User's Manual

R-IN32M4-CL2

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 : R18UZ0040EJ0100
Issue date : Oct 05, 2015

Renesas Electronics
www.renesas.com

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.

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

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.

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such. Please be understanding of this beforehand. In addition, because we make document at development, planning of each core, the related document may be the document for individual customers. Last four digits of document number (described as ****) indicate version information of each document. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

The document related to R-IN32M4-CL2

Document name	Document number
R-IN32M4-CL2 User's Manual	R18UZ0033EJ****
R-IN32M4-CL2 User's Manual Peripheral Functions	R18UZ0035EJ****
R-IN32M4-CL2 Programming Manual (Driver edition)	R18UZ0038EJ****
R-IN32M4-CL2 Programming Manual (OS edition)	This manual

The document related to OS

Document name	Document number
µITRON 4.0 Specification Ver.4.00.00 (ITRON Committee, TRON ASSOCIATION)	-

"µITRON 4.0 Specification" is the open real-time kernel specification developed led by TRON ASSOCIATION.

The µITRON 4.0 specification of this document is the extract from "µITRON 4.0 Specification Ver.4.00.00". Please refer to "µITRON 4.0 Specification Ver.4.00.00" about the whole aspect of specifications.

In addition, you can obtain "µITRON 4.0 Specification" from website of TRON ASSOCIATION

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

Contents

Contents of figures

Contents of tables

1. Outline

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-CL2”.

1.1 Features of Hardware Real-time OS

R-IN32M4-CL2 is loaded with a Hardware Real-time OS (HW-RTOS) to speed up the handling of Real-time OS.

The Hardware Real-time OS is able to handle the good responsive OS processing to handle the task scheduler and object such as the task, the eventflag and so on by hardware.

The number of settable object is shown on Table1.1. In Hardware Real-time OS, Settable number of semaphore and mutex is 128, because the semaphore and the mutex are realized with the same hardware. For example, when 100 are used for semaphore, the usable number for mutex is 28.

Table1.1 Settable object number

object	Settable number
Task number	64
Eventflag number	64
Mailbox number	64
Semaphore number	Total 128
Mutex number	

In addition, the characteristic function of the hardware Real-time OS includes Hardware ISR function.

By using the Hardware ISR functions, the registered service call can be automatically run when the interrupts are occurred.

The List of the settable service call by Hardware ISR is shown on Table1.2. For example, if using interrupt processing to run the set_flg only, the interrupt processing with CPU becomes needless by using this function. In addition, responsive good interrupt processing is enabled because the Hardware Real-time OS runs the task scheduling with the service call running at the same time.

Please refer to “6. Hardware ISR” for detail of setting.

Table1.2 List of the settable service call by Hardware ISR

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

1.2 Supported Service call

The list of service call supported by R-IN32M4 is shown as follow.

Table1.3 Supported Service call (1/3)

Items		Name	Descriptions	HW-RTOS Driver	µITRON Ver. 4.0 Standard Profile
Task management functions		act_tsk	Active Task	-	○
		iact_tsk	Active Task	-	○
		can_act	Cancel Task Activation Requests	-	○
		sta_tsk	Active Task (with a start code)	○	-
		ext_tsk	Terminate Invoking Task	○	○
		ter_tsk	Terminate task	○	○
		chg_pri	Change Task Priority	○	○
		get_pri	Reference Task Priority	○	○
Task dependent synchronization functions		slp_tsk	Put Task to Sleep	○	○
		tslp_tsk	Put Task to Sleep（with Timeout）	○	○
		wup_tsk	Wakeup Task	○	○
		iwup_tsk	Wakeup Task	○	○
		can_wup	Cancel Task Wakeup Requests	○	○
		rel_wai	Release Task from Waiting	○	○
		irel_wai	Release Task from Waiting	○	○
		sus_tsk	Suspend Task	-	○
		frsm_tsk	Forcibly Resume Suspend Task	-	○
		rsm_tsk	Resume Suspend Task	-	○
		dly_tsk	Delay Task	-	○
Task exception handling functions		ras_tex	Raise Task Exception Handling	-	○
		iras_tex	Raise Task Exception Handling	-	○
		dis_tex	Disable Task Exceptions	-	○
		ena_tex	Enable Task Exceptions	-	○
		sns_tex	Reference Task Exception Handling State	-	○
Synchronization and Communication Functions	Semaphores	cre_sem	Create Semaphore	-	-
		del_sem	Delete Semaphore	○	-
		wai_sem	Acquire Semaphore Resource	○	○
		pol_sem	Acquire Semaphore Resource (Polling)	○	○
		twai_sem	Acquire Semaphore Resource (with Timeout)	○	○
		sig_sem	Release Semaphore Resource	○	○
		isig_sem	Release Semaphore Resource	○	○

Note: “○”: Available, “-“: Not available

表1.3 Supported Service call (2/3)

Items		Name	Descriptions	HW-RTOS Driver	µITRON Ver. 4.0 Standard Profile
Synchronization and Communication Functions	Eventflags	cre_flg	Create Eventflag	-	-
		del_flg	Delete Eventflag	○	-
		set_flg	Set Eventflag	○	○
		iset_flg	Set Eventflag	○	○
		clr_flg	Clear Eventflag	○	○
		wai_flg	Wait for Eventflag	○	○
		pol_flg	Wait for Eventflag (Polling)	○	○
		twai_flg	Wait for Eventflag (with Timeout)	○	○
	Data Queues	snd_dtq	Send to Data Queue	-	○
		psnd_dtq	Send to Data Queue (Polling)	-	○
		ipsnd_dtq	Send to Data Queue	-	○
		tsnd_dtq	Send to Data Queue (with Timeout)	-	○
		fsnd_dtq	Forced Send to Data Queue	-	○
		ifsnd_dtq	Forced Send to Data Queue	-	○
		rcv_dtq	Receive from Data Queue	-	○
		prcv_dtq	Receive from Data Queue (Polling)	-	○
	Mailboxes	cre_mbx	Create Mailbox	-	-
		del_mbx	Delete Mailbox	○	-
		snd_mbx	Send to Mailbox	○	○
		rcv_mbx	Receive from Mailbox	○	○
		prcv_mbx	Receive from Mailbox (Polling)	○	○
		trcv_mbx	Receive from Mailbox (with Timeout)	○	○
Extended Synchronization and Communication Functions	Mutexes	cre_mtx	Create Mutex	-	-
		del_mtx	Delete Mutex	○	-
		loc_mtx	Lock Mutex	○	-
		ploc_mtx	Lock Mutex(Polling)	○	-
		tloc_mtx	Lock Mutex(with Timeout)	○	-
		unl_mtx	Unlock Mutex	○	-

Note: “○”: Available, “-“: Not available

表1.3 Supported Service call (3/3)

Items		Name	Descriptions	HW-RTOS Driver	µITRON Ver. 4.0 Standard Profile
Memory Pool Management Functions	Fixed-Sized	get_mpf	Acquire Fixed-Sized Memory Block	-	○
		pget_mpf	Acquire Fixed-Sized Memory Block (Polling)	-	○
		tget_mpf	Acquire Fixed-Sized Memory Block (with Timeout)	-	○
		rel_mpf	Release Fixed-Sized Memory Block	-	○
Time Management Functions	System Time Management	set_tim	Set System Time	○	○
		get_tim	Reference System Time	○	○
		isig_tim	Supply Time Tick	-	○
	Cyclic Handlers	sta_cyc	Start Cyclic Handler Operation	-	○
		stp_cyc	Stop Cyclic Handler Operation	-	○
System State Management Functions		rot_rdq	Rotate Task Precedence	○	○
		irot_rdq	Rotate Task Precedence	○	○
		get_tid	Reference Task ID in the RUNNING State	○	○
		iget_tid	Reference Task ID in the RUNNING State	○	○
		loc_cpu	Lock the CPU	○	○
		iloc_cpu	Lock the CPU	-	○
		unl_cpu	Unlock the CPU	○	○
		iunl_cpu	Unlock the CPU	-	○
		dis_dsp	Disable Dispatching	○	○
		ena_dsp	Enable Dispatching	○	○
		sns_ctx	Reference Contexts	-	○
		sns_loc	Reference CPU State	○	○
		sns_dsp	Reference Dispatching Sate	-	○
		sns_dpn	Reference Dispatch Pending State	-	○

Note: “○”: Available, “-“: Not available

1.3 Supported Static API

The list of static API supported by R-IN32M4 is shown as follow. Please refer to 5. Object static generation to use them.

Table1.4 Supported Static API

Items		Name	Descriptions	HW-RTOS Driver	µITRON Ver. 4.0 Standard Profile
Task Management Functions		CRE_TSK	Create Task	○	○
Task Exception Handling Functions		DEF_TEX	Define Task Exception Handling Routine	－	○
Synchronization and Communication Functions	Semaphores	CRE_SEM	Create Semaphore	○	○
	Eventflags	CRE_FLG	Create Eventflags	○	○
	Data Queues	CRE_DTQ	Create Data Queues	－	○
	Mailboxes	CRE_MBX	Create Mailbox	○	○
Extended Synchronization and Communication Functions	Mutexes	CRE_MTX	Create Mutex	○	－
Memory Pool Management Functions	Fixed-Sized	CRE_MPF	Create Fixed-Sized Memory Pool	－	○
Time Management Functions	Cyclic Handlers	CRE_CYC	Create Cyclic Handler	－	○
Interrupt Management Functions		DEF_INH	Define Interrupt Handler	○	○
System Configuration Management Functions		DEF_EXC	Define CPU Exception Handler	－	○
		ATT_INI	Attach Initialization Routine	－	○

Note: “○”: Available, “-“: Not available

1.4 Difference from standard profile

	R-IN32M4 µITRON specification	µITRON Ver. 4.0 Standard profile
Activation request queuing	Not supported	supported
Task priority	1 to 15	1 to 16
Max number of semaphore resource	31	at least 65535
Message priority	1 to 7	1 to 16 (greater than task priority)
Eventflag attribute	TA_WSGL is not supported but only TA_WMUL is supported.	TA_WSGL

As standard profile is extended, below services can be called from non-task context.

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

1.5 Development environment

The explanation of software development tools is shown as follow.

Table1.5 List of software development tools (Tool chain)

ツール チェーン	コンパイラ
ARM	ARM Compiler 5.05 update 1 (ARM)
GNU	GNU Tools for ARM Embedded Processors 4.9
IAR	Embedded Workbench for ARM V7.40.1 (IAR Systems)

Table1.6 List of software development tools (development environment)

Type of tool	Product name	Version	Tool vendors
Development environment	cygwin[Note]	1.7.33-2	Red Hat
Make tool	GNU make (for cygwin[Note])	4.0	GPL

Note: Please access http://cygwin.com/ for cygwin install.
Please install “GNU make” too, when you install cygwin.

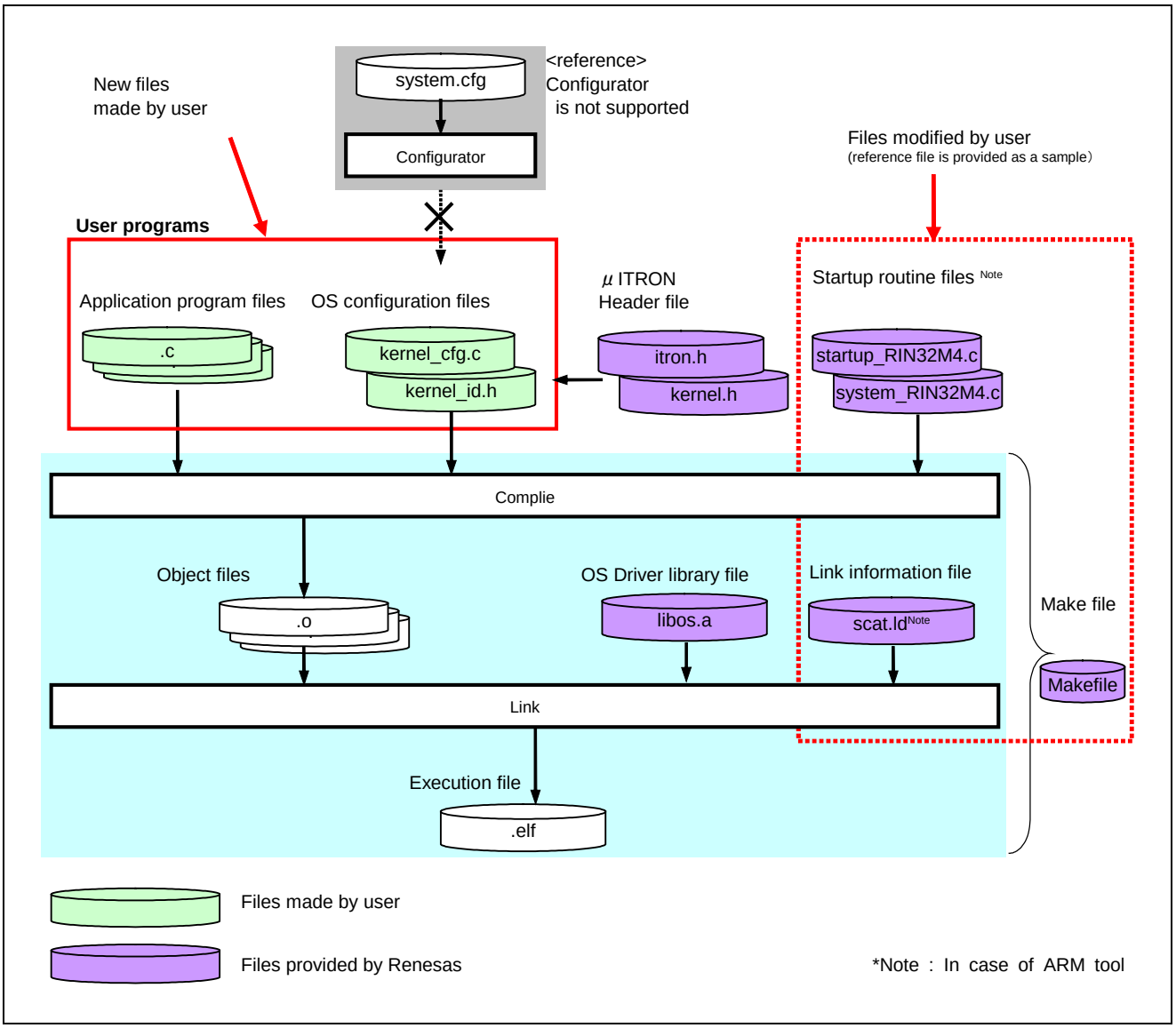
2. Software development procedure

This chapter explains the software development procedure.

2.1 Design flow

The file correlation diagram is shown on Figure 2.1File correlation diagram.
Please refer to R-IN32M4-CL2 Programming manual (Driver edition) for detail about the file structure.

Figure 2.1 File correlation diagram



2.2 Generation of the OS configuration file

Object, interrupt handler and Hardware ISR definition to be generated statically define in kernel_cfg.c.
Please refer to 5. Object static generation、6. Hardware ISR for the detail about definition method.

2.3 OS starts

When OS starts, at first, hwos_setup() function is called during startup. OS initialization finishes by this procedure. hwos_init() is empty function, but it is used for compatible to previous version.

2.3.1 OS setup

hwos_setup

(1) Description

Hardware RTOS setup

(2) C-language format

```
ER hwos_setup(void);
```

(3) Parameter

None

(4) Function

According to OS configuration file (kernel_cfg.c), OS resources are configured in this function.
This function sets parameters as follows:

- Configure stack pointer
  - Allocate stack areas for task in turn from the bottom address of stack area.
  - Set main stack pointer (MSP) to top address of interrupt stack.
  - Set process stack pointer (PSP) for first activated task.
- Configure semaphore
- Configure eventflag
- Configure mailbox
- Configure mutex
- Configure interrupt handler managed by OS
- Configure Hardware ISR

(5) Return value

Return Value	Meaning
ER_OK	Operation success

2.3.2 Initialize OS

hwos_init

(1) Description

Empty function for compatible to the R-IN32M3 Series OS library.

(2) C-language format

```
ER hwos_init(void);
```

(3) Parameter

None

(4) Function

Nothing to do

(5) Return value

Return Value	Meaning
ER_OK	none

2.4 Usage Precautions

The following explains notes for the OS library.

- Don't access to the HW-RTOS's peripheral register area (0x40080000 - 0x4008FFFF).
  On the debugger, to avoid accessing to the peripheral register area, set addresses with caution to the memory display feature or watch feature.
- Hardware ISR on the HW-RTOS does not stop even when the debugger is in break status.

3. Data Types and Macros

This chapter explains the Data type, data structure and macros when you execute service call supported by HW-RTOS library.

3.1 Data Types

The data types of each parameter specified at service call are shown as follow;

Table3.1 Data Types

macro	Data Types	Description
B	signed char	Signed 8-bit integer
H	signed short	Signed 16-bit integer
W	signed long	Signed 32-bit integer
UB	unsigned char	Unsigned 8-bit integer
UH	unsigned short	Unsigned 16-bit integer
UW	unsigned long	Unsigned 32-bit integer
VB	char	value with unknown data type (8 bit)
VH	short	value with unknown data type (16 bit)
VW	long	value with unknown data type (32 bit)
VP	void *	value with unknown data type (pointer)
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 (return parameter from service call)
ID	INT	Object ID number
ATR	UINT	Object attribute
STAT	UINT	Object state
MODE	UINT	Service call operational mode
PRI	INT	Priority of task or message
SIZE	UINT	Memory area size (Unit: Byte)
TMO	INT	Timeout (Unit: msec)
RELTIM	UINT	Relative time (Unit: msec)
SYSTIM	UINT	System time (Unit: msec)
VP_INT	VP	value with unknown data type (pointer), or Signed 32 -bit integer
ER_BOOL	ER	Error code, or a Boolean value (TRUE or FALSE)
ER_ID	ER	Error code, or an Object ID number
ER_UINT	ER	Error code, or Unsigned integer

3.2 Constants

The list of constants defined by OS Driver is shown as follows;

Table3.2 Constants (General Constants)

constants	Value	Description
NULL	0	Invalid pointer
TRUE	1	True
FALSE	0	False
E_OK	0	Normal completion

Table3.3 Constants (Object Attribute)

constants	Value	Description
TA_NULL	0	Object attribute unspecified
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 activated after creation
TA_RSTR	0x04	Limitation 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	The plural tasks are not allowed to wait for the event flag (Not supported).
TA_WMUL	0x02	The plural tasks are allowed to wait for the event flag
TA_CLR	0x04	Eventflag's bit pattern 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 task 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	Generate the period handler ins working status. (Not supported)
TA_PHS	0x04	Store the phase of the period handler (Not supported)

Table3.4 Constants (Timeout Specification)

constants	Value	Description
TMO_POL	0	Polling
TMO_FEVR	-1	Waiting forever
TMO_NBLK	-2	Non-blocking (Not supported)

Table3.5 Constants (Service call operation mode)

constants	Value	Description
TWF_ANDW	0x00	AND waiting condition for an eventflag
TWF_ORW	0x01	OR waiting condition for an eventflag

Table3.6 Constants (Other constants)

constants	Value	Description
TSK_SELF	0	Specifying invoking task
TSK_NONE	0	No applicable task (Not used)
TPRI_SELF	0	Specifying the base priority of the invoking task
TPRI_INI	0	Specifying the initial priority of the task

Table3.7 Constants (Error code)

constants	Value	Description
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

3.3 Data structure

3.3.1 µITRON V4 definition structure

T_CTSK

Description

Task generation information

Declaration of the structure

```
typedefstruct 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 of the structure

members		Description
ATR	tskatr	Task attribute When appointing TA_ACT, the task is generated in a started status. TA_ACT (2): Generate task in started status When appointing the following definitions, the movement does not change. TA_HLNG (0): Start through the high class language interface (Not Used) TA_ASM (1): Start through the assembler language interface (Not Used)
VP_INT	exinf	Task extended information (given as argument of task when TA_ACT is specified)
FP	task	Task start address
PRI	itskpri	Task initial priority value (1 to 15)[Note]: Task priority
SIZE	stksz	Task stack size (in bytes)
VP	stk	Base address of task stack space NULL (0): Setting base address by kernel (recommend) value: Set this value as base address

Note. In the µITRON 4.0 specification, the range of task priority is 1 to 16.

T_CSEM

Description

Semaphore creation information

Declaration of the structure

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

Members of the structure

members		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 count can be set :maxsem value)			
UINT	maxsem	Maximum semaphore resource count (maximum count can be set :31)			

T_CFLG

Description

Eventflag creation information

Declaration of the structure

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

Members of the structure

members		Description			
ATR	flgatr	Eventflag attribute			
		Task wait queue	TA_TFIFO	0x00	In FIFO order
			TA_TPRI	0x01	In task priority order
			TA_WMUL	0x02	Multi task wait enabled
			TA_CLR	0x04	Clear bit pattern when wait state released
FLGPTN	iflgptn	Initial value of the eventflag bit pattern			

Limitations

TA_WSGL is not supported. When TA_WSGL is specified, it is operated as TA_WMUL.

T_CMBX

Description

Mailbox creation information

Declaration of the structure

```
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 of the structure

members		Description			
ATR	mbxatr	Mailbox attribute			
		Task wait 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	Maximum message priority (priority can be set: 1 - 7)			
VP	mprihd	Start address of the area of message queue headers for each priority (Not used)			

Caution. In default, mailbox with TA_MPRI attribute is not available.
When you use the feature, refer to the “hwos_set_mpri_operation” section.

T_CMTX

Description

Mutex creation information

Declaration of the structure

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

Members of the structure

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

Caution. Mutex support neither TA_INHERIT nor TA_CEILING.

T_DINH

Description

Interrupt handler definition information

Declaration of the structure

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

Members of the structure

Members		Description
ATR	inhatr	Interrupt handler attribute (not supported)
FP	inthdr	Interrupt handler start address

3.3.2 R-IN32M4 specific definition structure

TSK_TBL

Description

Task static generation information (This defines the argument of CRE_TSK as structure)

Declaration of the structure

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

Members of the structure

members		Description
ID	id	ID number of the task to be generated statically Value(1 to 64): Target task ID
T_CTSK	t_ctsk	Task generation information

SEM_TBL

Description

Semaphore static generation information (This defines the argument of CRE_SEM as structure.)

Declaration of the structure

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

Members of the structure

members		Description
ID	id	ID number of the semaphore to be generated statically Value (1 to 128): Target semaphore ID
T_CSEM	pk_csem	semaphore generation information

FLG_TBL

Description

Eventflag static generation information (This defines the argument of CRE_FLG as structure.)

Declaration of the structure

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

Members of the structure

members		Description
ID	id	ID number of the eventflag to be generated statically Value (1 to 128): Target eventflag ID
T_CFLG	pk_cflg	Eventflag generation information

MBX_TBL

Description

Mailbox static generation information (This defines the argument of CRE_MBX as structure.)

Declaration of the structure

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

Members of the structure

members		Description
ID	id	ID number of the mailbox to be generated statically Value (1 to 64): Target mailbox ID
T_CMBX	pk_cmbx	Mailbox generation information

MTX_TBL

Description

Mutex static generation information (This defines the argument of CRE_MTX as structure.)

Declaration of the structure

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

Members of the structure

members		Description
ID	id	ID number of the mutex to be generated statically Value (1 to 128): Target mutex ID (shared with semaphore ID)
T_CMTX	pk_cmtx	Mutex generation information

INT_TBL

Description

Interrupt handler generation information (This defines the argument of DEF_INH as structure.)

Declaration of the structure

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

Members of the structure

members		Description
INHNO	inhno	Interrupt number of the Interrupt handler to be generated Value (0 to 117): Target interrupt number (specified the exception number defined by “Interrupt list” in in User’s manual minus 0x10)
T_DINH	pk_dinh	Pointer to the packet to put the interrupt handler definition information

HWISR_TBL

Description

Hardware ISR object generation information

Declaration of the structure

```
typedef struct hwisr_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 of the structure

members		Description
INHNO	inhno	Interrupt number of target hardware ISR Value (0 to 117): Target interrupt number
UINT	hwisr_syscall	Service call to operate at the time of interrupt outbreak automatically 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	Hardware ISR object ID
FLGPTN	setptn	Bit pattern to set (valid only in set_flg)

3.4 Global Variables

Global variables in OS library is as follows.

Table3.8 Global Variables

Type	Name	Meaning
HWRTOS_SBT	HWRTOS_Sbt	System management table
FP	HWRTOS_IntTable[]	Interrupt handler address entry table managed by OS

4. Service calls

4.1 Task Management Functions

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

Table4.1 Task Management Functions

Service Call	Function	Useful Range
sta_tsk	Active Task (with a start code)	Task, Non-task
ext_tsk	Terminate Invoking Task	Task
ter_tsk	Terminate Task	Task, Non-task
chg_pri	Change Task Priority	Task, Non-task
get_pri	Reference Task Priority	Task, Non-task

sta_tsk

Description

Active Task (with a Start Code)

C Language API

ER sta_tsk(ID tskid, VP_INT stacd);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task to be activated Value (1 to 64): Target task ID
I	VP_INT	stacd	Start code of the task

Function

The task specified by tskid is moved from the DORMANT state to the READY state.

The stacd is specified the expanded information to deliver to the target task.

Return parameter

Macro	Return value	Description
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

Description

Terminate Invoking Task

C Language API

void ext_tsk(void);

Parameter

None

Function

The invoking task is moved from the RUNNING state to the DORMANT state.

Return parameter

None

Limitation

If task exit with mutex locked, the mutex cannot be unlocked. Please finish task after unlock the mutex.

This function must not be called in CPU locked or in dispatch disabled or from interrupt handler. If this function is called in these states, the operation becomes unstable.

ter_tsk

Description

Terminate task

C Language API

ER ter_tsk(ID tskid);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task to be activated Value (1 to 64): Target task ID

Function

The task specified by tskid is moved to the DORMANT state forcibly.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	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 in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

Limitation

If task exit with mutex locked, the mutex cannot be unlocked. Please finish task after unlock the mutex.

chg_pri

Description

Change Task Priority

C Language API

ER chg_pri(ID tskid, PRI tskpri);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task whose priority is to be changed TSK_SELF (0): the invoking task ID value (1 to 64): the specified task ID
I	PRI	tskpri	New base priority of the task Note1 TPRI_INI (0): initial priority of the invoking task value (1 to 15) Note2: task's base priority

Note1. The base priority and current priority is always same, because mutex has not support inheritance and ceiling protocol

Note2. In the µITRON 4.0 specification, the range of task priority is 1 to 16.

Function

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

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tskpri is invalid)
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	CPU locked state
E_OBJ	-41	Object state error (specified task is in the DORMANT state)
E_NOEXS	-42	Non-existent object (specified task is not registered)

get_pri

Description

Reference Task Priority

C Language API

ER get_pri(ID tskid, PRI *p_tskpri);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task to reference TSK_SELF (0): the invoking task ID value (1 to 64): the specified task ID
O	PRI	*p_tskpri	Pointer to the memory area of a tskpri (current priority of specified task)

Function

This service call returns the current priority of the task specified by tskid through *p_tskpri.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_tskpri is NULL)
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	CPU locked state
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 following shows the service calls provided as the task dependent synchronization functions.

Table4.2 Task Dependent Synchronization Functions

Service Call	Function	Useful Range
slp_tsk	Put Task to Sleep	Task
tslp_tsk	Put Task to Sleep (with Timeout)	Task
wup_tsk	Wakeup Task	Task, Non-task
iwup_tsk	Wakeup Task	Non-task
can_wup	Cancel Task Wakeup Requests	Task, Non-task
rel_wai	Release Task from Waiting	Task, Non-task
irel_wai	Release Task from Waiting	Non-task

slp_tsk

Description

Put Task to Sleep

C Language API

ER slp_tsk (void);

Parameter

None

Function

This service call moves the invoking task to the sleeping state. However, if wakeup requests are queued, that is, if the wakeup request count for the invoking task is 1 or more, the count is decremented by 1 and the invoking task continues execution.

slp_tsk() is same as tslp_tsk(TMO_FEVR).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)

tslp_tsk

Description

Put Task to Sleep (with Timeout)

C Language API

ER tslp_tsk (TMO tmout);

Parameter

I/O	Parameter		Description
I	TMO	tmout	Specified timeout (Unit of time: ms) TMO_FEVR (-1): Waiting forever (same functionality as slp_tsk()) TMO_POL(0): Polling value: Waiting time

Function

This service call moves the invoking task to the sleeping state. However, if wakeup requests are queued, that is, if the wakeup request count for the invoking task is 1 or more, the count is decremented by 1 and the invoking task continues execution.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tmout is invalid)
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Timeout

wup_tsk / iwup_tsk

Description

Wakeup Task

C Language API

ER wup_tsk(ID tskid);

ER iwup_tsk(ID tskid);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task to be woken up TSK_SELF (0): the invoking task ID value (1 to 64): the specified task ID

Function

These service calls wake up the task specified by tskid from sleeping.

If the target task isn't WAITING state, the state doesn't change and wakeup request counter is incremented.

Remark The maximum value of wakeup request counter is 63.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	CPU locked state. Executed in task context. (only iwup_tsk)
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 exceeds 63)

can_wup

Description

Cancel Task Wakeup Requests

C Language API

ER_UINT can_wup(ID tskid);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task for cancelling wakeup requests TSK_SELF (0): the invoking task ID value (1 to 64): the specified task ID

Function

This service call cancels all queued wakeup requests for the task specified by tskid and returns the cancelled request count for the task.

Return parameter

Macro	Return value	Description
E_OK	0 or positive integer	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	CPU locked state
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

Description

Release Task from Waiting

C Language API

ER rel_wai(ID tskid);

ER irel_wai(ID tskid);

Parameter

I/O	Parameter		Description
I	ID	tskid	ID number of the task to be activated Value (1 to 64): Target task ID

Function

These service calls forcibly release the task specified by tskid from the waiting (waits for wakeup or semaphore, eventflag, message).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is invalid or unusable)
E_CTX	-25	CPU locked state Executed in task context. (only irel_wai)
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 (Semaphores)

The following shows the service calls provided as the Synchronization and Communication Functions (Semaphores).

Table4.3 Synchronization and Communication Functions (Semaphores)

Service Call	Function	Useful Range
del_sem	Delete Semaphore	Task
wai_sem	Acquire Semaphore Resource	Task
pol_sem	Acquire Semaphore Resource (Polling)	Task, Non-task
twai_sem	Acquire Semaphore Resource (with Timeout)	Task
sig_sem	Release Semaphore Resource	Task, Non-task
isig_sem	Release Semaphore Resource	Non-task

del_sem

Description

Delete Semaphore

C Language API

ER del_sem(ID semid);

Parameter

I/O	Parameter	Description
I	ID semid	ID number of the semaphore to be deleted Value (1 to 128): Target semaphore ID

Function

This service call deletes the semaphore specified by semid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object

wai_sem

Description

Acquire Semaphore Resource

C Language API

ER wai_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	ID number of the semaphore from which resource is acquired Value (1 to 128): Target semaphore ID

Function

This service call acquire one resource from the semaphore specified by semid.

If target semaphore resource is 0, self-task state is changed to waiting state.

This service call is same as twai_sem(semid, TMO_FEVR).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (target semaphore is deleted while task waits for it)

pol_sem

Description

Acquire Semaphore Resource (Polling)

C Language API

ER pol_sem(ID semid);

Parameter

I/O	Parameter		Description
I	ID	semid	ID number of the semaphore from which resource is acquired Value (1 to 128): Target semaphore ID

Function

This service calls acquire one resource from the semaphore specified by semid.

If target semaphore resource is 0, poling is failed. (Without state change)

This service calls is same as twai_sem(semid, TMO_POL).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is invalid or unusable)
E_CTX	-25	CPU locked state
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

twai_sem

Description

Acquire Semaphore Resource (with Timeout)

C Language API

ER twai_sem(ID semid, TMO tmout);

Parameter

I/O	Parameter		Description
I	ID	semid	ID number of the semaphore from which resource is acquired Value (1 to 128): Target semaphore ID
I	TMO	tmout	Specified timeout (Unit of time: ms) TMO_FEVR (-1): Waiting forever TMO_POL (0): Polling value: Positive number indicating a timeout duration

Function

This service calls acquire one resource from the semaphore specified by semid.

If target semaphore resource is 0, self-task state is changed to wait for a semaphore.

Return parameter

Macro	Return value	Description
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	CPU locked or dispatch disabled state, executed in non-task context. (Only if TMO_POL is specified, can be executed in non-task context)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Polling failure or timeout
E_DLT	-51	Waiting object deleted (target semaphore is deleted while task waits for it)

sig_sem / isig_sem

Description

Release Semaphore Resource

C Language API

ER sig_sem(ID semid);

ER isig_sem(ID semid);

Parameter

I/O	Parameter	Description
I	ID semid	ID number of the semaphore from which resource is acquired Value (1 to 128): Target semaphore ID

Function

These service calls release one resource to the semaphore specified by semid.

If there is the task which waits for target semaphore, release the head task of queue.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number（semid is invalid or unusable）
E_CTX	-25	CPU locked state. Executed in task context. (only isig_sem)
E_NOEXS	-42	Non-existent object（specified task is not registered）
E_QOVR	-43	Queue overflow (release over maximum resource count number (31))

4.4 Synchronization and Communication Functions (Eventflags)

The following shows the service calls provided as the Synchronization and Communication Functions (Eventflags).

Table4.4 Synchronization and Communication Functions (Eventflags)

Service Call	Function	Useful Range
del_flg	Delete Eventflag	Task
set_flg	Set Eventflag	Task, Non-task
iset_flg	Set Eventflag	Non-task
clr_flg	Clear Eventflag	Task, Non-task
wai_flg	Wait for Eventflag	Task
pol_flg	Wait for Eventflag (Polling)	Task, Non-task
twai_flg	Wait for Eventflag (with Timeout)	Task

del_flg

Description

Delete Eventflag

C Language API

ER del_flg(ID flgid);

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to be deleted Value (1 to 64): Target eventflag ID

Function

This service call deletes the eventflag specified by flgid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked state, executed in non-task context
E_NOEXS	-42	Non-existent object

set_flg / iset_flg

Description

Set Eventflag

C Language API

ER set_flg(ID flgid, FLGPTN setptn);

ER iset_flg(ID flgid, FLGPTN setptn);

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to be set Value (1 to 64): Target eventflag ID
I	FLGPTN	setptn	Bit pattern to set (16bit)

Function

These service calls set the bits specified by setptn in the eventflag specified by flgid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error（setptn is invalid)
E_ID	-18	Invalid ID number（flgid is invalid or unusable）
E_CTX	-25	CPU locked state Executed in task context (only iset_flg)
E_NOEXS	-42	Non-existent object（specified eventflag is not registered）

clr_flg

Description

Clear Eventflag

C Language API

ER clr_flg(ID flgid, FLGPTN clrptn);

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the eventflag to be cleared Value (1 to 64): Target eventflag ID
I	FLGPTN	clrptn	Bit pattern to clear (16bit)

Function

This service call clears the bits in the eventflag specified by flgid that correspond to 0 bit in clrptn.

Whatever clrptn has the value over 16 bit range, this service call doesn't return the error code. (This is different from set_flg)

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number（flgid is invalid or unusable）
E_CTX	-25	CPU locked state
E_NOEXS	-42	Non-existent object（specified eventflag is not registered）

wai_flg

Description

Wait for Eventflag

C Language API

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

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the semaphore from which resource is acquired Value (1 to 64): Target eventflag ID
I	FLGPTN	waiptn	Wait bit pattern (16bit) (If the value is over 16bit range or the value is 0x0000, error occurs)
I	MODE	wfmode	Wait mode TWF_ANDW (0): AND wait of eventflag TWF_ORW (1): OR wait of eventflag
O	FLGPTN	*p_flgptn	The pointer for bit pattern causing a task to be released from waiting

Function

This service call causes invoking task to wait until the eventflag specified by flgid satisfies the release condition. The release condition is determined by the bit pattern specified by waiptn and the wait mode specified by wfmode.

wai_flg(~) is same as twai_flg(~, TMO_FEVR).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn, wfmode or p_flgptn is invalid)
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (target eventflag is deleted while task waits for it)

pol_flg

Description

Wait for Eventflag (Polling)

C Language API

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

Parameter

I/O	Parameter		Description
I	ID	flgid	ID number of the semaphore from which resource is acquired Value (1 to 64): Target eventflag ID
I	FLGPTN	waiptn	Wait bit pattern (16bit) (If the value is over 16bit range or the value is 0x0000, error occurs)
I	MODE	wfmode	Wait mode TWF_ANDW (0): AND wait of eventflag TWF_ORW (1): OR wait of eventflag
O	FLGPTN	*p_flgptn	The pointer for bit pattern causing a task to be released from waiting

Function

This service call causes invoking task to wait until the eventflag specified by flgid satisfies the release condition. The release condition is determined by the bit pattern specified by waiptn and the wait mode specified by wfmode.

pol_flg(~) is same as twai_flg(~, TMO_POL).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn, wfmode or p_flgptn is invalid)
E_ID	-18	Invalid ID number (flgid is invalid)
E_CTX	-25	CPU locked state
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

twai_flg

Description

Wait for Eventflag (with Timeout)

C Language API

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 semaphore from which resource is acquired Value (1 to 64): Target eventflag ID
I	FLGPTN	waiptn	Wait bit pattern (16bit) (If the value is over 16bit range or the value is 0x0000, error occurs)
I	MODE	wfmode	Wait mode TWF_ANDW (0): AND wait of eventflag TWF_ORW (1): OR wait of eventflag
O	FLGPTN	*p_flgptn	The pointer for bit pattern causing a task to be released from waiting
I	TMO	tmout	Specified timeout（Unit of time: ms） TMO_FEVR (-1): Waiting forever TMO_POL (0): Polling value: Positive number indicating a timeout duration

Function

This service call causes invoking task to wait until the eventflag specified by flgid satisfies the release condition. The release condition is determined by the bit pattern specified by waiptn and the wait mode specified by wfmode.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (waiptn, wfmode or p_flgptn is invalid)
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Polling failure or timeout
E_DLT	-51	Waiting object deleted (target eventflag is deleted while task waits for it)

4.5 Synchronization and Communication Functions (Mailboxes)

The following shows the service calls provided as the Synchronization and Communication Functions (Mailboxes).

Table4.5 Synchronization and Communication Functions (Mailboxes)

Service Call	Function	Useful Range
del_mbx	Delete Mailbox	Task
snd_mbx	Send to Mailbox	Task, Non-task
isnd_mbx	Send to Mailbox	Non-task
rcv_mbx	Receive from Mailbox	Task
prcv_mbx	Receive from Mailbox (Polling)	Task, Non-task
trcv_mbx	Receive from Mailbox (with Timeout)	Task

del_mbx

Description

Delete Mailbox

C Language API

ER del_mbx(ID mbxid);

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox to be deleted Value (1 to 64): Target mailbox ID

Function

This service call deletes the mailbox specified by mbxid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object

snd_mbx / isnd_mbx

Description

Send to Mailbox

C Language API

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 which the message is sent Value (1 to 64): Target mailbox ID
I	T_ MSG *pk_msg	Start address of the message packet to be sent to the mailbox

Function

This service call sends the message whose start address is specified by pk_msg to the mailbox specified by mbxid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (pk_msg is invalid)
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked state Executed in task context (only isnd_mbx)
E_NOEXS	-42	Non-existent object
E_QOVR	-43	Queuing overflow (message queue number is over max value (192))

rcv_mbx

Description

Receive from Mailbox

C Language API

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox from which a message is received Value (1 to 64): Target mailbox ID
O	T_ MSG	**ppk_msg	The pointer for start address of the message packet received from the mailbox

Function

This service call receive a message from the mailbox specified by mbxid and return its start address through ppk_msg. If target mailbox is empty, self-task state is changed to wait for message state.

rcv_mbx(~) is same as trcv_mbx(~, TMO_FEVR).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (ppk_msg is invalid)
E_ID	-18	Invalid ID number (mbxid is invalid)
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_DLT	-51	Waiting object deleted (target mailbox is deleted while task waits for it)

prcv_mbx

Description

Receive from Mailbox (Polling)

C Language API

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

Parameter

I/O	Parameter		Description
I	ID	mbxid	ID number of the mailbox from which a message is received Value (1 to 64): Target mailbox ID
O	T_ MSG	**ppk_msg	The pointer for start address of the message packet received from the mailbox

Function

This service call receive a message from the mailbox specified by mbxid and return its start address through ppk_msg. If target mailbox is empty, polling failure is returned without entering wait state.

prcv_mbx(~) is same as trcv_mbx(~, TMO_POL).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (ppk_msg is invalid)
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked state
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

Limitation

This service call cannot be executed in interrupt handler.

trcv_mbx

Description

Receive from Mailbox (with Timeout)

C Language API

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 Value (1 to 64): Target mailbox ID
O	T_ MSG	**ppk_msg	The pointer for start address of the message packet received from the mailbox
I	TMO	tmout	Specified timeout（Unit of time: ms） TMO_FEVR（-1）: Waiting forever TMO_POL (0): Polling value: Positive number indicating a timeout duration

Function

This service call receive a message from mailbox specified by mbxid and return its start address through ppk_msg. If target mailbox is empty, self-task state is changed to wait for message state.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (ppk_msg is invalid)
E_ID	-18	Invalid ID number
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting（accept rel_wai while waiting）
E_TMOUT	-50	Polling failure or timeout
E_DLT	-51	Waiting object deleted (target mailbox is deleted while task waits for it)

Limitation

If TMO_POL is specified, this service call cannot be executed in interrupt handler.

4.6 Extended Synchronization and Communication Functions (Mutexes)

The following shows the service calls provided as the Extended Synchronization and Communication Functions (Mutexes).

Table4.6 Extended Synchronization and Communication Functions (Mutexes)

Service Call	Function	Useful Range
del_mtx	Delete Mutex	Task
loc_mtx	Lock Mutex	Task
ploc_mtx	Lock Mutex (Polling)	Task
tloc_mtx	Lock Mutex (with Timeout)	Task
unl_mtx	Unlock Mutex	Task

del_mtx

Description

Delete Mutex

C Language API

ER del_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be deleted Value (1 to 128): Target mutex ID

Function

This service call deletes the mutex specified by mtxid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is invalid or unusable)
E_CTX	-25	CPU locked state or executed in non-task context
E_NOEXS	-42	Non-existent object

loc_mtx

Description

Lock Mutex

C Language API

ER loc_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked Value (1 to 128): Target mutex ID

Function

This service call lock the mutex specified by mtxid.

If target mutex has already locked by other task, self-task state is changed to wait for mutex state.

loc_mtx(mtxid) is same as tloc_mtx(mtxid, TMO_FEVR).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number（mtxid is invalid or unusable）
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context
E_ILUSE	-28	Illegal service call use (self-task has already locked mutex)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting（accept rel_wai while waiting）
E_DLT	-51	Waiting object deleted (target mutex is deleted while task waits for it)

ploc_mtx

Description

Lock Mutex (Polling)

C Language API

ER ploc_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked Value (1 to 128): Target mutex ID

Function

This service call lock the mutex specified by mtxid.

If target mutex has already be locked by other task, polling failure is returned without entering wait state.

ploc_mtx(mtxid) is same as tloc_mtx(mtxid, TMO_POL).

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number（mtxid is invalid or unusable）
E_CTX	-25	CPU locked or dispatch disabled state
E_ILUSE	-28	Illegal service call use (self-task has already locked mutex)
E_NOEXS	-42	Non-existent object
E_TMOUT	-50	Polling failure

tloc_mtx

Description

Lock Mutex (with Timeout)

C Language API

ER tloc_mtx(ID mtxid, TMO tmout);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be locked Value (1 to 128): Target mutex ID
I	TMO	tmout	Specified timeout (Unit of time: ms) TMO_FEVR (-1): Waiting forever TMP_POL (0): Polling value: Positive number indicating a timeout duration

Function

This service call lock the mutex specified by mtxid.

If target mutex has already locked by other task, self-task state is changed to wait for mutex state.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (tmout is invalid)
E_ID	-18	Invalid ID number (mtxid is invalid or unusable)
E_CTX	-25	CPU locked or dispatch disabled state, executed in non-task context (If TMO_POL is specified, it can be executed in non-task context)
E_ILUSE	-28	Illegal service call use (self-task has already locked mutex)
E_NOEXS	-42	Non-existent object
E_RLWAI	-49	Forced release from waiting (accept rel_wai while waiting)
E_TMOUT	-50	Polling failure or timeout
E_DLT	-51	Waiting object deleted (target mutex is deleted while task waits for it)

unl_mtx

Description

Unlock Mutex

C Language API

ER unl_mtx(ID mtxid);

Parameter

I/O	Parameter		Description
I	ID	mtxid	ID number of the mutex to be unlocked Value (1 to 128): Target mutex ID

Function

This service call unlocks the mutex specified by mtxid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is invalid or unusable)
E_CTX	-25	CPU locked state, executed in non-task context
E_NOEXS	-42	Non-existent object

Limitations

In case of exiting task (ext_tsk or ter_tsk) remain locking mutex, in the original specification unlock procedure is done. But in Hardware Real time OS, the unlock procedure is not done. So please call unlock mutex before exiting task.

4.7 System Time Management

The following shows the service calls provided as the System Time Management.

Table4.7 System Time Management

Service Call	Function	Useful Range
set_tim	Set System Time	Task, Non-task
get_tim	Reference System Time	Task, Non-task

set_tim

Description

Set System Time

C Language API

ER set_tim(SYSTIM *p_systim);

Parameter

I/O	Parameter	Description
I	SYSTIM *p_systim	The pointer for time to set as system time

Function

This service call sets the system time to the value specified by p_systim.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_systim is invalid)
E_CTX	-25	CPU locked state

get_tim

Description

Reference System Time

C Language API

ER get_tim(SYSTIM *p_systim);

Parameter

I/O	Parameter	Description
O	SYSTIM *p_systim	The pointer for region current system time stored

Function

This service call returns the current system time through *p_systim.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error (p_systim is invalid)
E_CTX	-25	CPU locked state

4.8 System State Management Functions

The following shows the service calls provided as the System State Management Functions.

Table4.8 System State Management Functions

Service Call	Function	Useful Range
rot_rdq	Rotate Task Precedence	Task, Non-task
irot_rdq	Rotate Task Precedence	Non-task
get_tid	Reference Task ID in the RUNNING State	Task, Non-task
iget_tid	Reference Task ID in the RUNNING State	Non-task
loc_cpu	Lock the CPU	Task
unl_cpu	Unlock the CPU	Task
sns_loc	Reference CPU State	Task, Non-task
dis_dsp	Disable Dispatching	Task
ena_dsp	Enable Dispatching	Task

rot_rdq / irot_rdq

Description

Rotate Task Precedence

C Language API

ER rot_rdq(PRI tskpri);

ER irot_rdq(PRI tskpri);

Parameter

I/O	Parameter		Description
I	PRI	tskpri	Priority of the tasks whose precedence is rotated TPRI_SELF（0） Specifying the base priority of the invoking task Value (1 to 15) Priority of rotated task

Function

These service calls rotate the precedence of the tasks with the priority specified by tskpri.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error （tskpri is invalid or TPRI_SELF is specified from interrupt handler)
E_CTX	-25	CPU locked state Executed in task context (only irot_rdq)

get_tid / iget_tid

Description

Reference Task ID in the RUNNING State

C Language API

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 service calls reference the ID number of the task in the RUNNING state (this corresponds to the invoking task when the service call is invoked from task contexts) and return the task ID through *p_tskid.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_PAR	-17	Parameter error（p_tskid is invalid)
E_CTX	-25	CPU locked state Executed in task context (only iget_tid)

Limitation

If this service call is invoked from idle task, reference task ID is not TSK_NONE (=0) but idle task ID.

loc_cpu

Description

Lock the CPU

C Language API

ER loc_cpu(void);

Parameter

None

Function

This service call transition the system to the CPU locked state. If the system is in the CPU locked state, no action is required.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_CTX	-25	Executed in non-task context

Limitation

This service call cannot be executed in interrupt handler.

However CPU locked state, interrupt managed by RTOS can be accepted.

unl_cpu

Description

Unlock the CPU

C Language API

ER unl_cpu(void);

Parameter

None

Function

This service call transitions the system to the CPU unlocked state.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_CTX	-25	Executed in non-task context

Limitation

This service call cannot be executed in interrupt handler.

sns_loc

Description

Reference CPU State

C Language API

BOOL sns_loc(void);

Parameter

None

Function

This service call returns TRUE if the system is in the CPU locked state and returns FALSE if the system is in the CPU unlocked state

Return parameter

Macro	Return value	Description
TRUE	1	CPU locked state
FALSE	0	CPU unlocked state

dis_dsp

Description

Disable Dispatching

C Language API

ER dis_dsp(void);

Parameter

None

Function

This service call transitions the system to the dispatching disabled state.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_CTX	-25	Executed in non-task context

ena_dsp

Description

Enable Dispatching

C Language API

ER ena_dsp(void);

Parameter

None

Function

This service call transitions the system to the dispatching enabled state.

Return parameter

Macro	Return value	Description
E_OK	0	Normal completion
E_CTX	-25	Executed in non-task context

5. Object static generation

5.1 Task generation

The task can be generated in the time of OS startup routine (running hwos_setup() function) by writing task information in static_task_table array reserved by kernel.

The static_task_table array is defined with TSK_TBL structure and can be set by listing the task ID and member of T_CTSK structure.

In case of setting tskid to TASK_TBL_END (-1), the kernel considers it 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}}
};
```

Figure 5.1 Sequence setting example of static_task_table

Caution. Definition of task is necessary because this OS doesn't have idle task in kernel. The definition example of idle task is shown in Figure 5.2.

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

Figure 5.2 Definition example of the idle task

Caution. If multi tasks specified TA_ACT are created, the order of task ready queue is not correspond to the order of definition array. To control the order of task ready queue, please don't specify TA_ACT but use service call sta_tsk.
HW-RTOS supports only static object creation. After HW-RTOS operation starts, object cannot be created dynamically.
At least one of the tasks need to be specified as TA_ACT.

5.2 Semaphore generation

The semaphore can be generated in the time of OS startup routine (running hwos_init() function) by writing semaphore information in static_semaphore_table array reserved by kernel.

The static_semaphore_table array is defined with TSK_SEM structure and can be set by listing the argument of CRE_SEM.

In case of setting semid to SEMAPHORE_TBL_END (-1), the kernel considers it 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 Sequence setting example of static_semaphore_table

5.3 Eventflag generation

The eventflag can be generated in the time of OS startup routine (running hwos_init() function) by writing eventflag information in static_eventflag_table array reserved by kernel.

The static_eventflag_table array is defined with FLG_TBL structure and can be set by listing the event flag ID and member of T_CFLG structure.

In case of setting flgid to EVENTFLAG_TBL_END (-1), the kernel considers it 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}},
           {EVENTFLAG_TBL_END, {0,                  0}}
};
```

Figure 5.4 Sequence setting example of static_eventflag_table

Caution. If TA_WSGL is specified, but this OS treat as TA_WMUL.

5.4 Mailbox generation

The mailbox can be generated in the time of OS startup routine (running hwos_init() function) by writing mailbox information in static_mailbox_table array reserved by kernel.

The static_mailbox_table array is defined with MBX_TBL structure and can be set by listing mailbox ID and member of T_CMBX structure.

In case of setting mbxid to MAILBOX_TBL_END (-1), the kernel considers it the end of the table.

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

Figure 5.5 Sequence setting example of static_mailbox_table

Caution. In default, mailbox with TA_MPRI attribute is not available.
When you use the feature, refer to the “hwos_set_mpri_operation” section.

5.5 Mutex generation

The mutex can be generated in the time of OS startup routine (running hwos_init() function) by writing mutex information in static_mutex_table array reserved by kernel.

The static_mutex_table array is defined with MTX_TBL structure and can be set by listing mutex ID and member of T_CMTX structure.

In case of setting mtxid to MUTEX_TBL_END (-1), the kernel considers it 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}}
};
```

Figure 5.6 Sequence setting example of static_mutex_table

Caution. This OS supports neither mutex attribute TA_INHERIT nor TA_CEILING.
If these attribute are set, it is ignored.

5.6 Interrupt handler definition

The interrupt handler to be monitored by OS can be generated in the time of OS startup routine (running hwos_init() function) by writing interrupt handler information in static_interrupt_table array reserved by kernel.

The service call of OS can be executed in the interrupt handler to be generated by this definition.

The static_interrupt_table array is defined with INT_TBL structure and can be set by listing interrupt handler number and member of T_DINH structure.

In case of setting inhno to INT_TBL_END (0xFFFFFFFF), the kernel considers it 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}}
};
```

Figure 5.7 Sequence setting example of static_interrupt_table

Caution. All of the interrupt priority managed by OS kernel are set to 15 (bottom priority). Because of it, recursive interrupt doesn't occur.

6. Hardware ISR

The Hardware ISR can be registered in the time of OS startup routine (running hwos_init() function) by writing interrupt handler ISR information in static_interrupt_table array reserved by kernel.

The Hardware ISR is the function that the Hardware Real-time OS automatically runs the registered service call when the interrupt occurs. The overhead of operating interrupt by CPU can be reduced by using this function.

The service calls to be run by Hardware ISR are four kinds as shown in Table 1.2. (set_flg(), sig_sem(), rel_wai() and wup_tsk())

The static_hwisr_table array is defined with HWISR_TBL structure and can set 32 Hardware ISR. For example, in the first setting example of Figure 6.1, the Hardware Real-time OS executes the service call “set_flg(ID_APL_FLG1, 0x0001);” automatically when INTPZ1 interrupt occurs. In the second setting example of Figure 6.1, the Hardware Real-time OS executes the service call “wup_tsk(ID_TASK_MAIN);” automatically when INTPZ2 interrupt occurs.

In case of setting inhno to HWISR_TBL_END (0xFFFFFFFF), the kernel considers it 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}
};
```

Figure 6.1 Sequence setting example of static_hwisr_table

7. Utility Functions

rin_hwos_get_version

Description

Gets the version information.

C Language API

char *rin_hwos_get_version(uint8_t mode);

Parameter

I/O	Parameter	Description
I	uint8_t mode	Specifies the version information output mode. 0: Version only 1: Version with build date and hour

Function

This API obtains the OS library version information based on the mode which the parameter specifies.

Return parameter

Strings of version information

hwos_set_mpri_operation

Description

Enable / Disable to use TA_MPRI attribute for Mailbox

C Language API

void hwos_set_mpri_operation(int32_t flag);

Parameter

I/O	Parameter	Description
I	int32_t flag	Setting for TA_MPRI attribute of Mailbox. HWOS_DISABLE_MPRI (0): Disable (default) HWOS_ENABLE_MPRI (1): Enable

Function

This API sets availability of TA_MPRI attribute for Mailbox.

The TA_MPRI attribute is disabled as default if this API is not called. Therefore, you should call this API and enable the TA_MPRI attribute before sending / receiving message to / from Mailbox if you would like to use Mailbox that attribute is TA_MPRI. Note that system may not work correctly if you call this API after sending / receiving message. So, do not call this API after sending / receiving message.

In case of TA_MPRI attribute is enabled and sending message to Mailbox with TA_MPRI attribute, APIs of sending message returns error (E_QOVR) if the sending APIs are issued more than 255 times while one or more messages exist in the mailbox. When this error (E_QOVR) occurs, you should receive all message from the mailbox before sending new message to the mailbox because the error (E_QOVR) is returned until message of the mailbox becomes empty.

Item	Parameter	
	HWOS_DISABLE_MPRI	HWOS_ENABLE_MPRI
Create Mailbox with TA_MPRI attribute	Success	Success
Send message to Mailbox with TA_MPRI attribute	Fail API return parameter: E_RSATR (-11)	Success Note that sending message APIs return error (E_QOVR(-43)) if the sending APIs are issued more than 255 times while one or more messages exist in the mailbox.
Receive message from Mailbox with TA_MPRI attribute	Fail API return parameter: E_RSATR (-11)	Success
Supplement	-	Sending message APIs return E_QOVR until message(s) of the mailbox becomes empty when E_QOVR occurs.

Return parameter

None

8. Setting depending on development tool

This section explains the differences of each development tool.

If ARM or IAR tool is used, the library of compiler is used in startup routine.

OS library has 2 stack pointer, MSP and PSP, are shared with one region in case of IAR and GNU. In case of ARM compiler, PSP and MSP are independent.

PSP is used for task and MSP is used for except task (e.g. interrupt). PSP initial value indicates the stack pointer of 1st activated task.

8.1 ARM

8.1.1 Start up

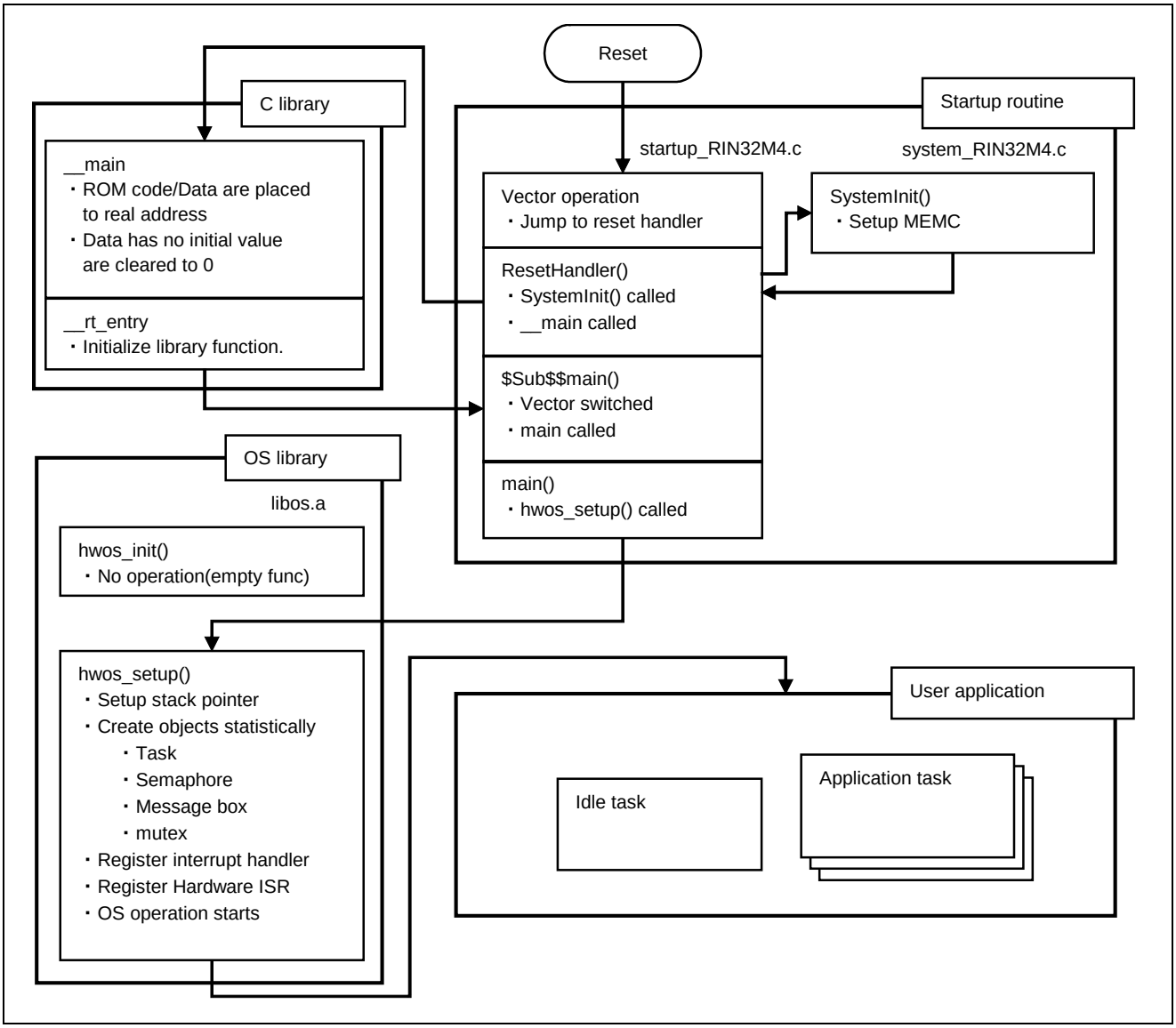


Figure 8.1 Startup routine in time of using ARM

8.1.2 Stack area

The stack area is showed in bellow figure after OS starts (hwos_setup() called). The arrow pointer in the figure shows direction of stack or heap pointer.

If ARM compiler is used, MSP shares with heap region.

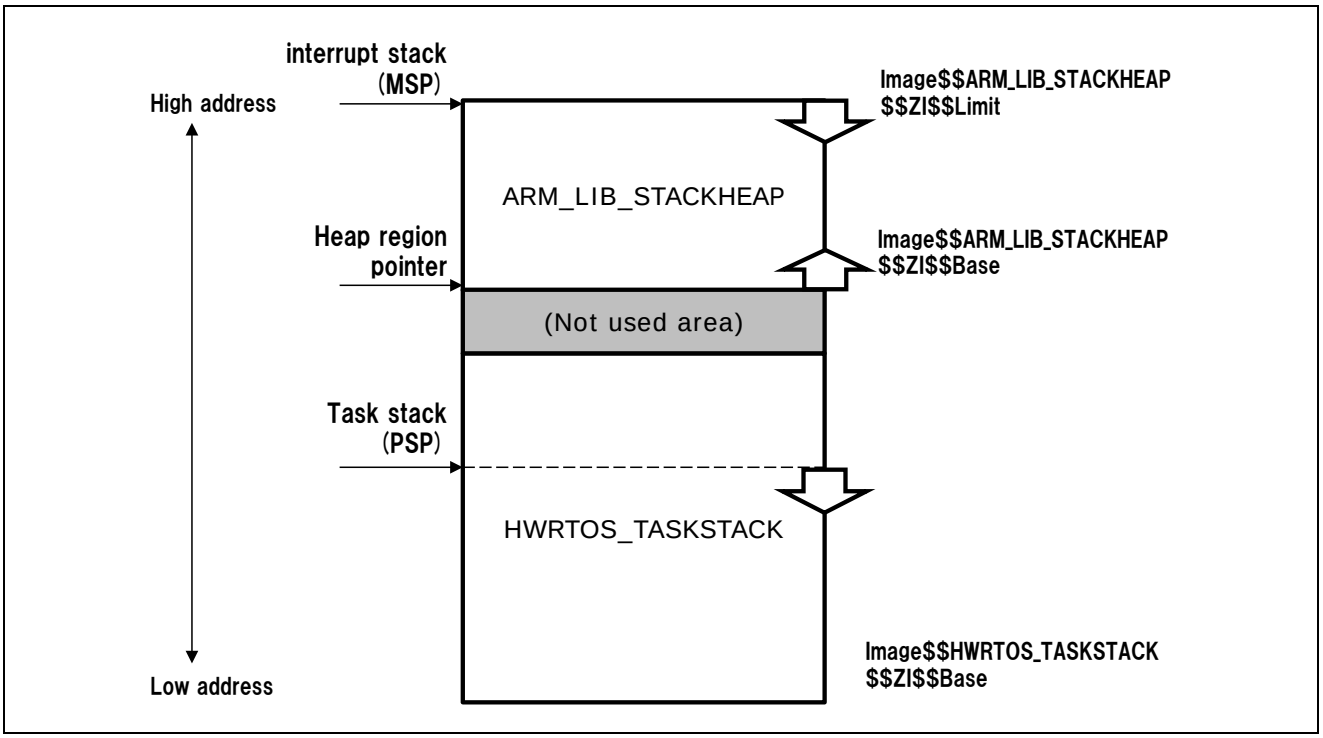


Figure 8.2 Stack area after OS starts (ARM case)

8.2 GNU

8.2.1 Start up

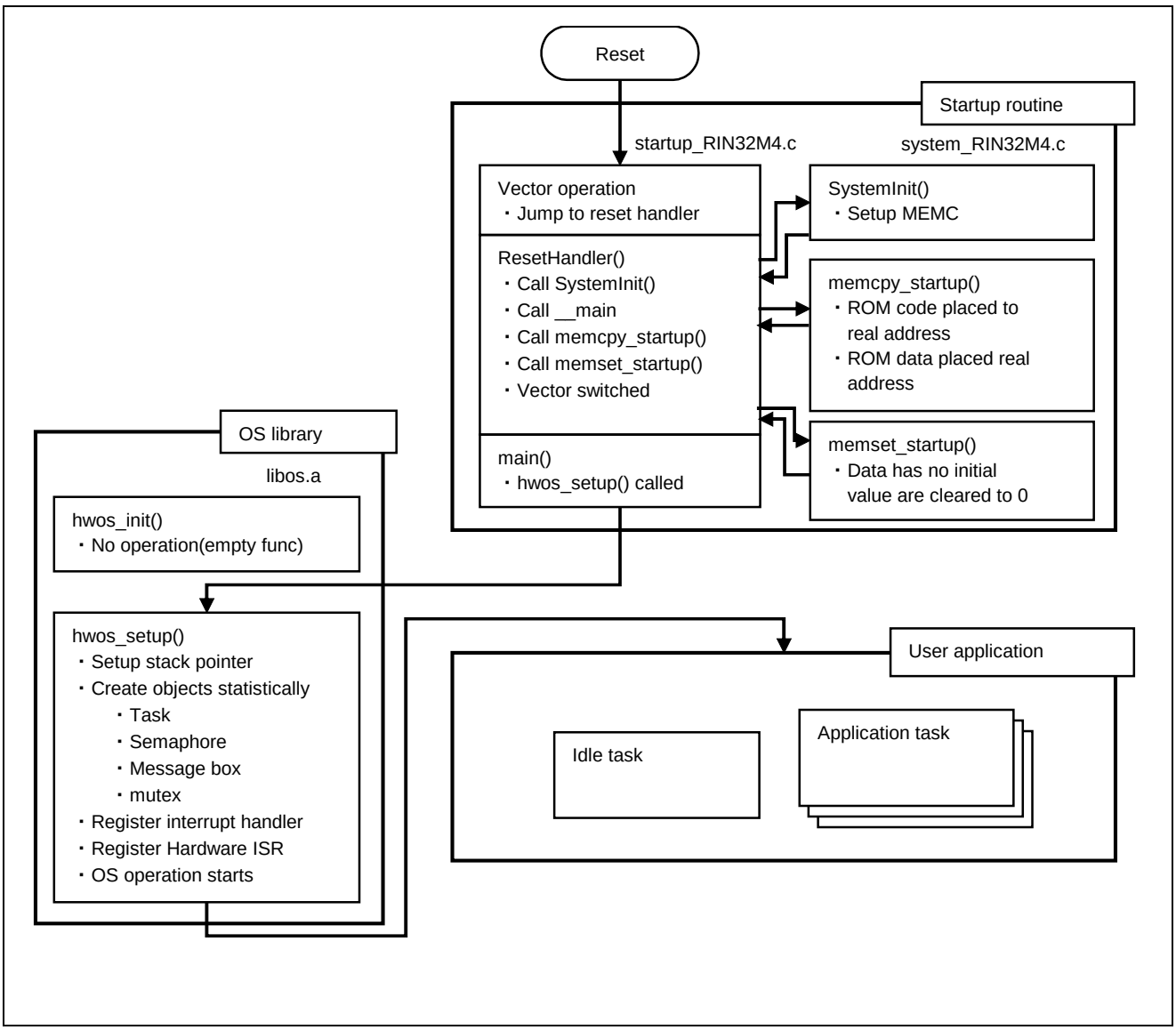


Figure 8.3 Startup routine in time of using GNU

8.2.2 Stack area

The stack area is showed in bellow figure after OS starts (hwos_setup() called). The arrow pointer in the figure shows direction of stack pointer.

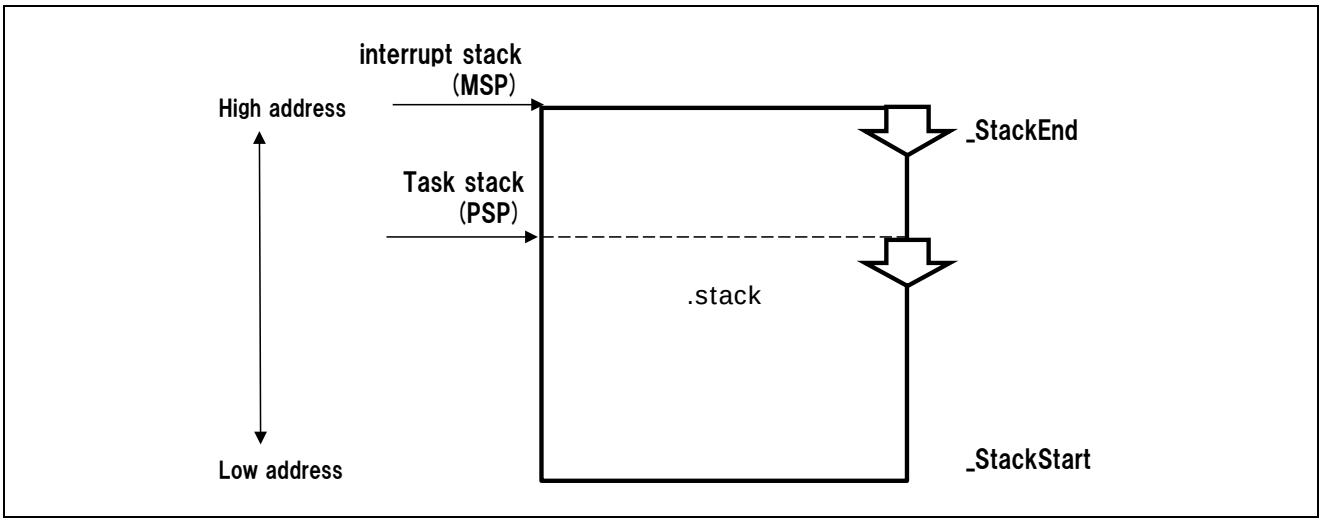


Figure 8.4 Stack area after OS starts (GNU case)

8.3 IAR

8.3.1 Start up

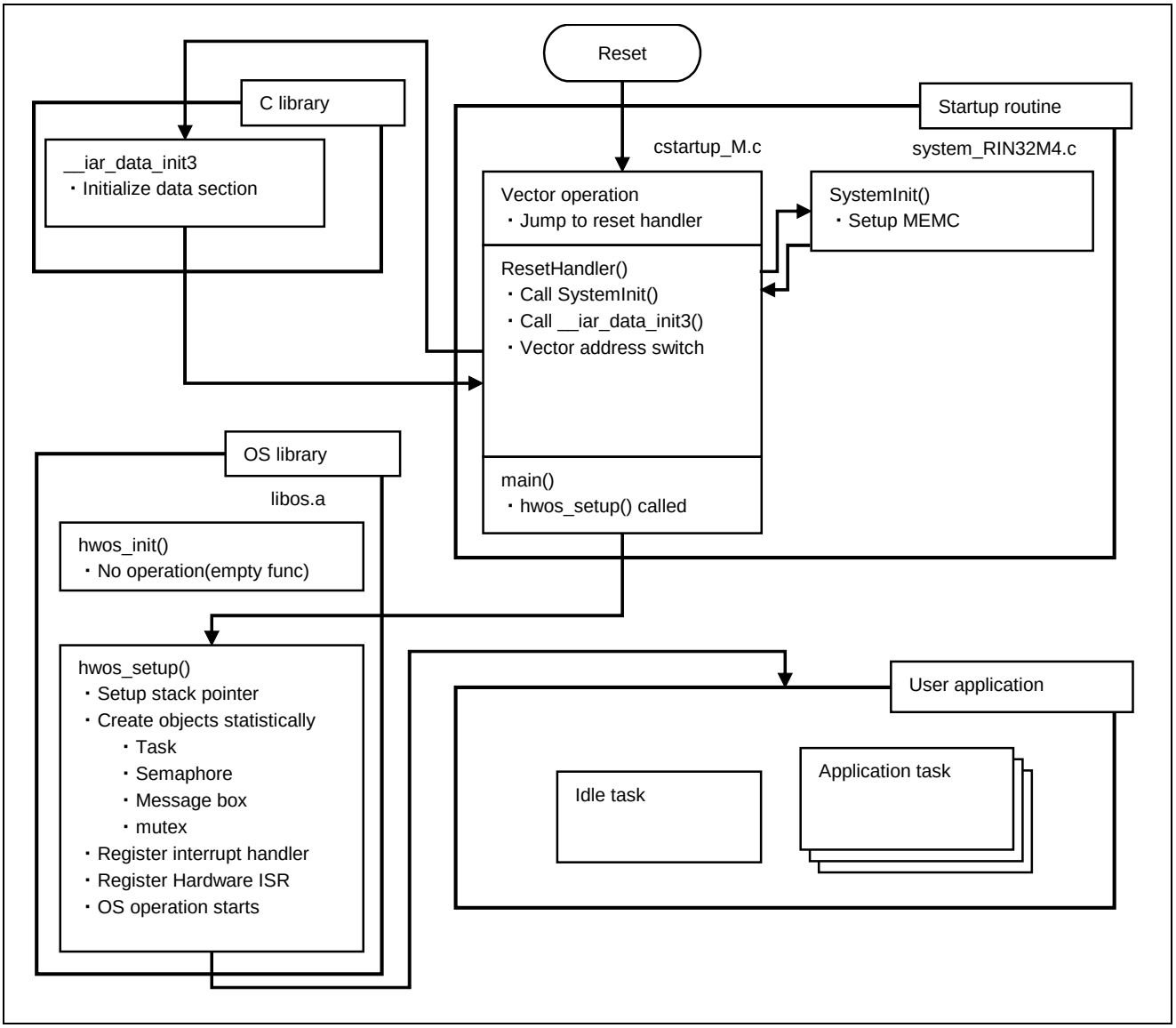


Figure 8.5 Startup routine in time of using IAR

8.3.2 Stack area

The stack area is showed in bellow figure after OS starts (hwos_setup() called). The arrow pointer in the figure shows direction of stack pointer.

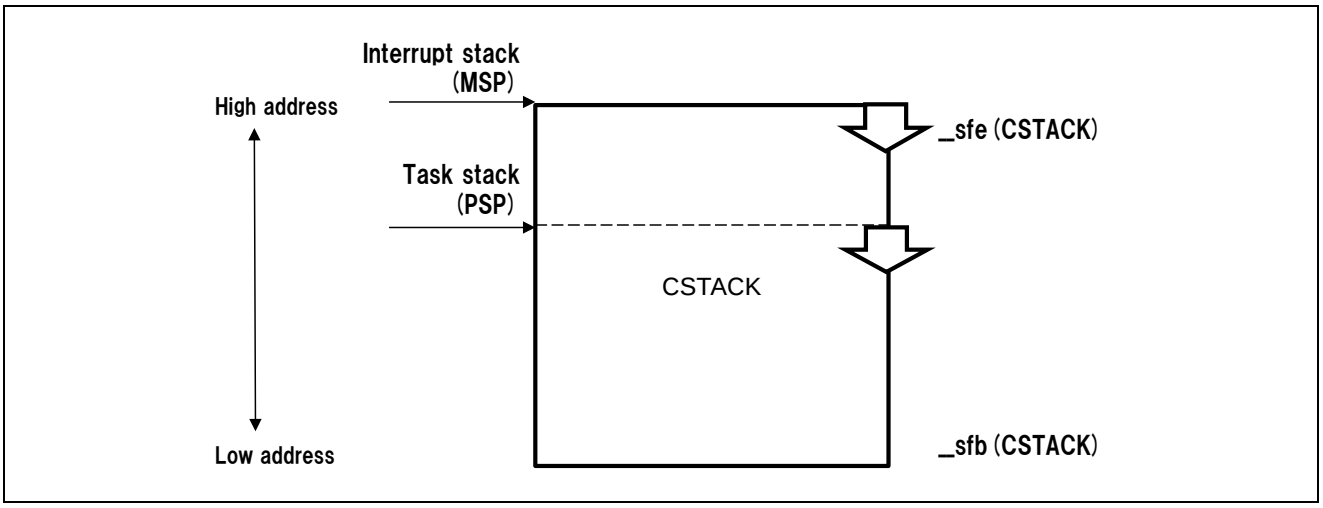


Figure 8.6 Stack area after OS starts (IAR case)

REVISION HISTORY	R-IN32M4-CL2 Programming Manual (OS edition)

Rev.	Date	Description	
		Page	Summary
1.00	Oct 05, 2015	-	First edition issued

[Memo]

R-IN32M4-CL2 Programming Manual (OS edition)

Publication Date: Rev.1.00 Oct 05, 2015

Published by: Renesas Electronics Corporation

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

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

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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

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

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

R-IN32M4-CL2
Programming Manual (OS edition)

Renesas Electronics Corporation

R18UZ0040EJ0100