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.
3. 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.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "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 etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.
6. 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.
7. 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 systems manufactured by you.
8. 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.
9. 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. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics 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.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
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.

Instructions for the use of product

In this section, the precautions are described for over whole of CMOS device.
Please refer to this manual about individual precaution.
When there is a mention unlike the text of this manual, a mention of the text takes first priority

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
     In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

How to use this manual

1. Purpose and target readers

This manual is intended for users who wish to understand the functions of Industrial Ethernet network LSI "R-IN32M4-CL2" for designing application of it.

Target users are expected to understand the fundamentals of electrical circuits, logic circuits, and microcomputers.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.
The mark "<R>" means the updated point in this revision. The mark "<R>" let users search for the updated point in this document.

Literature Literature may be preliminary versions. Note, however, that the following descriptions do not indicate "Preliminary".
Some documents on cores were created when they were planned or still under development. So, they may be directed to specific customers. Last four digits of document number (described as ****) indicate version information of each document. Please download the latest document from our web site and refer to it.

Document related to R-IN32M4-CL2

Document name	**Document number**
R-IN32M4-CL3 User's Manual	R18UZ0073EJ****
R-IN32M4-CL3 Programming Manual (Driver edition)	R18UZ0076EJ****
R-IN32M4-CL3 Programming Manual (OS edition)	This manual

Document related to this operating system

Document name	Document number
μITRON4.0 Specification Ver.4.02.00 (ITRON Specification Study Group, TRON Association)	-

The μITRON4.0 specification is the de-fact standard for the real-time kernel with the TRON Association at the center of its development.

Descriptions related to the μITRON4.0 specification in this manual are excerpt from the μITRON4.0 Specification. For more details on the specification, refer to the μITRON4.0 Specification itself.

The specification can be downloaded from the web site of the TRON Forum.

2. Notation of Numbers and Symbols

Weight in data notation: Left is high-order column, right is low-order column

Active low notation:

xxxZ (capital letter Z after pin name or signal name)

or xxx_N (capital letter _N after pin name or signal name)

or xxnx (pin name or signal name contains small letter n)

Note:

explanation of (Note) in the text

Caution:

Item deserving extra attention

Remark:

Supplementary explanation to the text

Numeric notation:

Binary ··· xxxx , xxxxB or n'bxxxx (n bits)

Decimal ··· xxxx

Hexadecimal ··· xxxxH or n'hxxxx (n bits)

Prefixes representing powers of 2 (address space, memory capacity):

K (kilo)··· $2^{10} = 1024$

M (mega)··· $2^{20} = 1024^2$

G (giga)··· $2^{30} = 1024^3$

Data Type:

Word ··· 32 bits

Halfword ··· 16 bits

Byte ··· 8 bits

Table of Contents

List of Figures

List of Tables

1. Overview

This document explains a procedure to use the Real-time OS (μITRON Ver. 4.0) and supporting service call in industrial Ethernet network LSI “R-IN32M4-CL3”.

The combination of the hardware real-time OS technique and OS library provides RTOS functionality for free. It means that hardware real-time OS technology does not require any cost such as license fee or maintenance cost. (however, no guarantee)

1.1 Features of the Hardware Real-time Operating System

The R-IN32M4-CL2 includes a hardware real-time operating system accelerator (HW-RTOS), that realizes faster processing of the real-time operating system. With the HW-RTOS, smoother responsiveness is ensured because the hardware handles objects such as tasks and eventflags and processing such as task scheduling.

The number of objects allowed in this system is given in Table1.1 As shown in the table, the objects semaphore and mutex share a hardware and the number of objects available for them is 128 in total. Which means, if 100 semaphores are to be used, only 28 objects are available to be used as mutex. Note that semaphore and mutex cannot share a same object ID. If IDs 1 to 10 are assigned to semaphore, available IDs for mutex are 11 and greater.

Table1.1 Maximum Number of Objects

object type	Maximum Number
Task number	64
Eventflag number	64
Mailbox number	64
Semaphore number	Total 128
Mutex number	

One of the major features of the HW-RTOS is the hardware interrupt service routine (hardware ISR). This is implemented in the hardware and handles interrupt service routines and some of the service calls run in the routines. With this function, when an interrupt is generated, the HW-RTOS automatically runs the service call previously registered in response to the interrupt. For example, if a service call set_flg is executed in an interrupt routine, the call is run without involving the CPU. The HW-RTOS handles task scheduling for the given service call, realizing service calls with smoother responsiveness to interrupts.

Table1.2 is the list of service calls allowed for the hardware ISR. For the setting procedure, see Section 6.Hardware ISRs.

Table1.2 Service Calls Available for the Hardware ISR

Service call name	Description
set_flg	Set eventflag
sig_sem	Release semaphore resource
rel_wai	Release Task from Waiting
wup_tsk	Wakeup Task

1.2 OS Library

This OS library is software that provides the functionality of service calls of μITRON through control of the HW-RTOS. This document describes the specification of the API functions for RTOS that is realized through the combination of the HW-RTOS and the OS library.

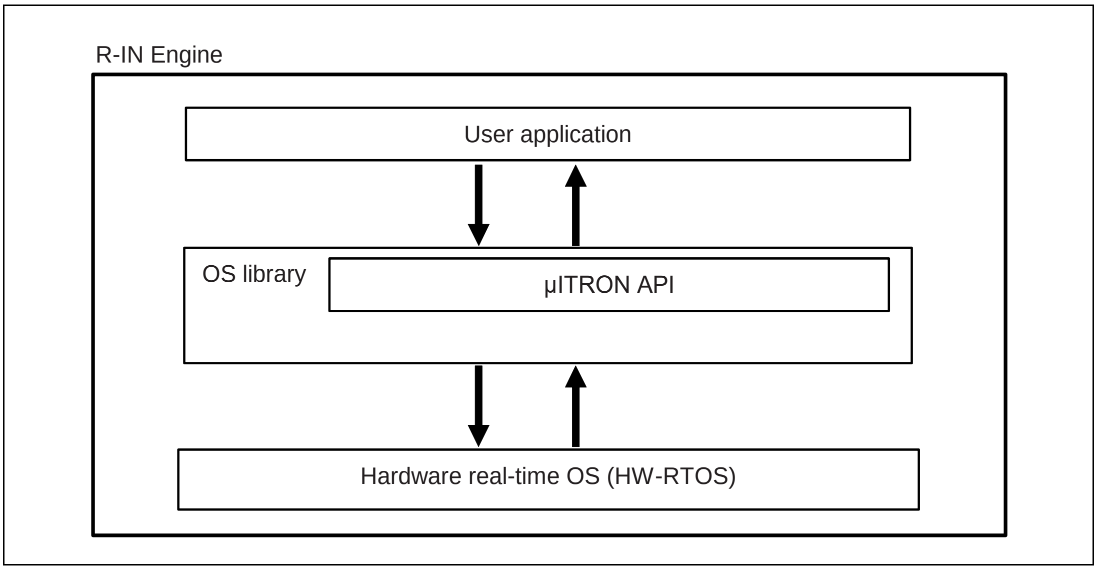

Figure1.1 System Configuration

1.2.1 The Version of OS Library

The target library version for this document is shown below.

File name for OS library	Version
libos.a	2.03

1.3 Supported Service calls

A list of the service calls supported by the R-IN32M4, and compared to those of the standard profile, is given below.

Table1.3 Supported Service call (1/3)

Category	Service Call Name	Descriptions	R-IN32M4 μITRON	μITRON Ver. 4.0 Standard Profile
Task management function	act_tsk	Activates task	-	√
	iact_tsk	Activates task	-	√
	can_act	Cancels task activation requests	-	√
	sta_tsk	Activates task (with a start code)	√	-
	ext_tsk	Terminates invoking task	√	√
	ter_tsk	Terminates task	√	√
	chg_pri	Changes task priority	√	√
	get_pri	References task priority	√	√
Task dependent synchronization function	slp_tsk	Puts task to sleep	√	√
	tslp_tsk	Puts task to sleep (with timeout)	√	√
	wup_tsk	Wakes up task	√	√
	iwup_tsk	Wakes up task	√	√
	can_wup	Cancels task wakeup requests	√	√
	rel_wai	Forcibly releases task from waiting	√	√
	irel_wai	Forcibly releases task from waiting	√	√
	sus_tsk	Forcibly suspends task	-	√
	frsm_tsk	Forcibly resumes suspended task	-	√
	rsm_tsk	Resumes forcibly suspended task	-	√
	dly_tsk	Delays task	-	√
Task exception handling function	ras_tex	Raises task exception handling request	-	√
	iras_tex	Raises task exception handling request	-	√
	dis_tex	Disables task exceptions	-	√
	ena_tex	Enables task exceptions	-	√
	sns_tex	References task exception handling state	-	√
Synchronization and communication functions — Semaphores	cre_sem	Creates semaphore	-	-
	del_sem	Deletes semaphore	√	-
	wai_sem	Acquires semaphore resource	√	√
	pol_sem	Acquires semaphore resource (by polling)	√	√
	twai_sem	Acquires semaphore resource (with timeout)	√	√
	sig_sem	Releases semaphore resource	√	√
	isig_sem	Releases semaphore resource	√	√

Note: √: Available, -: Not available

Table1.3 Supported Service call (2/3)

Category		Service Call Name	Descriptions	R-IN32M4 μITRON	μITRON Ver. 4.0 Standard Profile
Synchronization and communication functions	Eventflags	cre_flg	Creates eventflag	-	-
		del_flg	Deletes eventflag	√	-
		set_flg	Sets eventflag	√	√
		iset_flg	Sets eventflag	√	√
		clr_flg	Clears eventflag	√	√
		wai_flg	Waits for eventflag	√	√
		pol_flg	Waits for eventflag (by polling)	√	√
		twai_flg	Waits for eventflag (with timeout)	√	√
	Data queues	snd_dtq	Sends to data queue	-	√
		psnd_dtq	Sends to data queue (by polling)	-	√
		ipsnd_dtq	Sends to data queue	-	√
		tsnd_dtq	Sends to data queue (with timeout)	-	√
		fsnd_dtq	Forcibly sends to data queue	-	√
		ifsnd_dtq	Forcibly sends to data queue	-	√
		rcv_dtq	Receives from data queue	-	√
		prcv_dtq	Receives from data queue (by polling)	-	√
	Mailboxes	cre_mbx	Creates mailbox	-	-
		del_mbx	Deletes mailbox	√	-
		snd_mbx	Sends mailbox	√	√
		rcv_mbx	Receives mailbox	√	√
		prcv_mbx	Receives mailbox (by polling)	√	√
		trcv_mbx	Receives mailbox (with timeout)	√	√
Extended synchronization and communication functions	MutexesCreates mutex	cre_mtx	Creates mutex	-	-
		del_mtx	Deletes mutex	√	-
		loc_mtx	Locks mutex	√	-
		ploc_mtx	Locks mutex (by polling)	√	-
		tloc_mtx	Locks mutex (with timeout)	√	-
		unl_mtx	Unlocks mutex	√	-

Note: √: Available, -: Not available

Table1.3 Supported Service Calls(3/3)

Category		Service Call Name	Descriptions	R-IN32M4 μITRON	μITRON Ver. 4.0 Standard Profile
Memory pool management functions	Fixed-sized	get_mpf	Acquires fixed-sized memory block	-	√
		pget_mpf	Acquires fixed-sized memory block (by polling)	-	√
		tget_mpf	Acquires fixed-sized memory block (with timeout)	-	√
		rel_mpf	Releases fixed-sized memory block	-	√
Time management functions	System time management	set_tim	Sets system time	√	√
		get_tim	References system time	√	√
		isig_tim	Supplies time tick	-	√
	Cyclic handlers	sta_cyc	Starts cyclic handler operation	-	√
		stp_cyc	Stops cyclic handler operation	-	√
System state management functions		rot_rdq	Rotates task precedence	√	√
		irot_rdq	Rotates task precedence	√	√
		get_tid	References task ID in the RUNNING state	√	√
		iget_tid	References task ID in the RUNNING state	√	√
		loc_cpu	Locks the CPU	√	√
		iloc_cpu	Locks the CPU	-	√
		unl_cpu	Unlocks the CPU	√	√
		iunl_cpu	Unlocks the CPU	-	√
		dis_dsp	Disables dispatching	√	√
		ena_dsp	Enables dispatching	√	√
		sns_ctx	References contexts	-	√
		sns_loc	References CPU locked state	√	√
		sns_dsp	References dispatching disabled state	-	√
		sns_dpn	References dispatch pending state	-	√

Note: √: Available, -: Not available

1.4 Supported Static API Functions

A list of the static API functions supported by the R-IN32M4, compared to those of the standard profile, is given below. For details setting procedure, see Section 5. Static Creation Methods of Objects.

Table1.4 Supported Static API

Category		API Call Name	Description	R-IN32M4 μITRON	μITRON Ver. 4.0 Standard Profile
Task management function		CRE_TSK	Creates task	√	√
Task exception handling function		DEF_TEX	Defines task exception handling routine	–	√
Synchronization and communication functions	Semaphore	CRE_SEM	Creates semaphore	√	√
	Eventflag	CRE_FLG	Creates eventflag	√	√
	Data queue	CRE_DTQ	Creates data queue	–	√
	Mailbox	CRE_MBX	Creates mailbox	√	√
Extended synchronization and communication function	Mutex	CRE_MTX	Creates mutex	√	–
Memory pool management function	Fixed-sized length	CRE_MPF	Creates fixed-sized memory pool	–	√
Time management function	Cyclic handler	CRE_CYC	Creates cyclic handler	–	√
Interrupt management function		DEF_INH	Defines interrupt handler	√	√
System configuration management functions		DEF_EXC	Defines CPU exception handler	–	√
		ATT_INI	Attaches initialization routine	–	√

Note: √: Available, -: Not available

1.5 Differences from the Standard Profile

	μITRON for R-IN32M4	Standard profile of theμITRON Ver. 4.0
Queuing of activation requests	Not supported	Supported
Task priority levels	1 to 15	1 to 16
Maximum number of semaphore resources	31	65535 or more
Message priority levels	1 to 7	1 to 16 (greater than or equal to the number of task priority levels)
Eventflag attribute	TA_WSGL is not supported. Only TA_WMUL is supported.	TA_WSGL

The static API functions are extended from the standard profile. Among the service calls dedicated to task contexts, the functions listed below are also usable from non-task contexts.

sta_tsk	wup_tsk	pol_flg	rot_rdq
ter_tsk	can_wup	sig_sem	get_tid
chg_pri	rel_wai	set_flg	prcv_mbx
get_pri	pol_sem	clr_flg	snd_mbx

1.6 Operating Modes of the Processor

The ARM processor core supports two operating modes (thread and handler) and two access modes (privileged and non-privileged). In this system, they are used as follows.

・Task

Tasks are handled in thread mode with privileged access. Switching to non-privileged mode is not supported. Operations in this mode use process stack.

・Non-Task

Non-tasks, such as interrupt handler and dispatching process are handled in handler mode with privileged access. Operations in this mode use main stack.

Note that this core does not utilize the memory protection unit (MPU) because it does not support the management of memory protection.

1.7 OS Time Tick

Because OS tick is executed by hardware, the interrupt for tick doesn't happen. The timer for tick is implemented in hardware RTOS, and it is configured during boot sequence of hardware RTOS.

Tick period is set to 1 millisecond by default, but it can be changed by invoking the function "hwos_set_tick_time" before RTOS boot.

1.8 Development Environments

The software development tools which were used to build OS library are described as below.

Table1.5 Software Development Tools

Tool Vendor	Compiler
IAR	Embedded Workbench for ARM V8.42.1 (IAR Systems)

2. Procedure for Software Development

A series of procedures for software development is described here.

2.1 Design flow

Figure2.1 shows the correlation between files.

Please refer to R-IN32M4-CL3 Programming manual (Driver edition) for detail about the file structure.

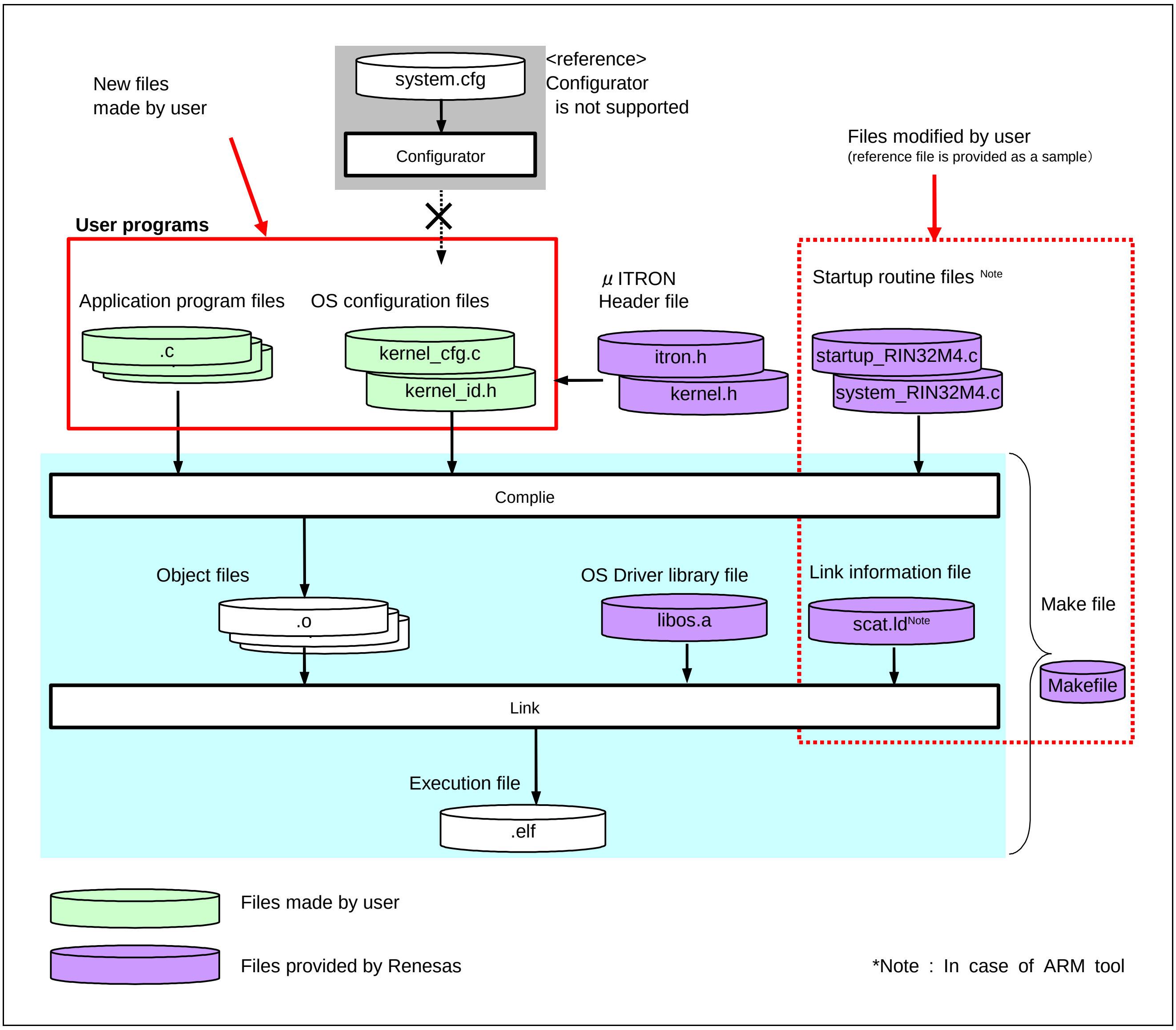

Figure2.1 Correlation Diagram between Files

2.2 Creation of OS Configuration Files

The objects to be statically created, the interrupt handlers, and the hardware ISRs are defined in the kernel_cfg.c file. For details on definition method, see Section 5. Static Creation Methods of Objects and 6. Hardware ISRs.

2.3 Starting the Operating System

Executing the function hwos_setup during startup procedure initializes the operating system and starts up the system. hwos_init is an empty function for keeping backward compatibility with previous versions.

2.3.1 Setting Up the Operating System

hwos_setup

(1) Synopsis

Sets up the hardware operating system.

(2) C language format

ER hwos_setup(void);

(3) Parameter

None

(4) Function

This function makes configurations of the following resources for the operating system based on the OS configuration file kernel_cfg.c.

- Stack pointers
  The addresses for the stack areas are assigned to the stack areas for tasks in order from the lowest address.
  The main stack pointer (MSP) is set to the highest address of the stack area for interrupts.
  The process stack pointer (PSP) is set as the stack pointer for the first task to be started.
- Semaphores
- Eventflags
- Mailboxes
- Mutexes
- Kernel Interrupts
- Hardware ISRs

(5) Returned Value

Return Value	Meaning
ER_OK	Successful setup

2.3.2 Initial Settings of the Operating System

hwos_init

(1) Synopsis

An empty function for keeping backward compatibility with the operating system of the R-IN32M3 series.

(2) C language format

ER hwos_init(void);

(3) Parameter

None

(4) Function

None

(5) Returned Value

Return Value	Meaning
ER_OK	None

2.4 Reboot OS

If reboot OS in order to update firmware or else purpose, should do the following procedure.

- Reset HW-RTOS itself by using the reset register for HW-RTOS module
- Disable interrupt and clear pending interrupt factor by using the interrupt controller for CPU (NVIC)
- Invoke hwos_setup

The example code for reboot is shown below.

```
/* Release register protection */
RIN_SYS->SYSPCMD = 0x000000A5;
RIN_SYS->SYSPCMD = 0x00000001;
RIN_SYS->SYSPCMD = 0x0000FFFE;
RIN_SYS->SYSPCMD = 0x00000001;

/* Assert reset */
RINACS->RTOSRST.LONG = 0x00000000;
/* Disable interrupt */
NVIC_DisableIRQ(HWRTOS_IRQn);
NVIC_ClearPendingIRQ(HWRTOS_IRQn);
/* Deassert reset */
RINACS->RTOSRST.LONG = 0x00000001;

/* Set register protection */
RIN_SYS->SYSPCMD = 0x00000000;

/* Start HW-RTOS */
hwos_setup();
```

Figure2.2 The example code for reboot procedure

2.5 Cautionary Notes

Cautionary notes on using the operating system library are given below.

- Do not access the area for the peripheral registers (address range from 0x40080000 to 0x4008FFFF) of the HW-RTOS. This area may be accessed by the debugger during debugging. To avoid this, make sure not to display this area in the display of memory and to omit this area from the target for monitoring.
- Hardware ISRs of HW-RTOS do not stop even when a program is stopped at a breakpoint by a debugger.
- When the Ethernet MAC is used, commands sent by its hardware function cannot be executed and result in an error if the system is in the dispatching disabled state or the CPU locked state. To avoid this, enable dispatching and unlock the CPU to allow the execution of commands.
- OS library refers to both of the SVCall interrupt handler and the HWRTOS interrupt handler, which are placed in vector table with symbol name "__vector_table". These two interrupt vectors should have the following function name for a compile.
  - SVCall interrupt vector：SVC_Handler
  - HWRTOS interrupt vector：HWRTOS_IRQHandler
- During OS boot, the vector table is overwritten by the registered interrupt handlers in "static_interrupt_table". Because OS library refers to the vector table with symbol name "__vector_table" at that time, this symbol name should not be modified.
- In the state that SVCall is disabled, OS service call cannot be invoked. (e.g. invoking service call after invoking __disable_irq, etc.)

3. Data Types and Macros

This section gives details on the data types, configurations, and macros used when issuing the service calls provided by this software.

3.1 Data Types

Data types of the parameters to be specified when issuing the service calls are listed below.

Table3.1 Software Development Tools(Development Environment)

Macro	Type	Meaning
B	signed char	Signed 8-bit integer
H	signed short	Signed 16-bit integer
W	signed long	Signed 32-bit integer
UB	unsigned char	Unsigned 8-bit integer
UH	unsigned short	Unsigned 16-bit integer
UW	unsigned long	Unsigned 32-bit integer
VB	char	8-bit value with unknown data type
VH	short	16-bit value with unknown data type
VW	long	32-bit value with unknown data type
VP	void *	Pointer to an unknown data type
FP	void (*)	Processing unit start address (pointer to a function)
INT	signed int	Signed 32-bit integer
UINT	unsigned int	Unsigned 32-bit integer
BOOL	INT	Boolean value (TRUE or FALSE)
FN	INT	Function code
ER	INT	Error code (returned value from a service call)
ID	INT	Management object ID number
ATR	UINT	Management object attribute
STAT	UINT	Management object state
MODE	UINT	Service call operational mode
PRI	INT	Priority of the task or the message
SIZE	UINT	Memory area size (in bytes)
TMO	INT	Waiting time for a task (in milliseconds)
RELTIM	UINT	Relative time (in milliseconds)
SYSTIM	UINT	System time (in milliseconds)
VP_INT	VP	Pointer to an unknown data type, or a signed 32-bit integer
ER_BOOL	ER	Error code or a Boolean value (TRUE or FALSE)
ER_ID	ER	Error code or a management object ID number
ER_UINT	ER	Error code or an unsigned integer
FLGPTN	UINT	Bit pattern of the eventflag

3.2 Constants

Constants defined in this system are listed below.

Table3.2 Constants (General)

Constants	Value	Meaning
NULL	0	Invalid pointer
TRUE	1	True
FALSE	0	False
E_OK	0	Normal completion

Table3.3 Constants (Object Attribute)

Constants	Value	Meaning
TA_NULL	0	Object attribute not specified.
Attributes specified when task/handler creation		
TA_HLNG	0x00	Start a processing unit through a high-level language interface.
TA_ASM	0x01	Start a processing unit through an assembly language interface.
TA_ACT	0x02	Task is executable after its creation.
TA_RSTR	0x04	Restricted task (not supported)
Attributes specified when Synchronization/ Extended synchronization communication functions(Semaphores, event flags, mailboxes, mutexes) creation		
TA_TFIFO	0x00	Task wait queue is in FIFO order.
TA_TPRI	0x01	Task wait queue is in task priority order.
Attribute specified when Synchronization communication functions(event flag) creation		
TA_WSGL	0x00	Only one task is allowed to be in the waiting state for the eventflag (not supported).
TA_WMUL	0x02	Multiple tasks are allowed to be in the waiting state for the eventflag.
TA_CLR	0x04	Eventflag is cleared when a task is released from the waiting state for that eventflag.
Attribute specified when Synchronization communication functions(mailboxes) creation		
TA_MFIFO	0x00	Message queue is in FIFO order.
TA_MPRI	0x02	Message queue is in message priority order.
Attribute specified when Extended synchronization communication functions(mutexes) creation		
TA_INHERIT	0x02	Mutex uses the priority inheritance protocol (not supported).
TA_CEILING	0x03	Mutex uses the priority ceiling protocol (not supported).
Attribute specified when generated period handler (Not supported)		
TA_STA	0x02	Cyclic handler is in an operational state after the creation (not supported).
TA_PHS	0x04	Cyclic handler is activated preserving the activation phase (not supported).

Table3.4 Constants (with Timeout)

Constants	Value	Meaning
TMO_POL	0	Polling
TMO_FEVR	-1	Waiting forever
TMO_NBLK	-2	Non-blocking (not supported)

Table3.5 Constants (Service Call Operating Mode)

Constants	Value	Meaning
TWF_ANDW	0x00	AND waiting condition for an eventflag
TWF_ORW	0x01	OR waiting condition for an eventflag

Table3.6 Constants (with Timeout)

Constants	Value	Meaning
TSK_SELF	0	Specifies invoking task.
TSK_NONE	0	No applicable task (not used)
TPRI_SELF	0	Specifies the base priority of the invoking task.
TPRI_INI	0	Specifies the initial priority of the task.

Table3.7 Constants (Error Code)

Constants	Value	Meaning
E_SYS	-5	System error
E_RSATR	-11	Reserved attribute
E_PAR	-17	Parameter error
E_ID	-18	Invalid ID number
E_CTX	-25	Context error
E_ILUSE	-28	Illegal service call use
E_OBJ	-41	Object state error
E_NOEXS	-42	Non-existent object
E_QOVR	-43	Queue overflow
E_RLWAI	-49	Forced release from waiting
E_TMOUT	-50	Polling failure or timeout
E_DLT	-51	Waiting object deleted
E_UNKNOWN	-99	Unknown error (illegal response by the HW-RTOS due to hardware errors)

3.3 Data structure

3.3.1 Structures Defined for μITRON V4

T_CTSK

Synopsis

Information required for creating a task.

Declaration

```
typedef struct t_ctsk {
        ATR      tskatr;     /*!< Task attribute                      */
        VP_INT   exinf;      /*!< Task extended information           */
        FP       task;       /*!< Task start address                  */
        PRI      itskpri;    /*!< Task initial priority               */
        SIZE     stksz;      /*!< Task stack size                     */
        VP       stk;        /*!< Base address of task stack space    */
} T_CTSK;
```

Members

Member		Description
ATR	tskatr	Task attribute When TA_ACT is specified, a task is activated when it is created. TA_ACT (2): Create a task in an activated state The following definitions can also be specified but make no difference to operation. TA_HLNG(0): Start a processing unit through a high-level language interface (not used) TA_ASM(1): Start a processing unit through an assembly language interface (not used)
VP_INT	exinf	Task extended information (the argument given to the task when TA_ACT is specified)
FP	task	Task start address
PRI	itskpri	Task initial priority An integer value (1 to 15)*: Task priority number
SIZE	stksz	Task stack size (in bytes)
VP	stk	Base address of task stack space NULL (0): Start address allocated by the kernel (recommended) Value: The value specified as the start address

Note. This range is 1 to 16 In the μITRON 4.0 specification.

T_CSEM

Synopsis

Information required for creating a semaphore.

Declaration

```
typedef struct t_csem {
        ATR     sematr;     /*!< Semaphore attribute                  */
        UINT    isemcnt;    /*!< Initial semaphore resource count     */
        UINT    maxsem;     /*!< Maximum semaphore resource count     */
} T_CSEM;
```

Members

Member		Description			
ATR	sematr	Semaphore attribute			
		Task wait queue:	TA_TFIFO	0x00	In FIFO order
			TA_TPRI	0x01	In task priority order
UINT	isemcnt	Initial semaphore resource count (maximum number: the value set in the maxsem member)			
UINT	maxsem	Maximum semaphore resource count (maximum number: 31)			

T_CFLG

Synopsis

Information required for creating an eventflag.

Declaration

```
typedef struct t_cflg {
        ATR       flgatr;       /*!< Eventflag attribute                        */
        FLGPTN    iflgptn;      /*!< Initial value of eventflag bit pattern     */
} T_CFLG;
```

Members

Member		Description			
ATR	flgatr	Eventflag attribute			
		Task waiting queue:	TA_TFIFO	0x00	In FIFO order
			TA_TPRI	0x01	In task priority order
			TA_WMUL	0x02	Multiple tasks are allowed to be in the waiting state
			TA_CLR	0x04	Eventflag's bit pattern is cleared when a task is released from the waiting state
FLGPTN	iflgptn	Initial value of the eventflag bit pattern (valid bit length: 16 bits)			

Restriction

TA_WSGL is not supported in this system. If TA_WSGL is specified, the eventflag behaves the same as it does with TA_WMUL.

T_CMBX

Synopsis

Information required for creating a mailbox.

Declaration

```
typedef struct t_cmbx {
        ATR     mbxatr;     /*!< Mailbox attribute                          */
        PRI     maxmpri;    /*!< Maximum message priority                   */
        VP      mprihd;     /*!< Start address of the area for message
                                 queue headers for each message priority    */
} T_CMBX;
```

Members

Member		Description			
ATR	mbxatr	Mailbox attribute			
		Task waiting queue:	TA_TFIFO	0x00	In FIFO order
			TA_TPRI	0x01	In task priority order
		Message Queue:	TA_MFIFO	0x00	In FIFO order
			TA_MPRI	0x02	In Message priority order
PRI	maxmpri	Highest message priority (allowable range: 1 to 7)			
VP	mprihd	Start address of the area for message queue headers for each message priority (not used)			

Caution: The mailbox attribute TA_MPRI cannot be used by default. To use this attribute, see the function "hwos_set_mpri_operation" in section 8. Utility Functions.

T_CMTX

Synopsis

Information required for creating a mutex.

Declaration

```
typedef struct t_cmtx {
        ATR     mtxatr;     /*!< Mutex attribute                 */
        PRI     ceilpri;    /*!< Mutex ceiling priority          */
} T_CMTX;
```

Members

Member		Description			
ATR	mtxatr	Mutex attribute			
		Task waiting queue:	TA_TFIFO	0x00	In FIFO order
			TA_TPRI	0x01	In task priority order
PRI	ceilpri	Mutex ceiling priority (not used)			

Caution: The mutex attributes TA_INHERIT and TA_CEILING are not supported.

T_DINH

Synopsis

The packet format of the information required for defining an interrupt handler.

Declaration

```
typedef struct t_dinh {
        ATR     inhatr;     /*!< Interrupt handler attribute          */
        FP      inthdr;     /*!< Interrupt handler start address      */
} T_DINH;
```

Members

Member		Description
ATR	inhatr	Interrupt handler attribute (not used)
FP	inthdr	Interrupt handler start address

T_MSG

Synopsis

Message header information.

Declaration

```
typedef struct t_msg {
            struct t_msg   *next;        /*!< Start address of the message packet from the mailbox    */
} T_MSG;
```

Members

Member		Description
t_msg	*next	Start address of the message packet from the mailbox

3.3.2 R-IN32M4-Specific Structures

TSK_TBL

Synopsis

Information required for static creation of a task. (the argument of CRE_TSK is defined as this structure)

Declaration

```
typedef struct task_table {
        ID        id;         /*!< Task ID                              */
        T_CTSK    t_ctsk;     /*!< Task creation information packet     */
} TSK_TBL;
```

Members

Member		Description
ID	id	ID number of the task to be created in static method An integer value (1 to 64): ID of the specified task
T_CTSK	t_ctsk	Task creation information

SEM_TBL

Synopsis

Information required for static creation of a semaphore. (the argument of CRE_SEM is defined as this structure)

Declaration

```
typedef struct semaphore_table {
        ID        id;          /*!< Semaphore ID                             */
        T_CSEM    pk_csem;     /*!< Semaphore creation information packet    */
} SEM_TBL;
```

Members

Member		Description	
ID	id	The semaphore ID to be created in static method	
		An integer value (1 to 128):	ID of the specified semaphore (This should not overlap with the mutex IDs)
T_CSEM	pk_csem	Semaphore creation information	

FLG_TBL

Synopsis

Information required for static creation of a flag. (the argument of CRE_FLG is defined as this structure)

Declaration

```
typedef struct flag_table {
        ID          id;           /*!< Eventflag ID                              */
        T_CFLG      pk_cflg;      /*!< Eventflag creation information packet    */
} FLG_TBL;
```

Members

Member		Description
ID	id	The Flag ID to be created in static method An integer value (1 to 64): ID of the specified flag
T_CFLG	pk_cflg	Flag creation information

MBX_TBL

Synopsis

Information required for static creation of a mailbox. (the argument of CRE_MBX is defined as this structure)

Declaration

```
typedef struct mailbox_table {
        ID        id;          /*!< Mailbox ID                           */
        T_CMBX    pk_cmbx;     /*!< Mailbox creation information packet  */
} MBX_TBL;
```

Members

Member		Description
ID	id	The mailbox ID to be created in static method An integer value (1 to 64): ID of the specified mailbox
T_CMBX	pk_cmbx	Mailbox creation information

MTX_TBL

Synopsis

Information required for static creation of a mutex. (the argument of CRE_MTX is defined as this structure)

Declaration of the structure

```
typedef struct mutex_table {
        ID          id;         /*!< Mutex ID                              */
        T_CMTX      pk_cmtx;    /*!< Mutex creation information packet     */
} MTX_TBL;
```

Members

Member		Description	
ID	id	The mutex ID to be created in static method	
		An integer value (1 to 128):	ID of the specified mutex (This should not overlap with the semaphore IDs.)
T_CMTX	pk_cmtx	Mutex creation information	

INT_TBL

Synopsis

Information required for creating an interrupt handler. (the argument of DEF_INH is defined as this structure)

Declaration

```
typedef struct interrupt_table {
        INHNO    id;          /*!< Interrupt handler number to be defined       */
        T_DINH   pk_dinh;     /*!< Pointer to the packet containing the
                                   interrupt handler definition information     */
} INT_TBL;
```

Members

Member		Description
INHNO	inhno	The interrupt number for which an interrupt handler is to be created An integer value (0 to 152): Specified interrupt number*
T_DINH	pk_dinh	Pointer to the packet that contains the interrupt handler definition information

Note: **This should be the value obtained by subtracting 16 from the exception number because interrupts are assigned to the exception numbers from 16. The given interrupt numbers are the exception numbers from 16, which are replaced by the numbers from 0. For the applicable interrupts, refer to List of Interrupts of the User's Manual.**

HWISR_TBL

Synopsis

Information required for creating a hardware ISRs.

Declaration

```
typedef struct hwist_table {
        INHNO   inhno;          /*!< Interrupt handler number to be defined    */
        UINT    hwisr_syscall;  /*!< System call                               */
        ID      id;             /*!< Target ID                                 */
        FLGPTN  setptn;         /*!< Bit pattern (only set_flg)                */
} HWISR_TBL;
```

Members

Member		Description
INHNO	inhno	Number of the interrupt for which a hardware ISR is to be created. An integer value (0 to 152): Target interrupt number*
UINT	hwisr_syscall	Service calls that are automatically executed on generation of the corresponding interrupt. HWISR_SET_FLG (1) set_flg() HWISR_SIG_SEM (2) sig_sem() HWISR_REL_WAI (3) rel_wai() HWISR_WUP_TSK (4) wup_tsk()
ID	id	ID of the specified object for which the service call is automatically executed on generation of the corresponding interrupt.
FLGPTN	setptn	Bit pattern to set (valid only when set_flg is specified)

3.4 Global Variables

The global variables used in this library are listed below. Make sure that you do not use these variable symbols in an application.

Table3.8 Global Variables

Variable Name
HWRTOS_Sbt
HWRTOS_Sit
HWRTOS_IntTable

4. Service calls

4.1 Task Management Function

The service calls for task management are listed below.

Table4.1 Task Management Functions

Service Call Name	Description	Range of Objects that Can Issue This Call
sta_tsk	Activates task (with a start code).	Tasks and non-tasks
ext_tsk	Terminates invoking task.	Tasks
ter_tsk	Forcibly terminates task.	Tasks and non-tasks
chg_pri	Changes task priority.	Tasks and non-tasks
get_pri	References task priority.	Tasks and non-tasks

Specification of this function is given below.

Table4.2 Task Management Functions

No.	Item	Content
1	Task ID numbers	1 to 64
2	Task priority levels	1 to 15
3	Wakeup request count	63
4	Task attribute	TA_HLNG: Activated through the high-level language interface (not used) TA_ASM: Activated through the assembler language interface (not used) TA_ACT : Task is executable after its creation

sta_tsk

Synopsis

Activates task.

C Language format

ER sta_tsk(ID tskid, VP_INT stacd);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			An integer value (1 to 64):	ID of specified task
I	VP_INT	stacd	Start code of the task	

Function

This call moves the task specified by tskid from the DORMANT state to the READY state.

The extended information is set in stacd and given to the specified task.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_OBJ	-41	Object state error (specified task is not in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

ext_tsk

Synopsis

Terminates task.

C Language format

void ext_tsk(void);

Parameter

None

Function

This call moves the invoking task from the RUNNING state to the DORMANT state.

This call does not return to its origin unless an error is detected. The errors include issuing this call while the CPU is locked, dispatching is disabled, or from an interrupt handler.

Return parameter

None

Restriction

If a task is terminated while its mutexes remain unlocked, they cannot be unlocked later. Be sure to unlock the mutexes before terminating a task.

Caution

Do not issue this call while the CPU is locked, dispatching is disabled, or from an interrupt handler. Otherwise, the operation is not guaranteed.

ter_tsk

Synopsis

Forcibly terminates task.

C Language API

ER ter_tsk(ID tskid);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			An integer value (1 to 64):	ID of specified task

Function

This call forcibly moves the task specified by tskid to the DORMANT state.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	Execution in the CPU locked state
E_ILUSE	-28	Illegal service call use (specified task is an invoking task)
E_OBJ	-41	Object state error (specified task is not in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

Restriction

If a task is terminated while its mutexes remain unlocked, they cannot be unlocked later. Be sure to unlock the mutexes before terminating a task.

chg_pri

Synopsis

Changes task priority.

C Language format

ER chg_pri(ID tskid, PRI tskpri);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			TSK_SELF (0):	ID of the invoking task
			An integer value (1 to 64):	ID of the specified task
I	PRI	tskpri	New base priority of the task*1	
			TPRI_INI (0):	Initial priority of the specified task
			An integer value (1 to 15) *2:	Base priority of the specified task

Note1. The base priority and the current priority are always the same because this system does not support the priority control facilities, including priority inheritance.

Note2. This range is 1 to 16 in the μITRON4.0 specification.

Function

This call changes the base priority of the task specified by tskid to the priority value specified by tskpri.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tskpri is invalid)
E_ID	-18	Invalid ID number (tskid is invalid or unusable) TSK_SELF is specified from an interrupt handler
E_CTX	-25	The call was invoked while the CPU is locked
E_OBJ	-41	Object state error (specified task is in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

Restriction

When chg_pri () service call is called for the task waiting for resources with TA_TFIFO attribution, order of the task moves to the last of the queue. In case of μITRON (ver4.03) specification, order does not change.

get_pri

Synopsis

References task priority.

C Language format

ER get_pri(ID tskid, PRI *p_tskpri);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			TSK_SELF (0):	ID of the invoking task
			An integer value (1 to 64):	ID of the specified task
O	PRI	*p_tskpri	Current Priority of the specified task	

Function

This call looks up the current priority of the task specified by tskid and returns the value through p_tskpri.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Null pointer is specified in p_tskpri
E_ID	-18	Invalid ID number (tskid is invalid or unusable) TSK_SELF is specified from an interrupt handler
E_CTX	-25	The call was invoked while the CPU is locked
E_OBJ	-41	Object state error (specified task is in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

4.2 Task Dependent Synchronization Functions

The service calls for task dependent synchronization function are listed below.

Table4.3 Task Dependent Synchronization Functions

Service Call Name	Description	Range of Objects that Can Issue This Call
slp_tsk	Puts task to sleep	Tasks
tslp_tsk	Puts task to sleep (with timeout)	Tasks
wup_tsk	Wakes up task	Tasks and non-tasks
iwup_tsk	Wakes up task	Non-tasks
can_wup	Cancels task wakeup request	Tasks and non-tasks
rel_wai	Releases task from waiting	Tasks and non-tasks
irel_wai	Releases task from waiting	Non-tasks

Specification of this function is given below.

Table4.4 Task Dependent Synchronization Function Specification

No.	Item	Content
1	Wakeup request count for the task	63

slp_tsk

Synopsis

Puts task to sleep.

C Language format

ER slp_tsk(void);

Parameter

None

Function

This call moves the invoking task from the RUNNING state to the sleeping state.

However, if wakeup requests are queued, that is, if the wakeup request count for the invoking task is other than 0x0, the count is decremented by 1 and the invoking task continues execution.

slp_tsk() has the same functionality as tslp_tsk(TMO_FEVR).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)

tslp_tsk

Synopsis

Puts task to sleep (with timeout)

C Language format

ER tslp_tsk(TMO tmout);

Parameter

I/O	Parameter		Description	
I	TMO	tmout	Specified timeout	
			TMO_FEVR(-1)	Wait forever(same processing as slp_tsk())
			TMO_POL(0)	Polling
			An integer value	Waiting time in milliseconds

Function

This call moves the invoking task from the RUNNING state to the sleeping state.

However, if wakeup requests are queued, that is, if the wakeup request count for the invoking task is other than 0x0, the count is decremented by 1 and the invoking task continues execution.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tmout is invalid)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout

wup_tsk / iwup_tsk

Synopsis

Wakes up task.

C Language format

ER wup_tsk(ID tskid);

ER iwup_tsk(ID tskid);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			TSK_SELF(0)	ID of the invoking task
			An integer value(1 to 64)	ID of the specified task

Function

These calls move the task specified by tskid from the sleeping state to the READY state.

However, if the task is not in the sleeping state when a call is issued, the wakeup request for the task is queued, that is, 0x1 is added to the count and the invoking task is not executed.

If the count exceeds the maximum possible count, an error code E_QOVR is returned.

Remark : The maximum wakeup request count is 63.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable) TSK_SELF is specified from an interrupt handler
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (iwup_tsk only)
E_OBJ	-41	Object state error (specified task is in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)
E_QOVR	-43	Queue overflow (wakeup request count exceeded 63)

can_wup

Synopsis

Cancels task wakeup request.

C Language format

ER_UINT can_wup(ID tskid);

Parameter

I/O	Parameter		Description	
I	ID	tskid	Task ID	
			TSK_SELF(0)	ID of the invoking task
			An integer value(1 to 64)	ID of the specified task

Function

This call cancels all queued wakeup requests for the task specified by tskid and clears the wakeup request count to 0. The value returned is the count before it was cleared.

Return value

Macro	Value	Meaning
-	0 or a positive integer	Queued wakeup request count
E_ID	-18	Invalid ID number (tskid is invalid or unusable) TSK_SELF is specified from an interrupt handler.
E_CTX	-25	The call was invoked while the CPU is locked.
E_OBJ	-41	Object state error (specified task is in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

rel_wai / irel_wai

Synopsis

Release task from waiting.

C Language format

ER rel_wai(ID tskid);

ER irel_wai(ID tskid);

Parameter

I/O	Parameter		Description
I	ID	tskid	Task ID
			An integer value(1 to 64) ID of the specified task

Function

These calls forcibly release the task specified by tskid from the WAITING states, that are, the states of waiting for a semaphore, an eventflag, or a message.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (irel_wai only)
E_OBJ	-41	Object state error (specified task is not in the WAITING state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

4.3 Synchronization and Communication Functions (Semaphore)

The service calls for synchronization and communication function (semaphore) are listed below.

Table4.5 Synchronization and Communication Function (Semaphore)

Service Call Name	Description	Range of Objects that Can Issue This Call
del_sem	Deletes semaphore.	Tasks
wai_sem	Acquires semaphore resource.	Tasks
pol_sem	Acquires semaphore resource (by polling).	Tasks and non-tasks
twai_sem	Acquires semaphore resource (with timeout).	Tasks
sig_sem	Releases semaphore resource.	Tasks and non-tasks
isig_sem	Releases semaphore resource.	Non-tasks

Specification of this function is given below.

Table4.6 Synchronization and Communication Function (Semaphore) Specification

No.	Item	Content
1	Semaphore ID numbers	1 to 128 (shared with mutexes)
2	Maximum semaphore resource count	31
3	Supported attributes	TA_TFIFO: Task wait queue is in FIFO order TA_TPRI: Task wait queue is in priority order

del_sem

Synopsis

Deletes semaphore.

C Language format

ER del_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	Semaphore ID An integer value(1 to 128)　ID of the specified semaphore

Function

This call deletes the semaphore with its ID specified by semid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler
E_NOEXS	-42	Non-existent object

wai_sem

Synopsis

Acquires semaphore resource.

C Language format

ER wai_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	Semaphore ID An integer value(1 to 128) ID of the specified semaphore

Function

This call acquires one resource from the semaphore specified by semid.
If the resource count of the specified semaphore is 0, the invoking task is moved to the semaphore waiting state.
wai_sem(smid) has the same functionality as twai_sem(semid, TMO_FEVR).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (semaphore is deleted while waiting)

pol_sem

Synopsis

Acquires semaphore resource (by polling).

C Language format

ER pol_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	Semaphore ID An integer value(1 to 128) ID of the specified semaphore

Function

This call acquires one resource from the semaphore specified by semid.

If the resource count of the specified semaphore is 0, the invoking task is not moved to the semaphore waiting state and the result will be failure of polling.

pol_sem(semid) has the same functionality as twai_sem(semid, TMO_POL).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked.
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

twai_sem

Synopsis

Acquires semaphore resource (with timeout).

C Language format

ER twai_sem(ID semid, TMO tmout);

Parameter

I/O	Parameter		Description	
I	ID	semid	Semaphore ID	
			An integer value(1 to 128)	ID of the specified semaphore
I	TMO	tmout	Specified timeout	
			TMO_POL (0)	Polling (same processing as pol_sem())
			TMO_FEVR (-1)	Wait forever (same processing as wai_sem())
			An integer value	Waiting time in milliseconds

Function

This call acquires one resource from the semaphore specified by semid.

If the resource count of the specified semaphore is 0, the invoking tasks is moved to the semaphore waiting state.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tmout is invalid)
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler. (It is invokable from an interrupt handler when TMO_POL is specified)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout or polling failure
E_DLT	-51	Waiting object deleted (semaphore is deleted while waiting).

sig_sem / isig_sem

Synopsis

Release semaphore resource

C Language format

ER sig_sem(ID semid);

ER isig_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	Semaphore ID An integer value(1 to 128) ID of the specified semaphore

Function

These calls release one resource from the semaphore specified by semid.

If there are any tasks waiting for the specified semaphore, the task at the head of the semaphore's wait queue is released from waiting.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (isig_sem only)
E_NOEXS	-42	Non-existent object (specified semaphore is not registered)
E_QOVR	-43	Queue overflow (release will exceed the maximum resource count, 31)

4.4 Synchronization and Communication Functions (Eventflag)

The service calls for synchronization and communication function (eventflag) are listed below.

Table4.7 Synchronization and Communication Function (Eventflag)

Service Call Name	Description	Range of Objects that Can Issue This Call
del_flg	Deletes eventflag.	Tasks
set_flg	Sets eventflag.	Tasks and non-tasks
iset_flg	Sets eventflag.	Non-tasks
clr_flg	Clears eventflag.	Tasks and non-tasks
wai_flg	Waits for eventflag.	Tasks
pol_flg	Waits for eventflag (by polling).	Tasks and non-tasks
twai_flg	Waits for eventflag (with timeout).	Tasks

Specification of this function is given below.

Table4.8 Synchronization and Communication Function (Eventflag) Specification

No.	Item	Content
1	Eventflag ID numbers	1 to 64
2	Number of bits in an eventflag	16 bits
3	Supported attributes	TA_TFIFO: Task wait queue is in FIFO order TA_TPRI: Task wait queue is in priority order TA_WMUL: Multiple tasks are allowed to be in the waiting state. TA_CLR: Eventflag is cleared when a task is released from the waiting state for that eventflag.

del_flg

Synopsis

Deletes eventflag.

C Language format

ER del_flg(ID flgid);

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to be deleted An integer value(1 to 64) ID of the specified eventflag

Function

This call deletes the eventflag with its ID specified by flgid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.
E_NOEXS	-42	Non-existent object

set_flg / iset_flg

Synopsis

Set evnetflag.

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	flgid	ID of the eventflag to be set An integer value(1 to 64) ID of the specified eventflag
I	FLGPTN	setptn	Bit pattern to be set (16 lower-order bits are effective)

Function

These calls set the bit pattern specified by setptn to the eventflag specified by flgid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (setptn is invalid or 1 is set to bit 16 or higher)
E_ID	-18	Invalid ID number (flgid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (iset_flg only)
E_NOEXS	-42	Non-existent object

clr_flg

Synopsis

Clears eventflag.

C Language format

ER clr_flg(ID flgid, FLGPTN clrptn);

Parameter

I/O	Parameter		Description
I	ID	flgid	ID of the eventflag to be set An integer value(1 to 64) ID of the specified eventflag
I	FLGPTN	setptn	Bit pattern to be cleared (16 lower-order bits are effective)

Function

This call clears the bits in the eventflag specified by flgid that correspond to the bits in clrptn having a value of 0. This call differs from set_flg in that it does not return an error when 1 is set to bit 16 or higher in clrptn.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked.
E_NOEXS	-42	Non-existent object

wai_flg

Synopsis

Waits for eventflag.

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to wait for An integer value(1 to 64) ID of the specified eventflag
I	FLGPTN	waiptn	Wait bit pattern (16 lower-order bits are effective) (an error is returned if 1 is set in bit 16 or higher, or if 0x0000 is set in the effective bits)
I	MODE	wfmode	Wait mode TWF_ANDW (0) AND waiting condition for an eventflag TWF_ORW (1) OR waiting condition for an eventflag
O	FLGPTN	*p_flgptn	Pointer to the location where the bit pattern is stored on release from waiting

Function

This call causes the invoking task to wait until the bit pattern of the eventflag specified by flgid satisfies the waiting conditions specified by waiptn and wfmode.
wai_flg(~) has the same functionality as twai_flg(~, TMO_FEVR).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn or wfmode is invalid)
E_ID	-18	Invalid ID (flgid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (specified eventflag is deleted while waiting)

pol_flg

Synopsis

Waits for eventflag (by polling).

C Language format

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

Parameter

I/O	Parameter		Description		
I	ID	flgid	ID number of the eventflag to wait for		
				An integer value(1 to 64)	ID of the specified eventflag
I	FLGPTN	waiptn	Wait bit pattern (16 lower-order bits are effective) (an error is returned if 1 is set in bit 16 or higher, or if 0x0000 is set in the effective bits)		
I	MODE	wfmode	Wait mode		
				TWF_ANDW (0)	AND waiting condition for an eventflag
				TWF_ORW (1)	OR waiting condition for an eventflag
O	FLGPTN	*p_flgptn	Pointer to the location where the bit pattern is stored on release from waiting		

Function

This call polls the bit pattern of the eventflag specified by flgid to see whether it satisfies the waiting conditions specified by waiptn and wfmode.

pol_flg(~) has the same functionality as twai_flg(~, TMO_POL).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn or wfmode is invalid)
E_ID	-18	Invalid ID (flgid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked.
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

twai_flg

Synopsis

Waits for eventflag (with timeout).

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to wait for An integer value(1 to 64) ID of the specified eventflag
I	FLGPTN	waiptn	Wait bit pattern (16 lower-order bits are effective) (an error is returned if 1 is set in bit 16 or higher, or if 0x0000 is set in the effective bits)
I	MODE	wfmode	Wait mode TWF_ANDW (0) AND waiting condition for an eventflag TWF_ORW (1) OR waiting condition for an eventflag
O	FLGPTN	*p_flgptn	Pointer to the location where the bit pattern is stored on release from waiting
I	TMO	tmout	Specified timeout TMO_POL (0) Polling (same processing as pol_flg()) TMO_FEVR (1) Wait forever (same processing as wai_flg()) An integer value Waiting time in milliseconds

Function

This call causes the invoking task to wait until the bit pattern of the eventflag specified by flgid satisfies the waiting conditions specified by waiptn and wfmode.

*p_flgptn holds the bit pattern of the eventflag when the conditions are met.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn is invalid)
E_ID	-18	Invalid ID
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler. (It is invokable from an interrupt handler when tmo_pol is specified)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout or polling failure
E_DLT	-51	Waiting object deleted (specified eventflag is deleted while waiting).

4.5 Synchronization and Communication Function (Mailbox)

The service calls for synchronization and communication function (mailbox) are listed below.

Table4.9 Synchronization and Communication Function (Mailbox)

Service Call Name	Description	Range of Objects that Can Issue This Call
del_mbx	Deletes mailbox	Tasks
snd_mbx	Sends to mailbox	Tasks and non-tasks
isnd_mbx	Sends to mailbox	Non-tasks
rcv_mbx	Receives from mailbox	Tasks
prcv_mbx	Receives from mailbox (by polling)	Tasks and non-tasks
trcv_mbx	Receives from mailbox (with timeout)	Tasks

Specification of this function is given below.

Table4.10 Synchronization and Communication Function (Mailbox) Specification

No.	Item	Content
1	Mailbox ID numbers	1 to 64
2	Message priority levels	1 to 7
3	Message queue count	192
4	Supported attributes	TA_TFIFO: Task wait queue is in FIFO order TA_TPRI: Task wait queue is in priority order TA_MFIFO: Message queue is in FIFO order. TA_MPRI: Message queue is in priority order (with functionality restrictions).

del_mbx

Synopsis
Deletes mailbox.

C Language format
ER del_mbx(ID mbxid);

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox to be deleted An integer value(1 to 64) ID of the specified mailbox

Function
This call deletes the mailbox with its ID specified by mbxid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID (mbxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.
E_NOEXS	-42	Non-existent object

snd_mbx / isnd_mbx

Synopsis

Send to mailbox.

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox to be sent An integer value(1 to 64) ID of the specified mailbox
I	T_MSG	*pk_msg	Start address of the message packet to be sent to the mailbox

Function

These calls send messages whose start address is specified by pk_msg to the mailbox specified by mbxid.

An implementation-dependent error code E_RSATR is added for this system. This error is returned if message packets are sent to a mailbox for which use of the TA_MPRI attribute has been disabled by using the utility function hwos_set_mpri_operation.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_RSATR	-11	Message was sent to a mailbox that was generated with the TA_MPRI attribute while HWOS_DISABLE_MPRI is specified by using the hwos_set_mpri_operation function.
E_PAR	-17	Parameter error (pk_msg or the message priority is invalid)
E_ID	-18	Invalid ID (mbxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (isnd_mbx only).
E_NOEXS	-42	Non-existent object
E_QOVR	-43	Queue overflow (message queue count exceed the maximum number 192)

rcv_mbx

Synopsis

Receive from Mailbox.

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox for which a message is received. An integer value(1 to 64) ID of the specified mailbox
O	T_MSG	*ppk_msg	Pointer to the location where the start address of the message packet received from the mailbox is stored.

Function

This call receives a message from the mailbox specified by mbxid and returns its start address through ppk_msg. If there are no messages in the specified mailbox, the invoking task is moved to the receiving waiting state for the mailbox.

rcv_mbx(~) has the same functionality as trcv_mbx(~, TMO_FEVR).

An implementation-dependent error code E_RSATR is added for this system. This error is returned if message packets are received from a mailbox for which use of the TA_MPRI attribute has been disabled by using the utility function hwos_set_mpri_operation.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_RSATR	-11	Message was sent to a mailbox that was generated with the TA_MPRI attribute while HWOS_DISABLE_MPRI is specified by using the hwos_set_mpri_operation function.
E_PAR	-17	Parameter error (ppk_msg is invalid)
E_ID	-18	Invalid ID (mbxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (specified mailbox was deleted while waiting)

prcv_mbx

Synopsis

Receives from mailbox (by polling).

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox for which a message is received. An integer value(1 to 64) ID of the specified mailbox
O	T_MSG	*ppk_msg	Pointer to the location where the start address of the message packet received from a mailbox is stored.

Function

This call receives a message from the mailbox specified by mbxid and returns its start address through ppk_msg.

If there are no messages in the specified mailbox, the invoking task is not moved to the waiting state and the result will be failure of polling.

prcv_mbx(~) has the same functionality as trcv_mbx(~, TMO_POL).

An implementation-dependent error code E_RSATR is added for this system. This error is returned if message packets are received from a mailbox for which use of the TA_MPRI attribute has been disabled by using the utility function hwos_set_mpri_operation.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_RSATR	-11	Message was received (by polling) from a mailbox that was generated with the TA_MPRI attribute while HWOS_DISABLE_MPRI is specified by using the hwos_set_mpri_operation function.
E_PAR	-17	Parameter error (ppk_msg is invalid)
E_ID	-18	Invalid ID (mbxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked.
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

trcv_mbx

Synopsis

Receives from mailbox (with timeout).

C Language format

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox from which a message is received. An integer value(1 to 64) ID of the specified mailbox
O	T_MSG	*ppk_msg	Pointer to the location where the start address of the message packet received from the mailbox is stored.
I	TMO	tmout	Specified timeout TMO_POL (0) Polling (same processing as prcv_mbx ()) TMO_FEVR (-1) Wait forever (same processing as rcv_mbx()) An integer value Waiting time in milliseconds

Function

This call receives a message from the mailbox specified by mbxid and return its start address through ppk_msg.

If there are no messages in the specified mailbox, the invoking task is moved to the message waiting state.

An implementation-dependent error code E_RSATR is added for this system. This error is returned if message packets are received from a mailbox for that use of the TA_MPRI attribute has been disabled by using the utility function hwos_set_mpri_operation.

This call is invokable from state of dispatch disable if TMO_POL is specified.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_RSATR	-11	Message was received (with timeout) from a mailbox that was generated with the TA_MPRI attribute while HWOS_DISABLE_MPRI is specified by using the hwos_set_mpri_operation function.
E_PAR	-17	Parameter error (ppk_msg or tmout is invalid)
E_ID	-18	Invalid ID (mbxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout or polling failure
E_DLT	-51	Waiting object deleted (specified mailbox deleted while waiting).

4.6 Extended Synchronization and Communication Function (Mutex)

The service calls for extended synchronization and communication function (mutex) are listed below.

Table4.11 Extended Synchronization and Communication Function (Mutex)

Service Call Name	Description	Range of Objects that Can Issue This Call
del_mtx	Deletes mutex	Tasks
loc_mtx	Locks mutex	Tasks
ploc_mtx	Locks mutex (by polling)	Tasks
tloc_mtx	Locks mutex (with timeout)	Tasks
unl_mtx	Unlocks mutex	Tasks

Specification of this function is given below.

Table4.12 Extended Synchronization and Communication Function (Mutex) Specification

No.	Item	Content
1	Mutex IDs	1 to 128 (shared with semaphores)
2	Supported attributes	TA_TFIFO: Task wait queue is in FIFO order TA_TPRI: Task wait queue is in priority order

del_mtx

Synopsis

Deletes mutex.

C Language format

ER del_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be deleted An integer value(1 to 128) ID of the specified mutex

Function

This call deletes the mutex with its ID specified by mtxid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID (mtxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.
E_NOEXS	-42	Non-existent object

loc_mtx

Synopsis

Locks mutex.

C Language format

ER loc_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked An integer value(1 to 128)　ID of the specified mutex

Function

This call locks the mutex specified by mtxid.

If the specified mutex is locked by another task, the invoking task is moved to the mutex waiting state.

loc_mtx(mtxid) has the same functionality as tloc_mtx(mtxid, TMO_FEVR).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID (mtxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.
E_ILUSE	-28	Illegal service call use (already locked by the invoking task)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (specified mutex deleted while waiting).

ploc_mtx

Synopsis

Locks mutex (by polling).

C Language format

ER ploc_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked An integer value(1 to 128) ID of the specified mutex

Function

This call locks the mutex specified by mtxid.

If the specified mutex is locked by another task, the invoking tasks is not moved to the mutex waiting state and the result will be failure of polling.

ploc_mtx(mtxid) has the same functionality as tloc_mtx(mtxid, TMO_POL).

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID (mtxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.
E_ILUSE	-28	Illegal service call use (already locked by the invoking task)
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

tloc_mtx

Synopsis

Locks mutex (with timeout).

C Language format

ER tloc_mtx(ID mtxid, TMO tmout);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked An integer value(1 to 128) ID of the specified mutex
I	TMO	tmout	Specified timeout TMO_POL (0) Polling (same processing as ploc_mtx()) TMO_FEVR (-1) Waiting forever (same processing as loc_mtx()) An integer value Waiting time in milliseconds

Function

This call locks the mutex specified by mtxid.

If the specified mutex is locked by another task, the invoking task is moved to the mutex waiting state.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tmout is invalid)
E_ID	-18	Invalid ID (mtxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked, dispatching is disabled, or from an interrupt handler.It is invokable in the dispatching disabled state if tmo_pol is specified.
E_ILUSE	-28	Illegal service call use (already locked by the invoking task)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout
E_DLT	-51	Waiting object deleted (specified mutex deleted while waiting).

unl_mtx

Synopsis

Unlocks mutex.

C Language format

ER unl_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be unlocked An integer value(1 to 128) ID of the specified mutex

Function

This call unlocks the mutex specified by mtxid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_ID	-18	Invalid ID (mtxid is invalid or unusable)
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.
E_ILUSE	-28	Illegal service call use (the invoking task does not have the specified mutex locked)
E_NOEXS	-42	Non-existent object

Restriction

In the original specification, mutexes such as ext_tsk and ter_tsk remain locked by a task when it is terminated will be unlocked later. However, in this system, locked mutexes are not unlocked after a task is terminated. So, be sure to unlock the mutexes before terminating a task.

4.7 System Time Management Functions

The service calls for system time management function are listed below.

Table4.13 System Time Management Function

Service Call Name	Description	Range of Objects that Can Issue This Call
set_tim	Sets system time	Tasks and non-tasks
get_tim	References system time	Tasks and non-tasks

Specification of this function is given below.

Table4.14 System Time Management Function Specification

No.	Item	Content
1	System time value	Unsigned 32-bit value
2	System time unit*	1[ms](default) 10 [us] ~ 100 [ms] can be set in 1 [us] precision.
3	Initial value of the system time (at initial start-up)	0x00000000

Note. The System time unit means tick period. It can be changed by calling hwos_set_tick_time function before HW-RTOS setup.

set_tim

Synopsis

Sets system time.

C Language format

ER set_tim(SYSTIM *p_systim);

Parameter

I/O	Parameter		Description
I	SYSTIM	*p_systim	Pointer to the location where the information of the time to be set for the system is stored.

Function

This call sets the system time to the value specified through p_systim.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_systim is invalid)
E_CTX	-25	The call was invoked while the CPU is locked.

get_tim

Synopsis

References system time.

C Language format

ER get_tim(SYSTIM *p_systim);

Parameter

I/O	Parameter		Description
O	SYSTIM	*p_systim	Pointer to the location where the information of the current system time is stored.

Function

This call returns the current system time through p_systim.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_systim is invalid)
E_CTX	-25	The call was invoked while the CPU is locked.

4.8 System State Management Functions

The service calls for system state management function are listed below.

Table4.15 System State Management Function

Service Call Name	Description	Range of Objects that Can Issue This Call
rot_rdq	Rotates task precedence	Tasks and non-tasks
irot_rdq	Rotates task precedence	Non-tasks
get_tid	References task ID in the RUNNING state	Tasks and non-tasks
iget_tid	References task ID in the RUNNING state	Non-tasks
loc_cpu	Transitions to the CPU locked state	Tasks
unl_cpu	Releases the CPU locked state	Tasks
sns_loc	References the CPU locked state	Tasks and non-tasks
dis_dsp	Disables dispatching	Tasks
ena_dsp	Enables dispatching	Tasks

rot_rdq / irot_rdq

Synopsis

Rotate task precedence.

C Language format

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

Parameter

I/O	Parameter		Description	
I	PRI	tskpri	Priority of the task whose precedence is rotated	
			TPRI_SELF (0)	Specify the base priority of the invoking task to be rotated
			An integer value (1 to 15)	Priority of the task specified for rotation

Function

These calls rotate the precedence of the tasks with the priority specified by tskpri. More specifically, the task with the highest precedence in the ready queue, whose priority is specified by tskpri, will have the lowest precedence among the tasks with the same priority after the precedence rotation. The next task in the queue will be executed.

If tskpri is TPRI_SELF (0), the ready queue of the base priority of the invoking task will be rotated.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tskpri is invalid or TPRI_SELF is specified from an interrupt handler)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (irot_rdq only).

get_tid / iget_tid

Synopsis

Reference task ID in the RUNNING state.

C Language format

ER get_tid(ID *p_tskid);

ER iget_tid(ID *p_tskid);

Parameter

I/O	Parameter		Description
O	ID	*p_tskid	ID number of the task in the RUNNING state

Function

These calls reference the ID number of the tasks in the RUNNING state and return it to p_tskid.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_tskid is invalid)
E_CTX	-25	The call was invoked while the CPU is locked. The call was issued by a task (iget_tid only).

Restriction

If these calls are issued while an idle task is running, the ID of the task defined as an idle task is returned instead of TSK_NONE (= 0).

loc_cpu

Synopsis

Transitions to the CPU locked state.

C Language format

ER loc_cpu(void);

Parameter

None

Function

This call moves the system to the CPU locked state. In this state, kernel management interrupts and task dispatching are disabled. In other words, the system can exclusively run programs except for the handler for the kernel management interrupt.

Issuable service calls in this state are limited to loc_cpu, unl_cpu, and sns_loc.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_CTX	-25	The call was invoked from an interrupt handler.

Restriction

This call cannot be invoked from an interrupt handler.

unl_cpu

Synopsis

Releases the CPU locked state.

C Language format

ER unl_cpu(void);

Parameter

None

Function

This call releases the system from the CPU locked state.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_CTX	-25	The call was invoked from an interrupt handler.

Restriction

This call cannot be invoked from an interrupt handler.

sns_loc

Synopsis

References the CPU locked state.

C Language format

BOOL sns_loc(void);

Parameter

None

Function

This call gets the CPU locked state.

Return value

Macro	Value	Meaning
TRUE	1	The CPU is locked.
FALSE	0	The CPU is not locked.

dis_dsp

Synopsis

Disables dispatching.

C Language format

ER dis_dsp(void);

Parameter

None

Function

This call moves the system to the dispatching disabled state.

In this state, task scheduling is disabled and the system can run a program exclusively against other tasks.

If this call is made while dispatching is disabled, it does not wait in a queue. Thus, no processing nor error handling will be performed.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.

ena_dsp

Synopsis
Enables dispatching.

C Language format
ER ena_dsp(void);

Parameter
None

Function
This call moves the system to the dispatching enabled state.

Return value

Macro	Value	Meaning
E_OK	0	Normal completion
E_CTX	-25	The call was invoked while the CPU is locked or from an interrupt handler.

5. Static Creation Methods of Objects

5.1 Creation Task

When the system is started by the execution of hwos_setup() in the startup routine, tasks are created based on the information written in the static_task_table array area reserved by the kernel.

This array is defined in the TSK_TBL structure. Task IDs and the members of the T_CTSK structure are listed in the table.

The kernel recognizes TASK_TBL_END (-1) set in tskid as the end of the table.

```
//-------------------------------------
// Task information
//-------------------------------------
const TSK_TBL static_task_table[] = {
// CRE_TSK( tskid,          {tskatr,          exinf,   task,           itskpri, stksz,  stk});
            {ID_TASK_INIT,  {TA_HLNG | TA_ACT, 0,       (FP)init_task,  1,       0x400,  NULL}},
            {ID_TASK_MAIN,  {TA_HLNG | TA_ACT, 0,       (FP)main_task,  2,       0x400,  NULL}},
            {ID_TASK_IDLE,  {TA_HLNG | TA_ACT, 0,       (FP)idle_task,  15,      0x100,  NULL}},
            {TASK_TBL_END,  {0,                0,       (FP)NULL,       0,       0,      NULL}}
};
```

Figure5.1 Configuration Example of the static_task_table Array

Caution: This operating system does not have an idle task in the kernel, so one should be defined in the application. Figure 5.2 shows an example of the definition of an idle task.

```
void idle_task(int exinf)
{
    while (1) {
        __NOP();
    }
}
```

Figure5.2 Configuration Example of an Idle Task

Caution: If two or more tasks with the same priority are created with TA_ACT specified, the tasks are not placed in the ready state in the order of the array. This order is controlled when tasks are created with TA_ACT not specified and invoked by the sta_tsk service call. Only static creation is possible and dynamic creation after starting up the system is not possible. Also, this operation requires at least one task to be created with TA_ACT specified.

5.2 Creating Semaphore

When the system is started by the execution of hwos_setup() in the startup routine, semaphores are created based on the information written in the static_semaphore_table array area reserved by the kernel.

This array is defined in the SEM_TBL structure. Semaphore IDs and the members of the T_CSEM structure are listed in the table.

The kernel recognizes SEMAPHORE_TBL_END (-1) set in semid as the end of the table.

```
//-------------------------------------
// Semaphore information
//-------------------------------------
const SEM_TBL static_semaphore_table[] = {
// CRE_SEM(  semid,              {sematr,          isemcnt, maxsem});
             {ID_APL_SEM1,       {TA_TFIFO,        0,       1}},
             {SEMAPHORE_TBL_END, {0,               0,       0}}
};
```

Figure 5.3 Configuration Example of the static_semaphore_table Array

5.3 Creating Eventflag

When the system is started by the execution of hwos_setup() in the startup routine, eventflags are created based on the information written in the static_eventflag_table array area reserved by the kernel.

This array is defined in the FLG_TBL structure. Eventflag IDs and the members of the T_CFLG structure are listed in the table.

The kernel recognizes EVENTFLAG_TBL_END (-1) set in flgid as the end of the table.

```
//-------------------------------------
// Eventflag information
//-------------------------------------
const FLG_TBL static_eventflag_table[] = {
// CRE_FLG(  flgid,              {flgatr,                        iflgptn});
             {ID_APL_FLG1,       {TA_TFIFO | TA_WMUL | TA_CLR,   0}},
             {EVENTFLG_TBL_END,  {0,                             0}}
};
```

Figure5.4 Configuration Example of the static_eventflag_table Array

Caution: If TA_WSGL is specified, the eventflag behaves the same as it does with TA_WMUL.

5.4 Creating Mailbox

When the system is started up by the execution of hwos_setup() in the startup routine, mailboxes are created based on the information written in the static_mailbox_table array area reserved by the kernel.

This array is defined in the MBX_TBL structure. Mailbox IDs and the members of the T_CMBX structure are listed in the table.

The kernel recognizes MAILBOX_TBL_END (-1) set in mbxid as the end of the table.

```
//-------------------------------------
// Mailbox information
//-------------------------------------
const MBX_TBL static_mailbox_table[] = {
// CRE_MBX( mbxid,            {mbxatr,                 maxpri,  mprihd});
            {ID_APL_MBX1,      {TA_TFIFO | TA_MFIFO,    0,       1}},
            {MAILBOX_TBL_END,  {0,                      0,       0}}
};
```

Figure5.5 Configuration Example of the static_mailbox_table Array

Caution: The mailbox attribute TA_MPRI cannot be used by default. To use this attribute, see the function "hwos_set_mpri_operation" in section 8, Utility Function.

5.5 Creating Mutex

When the system is started by the execution of hwos_setup() in the startup routine, mutexes are created based on the information written in the static_mutex_table array area reserved by the kernel.

This array is defined in the MTX_TBL structure. Mutex IDs and the members of the T_CMTX structure are listed in the table.

The kernel recognizes MUTEX_TBL_END (-1) set in mtxid as the end of the table.

```
//-------------------------------------
// Mutex information
//-------------------------------------
const MTX_TBL static_mutex_table[] = {
// CRE_MTX( mtxid,            {mtxatr,     ceilpri});
            {ID_APL_MTX1,      {TA_TFIFO,   0}},
            {MUTEX_TBL_END,    {0,          0}}
};
```

Figure5.6 Configuration Example of the static_mutex_table Array

Caution: The mutex attributes TA_INHERIT and TA_CEILING are not supported. If specified, they are ignored.

5.6 Defining Interrupt Handler

When the system is started by the execution of hwos_setup() in the startup routine, the interrupt handlers controlled by this operating system (kernel interrupt) are created based on the information written in the static_interrupt_table array area reserved by the kernel.

This array is defined in the INT_TBL structure. Interrupt handler numbers and the members of the T_DINH structure are listed in the table.

The kernel recognizes INT_TBL_END (0xFFFFFFFF) set in inhno as the end of the table.

```
//-------------------------------------
// Interrupt handler information
//-------------------------------------
const INT_TBL static_interrupt_table[] = {
// DEF_INH( inhno,             {inhatr,      inthdr});
            {INTPZ0_IRQn,      {TA_HLNG,     (FP)int_task}},
            {INT_TBL_END,      {0,           (FP)NULL}}
};
```

Figure5.7 Configuration Example of the static_interrupt_table Array

Caution: All kernel interrupts have the lowest priority, 15. This means that an interrupt will not take precedence over the current interrupt and lead to multiple interrupts.

The example code of interrupt handler is shown below.

```
void int_task(void)
{
    iset_flg(ID_APL_FLG1, 0x0001);
};
```

Figure5.8 Example code of interrupt handler

6. Hardware ISRs

When the system is started by the execution of hwos_setup() in the startup routine, the hardware ISRs are registered based on the information written in the static_hwisr_table array area reserved by the kernel.

With hardware ISRs, when an interrupt is generated, the HW-RTOS automatically runs the service call previously registered in response to the interrupt. This removes the overhead time of the CPU.

The service calls invokable by hardware ISRs are set_flg(), sig_sem(), rel_wai(), and wup_tsk(), as listed in Table 1.2.

Hardware ISR is detected at rising edge of interrupt signal.

The static_hwisr_table array is defined in the HWISR_TBL structure. Thirty-two hardware ISRs can be set in this array.

As shown in the first example in Figure6.1, the HW-RTOS automatically issues the service call "set_flg(ID_APL_FLG1, 0x0001);" when the INTPZ1 interrupt is generated.

In the second example in Figure6.1 the HW-RTOS automatically issues the service call "wup_tsk(ID_TASK_MAIN);", when the INTPZ2 interrupt is generated.

The kernel recognizes HWISR_TBL_END (0xFFFFFFFF) set in inhno as the end of the table.

```
//-------------------------------------
// Hardware ISR
//-------------------------------------
const HWISR_TBL static_hwisr_table[] = {
//  {inhno,           hwisr_syscall,    id,              setptn}
    {INTPZ1_IRQn,     HWISR_SET_FLG,    ID_APL_FLG1,     0x0001},
    {INTPZ2_IRQn,     HWISR_WUP_TSK,    ID_TASK_MAIN,    0},
    {HWISR_TBL_END,   0,                0,               0}
};
```

Figure6.1 Configuration Example of the static_hwisr_table Array

7. Interrupt Management Function

7.1 Types of Interrupts

Two types of interrupts specified in μITRON4.0 are available in this system, the kernel interrupts and non-kernel interrupts. In addition, there is hardware ISR which function is dedicated to HW-RTOS.

- Kernel interrupt

  The interrupts for which handlers are registered in the operating system are called "kernel interrupts".

  If the handler is registered in static_interrupt_table, the handler works as kernel interrupt.

  The kernel interrupt handler can issue service calls. If a kernel interrupt is generated while the system is processing a service call, the call is postponed until the system becomes ready to accept the interrupt.

  Kernel sets the priority of kernel interrupt to 15.

- Non-kernel interrupt

  The interrupts which is not masked during kernel operation are called “non-kernel interrupts”. The non-kernel interrupt handlers cannot issue service calls. If a non-kernel interrupt is generated while the system is processing a service call, the interrupt request is accepted immediately. Being independent of the kernel processing, a high-speed response is realized.

  User should set the priority of non-kernel interrupt to the higher priority than 14, that is 0 to 13.

- Hardware ISR

  The hardware ISR works independently in HW-RTOS itself.

  This ISR can invoke only limited service call, but it works without CPU handling. (refer chapter 6 for a detail)

7.2 Handling of CPU Exception

The SVCall exception as exception number 11, out of the Cortex-M exceptions as exception number 1 to 15, is handled by kernel. Other exceptions are handled as non-kernel interrupts.

7.3 Multiple Interrupts

All the kernel interrupts are configured to have the same priority. This means that an interrupt will not take precedence over the current interrupt and lead to multiple interrupts.

7.4 Interrupt Handler

An interrupt handler is a routine for exclusive processing whenever the given interrupt is generated.

The kernel starts up an interrupt handler after the necessary processing.

Interrupt handlers are registered in the initial settings. See Section 5.6 Defining Interrupt Handler for details.

8. Utility Functions

rin_hwos_get_version

Synopsis

Gets version information.

C Language format

char *rin_hwos_get_version(uint8_t mode);

Parameter

I/O	Parameter		Description
I	uint8_t	mode	Format of the version information to be output 0 Version information only 1 Version information with build date and time

Function

This call gets the version information of the operating system in the format specified by the parameter.

Return value

Version information in string format.

Example

```
printf("R-IN HWRTOS lib ver%s¥n",rin_hwos_get_version(0));
```

hwos_set_mpri_operation

Synopsis

Enables the TA_MPRI attribute.

C Language format

void hwos_set_mpri_operation(int32_t flag);

Parameter

I/O	Parameter		Description
I	int32_t	flag	Enables the TA_MPRI attribute HWOS_DISABLE_MPRI (0) Disabled (default) HWOS_ENABLE_MPRI (1) Enabled

Function

This call enables the use of a mailbox with the TA_MPRI attribute. It is disabled by default as long as this function is not called.

Call this API function before sending or receiving messages. Otherwise, the operation is undefined.

The error code E_QOVR is returned if this function is called 256 times or more while more than one message remains in the queue for the mailbox. Return of the same error code is continued until the mailbox becomes empty after all messages have been received by user operations.

	Parameter	
Item	HWOS_DISABLE_MPRI	HWOS_ENABLE_MPRI
Creating a mailbox with the TA_MPRI attribute	Successful	Successful
Sending messages to the mailbox with the TA_MPRI attribute	Failed Value returned by the function: E_RSATR (-11)	If messages are continuously sent while more than one message remains in the mailbox, this function fails on the 256th attempt. Value returned value by the function: E_QOVR (-43)
Receiving messages from the mailbox with the TA_MPRI attribute	Failed Value returned by the function: E_RSATR (-11)	Successful
Remark	-	If the error code E_QOVR is returned by the function, message transmission has failed and the error code continuous to be returned until the mailbox has become empty.

Return value

None

hwos_set_tick_time

Synopsis

Set the Tick Time.

C Language format

int32_t hwos_set_tick_time(uint32_t tick_time);

Parameter

I/O	Parameter		Description
I	uint32_t	tick_time	Set the Tick Time of the HW-RTOS. (1[us] precision) The setting range is 10-100000. (10[us]-100[ms])

Function

This API sets the Tick Time of the HW-RTOS.

This API must be called before setting up of the HW-RTOS (hwos_setup function).

Tick Time is the default value (1[ms]) if this API is not called or is called with invalid parameter.

Return value

Macro	Value	Meaning
TRUE	1	Tick Time setting has been successful.
FALSE	0	Tick Time setting has been failed. (Parameter is invalid)

Restriction

When this API is called after setting up of the HW-RTOS (hwos_setup function), The Tick Time is not changed even if TRUE is returned as a result of this API call.

9. Development Tool Dependent Configuration

This section explains the differences between development tools.

The IAR compilers use the libraries provided with the respective compilers during startup.

9.1 IAR

9.1.1 Startup

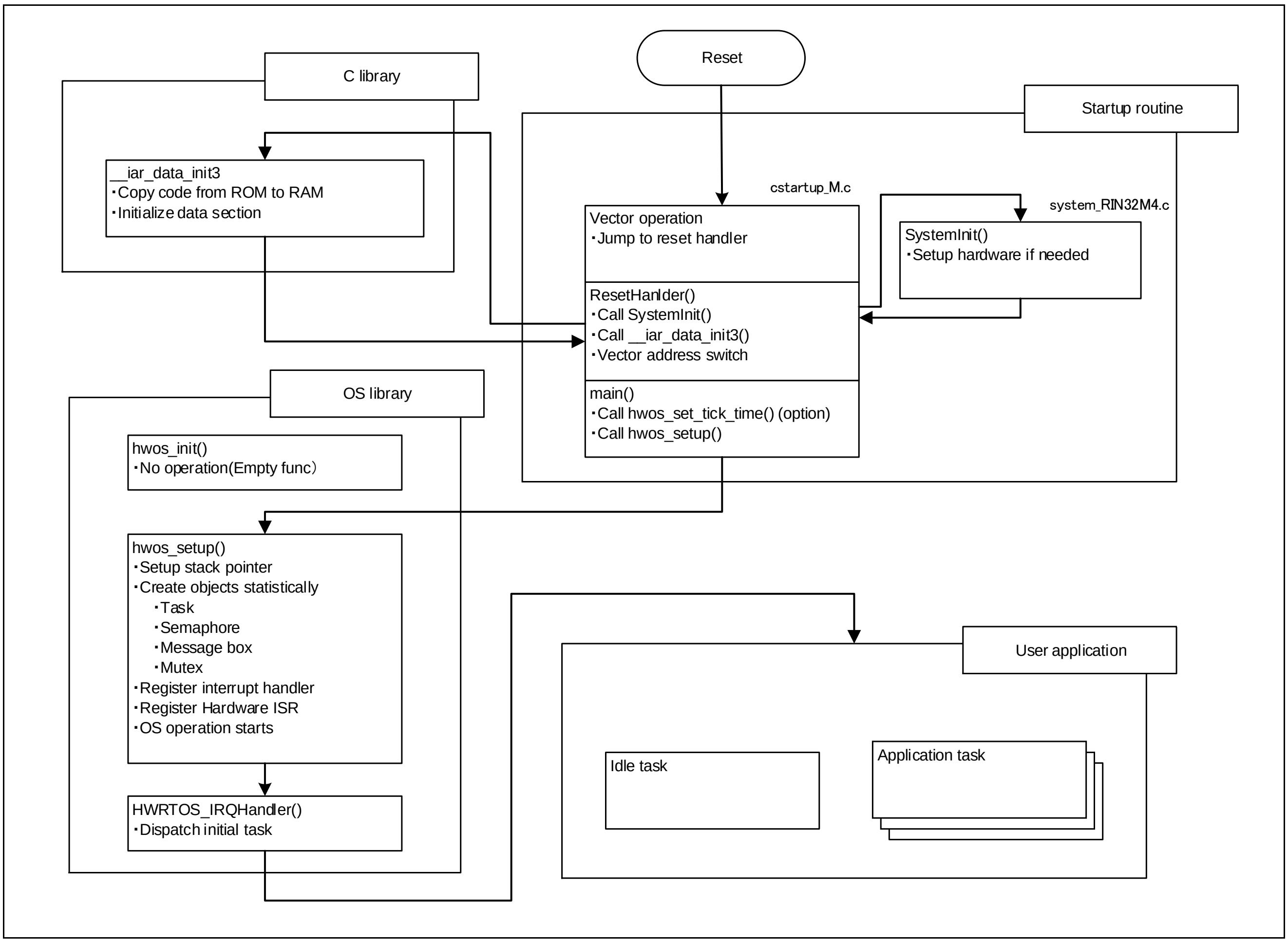

Figure9.1 Startup Routine with the IAR Compiler

9.1.2 Stack area

The initial state of the stack area at startup of the operating system (after execution of hwos_setup) is illustrated below. In the figure, the arrows indicate the pointer directions.

This operating system uses two stack pointers, the main stack pointer (MSP) and the process stack pointer (PSP). With the IAR compiler, these two pointers are used with the same area (CSTACK).

The PSP is used in normal task processing while the MSP is used in other processing, such as the handling of interrupts.

The initial value of PSP points to the stack for the first task to be activated. The pointer is switched to the end of the stack area for the destination task when it is dispatched.

The operating system gets the addresses where each section starts and ends by referring to the symbols shown in the figure below. Be sure to define the section names in the linker setting file (*.icf) with the same names as these symbols.

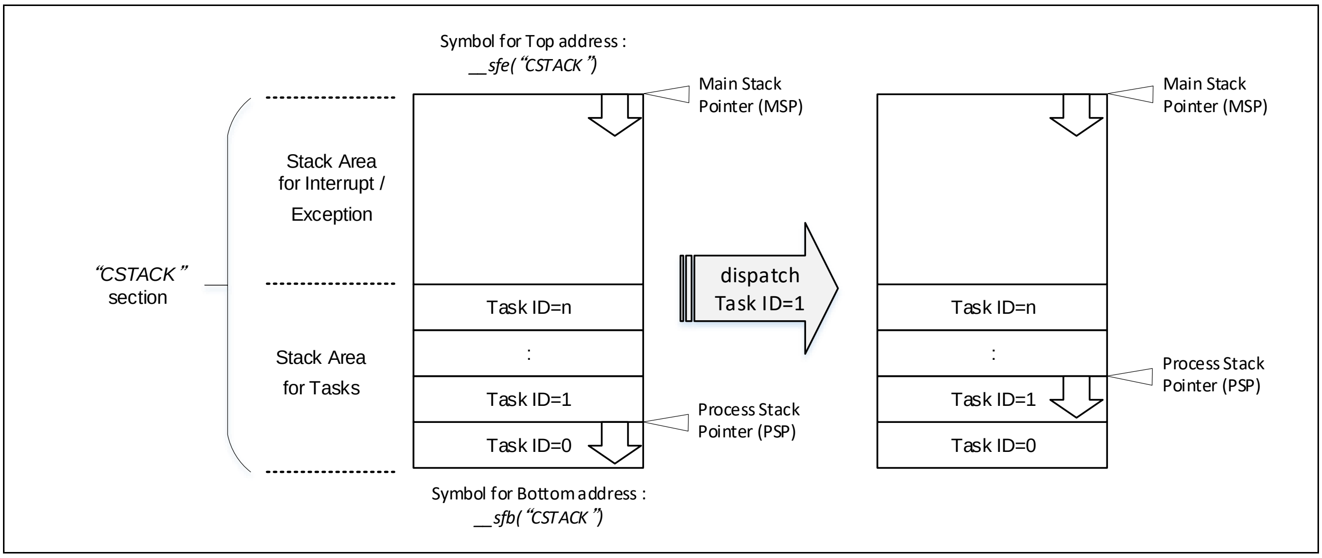

Figure9.2 Stack Area at Startup of the Operating System of the IAR Compiler

9.1.3 Compilation Options

Compilation options to use in creating the library for this operating system are listed below.

--cpu=Cortex-M4F	Target CPU: Cortex-M4
--fpu=VFPv4_sp	FPU type: Single floating point
--endian=little	Endian of generation code: Little
-e	Enables language extension.
-Ohs	Optimization level
--no_size_constraints	Removes measures to limit code size in optimization.

10. Resources

10.1 Hardware Resources

The hardware resources used in the library for this operating system are listed below.

Table10.1 Hardware Resources

Resource Name	Content
HW-RTOS	Hardware real-time OS
Exceptions (interrupts)	SVCall (exception number 11): A call of a system service by the SVC instruction INTHWRTOS (exception number 92): An HW-RTOS interrupt

[Note] The timer of the HW-RTOS provides tick, so timers of peripheral functions are not used.

10.2 Memory

The memory resources used in the library for this operating system are listed below. In addition, memory for stack area is also required. See Section 10.3

Stack.

Table10.2 Memory Usage

Category	Size[bytes]		
	ARM	GNU	IAR
code	--	--	7,240
RO Data	--	--	4
RW Data	--	--	0
ZI Data	--	--	3,744

10.3 Stack

Two types of stacks, the process stack and the main stack are available in this system.

The amount used for each stack is calculated by using the methods described in the subsequent sections. Definition of a stack area differs according to the compiler. Section 9. Development Tool Dependent Configuration.

10.3.1 Calculating the Size of the Process Stacks

Process stacks are used for tasks. The stack for each task consumes the amount of memory obtained by adding a) and b) below. To obtain the process stack requirements of a system, add up the amounts of stack for all tasks to be created, which is the sum of the values of stksz defined in the array for task creation, static_task_table.

a) Maximum amount of stack consumed for the function call tree which starts with the function that initiates the task.
b) The size taken up by storing the values of the task context registers, 72 bytes.

10.3.2 Calculating the Size of the Main Stack

Main stack is used for non-task processing.

Each non-task consumes the amount of stack obtained by adding a) and b) below, which equals the maximum amount that may be used in handling an interrupt, because multiple interrupts are never generated. To obtain the main stack requirements of a system, add up the amounts of stack for all non-tasks.

a) Maximum amount of stack consumed by the function call tree which starts with each interrupt handler initiating function.
b) The amount required for saving the contents of registers before handling an interrupt, 4 bytes.

REVISION HISTORY	R-IN32M4-CL3 Programming Manual (OS edition)

Rev.	Date	Description	
		Page	Summary
1.00	Oct 30, 2019	-	First edition issued

[Memo]

R-IN32M4-CL3 Programming Manual (OS edition)

Publication Date: Rev.1.00 Oct 30, 2019

Published by: Renesas Electronics Corporation

R-IN32M4-CL3
Programming Manual (OS edition)

Renesas Electronics Corporation

R18UZ0040EJ0102