

To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

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

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

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

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

HI7300/PX V.1.01

User's Manual

Renesas Microcomputer Development Environment System

R0R50730PRW01E

NOTICE:

There are corrections in the "Function" of "7.4.1 Initialize Cache (shx2_vini_cac)" on page 408 in this document.

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.

Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.

The information described here may contain technical inaccuracies or typographical errors.

Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.

Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (<http://www.renesas.com>).

4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

1. μ ITRON is an acronym of the "Micro Industrial TRON" and TRON is an acronym of "The Real Time Operating system Nucleus".
2. TRON, ITRON, and μ ITRON are the names of computer specifications and do not indicate a specific group of the commodity or the commodity.
3. The μ ITRON4.0 specification and μ ITRON4.0 protection function extension are open realtime-kernel specifications defined by the TRON association. The specifications of μ ITRON4.0 and μ ITRON4.0 protection function extension can be downloaded from the TRON association homepage (<http://www.assoc.tron.org>).
4. The copyright of the μ ITRON specification belongs to the TRON association.
5. Microsoft® Windows® 98, Microsoft® Windows® Millennium Edition (Windows® Me), Microsoft® Windows NT®, Microsoft® Windows® 2000, and Microsoft® Windows® XP operating systems are registered trademarks of Microsoft Corporation in the United States and/or other countries.
6. SuperH™ is a trademark of Renesas Technology Corp..
7. All other product names are trademarks or registered trademarks of the respective holders.

Preface

This manual describes how to use the HI7300/PX. Before using the HI7300/PX, please read this manual to fully understand the operating system.

Notes on Descriptions

HEW	Abbreviation of High-Performance Embedded Workshop, which is an integrated development tool.
H', 0x, and D'	For hexadecimal integers, prefix H' or 0x is attached. For decimal integers, prefix D' is attached. If no prefix is attached, a decimal integer is assumed.
<i>shnnnn</i>	Sample files are stored in the directory having the device name (e.g., 'sh73180' directory for the SH73180). This directory is referred to as <i>shnnnn</i> in this manual.
CFG_MAXTSKID	A variable name beginning with CGF_ is specified for the configurator. For details, refer to section 10.6, CFG Name.
samples\	'\' is used to delimit directories and a character string that ends with '\' indicates a directory name. Directory names are basically expressed as relative paths from the kernel installation directory. Note, however, that some paths written in this manual start from the sub-directories of the kernel installation directory when the paths become too long or redundant.

Renesas Technology Homepage

Various support information are available on the following Renesas Technology homepage:

<http://www.renesas.com/en/tools/>

Contents

Section 1	Configuration of This Manual	1
Section 2	Overview	3
2.1	Features.....	3
2.1.1	Memory Object Protection Function.....	3
2.1.2	Conformance to Industry-Standard μ ITRON Specifications	3
2.1.3	DSP/FPU Support	4
2.1.4	Configurator.....	4
2.1.5	Samples.....	4
2.1.6	Debugging Extension (Optional)	4
2.2	Operating Environment.....	5
Section 3	Introduction to Kernel.....	7
3.1	Principles of Kernel Operation	7
3.2	Service Calls	10
3.3	Objects	11
3.4	Tasks	12
3.4.1	Task State.....	12
3.4.2	Task Scheduling (Priority and Ready Queue).....	14
Section 4	Kernel Functions.....	17
4.1	Applications.....	17
4.2	System State.....	18
4.2.1	Task Context and Non-Task Context.....	18
4.2.2	Dispatch-Disabled State, CPU-Locked State, and Dispatch-Pended State	18
4.3	Protection Domains.....	19
4.4	Task Management.....	21
4.4.1	Task Creation.....	21
4.4.2	Domain of a Task.....	21
4.4.3	Task Initiation	21
4.4.4	Task Termination and Deletion.....	22
4.4.5	Priority Change	22
4.4.6	Task Execution Mode	22
4.4.7	Task State Reference.....	23
4.5	Stack Management.....	24
4.5.1	Non-Task Context Stack.....	24

4.5.2	Task Stacks	24
4.6	Task Synchronization.....	25
4.6.1	Synchronization by Task Wakeup	25
4.6.2	Forcible Cancellation of WAITING State	25
4.6.3	SUSPENDED State	26
4.6.4	Task Event Flag	26
4.7	Task Exception Processing	28
4.8	Semaphore	30
4.9	Event Flag	33
4.10	Data Queue	35
4.11	Mailbox.....	37
4.12	Mutex	40
4.13	Message Buffer	43
4.14	Fixed-Size Memory Pool	45
4.15	Variable-Size Memory Pool	48
4.15.1	Fragmentation	51
4.16	Time Management	52
4.16.1	Time Precision	52
4.16.2	System Clock Setting and Reference	53
4.16.3	Cyclic Handler	53
4.16.4	Alarm Handler	56
4.16.5	Overrun Handler	58
4.16.6	Timer Driver	59
4.16.7	Notes on Time Management.....	59
4.17	Optimized Timer Driver	60
4.17.1	Overview	60
4.17.2	Operation	61
4.17.3	Applicable Microcomputers.....	62
4.17.4	Hardware Initialization	63
4.18	Interrupt Management.....	63
4.18.1	Interrupt Handler	63
4.18.2	Kernel Level (CFG_KNLLVL)	64
4.18.3	Disabling Interrupts	64
4.19	CPU Exception	66
4.20	Extended Service Call and Trap.....	67
4.20.1	Extended Service Call.....	68
4.20.2	Trap.....	68
4.21	Memory Object Protection Function.....	68
4.21.1	Overview	68
4.21.2	Memory Object Types	71

4.21.3	Attribute and Domain	71
4.21.4	Access Permission Vector.....	73
4.21.5	Page Size.....	74
4.21.6	Detection of Illegal Access	75
4.21.7	TLB Miss Penalty	76
4.21.8	Access Permission Check (prb_mem)	77
4.21.9	Check for Errors in Address Parameters of Service Calls.....	77
4.21.10	MMU Initialization	77
4.22	Protected Memory Pool	78
4.23	Protected Mailbox	80
4.24	System Memory Management	83
4.24.1	System Pool	83
4.24.2	Resource Pool	84
4.25	DSP Standby Control.....	84
4.25.1	Overview.....	84
4.25.2	Applicable Microcomputers.....	85
4.25.3	Module Stop State of X/Y Memory when Initiating Programs.....	86
4.25.4	Notes	86
4.26	Performance Management	87
4.27	Service Call Trace.....	88
4.28	Other Functions.....	90
4.29	Kernel Idling	91
4.30	Resetting the CPU and Initiating the Kernel.....	92
4.31	Controlling Memory Fragmentation (VTA_UNFRAGMENT Attribute).....	95
4.32	Debugging Extension.....	98
Section 5 Logical Address Space.....		99
5.1	Overview.....	99
5.2	When Memory Object Protection Function Is Not Used	99
5.2.1	Overview.....	99
5.2.2	External Memory	101
5.2.3	On-Chip Memory.....	102
5.3	When Memory Object Protection Function Is Used	103
5.3.1	Overview.....	103
5.3.2	External Memory Space.....	105
5.3.3	On-Chip Memory.....	106
5.3.4	Note on Use	107
5.4	On-Chip Resources Allocated in P4 Area.....	107
5.5	On-Chip Resources whose Physical Addresses Are Allocated in Area 1	107
5.6	32-Bit Address Extended Mode	108

Section 6	Service Calls	109
6.1	C-Language API	109
6.1.1	Calling Form.....	109
6.1.2	Header File.....	109
6.1.3	Header Files Output from the Configurator	109
6.1.4	Basic Data Type.....	110
6.1.5	Constants and Macros.....	111
6.2	Register Contents Guaranteed after Issuing Service Call	115
6.3	Return Value of Service Call and Error Code.....	116
6.3.1	Overview	116
6.3.2	Parameter Check Function.....	116
6.3.3	Access Permission Check Function for Address Parameters.....	116
6.3.4	E_NOSPT Error.....	116
6.4	System State and Service Calls	117
6.4.1	CPU Exception Handler.....	117
6.4.2	Task Context and Non-Task Context.....	117
6.4.3	CPU-Locked State	118
6.4.4	Dispatch-Disabled State.....	118
6.4.5	When SR.IMASK is Modified to a Non-Zero Value through chg_ims in Task Context.....	118
6.5	Service Calls not in the μ TRON4.0 Specification.....	119
6.6	Service Call Description Form.....	120
6.7	Task Management.....	122
6.7.1	Create Task (cre_tsk, icre_tsk, acre_tsk, iacre_tsk).....	124
6.7.2	Delete Task (del_tsk)	132
6.7.3	Initiate Task (act_tsk, iact_tsk)	133
6.7.4	Cancel Task Initiation Request (can_act, ican_act)	135
6.7.5	Initiate Task and Specify Start Code (sta_tsk, ista_tsk).....	136
6.7.6	Exit Current Task (ext_tsk) and Exit and Delete Current Task (exd_tsk)	137
6.7.7	Forcibly Terminate Task (ter_tsk)	139
6.7.8	Change Task Priority (chg_pri, ichg_pri)	140
6.7.9	Refer to Task Priority (get_pri, iget_pri)	142
6.7.10	Refer to Task State (ref_tsk, iref_tsk).....	143
6.7.11	Refer to Task State (Simple Version) (ref_tst, iref_tst)	147
6.7.12	Change Task Execution Mode (vchg_tmd)	149
6.8	Task Synchronization.....	150
6.8.1	Sleep Task (slp_tsk, tslp_tsk)	152
6.8.2	Wake up Task (wup_tsk, iwup_tsk).....	153
6.8.3	Cancel Wakeup Request (can_wup, ican_wup).....	154
6.8.4	Cancel WAITING State Forcibly (rel_wai, irel_wai).....	155

6.8.5	Suspend Task (sus_tsk, isus_tsk).....	156
6.8.6	Resume Task (rsm_tsk, irsm_tsk) and Resume Task Forcibly (frsm_tsk, ifrsm_tsk).....	158
6.8.7	Delay Task (dly_tsk).....	159
6.8.8	Set Task Event Flag (vset_tfl, ivset_tfl).....	160
6.8.9	Clear Task Event Flag (vclr_tfl, ivclr_tfl).....	161
6.8.10	Wait for Task Event Flag (vwai_tfl, vpol_tfl, vtwai_tfl).....	162
6.9	Task Exception Processing Functions.....	164
6.9.1	Define Task Exception Processing Routine (def_tex, ndef_tex)	166
6.9.2	Request Task Exception Processing (ras_tex, iras_tex).....	169
6.9.3	Disable Task Exception Processing (dis_tex)	170
6.9.4	Enable Task Exception Processing (ena_tex)	171
6.9.5	Refer To Task Exception Processing Disabled State (sns_tex)	172
6.9.6	Refer to Task Exception Processing State (ref_tex, iref_tex)	173
6.10	Synchronization and Communication (Semaphore).....	175
6.10.1	Create Semaphore (cre_sem, icre_sem, acre_sem, iacre_sem).....	176
6.10.2	Delete Semaphore (del_sem)	178
6.10.3	Return Semaphore Resource (sig_sem, isig_sem)	179
6.10.4	Wait for Semaphore Resource (wai_sem, pol_sem, ipol_sem, twai_sem)	180
6.10.5	Refer to Semaphore State (ref_sem, iref_sem)	182
6.11	Synchronization and Communication (Event Flag).....	184
6.11.1	Create Event Flag (cre_flg, icre_flg, acre_flg, iacre_flg)	186
6.11.2	Delete Event Flag (del_flg).....	188
6.11.3	Set Event Flag (set_flg, iset_flg).....	189
6.11.4	Clear Event Flag (clr_flg, iclr_flg)	190
6.11.5	Wait for Event Flag Setting (wai_flg, pol_flg, ipol_flg, twai_flg)	191
6.11.6	Refer to Event Flag State (ref_flg, iref_flg).....	194
6.12	Synchronization and Communication (Data Queue).....	196
6.12.1	Create Data Queue (cre_dtq, icre_dtq, acre_dtq, iacre_dtq)	198
6.12.2	Delete Data Queue (del_dtq).....	200
6.12.3	Send Data to Data Queue (snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq).....	201
6.12.4	Receive Data from Data Queue (rcv_dtq, prcv_dtq, trcv_dtq)	203
6.12.5	Refer to Data Queue State (ref_dtq, iref_dtq).....	205
6.13	Synchronization and Communication (Mailbox).....	207
6.13.1	Create Mailbox (cre_mbx, icre_mbx, acre_mbx, iacre_mbx).....	209
6.13.2	Delete Mailbox (del_mbx).....	211
6.13.3	Send Message to Mailbox (snd_mbx, isnd_mbx)	212
6.13.4	Receive Message from Mailbox (rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx) ..	215
6.13.5	Refer to Mailbox State (ref_mbx, iref_mbx)	218

6.14	Synchronization and Communication (Mutex)	220
6.14.1	Create Mutex (cre_mtx, acre_mtx)	221
6.14.2	Delete Mutex (del_mtx)	223
6.14.3	Lock Mutex (loc_mtx, ploc_mtx, tloc_mtx)	224
6.14.4	Unlock Mutex (unl_mtx)	226
6.14.5	Refer to Mutex State (ref_mtx)	227
6.15	Extended Synchronization and Communication (Message Buffer)	229
6.15.1	Create Message Buffer (cre_mbf, icre_mbf, acre_mbf, iacre_mbf)	231
6.15.2	Delete Message Buffer (del_mbf)	234
6.15.3	Send Message to Message Buffer (snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)	235
6.15.4	Receive Message from Message Buffer (rcv_mbf, prcv_mbf, trcv_mbf)	238
6.15.5	Refer to Message Buffer State (ref_mbf, iref_mbf)	240
6.16	Memory Pool Management (Fixed-Size Memory Pool)	242
6.16.1	Create Fixed-Size Memory Pool (cre_mpf, icre_mpf, acre_mpf, iacre_mpf)	244
6.16.2	Create Fixed-Size Memory Pool and Specify Access Permission Vectors (icra_mpf)	248
6.16.3	Delete Fixed-Size Memory Pool (del_mpf)	250
6.16.4	Get Fixed-Size Memory Block (get_mpf, pget_mpf, ipget_mpf, tget_mpf)	251
6.16.5	Release Fixed-Size Memory Block (rel_mpf, irel_mpf)	253
6.16.6	Refer to Fixed-Size Memory Pool State (ref_mpf, iref_mpf)	254
6.17	Memory Pool Management (Variable-Size Memory Pool)	256
6.17.1	Create Variable-Size Memory Pool (cre_mpl, icre_mpl, acre_mpl, iacre_mpl)	258
6.17.2	Create Variable-Size Memory Pool and Specify Access Permission Vectors (ivcra_mpl)	264
6.17.3	Delete Variable-Size Memory Pool (del_mpl)	266
6.17.4	Get Variable-Size Memory Block (get_mpl, pget_mpl, ipget_mpl, tget_mpl)	267
6.17.5	Release Variable-Size Memory Block (rel_mpl, irel_mpl)	270
6.17.6	Refer to Variable-Size Memory Pool State (ref_mpl, iref_mpl)	271
6.18	Time Management (System Clock)	273
6.18.1	Set System Clock (set_tim, iset_tim)	274
6.18.2	Get System Clock (get_tim, iget_tim)	275
6.19	Time Management (Cyclic Handler)	276
6.19.1	Create Cyclic Handler (cre_cyc, icre_cyc, acre_cyc, iacre_cyc)	277
6.19.2	Delete Cyclic Handler (del_cyc)	281
6.19.3	Start Cyclic Handler Operation (sta_cyc, ista_cyc)	282
6.19.4	Stop Cyclic Handler Operation (stp_cyc, istp_cyc)	283
6.19.5	Refer to Cyclic Handler State (ref_cyc, iref_cyc)	284
6.20	Time Management (Alarm Handler)	286
6.20.1	Create Alarm Handler (cre_alm, icre_alm, acre_alm, iacre_alm)	287
6.20.2	Delete Alarm Handler (del_alm)	290

6.20.3	Start Alarm Handler Operation (sta_alm, ista_alm).....	291
6.20.4	Stop Alarm Handler Operation (stp_alm, istp_alm)	292
6.20.5	Refer to Alarm Handler State (ref_alm, iref_alm)	293
6.21	Time Management (Overrun Handler).....	295
6.21.1	Define Overrun Handler (def_ovr)	296
6.21.2	Start Overrun Handler Operation (sta_ovr, ista_ovr).....	298
6.21.3	Stop Overrun Handler Operation (stp_ovr, istp_ovr).....	299
6.21.4	Refer to Overrun Handler State (ref_ovr, iref_ovr)	300
6.22	System State Management	302
6.22.1	Rotate Ready Queue (rot_rdq, irot_rdq)	304
6.22.2	Get Task ID in RUNNING state (get_tid, iget_tid)	305
6.22.3	Get Domain ID of the Task in RUNNING State (get_did, iget_did).....	306
6.22.4	Lock CPU (loc_cpu, iloc_cpu).....	308
6.22.5	Unlock CPU (unl_cpu, iunl_cpu).....	310
6.22.6	Disable Dispatch (dis_dsp)	311
6.22.7	Enable Dispatch (ena_dsp)	312
6.22.8	Refer to Context (sns_ctx)	313
6.22.9	Refer to CPU-Locked State (sns_loc).....	314
6.22.10	Refer to Dispatch-Disabled State (sns_dsp).....	315
6.22.11	Refer to Dispatch-Pended State (sns_dpn).....	316
6.22.12	Start Kernel (vsta_knl, ivsta_knl)	317
6.22.13	System Down (vsys_dwn, ivsys_dwn)	320
6.22.14	Acquire Trace Information (vget_trc, ivget_trc).....	321
6.22.15	Acquire Start of Interrupt Handler as Trace Information (ivbgn_int).....	322
6.22.16	Acquire End of Interrupt Handler as Trace Information (ivend_int)	323
6.22.17	Change DSP (TA_COP0) Attribute (vchg_cop).....	324
6.23	Interrupt Management.....	326
6.23.1	Define Interrupt Handler (def_inh, idef_inh).....	327
6.23.2	Change Interrupt Mask (chg_ims, ichg_ims).....	331
6.23.3	Refer to Interrupt Mask (get_ims, iget_ims).....	333
6.24	Extended Service Call and Trap Management.....	334
6.24.1	Define Extended Service Call (def_svc, idef_svc).....	336
6.24.2	Issue Extended Service Call (cal_svc, ical_svc)	339
6.24.3	Define Trap Routine (vdef_trp, ivdef_trp).....	340
6.25	System Configuration Management.....	343
6.25.1	Define CPU Exception Handler (def_exc, idef_exc).....	344
6.25.2	Refer to Configuration Information (ref_cfg, iref_cfg).....	348
6.25.3	Refer to Version Information (ref_ver, iref_ver)	351
6.26	Memory Object Management Function	353
6.26.1	Change Access Permission Vector for Memory Object (sac_mem)	354

6.26.2	Check Access Permission for Memory Area (prb_mem)	356
6.26.3	Refer to the Memory Object State (ref_mem)	359
6.26.4	Lock TLB Entry (vloc_tlb)	361
6.26.5	Unlock TLB Entry (vunl_tlb)	363
6.27	Protected Memory Pool Management	364
6.27.1	Create Protected Memory Pool (icre_mpp)	365
6.27.2	Poll and Get Protected Memory Block (pget_mpp)	368
6.27.3	Release Protected Memory Block (rel_mpp)	370
6.27.4	Refer to Protected Memory Pool State (ref_mpp)	372
6.28	Protected Mailbox Management	374
6.28.1	Create Protected Mailbox (cre_mbp, icre_mbp, acre_mbp, iacre_mbp)	376
6.28.2	Delete Protected Mailbox (del_mbp)	378
6.28.3	Send Message to Protected Mailbox (snd_mbp)	379
6.28.4	Receive Message from Protected Mailbox (rcv_mbp, prcv_mbp, trcv_mbp)	382
6.28.5	Refer to Protected Mailbox State (ref_mbp, iref_mbp)	385
6.29	System Memory Management	387
6.29.1	Refer to System Pool State (vref_syp)	388
6.29.2	Refer to Resource Pool State (vref_rsp)	390
6.30	Performance Management	392
6.30.1	Start, Stop, or Initialize Performance Measurement (vchg_ppc, ivchg_ppc)	393
6.30.2	Refer to Performance Measurement Result (vref_ppc, ivref_ppc)	395
Section 7 Cache Support Functions		397
7.1	Overview	397
7.2	Notes	398
7.3	Functions in cache_sh4a.h	398
7.3.1	Initialize Cache (sh4a_vini_cac)	399
7.3.2	Clear Cache (sh4a_vclr_cac)	401
7.3.3	Flush Operand Cache (sh4a_vfls_cac)	403
7.3.4	Invalidate Cache (sh4a_vinv_cac)	405
7.4	Functions in cache_shx2.h	407
7.4.1	Initialize Cache (shx2_vini_cac)	407
7.4.2	Clear Cache (shx2_vclr_cac)	409
7.4.3	Flush Operand Cache (shx2_vfls_cac)	411
7.4.4	Invalidate Cache (shx2_vinv_cac)	413
Section 8 Application Program Creation		415
8.1	Tasks	415
8.1.1	Writing a Task	415
8.1.2	Rules on Using Registers	417

8.2	Task Exception Processing Routines	419
8.2.1	Writing a Task Exception Processing Routine.....	419
8.2.2	Rules on Using Registers	420
8.3	Extended Service Call Routines and Trap Routines	422
8.3.1	Writing an Extended Service Call Routine or a Trap Routine	422
8.3.2	Rules on Using Registers	424
8.4	Interrupt Handlers	427
8.4.1	Writing an Interrupt Handler.....	427
8.4.2	Rules on Using Registers	428
8.4.3	DSP and FPU	429
8.4.4	Notes on NMI	429
8.5	Interrupt and Exception Hook Routines.....	430
8.5.1	Overview.....	430
8.5.2	Writing a Hook Routine.....	431
8.5.3	Rules on Using Registers	431
8.5.4	Notes	433
8.6	Time Event Handlers	434
8.6.1	Writing a Time Event Handler.....	434
8.6.2	Rules on Using Registers	435
8.6.3	DSP and FPU	436
8.7	Initialization Routines	437
8.7.1	Writing an Initialization Routine	437
8.7.2	Rules on Using Registers	438
8.7.3	DSP and FPU	439
8.8	CPU Exception Handler.....	440
8.8.1	Writing the CPU Exception Handler	440
8.8.2	Macros Specialized for CPU Exception Handler.....	442
8.8.3	Rules on Using Registers	447
8.8.4	DSP and FPU	448
8.9	Memory Access Violation Handler.....	449
8.9.1	Overview.....	449
8.9.2	Writing the Memory Access Violation Handler	449
8.9.3	Macros Specialized for CPU Exception Handler.....	450
8.9.4	Rules on Using Registers	451
8.9.5	DSP and FPU	452
8.10	System Down Routine	453
8.10.1	Overview.....	453
8.10.2	Writing the System Down Routine	453
8.10.3	Rules on Using Registers	454

Section 9	Standard Timer Driver.....	455
9.1	Overview.....	455
9.2	Configuration of Functions	455
9.2.1	Timer Initialization Routine (_kernel_tmrini()).....	456
9.2.2	Timer Interrupt Routine (_kernel_tmrint())	458
Section 10	Configurator	459
10.1	Overview.....	459
10.2	Linkage Unit, Kernel Lock Mode, and [Kernel Side].....	460
10.3	Configuration Files Output from Configurator	460
10.3.1	Header Files for Application.....	461
10.3.2	System Definition Files	462
10.4	User Interface.....	463
10.4.1	Screen Configuration	463
10.4.2	Title Bar	463
10.4.3	Menu Bar:[File] Menu	464
10.4.4	Menu Bar:[View] Menu.....	465
10.4.5	Menu Bar:[Generate] Menu	466
10.4.6	Menu Bar:[Options] Menu.....	466
10.4.7	Menu Bar:[Help] Menu	467
10.4.8	Toolbar.....	467
10.4.9	Status Bar.....	467
10.4.10	[Navigation] Window	468
10.4.11	[Information Input] Window.....	469
10.5	Page Configuration	470
10.6	CFG Name	472
10.7	Specifications for Pages and Dialog Boxes	472
10.7.1	[Kernel] Page.....	472
10.7.2	[CPU] Page	476
10.7.3	[Definition of On-chip Memory] Dialog Box and [Modification of Information for On-chip Memory Definition] Dialog Box.....	480
10.7.4	[Time Management Function] Page.....	481
10.7.5	[Debugging Function] Page	484
10.7.6	[User Domain] Page.....	486
10.7.7	[Setting of User Domain ID] Dialog Box	487
10.7.8	[Performance] Page.....	487
10.7.9	[Service Call Selection] Page.....	489
10.7.10	[Interrupt/CPU Exception Handler] Page	493
10.7.11	[Modification of Interrupt/CPU Exception Information] Dialog Box	495
10.7.12	[Definition of Interrupt/CPU Exception Handler] Dialog Box.....	496

10.7.13 [Static Memory Object] Page.....	497
10.7.14 [Registration of Static Memory Object] Dialog Box and [Modification of Information for Static Memory Object Registration] Dialog Box	500
10.7.15 [Initialization Routine] Page	505
10.7.16 [Registration of Initialization Routine] Dialog Box and [Modification of Information for Initialization Routine Registration] Dialog Box.....	507
10.7.17 [Task] Page	508
10.7.18 [Modification of Task Information] Dialog Box.....	511
10.7.19 [Creation of Task] Dialog Box and [Modification of Information for Task Creation] Dialog Box	513
10.7.20 [Definition of Task Exception Processing Routine] Dialog Box	516
10.7.21 [Semaphore] Page	517
10.7.22 [Modification of Semaphore Information] Dialog Box	519
10.7.23 [Creation of Semaphore] Dialog Box and [Modification of Information for Semaphore Creation] Dialog Box	520
10.7.24 [Event Flag] Page.....	522
10.7.25 [Modification of Event Flag Information] Dialog Box	523
10.7.26 [Creation of Event Flag] Dialog Box and [Modification of Information for Event Flag Creation] Dialog Box.....	524
10.7.27 [Data Queue] Page	526
10.7.28 [Modification of Data Queue Information] Dialog Box	528
10.7.29 [Creation of Data Queue] Dialog Box and [Modification of Information for Data Queue Creation] Dialog Box	529
10.7.30 [Mailbox] Page	530
10.7.31 [Modification of Mailbox Information] Dialog Box.....	532
10.7.32 [Creation of Mailbox] Dialog Box and [Modification of Information for Mailbox Creation] Dialog Box	533
10.7.33 [Mutex] Page.....	535
10.7.34 [Modification of Mutex Information] Dialog Box	536
10.7.35 [Creation of Mutex] Dialog Box and [Modification of Information for Mutex Creation] Dialog Box.....	537
10.7.36 [Message Buffer] Page.....	538
10.7.37 [Modification of Message Buffer Information] Dialog Box	540
10.7.38 [Creation of Message Buffer] Dialog Box and [Modification of Information for Message Buffer Creation] Dialog Box.....	541
10.7.39 [Estimation of Message Buffer Area Size] Dialog Box	543
10.7.40 [Fixed-size Memory Pool] Page	543
10.7.41 [Modification of Fixed-size Memory Pool Information] Dialog Box.....	545
10.7.42 [Creation of Fixed-size Memory Pool] Dialog Box and [Modification of Information for Fixed-size Memory Pool Creation] Dialog Box	547

10.7.43	[Variable-size Memory Pool] Page.....	550
10.7.44	[Modification of Variable-size Memory Pool Information] Dialog Box	552
10.7.45	[Creation of Variable-size Memory Pool] Dialog Box and [Modification of Information for Variable-size Memory Pool Creation] Dialog Box	553
10.7.46	[Estimation of Variable-size Memory Pool Area Size] Dialog Box	557
10.7.47	[Cyclic Handler] Page.....	557
10.7.48	[Modification of Cyclic Handler Information] Dialog Box	560
10.7.49	[Creation of Cyclic Handler] Dialog Box and [Modification of Information for Cyclic Handler Creation] Dialog Box.....	561
10.7.50	[Alarm Handler] Page.....	563
10.7.51	[Modification of Alarm Handler Information] Dialog Box	565
10.7.52	[Creation of Alarm Handler] Dialog Box and [Modification of Information for Alarm Handler Creation] Dialog Box.....	566
10.7.53	[Overrun Handler] Page.....	567
10.7.54	[Protected Memory Pool] Page.....	569
10.7.55	[Modification of Protected Memory Pool Information] Dialog Box	571
10.7.56	[Creation of Protected Memory Pool] Dialog Box and [Modification of Information for Protected Memory Pool Creation] Dialog Box	572
10.7.57	[Estimation of Protected Memory Pool Area Size] Dialog Box	574
10.7.58	[Protected Mailbox] Page	575
10.7.59	[Modification of Protected Mailbox Information] Dialog Box.....	577
10.7.60	[Creation of Protected Mailbox] Dialog Box and [Modification of Information for Protected Mailbox Creation] Dialog Box.....	578
10.7.61	[Extended Service Call] Page	580
10.7.62	[Modification of Extended Service Call Information] Dialog Box.....	582
10.7.63	[Definition of Extended Service Call Routine] Dialog Box and [Modification of Information for Extended Service Call Routine Definition] Dialog Box	583
10.7.64	[Trap] Page	584
10.7.65	[Modification of Trap Information] Dialog Box.....	586
10.7.66	[Definition of Trap Routine] Dialog Box	587
10.8	Edit Box Specifications.....	589
10.9	Tuning.....	592
10.9.1	Reduction of Used RAM Size.....	592
10.9.2	Reduction of Used ROM Size.....	594
10.9.3	Performance Improvement	595
Section 11 Build		597
11.1	Load Modules	597
11.2	Directory Structure	600
11.3	Overview of Sample System.....	601

11.3.1	Overview.....	601
11.3.2	Lists of Kernel Objects	604
11.3.3	Task Exception Processing	607
11.4	Sample Applications	608
11.4.1	User domain 1 (dom1)	608
11.4.2	User domain 2 (dom2)	608
11.4.3	User domain 3 (dom3)	609
11.4.4	User domain 4 (dom4)	609
11.4.5	User domain 5 (dom5)	610
11.5	System Applications	611
11.5.1	System Down Routine (sysapp\sysdwn.c)	611
11.5.2	Memory Access Violation Handler (sysapp\mavhdr.c)	611
11.5.3	CPU Exception Handler (sysapp\exchdr.c).....	611
11.5.4	Interrupt and Exception Hook Routine (sysapp\inthook.src)	612
11.6	CPU-Dependent Processing.....	613
11.6.1	Standard Timer Driver (tmrdrv.c)	613
11.6.2	CPU Reset Processing.....	613
11.7	Standard Library Functions and Runtime Routines	614
11.7.1	Overview.....	614
11.7.2	Selecting Necessary Standard Library Functions.....	614
11.7.3	stdio.h.....	615
11.7.4	Kernel Objects to be Used	615
11.7.5	Functions Necessary to Use Standard Library Functions	616
11.7.6	Customizing Environment Settings for Standard Library Functions	617
11.7.7	Note on Standard Library Functions.....	618
11.7.8	Section Initialization Function (_INITSCT())	618
11.7.9	Runtime Routines	618
11.8	Monitor	619
11.8.1	Overview.....	619
11.8.2	Monitor Operation	619
11.8.3	Changing Monitor Interrupt.....	620
11.8.4	Monitor Commands	621
11.9	HEW Workspaces and Projects	623
11.9.1	Overview.....	623
11.9.2	Structure of Workspace Directories	624
11.9.3	HEW Build Configuration and Directories for Configurator Files	627
11.9.4	Moving HEW Workspaces	628
11.9.5	Option Settings for Build.....	628
11.10	knl_side.hwp Project in kernel.hws	630
11.10.1	Overview.....	630

11.10.2	Source Files to Be Registered in Project.....	631
11.10.3	Standard Library Generator Settings.....	632
11.10.4	Linkage Editor Settings	636
11.10.5	Build Execution	641
11.11	knl_side_sym.hwp Project in kernel.hws.....	642
11.12	runtime.hwp Project in kernel.hws	643
11.12.1	Overview	643
11.12.2	Standard Library Generator Settings.....	643
11.12.3	Build Execution	645
11.12.4	Notes on Section Initialization.....	645
11.13	env_side.hwp Project in kernel.hws.....	646
11.13.1	Overview	646
11.13.2	Source Files to Be Registered in Project.....	647
11.13.3	Standard Library Generator Settings.....	648
11.13.4	Linkage Editor Settings	650
11.13.5	Build Execution	652
11.14	app_dom5.hwp Project in app_dom5.hws	653
11.14.1	Overview	653
11.14.2	Source Files to Be Registered in Project.....	653
11.14.3	Standard Library Generator Settings.....	654
11.14.4	Linkage Editor Settings	655
11.14.5	Build Execution	657
11.15	Memory Allocation.....	658
11.15.1	Overview	658
11.15.2	Sections.....	658
11.15.3	Notes.....	664
11.15.4	Memory Map and Static Memory Objects.....	664
11.16	Execution on Simulator.....	673
11.16.1	Debugging Session	673
11.16.2	Execution	674
11.16.3	Monitor Startup.....	674
11.16.4	Detection of Illegal Access by Domain 4	674
11.16.5	Execution of Domain 5	675
Section 12 Calculation of Stack Size.....		677
12.1	Stack Types.....	677
12.2	Overview of Calculation Procedure for Stack Size.....	677
12.3	Stack Size Used by Each Task.....	677
12.3.1	Task Associated with User Domain.....	677
12.3.2	Task Associated with Kernel Domain	680

12.4	Calculation of Non-Task Context Stack Size.....	680
12.4.1	Stack Size Used by Each Initialization Routine and Timer Initialization Routine of Standard Timer Driver.....	681
12.4.2	Stack Size Used by Each Interrupt Handler, Time Event Handler, and Timer Interrupt Routine of Standard Timer Driver	682
12.4.3	Stack Size Used by NMI Interrupt Handler	682
12.4.4	Stack Size Used by Each CPU Exception Handler	682
Section 13	Estimation of Resource Pool Size.....	685
13.1	Overview.....	685
13.2	Requested Timing and Size.....	685
13.2.1	When Kernel is Started (vsta_knl)	685
13.2.2	When Object is Created	686
13.2.3	Sizes Used and Released at Other Timings.....	688
13.3	Calculation.....	690
Section 14	Estimation of System Pool Size.....	691
14.1	Overview.....	691
14.2	Requested Timing and Size.....	691
Section 15	Notes on FPU	693
15.1	Meaning of "Using FPU"	693
15.2	FPU Usage in Each Application	693
15.2.1	Task, Task Exception Processing Routine, Extended Service Call Routine, or Trap Routine	693
15.2.2	Other Applications.....	694
Section 16	System Down Handling	695
16.1	Information during System Down.....	695
16.2	Error at Kernel Start (vsta_knl).....	697
16.2.1	System Down Occurrence.....	697
16.2.2	When Object Specified in Configurator Cannot be Created	697
Section 17	Reference Listing	699
17.1	Service Call Reference.....	699
17.2	Service Call Error Code List.....	710

Section 1 Configuration of This Manual

This manual consists of the following sections:

Section 2 ‘Introduction’: Overview of the HI7300/PX

Section 3 ‘Introduction to Kernel’: Basic concept of kernel

Section 4 ‘Kernel Functions’: All the functions of kernel

Section 5 ‘Logical Address Space’: Handling of logical addresses

Section 6 ‘Service Calls’: Specifications of service calls

Section 7 ‘Cache Support Functions’: Specifications of cache support functions

Section 8 ‘Application Program Creation’: Methods for creating a task or a handler

Section 9 ‘Standard Timer Driver’: Methods for creating a standard timer driver

Section 10 ‘Configurator’: Position, functions, and usage of the configurator

Section 11 ‘Build for Sample System’: Description of sample programs and methods to generate a load module using these programs

Section 12 ‘Calculation of Stack Size’: Methods for calculating stack size

Section 13 ‘Estimation of Resource Pool Size’: Methods for estimating the required resource pool size

Section 14 ‘Estimation of System Pool Size’: Methods for estimating the required system pool size

Section 15 ‘Notes on FPU’: Notes on using an FPU

Section 16 ‘System Down Handling’: Methods for handling errors such as a system failure

Section 17 ‘Reference Listing’: References for service calls or error codes

Section 2 Overview

2.1 Features

2.1.1 Memory Object Protection Function

(1) Improving Debugging Efficiency

Generally, in a system that has no memory protection function, a user first notices damage to the memory contents (due to an illegal pointer, for example) when a problem arises. In other words, to identify the cause of a bug, the user had to analyze it using the trace functions of an emulator and this analysis used to take many hours. Since this HI7300/PX supports the memory protection function in instantly detecting illegal memory access, debugging efficiency will be dramatically improved.

(2) Realizing Highly Reliable Systems

The key sections of the system will be protected no matter what kinds of attempts to damage them are made after the shipment of the appliance. Thus, the operation of the appliance will be continued normally.

(3) Low Overhead

The memory object protection function is achieved by the utilization of the memory management unit (MMU) incorporated in the microcomputer. When the MMU is used, a TLB miss overhead is generated. This period is approximately one microsecond* in the HI7300/PX, proving exceptionally high performance. The `vloc_tlb` service call is also supported to avoid TLB misses in the specified memory area.

Note: This is the average value when the memory capacity is 64 Mbytes and a cache hit occurs while running at 400 MHz.

2.1.2 Conformance to Industry-Standard μ ITRON Specifications

The HI7300/PX conforms to the industry-standard μ ITRON4.0 specifications and supports almost all service calls except Rendezvous. The memory object protection function conforms to the μ ITRON4.0 protection function extension. Note, however, that the HI7300/PX does not support any protection function of kernel objects other than the memory object of the μ ITRON4.0 protection function extension.

2.1.3 DSP/FPU Support

The DSP and FPU are supported in the multitasking environment.

2.1.4 Configurator

The configurator is supported to allow easy kernel configuration on the GUI screen.

2.1.5 Samples

The following samples are provided:

- User domains as examples of service calls being used
- Simplified monitor (for exclusive use by the simulator) to view the states of the kernel objects
- System down routines
- Memory access violation handler
- CPU exception handlers
- Reset program and standard timer driver for various microcomputers
- Configurator setting file
- HEW workspaces for load module creation

2.1.6 Debugging Extension (Optional)

A debugging extension for adding multitasking debugging functions to the integrated development environment “HEW” is available. This debugging extension supports the following functions:

- Viewing the states of objects including tasks
- Operating objects (e.g. initiating tasks or setting event flags)
- Displaying service call history

The free debugging extension can be downloaded from our website.

2.2 Operating Environment

Item	Requirement
Target microcomputer	Microcomputer incorporating SH4SL-DSP or SH-4A as the CPU core
Host computer	Personal computer operated under Windows® 98, Windows® Millennium Edition (Windows® Me), Windows NT®4.0, Windows® 2000, or Windows® XP
Compiler	Renesas' C/C++ Compiler Package for SuperH™ RISC engine V.9.00 Release 03 or later

Section 3 Introduction to Kernel

3.1 Principles of Kernel Operation

The kernel is the nucleus program of a realtime operating system.

The kernel enables one microcomputer to appear as if multiple microcomputers are operating. How does the kernel do this?

As shown in figure 3.1, the kernel operates multiple tasks in a time-division manner; the kernel switches (schedules) running tasks at intervals to make it appear as if multiple tasks are running at the same time.

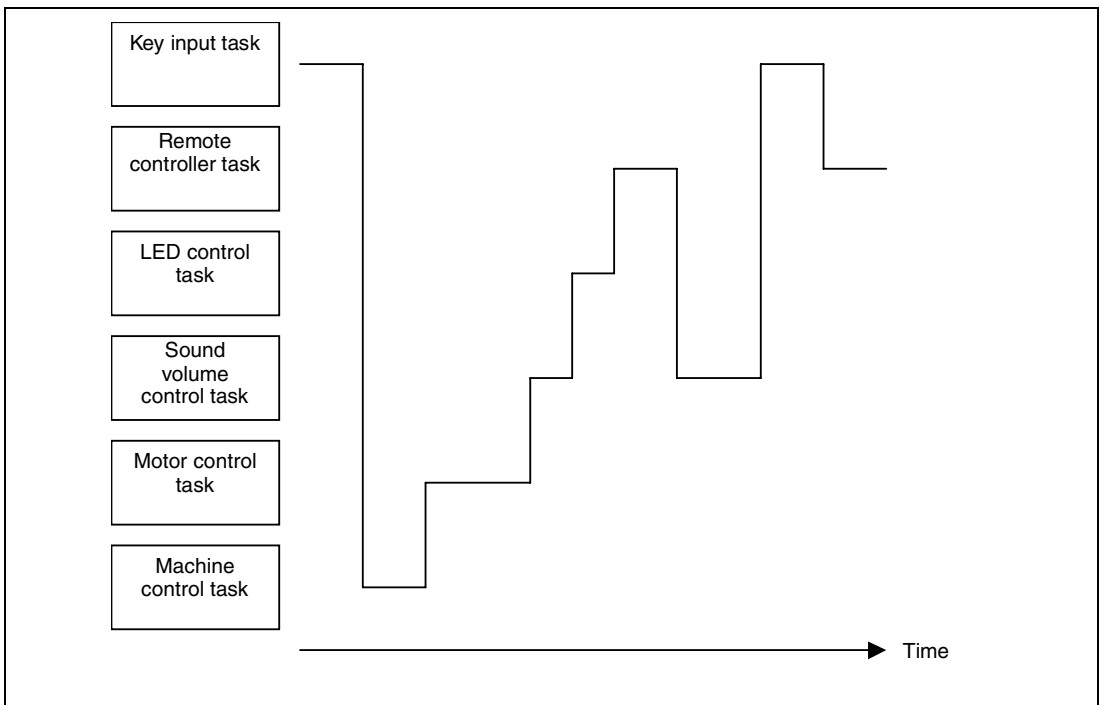


Figure 3.1 Time-Division Operation of Tasks

This task scheduling is also called task dispatch.

The kernel schedules (dispatches) tasks in the following cases.

- When a task itself requests a dispatch
- When an event (such as an interrupt) outside the current task requests a dispatch

This means that tasks are not switched at determined intervals as in a time-sharing system. This type of scheduling is generally called event-driven scheduling.

After tasks are scheduled, a task is resumed from the point where it is suspended (figure 3.2).

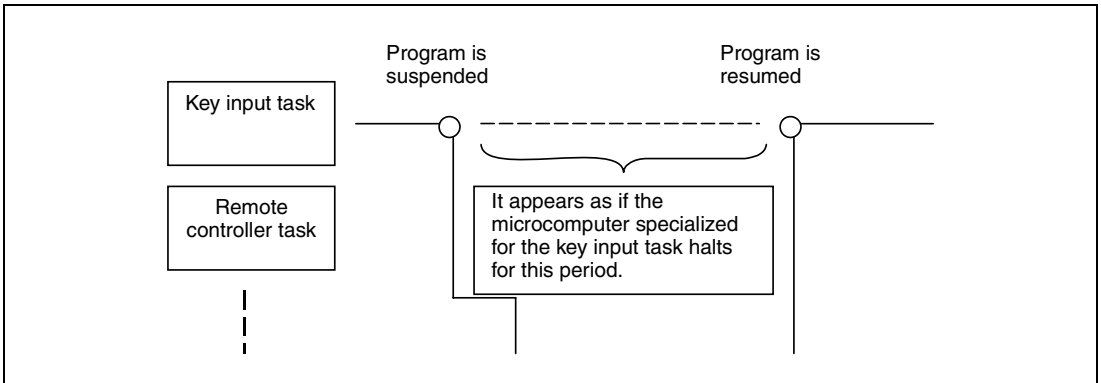


Figure 3.2 Suspending and Resuming a Task

In figure 3.2, while a task is running after obtaining control from the key input task, it appears to the programmer as if the microcomputer specialized for that program has halted.

The kernel restores the register contents stored when the task is suspended, and resumes the task in the state where it was suspended. In other words, task scheduling means saving the register contents for the current task in a memory area prepared for that task management and restoring the register contents of the next task to be resumed (figure 3.3).

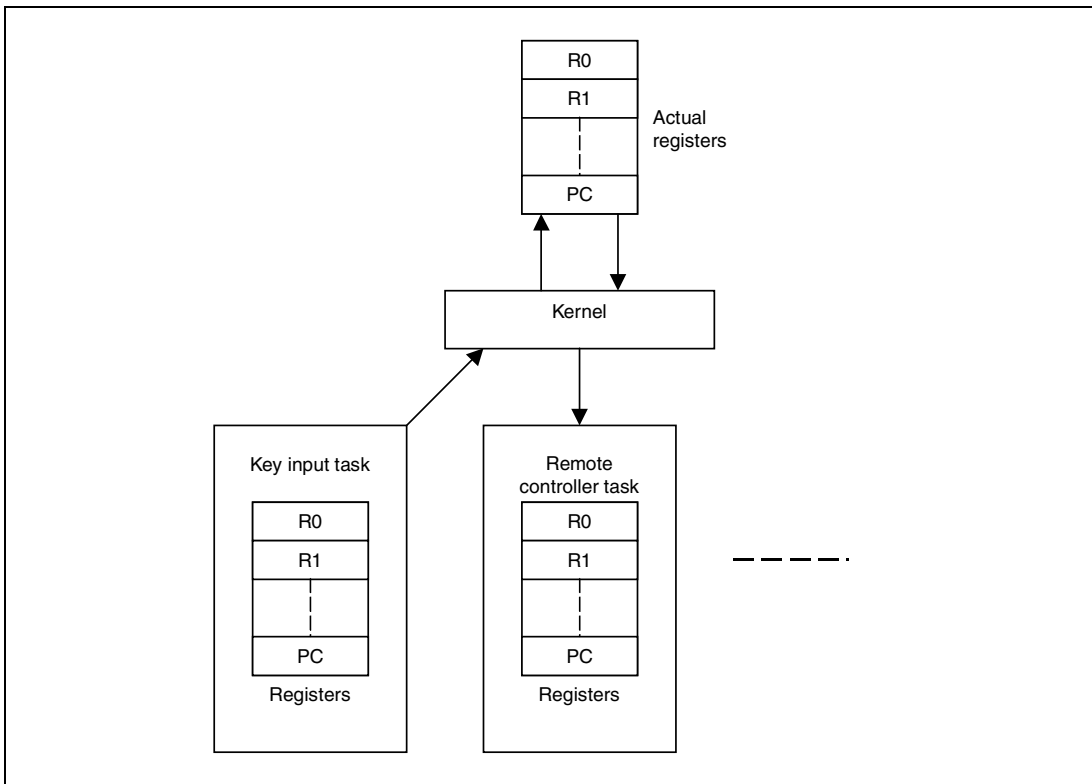


Figure 3.3 Task Scheduling

To execute tasks, stack areas are required in addition to registers. A separate stack area must be allocated for each task.

3.2 Service Calls

How should the programmer use kernel functions in a program?

To use kernel functions, they must be called in a program. This call is a service call. Through service calls, requests for various operations such as task initiation or wait for an event can be sent to the kernel.

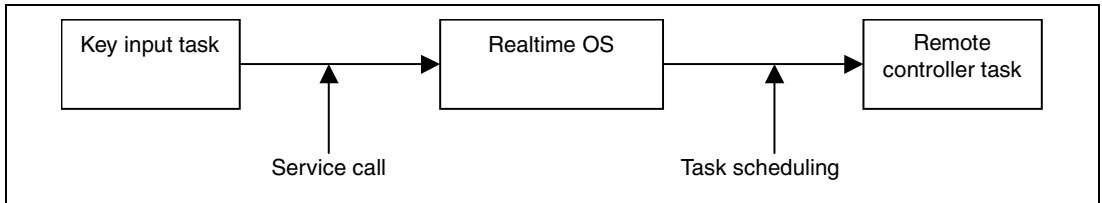


Figure 3.4 Service Call

In actual programs, a service call is issued as a C-language function.

```
act_tsk(1);
```

3.3 Objects

The targets of operation through service calls, such as tasks or semaphores, are called objects. Objects are distinguished by their IDs.

```
act_tsk(1);    /* Initiates the task with ID 1. */
```

Generally, IDs should be specified by the programmer when objects are created.

IDs can also be assigned automatically when objects are created through the configurator. In this case, the automatically assigned IDs are defined in header files (kernel_id.h and kernel_id_sys.h) output from the configurator as shown below.

```
#define ID_TASK1 1
```

By using this example definition, the above task initiating service call is described as follows.

```
act_tsk(ID_TASK1);    /* Initiates the task with ID "ID_TASK1". */
```

Through service calls whose names start with "acre", such as acre_tsk, the kernel assigns IDs automatically and returns the assigned IDs.

3.4 Tasks

The following describes how the kernel manages tasks.

3.4.1 Task State

The kernel checks the task state to control whether to start execution of a task. For example, figure 3.5 shows the state of the key input task and its execution control. When a key input is detected, the kernel must execute the key input task; that is, the key input task enters the running state. While waiting for a key input, the kernel does not need to execute the key input task; that is, the key input task is in the waiting state.

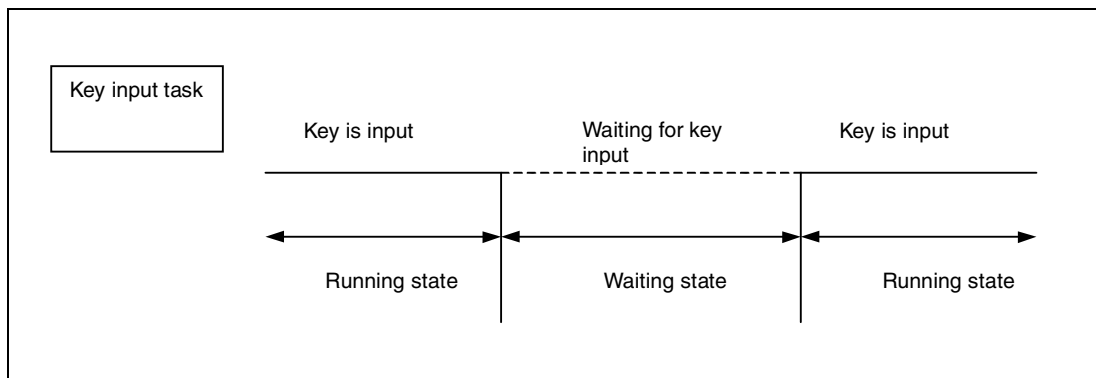


Figure 3.5 Task State

A task transits the seven states shown in figure 3.6.

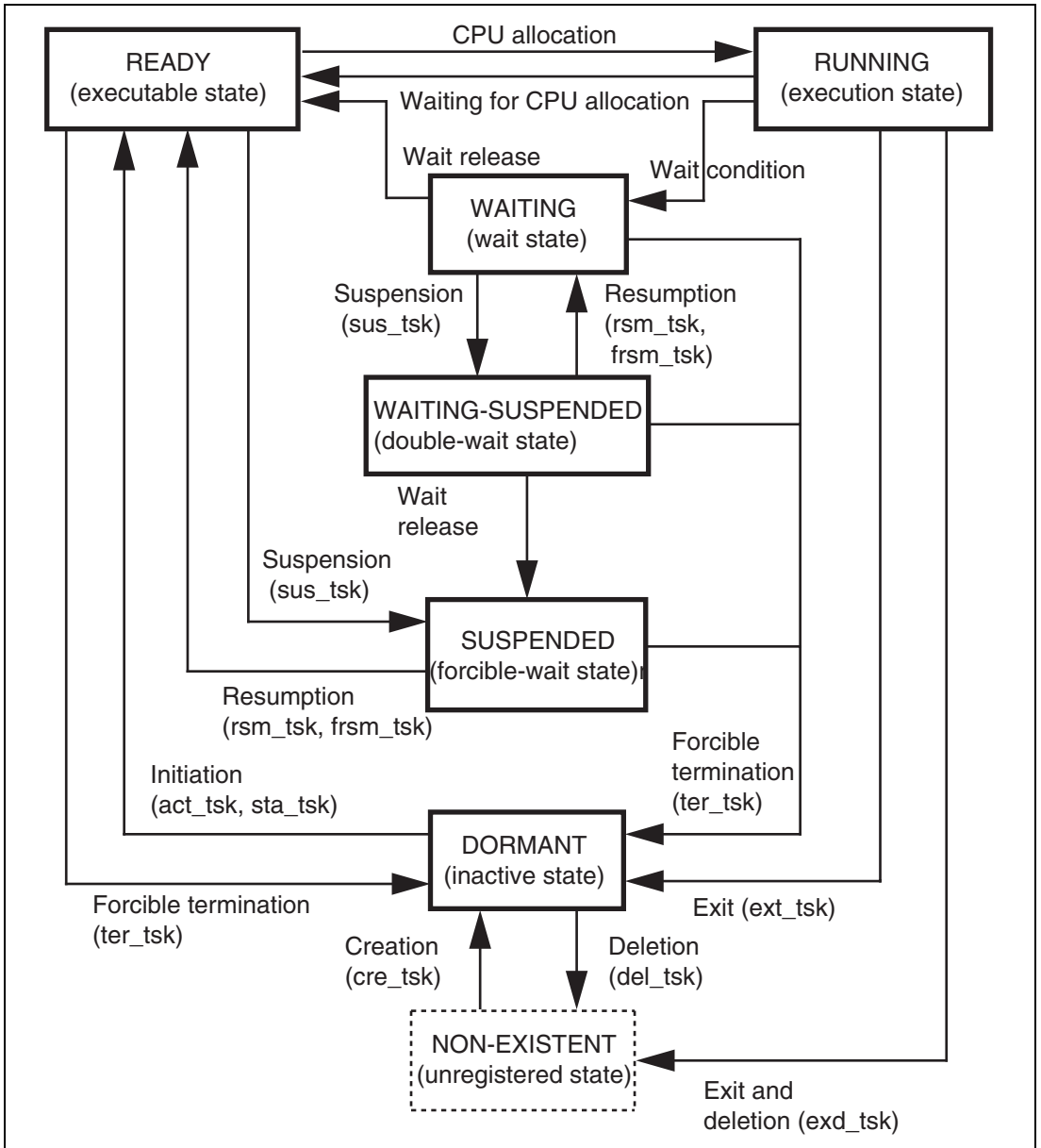


Figure 3.6 Task State Transition Diagram

(1) NON-EXISTENT State

The task has not been registered in the kernel. It is a virtual state.

(2) DORMANT State

The task has been registered in the kernel, but has not yet been initiated, or has already been terminated.

(3) READY (executable) State

The task is ready for execution, but cannot be executed because another higher priority task is currently running.

(4) RUNNING State

The task is currently running. The kernel puts the READY task with the highest priority in the RUNNING state.

(5) WAITING State

When the task issues a service call such as `tslp_tsk` and the specified conditions are not satisfied, the task enters the WAITING state. The task is released from the WAITING state when a service call such as `wup_tsk` to cancel the cause of the WAITING state is issued, and it then enters the READY state.

(6) SUSPENDED State

A task has been suspended by another task through `sus_tsk`.

(7) WAITING-SUSPENDED State

This state is a combination of the WAITING state and SUSPENDED state.

3.4.2 Task Scheduling (Priority and Ready Queue)

For each task, a task priority is assigned to determine the priority of processing. A smaller value indicates a higher priority level and level 1 is the highest priority.

The kernel selects the highest-priority task from the READY tasks and puts it in the RUNNING state.

The same priority can be assigned for multiple tasks. When there are multiple READY tasks with the highest priority, the kernel selects the first task that became READY and puts it in the

RUNNING state. To implement this behavior, the kernel has ready queues, which are READY task queues waiting for execution.

Figure 3.7 shows the ready queue configuration. A ready queue is provided for each priority level, and the kernel selects the task at the head of the ready queue for the highest priority and puts it in the RUNNING state.

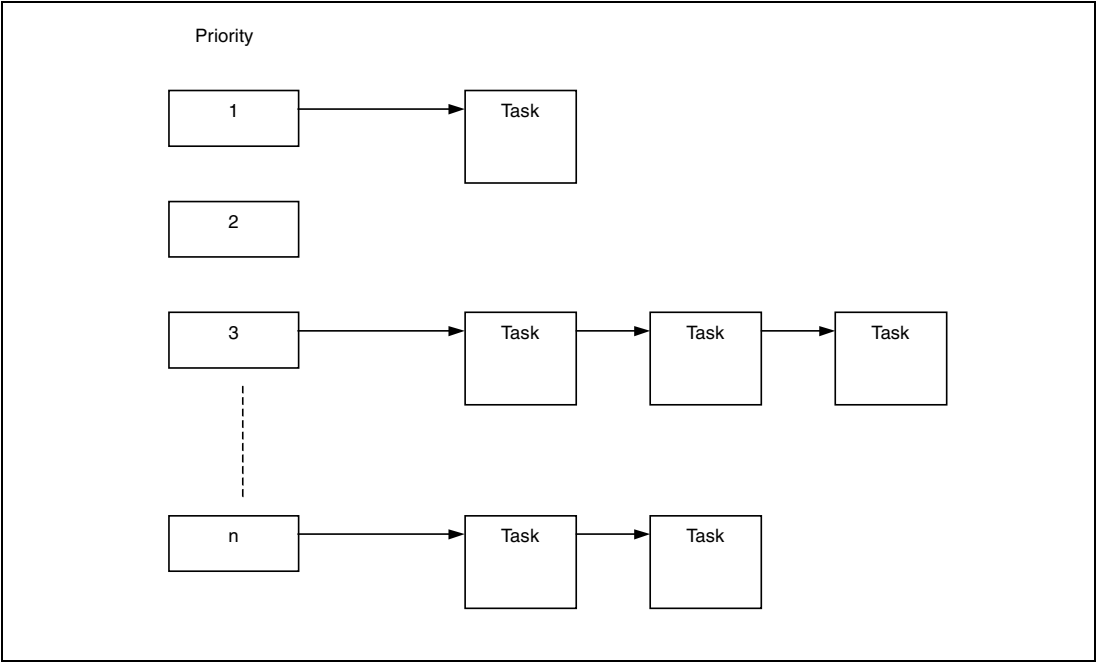


Figure 3.7 Ready Queues (Waiting for Execution)

Section 4 Kernel Functions

4.1 Applications

User applications can be classified into the following types.

(1) Task

A task is a unit controlled by multitasking.

(2) Task Exception Processing Routine

A task exception processing routine is executed when task exception processing is requested for a task (through service call `ras_tex` or `iras_tex`).

(3) Interrupt Handler

An interrupt handler is executed when an interrupt occurs.

(4) CPU Exception Handler

A CPU exception handler is executed when a CPU exception occurs.

(5) Time Event Handler (Cyclic Handler, Alarm Handler, or Overrun Handler)

A time event handler is executed when a specified cycle or time has been reached.

(6) Extended Service Call Routine

Programs created by the user can be registered in the kernel as extended service call routines. A registered extended service call routine is called through extended service call `cal_svc`.

(7) Trap Routine

Programs created by the user can be registered in the kernel as trap routines. A registered trap routine is called when a `TRAPA` instruction is executed.

(8) Initialization Routine

An initialization routine is executed only one time when the kernel is started.

In addition, the following special applications can be used. The symbol names for these applications are prescribed in the kernel specifications and cannot be modified.

(9) System Down Routine

This program is called when the system goes down.

(10) Interrupt or CPU Exception Hook Routine

When an interrupt or CPU exception occurs, the kernel generally calls an appropriate program for the interrupt or exception (interrupt handler, CPU exception handler, trap routine, or service call processing in the kernel). The interrupt or CPU exception hook routine is a program for debugging, and it is hooked and executed before the usual interrupt or exception processing is called.

(11) Memory Access Violation Handler (only when memory object protection function is selected)

This program is called when an illegal access to an MMU mapped area is attempted.

(12) Standard Timer Driver

The standard timer driver consists of a timer initialization routine and a timer interrupt routine. When the optimized timer driver is used, this program is not necessary.

4.2 System State

4.2.1 Task Context and Non-Task Context

The system is executed in either a task context state or a non-task context state. Available service calls depend on this context.

Tasks, task exception processing routines, and extended service call and trap routines called from tasks or task exception processing routines are all executed in task context. The other applications and the kernel are executed in non-task context.

The non-task context takes priority over the task context. Processing in the task context is executed only after all processing in the non-task context is completed. For example, if an interrupt occurs during task execution, the interrupt handler is initiated immediately, and the task execution is temporarily suspended.

Issuing `sns_ctx` can check whether execution is in the task or non-task context.

4.2.2 Dispatch-Disabled State, CPU-Locked State, and Dispatch-Pended State

The system is either in the dispatch-pended state or not in that state.

In the dispatch-pended state, tasks are not scheduled even when a task has a higher priority than the current task. In addition, if a service call to make a transition to the WAITING state is issued in the task context, an E_CTX error is returned.

The kernel updates all management information about the task state or ready queues regardless of whether the system is in the dispatch-pended state, but only stops task scheduling in the dispatch-pended state.

Any of the following cases is in the dispatch-pended state. Issuing sns_dpn can check whether the system is in the dispatch-pended state.

- Execution is in progress in the non-task context.
- The system is in the dispatch-disabled state.
- The system is in the CPU-locked state.
- The current task has modified the IMASK bits in SR to a non-zero value through chg_ims.

(1) Dispatch-Disabled State

Issuing dis_dsp shifts the system to the dispatch-disabled state. The dispatch-disabled state is canceled through ena_dsp.

Issuing sns_dsp can check whether the system is in the dispatch-disabled state.

(2) CPU-Locked State

Issuing loc_cpu or iloc_cpu shifts the system to the CPU-locked state. In the CPU-locked state, interrupts with interrupt levels equal to or lower than CFG_KNLLVL are masked. The CPU-locked state is canceled through unl_cpu or iunl_cpu.

Available service calls are limited in the CPU-locked state.

Issuing sns_loc can check whether the system is in the CPU-locked state.

Reference: Service calls available in the CPU-locked state → Section 6.4.3, CPU-Locked State

4.3 Protection Domains

Every application is assigned to a protection domain. There are two types of protection domain: user domain and kernel domain.

User domains are distinguished by domain IDs from 1 to 31. There is only one kernel domain in the system, and its domain ID is TDOM_KERNEL(-1).

Programs in the kernel domain are executed in the privileged mode (SR.MD = 1) of the CPU.
 Programs in user domains are executed in the user mode (SR.MD = 0) of the CPU.

The following restrictions are applied in the user mode.

- (1) Privileged instructions of the CPU cannot be executed. If this is attempted, a CPU exception will occur.
- (2) The accessible memory regions are limited. The exact limitations depend on whether the memory object protection function is selected.

When the memory protection function is not selected, only the classification between the kernel domain and user domains is valid, and distinction among user domains through IDs is ignored.

Table 4.1 shows the domain of each application.

Table 4.1 Application Domains

Application	Domain	Specification of Domain
Task	Can be assigned to any domain	Specified at task creation
Task exception processing routine	Assigned to the domain where the target task is assigned	—
Interrupt handler	Kernel domain	—
CPU exception handler	Kernel domain	—
Time event handler	Kernel domain	—
Extended service call routine	Kernel domain	—
Trap routine	Kernel domain	—
Initialization routine	Kernel domain	—
System down routine	Kernel domain	—
Interrupt hook routine	Kernel domain	—
Memory access violation handler	Kernel domain	—
Standard timer driver	Kernel domain	—

4.4 Task Management

4.4.1 Task Creation

Tasks can be created (shifted from the NON-EXISTENT state to the DORMANT state) in the following ways.

- Service call `cre_tsk` or `icre_tsk`
A task is created with a specified ID.
- Service call `acre_tsk` or `iacre_tsk`
A task is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

4.4.2 Domain of a Task

A task must be assigned to a domain. The domain for each task must be specified at creation. Tasks in user domains are executed in the user mode (SR.MD = 0) of the CPU; tasks in the kernel domain are executed in the privileged mode (SR.MD = 1) of the CPU.

4.4.3 Task Initiation

A task can be initiated (shifted from the DORMANT state to the READY state) in the following ways.

- Service call `act_tsk` or `iact_tsk`
If the specified task has already been initiated, the initiation request is placed in a queue.
- Service call `sta_tsk` or `ista_tsk`
If the specified task has already been initiated, an error is returned. Parameters can be specified so that they are passed to the task when it is executed.
- Specifying the TA_ACT attribute at task creation
The task becomes READY as soon as it is created.

The following service calls are also provided.

- Service calls `can_act` and `ican_act`
These service calls cancel the initiation request placed in a queue.

4.4.4 Task Termination and Deletion

- Service call `ext_tsk`
This service call terminates the current task, and the task enters the DORMANT state.
- Service call `exd_tsk`
This service call terminates and deletes the current task, and the task enters the NON-EXISTENT state.
- Service call `ter_tsk`
This service call forcibly terminates another task that is not in the DORMANT or NON-EXISTENT state and the task enters the DORMANT state.
- Service call `del_tsk`
This service call deletes another task that is in the DORMANT state, and the task enters the NON-EXISTENT state.

4.4.5 Priority Change

The priority of a task can be changed through `chg_pri` or `ichg_pri`. When the priority is changed, the order of the tasks in the ready queues and the queues arranged in the order of task priority (TA_TPRI) also change.

However, it is generally recommended that these service calls not be used because changing the priority affects the behavior of the entire system.

A task has two priority levels: base priority and current priority. In general operation, these two priority levels are the same; they differ only while the task locks a mutex. For details, refer to the following.

Reference: Difference between base priority and current priority → Section 4.12, Mutex

4.4.6 Task Execution Mode

The task execution mode is unique to the HI7300/PX and is not defined in the μ ITRON specifications.

A task may enter the DORMANT state with an unexpected timing before releasing the acquired resources, due to a forcible termination request (service call `ter_tsk`) issued from another task. In other cases, execution of a task may be suspended with an unexpected timing due to service call `sus_tsk`.

Service call `vchg_tmd` can mask termination or suspension requests issued by `ter_tsk` or `sus_tsk`.

4.4.7 Task State Reference

Service calls `ref_tsk` and `iref_tsk` are provided to refer to the state of a task. These service calls obtain detailed information about a task, such as the task state or the cause of a **WAITING** state.

In addition, service calls `ref_tst` and `iref_tst` are provided as simple versions of `ref_tsk` and `iref_tsk`.

4.5 Stack Management

Stacks are classified into two types: a non-task context stack or a stack for each task.

Reference: Section 12, Calculation of Stack Size

4.5.1 Non-Task Context Stack

There is only one non-task context stack in the system, which is used when execution is in the non-task context. Specify the size of the stack through `CFG_NTSKSTKSZ` in the configurator.

The kernel switches stacks to use the non-task context stack when execution shifts from the task context to the non-task context.

4.5.2 Task Stacks

(1) Task in User Domain

A task in a user domain has a system stack in addition to the stack used to execute the task. The kernel and the extended service call or trap routine called from the task uses the system stack to store the task context. The system stack cannot be accessed in the user mode.

The addresses of both stacks can be specified at task creation, or can be specified to be automatically assigned by the kernel. When a task is created through the configurator, stack addresses cannot be specified.

When the kernel automatically allocates stacks, the usual stack is allocated in the system pool and the system stack is allocated in the resource pool.

When the memory object protection function is selected, the stack allocated in the system pool by the kernel is handled as a memory object that can be read or written to only by the associated task.

(2) Task in Kernel Domain

A task in the kernel domain has one stack, which cannot be accessed in the user mode.

The address of the stack can be specified at task creation, or can be specified to be automatically assigned by the kernel. When a task is created through the configurator, the stack address cannot be specified.

When the kernel automatically allocates a stack, the stack is allocated in the resource pool.

4.6 Task Synchronization

Synchronization between tasks is achieved by using specialized functions associated with tasks.

4.6.1 Synchronization by Task Wakeup

Issuing slp_tsk or tslp_tsk shifts a task to the WAITING state until iwup_tsk or iwup_tsk wakes it up. In tslp_tsk, a timeout period before wakeup can also be specified.

By using these service calls, synchronization between tasks is achieved as shown in figure 4.1. This example shows the simplest way to ensure synchronization.

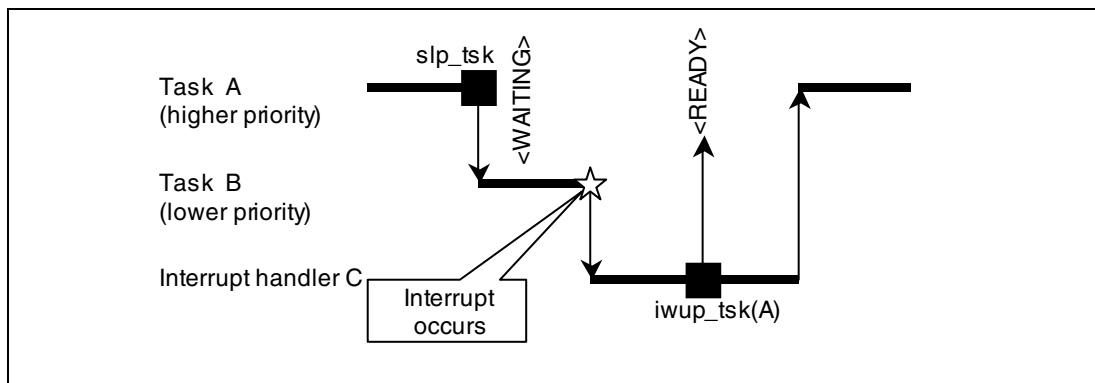


Figure 4.1 Example of Synchronization by Task Wakeup

While the task is not in the WAITING state caused by `slp_tsk` or `tslp_tsk`, the wakeup requests issued are placed in a queue. The wakeup requests can be canceled through `can_wup` or `ican_wup`.

4.6.2 Forcible Cancellation of WAITING State

Issuing `rel_wai` or `irel_wai` forcibly cancels the WAITING state of a task. Note that these service calls cannot cancel the SUSPENDED state.

4.6.3 SUSPENDED State

Issuing `sus_tsk` or `isus_tsk` forcibly suspends another task (the SUSPENDED state). The suspension requests are nested and stored. Service calls `rsm_tsk` and `irms_tsk` decrement the number of nested suspension requests, and when the number reaches 0, the SUSPENDED state is canceled. Service call `frsm_tsk` or `ifrms_tsk` immediately releases the task from the SUSPENDED state regardless of the number of nested requests.

If `sus_tsk` or `isus_tsk` is issued for a task that is already in the SUSPENDED state, the task enters the WAITING-SUSPENDED state.

The SUSPENDED state is generally used for debugging; it is recommended that the SUSPENDED state not be used in actual applications.

4.6.4 Task Event Flag

Task event flags are unique objects of the HI7300/PX and are not defined in the μ ITRON specifications.

Each task has a task event flag consisting of 32 bits; each bit corresponding to an event. A task can wait for a specified bit to be set (`vwai_tfl` or `vtwai_tfl`) or poll a bit (`vpol_tfl`). In `vtwai_tfl`, a timeout period can also be specified.

When an event is detected through one of these service calls, the task event flag is cleared to 0 and the value of the task event flag immediately before being cleared (the detected event) is returned through a parameter of the service call.

To notify a task of an event, use `vset_tfl` or `ivset_tfl`. Each service call sets the specified bits in the task event flag.

`vcclr_tfl` and `ivclr_tfl` clear the specified bits in the task event flag.

Figure 4.2 shows an example of task event flag operation.

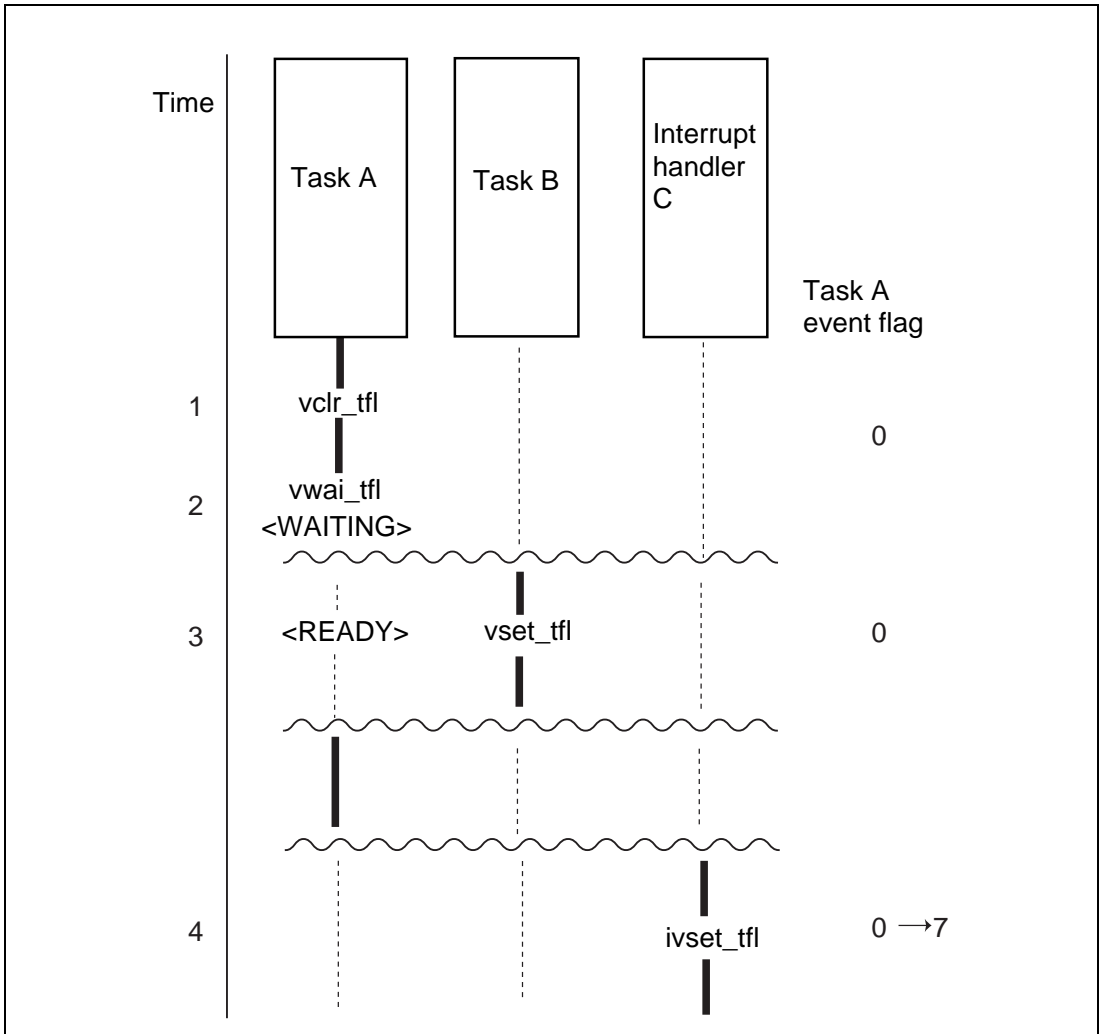


Figure 4.2 Example of Task Event Flag Operation

Description:

1. Task A issues `vclr_tfl` to clear all bits in its task event flag.
2. Task A issues `vwai_tfl` (waiting pattern = H'ffffff) to wait for an event.
3. Task B issues `vset_tfl` (set pattern = 1) to task A. Since this set pattern is included in the waiting pattern specified in task A, the `WAITING` state of task A is canceled, and the task A event flag is cleared to 0.

4. Interrupt handler C issues `ivset_tfl` (set pattern = 7) to set the event flag of task A. In this case, however, task A does not wait for an event, therefore the task event flag is logically ORed with a pattern specified by `ivset_tfl`.

4.7 Task Exception Processing

Task exception processing is performed when an exception occurs during task execution. Task exception processing is performed asynchronously with task processing and is similar to the function generally called "signal".

Task exceptions are handled by task exception processing routines, which can be defined in the following ways.

- Defined through service call `def_tex` or `idef_tex`
- Defined by the configurator

Task exception processing is controlled through the following service calls.

- Service call `ras_tex` or `iras_tex`: Requests task exception processing
- Service call `ena_tex`: Enables task exception processing for the current task
- Service call `dis_tex`: Disables task exception processing for the current task
- Service call `sns_tex`: Checks if the current task is in task exception processing disabled state
- Service call `ref_tex` or `iref_tex`: Refers to the task exception processing state

Figure 4.3 shows an example of task exception processing.

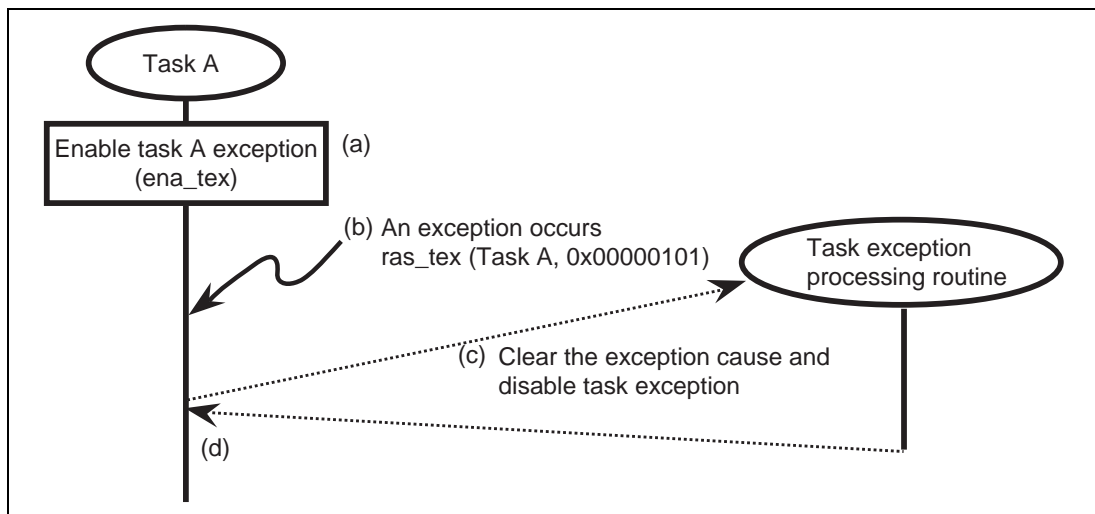


Figure 4.3 Example of Task Exception Processing

Description (Letters indicate the order of operation):

- (a) Task A enables a task exception through service call `ena_tex`.
- (b) An exception (exception cause = 0x00000101) is requested to task A through service call `ras_tex` during task A execution.
- (c) When task A is scheduled to execute, the task exception processing routine is initiated before the task A main routine is executed. During task exception processing, the task enters the task exception processing disabled state, and the task exception cause is cleared.
- (d) After returning from the task exception processing routine, the task A main routine is resumed.

4.8 Semaphore

A semaphore is an object used to prevent conflicts over resources such as devices shared by multiple tasks.

A semaphore has a semaphore counter. Acquiring or releasing each semaphore according to the value of the semaphore counter prevents resource conflicts.

Applications must be programmed so that the number of resources is specified as the initial value of a semaphore and each task acquires a semaphore before using a resource and releases it after completing use of the resource.

Semaphores can be created in the following ways.

- Service call `cre_sem` or `icre_sem`
A semaphore is created with a specified ID.
- Service call `acre_sem` or `iacre_sem`
A semaphore is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Semaphores are manipulated through the following service calls.

- Service call `del_sem`
Deletes the semaphore with the specified ID.
- Service call `wai_sem` or `twai_sem`
Acquires a semaphore. If no semaphore can be acquired (semaphore count = 0), the task enters the WAITING state. In `twai_sem`, a timeout period can also be specified.
- Service call `pol_sem` or `ipol_sem`
Acquires a semaphore. If no semaphore can be acquired (semaphore count = 0), an error is returned.
- Service call `sig_sem` or `isig_sem`
Releases a semaphore.
- Service call `ref_sem` or `iref_sem`
Refers to the state of a semaphore.

Figure 4.4 shows an example of semaphore usage.

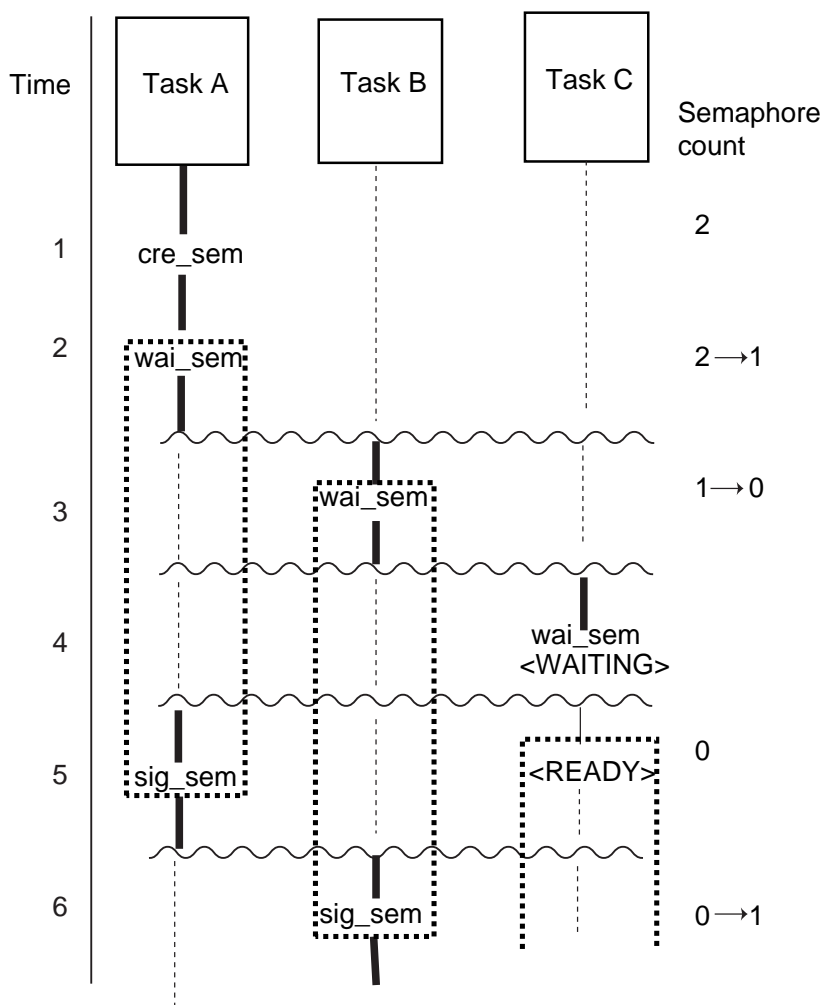


Figure 4.4 Example of Semaphore Usage

Description:

Dotted boxes represent the regions where tasks can exclusively access resources.

1. Task A creates a semaphore with initial counter value 2 through cre_sem.
2. Task A issues wai_sem and gets a semaphore, decrementing the semaphore count by 1. Task A continues execution.

3. Task B issues `wai_sem` in the same way as task A.
4. Task C issues `wai_sem`, but cannot get a semaphore because the semaphore count is 0. Task C enters the WAITING state.
5. Task A releases a semaphore by issuing `sig_sem`. The released semaphore is allocated to task C, and task C is released from the WAITING state.
6. Task B releases a semaphore by issuing `sig_sem`. There is no task waiting for a semaphore, and so the semaphore count is incremented by 1.

Priority Inversion:

When a semaphore is used, a problem called priority inversion may arise.

As shown in figure 4.5, when high-priority task A requests a semaphore, the time needed for task A to acquire a semaphore depends on the execution time of task B, but task B is not related to task C which has acquired the semaphore. To avoid this problem, use a mutex instead of a semaphore.

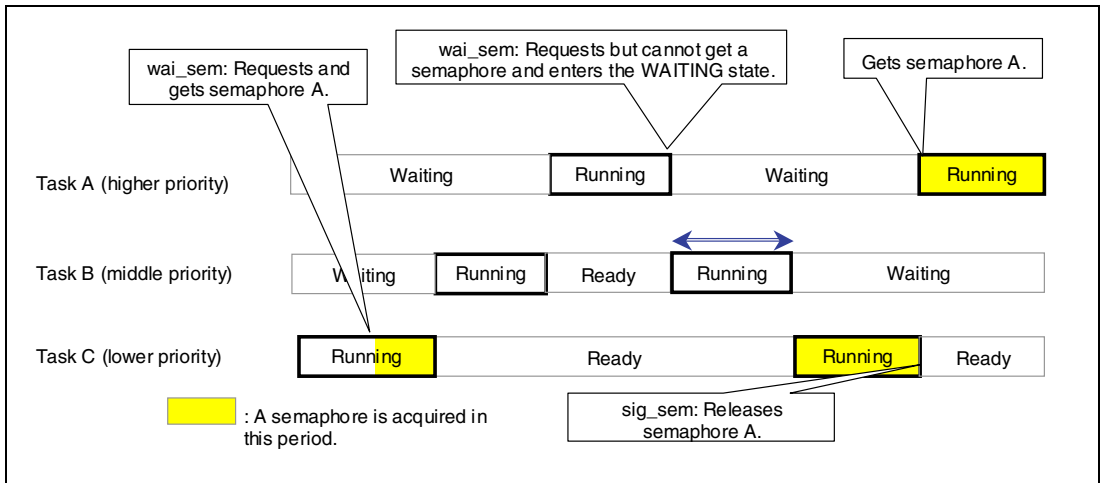


Figure 4.5 Priority Inversion Problem

High-priority task A cannot acquire a semaphore until low-priority task C releases the semaphore. However, while middle-priority task B is being executed, low-priority task C, which has acquired the semaphore, cannot be executed, and the timing for release of the semaphore is delayed for that period (indicated by \longleftrightarrow in the figure). As a result, high-priority task A is kept waiting by middle-priority task B which has not requested any semaphore.

4.9 Event Flag

An event flag is an object consisting of 32 bits; each bit corresponding to an event.

A task can wait for a specified bit or all bits to be set. Whether more than one task is allowed to wait for a specified bit to be set in an event flag can be specified at event flag creation.

Event flags can be created in the following ways.

- Service call `cre_flg` or `icre_flg`
An event flag is created with a specified ID.
- Service call `acre_flg` or `iacre_flg`
An event flag is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Event flags are manipulated through the following service calls.

- Service call `del_flg`
Deletes the event flag with the specified ID.
- Service call `wai_flg` or `twai_flg`
Checks if the specified bit in an event flag has been set. If the bit is not set, the task enters the WAITING state until the bit is set. In `twai_flg`, a timeout period can also be specified.
- Service call `pol_flg` or `ipol_flg`
Checks if the specified bit in an event flag has been set. If the bit is not set, an error is returned.
- Service call `set_flg` or `iset_flg`
Sets the specified bit in an event flag.
- Service call `clr_flg` or `iclr_flg`
Clears the specified bit in an event flag.
- Service call `ref_flg` or `iref_flg`
Refers to the state of an event flag.

Figure 4.6 shows an example of event flag usage.

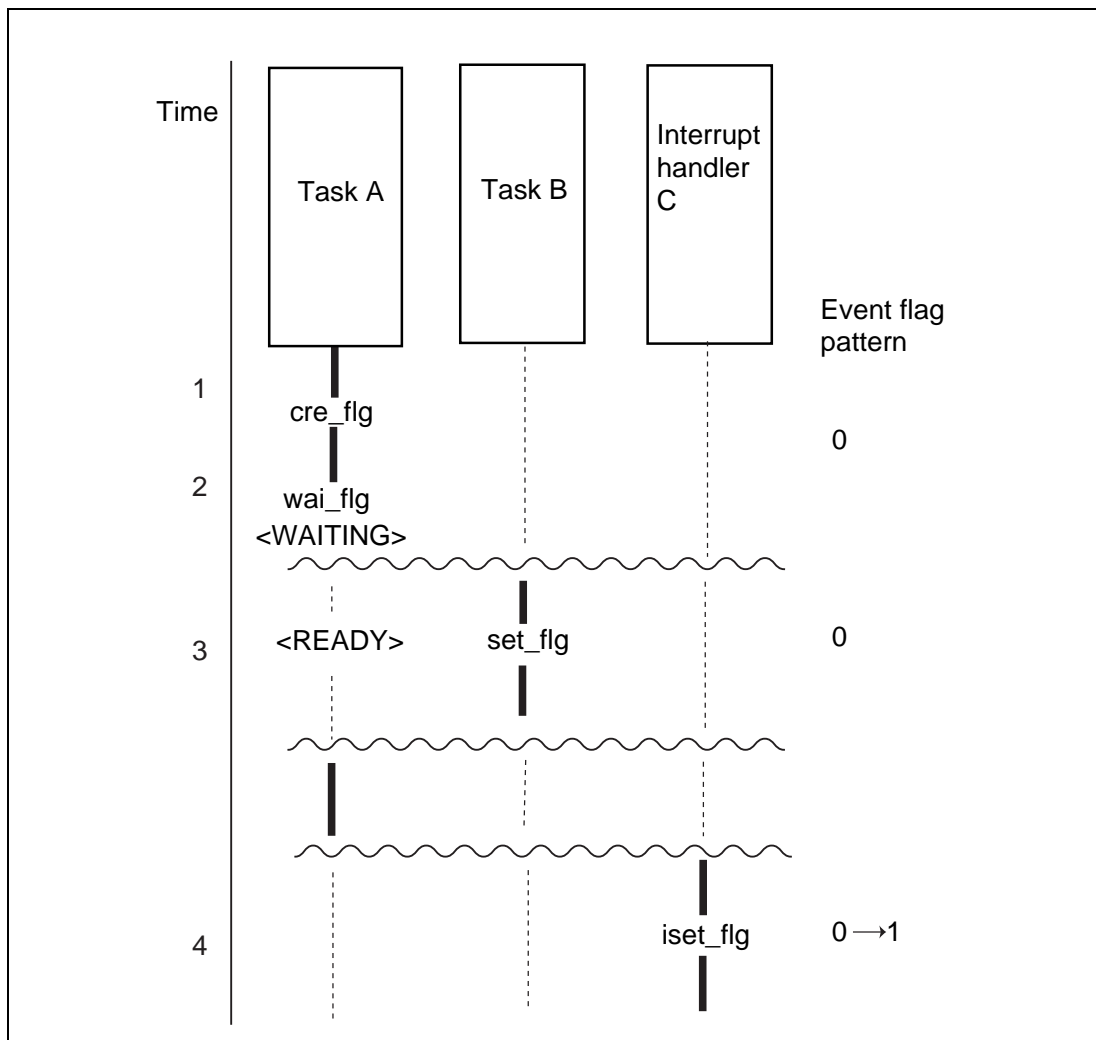


Figure 4.6 Example of Event Flag Usage

Description:

1. Task A issues `cre_flg` to create an event flag. The `TA_CLR` attribute (clear event flag to 0 when the `WAITING` state is canceled) is specified and the initial pattern is specified as 0.
2. Task A issues `wai_flg` (waiting pattern = 3, AND wait) to wait for an event.
3. Task B issues `set_flg` (set pattern = 7). Since all bits that task A is waiting for have been set, task A is released from the `WAITING` state. In addition, since the `TA_CLR` attribute has been specified, the event flag is cleared to 0.

4. Interrupt handler C sets the event flag by issuing `iset_flg` (set pattern = 1). In this case, there is no task waiting for an event, and so the event flag is ORed with the pattern specified by `iset_flg`.

4.10 Data Queue

A data queue is an object used to pass 1-word (4-byte) data. High-speed data communication can be achieved using data queues because communication using a data queue copies 1-word data itself. A pointer can also be specified as data.

A data queue area is allocated in the resource pool.

Data queues can be created in the following ways.

- Service call `cre_dtq`, `icre_dtq`
A data queue is created with a specified ID.
- Service call `acre_dtq`, `iacre_dtq`
A data queue is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Data queues are manipulated through the following service calls.

- Service call `del_dtq`
Deletes the data queue with the specified ID.
- Service call `snd_dtq` or `tsnd_dtq`
Sends data. When data cannot be sent (the data queue is full of data), the task enters the WAITING state. In `tsnd_dtq`, a timeout period can also be specified.
- Service call `psnd_dtq` or `ipsnd_dtq`
Sends data. When data cannot be sent (the data queue is full of data), an error is returned.
- Service call `fsnd_dtq` or `ifsnd_dtq`
Sends data. When data cannot be sent (the data queue is full of data), the oldest data is deleted and the new data is sent.
- Service call `rcv_dtq` or `trcv_dtq`
Receives data. When data cannot be received (the data queue has no data), the task enters the WAITING state. In `trcv_dtq`, a timeout period can also be specified.
- Service call `prcv_dtq`
Receives data. When data cannot be received (the data queue has no data), an error is returned.

- Service call `ref_dtq` or `iref_dtq`
Refers to the state of a data queue.

Figure 4.7 shows an example of data queue usage.

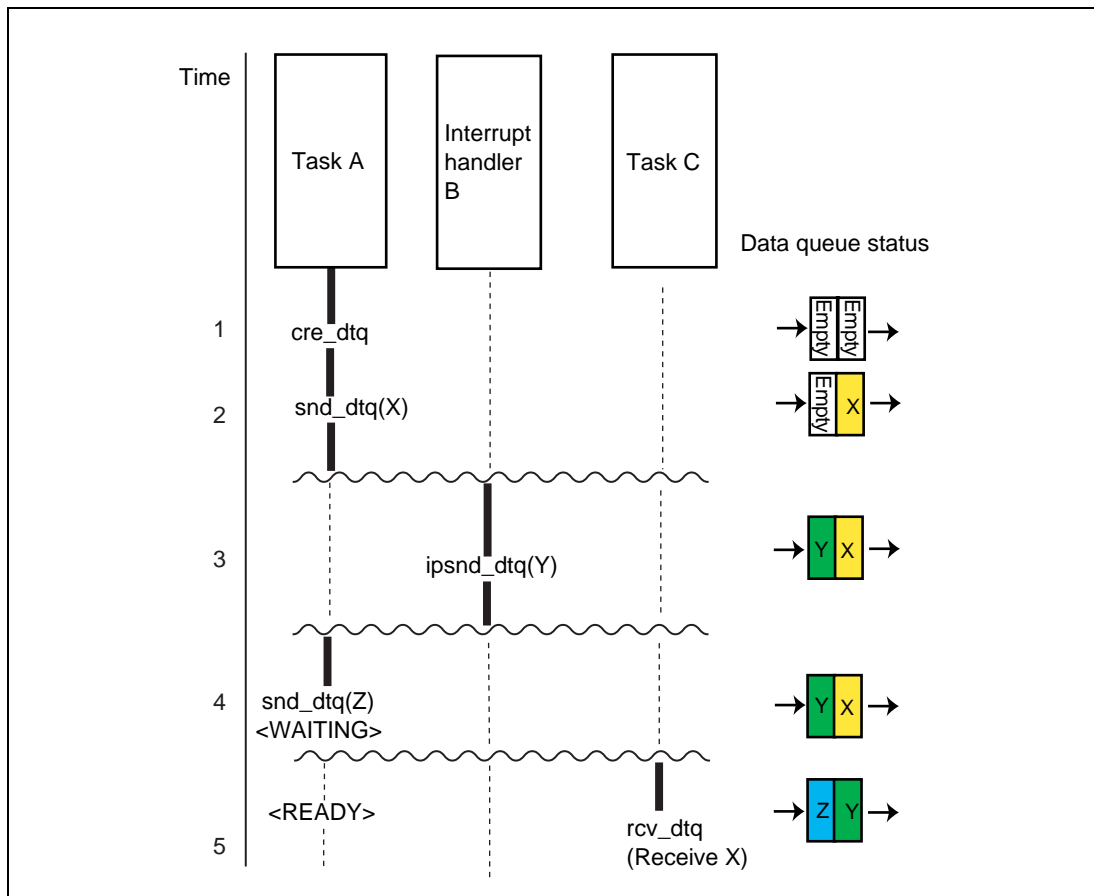


Figure 4.7 Example of Data Queue Usage

Description:

1. Task A issues `cre_dtq` to create a data queue with a size of two words.
2. Task A sends data X by issuing `snd_dtq`. Data X is copied to the data queue and task A continues execution.
3. Interrupt handler B sends data Y by issuing `ipsnd_dtq`.

4. Task A attempts to send data Z. At this time, since there is no free space in the data queue, task A enters the WAITING state.
5. Task C receives data from the data queue by issuing rcv_dtq. Task C gets data X, which is the oldest data that has been sent. At this time, since one entry in the data queue is released, data Z, which task A has attempted to send, is copied to the data queue, and task A is released from the WAITING state.

4.11 Mailbox

A mailbox is an object used to pass messages.

High-speed data communication can be achieved regardless of the message size because communication using a mailbox sends and receives only the start address of a message.

Mailboxes can be created in the following ways.

- Service call cre_mbx or icre_mbx
A mailbox is created with a specified ID.
- Service call acre_mbx or iacre_mbx
A mailbox is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Mailboxes are manipulated through the following service calls.

- Service call del_mbx
Deletes the mailbox with the specified ID.
- Service call snd_mbx or isnd_mbx
Sends a message to a mailbox.
- Service call rcv_mbx or trcv_mbx
Receives a message from a mailbox. When a message cannot be received (the mailbox has no message), the task enters the WAITING state. In trcv_mbx, a timeout period can also be specified.
- Service call prcv_mbx or iprcv_mbx
Receives a message from a mailbox. When a message cannot be received (the mailbox has no message), an error is returned.
- Service call ref_mbx or iref_mbx
Refers to the state of a mailbox.

Figure 4.8 shows an example of mailbox usage.

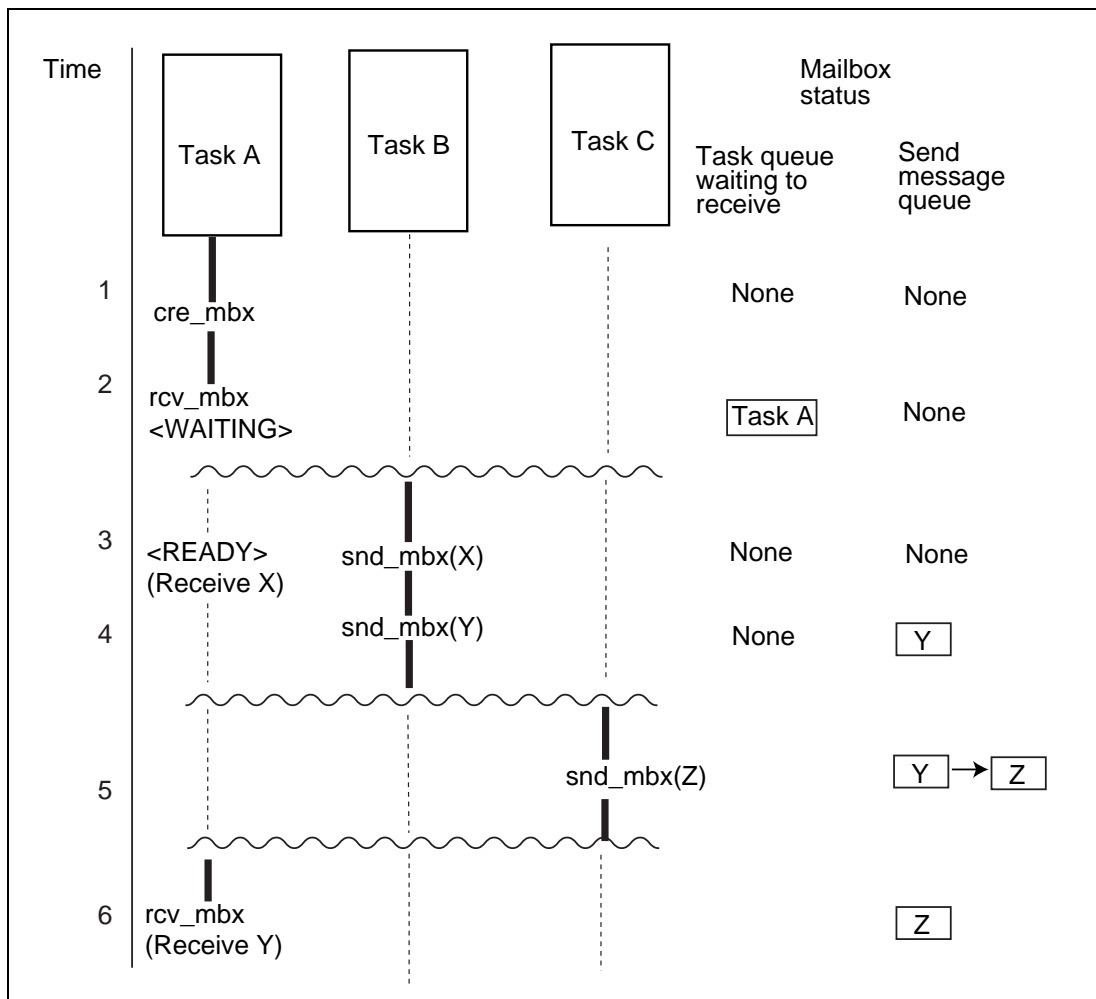


Figure 4.8 Example of Mailbox Usage

Description:

Bold lines represent executed processing. The following describes the mailbox operation with respect to time.

1. Task A issues `cre_mbx` to create a mailbox. The `TA_TFIFO` attribute (tasks waiting to receive are queued in FIFO) and the `TA_MFIFO` attribute (sent messages are queued in FIFO) are specified.
2. Task A attempts to receive a message by issuing `rcv_mbx`. Since no message is stored in the mailbox, task A enters the `WAITING` state.
3. Task B sends message X to the mailbox by issuing `snd_mbx`. At this time, task A is released from the `WAITING` state, and task A receives the address of message X.
4. Task B sends message Y to the mailbox by issuing `snd_mbx`. At this time, since no tasks are waiting for a message, message Y is placed in a message queue.
5. Task C sends message Z to the mailbox by issuing `snd_mbx`. In this case, message Z is placed behind message Y in the message queue because the `TA_MFIFO` attribute has been specified.
6. Task A issues `rcv_mbx`. Task A receives the address of message Y, which is at the head of the message queue.

4.12 Mutex

A mutex is an object used to achieve exclusive control. It differs from a semaphore in the following points.

- A priority ceiling protocol is provided to avoid priority inversion problems.
- A mutex can be used to exclusively control a single resource.

Applications must be programmed so that each task locks a mutex through `loc_mtx`, `tloc_mtx`, or `ploc_mtx` before using a resource, and then unlocks it through `unl_mtx` after completing the use of it.

When a task locks a mutex, the priority of the task is raised to the ceiling priority specified for the mutex. When the task unlocks the mutex, the priority returns to the previous level.

Mutexes can be created in the following ways.

- Service call `cre_mtx`
A mutex is created with a specified ID.
- Service call `acre_mtx`
A mutex is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Mutexes are manipulated through the following service calls.

- Service call `del_mtx`
Deletes the specified mutex.
- Service call `loc_mtx` or `tloc_mtx`
Locks a mutex. When the mutex cannot be locked (another task has already locked it), the task enters the WAITING state. In `tloc_mtx`, a timeout period can also be specified.
- Service call `ploc_mtx`
Locks a mutex. When the mutex cannot be locked (another task has already locked it), an error is returned.
- Service call `unl_mtx`
Unlocks a mutex.
- Service call `ref_mtx`
Refers to the state of a mutex.

Figure 4.9 shows an example of mutex usage.

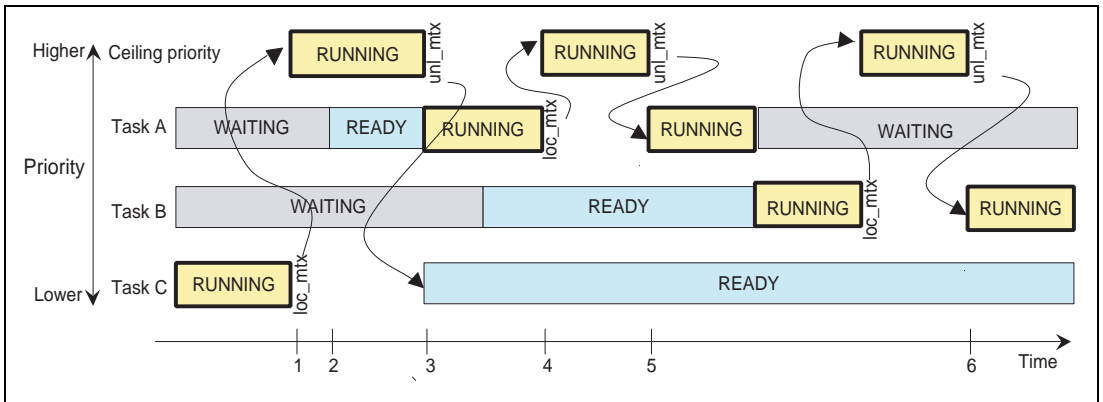


Figure 4.9 Example of Mutex Usage

Description:

1. Task C locks a mutex by issuing loc_mtx. The priority of task C is raised to the ceiling priority specified for the mutex.
2. Task A enters the READY state while task C is being executed at the ceiling priority. Although the priority of task A is higher than that of task C at initial specification, task C now locks a mutex to be executed at the ceiling priority which is higher than task A and task A cannot enter the RUNNING state. In other words, while task C locks a mutex, task C continues execution even if task A whose initial task priority is higher than task C becomes ready.
3. Task C unlocks the mutex by issuing unl_mtx. The priority of task C returns to the initial priority and higher-priority task A enters the RUNNING state.
4. Task A issues loc_mtx to raise its priority to the ceiling priority.
5. Task A issues unl_mtx to return its priority to the initial level.
6. Task B issues loc_mtx to raise its priority to the ceiling priority.
7. Task B issues unl_mtx to return its priority to the initial level.

Base Priority and Current Priority:

A task has two priority levels: base priority and current priority. Tasks are scheduled according to the current priority.

While a task does not lock a mutex, the current priority is always the same as the base priority.

When a task locks a mutex, only the current priority is raised to the ceiling priority specified for the mutex.

When priority-changing service call `chg_pri` or `ichg_pri` is issued, both the base priority and current priority are modified if the specified task has not locked any mutex. When the specified task has locked a mutex, only the base priority is modified. When the specified task has locked a mutex or is waiting to lock a mutex, if a priority higher than the ceiling priority of the locked or waited-for mutex is specified, an `E_ILUSE` error is returned.

The current priority can be checked through service call `get_pri` or `iget_pri`.

4.13 Message Buffer

A message buffer is an object used to pass messages by copying them. The message area becomes available immediately after the message has been sent regardless of whether a task has received the message or not.

A message buffer area is allocated in the resource pool.

Message buffers can be created in the following ways.

- Service call `cre_mbf` or `icre_mbf`
A message buffer is created with a specified ID.
- Service call `acre_mbf` or `iacre_mbf`
A message buffer is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Message buffers are manipulated through the following service calls.

- Service call `del_mbf`
Deletes the specified message buffer.
- Service call `snd_mbf` or `tsnd_mbf`
Sends a message to a message buffer. When the message cannot be sent (the message buffer is full or another task is waiting to send a message to the message buffer), the task enters the WAITING state. In `tsnd_mbf`, a timeout period can also be specified.
- Service call `psnd_mbf` or `ipsnd_mbf`
Sends a message to a message buffer. When the message cannot be sent (the message buffer is full or another task is waiting to send a message to the message buffer), an error is returned.
- Service call `rcv_mbf` or `trcv_mbf`
Receives a message from a message buffer. When the message cannot be received (the message buffer has no message), the task enters the WAITING state. In `trcv_mbf`, a timeout period can also be specified.
- Service call `prcv_mbf`
Receives a message from a message buffer. When the message cannot be received (the message buffer has no message), an error is returned.
- Service call `ref_mbf` or `iref_mbf`
Refers to the state of a message buffer.

Figure 4.10 shows an example of message buffer usage.

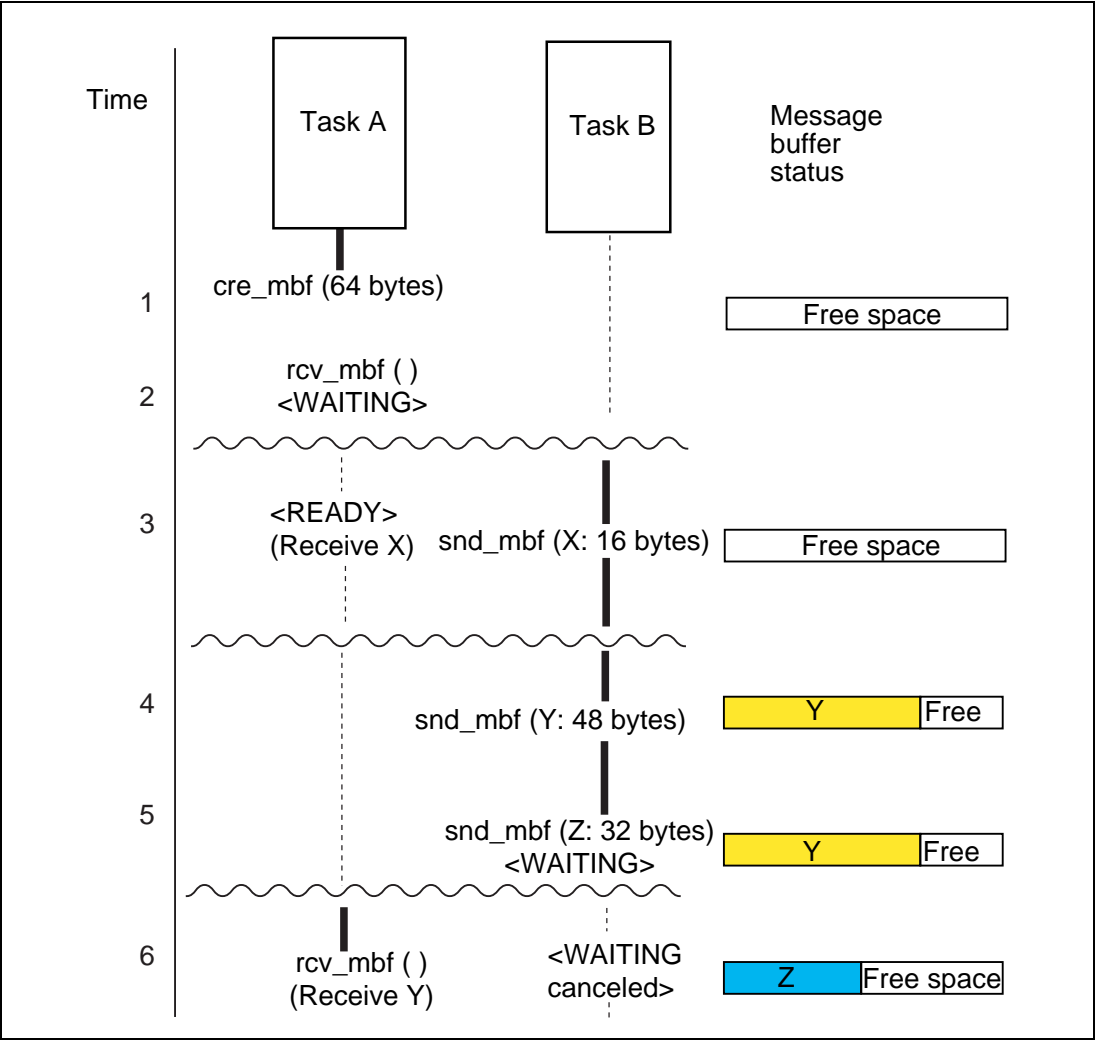


Figure 4.10 Example of Message Buffer Usage

Description:

1. Task A issues `cre_mbf` to create a 64-byte message buffer, where the maximum size of a message is 48 bytes.
2. Task A prepares a 48-byte memory area and issues `rcv_mbf` to receive a message. Task A is placed in the WAITING state since there are no messages in the message buffer.
3. Task B sends 16-byte message X by issuing `snd_mbf`. Task A is released from the WAITING state and message X is copied to the memory prepared by task A. Task A gets the received message size (16) as the return parameter.
4. Task B sends 48-byte message Y by issuing `snd_mbf`. Since there are no tasks waiting for a message, message Y is copied to the message buffer. Note that the kernel uses a 4-byte message buffer area when copying the message, but this is not shown in the figure.
5. Task B attempts to send 32-byte message Z by issuing `snd_mbf`. Since the message buffer does not have enough free space to store message Z, task B is placed in the WAITING state.
6. Task A prepares a 48-byte memory area and issues `rcv_mbf`. 48-byte message Y stored in the message buffer is copied to the memory prepared by task A. Task A gets the received message size (48) as the return parameter. At this time, since the message buffer has sufficient space to store message Z, task B is released from the WAITING state and message Z is copied to the message buffer.

4.14 Fixed-Size Memory Pool

A fixed-size memory pool is an object used to dynamically allocate a fixed-size memory area. Since the size of the memory pool is fixed, operation is faster than with a variable-size memory pool.

Fixed-size memory pools can be created in the following ways.

- Service call `cre_mpf` or `icre_mpf`
A fixed-size memory pool is created with a specified ID.
- Service call `acre_mpf` or `iacre_mpf`
A fixed-size memory pool is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

The address of a fixed-size memory pool area can be specified at task creation, or can be specified to be automatically assigned by the kernel. When a fixed-size memory pool is created through the configurator, its address cannot be specified.

When the kernel automatically allocates a memory pool, it is allocated in the system pool. When the memory object protection function is selected, the memory pool is handled as a memory object. An access permission vector can be specified for a memory pool when the memory pool is created by the configurator. For the memory pool created through a service call, an appropriate access permission vector is automatically assigned so that only the domain of the program that created the memory pool can read or write to the memory pool.

Fixed-size memory pools are manipulated through the following service calls.

- Service call `del_mpf`
Deletes the specified fixed-size memory pool.
- Service call `get_mpf` or `tget_mpf`
Acquires a fixed-size memory block. When the memory block cannot be acquired (no memory block is available in the memory pool), the task enters the WAITING state. In `tget_mpf`, a timeout period can also be specified.
- Service call `pget_mpf` or `ipget_mpf`
Acquires a fixed-size memory block. When the memory block cannot be acquired (no memory block is available in the memory pool), an error is returned.
- Service call `rel_mpf` or `irel_mpf`
Returns a fixed-size memory block.
- Service call `ref_mpf` or `iref_mpf`
Refers to the state of a fixed-size memory pool.

Figure 4.11 shows an example of fixed-size memory pool usage.

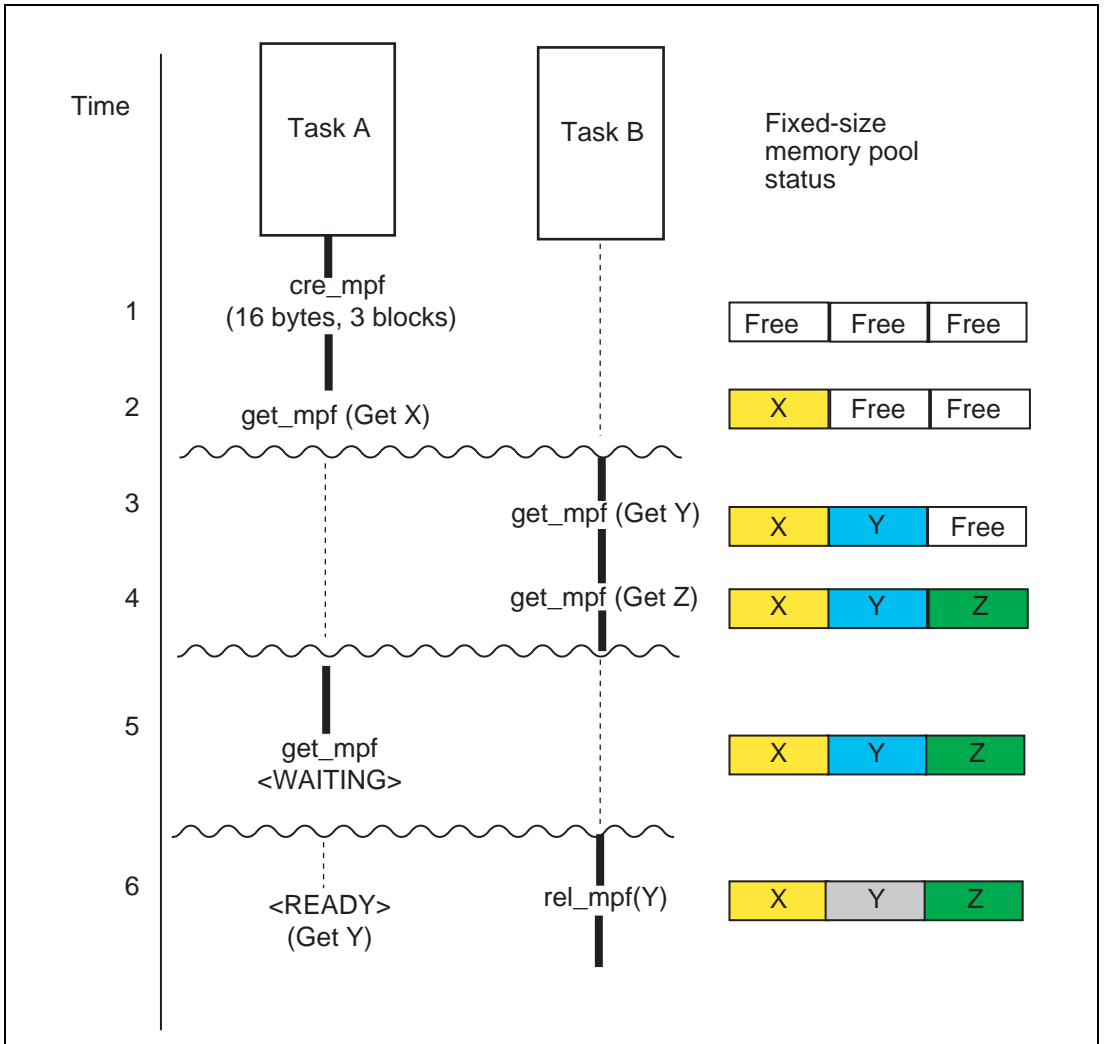


Figure 4.11 Example of Fixed-Size Memory Pool Usage

Description:

1. Task A issues `cre_mpf` to create a fixed-size memory pool that has three 16-byte memory blocks.
2. Task A gets block X by issuing `get_mpf`.
3. Task B gets block Y by issuing `get_mpf`.
4. Task B gets block Z by issuing `get_mpf`.

5. Task A attempts to get a block by issuing `get_mpf`. At this time, no memory blocks are available and task A enters the WAITING state.
6. Task B returns block Y by issuing `rel_mpf`. At this time, task A is released from the WAITING state and the returned block Y is allocated to task A.

4.15 Variable-Size Memory Pool

A variable-size memory pool is an object used to dynamically allocate a memory area with the size specified by the user.

Variable-size memory pools can be created in the following ways.

- Service call `cre_mpl` or `icre_mpl`
A variable-size memory pool is created with a specified ID.
- Service call `acre_mpl` or `iacre_mpl`
A variable-size memory pool is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

The address of a variable-size memory pool area can be specified at task creation, or can be specified to be automatically assigned by the kernel. When a variable-size memory pool is created through the configurator, its address cannot be specified.

When the kernel automatically allocates a memory pool, it is allocated in the system pool. When the memory object protection function is selected, the memory pool is handled as a memory object. An access permission vector can be specified for a memory pool when the memory pool is created by the configurator. For the memory pool created through a service call, an appropriate access permission vector is automatically assigned so that only the domain of the program that created the memory pool can read or write to the memory pool.

Variable-size memory pools are manipulated through the following service calls.

- Service call `del_mpl`
Deletes the specified variable-size memory pool.
- Service call `get_mpl` or `tget_mpl`
Acquires a variable-size memory block with a specified size. When the memory block cannot be acquired (no memory block is available in the memory pool or another task is waiting to acquire a memory block), the task enters the WAITING state. In `tget_mpl`, a timeout period can also be specified.

- Service call `pget_mpl` or `ipget_mpl`
Acquires a variable-size memory block. When the memory block cannot be acquired (no memory block is available in the memory pool or another task is waiting to acquire a memory block), an error is returned.
- Service call `rel_mpl` or `irel_mpl`
Returns a variable-size memory block.
- Service call `ref_mpl` or `iref_mpl`
Refers to the state of a variable-size memory pool.

Figure 4.12 shows an example of variable-size memory pool usage.

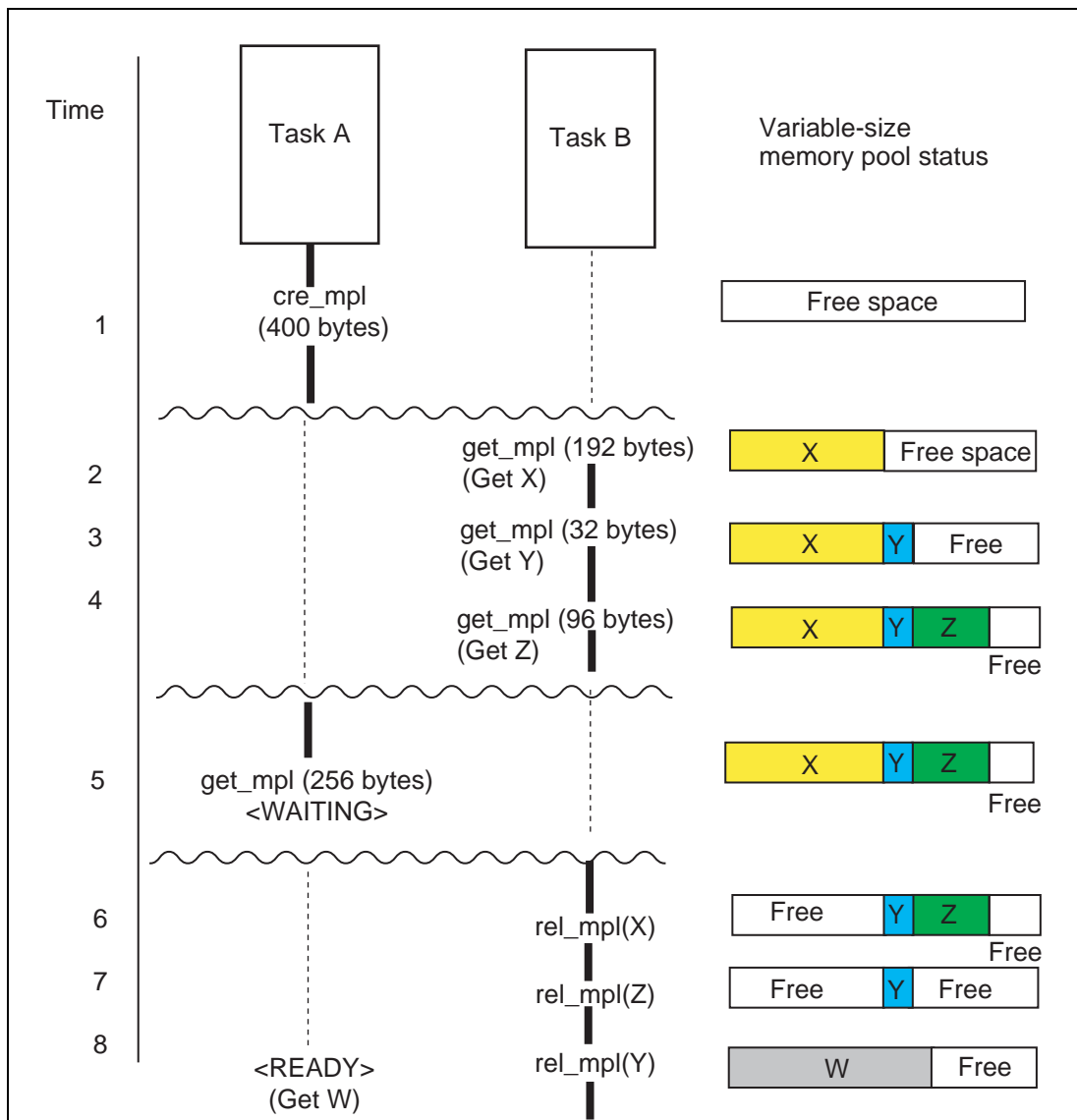


Figure 4.12 Example of Variable-Size Memory Pool Usage

Description:

1. Task A creates a 400-byte variable-size memory pool by issuing `cre_mpl`.
2. Task B acquires 192-byte memory block X by issuing `get_mpl`. Note that the kernel uses 16 bytes in the memory pool to allocate a block, but this is not shown in the figure.
3. Task B also acquires 32-byte memory block Y by issuing `get_mpl`.
4. Task B also acquires 96-byte memory block Z by issuing `get_mpl`.
5. Task A attempts to acquire a 256-byte memory block by issuing `get_mpl`. However, the available memory block is insufficient to assign a 256-byte memory block to task A, so task A enters the WAITING state.
6. Task B returns 192-byte memory block X by issuing `rel_mpl`. Since there is not 256 bytes of contiguous memory in the memory pool, task A remains in the WAITING state.
7. Task B returns 96-byte memory block Z by issuing `rel_mpl`. At this time, the total size of available memory blocks is more than 256 bytes, however there is not 256 bytes of contiguous memory in the memory pool, so task A remains in the WAITING state.
8. Task B returns 32-byte memory block Y by issuing `rel_mpl`. Since there is more than 256 bytes of contiguous memory in the memory pool, task A is released from the WAITING state and 256-byte memory block W is assigned to task A.

4.15.1 Fragmentation

Repeated acquisition and return of memory blocks from a variable-size memory pool causes "fragmentation" of the available memory area. When the memory area is fragmented, even if there is enough total space to acquire a required memory block, it cannot be acquired if the area is not contiguous.

The HI7300/PX provides a function for reducing fragmentation; the `VTA_UNFRAGMENT` attribute should be specified at creation of a variable-size memory pool to reduce fragmentation.

For details, refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

4.16 Time Management

The kernel provides the following functions related to time management.

- Reference to and setting of system clock
- Time event handler (cyclic handler, alarm handler, and overrun handler) execution control
- Task execution control such as timeout and dly_tsk

The unit of time used for setting time parameters is 1 ms.

The kernel uses the system clock (a 48-bit counter value) to implement the time management functions.

4.16.1 Time Precision

The unit of time used for setting time parameters, such as a timeout period, is 1 ms, but the precision of time is TIC_NUME/TIC_DENO [ms]. With this precision, the system clock is updated and time management is performed.

A time event (timeout occurrence or cyclic handler initiation) is generated after the specified time has passed.

Figure 4.13 shows examples of tslp_tsk(5) execution when the actual time is 9.2 ms.

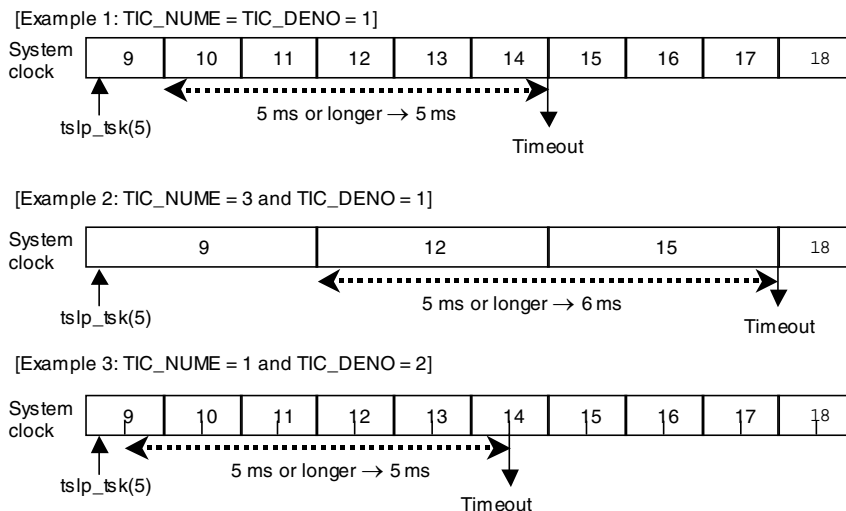


Figure 4.13 Time Precision

4.16.2 System Clock Setting and Reference

The current system clock can be checked through service call `get_tim` or `iget_tim`.

Through service call `set_tim` or `iset_tim`, the system clock can be modified to a specified value. Note that even after the system clock is modified, the actual time until the occurrence of an event (such as timeout) that has already been monitored will not change.

4.16.3 Cyclic Handler

A cyclic handler is a time event handler that is initiated at a specified interval after the specified initiation phase has been passed.

A cyclic handler is handled as the non-task context.

A cyclic handler is assigned to the kernel domain.

Cyclic handlers can be created in the following ways.

- Service call `cre_cyc` or `icre_cyc`
A cyclic handler is created with a specified ID. The created cyclic handler can also be started by specifying the `TA_STA` attribute.

- Service call `acre_cyc` or `iacre_cyc`
A cyclic handler is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Cyclic handlers are manipulated through the following service calls.

- Service call `del_cyc`
Deletes a cyclic handler.
- Service call `sta_cyc` or `ista_cyc`
Starts operation of a cyclic handler.
- Service call `stp_cyc` or `istp_cyc`
Stops operation of a cyclic handler.
- Service call `ref_cyc` or `iref_cyc`
Refers to the state of a cyclic handler.

There are two methods to initiate execution of a cyclic handler: storing the initiation phase, and not storing the initiation phase. When storing the initiation phase, execution of a cyclic handler is initiated based on the timing when the cyclic handler is created. When not storing the initiation phase, execution of a cyclic handler is initiated based on the timing when operation of the cyclic handler is started. Whether to store the initiation phase can be specified at creation of a cyclic handler.

Figure 4.14 shows an example of cyclic handler usage.

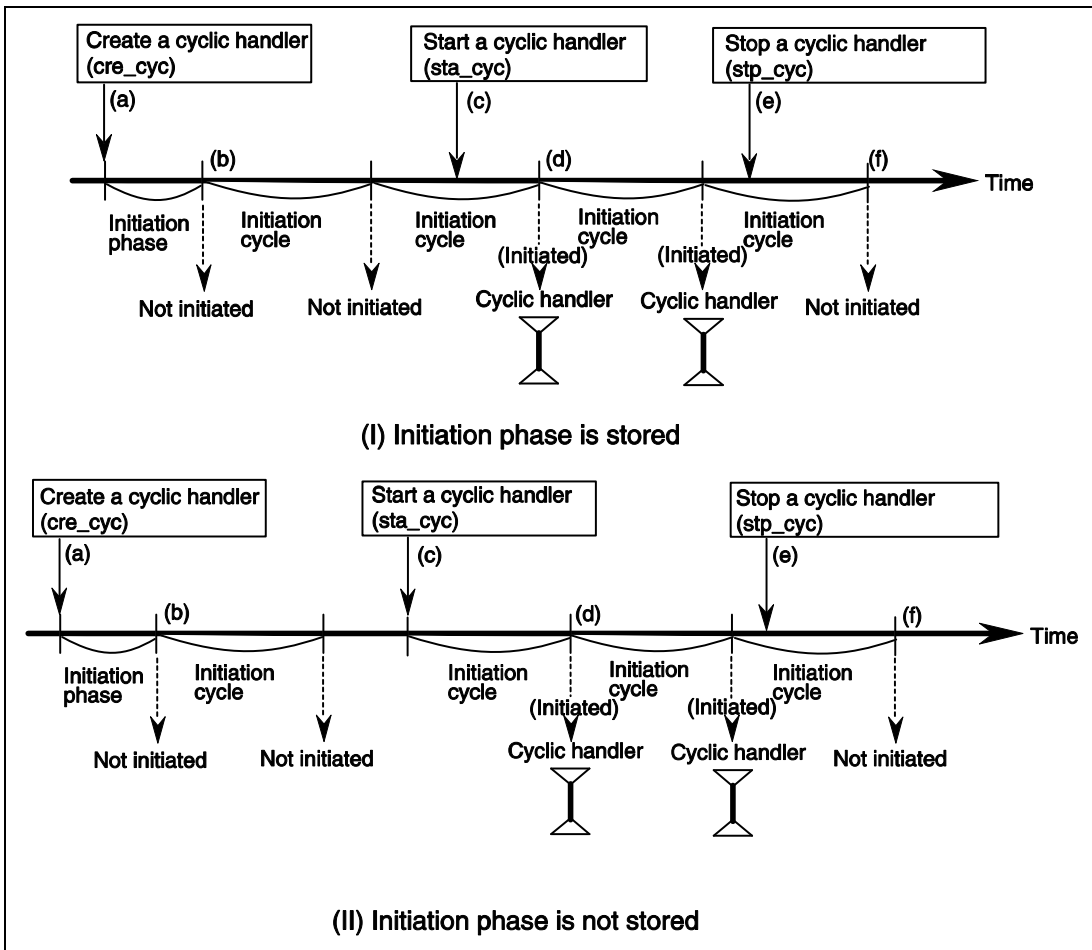


Figure 4.14 Example of Cyclic Handler Usage

Description:

- (a) A cyclic handler (without TA_STA attribute specification) is created.
- (b) The cyclic handler is not initiated after the initiation phase and cycle time have passed because the cyclic handler operation has not been started.
- (c) The cyclic handler operation is started by issuing the sta_cyc service call.
- (d) When the initiation phase is stored as shown in (I) in the figure, the cyclic handler is initiated based on the initiation cycle after the cyclic handler has been created. When the initiation phase is not stored as shown in (II) in the figure, the cyclic handler is initiated based on the initiation cycle after the sta_cyc service call has been issued.

- (e) The cyclic handler is stopped by issuing the `stp_cyc` service call.
- (f) The cyclic handler is not initiated after the cycle time has passed because the cyclic handler operation has been stopped.

4.16.4 Alarm Handler

An alarm handler is a time event handler that is initiated only once when the specified time has been reached.

An alarm handler is handled as the non-task context.

An alarm handler is assigned to the kernel domain.

Alarm handlers can be created in the following ways.

- Service call `cre_alm` or `icre_alm`
An alarm handler is created with a specified ID. The created alarm handler can also be started by specifying the `TA_STA` attribute.
- Service call `acre_alm` or `iacre_alm`
An alarm handler is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Alarm handlers are manipulated through the following service calls.

- Service call `del_alm`
Deletes an alarm handler.
- Service call `sta_alm` or `ista_alm`
Starts operation of an alarm handler.
- Service call `stp_alm` or `istp_alm`
Stops operation of an alarm handler.
- Service call `ref_alm` or `iref_alm`
Refers to the state of an alarm handler.

Figure 4.15 shows an example of alarm handler usage.

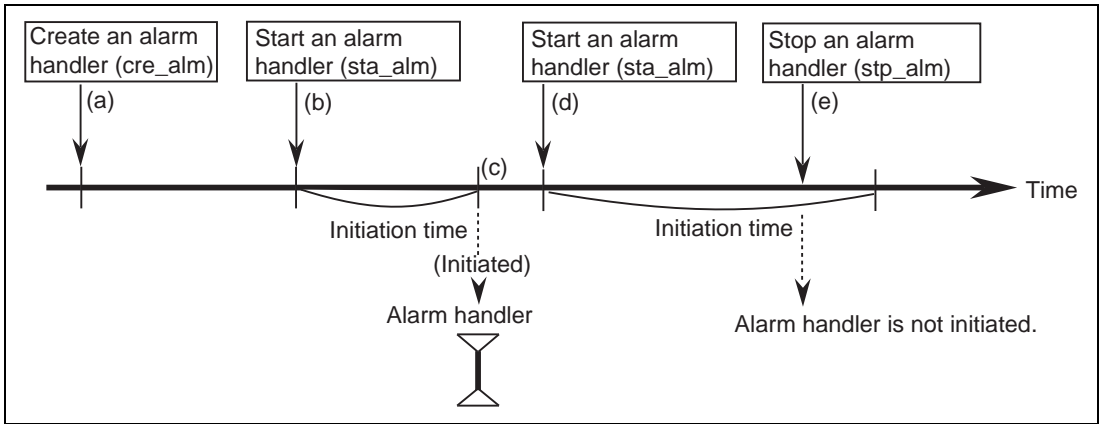


Figure 4.15 Example of Alarm Handler Usage

Description:

- (a) An alarm handler is created.
- (b) The alarm handler operation is started by issuing the sta_alm service call.
- (c) The alarm handler is initiated after the specified initiation time has passed.
- (d) If the sta_alm service call is issued by specifying another initiation time, the alarm handler starts operation again.
- (e) Since the stp_alm service call is issued before the initiation time has passed, the alarm handler is not initiated.

4.16.5 Overrun Handler

An overrun handler is a time event handler that is initiated when a task has used the processor over the time limit specified for the task. Only one overrun handler can be defined in a single system.

An overrun handler is handled as the non-task context.

An overrun handler is assigned to the kernel domain.

Overrun handlers can be created in the following ways.

- Service call `def_ovr`
Defines an overrun handler.
- Defined by the configurator

Overrun handlers are manipulated through the following service calls.

- Service call `sta_ovr` or `ista_ovr`
Specifies an overrun time for a task and starts monitoring whether the task overruns.
- Service call `stp_ovr` or `istp_ovr`
Stops monitoring whether the task overruns.
- Service call `ref_ovr` or `iref_ovr`
Refers to the state of an overrun handler.

Figure 4.16 shows an example of overrun handler usage.

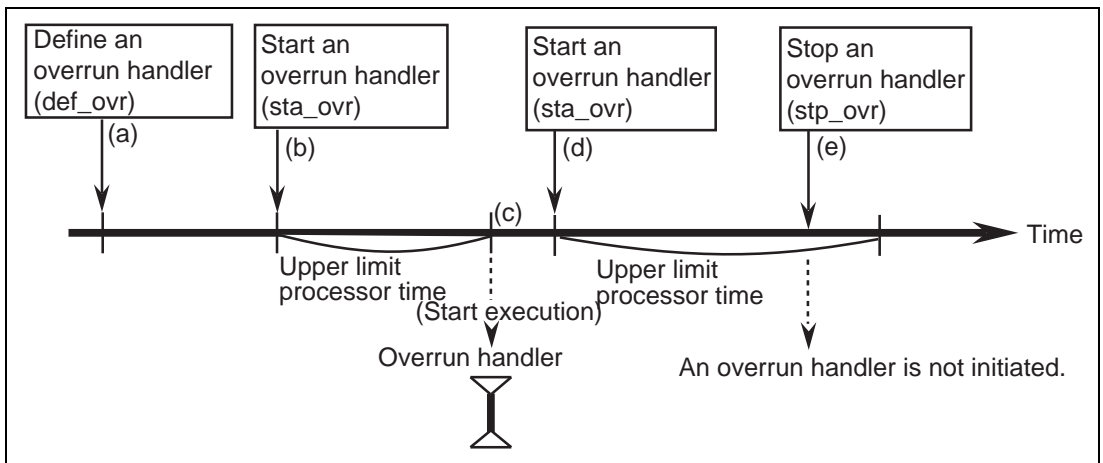


Figure 4.16 Example of Overrun Handler Usage

Description:

- (a) An overrun handler is defined.
- (b) The upper limit processor time for the task is specified by the `sta_ovr` service call. The operation of the overrun handler is started at this point.
- (c) If the total processor time used by the task exceeds the upper limit processor time, the overrun handler is initiated.
- (d) If the upper limit processor time is changed by the `sta_ovr` service call, the operation of the overrun handler is started again.
- (e) If the `stp_ovr` service call is issued before the total processor time has exceeded the upper limit processor time, the overrun handler is stopped. In this case, the overrun handler is not initiated even if the upper limit processor time is exceeded after that.

4.16.6 Timer Driver

A timer driver is required to enable the time-management functions. Timer drivers are of two types: a standard timer driver or an optimized timer driver. Either type should be selected through `CFG_OPTTMR` in the configurator.

The standard timer driver generates timer interrupts at the intervals specified by `TIC_NUME/TIC_DENO [ms]`, and must be created and embedded in the kernel by the user. The HI7300/PX provides sample drivers for the timers built into various microcomputers.

The optimized timer driver is provided in the kernel. It can reduce the frequency of timer interrupts, but the applicable hardware is limited.

Reference: Creating standard timer driver → Section 9, Standard Timer Driver
Optimized timer driver → Section 4.17, Optimized Timer Driver

4.16.7 Notes on Time Management

The kernel performs the following processing when a timer interrupt occurs.

- (a) Updates the system clock.
- (b) Initiates and executes alarm handlers.
- (c) Initiates and executes cyclic handlers.
- (d) Initiates and executes the overrun handler.
- (e) Performs task timeout processing specified by service calls with the timeout function such as `tslp_tsk`.

These processes are all performed with the CFG_KNLLVL or lower interrupt levels masked. Among these processes, (b), (c), and (e) may overlap for multiple tasks and handlers. In that case, the processing time of the kernel becomes very long and results in the following defects.

- Delay of the response to interrupts
- Delay of the system clock

To avoid these problems, the following steps must be taken.

- The time for time event handler processing must be as short as possible.
- The time event handler cycles and the timeout values specified by timeout service calls must be set to as large values as possible. For an extreme example, if the cycle time of a cyclic handler is 1 ms and the handler's processing takes longer than 1 ms, that cyclic handler will be executed forever; and the system will hang.

4.17 Optimized Timer Driver

4.17.1 Overview

The standard timer driver generates timer interrupts in the same cycle as the time precision for service calls (CFG_TICNUME/CFG_TICDENO [ms]). When the optimized timer driver is used, the frequency of interrupts can be reduced while the time precision for service calls is maintained.

- The frequency of timer-interrupt generation during the sleep mode of the CPU is reduced; this leads to improved power consumption.
- Reducing the frequency of timer interrupts lowers the percentage of CPU time taken up by timer-interrupt processing and improves the throughput of the system. Alternatively, the CPU may be placed in the power-down mode for a greater part of the time.

Unlike the standard timer driver, the optimized timer driver is developed for the TMU in specific microcomputers and built into the kernel. The user cannot create an optimized timer driver.

4.17.2 Operation

Figure 4.17 shows examples of operation where the standard timer driver and optimized timer driver are used to provide time precision of 1 ms (CFG_TICNUME/CFG_TICDENO).

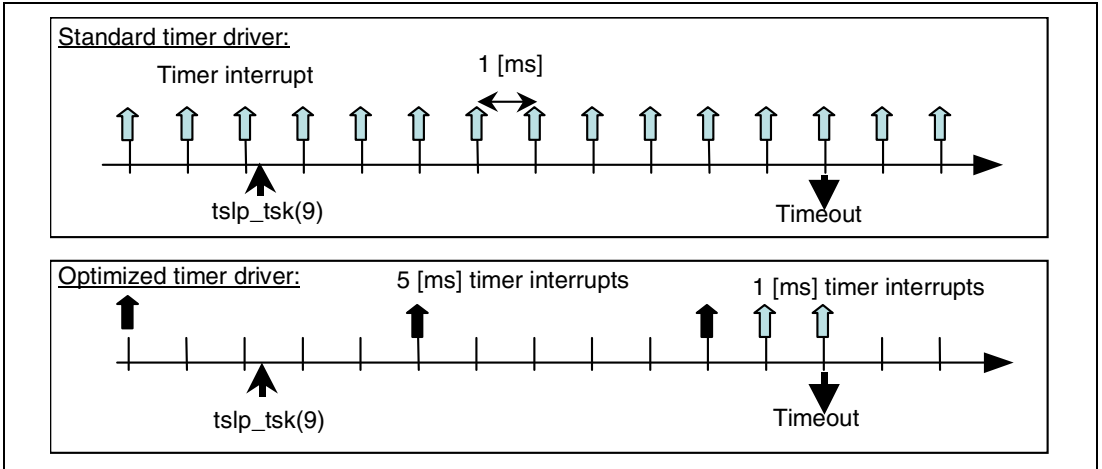


Figure 4.17 Example of Operation

Two TMU timer channels, one with a 1-ms cycle and the other with a 5-ms cycle, are used for the optimized timer driver as shown in figure 4.17; the respective timing cycles are called the high-precision cycle and low-precision cycle. The high-precision cycle is the result, in ms, of the division of the values specified by the configurator, i.e., CFG_TICNUME/CFG_TICDENO. The period of the low-precision cycle is an integer multiple of the high-precision period and is specified through CFG_LONGTICRATE in the configurator.

When the optimized timer driver is in use, the kernel investigates the following situations at the right time.

- Waiting tasks by service calls with timeout (txxx_yyy)
- Waiting tasks by dly_tsk service call
- Cyclic handlers
- Alarm handlers

The kernel uses the investigation results to determine whether or not interrupts at high-precision cycles are needed, and accordingly enables or disables high-precision interrupts.

Interrupts at low-precision cycles are always enabled.

Although not shown in figure 4.17, another TMU channel is used in monitoring for the overrun handler. This timer interrupt is only generated when a task has reached the upper limit on the allowed processor time.

Figure 4.18 shows two effects of using the optimized timer driver. This function reduces the frequency of timer interrupts, leading to the following advantages over the standard timer driver.

- Quicker transitions to the sleep mode (lower amount of CPU time consumed in timer-interrupt processing)
- Less frequent cancellation of the sleep mode for the processing of timer interrupts

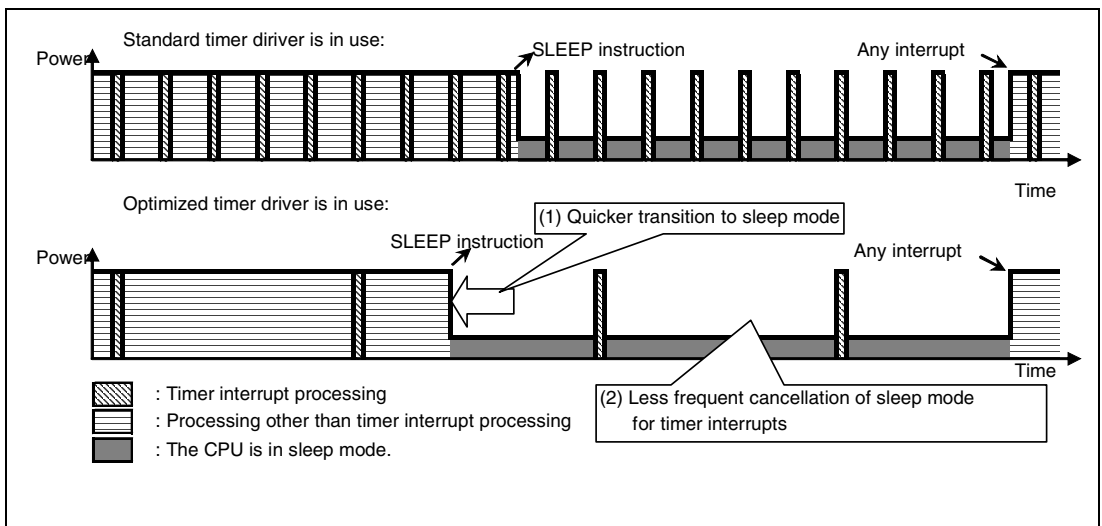


Figure 4.18 Schematic Illustration of the Optimized Timer Driver's Effects

4.17.3 Applicable Microcomputers

The optimized timer driver uses the TMU built into the microcomputer. However, not all microcomputers including TMU are applicable to the optimized timer driver. As of publication of this manual, the optimized timer driver can only be run with the following microcomputers:

SH73180 and SH73230

4.17.4 Hardware Initialization

The following processing is carried out to initialize the optimized timer driver when the kernel is started up.

- (1) Cancels the module stop state of TMU.
- (2) Initializes channels 0, 1, and 2.
- (3) Sets the interrupt levels of channels 0, 1, and 2 for the interrupt controller.

However, when the `def_ovr` service call has not been selected in the configurator, settings for channel 2 in the above (2) and (3) are not made.

Note that the optimized timer driver does not place the TMU in the module stop state.

4.18 Interrupt Management

4.18.1 Interrupt Handler

An interrupt can be requested through an external interrupt pin or from a peripheral module. When an interrupt occurs, the corresponding interrupt handler is executed via the kernel.

Interrupts are distinguished by the interrupt handler number. Each interrupt handler number corresponds to the `INTEVT` code of the CPU.

An interrupt handler is handled as the non-task context.

An interrupt handler is assigned to the kernel domain.

Interrupt handlers can be defined in the following ways.

- Service call `def_inh` or `idef_inh`
Defines an interrupt handler for the specified interrupt handler number.
- Defined by the configurator

When an interrupt for which no interrupt handler has been defined occurs, the system goes down.

Reference: Section 8.4, Interrupt Handlers

4.18.2 Kernel Level (CFG_KNLLVL)

The kernel level (CFG_KNLLVL) specifies the level of interrupts to be masked during kernel execution, and it is also used as the timer interrupt level. The kernel level can be specified by the configurator.

The kernel executes the critical sections and timer interrupt processing by masking the interrupts having levels equal to or lower than the CFG_KNLLVL level.

When the memory object protection function is selected, the kernel executes TLB-related exception processing by masking all interrupts (SR.BL = 1).

Interrupts having levels higher than the kernel level are immediately accepted even during critical section execution in the kernel. Note that the handlers for interrupts having levels higher than the kernel level are not allowed to issue a service call.

When the IMASK bit value in SR is set higher than the kernel level (an interrupt handler having a level higher than the kernel level always satisfies this condition), service calls must not be issued. If a service call is issued under this condition, the interrupt mask level in the kernel will be unexpectedly lowered.

4.18.3 Disabling Interrupts

Interrupts can be disabled in the following ways, regardless of the causes.

- Modify the IMASK bits in SR.
- Set the BL bit in SR to 1.

(1) Modifying the IMASK Bits in SR

Modifying the IMASK bits in SR can disable interrupts at the specified level or lower levels.

Note the following when modifying the IMASK bits.

- When the IMASK value in SR indicates a level higher than the kernel level (CFG_KNLLVL), service calls must not be issued. If this is attempted, correct system operation is not guaranteed.
- In an interrupt handler, the IMASK value must not be modified to lower than the interrupt level of that handler. During other non-task context execution, the IMASK value must not be modified to lower than the level at initiation.

The IMASK bits in SR can be modified in the following ways.

(a) Service call `loc_cpu` or `iloc_cpu`

Each call shifts the system to the CPU-locked state, in which the IMASK value in SR is modified to the kernel level (CFG_KNLLVL).

To cancel the CPU-locked state, issue `unl_cpu` or `iunl_cpu`.

Note the following when using service call `loc_cpu` or `iloc_cpu`.

- When the transition to the CPU-locked state is made in the non-task context, the CPU-locked state must be canceled within the handler.
- The service calls available in the CPU-locked state are limited.

Reference: Service calls available in the CPU-locked state → Section 6.4.3, CPU-Locked State

(b) Service call `chg_ims` or `ichg_ims`

Each call modifies the IMASK bits in SR to the specified value and interrupts having a level equal to or lower than the specified level are masked.

To cancel the mask, issue `chg_ims` to restore the IMASK value in SR to the value that the IMASK bits previously held.

Note the following when using service call `chg_ims` or `ichg_ims`.

- This method cannot modify the IMASK bits to a level higher than the kernel level (CFG_KNLLVL). If an attempt is made to specify a higher level through service call `chg_ims`, an E_PAR error is returned. To modify the IMASK bits to a level higher than the kernel level, use the method described in (c) Modifying the IMASK bits in SR without a service call.
- The system is in the dispatch-disabled state while the IMASK bits are set to a non-zero value by using this method in a task context.
- If service call `chg_ims` or `ichg_ims` is issued in the CPU-locked state, an E_CTX error is returned.

(c) Modifying the IMASK bits in SR without a service call

The IMASK bits in SR can be modified by the LDC instruction and interrupts having a level equal to or lower than the specified level are masked. In C language, intrinsic function `set_ims()` or `set_cr()` supported by the compiler should be used for this purpose.

To cancel the mask, restore the IMASK value in SR to the value that the IMASK bits previously held.

Note the following when modifying the IMASK bits through this method.

- In a task context, the IMASK bits are only modified to a level higher than the kernel level (CFG_KNLLVL) or 0. If another value is specified through this method, correct operation is not guaranteed.

- In the CPU-locked state, the IMASK bits must not be modified to a lower value than CFG_KNLLVL. If this is attempted, correct system operation is not guaranteed.
- LDC is a privileged instruction, and so this method cannot be used in a user domain.

(2) Modifying the BL Bit in SR

All interrupts are disabled through this method.

The BL bit in SR can be set to 1 through the LDC instruction. In C language, intrinsic function `set_cr()` supported by the compiler should be used for this purpose.

To cancel the mask, restore the BL value in SR to 0.

Note the following when modifying the BL bit through this method.

- If a CPU exception occurs while the BL bit is 1, the CPU is reset. To avoid this reset, applications must be programmed so that no CPU exception occurs in this state. Note especially that when the memory object protection function is selected, access to the MMU mapped area may generate a TLB miss exception except when the target address is locked through service call `vloc_tlb`, and so the MMU mapped area must not be accessed. In addition, a service call must not be issued for the same reason.
- LDC is a privileged instruction, and so this method cannot be used in a user domain.

Reference: Section 8.4.4, Notes on NMI

4.19 CPU Exception

A CPU exception occurs during program execution. When a CPU exception occurs, the corresponding CPU exception handler is executed via the kernel.

CPU exceptions are distinguished by the CPU exception handler number. Each CPU exception handler number corresponds to the EXPEVT code of the CPU.

A CPU exception handler is handled as the non-task context.

A CPU exception handler is assigned to the kernel domain.

CPU exception handlers can be defined in the following ways.

- Service call `def_exc` or `iddef_exc`
Defines a CPU exception handler for the specified CPU exception handler number.
- Defined by the configurator

When a CPU exception for which no CPU exception handler has been defined occurs, the system goes down.

A packet is sent to a CPU exception handler to pass the necessary information, such as the register values stored when the CPU exception occurred. To extract information from this packet, the following macros are provided.

- **VSNS_CTX**: Returns information as to whether execution was in the non-task context when the CPU exception occurred.
- **VSNS_LOC**: Returns information as to whether the system was in the CPU-locked state when the CPU exception occurred.
- **VSNS_DSP**: Returns information as to whether the system was in the dispatch-disabled state when the CPU exception occurred.
- **VSNS_DPN**: Returns information as to whether the system was in the dispatch-pended state when the CPU exception occurred.
- **VSNS_TEX**: Returns information as to whether the task exception was enabled for the task if execution was in a task context when the CPU exception occurred.
- **VGET_TID**: Returns the ID of the task that was running when the CPU exception occurred.
- **VGET_DID**: Returns the ID of the domain of the task that was running when the CPU exception occurred.

The state to be restored after a return from the CPU exception processing can be changed by modifying this packet.

Reference: Section 8.8, CPU Exception Handler

4.20 Extended Service Call and Trap

Extended service calls and traps are used to call programs (extended service call routines and trap routines) assigned to the kernel domain.

Both can be issued after corresponding user-created programs are defined in the kernel.

Reference: Section 8.3, Extended Service Call Routines and Trap Routines

The context type of an extended service call routine or a trap routine is the same as that of the caller of the routines.

Both routines are assigned to the kernel domain.

4.20.1 Extended Service Call

Extended service calls are distinguished by the function code. An extended service call routine should be defined for each function code. When service call `cal_svc` or `ical_svc` is issued, the extended service call routine corresponding to the function code specified in the call is executed.

Extended service calls are defined in the following ways.

- Service call `def_svc` or `idef_svc`
Defines an extended service call for the specified function.
- Defined by the configurator

4.20.2 Trap

Traps are distinguished by the trap number.

Trap numbers 0 to 15 are reserved for kernel use; trap numbers of 16 or above can be used in applications. When a TRAPA instruction is executed, the trap routine corresponding to the specified trap number is called.

Trap routines are defined in the following ways.

- Service call `vdef_trp` or `ivdef_trp`
Defines a trap for the specified number.
- Defined by the configurator

4.21 Memory Object Protection Function

4.21.1 Overview

The memory object protection function controls "which program can perform which type of access to which memory". Through this control, the following functions are implemented.

- Detect unauthorized access
- Check errors in address parameters passed through kernel service calls

By prohibiting unauthorized memory access through these functions, debugging efficiency is improved and the system's memory contents are protected against unexpected illegal memory access that might be done after shipment.

"Which program" in the above description means "which user domain ID". The programs in the kernel domain are not controlled by the memory object protection function, and these programs can always access any memory.

A task is assigned to a user domain or the kernel domain. A handler is assigned to the kernel domain.

"Which type of access" means "read access (including instruction fetch)" or "write access", and "which memory" means "which memory object". The kernel controls this information for each memory object. This management information is called an access permission vector.

For a memory object, the following attributes can be specified.

- Read-only or readable/writable
- Cacheable or non-cacheable, and write mode (copy-back or write-through) when cacheable

The kernel uses the memory management unit (MMU) built into the processor to implement this function.

To enable the memory object protection function, select CFG_PROTMEM.

Figure 4.19 gives an overview of the memory object protection function.

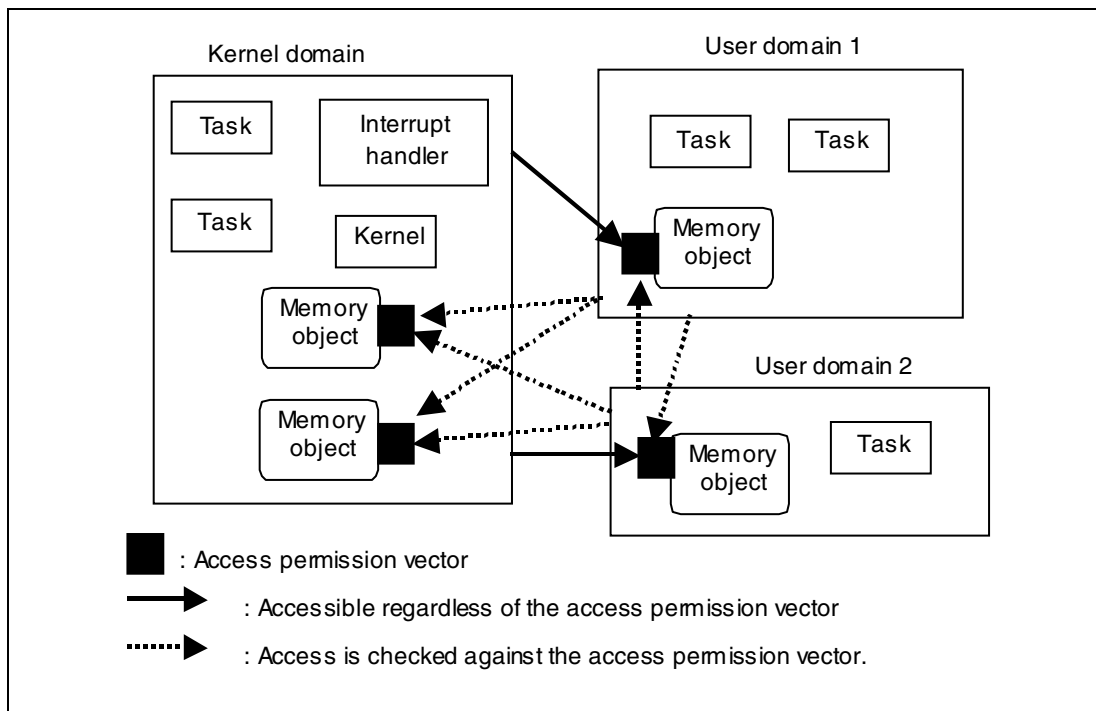


Figure 4.19 Overview of Memory Object Protection Function

4.21.2 Memory Object Types

Table 4.2 lists the memory object types.

Table 4.2 Memory Object Types

Memory Object		Creation
Memory object allocated in the system pool	Stack for the task in a user domain, which is allocated in the system pool by the kernel	Created at task creation
	Fixed-size memory pool area allocated in the system pool by the kernel	Created at fixed-size memory pool creation
	Variable-size memory pool area allocated in the system pool by the kernel	Created at variable-size memory pool creation
Protected memory block acquired from a protected memory pool		Created at protected memory block acquisition
Protected memory block received from a protected mailbox		
Static memory object defined by the configurator		—

4.21.3 Attribute and Domain

Each memory object has the following attributes supported by this kernel.

- (1) Read-only (TA_RO) or readable/writable (TA_RW)

When a memory object has the TA_RO attribute, an exception will be detected if the memory object is written to, regardless of the access permission vector to be described later.

- (2) Cacheable (TA_CACHE) or non-cacheable (TA_UNCACHE)

- (3) Cache write mode: Copy-back (TA_WBACK) or write-through (TA_WTHROUGH)

Note that the cache-related attributes are valid only while the cache is enabled. For an on-chip memory, the cache-related attributes are ignored, and the non-cacheable attribute is always assumed.

Reference: Section 5.3.3, On-Chip Memory

A memory object is assigned to a domain. Note the following about domains.

- (1) When a task returns a protected memory block to a protected memory pool (rel_mpp), the domain of the protected memory block must be the same as the domain of the task.
- (2) When a task sends a protected memory block to a protected mailbox (snd_mbp), the domain of the protected memory block must be the same as the domain of the task.

- (3) The initial access permission vector for a memory object that is allocated in the system pool depends on the assigned domain (see the next section).

Table 4.3 shows the domain of each memory object and the attributes of the domain. The attributes of each domain cannot be dynamically modified through service calls.

Table 4.3 Domain of Each Memory Object and Domain Attributes

Memory Object		Domain	Attribute
Memory object allocated in the system pool	Stack for the task in a user domain, which is allocated in the system pool by the kernel	Domain of the task (specified as the task attribute)	The following attributes are always applied. TA_RW
	Fixed-size memory pool area allocated in the system pool by the kernel	(1) When created by the configurator: Kernel domain	TA_CACHE
	Variable-size memory pool area allocated in the system pool by the kernel	(2) When created through a service call: Domain of the program that issued the service call	TA_WBACK
Protected memory block acquired from a protected memory pool		Domain of the task that acquired the memory block	Specified by the configurator at protected memory pool creation
Protected memory block received from a protected mailbox		Domain of the task that received the memory block	
Static memory object defined by the configurator		Kernel domain	Specified by the configurator at static memory object registration

4.21.4 Access Permission Vector

Each memory object has an access permission vector, which indicates which user domain can perform which type of access for that memory object. The kernel domain can access any memory object regardless of the access permission vector.

This kernel supports the access permission vectors shown in table 4.4.

Table 4.4 Access Permission Vectors

Access Permission Vector	User Domain that can Access to Memory Object	
	Write Access *	Read access
TACT_KERNEL	None (no user domain can access)	None (no user domain can access)
TACT_PRW(domid)	User domain with domid only	User domain with domid only
TACT_PRO(domid)	None (no user domain can access)	User domain with domid only
TACT_SRW	All user domains	All user domains
TACT_SRO	None (no user domain can access)	All user domains
TACT_SRPW(domid)	User domain with domid only	All user domains

Note: A memory object with the TA_RO attribute cannot be written to regardless of the access permission vector, even by the kernel domain.

The access permission vector can be modified through service call sac_mem.

Table 4.5 shows the initial access permission vector for each memory object.

Table 4.5 Access Permission Vector for Each Memory Object

Memory Object		Initial Access Permission Vector
Memory object allocated in the system pool	Stack for the task in a user domain, which is allocated in the system pool by the kernel	TACT_PRW(domid) domid indicates the domain of the task
	Fixed-size memory pool area allocated in the system pool by the kernel	(1) When created by the configurator: Value specified by the configurator
	Variable-size memory pool area allocated in the system pool by the kernel	(2) When created through a service call: An appropriate value is specified so that only the assigned domain can read or write to the memory object. For the kernel domain: TACT_KERNEL For a user domain: TACT_PRW(domid) domid indicates the assigned domain.
Protected memory block acquired from a protected memory pool		TACT_PRW(domid) domid indicates the domain of the task that acquired the memory block
Protected memory block received from a protected mailbox		TACT_PRW(domid) domid indicates the domain of the task that received the memory block
Static memory object defined by the configurator		Value specified by the configurator

4.21.5 Page Size

The MMU manages memory in page units. The basic page size in this kernel is 4 kbytes (CFG_PAGESZ). However, for static memory objects only, a value other than 4 kbytes can be specified as the page size. (Note that 1 kbyte is not allowed as the page size in the kernel specifications.)

The start address of each memory object must be aligned with a page boundary.

In static memory objects, specifying a large page size reduces the TLB entries to be used, which will decrease the TLB miss rate. However, it may degrade the efficiency of memory use.

For example, when a 40-kbyte area is used as a static memory object, if the page size is set to 4 kbytes, ten TLB entries are required. If the page size is set to 64 kbytes, only one TLB entry is required. In this case, when the page size is set to 64 kbytes, the TLB miss rate becomes 1/10 that for a 4-kbyte page when simply calculated. However, for a 64-kbyte page, an area between two

64-kbyte boundaries must be allocated and the last 24 kbytes in that area cannot be used for other memory objects, resulting in loss of available space.

Table 4.6 shows the difference in alignment among memory object types.

Table 4.6 Alignment of Each Memory Object

Memory Object	Alignment with Page Size
<div>Memory object allocated in the system pool</div> <hr/> <div>Stack for the task in a user domain, which is allocated in the system pool by the kernel</div> <hr/> <div>Fixed-size memory pool area allocated in the system pool by the kernel</div> <hr/> <div>Variable-size memory pool area allocated in the system pool by the kernel</div>	<div>Automatically aligned by the kernel.</div> <div>At linkage, the user must allocate the system pool section (BSCP_hisyspl) to align with a 4-kbyte boundary.</div>
Protected memory block acquired from a protected memory pool and protected memory block received from a protected mailbox	<div>Automatically aligned by the kernel.</div> <div>At linkage, the user must allocate the protected memory pool section to align with a 4-kbyte boundary.</div>
Static memory object defined by the configurator	<div>(1) Address specification</div> <div>The address of a boundary for the page size of the memory object must be specified. When a symbol is specified, make appropriate settings at linkage so that the symbol address is aligned with a boundary for the page size of the memory object.</div> <div>(2) Section specification</div> <div>The user must allocate the memory object section to align with a boundary for the page size of the memory object.</div>

4.21.6 Detection of Illegal Access

Refer to the following.

Reference: Section 5.3.1 (2), Detection of Illegal Access

4.21.7 TLB Miss Penalty

The MMU has a cache memory called the translation lookaside buffer (TLB), which stores protection information for each page. The kernel manages this TLB.

Note that the TLB is a cache and cannot indefinitely hold all information necessary for the system. If no valid information is found in the TLB when an MMU mapped area is accessed, the processor generates a TLB miss exception. In this case, the kernel appropriately updates the TLB according to the management information stored in memory and resumes exception processing. However, this means a large overhead for the respective section in the application because the kernel executes additional TLB miss exception processing when memory is accessed. This overhead is called a TLB miss penalty.

The following actions should be taken to avoid or reduce a TLB miss penalty.

(1) Allocating Target Area in MMU Non-Mapped Area

No TLB miss occurs when the target area is allocated in an MMU non-mapped area instead of a memory object. However, there is no MMU non-mapped area that can be accessed from a user domain. The on-chip memory is the only option that can be accessed from a user domain when "MMU non-mapped area, accessible in the user mode" is specified for CFG_IRAMUSAGE.

(2) Locking TLB

Through service call `vloc_tlb`, the page including the specified address can be temporarily locked in the TLB. Note that the number of pages locked at the same time is limited to the `CFG_MAXLOCPAGE` value (32 max.).

(3) Specifying Large Page Size for Static Memory Object

Specifying a large page size reduces the number of TLB entries used, resulting in a lower TLB miss rate.

(4) Reducing Number of Pages Used in the System at a Given Time to the Number of TLB Pages of the Microcomputer or Less

The sum of the following values is the number of pages used in the system at a given time.

- System pool size (`CFG_SYSPOOLSZ`)/4096
- Each static memory object size/page size for the memory object
- Each protected memory pool size/4096

When this sum is equal to or less than the number of TLB pages of the microcomputer, a TLB miss occurs only for the first access to each page, but will not occur again because that page will not be replaced in the TLB.

To avoid a TLB miss at the first access, access all pages intentionally during initialization before the main operation; after that, no TLB miss will occur during the main operation.

(5) Avoid Enabling MMU

When there is no possibility of memory access violation after debugging is completed, the kernel can be initiated (service call `vsta_knl`) with the MMU disabled. In this case, no TLB miss will occur. Note, however, that the exceptions described in section 4.21.6, Detection of Illegal Access, which can be detected by the MMU, will not be detected either.

Reference: Disabling MMU → Section 4.21.10, MMU Initialization

4.21.8 Access Permission Check (`prb_mem`)

Service call `prb_mem` can be used to check whether the specified domain is allowed to perform the specified access (read or write) to the specified address.

4.21.9 Check for Errors in Address Parameters of Service Calls

The address parameters for all service calls can be checked for errors in the same way as service call `prb_mem`. To enable this check function, select `CFG_MEMCHK`.

After the parameter correctness has been verified, deselecting this function can improve the performance.

4.21.10 MMU Initialization

(1) MMUCR Register

Some of the bits in MMUCR are initialized by `vsta_knl` (kernel initiation). The other bits must be initialized as needed by the application before `vsta_knl` is issued.

MMUCR must not be modified after `vsta_knl` is issued.

- AT bit: Not initialized by `vsta_knl`. Initialize it appropriately by the application before `vsta_knl`. Set 1 to enable the MMU or 0 to disable it.
- LRUI bits: Initialized to 0 by `vsta_knl`.

- URB bits: Initialized to 0 by vsta_knl.
- URC bits: Initialized to 0 by vsta_knl.
- SQMD bit (only for SH-4A): Not initialized by vsta_knl. Initialize it appropriately by the application before vsta_knl.
- SV bit: Initialized to 1 (single virtual memory mode) by vsta_knl.
- TI bit: Initialized to 1 by vsta_knl to invalidate all TLB entries.
- ME bit (when the CPU in use is the SH4AL-DSP or SH-4A with extended functions): Initialized to 1 by vsta_knl.

(2) TTB, PTEH, and PTEL Registers

The kernel uses these registers and they must not be modified after vsta_knl is issued (kernel initiation).

(3) PASCRC and IRMCRC Registers

These registers must be appropriately initialized by the application according to the system configuration before vsta_knl is issued. The kernel never accesses these registers.

Reference: In the sample system, these registers are initialized by
samples\shnnnn\kernel\knl_side\init_mmu.c.

4.22 Protected Memory Pool

A protected memory pool is an object used to dynamically allocate a memory object. Protected memory pools can be used only when the memory object protection function is selected.

A protected memory block acquired from a protected memory pool is assigned to the domain of the program that acquired the memory block, and is handled as a memory object that can be accessed only from that domain. Only protected memory blocks can be sent to protected mailboxes.

Protected memory pools can be created only by the configurator.

Protected memory pools are manipulated through the following service calls.

- Service call pget_mpp
Acquires a protected memory block with a specified size. When the memory block cannot be acquired (no memory block is available in the memory pool), an error is returned. The size of the block to be acquired is the value obtained by rounding the specified size up to a multiple of CFG_PAGESZ (4096).

- Service call `rel_mpp`
Returns a protected memory block. This is allowed only when the domain of the task that returns the protected memory block matches the domain of the memory block to be returned.
- Service call `ref_mpp`
Refers to the state of a protected memory pool.

Note:

The free space in protected memory pools may be fragmented. Refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

Figure 4.20 shows an example of protected memory pool usage.

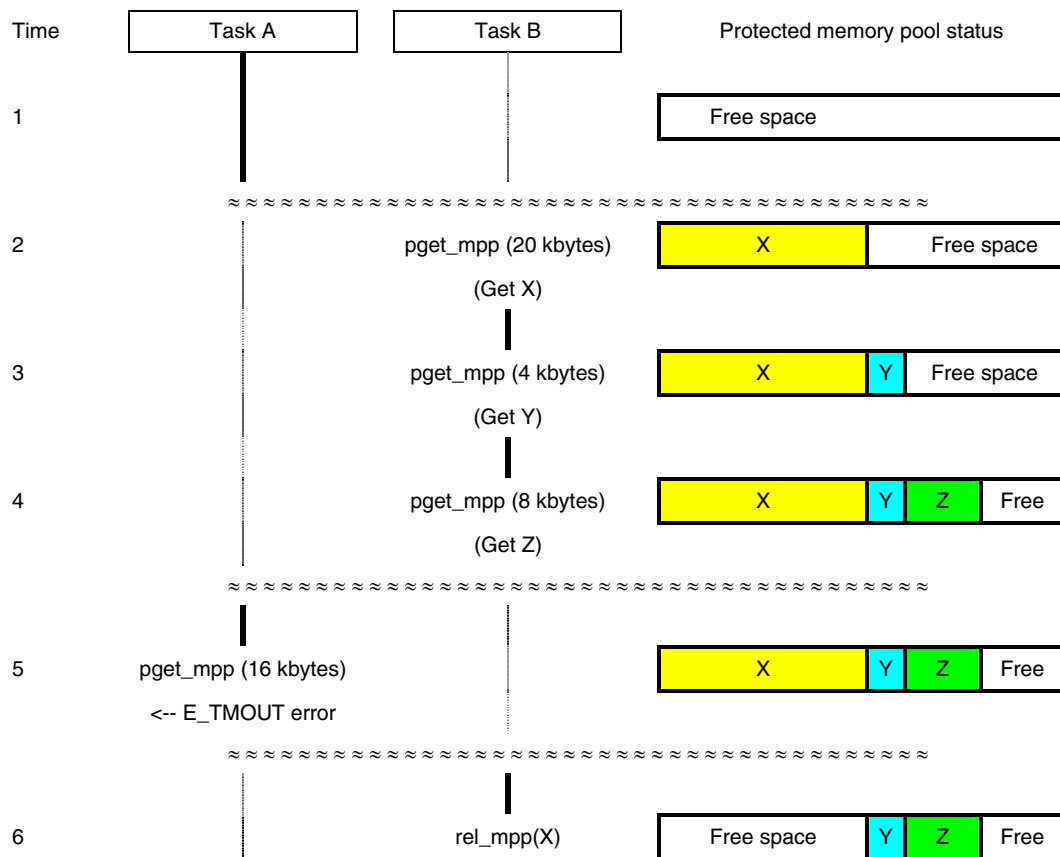


Figure 4.20 Example of Protected Memory Pool Usage

Description:

1. A 36-kbyte protected memory pool is created by the configurator.
2. Task B acquires 20-kbyte protected memory block X by issuing `pget_mpp`.
3. Task B acquires 4-kbyte protected memory block Y by issuing `pget_mpp`.
4. Task B acquires 8-kbyte protected memory block Z by issuing `pget_mpp`.
5. Task A attempts to acquire a 16-kbyte protected memory block by issuing `pget_mpp`. However, the free space is insufficient, and an `E_TMOUT` error is returned.
6. Task B returns 20-kbyte block X by issuing `rel_mpp`.

4.23 Protected Mailbox

A protected mailbox is an object used to pass messages between domains. It can be used only when the memory object protection function is selected.

High-speed data communication can be achieved because communication using a protected mailbox sends and receives only the start address of a message.

Only a protected memory block acquired from a protected memory pool can be used as a message.

Protected mailboxes can be created in the following ways.

- Service call `cre_mbp` or `icre_mbp`
A protected mailbox is created with a specified ID.
- Service call `acre_mbp` or `iacre_mbp`
A protected mailbox is created with the ID automatically assigned by the kernel.
- Created by the configurator
An ID name can be specified. When [Kernel side] is not selected in the configurator, the configurator can automatically assign an ID.

Protected mailboxes are manipulated through the following service calls.

- Service call `del_mbp`
Deletes the protected mailbox with the specified ID.
- Service call `snd_mbp`
Sends a protected memory block as a message to a protected mailbox. A message can be sent only when the domain of the task sending a message is the same as the domain of the protected memory block to be sent.
For a protected memory block sent to a protected mailbox, the access permission vector is changed to `TACT_KERNEL`.
- Service call `rcv_mbp` or `trcv_mbp`
Receives a message from a protected mailbox. When data cannot be received (the protected mailbox has no message), the task enters the `WAITING` state. In `trcv_mbp`, a timeout period can also be specified.
For the received message (protected memory block), the domain is changed to that of the task that received the message, and the access permission vector is changed so that only the task that received the message can read or write to the protected memory block.
- Service call `prcv_mbp`
Receives a message from a protected mailbox. When data cannot be received (the protected mailbox has no message), an error is returned.
For the received message (protected memory block), the domain is changed to that of the task that received the message, and the access permission vector is changed so that only the task that received the message can read or write to the protected memory block.
- Service call `ref_mbp` or `iref_mbp`
Refers to the state of a protected mailbox.

Figure 4.21 shows an example of protected mailbox usage.

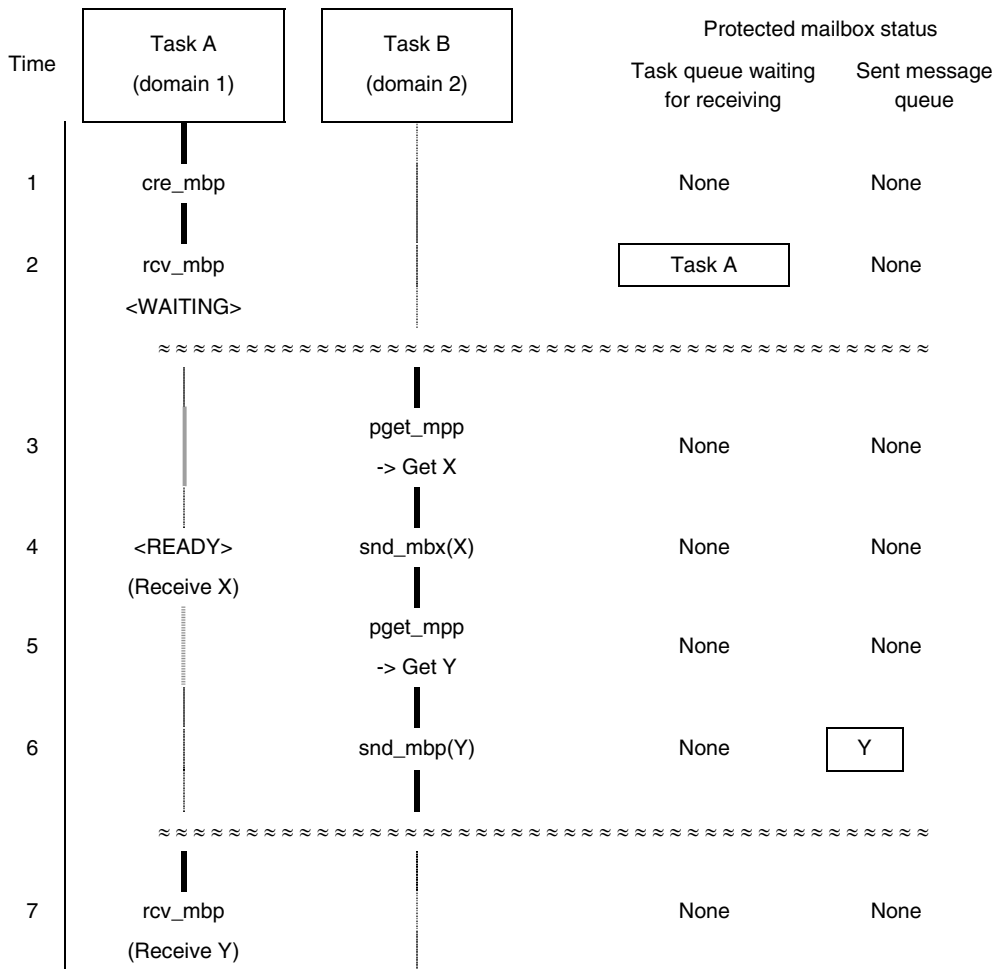


Figure 4.21 Protected Mailbox Usage

Description:

Bold lines represent executed processing. The following describes the mailbox operation with respect to time.

1. Task A issues **cre_mbp** to create a protected mailbox. The **TA_TFIFO** attribute (tasks waiting to receive are queued in FIFO) and the **TA_MFIFO** attribute (sent messages are queued in FIFO) are specified.
2. Task A attempts to receive a message by issuing **rcv_mbp**. Since no message is stored in the protected mailbox, task A enters the WAITING state.

3. Task B gets protected memory block X by issuing `pget_mpp` and creates a message in this block.
4. Task B sends protected memory block X by issuing `snd_mbp`. At this time, task A is released from the WAITING state, and task A receives the address of message X.
5. Task B gets protected memory block Y by issuing `pget_mpp` and creates a message in this block.
6. Task B sends protected memory block Y to the mailbox by issuing `snd_mbp`. At this time, since no tasks are waiting for a message, message Y is placed in a message queue.
7. Task A issues `rcv_mbp`. Task A receives the address of message Y, which is at the head of the message queue.

4.24 System Memory Management

4.24.1 System Pool

The system pool is an area where the kernel allocates the following areas.

- Stack areas for the tasks in user domains
- Fixed-size memory pool areas
- Variable-size memory pool areas

The system pool size should be specified through `CFG_SYSPOOLSZ` in the configurator.

If a service call requiring the system pool is issued when the system pool does not have sufficient free space, an `E_NOMEM` error is returned.

When the memory object protection function is selected, an area allocated in the system pool is handled as a memory object. The requested allocation size is rounded up to a multiple of `CFG_PAGESZ` (4 kbytes). The start address of an allocated area is aligned to a `CFG_PAGESZ`-size boundary.

When the memory object protection function is not selected, the requested allocation size is rounded up to a multiple of 64. The start address of an allocated area is aligned to a 32-byte boundary.

The current system pool status can be checked by issuing service call `vref_syp`.

Note that the kernel never creates management tables in the system pool.

The free space in the system pool may be fragmented. Refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

For calculation of the system pool size, refer to the following.

Reference: Section 14, Estimation of System Pool Size

4.24.2 Resource Pool

The resource pool is an area where the kernel allocates management tables, which are required in a dynamic manner in the kernel.

The resource pool size should be specified through CFG_RESPOOLSZ in the configurator.

The kernel acquires an area from the resource pool as necessary according to the service call issued. The requested allocation size is rounded up to a multiple of 20.

If a service call requiring the resource pool is issued when the resource pool does not have sufficient free space, an E_NOMEM error is returned.

The current resource pool status can be checked by issuing service call vref_rsp.

The free space in the resource pool may be fragmented. Refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

For the timing for resource pool requests and calculation of the resource pool size, refer to the following.

Reference: Section 13, Estimation of Resource Pool Size

4.25 DSP Standby Control

4.25.1 Overview

This function reduces power consumption by making the kernel automatically place the X/Y memory in the module stop state provided in the microcomputer when programs without the TA_COP0 attribute are running.

This function also supports the vchg_cop service call, which is used to dynamically change the TA_COP0 attribute. Using vchg_cop makes it possible to extend the periods over which hardware resources are placed in the module stop state.

Figure 4.22 gives an overview of this function.

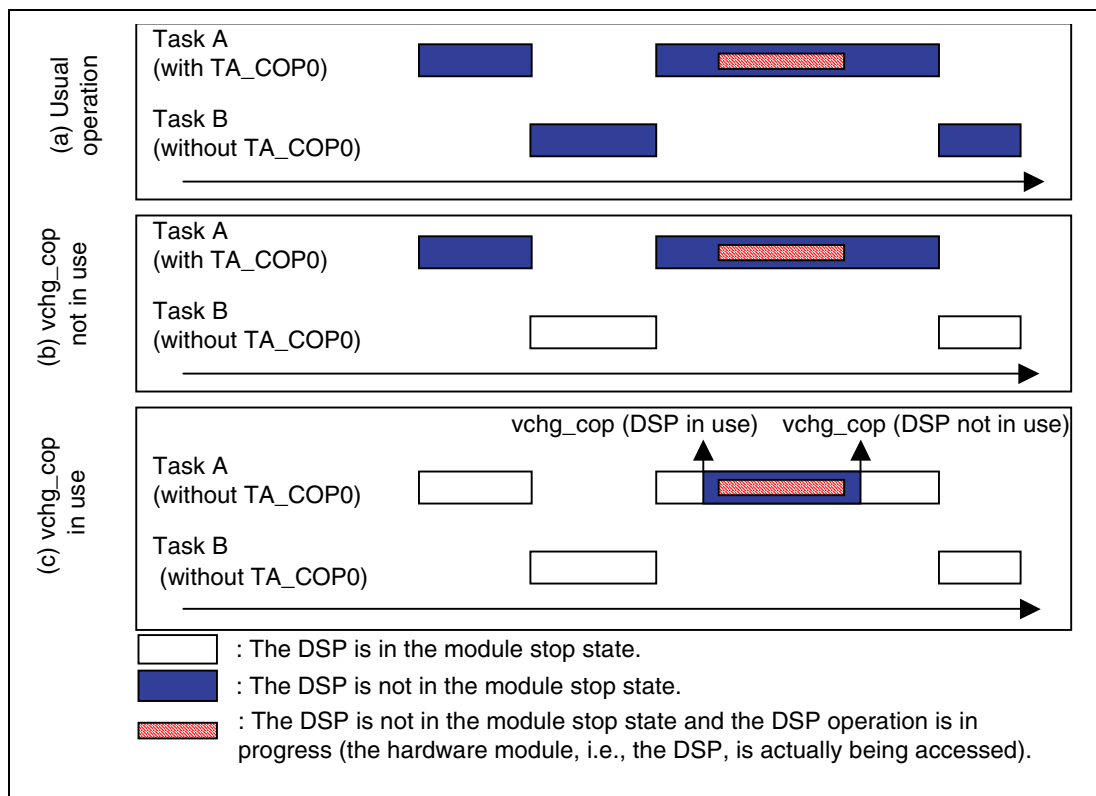


Figure 4.22 Overview of DSP Standby Control

4.25.2 Applicable Microcomputers

This function is only usable with microcomputers that include a DSP and satisfy the following condition.

- The register that specifies whether to stop the X/Y memory is 32 bits long.

4.25.3 Module Stop State of X/Y Memory when Initiating Programs

When processing of a program is initiated and the DSP-standby control function has been included, the X/Y memory status becomes as shown in table 4.7.

In the programs for which 'Undefined' are indicated in table 4.7, the X/Y memory associated with the TA_COP0 attribute must not be accessed. To use the X/Y memory, cancel the module stop state after saving the module stop state information on the program side when initiating the program, and return the X/Y memory to the saved state before leaving the program.

Table 4.7 Module Stop State when Initiating Programs

Program	Module Stop State
Task, task exception processing routine, extended service call routine, trap routine	With TA_COP0: Non-module stop state Without TA_COP0: Module stop state
Interrupt handler	Undefined (same state as before the interrupt)
CPU exception handler	Undefined (same state as before the CPU exception)
Time event handler	Undefined
Initialization routine	Undefined

When the kernel is idling (i.e., there is no READY task), the X/Y memory enters the module stop state.

4.25.4 Notes

For example, when task A with the TA_COP0 attribute specifies the X/Y memory as the source or destination of a DMA transfer and then starts the transfer, the transfer will not proceed correctly if execution is switched to task B that does not have the TA_COP0 attribute, since the kernel places the X/Y memory in module stop state during the execution of task B.

If an interrupt occurs during the above-described transfer by task A and the interrupt handler issues an extended service call that does not have the TA_COP0 attribute, the transfer will not proceed correctly since the kernel places the X/Y memory in module stop state during the execution of the extended service call.

This is because access to the X/Y memory by the DMA is asynchronous with the software operation, while the module stop control of the X/Y memory is synchronized with task execution.

To avoid this, do not use the DSP standby control function or take the following measures.

- Not using DMA transfer with the X/Y memory.
- Keeping the CPU-locked state until the completion of the DMA transfer.

4.26 Performance Management

(1) Overview

The performance management function measures performance such as the time and count of programs by using the program performance counters built into the microcomputer. Note that some microcomputers do not have program performance counters.

This function uses program performance counters 0 and 1. Each counter consists of 32 bits.

To include the performance management function, select CFG_PERFORM.

The user should fully understand the operation of the program performance counters. For details, refer to the appropriate document regarding the target microcomputer, such as SH-4A, SH4AL-DSP Program Performance Counters Application Notes.

(2) Measurement Items (CFG_PPC0TYPE, CFG_PPC1TYPE)

The items to be measured can be specified through CFG_PPC0TYPE and CFG_PPC1TYPE in the configurator. The former parameter specifies the measurement in counter 0, and the latter specifies it in counter 1.

Through each parameter, specify the value to be set to the CIT9 to CIT0 bits (10 bits) in the count condition set register (CCBR0 or CCBR1). The input value is masked with H'3ff.

For example, specifying 0 indicates the elapsed cycle count. If the measured count is H'100000 when the CPU is operating at 266 MHz, the elapsed time is $H'100000/266 \text{ MHz} = 3942 \mu\text{s}$.

(3) Connecting Counters

Each counter has only 32 bits and is quickly overflowed. To reduce overflows, two counters can be connected to become a 64-bit counter. To connect the counters, select CFG_CONNECT.

Note that when the counters are connected, the kernel temporarily stops counting when reading the counter, that is, an error in the measured value due to this stop will be accumulated.

(4) Measurement Targets

The kernel manages the accumulated values of the program performance counters according to the following classifications.

- Each task (including the extended service call routines and trap routines called from each task)
- Kernel idling state
- Others (non-task context, kernel)

When the memory object protection function is selected, the measured value regarding TLB-related exception processing by the kernel is included in the value for the context that generated the exception.

(5) Controlling Program Performance Counters

When service call `vsta_knl` is issued, the kernel initializes program performance counters 0 and 1 to 0 and starts counting. After that, counting does not stop except for the temporary stop described in (3) Connecting Counters.

(6) Service Calls

The following service calls are provided.

- `vchg_ppc`: Starts, stops, or initializes performance measurement
- `vref_ppc`: Refers to performance measurement result

(7) Using the HI7300/PX with an Emulator

While an emulator is used with the HI7300/PX, the emulator may occupy all of the program performance counters. Before using the performance measurement function of the kernel, read the user's manual or help of the emulator and make a setting so that the emulator will not occupy the counters. If you wish to use an E10A-USB emulator, for example, select [User] in the [PPC mode] list box of the [Configuration] dialog box or enter "user" with the `PPC_MODE` command in the HEW to release the counters to the user.

4.27 Service Call Trace

The service call trace function acquires the system operation history, such as the service calls issued or interrupts generated. When the performance management function is used, the program performance counter values are also traced. The acquired trace information can be displayed by using the debugging extension.

To use the trace function, select `CFG_TRACE`.

For details of the trace function, refer to the help information regarding the debugging extension.

(1) Trace Timing and Information to be Acquired

The trace information is acquired with the following timing.

- A service call
- Return from a service call
- Initiation of a task or handler
- Completion of a task or handler
- Transition to the kernel idling state

The following information is acquired.

- Parameters for service calls
- Error codes for service calls
- Value of the program counter (PC)
- Values of program performance counters 0 and 1 (only when the performance management function is used)

(2) Trace Type (`CFG_TRCTYPE`)

The trace information can be stored either in the buffer allocated in the RAM on the target system or in the trace memory in the simulator or emulator, which can be selected through `CFG_TRCTYPE`. The former is called a target trace and the latter is called a tool trace. For a target trace, specify the buffer size through `CFG_TRCBUFSZ`.

(3) Number of Objects (`CFG_TRCOBJCNT`)

In the debugging extension, the state of the objects specified by the user can also be acquired with the trace timing. The maximum number of objects that can be traced at one time is specified through `CFG_TRCOBJCNT`.

(4) User Event Trace (`vget_trc`, `ivget_trc`)

Use `vget_trc` or `ivget_trc` to acquire any user-specified information with the user-specified timing.

4.28 Other Functions

(1) Rotating Ready Task Queue (rot_rdq, irot_rdq)

By issuing these service calls at specified intervals (for example, issuing from a cyclic handler), the round-robin scheduling necessary for the time-sharing system can be implemented. (See figure 4.23)

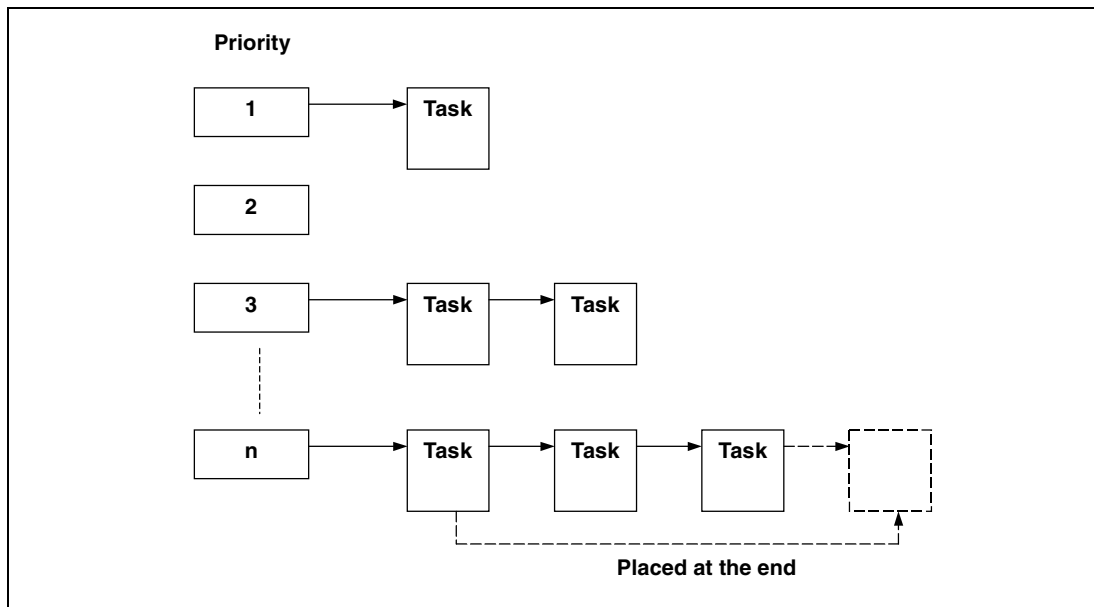


Figure 4.23 Ready Queue Manipulation through Service Call rot_rdq

(2) Acquiring ID of Current Running Task (get_tid, iget_tid)

The ID of the current running task can be acquired through these service calls.

(3) Acquiring ID of Domain of Current Running Task (get_did, iget_did)

The ID of the domain where the current running task is assigned can be acquired through these service calls.

Extended service call routines and trap routines are assigned to the kernel domain, but through these service calls, the domain ID of the task that has called an extended service call or trap routine can be checked.

(4) Referring to Configuration Information (ref_cfg, iref_cfg)

The configuration information such as the maximum ID for each object can be acquired through these service calls.

(5) Referring to Kernel Version Information (ref_ver, iref_ver)

The version information of the kernel can be acquired through these service calls. Part of the information obtained by ref_ver can also be checked through configuration constants.

Reference: Section 6.1.5 (1), Configuration Constants

4.29 Kernel Idling

When there is no READY task, the kernel enters an endless loop and waits for an interrupt.

To use a power-down mode of the CPU, the lowest-priority task is usually used for transition to that mode.

4.30 Resetting the CPU and Initiating the Kernel

Figure 4.24 shows a general flow from a CPU reset to kernel initiation (vsta_knl). Note that no parameters or codes will be returned from vsta_knl.

For the flow from a vsta_knl call to transition to multitasking environment, refer to the following.

Reference: Section 6.22.12, Start Kernel (vsta_knl, ivsta_knl)

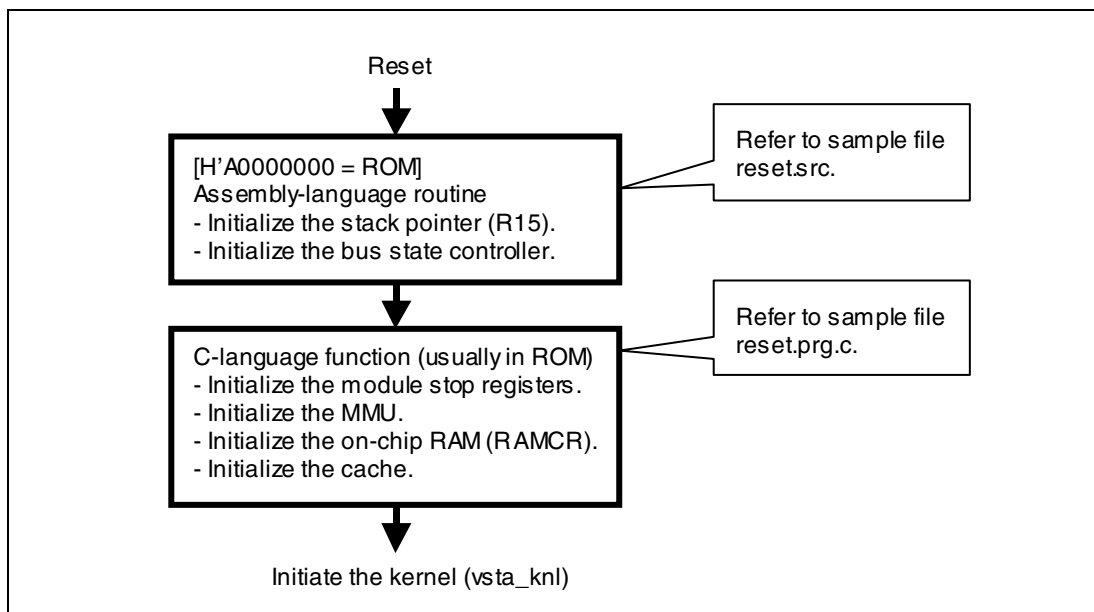


Figure 4.24 Flowchart from CPU Reset to Kernel Initiation

(1) Initializing Bus State Controller

To execute a C-language function, the RAM area to be used for stacks or data sections must be ready to be accessed. When a RAM, such as SDRAM, that should be initialized is used, initialize the bus state controller before executing a C-language function.

The bus state controller must be initialized according to the specifications of the target board.

(2) Initializing Module Stop Registers

When there is no need to use some modules in the system, these modules should be stopped to reduce power consumption. For the modules to be used in the system, the module stop state must be canceled.

When the memory object protection function is selected, cancel the module stop state for the TLB at this step.

(3) Initializing MMU

When the memory object protection function is not selected, initialize MMUCR to 0.

When the memory object protection function is selected, refer to the following.

Reference: Section 6.22.12, Start Kernel (vsta_knl, ivsta_knl)
Section 4.21.10, MMU Initialization

When the 32-bit address extended mode is used, refer to the following.

Reference: Section 5.6, 32-Bit Address Extended Mode

(4) Initializing RP and RMD Bits in RAMCR

These bits are initialized in vsta_knl execution when the memory object protection function is selected.

When the memory object protection function is not selected, the kernel does not initialize these bits. Refer to the following.

Reference: Section 5.2.3, On-Chip Memory

(5) Initializing Cache and IC2W and OC2W Bits in RAMCR

These are initialized when cache support function `sh4a_vini_cac()` is called.

(6) Disabling Interrupts and Suppressing CPU Exceptions

The kernel becomes ready for interrupt and CPU exception handling only after the kernel is initiated. Generally, all interrupts must be disabled and CPU exceptions must not be generated until the kernel is initiated.

To disable all interrupts, set $SR.BL = 1$. Immediately after a CPU reset, this state is automatically entered.

When the memory object protection function is selected, the MMU must be enabled on the application side before the kernel is initiated. Even in this case, MMU mapped areas must not be accessed to ensure that no CPU exception (TLB-related exception) occurs until the kernel is initiated.

If a CPU exception occurs while $SR.BL = 1$, execution branches to the reset vector.

For initialization of sections and standard library functions, refer to the following.

Reference: Section 11.7, Standard Library Functions and Runtime Routines

4.31 Controlling Memory Fragmentation (VTA_UNFRAGMENT Attribute)

The free space may be fragmented in the following areas.

- Variable-size memory pool
- Protected memory pool
- System pool
- Resource pool

Repeated acquisition and return of memory from these areas causes fragmentation of free space and contiguous free space can become insufficient even if the total size of the free space is sufficient; as a result, no large memory area can be acquired (figure 4.25).

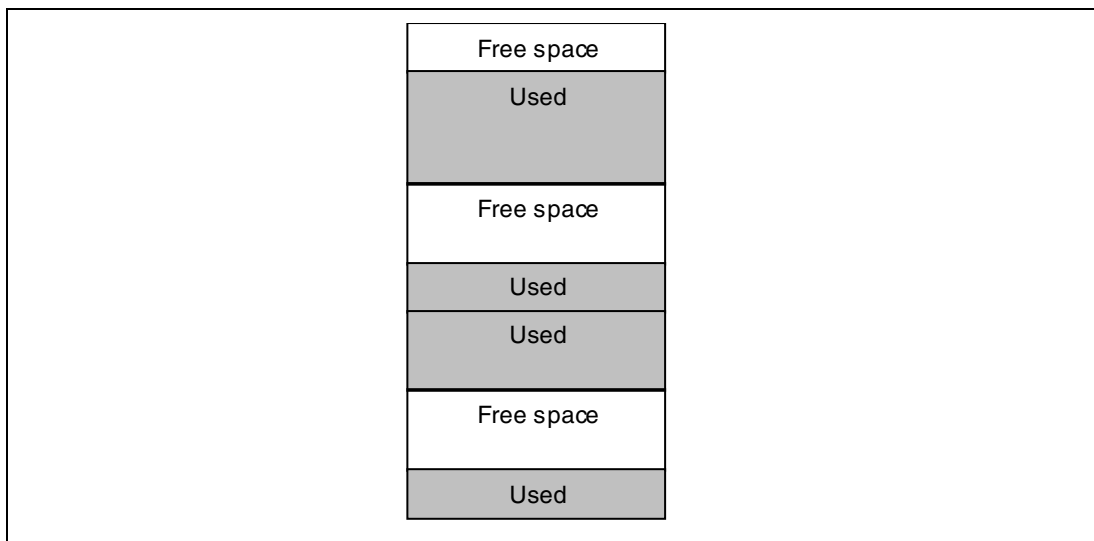


Figure 4.25 Fragmentation of Free Space

This kernel supports the sector management method to reduce this fragmentation; that is, the VTA_UNFRAGMENT attribute for variable-size memory pools and protected memory pools. The system pool and resource pool supports a similar measure.

The sector management method reduces fragmentation when a large number of small blocks and some large blocks are allocated in a large memory pool.

In this method, up to (minimum block size \times 8 bytes) is handled as a "small block" size. The size of each memory acquisition request is rounded up as shown in table 4.8.

When a "small block" is requested, the kernel creates a sector consisting of blocks whose size is a rounded value of the requested size. The size of the sector is always ($\text{minblksz} \times 32$). In other words, the number of blocks in the sector depends on the requested size.

Table 4.8 Handling of "Small Blocks"

Requested Block Size*	Rounded-up Block Size	Number of Blocks in the Sector
$0 < \text{blksz} \leq \text{minblksz}$	minblksz	32
$\text{minblksz} < \text{blksz} \leq \text{minblksz} \times 2$	$\text{minblksz} \times 2$	16
$\text{minblksz} \times 2 < \text{blksz} \leq \text{minblksz} \times 4$	$\text{minblksz} \times 4$	8
$\text{minblksz} \times 4 < \text{blksz} \leq \text{minblksz} \times 8$	$\text{minblksz} \times 8$	4

Note: blksz and minblksz mean the requested size and the minimum block size, respectively.

The kernel then assigns the memory blocks in the sector. The remaining blocks in the sector are reserved for later requests for memory blocks with this size or a smaller size.

In this manner, small blocks are allocated contiguously so that a larger free space is left available.

Figure 4.26 shows an example of a variable-size memory pool when the minimum block size is 32.

First a 32-byte memory block is requested. Sector [A] with $32 \times 32 = 1024$ bytes is allocated and 32-byte area [A-1] in the sector is assigned for the requested block (figure 4.26 (1)). When a 16-byte memory block is then requested, 32-byte area [A-2] in sector A is assigned (figure 4.26 (2)).

Next, a 36-byte memory block is requested. Since the size of each block in sector A is 32 bytes, no block in sector A can be assigned for this request. To respond to this request, new sector [B] is allocated for $16 \text{ blocks} \times 64 \text{ bytes}$ (the requested size, 36, is rounded up to a multiple of the minimum block size) = 1024 bytes, and 64-byte area [B-1] is assigned for the requested block (figure 4.26 (3)).

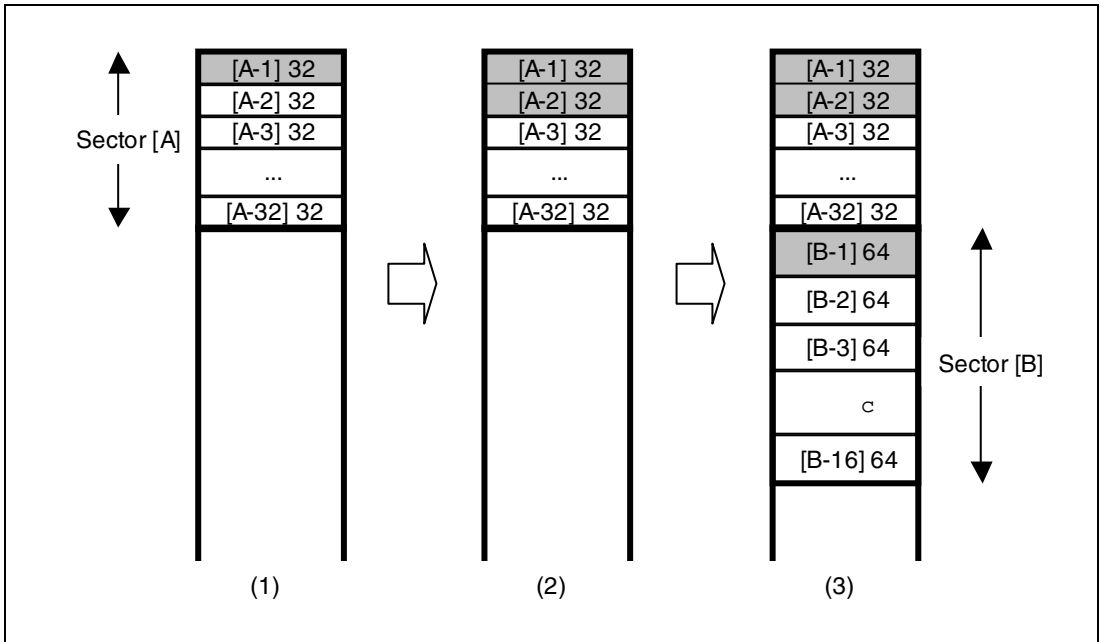


Figure 4.26 Example of Variable-Size Memory Pool

The minimum block size and the maximum number of sectors are determined according to the pool type as shown in table 4.9.

If the maximum number of sectors have already been used or free space is not sufficient to create a new sector, the requested size of the memory block is allocated without creating a sector. In this case, free space may be fragmented.

When all blocks in a sector are released, the sector itself is also released.

Table 4.9 Minimum Block Size and Maximum Number of Sectors for Each Pool

Pool		Minimum Block Size	Maximum Number of Sectors
Variable-size memory pool		Specified at creation	Specified at creation
Protected memory pool		Specified at creation	Specified at creation
System pool	Memory object protection function selected	CFG_PAGESZ (= 4 kbytes)	Specified by configurator (CFG_SYSPPOOLSCTNUM)
	Memory object protection function not selected	64	
Resource pool		20	As many sectors as possible can be created on request.

4.32 Debugging Extension

The debugging extension adds multitasking debugging functions to HEW. It can be downloaded from our website.

The debugging extension provides the following functions. For details, refer to the manual or help information for the debugging extension.

(1) Referring to Object States

The states of various objects such as tasks or semaphores can be displayed.

(2) Changing Object States (Issuing Service Calls)

Object states can be changed by initiating a task or setting an event flag. This function is available only when CFG_ACTION is selected in the configurator.

(3) Displaying Trace Information

This function is available only when CFG_TRACE is selected in the configurator.

Section 5 Logical Address Space

5.1 Overview

This kernel always assume logical addresses = physical addresses regardless of whether the memory object protection function is used.

In this kernel, the logical addresses for all code and data areas are determined by section allocation at linkage in principle. Each section must be allocated to an appropriate logical address according to the function of the section. The description in this section should be fully understood to determine appropriate addresses.

This section gives necessary information regarding actual addresses for external memory and on-chip memory.

5.2 When Memory Object Protection Function Is Not Used

5.2.1 Overview

When the memory object protection function is not selected, memory protection is available with distinction between the privileged mode (kernel domain) and user mode (user domain).

For the SH4AL-DSP or SH-4A CPU, the exceptions listed in table 5.1 can be detected. The exception conditions indicated in bold face in the table are related to memory protection. For details, refer to the manual of the target microcomputer.

Table 5.1 Protection-Related Exceptions Detected by CPU

Exception Code (EXPEVT)	Exception Conditions
H'0E0 * ²	<p>(1) Instruction address error: Detected under either of the following conditions.</p> <ul style="list-style-type: none"> — Instruction fetch from address H'80000000 or a higher address in the user mode *¹ — Instruction fetch from an odd address <p>(2) Data address error (read): Detected under either of the following conditions.</p> <ul style="list-style-type: none"> — Data read from address H'80000000 or a higher address in the user mode *¹ — Read access with illegal alignment
H'100 * ²	<p>Data address error (write): Detected under either of the following conditions.</p> <ul style="list-style-type: none"> • Write to address H'80000000 or a higher address in user mod *¹ • Write access with illegal alignment
H'180 * ²	<p>General illegal instruction exception: Detected under any of the following conditions.</p> <ul style="list-style-type: none"> • Decoding in the user mode of a privileged instruction not in a delay slot • Decoding of an undefined instruction not in a delay slot • DSP instruction execution while SR.DSP = 0 (SH4AL-DSP only)
H'1A0 * ²	<p>Slot illegal instruction exception: Detected under any of the following conditions.</p> <ul style="list-style-type: none"> • Decoding in the user mode of a privileged instruction in a delay slot • Decoding of an undefined instruction in a delay slot • Decoding of an instruction that modifies PC in a delay slot • Decoding of a PC-relative MOV or MOVA instruction in a delay slot
H'800	<p>General FPU disable exception (SH-4A only):</p> <p>Decoding of an FPU instruction not in a delay slot while SR.FD = 1</p>
H'820	<p>Slot FPU disable exception (SH-4A only):</p> <p>Decoding of an FPU instruction in a delay slot while SR.FD = 1</p>

Notes: *¹ On-chip memory can be accessed in some cases. For details, refer to section 5.2.3, On-Chip Memory.

*² The sample system defines a CPU exception handler (samples\sysapp\cpuexc.c) for these exception codes.

5.2.2 External Memory

The logical address space of the external memory is divided into areas as shown in table 5.2.

Table 5.2 Areas in External Memory Address Space

Area (Address Range)	Access in User Mode	Operation with Cache Enabled		Remarks
		Read	Write	
P0/U0 area (0 to H'7fffffff)	Allowed	Cacheable	Cacheable * ¹	
P1 area (H'80000000 to H'9fffffff)	Not allowed	Cacheable	Cacheable * ²	
P2 are (H'a0000000 to H'bfffffff)	Not allowed	Non-cacheable	Non-cacheable	
P3 area (H'c0000000 to H'dfffffff)	Not allowed	—	—	It is prescribed in the kernel specifications that the P3 area must not be used.
P4 area (H'e0000000 to H'ffffffff)	Not allowed	Non-cacheable	Non-cacheable	On-chip resources of the microcomputer are mapped to this area.

Notes: *1 The cache write mode is set to the write-through mode when the WT bit in CCR is 1 (TCAC_P0_WT is specified in vini_cac); otherwise, it is the copy-back mode.

*2 The cache write mode is set to the copy-back mode when the CB bit in CCR is 1 (TCAC_P1_CB is specified in vini_cac); otherwise, it is the write-through mode.

5.2.3 On-Chip Memory

(1) Logical Addresses for On-Chip Memory

The logical addresses of the on-chip memory are mapped to the P2 or P4 area, but the attributes of these addresses are different from those of the P2 or P4 area for external memory; the attributes are determined by the RMD bit setting in RAMCR.

Table 5.3 shows the attributes of the logical addresses for the on-chip memory. Access to the on-chip memory is always non-cacheable. The RMD bit in RAMCR must be initialized appropriately by the application, with reference to table 5.3.

Table 5.3 Attributes of Logical (Virtual) Addresses for On-Chip Memory

RAMCR.RMD Value	Access in User Mode	Operation with Cache Enabled	
		Read	Write
0	Not allowed *	Non-cacheable	Non-cacheable
1	Allowed	Non-cacheable	Non-cacheable

Note: When SR.DSP = 1, access is allowed even in a user domain.

(2) Enabling Cacheable Access

The following describes how to enable cacheable access to the on-chip memory.

Note that for some types of on-chip memory, such as X/Y memory or L memory, cacheable access is never allowed.

(a) On-chip memory whose physical addresses are mapped to area 1

Some microcomputer includes on-chip memory whose physical addresses are mapped to area 1. Such memory can be accessed through area 1 (in P0/U0 area) or using its shadow addresses in P1 area. The attributes of these addresses accessed in this way are the same as those of the external memory.

(b) Using 32-bit address extended mode

In a microcomputer supporting the 32-bit address extended mode, the physical addresses of the on-chip memory can be mapped to the logical addresses of the P1 or P2 area by making an appropriate setting in PMB. The cache operation can be controlled through C and WT bits in PMB. However, note that the P1 and P2 area cannot be accessed in the user mode.

5.3 When Memory Object Protection Function Is Used

5.3.1 Overview

(1) MMU Mapped Area and MMU Non-Mapped Area

All addresses in the logical address space are classified into either the MMU mapped area or the MMU non-mapped area.

(a) MMU Non-Mapped Area

An MMU non-mapped area is accessed without MMU intervention. This type of areas can be normally accessed only in the privileged mode. However, the on-chip memory can also be accessed in the user mode when CFG_IRAMUSAGE is appropriately set in the configurator.

(b) MMU Mapped Area

Access to an MMU mapped area is checked by the MMU. Areas to be used as memory objects, such as static memory objects, must be allocated in MMU mapped areas.

(2) Detection of Illegal Access

When the memory object protection function is selected, two types of memory protection functions work: one is provided by the CPU and the other by the MMU.

Protection by the CPU is exactly the same as the protection when the memory object protection function is not used. See section 5.2.1, Overview.

The MMU detects the following illegal access to the MMU mapped area.

1. No memory object is found in the accessed address.
2. Write access to a memory object having the TA_RO attribute was attempted.
3. A memory object was accessed from a user domain that is not allowed to access by the access permission vector for that memory object.

To implement this function, the kernel handles the following CPU exceptions. Even if a CPU exception handler is defined for any of these exception codes, the kernel ignores it.

- EXPEVT = H'040: Instruction TLB miss exception or data TLB miss exception (read)
- EXPEVT = H'060: Data TLB miss exception (write)
- EXPEVT = H'0A0: Instruction TLB protection violation exception or data TLB protection violation exception (read)

- EXPEVT = H'0C0: Data TLB protection violation exception (write)

If an illegal access is attempted, the memory access violation handler (samples\sysapp\mavhdr.c) is initiated. A memory access violation handler must be created and installed in the system.

The MMU hardware provides the following functions, but this kernel uses the MMU only for access protection and cache control, that is, address translation is not performed.

- Address translation
- Access protection
- Cache control

Note: Section 8.9, Memory Access Violation Handler

5.3.2 External Memory Space

The logical address space of the external memory is divided into areas as shown in table 5.4

Table 5.4 Areas in External Memory Address Space

Area (Address Range)	MMU Mapped/ Non-Mapped Area	Access in User Mode	Operation with Cache Enabled		Remarks
			Read	Write	
P0/U0 area (0 to H'7fffffff)	MMU mapped area	Depends on the access permission vector for the target memory object	Depends on the memory object attribute		
P1 area (H'80000000 to H'9fffffff)	MMU non- mapped area	Not allowed	Cacheable	Cacheable * ¹	
P2 area (H'a0000000 to H'bfffffff)	MMU non- mapped area	Not allowed	Non- cacheable	Non- cacheable	
P3 area (H'c0000000 to H'dfffffff)	—	Not allowed	—	—	It is prescribed in the kernel specifications that the P3 area must not be used.
P4 area (H'e0000000 to H'ffffffff)	MMU non- mapped area	Not allowed	Non- cacheable	Non- cacheable	On-chip resources of the microcomputer are mapped to this area.

Note: *¹ The cache write mode is set to the copy-back mode when the CB bit in CCR is 1 (TCAC_P1_CB is specified in vini_cac); otherwise, it is the write-through mode.

5.3.3 On-Chip Memory

(1) Logical Addresses for On-Chip Memory

The attributes of the logical addresses for the on-chip memory are determined according to the settings in the configurator. The kernel initializes the RP and RMD bits in RAMCR according to the configurator settings during execution of `vsta_knl`.

Table 5.5 shows the attributes of the logical addresses for the on-chip memory. Access to the on-chip memory is always non-cacheable.

Table 5.5 Attributes of Logical (Virtual) Addresses for On-Chip Memory and RAMCR Initialization by `vsta_knl`

Configurator Setting CFG_IRAMUSAGE	MMU Mapped/ Non-Mapped Area	Access in User Mode	Operation with Cache Enabled		RAMCR Initialization by <code>vsta_knl</code>
			Read	Write	
(1) MMU non-mapped area, accessible in any mode	MMU non-mapped area	Allowed	Non-cacheable	Non-cacheable	RP = 0 RMD = 1
(2) MMU non-mapped area, not accessible in user (non-DSP) mode	MMU non-mapped area	Not allowed ^{*1}	Non-cacheable	Non-cacheable	RP = 0 RMD = 0
(3) MMU mapped area	MMU mapped area	Depends on the access permission vector for the target memory object	Non-cacheable ^{*2}	Non-cacheable ^{*2}	RP = 1 RMD = 1

Notes: ^{*1} When SR.DSP = 1, access is allowed even from a user domain.

^{*2} Always non-cacheable regardless of the memory object attribute.

(2) Enabling Cacheable Access

The following describes how to enable cacheable access to the on-chip memory.

Note that for some types of on-chip memory, such as X/Y memory or L memory, cacheable access is never allowed.

(a) On-chip memory whose physical addresses are mapped to area 1

Some microcomputer includes on-chip memory whose physical addresses are mapped to area 1. Such memory can be accessed through area 1 (in P0/U0 area) or using its shadow addresses in P1 area. The attributes of these addresses accessed in this way are the same as those of the external memory.

(b) Using 32-bit address extended mod

In a microcomputer supporting the 32-bit address extended mode, the physical addresses of the on-chip memory can be mapped to the logical addresses of the P1 or P2 area by making an appropriate setting in PMB. The cache operation can be controlled through C and WT bits in PMB. However, note that the P1 and P2 area cannot be accessed in the user mode.

5.3.4 Note on Use

A TLB miss exception may occur during access to an MMU mapped area. If this exception occurs while the BL bit in SR is 1, the CPU is reset; so, do not access an MMU mapped area while BL = 1. Neither data to be accessed nor the program that accesses the data must be allocated in an MMU mapped area.

5.4 On-Chip Resources Allocated in P4 Area

As shown in table 5.2, the P4 area cannot be accessed from a user domain.

The only way to access an on-chip resource allocated in the P4 area from a user domain is to register the resource as a static memory object by using the memory object protection function.

5.5 On-Chip Resources whose Physical Addresses Are Allocated in Area 1

In usual operation, the first three bits of the physical address of such a resource must be modified to B'101 to access it as the P2 area. As shown in table 5.2, the P2 area cannot be accessed from a user domain.

The only way to access such a resource in the user mode is to register the resource as a static memory object by using the memory object protection function.

5.6 32-Bit Address Extended Mode

Some microcomputers supports the 32-bit address extended mode. To use this mode, take the following initialization steps before initiating the kernel. Refer also to sample file `samples\sh7780\kernel\knl_side\7780\init_mmu.c`.

1. Make a setting in PMB.
2. Turn on the SE bit in PASCR.
3. Enable the MMU if necessary.

When using the 32-bit address extended mode, note that the attributes of the P1 and P2 area depend on the PMB setting.

Section 6 Service Calls

6.1 C-Language API

6.1.1 Calling Form

All service calls are described in the following C language function call format.

```
ercd = act_tsk (1);
```

6.1.2 Header File

(1) include\itron.h

itron.h defines necessary information such as basic data types.

(2) include\kernel.h

kernel.h defines the specifications of the kernel and includes itron.h and the files listed in table 6.1.

Table 6.1 Files Included in kernel.h

File Name	Description
include\kernel_api.h	Defines the kernel service calls.
include\kernel_tsz.h	Defines the macros for memory size specifications.
kernel_macro.h	Defines the macros for kernel specifications. The configurator creates this header file.

6.1.3 Header Files Output from the Configurator

The configurator outputs three header files: kernel_macro.h, kernel_id.h, and kernel_id_sys.h.

(1) kernel_macro.h

This file contains the define statements for part of the configurator settings. For the detailed contents, refer to table 6.3.

kernel_macro.h is included in kernel.h; the application does not need to directly include kernel_macro.h.

Note that kernel_macro.h is not output when the configurator is in kernel-locked mode.

(2) kernel_id.h and kernel_id_sys.h

These files contain the define statements for the ID names specified and output by the configurator.

kernel_id_sys.h contains the ID names of the objects created with the [Kernel Side] checkbox being selected through the configurator and kernel_id.h contains the ID names of the objects created without the checkbox being selected.

Note that kernel_id_sys.h is not output when the configurator is in kernel-locked mode.

These files must be explicitly included in the application as necessary. Any kernel side file must not include kernel_id.h.

6.1.4 Basic Data Type

The basic data types defined in itron.h are shown below.

```
typedef signed char      B;          /* signed 8 bit integer      */
typedef signed short    H;          /* signed 16 bit integer     */
typedef signed long     W;          /* signed 32 bit integer     */
typedef signed long long D;         /* signed 64 bit integer     */

typedef unsigned char   UB;         /* unsigned 8 bit integer    */
typedef unsigned short  UH;         /* unsigned 16 bit integer   */
typedef unsigned long   UW;         /* unsigned 32 bit integer   */
typedef unsigned long long UD;      /* unsigned 64 bit integer   */

typedef B               VB;         /* variable data type (8 bit) */
typedef H               VH;         /* variable data type (16 bit) */
typedef W               VW;         /* variable data type (32 bit) */
typedef D               VD;         /* variable data type (64 bit) */

typedef void            *VP;         /* pointer to variable data type */
typedef void            (*FP)(void); /* program start address      */

typedef int             INT;         /* signed integer (CPU dependent) */
typedef unsigned int    UINT;        /* unsigned integer (CPU dependent) */

typedef INT             BOOL;        /* Bool value                  */

typedef W               FN;          /* function code                */
typedef W               ER;          /* error code                   */
typedef H               ID;          /* object ID (xxxid)            */
typedef UW              ATR;         /* attribute                    */
typedef UW              STAT;        /* object status                 */
typedef UW              MODE;        /* action mode                   */
typedef H               PRI;         /* task priority                 */
typedef UW              SIZE;        /* memory area size             */

typedef W               TMO;         /* time out                     */
typedef UW              RELTIM;      /* relative time                 */

typedef struct {
    UH      utime;          /* system clock                  */
    VH      _Hrsvl;         /* current date/time (upper)     */
    UW      ltime;          /* reserved                      */
    UW      /* current date/time (lower) */
} SYSTEM;

typedef INT             VP_INT;      /* integer or pointer to var. data */

typedef ER              ER_BOOL;     /* error code or bool value      */
typedef ER              ER_ID;       /* error code or object ID       */
typedef ER              ER_UINT;     /* error code or unsigned integer */
```

For the structures used in service calls, refer to the description of each service call.

6.1.5 Constants and Macros

(1) Configuration Constants

Configuration constants specify the kernel configuration information.

Some configuration constants are predetermined as the kernel specifications, and the others should be specified through the configurator. The former constants are defined in kernel.h, and the latter constants are defined in kernel_macro.h output from the configurator.

Table 6.2 Configuration Constants Defined in kernel.h

No.	Constant	Definition	Description
1	TMIN_TPRI	1	Minimum value of task priority
2	TMIN_MPRI	1	Minimum value of message priority
3	TKERNEL_MAKER	H'0115	Kernel maker code This value is same as parameter maker which is returned by a ref_ver service call.
4	TKERNEL_PRID	H'0012	Kernel ID This value is same as parameter prid which is returned by a ref_ver service call.
5	TKERNEL_SPVER	H'5402	ITRON specification version number This value is same as parameter spver which is returned by a ref_ver service call.
6	TKERNEL_PRVER	H'0100	Kernel version number This value is same as parameter prver which is returned by a ref_ver service call.
7	TKERNEL_PXVER	H'0100	Version number of the μ ITRON4.0 protection extension specification This value is same as parameter pxver which is returned by a ref_ver service call.
8	TBIT_TEXPTN	32	Number of task exception cause bits
9	TBIT_FLGPTN	32	Number of event flag bits
10	TMAX_MAXSEM	65535	Maximum number of semaphore resources

Table 6.3 Configuration Constants Defined in kernel_macro.h

No.	Constant	Related Configurator Setting	Description
1	TIC_NUME	CFG_TICNUME	Numerator of time tick cycle
2	TIC_DENO	CFG_TICDENO	Denominator of time tick cycle
3	TMAX_TPRI	CFG_MAXTSKPRI	Maximum value of task priority
4	TMAX_MPRI	CFG_MAXMSGPRI	Maximum value of message priority
5	TMAX_ACTCNT	CFG_MAXACTCNT	Maximum number of task initiation requests in a queue
6	TMAX_WUPCNT	CFG_MAXWUPCNT	Maximum number of task wakeup requests in a queue
7	TMAX_SUSCNT	CFG_MAXSUSCNT	Maximum number of nesting levels for task suspend requests
8	VTCFG_PROTMEM	CFG_PROTMEM	Memory object protection function selection 1 when CFG_PROTMEM is selected, or 0 when CFG_PROTMEM is not selected.
9	VTCFG_PAGESZ	CFG_PAGESZ	Default MMU page size 4096 when the memory object protection function is selected, or 0 when it is not selected.
10	VTCFG_TMRCLOCK	CFG_TMRCLOCK	Frequency input to the timer device
11	VTCFG_TIMINTNO	CFG_TIMINTNO	Standard timer driver interrupt number
12	VTCFG_KNLLVL	CFG_KNLLVL	Kernel interrupt mask level and timer interrupt level

(2) Error Codes

The error codes returned from service calls are defined in itron.h.

Reference: Section 17.2, Service Call Error Code List

The following macros are provided to manipulate error codes.

Table 6.4 Error Code Manipulating Macros (itron.h)

No.	Macro Name	Description
1	ER ercd = MERCD(ER ercd)	Returns the main error code for ercd.
2	ER ercd = SERCD(ER ercd)	Returns the suberror code for ercd. *

Note: For all errors returned from kernel service calls, the suberror code is set to -1.

(3) Macros for Size Calculation

The following macros are provided to calculate the size (bytes).

- Macros for calculating the area size to be specified on the application side (table 6.5)
- A macro for calculating the size to be used in the message buffer (table 6.6)
- Macros for calculating the size to be used in the resource pool

For the macros for resource pool size, refer to the following.

Reference: Section 13, Estimation of Resource Pool Size

Table 6.5 Macros for Calculating Area Size to be Specified on the Application Side (kernel_tsz.h)

Macro	Description
SIZE mbfsz = TSZ_MBFMB (UINT msgcnt, UINT msgsz)	Approximate size of the message buffer area that can hold the msgcnt number of msgsz-byte messages. Use this macro to estimate the approximate mbfsz value to be specified in cre_mbf or acre_mbf.
SIZE mbfsz = TSZ_MBF (UINT msgcnt, UINT msgsz)	Same as TSZ_MBFMB.
SIZE mpfsz = TSZ_MPF (UINT blkcnt, UINT blksz)	Size of the fixed-size memory pool area that can hold the blkcnt number of blksz-byte memory blocks. Use this macro to calculate the size of the fixed-size memory pool area when it should be allocated on the application side.
SIZE mplsz = TSZ_MPL (UINT blkcnt, UINT blksz)	Approximate size of the variable-size memory pool area that can hold the blkcnt number of blksz-byte memory blocks. Use this macro to estimate the approximate size of the variable-size memory pool area when it should be allocated on the application side.
SIZE mplsz = TSZ_MPP (UINT blkcnt, UINT memsz)	Approximate size of the protected memory pool area that can hold the blkcnt number of memsz-byte protected memory blocks.

Table 6.6 Macro for Calculating the Size to be Used in Message Buffer (kernel_tsz.h)

Macro	Description
SIZE size = VTSZ_MBFMSGMB(UINT msgsz)	Size of the message buffer area to be used when a msgsz-byte message is stored in the message buffer.

(4) Alignment Check Macros

The following macros are provided to check data alignment.

Table 6.7 Alignment Check Macros (itron.h)

No.	Macro Name	Description
1	BOOL align = ALIGN_VB(VP addr)	Returns TRUE (1) when data alignment at the addr address allows VB-type data access; otherwise, returns FALSE (0).
2	BOOL align = ALIGN_VH(VP addr)	Returns TRUE (1) when data alignment at the addr address allows VH-type data access; otherwise, returns FALSE (0).
3	BOOL align = ALIGN_VW(VP addr)	Returns TRUE (1) when data alignment at the addr address allows VW-type data access; otherwise, returns FALSE (0).
4	BOOL align = ALIGN_VD(VP addr)	Returns TRUE (1) when data alignment at the addr address allows VD-type data access; otherwise, returns FALSE (0).
5	BOOL align = ALIGN_VP(VP addr)	Returns TRUE (1) when data alignment at the addr address allows VP-type data access; otherwise, returns FALSE (0).

(5) Other Constants and Macros

The other constants and macros are described in the related service call descriptions.

6.2 Register Contents Guaranteed after Issuing Service Call

Some registers guarantee the contents after a service call is issued but some do not. This rule follows the Renesas C compiler. The details are shown below.

Table 6.8 Register Contents after Issuing Service Call

Register	Register State after Service Call Return
SR, R8 to R15, PR, GBR, MACH, MACL	The register contents are guaranteed. IMASK bits in SR are updated when service call chg_ims, ichg_ims, loc_cpu, iloc_cpu, unl_cpu, or iunl_cpu is issued.
R0	Normal termination (E_OK) or an error code is set.
R1 to R7	The register contents will not be guaranteed.
For DSP: DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	The register contents will be guaranteed in either one of the following states. <ul style="list-style-type: none">• When a service call is issued from a program with TA_COP0 attribute• When a service call is issued in dispatch-pended state *
For FPU: FPR0_BANK0 to FPR11_BANK0	(1) When FPSCR.FR = 0 The register contents will be guaranteed when a service call is issued in dispatch-pended state. * (2) When FPSCR.FR = 1 The register contents will be guaranteed in either one of the following states. <ul style="list-style-type: none">• When a service call is issued from a program with TA_COP1 TA_COP2 attribute• When a service call is issued in dispatch-pended state *
For FPU: FPR12_BANK0 to FPR15_BANK0, FPSCR, FPUL	The register contents will be guaranteed in either one of the following states. <ul style="list-style-type: none">• When a service call is issued from a program with TA_COP1 attribute• When a service call is issued in dispatch-pended state *
For FPU: FPR0_BANK1 to FPR11_BANK1	(1) When FPSCR.FR = 0 The register contents will be guaranteed in either one of the following states. <ul style="list-style-type: none">• When a service call is issued from a program with TA_COP1 TA_COP2 attribute• When a service call is issued in dispatch-pended state * (2) When FPSCR.FR = 1 The register contents will be guaranteed when a service call is issued in dispatch-pended state. *
For FPU: FPR12_BANK1 to FPR15_BANK1	The register contents will be guaranteed in either one of the following states. <ul style="list-style-type: none">• When a service call is issued from a program with TA_COP1 TA_COP2 attribute• When a service call is issued in dispatch-pended state *

Note: These registers are not accessed during the service call processing in the kernel.

6.3 Return Value of Service Call and Error Code

6.3.1 Overview

For service calls that have return values, a positive value or 0 (E_OK) indicates normal termination, and a negative value indicates an error code. However, for service calls that have a BOOL-type return value, this is not the case. The meaning of the return value at normal termination differs according to the service call; however, only E_OK is returned at normal termination for many service calls.

An error code consists of a main error code (lower 8 bits) and a suberror code (the remaining upper bits). The suberror code of this kernel is always set to -1.

The following macros are defined in standard header `itron.h`.

- `ER mercd = MERCD(ER ercd);` Extracts the main error code from the error code.
- `ER sercd = SERCD(ER ercd);` Extracts the suberror code from the error code.

6.3.2 Parameter Check Function

In this kernel, detection of parameter errors can be stopped. If the parameter check function is removed after debugging is completed, the overhead or code size can be reduced.

To remove the parameter check function, deselect `CFG_PARCHK` through the configurator.

6.3.3 Access Permission Check Function for Address Parameters

When the memory object protection function is selected, the address parameters such as packet addresses specified in service calls are checked whether their access permission is right. However, this check needs the processing equivalent to the `prb_mem` service call and causes a large overhead. When debugging has been properly performed and the access permission check is not necessary, this check can be omitted.

To remove the access permission check function for address parameters, deselect `CFG_MEMCHK`.

6.3.4 E_NOSPT Error

An E_NOSPT error will be returned if an unembedded service call is issued.

6.4 System State and Service Calls

Whether a service call can be issued depends on the system state.

6.4.1 CPU Exception Handler

The service calls that can be issued from the CPU exception handler are listed below.

- iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

No E_CTX error will be detected when a service call other than these is called from the CPU exception handler. In this case, correct operation cannot be guaranteed.

6.4.2 Task Context and Non-Task Context

(1) Special Service Calls

The following service calls can be issued in either a task context or a non-task context.

- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

(2) Service Calls Starting with sns

The service calls whose names start with "sns" can be issued in either a task context or a non-task context.

(3) Other Service Calls

The service calls whose names start with "i" are dedicated to non-task context, and the other service calls are for task context.

The service calls for task context are further classified into the following two types.

- (a) Service calls for which no corresponding service calls starting with "i" are provided (e.g. del_tsk; there is no idel_tsk)

If this type of service call is issued in a non-task context, an E_CTX error will be returned.

- (b) Service calls for which corresponding service calls starting with "i" are provided (e.g. act_tsk and iact_tsk)

In the kernel, the processing for a service call with "i" is the same as that for the corresponding service call without "i". Accordingly, when a service call starting with "i" is issued in a task context or when a service call without "i" is issued in a non-task context, no E_CTX error will be detected and the service call is processed correctly.

Note that this behavior is only for this version of kernel implementation, and it may change in a later version of the kernel.

It is recommended that any application should observe the rule that the service calls starting with "i" are for non-task context and the other service calls are for task context.

6.4.3 CPU-Locked State

Service calls that can be issued in the CPU-locked state are listed below. No E_CTX error is returned when a service call other than these is called in the CPU-locked state. In this case, correct system operation cannot be guaranteed. Note that, when a service call that shifts a task to the WAITING state is called, an E_CTX error is returned. Refer to "Error Code" in each service call description for this type of E_CTX error.

- ext_tsk (CPU-locked state will be canceled)
- exd_tsk (CPU-locked state will be canceled)
- sns_tex
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

6.4.4 Dispatch-Disabled State

When a service call that shifts a task to the WAITING state is issued in this state, an E_CTX error is returned. Refer to "Error Code" in each service call description for this type of E_CTX error.

6.4.5 When SR.IMASK is Modified to a Non-Zero Value through chg_ims in Task Context

When a service call that shifts a task to the WAITING state is issued in this state, an E_CTX error is returned. Refer to "Error Code" in each service call description for this type of E_CTX error.

6.5 Service Calls not in the μ ITRON4.0 Specification

Service calls whose name start with “v”, “iv”, or “V”, such as vset_tfl, are service calls that are not defined in the μ ITRON4.0 specification or the protection function extension of the μ ITRON4.0 specification.

The following "ixxx_yyy"-format service calls (starting with "i") are not defined in the μ ITRON4.0 specification. They are provided to enable the "xxx_yyy"-format service calls corresponding to the following service calls to be issued in a non-task context because the "xxx_yyy"-format service calls are defined to be issued only in a task context in the μ ITRON4.0 specification or the protection function extension of μ ITRON4.0 specification.

icre_tsk, iacre_tsk, ican_act, ista_tsk, ichg_pri, iget_pri, iref_tsk, iref_tst, ican_wup, isus_tsk, irsm_tsk, ifrsm_tsk, ideo_tsk, iref_tsk, icre_sem, iacre_sem, ipol_sem, iref_sem, icre_flg, iacre_flg, iclr_flg, ipol_flg, iref_flg, icre_dtq, iacre_dtq, iref_dtq, icre_mbx, iacre_mbx, isnd_mbx, iprv_mbx, iref_mbx, icre_mbf, iacre_mbf, ipsnd_mbf, iref_mbf, icre_mpf, iacre_mpf, ipget_mpf, irel_mpf, iref_mpf, icre_mpl, iacre_mpl, ipget_mpl, irel_mpl, iref_mpl, iset_tim, iget_tim, icre_cyc, iacre_cyc, ista_cyc, istp_cyc, iref_cyc, icre_alm, iacre_alm, ista_alm, istp_alm, iref_alm, ista_ovr, istp_ovr, iref_ovr, iget_did, ideo_inh, ichg_ims, iget_ims, ideo_svc, ical_svc, ideo_exc, iref_cfg, iref_ver, icre_mbp, iacre_mbp, iref_mbp

6.6 Service Call Description Form

Service calls are described in details as shown below in this section.

Section	Brief function description (Service call name)		
C-Language API:			
Service call issuing format			
Parameters:			
Type	Parameter name	Meaning	
Return Parameters:			
Type	Parameter name	Meaning	
Packet Structure:			
...		<--	See (1) below
Return Codes/Error Codes:			
Mnemonic	[Error type]	Meaning	<-- See (2) below
Function:			
Describes the function of the service call.			
Error Detection through CFG_MEMCHK:			
...		<--	See (2) below

Figure 6.1 Service Call Description Form

(1) Packet Structure

A packet structure is described in the following form.

typedef struct {				
ID	wtskid;	0	2	Wait task ID
UINT	semcnt;	+4	4	Current semaphore count
} T_RSEM;				
		↑	↑	↑
Offset from the beginning of the packet		Member size		Description of member

(2) Error Type

Errors are classified into the following types.

- [k]: Detected in all states.
- [p]: Detected only when CFG_PARCHK is selected through the configurator.
- [m]: Detected only when the memory object protection function (CFG_PROTMEM) and CFG_MEMCHK are selected through the configurator. For the error conditions, refer to "Error Detection through CFG_MEMCHK".

(3) Error Detection through CFG_MEMCHK

Describes the detailed error conditions for error type [m].

6.7 Task Management

Table 6.9 Service Calls for Task Management

Service Call¹		Description	System State²
			T/N/E/D/U/L/C
cre_tsk	[s]	Creates task	T/E/D/U
icre_tsk			N/E/D/U
acre_tsk		Creates task and assigns task ID automatically	T/E/D/U
iacre_tsk			N/E/D/U
del_tsk		Deletes task	T/E/D/U
act_tsk	[S]	Initiates task	T/E/D/U
iact_tsk	[S]		N/E/D/U
can_act	[S]	Cancels task initiation request	T/E/D/U
ican_act			N/E/D/U
sta_tsk		Initiates task and specifies start code	T/E/D/U
ista_tsk			N/E/D/U
ext_tsk	[S]	Exits current task	T/E/D/U/L
exd_tsk	[S]	Exits and deletes current task	T/E/D/U/L
ter_tsk	[S]	Forcibly terminates a task	T/E/D/U
chg_pri	[S]	Changes task priority	T/E/D/U
ichg_pri			N/E/D/U
get_pri	[S]	Refers to task priority	T/E/D/U
iget_pri			N/E/D/U
ref_tsk		Refers to task state	T/E/D/U
iref_tsk			N/E/D/U
ref_tst		Refers to task state (simple version)	T/E/D/U
iref_tst			N/E/D/U
vchg_tmd		Changes task execution mode	T/E/D/U

Notes: 1. [S]: Standard profile service calls

[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context

N: Can be called in a non-task context

E: Can be called in dispatch-enabled state

D: Can be called in dispatch-disabled state

U: Can be called in CPU-unlocked state

L: Can be called in CPU-locked state

C: Can be called from CPU exception handler

Table 6.10 Task Management Specifications

Item	Description
Task ID	1 to CFG_MAXTSKID (32767 max.)
Task priority	1 to CFG_MAXTSKPRI (255 max.) *
Maximum count of task initiation requests	CFG_MAXACTCNT (32767 max.)
Domain ID	Kernel domain: TDOM_KERNEL (-1) User domain: 1 to 31
Task attribute	TA_HLNG: The task is written in a high-level language. TA_ASM: The task is written in assembly language. TA_ACT: The task makes a transition to the READY state after the task has been created. TA_COP0: The task uses the DSP. TA_COP1: The task uses register bank 0 in the FPU. TA_COP2: The task uses register bank 1 in the FPU. TA_DOM(domid): The task is assigned to the domain indicated by domid.

Note: This value is the same as TMAX_TPRI defined in kernel_macro.h.

6.7.1 Create Task (cre_tsk, icre_tsk, acre_tsk, iacre_tsk)

C-Language API:

```
ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk);
ER ercd = icre_tsk(ID tskid, T_CTSK *pk_ctsk);
ER_ID tskid = acre_tsk(T_CTSK *pk_ctsk);
ER_ID tskid = iacre_tsk(T_CTSK *pk_ctsk);
```

Parameters:

T_CTSK	*pk_ctsk	Pointer to the packet where task creation information is stored
 <cre_tsk, icre_tsk >		
ID	tskid	Task ID

Return Parameters:

<cre_tsk, icre_tsk >		
ER	ercd	Normal termination (E_OK) or error code
<acre_tsk, iacre_tsk >		
ER_ID	tskid	Created task ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    tskatr;    0    4    Task attribute
    VP_INT exinf;    +4    4    Extended information
    FP    task;    +8    4    Task start address
    PRI    itskpri;    +12    2    Priority at task initiation
    SIZE    stksz;    +16    4    Task stack size
    VP    stk;    +20    4    Start address of task stack area
    SIZE    sstksz;    +24    4    System stack size for the task
    VP    sstk;    +28    4    Start address of the system stack
                                area for the task
    UW    inifpscr;    +32    4    Initial FPSCR value
}T_CTSK;
```

Error Codes:

E_RSATR	[p]	Reserved attribute <ul style="list-style-type: none"> (1) The bits other than TA_COP0, TA_COP1, TA_COP2, TA_ASM, TA_ACT, and upper eight bits in tskatr are not 0. (2) TA_COP0 is specified for tskatr while CFG_DSP is not selected. (3) TA_COP1 is specified for tskatr while CFG_FPU is not selected. (4) TA_COP2 is specified for tskatr while TA_COP1 is not specified. (5) Both TA_COP0 and TA_COP1 are specified for tskatr. (6) The upper eight bits of tskatr are neither a value within the range from 0 to 31 nor H'ff.
E_PAR	[p]	Parameter error <ul style="list-style-type: none"> (1) itskpri ≤ 0 (2) itskpri > CFG_MAXTSKPRI (3) stksz = 0 or sstksz = 0 while the task is assigned to a user domain. (4) stksz = 0 while the task is assigned to the kernel domain. (5) pk_ctsk is not a 4-byte boundary address. (6) task is an odd value. (7) stk is neither NULL nor a 4-byte boundary address. (8) sstk is neither NULL nor a 4-byte boundary address while the task is assigned to a user domain. (9) stk = NULL and stksz > CFG_SYSPOOLSZ while the task is assigned to a user domain. (10) sstk = NULL and sstksz > (CFG_RESPOOLSZ - VTSZ_RPLMB) while the task is assigned to a user domain. (11) stk = NULL and (stksz + sstksz) > (CFG_RESPOOLSZ - VTSZ_RPLMB) while the task is assigned to a kernel domain.
E_ID	[p]	Invalid ID number <ul style="list-style-type: none"> (1) tskid ≤ 0 (2) tskid > CFG_MAXTSKID
E_NOMEM	[k]	Insufficient memory <ul style="list-style-type: none"> (1) Insufficient space in the system pool (2) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_tsk)
E_OBJ	[k]	Invalid object state <ul style="list-style-type: none"> (1) Task specified by tskid already exists.

Function:

Each service call creates a task. The created task make a transition to the DORMANT state when the TA_ACT attribute is not specified, or to the READY state when the TA_ACT attribute is specified.

The processing that is performed at task creation is listed in table 6.11.

Table 6.11 Processing to be Performed at Task Creation

Contents

Clears the number of task initiation requests in the queue.
Resets the task state so that the task exception routine is not defined.
Resets the task state so that the upper-limit processor time is not specified.
Assigns a stack.
Clears the data in the performance counter and newly starts accumulation.

The following describes the meaning of the parameters.

(1) tskid

In service calls cre_tsk and icre_tsk, a value within the range from 1 to CFG_MAXTSKID can be specified for tskid. Service calls acre_tsk and iacre_tsk search for an unused task ID, create a task for the task ID with the contents specified by pk_ctsk, and return the ID as a return parameter.

(2) tskatr

Specify the logical OR of the following values for tskatr.

(a) Language

Specify either one of the following values.

- TA_HLNG (H'00000000): High-level language
- TA_ASM (H'00000001): Assembly language

(b) Task initiation

Specify TA_ACT to make the target task to enter the READY state. When TA_ACT is not specified, the task enters the DORMANT state.

- TA_ACT (H'00000002): The task makes a transition to the READY state after the task has been created.

- (c) Using a microcomputer with an on-chip DSP (when CFG_DSP is selected)
Specify TA_COP0 to use the DSP.
— TA_COP0 (H'00000100): The task uses the DSP.
- (d) Using a microcomputer with an on-chip FPU (when CFG_FPU is selected)
Specify TA_COP1 to use the FPU for floating-point operations. Specify TA_COP2 in addition to TA_COP1 when using both banks of the FPU for matrix operations.
— TA_COP1 (H'00000200): The task uses FPU register bank 0 (FPR0_BANK0 to FPR15_BANK0) and FPUL.
— TA_COP2 (H'00000400): The task uses FPU register bank 1 (FPR0_BANK1 to FPR15_BANK1).
To specify TA_COP2, be sure to specify TA_COP1 together; otherwise, an E_RSATR error is returned.
Also refer to description (7), inifpscr.

(e) SR at initiation and assigned domain

The following attribute can be specified (OR) to assign the task to a domain.

TA_DOM(domid)

Here, the following can be specified for domid.

- (i) 1 to 31: The task is assigned to the user domain of the specified domid.
- (ii) TDOM_SELF (0): The task is assigned to the domain of the caller. Note that when this service call is issued from an extended service call or trap routine being executed in a task context, the created task is assigned to the domain of the task that has called the extended service call or trap routine (the same domain ID that can be checked by issuing get_did from the extended service call or trap routine). When the service call is issued in a non-task context, the created task is assigned to the kernel domain.
- (iii) TDOM_KERNEL(-1): The task is assigned to the kernel domain.
- When TA_DOM(domid) is omitted, attribute (ii) is assumed.

The task in the kernel domain is executed in privileged mode (SR.MD = 1), and the task in a user domain is executed in user mode (SR.MD = 0).

When the memory object protection function is not selected:

Differences between user domain IDs are ignored, and operations only depend on the classification between the kernel domain and the user domain.

To be more specific, the classification between the kernel domain and the user domain only causes the following differences.

- The MD bit in SR at initiation is 1 (privileged mode) for the task in the kernel domain or 0 (user mode) for the task in the user domain.

— The stack to be used differs (refer to description (6), stksz, stk, sstksz, sstk).

(3) exinf

Parameter exinf can be widely used by the user, for example, to set information concerning tasks to be created. exinf is passed to the task as a parameter when the task is initiated through act_tsk.

(4) task

Specify the task start address.

(5) itskpri

Specify 1 to CFG_MAXTSKPRI as the task priority at initiation.

(6) stksz, stk, sstksz, sstk

These parameters specify stacks.

Note that stksz and sstksz are rounded up to multiples of four. In the following description, stksz and sstksz indicate multiples of four after being rounded up.

(a) Task in the user domain

A task in the user domain has a system stack in addition to a usual stack used for task execution. The system stack is used by the kernel and extended service call and trap routines called from the task to store the task context.

A stack is allocated to a stksz-byte area starting from address stk, and a system stack is allocated to a sstksz-byte area starting from address sstk.

When the memory object protection function is not selected:

A stack must be allocated in an area that can be accessed in user mode. Even if this rule is violated, the kernel does not check it. In this case, a CPU exception occurs when the stack is accessed in user mode.

A system stack must be allocated in an area that cannot be accessed in user mode. Even if this rule is violated, the kernel does not check it. In this case, the system stack can be accessed in user mode, which increases the risk of damaging the stack.

When the memory object protection function is selected:

A stack must be allocated in an area that can be read or written to from the domain including the target task. If this rule is violated, an E_MACV error will be returned.

A system stack must be allocated in an MMU non-mapped area that cannot be accessed in user mode. No memory object can be used as a system stack. If this rule is violated, an E_MACV error will be returned.

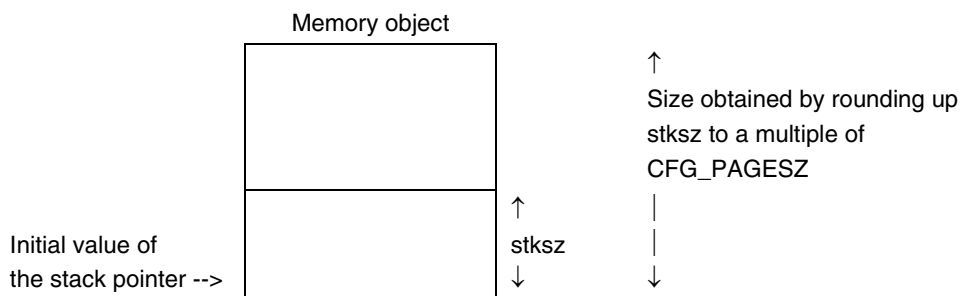
When NULL is specified as stk, the kernel allocates a stack in the system pool. At this time, the kernel consumes an area in the resource pool to manage the allocated stack. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.2 (1), Task
System pool consumption → Section 14.2 (1), When task is created

When the memory object protection function is selected:

The stack area allocated in the system pool by the kernel is a memory object having the following attributes.

(1) Size: stksz (stack size) is rounded up to a multiple of CFG_PAGESZ.



(2) Page size: 4 kbytes

(3) Domain: Domain where the target task is assigned

(4) Memory attribute: TA_RW|TA_CACHE|TA_WBACK

(5) Access permission vector: TACT_PRW(domid)

(domid is the ID of the domain where the target task is assigned)

When NULL is specified as sstk, the kernel allocates a system stack in the resource pool. For details, refer to the following.

Reference: Section 13.2.2 (1), Task

(b) Task in the kernel domain

Unlike a task in the user domain, a task in the kernel domain has only one stack and sstk is ignored.

The stack is used to execute the task and to store the task context by the kernel and the extended service call and trap routines called from the task.

A stack is allocated to a (stksz + stksz)-byte area starting from address stk.

When the memory object protection function is not selected:

A stack must be allocated in an area that cannot be accessed in user mode. Even if this rule is violated, the kernel does not check it. In this case, the privileged stack can be accessed in user mode, which increases the risk of damaging the stack.

When the memory object protection function is selected:

A stack must be allocated in an MMU non-mapped area that cannot be accessed in user mode. No memory object can be used as a stack. If this rule is violated, an E_MACV error will be returned.

When NULL is specified as stk, the kernel allocates a stack in the resource pool. For details, refer to the following.

Reference: Section 13.2.2 (1), Task

(7) inifpscr

inifpscr is a parameter not specified in the μ ITRON specification.

It is valid only when CFG_FPU is selected and the TA_COP1 attribute is specified. In other cases, it is ignored.

inifpscr specifies the FPSCR value at initiation. The kernel sets the inifpscr value in FPSCR without checking an error in the inifpscr value.

Also refer to the following.

Reference: Section 15, Notes on FPU

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for pk_ctsk, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_ctsk
 - size = sizeof(T_CTSK)
 - domid = Domain of the caller
 - pmmode = TPM_READ
- (2) The domain of the target task to be created does not have a read access permission for pk_ctsk->task, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_ctsk->task
 - size = 1
 - domid = Domain where the target task is assigned
 - pmmode = TPM_READ
- (3) When a task is to be created in a user domain, the domain of the task does not have a read/write access permission for the stack area specified by stk, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = stk
 - size = stksz
 - domid = Domain where the target task is assigned
 - pmmode = TPM_READ|TPM_WRITE
- (4) The system stack area specified by sstk for the task to be created in a user domain or the stack area specified by stk for the task to be created in the kernel domain is not allocated in an MMU non-mapped area that cannot be accessed in user mode. To be more specific, the specified area is not included in any of the following areas.
 - An area in the on-chip memory virtual address space specified through the configurator when CFG_IRAM specifies an MMU non-mapped area that cannot be accessed in user non-DSP mode
 - P1 or P2 area

6.7.2 Delete Task (del_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $tskid \leq 0$ (2) $tskid > \text{CFG_MAXTSKID}$
E_CTX	[k]	Context error (1) Called in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is not in DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Service call del_tsk deletes the task specified by parameter tskid. The deleted task makes a transition to the NON-EXISTENT state.

The areas allocated in the system pool and resource pool when the target task was created are released.

6.7.3 Initiate Task (act_tsk, iact_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_QOVR	[k]	Queuing overflow (1) The number of initiation requests queued for the task has reached CFG_MAXACTCNT.

Function:

Each service call initiates the task specified by parameter tskid. The initiated task makes a transition from the DORMANT state to the READY state.

The processing that is performed during task initiation is listed in table 6.12.

Table 6.12 Processing to be Performed during Task Initiation

Contents

Initializes base priority and current priority of the task.
Clears the number of initiation requests in the queue.
Clears the number of suspend request nesting levels.
Clears pended exception causes.
Sets task exception processing disabled state.
Clears the flag pattern of the task event flag.

By specifying tskid = TSK_SELF (0), the current task is specified.

Extended information of the task specified at task creation will be passed to the task as the parameter.

When the task is not in the DORMANT state, up to the CFG_MAXACTCNT number of task initiation requests through these service calls can be kept waiting.

6.7.4 Cancel Task Initiation Request (can_act, ican_act)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER_UINT	actcnt	Number of queued initiation requests (positive value or 0), or error code
---------	--------	--

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

The number of initiation requests queued for the task specified by tskid is determined, the result is returned as the return parameter, and at the same time the initiation requests are all cancelled.

By specifying tskid = TSK_SELF (0), the current task is specified.

A task in the DORMANT state can also be specified; in this case the return parameter is 0.

6.7.5 Initiate Task and Specify Start Code (sta_tsk, ista_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
VP_INT	stacd	Task start code

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $tskid \leq 0$ (2) $tskid > CFG_MAXTSKID$
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is not in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call initiates the task indicated by parameter tskid. The initiated task makes a transition from the DORMANT state to the READY state. At this time, the processing to be performed during task initiation (table 6.12) is performed. The task start code indicated by parameter stacd will be passed to the initiated task as the parameter.

6.7.6 Exit Current Task (ext_tsk) and Exit and Delete Current Task (exd_tsk)

C-Language API:

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

Parameters:

None

Return Parameters:

Execution does not return to the caller of these tasks.
In addition, if the following error occurs, control is passed to the system down routine.

E_CTX	[k]	Context error
		(1) Called in a non-task context.

Function:

Service call ext_tsk terminates the current task normally. After the execution of service call ext_tsk, the current task makes a transition from the RUNNING state to the DORMANT state. When an initiation request is queued, service call ext_tsk terminates the current task and then restarts the task.

The processing that is performed at task termination is listed in table 6.13.

Table 6.13 Processing to be Performed at Task Termination

Contents

Unlocks the mutex locked by the task.
Releases upper-limit processor time.

Service call exd_tsk terminates the current task normally and deletes it. After the execution of service call exd_tsk, the current task makes a transition from the RUNNING state to the NON-EXISTENT state.

Service calls ext_tsk and exd_tsk do not release the resources acquired by the task (such as semaphores and memory blocks) except for mutexes. Therefore, the user must issue service calls to release resources before exiting the task.

In service call exd_tsk, the areas acquired from the system pool and resource pool at task creation.

Service calls ext_tsk and exd_tsk can be issued while task dispatch is disabled, the CPU is locked, or the interrupt mask has been changed to a non-zero value through chg_ims. After either of the

service calls is issued in any one of these states, the dispatch-disabled state or CPU-locked state is cancelled and the interrupt mask is restored to 0.

Note that when execution returns from the task start function, the same operation as for service call `ext_tsk` will be performed.

6.7.7 Forcibly Terminate Task (ter_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $tskid \leq 0$ (2) $tskid > \text{CFG_MAXTSKID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_ILUSE	[k]	Illegal use of service call (1) The current task is specified as the target task.
E_OBJ	[k]	Invalid object state (1) Task specified by <code>tskid</code> is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by <code>tskid</code> does not exist.

Function:

Service call `ter_tsk` forces a task specified by `tskid` to terminate execution. The terminated task enters the DORMANT state. At this time, the processing shown in table 6.13 is performed.

When an initiation request is queued, the processing to be performed during task initiation is performed, and the target task enters the READY state.

A termination request through this service call is delayed in the following case:

- If the task specified by `tskid` masks forcible termination requests by service call `vchg_tmd`

Service call `ter_tsk` does not automatically release the resources acquired by the task (such as semaphores and memory blocks) except for the mutexes. Therefore, the user must issue service calls to release the resources before issuing service call `ter_tsk`.

6.7.8 Change Task Priority (chg_pri, ichg_pri)

C-Language API:

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

Parameters:

ID	tskid	Task ID
PRI	tskpri	Base priority of task

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) tskpri < 0 (2) tskpri > CFG_MAXTSKPRI
E_ID	[p]	Invalid ID number (1) tskpri < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_ILUSE	[k]	Illegal use of service call (1) Ceiling priority is exceeded.
E_OBJ	[k]	Invalid object state (1) Task is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call changes the base priority of the task specified by parameter tskid to the value specified by parameter tskpri. By specifying tskid = TSK_SELF (0), the current task is specified.

Specifying tskpri = TPRI_INI (0) returns the task priority to the initial priority that was specified at task creation.

When the target task locks mutexes or waits for locking mutexes, an E_ILUSE is returned if the specified priority is higher than the ceiling priority of any one of the mutexes.

When the target task does not lock mutexes, the current priority is also changed to the tskpri value in addition to the base priority.

The base priority changed by the service calls is valid until the task is terminated or until a priority changing service call is issued again. When a task makes a transition to the DORMANT state, the

base priority before termination becomes invalid. When the task is initiated again, the base priority returns to the initial task priority specified at task creation.

If the task specified by `tskid` is in a `WAITING` state and `TA_TPRI` is specified for the object attribute, the wait queue may be changed by the service calls and as a result, the task at the head of the wait queue may be released from the `WAITING` state.

6.7.9 Refer to Task Priority (get_pri, iget_pri)

C-Language API:

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

Parameters:

ID	tskid	Task ID
PRI	*p_tskpri	Pointer to the area where the current priority of the target task is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
PRI	*p_tskpri	Pointer to the area where the current priority of the target task is stored

Error Codes:

E_PAR	[p]	Parameter error (1) p_tskpri is not a 2-byte boundary address.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the current priority of the task specified by parameter tskid, and returns it to the area indicated by parameter p_tskpri. By specifying tskid = TSK_SELF (0), the current task is specified.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_tskpri, which means that an error will be returned if prb_mem is issued with the following parameters.
- base = p_tskpri
 - size = sizeof(PRI)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.7.10 Refer to Task State (ref_tsk, iref_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
T_RTSK	*pk_rtsk	Pointer to the packet where the task state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RTSK	*pk_rtsk	Pointer to the packet where the task state is stored

Packet Structure:

```
typedef struct {  
    STAT    tskstat;    0    4    Task state  
    PRI     tskpri;     +4   2    Current priority of the task  
    PRI     tskbpri;    +6   2    Base priority of the task  
    STAT    tskwait;    +8   4    Wait cause  
    ID      wobjid;     +12  2    Wait object ID  
    TMO     lefttmo;    +16  4    Time to timeout  
    UINT    actcnt;     +20  4    Number of queued initiation requests  
    UINT    wupcnt;     +24  4    Number of queued wakeup requests  
    UINT    suscnd;     +28  4    Suspend request nest count  
    MODE    tskmode     +32  4    Task execution mode  
    FLGPTN  tflptn;     +36  4    Current task event flag value  
    ID      domid;      +40  2    ID of the domain where the task is  
                                   assigned  
}T_RTSK;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rtsk is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the task indicated by parameter `tskid`. By specifying `tskid = TSK_SELF (0)`, the current task is specified.

The following values are returned to the area indicated by `pk_rtsk`. Note that data with an asterisk (*) is invalid when the task is in the DORMANT state. If referenced information is related to a function that is not installed, the referenced information will be undefined.

- `tskstat`

Indicates the current task state. The following values are returned.

- `TTS_RUN (H'00000001)`: RUNNING state
- `TTS_RDY (H'00000002)`: READY state
- `TTS_WAI (H'00000004)`: WAITING state
- `TTS_SUS (H'00000008)`: SUSPENDED state
- `TTS_WAS (H'0000000c)`: WAITING-SUSPENDED state
- `TTS_DMT (H'00000010)`: DORMANT state

- `tskpri`

Indicates the current task priority. When the task is in the DORMANT state, the initial priority of the task is returned.

- `tskbpri`

Indicates the base priority of the task. When the task is in the DORMANT state, the initial priority of the task is returned.

- `tskwait*`

Valid only when `TTS_WAI` or `TTS_WAS` is returned to `tskstat`. The following values are returned.

- `TTW_SLP (H'00000001)`: WAITING state caused by `slp_tsk` or `tslp_tsk`
- `TTW_DLY (H'00000002)`: WAITING state caused by `dly_tsk`
- `TTW_SEM (H'00000004)`: WAITING state caused by `wai_sem` or `twai_sem`
- `TTW_FLG (H'00000008)`: WAITING state caused by `wai_flg` or `twai_flg`
- `TTW_SDTQ (H'00000010)`: WAITING state caused by `snd_dtq` or `tsnd_dtq`
- `TTW_RDTQ (H'00000020)`: WAITING state caused by `rcv_dtq` or `trcv_dtq`
- `TTW_MBX (H'00000040)`: WAITING state caused by `rcv_mbx` or `trcv_mbx`
- `TTW_MTX (H'00000080)`: WAITING state caused by `loc_mtx` or `tloc_mtx`
- `TTW_SMBF (H'00000100)`: WAITING state caused by `snd_mbf` or `tsnd_mbf`
- `TTW_RMBF (H'00000200)`: WAITING state caused by `rcv_mbf` or `trcv_mbf`
- `TTW_MPF (H'00002000)`: WAITING state caused by `get_mpf` or `tget_mpf`
- `TTW_MPL (H'00004000)`: WAITING state caused by `get_mpl` or `tget_mpl`
- `TTW_TFL (H'00008000)`: WAITING state caused by `vwai_tfl` or `vtwai_tfl`
- `TTW_MBP (H'00020000)`: WAITING state caused by `rcv_mbp` or `trcv_mbp`

- wobjid*
Valid only when TTS_WAI or TTS_WAS is returned to tskstat and the waiting target object ID is returned.
- lefttmo*
The time until the target task times out is returned. Note that when the target task is in the WAITING state according to service call dly_tsk, the value is undefined.
- actcnt*
The current initiation request queue count is returned.
- wupcnt*
The current wakeup request queue count is returned.
- suscnt*
The current suspend request nesting count is returned.
- tskmode*
tskmode is a parameter not defined in the μ ITRON specification.
tskmode indicates the task execution mode set through service call vchg_tmd, and whether there is a request that is delayed by service call vchg_tmd.
The following value is returned to tskmode.
 - ECM_SUS (H'00000001): A suspend request is masked
 - ECM_TER (H'00000002): A forcible termination request is masked
 - PND_SUS (H'00000004): A suspend request is delayed
 - PND_TER (H'00000008): A forcible termination request is delayed
- tflptn*
tflptn is a parameter not defined in the μ ITRON specification.
The current task event flag value is returned. However, if the task event flag function was not installed at system creation, an undefined value is returned.
- domid
domid is a parameter not specified in the μ ITRON specification.
The ID of the domain where the target task is assigned is returned through domid.
TDOM_KERNEL (-1) is returned when the task is in the kernel domain.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rtsk, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rtsk
- size = sizeof(T_RTSK)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.7.11 Refer to Task State (Simple Version) (ref_tst, iref_tst)

C-Language API:

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

Parameters:

ID	tskid	Task ID
T_RTST	*pk_rtst	Pointer to the packet where the task state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RTST	*pk_rtst	Pointer to the packet where the task state is stored

Packet Structure:

```
typedef struct {  
    STAT  tskstat;      0    4    Task state  
    STAT  tskwait;     +4    4    Wait cause  
}T_RTST;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rtst is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the task indicated by parameter tskid. By specifying tskid = TSK_SELF (0), the current task is specified.

The obtained values are returned to the area indicated by pk_rtst. The members of pk_rtst are the same as those with the same names in pk_rtsk, which will be returned when ref_tsk is issued. For details, refer to the description of ref_tsk.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for pk_rtsk, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rtst
- size = sizeof(T_RTST)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.7.12 Change Task Execution Mode (vchg_tmd)

C-Language API:

```
ER ercd = vchg_tmd(MODE tmd);
```

Parameters:

UINT	tmd	Task execution mode to change
------	-----	-------------------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) tmd is invalid.
E_CTX	[k]	Context error (1) Called in a non-task context

Function:

System call vchg_tmd changes the execution mode of the current task. A mask for requests from other tasks can be specified in tmd as the task execution mode.

- ECM_SUS (H'00000001): Suspend request is masked
- ECM_TER (H'00000002): Forcible termination request is masked

When the suspend request is masked, even if service call sus_tsk or isus_tsk is issued, its request is delayed until the mask is cancelled through service call vchg_tmd.

When the forcible termination request is masked, even if service call ter_tsk is called, its request is delayed until the mask is cancelled through service call vchg_tmd.

The task execution mode is not changed by an extended service call or a return from it, or by an initiation of a task exception processing routine or a return from it.

Delays of suspend requests and forcible termination requests can be referenced through service calls ref_tsk and iref_tsk.

6.8 Task Synchronization

Table 6.14 Service Calls for Task Synchronization

		System State ²
Service Call ¹	Description	T/N/E/D/U/L/C
slp_tsk [S]	Shifts current task to the WAITING state	T/E/U
tslp_tsk [S]	Shifts current task to the WAITING state with timeout function	T/E/U
wup_tsk [S]	Wakes up task	T/E/D/U
iwup_tsk [S]		N/E/D/U
can_wup [S]	Cancels wakeup request	T/E/D/U
ican_wup		N/E/D/U
rel_wai [S]	Cancels the WAITING state forcibly	T/E/D/U
irel_wai [S]		N/E/D/U
sus_tsk [S]	Shifts to the SUSPENDED state	T/E/D/U
isus_tsk		N/E/D/U
rsm_tsk [S]	Resumes the execution of a task in the SUSPENDED state	T/E/D/U
irsm_tsk		N/E/D/U
frsm_tsk [S]	Forcibly resumes the execution of a task in the SUSPENDED state	T/E/D/U
ifrm_tsk		N/E/D/U
dly_tsk [S]	Delays the current task	T/E/U
vset_tfl	Sets the task event flag	T/E/D/U
ivset_tfl		N/E/D/U
vclr_tfl	Clears the task event flag	T/E/D/U
ivclr_tfl		N/E/D/U
vwai_tfl	Waits for the task event flag	T/E/U
vpol_tfl	Polls and waits for the task event flag	T/E/D/U
vtwai_tfl	Waits for the task event flag with timeout function	T/E/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.15 Task Synchronization Specifications

Item	Description
Maximum number of task wakeup requests	CFG_MAXWUPCNT (32767 max.)
Maximum number of nesting levels for task suspend requests	CFG_MAXSUSCNT (32767 max.)
Number of task event flag bits	32 bits (lower 16 bits are reserved for future expansion)
Initial value of task event flag	0
Wait condition of task event flag	OR wait

6.8.1 Sleep Task (slp_tsk, tslp_tsk)

C-Language API:

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

Parameters:

<tslp_tsk>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) $tmout \leq -2$
E_CTX	[k]	Context error (1) Called in the dispatch-pended state
E_RLWAI	[k]	WAITING state is forcibly cancelled (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to the WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Timeout

Function:

Each service call shifts the current task to the wakeup WAITING state. However, if wakeup requests are queued for the current task, the wakeup request count is decremented by one and task execution continues. The WAITING state is cancelled by service call wup_tsk or iwup_tsk.

Parameter tmout specified by service call tslp_tsk specifies the timeout period. If a positive value is specified for parameter tmout, the WAITING state is released and error code E_TMOUT is returned when the tmout period has passed without the wait release conditions being satisfied.

If $tmout = TMO_POL$ (0) is specified, the task continues execution by decrementing the wakeup request count by one if the count is a positive value. If the wakeup request count is 0, error code E_TMOUT is returned.

If $tmout = TMO_FEVR$ (-1) is specified, the same operation as for service call slp_tsk will be performed. In other words, timeout will not be monitored.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7\text{ffffff}/CFG_TICDENO$. If a value larger than this is specified, operation is not guaranteed.

6.8.2 Wake up Task (wup_tsk, iwup_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_QOVR	[k]	Queuing overflow (1) The number of wakeup requests queued for the task has reached CFG_MAXWUPCNT.

Function:

Each service call releases a task from the WAITING state after the task was assigned to the WAITING state by service call slp_tsk or tslp_tsk. If the target task did not enter the WAITING state by service call slp_tsk or tslp_tsk, up to the CFG_MAXWUPCNT number of requests to wake up a task can be kept being pending.

By specifying tskid = TSK_SELF (0), the current task is specified.

6.8.3 Cancel Wakeup Request (can_wup, ican_wup)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER_UINT	wupcnt	Number of queued task wakeup requests (0 or a positive value) or error code
---------	--------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call obtains the number of wakeup requests queued for the task specified by tskid, returns the result as a return parameter, and invalidates all of those requests.

By specifying tskid = TSK_SELF (0), the current task is specified.

6.8.4 Cancel WAITING State Forcibly (rel_wai, irel_wai)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER_UINT	ercd	Normal termination (E_OK) or error code
---------	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

When the task specified by tskid is in some kind of WAITING state (not including a SUSPENDED state), it is forcibly cancelled. E_RLWAI is returned as the error code for the task for which the WAITING state is cancelled by service call rel_wai or irel_wai.

When the target task is executing an extended service call or trap routine, the target task is shifted to the WAITING-disabled state. The WAITING-disabled state is cancelled when all processing of the extended service call or trap routine called from the target task is completed, that is, the target task is in the WAITING-disabled state only during execution of the extended service call or trap routine.

In the WAITING-disabled state, if a service call causing a transition to the WAITING state is issued and the transition condition is satisfied, an E_RLWAI error will be generated.

By specifying tskid = TSK_SELF (0), the current task is specified.

If service call rel_wai or irel_wai is issued for a task in a WAITING-SUSPENDED state, the task enters the SUSPENDED state. After that, if service call rsm_tsk, irsm_tsk, frsm_tsk, or ifrsm_tsk is issued and the SUSPENDED state is cancelled, E_RLWAI is returned as the error code for the task.

For canceling SUSPENDED state, rsm_tsk, irsm_tsk, frsm_tsk or ifrsm_tsk should be used.

6.8.5 Suspend Task (sus_tsk, isus_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_CTX	[k]	Context error (1) A task being executed was specified in dispatch-pended state.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_QOVR	[k]	Queuing overflow (1) The number of suspend requests queued for the task has reached CFG_MAXSUSCNT.

Function:

Each service call suspends execution of the task specified by tskid and shifts the task to the SUSPENDED state. If the specified task is in the WAITING state, the task shifts to the WAITING-SUSPENDED state.

By specifying tskid = TSK_SELF (0), the current task is specified.

The SUSPENDED state can be cancelled by service call rsm_tsk, irsm_tsk, frsm_tsk, or ifrsm_tsk.

Requests to suspend a task by service calls sus_tsk and isus_tsk are nested. Up to the CFG_MAXSUSCNT number of requests can be kept being pended.

When the task specified by tskid masks the suspend request by service call vchg_tmd, the task enters the SUSPENDED state immediately after the suspend request mask is cancelled by service call vchg_tmd (by specifying tmd = 0).

Delayed requests to suspend a task can be cancelled by service call `rsm_tsk`, `irms_tsk`, `frsm_tsk`, or `ifrsn_tsk`. Therefore, tasks are suspended if there are one or more delayed suspend requests when the delay is canceled.

6.8.6 Resume Task (rsm_tsk, irsm_tsk) and Resume Task Forcibly (frsm_tsk, ifrsm_tsk)

C-Language API:

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

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $tskid \leq 0$ (2) $tskid > \text{CFG_MAXTSKID}$
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state. (2) Task specified by tskid is not in the SUSPENDED state. (3) Task specified by tskid is the current task.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call releases the task specified by parameter tskid from the SUSPENDED state. Service calls rsm_tsk and irsm_tsk decrement, by one, the number of nested requests to suspend the task specified by tskid, and release the task from the SUSPENDED state when the number of the nested requests becomes 0. Service calls frsm_tsk and ifrsm_tsk modify the number of nested requests to 0, and release the task from the SUSPENDED state. When the task is in the WAITING-SUSPENDED state, the task is shifted to the WAITING state.

6.8.7 Delay Task (dly_tsk)

C-Language API:

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

Parameters:

RELTIM	dlytim	Delayed time
--------	--------	--------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_CTX	[k]	Context error (1) Called in the dispatch-delayed state
E_RLWAI	[k]	WAITING state is forcibly cancelled. (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.

Function:

The current task is shifted from the RUNNING state to a timed WAITING state, and waits until the time specified by dlytim has expired. When the time specified by dlytim has elapsed, the state of the current task is shifted to the READY state. The current task is put into a WAITING state even if dlytim = 0 is specified.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for dlytim is $H'ffffff/CFG_TICDENO$. If a value larger than this is specified, operation is not guaranteed.

This service call differs from service call tslp_tsk in that it ends normally when execution is terminated after being delayed by the amount of time specified by dlytim. In addition, even if service call wup_tsk or iwup_tsk is executed during the delay, the WAITING state is not cancelled. The WAITING state is cancelled before the delay time has elapsed only when service call rel_wai, irel_wai, or ter_tsk is issued.

6.8.8 Set Task Event Flag (vset_tfl, ivset_tfl)

C-Language API:

```
ER ercd = vset_tfl(ID tskid, FLGPTN setptn);  
ER ercd = ivset_tfl(ID tskid, FLGPTN setptn);
```

Parameters:

ID	tskid	Task ID
FLGPTN	setptn	Bit pattern to set

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

The task event flag of the task indicated by parameter tskid is logically ORed with the value indicated by parameter setptn and is updated to the resultant value. Note that the lower 16 bits of the bit pattern in parameter setptn must be set to 0 because the corresponding bits of the event flag are reserved for future expansion.

By specifying tskid = TSK_SELF (0), the current task is specified.

In service calls vset_tfl and ivset_tfl, when the logical OR of the updated pattern of the task event flag and the waiting pattern becomes a non-zero value, the task is released from the WAITING state. At this time, the task event flag is cleared to 0.

6.8.9 Clear Task Event Flag (vclr_tfl, ivclr_tfl)

C-Language API:

```
ER ercd = vclr_tfl(ID tskid, FLGPtn clrptn);  
ER ercd = ivclr_tfl(ID tskid, FLGPtn clrptn);
```

Parameters:

ID	tskid	Task ID
FLGPtn	clrptn	Bit pattern to clear

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state.
E_NOEXS	[p]	Undefined (1) Task specified by tskid does not exist.

Function:

The task event flag of the task indicated by parameter tskid are logically ANDed with the value indicated by parameter clrptn and is updated to the resultant value. Note that the lower 16 bits of the bit pattern in parameter clrptn must be set to H'ffff because the corresponding bits of the event flag are reserved for future expansion.

By specifying tskid = TSK_SELF (0), the current task is specified.

6.8.10 Wait for Task Event Flag (vwai_tfl, vpol_tfl, vtwai_tfl)

C-Language API:

```
ER ercd = vwai_tfl(UINT waiptn, FLGPTN *p_tflptn);
ER ercd = vpol_tfl(UINT waiptn, FLGPTN *p_tflptn);
ER ercd = vtwai_tfl(UINT waiptn, FLGPTN *p_tflptn, TMO tmout);
```

Parameters:

FLGPTN	waiptn	Bit pattern to wait
FLGPTN	*p_tflptn	Pointer to the area where the bit pattern when releasing the WAITING state is to be returned
<vtwai_tfl>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
FLGPTN	*p_tflptn	Pointer to the area where the bit pattern when releasing the WAITING state is stored

Error Codes:

E_PAR	[p]	Parameter error (1) waiptn = 0 (2) tmout ≤ -2 (3) p_tflptn is not a 4-byte boundary address.
E_CTX	[k]	Context error (1) Called in a non-task context (2) Called in the dispatch-pended state in a task context (only for vwai_tfl and twai_tfl)
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for vwai_tfl and vtwai_tfl). (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Timeout
E_MACV	[m]	Memory access violation

Function:

Each service call waits for any bit of the task event flag specified by parameter waiptn to be set. When the wait release condition is satisfied, the bit pattern of the task event flag at that time is returned to the area indicated by parameter p_tflptn. At the same time, the task event flag value is cleared to 0.

Each service call immediately terminates processing if any bit specified by waip_{tn} is already set when a service call is issued. If no bit is set, the task that issued service call vwai_{tfl} or vtwai_{tfl} enters the WAITING state. With service call vpol_{tfl}, error code E_TMOUT is immediately returned in this case. Tasks are released from the WAITING state when any bit specified by waip_{tn} is set by service call vset_{tfl}.

The task event flag value is 0 at task initiation.

In service call vtwai_{tfl}, parameter tmout specifies the timeout period.

If a positive value is specified for parameter tmout, error code E_TMOUT is returned when tmout period has passed without the wait release condition being satisfied. If tmout = TMO_POL (0) is specified, the same operation as for service call vpol_{tfl} will be performed. If tmout = TMO_FEVR (-1) is specified, the timeout monitoring is not performed. In other words, the same operation as for service call vwai_{tfl} will be performed.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_{tfl}ptn, which means that an error will be returned if prb_{mem} is issued with the following parameters.
- base = p_{tfl}ptn
 - size = sizeof(T_RTsk)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.9 Task Exception Processing Functions

Table 6.16 Service Calls for Task Exception Processing

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
def_tex [s]	Defines the task exception processing routine	T/E/D/U
idef_tex		N/E/D/U
ras_tex [S]	Requests the task exception processing	T/E/D/U
iras_tex [S]		N/E/D/U/C
dis_tex [S]	Disables the task exception processing	T/E/D/U
ena_tex [S]	Enables the task exception processing	T/E/D/U
sns_tex [S]	Refers to the task exception processing disabled state	T/N/E/D/U/L
ref_tex	Refers to the task exception processing state	T/E/D/U
iref_tex		N/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.17 Task Exception Processing Specifications

Item	Description
Exception cause	32 bits
Task exception processing routine attributes	TA_HLNG: The processing is written in a high-level language. TA_ASM: The processing is written in assembly language. TA_COP0: The routine uses the DSP. TA_COP1: The routine uses FPU register bank 0. TA_COP2: The routine uses FPU register bank 1.

The task exception routine is initiated as a task context when the following conditions are satisfied.

- Task exception processing enabled state
- Pended exception cause is not 0
- The CPU is unlocked
- The interrupt mask is not changed to a non-zero value by service call `chg_ims`
- Task is not executing an extended service call routine or a trap routine.

When execution returns from a task exception processing routine, the processing that was performed before the task exception processing routine was initiated is continued. At this time, the task enters the task exception enabled state. When the pended exception cause is not 0 at this time, the task exception processing routine is initiated again.

Note that the following states do not change before and after the task exception processing routine is initiated and terminated.

- Task or non-task context
- Dispatch-disabled or enabled state
- CPU-locked or unlocked state
- Domain type (kernel or user domain)

6.9.1 Define Task Exception Processing Routine (def_tex, idef_tex)

C-Language API:

```
ER ercd = def_tex(ID tskid, T_DTEX *pk_dtex);
ER ercd = idef_tex(ID tskid, T_DTEX *pk_dtex);
```

Parameters:

ID	tskid	Task ID
T_DTEX	*pk_dtex	Pointer to the packet where task exception-processing-routine definition information is stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure

```
typedef struct {
    ATR    texatr;      0    4    Task exception processing routine
                           attribute
    FP    texrtn;      +4    4    Task exception processing routine
                           initiation address
    UW    inifpscr;    +8    4    Initial FPSCR value
}T_DTEX;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_COP0, TA_COP1, TA_COP2, and TA_ASM in texatr are not 0. (2) TA_COP0 is specified for texatr while CFG_DSP is not selected. (3) TA_COP1 is specified for texatr while CFG_FPU is not selected. (4) TA_COP2 is specified for texatr while TA_COP1 is not specified. (5) Both TA_COP0 and TA_COP1 are specified for texatr.
E_PAR	[p]	Parameter error (1) pk_dtex is not a 4-byte boundary address. (2) texrtn is an odd value.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

The task exception processing routine is defined. The following describes each parameter function.

(1) tskid

Parameter tskid specifies the ID of the task to be defined. By specifying tskid = TSK_SELF (0), the current task is specified.

The task exception processing routine is assigned to the same domain as the target task.

The task exception processing routine for a task in the kernel domain is executed in privileged mode (SR.MD = 1), and that for a task in a user domain is executed in user mode (SR.MD = 0).

(2) texatr

Specify the logical OR of the following values for texatr.

(a) Language

Specify either one of the following values.

- TA_HLNG (H'00000000): High-level language
- TA_ASM (H'00000001): Assembly language

(b) Using a microcomputer with an on-chip DSP (when CFG_DSP is selected)

Specify TA_COP0 to use the DSP.

- TA_COP0 (H'00000100): The routine uses the DSP.

(c) Using a microcomputer with an on-chip FPU (when CFG_FPU is selected)

Specify TA_COP1 to use the FPU for floating-point operations. Specify TA_COP2 in addition to TA_COP1 when using both banks of the FPU for matrix operations.

- TA_COP1 (H'00000200): The routine uses FPU register bank 0 (FPR0_BANK0 to FPR15_BANK0) and FPUL.
- TA_COP2 (H'00000400): The routine uses FPU register bank 1 (FPR0_BANK1 to FPR15_BANK1).

To specify TA_COP2, be sure to specify TA_COP1 together; otherwise, an E_RSATR error will be returned.

Also refer to description (4), inifpscr.

(3) texrtn

texrtn specifies the start address of the task exception processing routine.

When `pk_dtex=NULL (0)` is specified, the definition of the task exception processing routine for the task specified by `tskid` is cancelled. At this time, the pending exception cause for the task is cleared to 0, and the task is shifted to the task exception processing disabled state.

If a task exception processing routine has already been defined, the previous definition is cancelled and is replaced with the new definition. At this time, the pending exception cause is not cleared and task exception processing is not disabled.

(4) `inifpscr`

`inifpscr` is a parameter not specified in the μ ITRON specification.

It is valid only when `CFG_FPU` is selected and the `TA_COP1` attribute is specified. In other cases, it is ignored.

`inifpscr` specifies the FPSCR value at initiation. The kernel sets the `inifpscr` value in FPSCR without checking an error in the `inifpscr` value.

Also refer to the following.

Reference: Section 15, Notes on FPU

Error Detection through `CFG_MEMCHK`:

An `E_MACV` error will be returned in the following cases.

- (1) When `pk_dtex != NULL`, the domain of the caller does not have a read access permission for `pk_dtex`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = pk_dtex`
 - `size = sizeof(T_DTEX)`
 - `domid = Domain of the caller`
 - `pmode = TPM_READ`
- (2) When `pk_dtex != NULL`, the domain of the task indicated by `tskid` does not have a read access permission for `pk_dtex->texrtn`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = pk_dtex->texrtn`
 - `size = 1`
 - `domid = Domain where the task indicated by tskid is assigned`
 - `pmode = TPM_READ`

6.9.2 Request Task Exception Processing (ras_tex, iras_tex)

C-Language API:

```
ER ercd = ras_tex(ID tskid, TEXPTN rasptn);  
ER ercd = iras_tex(ID tskid, TEXPTN rasptn);
```

Parameters:

ID	tskid	Task ID
TEXPTN	rasptn	Task exception cause of task exception processing to be requested

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) rasptn = 0
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state. (2) Task exception processing routine is not defined for the task specified by tskid.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call requests task exception processing through the task exception cause specified by rasptn, for the task specified by tskid. That is, the pending exception cause for the task is logically ORed with the value indicated by parameter rasptn.

By specifying tskid=TSK_SELF (0), the current task is specified.

When the conditions for starting task exception processing routine are satisfied through this service call, the task exception processing routine is initiated.

6.9.3 Disable Task Exception Processing (dis_tex)

C-Language API:

```
ER ercd = dis_tex( );
```

Parameters:

None

Return Parameters:

ER ercd Normal termination (E_OK) or error code

Error Codes:

E_CTX	[k]	Context error (1) Called in a non-task context
E_OBJ	[k]	Invalid object state (1) Task exception processing routine is not defined for the current task.

Function:

The current task is shifted to the task exception processing disabled state.

6.9.4 Enable Task Exception Processing (ena_tex)

C-Language API:

```
ER ercd = ena_tex( );
```

Parameters:

None

Return Parameters:

ER ercd Normal termination (E_OK) or error code

Error Codes:

E_CTX	[k]	Context error (1) Called in a non-task context
E_OBJ	[k]	Invalid object state (1) Task exception processing routine is not defined for the current task.

Function:

The current task is shifted to the task exception enabled state.

When conditions for starting the task exception processing routine are satisfied through this service call, the task exception processing routine is initiated.

6.9.5 Refer To Task Exception Processing Disabled State (sns_tex)

C-Language API:

```
BOOL state= sns_tex( );
```

Parameters:

None

Return Parameters:

BOOL state Task exception processing disabled state

Error Codes:

None

Function:

When a task in the RUNNING state is in the task exception processing disabled state, TRUE is returned; when in the task exception processing enabled state, FALSE is returned. A task in the RUNNING state is the current task when this service call is issued in a task context, or when issued in a non-task context, is the task which had run immediately prior to the transition to the non-task context. When the service call is issued in a non-task context and no task is in the RUNNING state, TRUE is returned.

Tasks for which no task exception processing routines are defined are held in the task exception processing disabled state, so when no task exception processing routine has been defined for a task in the RUNNING state, this service call returns TRUE.

This service call can also be issued in the CPU-locked state and from the CPU exception handler.

6.9.6 Refer to Task Exception Processing State (ref_tex, iref_tex)

C-Language API:

```
ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex);  
ER ercd = iref_tex(ID tskid, T_RTEX *pk_rtex);
```

Parameters:

ID	tskid	Task ID
T_RTEX	*pk_rtex	Pointer to the packet where the task exception processing state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RTEX	*pk_rtex	Pointer to the packet where the task exception processing state is stored

Packet Structure:

```
typedef struct {  
    STAT    texstat;    0    4    Task exception processing state  
    TEXPTN  pndptn;    +4    4    Pended exception cause  
}T_RTEX;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rtex is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Task specified by tskid is in the DORMANT state. (2) Task exception processing routine is not defined for the task specified by tskid.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_MACV	[m]	Memory access violation

Function:

The state of the task exception processing for the task specified by tskid is referenced. By specifying tskid = TSK_SELF (0), the current task is specified.

The following values are returned to the area indicated by pk_rtex.

- **texstat**

One of the following values is returned for texstat, according to whether the target task is in a task exception enabled state or a task exception processing disabled state.

- TTEX_ENA (H'00000000): Task exception processing enabled state
- TTEX_DIS (H'00000001): Task exception processing disabled state

- **pndptn**

The pending exception cause for the target task is returned as pndptn. If there are no unprocessed exception processing requests, 0 is returned as pndptn.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rtex, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rtex
- size = sizeof(T_RTEX)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.10 Synchronization and Communication (Semaphore)

Table 6.18 Service Calls for Synchronization and Communication (Semaphore)

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_sem [s]	Creates semaphore	T/E/D/U
icre_sem		N/E/D/U
acre_sem	Creates semaphore and assigns semaphore ID automatically	T/E/D/U
iacre_sem		N/E/D/U
del_sem	Deletes semaphore	T/E/D/U
sig_sem [S]	Returns semaphore resource	T/E/D/U
isig_sem [S]		N/E/D/U
wai_sem [S]	Waits for semaphore resource	T/E/U
pol_sem [S]	Polls and waits for semaphore resource	T/E/D/U
ipol_sem		N/E/D/U
twai_sem [S]	Waits for semaphore resource with timeout function	T/E/U
ref_sem	Refers to semaphore state	T/E/D/U
iref_sem		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.19 Semaphore Specifications

Item	Description
Semaphore ID	1 to CFG_MAXSEMIC (32767 max.)
Maximum semaphore count	65535
Semaphore attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis. TA_TPRI: Wait task queue is managed on the current priority.

6.10.1 Create Semaphore (cre_sem, icre_sem, acre_sem, iacre_sem)

C-Language API:

```
ER ercd = cre_sem(ID semid, T_CSEM *pk_csem);
ER ercd = icre_sem(ID semid, T_CSEM *pk_csem);
ER_ID semid = acre_sem(T_CSEM *pk_csem);
ER_ID semid = iacre_sem(T_CSEM *pk_csem);
```

Parameters:

T_CSEM	*pk_csem	Pointer to the packet where semaphore creation information is stored
<cre_sem, icre_sem>		
ID	semid	Semaphore ID

Return Parameters:

<cre_sem, icre_sem>		
ER	ercd	Normal termination (E_OK) or error code
<acre_sem, iacre_sem>		
ER_ID	semid	ID of created semaphore (a positive value) or error code

Packet Structure

```
typedef struct {
    ATR    sematr;        0   4   Semaphore attribute
    UINT   isemcnt        +4   4   Initial value of semaphore resource
                                   count
    UINT   maxsem;        +8   4   Maximum number of semaphore resources
}T_CSEM;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) sematr is invalid.
E_PAR	[p]	Parameter error (1) maxsem = 0 or maxsem > H'ffff (2) isemcnt > maxsem (3) pk_csem is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) semid ≤ 0 (2) semid > CFG_MAXSEMID
E_NOID	[k]	No ID available (only for acre_sem)
E_OBJ	[k]	Invalid object state (1) Semaphore specified by semid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls `cre_sem` and `icre_sem` create a semaphore with the ID specified by `semid` using the contents specified by parameter `pk_csem`.

Service calls `acre_sem` and `iacre_sem` search for an unused semaphore ID, create a semaphore for that ID with the contents specified by parameter `pk_csem`, and return the ID as a return parameter. The range to search for an unused semaphore ID is 1 to `CFG_MAXSEMICID`.

Parameter `sematr` specifies the order of the tasks in the queue waiting for the semaphore resource.

`sematr := (TA_TFIFO || TA_TPRI)`

- `TA_TFIFO` (H'00000000): Wait task queue is managed on a FIFO basis
- `TA_TPRI` (H'00000001): Wait task queue is managed on the current priority

Parameter `isemcnt` specifies the initial value of the semaphore to be created. It can range from 0 to `maxsem`.

Parameter `maxsem` specifies the maximum number of resources of the semaphore to be created. It can range from 1 to 65535.

A semaphore can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_csem`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_csem`
 - `size` = `sizeof(T_CSEM)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ`

6.10.2 Delete Semaphore (del_sem)

C-Language API:

```
ER ercd = del_sem(ID semid);
```

Parameters:

ID	semid	Semaphore ID
----	-------	--------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{semid} \leq 0$ (2) $\text{semid} > \text{CFG_MAXSEMID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Semaphore specified by semid does not exist.

Function:

Service call del_sem deletes the semaphore indicated by parameter semid.

No error will occur even if there is a task waiting to acquire a resource with the semaphore indicated by semid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

6.10.3 Return Semaphore Resource (sig_sem, isig_sem)

C-Language API:

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

Parameters:

ID	semid	Semaphore ID
----	-------	--------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{semid} \leq 0$ (2) $\text{semid} > \text{CFG_MAXSEMID}$
E_NOEXS	[k]	Undefined (1) Semaphore specified by semid does not exist.
E_QOVR	[k]	Queuing overflow (1) The semaphore count has already reached the maximum number of semaphore resources specified when the semaphore was created.

Function:

Each service call returns one resource to the semaphore indicated by semid. If there is a task waiting for the semaphore indicated by semid, the task at the head of the wait queue is released from the WAITING state, and the resource is assigned to the task. If there are no tasks in the wait queue, the semaphore count is incremented by one.

The maximum semaphore count is maxsem, which is specified at semaphore creation.

6.10.4 Wait for Semaphore Resource (wai_sem, pol_sem, ipol_sem, twai_sem)

C-Language API:

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

Parameters:

ID	semid	Semaphore ID
<twai_sem>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) tmout ≤ -2
E_ID	[p]	Invalid ID number (1) semid ≤ 0 (2) semid > CFG_MAXSEMIC
E_CTX	[k]	Context error (only for wai_sem and twai_sem) (1) Called in the dispatch-pended state
E_NOEXS	[k]	Undefined (1) Semaphore specified by semid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for wai_sem and twai_sem). (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Semaphore specified by semid was deleted.

Function:

Each service call acquires one resource from the semaphore specified by semid.

Each service call decrements the number of resources of the target semaphore by one if the number of resources of the target semaphore is equal to or greater than 1, and the task issuing the service call continues execution. If no resources exist, the task issuing service call wai_sem or twai_sem is placed in the wait queue for the semaphore, and with service call pol_sem or

ipol_sem, error code E_TMOUT is immediately returned. The wait queue is managed according to the attribute specified at creation.

Parameter tmout specified by service call twai_sem specifies the timeout period. If a positive value is specified for parameter tmout, error code E_TMOUT is returned when the tmout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the same operation as for service call pol_sem will be performed.

If tmout = TMO_FEVR (-1) is specified, the timeout monitoring is not performed. In this case, the same operation as for service call wai_sem will be performed.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7ffffff/CFG_TICDENO$. If a value larger than this is specified, operation is not guaranteed.

6.10.5 Refer to Semaphore State (ref_sem, iref_sem)

C-Language API:

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

Parameters:

ID	semid	Semaphore ID
T_RSEM	*pk_rsem	Pointer to the packet where the semaphore state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RSEM	*pk_rsem	Pointer to the packet where the semaphore state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;          0      2      Wait task ID  
    UINT    semcnt;         +4     4      Current semaphore count  
}T_RSEM;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rsem is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) semid ≤ 0 (2) semid > CFG_MAXSEMIC
E_NOEXS	[k]	Undefined (1) Semaphore specified by semid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the semaphore indicated by parameter semid. Each service call returns the task ID at the head of the semaphore wait queue (wtskid) and the current semaphore count (semcnt), to the area specified by parameter pk_rsem. If there is no task waiting for the semaphore, TSK_NONE (0) is returned as wtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for pk_rsem, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rsem
- size = sizeof(T_RSEM)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.11 Synchronization and Communication (Event Flag)

Table 6.20 Service Calls for Synchronization and Communication (Event Flag)

Service Call ¹		Description	System State ²
			T/N/E/D/U/L/C
cre_flg	[s]	Creates event flag	T/E/D/U
icre_flg			N/E/D/U
acre_flg		Creates event flag and assigns event flag ID automatically	T/E/D/U
iacre_flg			N/E/D/U
del_flg		Deletes event flag	T/E/D/U
set_flg	[S]	Sets event flag	T/E/D/U
iset_flg			N/E/D/U
clr_flg	[S]	Clears event flag	T/E/D/U
iclr_flg			N/E/D/U
wai_flg	[S]	Waits for event flag	T/E/U
pol_flg	[S]	Polls and waits for event flag	T/E/D/U
ipol_flg			N/E/D/U
twai_flg	[S]	Waits for event flag with timeout function	T/E/U
ref_flg		Refers to event flag state	T/E/D/U
iref_flg			N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.21 Event Flag Specifications

Item	Description
Event flag ID	1 to CFG_MAXFLGID (32767 max.)
Event flag size	32 bits
Event flag attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis. TA_TPRI: Wait task queue is managed on the current priority. TA_WSGL: Does not permit multiple tasks to wait for the event flag. TA_WMUL: Permits multiple tasks to wait for the event flag. TA_CLR: Clears event flag at the time of waiting release.

6.11.1 Create Event Flag (cre_flg, icre_flg, acre_flg, iacre_flg)

C-Language API:

```
ER ercd = cre_flg(ID flgid, T_CFLG *pk_cflg);
ER ercd = icre_flg(ID flgid, T_CFLG *pk_cflg);
ER_ID flgid = acre_flg(T_CFLG *pk_cflg);
ER_ID flgid = iacre_flg(T_CFLG *pk_cflg);
```

Parameters:

T_CFLG	*pk_cflg	Pointer to the packet where the event flag creation information is stored
<cre_flg, icre_flg>		
ID	flgid	Event flag ID

Return Parameters:

<cre_flg, icre_flg>		
ER	ercd	Normal termination (E_OK) or error code
<acre_flg, iacre_flg>		
ER_ID	flgid	Created event flag ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    flgatr;      0    4    Event flag attribute
    FLGPNTN iflgpntn;  +4    4    Initial value of event flag
}T_CFLG;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) flgatr is invalid.
E_PAR	[p]	Parameter error (1) pk_cflg is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_NOID	[k]	No ID available (only for acre_flg)
E_OBJ	[k]	Invalid object state (1) Event flag specified by flgid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls `cre_flg` and `icre_flg` create an event flag with the ID specified by `flgid` using the contents specified by `pk_cflg`.

Service calls `acre_flg` and `iacre_flg` search for an unused event flag ID and create an event flag for that ID with the contents specified by parameter `pk_cflg`. The created event flag ID is returned as a return parameter. The range to search for an unused event flag ID is 1 to `CFG_MAXFLGID`.

Parameter `flgatr` specifies the order of the tasks in the queue waiting for the event flag and the number of tasks allowed to wait for the event flag.

$$\text{flgatr} := ((\text{TA_TFIFO} \parallel \text{TA_TPRI}) \mid (\text{TA_WSGL} \parallel \text{TA_WMUL}) \mid [\text{TA_CLR}])$$

- `TA_TFIFO` (H'00000000): Wait task queue is managed on a FIFO basis.
- `TA_TPRI` (H'00000001): Wait task queue is managed on the current priority.
- `TA_WSGL` (H'00000000): Does not permit multiple tasks to wait for the event flag.
- `TA_WMUL` (H'00000002): Permits multiple tasks to wait for the event flag.
- `TA_CLR` (H'00000004): Clears event flag at the time of waiting release.

If `TA_WSGL` attribute is specified for `flgatr`, only one task can wait for the created event flag. In this case, the event flag performs the same operation when either attribute `TA_TFIFO` or `TA_TPRI` is specified. On the other hand, multiple tasks can enter the WAITING state when the `TA_WMUL` attribute is specified. If `TA_CLR` attribute is specified for `flgatr`, all bits of the event flag bit pattern are cleared when the wait release condition is satisfied.

Parameter `iflgptn` specifies the initial value of the event flag to be created.

An event flag can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cflg`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_cflg`
 - `size` = `sizeof(T_CFLG)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ`

6.11.2 Delete Event Flag (del_flg)

C-Language API:

```
ER ercd = del_flg(ID flgid);
```

Parameters:

ID	flgid	Event flag ID
----	-------	---------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Event flag specified by flgid does not exist.

Function:

Service call del_flg deletes the event flag indicated by parameter flgid.

No error will occur even if there is a task waiting for the conditions to be met in the event flag indicated by flgid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

6.11.3 Set Event Flag (set_flg, iset_flg)

C-Language API:

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

Parameters:

ID	flgid	Event flag ID
FLGPTN	setptn	Bit pattern to set

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_NOEXS	[k]	Undefined (1) Event flag specified by flgid does not exist.

Function:

The event flag specified by flgid is logically ORed with the value indicated by parameter setptn and is updated to the resultant value.

Each service call shifts a task to the READY state after the event flag value has been changed and when the wait release conditions of a task waiting for an event flag are satisfied. Wait release conditions are checked in the queue order. All bits of the event flag bit pattern are cleared when the TA_CLR attribute is set to the target event flag attribute and service call processing ends.

When the TA_WMUL attribute is set to the event flag and the TA_CLR attribute is not specified, multiple wait tasks may satisfy the release conditions when service call set_flg is called only once. When multiple wait tasks satisfy the release conditions, the tasks are released in the queue order of the event flag.

6.11.4 Clear Event Flag (clr_flg, iclr_flg)

C-Language API:

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

Parameters:

ID	flgid	Event flag ID
FLGPTN	clrptn	Bit pattern to clear

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_NOEXS	[k]	Undefined (1) Event flag specified by flgid does not exist.

Function:

The event flag specified by flgid is logically ANDed with the value indicated by parameter clrptn and is updated to the resultant value.

6.11.5 Wait for Event Flag Setting (wai_flg, pol_flg, ipol_flg, twai_flg)

C-Language API:

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

Parameters:

ID	flgid	Event flag ID
FLGPTN	waiptn	Wait bit pattern
MODE	wfmode	Wait mode
FLGPTN	*p_flgptn	Pointer to the area where the bit pattern at waiting release is to be returned
<twai_flg>		
TMO	tmout	Timeout value

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
FLGPTN	*p_flgptn	Pointer to the area where the bit pattern at waiting release is stored

Error Codes:

E_PAR	[p]	Parameter error (1) waiptn = 0 (2) wfmode is invalid. (3) tmout ≤ -2 (4) p_flgptn is not a 4-byte boundary address.
E_ID	[p]	Invalid ID (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_CTX	[k]	Context error (only for wai_flg and twai_flg) (1) Called in the dispatch-pended state
E_ILUSE	[k]	Illegal use of service call (1) The target event flag has the TA_WSGL attribute and a task is waiting for the event flag.
E_NOEXS	[k]	Undefined (1) Event flag specified by flgid does not exist.

E_RLWAI	[k]	<p>WAITING state was forcibly cancelled (only for wai_flg and twai_flg).</p> <p>(1) rel_wai service call was issued in the WAITING state.</p> <p>(2) An attempt was made to shift to WAITING state in WAITING-disabled state.</p>
E_TMOU	[k]	Polling failed or timeout
E_DLT	[k]	<p>Waiting object deleted</p> <p>(1) Event flag specified by flgid was deleted.</p>
E_MACV	[m]	Memory access violation

Function:

A task that has called one of these service calls waits until the event flag specified by parameter flgid is set to satisfy the waiting conditions indicated by parameters waipn and wfmode. Each service call returns the bit pattern of the event flag to the area indicated by p_flgptn when the wait release condition is satisfied.

If the attribute of the target event flag is TA_WSG and another task is waiting for the event flag, error code E_ILUSE is returned.

If the wait release conditions are met before a task issues service call wai_flg, pol_flg, ipol_flg, or twai_flg, the service call will be completed immediately. If they are not met, the task will be sent to the wait queue when service call wai_flg or twai_flg is called. With service call pol_flg or ipol_flg, error code E_TMOU is immediately returned, then the processing ends.

The parameter wfmode is specified in the following format.

wfmode:= ((TWF_ANDW || TWF_ORW))

- TWF_ANDW (H'00000000): AND wait
- TWF_ORW (H'00000001): OR wait

If TWF_ANDW is specified as wfmode, the task waits until all the bits specified by waipn have been set. If TWF_ORW is specified as wfmode, the task waits until any one of the bits specified by waipn has been set in the specified event flag.

Parameter tmout for service call twai_flg specifies the timeout period. If a positive value is specified for parameter tmout, error code E_TMOU is returned when the timeout period has passed without the waiting release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the same operation as for service call pol_flg will be performed.

If `tmout = TMO_FEVR (-1)` is specified, the timeout monitoring is not performed. In this case, the same operation as for service call `wai_flg` will be performed.

If a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `p_flgptn`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = p_flgptn`
 - `size = sizeof(FLGPTN)`
 - `domid = Domain of the caller`
 - `pmmode = TPM_READ|TPM_WRITE`

6.11.6 Refer to Event Flag State (ref_flg, iref_flg)

C-Language API

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

Parameters:

ID	flgid	Event flag ID
T_RFLG	*pk_rflg	Pointer to the packet where the event flag state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RFLG	*pk_rflg	Pointer to the packet where the event flag state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;      0      2      Wait task ID  
    FLGPNTN flgpntn;    +4      4      Event flag bit pattern  
}T_RFLG;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rflg is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) flgid ≤ 0 (2) flgid > CFG_MAXFLGID
E_NOEXS	[k]	Undefined (1) Event flag specified by flgid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the event flag indicated by parameter flgid.

Each service call returns the task ID at the head of the event flag wait queue (wtskid) and the current event flag bit pattern (flgpntn), to the area specified by parameter pk_rflg.

If there is no task waiting for the specified event flag, TSK_NONE (0) is returned as wtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rflg, which means that an error will be returned if prb_mem is issued with the following parameters.

- base= pk_rflg
- size = sizeof(T_RFLG)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.12 Synchronization and Communication (Data Queue)

Table 6.22 Service Calls for Synchronization and Communication (Data Queue)

		System State ²
Service Call ¹	Description	T/N/E/D/U/L/C
cre_dtq [s]	Creates data queue	T/E/D/U
icre_dtq		N/E/D/U
acre_dtq	Creates data queue and assigns data queue ID automatically	T/E/D/U
iacre_dtq		N/E/D/U
del_dtq	Deletes data queue	T/E/D/U
snd_dtq [S]	Sends data to data queue	T/E/U
psnd_dtq [S]	Polls and sends data to data queue	T/E/D/U
ipsnd_dtq [S]		N/E/D/U
tsnd_dtq [S]	Sends data to data queue with timeout function	T/E/U
fsnd_dtq [S]	Forcibly sends data to data queue	T/E/D/U
ifsnd_dtq [S]		N/E/D/U
rcv_dtq [S]	Receives data from data queue	T/E/U
prcv_dtq [S]	Polls and receives data from data queue	T/E/D/U
trcv_dtq [S]	Receives data from data queue with timeout function	T/E/U
ref_dtq	Refers to data queue state	T/E/D/U
iref_dtq		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.23 Data Queue Specifications

Item	Description
Data queue ID	1 to CFG_MAXDTQID (32767 max.)
One word	32 bits
Data queue attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis. TA_TPRI: Wait task queue is managed on the current priority.

6.12.1 Create Data Queue (cre_dtq, icre_dtq, acre_dtq, iacre_dtq)

C-Language API:

```
ER ercd = cre_dtq(ID dtqid, T_CDTQ *pk_cdtq);
ER ercd = icre_dtq (ID dtqid, T_CDTQ *pk_cdtq);
ER_ID dtqid = acre_dtq (T_CDTQ *pk_cdtq);
ER_ID dtqid = iacre_dtq (T_CDTQ *pk_cdtq);
```

Parameters:

T_CDTQ	*pk_cdtq	Pointer to the packet where the data queue creation information is stored
<cre_dtq, icre_dtq>		
ID	dtqid	Data queue ID

Return Parameters:

<cre_dtq, icre_dtq>		
ER	ercd	Normal termination (E_OK) or error code
<acre_dtq, iacre_dtq>		
ER_ID	dtqid	Created data queue ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    dtqatr;        0    4    Data queue attribute
    UINT   dtqcnt;        +4   4    Size of data queue area (the number
                                   of data values)
    VP     dtqmb;        +8    4    Start address of data queue area
}T_CDTQ;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) dtqatr is invalid.
E_PAR	[p]	Parameter error (1) dtqcnt != 0 and TSZ_DTQMB(dtqcnt) > (CFG_RESPOOLSZ - VTSZ_RPLMB) (2) pk_cdtq is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) dtqid ≤ 0 (2) dtqid > CFG_MAXDTQID
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_dtq)
E_OBJ	[k]	Invalid object state (1) Data queue specified by dtqid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls `cre_dtq` and `icre_dtq` create a data queue with the ID specified by `dtqid` using the contents specified by `pk_cdtq`.

Service calls `acre_dtq` and `iacre_dtq` search for an unused data queue ID and create a data queue for that ID with the contents specified by `pk_cdtq`, and return the ID as a return parameter. The range to search for unused data queue IDs is from 1 to `CFG_MAXDTQID`.

Attribute `dtqatr` specifies the order of the tasks in the queue waiting for sending a message to the data queue.

`dtqatr` = (`TA_TFIFO` || `TA_TPRI`)

- `TA_TFIFO` (H'00000000): Wait task queue is managed on a FIFO basis.
- `TA_TPRI` (H'00000001): Wait task queue is managed on the current priority.

The wait queue for receiving a message from the data queue is always managed on a FIFO basis. In addition, data to be sent to a data queue is also managed on a FIFO basis in the data queue, without priority.

Parameter `dtqcnt` specifies the number of data items that can be stored in the data queue area. It is also possible to specify a value of 0 for `dtqcnt`; in this case, data sending tasks and data receiving tasks are completely synchronized.

Parameter `dtqmb` is ignored in this kernel. To ensure the portability of programs, specify `NULL` for `dtqmb`.

The kernel allocates a data queue area in the resource pool. For details, refer to the following.

Reference: Section 13.2.2 (2), Data queue

Data queues can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cdtq`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_cdtq`
 - `size` = `sizeof(T_CDTQ)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ`

6.12.2 Delete Data Queue (del_dtq)

C-Language API:

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

Parameters:

ID	dtqid	Data queue ID
----	-------	---------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $dtqid \leq 0$ (2) $dtqid > CFG_MAXDTQID$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Data queue specified by dtqid does not exist.

Function:

The data queue specified by dtqid is deleted.

No error occurs even if there is a send-waiting task or receive-waiting task in the data queue specified by dtqid. However, the WAITING state of the task is cancelled, and an error code E_DLT is returned.

On deletion, the data queue area allocated in the resource pool is released.

6.12.3 Send Data to Data Queue (snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq)

C-Language API:

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

Parameters:

ID	dtqid	Data Queue ID
VP_INT	data	Data to be sent to data queue
<tsnd_dtq>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) tmout ≤ -2
E_ID	[p]	Invalid ID number (1) dtqid ≤ 0 (2) dtqid > CFG_MAXDTQID
E_CTX	[k]	Context error (only for snd_dtq and tsnd_dtq) (1) Called in the dispatch-pended state
E_ILUSE	[k]	Illegal use of service call (1) fsnd_dtq or ifsnd_dtq is issued for the data queue whose dtqcnt is 0.
E_NOEXS	[k]	Undefined (1) Data queue specified by dtqid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for snd_dtq and tsnd_dtq). (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Data queue specified by dtqid was deleted.

Function:

The 4-byte data specified by parameter data is sent to the data queue specified by dtqid.

When a task is waiting to receive data in the target data queue, the data is passed to the head task in the receive-waiting queue and the waiting state of the task is canceled.

When a task is waiting to send data in the target data queue, service calls snd_dtq and tsnd_dtq place the calling task in the queue for waiting a free space in the data queue (send-waiting queue), or service calls psnd_dtq and ipsnd_dtq are immediately terminated with returning an E_TMOOUT error. The send-waiting queue is managed according to the attribute specified when the data queue was created.

When neither a receive-waiting task nor a send-waiting task exists, the data is stored in the data queue. The count of the data queue is incremented by one.

When the data queue count has not reached the maximum data queue count, the calling task is connected to the send-waiting queue.

In service call tsnd_dtq, the wait time is specified for tmout.

If a positive value is specified for parameter tmout, error code E_TMOOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the same operation as for service call psnd_dtq will be performed.

If tmout = TMO_FEVR (-1) is specified, timeout monitoring is not performed. In other words, the same operation as for service call snd_dtq will be performed.

When a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

In fsnd_dtq and ifsnd_dtq, when a task is waiting to send data in the target data queue or when no free space is found in the data queue even if no task is waiting to send data in the target data queue, the oldest data in the data queue is erased and data is sent to that area. In other cases, service calls fsnd_dtq and ifsnd_dtq operate in the same way as snd_dtq and isnd_dtq, respectively.

Note that neither fsnd_dtq nor ifsnd_dtq can be issued for the data queue whose data size is specified as 0. If any one of these service calls is issued, an E_ILUSE error is returned.

6.12.4 Receive Data from Data Queue (rcv_dtq, prcv_dtq, trcv_dtq)

C-Language API:

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

Parameters:

ID	dtqid	Data queue ID
VP_INT	*p_data	Start address of the area where received data is to be returned
<trcv_dtq>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
VP_INT	*p_data	Pointer to the area where received data is stored

Error Codes:

E_PAR	[p]	Parameter error (1) tmout \leq -2 (2) p_data is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) dtqid \leq 0 (2) dtqid > CFG_MAXDTQID
E_CTX	[k]	Context error (1) Called in a non-task context (2) Called in dispatch-pended state in a task context (only for rcv_dtq and trcv_dtq)
E_NOEXS	[k]	Undefined (1) Data queue specified by dtqid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for rcv_dtq and trcv_dtq). (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Data queue specified by dtqid was deleted.
E_MACV	[m]	Memory access violation

Function:

Data is received from the data queue specified by `dtqid`, and stored to the area indicated by parameter `p_data`.

If there is data in the data queue, the first data (the oldest message) is received. On receiving data from the data queue, the data queue count is decremented by 1. As a result, if data can be stored in the data queue, data sending processing is performed for a task in the send-waiting queue in the order of the wait queue.

If there is no data in the data queue, and there exists a send-waiting task (such a circumstance can occur only when the data queue area capacity is 0), the data of the task at the head of data send-waiting queue is received. As a result, the WAITING state of the data send-waiting task is cancelled.

If there is no data in the data queue and there are no send-waiting tasks either, service call `rcv_dtq` or `trcv_dtq` causes the calling task to be placed in the queue for waiting for message arrival (receive-waiting queue). In service call `prcv_dtq`, the call returns immediately with an `E_TMOUT` error. The receive-waiting queue is managed on a FIFO basis.

In service call `trcv_dtq`, `tmout` specifies the wait time.

If a positive value is specified for parameter `tmout`, error code `E_TMOUT` is returned when the timeout period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `prcv_dtq` will be performed. If `tmout = TMO_FEVR (-1)` is specified, timeout monitoring is not performed. In other words, the same operation as for service call `rcv_dtq` will be performed.

When a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for `p_data`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base = p_data`
- `size = sizeof(VP_INT)`
- `domid = Domain of the caller`
- `pmmode = TPM_READ|TPM_WRITE`

6.12.5 Refer to Data Queue State (ref_dtq, iref_dtq)

C-Language API:

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

Parameters:

ID	dtqid	Data queue ID
T_RDTQ	*pk_rdtq	Pointer to the packet where data queue state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RDTQ	*pk_rdtq	Pointer to the packet where data queue state is stored

Packet Structure:

```
typedef struct {  
    ID      stskid;      0      2      Task ID waiting for sending  
    ID      rtskid;      +2     2      Task ID waiting for receiving  
    UINT    sdtqcnt;     +4     4      The number of data in the data queue  
}T_RDTQ;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rdtq is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) dtqid ≤ 0 (2) dtqid > CFG_MAXDTQID
E_NOEXS	[k]	Undefined (1) Data queue specified by dtqid does not exist.
E_MACV	[m]	Memory access violation

Function:

The state of the data queue specified by dtqid is referenced, and the send-waiting task IDs (stskid), the receive-waiting task IDs (rtskid), and the number of data items in the data queue (sdtqcnt) are returned to the area specified by pk_rdtq. When no task is waiting for sending in the target data queue, TSK_NONE (0) is returned through stskid. When no task is waiting for receiving in the target data queue, TSK_NONE (0) is returned through rtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rdtq, which means that an error will be returned if prb_mem is issued with the following parameters.

- base= pk_rdtq
- size = sizeof(T_RDTQ)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.13 Synchronization and Communication (Mailbox)

Table 6.24 Service Calls for Synchronization and Communication (Mailbox)

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_mbx [s]	Creates mailbox	T/E/D/U
icre_mbx		N/E/D/U
acre_mbx	Creates mailbox and assigns mailbox ID automatically	T/E/D/U
iacre_mbx		N/E/D/U
del_mbx	Deletes mailbox	T/E/D/U
snd_mbx [S]	Sends data to mailbox	T/E/D/U
isnd_mbx		N/E/D/U
rcv_mbx [S]	Receives data from mailbox	T/E/U
prcv_mbx [S]	Polls and receives data from mailbox	T/E/D/U
iprcv_mbx		N/E/D/U
trcv_mbx [S]	Receives data from mailbox with timeout function	T/E/U
ref_mbx	Refers to mailbox state	T/E/D/U
iref_mbx		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.25 Mailbox Specifications

Item	Description
Mailbox ID	1 to CFG_MAXMBXID (32767 max.)
Message priority	1 to CFG_MAXMSGPRI* (255 max.)
Mailbox attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis. TA_TPRI: Wait task queue is managed on the current priority. TA_MFIFO: Message queue is managed on a FIFO basis. TA_MPRI: Message queue is managed on the current priority.

Note: This value is same as TMAX_MPRI defined in kernel_macro.h.

6.13.1 Create Mailbox (cre_mbx, icre_mbx, acre_mbx, iacre_mbx)

C-Language API:

```
ER ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx);
ER ercd = icre_mbx(ID mbxid, T_CMBX *pk_cmbx);
ER_ID mbxid = acre_mbx(T_CMBX *pk_cmbx);
ER_ID mbxid = iacre_mbx(T_CMBX *pk_cmbx);
```

Parameters:

T_CMBX	*pk_cmbx	Pointer to the packet where the mailbox creation information is stored
<cre_mbx, icre_mbx>	ID	mbxid Mailbox ID

Return Parameters:

<cre_mbx, icre_mbx>	ER	ercd	Normal termination (E_OK) or error code
<acre_mbx, iacre_mbx>	ER_ID	mbxid	Created mailbox ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    mbxatr;      0    4    Mailbox attribute
    UINT    mbxcnt;     +4    4    Number of messages that can be stored
    PRI    maxmpri;     +8    2    Highest message priority
    VP    mbxmb;       +12   4    Start address of mailbox management
                                   area
}T_CMBX;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) mbxatr is invalid.
E_PAR	[p]	Parameter error (1) maxmpri ≤ 0 (2) maxmpri > CFG_MAXMSGPRI (3) pk_cmbx is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbxid ≤ 0 (2) mbxid > CFG_MAXMBXID
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_mbx)
E_OBJ	[k]	Invalid object state (1) Mailbox specified by mbxid already exists.

Function:

Service calls `cre_mbx` and `icre_mbx` create a mailbox with the ID specified by `mbxid` using the contents specified by `pk_cmbx`.

Service calls `acre_mbx` and `iacre_mbx` search for an unused mailbox ID and create a mailbox for that ID with the contents specified by parameter `pk_cmbx`. The created mailbox ID is returned as a return parameter. The range to search for an unused mailbox ID is 1 to `CFG_MAXMBXID`.

Parameter `mbxatr` specifies the order of the receive-waiting tasks and messages in the wait queues.

`mbxatr` := ((TA_TFIFO || TA_TPRI) | TA_MFIFO || TA_MPRI)

- TA_TFIFO (H'00000000): Message receive-waiting queue is managed on a FIFO basis.
- TA_TPRI (H'00000001): Message receive-waiting queue is managed on the current priority.
- TA_MFIFO (H'00000000): Message queue is managed on a FIFO basis.
- TA_MPRI (H'00000002): Message queue is managed on the current priority.

`mbxcnt` and `mbxmb` are always ignored in this kernel. To ensure the portability of programs, specify an appropriate value for `mbxcnt` and NULL for `mbxmb`.

When the TA_MPRI attribute is specified and `maxmpri > 1`, the kernel uses an area in the resource pool to manage the mailbox. For details, refer to the following.

Reference: Section 13.2.2 (3), Mailbox

A mailbox can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cmbx`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base` = `pk_cmbx`
- `size` = `sizeof(T_CMBX)`
- `domid` = Domain of the caller
- `pmmode` = `TPM_READ`

6.13.2 Delete Mailbox (del_mbx)

C-Language API:

```
ER ercd = del_mbx(ID mbxid);
```

Parameters:

ID	mbxid	Mailbox ID
----	-------	------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $mbxid \leq 0$ (2) $mbxid > CFG_MAXMBXID$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Mailbox specified by mbxid does not exist.

Function:

Service call del_mbx deletes the mailbox indicated by parameter mbxid.

After the mailbox is deleted, the management area that was allocated in the resource pool to create the mailbox and send messages is released.

No error will occur even if there is a task waiting for a message in the mailbox indicated by mbxid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned. If there is a message in the mailbox, no error will occur, but the kernel will not perform any processing for the message area. For example, the kernel will not automatically return the message area to the memory pool when a memory block acquired from the memory pool is used for a message.

6.13.3 Send Message to Mailbox (snd_mbx, isnd_mbx)

C-Language API:

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

Parameters:

ID	mbxid	Mailbox ID
T_MSG	*pk_msg	Start address of the message to be sent

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

<Mailbox message header>

```
typedef struct {
    VP    msghead;    0    4    Kernel management area
}T_MSG;
```

msghead is provided only to ensure the compatibility of the message format with the former versions. It does not need to be cleared to zero when data is sent to the mailbox.

<Mailbox message header with priority>

```
typedef struct {
    T_MSG  msgque;    0    4    Message header
    PRI    msgpri;    +4    2    Message priority
}T_MSG_PRI;
```

Error Codes:

E_PAR		Parameter error
	[p]	(1) pk_msg is not a 4-byte boundary address.
	[k]	(2) The TA_MPRI attribute is specified for the target mailbox and msgpri ≤ 0 or msgpri > (highest message priority specified when the mailbox was created).
E_ID	[p]	Invalid ID number
		(1) mbxid ≤ 0
		(2) mbxid > CFG_MAXMBXID
E_NOMEM	[k]	Insufficient memory
		(1) Insufficient space in the resource pool
E_NOEXS	[k]	Undefined
		(1) Mailbox specified by mbxid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call sends a message specified by `pk_msg` to the mailbox specified by `mbxid`.

If there is a task waiting to receive a message in the mailbox, the task at the head of the wait queue receives the message and is released from the WAITING state. On the other hand, if there are no tasks waiting to receive a message, the message specified by `pk_msg` is placed at the end of the message queue. The message queue is managed according to the attribute specified at creation. Here, the kernel uses an area in the resource pool to manage the message. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.3 (1), Mailbox: `snd_mbx`, `isnd_mbx`

To send a message to a mailbox that has the `TA_MFIFO` attribute, the message must have the `T_MSG` structure at the head of the message, as shown in Figure 6.2.

To send a message to a mailbox that has the `TA_MPRI` attribute, the message must have the `T_MSG_PRI` structure at the head of the message, as shown in Figure 6.3.

Messages must be created in RAM.

```
typedef struct {
    T_MSG    t_msg;        /* T_MSG structure */
    B        data[8];      /* Example of user message data structure (any structure) */
} USER_MSG;
```

Figure 6.2 Example of a Message Form

```
typedef struct {
    T_MSG_PRI t_msg;       /* T_MSG_PRI structure */
    B        data[8];      /* Example of user message data structure (any structure) */
} USER_MSG;
```

Figure 6.3 Example of a Message Form with Priority

The sent message is read by the receiving task. Accordingly, note the following when creating a message.

- (1) In principle, messages must not be created as local variables.
- (2) When the memory protection function is used, a message must be created in an area for which the receiving task has a read access permission. In general, it is recommended to use a mailbox for communications between tasks within the same domain. To transfer messages between

domains, consider the use of a data queue (only one word), a message buffer, or a protected mailbox.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for pk_msg, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_msg

- size = sizeof(T_MSG_PRI)

Note that this size should be checked even when the TA_MFIFO attribute is specified.

- domid = Domain of the caller

- pmmode = TPM_READ

6.13.4 Receive Message from Mailbox (rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx)

C-Language API:

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

Parameters:

ID	mbxid	Mailbox ID
T_MSG	**ppk_msg	Pointer to the area where the start address of the received message is to be returned
<trcv_mbx>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_MSG	**ppk_msg	Pointer to the area where the start address of the received message is stored

Packet Structure:

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

Error Codes:

E_PAR	[p]	Parameter error (1) tmout ≤ -2 (2) ppk_msg is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbxid ≤ 0 (2) mbxid > CFG_MAXMBXID)
E_CTX	[k]	Context error (only for rcv_mbx and trcv_mbx) (1) Called in the dispatch-pended state

E_RLWAI	[k]	<p>WAITING state is forcibly cancelled (only for rcv_mbx and trcv_mbx).</p> <p>(1) rel_wai service call was issued in the WAITING state.</p> <p>(2) An attempt was made to shift to WAITING state in WAITING-disabled state.</p>
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	<p>Waiting object deleted</p> <p>(1) Mailbox specified by mbxid was deleted.</p>
E_MACV	[m]	Memory access violation

Function:

Each service call receives a message from the mailbox specified by parameter mbxid. Then the start address of the received message is returned to the area indicated by parameter ppk_msg.

After a message is received, the management area that was acquired from the resource pool by the kernel to manage the message when the message was sent is released.

With service calls rcv_mbx and trcv_mbx, if there are no messages in the mailbox, the task that issued the service call is placed in the wait queue to receive a message (receive-waiting queue). With service calls prcv_mbx and iprcv_mbx, if there are no messages in the mailbox, error code E_TMOUT is returned immediately. The wait queue is managed according to the attribute specified at creation.

Parameter tmout specified by service call trcv_mbx specifies the timeout period.

If a positive value is specified for parameter tmout, error code E_TMOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the same operation as for service call prcv_mbx will be performed.

If tmout = TMO_FEVR (−1) is specified, timeout monitoring is not performed. In other words, the same operation as for service call rcv_mbx will be performed.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `ppk_msg`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
- `base = ppk_msg`
 - `size = sizeof(T_MSG *)`
 - `domid = Domain of the caller`
 - `pmmode = TPM_READ|TPM_WRITE`

6.13.5 Refer to Mailbox State (ref_mbx, iref_mbx)

C-Language API:

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

Parameters:

ID	mbxid	Mailbox ID
T_RMBX	*pk_rmbx	Pointer to the packet where the mailbox state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMBX	*pk_rmbx	Pointer to the packet where the mailbox state is stored

Packet Structure:

```
(1) T_RMBX  
typedef struct {  
    ID      wtskid;      0    2    Wait task ID  
    T_MSG   *pk_msg;    +4    4    Start address of the message to be  
                                   received next  
}T_RMBX;  
(2) T_MSG  
<Mailbox message header>  
typedef struct {  
    VP      msghead;    0    4    Kernel management area  
}T_MSG;  
<Mailbox message header with priority>  
typedef struct {  
    T_MSG   msgque;      0    4    Message header  
    PRI     msgpri;     +4    2    Message priority  
}T_MSG_PRI;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmbx is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbxid ≤ 0 (2) mbxid > CFG_MAXMBXID
E_NOEXS	[k]	Undefined (1) Mailbox specified by mbxid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the mailbox specified by parameter mbxid.

Each service call returns the wait task ID (wtskid) and the start address of the message to be received next (pk_msg) to the area indicated by pk_rmbx.

If there is no task waiting in the specified mailbox, TSK_NONE (0) is returned as wtskid.

If there is no message to be received next, NULL (0) is returned as pk_msg.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for pk_rmbx, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rmbx
- size = sizeof(T_RMBX)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.14 Synchronization and Communication (Mutex)

Table 6.26 Service Calls for Synchronization and Communication (Mutex)

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_mtx	Creates mutex	T/E/D/U
acre_mtx	Creates mutex and assigns mutex ID automatically	T/E/D/U
del_mtx	Deletes mutex	T/E/D/U
loc_mtx	Locks mutex	T/E/U
ploc_mtx	Polls and locks mutex	T/E/D/U
tloc_mtx	Locks mutex with timeout function	T/E/U
unl_mtx	Unlocks mutex	T/E/D/U
ref_mtx	Refers to mutex state	T/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.27 Mutex Specifications

Item	Description
Mutex ID	1 to CFG_MAXMTXID (32767 max.)
Mutex attributes	TA_CEILING: Ceiling priority protocol

Note: This kernel only supports the TA_CEILING attribute (ceiling priority protocol). In this kernel, the mutex is managed by "simplified priority control rule". Under this rule, the management which changes the task's current priority to a higher value is always done, but the management which changes the task's priority to a lower value is done only when the task releases all of mutexes.

6.14.1 Create Mutex (cre_mtx, acre_mtx)

C-Language API:

```
ER ercd = cre_mtx(ID mtxid, T_CMTX *pk_cmtx);  
ER_ID mtxid = acre_mtx(T_CMTX *pk_cmtx);
```

Parameters:

T_CMTX	*pk_cmtx	Pointer to the packet where the mutex creation information is stored
--------	----------	--

<cre_mtx>

ID	mtxid	Mutex ID
----	-------	----------

Return Parameters:

<cre_mtx>

ER	ercd	Normal termination (E_OK) or error code
----	------	---

<acre_mtx>

ER_ID	mtxid	Created mutex ID (a positive value) or error code
-------	-------	---

Packet Structure:

```
typedef struct {  
    ATR    mtxatr;      0    4    Mutex attribute  
    PRI    ceilpri;     +4   2    Ceiling priority of mutex  
}T_CMTX;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) mtxatr is invalid.
E_PAR	[p]	Parameter error (1) ceilpri ≤ 0 (2) ceilpri > CFG_MAXTSKPRI (3) pk_cmtx is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mtxid ≤ 0 (2) mtxid > CFG_MAXMTXID
E_NOID	[k]	No ID available (only for acre_mtx)
E_OBJ	[k]	Invalid object state (1) Mutex specified by mtxid already exists.
E_MACV	[m]	Memory access violation

Function:

Service call `cre_mtx` creates a mutex with the ID specified by `mtxid` using the contents specified by `pk_cmtx`.

Service call `acre_mtx` searches for an unused mutex ID and creates a mutex for that ID with the contents specified by `pk_cmtx`, and returns the ID as a return parameter. The range to search for an unused mutex ID is from 1 to `CFG_MAXMTXID`.

As the `mtxatr` attribute, only the ceiling priority protocol (`TA_CEILING`) can be specified.

`mtxatr:= (TA_CEILING)`

- `TA_CEILING` (H'00000003): Ceiling priority protocol

Wait task queue is always managed on the current priority.

Parameter `ceilpri` specifies the ceiling priority for the mutex to be created. The range of values which can be specified is 1 to `CFG_MAXTSKPRI`.

A mutex can also be created statically by the configurator.

This service call must not be issued in a non-task context, but even if it is attempted, no `E_CTX` error is detected.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cmtx`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base` = `pk_cmtx`
- `size` = `sizeof(T_CMTX)`
- `domid` = Domain of the caller
- `pmmode` = `TPM_READ`

6.14.2 Delete Mutex (del_mtx)

C-Language API:

```
ER ercd = del_mtx(ID mtxid);
```

Parameters:

ID	mtxid	Mutex ID
----	-------	----------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{mtxid} \leq 0$ (2) $\text{mtxid} > \text{CFG_MAXMTXID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Mutex specified by mtxid does not exist.

Function:

Service call del_mtx deletes the mutex specified by parameter mtxid.

No error occurs even when there is a lock-waiting task for the mutex specified by mtxid; but the WAITING state of the task is cancelled, and E_DLT is returned as an error code.

When the target mutex is locked, the lock for the task that locks the mutex is cancelled. As a result, only when all mutexes locked by the task are unlocked, the task priority is returned to the base priority.

The task locking the deleted mutex is not notified that the mutex has been deleted. If an attempt is later made to release the mutex lock, an error is returned.

6.14.3 Lock Mutex (loc_mtx, ploc_mtx, tloc_mtx)

C-Language API:

```
ER ercd = loc_mtx(ID mtxid);  
ER ercd = ploc_mtx(ID mtxid);  
ER ercd = tloc_mtx(ID mtxid, TMO tmout);
```

Parameters:

ID	mtxid	Mutex ID
<tloc_mtx>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) tmout \leq -2
E_ID	[p]	Invalid ID (1) mtxid \leq 0 (2) mtxid > CFG_MAXMTXID
E_CTX	[k]	Context error (1) Called in a non-task context (2) Called in the dispatch-pended state in a task context (only for loc_mtx and tloc_mtx)
E_ILUSE	[k]	Illegal use of service call (1) The target mutex is already locked by the calling task. (2) Ceiling priority violation (The base priority of the calling task is less than the ceiling priority specified at mutex creation.)
E_NOEXS	[k]	Undefined (1) Mutex specified by mtxid does not exist.
E_RLWAI	[k]	The WAITING state was forcibly cancelled (only for loc_mtx and tloc_mtx). (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Mutex specified by mtxid was deleted.

Function:

Each service call locks the mutex specified by parameter `mtxid`.

When the target mutex is not locked, the current task locks the mutex, and the service call processing is completed. At this time, the priority of the current task is raised to the ceiling priority of the mutex.

If the target mutex is locked, the current task is placed in a wait queue, and the current task enters the mutex lock-wait state. The wait queue is managed in priority order.

Parameter `tmout` specified by service call `tloc_mtx` specifies the timeout period.

If a positive value is specified for parameter `tmout`, error code `E_TMOUT` is returned when the timeout period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `ploc_mtx` will be performed.

If `tmout = TMO_FEVR (-1)` is specified, timeout monitoring is not performed. In other words, the same operation as for service call `loc_mtx` will be performed.

When a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

6.14.4 Unlock Mutex (unl_mtx)

C-Language API:

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

Parameters:

ID	mtxid	Mutex ID
----	-------	----------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID (1) $\text{mtxid} \leq 0$ (2) $\text{mtxid} > \text{CFG_MAXMTXID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_ILUSE	[k]	Illegal use of service call (1) The calling task has not locked the target mutex.
E_NOEXS	[k]	Undefined (1) Mutex specified by mtxid does not exist.

Function:

The lock for the mutex specified by mtxid is released. If there is a task waiting for the lock for the specified mutex, the WAITING state for the task at the head of the mutex wait queue is released, and the task whose WAITING state has been released is put into a state which locks the mutex. At this time, the priority of the locking task is raised to the ceiling priority of the mutex. If there are no tasks waiting for the mutex, the mutex is put into the unlocked state.

Through this service call, only when all the mutexes that are locked by the current task are unlocked, the current priority of the task is returned to the base priority.

6.14.5 Refer to Mutex State (ref_mtx)

C-Language API:

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

Parameters:

ID	mtxid	Mutex ID
T_RMTX	*pk_rmtx	Pointer to the area where the mutex status is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMTX	*pk_rmtx	Pointer to the packet where the mutex status is stored

Packet Structure:

```
typedef struct {  
    ID    htskid;      0    2    Task ID locking a mutex  
    ID    wtskid;      +2   2    ID of the task at the head of mutex  
                                   waiting queue  
}T_RMTX;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmtx is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mtxid ≤ 0 (2) mtxid > CFG_MAXMTXID
E_NOEXS	[k]	Undefined (1) Mutex specified by mtxid does not exist.
E_MACV	[m]	Memory access violation

Function:

Service call ref_mtx refers to the state of the mutex specified by mtxid. Service call ref_mtx returns the task ID that locks the mutex (htskid) and the ID of the task placed at the head of the mutex wait queue (wtskid) to the area indicated by pk_rmtx. If there is no task that locks the target mutex, TSK_NONE (0) is returned to htiskid. If there is no task waiting for the target mutex, TSK_NONE (0) is returned to wtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rmtx, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rmtx
- size = sizeof(T_RMTX)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.15 Extended Synchronization and Communication (Message Buffer)

Table 6.28 Service Calls for Extended Synchronization and Communication (Message Buffer)

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_mbf	Creates message buffer	T/E/D/U
icre_mbf		N/E/D/U
acre_mbf	Creates message buffer and assigns message	T/E/D/U
iacre_mbf	buffer ID automatically	N/E/D/U
del_mbf	Deletes message buffer	T/E/D/U
snd_mbf	Sends message to message buffer	T/E/U
psnd_mbf	Polls and sends message to message buffer	T/E/D/U
ipsnd_mbf		N/E/D/U
tsnd_mbf	Sends message to message buffer with timeout function	T/E/U
rcv_mbf	Receives message from message buffer	T/E/U
prcv_mbf	Polls and receives message from message buffer	T/E/D/U
trcv_mbf	Receives message from message buffer with timeout function	T/E/U
ref_mbf	Refers to message buffer state	T/E/D/U
iref_mbf		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.29 Message Buffer Specifications

Item	Description
Message buffer ID	1 to CFG_MAXMBFID (32767 max.)
Message buffer attributes	TA_TFIFO: Task queue waiting for sending a message is managed on a FIFO basis TA_TPRI: Task queue waiting for sending a message is managed on the current priority

6.15.1 Create Message Buffer (cre_mbf, icre_mbf, acre_mbf, iacre_mbf)

C-Language API:

```
ER ercd = cre_mbf(ID mbfid, T_CMBF *pk_cmbf);
ER ercd = icre_mbf(ID mbfid, T_CMBF *pk_cmbf);
ER_ID mbfid = acre_mbf(T_CMBF *pk_cmbf);
ER_ID mbfid = iacre_mbf(T_CMBF *pk_cmbf);
```

Parameters:

T_CMBF	*pk_cmbf	Pointer to the packet where the message buffer creation information is stored
<cre_mbf, icre_mbf>		
ID	mbfid	Message buffer ID

Return Parameters:

<cre_mbf, icre_mbf>		
ER	ercd	Normal termination (E_OK) or error code
<acre_mbf, iacre_mbf>		
ER_ID	mbfid	Created message buffer ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    mbfatr;      0  4    Message buffer attribute
    UINT   maxmsz;      +4  4    Maximum message size (number of bytes)
    SIZE   mbfsz;       +8  4    Message buffer size (number of bytes)
    VP     mbfmb;       +12 4    Start address of message buffer
                                management area
}T_CMBF;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) mbfatr is invalid.
E_PAR	[p]	Parameter error (1) When mbfsz != 0, mbfsz < TSZ_MBFMB(1,maxmsz) or mbfsz > (CFG_RESPOOLSZ - VTSZ_RPLMB) (2) maxmsz = 0 (3) pk_cmbf is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbfid ≤ 0 (2) mbfid > CFG_MAXMBFID
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_mbf)

E_OBJ	[k]	Invalid object state (1) Message buffer specified by mbfid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls cre_mbf and icre_mbf create a message buffer with the ID specified by mbfid using the contents specified by pk_cmbf.

Service calls acre_mbf and iacre_mbf search for an unused message buffer ID and create a message buffer for that ID with the contents specified by parameter pk_cmbf. The created message buffer ID is returned as a return parameter. The range to search for an unused message buffer ID is 1 to CFG_MAXMBFID.

Parameter mbfatr specifies the order of the tasks in the queue waiting for sending a message to the message buffer.

mbfatr:= (TA_TFIFO || TA_TPRI)

- TA_TFIFO (H'00000000): Task queue waiting for sending a message is managed on a FIFO basis.
- TA_TPRI (H'00000001): Task queue waiting for sending a message is managed on the current priority.

The message queue and the task queue waiting for receiving a message are managed on a first-in first-out (FIFO) basis regardless of the mbfatr specification.

Parameter maxmsz specifies the maximum length of a message that can be held in a message buffer.

Parameter mbfsz specifies the size of the message buffer to be created. The mbfsz value is rounded up to a multiple of four during processing. The following macro is provided to estimate the approximate size to be specified for mbfsz.

SIZE mbfsz = TSZ_MBFMB(UINT msgcnt, UINT msgsz)

Approximate size (bytes) of a message buffer area required to hold the msgcnt number of msgsz-byte messages

The mbfsz value must be equal to or larger than TSZ_MBFSZ(1,maxmsz).

A message buffer of mbfsz = 0 can also be created. In this case, no message can be stored in the message buffer, and the message-receiving task completely synchronizes with the message-sending task. In other words, when a service call to send a message is issued, the task stays in the

WAITING state until another task issues a service call to receive a message. Similarly, when a task issues a service call to receive a message, the task stays in the WAITING state until another task issues a service call to send a message.

When `mbfsz != 0`, the kernel allocates a message buffer area in the resource pool. For details, refer to the following.

Reference: Section 13.2.2 (4), Message buffer

Parameter `mbfmb` is ignored in this kernel. To ensure the portability of programs, specify `NULL` for `mbfmb`.

A message buffer can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cmbf`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = pk_cmbf`
 - `size = sizeof(T_CMBF)`
 - `domid = Domain of the caller`
 - `pmmode = TPM_READ`

6.15.2 Delete Message Buffer(del_mbf)

C-Language API:

```
ER ercd = del_mbf(ID mbfid);
```

Parameters:

ID	mbfid	Message buffer ID
----	-------	-------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $mbfid \leq 0$ (2) $mbfid > CFG_MAXMBFID$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Message buffer specified by mbfid does not exist.

Function:

Service call del_mbf deletes the message buffer specified by parameter mbfid.

No error will occur even if there is a task waiting for receiving or sending a message in the message buffer indicated by mbfid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned. In addition, if there is a message in the message buffer, no error will occur, but all stored messages will be deleted.

On deletion, the message buffer area allocated in the resource pool is released.

6.15.3 Send Message to Message Buffer (snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

C-Language API:

```
ER ercd = snd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = psnd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = ipsnd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = tsnd_mbf(ID mbfid, VP msg, UINT msgsz, TMO tmout);
```

Parameters:

ID	mbfid	Message buffer ID
VP	msg	Start address of the message to send
UINT	msgsz	Size of the message to send (number of bytes)
<tsnd_mbf>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR		Parameter error
	[p]	(1) msgsz = 0 (2) tmout ≤ -2 (3) msg is not a 4-byte boundary address.
	[k]	msgsz > (maximum message size specified at creation)
E_ID	[p]	Invalid ID number
		(1) mbfid ≤ 0 (2) mbfid > CFG_MAXMBFID
E_CTX	[k]	Context error (only for snd_mbf and tsnd_mbf)
		(1) Called in dispatch-pended state
E_NOEXS	[k]	Undefined
		(1) Message buffer specified by mbfid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for snd_mbf and tsnd_mbf)
		(1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted
		(1) Message buffer specified by mbfid was deleted.
E_MACV	[m]	Memory access violation

Function:

Each service call sends a message specified by `msg` to the message buffer specified by `mbfid`. The message size is specified by parameter `msgsz`.

If there is a task waiting to receive a message from the specified message buffer, the message sent by the service call is not placed in the message buffer. Instead, the message is passed to the task at the head of the receive-waiting queue, releasing the task from the WAITING state.

If there are already tasks waiting to send a message to the message buffer, the task that issued service call `snd_mbf` or `tsnd_mbf` is placed in the queue to wait for free space in the message buffer (send-waiting queue). With service calls `psnd_mbf` and `ipsnd_mbf`, error code `E_TMOU`T is immediately returned. The wait queue is managed according to the attribute specified at creation.

If there are no tasks waiting to send or receive a message, the message sent from a task is stored in the message buffer. After that, the size of the free space in the message buffer will decrease by `VTSZ_MBFMSGMB(msgsz)` bytes. If the free space in the message buffer is less than this size (including when the buffer size is 0), the task that issued the service call is placed in the send-waiting queue.

`ipsnd_mbf` can also be issued in a non-task context. Since the priority of a non-task context is higher than that of a task when the target message buffer has `TA_TPRI` attribute, the specified message is copied to the buffer when the buffer has enough free space for the required size, even if there exists a task that has been waiting to send a message.

In service call `tsnd_mbf`, parameter `tmout` specifies the timeout period. If a positive value is specified for parameter `tmout`, error code `E_TMOU`T is returned when the `tmout` period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `psnd_mbf` will be performed. If `tmout = TMO_FEVR (-1)` is specified, the same operation as for service call `snd_mbf` will be performed. In other words, timeout monitoring is not performed.

If a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `msg`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- base = msg
- size = msgsz
- domid = Domain of the caller
- pmmode = TPM_READ

6.15.4 Receive Message from Message Buffer (rcv_mbf, prcv_mbf, trcv_mbf)

C-Language API:

```
ER_UINT msgsz = rcv_mbf(ID mbfid, VP msg);  
ER_UINT msgsz = prcv_mbf(ID mbfid, VP msg);  
ER_UINT msgsz = trcv_mbf(ID mbfid, VP msg, TMO tmout);
```

Parameters:

ID	mbfid	Message buffer ID
VP	msg	Start address of the area where the received message is to be returned
<trcv_mbf>		
TMO	tmout	Timeout specification

Return Parameters:

ER_UINT	msgsz	Size of the received message (number of bytes, a positive value) or error code
VP	msg	Start address of the area where the received message is stored

Error Codes:

E_PAR	[p]	Parameter error (1) tmout ≤ -2 (2) msg is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbfid ≤ 0 (2) mbfid > CFG_MAXMBFID
E_CTX	[k]	Context error (1) Called in dispatch-pended state
E_NOEXS	[k]	Undefined (1) Message buffer specified by mbfid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for rcv_mbf and trcv_mbf) (1) rel_wai service call was issued in the WAITING state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Message buffer specified by mbfid was deleted.
E_MACV	[m]	Memory access violation

Function:

Each service call receives a message from the message buffer specified by parameter mbfid and stores the received message in the area specified by msg. The received message size is returned as

the return parameter. Through parameter msg, a free area that can hold the maximum message size (maxmsz) specified at message buffer creation must be specified.

If there are already messages in the message buffer, the task receives the message of the head of the queue (the oldest message). After the message has been received, the size of the free space in the message buffer will increase by VTSZ_MBFMSGMB(msgsz) bytes.

If, as a result, the free space in the message buffer becomes larger than the size of the message to be sent by the task at the head of the send-waiting queue, the message is stored in the message buffer and the task is released from the WAITING state. The same process will be done for the remaining tasks in the order of the wait queue if the remaining message buffer size still has enough contiguous free space.

If there are no messages in the message buffer and there are tasks waiting to send a message, the message of the task at the head of the wait queue is received. As a result, the sending task is released from the WAITING state.

If there are no messages in the message buffer and there are no tasks in the queue to send a message, the task that issued service call rcv_mbf or trcv_mbf is placed in the wait queue to receive a message (receive-waiting queue). With service call prcv_mbf, error code E_TMOU is immediately returned. The wait queue is managed on a FIFO basis.

In service call trcv_mbf, parameter tmout specifies the timeout period. If a positive value is specified for parameter tmout, error code E_TMOU is returned when the tmout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the same operation as for service call prcv_mbf will be performed. If tmout = TMO_FEVR (-1) is specified, timeout monitoring is not performed. In other words, the same operation as for service call rcv_mbf will be performed.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for tmout is $H'7ffffff/CFG_TICDENO$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for msg, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = msg
 - size = Maximum message size specified at creation
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.15.5 Refer to Message Buffer State (ref_mbf, iref_mbf)

C-Language API:

```
ER ercd = ref_mbf(ID mbfid, T_RMBF *pk_rmbf);  
ER ercd = iref_mbf(ID mbfid, T_RMBF *pk_rmbf);
```

Parameters:

ID	mbfid	Message buffer ID
T_RMBF	*pk_rmbf	Pointer to the packet where the message buffer state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMBF	*pk_rmbf	Pointer to the packet where the message buffer state is stored

Packet Structure:

```
typedef struct {  
    ID      stskid;      0      2      ID of the task at the head of the  
                                   queue waiting to send a message  
    ID      rtskid;      +2     2      ID of the task at the head of the  
                                   queue waiting to receive a message  
    UINT    msgcnt;      +4     4      Number of messages in message buffer  
    SIZE    fmbfsz;      +8     4      Size of free buffer (number of bytes)  
}T_RMBF;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmbf is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbfid ≤ 0 (2) mbfid > CFG_MAXMBFID
E_NOEXS	[k]	Undefined (1) Message buffer specified by mbfid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the message buffer specified by parameter mbfid and returns the ID of the task waiting to send a message (stskid), task waiting to receive a message (rtskid), the number of the messages stored in the message buffer (msgcnt), and the available free buffer size (fmbfsz) to the area indicated by pk_rmbf.

If no task is waiting to receive or send a message in the target message buffer, TSK_NONE (0) is returned as wtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rmbf, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_rmbf
 - size = sizeof(T_RMBF)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.16 Memory Pool Management (Fixed-Size Memory Pool)

Table 6.30 Service Calls for Memory Pool Management (Fixed-Size Memory Pool)

Service Call ¹	Description	System State ²
		T/E/D/U/L/C
cre_mpf [s]	Creates fixed-size memory pool	T/E/D/U
icre_mpf		N/E/D/U
icra_mpf	Creates fixed-size memory pool and specifies access permission vectors	See note 3 below
acre_mpf	Creates fixed-size memory pool and assigns fixed-size memory pool ID automatically	T/E/D/U
iacre_mpf		N/E/D/U
del_mpf	Deletes fixed-size memory pool	T/E/D/U
get_mpf [S]	Acquires fixed-size memory block	T/E/U
pget_mpf [S]	Polls and acquires fixed-size memory block	T/E/D/U
ipget_mpf		N/E/D/U
tget_mpf [S]	Acquires fixed-size memory block with timeout function	T/E/U
rel_mpf [S]	Returns fixed-size memory block	T/E/D/U
irel_mpf		N/E/D/U
ref_mpf	Refers to fixed-size memory pool state	T/E/D/U
iref_mpf		N/E/D/U

Notes: 1. [S]: Standard profile service calls

[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context

N: Can be called in a non-task context

E: Can be called in dispatch-enabled state

D: Can be called in dispatch-disabled state

U: Can be called in CPU-unlocked state

L: Can be called in CPU-locked state

C: Can be called from CPU exception handler

3. icra_mpf is dedicated to use in the initial definition routines created by the configurator. If it is used outside the initial definition routines, correct operation is not guaranteed.

Table 6.31 Fixed-Size Memory Pool Specifications

Item	Description
Fixed-size memory pool ID	1 to CFG_MAXMPFID (32767 max.)
Fixed-size memory pool attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis TA_TPRI: Wait task queue is managed on the current priority

6.16.1 Create Fixed-Size Memory Pool (cre_mpf, icre_mpf, acre_mpf, iacre_mpf)

C-Language API:

```
ER ercd = cre_mpf(ID mpfid, T_CMPF *pk_cmpf);
ER ercd = icre_mpf(ID mpfid, T_CMPF *pk_cmpf);
ER_ID mpfid = acre_mpf(T_CMPF *pk_cmpf);
ER_ID mpfid = iacre_mpf(T_CMPF *pk_cmpf);
```

Parameters:

T_CMPF	*pk_cmpf	Pointer to the packet where the fixed-size memory pool creation information is stored
<cre_mpf, icre_mpf>		
ID	mpfid	Fixed-size memory pool ID

Return Parameters:

<cre_mpf, icre_mpf>		
ER	ercd	Normal termination (E_OK) or error code
<acre_mpf, iacre_mpf>		
ER_ID	mpfid	Created fixed-size memory pool ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    mpfatr;      0    4    Fixed-size memory pool attribute
    UINT   blkcnt;      +4   4    Number of blocks in memory pool
    UINT   blkksz;      +8   4    Block size of fixed-size memory pool
                                   (number of bytes)
    VP     mpf;         +12  4    Start address of the fixed-size
                                   memory pool area
    VP     mpfmb;       +16  4    Start address of the fixed-size
                                   memory pool management area
}T_CMPF;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) mpfatr is invalid.
E_PAR	[p]	Parameter error (1) blkcnt = 0 (2) blkksz = 0 (3) mpf = NULL and TSZ_MPF(blkcnt, blkksz) > CFG_SYSPOOLSZ (4) pk_cmpf is not a 4-byte boundary address. (5) mpf != NULL and mpf is not a 4-byte boundary address.

E_ID	[p]	Invalid ID number (1) mpfid ≤ 0 (2) mpfid > CFG_MAXMPFID
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the system pool (2) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_mpf)
E_OBJ	[k]	Invalid object state (1) Fixed-size memory pool specified by mpfid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls `cre_mpf` and `icre_mpf` create a fixed-size memory pool with the ID specified by `mpfid` using the contents specified by `pk_mpf`.

Service calls `acre_mpf` and `iacre_mpf` search for an unused fixed-size memory pool ID, create a fixed-size memory pool for that ID with the contents specified by parameter `pk_mpf`, and return the ID as a return parameter. The range to search for an undefined fixed-size memory pool ID is 1 to `CFG_MAXMPFID`.

Parameter `mpfatr` specifies the order of the tasks in the queue waiting to acquire a memory block.

`mpfatr := (TA_TFIFO || TA_TPRI)`

- `TA_TFIFO` (H'00000000): Task queue waiting to acquire a memory block is managed on a FIFO basis
- `TA_TPRI` (H'00000001): Task queue waiting to acquire a memory block is managed by the current priority

Parameter `blkcnt` specifies the total number of memory blocks in the memory pool to be created.

The size of the memory block is specified by `blksz`. `blksz` is rounded up to a multiple of four during processing.

Parameter `mpf` specifies the start address of a free area to be used as a fixed-size memory pool. The kernel allocates a `TSZ_MPF(blkcnt, blksz)`-byte area starting from address `mpf` as a fixed-size memory pool. Note that the kernel does not check which domain can access the specified area. For example, if an address in the P1 or P2 area is specified for a fixed-size memory pool area, the area cannot be accessed from a user domain, but the kernel does not detect it.

When the memory object protection function is selected:

An area that can be read or written to from the kernel domain must be specified as a fixed-size memory pool area. If this rule is violated, an E_MACV error will be returned.

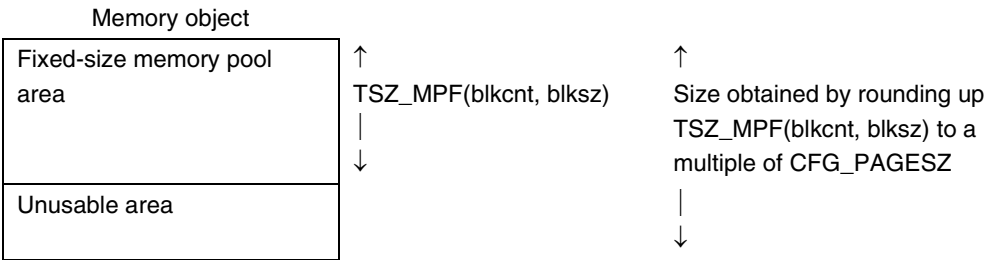
When NULL is specified for mpf, the kernel allocates a TSZ_MPF(blkcnt, blksz)-byte memory pool area in the system pool. At this time, the kernel consumes an area in the resource pool to manage the allocated memory pool area. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.2 (5), Fixed-size memory pool
System pool consumption → Section 14.2 (2), When fixed-size memory pool is created

When the memory object protection function is selected:

The memory pool area allocated in the system pool by the kernel is a memory object having the following attributes.

- (1) Size: TSZ_MPF(blkcnt, blksz) is rounded up to a multiple of CFG_PAGESZ.
- However, note that only the bytes calculated by TSZ_MPF(blkcnt, blksz) can be used for a memory pool. As shown in the following figure, since the unusable area is included in the memory object where a fixed-size memory pool area is allocated, the unusable area has the same access permission as the memory pool area, but the unusable area is not handled as a memory pool.



- (2) Domain: When the service call is issued in a task context, the domain of the issuing task is assigned to the memory pool, which is the same as the domain ID that can be obtained by calling get_did. When the service call is issued in a non-task context, the kernel domain is assigned.

- (3) Memory attribute: TA_RW|TA_CACHE|TA_WBACK
- (4) Access permission vector: An appropriate vector is specified so that only the assigned domain can read or write to the memory pool as follows.
 For the kernel domain: TACT_KERNEL
 For a user domain: TACT_PRW(domid)
 (domid is the ID of the domain where the target memory pool is assigned)

Pointer mpfmb is ignored in this kernel. To ensure the portability of programs, specify NULL for mpfmb.

Regardless of whether NULL is specified for mpf, the kernel consumes an area in the resource pool to manage the memory blocks in the created fixed-size memory pool. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.2 (5), Fixed-size memory pool

Fixed-size memory pools can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for pk_cmpf, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_cmpf
 - size = sizeof(T_CMPF)
 - domid = Domain of the caller
 - pmmode = TPM_READ
- (2) When pk_cmpf->mpf != NULL, the kernel does not have a read/write access permission for the TSZ_MPF(blkcnt, blksize) byte area starting from address mpf, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_cmpf->mpf
 - size = TSZ_MPF(blkcnt, blksize)
 - domid = Kernel domain
 - pmmode = TPM_READ|TPM_WRITE

6.16.2 Create Fixed-Size Memory Pool and Specify Access Permission Vectors (icra_mpf)

C-Language API:

```
ER ercd = icra_mpf(ID mpfid, T_CMPF *pk_cmpf, ACVCT *p_acvct);
```

Parameters:

ID	mpfid	Fixed-size memory pool ID
T_CMPF	*pk_cmpf	Pointer to the packet where the fixed-size memory pool creation information is stored
ACVCT	p_acvct	Pointer to the packet where the access permission vectors are stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {
    ATR  mpfatr;      0    4    Fixed-size memory pool attribute
    UINT blkcnt;      +4   4    Number of blocks in memory pool
    UINT blksz;       +8   4    Block size of fixed-size memory pool
                                (number of bytes)
    VP   mpf;         +12  4    Start address of the fixed-size
                                memory pool area
    VP   mpfmb;       +16  4    Start address of the fixed-size
                                memory pool management area
}T_CMPF;
typedef struct {
    ACPTN acptn1;      0    4    Write access permission pattern
    ACPTN acptn2;      +4   4    Read access permission pattern
}ACVCT;
```

Error Codes:

E_PAR	[p]	Parameter error (1) TSZ_MPF(blkcnt, blksz) > CFG_SYSPOOLSZ)
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the system pool (2) Insufficient space in the resource pool

Function:

Service call icra_mpf creates a fixed-size memory pool with the ID specified by mpfid using the contents specified by pk_cmpf and p_acvct.

This service call must not be issued in any application. This service call is issued only in the initial definition routines created by the configurator when the memory object protection function is selected and creation of fixed-size memory pools is specified through the configurator. This service call is implemented only for this purpose, and most error detection functions are omitted.

This service call differs from `cre_mpf` in the following points.

- (1) Only NULL can be specified for `pk_cmpf->mpf`, that is, a memory pool area is always allocated in the system pool. No specific address can be specified for a memory pool area.
- (2) Through parameter `p_acvct`, access permission vectors can be specified for the memory pool area created as a memory object. However, when the memory protection function is not selected, `p_acvct` is ignored.
- (3) The `E_MACV` error is never detected.

6.16.3 Delete Fixed-Size Memory Pool (del_mpf)

C-Language API:

```
ER ercd = del_mpf(ID mpfid);
```

Parameters:

ID	mpfid	Fixed-size memory pool ID
----	-------	---------------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{mpfid} \leq 0$ (2) $\text{mpfid} > \text{CFG_MAXMPFID}$
E_CTX	[k]	Context error (1) Called in a non-task context.
E_NOEXS	[k]	Undefined (1) Fixed-size memory pool specified by mpfid does not exist.

Function:

Service call del_mpf deletes the fixed-size memory pool specified by mpfid.

No error will occur even if there is a task waiting to acquire a memory block in the fixed-size memory pool area indicated by mpfid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

On deletion, the memory pool area allocated in the system pool and the management area allocated in the resource pool are released.

The kernel will not perform any processing and delete the memory pool even when a block in the memory pool has not been released yet.

6.16.4 Get Fixed-Size Memory Block (get_mpf, pget_mpf, ipget_mpf, tget_mpf)

C-Language API:

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

Parameters:

ID	mpfid	Fixed-size memory pool ID
VP	*p_blk	Pointer to the area where the start address of the memory block is to be returned
<tget_mpf>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
VP	*p_blk	Pointer to the area where the start address of the memory block is stored

Error Codes:

E_PAR	[p]	Parameter error (1) tmout ≤ -2 (2) p_blk is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mpfid ≤ 0 (2) mpfid > CFG_MAXMPFID
E_CTX	[k]	Context error (only for get_mpf and tget_mpf) (1) Called in dispatch-pended state
E_NOEXS	[k]	Undefined (1) Fixed-size memory pool specified by mpfid does not exist.
E_RLWAI	[k]	WAITING state was forcibly cancelled (only for get_mpf and tget_mpf). (1) rel_wai service call was issued in the WAITING state (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Fixed-size memory pool specified by mpfid was deleted.
E_MACV	[m]	Memory access violation

Function:

Each service call gets one memory block from the fixed-size memory pool specified by `mpfid`, and returns the start address of the acquired memory block to the area specified by `p_blk`.

If there are tasks already waiting for a memory block or if no task is waiting but there is no memory block available in the fixed-size memory pool, the task issued service call `get_mpf` or `tget_mpf` is placed in the memory acquiring wait queue, and the task that issued service call `pget_mpf` or `ipget_mpf` immediately returns error code `E_TMOUT`. The queue is managed according to the attribute specified at creation.

Parameter `tmout` of service call `tget_mpf` specifies the timeout period. If a positive value is specified for parameter `tmout`, error code `E_TMOUT` is returned when the `tmout` period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `pget_mpf` will be performed. If `tmout = TMO_FEVR (-1)` is specified, timeout monitoring is not performed. In other words, the same operation as for service call `get_mpf` will be performed.

If a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for `p_blk`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base = p_blk`
- `size = sizeof(VP)`
- `domid = Domain of the caller`
- `pmmode = TPM_READ|TPM_WRITE`

6.16.5 Release Fixed-Size Memory Block (rel_mpf, irel_mpf)

C-Language API:

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

Parameters:

ID	mpfid	Fixed-size memory pool ID
VP	blk	Start address of memory block

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR		Parameter error
	[p]	(1) blk is not a 4-byte boundary address.
	[k]	(2) A value other than the start address of a memory block or released blk was specified.
E_ID	[p]	Invalid ID number
		(1) mpfid ≤ 0
		(2) mpfid > CFG_MAXMPFID
E_NOEXS	[k]	Undefined
		(1) Fixed-size memory pool specified by mpfid does not exist.

Function:

Each service call returns the memory block specified by blk to the fixed-size memory pool indicated by mpfid.

The start address of a memory block acquired by service call get_mpf, pget_mpf, ipget_mpf, or tget_mpf must be specified for parameter blk.

If there are tasks waiting to get a memory block in the target fixed-size memory pool, the memory block returned by this service call is passed to the task at the head of the wait queue, releasing it from the WAITING state.

6.16.6 Refer to Fixed-Size Memory Pool State (ref_mpf, iref_mpf)

C-Language API:

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

Parameters:

ID	mpfid	Fixed-size memory pool ID
T_RMPF	*pk_rmpf	Pointer to the packet where the fixed-size memory pool state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMPF	*pk_rmpf	Pointer to the packet where the fixed-size memory pool state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;          0      2      Wait task ID  
    UINT    fblkcnt;        +4      4      Number of blocks of memory space  
                                           available  
}T_RMPF;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmpf is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mpfid ≤ 0 (2) mpfid > CFG_MAXMPFID
E_NOEXS	[k]	Undefined (1) Fixed-size memory pool specified by mpfid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the fixed-size memory pool specified by mpfid.

Each service call returns the wait task ID (wtskid) and the number of blocks of memory space available (fblkcnt) to the area indicated by pk_rmpf.

If there is no task waiting for the specified memory pool, TSK_NONE (0) is returned as wtskid.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rmpf, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_rmpf
 - size = sizeof(T_RMPF)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.17 Memory Pool Management (Variable-Size Memory Pool)

Table 6.32 Service Calls for Memory Pool Management (Variable-Size Memory Pool)

Service Call ^{*1}	Description	System State ^{*2}
		T/N/E/D/U/L/C
cre_mpl	Creates variable-size memory pool	T/E/D/U
icre_mpl		N/E/D/U
ivcra_mpl	Creates variable-size memory pool and specifies access permission vectors	See note 3 below
acre_mpl	Creates variable-size memory pool and assigns variable-size memory pool ID automatically	T/E/D/U
iacre_mpl		N/E/D/U
del_mpl	Deletes variable-size memory pool	T/E/D/U
get_mpl	Acquires variable-size memory block	T/E/U
pget_mpl	Polls and acquires variable-size memory block	T/E/D/U
ipget_mpl		N/E/D/U
tget_mpl	Acquires variable-size memory block with timeout function	T/E/U
rel_mpl	Returns variable-size memory block	T/E/D/U
irel_mpl		N/E/D/U
ref_mpl	Refers to variable-size memory pool state	T/E/D/U
iref_mpl		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler
3. ivcra_mpl is dedicated to use in the initial definition routines created by the configurator. If it is used outside the initial definition routines, correct operation is not guaranteed.

Table 6.33 Variable-Size Memory Pool Specifications

Item	Description
Variable-size memory pool ID	1 to CFG_MAXMPLID (32767 max.)
Variable-size memory pool attributes	TA_TFIFO: Wait task queue is managed on a FIFO basis VTA_UNFRAGMENT: Sector management (reducing fragmentation in free space) VTA_ALIGN16: Memory block addresses are adjusted to 16-byte boundaries. VTA_ALIGN32: Memory block addresses are adjusted to 32-byte boundaries.

Also refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

6.17.1 Create Variable-Size Memory Pool (cre_mpl, icre_mpl, acre_mpl, iacre_mpl)

C-Language API:

```
ER ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl);
ER ercd = icre_mpl(ID mplid, T_CMPL *pk_cmpl);
ER_ID mplid = acre_mpl(T_CMPL *pk_cmpl);
ER_ID mplid = iacre_mpl(T_CMPL *pk_cmpl);
```

Parameters:

T_CMPL	*pk_cmpl	Pointer to the packet where the variable-size memory pool creation information is stored
<cre_mpl, icre_mpl>		
ID	mplid	Variable-size memory pool ID

Return Parameters:

<cre_mpl, icre_mpl>		
ER	ercd	Normal termination (E_OK) or error code
<acre_mpl, iacre_mpl>		
ER_ID	mplid	Created variable-size memory pool ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    mplatr;          0    4    Variable-size memory pool attribute
    SIZE   mplsz;          +4    4    Size of memory pool (number of
                                     bytes)
    VP     mpl;            +8    4    Start address of the variable-size
                                     memory pool area
    VP     mplmb;          +12   4    Start address of management area for
                                     variable-size memory pool
    UINT   minblksz;       +16   4    Minimum block size
    UINT   sctnum;        +20   4    Maximum number of sectors
}T_CMPL;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) mplatr is invalid.
---------	-----	--

E_PAR	[p]	Parameter error <ul style="list-style-type: none"> (1) pk_cmpl is not a 4-byte boundary address. (2) mplsz < TSZ_MPL(1,4) (3) mpl = NULL and mplsz > CFG_SYSPOLLSZ (4) mpl != NULL and mpl is not a 4-byte boundary address. (5) When attribute VTA_UNFRAGMENT is specified, minblksz is 0. (6) When attributes VTA_UNFRAGMENT and VTA_ALIGN16 are specified, minblksz is not a multiple of 16. (7) When attributes VTA_UNFRAGMENT and VTA_ALIGN32 are specified, minblksz is not a multiple of 32. (8) When attribute VTA_UNFRAGMENT is specified and neither attribute VTA_ALIGN16 nor VTA_ALIGN32 is specified, minblksz is not a multiple of 4. (9) When attribute VTA_UNFRAGMENT is specified, sctnum = 0. (10) When attribute VTA_UNFRAGMENT is specified, mplsz < minblksz * 32.
E_ID	[p]	Invalid ID number <ul style="list-style-type: none"> (1) mplid ≤ 0 (2) mplid > CFG_MAXMPFID
E_NOMEM	[k]	Insufficient memory <ul style="list-style-type: none"> (1) Insufficient space in the system pool (2) Insufficient space in the resource pool
E_NOID	[k]	No ID available (only for acre_mpl)
E_OBJ	[k]	Invalid object state <ul style="list-style-type: none"> (1) Variable-size memory pool specified by mplid already exists.
E_MACV	[m]	Memory access violation

Function:

Service calls cre_mpl and icre_mpl create a variable-size memory pool with the ID specified by mplid using the contents specified by pk_cmpl.

Service calls acre_mpl and iacre_mpl search for an unused variable-size memory pool ID, create a variable-size memory pool for that ID with the contents specified by parameter pk_cmpl, then return the ID as a return parameter. The range to search for the variable-size memory pool ID is 1 to CFG_MAXMPFID.

(1) mplatr

Specify the logical OR of the following values for mplatr.

(a) Order of tasks in the queue for waiting for memory block acquisition

Only TA_TFIFO can be specified.

— TA_TFIFO (H'00000000): Task queue waiting for memory is managed on a FIFO basis.

(b) Management method

VTA_UNFRAGMENT can be specified.

— VTA_UNFRAGMENT (H'80000000): Sector management (reducing fragmentation in free space)

The VTA_UNFRAGMENT attribute is suitable for a memory pool from which a large number of small memory blocks are to be acquired. When this attribute is specified, small blocks are collectively allocated in specialized contiguous areas to leave larger possible contiguous areas.

Only when attribute VTA_UNFRAGMENT is specified, minblksz and sctnum become valid.

When sctnum is set to a larger value than $\text{mplsz} / (\text{minblksz} \times 32)$, $\text{mplsz} / (\text{minblksz} \times 32)$ is assumed.

For details, refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

When attribute VTA_UNFRAGMENT is specified, the kernel consumes an area in the resource pool to manage the sectors. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.2 (6), Variable-size memory pool

(c) Alignment of memory block addresses

For alignment of the addresses of memory blocks to be acquired from a memory pool, any one of the following attributes can be specified when necessary.

— VTA_ALIGN16 (H'00000010): Memory block addresses are adjusted to 16-byte boundaries.

— VTA_ALIGN32 (H'00000020): Memory block addresses are adjusted to 32-byte boundaries.

When neither one of them is specified, memory block addresses are adjusted to 4-byte boundaries.

(2) mplsz

Parameter mplsz specifies the size of the variable-size memory pool to be created. The following macro is provided to estimate the approximate size to be specified for mplsz.

`SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)`

Approximate size (bytes) of a variable-size memory pool area required to hold the blkcnt number of blksz-byte memory blocks

Note that `mplsz` is rounded up to a multiple of four during processing. In the following description, `mplsz` indicates a multiple of four after being rounded up.

(3) `mpl`

Parameter `mpl` specifies the start address of a free area to be used as a variable-size memory pool. The kernel allocates an `mplsz`-byte area starting from address `mpl` as a fixed-size memory pool. When attribute `VTA_ALIGN16` or `VTA_ALIGN32` is specified, the actual memory pool area to be used starts from an address obtained by adjusting address `mpl` to a 16-byte or 32-byte boundary, which means that the usable memory pool size decreases for the adjusted size.

Note that the kernel does not check which domain can access the specified area. For example, if an address in the P1 or P2 area is specified for a variable-size memory pool area, the area cannot be accessed from a user domain, but the kernel does not detect it.

When the memory object protection function is selected:

An area that can be read or written to from the kernel domain must be specified as a variable-size memory pool area. If this rule is violated, an `E_MACV` error will be returned.

When `NULL` is specified for `mpl`, the kernel allocates an `mplsz`-byte memory pool area in the system pool. At this time, the kernel consumes an area in the resource pool to manage the allocated memory pool area. For details, refer to the following.

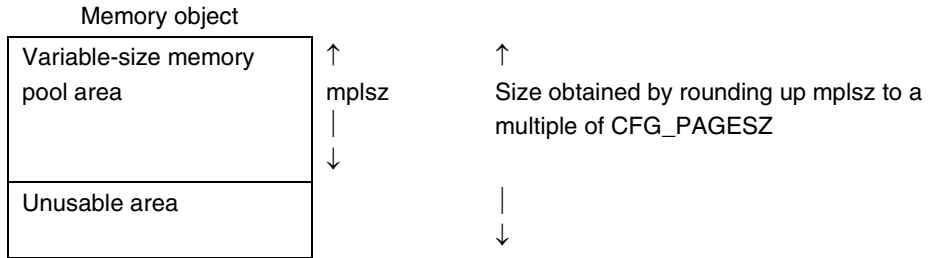
Reference: Resource pool consumption → Section 13.2.2(6), Variable-size memory pool
System pool consumption → Section 14.2 (3), When variable-size memory pool is created

When the memory object protection function is selected:

The memory pool area allocated in the system pool by the kernel is a memory object having the following attributes.

(1) Size: `mplsz` is rounded up to a multiple of `CFG_PAGESZ`.

However, note that only the `mplsz` bytes can be used for a memory pool. As shown in the following figure, since the unusable area is included in the memory object where a variable-size memory pool area is allocated, the unusable area has the same access permission as the memory pool area, but the unusable area is not handled as a memory pool.



- (2) Domain: When the service call is issued in a task context, the domain of the issuing task is assigned to the memory pool, which is the same as the domain ID that can be obtained by calling `get_did`. When the service call is issued in a non-task context, the kernel domain is assigned.
- (3) Memory attribute: TA_RW|TA_CACHE| TA_WBACK
- (4) Access permission vector: An appropriate vector is specified so that only the assigned domain can read or write to the memory pool as follows.
 For the kernel domain: TACT_KERNEL
 For a user domain: TACT_PRW(domid)
 (domid is the ID of the domain where the target memory pool is assigned)
-

(4) minblksz and sctnum

These are parameters not defined in the μ ITRON specification.

These parameters are valid only when attribute VTA_UNFRAGMENT is specified. For details, refer to the above description of attribute VTA_UNFRAGMENT.

(5) mplmb

mplmb is a parameter not defined in the μ ITRON specification.

Parameter mplmb is ignored in this kernel. To ensure the portability of programs, specify NULL for mplmb.

Variable-size memory pools can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for `pk_cmpl`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- base = pk_cmpl
 - size = sizeof(T_CMPL)
 - domid = Domain of the caller
 - pmmode = TPM_READ
- (2) When pk_cmpl->mpl != NULL, the kernel does not have a read/write access permission for the mplsz-byte area starting from address mpl, which means that an error will be returned if prb_mem is issued with the following parameters.
- base = pk_cmpl->mpl
 - size = pk_cmpl->mplsz
 - domid = Kernel domain
 - pmmode = TPM_READ|TPM_WRITE

6.17.2 Create Variable-Size Memory Pool and Specify Access Permission Vectors (ivcra_mpl)

C-Language API:

```
ER ercd = ivcra_mpl (ID mplid, T_CMPL *pk_cmpl, ACVCT *p_acvct);
```

Parameters:

ID	mplid	Variable-size memory pool ID
T_CMPL	*pk_cmpl	Pointer to the packet where the variable-size memory pool creation information is stored
ACVCT	*p_acvct	Pointer to the packet where the access permission vectors are stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {
    ATR    mplatr;      0    4    Variable-size memory pool attribute
    SIZE   mplsz;       +4   4    Size of memory pool (number of
                                bytes)
    VP     mpl;         +8   4    Start address of the variable-size
                                memory pool area
    VP     mplmb;       +12  2    Start address of management area for
                                variable-size memory pool
    UINT   minblksz;    +16  4    Minimum block size
    UINT   sctnum;      +20  4    Maximum number of sectors
}T_CMPL;
typedef struct {
    ACPTN   acptn1;      0    4    Write access permission pattern
    ACPTN   acptn2;     +4   4    Read access permission pattern
}ACVCT;
```

Error Codes:

E_PAR	[p]	Parameter error (1) mplsz > CFG_SYSPOOLSZ
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the system pool (2) Insufficient space in the resource pool

Function:

Service call ivcra_mpl creates a variable-size memory pool with the ID specified by mplid using the contents specified by pk_cmpl and p_acvct.

This service call must not be issued in any application. This service call is issued only in the initial definition routines created by the configurator when the memory object protection function is selected and creation of variable-size memory pools is specified through the configurator. This service call is implemented only for this purpose, and most error detection functions are omitted.

This service call differs from `cre_mpl` in the following points.

- (1) Only NULL can be specified for `pk_cmpl->mpl`, that is, a memory pool area is always allocated in the system pool. No specific address can be specified for a memory pool area.
- (2) Through parameter `p_acvct`, access permission vectors can be specified for the memory pool area created as a memory object. However, when the memory protection function is not selected, `p_acvct` is ignored.
- (3) The `E_MACV` error is never detected.

6.17.3 Delete Variable-Size Memory Pool (del_mpl)

C-Language API:

```
ER ercd = del_mpl(ID mplid);
```

Parameters:

ID	mplid	Variable-size memory pool ID
----	-------	------------------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{mplid} \leq 0$ (2) $\text{mplid} > \text{CFG_MAXMPFID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Variable-size memory pool specified by mplid does not exist.

Function:

Service call del_mpl deletes the variable-size memory pool specified by mplid.

No error will occur even if there is a task waiting to acquire a memory block in the variable-size memory pool area. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

On deletion, the memory pool area allocated in the system pool and the management area allocated in the resource pool are released.

The kernel will not perform any processing and delete the memory pool even when a block in the memory pool has not been released yet.

6.17.4 Get Variable-Size Memory Block (get_mpl, pget_mpl, ipget_mpl, tget_mpl)

C-Language API:

```
ER ercd = get_mpl (ID mplid, UINT blksize, VP *p_blk);
ER ercd = pget_mpl (ID mplid, UINT blksize, VP *p_blk);
ER ercd = ipget_mpl (ID mplid, UINT blksize, VP *p_blk);
ER ercd = tget_mpl (ID mplid, UINT blksize, VP *p_blk);
```

Parameters:

ID	mplid	Variable-size memory pool ID
UINT	blksize	Memory block size (number of bytes)
VP	*p_blk	Pointer to the area where the start address of the memory block is to be returned
<tget_mpl>		
TMO	tmout	Timeout specification

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
VP	*p_blk	Pointer to the area where the start address of the memory block is stored

Error Codes:

E_PAR		Parameter error
	[p]	(1) tmout ≤ -2 (2) blksize = 0 (3) p_blk is not a 4-byte boundary address.
	[k]	blksize > (memory pool size specified at creation)
E_ID	[p]	Invalid ID number
		(1) mplid ≤ 0 (2) mplid > CFG_MAXMPFID
E_CTX	[k]	Context error (only for get_mpl and tget_mpl)
		(1) Called in dispatch-pended state
E_NOMEM	[k]	Insufficient memory
		(1) Insufficient space in the resource pool
E_NOEXS	[k]	Undefined
		(1) Variable-size memory pool specified by mplid does not exist.
E_RLWAI	[k]	WAITING state was forcibly cancelled (only for get_mpl and tget_mpl)
		(1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout

E_DLT	[k]	Waiting object deleted (1) The memory pool specified by mplid was deleted.
E_MACV	[m]	Memory access violation

Function:

Each service call acquires a memory block with the size specified by blksize (number of bytes) from the variable-size memory pool indicated by mplid, and returns the start address of the acquired memory block to the area indicated by p_blk.

blksize is rounded up during processing as shown in table 6.34.

Table 6.34 Rounding up blksize

VTA_ALIGN16	VTA_ALIGN32	VTA_UNFRAGMENT	Rounding up blksize
Not specified	Not specified	Not specified	Rounded up to a multiple of 4
Specified	Not specified	Not specified	Rounded up to a multiple of 16
Not specified	Specified	Not specified	Rounded up to a multiple of 32
Not specified	Not specified	Specified	(1) When $\text{blksize} \leq (\text{minblksize} \times 8)$: Rounded up to minblksize, minblksize \times 2, minblksize \times 4, or minblksize \times 8 depending on the value of blksize (2) When $\text{blksize} > (\text{minblksize} \times 8)$: Rounded up to a multiple of 4
Specified	Not specified	Specified	(1) When $\text{blksize} \leq (\text{minblksize} \times 8)$: Rounded up to minblksize, minblksize \times 2, minblksize \times 4, or minblksize \times 8 depending on the value of blksize (2) When $\text{blksize} > (\text{minblksize} \times 8)$: Rounded up to a multiple of 16
Not specified	Specified	Specified	(1) When $\text{blksize} \leq (\text{minblksize} \times 8)$: Rounded up to minblksize, minblksize \times 2, minblksize \times 4, or minblksize \times 8 depending on the value of blksize (2) When $\text{blksize} > (\text{minblksize} \times 8)$: Rounded up to a multiple of 32

After the memory block has been acquired, the size of the free space in the variable-size memory pool will decrease by the size of rounded blksize.

For a memory pool with attribute VTA_ALIGN16 or VTA_ALIGN32, the memory block address is a 16-byte or 32-byte boundary address, respectively.

If there are tasks already waiting for the memory pool, or if no task is waiting but there is no memory block available, the task that issued service call `get_mpl` or `tget_mpl` is placed in the memory block wait queue, and the task that issued service call `pget_mpl` or `ipget_mpl` is immediately returns the error code `E_TMOUT`. The queue is managed on a FIFO basis.

Parameter `tmout` of service call `tget_mpl` specifies the timeout period. If a positive value is specified for parameter `tmout`, error code `E_TMOUT` is returned when the timeout period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `pget_mpl` will be performed. If `tmout = TMO_FEVR (-1)` is specified, timeout watch is not performed. In other words, the same operation as for service call `get_mpl` will be performed.

The kernel consumes an area in the resource pool to manage the memory blocks. If there is not sufficient free space in the resource pool, an `E_NOMEM` error will be returned immediately. This processing is always done regardless of whether a memory block can be acquired immediately or the memory waiting state is entered.

For resource pool consumption, refer to the following.

Reference: Section 13.2.3 (2), Variable-size memory pool: `get_mpl`, `pget_mpl`, `ipget_mpl`, `tget_mpl`

If a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `p_blk`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = p_blk`
 - `size = sizeof(VP)`
 - `domid = Domain of the caller`
 - `pmmode = TPM_READ|TPM_WRITE`

6.17.5 Release Variable-Size Memory Block (rel_mpl, irel_mpl)

C-Language API:

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

Parameters:

ID	mplid	Variable-size memory pool ID
VP	blk	Start address of memory block

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR		Parameter error
	[p]	(1) blk is not a 4-byte boundary address.
	[k]	(2) A value other than the start address of a memory block or released blk was specified.
E_ID	[p]	Invalid ID number
		(1) mplid ≤ 0
		(2) mplid > CFG_MAXMPFID
E_NOEXS	[k]	Undefined
		(1) Variable-size memory pool specified by mplid does not exist.

Function:

Each service call returns the memory block specified by blk to the variable-size memory pool specified by mplid.

The start address of a memory block acquired by service call get_mpl, pget_mpl, ipget_mpl, or tget_mpl must be specified as parameter blk.

After the memory block has been returned, if the target variable-size memory pool has a contiguous free area of the size requested by the task at the head of the memory block acquisition wait queue, a memory block is assigned to that task and the task is released from the WAITING state.

The same process will be done for the remaining tasks in the order of the wait queue if the remaining memory pool size still has enough contiguous free space.

After the memory block has been returned, the area allocated in the resource pool to manage the memory block when the memory block was acquired is released.

6.17.6 Refer to Variable-Size Memory Pool State (ref_mpl, iref_mpl)

C-Language API:

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

Parameters:

ID	mplid	Variable-size memory pool ID
T_RMPL	*pk_rmpl	Pointer to the packet where the variable-size memory pool state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMPL	*pk_rmpl	Pointer to the packet where the variable-size memory pool state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;      0    2    Wait task ID  
    SIZE    fmplsz;      +4   4    Total size of available memory area  
                                   (number of bytes)  
    UINT    fblks;       +8   4    Maximum memory area available  
                                   (number of bytes)  
    SIZE    mplsz;       +12  4    Size of variable-size memory pool  
}T_RMPL;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmpl is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mplid ≤ 0 (2) mplid > CFG_MAXMPFID
E_NOEXS	[k]	Undefined (1) Variable-size memory pool specified by mplid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the status of the variable-size memory pool specified by mplid and returns the wait task ID (wtskid), the current free memory area total size (fmplsz), the maximum memory block size available (fblks), and the size of the variable-size memory pool (mplsz: a parameter not defined in μ ITRON specification) to the area indicated by pk_rmpl. If there is no task waiting to get a memory block, TSK_NONE (0) is returned as wtskid.

The free space is usually fragmented. The maximum contiguous free space is returned to parameter `fbksz`. The block up to the size `fbksz` can be acquired immediately by service call `get_mpl`, `pget_mpl`, `ipget_mpl`, or `tget_mpl`.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `pk_rmpl`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_rmpl`
 - `size` = `sizeof(T_RMPL)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ|TPM_WRITE`

6.18 Time Management (System Clock)

Table 6.35 Service Calls for System Clock Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
set_tim [S]	Sets system clock	T/E/D/U
iset_tim		N/E/D/U
get_tim [S]	Gets system clock	T/E/D/U
iget_tim		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.36 System Clock Management Specifications

Item	Description
System clock value	Unsigned 48 bits
System clock unit	1 ms
System clock update cycle	CFG_TICNUME/CFG_TICDENO [ms]*
System clock initial value (at initialization)	H'000000000000

Note: The values of TIC_NUME and TIC_DENO defined in kernel_macro.h are same as the values of CFG_TICNUME and CFG_TICDENO, respectively.

The system clock is expressed as a 48-bit unsigned integer by using a structure of data type “SYSTIM”. The maximum value of the system clock is shown below.

When $\text{CFG_TICNUME}/\text{CFG_TICDENO} \leq 1$:

Maximum value = $\text{H'ffffffff}/\text{CFG_TICDENO}$

When $\text{CFG_TICNUME}/\text{CFG_TICDENO} > 1$:

Maximum value = H'ffffffff

The system clock is incremented at timer interrupts. If the above maximum value is exceeded, the system clock is initialized to 0.

6.18.1 Set System Clock (set_tim, iset_tim)

C-Language API:

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

Parameters:

SYSTIM	*p_systim	Pointer to the packet where the current time data is stored
--------	-----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    UH    utime;      0    2    Current time data (upper)  
    UW    ltime;      +4   4    Current time data (lower)  
}SYSTIM;
```

Error Codes:

E_PAR	[p]	Parameter error (1) p_systim is not a 4-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call changes the current system clock retained in the system to a value specified by p_systim.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

Even after the system clock is modified, the actual time required to generate the time events (timeout or cyclic handler initiation) that have been requested will not change.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The domain of the caller does not have a read access permission for p_systim, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = p_systim
- size = sizeof(SYSTIM)
- domid = Domain of the caller
- pmmode = TPM_READ

6.18.2 Get System Clock (get_tim, iget_tim)

C-Language API:

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

Parameters:

SYSTIM	*p_systim	Start address of the packet where the current time data is to be returned
--------	-----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
SYSTIM	*p_systim	Start address of the packet where the current time data is stored

Packet Structure:

```
typedef struct {  
    UH    utime;        0    2    Current time data (upper)  
    UW    ltime;        +4   4    Current time data (lower)  
}SYSTIM;
```

Error Codes:

E_PAR	[p]	Parameter error (1) p_systim is not a 4-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call reads the current system clock and returns it to the area indicated by p_systim.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_systim, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = p_systim
- size = sizeof(SYSTIM)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.19 Time Management (Cyclic Handler)

Table 6.37 Service Calls for Cyclic Handler

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_cyc [s]	Creates cyclic handler	T/E/D/U
icre_cyc		N/E/D/U
acre_cyc	Creates cyclic handler and assigns cyclic handler ID automatically	T/E/D/U
iacre_cyc		N/E/D/U
del_cyc	Deletes cyclic handler	T/E/D/U
sta_cyc [S]	Starts cyclic handler operation	T/E/D/U
ista_cyc		N/E/D/U
stp_cyc [S]	Stops cyclic handler operation	T/E/D/U
istp_cyc		N/E/D/U
ref_cyc	Refers to the cyclic handler state	T/E/D/U
iref_cyc		N/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.38 Cyclic Handler Specifications

Item	Description
Cyclic handler ID	1 to CFG_MAXCYCID (254 max.)
Cyclic handler attributes	TA_HLNG: The handler is written in a high-level language. TA_ASM: The handler is written in assembly language. TA_STA: Starts cyclic handler operation. TA_PHS: Reserves initiation phase.

6.19.1 Create Cyclic Handler (cre_cyc, icre_cyc, acre_cyc, iacre_cyc)

C-Language API:

```
ER ercd = cre_cyc (ID cycid, T_CCYC *pk_ccyc);
ER ercd = icre_cyc (ID cycid, T_CCYC *pk_ccyc);
ER_ID cycid = acre_cyc (T_CCYC *pk_ccyc);
ER_ID cycid = iacre_cyc (T_CCYC *pk_ccyc);
```

Parameters:

T_CCYC	*pk_ccyc	Pointer to the packet where the cyclic handler creation information is stored
<cre_cyc, icre_cyc>		
ID	cycid	Cyclic handler ID

Return Parameters:

<cre_cyc, icre_cyc>		
ER	ercd	Normal termination (E_OK) or error code
<acre_cyc, iacre_cyc>		
ER_ID	cycid	Created cyclic handler ID number (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    cycatr;    0    4    Cyclic handler attribute
    VP_INT exinf;    +4    4    Extended information
    FP     cychdr;    +8    4    Cyclic handler address
    RELTIM cyctim;    +12   4    Cyclic handler initiation cycle
    RELTIM cycphs;    +16   4    Cyclic handler initiation phase
}T_CCYC;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) cycatr is invalid.
E_PAR	[p]	Parameter error (1) cyctim = 0 (2) cycphs > cyctim (3) pk_ccyc is not a 4-byte boundary address. (4) cychdr is an odd address.
E_ID	[p]	Invalid ID number (1) cycid ≤ 0 (2) cycid > CFG_MAXCYCID
E_NOID	[k]	No ID available (only for acre_cyc)
E_OBJ	[k]	Invalid object state (1) Cyclic handler specified by cycid already exists.

Function:

Each service call creates a cyclic handler. The cyclic handler is a time event handler for a non-task context and is initiated at specified time intervals. The cyclic handler is assigned to the kernel domain and is executed in privileged mode.

The following describes each parameter function.

(1) cycid

For service calls `cre_cyc` and `icre_cyc`, specify a value within the range from 1 to `CFG_MAXCYCID` for parameter `cycid`. Service calls `acre_cyc` and `iacre_cyc` search for an unused cyclic handler ID, define a cyclic handler for that ID with the contents specified by parameter `pk_ccyc`, and return the defined cyclic handler ID as a return parameter.

(2) cycatr

Specify the logical OR of the following values for `cycatr`.

The cyclic handler is always assigned to the kernel domain.

In the PX specification, `TA_DOM(TDOM_KERNEL)` must be specified for `cycatr` so that the cyclic handler is assigned to the kernel domain. To ensure the portability of programs, always specify `TA_DOM(TDOM_KERNEL)`. Note that `TA_DOM()` is ignored in this kernel, and even if a user domain is assigned through `TA_DOM()`, no error will occur and the kernel domain is always assumed.

(a) Language

Specify either one of the following values.

- `TA_HLNG` (H'00000000): High-level language
- `TA_ASM` (H'00000001): Assembly language

(b) Initiating the cyclic handler

Specify `TA_STA` to initiate the handler immediately. When `TA_STA` is specified, the cyclic handler is set to the operating state after it is created. When `TA_STA` is not specified, the cyclic handler does not operate until service call `sta_cyc` or `ista_cyc` is issued.

- `TA_STA` (H'00000002): Initiates the cyclic handler operation.

(c) Keeping the initiation phase

When `TA_PHS` is specified, the initiation phase of the cyclic handler is kept before activating the cyclic handler, and the next time to initiate the handler is determined.

When TA_PHS is not specified, once the cyclic handler is stopped by service call `stp_cyc` or `istp_cyc`, the cyclic handler is initiated at intervals `cycetim` after the next `sta_cyc` or `ista_cyc` service call, that is, the first time the cyclic handler is initiated is `cycetim` after a `sta_cyc` or `ista_cyc` service call.

When TA_PHS is specified, even after the cyclic handler is stopped by service call `stp_cyc` or `istp_cyc`, the kernel continues to manage the correct time to initiate the cyclic handler. If service call `sta_cyc` or `ista_cyc` is issued after that, the kernel initiates the cyclic handler when the correct time is reached.

— TA_PHS (H'00000004): Retains the initiation phase.

(3) `exinf`

Parameter `exinf` can be widely used by the user, for example, to set information concerning cyclic handlers to be defined. `exinf` is passed to the cyclic handler as a parameter.

(4) `cychdr`

Parameter `cychdr` specifies the start address of the cyclic handler.

(5) `cycetim` and `cycphs`

Parameter `cycetim` specifies the handler initiation cycle, and parameter `cycphs` specifies the handler initiation phase.

`cycphs` is valid only when either TA_STA or TA_PHS is specified. If `cycphs` is valid, the first time to initiate the cyclic handler is the time when the specified `cycphs` (initiation phase time) or a longer time has passed since a cyclic handler creating service call is issued. At this time, if start of the cyclic handler is specified through the TA_STA attribute or service call `stp_cyc` or `istp_cyc`, the cyclic handler is actually executed.

If a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for `cycetim` and `cycphs` is $H'7\text{ffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

The cyclic handler can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

(1) The domain of the caller does not have a read access permission for `pk_ccyc`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base` = `pk_ccyc`
- `size` = `sizeof(T_CCYC)`
- `domid` = Domain of the caller

— pmmode = TPM_READ

(2) The kernel domain does not have a read access permission for pk_ccyc->cychdr, which means that an error will be returned if prb_mem is issued with the following parameters.

— base = pk_ccyc->cychdr

— size = 1

— domid = Kernel domain

— pmmode = TPM_READ

6.19.2 Delete Cyclic Handler (del_cyc)

C-Language API:

```
ER ercd = del_cyc (ID cycid);
```

Parameters:

ID	cycid	Cyclic handler ID
----	-------	-------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) cycid ≤ 0 (2) cycid > CFG_MAXCYCID
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Cyclic handler specified by cycid does not exist.

Function:

Service call del_cyc deletes the cyclic handler specified by parameter cycid.

6.19.3 Start Cyclic Handler Operation (sta_cyc, ista_cyc)

C-Language API:

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

Parameters:

ID	cycid	Cyclic handler ID
----	-------	-------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) cycid ≤ 0 (2) cycid > CFG_MAXCYCID
E_NOEXS	[k]	Undefined (1) Cyclic handler specified by cycid does not exist.

Function:

Each service call causes the cycle handler specified by cycid to enter the operational state.

If TA_PHS is not specified as a cyclic handler attribute, the cyclic handler is started each time the start cycle has passed, based on the timing at which service call sta_cyc or ista_cyc is issued.

If the cyclic handler specified by cycid is in the operational state and TA_PHS is not specified as its attribute, the next timing of initiation is set after the service call is issued.

If the cyclic handler specified by cycid is in the operating state and TA_PHS is specified as its attribute, the next timing of initiation is not set.

6.19.4 Stop Cyclic Handler Operation (stp_cyc, istp_cyc)

C-Language API:

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

Parameters:

ID	cycid	Cyclic handler ID
----	-------	-------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{cycid} \leq 0$ (2) $\text{cycid} > \text{CFG_MAXCYCID}$
E_NOEXS	[k]	Undefined (1) Cyclic handler specified by cycid does not exist.

Function:

Each service call causes the cyclic handler specified by parameter cycid to enter the not-operating state.

6.19.5 Refer to Cyclic Handler State (ref_cyc, iref_cyc)

C-Language API:

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

Parameters:

ID	cycid	Cyclic handler ID
T_RCYC	*pk_rcyc	Pointer to the packet where the cyclic handler state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RCYC	*pk_rcyc	Pointer to the packet where the cyclic handler state is stored

Packet Structure:

```
typedef struct {  
    STAT    cycstat;      0    4    Cyclic handler operating state  
    RELTIM  lefttim;     +4    4    Remaining time until the cyclic  
                                    handler is initiated  
}T_RCYC;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rcyc is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) cycid ≤ 0 (2) cycid > CFG_MAXCYCID
E_NOEXS	[k]	Undefined (1) Cyclic handler specified by cycid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call reads the cyclic handler state specified by cycid and returns the cyclic handler operating state (cycstat) and the time remaining until the cyclic handler is initiated (lefttim) to the area indicated by parameter pk_rcyc.

The target cyclic handler operating state is returned to parameter cycstat.

- TCYC_STP (H'00000000): The cyclic handler is not in the operational state
- TCYC_STA (H'00000001): The cyclic handler is in the operational state

The relative time until the target cyclic handler is next initiated is returned to parameter `lefttim`. When the target cyclic handler is not in the operational state, `lefttim` is undefined.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `pk_rcyc`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_rcyc`
 - `size` = `sizeof(T_RCYC)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ|TPM_WRITE`

6.20 Time Management (Alarm Handler)

Table 6.39 Service Calls for Alarm Handler

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_alm	Creates alarm handler	T/E/D/U
icre_alm		N/E/D/U
acre_alm	Creates alarm handler and assigns alarm handler ID automatically	T/E/D/U
iacre_alm		N/E/D/U
del_alm	Deletes alarm handler	T/E/D/U
sta_alm	Starts alarm handler operation	T/E/D/U
ista_alm		N/E/D/U
stp_alm	Stops alarm handler operation	T/E/D/U
istp_alm		N/E/D/U
ref_alm	Refers to the alarm handler state	T/E/D/U
iref_alm		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.40 Alarm Handler Specifications

Item	Description
Alarm handler ID	1 to CFG_MAXALMID (255 max.)
Alarm handler attributes	TA_HLNG: The handler is written in a high-level language. TA_ASM: The handler is written in assembly language.

6.20.1 Create Alarm Handler (cre_alm, icre_alm, acre_alm, iacre_alm)

C-Language API:

```
ER ercd = cre_alm (ID almid, T_CALM *pk_calm);
ER ercd = icre_alm (ID almid, T_CALM *pk_calm);
ER_ID almid = acre_alm (T_CALM *pk_calm);
ER_ID almid = iacre_alm (T_CALM *pk_calm);
```

Parameters:

T_CALM	*pk_calm	Pointer to the packet where the alarm handler creation information is stored
<cre_alm, icre_alm>		
ID	almid	Alarm handler ID

Return Parameters:

<cre_alm, icre_alm>		
ER	ercd	Normal termination (E_OK) or error code
<acre_alm, iacre_alm>		
ER_ID	almid	Created alarm handler ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    almatr;    0    4    Alarm handler attribute
    VP_INT exinf;     +4   4    Extended information
    FP     almhdr;    +8   4    Alarm handler address
}T_CALM;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) almatr is invalid.
E_PAR	[p]	Parameter error (1) pk_calm is not a 4-byte boundary address. (2) almhdr is an odd value.
E_ID	[p]	Invalid ID number (1) almid ≤ 0 (2) almid > CFG_MAXALMID.
E_NOID	[k]	No ID available (only for acre_alm)
E_OBJ	[k]	Invalid object state (1) Alarm handler specified by almid already exists.
E_MACV	[m]	Memory access violation

Function:

Each service call creates an alarm handler. The alarm handler is a time event handler for a non-task context and is initiated at the specified time only once. The alarm handler is assigned to the kernel domain and is executed in privileged mode.

The following describes each parameter function.

(1) almid

For service calls `cre_alm` and `icre_alm`, specify a value within the range from 1 to `CFG_MAXALMID` for parameter `almid`. Service calls `acre_alm` and `iacre_alm` search for an unused alarm handler ID, define an alarm handler for that ID with the contents specified by parameter `pk_calm`, and return the defined alarm handler ID as a return parameter.

(2) almatr

Specify either one of the following values.

- `TA_HLNG` (H'00000000): High-level language
- `TA_ASM` (H'00000001): Assembly language

The alarm handler is always assigned to the kernel domain.

In the PX specification, `TA_DOM(TDOM_KERNEL)` must be specified for `almatr` so that the alarm handler is assigned to the kernel domain. To ensure the portability of programs, always specify `TA_DOM(TDOM_KERNEL)`. Note that `TA_DOM()` is ignored in this kernel, and even if a user domain is assigned through `TA_DOM()`, no error will occur and the kernel domain is always assumed.

(3) exinf

Parameter `exinf` can be widely used by the user, for example, to set information concerning alarm handlers to be defined. `exinf` is passed to the alarm handler as a parameter.

The time to initiate the alarm handler is not set immediately after creating the alarm handler. The alarm handler is in the stopped state.

The alarm handler can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for pk_calm, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_calm
 - size = sizeof(T_CALM)
 - domid = Domain of the caller
 - pmmode = TPM_READ
- (2) The kernel domain does not have a read access permission for pk_calm->almhdr, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_calm->almhdr
 - size = 1
 - domid = Kernel domain
 - pmmode = TPM_READ

6.20.2 Delete Alarm Handler (del_alm)

C-Language API:

```
ER ercd = del_alm (ID almid);
```

Parameters:

ID	almid	Alarm handler ID
----	-------	------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{almid} \leq 0$ (2) $\text{almid} > \text{CFG_MAXALMID}$
E_CTX	[k]	Context error (1) Called in a non-task context
E_NOEXS	[k]	Undefined (1) Alarm handler specified by almid does not exist.

Function:

Service call del_alm deletes the alarm handler specified by parameter almid.

6.20.3 Start Alarm Handler Operation (sta_alm, ista_alm)

C-Language API:

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

Parameters:

ID	almid	Alarm handler ID
RELTIM	almtim	Alarm handler initiation time

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{almid} \leq 0$ (2) $\text{almid} > \text{CFG_MAXALMID}$
E_NOEXS	[k]	Undefined (1) Alarm handler specified by almid does not exist.

Function:

The initiation time for the alarm handler specified by almid is set to the relative time specified by almtim after the moment at which the service call is issued, to start operation of the alarm handler.

If a time is set for an alarm handler already in operation, the previous initiation time setting is cancelled, and the new initiation time is set.

If almtim is set to 0, the alarm handler is started at the next time tick.

When a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for almtim is $\text{H'ffffffff}/\text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

6.20.4 Stop Alarm Handler Operation (stp_alm, istp_alm)

C-Language API:

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

Parameters:

ID	almid	Alarm handler ID
----	-------	------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $\text{almid} \leq 0$ (2) $\text{almid} > \text{CFG_MAXALMID}$
E_NOEXS	[k]	Undefined (1) Alarm handler specified by almid does not exist.

Function:

Each service call cancels the initiation time for the alarm handler specified by parameter almid, and stops alarm handler operation.

6.20.5 Refer to Alarm Handler State (ref_alm, iref_alm)

C-Language API:

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

Parameters:

ID	almid	Alarm handler ID
T_RALM	*pk_ralm	Pointer to the packet where the alarm handler state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RALM	*pk_ralm	Pointer to the packet where the alarm handler state is stored

Packet Structure:

```
typedef struct {  
    STAT    almstat;      0    4    Alarm handler operation state  
    RELTIM  lefttim;      +4   4    Remaining time until the alarm  
                                     handler is initiated  
}T_RALM;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_ralm is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) almid ≤ 0 (2) almid > CFG_MAXALMID
E_NOEXS	[k]	Undefined (1) Alarm handler specified by almid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call reads the alarm handler state specified by almid and returns the alarm handler operating state (almstat) and remaining time until the alarm handler is initiated (lefttim) to the area indicated by parameter pk_ralm.

The target alarm handler operating state is returned to parameter almstat.

- TALM_STP (H'00000000): The alarm handler is not operating
- TALM_STA (H'00000001): The alarm handler is operating

The relative time until the target alarm handler is initiated next is returned to parameter lefttim. When the target alarm handler is not operating, lefttim is undefined.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_ralm, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_ralm
- size = sizeof(T_RALM)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.21 Time Management (Overflow Handler)

Table 6.41 Service Calls for Overflow Handler

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
def_ovr	Defines overflow handler	T/E/D/U
sta_ovr	Starts overflow handler operation	T/E/D/U
ista_ovr		N/E/D/U
stp_ovr	Stops overflow handler operation	T/E/D/U
istp_ovr		N/E/D/U
ref_ovr	Refers to overflow handler state	T/E/D/U
iref_ovr		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.42 Overflow Handler Specifications

Item	Description
Processor time unit (OVRTIM)	Same as system clock (1 [ms])
Overflow handler attributes	TA_HLNG: The handler is written in a high-level language. TA_ASM: The handler is written in assembly language.

Only one overflow handler can be defined in the system. The overflow handler is a time event handler.

The processor time used by the task includes the execution times of a task, the service calls issued by the task, and the interrupt handler that is initiated during execution of the task. Used processor time is not counted while the task is not in the RUNNING state.

6.21.1 Define Overrun Handler (def_ovr)

C-Language API:

```
ER ercd = def_ovr (T_DOVR *pk_dovr);
```

Parameters:

T_DOVR	*pk_dovr	Pointer to the packet where the overrun handler definition information is stored
--------	----------	--

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ATR  ovratr;      0    4    Overrun handler attribute  
    FP   ovrhdr;      +4    4    Overrun handler address  
}T_DOVR;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_ASM in ovratr are not 0.
E_PAR	[p]	Parameter error (1) pk_dovr is not a 4-byte boundary address. (2) ovrhdr is an odd value.
E_MACV	[m]	Memory access violation

Function:

Service call def_ovr defines an overrun handler.

The overrun handler is a time event handler for a non-task context which is started when the processor is used by a task for a time exceeding a preset time. The overrun handler is assigned to the kernel domain and is executed in privileged mode.

The following describes each parameter function.

(1) ovratr

Specify either one of the following values.

- TA_HLNG (H'00000000): High-level language
- TA_ASM (H'00000001): Assembly language

(2) ovrhdr

Parameter ovrhdr specifies the start address of the overrun handler.

When `pk_dovr=NULL (0)` is specified, the overrun handler definition is cancelled.

When an overrun handler has already been defined, if this service call is issued, the preceding definition is cancelled and the new definition takes its place.

This service call must not be issued in a non-task context, but even if attempted, the `E_CTX` error will not be detected.

An overrun handler can also be defined statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for `pk_dovr`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = pk_dovr`
 - `size = sizeof(T_DOVR)`
 - `domid = Domain of the caller`
 - `pmmode = TPM_READ`
- (2) The kernel domain does not have a read access permission for `pk_dovr->ovrhdr`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base = pk_dovr->ovrhdr`
 - `size = 1`
 - `domid = Kernel domain`
 - `pmmode = TPM_READ`

6.21.2 Start Overrun Handler Operation (sta_ovr, ista_ovr)

C-Language API:

```
ER ercd = sta_ovr (ID tskid, OVRTIM ovrtime);  
ER ercd = ista_ovr (ID tskid, OVRTIM ovrtime);
```

Parameters:

ID	tskid	Task ID
OVRTIM	ovrtim	Upper processor time limit

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Overrun handler has not been defined.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Overrun handler operation begins for the task specified by tskid.

By specifying tskid=TSK_SELF (0), the current task is specified.

The upper processor time limit for the task is set to the time specified by ovrtime, and the processor time used is cleared to 0. If the specified overrun handler has already been operating, the upper processor time limit previously specified is cancelled, and the new processor time limit is set.

When the processor time used exceeds the upper processor time limit, the overrun handler is started.

When a value larger than 1 is specified for CFG_TICDENO (the denominator for time tick cycles), the maximum value that can be specified for ovrtime is $H'ffffff / \text{CFG_TICDENO}$. If a value larger than this is specified, operation is not guaranteed.

If 0 is specified for ovrtime, the overrun handler is started on the first time tick after the task begins to use the processor.

6.21.3 Stop Overrun Handler Operation (stp_ovr, istp_ovr)

C-Language API:

```
ER ercd = stp_ovr (ID tskid);  
ER ercd = istp_ovr (ID tskid);
```

Parameters:

ID	tskid	Task ID
----	-------	---------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Overrun handler has not been defined.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.

Function:

Each service call releases the upper processor time limit for the task specified by parameter tskid and stops overrun handler operation.

By specifying tskid = TSK_SELF (0), the current task is specified.

6.21.4 Refer to Overrun Handler State (ref_ovr, iref_ovr)

C-Language API:

```
ER ercd = ref_ovr (ID tskid, T_ROVR *pk_rovr);  
ER ercd = iref_ovr (ID tskid, T_ROVR *pk_rovr);
```

Parameters:

ID	tskid	Task ID
T_ROVR	*pk_rovr	Pointer to the packet where the overrun handler state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_ROVR	*pk_rovr	Pointer to the packet where the overrun handler state is stored

Packet Structure:

```
typedef struct {  
    STAT    ovrstat;      0    4    Overrun handler operating state  
    OVRTIM  leftotm;     +4    4    Remaining processor time  
}T_ROVR;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rovr is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) tskid < 0 (2) tskid > CFG_MAXTSKID (3) tskid = TSK_SELF (0) is specified in a non-task context.
E_OBJ	[k]	Invalid object state (1) Overrun handler has not been defined.
E_NOEXS	[k]	Undefined (1) Task specified by tskid does not exist.
E_MACV	[m]	Memory access violation

Function:

The state of the overrun handler for the task specified by tskid is referenced. By specifying tskid = TSK_SELF (0), the current task is specified.

The state of operation of the overrun handler (ovrstat) and the remaining processor time (leftotm) are returned to the area specified by pk_rovr. As the operating state of the overrun handler, the upper processor time limit setting is returned as ovrstat.

- TOVR_STP (H'00000000): No upper processor time limit is set

- TOVR_STA (H'00000001): An upper processor time limit is set

The processor time remaining until the overrun handler is started due to the target task is returned as leftotm. If no upper processor time limit is set for the task, the value of leftotm is undefined.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rovr, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_rovr
 - size = sizeof(T_ROVR)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.22 System State Management

Table 6.43 Service Calls for System State Management

Service Call¹		Description	System State²
			T/N/E/D/U/L/C
rot_rdq	[S]	Rotates ready queue	T/E/D/U
irrot_rdq	[S]		N/E/D/U
get_tid	[S]	Refers to task ID in RUNNING state	T/E/D/U
iget_tid	[S]		N/E/D/U
get_did		Refers to domain ID of the task in RUNNING state	T/E/D/U
iget_did			N/E/D/U
loc_cpu	[S]	Locks CPU	T/E/D/U/L
iloc_cpu	[S]		N/E/D/U/L
unl_cpu	[S]	Unlocks CPU	T/E/D/U/L
iunl_cpu	[S]		N/E/D/U/L
dis_dsp	[S]	Disables task dispatch	T/E/D/U
ena_dsp	[S]	Enables task dispatch	T/E/D/U
sns_ctx	[S]	Refers to task context	T/N/E/D/U/L
sns_loc	[S]	Refers to CPU-locked state	T/N/E/D/U/L
sns_dsp	[S]	Refers to dispatch-disabled state	T/N/E/D/U/L
sns_dpn	[S]	Refers to dispatch-pended state	T/N/E/D/U/L
vsta_knl	[s]	Starts kernel	T/N/E/D/U/L/C
ivsta_knl	[s]		T/N/E/D/U/L/C
vsys_dwn	[s]	Terminates the system	T/N/E/D/U/L/C
ivsys_dwn	[s]		T/N/E/D/U/L/C
vget_trc		Acquires trace information	T/E/D/U
ivget_trc			N/E/D/U
ivbgn_int		Acquires start of interrupt handler to trace	N/E/D/U
ivend_int		Acquires end of interrupt handler to trace	N/E/D/U
vchg_cop		Changes DSP (TA_COP0) attribute	T/E/D/U

Notes: 1. [S]: Standard profile service calls

[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

- 2. T: Can be called in a task context
- N: Can be called in a non-task context
- E: Can be called in dispatch-enabled state
- D: Can be called in dispatch-disabled state
- U: Can be called in CPU-unlocked state
- L: Can be called in CPU-locked state
- C: Can be called from CPU exception handler

6.22.1 Rotate Ready Queue (rot_rdq, irot_rdq)

C-Language API:

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

Parameters:

PRI	tskpri	Task priority
-----	--------	---------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error
-------	-----	-----------------

- (1) tskpri < 0
- (2) tskpri > CFG_MAXTSKPRI
- (3) tskpri = TPRI_SELF (0) is specified in a non-task context.

Function:

Each service call rotates the ready queue for the task priority specified by parameter tskpri. In other words, the task at the head of the ready queue for the task priority is sent to the end of the queue, enabling the second task in the ready queue to be executed.

Specifying tskpri = TPRI_SELF (0) rotates the ready queue for the base priority of the current task. The base priority is the same as the current priority when the mutex function is not used; however, the current priority is not the same as the base priority while the mutex is locked. Thus, the ready queue for the priority where the current task is included, cannot be rotated even when TPRI_SELF is specified.

6.22.2 Get Task ID in RUNNING state (get_tid, iget_tid)

C-Language API:

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

Parameters:

ID	*p_tskid	Pointer to the area where the task ID is to be returned
----	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
ID	*p_tskid	Pointer to the task ID

Error Codes:

E_PAR	[p]	Parameter error (1) p_tskid is not a 2-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call gets the task ID in the RUNNING state and returns it to the area specified by p_tskid. If each service call is issued in a task context, the current task ID is returned. If each service call is issued in a non-task context, the task ID that is being executed is returned. If there is no task in the RUNNING state, TSK_NONE (0) is returned.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_tskid, which means that an error will be returned if prb_mem is issued with the following parameters.
- base = p_tskid
 - size = sizeof(ID)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.22.3 Get Domain ID of the Task in RUNNING State (get_did, iget_did)

C-Language API:

```
ER ercd = get_did(ID *p_domid);  
ER ercd = iget_did(ID *p_domid);
```

Parameters:

ID	*p_domid	Pointer to the area where the domain ID is to be returned
----	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
ID	*p_domid	Pointer to the domain ID

Error Codes:

E_PAR	[p]	Parameter error (1) p_domid is not a 2-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call gets the ID of the domain where the task in the RUNNING state is assigned and returns it to the area specified by p_domid.

The actual processing depends on where the service call is issued as follows.

- (1) When the service call is issued in a task or a task exception processing routine, the service call returns the ID of the domain where the task is assigned.
- (2) When the service call is issued in an extended service call or trap routine that was called from a task or a task exception processing routine, the service call returns the ID of the domain where the task which was in the RUNNING state before the extended service call or trap routine was called is assigned.
- (3) When the service call is issued in a non-task context, the service call returns the ID of the domain where the task which is in RUNNING state when the service call is issued is assigned. If there is no task in the RUNNING state, TDOM_NONE (-2) is returned.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_domid, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = p_domid
 - size = sizeof(ID)

- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.22.4 Lock CPU (loc_cpu, iloc_cpu)

C-Language API:

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

Parameters:

None

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_CTX	[k]	Context error (1) Called in a task context while interrupts are masked through chg_ims.
-------	-----	--

Function:

Each service call locks the CPU and inhibits interrupts and task dispatches.

The following describes the CPU-locked state:

- ◆ Tasks cannot be scheduled while the CPU is locked.
- ◆ Task exception processing routines cannot be initiated while the CPU is locked.
- ◆ Interrupts, having a level equal to or below the kernel interrupt mask level (CFG_KNLMSKLVL) defined by the configurator, are inhibited.
- ◆ Only the following service calls can be issued in the CPU-locked state. The system operation cannot be guaranteed when a service call other than the followings is issued. When a service call that shifts a task to the WAITING state is issued or chg_ims is issued, an E_CTX error is returned.
 - ext_tsk (automatically unlocks the CPU)
 - exd_tsk (automatically unlocks the CPU)
 - sns_tex
 - loc_cpu, iloc_cpu
 - unl_cpu, iunl_cpu
 - sns_ctx
 - sns_loc
 - sns_dsp
 - sns_dpn
 - vsta_knl, ivsta_knl
 - vsys_dwn, ivsys_dwn

When the following service calls are issued in the CPU-locked state, the system returns to the CPU-unlocked state.

- unl_cpu or iunl_cpu
- ext_tsk or exd_tsk

When an interrupt handler, a time event handler, a CPU exception handler, or an initialization routine is completed, the system returns to the state before handler initiation (CPU-locked or CPU-unlocked state).

If service calls loc_cpu and iloc_cpu are issued while the CPU is locked, no error will occur. In this case, queuing will not be done.

6.22.5 Unlock CPU (unl_cpu, iunl_cpu)

C-Language API:

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

Parameters:

None

Return Parameters:

ER ercd Normal termination (E_OK) or error code

Error Codes:

E_CTX [k] Context error
 (1) Called in a task context while interrupts are masked
 through chg_ims.

Function:

Each service call unlocks the CPU, which was locked by service call loc_cpu or iloc_cpu. If service call unl_cpu is issued in the dispatch-enabled state, the task scheduling is performed.

The CPU-locked state and dispatch-disabled state are managed individually. Thus, service call unl_cpu or iunl_cpu does not enable the task dispatch which was disabled by service call dis_dsp.

If service calls unl_cpu and iunl_cpu are called in CPU-unlocked state, no error will occur, but queuing will not be done.

6.22.6 Disable Dispatch (dis_dsp)

C-Language API:

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

Parameters:

None

Return Parameters:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

E_CTX	[k]	Context error
		(1) Called in a non-task context.

Function:

Service call `dis_dsp` disables task dispatch.

The following describes the dispatch-disabled state:

- ◆ Tasks cannot be scheduled.
- ◆ When a service call that shifts a task to the `WAITING` state is issued in a task context, an `E_CTX` error is returned.

When the following service calls are issued while task dispatch is disabled, the system returns to the task dispatch-enabled state.

- ◆ `ena_dsp`
- ◆ `ext_tsk` or `exd_tsk`

The transition between dispatch-disabled state and dispatch-enabled state occurs only when `dis_dsp`, `ena_dsp`, `ext_tsk`, or `exd_tsk` service call is issued.

When task dispatch is disabled by this service call, the task state is undefined. Therefore, if the current task refers to its state by service call `ref_tsk`, the returned state is not always the `RUNNING` state.

An error will not occur when service call `dis_dsp` is issued while the task dispatch is disabled; however, queuing will not be done.

6.22.7 Enable Dispatch (ena_dsp)

C-Language API:

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

Parameters:

None

Return Parameters:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

E_CTX	[k]	Context error
-------	-----	---------------

(1) Called in a non-task context.

Function:

Service call ena_dsp enables task dispatch disabled by service call dis_dsp. Task scheduling is then performed after the service call.

An error will not occur when service call ena_dsp is called during task dispatch-enabled state; however, queuing will not be done.

6.22.8 Refer to Context (sns_ctx)

C-Language API:

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

Parameters:

None

Return Parameters:

BOOL state Context

Function:

Service call sns_ctx returns TRUE when it is issued in a non-task context, or FALSE when in a task context.

6.22.9 Refer to CPU-Locked State (sns_loc)

C-Language API:

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

Parameters:

None

Return Parameters:

BOOL state CPU-locked state

Function:

Service call sns_loc returns TRUE when the CPU is locked, or FALSE when the CPU is unlocked.

6.22.10 Refer to Dispatch-Disabled State (sns_dsp)

C-Language API:

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

Parameters:

None

Return Parameters:

BOOL state Dispatch-disabled state

Function:

Service call sns_dsp returns TRUE when task dispatch is disabled, or FALSE when task dispatch is enabled.

6.22.11 Refer to Dispatch-Pended State (sns_dpn)

C-Language API:

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

Parameters:

None

Return Parameters:

BOOL state Dispatch-pended state

Function:

Service call sns_dpn returns TRUE when the task dispatch is pended. Otherwise, this service call returns FALSE.

When the following conditions are all satisfied, FALSE is returned. Otherwise, TRUE is returned.

- Task dispatch is not disabled.
- The CPU is unlocked.
- Execution is in a task context.
- Interrupts are not masked by service call chg_ims.

6.22.12 Start Kernel (vsta_knl, ivsta_knl)

C-Language API:

```
void vsta_knl(void);  
void ivsta_knl(void);
```

Parameters:

None

Return Parameters:

No parameters are returned to the caller.

Function:

Each service call starts the kernel.

If the kernel has already been started, the multitasking environment up to that point is all nullified. In addition, control does not return to the caller.

These service calls must be issued while SR.MD = 1 and SR.BL = 1. An application issuing these service calls must be linked with the kernel library.

Note that even when the memory object protection function is selected, the kernel does not enable the MMU during initialization. The application must initialize MMUCR.AT to determine whether to enable the MMU before starting the kernel.

After enabling the MMU, do not access the MMU mapped area before calling vsta_knl; MMU-related interrupts, which may be generated by access to the MMU mapped area, cannot be handled during this period because the kernel has not been started.

The following shows the detailed initialization processing performed by these service calls.

- (1) Sets SR.BL = 1 (disables all interrupts)
- (2) Sets the stack pointer (R15) to a non-task context stack (section BSCP_hintsstk).
- (3) Initializes the VBR.

When an interrupt handler is initiated, the IMASK bits in SR are handled according to the value of the INTMU bit in the CPUOPM set when the kernel is started. Initialize the bit as necessary before starting the kernel.

- (4) Initializes the RAMCR.

Only when the memory object protection function is selected (CFG_PROTMEM is selected) and CFG_IRAM is selected, the RP and RMD bits in the RAMCR are initialized according to the CFG_IRAMUSAGE setting as follows. The other bits in the register are not initialized.

CFG_IRAMUSAGE Setting	Initial RAMCR.RP Value	Initial RAMCR.RMD Value
MMU non-mapped area, accessible in any mode	0	1
MMU non-mapped area, accessible in user non-DSP mode only	0	0
MMU mapped area	1	1

(5) Initializes the MMUCR.

Only when the memory object protection function is selected (CFG_PROTMEM is selected), the LRUI, URB, and URC bits in the MMUCR are initialized to 0. The other bits in the register are not initialized.

To enable the MMU, set MMUCR.AT = 1 before starting the kernel. If the kernel is started while MMUCR.AT = 0, the MMU remains disabled. In this state, no illegal memory access can be detected but no TLB miss will occur.

(6) Initializes the kernel management information.

(7) Initializes the static memory objects specified through the configurator.

(8) Initializes the timers when CFG_OPTTMR is selected.

(9) Sets SR.BL = 0 and IMASK = 15.

(10) Creates and defines necessary objects.

The objects specified through the configurator are created and defined.

These objects are created and defined through appropriate service calls issued in the initial definition routines output from the configurator.

If the settings through the configurator are incorrect, these service calls may end with an error. In this case, the initial definition routine enters an infinite loop at the point where the service call causing the error is issued.

There are two initial definition routines: kernel_def_inireg.h and kernel_cfg_inireg.h.

kernel_def_inireg.h creates and defines the objects for which [Kernel side] has been selected through the configurator, and kernel_cfg_inireg.h is for the objects for which [Kernel side] has not been selected.

kernel_def_inireg.h is called first, then kernel_cfg_inireg.h is called. In each routine, objects are created and defined in the specified order shown below. Among the same type of objects, tasks and cyclic handlers are created in the order of the list displayed in the configurator, but the other objects are created in a random order.

Order	Object
1	Interrupt and CPU exception handlers
2	Overrun handler
3	Cyclic handlers
4	Alarm handlers
5	Extended service call routines
6	Trap routines
7	Semaphores
8	Event flags
9	Data queues
10	Mailboxes
11	Mutexes
12	Message buffers
13	Fixed-size memory pools
14	Variable-size memory pools
15	Protected memory pools
16	Protected mailboxes
17	Tasks and task exception processing routines

(11) Calls the timer initialization routine (`_kernel_tmrini()`) only when `CFG_OPTTMR` is not selected.

(12) Calls the initialization routines.

The initial definition routines defined through the configurator are called.

First, the routines for which [Kernel side] is not selected are called in the order of the list displayed in the configurator, and then the routines for which [Kernel side] is not selected are called in the order of the list displayed in the configurator.

(13) Initializes the program performance counters only when `CFG_PERFORM` is selected.

After the above initialization, the kernel enters the multitasking environment.

`ivsta_knl` is an API implemented to conform to the naming convention of the μ ITRON 4.0 specification, but its actual code is the same as that for `vsta_knl`. In the header file, `ivsta_knl()` is defined as `vsta_knl()`.

6.22.13 System Down (vsys_dwn, ivsys_dwn)

C-Language API:

```
void vsys_dwn (ER type, VW inf1, VW inf2, VW inf3);  
void ivsys_dwn (ER type, VW inf1, VW inf2, VW inf3);
```

Parameters:

ER	type	Error type
VW	inf1	System abnormal information 1
VW	inf2	System abnormal information 2
VW	inf3	System abnormal information 3

Return Parameters:

No parameters are returned to the caller.

Function:

Each service call passes control to the system down routine. The system down routine is assigned to the kernel domain and is executed in privileged mode.

A value (1 to H'7fffffff) corresponding to the error type must be specified for the parameter type. Value 0 or smaller values are reserved for system use.

The system down routine is also executed when abnormal operation is detected in the kernel.

Service calls vsys_dwn and ivsys_dwn can be issued in the CPU-locked state and from the CPU exception handler.

ivsys_dwn is an API implemented to conform to the naming convention of the μ ITRON 4.0 specification, but its actual code is the same as that for vsys_dwn. In the header file, ivsys_dwn is defined as vsys_dwn.

As these service calls use a TRAPA instruction in the same way as in the other service calls, they must not be issued while SR.BL = 1. To pass control to the system down routine while SR.BL = 1, call the system down routine directly.

Reference: Section 8.10, System Down Routine

6.22.14 Acquire Trace Information (vget_trc, ivget_trc)

C-Language API:

```
ER ercd = vget_trc(VW para1, VW para2, VW para3, VW para4);  
ER ercd = ivget_trc(VW para1, VW para2, VW para3, VW para4);
```

Parameters:

VW	para1	Parameter 1
VW	para2	Parameter 2
VW	para3	Parameter 3
VW	para4	Parameter 4

Return Parameters:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

None

Function:

A trace of information required by the user is obtained.

Parameters para1 to para4 can be used freely by the user to distinguish the information to be acquired.

The acquired trace information can be displayed by using a debugging extension.

If CFG_TRACE is not selected by the configurator, this service call always ends normally and does not perform any processing.

ivget_trc is an API implemented to conform to the naming convention of the μ ITRON 4.0 specification, but its actual code is the same as that for vget_trc. In the header file, ivget_trc is defined as vget_trc.

6.22.15 Acquire Start of Interrupt Handler as Trace Information (ivbgn_int)

C-Language API:

```
ER ercd = ivbgn_int(UINT dintno);
```

Parameters:

UINT	dintno	Interrupt handler number
------	--------	--------------------------

Return Parameters:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

None

Function:

This API is implemented only in the C-language interface to ensure the compatibility with the HI7000/4 series. In the C-language interface, this is defined as follows.

```
#define ivbgn_int(dintno) E_OK /* Always returns E_OK. */
```

This kernel always acquires trace information about the start and end of interrupt handlers when CFG_TRACE is selected.

6.22.16 Acquire End of Interrupt Handler as Trace Information (ivend_int)

C-Language API:

```
ER ercd = ivend_int(UINT dintno);
```

Parameters:

UINT	dintno	Interrupt handler number
------	--------	--------------------------

Return Parameters:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

None

Function:

This API is implemented only in the C-language interface to ensure the compatibility with the HI7000/4 series. In the C-language interface, this is defined as follows.

```
#define ivend_int(dintno) E_OK /* Always returns E_OK. */
```

This kernel always acquires trace information about the start and end of interrupt handlers when CFG_TRACE is selected.

6.22.17 Change DSP (TA_COP0) Attribute (vchg_cop)

C-Language API:

```
ER_UINT oldatr = vchg_cop(ATR newatr);
```

Parameters:

ATR	newatr	Attribute after change
-----	--------	------------------------

Return Parameters:

ER_UINT	oldatr	Attribute before change (a positive value or 0) or error code
---------	--------	---

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_COP0 in newatr are not 0.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_ILUSE	[k]	Illegal use of service call (1) Called in a program with the TA_COP1 or TA_COP2 attribute

Function:

Service call vchg_cop changes the current TA_COP0 attribute of the caller to that specified by newatr.

This service call is implemented to refine the control of DSP standby function.

This service call changes the current attribute of the caller (a task when called from the task, a task exception processing routine when called from the routine, or an extended service call or trap routine when called from the routine that has been called in a task context).

The change is temporary, just for this one time.

For a task, after it is completed and then initiated again, the attribute returns to the initial state specified at task creation.

For a task exception processing routine, after it is completed and then initiated again, the attribute returns to the initial state defined at creation of the task exception processing routine.

For an extended service call or trap routine called in a task context, the attribute change is only valid in the context of the current task being executed. When the routine is called again from another task after that, the attribute returns to the initial state specified at definition of the routine.

The following can be specified for newatr. FPU-related attributes (TA_COP1 and TA_COP2) cannot be specified.

- TA_COP0 (H'00000100): The task or routine uses the DSP.
- TA_NULL (H'00000000): The task or routine does not use the DSP.

The attribute before change (TA_COP0 or TA_NULL) is returned through oldatr.

6.23 Interrupt Management

Table 6.44 Service Calls for Interrupt Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
def_inh	Defines interrupt handler	T/E/D/U
idef_inh		N/E/D/U
chg_ims	Changes interrupt mask	T/E/D/U
ichg_ims		N/E/D/U
get_ims	Refers to interrupt mask	T/E/D/U
iget_ims		N/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.45 Interrupt Management Specifications

Item	Description
Interrupt handler number	A multiple of H'20 within the range from 0 to CFG_MAXINTNO (H'3fe0 max.)
Interrupt handler attributes	TA_HLNG: The handler is written in a high-level language. TA_ASM: The handler is written in assembly language.

6.23.1 Define Interrupt Handler (def_inh, ndef_inh)

C-Language API:

```
ER ercd = def_inh(INHNO inhno, T_DINH *pk_dinh);  
ER ercd = ndef_inh(INHNO inhno, T_DINH *pk_dinh);
```

Parameters:

INHNO	inhno	Interrupt handler number
T_DINH	*pk_dinh	Pointer to the packet where the definition information of interrupt handler is stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ATR    inhatr;      0    4    Handler attribute  
    FP    inthdr;      +4    4    Handler address  
    UINT    inhsr;      +8    4    SR at initiation  
}T_DINH;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_ASM in inhatr are not 0.
E_PAR	[p]	Parameter error (1) inhno after being rounded down to a multiple of 0x20 is 0, H'20, H'140, H'160, or a value larger than CFG_MAXINTNO. (2) pk_dinh is not a 4-byte boundary address. (3) inthdr is an odd value.
E_MACV	[m]	Memory access violation

Function:

Each service call defines an interrupt handler. The interrupt handler is assigned to the kernel domain and is executed in privileged mode.

The actual codes of these service calls are the same as that for def_exc, which means that these service calls can be used to define a CPU exception handler and def_exc can be used to define an interrupt handler.

The following describes each parameter function.

(1) inhno

Parameter inhno is an interrupt handler number. Specify an INTEVT code of the CPU for the interrupt handler number. inhno is rounded down to a multiple of H'20 during processing.

Note that the interrupt code (INTEVT code) and exception code (EXPEVT code) are managed in the same code system in the SH microcomputer. This service call does not check whether the specified inhno value is an interrupt code or an exception code. If an exception code is specified, the service call operates in the same way as def_exc.

For some INTEVT and EXPEV codes, no handlers can be defined, or even when handlers can be defined, the handlers will not work. Table 6.46 shows these cases.

Table 6.46 Exceptional INTEVT and EXPEVT Codes

INTEVT or EXPEVT Code	Cause	Handler Definition	CPU Operation for Specified Interrupt or Exception	
			CFG_PROTMEM Is Not Selected	CFG_PROTMEM Is Selected
0	Power-on reset or H-UDI reset	Not possible	Branches to the reset vector (H'A0000000).	
H'20	Manual reset	Not possible		
H'40	TLB miss exception (read)	Possible	Initiates the defined handler. *	The kernel updates the TLB. If the TLB cannot be updated, the kernel initiates the memory access violation handler. The defined handler is never executed; its definition has no effect.
H'60	TLB miss exception (write)			
H'A0	TLB protection violation exception (read)			
H'C0	TLB protection violation exception (write)			
H'140	Instruction TLB multiple-hit exception or data TLB multiple-hit exception	Not possible	Branches to the reset vector (H'A0000000). *	Branches to the reset vector (H'A0000000).
H'160	TRAPA instruction	Not possible	According to the trap number, a kernel service call or user-defined trap routine is executed.	

Note: This type of exception will not occur usually unless the application enables the MMU.

Do not specify the following codes.

(a) CFG_TIMINTNO

CFG_TIMINTNO is an interrupt number used in the standard timer driver when CFG_OPTTMR is not selected. If a handler is specified for this number, the standard timer driver will not operate correctly.

(b) H'400, H'420, and H'440

These are interrupt numbers used in the optimized timer driver when CFG_OPTTMR is selected. If a handler is specified for any one of these numbers, the optimized timer driver will not operate correctly.

When def_ovr is not installed, H'440 is not used in the optimized timer driver and an interrupt handler can be defined for this number.

(2) inhtr

Specify either one of the following values.

- TA_HLNG (H'00000000): High-level language
- TA_ASM (H'00000001): Assembly language

(3) inhsr

inhsr is a parameter not specified in the μ ITRON specification.

Parameter inhsr specifies the value of the status register (SR) on startup of the interrupt handler. inhsr is specified using the same bit position as the SR. Note that the SR value becomes as shown below when an interrupt handler is actually initiated; only the block (BL) and interrupt mask level (IMASK) bits take effect and the other bits are ignored. A value equal to or greater than the level of the target interrupt should always be specified as the interrupt mask bits. If a value lower than the interrupt level is specified, correct system operation is not guaranteed.

- Mode (MD) bit: Always 1
- Register bank (RB) bit: Always 0
- Block (BL) bit: Set to the inhsr value.
- DSP bit (SH4AL-DSP): 0
- FPU disable (FD) bit (SH-4A): 1
- Interrupt mask level (IMASK) bits: Set to the inhsr value if the INTMU bit in the CPUOPM register is 0 when the kernel is started. If the INTMU bit is 1, the level of the generated interrupt is set here.
- Other bits: Undefined

When pk_dinh = NULL (0) is specified, the definition of the interrupt handler is cancelled.

If this service call is issued while an interrupt handler has been defined, the previous definition is canceled and replaced with the new definition.

An interrupt handler can be statically defined through the configurator.

If an interrupt for a number for which no handler has been defined occurs, control is passed to the system down routine.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for pk_dinh, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = pk_dinh
 - size = sizeof(T_DINH)
 - domid = Domain of the caller
 - pmmode = TPM_READ
- (2) The kernel domain does not have a read access permission for pk_dinh->inthdr, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base= pk_dinh->inthdr
 - size = 1
 - domid = Kernel domain
 - pmmode = TPM_READ

6.23.2 Change Interrupt Mask (chg_ims, ichg_ims)

C-Language API:

```
ER ercd = chg_ims(IMASK imask);  
ER ercd = ichg_ims(IMASK imask);
```

Parameters:

IMASK	imask	Interrupt mask value
-------	-------	----------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[p]	Parameter error (1) A value other than SR_IMS00 to SR_IMS15 was specified for imask.
E_CTX	[k]	Context error (1) Called in CPU-locked state in a task context

Function:

Each service call changes the current interrupt mask to the level specified by imask.

The imask can be specified as follows:

- SR_IMSnn (H'0000000m): Changes interrupt mask level to nn.
nn: Character string indicating two-digit decimal number from 0 to 15 (00, 01, 02, ... , 15).
m: nn converted to a hexadecimal number.

Note the following precautions for the period while the interrupt mask is changed to a non-zero value through this service call in a task context.

1. Tasks are not scheduled, that is, the system enters dispatch-disabled state. No service call shifting a task to the WAITING state can be issued. If attempted, an E_CTX error is returned.
2. No task exception processing routine is initiated.
3. Interrupts having a level lower than the imask value are disabled.
4. Neither loc_cpu nor unl_cpu can be issued. If attempted, an E_CTX error is returned.

Use service call chg_ims or ichg_ims when changing the interrupt mask level in the following cases. The SR can be directly changed when changing the interrupt mask level in the other cases.

1. When the interrupt mask level is changed from level 0 to a level other than 0 in a task context.
2. When the interrupt mask level is returned to 0 after the above case.

Otherwise, normal system operation cannot be guaranteed.

Note that service calls must not be issued while the interrupt mask level is made higher than the kernel interrupt mask level (CFG_KNLMSKLVL) unless this service call is used to lower the interrupt mask level to a level equal to or below the kernel interrupt mask level. Otherwise, normal system operation cannot be guaranteed.

6.23.3 Refer to Interrupt Mask (get_ims, iget_ims)

C-Language API:

```
ER ercd = get_ims(IMASK *p_ims);  
ER ercd = iget_ims(IMASK *p_ims);
```

Parameters:

IMASK	*p_ims	Start address of the area where the interrupt mask level is to be returned
-------	--------	--

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
IMASK	*p_ims	Start address of the area where the interrupt mask level is stored

Error Codes:

E_PAR	[p]	Parameter error (1) p_ims is not a 4-byte boundary.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the interrupt mask bits (IMASK bits) of the current CPU status register (SR) and returns the interrupt mask level to the area indicated by p_ims.

The value to be returned to p_ims has the same format as parameter imask used by service call chg_ims.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_ims, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = p_ims
 - size = sizeof(IMASK)
 - domid = Domain of the caller
 - pmmode = TPM_READ|TPM_WRITE

6.24 Extended Service Call and Trap Management

Table 6.47 Service Calls for Service Call Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
def_svc	Defines extended service call	T/E/D/U
idef_svc		N/E/D/U
cal_svc	Issues extended service call	T/E/D/U
ical_svc		N/E/D/U
vdef_trp	Defines trap	T/E/D/U
ivdef_trp		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.48 Service Call Management Specifications

Item	Description
Function code of extended service call	1 to CFG_MAXSVCCD (32767 max.)
Extended service call routine attributes	TA_HLNG: High-level language TA_ASM: Assembly language TA_COP0: The routine uses the DSP. TA_COP1: The routine uses register bank 0 in the FPU. TA_COP2: The routine uses register bank 1 in the FPU.
Trap number	16 to CFG_MAXTRPNO (255 max.) Note that 0 to 15 are reserved for the kernel use and cannot be specified.
Trap routine attributes	TA_HLNG: High-level language TA_ASM: Assembly language TA_COP0: The routine uses the DSP. TA_COP1: The routine uses register bank 0 in the FPU. TA_COP2: The routine uses register bank 1 in the FPU.

The extended service call and trap have the following features.

- Both the extended service call routine and trap routine are executed in privileged mode, and the access types that are prohibited in the user domain are allowed.
- These routines can be called without being linked to a program.
- When an extended service call or a trap is called in a task context, the task exception processing routine is not initiated while the extended service call or trap routine is being executed.
- When an extended service call or a trap is called in a task context, the WAITING-disabled state is entered if a rel_wai service call is issued while the extended service call or trap routine is being executed.

6.24.1 Define Extended Service Call (def_svc, idef_svc)

C-Language API:

```
ER ercd = def_svc (FN fncd, T_DSVC *pk_dsvc);  
ER ercd = idef_svc (FN fncd, T_DSVC *pk_dsvc);
```

Parameters:

FN	fncd	Function code of extended service call
T_DSVC	*pk_dsvc	Start address of the extended service call routine definition information

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ATR    svcatr;      0  4  Extended service call routine attribute  
    FP     svcrtn;      +4 4  Extended service call routine address  
    UW     inifpscr;    +8 4  Initial FPSCR value  
}T_DSVC;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_COP0, TA_COP1, TA_COP2, and TA_ASM in svcatr are not 0. (2) TA_COP0 is specified for svcatr while CFG_DSP is not selected. (3) TA_COP1 is specified for svcatr while CFG_FPU is not selected. (4) TA_COP2 is specified for svcatr while TA_COP1 is not specified. (5) Both TA_COP0 and TA_COP1 are specified for svcatr.
E_PAR	[p]	Parameter error (1) $fncd \leq 0$ (2) $fncd > CFG_MAXSVCCD$ (3) <i>pk_dsvc</i> is not a 4-byte boundary address. (4) <i>svcrtn</i> is an odd value.
E_MACV	[m]	Memory access violation

Function:

Service calls *def_svc* and *idef_svc* define an extended service call routine. The extended service call routine is called through *cal_svc*.

The extended service call routine is assigned to the kernel domain and is executed in privileged mode.

Note that the following states do not change before and after the extended service call routine is initiated and terminated.

- Task or non-task context
- Dispatch-disabled or enabled state
- CPU-locked or unlocked state

While the extended service call routine called from a task is being executed, the task exception processing routine for that task is not initiated. When `rel_wai` is issued for that task, the task enters the WAITING-disabled state.

The following describes each parameter function.

(1) `fncd`

Parameter `fncd` specifies the function code for the extended service call routine. Specify a value within the range from 1 to `CFG_MAXSVCCD`.

(2) `svcatr`

Specify the logical OR of the following (a) to (c) values for `svcatr`.

(a) Language

Specify either one of the following values.

- `TA_HLNG` (H'00000000): High-level language
- `TA_ASM` (H'00000001): Assembly language

(b) Using a microcomputer with an on-chip DSP (when `CFG_DSP` is selected)

Specify `TA_COP0` to use the DSP.

- `TA_COP0` (H'00000100): The routine uses the DSP.

(c) Using a microcomputer with an on-chip FPU (when `CFG_FPU` is selected)

Specify `TA_COP1` to use the FPU for floating-point operations. Specify `TA_COP2` in addition to `TA_COP1` when using both banks of the FPU for matrix operations.

- `TA_COP1` (H'00000200): The routine uses FPU register bank 0 (`FPR0_BANK0` to `FPR15_BANK0`) and `FPUL`.
- `TA_COP2` (H'00000400): The routine uses FPU register bank 1 (`FPR0_BANK1` to `FPR15_BANK1`).

To specify `TA_COP2`, be sure to specify `TA_COP1` together; otherwise, an `E_RSATR` error will be returned.

Also refer to description (4), `inifpscr`.

(3) `svcrtn`

Parameter `svcrtn` specifies the start address of the extended service call routine.

(4) `inifpscr`

`inifpscr` is a parameter not specified in the μ ITRON specification.

It is valid only when `CFG_FPU` is selected and the `TA_COP1` attribute is specified. In other cases, it is ignored.

`inifpscr` specifies the FPSCR value at initiation. The kernel sets the `inifpscr` value in FPSCR without checking an error in the `inifpscr` value.

Also refer to the following.

Reference: Section 15, Notes on FPU

An extended service call routine can also be defined statically by the configurator.

Error Detection through `CFG_MEMCHK`:

An `E_MACV` error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for `pk_dsvc`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_dsvc`
 - `size` = `sizeof(T_DSVC)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ`
- (2) The kernel domain does not have a read access permission for `pk_dsvc->svcrtn`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_dsvc->svcrtn`
 - `size` = 1
 - `domid` = Kernel domain
 - `pmmode` = `TPM_READ`

6.24.2 Issue Extended Service Call (cal_svc, ical_svc)

C-Language API:

```
ER_UINT ercd = cal_svc (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT
par4);
ER_UINT ercd = ical_svc (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT
par4);
```

Parameters:

FN	fncd	Function code of extended service call
VP_INT	par1	Parameter 1
VP_INT	par2	Parameter 2
VP_INT	par3	Parameter 3
VP_INT	par4	Parameter 4

Return Parameters:

ER_UINT	ercd	Return value from service call
---------	------	--------------------------------

Error Codes:

E_RSFN	[k]	Reserved function code (fncd is invalid or cannot be used)
--------	-----	--

Function:

Each service call executes the extended service call routine corresponding to the function code specified by parameter fncd.

par1 to par4 are passed to the extended service call routine as parameters.

6.24.3 Define Trap Routine (vdef_trp, ivdef_trp)

C-Language API:

```
ER ercd = vdef_trp(UINT dtrpno, VT_DTRP *pk_dtrp);
ER ercd = ivdef_trp(UINT dtrpno, VT_DTRP *pk_dtrp);
```

Parameters:

UINT	dtrpno	Trap number
VT_DTRP	*pk_dtrp	Pointer to the packet where the trap routine definition information is stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {
    ATR    trpatr;    0    4    Trap routine attribute
    FP     trprtn;    +4   4    Trap routine address
    UW     inifpscr;  +8   4    Initial FPSCR value
}T_DSVC;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_COP0, TA_COP1, TA_COP2, and TA_ASM in trpatr are not 0. (2) TA_COP0 is specified for trpatr while CFG_DSP is not selected. (3) TA_COP1 is specified for trpatr while CFG_FPU is not selected. (4) TA_COP2 is specified for trpatr while TA_COP1 is not specified. (5) Both TA_COP0 and TA_COP1 are specified for trpatr.
E_PAR	[p]	Parameter error (1) $0 \leq \text{trpno} \leq 15$ (2) $\text{trpno} > \text{CFG_NAXTRPNO}$ (3) pk_dtrp is not a 4-byte boundary address. (4) trprtn is an odd value.
E_MACV	[m]	Memory access violation

Function:

Defines a trap routine using the contents specified by pk_dtrp. The trap routine is called through a TRAPA instruction with the number specified by dtrpno.

The trap routine is assigned to the kernel domain and is executed in privileged mode.

Note that the following states do not change before and after the trap routine is initiated and terminated.

- Task or non-task context
- Dispatch-disabled or enabled state
- CPU-locked or unlocked state

While the trap routine called from a task is being executed, the task exception processing routine for that task is not initiated. When `rel_wai` is issued for that task, the task enters the WAITING-disabled state.

The following describes each parameter function.

(1) `dtrpno`

Parameter `dtrpno` specifies the target TRAPA number. Specify a value within the range from 16 to `CFG_MAXTRPNO`. 0 to 15 are reserved numbers for system use and must not be specified.

(2) `trpatr`

Specify the logical OR of the following values for `trpatr`.

(a) Language

Specify either one of the following values.

- `TA_HLNG` (H'00000000): High-level language
- `TA_ASM` (H'00000001): Assembly language

(b) Using a microcomputer with an on-chip DSP (when `CFG_DSP` is selected)

Specify `TA_COP0` to use the DSP.

- `TA_COP0` (H'00000100): The routine uses the DSP.

(c) Using a microcomputer with an on-chip FPU (when `CFG_FPU` is selected)

Specify `TA_COP1` to use the FPU for floating-point operations. Specify `TA_COP2` in addition to `TA_COP1` when using both banks of the FPU for matrix operations.

- `TA_COP1` (H'00000200): The routine uses FPU register bank 0 (`FPR0_BANK0` to `FPR15_BANK0`) and `FPUL`.
- `TA_COP2` (H'00000400): The routine uses FPU register bank 1 (`FPR0_BANK1` to `FPR15_BANK1`).

To specify `TA_COP2`, be sure to specify `TA_COP1` together; otherwise, an `E_RSATR` error will be returned.

Also refer to description (4), `inifpscr`.

(3) `trprtn`

Parameter `trprtn` specifies the start address of the trap routine.

(4) `inifpscr`

`inifpscr` is a parameter not specified in the μ ITRON specification.

It is valid only when `CFG_FPU` is selected and the `TA_COP1` attribute is specified. In other cases, it is ignored.

`inifpscr` specifies the FPSCR value at initiation. The kernel sets the `inifpscr` value in FPSCR without checking an error in the `inifpscr` value.

Also refer to the following.

Reference: Section 15, Notes on FPU

A trap routine can also be defined statically by the configurator.

If a TRAPA instruction for an undefined trap number is executed, `E_RSFN` is returned through the R0 register.

Error Detection through `CFG_MEMCHK`:

An `E_MACV` error will be returned in the following cases.

(1) The domain of the caller does not have a read access permission for `pk_dtrp`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base = pk_dtrp`
- `size = sizeof(VT_DTRP)`
- `domid = Domain of the caller`
- `pmmode = TPM_READ`

(2) The kernel domain does not have a read access permission for `pk_dtrp->trprtn`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base = pk_dtrp->trprtn`
- `size = 1`
- `domid = Kernel domain`
- `pmmode = TPM_READ`

6.25 System Configuration Management

Table 6.49 Service Calls for System Configuration Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
def_exc	Defines CPU exception handler	T/E/D/U
idef_exc		N/E/D/U
ref_cfg	Refers to configuration information	T/E/D/U
iref_cfg		N/E/D/U
ref_ver	Refers to version information	T/E/D/U
iref_ver		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.50 System Configuration Management Specifications

Item	Description
CPU exception handler number	A multiple of H'20 within the range from 0 to CFG_MAXINTNO (H'3fe0 max.)
CPU exception handler attributes	TA_HLNG: The handler is written in a high-level language. TA_ASM: The handler is written in assembly language.

6.25.1 Define CPU Exception Handler (def_exc, idf_exc)

C-Language API:

```
ER ercd = def_exc(EXCNO excno, T_DEXC *pk_dexc);  
ER ercd = idf_exc(EXCNO excno, T_DEXC *pk_dexc);
```

Parameters:

EXCNO	excno	CPU exception handler number
T_DEXC	*pk_dexc	Start address of the definition information of CPU exception handler

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ATR  excatr;      0    4    Handler attribute  
    FP   exchdr;     +4    4    Handler address  
    UW   excsr;      +8    4    SR at initiation  
}T_DEXC;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (1) The bits other than TA_ASM in excatr are not 0.
E_PAR	[p]	Parameter error (1) excno after being rounded down to a multiple of 0x20 is 0, H'20, H'140, H'160, or a value larger than CFG_MAXINTNO. (2) pk_dexc is not a 4-byte boundary address. (3) exchdr is an odd value.
E_MACV	[m]	Memory access violation

Function:

Each service call defines a CPU exception handler. The CPU exception handler is assigned to the kernel domain and is executed in privileged mode.

The actual codes of these service calls are the same as that for def_inh, which means that these service calls can be used to define an interrupt handler and def_inh can be used to define a CPU exception handler.

The following describes each parameter function.

(1) excno

Parameter excno specifies a CPU exception handler number. Specify an EXPEVT code of the CPU. excno is rounded down to a multiple of H'20 during processing.

Note that the interrupt code (INTEVT code) and exception code (EXPEVT code) are managed in the same code system in the SH microcomputer. This service call does not check whether the specified exeno value is an exception code or an interrupt code. If an interrupt code is specified, the service call operates in the same way as def_inh.

For some INTEVT and EXPEV codes, no handlers can be defined, or even when handlers can be defined, the handlers will not work. Table 6.51 shows these cases.

Table 6.51 Exceptional INTEVT and EXPEVT Codes

INTEVT or EXPEVT Code	Cause	Handler Definition	CPU Operation for Specified Interrupt or Exception	
			CFG_PROTMEM Is Not Selected	CFG_PROTMEM Is Selected
0	Power-on reset or H-UDI reset	Not possible	Branches to the reset vector (H'A0000000).	
H'20	Manual reset	Not possible		
H'40	TLB miss exception (read)	Possible	Initiates the defined handler. *	The kernel updates the TLB. If the TLB cannot be updated, the kernel initiates the memory access violation handler. The defined handler is never executed; its definition has no effect.
H'60	TLB miss exception (write)			
H'A0	TLB protection violation exception (read)			
H'C0	TLB protection violation exception (write)			
H'140	Instruction TLB multiple-hit exception or data TLB multiple-hit exception	Not possible	Branches to the reset vector (H'A0000000). *	Branches to the reset vector (H'A0000000).
H'160	TRAPA instruction	Not possible	According to the trap number, a kernel service call or user-defined trap routine is executed.	

Note: This type of exception will not occur usually unless the application enables the MMU.

Do not specify the following codes.

(a) CFG_TIMINTNO

CFG_TIMINTNO is an interrupt number used in the standard timer driver when CFG_OPTTMR is not selected. If a handler is specified for this number, the standard timer driver will not operate correctly.

(b) H'400, H'420, and H'440

These are interrupt numbers used in the optimized timer driver when CFG_OPTTMR is selected. If a handler is specified for any one of these numbers, the optimized timer driver will not operate correctly.

When def_ovr is not installed, H'440 is not used in the optimized timer driver and an interrupt handler can be defined for this number.

(2) excatr

Specify either one of the following values.

- TA_HLNG (H'00000000): High-level language
- TA_ASM (H'00000001): Assembly language

(3) excsr

excsr is a parameter not specified in the μ ITRON specification.

Parameter excsr specifies the value of the status register (SR) on startup of the interrupt handler. excsr is specified using the same bit position as the SR. Note that the SR value becomes as shown below when a CPU exception handler is actually initiated; only the block (BL) bit takes effect and the other bits are ignored.

- Mode (MD) bit: Always 1
- Register bank (RB) bit: Always 0
- Block (BL) bit: Set to the excsr value.
- Other bits: Value before the CPU exception occurs

When pk_dexc = NULL (0) is specified, the definition of the CPU exception handler is cancelled.

If this service call is issued while a CPU exception handler has been defined, the previous definition is canceled and replaced with the new definition.

A CPU exception handler can be statically defined through the configurator.

If a CPU exception for a number for which no handler has been defined occurs, control is passed to the system down routine.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following cases.

- (1) The domain of the caller does not have a read access permission for pk_dexc, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_dexc
- size = sizeof(T_DEXC)

- domid = Domain of the caller
 - pmmode = TPM_READ
- (2) The kernel domain does not have a read access permission for `pk_dexc->exchdr`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
- base= `pk_dexc->exchdr`
 - size = 1
 - domid = Kernel domain
 - pmmode = TPM_READ

6.25.2 Refer to Configuration Information (ref_cfg, iref_cfg)

C-Language API:

```
ER ercd = ref_cfg(T_RCFG *pk_rcfg);  
ER ercd = iref_cfg(T_RCFG *pk_rcfg);
```

Parameters:

T_RCFG	*pk_rcfg	Pointer to the packet where the configuration information is to be returned
--------	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RCFG	*pk_rcfg	Pointer to the packet where the configuration information is stored

Packet Structure:

```
typedef struct {  
    ID    maxtskid;    0    2    Maximum task ID  
    ID    rsv;         +2   2    (Reserved)  
    ID    maxsemid;    +4   2    Maximum semaphore ID  
    ID    maxflgid;    +6   2    Maximum event flag ID  
    ID    maxdtqid;    +8   2    Maximum data queue ID  
    ID    maxmbxid;    +10  2    Maximum mailbox ID  
    ID    maxmtxid;    +12  2    Maximum mutex ID  
    ID    maxmbfid;    +14  2    Maximum message buffer ID  
    ID    maxmplid;    +16  2    Maximum variable-size memory pool ID  
    ID    maxmpfid;    +18  2    Maximum fixed-size memory pool ID  
    ID    maxcycid;    +20  2    Maximum cyclic handler ID  
    ID    maxalmid;    +22  2    Maximum alarm handler ID  
    ID    maxs_fncd;    +24  4    Maximum function code of extended  
                                service call  
    ID    maxdomid;    +28  2    Maximum domain ID  
    ID    maxmppid;    +30  2    Maximum protected memory pool ID  
    ID    maxmbpid;    +32  2    Maximum protected mailbox ID  
}  
T_RCFG;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rcfg is not a 4-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call returns the system configuration information to the area indicated by `pk_rcfg`.

The following parameters are returned to the packet specified by `pk_rcfg`. The name enclosed in parentheses is the corresponding items to be set in the configurator.

- `maxtskid`: Returns the maximum task ID (`CFG_MAXTSKID`)
- `maxsemid`: Returns the maximum semaphore ID (`CFG_MAXSEMICID`)
- `maxflgid`: Returns the maximum event flag ID (`CFG_MAXFLGID`)
- `maxdtqid`: Returns the maximum data queue ID (`CFG_MAXDTQID`)
- `maxmbxid`: Returns the maximum mailbox ID (`CFG_MAXMBXID`)
- `maxmtxid`: Returns the maximum mutex ID (`CFG_MAXMTXID`)
- `maxmbfid`: Returns the maximum message buffer ID (`CFG_MAXMBFID`)
- `maxmplid`: Returns the maximum variable-size memory pool ID (`CFG_MAXMPLID`)
- `maxmpfid`: Returns the maximum fixed-size memory pool ID (`CFG_MAXMPFID`)
- `maxcycid`: Returns the maximum cyclic handler ID
`CFG_MAXCYCID + 1` is returned when `CFG_ACTION` is selected; otherwise, `CFG_MAXCYCID` is returned. In the former case, the cyclic handler with ID `CFG_MAXCYCID + 1` indicates the cyclic handler used by the debugging extension.
- `maxalmid`: Returns the maximum alarm handler ID (`CFG_MAXALMID`)
- `maxs_fnccd`: Returns the maximum extended SVC function code (`CFG_MAXSVCCD`)
- `maxdomid`: Maximum domain ID (always 31)
- `maxmppid`: Maximum protected memory pool ID (`CFG_MAXMPPID`)
0 is returned when the memory object protection function is not selected.
- `maxmbpid`: Maximum protected mailbox ID (`CFG_MAXMBPID`)
0 is returned when the memory object protection function is not selected.

The members of the `T_RCFG` structure are not defined in the μ ITRON specification; the μ ITRON specification does not define anything about the contents of the `T_RCFG` structure.

Error Detection through `CFG_MEMCHK`:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `pk_rcfg`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_rcfg`
 - `size` = `sizeof(T_RCFG)`
 - `domid` = Domain of the caller

— pmmode = TPM_READ|TPM_WRITE

6.25.3 Refer to Version Information (ref_ver, iref_ver)

C-Language API:

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

Parameters:

T_RVER	*pk_rver	Pointer to the packet where version information is to be returned
--------	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RVER	*pk_rver	Pointer to the packet where version information is stored

Packet Structure:

```
typedef struct {  
    UH    maker;           0      2    Manufacturer  
    UH    prid;            +2     2    Identification number  
    UH    spver;           +4     2    Specification version  
    UH    prver;           +6     2    Product version  
    UH    prno [4];        +8     8    Product management information  
    UH    pxver;           +16    2    Version number of the protection  
                                     function extension of μITRON4.0  
                                     specification  
}T_RVER;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rver is not a 2-byte boundary address.
E_MACV	[m]	Memory access violation

Function:

Each service call reads information on the version of the kernel currently in use and returns it to the area indicated by pk_rver.

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

(1) maker

Parameter maker indicates the manufacturer of this kernel. The value for this kernel is H'0115, which means Renesas.

(2) `prid`

Parameter `prid` indicates the number to identify the OS or VLSI type. The value for this kernel is H'12.

(3) `spver`

Parameter `spver` indicates the specifications to which the kernel conforms to, as follows.

- Bits 15 to 12: MAGIC (Number to identify the TRON specification series)
H'5 (μ ITRON specifications) for this kernel
- Bits 11 to 0: SpecVer (Version number of the TRON specification on which the product is based)
H'402 (version 4.02) for this kernel

(4) `prver`

Parameter `prver` indicates the version number of the kernel. For example, the value is H'012D when the version of the kernel is V.1.02.13.456.

(5) `prno`

Parameter `prno` indicates the product management information and the product number. The `prno[0]` to `prno[3]` values of this kernel are all H'0000.

(6) `pxver`

Parameter `pxver` indicates the version of the protection function extension of the μ ITRON 4.0 specification which the kernel conforms to. The value for this kernel is H'0100 (Ver.1.00).

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `pk_rver`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base = pk_rver`
- `size = sizeof(T_RVER)`
- `domid` = Domain of the caller
- `pmmode` = `TPM_READ|TPM_WRITE`

6.26 Memory Object Management Function

Table 6.52 Service Calls for Memory Object Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
sac_mem	Changes access permission vector for memory object	T/E/D/U
prb_mem	Checks access right for memory area	T/E/D/U
ref_mem	Refers to memory object state	T/E/D/U
vloc_tlb	Locks TLB entry	T/E/D/U
vunl_tlb	Unlocks TLB entry	T/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

The address space is classified into MMU mapped areas and MMU non-mapped areas. All memory objects must be allocated in MMU mapped areas. For details, refer to the following.

Reference: Section 5, Logical Address Space

Access to memory objects are controlled according to the attribute and access permission vector assigned to each memory object. For details, refer to the following.

Reference: Section 4.21, Memory Object Protection Function

6.26.1 Change Access Permission Vector for Memory Object (sac_mem)

C-Language API:

```
ER ercd = sac_mem(VP base, ACVCT *p_acvct);
```

Parameters:

VP	base	Memory object address
ACVCT	*p_acvct	Address of the packet where access permission vector for the memory object is stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ACPTN acptn1;    0    4    Write access permission pattern  
    ACPTN acptn2;    +4   4    Read access permission pattern  
}ACVCT;
```

Error Codes:

E_PAR	[p]	Parameter error (1) p_acvct is invalid. (2) p_acvct is not a 4-byte boundary address.
E_CTX	[k]	Context error (Called in a non-task context)
E_ILUSE	[k]	Illegal use of service call (1) The memory object including the address specified by base is placed in the protected mailbox queue (2) The memory object including the address specified by base contains a TLB-locked page.
E_NOEXS	[k]	Undefined (1) The memory object including the address specified by base does not exit.
E_MACV	[m]	Memory access violation

Function:

Service call sac_mem changes the access permission vector for the memory object including the address specified by base to a new access permission vector specified by p_acvct. If either one of the following memory objects is specified, an E_ILUSE error is returned.

- A protected memory block placed in a protected mailbox queue
- A memory object containing a TLB-locked page

Table 6.53 shows the access permission vectors that can be specified for p_acvct. They cannot be ORed when specified. If a value that is not listed in the table is specified, an E_PAR error is returned. Note that if the specified vector does not match the access permission for the calling domain, no error is returned. A memory object with the TA_RO attribute cannot be written to by any program, even from the kernel domain, regardless of the specified write access permission pattern.

Table 6.53 Specifiable Access Permission Vector

Access Permission Vector	acptn1 (Write Access Permission Pattern)	acptn2 (Read Access Permission Pattern)
TACT_KERNEL	TACP_KERNEL	TACP_KERNEL
TACT_PRW(domid) *	TACP(domid) *	TACP(domid) *
TACT_PRO(domid) *	TACP_KERNEL	TACP(domid) *
TACT_SRW	TACP_SHARED	TACP_SHARED
TACT_SRO	TACP_KERNEL	TACP_SHARED
TACT_SRPW(domid) *	TACP(domid) *	TACP_SHARED

Note: For parameter domid in these macros, a value within the range from 1 to 31 can be specified.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The calling domain does not have a read access permission for p_acvct, which means that an error will be returned if prb_mem is issued with the following parameters.
 - base = p_acvct
 - size = sizeof(ACVCT)
 - domid = Domain of the caller
 - pmmode = TPM_READ

6.26.2 Check Access Permission for Memory Area (prb_mem)

C-Language API:

```
ER ercd = prb_mem(VP base, SIZE size, ID domid, MODE pmmode);
```

Parameters:

VP	base	Start address of memory area
SIZE	size	Size of memory area (number of bytes)
ID	domid	ID of the domain accessing the memory area
MODE	pmmode	Access mode

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) domid < -1 (2) domid > 31
E_PAR	[p]	Parameter error (1) pmmode is invalid. (2) size = 0 (3) base + size exceeds 32 bits.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_OBJ	[k]	Invalid object (1) The address specified by base is included in a memory object, but the address specified by (base + size - 1) is outside the memory object.
E_NOEXS	[k]	Undefined (1) The memory object including the address specified by base does not exist.
E_MACV	[m]	Memory access violation (For the detailed error conditions, refer to Function below.)

Function:

Service call prb_mem checks whether the domain specified by domid has the access permission for the memory area which starts from the address specified by base and has the size specified by parameter size, and returns E_OK when the access is allowed.

The following values can be specified for domid.

(a) 1 to 31: User domain with the specified domid.

- (b) TDOM_SELF(0): Calling domain. When this service call is issued by an extended service call or trap routine being executed in a task context, the domain of the task that called the routine (the domain ID that can be acquired through get_did issued in the extended service call or trap routine) is specified.
- (c) TDOM_KERNEL(-1): Kernel domain.

Parameter pmmode specifies the access type to be checked. Either one or both of the following values can be specified for pmmode.

- TPM_READ (H'00000001): Checks whether read access is allowed.
- TPM_WRITE (H'00000002): Checks whether write access is allowed.

In specific, the following three modes can be specified.

(1) TPM_READ

Checks whether read access is allowed.

(2) TPM_WRITE

Checks whether write access is allowed. In this kernel, read access is always allowed when write access is allowed, which means that this mode is actually the same as specification (3) below.

(3) TPM_READ|TPM_WRITE

Checks whether both read access and write access are allowed.

The processing of this service call depends on whether the address specified by base is included in a memory object, as shown below.

(1) When base specifies an address in the MMU mapped area

When there is no memory object that includes the address specified by base, an E_NOEXS error is returned.

When a memory object includes the address specified by base but the specified area extends beyond the memory object, an E_OBJ error is returned.

In either of the following cases, an E_MACV error is returned.

- TPM_WRITE is specified for pmmode, and the memory object has the TA_RO attribute.
- domid specifies a user domain, and the access permission vector assigned for the memory object does not allow the domain specified by domid to access in the mode specified by pmmode.

(2) When base specifies an address in the MMU non-mapped area
pmmode is ignored, and domid is only used for distinction between the kernel domain and a user domain.

(a) When base specifies a logical address in the on-chip memory

This is the case when "MMU non-mapped area, accessible in any mode" or "MMU non-mapped area, not accessible in user non-DSP mode" is specified for CFG_IRAMUSAGE through the configurator and parameter base specifies an address in an on-chip memory area specified in "On-chip memory list". Note that when "MMU mapped area" is specified, description (1), When base specifies an address in the MMU mapped area, is applied.

When $(\text{base} + \text{size} - 1)$ is outside the on-chip memory area, an E_OBJ error is returned.

In other cases, E_OK is returned when "MMU non-mapped area, accessible in any mode" is specified. When "MMU non-mapped area, not accessible in user non-DSP mode" is specified, E_MACV is returned if domid specifies a user domain. Note that this service call always returns an error in this case while in actual operation, the on-chip memory can be accessed even from a user domain ($\text{SR.MD} = 0$) while $\text{SR.DSP} = 1$. When domid specifies the kernel domain, E_OK is returned.

(b) P3 or P4 area

When the area specified by base and size overlaps the P3 or P4 area, an E_MACV error is returned.

As the P3 area must not be used in this kernel, E_MACV is always returned.

The P4 area is accessible only from the kernel domain (privileged mode). However, the P4 area is outside the scope of inspection by this service call and this service call always returns E_MACV.

(c) Other cases (P1 or P2 area)

Any user domain must not access these areas. When domid specifies a user domain, an E_MACV error is returned. When domid specifies the kernel domain, E_OK is returned.

6.26.3 Refer to the Memory Object State (ref_mem)

C-Language API:

```
ER ercd = ref_mem(VP base, T_RMEM *pk_rmem);
```

Parameters:

VP	base	Memory object address
T_RMEM	*pk_rmem	Address of the packet where the memory object state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMEM	*pk_rmem	Address of the packed where the memory object state is stored

Packet Structure:

```
typedef struct {
    ACVCT acvct;          0      8      Access permission vector for the
                                memory object
}T_RMEM;
typedef struct {
    ACPTN acptn1;         0      4      Write access permission pattern
    ACPTN acptn2;        +4      4      Read, management, and reference access
                                permission pattern
}ACVCT;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmem is not a multiple of four.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_NOEXS	[k]	Undefined (1) The memory object including the address specified by base does not exist.
E_MACV	[m]	Memory access violation

Function:

Service call ref_mem refers to the state of the memory object that includes the address specified by base and returns the access permission vector (acvct) assigned for the memory object to the area specified by pk_rsem.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The calling domain does not have a read/write access permission for pk_rmem, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rmem
- size = sizeof(T_RMEM)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.26.4 Lock TLB Entry (vloc_tlb)

C-Language API:

```
ER ercd = vloc_tlb(VP base);
```

Parameters:

VP	base	Address in a page which is to be locked in TLB
----	------	--

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	[k]	Parameter error (1) The CFG_MAXLOCPAGE number of pages have been locked.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_ILUSE	[k]	Illegal use of service call (1) The memory object including the address specified by base is placed in a protected mailbox queue. (2) The page including the address specified by base has been locked. (3) The memory object including the address specified by base is not allowed to be locked.
E_NOEXS	[k]	Undefined (1) The memory object including the address specified by base does not exist.

Function:

Service call vloc_tlb registers in the TLB the MMU page including the memory object address specified by base and locks the TLB entry. When a page in either one of the following memory objects is specified, an E_ILUSE error is returned.

- A protected memory block placed in a protected mailbox queue.
- A memory object having one of the combinations of memory attribute and access permission vector shown in table 6.54.

Table 6.54 Memory Objects that Cannot be Locked

Access Permission Vector	Memory Attribute
TACT_PRO(domid)	TA_RW
TACT_SRO	TA_RW
TACT_SRPW(domid)	TA_RW

When the specified page has already been registered in the TLB but has not been locked, that entry is purged and the specified page is newly registered in another TLB entry.

When the specified page has already been locked, an E_ILUSE error is returned.

The locked page can be unlocked by a vunl_tlb service call.

After this service call is executed, the number of pages that can be locked in the system decreases by one. The maximum pages that can be locked is defined for CFG_MAXLOCPAGE through the configurator.

The locked TLB entry is never deleted, and no TLB miss occurs during access to the corresponding page. However, note that when a page is locked in the TLB, the rate of TLB miss for access to the other unlocked pages will increase.

The sac_mem and snd_mbp service calls cannot be used for the memory object including the locked page.

Before deleting the memory object including the locked page (the operations shown in table 6.55), be sure to unlock the page. If such an operation is attempted before the page is unlocked, the number of pages that can be locked will become less than the CFG_MAXLOCPAGE number.

Table 6.55 Memory Object Deleting Operations

Memory Object Including the Address Specified by base	Memory Object Deleting Operation
Stack area acquired from the system pool for the task in a user domain	del_tsk, exd_tsk
Fixed-size memory pool area acquired from the system pool	del_mpf
Variable-size memory pool area acquired from the system pool	del_mpl
Protected memory block acquired from a protected memory pool or protected memory block received from a protected mailbox	rel_mpp
Static memory object	None

Also refer to the following.

Reference: Section 10.7.17

Table 10.14 Page Size for SH4AL-DSP or SH-4A without Extended Functions

6.26.5 Unlock TLB Entry (vunl_tlb)

C-Language API:

```
ER ercd = vunl_tlb(VP base);
```

Parameters:

VP	base	Address in a page which is to be unlocked in TLB
----	------	--

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_CTX	[k]	Context error (1) Called in a non-task context.
E_ILUSE	[k]	Illegal use of service call (1) The page including the address specified by base has not been locked.
E_NOEXS	[k]	Undefined (1) The memory object including the address specified by base does not exist.

Function:

Service call `vunl_tlb` unlocks the TLB entry for the MMU page including the memory object address specified by `base`.

When the page including the specified address has not been locked, an `E_ILUSE` error is returned.

After this service call is executed, the number of pages that can be locked in the system increases by one.

Also refer to the following.

Reference: Section 10.7.17

Table 10.14 Page Size for SH4AL-DSP or SH-4A without Extended Functions

6.27 Protected Memory Pool Management

Table 6.56 Service Calls for Protected Memory Pool Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
icre_mpp	Creates protected memory pool	See note 3 below
pget_mpp	Polls and gets protected memory block	T/E/D/U
rel_mpp	Releases protected memory block	T/E/D/U
ref_mpp	Refers to protected memory pool state	T/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler
3. icre_mpp is dedicated to use in the initial definition routines created by the configurator. If it is used outside the initial definition routines, correct operation is not guaranteed.

Table 6.57 Protected Memory Pool Management Specifications

Item	Description
Protected memory pool ID	1 to CFG_MAXMPPID (31 max.)
Variable-size memory pool attribute	VTA_UNFRAGMENT: Sector management (reducing fragmentation in free space)

Protected memory pools are statically created by the configurator; they cannot be created through service calls.

Also refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

6.27.1 Create Protected Memory Pool (icre_mpp)

C-Language API:

```
ER ercd = icre_mpp(ID mppid, T_CMPP *pk_cmpp);
```

Parameters:

ID	mppid	Protected memory pool ID
T_CMPP	*pk_cmpp	Pointer to the packet where the protected memory pool creation information is stored

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Packet Structure:

```
typedef struct {  
    ATR    mppatr;      0    4    Protected memory pool attribute  
    SIZE   mppsz;      +4   4    Size of protected memory pool area  
                                   (number of bytes)  
    VP     mpp;        +8   4    Start address of protected memory  
                                   pool area  
    VP     mppmb;      +12  4    Start address of management area for  
                                   protected memory pool  
    UINT   minblkksz;  +16  4    Minimum block size  
    UINT   sctnum;     +20  4    Maximum number of sectors  
}T_CMPP;
```

Error Codes:

E_PAR	[k]	Parameter error
		(1) The mppsz bytes starting from the address specified by mpp are not in the MMU mapped area.
		(2) mpp is not a CFG_PAGESZ boundary address.
		(3) The VTA_UNFRAGMENT attribute is specified and sctnum = 0.

Function:

Service call `icre_mpp` creates a protected memory pool with the ID specified by `mpfid` using the contents specified by `pk_cmpp`.

This service call must not be issued in any application. This service call is issued only in the initial definition routines created by the configurator when creation of protected memory pools is specified through the configurator. This service call is implemented only for this purpose, and most error detection functions are omitted.

The following describes each parameter function.

(1) mppatr

Specify the logical OR of the following values for mppatr.

(a) Order of tasks in the queue for waiting for memory block acquisition

Only TA_TFIFO can be specified.

— TA_TFIFO (H'00000000): Task queue waiting for memory is managed on a FIFO basis.

(b) Read-only or readable/writable attribute specification

Either TA_RW or TA_RO can be specified.

— TA_RW (H'00000000): Readable/writable memory (RAM)

— TA_RO (H'00000001): Read-only memory (ROM)

(c) Cache specification

The following attributes can be specified.

TA_UNCACHE || (TA_CHCHE | [TA_WBACK || TA_WTHROUGH])

— TA_CACHE (H'00000000): Cached during read/write access

— TA_UNCACHE (H'00000002): Not cached during read/write access

— TA_WBACK (H'00000000): Copy-back operation for write access

— TA_WTHROUGH (H'00000004): Write-through operation for write access

(d) Management method

VTA_UNFRAGMENT can be specified.

— VTA_UNFRAGMENT (H'80000000): Sector management (reducing fragmentation in free space)

The VTA_UNFRAGMENT attribute is suitable for a memory pool from which a large number of small memory blocks are to be acquired. When this attribute is specified, small blocks are collectively allocated in specialized contiguous areas to leave larger possible contiguous areas.

Only when attribute VTA_UNFRAGMENT is specified, sctnum becomes valid. When sctnum is set to a larger value than $mppsz / (4096 \times 32)$, $mppsz / (4096 \times 32)$ is assumed.

For details, refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

(2) mpp and mppsz

Parameter mpp specifies the address of the protected memory pool to be created and mppsz specifies the size of the protected memory pool.

The specified memory area must be in the MMU mapped area and aligned with a CFG_PAGESZ boundary; otherwise, an E_PAR error is returned.

(3) minblksz and sctnum

These are parameters not defined in the μ ITRON specification.

These parameters are valid only when attribute VTA_UNFRAGMENT is specified but minblksz is always assumed as CFG_PAGESZ (4096). For details, refer to the above description of attribute VTA_UNFRAGMENT.

(4) mppmb

Parameter mppmb specifies the address of the kernel management area. The entity of mppmb is generated by the configurator. The mppmb address must be in the MMU non-mapped area and in an area that cannot be accessed in user mode.

6.27.2 Poll and Get Protected Memory Block (pget_mpp)

C-Language API:

```
ER_UINT blksize = pget_mpp(ID mppid, UINT memsize, VP *p_blk);
```

Parameters:

ID	mppid	Protected memory pool ID
UINT	memsize	Size of memory block to be acquired
VP	*p_blk	Pointer to the packet where the start address of the memory block is to be returned

Return Parameters:

ER_UINT	blksize	Size of acquired memory block (a positive value) or error code
VP	*p_blk	Pointer to the packet where the start address of the memory block is stored

Error Codes:

E_PAR		Parameter error
	[p]	(1) p_blk is not a 4-byte boundary address.
	[k]	(2) memsize > size of target protected memory pool
E_ID	[p]	Invalid ID number
		(1) mppid ≤ 0
		(2) mppid > CFG_MAXMPPID
E_CTX	[k]	Context error
		(1) Called in a non-task context.
E_NOMEM	[k]	Insufficient memory
		(1) Insufficient space in the resource pool
E_NOEXS	[k]	Undefined
		(1) Protected memory pool specified by mppid does not exist.
E_TMOUT	[k]	Polling failed
E_MACV	[m]	Memory access violation

Function:

Service call pget_mpp gets a memory block for the size (bytes) specified by memsize which is rounded up to a multiple of CFG_PAGESZ, from the protected memory pool specified by mppid, returns the start address of the acquired memory block to the area indicated by p_blk, and returns the block size through blksize.

The acquired memory block is handled as a memory object having the following attributes.

- Domain: Domain of the calling task, which is the same as the domain ID that can be obtained by calling get_did.

- Memory attribute: Same attribute as the protected memory pool specified by mppid (specified through the configurator)
- Access permission vector:
 - (a) For the kernel domain: TACT_KERNEL
 - (b) For a user domain: TACT_PRW(domid)
(domid is the ID of the domain where the target memory block is assigned)
- Start address: A 4-kbyte boundary address

Use `sac_mem` to change the access permission vector.

After the memory block has been acquired, the size of the free space in the protected memory pool will decrease by `blksz`. If there is not sufficient contiguous free space in the protected memory pool, an `E_TMOUT` error is returned.

At this time, the kernel consumes an area in the resource pool to manage the acquired memory blocks as memory objects. For details, refer to the following.

Reference: Section 13.2.3 (3), Protected memory pool: `pget_mpp`

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `p_blk`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `p_blk`
 - `size` = `sizeof(VP)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ|TPM_WRITE`

6.27.3 Release Protected Memory Block (rel_mpp)

C-Language API:

```
ER ercd = rel_mpp(ID mppid, VP blk);
```

Parameters:

ID	mppid	Protected memory pool ID
VP	blk	Start address of memory block

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR		Parameter error
	[p]	(1) blk is not a CFG_PAGESZ boundary address.
	[k]	(2) blk is an address outside the target protected memory pool area.
E_ID	[p]	Invalid ID number
		(1) mppid ≤ 0
		(2) mppid > CFG_MAXMPPID
E_CTX	[k]	Context error
		(1) Called in a non-task context.
E_ILUSE	[k]	Illegal use of service call
		(1) blk is not the start address of a protected memory block acquired from the target protected memory pool.
		(2) The domain of the protected memory block specified by blk differs from the domain of the task in RUNNING state.
		(3) The protected memory block specified by blk is placed in a protected mailbox queue.
E_NOEXS	[k]	Undefined
		(1) Protected memory pool specified by mppid does not exist.

Function:

Service call rel_mpp returns the memory block specified by blk to the protected memory pool specified by mppid.

The start address of the memory block acquired by service call pget_mpp must be specified as parameter blk.

The specified memory block can be released when the domain of the task in RUNNING state (domain obtained by get_did) matches the domain of the target memory block; otherwise, an E_ILUSE error is returned. The following shows examples.

- (1) When the memory block is in domain A and a task in domain B issues this service call, an E_ILUSE error is returned.
- (2) When the memory block is in domain A and the extended service call routine that was called by a task in the same domain (domain A) issues this service call, the memory block can be released because "the domain of the task in RUNNING state" is domain A where the task is assigned even if the extended service call routine is assigned to the kernel domain.

After the memory block has been released, the size of the free space in the protected memory pool will increase by the size of the released block. The management area allocated in the resource pool is also released.

6.27.4 Refer to Protected Memory Pool State (ref_mpp)

C-Language API:

```
ER ercd = ref_mpp(ID mppid, T_RMPP *pk_rmpp);
```

Parameters:

ID	mppid	Protected memory pool ID
T_RMPP	*pk_rmpp	Pointer to the packet where protected memory pool state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMPP	*pk_rmpp	Pointer to the packet where protected memory pool state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;      0    2    Wait task ID  
    SIZE    fmppsz;      +4    4    Total size of available memory area  
                                   (number of bytes)  
    UINT    fblksz;      +8    4    Maximum memory area available (number  
                                   of bytes)  
    SIZE    mppsz;       +12   4    Size of protected memory pool  
}T_RMPP;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmpp is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mppid ≤ 0 (2) mppid > CFG_MAXMPPID
E_CTX	[k]	Context error (1) Called in a non-task context.
E_NOEXS	[k]	Undefined (1) Protected memory pool specified by mppid does not exist.
E_MACV	[m]	Memory access violation

Function:

Service call ref_mpp refers to the status of the protected memory pool specified by mppid and returns the wait task ID (wtskid), the total size of current available memory area (fmppsz), the size of the maximum memory block available (fblksz), and the size of the protected memory pool (mppsz) to the area indicated by pk_rmpp. mppsz is a parameter not defined in the μ ITRON specification.

The free space is usually fragmented. The maximum contiguous free space is returned to parameter `fblksz`. The block up to the size indicated by `fblksz` can be acquired immediately by calling `pget_mpp`.

`wtskid` is a parameter reserved for future extension, and the kernel always returns 0 through it.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for `pk_rmpp`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base` = `pk_rmpp`
- `size` = `sizeof(T_RMPP)`
- `domid` = Domain of the caller
- `pmmode` = `TPM_READ|TPM_WRITE`

6.28 Protected Mailbox Management

Table 6.58 Service Calls for Protected Mailbox Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
cre_mbp	Creates protected mailbox	T/E/D/U
icre_mbp		N/E/D/U
acre_mbp	Creates protected mailbox and assigns mailbox ID automatically	T/E/D/U
iacre_mbp		N/E/D/U
del_mbp	Deletes protected mailbox	T/E/D/U
snd_mbp	Sends data to protected mailbox	T/E/D/U
rcv_mbp	Receives data from protected mailbox	T/E/U
prcv_mbp	Polls and receives data from protected mailbox	T/E/D/U
trcv_mbp	Receives data from protected mailbox with timeout function	T/E/U
ref_mbp	Refers to protected mailbox state	T/E/D/U
iref_mbp		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

Table 6.59 Protected Mailbox Management Specifications

Item	Description
Protected mailbox ID	1 to CFG_MAXMBPID (32767 max.)
Protected message priority	1 to CFG_MAXMSGPRI * (255 max.)
Attributes supported	TA_TFIFO: Wait task queue is managed on a FIFO basis. TA_TPRI: Wait task queue is managed on the current priority. TA_MFIFO: Message queue is managed on a FIFO basis. TA_MPRI: Message queue is managed on the current priority.
Note: This value is same as TMAX_MPRI defined in kernel_macro.h	

6.28.1 Create Protected Mailbox (cre_mbp, icre_mbp, acre_mbp, iacre_mbp)

C-Language API:

```
ER ercd = cre_mbp(ID mbpid, T_CMBP *pk_cmbp);
ER ercd = icre_mbp(ID mbpid, T_CMBP *pk_cmbp);
ER_ID mbpid = acre_mbp(T_CMBP *pk_cmbp);
ER_ID mbpid = iacre_mbp(T_CMBP *pk_cmbp);
```

Parameters:

T_CMBP	*pk_cmbp	Pointer to the packet where the protected mailbox creation information is stored
<cre_mbp, icre_mbp>		
ID	mbpid	Protected mailbox ID

Return Parameters:

<cre_mbp, icre_mbp>		
ER	ercd	Normal termination (E_OK) or error code
<acre_mbp, iacre_mbp>		
ER_ID	mbpid	Created protected mailbox ID (a positive value) or error code

Packet Structure:

```
typedef struct {
    ATR    mbpatr;      0    4    Protected mailbox attribute
    UINT   mbpcnt;      +4   4    Number of messages that can be stored
    PRI    maxmpri;     +8   2    Highest message priority
    VP     mbpmb;       +12  4    Start address of management area for
                                protected mailbox
}T_CMBP;
```

Error Codes:

E_RSATR	[p]	Reserved attribute (mbpatr is invalid.)
E_PAR	[p]	Parameter error (1) maxmpri ≤ 0 (2) maxmpri > CFG_MAXMSGPRI (3) pk_cmbp is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbpid ≤ 0 (2) mbpid > CFG_MAXMBPID
E_NOMEM	[k]	Insufficient memory (1) Insufficient space in the resource pool
E_NOID	[k]	NO ID available (only for acre_mbp)
E_OBJ	[k]	Invalid object state (1) Protected mailbox specified by mbpid already exists.

Function:

Service calls `cre_mbp` and `icre_mbp` create a protected mailbox with the ID specified by `mbpid` using the contents specified by `pk_cmbp`.

Service calls `acre_mbp` and `iacre_mbp` search for an unused protected mailbox ID and create a protected mailbox for that ID with the contents specified by parameter `pk_cmbp`. The created protected mailbox ID is returned as a return parameter. The range to search for an unused protected mailbox ID is 1 to `CFG_MAXMBPID`.

Parameter `mbpatr` specifies the order of the receive-waiting tasks and messages in the wait queues.

`mbpatr := ((TA_TFIFO || TA_TPRI) | (TA_MFIFO || TA_MPRI))`

- `TA_TFIFO` (H'00000000): Message receive-waiting queue is managed on a FIFO basis.
- `TA_TPRI` (H'00000001): Message receive-waiting queue is managed on the current priority.
- `TA_MFIFO` (H'00000000): Message queue is managed on a FIFO basis.
- `TA_MPRI` (H'00000002): Message queue is managed on the current priority.

`mbpcnt` and `mbpmb` are always ignored in this kernel. To ensure the portability of programs, specify an appropriate value for `mbpcnt` and `NULL` for `mbpmb`.

When the `TA_MPRI` attribute is specified and `maxmpri > 1`, the kernel uses an area in the resource pool to manage the mailbox. For details, refer to the following.

Reference: Section 13.2.2 (7), Protected mailbox

A protected mailbox can also be created statically by the configurator.

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

- (1) The domain of the caller does not have a read access permission for `pk_cmbp`, which means that an error will be returned if `prb_mem` is issued with the following parameters.
 - `base` = `pk_cmbp`
 - `size` = `sizeof(T_CMBP)`
 - `domid` = Domain of the caller
 - `pmmode` = `TPM_READ`

6.28.2 Delete Protected Mailbox (del_mbp)

C-Language API:

```
ER ercd = del_mbp(ID mbpid);
```

Parameters:

ID	mbpid	Protected mailbox ID
----	-------	----------------------

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_ID	[p]	Invalid ID number (1) $mbpid \leq 0$ (2) $mbpid > CFG_MAXMBPID$
E_CTX	[k]	Context error (1) Called in a non-task context.
E_OBJ	[k]	Invalid object state (1) A message is placed in the queue for the target protected mailbox.
E_NOEXS	[k]	Undefined (1) Protected mailbox specified by mbpid does not exist.

Function:

Service call del_mbp deletes the protected mailbox specified by parameter mbpid.

No error will occur even if there is a task waiting for a message in the protected mailbox indicated by mbpid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned. If there is a message in the protected mailbox, an E_OBJ error will be returned.

On deletion, the management area allocated in the resource pool is released.

6.28.3 Send Message to Protected Mailbox (snd_mbp)

C-Language API:

```
ER ercd = snd_mbp(ID mbpid, VP blk, PRI msgpri);
```

Parameters:

ID	mbpid	Protected mailbox ID
VP	blk	Start address of the protected memory block where send message is stored
PRI	msgpri	Message priority

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR		Parameter error
	[p]	(1) blk is not a CFG_PAGESZ boundary address.
	[k]	(2) The TA_MPRI attribute is specified for the target protected mailbox and msgpri ≤ 0 or msgpri > (highest message priority specified at creation)
E_ID	[p]	Invalid ID number
		(1) mbpid ≤ 0
		(2) mbpid > CFG_MAXMBPID
E_CTX	[k]	Context error
		(1) Called in a non-task context.
E_NOMEM	[k]	Insufficient memory
		(1) Insufficient space in the resource pool
E_ILUSE	[k]	Illegal use of service call
		(1) Protected memory block starting from the address specified by blk does not exist.
		(2) The domain of the protected memory block specified by blk does not match the domain of the task in RUNNING state.
		(3) The protected memory block specified by blk is placed in the protected mailbox queue.
		(4) The memory object including the address specified by blk contains a TLB-locked page.
E_NOEXS	[k]	Undefined
		(1) Protected mailbox specified by mbpid does not exist.

Function:

Service call snd_mbp sends a protected memory block starting from the address specified by blk as a message with the priority specified by msgpri to the protected mailbox specified by mbpid.

When the TA_MPRI attribute is not specified for the protected mailbox specified by mbpid, msgpri is ignored.

For parameter blk, only the start address of a protected memory block (a memory block acquired through pget_mpp) can be specified; otherwise, an E_ILUSE error is returned.

In addition, neither protected memory block shown below can be sent to a protected mailbox. If attempted, an E_ILUSE error is returned.

- A protected memory block placed in a protected mailbox queue
- A protected memory block including a TLB-locked page

The domain of the protected memory block specified by blk must be the same as the domain of the task in RUNNING state (domain obtained by get_did); otherwise, an E_ILUSE error is returned. The following shows examples.

- (1) When the memory block is in domain A and a task in domain B issues this service call, an E_ILUSE error is returned.
- (2) When the memory block is in domain A and the extended service call routine that was called by a task in the same domain (domain A) issues this service call, the memory block can be sent to the protected mailbox because "the domain of the task in RUNNING state" is domain A where the task is assigned even if the extended service call routine is assigned to the kernel domain.

If there is a task waiting to receive a message in the protected mailbox, the task at the head of the wait queue receives the message and is released from the WAITING state. At this time, the sent protected memory block changes its attributes as follows.

- Domain: Domain of the receiving task, which is the same as the domain ID that can be obtained by calling get_did.
- Access permission vector:
 - (a) For the kernel domain: TACT_KERNEL
 - (b) For a user domain: TACT_PRW(domid)
(domid is the ID of the domain where the target memory block is assigned)

If there are no tasks waiting to receive a message, the specified message is placed in the message-waiting queue. At this time, the kernel consumes an area in the resource pool to manage the messages. For details, refer to the following.

Reference: Resource pool consumption → Section 13.2.3 (4), Protected mailbox: snd_mbp

The message queue is managed according to the attribute specified at creation. At this time, the sent protected memory block attributes are changed as follows.

- Domain: Kernel domain
- Access permission vector: TACT_KERNEL

6.28.4 Receive Message from Protected Mailbox (rcv_mbp, prcv_mbp, trcv_mbp)

C-Language API:

```
ER_UINT blksize = rcv_mbp(ID mbpid, VP *p_blk);  
ER_UINT blksize = prcv_mbp(ID mbpid, VP *p_blk);  
ER_UINT blksize = trcv_mbp(ID mbpid, VP *p_blk, TMO tmout);
```

Parameters:

ID	mbpid	Protected mailbox ID
VP	*p_blk	Pointer to the area where the start address of the protected memory block holding the received message is to be returned
<trcv_mbp>		
TMO	tmout	Timeout specification

Return Parameters:

ER_UINT	blksize	Size of received protected memory block (a positive value) or error code
VP	*p_blk	Pointer to the start address of the protected memory block holding the received message

Error Codes:

E_PAR	[p]	Parameter error (1) tmout \leq -2 (2) p_blk is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbpid \leq 0 (2) mbpid > CFG_MAXMBPID
E_CTX	[k]	Context error (1) Called in a non-task context. (2) Called in dispatch-pended state in a task context (only for rcv_mbp and trcv_mbp)
E_NOEXS	[k]	Undefined (1) Protected mailbox specified by mbpid does not exist.
E_RLWAI	[k]	WAITING state is forcibly cancelled (only for rcv_mbp and trcv_mbp) (1) rel_wai service call was issued in the WAITING state. (2) An attempt was made to shift to WAITING state in WAITING-disabled state.
E_TMOUT	[k]	Polling failed or timeout
E_DLT	[k]	Waiting object deleted (1) Protected mailbox specified by mbpid was deleted.
E_MACV	[m]	Memory access violation

Function:

Each service call receives a message stored in a protected memory block in the protected mailbox specified by parameter `mbpid`. Then the start address of the protected memory block is returned through `p_blk` and the size through `blksz`.

With service calls `rcv_mbp` and `trcv_mbp`, if there are no messages in the protected mailbox, the calling task is placed in the wait queue to receive a message (receive-waiting queue). With service call `prcv_mbp`, if there are no messages in the mailbox, error code `E_TMOUT` is returned immediately. The wait queue is managed according to the attribute specified at creation.

After a message is received, the management area acquired from the resource pool by the kernel to manage the message when the message was sent is released.

The received protected memory block changes its attributes as follows.

- Domain: Domain of the caller. When this service call is issued from an extended service call or trap routine being executed in a task context, the protected memory block is assigned to the domain of the task that has called the extended service call or trap routine (the same domain ID that can be checked by issuing `get_did` from the extended service call or trap routine).
- Access permission vector:
 - (a) When a task in the kernel domain issued a receiving service call (`rcv_mbp`, `prcv_mbp`, or `trcv_mbp`): `TACT_KERNEL`
 - (b) When a task in a user domain issued a receiving service call (`rcv_mbp`, `prcv_mbp`, or `trcv_mbp`) or an extended service call or trap routine called from a task in a user domain issued a receiving service call (`rcv_mbp` or `trcv_mbp`): `TACT_PRW(domid)`; `domid` is the ID of the domain where the task is assigned.

Parameter `tmout` specified by service call `trcv_mbp` specifies the timeout period.

If a positive value is specified for parameter `tmout`, error code `E_TMOUT` is returned when the timeout period has passed without the wait release conditions being satisfied.

If `tmout = TMO_POL (0)` is specified, the same operation as for service call `prcv_mbp` will be performed.

If `tmout = TMO_FEVR (-1)` is specified, timeout monitoring is not performed. In other words, the same operation as for service call `rcv_mbp` will be performed.

If a value larger than 1 is specified for `CFG_TICDENO` (the denominator for time tick cycles), the maximum value that can be specified for `tmout` is `H'7fffffff/CFG_TICDENO`. If a value larger than this is specified, operation is not guaranteed.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for p_blk, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = p_blk
- size = sizeof(VP)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.28.5 Refer to Protected Mailbox State (ref_mbp, iref_mbp)

C-Language API:

```
ER ercd = ref_mbp(ID mbpid, T_RMBP *pk_rmbp);  
ER ercd = iref_mbp(ID mbpid, T_RMBP *pk_rmbp);
```

Parameters:

ID	mbpid	Protected mailbox ID
T_RMBP	*pk_rmbp	Pointer to the area where the protected mailbox state is to be returned

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
T_RMBP	*pk_rmbp	Pointer to the packet where the protected mailbox state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;      0      2      Wait task ID  
    VP      blk;         +4     4      Start address of the protected  
                                     memory block at the head of the  
                                     message queue  
    UINT    blkksz;      +8     4      Size of the protected memory block  
                                     at the head of the message queue  
}T_RMBP;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rmbp is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) mbpid ≤ 0 (2) mbpid > CFG_MAXMBPID
E_NOEXS	[k]	Undefined (1) Protected mailbox specified by mbpid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the state of the protected mailbox specified by parameter mbpid.

Each service call returns the wait task ID (wtskid), the start address of the protected memory block to be received next (pk_msg), and the size of the protected memory block (blksz) to the area specified by pk_rmbp.

If there is no task waiting in the specified protected mailbox, TSK_NONE (0) is returned as wtskid.

If there is no message to be received next, NULL (0) is returned through blk and an undefined value through blksz.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for pk_rmbp, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rmbp
- size = sizeof(T_RMBP)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.29 System Memory Management

Table 6.60 Service Calls for System Memory Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
vref_syp	Refers to system pool state	T/E/D/U
vref_rsp	Refers to resource pool state	T/E/D/U

Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function

2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

For details of the system pool and resource pool, refer to the following.

Reference: Section 4.24, System Memory Management

Also refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

6.29.1 Refer to System Pool State (vref_syp)

C-Language API:

```
ER ercd = vref_syp(VT_RSYP *pk_rsyp);
```

Parameters:

VT_RSYP	*pk_rsyp	Pointer to the packet where the system pool state is to be returned
---------	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
VT_RSYP	*pk_rsyp	Pointer to the packet where the system pool state is stored

Packet Structure:

```
typedef struct {
    ID      wtskid;      0      2      Wait task ID
    SIZE    freesz;      +4     4      Total size of available memory area
                                         (number of bytes)
    UINT    fblksiz;     +8     4      Maximum memory area available (number
                                         of bytes)
    SIZE    sypsz;       +12    4      Size of system pool
}VT_RSYP;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rsyp is not a 4-byte boundary address.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_MACV	[m]	Memory access violation

Function:

Service call vref_syp refers to the status of the system pool and returns the total size of current available memory area (freesz), the size of the maximum memory block available (fblksiz), and the size of the system pool (sypsz) to the area indicated by pk_rsyp. For wtskid, NTSK is always returned.

The free space is usually fragmented. The maximum contiguous free space is returned to parameter fblksiz.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rsymp, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rsymp
- size = sizeof(VT_RSYMP)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.29.2 Refer to Resource Pool State (vref_rsp)

C-Language API:

```
ER ercd = vref_rsp(VT_RRSP *pk_rrsp);
```

Parameters:

VT_RRSP	*pk_rrsp	Pointer to the packet where the resource pool state is to be returned
---------	----------	---

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
VT_RRSP	*pk_rrsp	Pointer to the packet where the resource pool state is stored

Packet Structure:

```
typedef struct {  
    ID      wtskid;      0      2      Wait task ID  
    SIZE    freesz;      +4     4      Total size of available memory area  
                                         (number of bytes)  
    UINT    fblks;      +8     4      Maximum memory area available (number  
                                         of bytes)  
    SIZE    rspsz;      +12    4      Size of resource pool  
}VT_RRSP;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rrsp is not a 4-byte boundary address.
E_CTX	[k]	Context error (1) Called in a non-task context.
E_MACV	[m]	Memory access violation

Function:

Service call vref_rsp refers to the status of the resource pool and returns the total size of current available memory area (freesz), the size of the maximum memory block available (fblks), and the size of the resource pool (rspsz) to the area indicated by pk_rrsp. For wtskid, NTSK is always returned.

The free space is usually fragmented. The maximum contiguous free space is returned to parameter fblks.

Error Detection through CFG_MEMCHK:

An E_MACV error will be returned in the following case.

- (1) The domain of the caller does not have a read/write access permission for pk_rrsp, which means that an error will be returned if prb_mem is issued with the following parameters.

- base = pk_rrsp
- size = sizeof(VT_RRSP)
- domid = Domain of the caller
- pmmode = TPM_READ|TPM_WRITE

6.30 Performance Management

Table 6.61 Service Calls for Performance Management

Service Call ¹	Description	System State ²
		T/N/E/D/U/L/C
vchg_ppc	Starts, stops, or initializes performance measurement	T/E/D/U
ivchg_ppc		N/E/D/U
vref_ppc	Refers to performance measurement result	T/E/D/U
ivref_ppc		N/E/D/U

- Notes: 1. [S]: Standard profile service calls
[s]: Service calls that are not standard profile service calls but are needed in order to use the standard profile function
2. T: Can be called in a task context
N: Can be called in a non-task context
E: Can be called in dispatch-enabled state
D: Can be called in dispatch-disabled state
U: Can be called in CPU-unlocked state
L: Can be called in CPU-locked state
C: Can be called from CPU exception handler

The performance management function measures performance such as the task execution time by using the program performance counters (PPC) in the microcomputer. This function can be used only when the target microcomputer has program performance counters.

Reference: Section 4.26, Performance Management

6.30.1 Start, Stop, or Initialize Performance Measurement (vchg_ppc, ivchg_ppc)

C-Language API:

```
ER_UINT oldmode = vchg_ppc(ID ctxid, MODE mode);  
ER_UINT oldmode = ivchg_ppc(ID ctxid, MODE mode);
```

Parameters:

ID	ctxid	Target context
MODE	mode	New accumulation mode

Return Parameters:

ER_UINT	oldmode	Previous accumulation mode
---------	---------	----------------------------

Error Codes:

E_PAR	[k]	Parameter error (1) mode is invalid.
E_ID	[p]	Invalid ID number (1) ctxid < -3 (2) ctxid > CFG_MAXTSKID (3) ctxid = TSK_SELF(0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by ctxid does not exist.

Function:

Each service call starts, stops, or initializes performance measurement for the context specified by ctxid.

The following values can be specified for ctxid.

- 1 to CFG_MAXTSKID: Task with task ID ctxid
- TSK_SELF(0): Calling task. When the service call is issued in a non-task context, an E_ID error is returned.
- -1: Kernel idling state
- -2: Non-task context + kernel
- -3: All programs

Parameter mode specifies the operation of the counters (start or stop).

```
mode:=(VTPPC_STA0||VTPPC_STP0) | (VTPPC_STA1||VTPPC_STP1)  
[|VTPPC_INI0][|VTPPC_INI1]
```

- VTPPC_STA0 (H'00000001): Resumes accumulation for counter 0.
- VTPPC_STP0 (H'00000000): Stops accumulation for counter 0.

- VTPPC_INI0 (H'00000004): Clears the accumulated data for counter 0 to 0.
- VTPPC_STA1 (H'00000002): Resumes accumulation for counter 1.
- VTPPC_STP1 (H'00000000): Stops accumulation for counter 1.
- VTPPC_INI1 (H'00000008): Clears the accumulated data for counter 1 to 0.

When a value other than -3 is specified for ctxid, the previous accumulation mode (shown below) is returned through oldmode.

oldmode:= (VTPPC_STA0|VTPPC_STP0) | (VTPPC_STA1|VTPPC_STP1)

- VTPPC_STA0 (H'00000001): Accumulation for counter 0 is in progress.
- VTPPC_STP0 (H'00000000): Accumulation for counter 0 stops.
- VTPPC_STA1 (H'00000002): Accumulation for counter 1 is in progress.
- VTPPC_STP1 (H'00000000): Accumulation for counter 1 stops.

When -3 is specified for ctxid, an undefined value is returned through oldmode.

When CFG_CONNECT is selected, the mode setting for counter 1 is ignored and the oldmode information about counter 1 has no meaning.

The performance measurement for kernel idling state and non-task context + kernel is initialized and then newly started when the kernel is started by vsta_kernel. The performance measurement for a task, on the other hand, is initialized and then newly started when the task is created.

Note that each performance counter itself cannot be started, stopped, or initialized.

6.30.2 Refer to Performance Measurement Result (vref_ppc, ivref_ppc)

C-Language API:

```
ER_UINT sts = vref_ppc(ID ctxid, VT_RPPC *pk_rppc);  
ER_UINT sts = ivref_ppc(ID ctxid, VT_RPPC *pk_rppc);
```

Parameters:

ID	ctxid	Target context
VT_RPPC	*pk_rppc	Pointer to the packet where the accumulated value is to be returned

Return Parameters:

ER_UINT	sts	PPC state (a positive value) or error code
VT_RPPC	*pk_rppc	Pointer to the packet where the accumulated value is stored

Packet Structure:

```
typedef struct {  
    UW    ppc0;      0      4      PPC0 counter  
    UW    ppc1;      +4     4      PPC1 counter  
}VT_RPPC;
```

Error Codes:

E_PAR	[p]	Parameter error (1) pk_rppc is not a 4-byte boundary address.
E_ID	[p]	Invalid ID number (1) ctxid < -2 (2) ctxid > CFG_MAXTSKID (3) ctxid = TSK_SELF(0) is specified in a non-task context.
E_NOEXS	[k]	Undefined (1) Task specified by ctxid does not exist.
E_MACV	[m]	Memory access violation

Function:

Each service call refers to the accumulated values of the performance counters.

The following values can be specified for ctxid.

- 1 to CFG_MAXTSKID: Task with task ID ctxid
- TSK_SELF(0): Calling task. When the service call is issued in a non-task context, an E_ID error is returned.
- -1: Kernel idling state

- -2: Non-task context

The accumulated values of the performance counters for the context specified by `ctxid` are returned through `pk_rppc`.

The following information is returned through `sts`. When 1 or 2 is returned, the information returned through `pk_rppc` may be incorrect, but there is no means of checking whether the information is correct.

- 0: No overflow in performance counters 0 and 1
- 1: Overflow in performance counter 0 (this value is never returned when `CFG_CONNECT` is selected)
- 2: Overflow in performance counters 0 and 1

Error Detection through CFG_MEMCHK:

An `E_MACV` error will be returned in the following case.

(1) The domain of the caller does not have a read/write access permission for `pk_rppc`, which means that an error will be returned if `prb_mem` is issued with the following parameters.

- `base` = `pk_rppc`
- `size` = `sizeof(VT_RPPC)`
- `domid` = Domain of the caller
- `pmmode` = `TPM_READ|TPM_WRITE`

Section 7 Cache Support Functions

7.1 Overview

The cache support functions provide cache-related operations such as writing the cache contents back to memory or clearing the cache contents.

The header file for the cache support functions is stored in the include\ directory. To use these functions, include the header file.

The actual code for the cache support functions is stored as a relocatable object in the lib\elf\ directory. This relocatable object should be embedded on the kernel side.

Table 7.1 shows the cache support functions provided by V.1.01 Release00 of the HI7300/PX.

Table 7.1 Cache Support Functions

Overview of Target Cache Hardware Specifications	Target CPU (Typical Microcomputers Including Cache)	Header File	Relocatable Objects
<ul style="list-style-type: none">• Separate instruction cache and operand cache• Four-way set-associative• Virtual address index/physical address tag• Line size: 32 bytes	SH4AL-DSP, SH-4A (without extended functions), SH73180, and SH7780	cache_sh4a.h	<ul style="list-style-type: none">• cache_sh4a_big.rel (for big endian)• cache_sh4a_little.rel (for little endian)
<ul style="list-style-type: none">• Separate instruction cache and operand cache• Four-way set-associative• Virtual address index/physical address tag• Line size: 32 bytes• Way prediction in the instruction cache	SH4AL-DSP, SH-4A (with extended functions), SH7343, and SH7785	cache_shx2.h	<ul style="list-style-type: none">• cache_shx2_big.rel (for big endian)• cache_shx2_little.rel (for little endian)

7.2 Notes

- (1) A cache support function can be called only in the privileged mode (from a program in the kernel domain). If a cache support function is called in the user mode, an exception usually occurs within the function.
- (2) Note that incorrect use of a cache support function may affect system operation; for example, coherence between the cache and memory may not be maintained. Before using cache support functions, fully understand the specifications of the cache in the target microcomputer and the behavior of the functions.

7.3 Functions in cache_sh4a.h

The following functions are provided by cache_sh4a.h.

- sh4a_vini_cac(): Initializes the cache.
- sh4a_vclr_cac(): Clears the cache.
- sh4a_vfls_cac(): Flushes the operand cache.
- sh4a_vinv_cac(): Invalidates the cache.

7.3.1 Initialize Cache (sh4a_vini_cac)

C-Language API:

```
ER ercd = sh4a_vini_cac(ATR cacatr, UINT icsize, UINT ocsize);
```

Parameters:

ATR	cacatr	Cache attribute
UINT	icsize	Size of the instruction cache
UINT	ocsiz	Size of the operand cache

Return Parameter:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

No error code is returned

Function:

This function initializes the cache. To be more specific, CCR and RAMCR in the processor are set to the values determined by the specified cacatr as described later.

CCR and RAMCR are modified by instructions placed in the P2 area while the BL bit in SR is 1.

A logical OR of the following values can be specified for cacatr. The kernel does not check errors for the value specified for cacatr.

This function writes 1 to the ICI and OCI bits in CCR regardless of the cacatr setting; that is, the cache contents before this function call are all cleared.

- TCAC_IC_ENABLE (H'00000100)
Setting this value enables the instruction cache (CCR.ICE = 1); otherwise, the instruction cache is disabled (CCR.ICE = 0).
- TCAC_OC_ENABLE (H'00000001)
Setting this value enables the operand cache (CCR.OCE = 1); otherwise, the operand cache is disabled (CCR.OCE = 0).
- TCAC_IC_2WAY (H'00800000)
Setting this value specifies 2-way instruction cache (RAMCR.IC2W = 1); otherwise, 4-way instruction cache is specified (RAMCR.IC2W = 0).

- TCAC_OC_2WAY (H'00400000)
Setting this value specifies 2-way operand cache (RAMCR.OC2W = 1); otherwise, 4-way operand cache is specified (RAMCR.OC2W = 0).
- TCAC_P1_CB (H'00000004)
Setting this value selects the copy-back mode as the write mode for the P1 area (CCR.CB = 1); otherwise, the write-through mode is selected (CCR.CB = 0).
- TCAC_P0_WT (H'00000002)
Setting this value selects the write-through mode as the write mode for the P0/U0 area (CCR.WT = 1); otherwise, the copy-back mode is selected (CCR.WT = 0).

Specify the sizes (bytes) of the instruction cache and operand cache in the target microcomputer through icsize and ocsiz, respectively. The kernel does not check whether the specified sizes are correct.

Be sure to call this function before using other cache support functions.

7.3.2 Clear Cache (sh4a_vclr_cac)

C-Language API:

```
ER ercd = sh4a_vclr_cac(VP clradr1, VP clradr2, MODE mode);
```

Parameters:

VP	clradr1	Start address of cache clearing
VP	clradr2	End address of cache clearing
MODE	mode	Target cache

Return Parameter:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (1) clradr1 > clradr2 (2) mode is invalid.
E_OBJ	Target cache specified by mode is disabled.

Function:

This function clears the cache. To be more specific, the cache contents are invalidated, and if the operand cache has data that has not been written back to memory, the data is written to memory.

The target cache is specified by mode. Any one of the following values can be specified for mode.

- TC_FULL (H'00000000): Clears both the instruction cache and operand cache.
- TC_EXCLUDE_IC (H'00000001): Clears only the operand cache (excludes the instruction cache).
- TC_EXCLUDE_OC (H'00000002): Clears only the instruction cache (excludes the operand cache).

The address range to be cleared is specified by clradr1 and clradr2. clradr1 is rounded down to a multiple of 32, and clradr2 is rounded up to (a multiple of 32) - 1.

(1) Clearing Specified Address Range

This function clears the entries corresponding to the logical address range from clradr1 to clradr2 in the cache specified by mode. When the operand cache is specified as a target (when TC_FULL

or TC_EXCLUDE_IC is specified for mode), this function copies dirty entries (entries that have not been written to memory) back to memory before clearing the entries.

This function repeats execution of the following instructions for the range from cladr1 to cladr2.

- When mode = TC_FULL: ICBI and OCBP instructions
- When mode = TC_EXCLUDE_IC: OCBP instruction
- When mode = TC_EXCLUDE_OC: ICBI instruction

During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit in SR is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for cladr1 and cladr2. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Clearing All Entries

Specifying cladr1 = 0 and cladr2 = H'ffffff clears all entries in the cache specified by mode. This function performs the following processing.

- (a) When TC_FULL or TC_EXCLUDE_OC is specified for mode, this function sets the ICI bit in CCR to 1 to invalidate all entries in the instruction cache. CCR is modified through an instruction placed in the P2 area while the BL bit in SR is 1.
- (b) After step (a), when TC_FULL or TC_EXCLUDE_IC is specified for mode, this function writes V = 0 and U = 0 to all entries in the memory-mapped operand cache. At the same time, the dirty entries (U = 1) are copied back to memory. During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. This function can be called even while the BL bit in SR is 1 when clearing all entries.

7.3.3 Flush Operand Cache (sh4a_vfls_cac)

C-Language API:

```
ER ercd = sh4a_vfls_cac(VP flsadr1, VP flsadr2);
```

Parameters:

VP	flsadr1	Start address of cache flushing
VP	flsadr2	End address of cache flushing

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (flsadr1 > flsadr2)
E_OBJ	Operand cache is disabled.

Function:

This function flushes the operand cache. To be more specific, when the operand cache has data that has not been written to memory, the data is copied back to memory.

The address range to be flushed is specified by flsadr1 and flsadr2. flsadr1 is rounded down to a multiple of 32, and flsadr2 is rounded up to (a multiple of 32) - 1.

(1) Flushing Specified Address Range

This function flushes the entries corresponding to the logical address range from flsadr1 to flsadr2 in the operand cache, that is, when the specified entries have not been written to memory, the entries are copied back to memory.

This function repeats execution of the OCBWB instruction for the range from flsadr1 to flsadr2. During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for flsadr1 and flsadr2. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Flushing All Entries

Specifying flsadr1 = 0 and flsadr2 = H'ffffffff flushes all entries in the operand cache. This function performs the following processing.

- This function reads all entries in the memory-mapped operand cache, and writes V = 1 and U = 0 to the valid (V = 1) entries. At the same time, the dirty entries (U = 1) are copied back to memory. This read and write processing is done by temporarily setting the BL bit in SR to 1. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. This function can be called even while the BL bit in SR is 1 when flushing all entries.

7.3.4 Invalidate Cache (sh4a_vinv_cac)

C-Language API:

```
ER ercd = sh4a_vinv_cac(VP invadr1, VP invadr2, MODE mode);
```

Parameters:

VP	invadr1	Start address of cache invalidation
VP	invadr2	End address of cache invalidation
MODE	mode	Target cache

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (1) invadr1 > invadr2 (2) mode is invalid.
E_OBJ	Target cache is disabled.

Function:

This function invalidates the cache.

The target cache is specified by mode. Any one of the following values can be specified for mode.

- TC_FULL (H'00000000): Invalidates both the instruction cache and operand cache.
- TC_EXCLUDE_IC (H'00000001): Invalidates the operand cache only (excludes the instruction cache).
- TC_EXCLUDE_OC (H'00000002): Invalidates the instruction cache only (excludes the operand cache).

The address range to be invalidated is specified by invadr1 and invadr2. invadr1 is rounded down to a multiple of 32, and invadr2 is rounded up to (a multiple of 32) - 1.

(1) Invalidating Specified Address Range

This function invalidates the entries corresponding to the logical address range from invadr1 to invadr2 in the cache specified by mode. When the operand cache is specified as a target (when TC_FULL or TC_EXCLUDE_IC is specified for mode), this function does not copy dirty entries

(the entries that have not been written to memory) back to memory, that is, the data in the entries will be lost.

This function repeats execution of the following instructions for the range from `invadr1` to `invadr2`.

- When mode = `TC_FULL`: ICBI and OCBI instructions
- When mode = `TC_EXCLUDE_IC`: OCBI instruction
- When mode = `TC_EXCLUDE_OC`: ICBI instruction

During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for `invadr1` and `invadr2`. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Flushing All Entries

Specifying `invadr1 = 0` and `invadr2 = 0xffffffff` flushes all entries in the cache specified by mode. This function manipulates the following bits in CCR according to the specified mode. CCR is modified through an instruction placed in the P2 area while the BL bit in SR is 1.

- When mode = `TC_FULL`: Sets the ICI and OCI bits in CCR to 1.
- When mode = `TC_EXCLUDE_IC`: Sets the OCI bit in CCR to 1.
- When mode = `TC_EXCLUDE_OC`: Sets the ICI bit in CCR to 1.

7.4 Functions in cache_shx2.h

The following functions are provided by cache_shx2.h.

- shx2_vini_cac(): Initializes the cache.
- shx2_vclr_cac(): Clears the cache.
- shx2_vfls_cac(): Flushes the operand cache.
- shx2_vinv_cac(): Invalidates the cache.

7.4.1 Initialize Cache (shx2_vini_cac)

C-Language API:

```
ER ercd = shx2_vini_cac(ATR cacatr, UINT icsize, UINT ocsiz);
```

Parameters:

ATR	cacatr	Cache attribute
UINT	icsize	Size of the instruction cache
UINT	ocsiz	Size of the operand cache

Return Parameter:

ER	ercd	Normal termination (E_OK)
----	------	---------------------------

Error Codes:

No error code is returned

Function:

This function initializes the cache. To be more specific, CCR and RAMCR in the processor are set to the values determined by the specified cacatr as described later.

CCR and RAMCR are modified by instructions placed in the P2 area while the BL bit in SR is 1.

A logical OR of the following values can be specified for cacatr. The kernel does not check errors for the value specified for cacatr.

This function writes 1 to the ICI and OCI bits in CCR regardless of the cacatr setting; that is, the cache contents before this function call are all cleared.

- **TCAC_IC_ENABLE (H'00000100)**
Setting this value enables the instruction cache (CCR.ICE = 1); otherwise, the instruction cache is disabled (CCR.ICE = 0).
 - **TCAC_OC_ENABLE (H'00000001)**
Setting this value enables the operand cache (CCR.OCE = 1); otherwise, the operand cache is disabled (CCR.OCE = 0).
 - **TCAC_IC_2WAY (H'00800000)**
Setting this value specifies 2-way instruction cache (RAMCR.IC2W = 1); otherwise, 4-way instruction cache is specified (RAMCR.IC2W = 0).
 - **TCAC_OC_2WAY (H'00400000)**
Setting this value specifies 2-way operand cache (RAMCR.OC2W = 1); otherwise, 4-way operand cache is specified (RAMCR.OC2W = 0).
 - **TCAC_P1_CB (H'00000004)**
Setting this value selects the copy-back mode as the write mode for the P1 area (CCR.CB = 1); otherwise, the write-through mode is selected (CCR.CB = 0).
 - **TCAC_P0_WT (H'00000002)**
Setting this value selects the write-through mode as the write mode for the P0/U0 area (CCR.WT = 1); otherwise, the copy-back mode is selected (CCR.WT = 0).
 - **TCAC_IC_WPD (H'00200000)**
Setting this value enables way prediction in the instruction cache (CCR.ICWPD = 1); otherwise, way prediction in the instruction cache is disabled (CCR.ICWPD = 0).
 - ~~• **TCAC_L2_ENABLE (H'00010000)**
Setting this value enables the level-2 cache (RAMCR.L2E = 1); otherwise, the level-2 cache is disabled (RAMCR.L2E = 0).~~
 - ~~• **TCAC_L2_FC (H'00020000)**
Setting this value selects the level-2 cache forcible coherency mode (RAMCR.L2FC = 1); otherwise, the level-2 cache forcible coherency mode is not selected (RAMCR.L2FC = 0).~~
- ~~Do not specify TCAC_L2_ENABLE or TCAC_L2_FC if the microcomputer being used does not support the level-2 cache. Specify the sizes (bytes) of the instruction cache and operand cache in the target microcomputer through icsize and ocsz, respectively. The kernel does not check whether the specified sizes are correct.~~

Be sure to call this function before using other cache support functions.

7.4.2 Clear Cache (shx2_vclr_cac)

C-Language API:

```
ER ercd = shx2_vclr_cac(VP clradr1, VP clradr2, MODE mode);
```

Parameters:

VP	clradr1	Start address of cache clearing
VP	clradr2	End address of cache clearing
MODE	mode	Target cache

Return Parameter:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (1) clradr1 > clradr2 (2) mode is invalid.
E_OBJ	Target cache specified by mode is disabled.

Function:

This function clears the cache. To be more specific, the cache contents are invalidated, and if the operand cache has data that has not been written back to memory, the data is written to memory.

The target cache is specified by mode. Any one of the following values can be specified for mode.

- TC_FULL (H'00000000): Clears both the instruction cache and operand cache.
- TC_EXCLUDE_IC (H'00000001): Clears only the operand cache (excludes the instruction cache).
- TC_EXCLUDE_OC (H'00000002): Clears only the instruction cache (excludes the operand cache).

The address range to be cleared is specified by clradr1 and clradr2. clradr1 is rounded down to a multiple of 32, and clradr2 is rounded up to (a multiple of 32) - 1.

(1) Clearing Specified Address Range

This function clears the entries corresponding to the logical address range from clradr1 to clradr2 in the cache specified by mode. When the operand cache is specified as a target (when TC_FULL

or TC_EXCLUDE_IC is specified for mode), this function copies dirty entries (entries that have not been written to memory) back to memory before clearing the entries.

This function repeats execution of the following instructions for the range from cladr1 to cladr2.

- When mode = TC_FULL: ICBI and OCBP instructions
- When mode = TC_EXCLUDE_IC: OCBP instruction
- When mode = TC_EXCLUDE_OC: ICBI instruction

During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit in SR is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for cladr1 and cladr2. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Clearing All Entries

Specifying cladr1 = 0 and cladr2 = H'ffffff clears all entries in the cache specified by mode. This function performs the following processing.

- (a) When TC_FULL or TC_EXCLUDE_OC is specified for mode, this function sets the ICI bit in CCR to 1 to invalidate all entries in the instruction cache. CCR is modified through an instruction placed in the P2 area while the BL bit in SR is 1.
- (b) After step (a), when TC_FULL or TC_EXCLUDE_IC is specified for mode, this function executes an OCBP instruction for all entries in the memory-mapped operand cache. Thus, the dirty entries (U = 1) are copied back to memory and invalidated (V = 0). During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. This function can be called even while the BL bit in SR is 1 when clearing all entries.

7.4.3 Flush Operand Cache (shx2_vfls_cac)

C-Language API:

```
ER ercd = shx2_vfls_cac(VP flsadr1, VP flsadr2);
```

Parameters:

VP	flsadr1	Start address of cache flushing
VP	flsadr2	End address of cache flushing

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (flsadr1 > flsadr2)
E_OBJ	Operand cache is disabled.

Function:

This function flushes the operand cache. To be more specific, when the operand cache has data that has not been written to memory, the data is copied back to memory.

The address range to be flushed is specified by flsadr1 and flsadr2. flsadr1 is rounded down to a multiple of 32, and flsadr2 is rounded up to (a multiple of 32) - 1.

(1) Flushing Specified Address Range

This function flushes the entries corresponding to the logical address range from flsadr1 to flsadr2 in the operand cache, that is, when the specified entries have not been written to memory, the entries are copied back to memory.

This function repeats execution of the OCBWB instruction for the range from flsadr1 to flsadr2. During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for flsadr1 and flsadr2. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Flushing All Entries

Specifying flsadr1 = 0 and flsadr2 = H'ffffffff flushes all entries in the operand cache. This function performs the following processing.

- This function reads all entries in the memory-mapped operand cache, and writes V = 1 and U = 0 to the valid (V = 1) entries. At the same time, the dirty entries (U = 1) are copied back to memory. This read and write processing is done by temporarily setting the BL bit in SR to 1. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. This function can be called even while the BL bit in SR is 1 when flushing all entries.

7.4.4 Invalidate Cache (shx2_vinv_cac)

C-Language API:

```
ER ercd = shx2_vinv_cac(VP invadr1, VP invadr2, MODE mode);
```

Parameters:

VP	invadr1	Start address of cache invalidation
VP	invadr2	End address of cache invalidation
MODE	mode	Target cache

Return Parameters:

ER	ercd	Normal termination (E_OK) or error code
----	------	---

Error Codes:

E_PAR	Parameter error (1) invadr1 > invadr2 (2) mode is invalid.
E_OBJ	Target cache is disabled.

Function:

This function invalidates the cache.

The target cache is specified by mode. Any one of the following values can be specified for mode.

- TC_FULL (H'00000000): Invalidates both the instruction cache and operand cache.
- TC_EXCLUDE_IC (H'00000001): Invalidates the operand cache only (excludes the instruction cache).
- TC_EXCLUDE_OC (H'00000002): Invalidates the instruction cache only (excludes the operand cache).

The address range to be invalidated is specified by invadr1 and invadr2. invadr1 is rounded down to a multiple of 32, and invadr2 is rounded up to (a multiple of 32) - 1.

(1) Invalidating Specified Address Range

This function invalidates the entries corresponding to the logical address range from invadr1 to invadr2 in the cache specified by mode. When the operand cache is specified as a target (when TC_FULL or TC_EXCLUDE_IC is specified for mode), this function does not copy dirty entries

(the entries that have not been written to memory) back to memory, that is, the data in the entries will be lost.

This function repeats execution of the following instructions for the range from `invadr1` to `invadr2`.

- When mode = `TC_FULL`: ICBI and OCBP instructions
- When mode = `TC_EXCLUDE_IC`: OCBI instruction
- When mode = `TC_EXCLUDE_OC`: ICBI instruction

During this processing, the SR value remains the same as when this function is called. When no interrupt should be accepted during this function processing, mask interrupts and then call this function. Note that this function must not be called while the BL bit in SR is 1 when the MMU is enabled and an MMU mapped area is specified; while the BL bit is 1, a TLB-related exception may occur in the above instruction execution and the CPU is reset in this case.

In this function, only the basic error check shown in the Error Codes description is performed for `invadr1` and `invadr2`. Accordingly, make sure that the addresses such as those listed below are not included in the address range.

- P2, P3, and P4 areas
- An address corresponding to a physical address in the control register area
- An address corresponding to a physical address in the X/Y memory
- An address that is in the P0/U0 area and is not in a memory object

(2) Flushing All Entries

Specifying `invadr1 = 0` and `invadr2 = 0xffffffff` flushes all entries in the cache specified by mode. This function manipulates the following bits in CCR according to the specified mode. CCR is modified through an instruction placed in the P2 area while the BL bit in SR is 1.

- When mode = `TC_FULL`: Sets the ICI and OCI bits in CCR to 1.
- When mode = `TC_EXCLUDE_IC`: Sets the OCI bit in CCR to 1.
- When mode = `TC_EXCLUDE_OC`: Sets the ICI bit in CCR to 1.

Section 8 Application Program Creation

8.1 Tasks

8.1.1 Writing a Task

A task must be written as a C-language function as shown in figure 8.1. Use an `ext_tsk` or `exd_tsk` service call to end a task. If execution is returned from a task without issuing `ext_tsk` or `exd_tsk`, it is assumed that `ext_tsk` has been issued and the same operation as `ext_tsk` is performed.

```
#include "kernel.h"
#pragma nogsave(Task)          ← (1)
void Task(VP_INT exinf)       ← (2)
{
    /* Processing */
    ext_tsk();
}
```

Figure 8.1 Example of a Task

Description:

- (1) `#pragma nogsave` can be specified to reduce the amount of the stack area to be used, except when execution is returned from a task function.
- (2) When a task is initiated by `sta_tsk`, `stacd` specified by `sta_tsk` is passed through `exinf`. When a task is initiated by `act_tsk` or the `TA_ACT` attribute specified at task creation, the extended information regarding the task is passed through `exinf`.

A task function can also be written as an infinite loop as shown in figure 8.2.

```

#include "kernel.h"

#pragma noregsave(Task)                ← (1)
void Task(VP_INT exinf)
{
    for(;;) {
        /* Processing */
    }
}

```

Figure 8.2 Example of a Task Written as an Infinite Loop

Description:

(1) #pragma noregsave can be specified to reduce the amount of the stack area to be used.

8.1.2 Rules on Using Registers

Table 8.1 shows the rules on using the registers and their initial values in a task.

Table 8.1 Rules on Using Registers and Initial Register Values in a Task

No.	Register	Use * ¹	Initial Value
1	PC	—	Task address
2	SR	O	See table 8.2
3	R0 to R3	√	Undefined
4	R4	√	When activated by TA_ACT attribute or act_tsk: exinf specified at task creation When activated by sta_tsk: stacd specified by sta_tsk
5	R5	√	Undefined
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	√	Undefined
9	R15	O	End address of stack area for the task
10	PR	O	Address of task end processing in the kernel
11	[DSP] DSR	√ * ²	0
12	[DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	√ * ²	Undefined
13	[FPU] FPSCR	√ * ³	inifpscr specified at task creation when the TA_COP1 attribute is specified; otherwise, undefined
14	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0	√ * ³	Undefined
15	[FPU] FPR0_BANK1 to FPR15_BANK1	√ * ⁴	Undefined

Notes: *¹ √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at task initiation when execution is returned from the task function.

*² The register can be used only when the TA_COP0 attribute is specified.

*³ The register can be used only when the TA_COP1 attribute is specified.

*⁴ The register can be used only when the TA_COP1|TA_COP2 attribute is specified.

Table 8.2 SR at Task Initiation

TA_COPn Attribute	Assigned Domain	SR at Initiation (The values of the bits not shown here are undefined.)					
		MD	RB	BL	DSP	FD	IMASK
None	Kernel domain	1	0	0	0	1	0
	User domain	0					
TA_COP0	Kernel domain	1			1		
	User domain	0					
TA_COP1 (TA_COP2)	Kernel domain	1			0	0	
	User domain	0					

8.2 Task Exception Processing Routines

8.2.1 Writing a Task Exception Processing Routine

A task exception processing routine must be written as a C-language function as shown in table 8.3.

```
#include "kernel.h"

void Texrtn(TEXPTN texptn, VP_INT exinf)    ← (1)
{
    /* Processing */
}
```

Figure 8.3 Example of a Task Exception Processing Routine

Description:

- (1) The task exception source pattern is passed through texptn, and the task extended information is passed through exinf.

8.2.2 Rules on Using Registers

Table 8.3 shows the rules on using the registers and their initial values in a task exception processing routine.

Table 8.3 Rules on Using Registers and Initial Register Values in a Task Exception Processing Routine

No.	Register	Use * ¹	Initial Value
1	PC	—	Address of task exception processing routine
2	SR	O	See table 8.4
3	R0 to R3	√	Undefined
4	R4	√	Exception source pattern
5	R5	√	Task extended information
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	√	Undefined
9	R15	O	Stack area for the task
10	PR	O	Address of the end processing for the task exception processing routine in the kernel
11	[DSP] DSR	√ * ²	0
12	[DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	√ * ²	Undefined
13	[FPU] FPSCR	√ * ³	inifpscr specified at creation of the task exception processing routine when the TA_COP1 attribute is specified; otherwise, undefined
14	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0	√ * ³	Undefined
15	[FPU] FPR0_BANK1 to FPR15_BANK1	√ * ⁴	Undefined

Notes: *¹ √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the task exception processing routine function.

*² The register can be used only when the TA_COP0 attribute is specified.

*³ The register can be used only when the TA_COP1 attribute is specified.

*⁴ The register can be used only when the TA_COP1|TA_COP2 attribute is specified.

Table 8.4 SR at Initiation of Task Exception Processing Routine

TA_COPn Attribute	Assigned Domain	SR at Initiation (The values of the bits not shown here are undefined.)					
		MD	RB	BL	DSP	FD	IMASK
None	Kernel domain	1	0	0	0	1	0
	User domain	0					
TA_COP0	Kernel domain	1			1		
	User domain	0					
TA_COP1 (TA_COP2)	Kernel domain	1			0	0	
	User domain	0					

8.3 Extended Service Call Routines and Trap Routines

8.3.1 Writing an Extended Service Call Routine or a Trap Routine

(1) Extended Service Call Routines

An extended service call routine is called through a `cal_svc` service call.

An extended service call routine must be written as a C-language function as shown in figure 8.4.

```
#include "kernel.h"
ER_UINT Svcrtm(VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4) ← (1)
{
    /* Processing */
    return E_OK;
} ← (2)
```

Figure 8.4 Example of an Extended Service Call Routine (1)

Description:

- (1) An extended service call routine receives four VP_INT-type parameters specified through `cal_svc`.
- (2) An ER_UINT-type value is passed as a return value from `cal_svc`.

When only one or two parameters are to be passed, an extended service call routine can be written as shown in figure 8.5.

```
#include "kernel.h"
ER_UINT Svcrtm(VP_INT par1, VP_INT par2) ← (1)
{
    /* Processing */
    return E_OK;
}
```

Figure 8.5 Example of an Extended Service Call Routine (2)

Description:

- (1) An extended service call routine receives only two VW-type parameters, `par1` and `par2`, specified through `cal_svc`.

(2) Trap Routines

A trap routine is called when a TRAPA instruction is executed.

A trap routine must be written as a C-language function as shown in figure 8.6.

```
#include "kernel.h"
void Trprtn(VT_TRAP *pk_trap)      ← (1)
{
    /* Processing */
    return;
}
```

Figure 8.6 Example of a Trap Routine

Description:

(1) pk_trap indicates the address of the packet where the register information is saved when a TRAPA instruction is executed.

Figure 8.7 shows the VT_TRAP definition.

This packet holds the value of each register when a TRAPA instruction is executed.

The kernel restores each register value to the respective value saved in this packet when the execution of a trap routine is completed.

Note that ctxid, ssr, and r15 in this packet must not be modified. If they are modified, correct operation is not guaranteed.

```

typedef struct {
    UW    r0;        /* R0_BANK0 register */
    UW    r1;        /* R1_BANK0 register */
    UW    r2;        /* R2_BANK0 register */
    UW    r3;        /* R3_BANK0 register */
    UW    r4;        /* R4_BANK0 register */
    UW    r5;        /* R5_BANK0 register */
    UW    r6;        /* R6_BANK0 register */
    UW    r7;        /* R7_BANK0 register */
    UW    pr;        /* PR register */
    UW    spc;       /* SPC register */
    UW    ssr;       /* SSR register */
    UW    ctxid;     /* ctxid information (kernel internal information) */
    UW    r15;       /* R15 register */
} VT_REG0;

typedef VT_REG0 VT_TRAP;

```

Figure 8.7 VT_TRAP Type

8.3.2 Rules on Using Registers

Table 8.5 shows the rules on using the registers and their initial values in an extended service call routine or a trap routine. The return value from an extended service call routine is stored in register R0.

Table 8.5 Rules on Using Registers and Initial Register Values in an Extended Service Call Routine or a Trap Routine

No.	Register	Use ^{*1}	Initial Value
1	PC	—	Routine address
2	SR	O	See table 8.6
3	R0 to R3	√	Undefined
4	R4	√	Extended service call routine: par1 Trap routine: pk_trap
5	R5	√	Extended service call routine: par2 Trap routine: Undefined
6	R6	√	Extended service call routine: par3 Trap routine: Undefined
7	R7	√	Extended service call routine: par4 Trap routine: Undefined
8	R8 to R14, MACH, MACL, GBR	O	Undefined
9	R15	O	(1) When called in a task context: Privileged stack area for the calling task (2) When called in a non-task context: Stack used before the routine is called (non-task stack)
10	PR	O	Address of the return processing in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	√ ^{*2}	Undefined
12	[FPU] FPSCR	√ ^{*3}	inifpscr specified at routine definition when the TA_COP1 attribute is specified; otherwise, undefined
13	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0	√ ^{*3}	Undefined
14	[FPU] FPR0_BANK1 to FPR15_BANK1	√ ^{*4}	Undefined

Notes: ^{*1} √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the routine function.

^{*2} The register can be used only when the TA_COP0 attribute is specified.

^{*3} The register can be used only when the TA_COP1 attribute is specified.

^{*4} The register can be used only when the TA_COP1|TA_COP2 attribute is specified.

Table 8.6 SR at Initiation of Extended Service Call Routine or Trap Routine

TA_COPn Attribute	SR at Initiation (The values of the bits not shown here are undefined.)					
	MD	RB	BL	DSP	FD	IMASK
None	1	0	0	0	1	Same as the value before the extended service call or TRAPA instruction
TA_COP0				1	1	
TA_COP1 (TA_COP2)				0	0	

8.4 Interrupt Handlers

8.4.1 Writing an Interrupt Handler

An interrupt handler must be written as a general C-language function as shown in figure 8.8.

```
#include "kernel.h"
void IntHandler(void)
{
    /* Processing */
}
```

Figure 8.8 Example of an Interrupt Handler

8.4.2 Rules on Using Registers

Table 8.7 shows the rules on using the registers and their initial values in an interrupt handler.

Table 8.7 Rules on Using Registers and Initial Register Values in an Interrupt Handler

No.	Register	Use *1	Initial Value
1	PC	—	Interrupt handler address
2	SR	O	See the description below
3	R0 to R3	√	Undefined
4	R4	√	Undefined
5	R5	√	Undefined
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	O	Undefined
9	R15	O	Non-task stack
10	PR	O	Address of the return processing for an interrupt in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		Same as the value before the interrupt
12	[FPU] FPSCR		Same as the value before the interrupt
13	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0		Same as the value before the interrupt
14	[FPU] FPR0_BANK1 to FPR15_BANK1		Same as the value before the interrupt

Notes: *1 √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the handler function.

The register with no indication in the Use column must not be used (accessed).

The bits in SR hold the following values when an interrupt handler is initiated.

- Mode (MD) bit: Always 1.
- Register bank (RB) bit: Always 0.
- Block (BL) bit: Depends on the inhsr value specified at definition of the interrupt handler.
- DSP bit (SH4AL-DSP): 0
- FPU disable (FD) bit (SH-4A): 1
- Interrupt mask level (IMASK) bits: Depend on the inhsr value if the INTMU bit in CPUOPM register is 0 when the kernel is started, or hold the level of the generated interrupt if the INTMU bit is 1.
- Other bits: Undefined

In an interrupt handler, the BL and IMASK bits in SR must be specified so that interrupts with the current interrupt level are not accepted.

8.4.3 DSP and FPU

In an interrupt handler, operation using the DSP or FPU is not allowed. Note, however, that an extended service call routine or a trap routine called in an interrupt handler can use the coprocessor corresponding to the attribute (TA_COP0, TA_COP1, or TA_COP2) specified for the routine.

8.4.4 Notes on NMI

- (1) The BL bit in SR must be set to 1 at initiation of the NMI interrupt handler, which can be specified when the handler is defined, and must not be cleared within the NMI interrupt handler.
- (2) Although the interrupt controller can be used to specify whether to accept an NMI while the BL bit in SR is 1, it must be specified so that no NMI is accepted in usual operation.
If accepting an NMI is specified, the amount of the stack used by the NMI interrupt handler increases. When the memory object protection function is used, do not access MMU mapped areas through the NMI interrupt handler. If a TLB miss occurs during access to an MMU mapped area, the CPU is reset.

Reference: Section 12.4.3, Stack Size Used by NMI Interrupt Handler
 Section 5, Logical Address Space
 Section 4.18.3 (2), Modifying the BL Bit in SR

8.5 Interrupt and Exception Hook Routines

8.5.1 Overview

When a CPU exception or an interrupt occurs, the processor passes control to the following address.

- General exception: VBR + H'100
- TLB miss exception: VBR + H'400
- Interrupt: VBR + H'600

The kernel initializes VBR through `vsta_knl` so that the above addresses point to the respective processing routines inside the kernel. Each routine analyzes the source of the exception or interrupt and initiates the handler corresponding to the source.

However, in some cases, user-specified processing should be performed for debugging before the handler is initiated.

Select `CFG_INTHOOK` in the configurator to call a hook routine when a CPU exception or interrupt occurs.

The symbol names of hook routines are determined according to the exception or interrupt type (shown above) as follows.

- General exception: `__kernel_hook_exp` (name in the assembly-language level)
- TLB miss exception: `__kernel_hook_tlb` (name in the assembly-language level)
- Interrupt: `__kernel_hook_int` (name in the assembly-language level)

Refer to the sample hook routines stored in `samples\sysapp\inthook.src`.

8.5.2 Writing a Hook Routine

A hook routine must be written in the assembly language as shown in figure 8.9.

```
.section  PSCP_hiknl, code, align=4 ; Any section name can be used.

.export  __kernel_hook_exp
__kernel_hook_exp:                ; Symbol for the start address of the hook routine for general exceptions
    ; General exception processing (VBR + H'100) including TRAPA
    ; Write user-code here.

    rts    ; The RTS instruction returns execution to the usual general exception processing in the kernel.
nop
.pool

.export  __kernel_hook_tlb
__kernel_hook_tlb:                ; Symbol for the start address of the hook routine for TLB miss exceptions
    ; TLB miss exception processing (VBR + H'400)
    ; Write user-code here.

    rts    ; The RTS instruction returns execution to the standard TLB exception processing in the
kernel.
nop
.pool

.export  __kernel_hook_int
__kernel_hook_int:                ;Symbol for the start address of the hook routine for interrupts
    ; Interrupt processing (VBR + H'600)
    ; Write user-code here.

    rts    ; The RTS instruction returns execution to the standard interrupt processing in the
kernel.
nop
.pool
.end
```

Figure 8.9 Examples of Hook Routines

8.5.3 Rules on Using Registers

Table 8.8 shows the rules on using the registers and their initial values in a hook routine.

Table 8.8 Rules on Using Registers and Initial Register Values in a Hook Routine

No.	Register	Use *1	Initial Value
1	PC	—	Hook routine address
2	SR	O	Determined by the interrupt or exception processing in the processor. Note especially the following. <ul style="list-style-type: none"> • MD bit: 1 • RB bit: 1 • BL bit: 1 The BL bit must not be changed to 0.
3	R0 to R2_BANK1	✓	Undefined
4	R3 to R7_BANK1	O	Undefined
5	R0 to R7_BANK0	O	Same as the value before the interrupt or exception
6	R8 to R14, MACH, MACL, GBR	O	Same as the value before the interrupt or exception
7	R15	O	Same as the value before the interrupt or exception
8	PR	O	Address of the interrupt or exception processing in the kernel
9	SPC, SSR	O	Value specified by the exception processing in the processor
10	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		Same as the value before the interrupt or exception
11	[FPU] FPSCR		Same as the value before the interrupt or exception
12	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0		Same as the value before the interrupt or exception
13	[FPU] FPR0_BANK1 to FPR15_BANK1		Same as the value before the interrupt or exception

Notes: *1 ✓: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the routine function.

The register with no indication in the Use column must not be used (accessed).

8.5.4 Notes

- (1) Only R0_BANK1 to R2_BANK1 can be used (the contents can be modified) in a hook routine.
- (2) When a hook routine is initiated, both the BL and RB bits in SR are 1; the BL bit must not be changed to 0.
- (3) If a CPU exception occurs while the BL bit in SR is 1, the CPU is reset; do not generate a CPU exception.

When the memory object protection function is selected, exceptions related to the TLB must not be generated for the same reason; that is, a hook routine must not access any MMU mapped area.

- (4) A hook routine is executed with keeping BL = 1 in SR, and so it is not reentered.
- (5) When using a stack, allocate a stack area in an MMU non-mapped area in advance and switch to that area.

8.6 Time Event Handlers

8.6.1 Writing a Time Event Handler

A time event handler must be written as a C-language function. Figure 8.10 shows an example of a cyclic or alarm handler, and figure 8.11 shows an example of an overrun handler.

```
#include "kernel.h"

void Handler(VP_INT exinf)          ← (1)
{
    /* Processing */
}
```

Figure 8.10 Example of a Cyclic or Alarm Handler

Description:

(1) The handler extended information is passed through exinf.

```
#include "kernel.h"

void Ovrhdr(ID tskid, VP_INT exinf) ← (1)
{
    /* Processing */
}
```

Figure 8.11 Example of an Overrun Handler

Description:

(1) The target task ID is passed through tskid, and the extended information for that task is passed through exinf.

8.6.2 Rules on Using Registers

Table 8.9 shows the rules on using the registers and their initial values in a time event handler.

Table 8.9 Rules on Using Registers and Initial Register Values in a Time Event Handler

No.	Register	Use *1	Initial Value
1	PC	—	Handler address
2	SR	O	See the description below
3	R0 to R3	√	Undefined
4	R4	v	Cyclic handler or alarm handler: Handler extended information Overrun handler: Target task ID
5	R5	√	Cyclic handler or alarm handler: Undefined Overrun handler: Extended information for the target task
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	O	Undefined
9	R15	O	Non-task stack
10	PR	O	Address of the return processing in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		Undefined
12	[FPU] FPSCR		Undefined
13	[FPU] FPUL		Undefined
14	[FPU] FPR0_BANK0 to FPR15_BANK0		Undefined
15	[FPU] FPR0_BANK1 to FPR15_BANK1		Undefined

Notes: *1 √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at routine initiation when execution is returned from the handler function.

The register with no indication in the Use column must not be used (accessed).

The bits in SR hold the following values when a time event handler is initiated.

- Mode (MD) bit: Always 1.
- Register bank (RB) bit: Always 0.
- Block (BL) bit: Always 0.
- DSP bit (SH4AL-DSP): Always 0.
- FPU disable (FD) bit (SH-4A): Always 1.
- Interrupt mask level (IMASK) bits: CFG_KNLMSKLV
- Other bits: Undefined

In a time event handler, the BL and IMASK bits in SR must be specified so that interrupts with the level masked at initiation of the handler are not accepted.

8.6.3 DSP and FPU

In a time event handler, operation using the DSP or FPU is not allowed. Note, however, that an extended service call routine or a trap routine called in a time event handler can use the coprocessor corresponding to the attribute (TA_COP0, TA_COP1, or TA_COP2) specified for the routine.

8.7 Initialization Routines

8.7.1 Writing an Initialization Routine

The initialization routines defined through the [Initialization routine] page in the configurator are executed immediately before the multi-tasking environment is entered after the kernel is started.

Reference: Section 6.22.12, Start Kernel (vsta_knl, ivsta_knl)

An initialization routine must be written as a C-language function as shown in figure 8.12.

```
#include "kernel.h"
void InitRoutine(VP_INT exinf)      ← (1)
{
    /* Processing */
}
```

Figure 8.12 Example of an Initialization Routine

Description:

(1) The extended information for the initialization routine is passed through exinf.

8.7.2 Rules on Using Registers

Table 8.10 shows the rules on using the registers and their initial values in an initialization routine.

Table 8.10 Rules on Using Registers and Initial Register Values in an Initialization Routine

No.	Register	Use *1	Initial Value
1	PC	—	Routine address
2	SR	O	See the description below
3	R0 to R3	√	Undefined
4	R4	√	Initialization routine extended information
5	R5	√	Undefined
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	O	Undefined
9	R15	O	Non-task stack
10	PR	O	Address of the return processing in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		Undefined
12	[FPU] FPSCR		Undefined
13	[FPU] FPUL		Undefined
14	[FPU] FPR0_BANK0 to FPR15_BANK0		Undefined
15	[FPU] FPR0_BANK1 to FPR15_BANK1		Undefined

Notes: *1 √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at routine initiation when execution is returned from the routine function.

The register with no indication in the Use column must not be used (accessed).

The bits in SR hold the following values when an initialization routine is initiated.

- Mode (MD) bit: Always 1.
- Register bank (RB) bit: Always 0.
- Block (BL) bit: Always 0.
- DSP bit (SH4AL-DSP): Always 0.
- FPU disable (FD) bit (SH-4A): Always 1.
- Interrupt mask level (IMASK) bits: 15
- Other bits: Undefined

In an initialization routine, the BL and IMASK bits in SR must be specified so that interrupts with the level masked at initiation of the routine are not accepted.

8.7.3 DSP and FPU

In an initialization routine, operation using the DSP or FPU is not allowed. Note, however, that an extended service call routine or a trap routine called in an initialization routine can use the coprocessor corresponding to the attribute (TA_COP0, TA_COP1, or TA_COP2) specified for the routine.

Before a DSP operation is performed, the DSR must be correctly initialized. For appropriate initial values, refer to the hardware manual of the microcomputer being used.

8.8 CPU Exception Handler

Refer to the sample CPU exception handler stored in `samples\sysapp\exchdr.c`.

8.8.1 Writing the CPU Exception Handler

The CPU exception handler must be written as a C-language function as shown in figure 8.13.

```
#include "kernel.h"

void Exchdr(VT_EXC *pk_exc)          ← (1)
{
    /* Processing */
}
```

Figure 8.13 Example of CPU Exception Handler

Description:

(1) `pk_exc` indicates the address of the packet where the register information is saved when a CPU exception occurs.

The `VT_EXC`-type structure consists of `VT_EXCINF`-type, `VT_REG0`-type, and `VT_REG1`-type structures as shown in figure 8.14. Figures 8.15 to 8.17 show the definition of each type.

```
typedef struct {
    VT_EXCINF    vt_excinf;
    VT_REG1      vt_reg1;
    VT_REG0      vt_reg0;
} VT_EXC;
```

Figure 8.14 VT_EXC Type


```

typedef struct {
    UW    syssts;    /* System status (kernel internal information) before the exception occurs */
    ID    tskid;     /* ID of the task executed before the exception.
                     TSK_NONE(0) when no task was executed */
    ID    domid;     /* ID of the domain where the task executed before the exception is assigned.
                     TDOM_NONE(-2) when no task was executed */
    STAT  texstat;   /* Exception processing state of the task with tskid.
                     Invalid (undefined value) when tskid = TSK_NONE(0) */
    UW    expevt;    /* EXPEVT register */
    UW    tra;       /* TRA register */
    UW    tea;       /* TEA register */
} VT_EXCINF;

```

Figure 8.15 VT_EXCINF Type

```

typedef struct {
    UW    r0;        /* R0_BANK0 register */
    UW    r1;        /* R1_BANK0 register */
    UW    r2;        /* R2_BANK0 register */
    UW    r3;        /* R3_BANK0 register */
    UW    r4;        /* R4_BANK0 register */
    UW    r5;        /* R5_BANK0 register */
    UW    r6;        /* R6_BANK0 register */
    UW    r7;        /* R7_BANK0 register */
    UW    pr;        /* PR register */
    UW    spc;        /* SPC register */
    UW    ssr;        /* SSR register */
    UW    ctxid;     /* ctxid information (kernel internal information) */
    UW    r15;       /* R15 register */
} VT_REG0;

```

Figure 8.16 VT_REG0 Type

```
typedef struct {
    UW    r8;        /* R8 register */
    UW    r9;        /* R9 register */
    UW    mach;      /* MACH register */
    UW    r10;       /* R10 register */
    UW    mac1;      /* MACL register */
    UW    r11;       /* R11 register */
    UW    gbr;       /* GBR register */
    UW    r12;       /* R12 register */
    UW    r13;       /* R13 register */
    UW    r14;       /* R14 register */
} VT_REG1;
```

Figure 8.17 VT_REG1 Type

The kernel restores each register value to the respective values saved in `vt_reg0` or `vt_reg1` when the execution of the CPU exception handler is completed. To modify the register contents for exception processing, modify the packet contents. Note, however, that `ssr`, `ctxid`, and `r15` in `vt_reg0` must not be modified. If they are modified, correct operation is not guaranteed.

The FPU register values and DSP register values are not saved in this packet. The kernel neither saves these register values when initiating a handler, nor restores the registers when execution of the handler is completed. To modify the DSP or FPU registers for exception processing, modify the desired registers directly in the handler; note that direct modification is only allowed for a DSP or FPU exception. If the current exception is not related to the DSP or FPU, manipulation of the DSP or FPU registers may not be possible. The program for modifying the registers must be written in the assembly language.

8.8.2 Macros Specialized for CPU Exception Handler

The following C-language macros specialized for the CPU exception handler are provided to refer to various states when a CPU exception occurs. All macros specify `pk_exc`, which is passed to the CPU exception handler as a parameter. The kernel does not detect errors in the parameters specified in these macros.

(1) Referring to the Context at CPU Exception: VSNS_CTX Macro

C-Language API:

```
BOOL state = VSNS_CTX(VT_EXC *pk_exc);
```

Return Parameter:

```
BOOL    state    Context
```

Function:

The return value is determined in the same way as in the `sns_ctx` service call; that is, TRUE is returned when a non-task context is executed when a CPU exception occurs, or FALSE when a task context is executed.

(2) Referring to the ID of the Task in RUNNING State at CPU Exception: VGET_TID Macro

C-Language API:

```
ID tskid = VGET_TID(VT_EXC *pk_exc);
```

Return Parameter:

```
ID      tskid    Task ID
```

Function:

The return value is determined in the same way as in the `iget_tid` service call; that is, the ID of the task in RUNNING state when a CPU exception occurs is returned. To be more specific, when a CPU exception occurs in a task context, the ID of the task is returned. When a CPU exception occurs in a non-task context, the ID of the task in RUNNING state before the non-task context was entered is returned. If there is no task in RUNNING state, `TSK_NONE(0)` is returned.

Note that the ID of the task in RUNNING state before a non-task context was entered is returned when a CPU exception occurs in the non-task context. Use `VSNS_CTX()` to check whether a CPU exception occurred in a task context or a non-task context.

(3) Referring to the ID of the Domain for the Task in RUNNING State at CPU Exception: VGET_DID Macro

C-Language API:

```
ID domid = VGET_DID(VT_EXC *pk_exc);
```

Return Parameter:

ID domid Domain ID

Function:

The return value is determined in the same way as in the `iget_did` service call; that is, the ID of the domain for the task in RUNNING state when a CPU exception occurs is returned. To be more specific, when a CPU exception occurs in a task context, the ID of the domain for the task is returned. When a CPU exception occurs in a non-task context, the ID of the domain for the task in RUNNING state before the non-task context was entered is returned. If there is no task in RUNNING state, `TDOM_NONE(-2)` is returned.

Note that the ID of the domain for the task in RUNNING state before a non-task context was entered is returned when a CPU exception occurs in the non-task context. Use `VSNS_CTX()` to check whether a CPU exception occurred in a task context or a non-task context.

(4) Referring to the CPU-Locked State at CPU Exception: VSNS_LOC Macro

C-Language API:

```
BOOL state = VSNS_LOC(VT_EXC *pk_exc);
```

Return Parameter:

BOOL state CPU-locked state

Function:

The return value is determined in the same way as in the `sns_loc` service call; that is, `TRUE` is returned when a CPU exception occurs in the CPU-locked state, or `FALSE` in the CPU-unlocked state.

(5) Referring to the Dispatch-Disabled State at CPU Exception: VSNS_DSP Macro

C-Language API:

```
BOOL state = VSNS_DSP(VT_EXC *pk_exc);
```

Return Parameter:

BOOL state Dispatch-disabled state

Function:

The return value is determined in the same way as in the sns_dsp service call; that is, TRUE is returned when a CPU exception occurs in the dispatch-disabled state, or FALSE in the dispatch-enabled state.

(6) Referring to the Dispatch-Pended State at CPU Exception: VSNS_DPN Macro

C-Language API:

```
BOOL state = VSNS_DPN(VT_EXC *pk_exc);
```

Return Parameter:

BOOL state Dispatch-pended state

Function:

The return value is determined in the same way as in the sns_dpn service call; that is, TRUE is returned when a CPU exception occurs in the dispatch-pended state; otherwise, FALSE is returned.

**(7) Referring to the Task Exception-Disabled State for the Task in RUNNING State at CPU
Exception: VSNS_TEX Macro**

C-Language API:

```
BOOL state = VSNS_TEX(VT_EXC *pk_exc);
```

Return Parameter:

BOOL state Task exception-disabled state

Function:

The return value is determined in the same way as in the sns_ctx service call; that is, FALSE is returned when the return value from VGET_TID is not TSK_NONE(0) and the task is in the task exception-enabled state; otherwise, TRUE is returned.

8.8.3 Rules on Using Registers

Table 8.11 shows the rules on using the registers and their initial values in the CPU exception handler.

Table 8.11 Rules on Using Registers and Initial Register Values in CPU Exception Handler

No.	Register	Use ^{*1}	Initial Value
1	PC	—	Handler address
2	SR	O	See the description below
3	R0 to R3	√	Undefined
4	R4	√	pk_exc
5	R5	√	Undefined
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	√	Undefined
9	R15	O	Non-task stack
10	PR	O	Address of the return processing for a CPU exception in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	O ^{*2}	Same as the value before the CPU exception
12	[FPU] FPSCR	O ^{*2}	Same as the value before the CPU exception
13	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0	O ^{*2}	Same as the value before the CPU exception
15	[FPU] FPR0_BANK1 to FPR15_BANK1	O ^{*2}	Same as the value before the CPU exception

Notes: ^{*1} √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the handler function.

^{*2} Modify the required register only when the exception cause should be cleared.

The bits in SR hold the following values when the CPU exception handler is initiated.

- Mode (MD) bit: Always 1.
- Register bank (RB) bit: Always 0.
- Block (BL) bit: Depends on the excsr value specified at definition of the CPU exception handler
- Other bit: Same as the value before the CPU exception

In the CPU exception handler, the BL and IMASK bits in SR must be specified so that interrupts with the level masked at initiation of the handler are not accepted.

8.8.4 DSP and FPU

In the CPU exception handler, operation using the DSP or FPU is not allowed except when registers are modified for exception processing.

8.9 Memory Access Violation Handler

8.9.1 Overview

The memory access violation handler is initiated when an illegal memory access is performed for an MMU mapped area while the memory object protection function is used. When the memory object protection function is used, a memory access violation handler must be created and embedded in the kernel.

Refer to the sample memory access violation handler stored in `samples\sysapp\mavhdr.c`.

8.9.2 Writing the Memory Access Violation Handler

The memory access violation handler must be written as a C-language function as shown in figure 8.18.

```
#include "kernel.h"

void _kernel_mavhdr (VT_MAV *pk_mav, VT_EXC *pk_exc)    ← (1)
{
    /* Processing */
}
```

Figure 8.18 Example of Memory Access Violation Handler

Description:

- (1) The name of the memory access violation handler is determined as `"_kernel_mavhdr"`. `pk_mav` indicates the type of access violation, and `pk_exc` indicates the address of the packet where the register information is saved when memory access violation occurs.

For the `VT_EXC` type definition, see figures 8.14 to 8.17.

The kernel restores each register value to the respective values saved in `vt_reg0` or `vt_reg1` when the execution of the memory access violation handler is completed. To modify the register contents for exception processing, modify the packet contents. Note, however, that `ssr`, `ctxid`, and `r15` in `vt_reg0` must not be modified. If they are modified, correct operation is not guaranteed.

The FPU register values and DSP register values are not saved in this packet. The kernel neither saves these register values when initiating a handler, nor restores the registers when execution of the handler is completed. To modify the DSP or FPU registers for exception processing, modify the desired registers directly in the handler; note that direct modification is only allowed for a DSP

or FPU exception. If the current exception is not related to the DSP or FPU, manipulation of the DSP or FPU registers may not be possible. The program for modifying the registers must be written in the assembly language.

Figure 8.19 shows the definition of the VT_MAV type.

```
typedef struct {  
    UW      type;    /* Error type */  
    UW      access;  /* Read or write */  
} VT_MAV;
```

Figure 8.19 VT_MAV Type

Either one of the following values is passed through type.

- E_NOEXS: An address that is not specified as a memory object in an MMU mapped area is accessed or the P3 area is accessed in the privileged mode.
- E_MACV: There is no access permission when an existing memory object is accessed.

Through access, TPM_READ(0) is passed for read access, or TPM_WRITE(1) for write access.

8.9.3 Macros Specialized for CPU Exception Handler

The macros described in section 8.8.2, Macros Specialized for CPU Exception Handler, can also be used in the memory access violation handler.

8.9.4 Rules on Using Registers

Table 8.12 shows the rules on using the registers and their initial values in the memory access violation handler.

Table 8.12 Rules on Using Registers and Initial Register Values in Memory Access Violation Handler

No.	Register	Use ^{*1}	Initial Value
1	PC	—	Handler address
2	SR	O	See the description below
3	R0 to R3	√	Undefined
4	R4	√	pk_mav
5	R5	√	pk_exc
6	R6	√	Undefined
7	R7	√	Undefined
8	R8 to R14, MACH, MACL, GBR	√	Undefined
9	R15	O	Non-task stack
10	PR	O	Address of the return processing for memory access violation in the kernel
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	O ^{*2}	Same as the value before the memory access violation
12	[FPU] FPSCR	O ^{*2}	Same as the value before the memory access violation
13	[FPU] FPUL, FPR0_BANK0 to FPR15_BANK0	O ^{*2}	Same as the value before the memory access violation
15	[FPU] FPR0_BANK1 to FPR15_BANK1	O ^{*2}	Same as the value before the memory access violation

Notes: ^{*1} √: The register can be used without saving or restoring the register value.

O: The register value must be restored to its initial value held at initiation when execution is returned from the handler function.

^{*2} Modify the required register only when the exception source should be clarified.

The bits in SR hold the following values when the memory access violation handler is initiated. Note especially that the handler is initiated with keeping BL = 1. Any service call must not be issued while BL = 1.

- Mode (MD) bit: Always 1.
- Register bank (RB) bit: Always 0.
- Block (BL) bit: Always 0.
- Other bits: Same as the value before the memory access violation

In the memory access violation handler, the BL and IMASK bits in SR must be specified so that interrupts with the level masked at initiation of the handler are not accepted.

8.9.5 DSP and FPU

In the memory access violation handler, operation using the DSP or FPU is not allowed except when registers are modified for exception processing.

8.10 System Down Routine

8.10.1 Overview

The system down routine is called when a `vsys_dwn` service call is issued or an abnormal state is detected in the kernel. Various information regarding the cause of abnormality is passed to the system down routine. A system down routine must always be created and embedded in the kernel.

Refer to the sample system down routine stored in `samples\sysapp\sysdwn.c`.

8.10.2 Writing the System Down Routine

The system down routine must be written as a C-language function as shown in figure 8.20.

```
#include "kernel.h"

void    _kernel_sysdwn ( ER type, VW inf1, VW inf2, VW inf3)
{
    /* Processing */
    while(1);
}
```

Figure 8.20 Example of System Down Routine

The function name of the system down routine is determined as `"_kernel_sysdwn"`.

Execution should not return from the system down routine.

8.10.3 Rules on Using Registers

Since execution does not return from the system down routine, any register can be freely used in the system down routine.

Parameters are passed through the following registers.

- R4: type
- R5: inf1
- R6: inf2
- R7: inf3

For the function of each parameter, refer to the following.

Reference: Section 16.1, Information during System Down

Section 9 Standard Timer Driver

9.1 Overview

If CFG_OPTTMR is not selected in the configurator, a standard timer driver must be created and installed in the kernel.

The HI7300/PX provides a sample of the standard timer driver (samples\shnnnn\kernel\knl_side\tmrdrv.c). Table 9.1 lists the samples of standard timer drivers.

Table 9.1 Samples of Standard Timer Drivers Provided for HI7300/PX V.1.00

shnnnn	Target Microcomputer and Internal Timer Module	
73180	SH73180	TMU CH0
7343	SH7343	TMU CH0
7780	SH7780	TMU CH3
7785	SH7785	TMU CH3

9.2 Configuration of Functions

The standard timer driver is composed of timer initialization and timer interrupt routines. The timer interrupt routine is called from the timer interrupt handler _kernel_isig_tim() of the kernel, and clears interrupt factors. In addition, the timer initialization routine is executed as an initialization routine.

These function names are fixed as follows:

- Timer initialization routine: _kernel_tmrini()
- Timer interrupt routine: _kernel_tmrint()

9.2.1 Timer Initialization Routine (`_kernel_tmrini()`)

In the timer initialization routine, the macros listed in table 9.2, which are output to `kernel_macro.h` by the configurator, are used to initialize the timer.

Table 9.2 Macros Used in Timer Initialization Routine

Macro Name	Item Set by the Configurator	Description
TIC_NUME	CFG_TICNUME in the [Time Management Function] page	Numerator of time tick period [msec] = TIC_NUME/TIC_DENO
TIC_DENO	CFG_TICDENO in the [Time Management Function] page	
VTCFG_TIMINTNO	CFG_TIMINTNO in the [Time Management Function] page	Timer interrupt number
VTCFG_TMRCLOCK	CFG_TMRCLOCK in the [Time Management Function] page	Clock supplied to the timer [Hz]
VTCFG_KNLLVL	CFG_KNLLVL in the [Kernel] page	Kernel interrupt mask level or timer interrupt level

The following operations are performed in the timer initialization routine:

- Definition of the timer interrupt handler
- Initialization of the timer device or interrupt controller

Figure 9.1 shows an example of the timer initialization routine.


```

extern void _kernel_isig_tim(void);
void _kernel_tmrini(void)
{
    /*** Interrupt handler definition packet ***/
    const T_DINH dinh                                     <- (1)
    = { TA_HLNG, &_kernel_isig_tim, (MD_BIT|BL_BIT) | ((VTCFG_KNLLVL)<<4) };

    INT    old_sr;

    /* Define timer interrupt handler */                                     <- (1)
    if((def_inh(VTCFG_TIMINTNO, &dinh)) != E_OK) {
        while(1) {

            }
        }

    /* Save current SR */
    old_sr = get_cr();

    /* Set SR.BL=1 */
    set_cr(BL_BIT | old_sr);

    /* Cancel TMU module-stop */                                     <- (2)
    TIMER_MSTOP_CANCEL();

    /* Initialize INTC for TMU */                                     <- (3)
    TIMER_INTC_SET(VTCFG_KNLLVL);

    /* Initialize TMU.CH0 */                                     <- (4)
    TIMER_INITIALIZE();

    /* Restore SR */
    set_cr(old_sr);
}

```

Figure 9.1 Example of Timer Initialization Routine

(1) The timer interrupt handler must be defined as follows:

- Interrupt number: VTCFG_TIMINTNO
- Attribute of the handler: TA_HLNG
- Handler address: ‘_kernel_isig_tim’ (internal kernel module)
- SR at activation: (MD_BIT | BL_BIT) | ((VTCFG_KNLLVL) <<4)

(2) The module stop of the timer device is canceled. The TIMER_MSTOP_CANCEL() has been defined by mstop_tmu.h.

(3) The interrupt controller is set so that the timer interrupt level will be VTCFG_KNLLVL. The TIMER_INTC_SET() is defined by intc_tmu.h.

(4) The timer device is initialized so that the condition of the input clock to the timer will be VTCFG_TMRCLOCK[Hz] and the interrupt cycle will be TIC_NUME/TIC_DENO[msec]. The TIMER_INITIALIZE() is defined with tmu.h.

9.2.2 Timer Interrupt Routine (_kernel_tmrint())

In the timer interrupt routine, interrupt factors are cleared.

```
void _kernel_tmrint(void)

{

    TIMER_INTERRUPT();                                     <- (1)

}
```

Figure 9.2 Example of Timer Interrupt Routine

(1) The interrupt factor is cleared. The TIMER_INTERRUPT() is defined with tmu.h.

Section 10 Configurator

10.1 Overview

The configurator is a tool used for setting the operating parameters of the kernel. The configurator creates the following files according to the settings. These files created by the configurator are totally referred to as "configuration files".

- ID name header file

The name specified for each object by the user is defined as the ID number in the configurator. This file is included from an application.

- System definition files

The system configuration information is output to these files. These files are included from two files, `kernel_def.c` and `kernel_cfg.c`. These two files are contained in the `system\` directory. `kernel_def.c` is linked with the kernel libraries. In other words, these files are used to extract the necessary modules from the kernel libraries.

The configurator settings can be saved in a file whose extension is `hcf`. This file is called the "configurator setting file" or "HCF file".

For the role played by the configurator in system configuration, refer to the following.

Reference: Section 11.1, Load Module Types

10.2 Linkage Unit, Kernel Lock Mode, and [Kernel Side]

All setting items of the configurator are classified into the "kernel side" and "kernel environment side".

The "kernel side" stands for information that will be included in a kernel load module (knl_side), and the "kernel environment side" stands for information that will be included in a kernel environment load module (env_side).

To prevent update of a kernel load module, set "kernel lock mode" for the configurator. In kernel lock mode, changing information on the "kernel side" is limited and no configuration file is output on the kernel side.

To enter kernel lock mode, select [Generate -> Kernel Lock Mode] from the menu bar.

In the case of creating objects, such as tasks, which item is on the kernel side can be set and confirmed as shown below.

- The dialog box for creating an object has a [Kernel Side] check box. If this check box is selected, that object is on the kernel side.
- The objects marked by a flag icon in the list box showing a list of objects are on the kernel side.

For other setting items, refer to the subsequent sections.

In addition, refer to the following.

Reference: Section 11.1, Load Module Types

When specifying a C-language symbol or section name of an application, the symbol entity must be included in the suitable linkage unit (load module on the kernel side or kernel environment side).

10.3 Configuration Files Output from Configurator

Configuration files consist of header files for the application and system definition files. System definition files are included only from the following two files which fetch the configuration result.

- kernel_def.c: Kernel side
- kernel_cfg.c: Kernel environment side

kernel_def.c and kernel_cfg.c are stored in the system\ directory.

Table 10.1 lists the configuration files. In kernel lock mode, files on the kernel side are not output.

Table 10.1 Configuration Files Output from Configurator

Classification	File Name	Linkage Unit	Included from kernel_def.c	Included from kernel_cfg.c
Header files for application	kernel_macro.h	Kernel side	O (included from kernel.h)	
	kernel_id_sys.h	Kernel side	O	O
	kernel_id.h	Kernel environment side	—	O
System definition files	kernel_def_main.h	Kernel side	O	O
	kernel_def_import.h	Kernel side	O	O
	kernel_def_inireg.h	Kernel side	O	—
	kernel_def_inirtn.h	Kernel side	O	—
	kernel_def_attmem.h	Kernel side	O	—
	kernel_cfg_main.h	Kernel environment side	—	O
	kernel_cfg_import.h	Kernel environment side	—	O
	kernel_cfg_inireg.h	Kernel environment side	—	O
	kernel_cfg_inirtn.h	Kernel environment side	—	O
	kernel_cfg_attmem.h	Kernel environment side	—	O

10.3.1 Header Files for Application

(1) kernel_macro.h (kernel side)

This file is included from the header, kernel.h.

(2) kernel_id_sys.h (kernel side), kernel_id.h (kernel environment side)

Definition of the ID name specified when creating various objects in the configurator is output to these files in the following form:

```
#define ID_TASK_A 1 /* ID value of ID name "ID_TASK_A" is 1 */
```

Including these files from an application as required allows the ID name to be used as an ID value of an object.

If [Kernel Side] is selected at object creation, the ID name is output to kernel_id_sys.h, whereas if not selected, it is output to kernel_id.h. Note that if [Kernel Side] is not selected, the configurator can assign ID numbers automatically.

The domain name is always output to kernel_id_sys.h.

kernel_id.h is a file on the kernel environment side so it should not be included in an application that will be linked to the kernel side.

10.3.2 System Definition Files

(1) kernel_def_main.h (kernel side)

Information, such as the result of service call selection, which is used for selecting the necessary function modules from the kernel libraries is output.

(2) kernel_cfg_main.h (kernel environment side)

Information, such as the number of tasks and the resource pool size, which is related to the system size is output.

(3) kernel_def_import.h (kernel side), kernel_cfg_import.h (kernel environment side)

When a C-language symbol is specified in the configurator, the external reference statement for that symbol is output to these files. The symbols on the kernel side are output to kernel_def_import.h, while the symbols on the kernel environment side are output to kernel_cfg_import.h.

(4) kernel_def_inireg.h (kernel side), kernel_cfg_inireg.h (kernel environment side)

The "initial registration routine" program for creating and defining various objects set in the configurator is output. Objects on the kernel side are created and defined in kernel_def_inireg.h, while objects on the kernel environment side are created and defined in kernel_cfg_inireg.h.

In addition, refer to the following.

Reference: • Section 6.22.12, Start Kernel (vsta_knl, ivsta_knl)
• Section 16.2.2, When Object Specified in Configurator Cannot be Created

(5) kernel_def_inirtn.h (kernel side), kernel_cfg_inirtn.h (kernel environment side)

The initialization routine information is output. The initialization routine information on the kernel side is output to kernel_def_inirtn.h, while the initialization routine information on the kernel environment side is output to kernel_cfg_inirtn.h.

(6) kernel_def_attmem.h (kernel side), kernel_cfg_attmem.h (kernel environment side)

The static memory object information is output. The static memory object information on the kernel side is output to kernel_def_attmem.h, while the static memory object information on the kernel environment side is output to kernel_cfg_attmem.h.

10.4 User Interface

10.4.1 Screen Configuration

Figure 10.1 shows the screen configuration of the configurator.

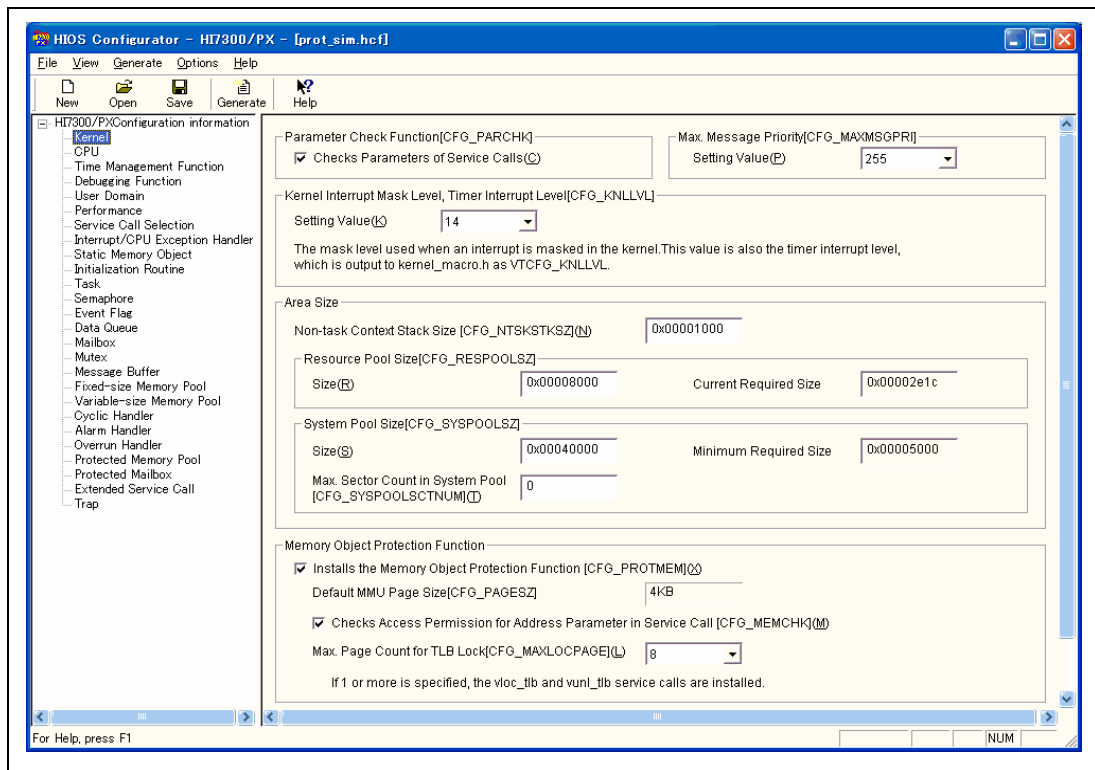


Figure 10.1 Screen Configuration of Configurator




10.4.2 Title Bar

The title bar at the top of the window displays the application name or document name.



The title bar includes the following elements:



- (1) Control menu button of the application
- (2) Application name (HIOS Configurator – HI7300/PX –)
- (3) HCF file name
- (4) <Minimize> button 
- (5) <Maximize>/<Restore Down> button 
- (6) <Close> button 

10.4.3 Menu Bar:[File] Menu

All information set by the user can be saved in the HCF files. The previously set contents can be recovered by reading an HCF file.

The [File] menu is for creating, opening, or saving an HCF file.

The following commands are available.

- [New]
- [Open...]
- [Save]
- [Save As...]
- [Exit]

Up to four recently used HCF files can be displayed.

(1) [New] command ([File] menu)

Creates a new configurator setting file (untitled.hcf) and opens it.

Shortcut:

Tool bar:



Keyboard: CTRL + N

(2) [Open...] command ([File] menu)

Opens an existing HCF file.

Shortcut:

Tool bar:



Keyboard: CTRL + O

(3) [Save] command ([File] menu)

Saves the currently edited HCF file without changing the file name and storage location. The first time a new configurator setting file (untitled.hcf) is saved, the [Save As] dialog box is displayed so that the file can be saved with a suitable file name. To save a file with a different file name or storage location, use the [Save As...] command.

Shortcut:

Tool bar:



Keyboard: CTRL + S

(4) [Save As...] command ([File] menu)

Saves the currently edited contents in a new HCF file.

(5) [Exit] command ([File] menu)

Terminates the configurator. If the setting contents are not yet saved, a dialog box confirming whether the contents should be saved or not is displayed.

10.4.4 Menu Bar:[View] Menu

The [View] menu is for enabling or disabling display of the toolbar and status bar.

The following commands are available.

- [Toolbar]
- [Status Bar]

(1) [Toolbar] command ([View] menu)

Enables or disables display of the toolbar. The toolbar includes tools which have the same functions as the most frequently used commands, such as [Open...]. When the toolbar is displayed, a check mark is put next to this command name in the [View] menu.

(2) [Status Bar] command ([View] menu)

Enables or disables display of the status bar. The status bar displays a simple description of a command if a menu command or toolbar button is selected, and also the ON/OFF state of the special keys on the keyboard. When the status bar is displayed, a check mark is put next to this command name in the [View] menu.

10.4.5 Menu Bar:[Generate] Menu

The [Generate] menu has the following commands.

- [Kernel Lock Mode]
- [Generate Configuration Files]

(1) [Kernel Lock Mode] command ([Generate] menu)

Sets or cancels kernel lock mode.

In kernel lock mode, "Kernel Lock" is displayed on the status bar.

Whether or not the configurator was in kernel lock mode is also saved in the HCF file.

Reference: 10.2 Linkage Unit, Kernel Lock Mode, and [Kernel Side]

(2) [Generate Configuration Files] command ([Generate] menu)

Generates the configuration files.

Selecting this command opens the [Generation of Configuration Files] dialog box in which the file generation location is specified. After specifying the file generation location, click the [Generate] button to generate the configuration files at the specified location.

Shortcut:

Tool bar:



Keyboard: CTRL + G

10.4.6 Menu Bar:[Options] Menu

The [Options] menu has the following command.

- [Open the file used last time]

(1) [Open the file used last time] command ([Options] menu)

The file used last time is opened automatically when this command is selected.

10.4.7 Menu Bar:[Help] Menu

The [Help] menu has the following commands.

- [Help Topics]
- [About HIOS Configurator...]

(1) [Help Topics] command ([Help] menu)

Opens the help files.

(2) [About HIOS Configurator...] command ([Help] menu)

Displays the version number and copyright of the configurator.






10.4.8 Toolbar

The toolbar is displayed immediately under the menu bar at the top of the application window. In the toolbar, frequently used functions are registered as buttons.



To enable or disable display of the toolbar, select the [Toolbar] command in the [View] menu.

Each button is related with a command as shown below.

 New	[New] command
 Open	[Open...] command
 Save	[Save] command
 Generate	[Generate Configuration Files] command
 Help	Help Initiation of context help mode

10.4.9 Status Bar

The status bar is displayed at the bottom of the main window. To enable or disable display of the status bar, select the [Status Bar] command in the [View] menu.

On the left side of the status bar, a brief description of a menu command is displayed when selected. Similarly, when the cursor is placed on a toolbar button, a brief description is displayed. To halt execution of a toolbar command after reading its description, move the mouse pointer to a different location than that toolbar button and release the mouse button.

On the right side of the status bar, the following status is displayed.

- Kernel Lock: Kernel lock mode
- CAP: [Caps Lock] key is ON
- NUM: [Num Lock] key is ON
- SCRL: [Scroll Lock] key is ON

10.4.10 [Navigation] Window

The [Navigation] window displays the page that will be shown in the [Information Input] window. Selecting an item in the list using the mouse or keyboard displays the page corresponding to the selected item in the [Information Input] window.

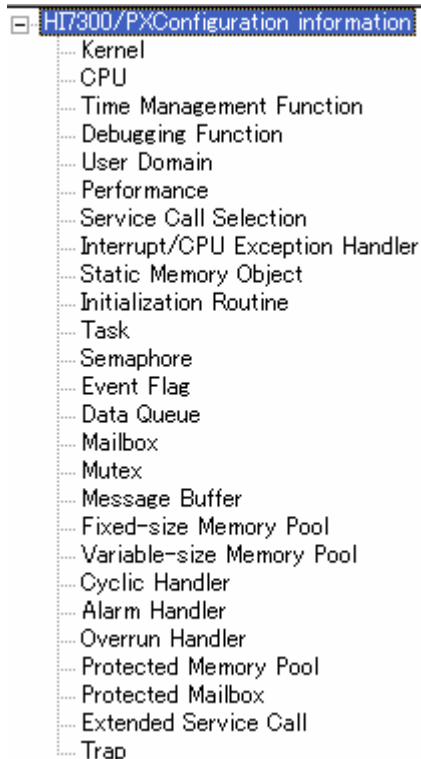


Figure 10.2 Navigation Window

10.4.11 [Information Input] Window

The [Information Input] window is for entering configuration information. This window displays the page selected in the [Navigation] window.

For details of each page, refer to the subsequent sections.

10.5 Page Configuration

Table 10.2 lists the pages.

Table 10.2 List of Pages

Page	Setting Items
[Kernel] page	Common items of the kernel
[CPU] page	Information related to the microcomputer used
[Time Management Function] page	Items related to the time management function
[Debugging Function] page	Items related to the debugging function (Debugging Extension)
[User Domain] page	ID names of user domains
[Performance] page	Items related to the performance management function using the program performance counter (PPC) in the CPU
[Service Call Selection] page	Service calls to be installed
[Interrupt/CPU Exception Handler] page	Items related to an interrupt or CPU exception
[Static Memory Object] page	Registration of a static memory object
[Initialization Routine] page	Registration of an initialization routine
[Task] page	Items related to a task
[Semaphore] page	Items related to a semaphore
[Event Flag] page	Items related to an event flag
[Data Queue] page	Items related to a data queue
[Mailbox] page	Items related to a mailbox
[Mutex] page	Items related to a mutex
[Message Buffer] page	Items related to a message buffer
[Fixed-size Memory Pool] page	Items related to a fixed-size memory pool
[Variable-size Memory Pool] page	Items related to a variable-size memory pool
[Cyclic Handler] page	Items related to a cyclic handler
[Alarm Handler] page	Items related to an alarm handler
[Overrun Handler] page	Definition of the overrun handler
[Protected Memory Pool] page	Items related to a protected memory pool
[Protected Mailbox] page	Items related to a protected mailbox
[Extended Service Call] page	Items related to an extended service call
[Trap] page	Items related to a trap

10.6 CFG Name

Most of the items set in the configurator affect kernel operation. Such setting items have a "CFG name" that begins with "CFG_". This name is not only displayed on the configurator screen but is also used in this manual. However, not all setting items are given names.

For example, CFG_SYSPoolsSZ in the [Kernel] page stands for the system pool size.

10.7 Specifications for Pages and Dialog Boxes

10.7.1 [Kernel] Page

In this page, set the common items of the kernel.

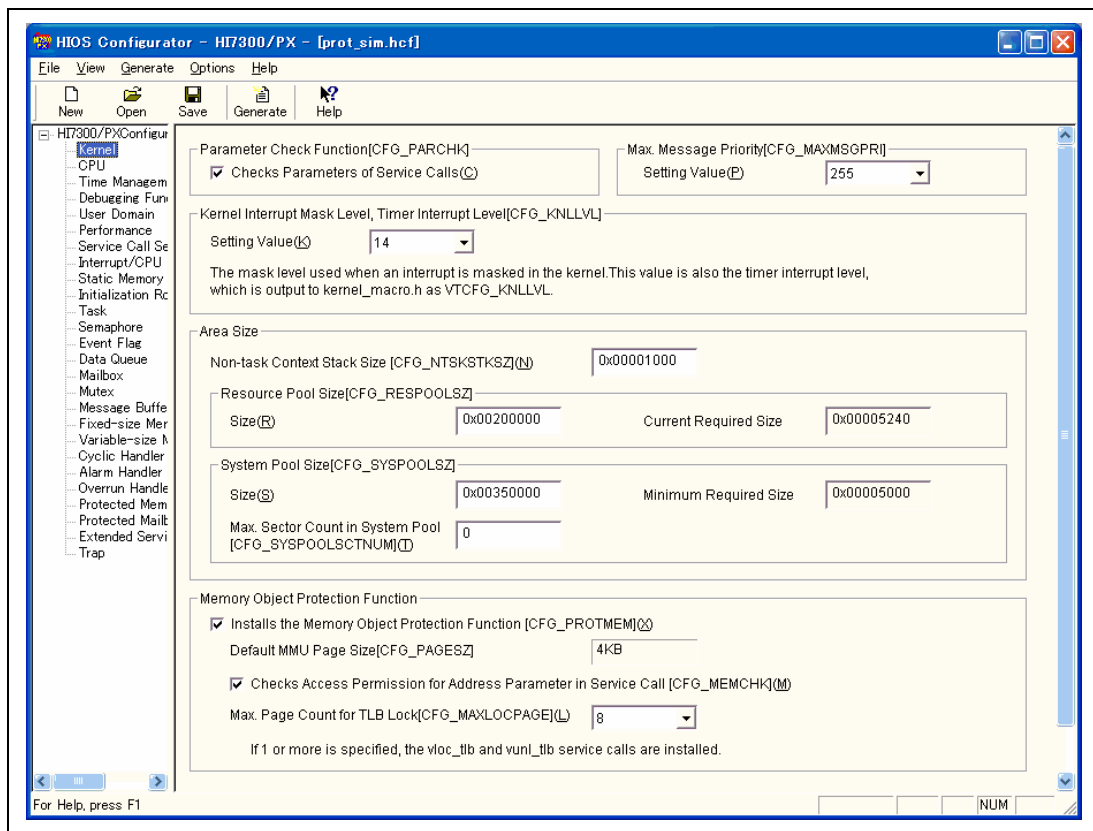


Figure 10.3 [Kernel] Page

Table 10.3 lists the [Kernel] page items.

Table 10.3 [Kernel] Page Items

Item	CFG Name	Linkage Unit
Parameter Check Function	CFG_PARCHK	Kernel side
Max. Message Priority	CFG_MAXMSGPRI	Kernel side
Kernel Interrupt Mask Level, Timer Interrupt Level	CFG_KNLLVL	Kernel side
Non-task Context Stack Size	CFG_NTSKSTKSZ	Kernel environment side
Resource Pool Size	CFG_RESPOOLSZ	Kernel environment side
System Pool Size	CFG_SYSPOLSZ	Kernel environment side
Max. Sector Count in System Pool	CFG_SYSPOLoSCTNUM	Kernel environment side
Installs the Memory Object Protection Function	CFG_PROTMEM	Kernel side
Default MMU Page Size	CFG_PAGESZ	Kernel side
Checks Access Permission for Address Parameter in Service Call	CFG_MEMCHK	Kernel side
Max. Page Count for TLB Lock	CFG_MAXLOCPAGE	Kernel side

(1) [Parameter Check Function [CFG_PARCHK]]

If this check box is selected, the parameter errors in a service call are checked.

If this check box is not selected, the selection of CFG_MEMCHK described later is automatically canceled.

Reference: Section 6.3.2, Parameter Check Function

(2) [Max. Message Priority [CFG_MAXMSGPRI]]

Select the maximum value of the message priority used in the mailboxes and protected mailboxes from 1 to 255. When both mailboxes and protected mailboxes are not used, any value can be selected because this item has no meaning.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TMAX_MPRI 255 /* Example when 255 is selected */
```

(3) [Kernel Interrupt Mask Level, Timer Interrupt Level [CFG_KNLLVL]]

In the case of executing a critical section, the kernel sets the I bit in the SR register to CFG_KNLLVL. An interrupt whose interrupt level is higher than this value is accepted without delay even while the kernel is executing a critical section, but a service call cannot be issued from that handler. A value from 1 to 15 can be selected.

In addition, CFG_KNLLVL is also the timer interrupt level.

The following statement is output to kernel_macro.h in response to this setting.

```
#define VTKNL_LVL 15 /* Example when 15 is selected */
```

When CFG_OPTTMR in the [Time Management Function] page is not selected, in other words, when the standard timer driver is used, the timer interrupt level should be initialized to VTKNL_LVL in the initialization routine of the standard timer driver.

When the optimized timer driver is used, the kernel initializes the timer interrupt level to CFG_KNLLVL.

Reference: Creating standard timer driver → Section 9, Standard Timer Driver

(4) [Area Size] group

(a) [Non-task Context Stack Size [CFG_NTSKSTKSZ]]

Specify the stack size for the non-task context in bytes. An integer between 256 and 0x20000000 can be specified. The specified value is rounded up to a multiple of four.

Reference: Section 12.4, Calculation of Non-task Context Stack Size

(b) [Resource Pool Size [CFG_RESPOOLSZ]]

Specify the resource pool size in bytes. An integer between 256 and 0x20000000 can be specified. The specified value is rounded up to a multiple of four.

[Current Required Size] shows the required size for the objects that use the resource pool and were created in the configurator. If a value smaller than this size is specified, the following error message is displayed.

```
Resource pool size must be at least [Current Required Size]
```

However, this message may not be displayed even when the size is insufficient because the resource pool is consumed even during system operation. A sufficient size must be specified taking into consideration the cases where the resource pool is consumed.

Reference: • Section 4.31, Controlling Memory Fragmentation
• Section 13, Estimation of Resource Pool Size

(c) [System Pool Size [CFG_SYSPoolsSZ]]

Specify the system pool size in bytes. An integer between 0 and 0x20000000 can be specified. The specified value is rounded up to a multiple of 64 when CFG_PROTMEM is not selected and to CFG_PAGESZ (= 4 kbytes) when CFG_PROTMEM is selected.

[Minimum Required Size] shows the minimum required size for the objects that use the system pool and were created in the configurator. If a value smaller than this size is specified, the following error message is displayed.

System pool size must be at least [Minimum Required Size]

The value shown in [Minimum Required Size] is the size when sector management is not performed (CFG_SYSPoolSCTNUM is 0) for the system pool. If a value other than 0 is set to CFG_SYSPoolSCTNUM, the system pool may be insufficient even though this error message is not displayed. In such a case, the specified object cannot be created when the kernel is initiated and system initiation fails.

Reference: Section 16.2.2, When Object Specified in Configurator Cannot be Created

When dynamically creating an object that uses the system pool in a service call, a sufficient size must be specified taking into consideration the cases where the system pool is consumed.

Reference: • Section 4.31, Controlling Memory Fragmentation
• Section 14, Estimation of System Pool Size

(d)[Max. Sector Count in System Pool [CFG_SYSPoolSCTNUM]]

Specify the maximum number of sectors in the system pool. If 0 is specified, sector management is not performed.

If a value greater than the value calculated from the relevant equation below is specified, the actual maximum number of sectors will be corrected to the value calculated from the relevant equation by the kernel.

- When CFG_PROTMEM is selected: $\text{CFG_SYSPoolSZ} / (4096 \times 32)$
- When CFG_PROTMEM is not selected: $\text{CFG_SYSPoolSZ} / (64 \times 32)$

(5) [Memory Object Protection Function] group

In this group, settings related to the memory object protection function are made.

Reference: Section 4.21, Memory Object Protection Function

(a) [Installs the Memory Object Protection Function [CFG_PROTMEM]]

Select this check box when installing the memory object protection function.

The following statement is output to kernel_macro.h in response to this setting.

```
#define VTCFG_PROTMEM 1 /* 1 for installation, 0 for no installation */
```

(b) [Default MMU Page Size [CFG_PAGESZ]]

The default page size is the MMU page size used by memory objects other than static memory objects. The size is fixed at 4 kbytes (4096) and cannot be modified. For static memory objects, a page size other than 4 kbytes can be selected independently.

The following statement is output to kernel_macro.h in response to this setting irrespective of CFG_PROTMEM. However, this definition does not have any meaning unless CFG_PROTMEM is selected.

```
#define VTCFG_PAGESZ 4096
```

(c) [Checks Access Permission for Address Parameter in Service Call [CFG_MEMCHK]]

When this check box is selected, whether access permission for the address parameter in a service call is appropriate or not is checked. However, since this check has a large overhead, as long as debugging has been performed sufficiently and this check will be redundant, this error check can be skipped by canceling the selection of CFG_MEMCHK.

Note that when this check box is selected, CFG_PARCHK is also automatically selected.

Reference: Section 6.3.3, Access Permission Check Function for Address Parameters

(d) [Max. Page Count for TLB Lock [CFG_MAXLOCPAGE]]

Specify the maximum number of pages that can be TLB locked simultaneously from 0 to 32.

If a value other than 0 is selected, the vloc_tlb and vunl_tlb service calls are installed. However, these service calls cannot be selected in the [Service Call Selection] page.

10.7.2 [CPU] Page

In this page, set information related to the microcomputer used.

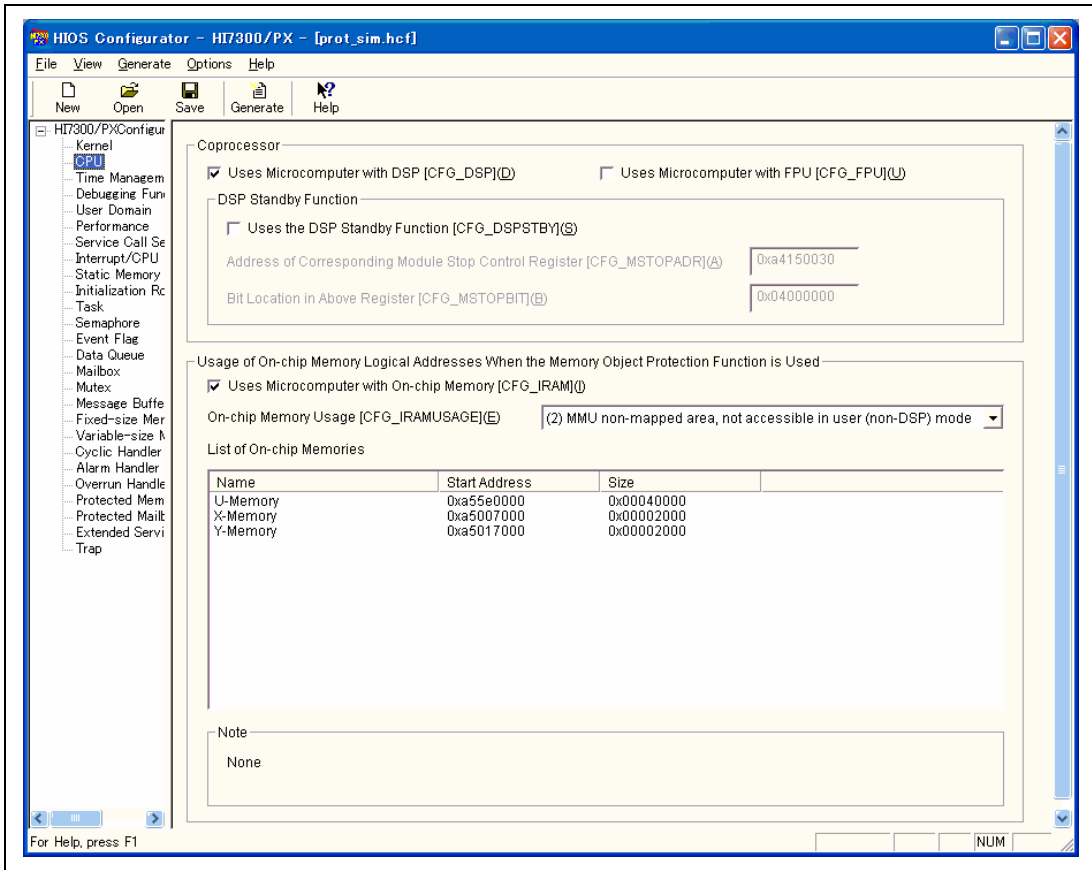


Figure 10.4 [CPU] Page

Table 10.4 lists the [CPU] page items.

Table 10.4 [CPU] Page Items

Item	CFG Name	Linkage Unit
Uses Microcomputer with DSP	CFG_DSP	Kernel side
Uses Microcomputer with FPU	CFG_FPU	Kernel side
Uses the DSP Standby Function	CFG_DSPSTBY	Kernel side
Address of Corresponding Module Stop Control Register	CFG_MSTOPADR	Kernel side
Bit Location in Above Register	CFG_MSTOPBIT	Kernel side
Uses Microcomputer with On-chip Memory	CFG_IRAM	Kernel side
On-chip Memory Usage	CFG_IRAMUSAGE	Kernel side
List of On-chip Memories	—	Kernel side

(1) [Coprocessor] group

(a) [Uses Microcomputer with DSP [CFG_DSP]] or [Uses Microcomputer with FPU [CFG_FPU]]

Select CFG_DSP when a microcomputer with DSP mounted is used, and select CFG_FPU when a microcomputer with FPU mounted is used. Note not to select CFG_DSP and CFG_FPU at the same time.

The TA_COP0 attribute for tasks or handlers can be used only when CFG_DSP is selected. Similarly, the TA_COP1 and TA_COP2 attributes can be used only when CFG_FPU is selected.

(b) [Uses the DSP Standby Function [CFG_DSPSTBY]]

The DSP standby function is used to reduce power consumption by halting clock supply (module stop) to the X/Y memory when executing a task whose attribute is not TA_COP0.

Select this check box when using the DSP standby function. When selected, the vchg_cop service call is installed. Note that the vchg_cop service call cannot be selected in the [Service Call Selection] page.

However, this item cannot be modified and has no meaning unless CFG_DSP is selected.

Reference: DSP standby function → Section 4.25, DSP Standby Control

(c) [Address of Corresponding Module Stop Control Register [CFG_MSTOPADR]] and [Bit Location in Above Register [CFG_MSTOPBIT]]

Specify the address and bit location of the module stop control register that is controlled by the DSP standby function, with reference to the hardware manual of the microcomputer used.

The value that can be specified in [Address] ranges from 0xa0000000 to 0xbfffffff or 0xe0000000 to 0xffffffff and must also be a multiple of four.

In [Bit Location], specify a bit pattern in which the bit that corresponds to the specified module is 1. Normally, only the X/Y memory should be specified.

Note that when CFG_DSP or CFG_DSPSTBY is not selected, these items cannot be modified and have no meaning.

(2) [Usage of On-chip Memory Logical Addresses When the Memory Object Protection Function is Used] group

This group is valid only when the memory object protection function is used. If CFG_PROTMEM is not selected in the [Kernel] page, the items in this group cannot be modified and have no meaning.

Reference: Section 5.3.3, On-Chip Memory

(a) [Uses Microcomputer with On-chip Memory [CFG_IRAM]]

Select this check box when a microcomputer with on-chip memory is used.

If this check box is not selected, the subsequent items cannot be modified and have no meaning.

(b) [On-chip Memory Usage [CFG_IRAMUSAGE]]

Select from the following how the logical addresses of the on-chip memory allocated to the P2 or P4 area are used.

- (1) MMU non-mapped area, accessible in user mode
- (2) MMU non-mapped area, not accessible in user (non-DSP) mode
- (3) MMU mapped area

The kernel initializes the RP and RMD bits in RAMCR according to this setting when the vsta_knl service call is issued.

However, if CFG_IRAM is not selected, the kernel does not initialize RAMCR.

(c) [List of On-chip Memories]

The logical addresses of the specified on-chip memories are handled as specified in CFG_IRAMUSAGE.

The following items are in the pop-up menu.

- [Define]: Opens the [Definition of On-chip Memory] dialog box to add an on-chip memory definition
- [Delete]: Deletes the selected on-chip memory definition
- [Modify]: Opens the [Modification of Information for On-chip Memory Definition] dialog box to modify the selected on-chip memory definition

(d) [Note]

All setting items in this group become invalid when CFG_PROTMEM is not selected in the [Kernel] page. In addition, all items in this group cannot be modified.

In this case, the message shown in table 10.5 is displayed in [Note].

Table 10.5 [Note] in [Usage of On-chip Memory Logical Addresses When the Memory Object Protection Function is Used] Group

Condition	Display Message
CFG_PROTMEM is not selected	All setting items in this group are invalid because CFG_PROTMEM is not selected.

10.7.3 [Definition of On-chip Memory] Dialog Box and [Modification of Information for On-chip Memory Definition] Dialog Box

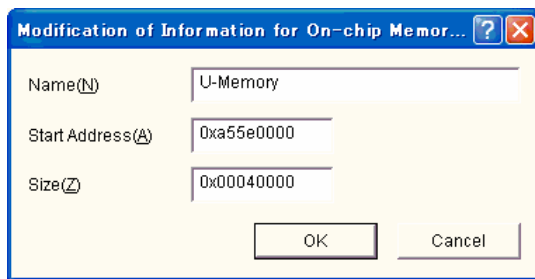


Figure 10.5 [Definition of On-chip Memory] Dialog Box

Selecting [Define] from the pop-up menu in the [CPU] page opens the [Definition of On-chip Memory] dialog box. Selecting [Modify] from the pop-up menu in the [CPU] page opens the [Modification of Information for On-chip Memory Definition] dialog box. These two dialog boxes have the same configuration.

The same information as the on-chip memory mounted on the microcomputer used must be defined here.

(1) [Name]

Specify the name of the on-chip memory. The only purpose of this name is for the user to distinguish on-chip memories in the [CPU] page. The configurator will not check whether this input is correct.

(2) [Start Address]

Specify the logical address of the on-chip memory as a numeric value. The specified address must be in the P2 or P4 area. The specified value is rounded down at the 4-kbyte boundary.

Make sure the range from [Start Address] to [Size] does not overlap with other on-chip memory ranges.

(3) [Size]

Specify the on-chip memory size as a numeric value. The specifiable size is from 4 kbytes to 2 Mbytes and must also be a multiple of 4 kbytes.

(4) [Define] button (only in [Definition of On-chip Memory] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next on-chip memory can be defined without break. This dialog box is not closed.

(5) [OK] button (only in [Modification of Information for On-chip Memory Definition] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(6) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.4 [Time Management Function] Page

In this page, set items related to the time management function.

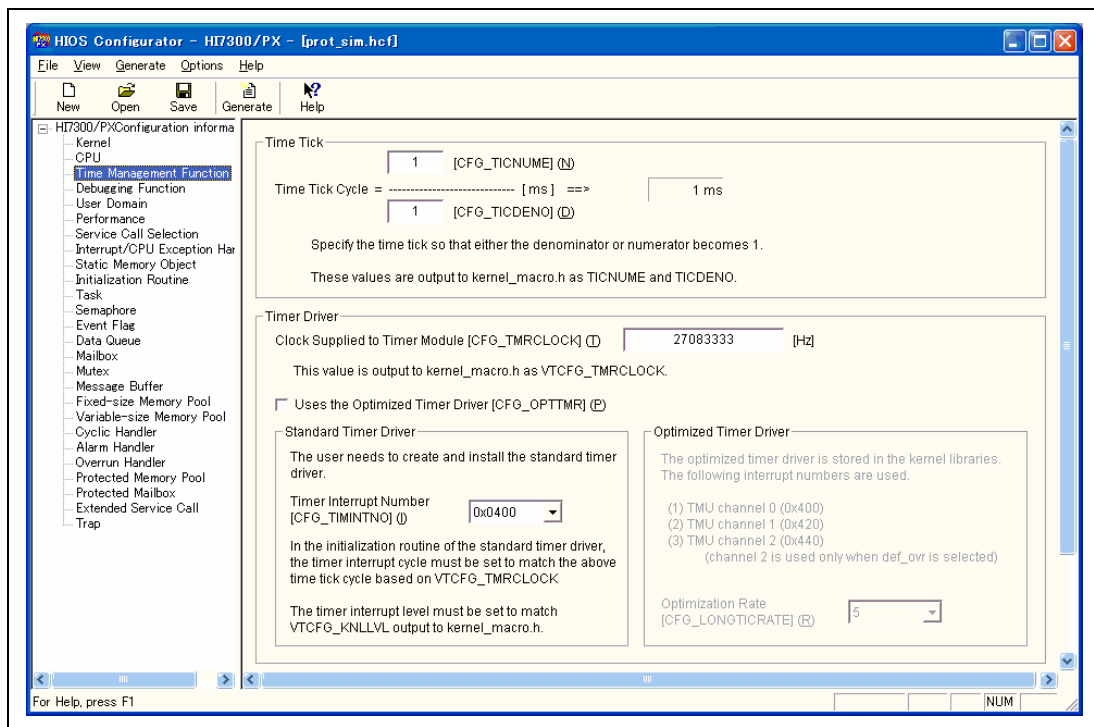


Figure 10.6 [Time Management Function] Page

Table 10.6 lists the [Time Management Function] page items.

Table 10.6 [Time Management Function] Page Items

Item	CFG Name	Linkage Unit
Time Tick Cycle Numerator	CFG_TICNUM	Kernel side
Time Tick Cycle Denominator	CFG_TICDENO	Kernel side
Clock Supplied to Timer Module	CFG_TMRCLOCK	Kernel side
Uses the Optimized Timer Driver	CFG_OPTTMR	Kernel side
Timer Interrupt Number	CFG_TIMINTNO	Kernel side
Optimization Rate	CFG_LONGTICRATE	Kernel side

(1) [Time Tick]

The time tick cycle is set to CFG_TICNUM/CFG_TICDENO [ms]. An integer between 1 and 65535 can be specified for CFG_TICNUM, and an integer between 1 and 100 can be specified for CFG_TICDENO. However, either CFG_TICNUM or CFG_TICDENO has to be 1.

The time tick cycle represents the time precision for task timeout or cyclic handlers. A small time tick cycle will increase the precision of time management by the kernel. However, timer interrupts will occur more frequently and the overhead increased.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TIC_NUME 1 /* When 1 is set to CFG_TICNUME */  
#define TIC_DENO 1 /* When 1 is set to CFG_TICDENO */
```

(2) [Timer Driver] group

(a) [Clock Supplied to Timer Module [CFG_TMRCLOCK]]

Specify the clock frequency supplied to the timer module used in Hz units. An integer between 1 and 0x40000000 (approximately 1 GHz) can be specified.

Normally, specify the clock frequency supplied to the on-chip TMU.

The following statement is output to kernel_macro.h in response to this setting.

```
#define VTCFG_TMRCLOCK 10000000 /* When 10000000 (10 MHz) is specified */
```

When CFG_OPTTMR is not selected, in other words, when the standard timer driver is used, the timer interrupt cycle should be initialized to match TICNUME/TICDENO [ms] in the initialization routine of the standard timer driver.

(b) [Uses the Optimized Timer Driver [CFG_OPTTMR]]

Select this check box when using the optimized timer driver provided by the kernel as the timer driver. Cancel the selection when using the standard timer driver.

(c) [Timer Interrupt Number of Standard Timer Driver [CFG_TIMINTNO]]

Select the interrupt number (INTEVT code) used in the standard timer driver.

The following statement is output to kernel_macro.h in response to this setting.

```
#define VTCFG_TIMINTNO 0x400 /* When 0x400 is selected */
```

In the initialization routine of the standard timer driver, the timer interrupt handler must be defined for this number using the idf_inh service call.

However, this item cannot be modified and has no meaning when CFG_OPTTMR is selected.

(d) [Optimization Rate [CFG_LONGTICRATE]]

Select the optimization rate of the optimized timer driver from 2 to 255.

However, this item cannot be modified and has no meaning unless CFG_OPTTMR is selected.

Reference: • Creating standard timer driver → Section 9, Standard Timer Driver
• Optimized timer driver → Section 4.17, Optimized Timer Driver

10.7.5 [Debugging Function] Page

In this page, set items related to the debugging function (Debugging Extension).

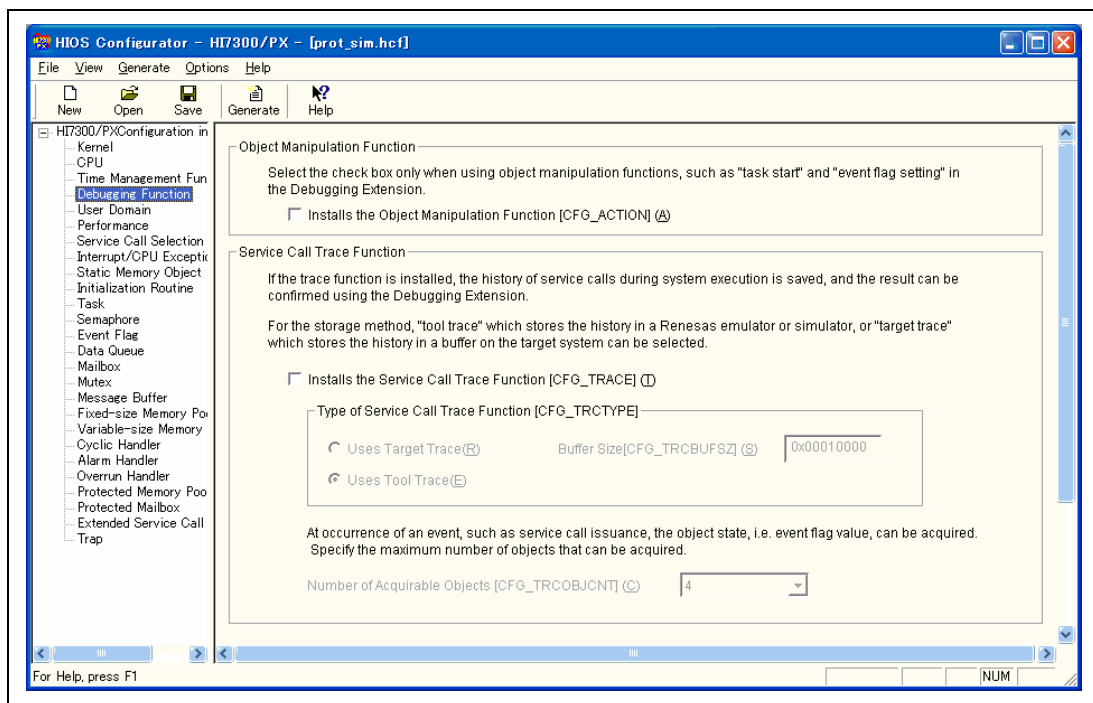


Figure 10.7 [Debugging Function] Page

Table 10.7 lists the [Debugging Function] page items.

Table 10.7 [Debugging Function] Page Items

Item	CFG Name	Linkage Unit
Installs the Object Manipulation Function	CFG_ACTION	Kernel side
Installs the Service Call Trace Function	CFG_TRACE	Kernel side
Type of Service Call Trace Function	CFG_TRCTYPE	Kernel side
Buffer Size	CFG_TRCBUFSZ	Kernel environment side
Number of Acquirable Objects	CFG_TRCOBJCNT	Kernel environment side

(1) [Object Manipulation Function] group

When CFG_ACTION is selected, functions using service calls, such as "task start" and "event flag setting", can be used in the Debugging Extension.

(2) [Service Call Trace Function] group

In this group, settings related to the service call trace function are made. In addition, refer to the following.

Reference: Section 4.27, Service Call Trace

(a) [Installs the Service Call Trace Function [CFG_TRACE]]

When this check box is selected, the service call trace function is installed and also the vget_trc service call is installed. Note that the vget_trc service call cannot be selected in the [Service Call Selection] page.

The trace result can be referenced by the Debugging Extension.

If this check box is not selected, the subsequent items cannot be modified and have no meaning.

(b) [Type of Service Call Trace Function [CFG_TRCTYPE]]

Select either "target trace" or "tool trace" as the type of the service call trace function.

For target trace, a trace buffer to store the trace information must be prepared in the target system.

(c) [Buffer Size [CFG_TRCBUFSZ]]

Specify the buffer size that will be used for target trace. The specified value is rounded up to a multiple of four. An integer between 512 and 0x20000000 can be specified.

However, this item cannot be modified and has no meaning when "tool trace" is selected in CFG_TRCTYPE.

(d) [Number of Acquirable Objects [CFG_TRCOBJCNT]]

The object state at the moment of trace acquisition can be acquired. The kind of object state to be acquired can be set in the Debugging Extension. Here, select the number (maximum value) of objects that can be acquired simultaneously as [No Acquisition] or from 1 to 32.

10.7.6 [User Domain] Page

In this page, set the ID names of user domains.

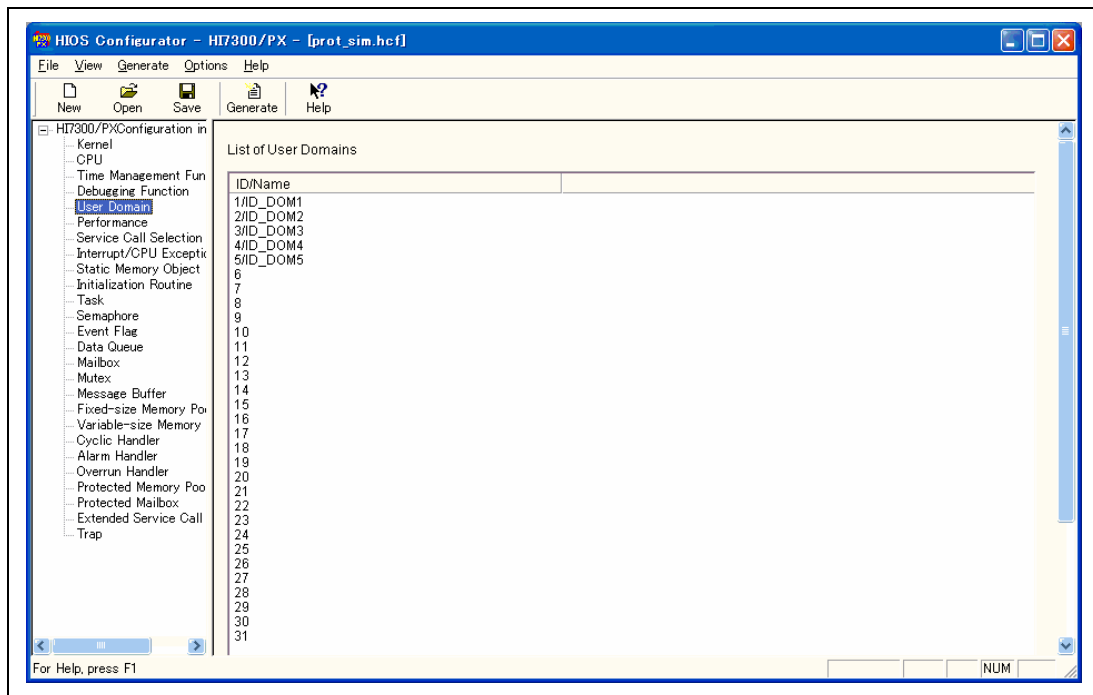


Figure 10.8 [User Domain] Page

According to the specifications, this kernel always has user domains with domain IDs from 1 to 31.

In this page, the ID names for these user domains are set. The ID name of a user domain is always on the kernel side and output to kernel_id_sys.h.

The following items are in the pop-up menu.

- [Edit]: Opens the [Setting of User Domain ID] dialog box to input an ID name for the selected domain ID

- [Delete]: Deletes the ID name of the selected domain ID

10.7.7 [Setting of User Domain ID] Dialog Box

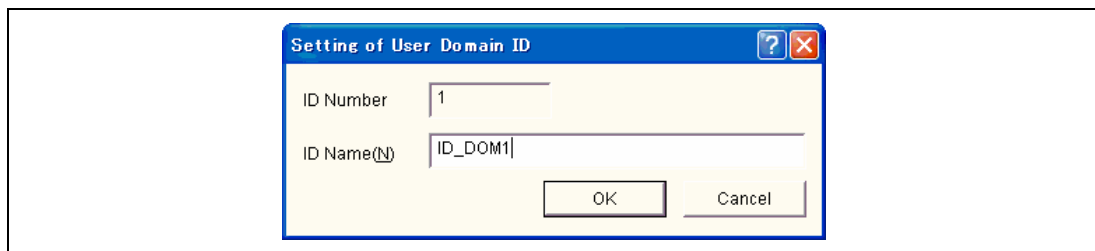


Figure 10.9 [Setting of User Domain ID] Dialog Box

Selecting [Edit] from the pop-up menu in the [User Domain] page opens this dialog box.

[ID Number] displays the domain ID number selected in the [User Domain] page.

Input the ID name to be given in [ID Name].

Clicking the [OK] button closes this dialog box after making the settings in this dialog box effective. Clicking the [Cancel] button closes this dialog box without saving the settings in this dialog box.

10.7.8 [Performance] Page

In this page, set items related to the performance management function using the program performance counter (PPC) in the CPU.

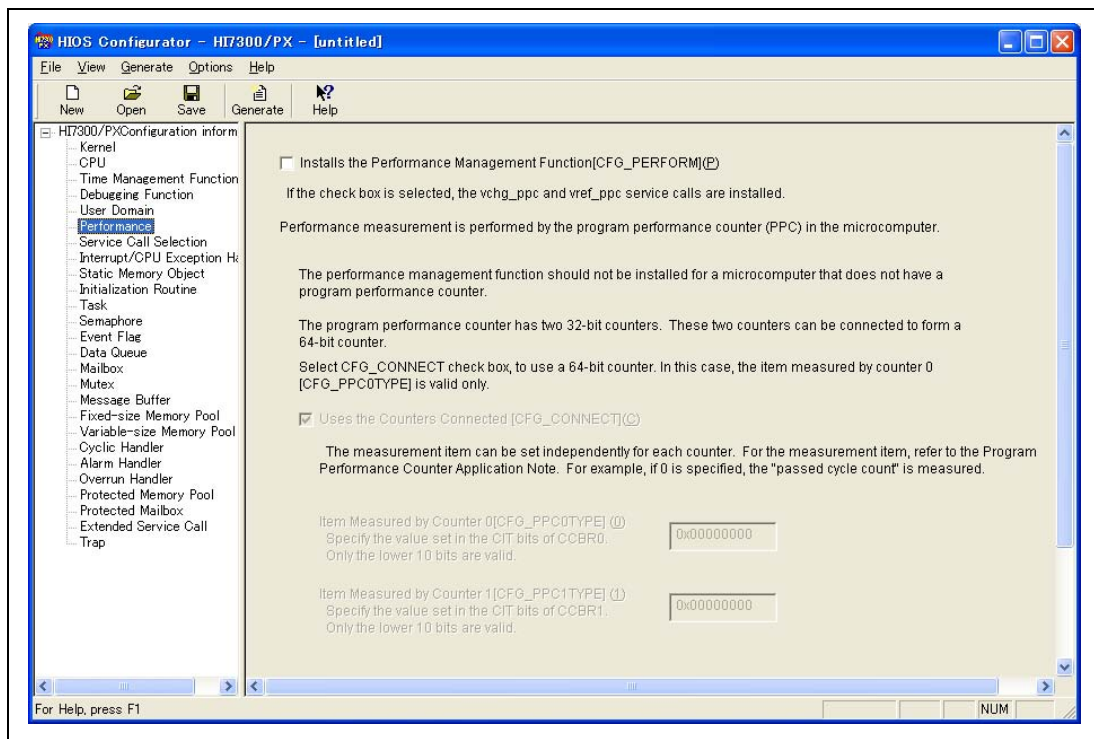


Figure 10.10 [Performance] Page

Knowledge on a program performance counter is required to use the performance management function. Refer to the SH-4A, SH4AL-DSP Program Performance Counter Application Note.

Reference: Section 4.26, Performance Management

Table 10.8 lists the [Performance] page items.

Table 10.8 [Performance] Page Items

Item	CFG Name	Linkage Unit
Installs the Performance Management Function	CFG_PERFORM	Kernel side
Uses the Counters Connected	CFG_CONNECT	Kernel side
Item Measured by Counter 0	CFG_PPC0TYPE	Kernel side
Item Measured by Counter 1	CFG_PPC1TYPE	Kernel side

(1) [Installs the Performance Management Function [CFG_PERFORM]]

Select this check box when using the performance management function. When selected, the `vchg_ppc` and `vref_ppc` service calls are installed. Note that these service calls cannot be selected in the [Service Call Selection] page.

If the microcomputer used does not include a program performance counter, be sure to cancel the selection.

If this check box is not selected, the subsequent items cannot be modified and have no meaning.

(2) [Uses the Counters Connected [CFG_CONNECT]]

Select this check box when the two program performance counters are used connected.

Each program performance counter is 32 bits. For example, in the case of measuring the "passed cycle count", a 32-bit counter overflows at approximately 10 s for a CPU with an internal operating frequency of 400 MHz. Since a correct result cannot be obtained at an overflow, use the counters connected in such cases.

(3) [Item Measured by Counter 0 [CFG_PPC0TYPE]], [Item Measured by Counter 1 [CFG_PPC1TYPE]]

Specify the items to be measured by counters 0 and 1, respectively. The value to be specified is the CIT bit value in CCBR0 or CCBR1 of the PPC, which is assigned from bit 0.

For example, if 0 is specified, the passed cycle count (number of CPU cycles) is measured. For details, refer to the SH-4A, SH4AL-DSP Program Performance Counter Application Note.

10.7.9 [Service Call Selection] Page

In this page, select the service calls that will be installed.

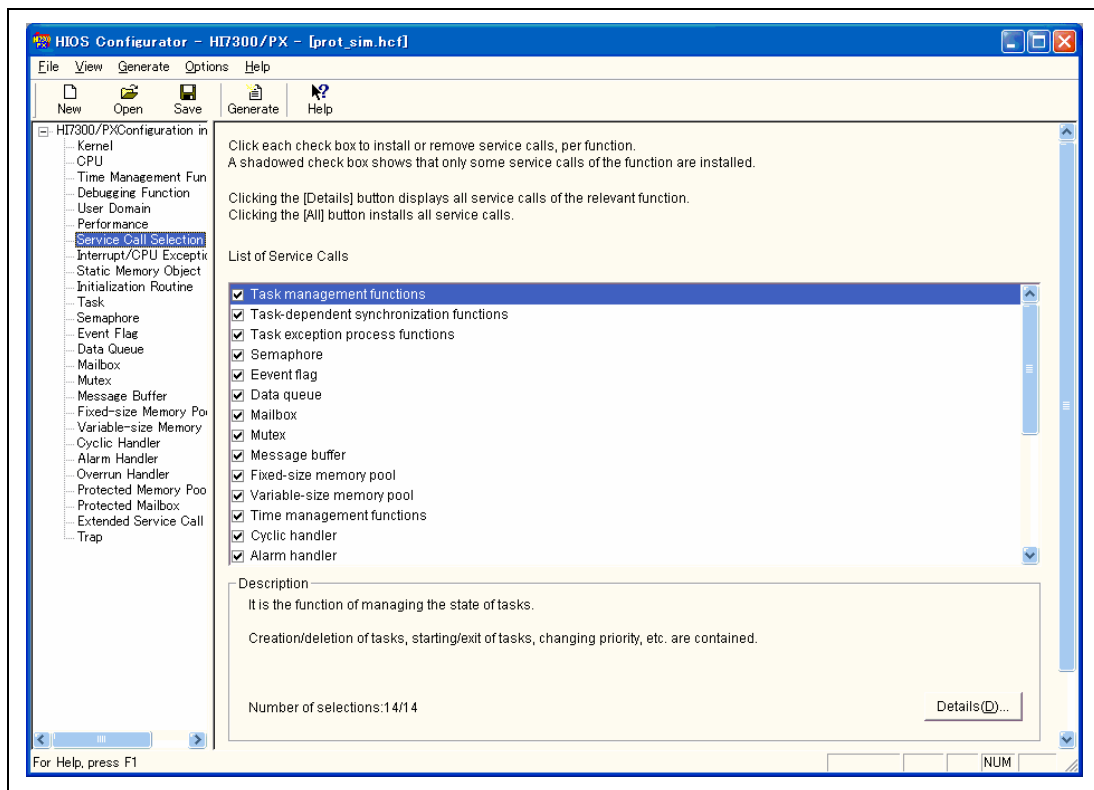


Figure 10.11 [Service Call Selection] Page

In [List of Service Calls], a check box is assigned to each function group, such as "Task management functions" and "Synchronization and communication functions (semaphore)".

After selecting a function group, double-clicking it or clicking the [Details] button opens a dialog box to individually select the service calls belonging to that function group.

Clicking the [All] button selects all service calls.

Some service calls have two types of names; one prefixed with "i" and the other without "i" at the beginning, though they both have the same function, i.e. set_flg and iset_flg. These service calls are collectively selected as "set_flg" in this page.

Note that all setting items in this page are on the kernel side.

(1) Service calls that cannot be selected

The service calls listed in table 10.9 cannot be selected in this page. Though they cannot be selected, whether they will be installed or not can be confirmed in this page.

Table 10.9 Unselectable Service Calls

Function Group	Service Call	Condition for Installation
Task management	cre_tsk	Always installed
	ext_tsk	Always installed
Fixed-size memory pool management	icra_mpf	When CFG_PROTMEM is selected in the [Kernel] page, and at the same time cre_mpf is selected
Variable-size memory pool management	ivcra_mpl	When CFG_PROTMEM is selected in the [Kernel] page, and at the same time cre_mpl is selected
System state management	vsta_knl	Always installed
	vsys_dwn	Always installed
	vget_trc	When CFG_TRACE is selected in the [Debugging Function] page
	ivbgn_int, ivend_int	Only the APIs of these service calls are provided for merely the compatibility with the HI7000/4 series, and their entities do not exist
	vchg_cop	When CFG_DSP and CFG_DSPSTBY are both selected in the [CPU] page
Interrupt management	def_inh	Always installed
System configuration management	def_exc	Always installed
Performance management	vchg_ppc, vref_ppc	When CFG_PERFORM is selected in the [Performance] page
Memory object management	vloc_tlb, vunl_tlb	When CFG_PROTMEM is selected in the [Kernel] page, and at the same time CFG_MAXLOCPAGE is set to a value other than 0 in the [Kernel] page

(2) Service calls required for object creation

The objects shown in table 10.10 cannot be used unless its creation or definition service call is selected. To be specific, creation or definition of the object in each page of the configurator will be ignored. To use these objects, be sure to select its creation or definition service call.

Table 10.10 Creation or Definition Service Call Required for Each Object

Object	Creation or Definition Service Call	Object	Creation or Definition Service Call
Task exception processing routine	def_tex	Variable-size memory pool	cre_mpl
Semaphore	cre_sem	Cyclic handler	cre_cyc
Event flag	cre_flg	Alarm handler	cre_alm
Data queue	cre_dtq	Overrun handler	def_ovr
Mailbox	cre_mbx	Extended service call routine	def_svc
Mutex	cre_mtx	Trap routine	vdef_trp
Message buffer	cre_mbf	Protected memory pool	icre_mpp
Fixed-size memory pool	cre_mpf	Protected mailbox	cre_mbp

In addition, if a creation or definition service call shown in table 10.10 is not selected, all service calls in the same function group cannot be installed regardless of the settings of this page.

When selection of a creation or definition service call shown in table 10.10 is attempted to be canceled, the warning dialog box shown in figure 10.12 is displayed.

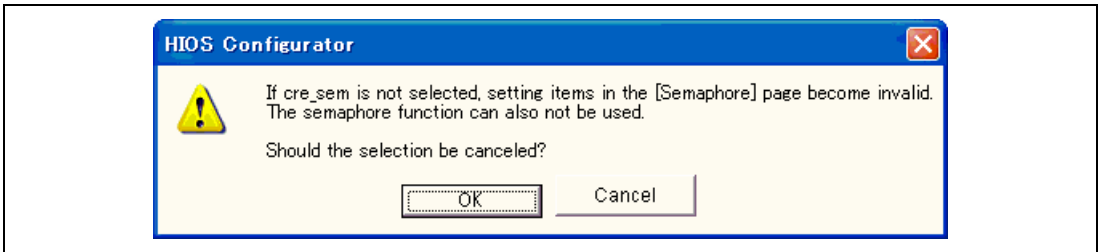


Figure 10.12 Warning Dialog Box in [Service Call Selection] Page

(3) Service calls related to memory object protection function

If CFG_PROTMEM is not selected in the [Kernel] page, the service calls belonging to the function groups below cannot be installed regardless of the settings of this page.

- Memory object management functions
- Protected memory pool management functions
- Protected mailbox management functions

10.7.10 [Interrupt/CPU Exception Handler] Page

In this page, set items related to an interrupt or CPU exception.

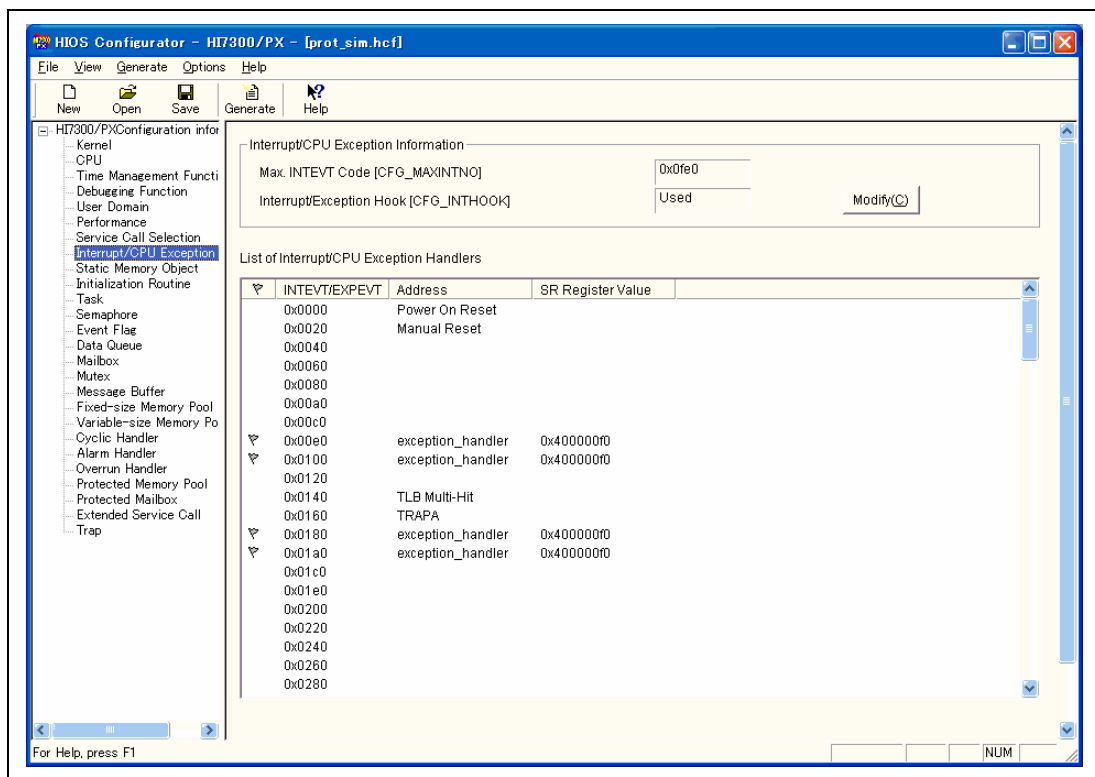


Figure 10.13 [Interrupt/CPU Exception Handler] Page

Table 10.11 lists the [Interrupt/CPU Exception Handler] page items.

Table 10.11 [Interrupt/CPU Exception Handler] Page Items

Item	CFG Name	Linkage Unit
Max. INTEVT Code	CFG_MAXINTNO	Kernel environment side
Interrupt/Exception Hook	CFG_INTHOOK	Kernel side
Definition of Interrupt/CPU Exception Handler	—	Kernel side/kernel environment side

(1) [Interrupt/CPU Exception Information] group

[Max. INTEVT Code] displays the maximum INTEVT code among the interrupts used in the target system.

[Interrupt/Exception Hook] displays whether the interrupt/exception hook is used or not.

Clicking the [Modify] button opens the [Modification of Interrupt/CPU Exception Information] dialog box in which these settings can be changed.

(2) [List of Interrupt/CPU Exception Handlers] group

The status of the definitions of the handlers corresponding to all INTEVT/EXPEVT codes from 0 to CFG_MAXINTNO is displayed here. The flag icon indicates that the handler is defined to belong to the kernel side.

The following items are in the pop-up menu. In kernel lock mode, these pop-up menu items cannot be selected for the handlers on the kernel side.

- [Define]: Opens the [Definition of Interrupt/CPU Exception Handler] dialog box to define a handler for the selected INTEVT/EXPEVT code
- [Cancel]: Cancels the handler definition for the selected INTEVT/EXPEVT code

(3) Special INTEVT/EXPEVT

The INTEVT/EXPEVT codes shown in table 10.12 cannot be defined or canceled in some cases.

Table 10.12 INTEVT/EXPEVT for which [Define] or [Cancel] is Disabled

INTEVT/EXPEVT Code	Source	Display in List	Condition Disabling [Define] or [Cancel]
0	Power-on reset	Power On Reset	Always disabled
0x20	Manual reset	Manual Reset	Always disabled
0x140	TLB multiple hit	TLB Multi-Hit	Always disabled
0x160	Unconditional trap	TRAPA	Always disabled
Value selected in CFG_TIMINTNO in the [Time Management Function] page	—	Timer (Standard)	When CFG_OPTTMR is not selected in the [Time Management Function] page
0x400	TMU channel 0	Timer CH0 (Optimized)	When CFG_OPTTMR is selected in the [Time Management Function] page
0x420	TMU channel 1	Timer CH1 (Optimized)	When CFG_OPTTMR is selected in the [Time Management Function] page
0x440	TMU channel 2	Timer CH2 (Optimized)	When CFG_OPTTMR is selected in the [Time Management Function] page, and at the same time def_ovr is selected in the [Service Call Selection] page

10.7.11 [Modification of Interrupt/CPU Exception Information] Dialog Box

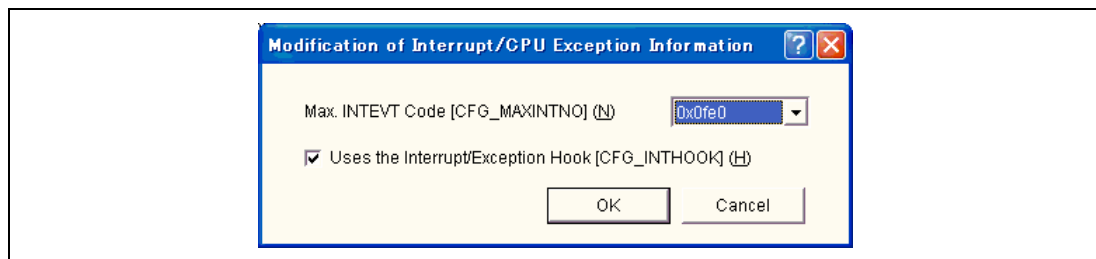


Figure 10.14 [Modification of Interrupt/CPU Exception Information] Dialog Box

Clicking the [Modify] button in the [Interrupt/CPU Exception Handler] page opens this dialog box.

(1) [Max. INTEVT Code [CFG_MAXINTNO]]

Select the maximum INTEVT code among the interrupts used in the target system. The selectable code is a multiple of 0x20, and the minimum value is 0x400 and the maximum value 0x3fe0. However, a value smaller than the following cannot be selected.

- CFG_TIMINTNO in the [Time Management Function] page (CFG_OPTTMR is not selected)
- 0x420 (def_ovr is not selected when CFG_OPTTMR is selected)
- 0x440 (def_ovr is selected when CFG_OPTTMR is selected)
- INTEVT code of a handler already defined

(2) [Uses the Interrupt/Exception Hook [CFG_INTHOOK]]

Select this check box when using the interrupt/exception hook function.

Reference: Section 8.5, Interrupt and Exception Hook Routines

(3) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(4) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.12 [Definition of Interrupt/CPU Exception Handler] Dialog Box

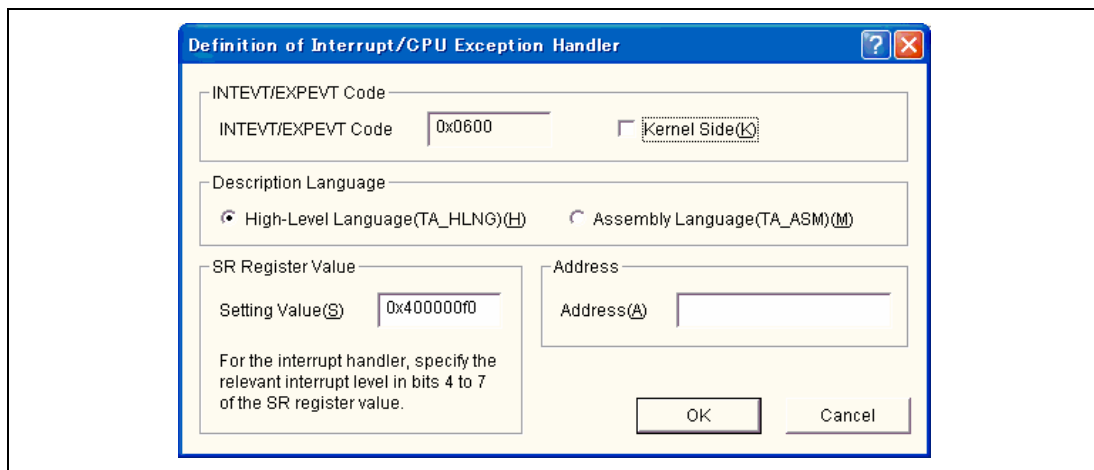


Figure 10.15 [Definition of Interrupt/CPU Exception Handler] Dialog Box

Selecting [Define] from the pop-up menu in the [Interrupt/CPU Exception Handler] page opens this dialog box. These handlers can also be dynamically defined using the `def_inh` or `def_exc` service call.

(1) [INTEVT/EXPEVT Code]

Displays the INTEVT or EXPEVT code selected in the [Interrupt/CPU Exception Handler] page.

(2) [Kernel Side]

Select this check box when specifying the handler defined to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(3) [Description Language]

Select [High-Level Language (TA_HLNG)] when the handler is written in a high-level language, and select [Assembly Language (TA_ASM)] when the handler is written in assembly language.

(4) [SR Register Value]

Enter the SR register value at handler initiation as a numeric value with reference to the following.

Reference: • SR at interrupt handler initiation → Section 8.4.2, Rules on Using Registers
• SR at CPU exception handler initiation → Section 8.8.3, Rules on Using Registers

(5) [Address]

Specify the start address of the handler as a C-language symbol or numeric value.

(6) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(7) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.13 [Static Memory Object] Page

In this page, register a static memory object.

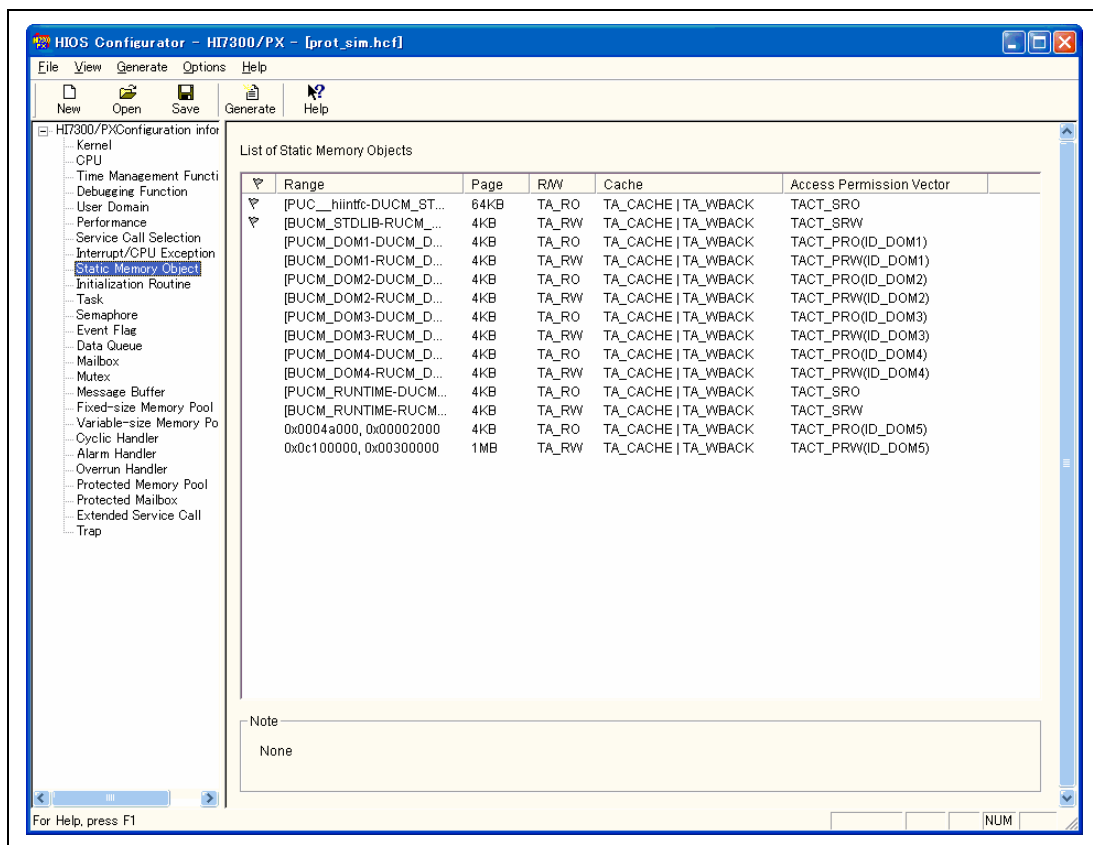


Figure 10.16 [Static Memory Object] Page

(1) [List of Static Memory Objects]

The static memory objects already registered are displayed in the list. The flag icon indicates that the static memory object is registered to belong to the kernel side.

The following items are in the pop-up menu. In kernel lock mode, [Delete] and [Modify] cannot be selected for the static memory objects on the kernel side.

- [Register]: Opens the [Registration of Static Memory Object] dialog box to register a static memory object
- [Delete]: Deletes the selected static memory object
- [Modify]: Opens the [Modification of Information for Static Memory Object Registration] dialog box to modify the setting of the selected static memory object

(2) [Note]

All setting items in this page become invalid when CFG_PROTMEM is not selected in the [Kernel] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.13 is displayed in [Note].

Table 10.13 [Note] in [Static Memory Object] Page

Condition	Display Message
CFG_PROTMEM is not selected	All setting items in this page are invalid because CFG_PROTMEM is not selected.

10.7.14 [Registration of Static Memory Object] Dialog Box and [Modification of Information for Static Memory Object Registration] Dialog Box

Modification of Information for Static Memory Object Registration

Memory Object Area

☒ Kernel Side(K)

☐ Specifies the Address(B)

Address(A)

Size(Z)

☒ Specifies the Section Range(E)

Start Section Name(G) BUCM_STDLIB

End Section Name(L) RUCM_STDLIB

Page Size

Page Size(P) 4KB

Read/Write

☐ Read-only(TA_RO)(Q)

☒ Readable/Writable(TA_RW)(W)

Cache Setting

☒ Cacheable in write-back mode(TA_CACHE|TA_WBACK)(C)

☐ Cacheable in write-through mode(TA_CACHE|TA_WTHROUGH)(T)

☐ Non-cacheable(TA_UNCACHE)(N)

Access Permission Vector

Setting

Access Permission Vector(V) (4) TACT_SRW

Specified User Domain ID(D) 1/ID_DOM1

Setting Result

User Domain Possible to Write All user domain

User Domain Possible to Read All user domain

OK Cancel

Figure 10.17 [Registration of Static Memory Object] Dialog Box

Selecting [Register] from the pop-up menu in the [Static Memory Object] page opens the [Registration of Static Memory Object] dialog box. Selecting [Modify] from the pop-up menu in the [Static Memory Object] page opens the [Modification of Information for Static Memory Object Registration] dialog box. These two dialog boxes have the same configuration.

Static memory objects can be located only in an MMU mapped area, and its start address must be at the [Page Size] boundary.

Reference: Section 5.3.1 (1), MMU Mapped Area and MMU Non-Mapped Area

All static memory objects are associated with the kernel domain.

(1) [Memory Object Area] group

(a) [Kernel Side]

Select this check box when specifying the static memory object registered to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(b) [Specifies the Address] or [Specifies the Section Range]

Select either of them as the method to specify the address of the static memory object.

When [Specifies the Address] is selected, enter [Address] and [Size].

When [Specifies the Section Range] is selected, enter [Start Section Name] and [End Section Name].

(c) [Address] and [Size]

In [Address], specify the start address of the static memory object as a numeric value or C-language symbol. Though the range determined by [Address] and [Size] must be an MMU mapped area, the configurator does not fully check whether the inputs are correct so the specifications must be made carefully.

In [Size], specify the size (number of bytes) of the static memory object. An integer between 1 and 0x20000000 can be specified in [Size]. The [Size] value is rounded up to a multiple of [Page Size].

(d) [Start Section Name] and [End Section Name]

A single memory object is located in the range from [Start Section Name] to [End Section Name]. Taking this in consideration, a specified section must be allocated at the [Page Size] boundary of an MMU mapped area at linkage.

Example: To specify the three sections of PUCM_DOM1, CUCM_DOM1, and DUCM_DOM1 as one static memory object:

In this dialog box, specify PUCM_DOM1 in [Start Section Name] and DUCM_DOM1 in [End Section Name]. At linkage, locate PUCM_DOM1 at the [Page Size] boundary address of an MMU mapped area.

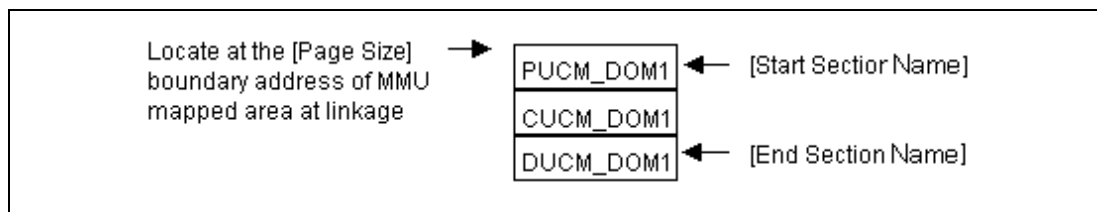


Figure 10.18 [Specifies the Section Range] for Static Memory Object

Reference: Section 11.15, Memory Allocation

(2) [Page Size] group

Select the page size as 4 kbytes, 8 kbytes, 64 kbytes, 256 kbytes, 1 Mbyte, 4 Mbytes, or 64 Mbytes. Note, however, that if the SH4AL-DSP or SH-4A without extended functions is used, the actual page size applied will be as shown in table 10.14.

Table 10.14 Page Size for SH4AL-DSP or SH-4A without Extended Functions

Selected Page Size	Actual Page Size Applied
4 kbytes	4 kbytes
8 kbytes	
64 kbytes	64 kbytes
256 kbytes	
1 Mbyte	1 Mbyte
4 Mbytes	
64 Mbytes	

Reference: Section 4.21.5, Page Size

(3) [Read/Write] group

Select read-only (TA_RO) or readable/writable (TA_RW) for the static memory object.

(4) [Cache Setting] group

Select how to handle the static memory object when cache is enabled from the following:

- Cacheable in write-back mode (TA_CACHE|TA_WBACK)
- Cacheable in write-through mode (TA_CACHE|TA_WTHROUGH)
- Non-cacheable (TA_UNCACHE)

(5) [Access Permission Vector] group

In [Access Permission Vector], select one from the following:

- (1) TACT_KERNEL
- (2) TACT_PRW (domid)
- (3) TACT_PRO (domid)
- (4) TACT_SRW
- (5) TACT_SRO
- (6) TACT_SRPW (domid)

Only when (2), (3), or (6) is selected, [Specified User Domain ID] becomes valid. In this case, select the user domain IDs to be permitted in [Specified User Domain ID].

[Setting Result] shows from which user domains write or read is possible, according to the settings of [Read/Write], [Access Permission Vector], and [Specified User Domain ID].

Table 10.15 shows the displayed contents of [Setting Result].

Table 10.15 Displayed Contents of [Setting Result]

Settings			[Setting Result] Display	
Read/Write	Access Permission Vector	Specified User Domain	User Domain Possible to Write	User Domain Possible to Read
TA_RO	TACT_KERNEL	Invalid	No user domain (even not the kernel domain)	No user domain
	TACT_PRW (domid)	Valid		Specified user domain only
	TACT_PRO (domid)	Valid		Specified user domain only
	TACT_SRW	Invalid		All user domains
	TACT_SRO	Invalid		All user domains
	TACT_SRPW (domid)	Valid		All user domains
TA_RW	TACT_KERNEL	Invalid	No user domain	No user domain
	TACT_PRW (domid)	Valid	Specified user domain only	Specified user domain only
	TACT_PRO (domid)	Valid	No user domain	Specified user domain only
	TACT_SRW	Invalid	All user domains	All user domains
	TACT_SRO	Invalid	No user domain	All user domains
	TACT_SRPW (domid)	Valid	Specified user domain only	All user domains

(6) [Register] button (only in [Registration of Static Memory Object] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next static memory object can be registered without break. This dialog box is not closed.

(7) [OK] button (only in [Modification of Information for Static Memory Object Registration] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(8) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

(9) [Notes]

- A static memory object can be located only in an MMU mapped area. However, the configurator and kernel do not fully check this. If located in an MMU non-mapped area, when that memory object is accessed, the memory attribute and access permission vector are ignored and it has no meaning as a memory object.
- The start address of a static memory object must be located at the specified [Page Size] boundary. However, except for when a value is specified in [Address], the configurator does not check this. If the start address is not at the [Page Size] boundary, the system goes down at kernel initiation.
- The range of the specified static memory object must not overlap with another static memory object or the system pool. Overlap refers to not only overlap in the logical address space but overlap in the physical address space. However, the configurator and kernel do not detect such kind of error. In this case, normal system operation cannot be guaranteed.

10.7.15 [Initialization Routine] Page

In this page, register an initialization routine.

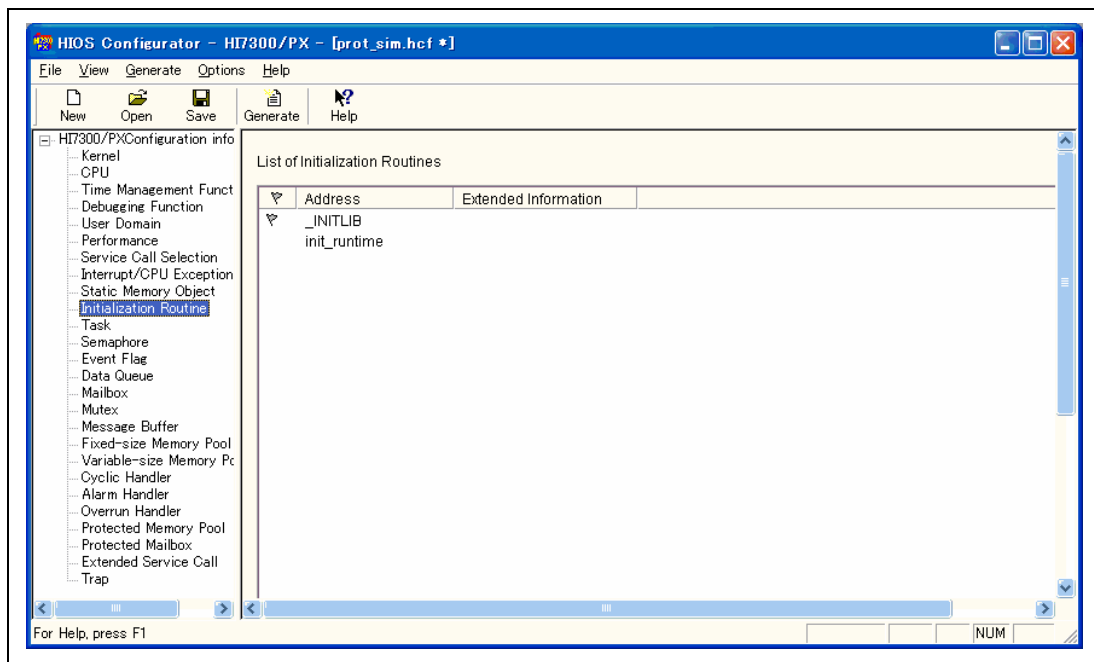


Figure 10.19 [Initialization Routine] Page

The initialization routines already registered are displayed in the list. The flag icon indicates that the initialization routine is registered to belong to the kernel side.

When the kernel is initiated, initialization routines on the kernel side (with the flag icon) are executed in sequence from the top of this list, and then initialization routines on the kernel environment side (without the flag icon) are executed in sequence from the top of this list.

The following items are in the pop-up menu.

- [Register]: Opens the [Registration of Initialization Routine] dialog box to register an initialization routine
- [Delete]: Deletes the selected initialization routine
- [Modify]: Opens the [Modification of Information for Initialization Routine Registration] dialog box to modify the selected initialization routine setting
- [Up]: Switches the selected initialization routine with the initialization routine immediately above
- [Down]: Switches the selected initialization routine with the initialization routine immediately below

10.7.16 [Registration of Initialization Routine] Dialog Box and [Modification of Information for Initialization Routine Registration] Dialog Box

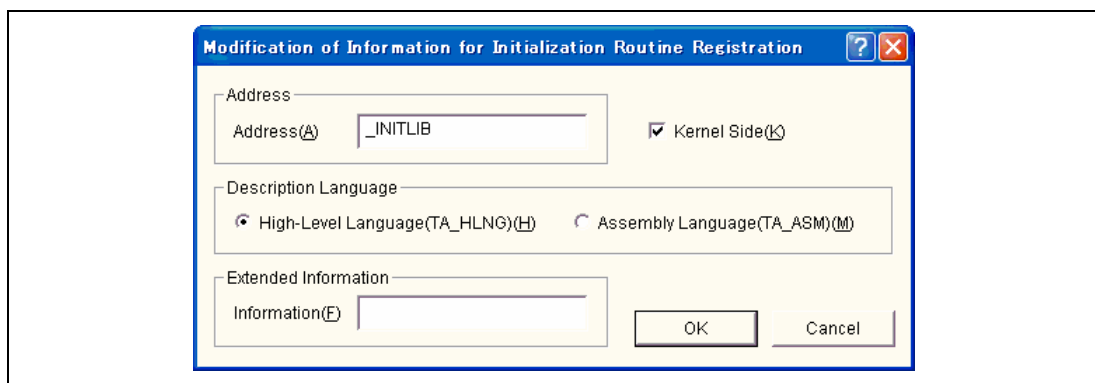


Figure 10.20 [Registration of Initialization Routine] Dialog Box

Selecting [Register] from the pop-up menu in the [Initialization Routine] page opens the [Registration of Initialization Routine] dialog box. Selecting [Modify] from the pop-up menu in the [Initialization Routine] page opens the [Modification of Information for Initialization Routine Registration] dialog box. These two dialog boxes have the same configuration.

(1) [Address]

Specify the address of the initialization routine as a C-language symbol or numeric value.

(2) [Kernel Side]

Select this check box when specifying the initialization routine registered to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(3) [Description Language]

Select [High-Level Language (TA_HLNG)] when the initialization routine is written in a high-level language, and select [Assembly Language (TA_ASM)] when the initialization routine is written in assembly language.

(4) [Extended Information]

The extended information is passed to the initialization routine as a parameter. Specify it as a C-language symbol or numeric value.

(5) [Register] button (only in [Registration of Initialization Routine] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next initialization routine can be registered without break. This dialog box is not closed.

(6) [OK] button (only in [Modification of Information for Initialization Routine Registration] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(7) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.17 [Task] Page

In this page, set items related to a task.

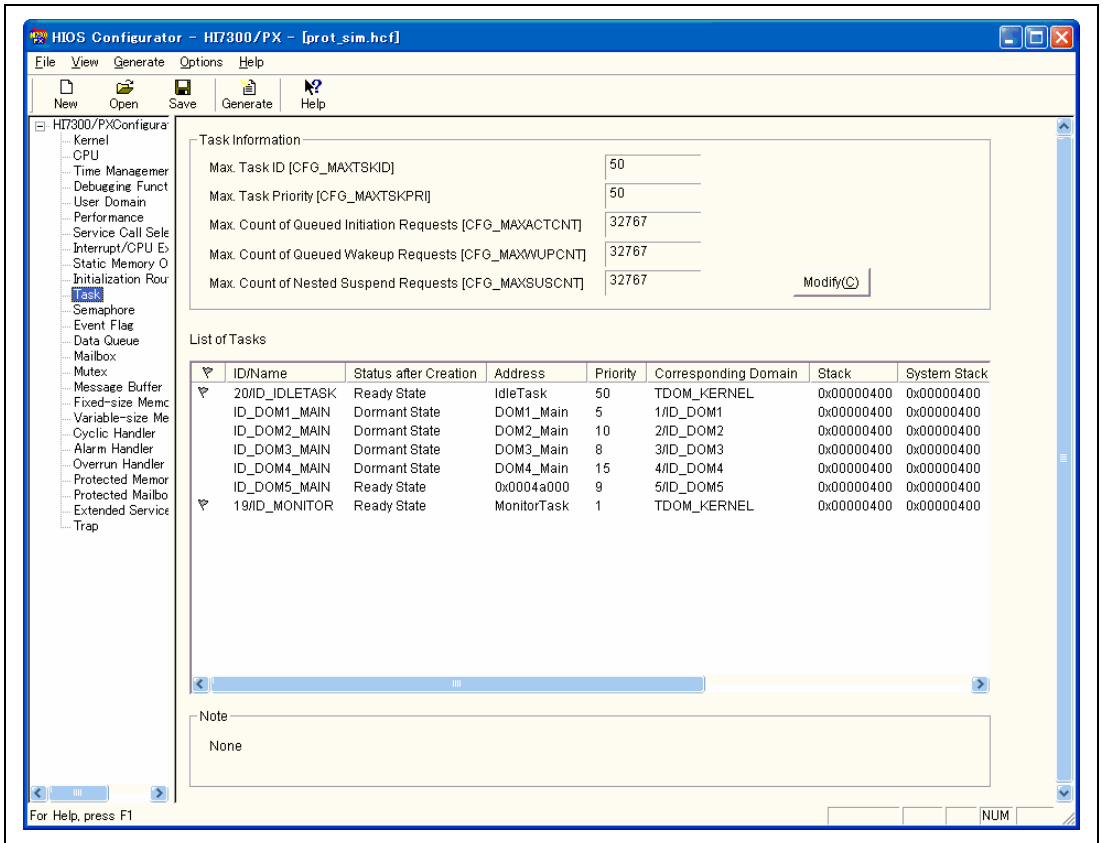


Figure 10.21 [Task] Page

Table 10.16 lists the [Task] page items.

Table 10.16 [Task] Page Items

Item	CFG Name	Linkage Unit
Max. Task ID	CFG_MAXTSKID	Kernel environment side
Max. Task Priority	CFG_MAXTSKPRI	Kernel side
Max. Count of Queued Initiation Requests	CFG_MAXACTCNT	Kernel side
Max. Count of Queued Wakeup Requests	CFG_MAXWUPCNT	Kernel side
Max. Count of Nested Suspend Requests	CFG_MAXSUSCNT	Kernel side
Creation of Task, Definition of Task	—	Kernel side/kernel
Exception Processing Routine		environment side

(1) [Task Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Task Information] dialog box in which the information can be changed.

(a) [Max. Task ID [CFG_MAXTSKID]]

The range of usable task IDs is between 1 and CFG_MAXTSKID.

(b) [Max. Task Priority [CFG_MAXTSKPRI]]

The priority range of usable tasks is between 1 and CFG_MAXTSKPRI.

(c) [Max. Count of Queued Initiation Requests [CFG_MAXACTCNT]]

The maximum number of queued initiation requests (act_tsk) for a task.

(d) [Max. Count of Queued Wakeup Requests [CFG_MAXWUPCNT]]

The maximum number of queued wakeup requests (wup_tsk) for a task.

(e) [Max. Count of Nested Suspend Requests [CFG_MAXSUSCNT]]

The maximum number of nested suspend requests (sus_tsk) for a task.

(2) [List of Tasks] group

In this group, the tasks already created are displayed. The flag icon indicates that the task is created to belong to the kernel side.

When the kernel is initiated, tasks on the kernel side (with the flag icon) are created in sequence from the top of this list, and then tasks on the kernel environment side (without the flag icon) are created in sequence from the top of this list. Note that if [Start Task after Creation (TA_ACT)] has been specified at task creation, the tasks enter the READY state in this sequence.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Task] dialog box to create a task
- [Delete]: Deletes the selected task
- [Modify]: Opens the [Modification of Information for Task Creation] dialog box to modify the selected task setting
- [Up]: Switches the selected task with the task immediately above
- [Down]: Switches the selected task with the task immediately below

(3) [Note]

All definitions for the task exception processing routine of this page become invalid when def_tex is not selected in the [Service Call Selection] page. In addition, the task exception processing routine cannot be defined.

In this case, the message shown in table 10.17 is displayed in [Note].

Table 10.17 [Note] in [Task] Page

Condition	Display Message
def_tex is not selected	All definitions for the task exception processing routine of this page are invalid because the setting to install def_tex is not made.

10.7.18 [Modification of Task Information] Dialog Box

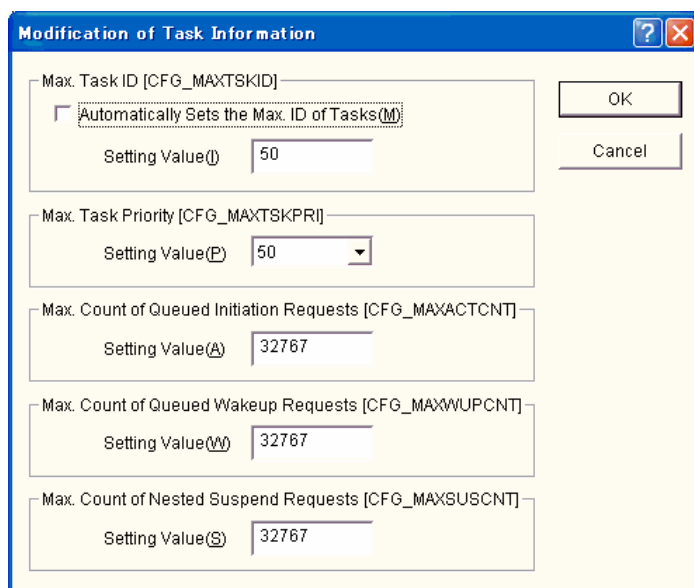


Figure 10.22 [Modification of Task Information] Dialog Box

Clicking the [Modify] button in the [Task] page opens this dialog box.

(1) [Max. Task ID [CFG_MAXTSKID]]

A task ID between 1 and CFG_MAXTSKID can be used. An integer between 1 and 32767 can be specified.

If [Automatically Sets the Max. ID of Tasks] is selected, the configurator automatically calculates the maximum ID based on the tasks created in the [Task] page.

(2) [Max. Task Priority [CFG_MAXTSKPRI]]

A task priority between 1 and CFG_MAXTSKPRI can be used. An integer between 1 and 255 can be selected.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TMAX_TPRI 255 /* When 255 is specified */
```

(3) [Max. Count of Queued Initiation Requests [CFG_MAXACTCNT]]

Specify the maximum number of queued initiation requests by the act_tsk and iact_tsk service calls. An integer between 1 and 32767 can be specified. If the same number of initiation requests are already queued for the specified task using the act_tsk or iact_tsk service call, the act_tsk or iact_tsk service call results in the E_QOVR error.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TMAX_ACTCNT 32767 /* When 32767 is specified */
```

(4) [Max. Count of Queued Wakeup Requests [CFG_MAXWUPCNT]]

Specify the maximum number of queued wakeup requests by the wup_tsk and iwup_tsk service calls. An integer between 1 and 32767 can be specified. If the same number of wakeup requests are already queued for the specified task using the wup_tsk or iwup_tsk service call, the wup_tsk or iwup_tsk service call results in the E_QOVR error.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TMAX_WUPCNT 32767 /* When 32767 is specified */
```

(5) [Max. Count of Nested Suspend Requests [CFG_MAXSUSCNT]]

Specify the maximum number of nested suspend requests by the sus_tsk and isus_tsk service calls. An integer between 1 and 32767 can be specified. If the same number of suspend requests are already nested for the specified task using the sus_tsk or isus_tsk service call, the sus_tsk or isus_tsk service call results in the E_QOVR error.

The following statement is output to kernel_macro.h in response to this setting.

```
#define TMAX_SUSCNT 32767 /* When 32767 is specified */
```

(6) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(7) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.19 [Creation of Task] Dialog Box and [Modification of Information for Task Creation] Dialog Box

Modification of Information for Task Creation

Task ID

☒ Automatic Assignment of ID Number

ID Number Corresponding Domain ID

ID Name ☐ Kernel Side

Task Address

Address

Task Initiation Priority

Priority

Coprocessor

☐ Uses DSP(TA_COP0)

☐ Uses FPU (bank 0)(TA_COP1)

☐ Uses FPU (bank 1)(TA_COP2)

Initial FPSCR Value

Stack

Application Stack Size

System Stack Size

The stack address cannot be specified. To specify the stack address, create a task using the service call.

Status after Creation

☐ Status after Creation(TA_ACT)

Description Language

☒ High-Level Language(TA_HLNG)

☐ Assembly Language(TA_ASM)

Extended Information

Information

Define Task Exception Processing Routine...

OK Cancel

Figure 10.23 [Creation of Task] Dialog Box

Selecting [Create] from the pop-up menu in the [Task] page opens the [Creation of Task] dialog box. Selecting [Modify] from the pop-up menu in the [Task] page opens the [Modification of Information for Task Creation] dialog box. These two dialog boxes have the same configuration.

A task can also be dynamically created using the cre_tsk or acre_tsk service call.

(1) [Task ID and Corresponding Domain] group

(a) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(b) [ID Number]

Enter the task ID as a numeric value. A value between 1 and CFG_MAXTSKID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(c) [ID Name]

Specify the ID name.

If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(d) [Corresponding Domain ID]

Select the domain to be associated. TDOM_KERNEL (kernel domain) or user domains with domain IDs from 1 to 31 can be selected.

(e) [Kernel Side]

Select this check box when specifying the task created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(2) [Task Address] group

Specify the start address of the task as a numeric value or C-language symbol.

(3) [Task Initiation Priority] group

Select the priority at task initiation.

(4) [Coprocessor] group

(a) [Uses DSP (TA_COP0)], [Uses FPU (bank 0) (TA_COP1)], or [Uses FPU (bank 1) (TA_COP2)]

TA_COP0 is valid only when CFG_DSP is selected in the [CPU] page. Select this check box when performing DSP calculation.

TA_COP1 or TA_COP2 is valid only when CFG_FPU is selected in the [CPU] page. For normal FPU calculation, select only TA_COP1. For cases using both FPU banks, i.e. matrix calculation,

select both TA_COP1 and TA_COP2. Selecting only TA_COP2 and not TA_COP1 is not possible.

TA_COP0 cannot be selected together with TA_COP1 or TA_COP2.

(b) [Initial FPSCR Value]

The initial FPSCR value has a meaning only when either TA_COP1 or TA_COP2 is selected. An integer between 0 and 0xffffffff can be specified. Specify this value with reference to the following.

Reference: Section 15, Notes on FPU

(5) [Stack] group

Specify the stack size and system stack size (bytes) as numeric values. A positive integer equal to or lower than 0x20000000 can be specified for each size.

The stack is allocated to the system pool or resource pool. If the system pool or resource pool does not have enough area for the specified size, an error message is displayed to inform it.

When creating a task using a service call, a specification to use the allocated area as the stack can be made by the application but not in the configurator.

The size specified here corresponds to stksz and sstksz specified in the cre_tsk service call. For details, refer to the following.

Reference: Section 6.7.1, Create Task (cre_tsk, icre_tsk, acre_tsk, iacre_tsk)

(6) [Status after Creation] group

To start the task after creation, select TA_ACT.

(7) [Description Language] group

Select [High-Level Language (TA_HLNG)] when the task is written in a high-level language, and select [Assembly Language (TA_ASM)] when the task is written in assembly language.

(8) [Extended Information] group

Specify it as a C-language symbol or numeric value.

(9) [Define Task Exception Processing Routine] button

Opens the [Definition of Task Exception Processing Routine] dialog box to define the task exception processing routine for the task to be created in this dialog box. Also, click this button to cancel definition of the task exception processing routine.

(10) [Create] button (only in [Creation of Task] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next task can be created without break. This dialog box is not closed.

(11) [OK] button (only in [Modification of Information for Task Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(12) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.20 [Definition of Task Exception Processing Routine] Dialog Box

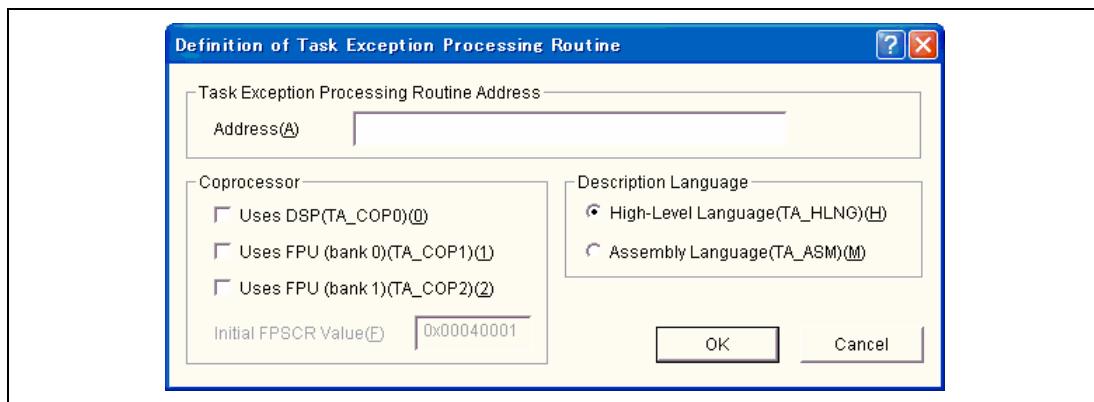


Figure 10.24 [Definition of Task Exception Processing Routine] Dialog Box

Clicking the [Define Task Exception Processing Routine] button in the [Creation of Task] dialog box or [Modification of Information for Task Creation] dialog box opens the [Definition of Task Exception Processing Routine] dialog box.

The distinction between the kernel side and kernel environment side in this dialog box is in accordance with the state of the [Kernel Side] check box in the [Creation of Task] dialog box or [Modification of Information for Task Creation] dialog box.

A task exception processing routine can also be dynamically defined using the def_tex service call.

(1) [Address]

Specify the start address of the task exception processing routine as a numeric value or C-language symbol.

If this edit box is left blank, definition of the task exception processing routine is canceled.

(2) [Uses DSP (TA_COP0)], [Uses FPU (bank 0) (TA_COP1)], or [Uses FPU (bank 1) (TA_COP2)]

TA_COP0 is valid only when CFG_DSP is selected in the [CPU] page. Select this check box when performing DSP calculation.

TA_COP1 or TA_COP2 is valid only when CFG_FPU is selected in the [CPU] page. For normal FPU calculation, select only TA_COP1. For cases using both FPU banks, i.e. matrix calculation, select both TA_COP1 and TA_COP2. Selecting only TA_COP2 and not TA_COP1 is not possible.

TA_COP0 cannot be selected together with TA_COP1 or TA_COP2.

(3) [Initial FPSCR Value]

The initial FPSCR value has a meaning only when either TA_COP1 or TA_COP2 is selected. An integer between 0 and 0xffffffff can be specified. Specify this value with reference to the following.

Reference: Section 15, Notes on FPU

(4) [Description Language]

Select [High-Level Language (TA_HLNG)] when the task exception processing routine is written in a high-level language, and select [Assembly Language (TA_ASM)] when the task exception processing routine is written in assembly language.

(5) [OK] button

Closes this dialog box after making the settings in this dialog box effective, and then returns to the former [Creation of Task] dialog box or [Modification of Information for Task Creation] dialog box.

(6) [Cancel] button

Closes this dialog box without saving the settings in this dialog box, and then returns to the former [Creation of Task] dialog box or [Modification of Information for Task Creation] dialog box.

10.7.21 [Semaphore] Page

In this page, set items related to a semaphore.

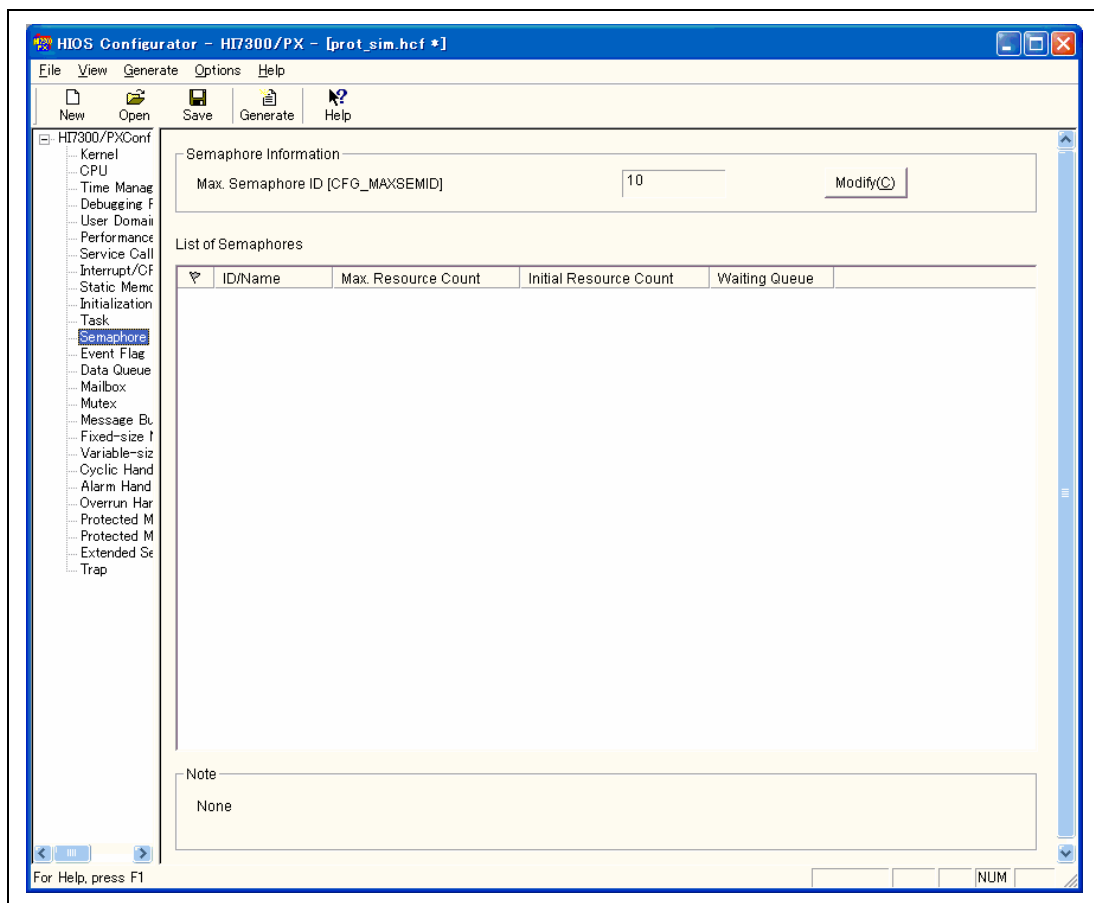


Figure 10.25 [Semaphore] Page

Table 10.18 lists the [Semaphore] page items.

Table 10.18 [Semaphore] Page Items

Item	CFG Name	Linkage Unit
Max. Semaphore ID	CFG_MAXSEMD	Kernel environment side
Creation of Semaphore	—	Kernel side/kernel environment side

(1) [Semaphore Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Semaphore Information] dialog box in which the information can be changed.

(a) [Max. Semaphore ID [CFG_MAXSEMICID]]

The range of usable semaphore IDs is between 1 and CFG_MAXSEMICID.

(2) [List of Semaphores] group

In this group, the semaphores already created are displayed. The flag icon indicates that the semaphore is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Semaphore] dialog box to create a semaphore
- [Delete]: Deletes the selected semaphore
- [Modify]: Opens the [Modification of Information for Semaphore Creation] dialog box to modify the selected semaphore setting

(3) [Note]

All setting items in this page become invalid when cre_sem is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.19 is displayed in [Note].

Table 10.19 [Note] in [Semaphore] Page

Condition	Display Message
cre_sem is not selected	All setting items in this page are invalid because the setting to install cre_sem is not made.

10.7.22 [Modification of Semaphore Information] Dialog Box

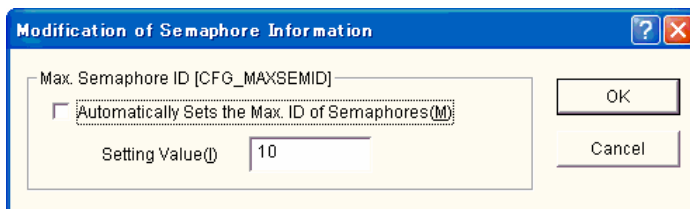


Figure 10.26 [Modification of Semaphore Information] Dialog Box

Clicking the [Modify] button in the [Semaphore] page opens this dialog box.

(1) [Max. Semaphore ID [CFG_MAXSEMICID]]

A semaphore ID between 1 and CFG_MAXSEMICID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, semaphores cannot be used. However, since a variable area for managing the semaphores does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Semaphores] is selected, the configurator automatically calculates the maximum ID based on the semaphores created in the [Semaphore] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.23 [Creation of Semaphore] Dialog Box and [Modification of Information for Semaphore Creation] Dialog Box

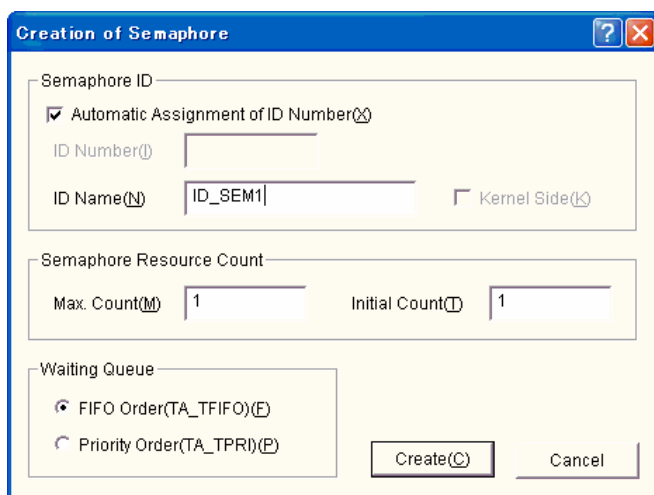


Figure 10.27 [Creation of Semaphore] Dialog Box

Selecting [Create] from the pop-up menu in the [Semaphore] page opens the [Creation of Semaphore] dialog box. Selecting [Modify] from the pop-up menu in the [Semaphore] page opens the [Modification of Information for Semaphore Creation] dialog box. These two dialog boxes have the same configuration.

A semaphore can also be dynamically created using the cre_sem or acre_sem service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the semaphore ID as a numeric value. A value between 1 and CFG_MAXSEMIC can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the semaphore created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Max. Count] of [Semaphore Resource Count]

Specify the maximum value of the semaphore resources. An integer between 1 and 65535 can be specified.

(6) [Initial Count] of [Semaphore Resource Count]

Specify the initial value of the semaphore resources. An integer between 0 and [Max. Count] of [Semaphore Resource Count] can be specified.

(7) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(8) [Create] button (only in [Creation of Semaphore] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next semaphore can be created without break. This dialog box is not closed.

(9) [OK] button (only in [Modification of Information for Semaphore Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(10) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.24 [Event Flag] Page

In this page, set items related to an event flag.

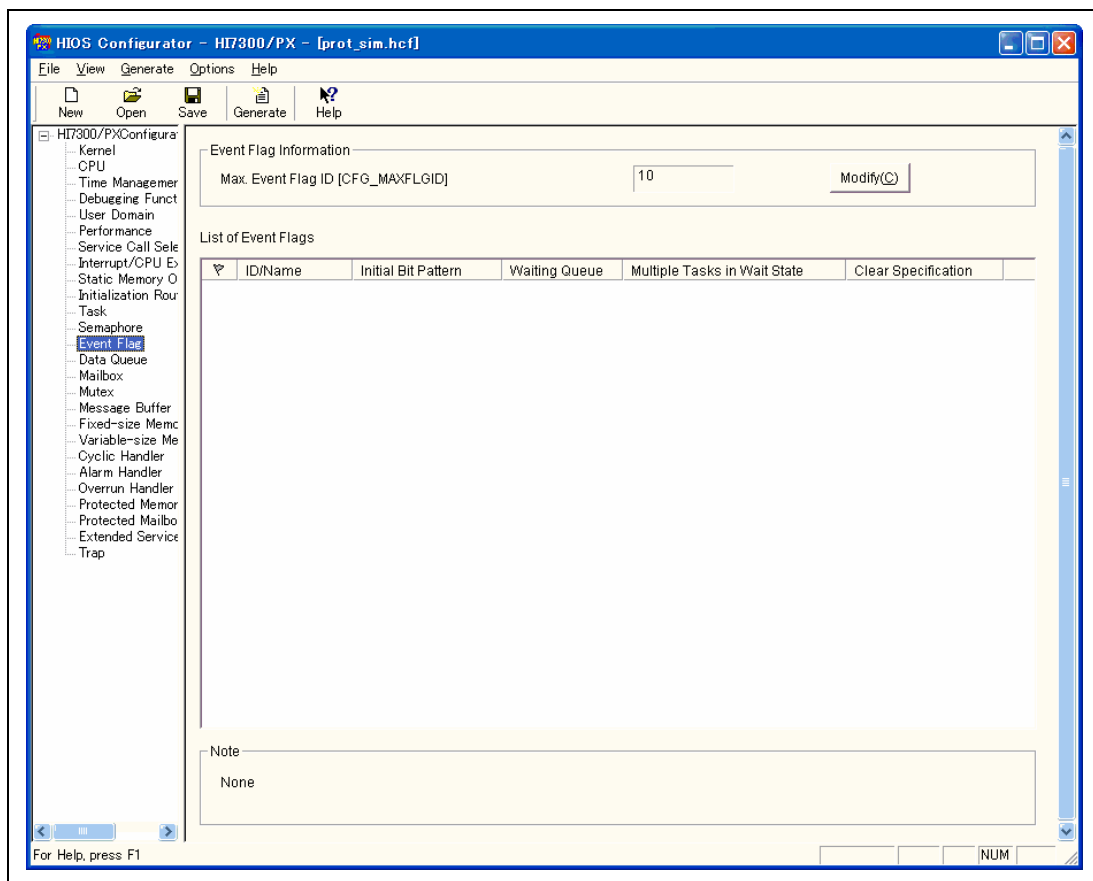


Figure 10.28 [Event Flag] Page

Table 10.20 lists the [Event Flag] page items.

Table 10.20 [Event Flag] Page Items

Item	CFG Name	Linkage Unit
Max. Event Flag ID	CFG_MAXFLGID	Kernel side
Creation of Event Flag	—	Kernel side/kernel environment side

(1) [Event Flag Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Event Flag Information] dialog box in which the information can be changed.

(a) [Max. Event Flag ID [CFG_MAXFLGID]]

The range of usable event flag IDs is between 1 and CFG_MAXFLGID.

(2) [List of Event Flags] group

In this group, the event flags already created are displayed. The flag icon indicates that the event flag is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Event Flag] dialog box to create an event flag
- [Delete]: Deletes the selected event flag
- [Modify]: Opens the [Modification of Information for Event Flag Creation] dialog box to modify the selected event flag setting

(3) [Note]

All setting items in this page become invalid when cre_flg is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.21 is displayed in [Note].

Table 10.21 [Note] in [Event Flag] Page

Condition	Display Message
cre_flg is not selected	All setting items in this page are invalid because the setting to install cre_flg is not made.

10.7.25 [Modification of Event Flag Information] Dialog Box

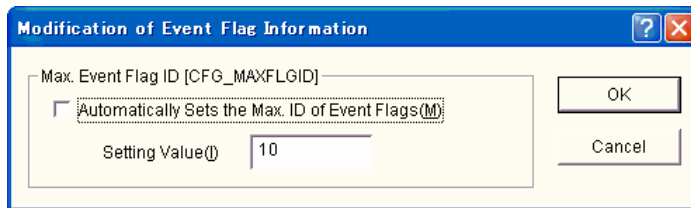


Figure 10.29 [Modification of Event Flag Information] Dialog Box

Clicking the [Modify] button in the [Event Flag] page opens this dialog box.

(1) [Max. Event Flag ID [CFG_MAXFLGID]]

An event flag ID between 1 and CFG_MAXFLGID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, event flags cannot be used. However, since a variable area for managing the event flags does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Event Flags] is selected, the configurator automatically calculates the maximum ID based on the event flags created in the [Event Flag] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.26 [Creation of Event Flag] Dialog Box and [Modification of Information for Event Flag Creation] Dialog Box

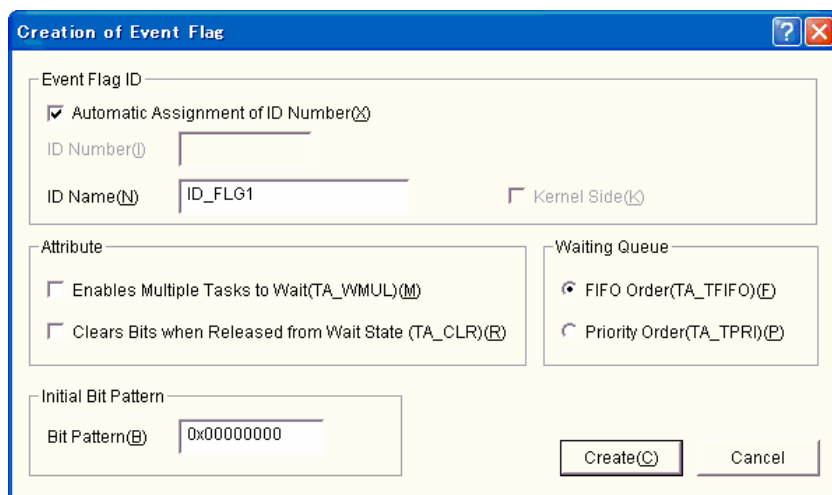


Figure 10.30 [Creation of Event Flag] Dialog Box

Selecting [Create] from the pop-up menu in the [Event Flag] page opens the [Creation of Event Flag] dialog box. Selecting [Modify] from the pop-up menu in the [Event Flag] page opens the [Modification of Information for Event Flag Creation] dialog box. These two dialog boxes have the same configuration.

An event flag can also be dynamically created using the `cre_flg` or `acre_flg` service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the event flag ID as a numeric value. A value between 1 and `CFG_MAXFLGID` can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the event flag created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Enables Multiple Tasks to Wait (TA_WMUL)]

Select this check box when making more than one task to wait for the event flag.

(6) [Clears Bits when Released from Wait State (TA_CLR)]

If this check box is selected, the event flag is cleared to 0 when `E_OK` is returned in the `wai_flg`, `twai_flg`, or `pol_flg` service call.

(7) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(8) [Initial Bit Pattern]

Specify the initial value of the event flag as an integer between 0 and `0xffffffff`.

(9) [Create] button (only in [Creation of Event Flag] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next event flag can be created without break. This dialog box is not closed.

(10) [OK] button (only in [Modification of Information for Event Flag Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(11) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.27 [Data Queue] Page

In this page, set items related to a data queue.

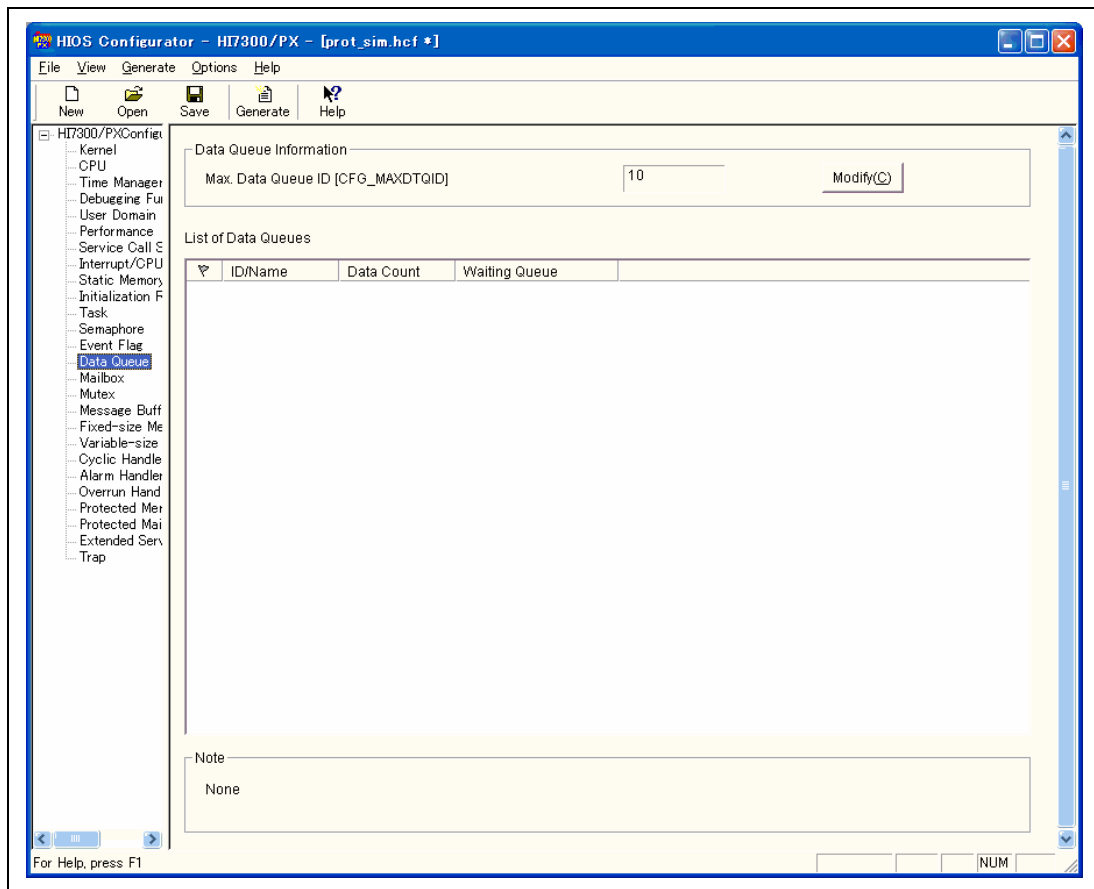


Figure 10.31 [Data Queue] Page

Table 10.22 lists the [Data Queue] page items.

Table 10.22 [Data Queue] Page Items

Item	CFG Name	Linkage Unit
Max. Data Queue ID	CFG_MAXDTQID	Kernel environment side
Creation of Data Queue	—	Kernel side/kernel environment side

(1) [Data Queue Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Data Queue Information] dialog box in which the information can be changed.

(a) [Max. Data Queue ID [CFG_MAXDTQID]]

The range of usable data queue IDs is between 1 and CFG_MAXDTQID.

(2) [List of Data Queues] group

In this group, the data queues already created are displayed. The flag icon indicates that the data queue is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Data Queue] dialog box to create a data queue
- [Delete]: Deletes the selected data queue
- [Modify]: Opens the [Modification of Information for Data Queue Creation] dialog box to modify the selected data queue setting

(3) [Note]

All setting items in this page become invalid when cre_dtq is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.23 is displayed in [Note].

Table 10.23 [Note] in [Data Queue] Page

Condition	Display Message
cre_dtq is not selected	All setting items in this page are invalid because the setting to install cre_dtq is not made.

10.7.28 [Modification of Data Queue Information] Dialog Box

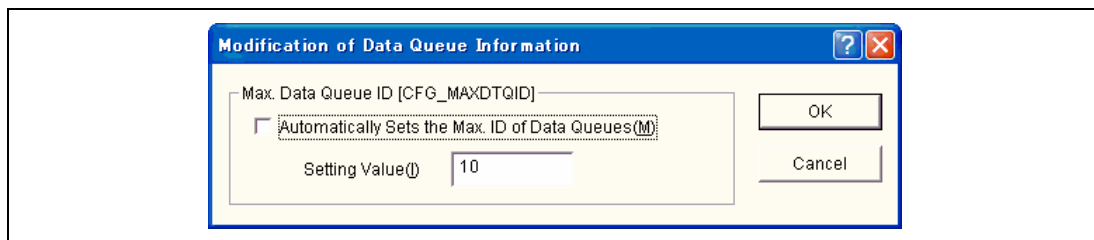


Figure 10.32 [Modification of Data Queue Information] Dialog Box

Clicking the [Modify] button in the [Data Queue] page opens this dialog box.

(1) [Max. Data Queue ID [CFG_MAXDTQID]]

A data queue ID between 1 and CFG_MAXDTQID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, data queues cannot be used. However, since a variable area for managing the data queues does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Data Queues] is selected, the configurator automatically calculates the maximum ID based on the data queues created in the [Data Queue] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.29 [Creation of Data Queue] Dialog Box and [Modification of Information for Data Queue Creation] Dialog Box

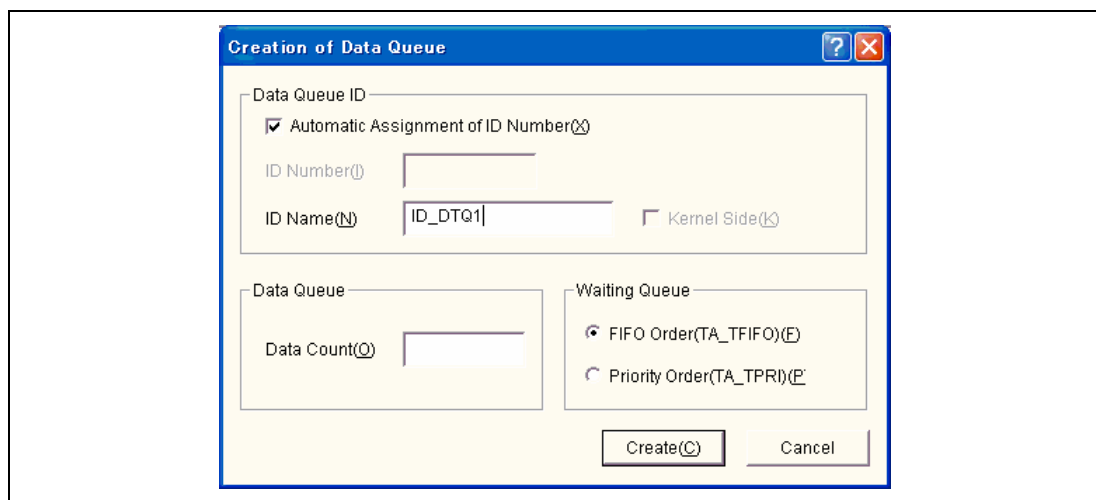


Figure 10.33 [Creation of Data Queue] Dialog Box

Selecting [Create] from the pop-up menu in the [Data Queue] page opens the [Creation of Data Queue] dialog box. Selecting [Modify] from the pop-up menu in the [Data Queue] page opens the [Modification of Information for Data Queue Creation] dialog box. These two dialog boxes have the same configuration.

A data queue can also be dynamically created using the cre_dtq or acre_dtq service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the data queue ID as a numeric value. A value between 1 and CFG_MAXDTQID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the data queue created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Data Count]

Specify the number of data that can be stored in the data queue. 0 can be specified.

(6) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(7) [Create] button (only in [Creation of Data Queue] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next data queue can be created without break. This dialog box is not closed.

(8) [OK] button (only in [Modification of Information for Data Queue Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(9) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.30 [Mailbox] Page

In this page, set items related to a mailbox.

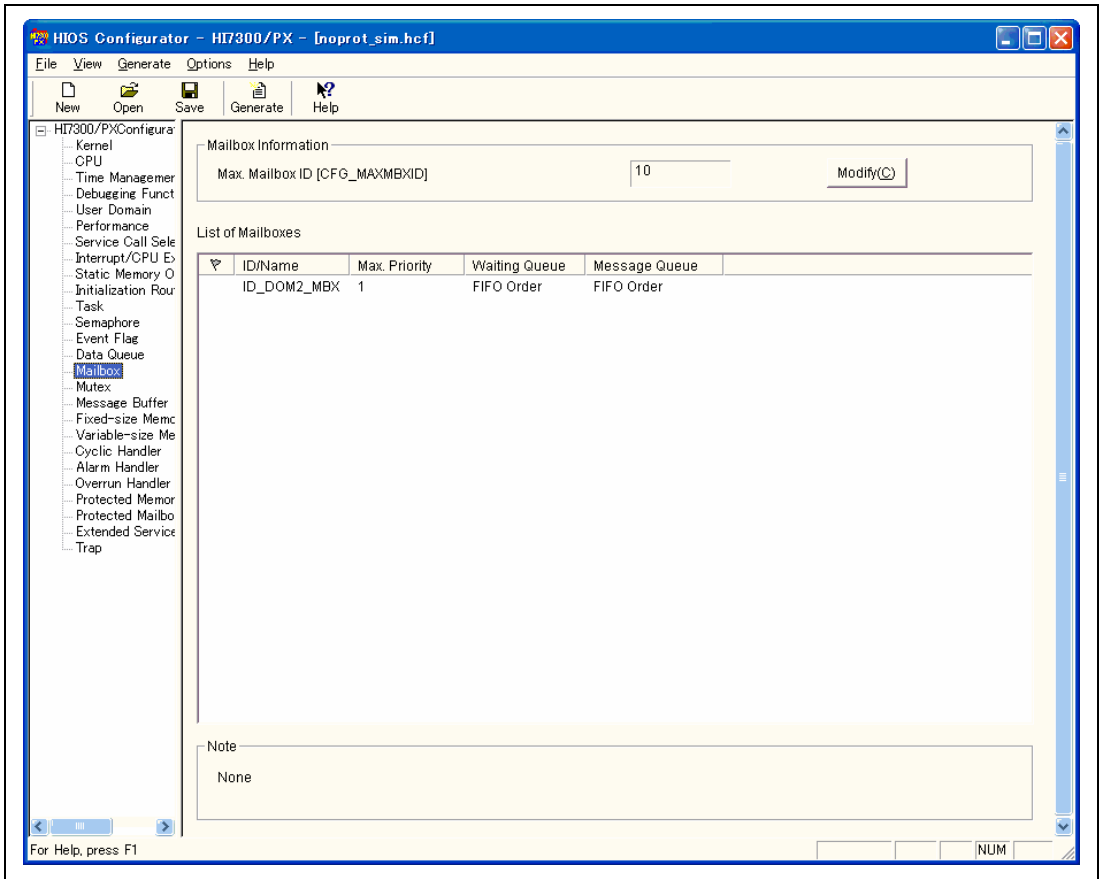


Figure 10.34 [Mailbox] Page

Table 10.24 lists the [Mailbox] page items.

Table 10.24 [Mailbox] Page Items

Item	CFG Name	Linkage Unit
Max. Mailbox ID	CFG_MAXMBXID	Kernel environment side
Creation of Mailbox	—	Kernel side/kernel environment side

(1) [Mailbox Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Mailbox Information] dialog box in which the information can be changed.

(a) [Max. Mailbox ID [CFG_MAXMBXID]]

The range of usable mailbox IDs is between 1 and CFG_MAXMBXID.

(2) [List of Mailboxes] group

In this group, the mailboxes already created are displayed. The flag icon indicates that the mailbox is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Mailbox] dialog box to create a mailbox
- [Delete]: Deletes the selected mailbox
- [Modify]: Opens the [Modification of Information for Mailbox Creation] dialog box to modify the selected mailbox setting

(3) [Note]

All setting items in this page become invalid when cre_mbx is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.25 is displayed in [Note].

Table 10.25 [Note] in [Mailbox] Page

Condition	Display Message
cre_mbx is not selected	All setting items in this page are invalid because the setting to install cre_mbx is not made.

10.7.31 [Modification of Mailbox Information] Dialog Box

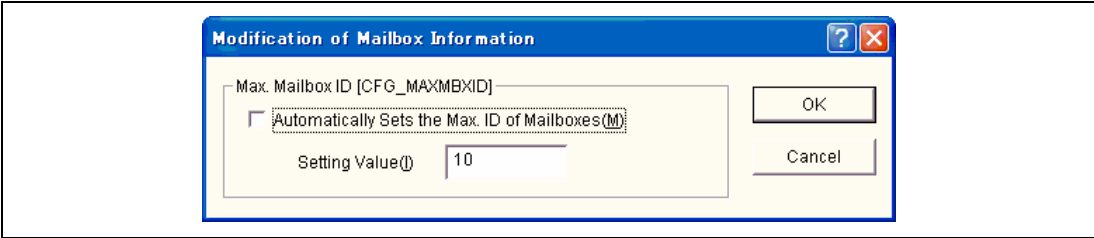


Figure 10.35 [Modification of Mailbox Information] Dialog Box

Clicking the [Modify] button in the [Mailbox] page opens this dialog box.

(1) [Max. Mailbox ID [CFG_MAXMBXID]]

A mailbox ID between 1 and CFG_MAXMBXID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, mailboxes cannot be used. However, since a variable area for managing the mailboxes does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Mailboxes] is selected, the configurator automatically calculates the maximum ID based on the mailboxes created in the [Mailbox] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.32 [Creation of Mailbox] Dialog Box and [Modification of Information for Mailbox Creation] Dialog Box

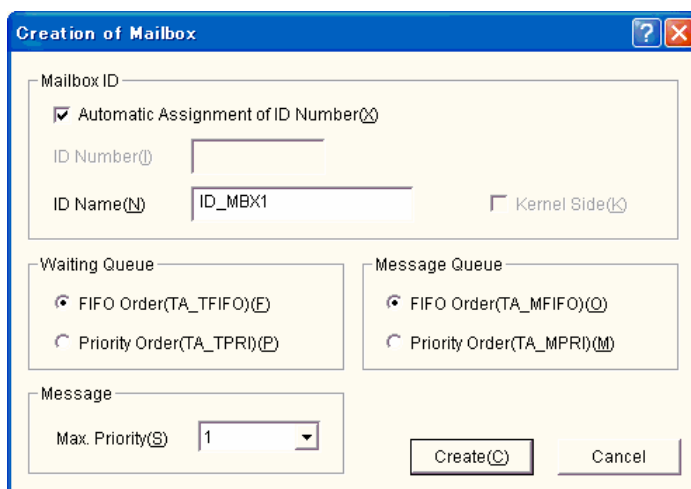


Figure 10.36 [Creation of Mailbox] Dialog Box

Selecting [Create] from the pop-up menu in the [Mailbox] page opens the [Creation of Mailbox] dialog box. Selecting [Modify] from the pop-up menu in the [Mailbox] page opens the [Modification of Information for Mailbox Creation] dialog box. These two dialog boxes have the same configuration.

A mailbox can also be dynamically created using the cre_mbx or acre_mbx service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the mailbox ID as a numeric value. A value between 1 and CFG_MAXMBXID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the mailbox created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(6) [Message Queue]

Select the FIFO order or priority order as the method to queue the messages.

(7) [Max. Priority]

When the priority order is selected in [Message Queue], select the maximum priority of the message. It can be selected between 1 and CFG_MAXMSGPRI.

When the FIFO order is selected in [Message Queue], this item has no meaning.

(8) [Create] button (only in [Creation of Mailbox] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next mailbox can be created without break. This dialog box is not closed.

(9) [OK] button (only in [Modification of Information for Mailbox Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(10) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.33 [Mutex] Page

In this page, set items related to a mutex.

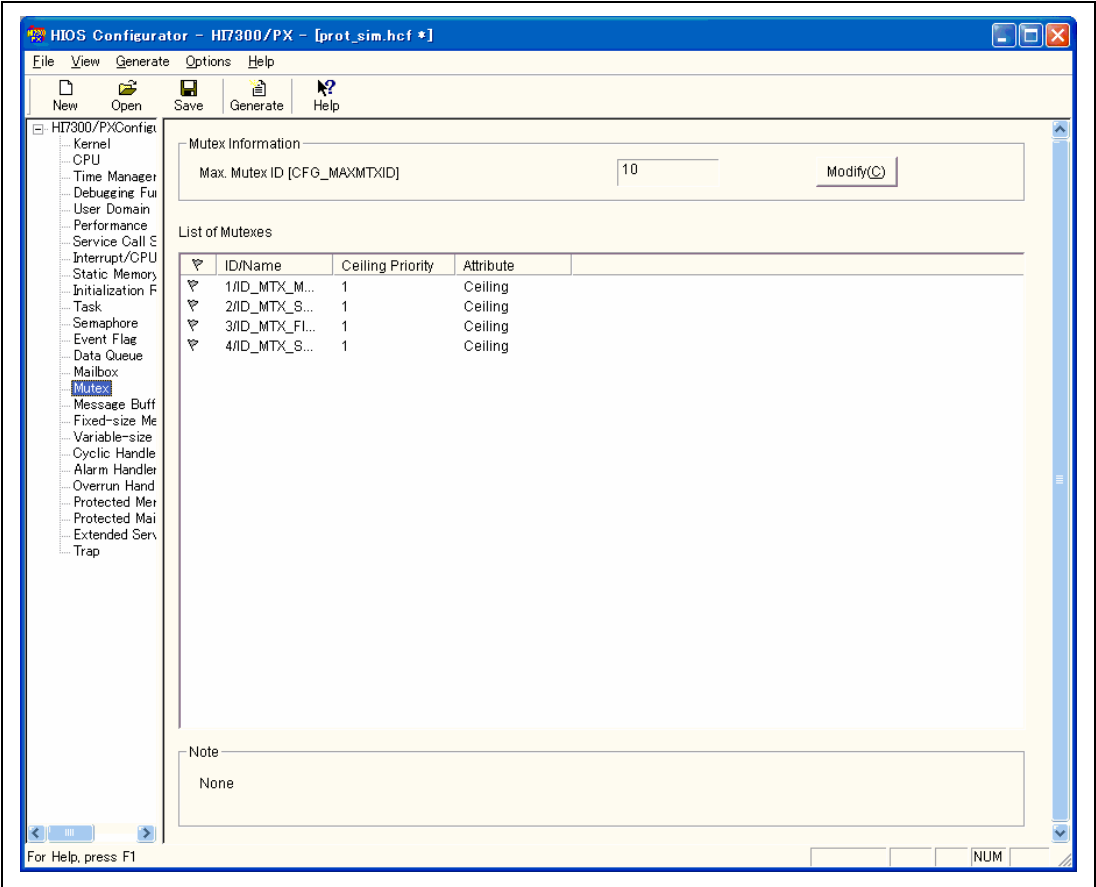


Figure 10.37 [Mutex] Page

Table 10.26 lists the [Mutex] page items.

Table 10.26 [Mutex] Page Items

Item	CFG Name	Linkage Unit
Max. Mutex ID	CFG_MAXMTXID	Kernel environment side
Creation of Mutex	—	Kernel side/kernel environment side

(1) [Mutex Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Mutex Information] dialog box in which the information can be changed.

(a) [Max. Mutex ID [CFG_MAXMTXID]]

The range of usable mutex IDs is between 1 and CFG_MAXMTXID.

(2) [List of Mutexes] group

In this group, the mutexes already created are displayed. The flag icon indicates that the mutex is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Mutex] dialog box to create a mutex
- [Delete]: Deletes the selected mutex
- [Modify]: Opens the [Modification of Information for Mutex Creation] dialog box to modify the selected mutex setting

(3) [Note]

All setting items in this page become invalid when cre_mtx is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.27 is displayed in [Note].

Table 10.27 [Note] in [Mutex] Page

Condition	Display Message
cre_mtx is not selected	All setting items in this page are invalid because the setting to install cre_mtx is not made.

10.7.34 [Modification of Mutex Information] Dialog Box

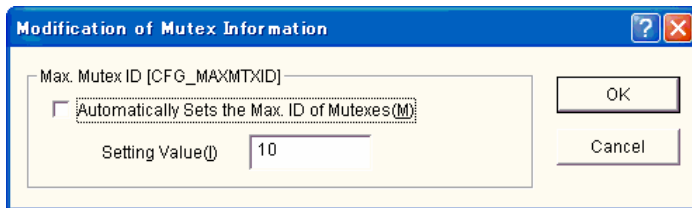


Figure 10.38 [Modification of Mutex Information] Dialog Box

Clicking the [Modify] button in the [Mutex] page opens this dialog box.

(1) [Max. Mutex ID [CFG_MAXMTXID]]

A mutex ID between 1 and CFG_MAXMTXID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, mutexes cannot be used. However, since a variable area for managing the mutexes does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Mutexes] is selected, the configurator automatically calculates the maximum ID based on the mutexes created in the [Mutex] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.35 [Creation of Mutex] Dialog Box and [Modification of Information for Mutex Creation] Dialog Box

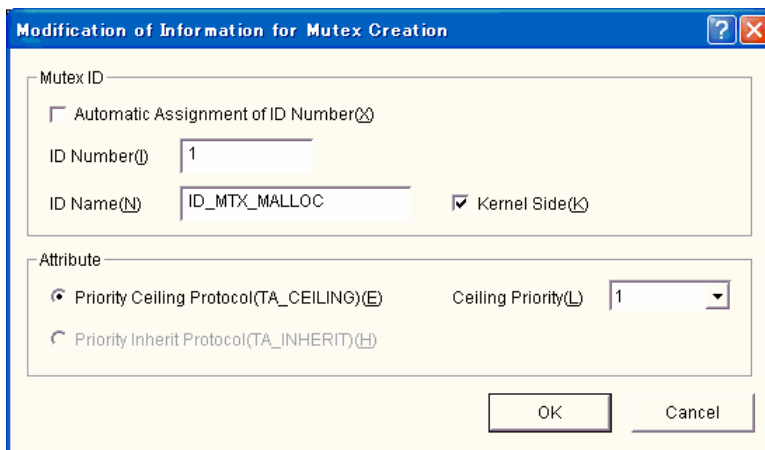


Figure 10.39 [Creation of Mutex] Dialog Box

Selecting [Create] from the pop-up menu in the [Mutex] page opens the [Creation of Mutex] dialog box. Selecting [Modify] from the pop-up menu in the [Mutex] page opens the [Modification of Information for Mutex Creation] dialog box. These two dialog boxes have the same configuration.

A mutex can also be dynamically created using the `cre_mtx` or `acre_mtx` service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the mutex ID as a numeric value. A value between 1 and `CFG_MAXMTXID` can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the mutex created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Priority Ceiling Protocol]

Only the priority ceiling protocol can be selected as an attribute.

(6) [Ceiling Priority]

Select the ceiling priority. It can be selected between 1 and `CFG_MAXTSKPRI`.

(7) [Create] button (only in [Creation of Mutex] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next mutex can be created without break. This dialog box is not closed.

(8) [OK] button (only in [Modification of Information for Mutex Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(9) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.36 [Message Buffer] Page

In this page, set items related to a message buffer.

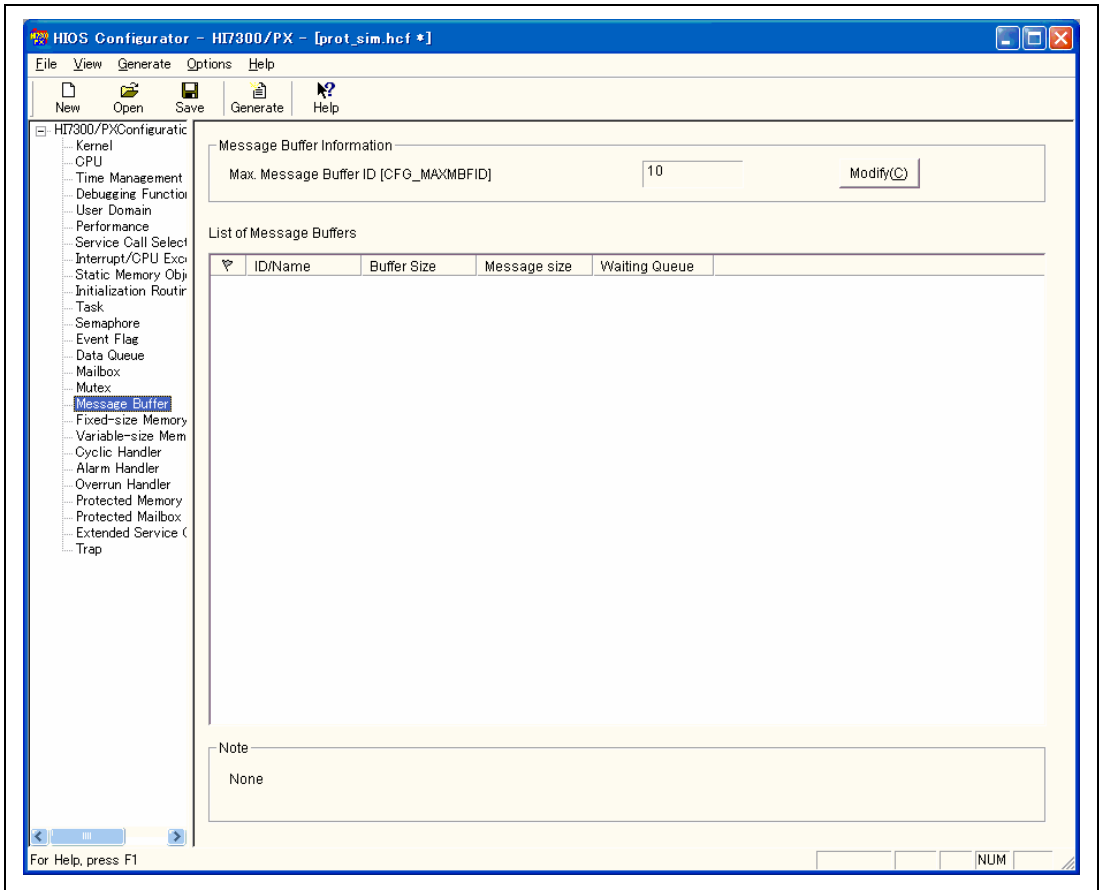


Figure 10.40 [Message Buffer] Page

Table 10.28 lists the [Message Buffer] page items.

Table 10.28 [Message Buffer] Page Items

Item	CFG Name	Linkage Unit
Max. Message Buffer ID	CFG_MAXMBFID	Kernel environment side
Creation of Message Buffer	—	Kernel side/kernel environment side

(1) [Message Buffer Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Message Buffer Information] dialog box in which the information can be changed.

(a) [Max. Message Buffer ID [CFG_MAXMBFID]]

The range of usable message buffer IDs is between 1 and CFG_MAXMBFID.

(2) [List of Message Buffers] group

In this group, the message buffers already created are displayed. The flag icon indicates that the message buffer is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Message Buffer] dialog box to create a message buffer
- [Delete]: Deletes the selected message buffer
- [Modify]: Opens the [Modification of Information for Message Buffer Creation] dialog box to modify the selected message buffer setting

(3) [Note]

All setting items in this page become invalid when cre_mbf is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.29 is displayed in [Note].

Table 10.29 [Note] in [Message Buffer] Page

Condition	Display Message
cre_mbf is not selected	All setting items in this page are invalid because the setting to install cre_mbf is not made.

10.7.37 [Modification of Message Buffer Information] Dialog Box

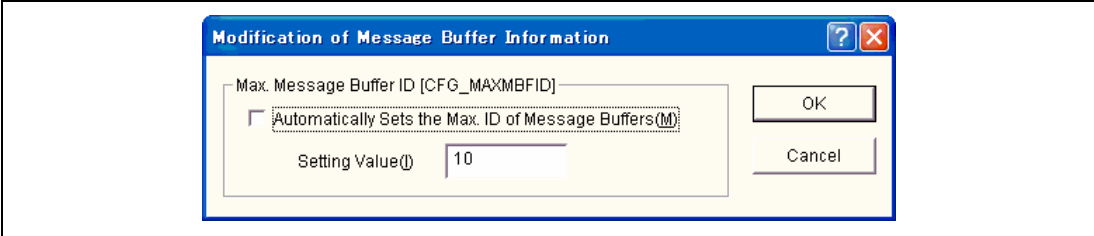


Figure 10.41 [Modification of Message Buffer Information] Dialog Box

Clicking the [Modify] button in the [Message Buffer] page opens this dialog box.

(1) [Max. Message Buffer ID [CFG_MAXMBFID]]

A message buffer ID between 1 and CFG_MAXMBFID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, message buffers cannot be used. However, since a variable area for managing the message buffers does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Message Buffers] is selected, the configurator automatically calculates the maximum ID based on the message buffers created in the [Message Buffer] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.38 [Creation of Message Buffer] Dialog Box and [Modification of Information for Message Buffer Creation] Dialog Box

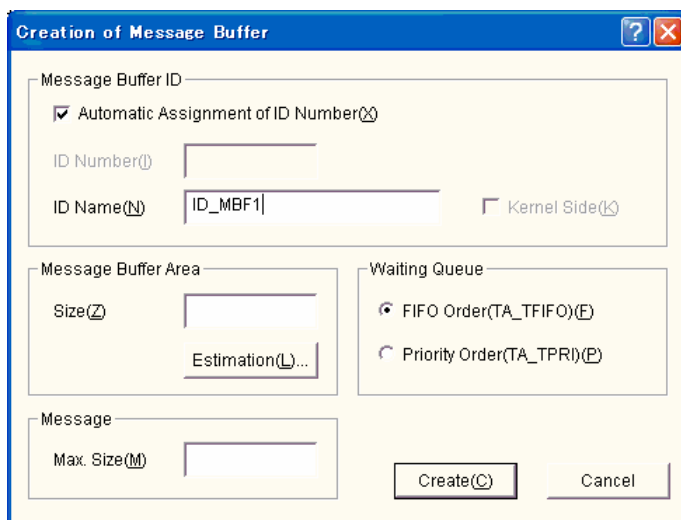


Figure 10.42 [Creation of Message Buffer] Dialog Box

Selecting [Create] from the pop-up menu in the [Message Buffer] page opens the [Creation of Message Buffer] dialog box. Selecting [Modify] from the pop-up menu in the [Message Buffer]

page opens the [Modification of Information for Message Buffer Creation] dialog box. These two dialog boxes have the same configuration.

A message buffer can also be dynamically created using the cre_mbf or acre_mbf service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the message buffer ID as a numeric value. A value between 1 and CFG_MAXMBFID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the message buffer created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Message Buffer Area Size]

Specify the message buffer size. A value between 0 and 0x20000000 can be specified. The specified value is rounded up to a multiple of four.

Clicking the [Estimation] button opens the [Estimation of Message Buffer Area Size] dialog box to calculate the estimated value of [Size].

(6) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(7) [Max. Size]

Specify the maximum size of the message to be transmitted to the message buffer. A value between 0 and 0x20000000 can be specified. The specified value is rounded up to a multiple of four.

(8) [Create] button (only in [Creation of Message Buffer] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next message buffer can be created without break. This dialog box is not closed.

(9) [OK] button (only in [Modification of Information for Message Buffer Creation] dialog box)
Closes this dialog box after making the settings in this dialog box effective.

(10) [Cancel] button
Closes this dialog box without saving the settings in this dialog box.

10.7.39 [Estimation of Message Buffer Area Size] Dialog Box

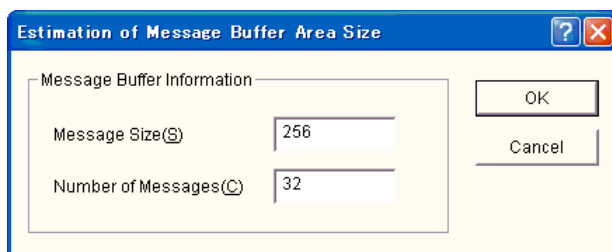


Figure 10.43 [Estimation of Message Buffer Area Size] Dialog Box

Clicking the [Estimation] button in the [Creation of Message Buffer] dialog box or [Modification of Information for Message Buffer Creation] dialog box opens this dialog box.

In this dialog box, the size of the message buffer that can store a message of the size specified in [Message Size] for the number of messages specified in [Number of Messages] is calculated. To be specific, the same calculation as the TSZ_MBFMB or TSZ_MBF macro is performed.

Clicking the [OK] button returns the display to the [Creation of Message Buffer] dialog box or [Modification of Information for Message Buffer Creation] dialog box, and [Size] of the message buffer area in either of these dialog boxes is updated to the calculation result of this dialog box.

Clicking the [Cancel] button returns the display to the [Creation of Message Buffer] dialog box or [Modification of Information for Message Buffer Creation] dialog box, but [Size] of the message buffer area in either of these dialog boxes is not updated.

10.7.40 [Fixed-size Memory Pool] Page

In this page, set items related to a fixed-size memory pool.

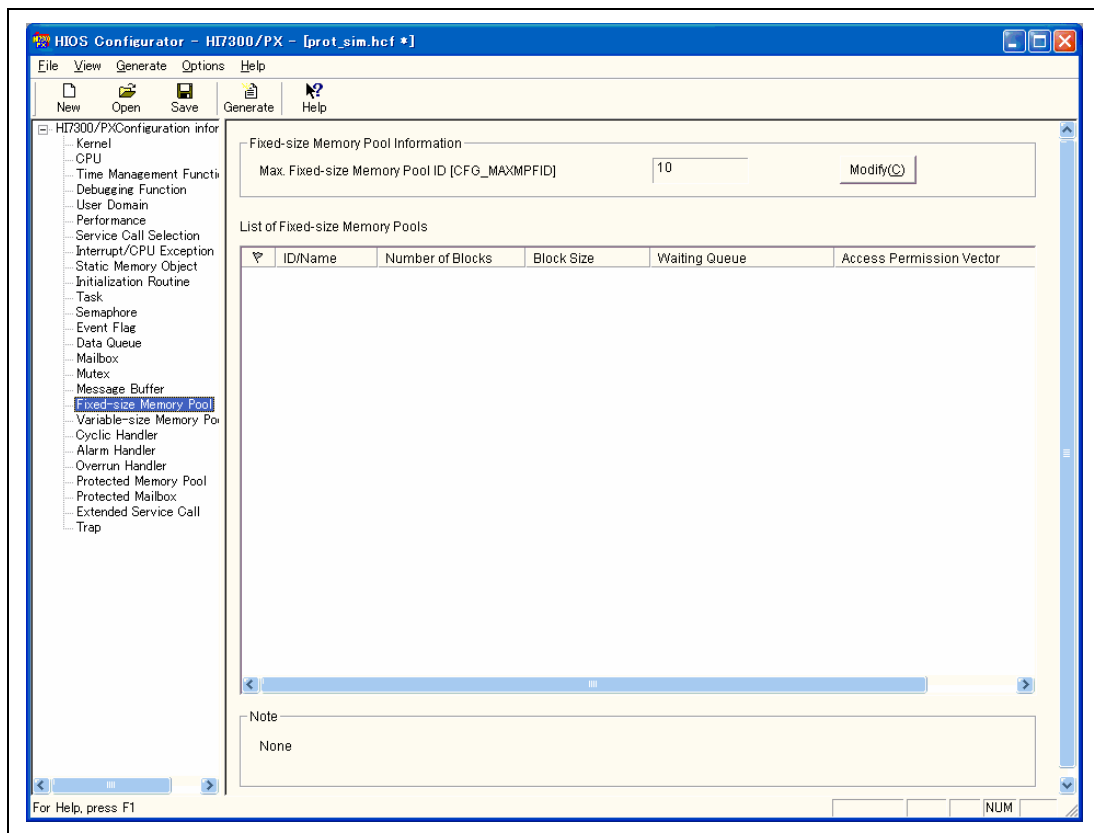


Figure 10.44 [Fixed-size Memory Pool] Page

Table 10.30 lists the [Fixed-size Memory Pool] page items.

Table 10.30 [Fixed-size Memory Pool] Page Items

Item	CFG Name	Linkage Unit
Max. Fixed-size Memory Pool ID	CFG_MAXMPFID	Kernel environment side
Creation of Fixed-size Memory Pool	—	Kernel side/kernel environment side

(1) [Fixed-size Memory Pool Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Fixed-size Memory Pool Information] dialog box in which the information can be changed.

(a) [Max. Fixed-size Memory Pool ID [CFG_MAXMPFID]]

The range of usable fixed-size memory pool IDs is between 1 and CFG_MAXMPFID.

(2) [List of Fixed-size Memory Pools] group

In this group, the fixed-size memory pools already created are displayed. The flag icon indicates that the fixed-size memory pool is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Fixed-size Memory Pool] dialog box to create a fixed-size memory pool
- [Delete]: Deletes the selected fixed-size memory pool
- [Modify]: Opens the [Modification of Information for Fixed-size Memory Pool Creation] dialog box to modify the selected fixed-size memory pool setting

(3) [Note]

All setting items in this page become invalid when cre_mpf is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.31 is displayed in [Note].

Table 10.31 [Note] in [Fixed-size Memory Pool] Page

Condition	Display Message
cre_mpf is not selected	All setting items in this page are invalid because the setting to install cre_mpf is not made.

10.7.41 [Modification of Fixed-size Memory Pool Information] Dialog Box

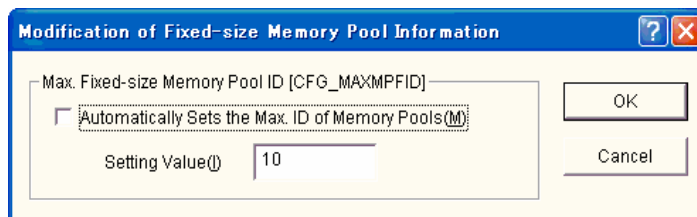


Figure 10.45 [Modification of Fixed-size Memory Pool Information] Dialog Box

Clicking the [Modify] button in the [Fixed-size Memory Pool] page opens this dialog box.

(1) [Max. Fixed-size Memory Pool ID [CFG_MAXMPFID]]

A fixed-size memory pool ID between 1 and CFG_MAXMPFID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, fixed-size memory pools cannot be used.

However, since a variable area for managing the fixed-size memory pools does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Memory Pools] is selected, the configurator automatically calculates the maximum ID based on the fixed-size memory pools created in the [Fixed-size Memory Pool] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.42 [Creation of Fixed-size Memory Pool] Dialog Box and [Modification of Information for Fixed-size Memory Pool Creation] Dialog Box

Creation of Fixed-size Memory Pool

Fixed-size Memory Pool ID

☒ Automatic Assignment of ID Number(X)

ID Number(I)

ID Name(N) ☐ Kernel Side(K)

Memory Block

Number of Blocks(Q) Block Size(Z)

Waiting Queue

☒ FIFO Order(TA_TFIFO)(F) ☐ Priority Order(TA_TPRI)(P)

Access Permission Vector

The address cannot be specified. To specify the address, create a fixed-size memory pool using the service call.

Memory Object Setting

This setting is ignored when CFG_PROTMEM is not selected (memory object protection function is not used).

Access Permission Vector(V)

Specified User Domain ID(D)

Setting Result

User Domain Possible to Write

User Domain Possible to Read

Create(C) Cancel

Figure 10.46 [Creation of Fixed-size Memory Pool] Dialog Box

Selecting [Create] from the pop-up menu in the [Fixed-size Memory Pool] page opens the [Creation of Fixed-size Memory Pool] dialog box. Selecting [Modify] from the pop-up menu in the [Fixed-size Memory Pool] page opens the [Modification of Information for Fixed-size Memory Pool Creation] dialog box. These two dialog boxes have the same configuration.

A fixed-size memory pool can also be dynamically created using the `cre_mpf` or `acre_mpf` service call. When creating a fixed-size memory pool using these service calls, a specification to use the

allocated area as a memory pool can be made by the application. However, since this specification cannot be made in the configurator, fixed-size memory pools are always allocated to the system pool.

(1) [Fixed-size Memory Pool ID] group

(a) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(b) [ID Number]

Enter the fixed-size memory pool ID as a numeric value. A value between 1 and CFG_MAXMPFID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(c) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(d) [Kernel Side]

Select this check box when specifying the fixed-size memory pool created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(2) [Memory Block] group

Create a fixed-size memory pool that can acquire a memory block of the size specified in [Block Size] for the number of blocks specified in [Number of Blocks].

In [Number of Blocks], specify a value between 1 and 0x08000000.

In [Block Size], specify a value between 4 and 0x20000000. The specified value is rounded up to a multiple of four.

(3) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(4) [Access Permission Vector] group

All items in this group have meaning only when CFG_PROTMEM is selected in the [Kernel] page. In [Access Permission Vector], select one from the following:

(1) TACT_KERNEL

- (2) TACT_PRW (domid)
- (3) TACT_PRO (domid)
- (4) TACT_SRW
- (5) TACT_SRO
- (6) TACT_SRPW (domid)

Only when (2), (3), or (6) is selected, [Specified User Domain ID] becomes valid. In this case, select the user domain ID to be permitted in [Specified User Domain ID].

[Setting Result] shows from which user domains write or read is possible, according to the settings of [Access Permission Vector] and [Specified User Domain ID].

Table 10.32 shows the displayed contents of [Setting Result].

The following memory attributes are always used.

- TA_RW (Readable/Writable)
- TA_CACHE (Cacheable)
- TA_WBACK (Cacheable in write-back mode)

Table 10.32 Displayed Contents of [Setting Result]

Settings		[Setting Result] Display	
Access Permission Vector	Specified User Domain	User Domain Possible to Write	User Domain Possible to Read
TACT_KERNEL	Invalid	No user domain	No user domain
TACT_PRW (domid)	Valid	Specified user domain only	Specified user domain only
TACT_PRO (domid)	Valid	No user domain	Specified user domain only
TACT_SRW	Invalid	All user domains	All user domains
TACT_SRO	Invalid	No user domain	All user domains
TACT_SRPW (domid)	Valid	Specified user domain only	All user domains

(5) [Create] button (only in [Creation of Fixed-size Memory Pool] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next fixed-size memory pool can be created without break. This dialog box is not closed.

(6) [OK] button (only in [Modification of Information for Fixed-size Memory Pool Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(7) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.43 [Variable-size Memory Pool] Page

In this page, set items related to a variable-size memory pool.

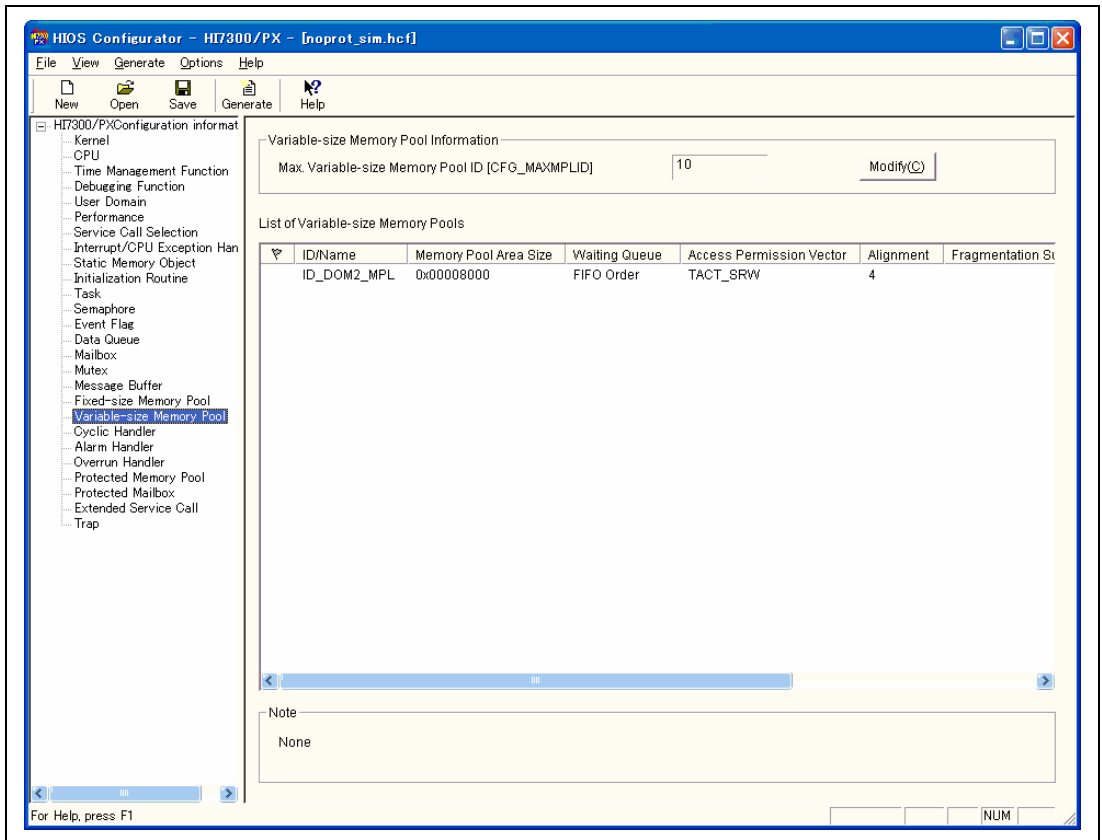


Figure 10.47 [Variable-size Memory Pool] Page

Table 10.33 lists the [Variable-size Memory Pool] page items.

Table 10.33 [Variable-size Memory Pool] Page Items

Item	CFG Name	Linkage Unit
Max. Variable-size Memory Pool ID	CFG_MAXMPLID	Kernel environment side
Creation of Variable-size Memory Pool	—	Kernel side/kernel environment side

(1) [Variable-size Memory Pool Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Variable-size Memory Pool Information] dialog box in which the information can be changed.

(a) [Max. Variable-size Memory Pool ID [CFG_MAXMPLID]]

The range of usable variable-size memory pool IDs is between 1 and CFG_MAXMPLID.

(2) [List of Variable-size Memory Pools] group

In this group, the variable-size memory pools already created are displayed. The flag icon indicates that the variable-size memory pool is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Variable-size Memory Pool] dialog box to create a variable-size memory pool
- [Delete]: Deletes the selected variable-size memory pool
- [Modify]: Opens the [Modification of Information for Variable-size Memory Pool Creation] dialog box to modify the selected variable-size memory pool setting

(3) [Note]

All setting items in this page become invalid when cre_mpl is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.34 is displayed in [Note].

Table 10.34 [Note] in [Variable-size Memory Pool] Page

Condition	Display Message
cre_mpl is not selected	All setting items in this page are invalid because the setting to install cre_mpl is not made.

10.7.44 [Modification of Variable-size Memory Pool Information] Dialog Box

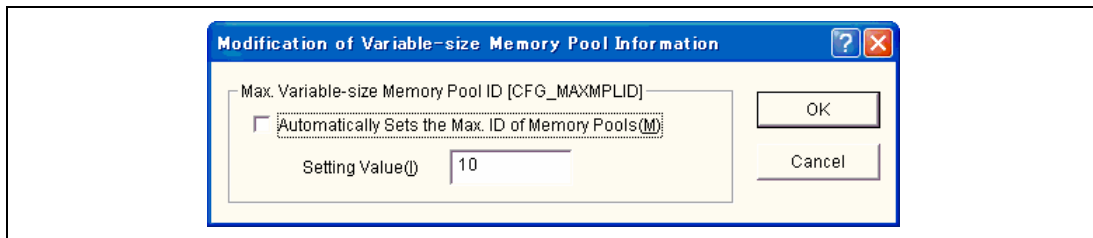


Figure 10.48 [Modification of Variable-size Memory Pool Information] Dialog Box

Clicking the [Modify] button in the [Variable-size Memory Pool] page opens this dialog box.

(1) [Max. Variable-size Memory Pool ID [CFG_MAXMPLID]]

A variable-size memory pool ID between 1 and CFG_MAXMPLID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, variable-size memory pools cannot be used. However, since a variable area for managing the variable-size memory pools does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Memory Pools] is selected, the configurator automatically calculates the maximum ID based on the variable-size memory pools created in the [Variable-size Memory Pool] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.45 [Creation of Variable-size Memory Pool] Dialog Box and [Modification of Information for Variable-size Memory Pool Creation] Dialog Box

Modification of Information for Variable-size Memory Pool Creation

Variable-size Memory Pool ID

- ☒ Automatic Assignment of ID Number
- ID Number:
- ID Name: ☐ Kernel Side

Waiting Queue

- ☒ FIFO Order(TA_TFIFO)
- ☐ Priority Order(TA_TPRI)

Sector Management

- ☐ Sector Management(VTA_UNFRAGMENT)
- Min. Block Size:
- Max. Sector Count:

Alignment Adjustment

- ☒ Aligns the memory block address to a multiple of 4(4)
- ☐ Aligns the memory block address to a multiple of 16(VTA_ALIGN16)(6)
- ☐ Aligns the memory block address to a multiple of 32(VTA_ALIGN32)(3)

Memory Pool Area

The address cannot be specified. To specify the address, create a variable-size memory pool using the service call.

Size: Estimation:

Memory Object Setting

This setting is ignored when CFG_PROTMEM is not selected (memory object protection function is not used).

Access Permission Vector:

Specified User Domain ID:

Setting Result

User Domain Possible to Write:

User Domain Possible to Read:

OK Cancel

Figure 10.49 [Creation of Variable-size Memory Pool] Dialog Box

Selecting [Create] from the pop-up menu in the [Variable-size Memory Pool] page opens the [Creation of Variable-size Memory Pool] dialog box. Selecting [Modify] from the pop-up menu in the [Variable-size Memory Pool] page opens the [Modification of Information for Variable-size Memory Pool Creation] dialog box. These two dialog boxes have the same configuration.

A variable-size memory pool can also be dynamically created using the `cre_mpl` or `acre_mpl` service call. When creating a variable-size memory pool using these service calls, a specification to use the allocated area as a memory pool can be made by the application. However, since this

specification cannot be made in the configurator, variable-size memory pools are always allocated to the system pool.

(1) [Variable-size Memory Pool ID] group

(a) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(b) [ID Number]

Enter the variable-size memory pool ID as a numeric value. A value between 1 and CFG_MAXMPLID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(c) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(d) [Kernel Side]

Select this check box when specifying the variable-size memory pool created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(2) [Waiting Queue] group

Only the FIFO order can be selected as the method to queue the waiting tasks.

(3) [Sector Management] group

When [Sector Management (VTA_UNFRAGMENT)] is selected, variable-size memory pools are managed by sectors. [Min. Block Size] and [Max. Sector Count] are valid only in this case.

The sector management method is an attribute suitable for a memory pool that acquires a large quantity of small memory blocks. By using small blocks continuously as much as possible, a continuous free area of a large size can be maintained more easily. For details on sector management and the meanings of [Min. Block Size] and [Max. Sector Count], refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

In [Min. Block Size], an integer other than 0 can be specified, and it is rounded up to a multiple of the size selected in the [Alignment Adjustment] group.

In [Max. Sector Count], an integer other than 0 can be specified. If 0 is specified, it is corrected to 1.

When [Max. Sector Count] is greater than $[\text{Size}] / ([\text{Min. Block Size}] \times 32)$, the actual Max. sector count is corrected to $[\text{Size}] / ([\text{Min. Block Size}] \times 32)$ by the kernel.

Note that if [Sector Management (VTA_UNFRAGMENT)] is selected, the size of the memory block acquired from the memory pool is rounded up to a multiple of [Min. Block Size].

(4) [Alignment Adjustment] group

Make a specification related to adjusting alignment of the address of the memory block acquired from the memory pool.

Select one from the three alignment methods below.

- (a) [Aligns the memory block address to a multiple of 4]
- (b) [Aligns the memory block address to a multiple of 16 (VTA_ALIGN16)]
- (c) [Aligns the memory block address to a multiple of 32 (VTA_ALIGN32)]

(5) [Memory Pool Area] group

In [Size], specify the memory pool size. A value between 4 and 0x20000000 can be specified. The specified value is rounded up to a multiple of four.

Clicking the [Estimation] button opens the [Estimation of Variable-size Memory Pool Area Size] dialog box to calculate the estimated value of [Size].

[Access Permission Vector] has meaning only when CFG_PROTMEM is selected in the [Kernel] page. In [Access Permission Vector], select one from the following:

- (1) TACT_KERNEL
- (2) TACT_PRW (domid)
- (3) TACT_PRO (domid)
- (4) TACT_SRW
- (5) TACT_SRO
- (6) TACT_SRPW (domid)

Only when (2), (3), or (6) is selected, [Specified User Domain ID] becomes valid. In this case, select the user domain ID to be permitted in [Specified User Domain ID].

[Setting Result] shows from which user domains write or read is possible, according to the settings of [Access Permission Vector] and [Specified User Domain ID].

Table 10.35 shows the displayed contents of [Setting Result].

The following memory attributes are always used.

- TA_RW (Readable/Writable)
- TA_CACHE (Cacheable)
- TA_WBACK (Cacheable in write-back mode)

Table 10.35 Displayed Contents of [Setting Result]

Settings		[Setting Result] Display	
Access Permission Vector	Specified User Domain	User Domain Possible to Write	User Domain Possible to Read
TACT_KERNEL	Invalid	No user domain	No user domain
TACT_PRW (domid)	Valid	Specified user domain only	Specified user domain only
TACT_PRO (domid)	Valid	No user domain	Specified user domain only
TACT_SRW	Invalid	All user domains	All user domains
TACT_SRO	Invalid	No user domain	All user domains
TACT_SRPW (domid)	Valid	Specified user domain only	All user domains

(6) [Create] button (only in [Creation of Variable-size Memory Pool] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next variable-size memory pool can be created without break. This dialog box is not closed.

(7) [OK] button (only in [Modification of Information for Variable-size Memory Pool Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(8) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.46 [Estimation of Variable-size Memory Pool Area Size] Dialog Box

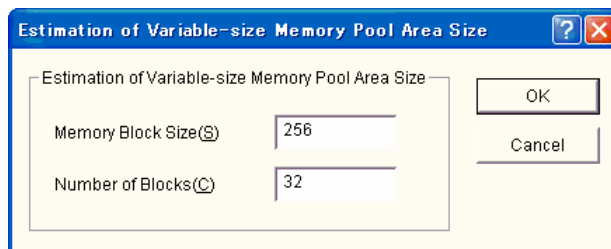


Figure 10.50 [Estimation of Variable-size Memory Pool Area Size] Dialog Box

Clicking the [Estimation] button in the [Creation of Variable-size Memory Pool] dialog box or [Modification of Information for Variable-size Memory Pool Creation] dialog box opens this dialog box.

In this dialog box, the size of the variable-size memory pool that can acquire a memory block of the size specified in [Memory Block Size] for the number of blocks specified in [Number of Blocks] is calculated. To be specific, the same calculation as the TSZ_MPL macro is performed.

Clicking the [OK] button returns the display to the [Creation of Variable-size Memory Pool] dialog box or [Modification of Information for Variable-size Memory Pool Creation] dialog box, and [Size] of the variable-size memory pool area in either of these dialog boxes is updated to the calculation result of this dialog box.

Clicking the [Cancel] button returns the display to the [Creation of Variable-size Memory Pool] dialog box or [Modification of Information for Variable-size Memory Pool Creation] dialog box, but [Size] of the variable-size memory pool area in either of these dialog boxes is not updated.

10.7.47 [Cyclic Handler] Page

In this page, set items related to a cyclic handler.

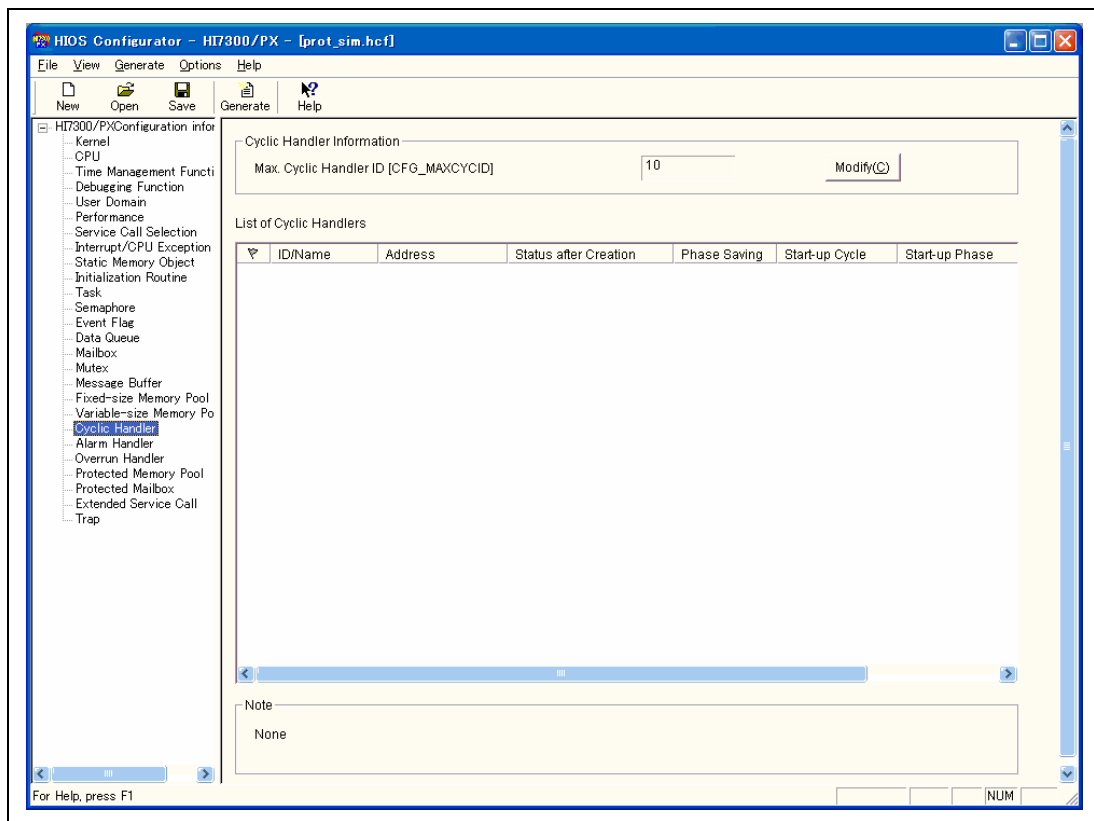


Figure 10.51 [Cyclic Handler] Page

Table 10.36 lists the [Cyclic Handler] page items.

Table 10.36 [Cyclic Handler] Page Items

Item	CFG Name	Linkage Unit
Max. Cyclic Handler ID	CFG_MAXCYCID	Kernel environment side
Creation of Cyclic Handler	—	Kernel side/kernel environment side

(1) [Cyclic Handler Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Cyclic Handler Information] dialog box in which the information can be changed.

(a) [Max. Cyclic Handler ID [CFG_MAXCYCID]]

The range of usable cyclic handler IDs is between 1 and CFG_MAXCYCID.

(2) [List of Cyclic Handlers] group

In this group, the cyclic handlers already created are displayed. The flag icon indicates that the cyclic handler is created to belong to the kernel side.

When the kernel is initiated, cyclic handlers on the kernel side (with the flag icon) are created in sequence from the top of this list, and then cyclic handlers on the kernel environment side (without the flag icon) are created in sequence from the top of this list. Note that if [Start Handler after Creation (TA_STA)] or [Save Initiation Phase (TA_PHS)] has been specified at handler creation, the cyclic handlers are created in this sequence.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Cyclic Handler] dialog box to create a cyclic handler
- [Delete]: Deletes the selected cyclic handler
- [Modify]: Opens the [Modification of Information for Cyclic Handler Creation] dialog box to modify the selected cyclic handler setting
- [Up]: Switches the selected cyclic handler with the cyclic handler immediately above
- [Down]: Switches the selected cyclic handler with the cyclic handler immediately below

(3) [Note]

All setting items in this page become invalid when cre_cyc is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.37 is displayed in [Note].

Table 10.37 [Note] in [Cyclic Handler] Page

Condition	Display Message
cre_cyc is not selected	All setting items in this page are invalid because the setting to install cre_cyc is not made.

10.7.48 [Modification of Cyclic Handler Information] Dialog Box

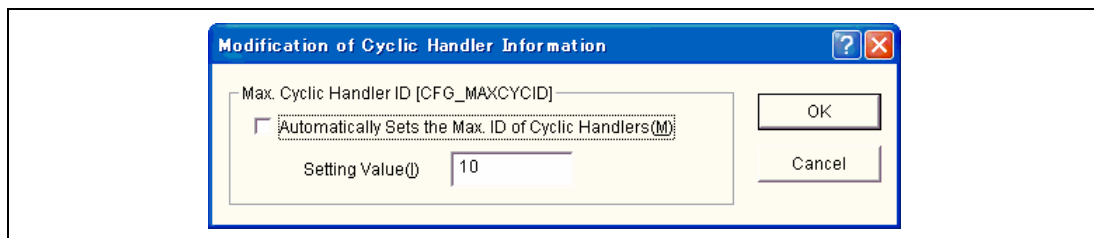


Figure 10.52 [Modification of Cyclic Handler Information] Dialog Box

Clicking the [Modify] button in the [Cyclic Handler] page opens this dialog box.

(1) [Max. Cyclic Handler ID [CFG_MAXCYCID]]

A cyclic handler ID between 1 and CFG_MAXCYCID can be used. An integer between 0 and 254 can be specified. If 0 is specified, cyclic handlers cannot be used. However, since a variable area for managing the cyclic handlers does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Cyclic Handlers] is selected, the configurator automatically calculates the maximum ID based on the cyclic handlers created in the [Cyclic Handler] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.49 [Creation of Cyclic Handler] Dialog Box and [Modification of Information for Cyclic Handler Creation] Dialog Box

Creation of Cyclic Handler

Cyclic Handler ID

☒ Automatic Assignment of ID Number(X)

ID Number(I)

ID Name(N) ☐ Kernel Side(K)

Handler Address

Address(A)

Extended Information

Information(E)

Initiation Information

Start-up Cycle(Y) Start-up Phase(P)

Attribute

☐ Start Handler after Creation(TA_STA)(I)

☐ Save Start-up Phase(TA_PHS)(S)

Description Language

☒ High-Level Language(TA_HLNG)(H)

☐ Assembly Language(TA_ASM)(M)

Create(C) Cancel

Figure 10.53 [Creation of Cyclic Handler] Dialog Box

Selecting [Create] from the pop-up menu in the [Cyclic Handler] page opens the [Creation of Cyclic Handler] dialog box. Selecting [Modify] from the pop-up menu in the [Cyclic Handler] page opens the [Modification of Information for Cyclic Handler Creation] dialog box. These two dialog boxes have the same configuration.

A cyclic handler can also be dynamically created using the `cre_cyc` or `acre_cyc` service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the cyclic handler ID as a numeric value. A value between 1 and `CFG_MAXCYCID` can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the cyclic handler created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Address]

Specify the address of the cyclic handler as a C-language symbol or numeric value.

(6) [Extended Information]

The extended information is passed to the cyclic handler as a parameter. Specify it as a C-language symbol or numeric value.

(7) [Start-up Cycle] and [Start-up Phase]

Specify the initiation cycle and initiation phase for the cyclic handler. A value between 1 and 0x7fffffff can be specified for each. However, initiation cycle \geq initiation phase must be satisfied.

In the case of $CFG_TICDENO > 1$, these values must not exceed $0x7fffffff/CFG_TICDENO$. Since the configurator does not check this error, the specifications must be made carefully.

(8) [Start Handler after Creation (TA_STA)]

If this check box is selected, the cyclic handler enters the operating state at kernel initiation.

(9) [Save Start-up Phase (TA_PHS)]

If this check box is selected, the initiation phase is saved even when the cyclic handler is not operating.

(10) [Description Language]

Select [High-Level Language (TA_HLNG)] when the cyclic handler is written in a high-level language, and select [Assembly Language (TA_ASM)] when the cyclic handler is written in assembly language.

(11) [Create] button (only in [Creation of Cyclic Handler] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next cyclic handler can be created without break. This dialog box is not closed.

(12) [OK] button (only in [Modification of Information for Cyclic Handler Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(13) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.50 [Alarm Handler] Page

In this page, set items related to an alarm handler.

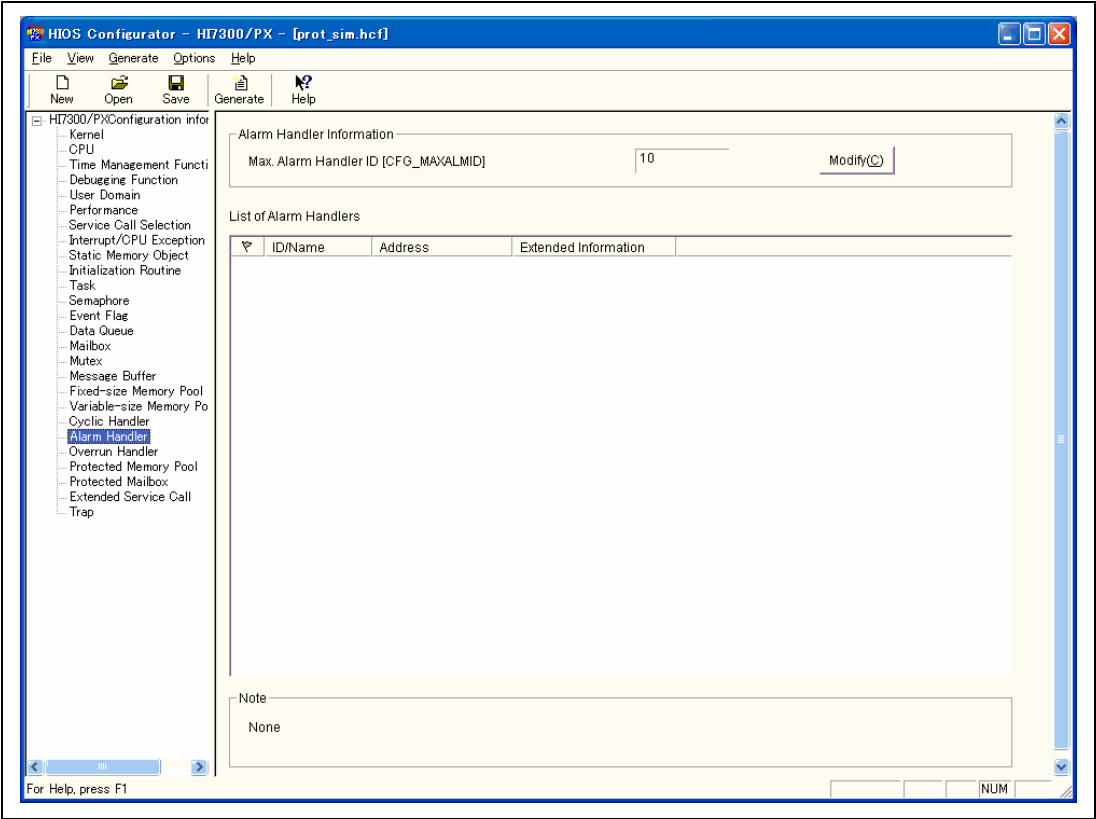


Figure 10.54 [Alarm Handler] Page

Table 10.38 lists the [Alarm Handler] page items.

Table 10.38 [Alarm Handler] Page Items

Item	CFG Name	Linkage Unit
Max. Alarm Handler ID	CFG_MAXALMID	Kernel environment side
Creation of Alarm Handler	—	Kernel side/kernel environment side

(1) [Alarm Handler Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Alarm Handler Information] dialog box in which the information can be changed.

(a) [Max. Alarm Handler ID [CFG_MAXALMID]]

The range of usable alarm handler IDs is between 1 and CFG_MAXALMID.

(2) [List of Alarm Handlers] group

In this group, the alarm handlers already created are displayed. The flag icon indicates that the alarm handler is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Alarm Handler] dialog box to create an alarm handler
- [Delete]: Deletes the selected alarm handler
- [Modify]: Opens the [Modification of Information for Alarm Handler Creation] dialog box to modify the selected alarm handler setting

(3) [Note]

All setting items in this page become invalid when cre_alm is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.39 is displayed in [Note].

Table 10.39 [Note] in [Alarm Handler] Page

Condition	Display Message
cre_alm is not selected	All setting items in this page are invalid because the setting to install cre_alm is not made.

10.7.51 [Modification of Alarm Handler Information] Dialog Box

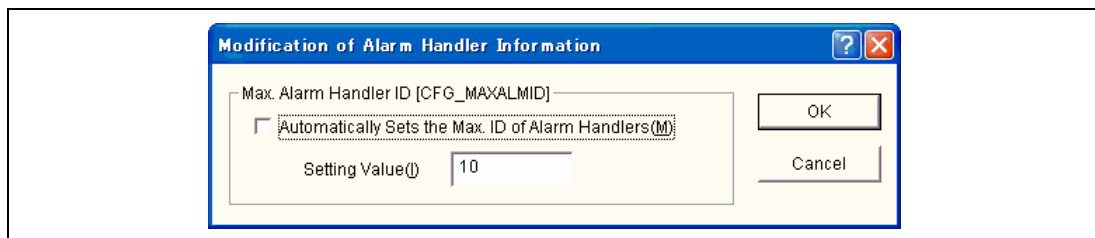


Figure 10.55 [Modification of Alarm Handler Information] Dialog Box

Clicking the [Modify] button in the [Alarm Handler] page opens this dialog box.

(1) [Max. Alarm Handler ID [CFG_MAXALMID]]

An alarm handler ID between 1 and CFG_MAXALMID can be used. An integer between 0 and 255 can be specified. If 0 is specified, alarm handlers cannot be used. However, since a variable area for managing the alarm handlers does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Alarm Handlers] is selected, the configurator automatically calculates the maximum ID based on the alarm handlers created in the [Alarm Handler] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.52 [Creation of Alarm Handler] Dialog Box and [Modification of Information for Alarm Handler Creation] Dialog Box

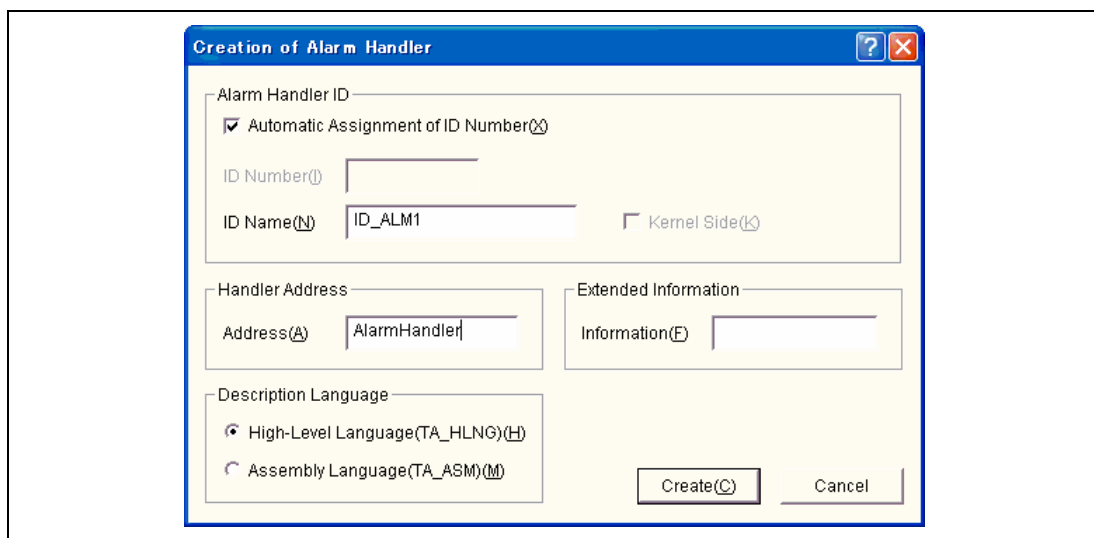


Figure 10.56 [Creation of Alarm Handler] Dialog Box

Selecting [Create] from the pop-up menu in the [Alarm Handler] page opens the [Creation of Alarm Handler] dialog box. Selecting [Modify] from the pop-up menu in the [Alarm Handler] page opens the [Modification of Information for Alarm Handler Creation] dialog box. These two dialog boxes have the same configuration.

An alarm handler can also be dynamically created using the cre_alm or acre_alm service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the alarm handler ID as a numeric value. A value between 1 and CFG_MAXALMID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the alarm handler created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Address]

Specify the address of the alarm handler as a C-language symbol or numeric value.

(6) [Extended Information]

The extended information is passed to the alarm handler as a parameter. Specify it as a C-language symbol or numeric value.

(7) [Description Language]

Select [High-Level Language (TA_HLNG)] when the alarm handler is written in a high-level language, and select [Assembly Language (TA_ASM)] when the alarm handler is written in assembly language.

(8) [Create] button (only in [Creation of Alarm Handler] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next alarm handler can be created without break. This dialog box is not closed.

(9) [OK] button (only in [Modification of Information for Alarm Handler Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(10) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.53 [Overrun Handler] Page

In this page, define the overrun handler. The overrun handler can also be dynamically defined using the `def_ovr` service call.

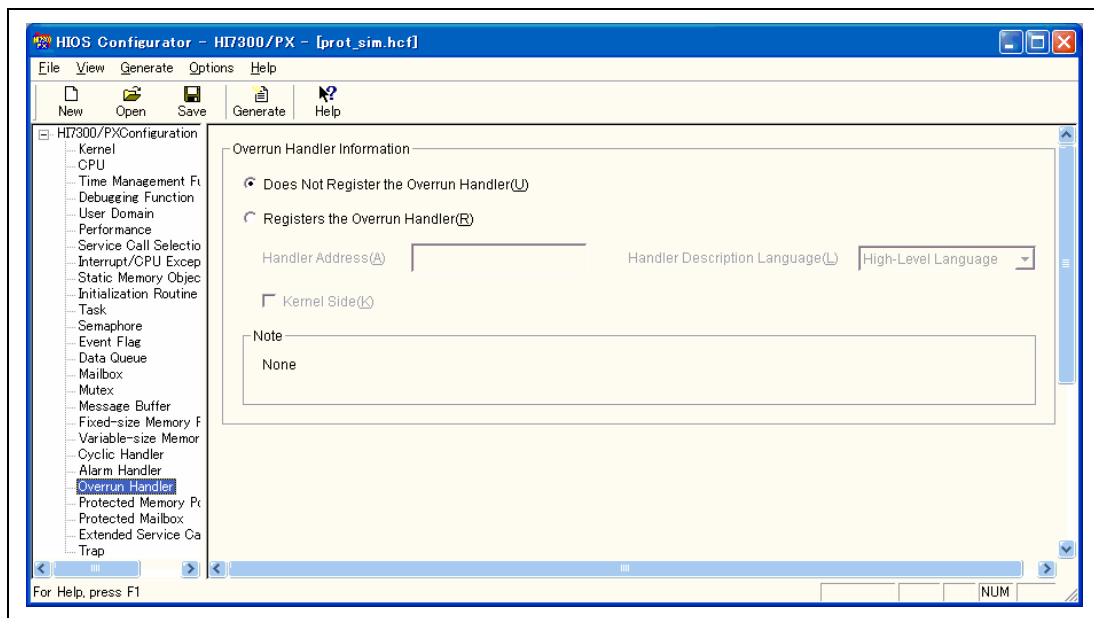


Figure 10.57 [Overrun Handler] Page

(1) [Does Not Register the Overrun Handler] or [Registers the Overrun Handler]

To register the overrun handler, select [Registers the Overrun Handler]. In this case, the subsequent items become valid.

(2) [Handler Address]

Specify the address of the overrun handler as a C-language symbol or numeric value.

(3) [Handler Description Language]

Select [High-Level Language] when the overrun handler is written in a high-level language, and select [Assembly Language] when the overrun handler is written in assembly language.

(4) [Kernel Side]

Select this check box when specifying the overrun handler created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Note]

All setting items in this page become invalid when `def_ovr` is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.40 is displayed in [Note].

Table 10.40 [Note] in [Overrun Handler] Page

Condition	Display Message
def_ovr is not selected	All setting items in this page are invalid because the setting to install def_ovr is not made.

10.7.54 [Protected Memory Pool] Page

In this page, set items related to a protected memory pool.

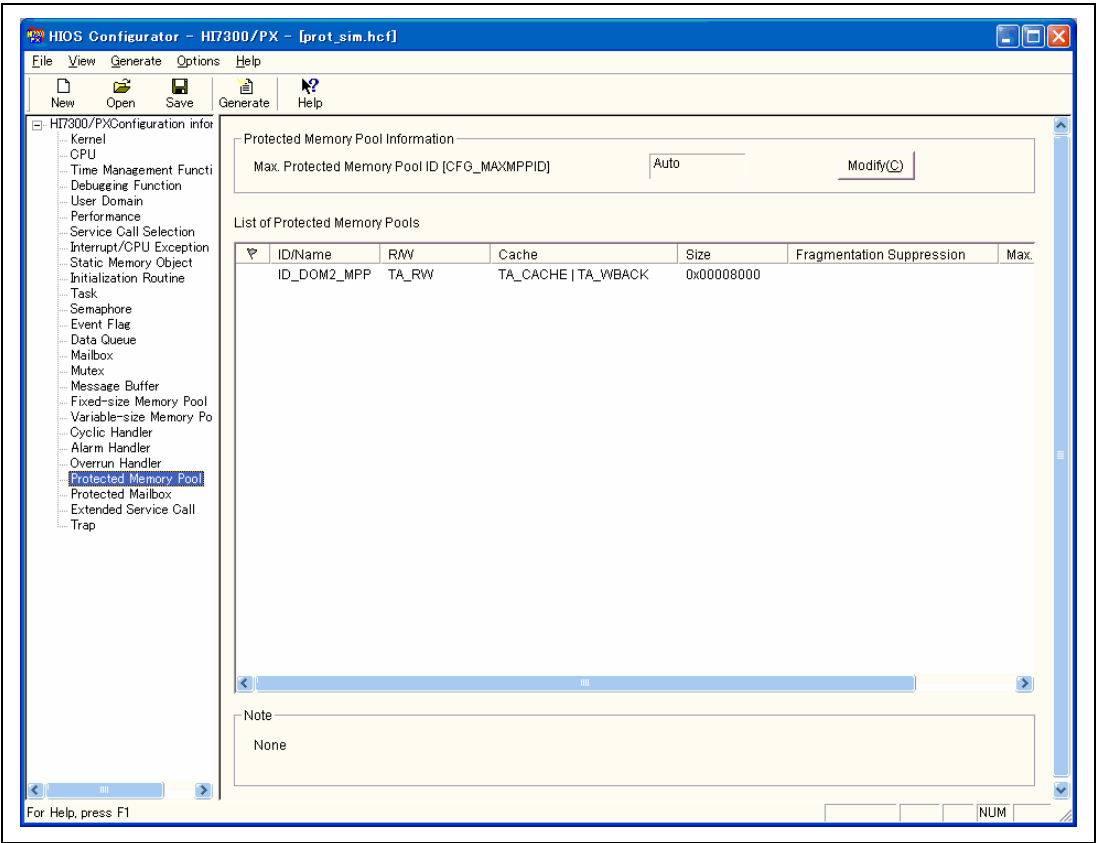


Figure 10.58 [Protected Memory Pool] Page

Table 10.41 lists the [Protected Memory Pool] page items.

Table 10.41 [Protected Memory Pool] Page Items

Item	CFG Name	Linkage Unit
Max. Protected Memory Pool ID	CFG_MAXMPPID	Kernel environment side
Creation of Protected Memory Pool	—	Kernel side/kernel environment side

(1) [Protected Memory Pool Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Protected Memory Pool Information] dialog box in which the information can be changed.

(a) [Max. Protected Memory Pool ID [CFG_MAXMPPID]]

The range of usable protected memory pool IDs is between 1 and CFG_MAXMPPID.

(2) [List of Protected Memory Pools] group

In this group, the protected memory pools already created are displayed. The flag icon indicates that the protected memory pool is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Protected Memory Pool] dialog box to create a protected memory pool
- [Delete]: Deletes the selected protected memory pool
- [Modify]: Opens the [Modification of Information for Protected Memory Pool Creation] dialog box to modify the selected protected memory pool setting

(3) [Note]

All setting items in this page become invalid when CFG_PROTMEM is not selected in the [Kernel] page or icre_mpp is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.42 is displayed in [Note].

Table 10.42 [Note] in [Protected Memory Pool] Page

Condition	Display Message
CFG_PROTMEM is not selected or icre_mpp is not selected	All setting items in this page are invalid because CFG_PROTMEM is not selected or the setting to install icre_mpp is not made.

10.7.55 [Modification of Protected Memory Pool Information] Dialog Box

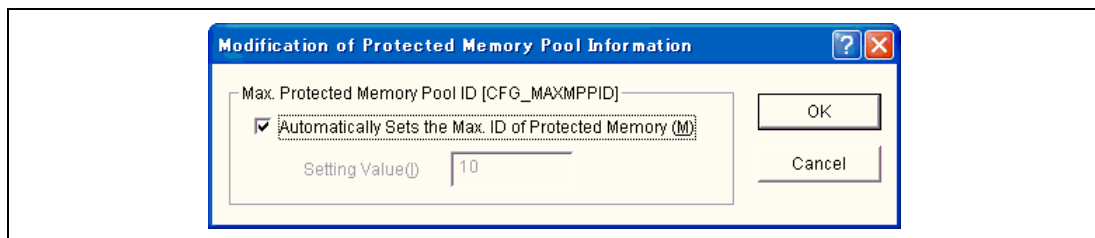


Figure 10.59 [Modification of Protected Memory Pool Information] Dialog Box

Clicking the [Modify] button in the [Protected Memory Pool] page opens this dialog box.

(1) [Max. Protected Memory Pool ID [CFG_MAXMPPID]]

A protected memory pool ID between 1 and CFG_MAXMPPID can be used. An integer between 0 and 31 can be specified. If 0 is specified, protected memory pools cannot be used. However, since a variable area for managing the protected memory pools does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Protected Memory Pools] is selected, the configurator automatically calculates the maximum ID based on the protected memory pools created in the [Protected Memory Pool] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.56 [Creation of Protected Memory Pool] Dialog Box and [Modification of Information for Protected Memory Pool Creation] Dialog Box

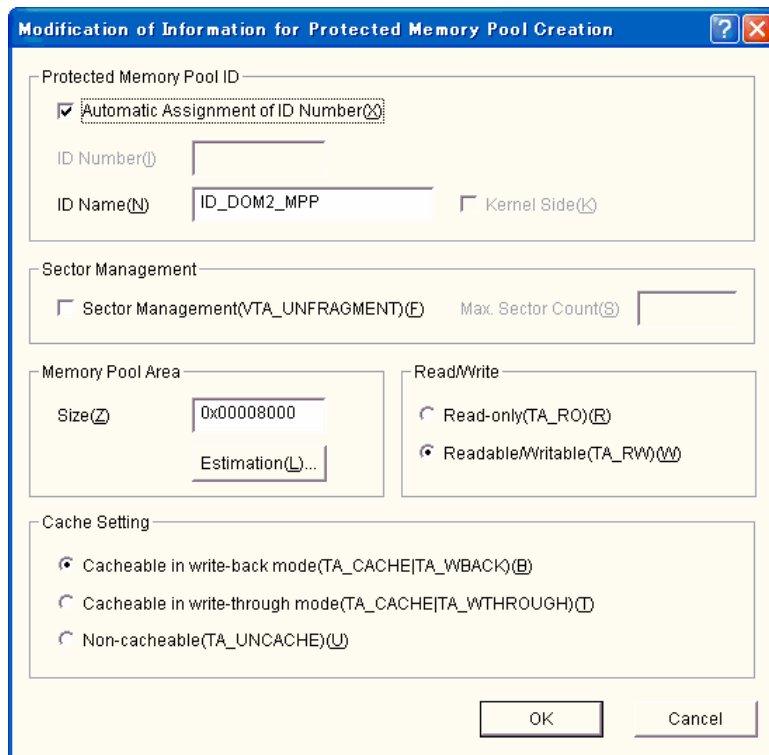


Figure 10.60 [Creation of Protected Memory Pool] Dialog Box

Selecting [Create] from the pop-up menu in the [Protected Memory Pool] page opens the [Creation of Protected Memory Pool] dialog box. Selecting [Modify] from the pop-up menu in the [Protected Memory Pool] page opens the [Modification of Information for Protected Memory Pool Creation] dialog box. These two dialog boxes have the same configuration.

(1) [Protected Memory Pool ID] group

(a) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(b) [ID Number]

Enter the protected memory pool ID as a numeric value. A value between 1 and CFG_MAXMPPID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(c) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(d) [Kernel Side]

Select this check box when specifying the protected memory pool created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(2) [Sector Management] group

When [Sector Management (VTA_UNFRAGMENT)] is selected, protected memory pools are managed by sectors. [Max. Sector Count] is valid only in this case.

The sector management method is an attribute suitable for a memory pool that acquires a large quantity of small memory blocks. By using small blocks continuously as much as possible, a continuous free area of a large size can be maintained more easily. For details on sector management and the meaning of [Max. Sector Count], refer to the following.

Reference: Section 4.31, Controlling Memory Fragmentation

In [Max. Sector Count], an integer other than 0 can be specified. If 0 is specified, it is corrected to 1.

When [Max. Sector Count] is greater than [Size]/(4096 × 32), the actual Max. sector count is corrected to [Size]/(4096 × 32) by the kernel.

(3) [Memory Pool Area] group

In [Size], specify the memory pool size. A value between 1 and 0x20000000 can be specified. The specified value is rounded up to a multiple of CFG_PAGESZ (4096).

The protected memory pool area is created with the following section name.

BUCM_himpp_<ID>

<ID> shows the ID name when an ID name is specified, otherwise it shows the decimal notation of the ID number. At linkage, this section must be allocated to an MMU mapped area and at the boundary address of CFG_PAGESZ (4096).

Clicking the [Estimation] button opens the [Estimation of Protected Memory Pool Area Size] dialog box to calculate the estimated value of [Size].

(4) [Read/Write] group

Select read-only or readable/writable for the protected memory pool.

(5) [Cache Setting] group

Select how the protected memory pool is handled when cache is enabled from the following:

- Cacheable in write-back mode (TA_CACHE|TA_WBACK)
- Cacheable in write-through mode (TA_CACHE|TA_WTHROUGH)
- Non-cacheable (TA_UNCACHE)

(6) [Create] button (only in [Creation of Protected Memory Pool] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next protected memory pool can be created without break. This dialog box is not closed.

(7) [OK] button (only in [Modification of Information for Protected Memory Pool Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(8) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.57 [Estimation of Protected Memory Pool Area Size] Dialog Box

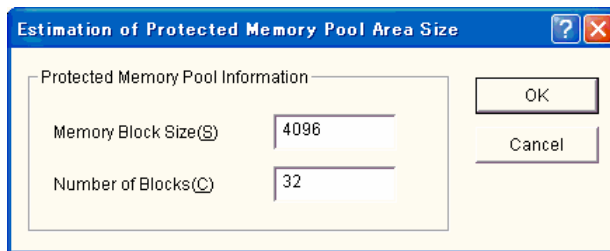


Figure 10.61 [Estimation of Protected Memory Pool Area Size] Dialog Box

Clicking the [Estimation] button in the [Creation of Protected Memory Pool] dialog box or [Modification of Information for Protected Memory Pool Creation] dialog box opens this dialog box.

In this dialog box, the size of the protected memory pool that can acquire a memory block of the size specified in [Memory Block Size] for the number of blocks specified in [Number of Blocks] is calculated. To be specific, the same calculation as the TSZ_MPP macro is performed.

Clicking the [OK] button returns the display to the [Creation of Protected Memory Pool] dialog box or [Modification of Information for Protected Memory Pool Creation] dialog box, and [Size] of the protected memory pool area in either of these dialog boxes is updated to the calculation result of this dialog box.

Clicking the [Cancel] button returns the display to the [Creation of Protected Memory Pool] dialog box or [Modification of Information for Protected Memory Pool Creation] dialog box, but [Size] of the protected memory pool area in either of these dialog boxes is not updated.

10.7.58 [Protected Mailbox] Page

In this page, set items related to a protected mailbox.

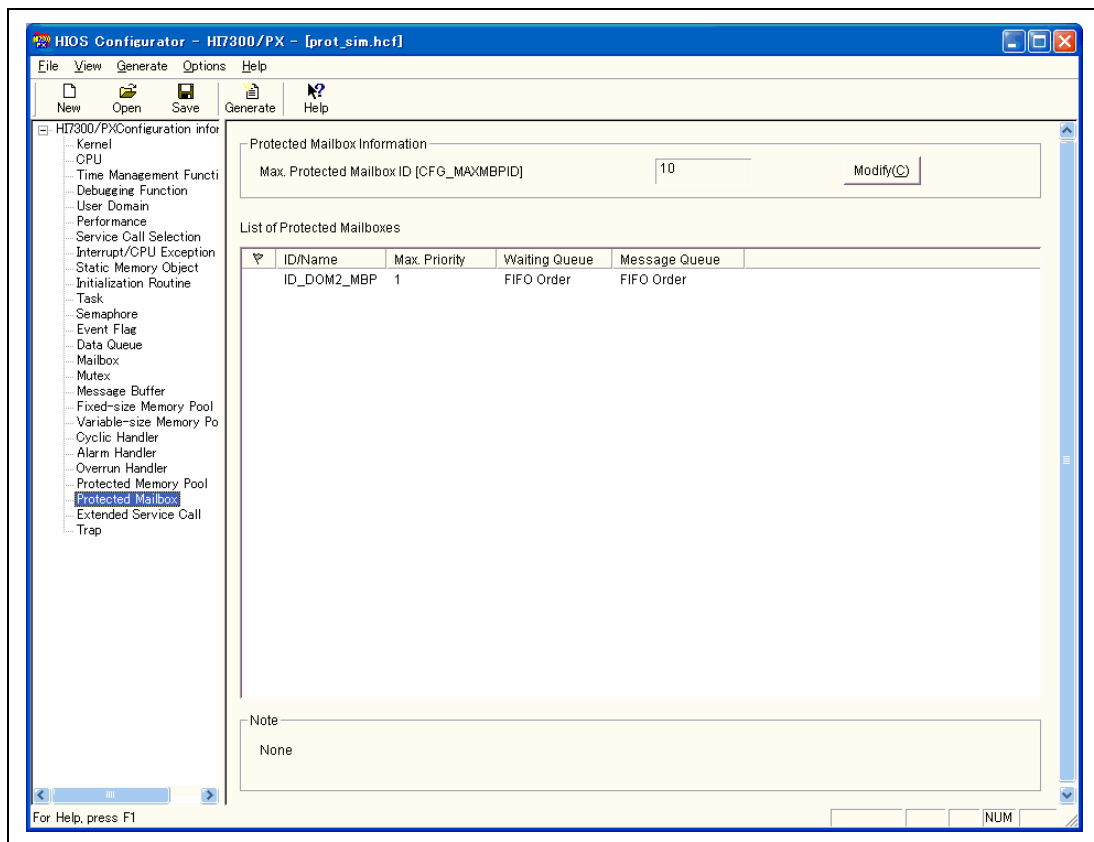


Figure 10.62 [Protected Mailbox] Page

Table 10.43 lists the [Protected Mailbox] page items.

Table 10.43 [Protected Mailbox] Page Items

Item	CFG Name	Linkage Unit
Max. Protected Mailbox ID	CFG_MAXMBPID	Kernel environment side
Creation of Protected Mailbox	—	Kernel side/kernel environment side

(1) [Protected Mailbox Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Protected Mailbox Information] dialog box in which the information can be changed.

(a) [Max. Protected Mailbox ID [CFG_MAXMBPID]]

The range of usable protected mailbox IDs is between 1 and CFG_MAXMBPID.

(2) [List of Protected Mailboxes] group

In this group, the protected mailboxes already created are displayed. The flag icon indicates that the protected mailbox is created to belong to the kernel side.

The following items are in the pop-up menu.

- [Create]: Opens the [Creation of Protected Mailbox] dialog box to create a protected mailbox
- [Delete]: Deletes the selected protected mailbox
- [Modify]: Opens the [Modification of Information for Protected Mailbox Creation] dialog box to modify the selected protected mailbox setting

(3) [Note]

All setting items in this page become invalid when CFG_PROTMEM is not selected in the [Kernel] page or cre_mbp is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.44 is displayed in [Note].

Table 10.44 [Note] in [Protected Mailbox] Page

Condition	Display Message
CFG_PROTMEM is not selected or cre_mbp is not selected	All setting items in this page are invalid because CFG_PROTMEM is not selected or the setting to install cre_mbp is not made.

10.7.59 [Modification of Protected Mailbox Information] Dialog Box

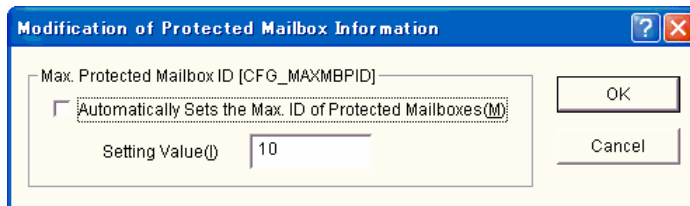


Figure 10.63 [Modification of Protected Mailbox Information] Dialog Box

Clicking the [Modify] button in the [Protected Mailbox] page opens this dialog box.

(1) [Max. Protected Mailbox ID [CFG_MAXMBPID]]

A protected mailbox ID between 1 and CFG_MAXMBPID can be used. An integer between 0 and 32767 can be specified. If 0 is specified, protected mailboxes cannot be used. However, since a variable area for managing the protected mailboxes does not need to be allocated, the used RAM size can be reduced.

If [Automatically Sets the Max. ID of Protected Mailboxes] is selected, the configurator automatically calculates the maximum ID based on the protected mailboxes created in the [Protected Mailbox] page.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.60 [Creation of Protected Mailbox] Dialog Box and [Modification of Information for Protected Mailbox Creation] Dialog Box

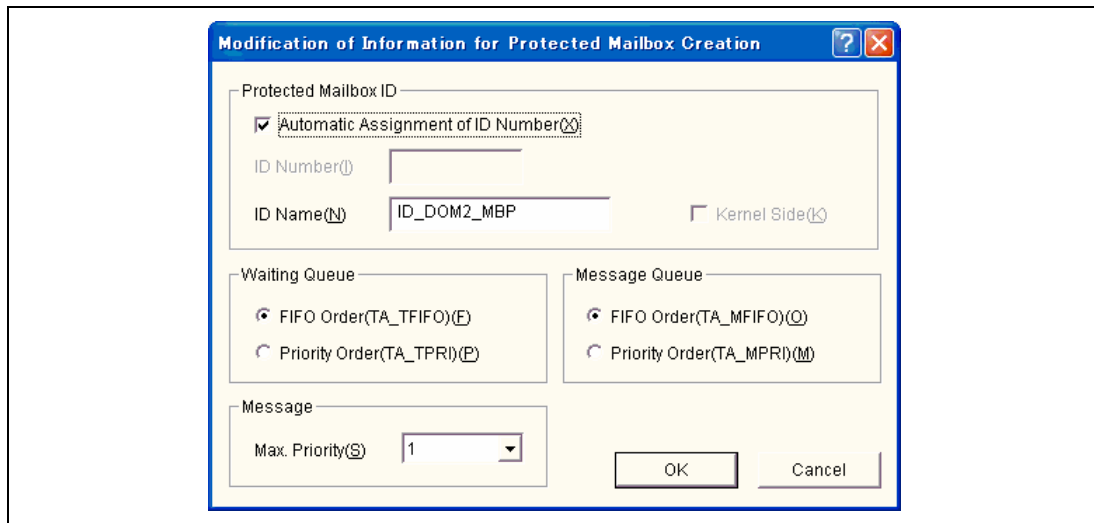


Figure 10.64 [Creation of Protected Mailbox] Dialog Box

Selecting [Create] from the pop-up menu in the [Protected Mailbox] page opens the [Creation of Protected Mailbox] dialog box. Selecting [Modify] from the pop-up menu in the [Protected

Mailbox] page opens the [Modification of Information for Protected Mailbox Creation] dialog box. These two dialog boxes have the same configuration.

A protected mailbox can also be dynamically created using the cre_mbp or acre_mbp service call.

(1) [Automatic Assignment of ID Number]

If this check box is selected, the configurator automatically assigns an ID number. However, if this check box is selected, [Kernel Side] cannot be selected.

(2) [ID Number]

Enter the protected mailbox ID as a numeric value. A value between 1 and CFG_MAXMBPID can be specified. However, if [Automatic Assignment of ID Number] is selected, the ID number cannot be specified.

(3) [ID Name]

Specify the ID name. If [Automatic Assignment of ID Number] is selected, the name must be specified. In other cases, this edit box can be left blank.

(4) [Kernel Side]

Select this check box when specifying the protected mailbox created to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(5) [Waiting Queue]

Select the FIFO order or priority order as the method to queue the waiting tasks.

(6) [Message Queue]

Select the FIFO order or priority order as the method to queue the messages.

(7) [Max. Priority]

When the priority order is selected in [Message Queue], select the maximum priority of the message. It can be selected between 1 and CFG_MAXMSGPRI.

When the FIFO order is selected in [Message Queue], this item has no meaning.

(8) [Create] button (only in [Creation of Protected Mailbox] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next protected mailbox can be created without break. This dialog box is not closed.

(9) [OK] button (only in [Modification of Information for Protected Mailbox Creation] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(10) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.61 [Extended Service Call] Page

In this page, set items related to an extended service call.

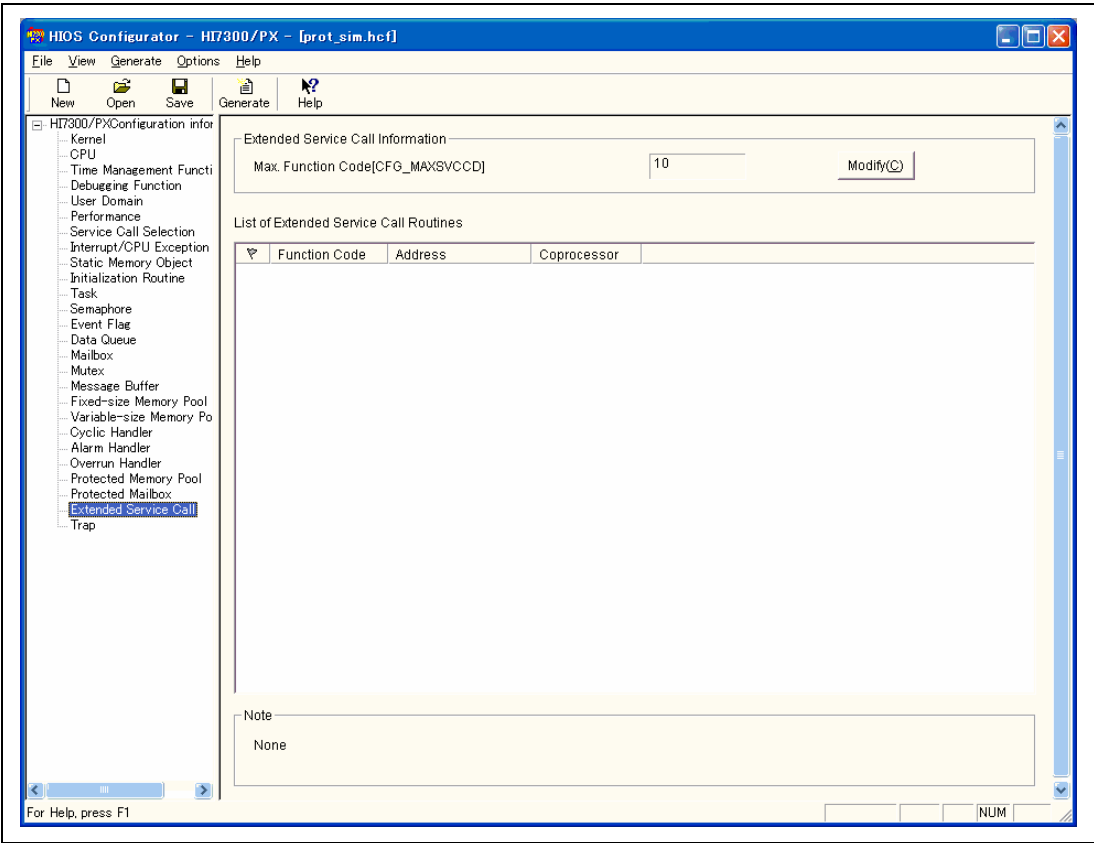


Figure 10.65 [Extended Service Call] Page

Table 10.45 lists the [Extended Service Call] page items.

Table 10.45 [Extended Service Call] Page Items

Item	CFG Name	Linkage Unit
Max. Function Code	CFG_MAXSVCCD	Kernel environment side
Definition of Extended Service Call Routine	—	Kernel side/kernel environment side

(1) [Extended Service Call Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Extended Service Call Information] dialog box in which the information can be changed.

(a) [Max. Function Code [CFG_MAXSVCCD]]

The range of usable function codes is between 1 and CFG_MAXSVCCD.

(2) [List of Extended Service Call Routines] group

In this group, the extended service call routines already defined are displayed. The flag icon indicates that the routine is defined to belong to the kernel side.

The following items are in the pop-up menu. In kernel lock mode, these pop-up menu items cannot be selected for the routines on the kernel side.

- [Define]: Opens the [Definition of Extended Service Call Routine] dialog box to define an extended service call routine
- [Delete]: Cancels definition of the selected extended service call routine
- [Modify]: Opens the [Modification of Information for Extended Service Call Routine Definition] dialog box to modify the selected extended service call routine setting

(3) [Note]

All setting items in this page become invalid when def_svc is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.46 is displayed in [Note].

Table 10.46 [Note] in [Extended Service Call] Page

Condition	Display Message
def_svc is not selected	All setting items in this page are invalid because the setting to install def_svc is not made.

10.7.62 [Modification of Extended Service Call Information] Dialog Box

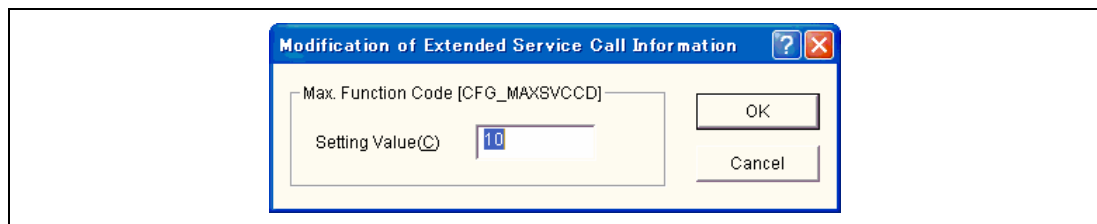


Figure 10.66 [Modification of Extended Service Call Information] Dialog Box

Clicking the [Modify] button in the [Extended Service Call] page opens this dialog box.

(1) [Max. Function Code [CFG_MAXSVCCD]]

Function codes between 1 and CFG_MAXSVCCD can be used. An integer between 0 and 32767 can be specified. If 0 is specified, extended service calls cannot be used. However, since a variable area for managing the extended service calls does not need to be allocated, the used RAM size can be reduced.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.63 [Definition of Extended Service Call Routine] Dialog Box and [Modification of Information for Extended Service Call Routine Definition] Dialog Box

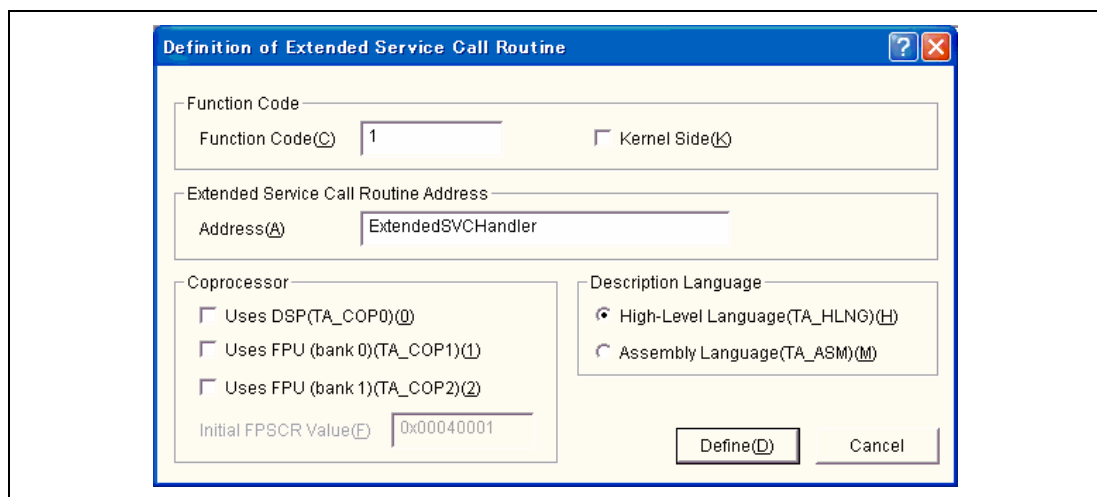


Figure 10.67 [Definition of Extended Service Call Routine] Dialog Box

Selecting [Define] from the pop-up menu in the [Extended Service Call] page opens the [Definition of Extended Service Call Routine] dialog box. Selecting [Modify] from the pop-up menu in the [Extended Service Call] page opens the [Modification of Information for Extended Service Call Routine Definition] dialog box. These two dialog boxes have the same configuration.

An extended service call can also be dynamically defined using the `def_svc` service call.

(1) [Function Code]

Enter the function code as a numeric value. A value between 1 and `CFG_MAXSVCCD` can be specified.

(2) [Kernel Side]

Select this check box when specifying the extended service call routine defined to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(3) [Extended Service Call Routine Address]

Specify the address of the extended service call routine as a C-language symbol or numeric value.

(4) [Uses DSP (TA_COP0)], [Uses FPU (bank 0) (TA_COP1)], or [Uses FPU (bank 1) (TA_COP2)]

TA_COP0 is valid only when CFG_DSP is selected in the [CPU] page. Select this check box when performing DSP calculation.

TA_COP1 or TA_COP2 is valid only when CFG_FPU is selected in the [CPU] page. For normal FPU calculation, select only TA_COP1. For cases using both FPU banks, i.e. matrix calculation, select both TA_COP1 and TA_COP2. Selecting only TA_COP2 and not TA_COP1 is not possible.

TA_COP0 cannot be selected together with TA_COP1 or TA_COP2.

(5) [Initial FPSCR Value]

The initial FPSCR value has a meaning only when either TA_COP1 or TA_COP2 is selected. An integer between 0 and 0xffffffff can be specified. Specify this value with reference to the following.

Reference: Section 15, Notes on FPU

(6) [Description Language]

Select [High-Level Language (TA_HLNG)] when the extended service call routine is written in a high-level language, and select [Assembly Language (TA_ASM)] when the extended service call routine is written in assembly language.

(7) [Define] button (only in [Definition of Extended Service Call Routine] dialog box)

Makes the settings in this dialog box effective. Then, returns the display of this dialog box to its initial state so that the next extended service call routine can be defined without break. This dialog box is not closed.

(8) [OK] button (only in [Modification of Information for Extended Service Call Routine Definition] dialog box)

Closes this dialog box after making the settings in this dialog box effective.

(9) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.64 [Trap] Page

In this page, set items related to a trap.

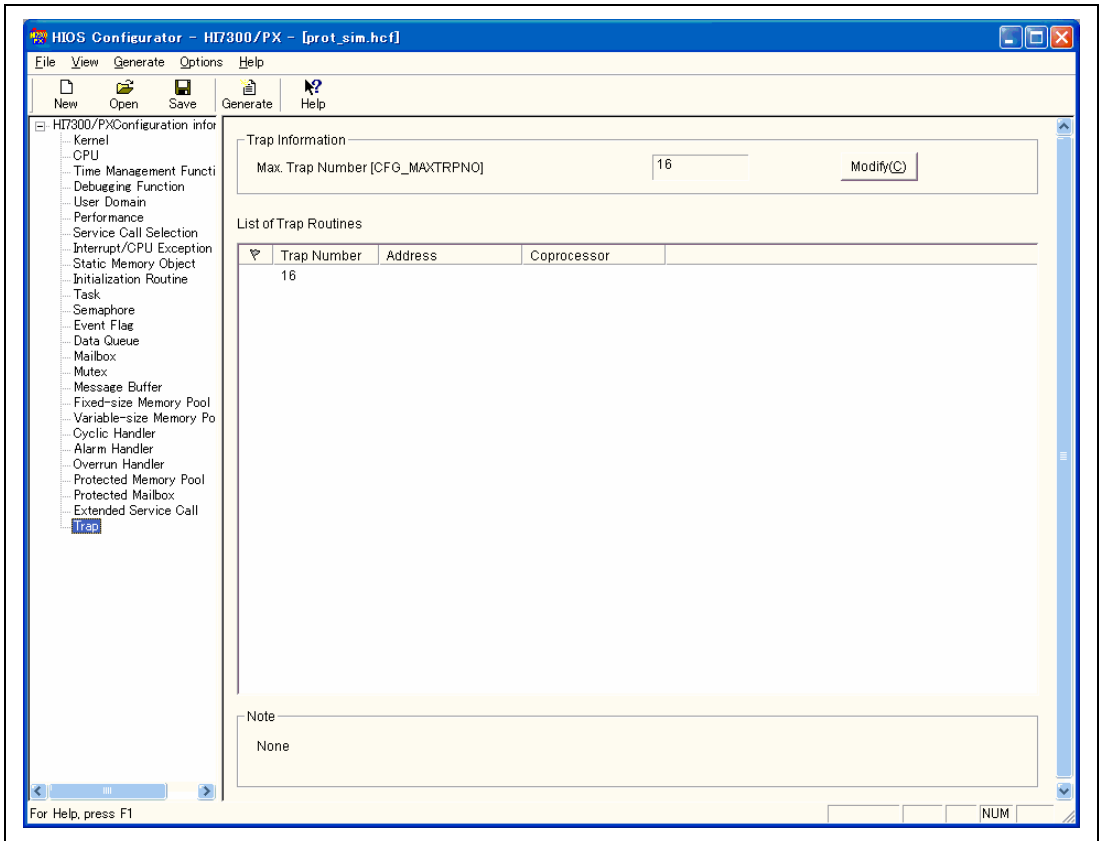


Figure 10.68 [Trap] Page

Table 10.47 lists the [Trap] page items.

Table 10.47 [Trap] Page Items

Item	CFG Name	Linkage Unit
Max. Trap Number	CFG_MAXTRPNO	Kernel environment side
Definition of Trap Routine	—	Kernel side/kernel environment side

(1) [Trap Information] group

In this group, the following information is displayed. Clicking the [Modify] button opens the [Modification of Trap Information] dialog box in which the information can be changed.

(a) [Max. Trap Number [CFG_MAXTRPNO]]

The range of usable trap numbers is between 16 and CFG_MAXTRPNO.

(2) [List of Trap Routines] group

In this group, the trap routines already defined are displayed. The flag icon indicates that the routine is defined to belong to the kernel side.

The following items are in the pop-up menu. In kernel lock mode, these pop-up menu items cannot be selected for the routines on the kernel side.

- [Define]: Opens the [Definition of Trap Routine] dialog box to define a trap routine
- [Cancel]: Cancels definition of the selected trap routine

(3) [Note]

All setting items in this page become invalid when vdef_trp is not selected in the [Service Call Selection] page. In addition, all items in this page cannot be modified.

In this case, the message shown in table 10.48 is displayed in [Note].

Table 10.48 [Note] in [Trap] Page

Condition	Display Message
vdef_trp is not selected	All setting items in this page are invalid because the setting to install vdef_trp is not made.

10.7.65 [Modification of Trap Information] Dialog Box

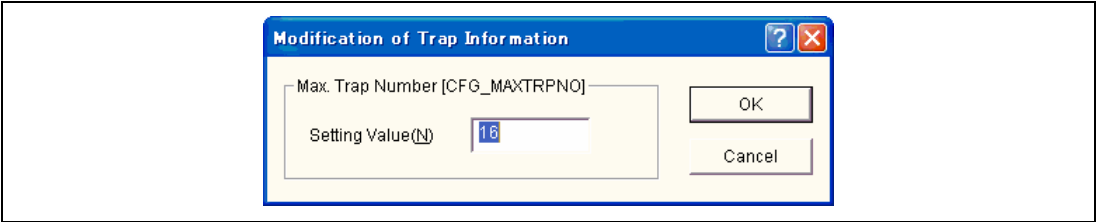


Figure 10.69 [Modification of Trap Information] Dialog Box

Clicking the [Modify] button in the [Trap] page opens this dialog box.

(1) [Max. Trap Number [CFG_MAXTRPNO]]

A trap number between 16 and CFG_MAXTRPNO can be used. An integer between 16 and 255 can be specified.

(2) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(3) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.7.66 [Definition of Trap Routine] Dialog Box

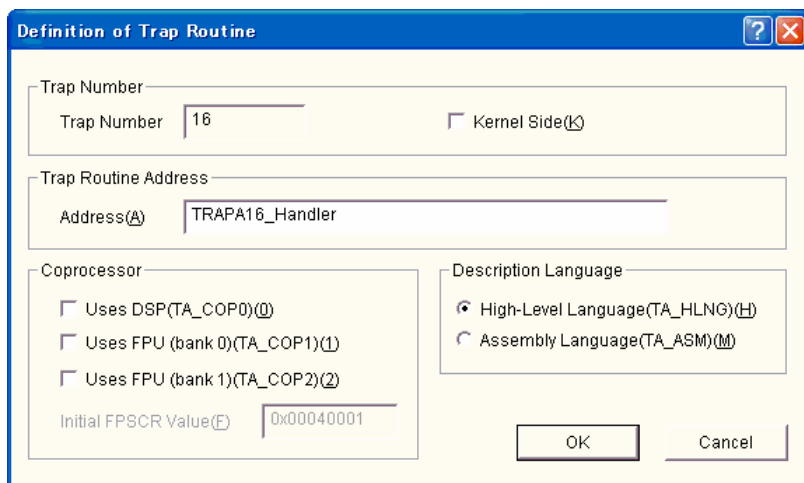


Figure 10.70 [Definition of Trap Routine] Dialog Box

Selecting [Define] from the pop-up menu in the [Trap] page opens the [Definition of Trap Routine] dialog box.

A trap can also be dynamically defined using the vdef_trp service call.

(1) [Trap Number]

The trap number selected in the [Trap] page is displayed.

(2) [Kernel Side]

Select this check box when specifying the trap routine defined to belong to the kernel side.

In kernel lock mode, this check box cannot be selected at all times.

(3) [Trap Routine Address]

Specify the address of the trap routine as a C-language symbol or numeric value.

(4) [Uses DSP (TA_COP0)], [Uses FPU (bank 0) (TA_COP1)], or [Uses FPU (bank 1) (TA_COP2)]

TA_COP0 is valid only when CFG_DSP is selected in the [CPU] page. Select this check box when performing DSP calculation.

TA_COP1 or TA_COP2 is valid only when CFG_FPU is selected in the [CPU] page. For normal FPU calculation, select only TA_COP1. For cases using both FPU banks, i.e. matrix calculation, select both TA_COP1 and TA_COP2. Selecting only TA_COP2 and not TA_COP1 is not possible.

TA_COP0 cannot be selected together with TA_COP1 or TA_COP2.

(5) [Initial FPSCR Value]

The initial FPSCR value has a meaning only when either TA_COP1 or TA_COP2 is selected. An integer between 0 and 0xffffffff can be specified. Specify this value with reference to the following.

Reference: Section 15, Notes on FPU

(6) [Description Language]

Select [High-Level Language (TA_HLNG)] when the trap routine is written in a high-level language, and select [Assembly Language (TA_ASM)] when the trap routine is written in assembly language.

(7) [OK] button

Closes this dialog box after making the settings in this dialog box effective.

(8) [Cancel] button

Closes this dialog box without saving the settings in this dialog box.

10.8 Edit Box Specifications

Number of Characters that Can be Input: The number of characters that can be entered in an edit box is up to 255 bytes. An input exceeding this limit is not accepted.

Characters that Can be Input:

(1) Identification of numeric values and character strings

Only the ASCII code should be input to an edit box. Do not input any other character code. The following cases of input are handled as numeric values, and other cases are handled as character strings.

(1) Character string consisting of only 0 to 9: Handled as a decimal number

(2) Character string starting with "0x" or "0X", and the subsequent one to eight characters consist of only 0 to 9, a to f, and A to F: Handled as a hexadecimal number

For an input handled as a numeric value, which cannot be represented with 32 bits, the following error message is displayed.

Numeric value exceeding 0xffffffff cannot be input

In the above case (2), if there are nine or more characters following "0x" or "0X", the above error message is also displayed.

Examples are shown below.

— 123: Handled as a decimal number of 123

— 0x1000: Handled as a hexadecimal number of 0x1000 (4096)

— 0x012345678: Error message "Numeric value exceeding 0xffffffff cannot be input" is displayed

— 0x0123Z: Handled as a character string

(2) Character string

In the configurator, only the following have meaning as a character string.

— ID name (C-language macro name)

— C-language symbol

— Section name

Whether the C-language grammar in these inputs is correct or not is not tested. If not correct, a grammar error is detected at compilation.

Blank: If there is no input in the edit boxes shown in table 10.49, the default setting is used.

If other edit boxes are left blank, one of the following error messages will be displayed.

Input a value

Input a character string

Input a value or character string

Table 10.49 Edit Boxes Allowed to be Blank

Page/Dialog Box	Edit Box	Condition for Allowing Blank	Default Setting
[Creation of Task] and [Modification of Information for Task Creation] dialog boxes	Extended information	Always	Handled as 0
[Registration of Initialization Routine] and [Modification of Information for Initialization Routine Registration] dialog boxes			
[Creation of Cyclic Handler] and [Modification of Information for Cyclic Handler Creation] dialog boxes			
[Creation of Alarm Handler] and [Modification of Information for Alarm Handler Creation] dialog boxes			
[Definition of Task Exception Processing Routine] dialog box	Address	Always	Definition is canceled
[Creation] and [Modification of Information for Creation] dialog boxes for objects	ID name	[Automatic Assignment of ID Number] is not selected	Handled as an object with no name

Duplication of ID Name, C-Language Symbol, or Section Name: The following error message is displayed in the cases below.

Specified symbol (name) already used. Specify another symbol (name).

- Input of ID name: The specified ID name is the same as an ID name, C-language symbol, or section name, which is already registered.
- Input of C-language symbol: The specified C-language symbol is the same as an ID name or section name, which is already registered.
- Input of section name: The specified section name is the same as an ID name, C-language symbol, or section name, which is already registered.

10.9 Tuning

10.9.1 Reduction of Used RAM Size

Table 10.50 lists the configurator setting items which are related to the used RAM size.

Table 10.50 Reduction of Used RAM Size

Page	Item	Relevant Section Name	Method to Reduce Size
[Kernel] page	CFG_NTSKSTKSZ	BSCP_hintskstk	Make it small
	CFG_RESPOOLSZ	BSCP_hirespl	Make it small
	CFG_SYSPOOLSZ	BSCP_hisyspl	Make it small
	CFG_PROTMEM	BSCP_hidef, BSCP_hicfg, BSCP_hiwrk	Do not select
[Debugging Function] page	CFG_ACTION	BSCP_hiwrk	Do not select
	CFG_TRACE	BSCP_hiwrk	Do not select
	CFG_TRCOBJCNT	BSCP_hiwrk	Make it small
	CFG_TRCBUFSZ	BSCP_hitrcbuf	Make it small
[Time Management Function] page	CFG_OPTTMR	BSCP_hiwrk	Do not select
[Performance] page	CFG_PERFORM	BSCP_hiwrk	Do not select
[Service Call Selection] page	vset_tfl, vclr_tfl, vwai_tfl, vtwai_tfl, vpwai_tfl	BSCP_hiwrk	Do not select any
	def_tex	BSCP_hiwrk	Do not select
	cre_sem	BSCP_hiwrk	Do not select
	cre_flg	BSCP_hiwrk	Do not select
	cre_dtq	BSCP_hiwrk	Do not select
	cre_mbx	BSCP_hiwrk	Do not select
	cre_mtx	BSCP_hiwrk	Do not select
	cre_mbf	BSCP_hiwrk	Do not select
	cre_mpf	BSCP_hiwrk	Do not select
	cre_mpl	BSCP_hiwrk	Do not select
	cre_cyc	BSCP_hiwrk	Do not select

Table 10.50 Reduction of Used RAM Size (cont)

Page	Item	Relevant Section Name	Method to Reduce Size
[Service Call Selection] page (cont)	cre_alm	BSCP_hiwrk	Do not select
	def_ovr	BSCP_hiwrk	Do not select
	icre_mpp	BSCP_hiwrk	Do not select
	cre_mbp	BSCP_hiwrk	Do not select
	def_svc	BSCP_hiwrk	Do not select
	vdef_trp	BSCP_hiwrk	Do not select
[Interrupt/CPU Exception Handler] page	CFG_MAXINTNO	BSCP_hiwrk	Make it small
[Task] page	CFG_MAXTSKID	BSCP_hiwrk	Make it small
	CFG_MAXTSKPRI	BSCP_hiwrk	Make it small
[Semaphore] page	CFG_MAXSEMED	BSCP_hiwrk	Make it small
[Event Flag] page	CFG_MAXFLGID	BSCP_hiwrk	Make it small
[Data Queue] page	CFG_MAXDTQID	BSCP_hiwrk	Make it small
[Mailbox] page	CFG_MAXMBXID	BSCP_hiwrk	Make it small
[Mutex] page	CFG_MAXMTXID	BSCP_hiwrk	Make it small
[Message Buffer] page	CFG_MAXMBFID	BSCP_hiwrk	Make it small
[Fixed-size Memory Pool] page	CFG_MAXMPFID	BSCP_hiwrk	Make it small
[Variable-size Memory Pool] page	CFG_MAXMPLID	BSCP_hiwrk	Make it small
[Cyclic Handler] page	CFG_MAXCYCID	BSCP_hiwrk	Make it small
[Alarm Handler] page	CFG_MAXALMID	BSCP_hiwrk	Make it small
[Protected Memory Pool] page	CFG_MAXMPPID	BSCP_hiwrk	Make it small
[Protected Mailbox] page	CFG_MAXMBPID	BSCP_hiwrk	Make it small
[Extended Service Call] page	CFG_MAXSVCCD	BSCP_hiwrk	Make it small
[Trap] page	CFG_MAXTRPNO	BSCP_hiwrk	Make it small

10.9.2 Reduction of Used ROM Size

The most effective way to reduce the used ROM size is to decrease the number of service calls selected in the [Service Call Selection] page. In particular, not selecting the def_??? or cre_??? service calls has a large effect since their function modules will not be installed.

Table 10.51 lists the configurator setting items which are related to the used ROM size.

Table 10.51 Reduction of Used ROM Size

Page	Item	Relevant Section Name	Method to Reduce Size
[Kernel] page	CFG_PARCHK	PSCP_hiknl	Do not select
	CFG_PROTMEM	PSCP_hiknl, CSCP_hidef, CSCP_hicfg	Do not select
	CFG_MEMCHK	PSCP_hiknl	Do not select
	CFG_MAXLOCPAGE	PSCP_hiknl	Set it to 0
[CPU] page	CFG_DSPSTBY	PSCP_hiknl, PSCP_hidef	Do not select
[Time Management Function] page	CFG_OPTTMR	PSCP_hiknl	Do not select
[Debugging Function] page	CFG_ACTION	PSCP_hiknl	Do not select
	CFG_TRACE	PSCP_hiknl	Do not select
[Performance] page	CFG_PERFORM	PSCP_hiknl	Do not select
All pages	Initial registration of objects	PSCP_hidef, CSCP_hidef, PSCP_hicfg, CSCP_hicfg	Make it small

10.9.3 Performance Improvement

Table 10.52 lists the configurator setting items that affect the performance.

Table 10.52 Performance Improvement

Page	Item	Description
[Kernel] page	CFG_PARCHK	If not selected, the processing time of the service call becomes shorter.
	CFG_KNLLVL	A small size allows more interrupt levels to be accepted even while the kernel is executing a critical section.
	CFG_PROTMEM	If not selected, TLB-miss overhead will not occur.
	CFG_MEMCHK	If not selected, the processing time of the service call with an address parameter becomes shorter.
[Time Management Function] page	CFG_TICNUME, CFG_TICDENO	A large time tick reduces the load caused by a timer interrupt, but precision of time management by the kernel will be degraded.
	CFG_OPTTMR	If selected, the processing time of the service call requesting time management processing has the possibility of becoming longer. However, occurrence of timer interrupts will become less frequent.
[Debugging Function] page	CFG_ACTION	If selected, a cyclic handler with a 100-ms cycle is executed, thus increasing the load.
	CFG_TRACE	If selected, the entire performance is degraded.
	CFG_TRCOBJCNT	A large number degrades the entire performance.
[Performance] page	CFG_PERFORM	If selected, the entire performance is degraded.
[Service Call Selection] page	def_tex	If selected, the task switching time gets longer.

Section 11 Build

This section describes how to create load modules in the absolute address format, which are to be installed in the target system, with referring to the provided sample.

11.1 Load Modules

(1) Load Module Types

The system using the HI7300/PX should consist of the following three types of load modules.

(a) Kernel load module (knl_side)

The kernel load module includes the following.

- Code of the kernel
- Cache support functions
- Statically allocated variable area for the kernel
- Code and data of the applications (when necessary)

(b) Kernel environment load module (env_side)

The kernel environment load module includes the following.

- Statically allocated variable area for the kernel
- System pool
- Resource pool
- Stack area for non-task context
- Code and data of the applications (when necessary)

(c) Application load modules

An application load module consists of applications only. More than one application load module can be created. Alternatively, if all applications are included in the kernel load module or kernel environment load module, there is no need to create any application load module.

Both knl_side and env_side are necessary for kernel operation.

These load modules have the following dependencies.

Kernel load module

--> Kernel environment load module

--> Application load modules

As shown, when the kernel load module is updated, the kernel environment load module and application load modules must be updated. In the same way, when the kernel environment load module is updated, the application load modules must be updated.

This dependency structure has the following advantages.

- After only the kernel load module is stored in ROM, the kernel environment load module and application load modules can be modified.
- By including the target of debugging in the kernel environment load module or an application load module, the build and download time can be reduced in debugging process.

(2) [Kernel Side] Checkbox in the Configurator

When the [Kernel side] checkbox is selected to specify C-language symbols and section names during creation or definition of task addresses or static memory objects through the configurator, the addresses for these symbols and sections must be determined at linkage of the kernel load module. In the same way, when the [Kernel side] checkbox is not selected, the addresses for these symbols and section names must be determined at linkage of the kernel environment load module.

However, to be more exact, the entities of C-language symbols do not always need to be linked if the symbols can be determined at linkage through forced definition of symbol values or symbol file input.

(3) Note on ID Name Header File

To avoid dependency of the kernel load module upon other load modules, the applications included in the kernel load module must not include `kernel_id.h`, which is on the kernel environment side.

(4) Summary

Figure 11.1 gives the summary of load module creation described above.

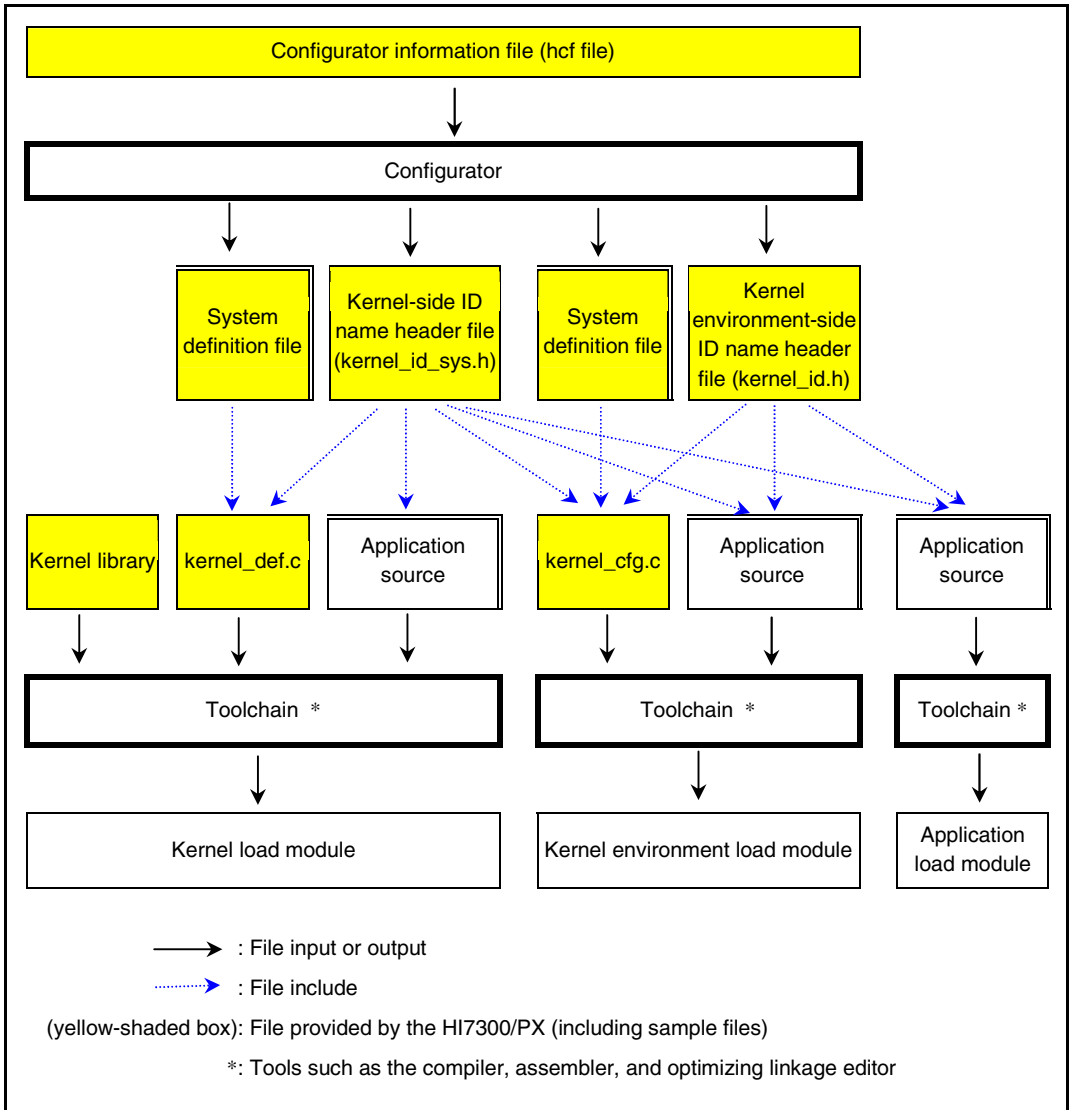


Figure 11.1 Load Module Creation

11.2 Directory Structure

Figure 11.2 shows the structure of the directories where the kernel is installed.

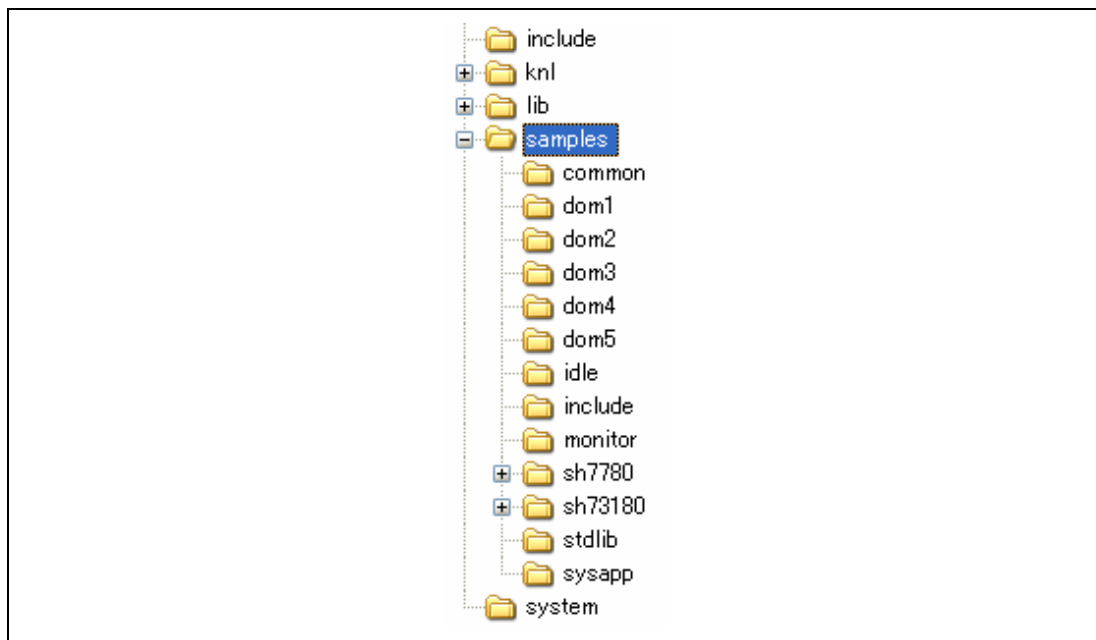


Figure 11.2 Directory Structure

(1) include\

Contains header files such as `itron.h`.

This directory must be specified as an include path when any application is compiled or assembled.

The files in this directory must not be modified.

(2) knl\

Contains the source code of the kernel, which is only provided under the source code license.

(3) lib\

The `elf\` directory under this directory contains the relocatable object files of the kernel library and cache support functions.

(4) system\

Contains the system definition files, which are used to compile the files output by the configurator.

The files in this directory must not be modified.

(5) samples\

The directories under this directory contain the sample files (such as sample programs and HEW workspace).

11.3 Overview of Sample System

11.3.1 Overview

The sample system stored in the directories under samples\ includes the following applications. Each application displays messages in the simulated I/O window through standard library functions when it is executed in the simulator.

- User domain 1 (dom1)
A sample with domain ID = 1.
This sample application performs data communications within a domain by using a fixed-size memory pool and a mailbox.
- User domain 2 (dom2) and user domain 3 (dom3)
Samples with domain ID = 2 and domain ID = 3.
These sample applications perform data communications between domains. When the memory object protection function is selected, a protected memory pool and a protected mailbox are used. When the memory object protection function is not selected, a variable-size memory pool and a mailbox are used.
Domain 2 is the receiver and domain 3 is the sender.
- User domain 4 (dom4)
A sample with domain ID = 4.
This sample application intentionally performs an illegal access.
- User domain 5 (dom5)
A sample with domain ID = 5.
This sample application uses various service calls of the kernel. A task is shifted to the WAITING state and another task cancels the WAITING state.
- Idling task (idle)
The lowest-priority task in the system. This sample application simply performs an infinite loop.
The idling task is assigned to the kernel domain.

The following application is specialized for simulator use.

- Monitor (monitor)

This application displays the response to the input by the user in the simulated I/O window of the simulator. Various monitoring operations such as status reference to kernel objects are available through commands.

The monitor task is assigned to the kernel domain.

The following programs are provided as system applications.

- Memory access violation handler
- System down routine
- CPU exception handler
- Interrupt and exception hook routine

In this sample system, the standard library functions are included.

Figure 11.3 shows the directory structure under samples\.

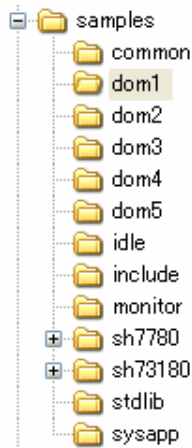


Figure 11.3 Directory Structure under samples

- (1) dom1\
Contains the sample source code for domain 1.
- (2) dom2\
Contains the sample source code for domain 2.
- (3) dom3\
Contains the sample source code for domain 3.
- (4) dom4\
Contains the sample source code for domain 4.
- (5) dom5\
Contains the sample source code for domain 5.
- (6) idle\
Contains the idling task.
- (7) include\
Contains the common header files used in the sample system.
- (8) stdlib\
Contains the source code necessary to include the standard library.
- (9) sysapp\
Contains the system application files.
- (10) *shnnnn*\
Contains the sample source code, configurator files, and HEW workspaces and projects specialized for the SHnnnn microcomputer.

11.3.2 Lists of Kernel Objects

The following shows lists of the kernel objects used in this sample system.

Table 11.1 Tasks

Classification	ID	Function Name	Creation and Initiation	Priority	Remarks
dom1	ID_DOM1_MAIN * ²	DOM1_Main()	Created and initiated by configurator	5	
	Assigned dynamically * ¹	DOM1_Input()	Created and initiated by DOM1_Main	6	
	Assigned dynamically * ¹	DOM1_Output()	Created and initiated by DOM1_Main	7	
dom2	ID_DOM2_MAIN * ²	DOM2_Main()	Created and initiated by configurator	10	
dom3	ID_DOM3_Main * ²	DOM3_Main()	Created and initiated by configurator	8	
dom4	ID_DOM4_Main * ²	DOM4_Main()	Created and initiated by configurator	15	
dom5	ID_DOM5_Main * ²	DOM5_Main()	Created by configurator * ³	9	
	Assigned dynamically * ¹	DOM1_Sub()	Created and initiated by DOM5_Main	10	
Idling task	ID_IDLETASK(20)	IdleTask()	Created and initiated by configurator	20	
Monitor	ID_MONITOR(19)	MonitorTask()	Created and initiated by configurator	1	Only when simulator is used

Notes: *¹ Automatically assigned through the acre_tsk service call.

*² Automatically assigned by the configurator.

*³ It is only created but is not initiated with the settings at shipment.

Table 11.2 Semaphore

Classification	ID	Creation
dom5	Assigned dynamically *	Created by DOM5_Main

Note: * Automatically assigned through the acre_sem service call.

Table 11.3 Data Queue

Classification	ID	Creation
dom5	Assigned dynamically *	Created by DOM5_Main

Note: * Automatically assigned through the acre_dtq service call.

Table 11.4 Mailboxes

Classification	ID	Creation	Remarks
dom1	Assigned dynamically * ¹	Created by DOM1_Main	
dom2	ID_DOM2_MBX * ²	Created by configurator	Only when the memory object protection function is not selected

Notes: *¹ Automatically assigned through the acre_mbx service call.

*² Automatically assigned by the configurator.

Table 11.5 Mutexes

Classification	ID	Creation	Remarks
stdlib	ID_MTX_MALLOC(1)	Created by configurator	
stdlib	ID_MTX_STRTOK(2)	Created by configurator	
stdlib	ID_MTX_FILETBL(3)	Created by configurator	Only when the simulator is used
stdlib	ID_MTX_STDIO(4)	Created by configurator	Only when the simulator is used

Table 11.6 Fixed-Size Memory Pool

Classification	ID	Creation
dom1	Assigned dynamically *	Created by DOM1_Main

Note: * Automatically assigned through the acre_mpf service call.

Table 11.7 Variable-Size Memory Pool

Classification	ID	Creation	Remarks
dom1	ID_DOM2_MPL *	Created by configurator	Only when the memory object protection function is not selected

Note: * Automatically assigned by the configurator.

Table 11.8 Protected Memory Pool

Classification	ID	Creation	Remarks
dom2	ID_DOM2_MPP *	Created by configurator	Only when the memory object protection function is selected

Note: * Automatically assigned by the configurator.

Table 11.9 Protected Mailbox

Classification	ID	Creation	Remarks
dom2	ID_DOM2_MBP *	Created by configurator	Only when the memory object protection function is selected

Note: * Automatically assigned by the configurator.

Table 11.10 Interrupt and CPU Exception Handlers

Interrupt or Exception Code	Function Name	Definition	Remarks
H'0E0	exception_handler() in samples\sysapp\exchdr.c	Defined by configurator	Instruction address error Data address error (read)
H'100		Defined by configurator	Data address error (write)
H'180		Defined by configurator	General illegal instruction exception
H'1A0		Defined by configurator	Slot illegal instruction exception
H'E00	MonitorWakeup() in samples\monitor\monitor.c	Created by MonitorTask	Interrupt for initiating the monitor (specialized for simulator use)
CFG_TIMINTNO	_kernel_tmrint() in samples\shnnnn\kernel\knl_side\tmrdrv.c	Defined by configurator	Standard timer driver

For static memory objects, refer to the following.

Reference: Section 11.15.4, Memory Map and Static Memory Objects

11.3.3 Task Exception Processing

This sample system uses the task exception processing function of the kernel.

A task exception processing routine is defined for each task, and when a CPU exception such as access violation occurs, a task exception processing is requested to the task. The address accessed illegally (TEA register value) is used for the task exception request pattern. Note that address 0 is handled as task exception request pattern H'ffffffff because 0 is not allowed for a pattern.

The task exception processing routine for each task suspends the processing that generated the CPU exception.

A task exception is requested in the following handlers.

- Memory access violation handler (only when the memory object protection function is used) (sysapp\mavhdr.c)
- CPU exception handler (sysapp\exchr.c)

Only domain 4 generates an access violation exception in the sample system.

For details, refer to the related source codes.

11.4 Sample Applications

Each sample application displays messages in the simulated I/O window by using `printf()` when it is executed in the simulator. Other functions in the standard library are not used in the sample applications.

11.4.1 User domain 1 (dom1)

The sample application with domain ID = 1, which performs data communications within a domain by using a fixed-size memory pool and a mailbox.

The main task creates an input task and an output task.

The input task acquires a memory block from a fixed-size memory pool, creates a message in the memory block, and sends it to a mailbox.

The output task receives data from the mailbox and returns the message area to the fixed-size memory pool.

The sample application uses the following kernel objects.

- Main task (DOM1_Main() in dom1_main.c)
The first task to be executed in domain 1. It is created and initiated by the configurator.
- Input task (DOM1_Input() in dom1_input.c)
It is created and initiated by the main task.
- Output task (DOM1_Output() in dom1_output.c)
It is created and initiated by the main task.
- Fixed-size memory pool and mailbox
They are created by the main task.

11.4.2 User domain 2 (dom2)

The sample application with domain ID = 2, which performs data communications between domains.

When the memory object protection function is selected, this sample application uses a protected memory pool and a protected mailbox. When the memory object protection function is not selected, it uses a variable-size memory pool and a mailbox.

The main task receives a message from a protected mailbox or the mailbox and returns the message area to a protected memory pool or a variable-size memory pool.

The sample application uses the following kernel objects.

- Main task (DOM2_Main() in dom2.c)
It is created and initiated by the configurator.
- Protected memory pool and protected mailbox (when the memory object protection function is used)
They are created by the configurator.
- Variable-size memory pool and mailbox (when the memory object protection function is not used)
They are created by the configurator.

11.4.3 User domain 3 (dom3)

The sample application with domain ID = 3, which sends data to user domain 2.

When the memory object protection function is selected, this sample application acquires a memory block from the protected memory pool in domain 2, creates a message in the memory block, and sends it to the protected mailbox in domain 2.

When the memory object protection function is not selected, the sample application acquires a memory block from the variable-size memory pool in domain 2, creates a message in the memory block, and sends it to the mailbox in domain 2.

The sample application uses the following kernel object.

- Main task (DOM3_Main() in dom3.c)
It is created and initiated by the configurator.

11.4.4 User domain 4 (dom4)

The sample application with domain ID = 4, which intentionally accesses an illegal address.

When the memory object protection function is selected, this illegal access is detected and the memory access violation handler requests a task exception. After that, a task exception processing routine is initiated and the instruction that caused the access violation will not be executed again.

The task exception processing routine restarts the current task so that an access violation exception is generated again.

The sample application uses the following kernel object.

- Main task (DOM4_Main() in dom4.c)
It is created and initiated by the configurator.

11.4.5 User domain 5 (dom5)

The sample application with domain ID = 5, which generates the WAITING state and cancels it by using various kernel functions.

The main task creates a sub-task, and then issues the following service calls to shift the main task to the WAITING state.

- slp_tsk
- wai_sem
- wai_flg
- rcv_dtq

The sub-task issues the following service calls to cancel the WAITING state of the main task.

- wup_tsk
- sig_sem
- set_flg
- psnd_dtq

The sample application uses the following kernel objects.

- Main task (DOM5_Main() in dom5.c)
It is created by the configurator but cannot be initiated with the settings at shipment. To initiate it, enter the ACT or STA command through the monitor or make appropriate settings in the application or configurator.
- Sub-task (DOM5_Sub() in dom5.c)
It is created and initiated by DOM5_Main.
- Semaphore, event flag, and data queue
They are created by DOM5_Main.

11.5 System Applications

11.5.1 System Down Routine (sysapp\sysdwn.c)

A system down routine (`_kernel_sysdwn()`) must always be included in the kernel.

The sample system down routine simply performs an infinite loop.

Reference: Creating system down routine -> Section 8.10, System Down Routine

11.5.2 Memory Access Violation Handler (sysapp\mavhdr.c)

To use the memory object protection function, a memory access violation handler (`_kernel_mavhdr()`) must be included in the kernel.

When access violation is generated in a task context, the sample memory access violation handler requests a task exception to that task (through service call `iras_tex`). When access violation is generated in a non-task context, the handler makes the system go down.

When the memory object protection function is not used, an empty object code is created for this handler through conditional compile.

Reference: Creating memory access violation handler -> Section 8.9, Memory Access Violation Handler

11.5.3 CPU Exception Handler (sysapp\exchdr.c)

The sample system defines `exception_handler()` in `exchdr.c` as the CPU exception handler for the following exception codes.

- Exception code H'0E0: Instruction address error or data address error (read)
- Exception code H'100: Data address error (write)
- Exception code H'180: General illegal instruction exception
- Exception code H'1A0: Slot illegal instruction exception

When an exception is generated in a task context, `exception_handler()` requests a task exception to that task (through service call `iras_tex`). When an exception is generated in a non-task context, the handler makes the system go down.

Reference: Creating CPU exception handler -> Section 8.8, CPU Exception Handler

11.5.4 Interrupt and Exception Hook Routine (sysapp\inthook.src)

Whether to use an interrupt and exception hook routine is specified through CFG_INTHOOK in the configurator. This sample system specifies that the hook routine is used.

However, this sample hook routine does not perform any processing and simply returns control.

Reference: Creating interrupt and exception hook routine -> Section 8.5, Interrupt and Exception Hook Routine

11.6 CPU-Dependent Processing

The source codes for the CPU-dependent processing shown below are stored in samples\shnnnn\kernel\knl_side\.

11.6.1 Standard Timer Driver (tmrdrv.c)

A sample code of the standard timer driver for the timer module (generally, the TMU) in the target microcomputer is provided.

Reference: Creating standard timer driver -> Section 9, Standard Timer Driver

11.6.2 CPU Reset Processing

The following files are provided for the CPU reset processing.

- reset.src
- resetprg.c
- init_mmu.c

reset.src is the first program to be executed immediately after a CPU reset, which is written in the assembly language. It initializes the stack pointer and bus state controller, then calls PowerON_Reset_PC() in resetprg.c.

PowerON_Reset_PC() initializes the MMU (InitializeMMU() in init_mmu.c) and cache, then start the kernel through the vsta_knl service call. After the vsta_knl call, execution does not return to the reset processing.

In general systems, these files should be modified according to the target environment.

11.7 Standard Library Functions and Runtime Routines

11.7.1 Overview

The standard library functions and runtime routines are output to a single library file created by the standard library generator. In other words, in general systems, the library file must be linked and initialized in every linkage unit.

However, in this kernel, the standard library created by the standard library generator is only linked to the kernel load module (knl_side) and it does not need to be linked to the other linkage units so that the system can be built with multiple linkage units. To be more specific, the library functions necessary for the target system should be selected in advance, and only those library functions should be embedded in the kernel load module (knl_side). In the other linkage units, the addresses of the referenced library functions are determined according to the symbol file that is output at kernel load module creation.

For runtime routines, this method cannot be applied because necessary runtime routines are determined when programs are compiled. A library specialized for runtime routines should be created (runtime.lib) and linked to each linkage unit.

11.7.2 Selecting Necessary Standard Library Functions

stdlib\select.c selects the necessary library functions to be included in the system.

Only the selected functions are extracted from the library created by the standard library generator and embedded in the kernel load module.

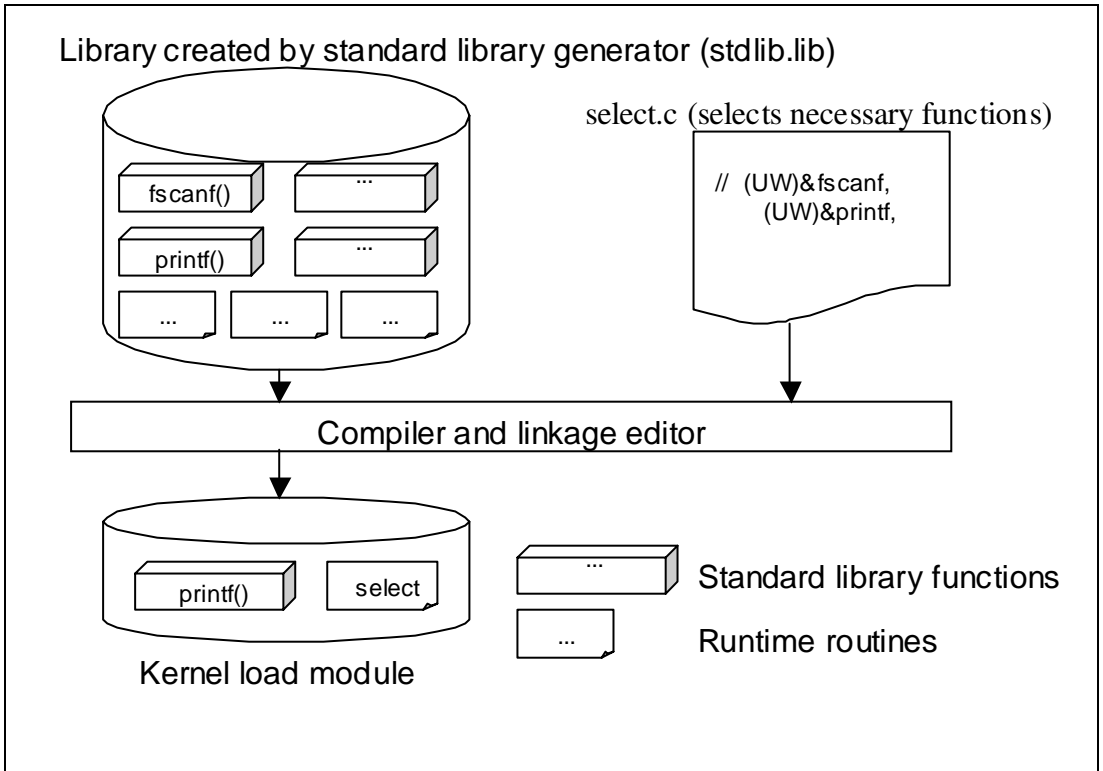


Figure 11.4 Embedding Standard Library Functions

11.7.3 stdio.h

This sample only supports "stdin", "stdio", and "stderr" when the simulator is used. They are used for input/output interface with the simulated I/O window in the simulator.

Input from "stdin" is polled within low-level interface routine charget() (samples\stdlib\lowsr.c), and therefore, execution of every program stops until an input is detected.

When the simulator is not used, the sample does not support stdio.h.

11.7.4 Kernel Objects to be Used

The mutexes with the following ID numbers, which are created by the configurator, are used in the sample.

(1) ID_MTX_MALLOC(1)

This mutex is necessary when the standard library functions are generated as a reentrant library.

It is used for exclusive control for the malloc-type functions.

This ID number is determined according to the specifications of the standard library generator.

(2) ID_MTX_STRTOK(2)

This mutex is necessary when the standard library functions are generated as a reentrant library.

It is used for exclusive control for the strtok() function.

This ID number is determined according to the specifications of the standard library generator.

(3) ID_MTX_FILETBL(3)

This mutex is necessary when the standard library functions are generated as a reentrant library.

It is used for exclusive control for the __job file table.

This ID number is determined according to the specifications of the standard library generator.

When the simulator is not used, the sample does not support stdio.h and so does not use this mutex.

(4) ID_MTX_STDIO(4)

In the sample system, this mutex is locked and unlocked on a line-by-line basis so that the input/output through the simulated I/O window does not become disordered.

When the simulator is not used, the sample does not support stdio.h and so does not use this mutex.

11.7.5 Functions Necessary to Use Standard Library Functions

The following functions must be implemented to use the standard library functions.

- Function for initializing library functions
- Low-level interface routines

These functions are implemented in stdlib\lowsrc.c.

_INITLIB() is the initialization function.

_INITLIB() is defined as an initialization routine by the configurator.

11.7.6 Customizing Environment Settings for Standard Library Functions

The following section in samples\include\lowsrc.h should be modified as required.

```
/* *****  
 * User setting  
***** */  
  
#ifdef SIM  
#define SIM_IO      4 /* Simulated I/O system call address */ <-(1)  
#endif  
  
#define IOSTREAM      3UL /* Number of I/O Stream*/  
#define HEAPSIZE      8192UL /* for malloc() */ <-(2)  
#define HEAPSIZE__X 1024UL /* for malloc__X() */ <-(3)  
#define HEAPSIZE__Y 1024UL /* for malloc__Y() */ <-(4)  
  
#ifdef _REENTRANT  
#define MAXTSKID 20 /* Define maximum task ID which uses standard library */ <-(5)  
#define TMOUT 300000L /* Timeout when cannot lock mutex. */ <-(6)  
#endif
```

- (1) The address of the system call to the simulator, which is used for I/O simulation.
- (2) The heap size used by malloc().
- (3) The heap size used by malloc__X(). Note that malloc__X() is not supported with the initial settings at shipment.
- (4) The heap size used by malloc__Y(). Note that malloc__Y() is not supported with the initial settings at shipment.
- (5) When the standard library is generated as a reentrant library, specify here the value of CFG_MAXTSKID that is set in the configurator.
- (6) Specify this value when the standard library is generated as a reentrant library. This value is the timeout period for locking a mutex by low-level interface routine wait_sem(). When

TMO_FEVR(-1) is set, waiting state lasts forever. When TMO_POL(0) is set, polling is specified.

11.7.7 Note on Standard Library Functions

When using a reentrant library, do not call library functions that use low-level routine `wait_sem()`, such as `printf()` or `malloc()`, in a non-task context. If this is attempted, an error is returned from the library function.

11.7.8 Section Initialization Function (`_INITSCT()`)

A section initialization function should be prepared although it is not a standard library function. The section initialization function clears the B sections to 0 and copies the data in the section that is specified by the ROM support function of the linkage editor from ROM (D section) to RAM (R section).

The standard library generator creates standard section initialization function `_INITSCT()` in the library file. However, this `_INITSCT()` acquires information regarding the addresses and sizes of B, D, and R sections from the tables created with the fixed section names (C\$BSEC section and C\$DSEC section). Accordingly, when there are multiple sections to be initialized and they have different access permissions, `_INITSCT()` can be executed only in the kernel domain.

To avoid this restriction, this sample system provides an original `_INITSCT()` function, which acquires the section initialization information as parameters.

`_INITSCT()` is included in `stdlib\initsct.c`. For details about the interface specifications, refer to the source code.

11.7.9 Runtime Routines

The runtime routines require section initialization as part of its initialization processing.

(1) Linkage unit other than `kn1_side`

When runtime routines are required in a linkage unit other than `kn1_side`, `runtime.lib` created by `runtime.hwp` and `common\init_runtime.c` should be linked. `init_runtime()` in `init_runtime.c` works as a section initialization function for `runtime.lib`.

`init_runtime()` must be executed first in the linkage unit.

In this sample system, `init_runtime()` is executed as follows.

- `env_side: init_runtime()` is defined as an initialization routine by the configurator.
- `app_dom5: init_runtim()` is called at the start of the entry function (`DOM5_Main()`) in domain 5.

(2) `kn1_side` Linkage Unit

In `kn1_side`, runtime routines are linked from `stdlib.lib`. Since section initialization for runtime routines is also performed in `_INITLIB()`, there is no need to execute `init_rintime.c`.

11.8 Monitor

11.8.1 Overview

The monitor is a program that handles commands input by the user through the simulated I/O window of the simulator. It only operates in the simulator and does not operate in the actual target system.

The sample application uses the following kernel objects.

- Monitor task (the `MonitorTask` function in `monitor.c`)
It is created and initiated by the configurator.
- Monitor interrupt handler (the `MonitorWakeup` function in `monitor.c`)
It is defined by the monitor task.

The monitor uses standard library functions.

11.8.2 Monitor Operation

After the system is started, the monitor task is initiated and a startup message appears in the simulated I/O window. Then, the monitor task enters the `WAITING` state, in which no commands can be input.

To make the monitor ready for command input, generate an interrupt with interrupt code `H'FE0`. In the HEW debugging session configured at shipment, this interrupt is assigned to a trigger button; click this button to shift the monitor to the command input waiting state.

In command input waiting state, a prompt, `"MON>"`, appears on the simulated I/O window.

The monitor locks mutex `ID_MTX_SIMIO` while waiting for command input or processing a command; during this period, input and output to the simulated I/O window by other tasks is inhibited.

Entering the Q command makes the monitor unlock ID_MTX_SIMIO and shifts the monitor back to the interrupt waiting state.

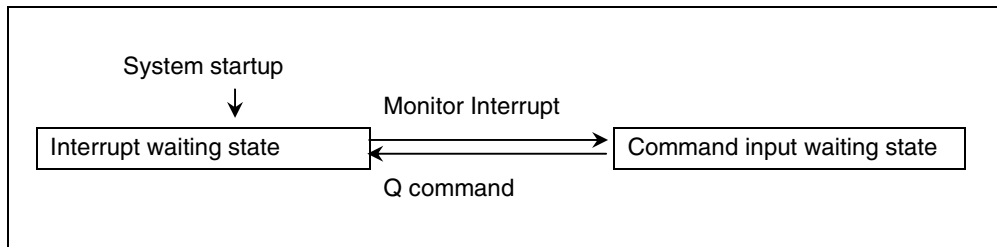


Figure 11.5 Monitor State Transition

11.8.3 Changing Monitor Interrupt

To change the interrupt number, modify the following section in monitor.h.

```

/*****
 * User setting
 *****/
/** Please define interrupt number to wake-up monitor */
#define INTNO_MONITOR 0xfe0UL      <- Change to a new interrupt number.

```

To set up a trigger button, select [Setting...] from the popup menu in the [Trigger] window to open the [Trigger Setting] dialog box and set the following conditions in the dialog box.

- Interrupt Type: INTNO_MONITOR value shown above
- Priority: A value within the range from 1 to CFG_KNLLVL

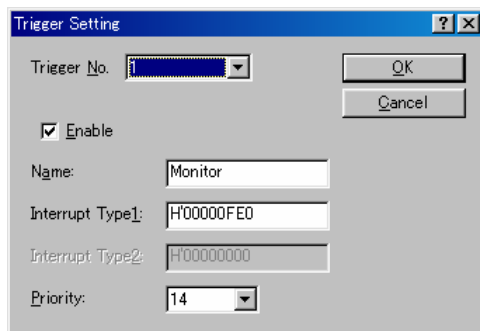


Figure 11.6 Trigger Setting Dialog Box

11.8.4 Monitor Commands

Commands should be input in the following syntax.

```
MON> <command> [Δ<parameter1> Δ<parameter2>...]
```

The detailed syntax for each command can be displayed by entering "?Δ<command>".

Command	Function
TSK	Displays the state of a task
ACT	Initiates a task (through service call act_tsk)
STA	Initiates a task (through service call sta_tsk)
TER	Forcibly terminates a task (through service call ter_tsk)
SEM	Displays the state of a semaphore
FLG	Displays the state of an event flag
DTQ	Displays the state of a data queue
MBX	Displays the state of a mailbox
MTX	Displays the state of a mutex
MBF	Displays the state of a message buffer
MPF	Displays the state of a fixed-size memory pool
MPL	Displays the state of a variable-size memory pool
TIM	Displays the current system clock time
CYC	Displays the state of a cyclic handler
ALM	Displays the state of an alarm handler
OVR	Displays the state of an overrun handler
MPP	Displays the state of a protected memory pool
MBP	Displays the state of a protected mailbox
RSP	Displays the state of a resource pool
SYP	Displays the state of a system pool
MEM	Displays the state of a memory object
PRB	Checks the access permission for a memory area
?	Displays the help information
Q	Returns the monitor to the interrupt waiting state

Notes:

- (1) For input as a parameter, a value with prefix "0x" is handled as hexadecimal, and a value without a prefix is handled as decimal.
- (2) The backspace key is ignored.

11.9 HEW Workspaces and Projects

11.9.1 Overview

The samples\shnnnn\ directory contains HEW workspaces and projects for creating load modules, in addition to the SHnnnn microcomputer-dependent source codes.

This sample system uses two workspaces to generate three load modules in the absolute address format.

One workspace is samples\shnnn\kernel\kernel.hws, which generates load modules for the kernel, and another workspace is samples\shnnnn\app_dom5\app_dom5.hws, which generates a load module containing domain 5 only.

Figure 11.7 shows the relationship between the workspaces and projects.

kenrel.hws contains the following projects.

- knl_side.hwp
It generates a load module on the kernel side. The items contained in this load module are shown in figure 11.7.
- knl_side_sym.hwp
It generates an object of the symbol file for the load module on the kernel side. The symbols on the kernel side can be referred to by inputting this object during linkage of other units.
- env_side.hwp
It generates a load module on the kernel environment side. The items contained in this load module are shown in figure 11.7.
- runtime.hwp
It generates a library which contains runtime routines only. This library should be linked during linkage of units other than knl_side. When using this library, link common\init_runtime.c so that init_runtime() in init_runtime.c is executed first.

app_dom5.hwp is the only project contained in app_dom5.hws. This project generates a load module which contains domain 5 only.

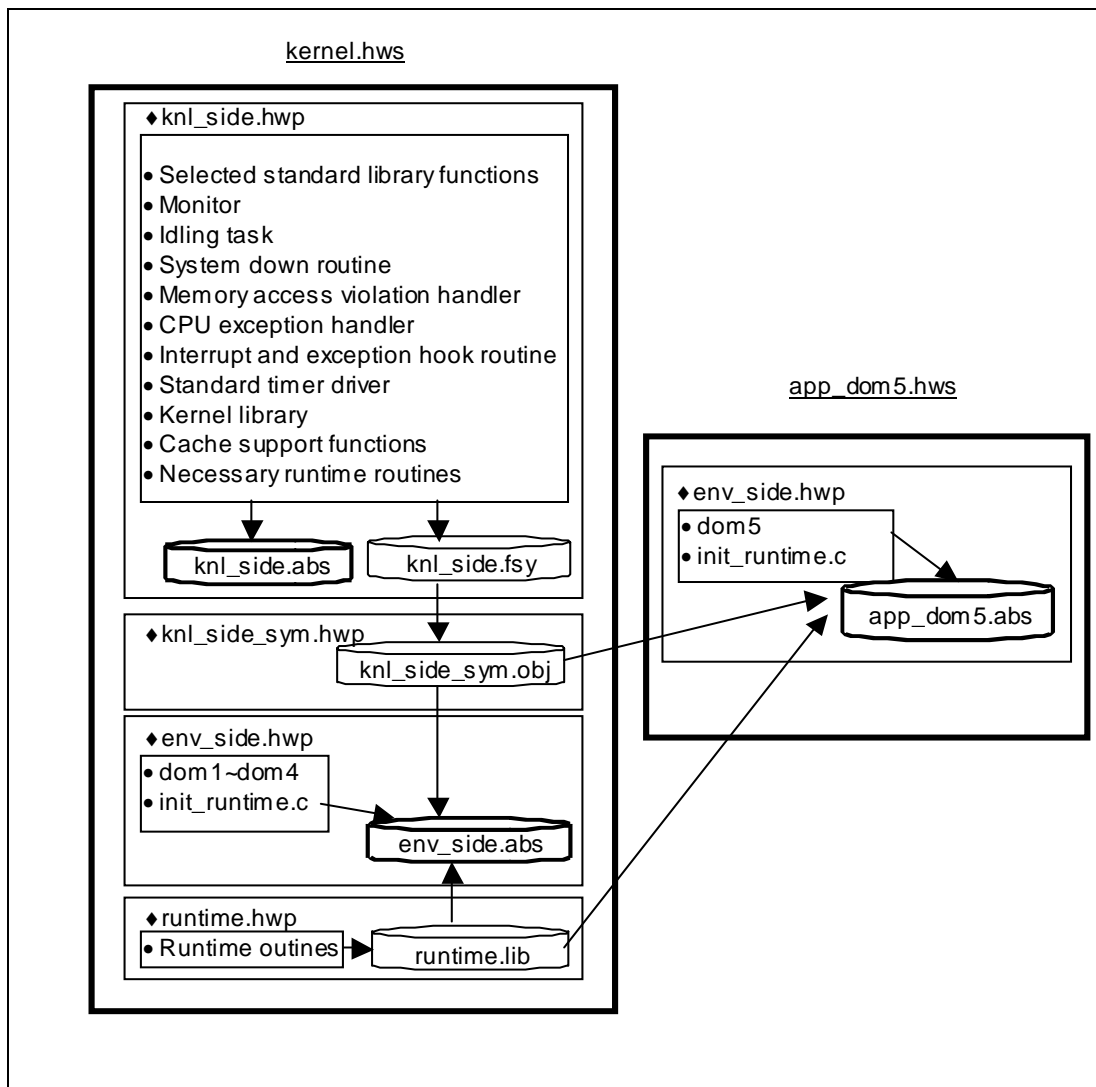


Figure 11.7 Configuration of HEW Workspaces and Projects◆◆

11.9.2 Structure of Workspace Directories

This version (V.1.01) of product provides the *shnnnn* directories shown in table 11.11. When the target microcomputer is not shown in the table, select the microcomputer that is closer in function to the target microcomputer and modify the contents of its directory as necessary.

The microcomputer-dependent source codes are stored in samples\shnnnn\kernel\kn1_side\.

Table 11.11 *shnnnn*

<i>shnnnn</i>	CPU Core	Microcomputer	Version of HEW (Compiler Package) Used for Workspace Creation
sh73180	SH4AL-DSP	SH73180	4.00.02 (9.00, Release 03)
sh7343	SH4AL-DSP (with extended functions)	SH7343	4.00.02 (9.00, Release 03)
sh7780	SH-4A	SH7780	4.00.02 (9.00, Release 03)
sh7785	SH-4A (with extended functions),	SH7785	4.00.02 (9.00, Release 03)

Figure 11.8 shows the directory structure under samples\shnnnn\.

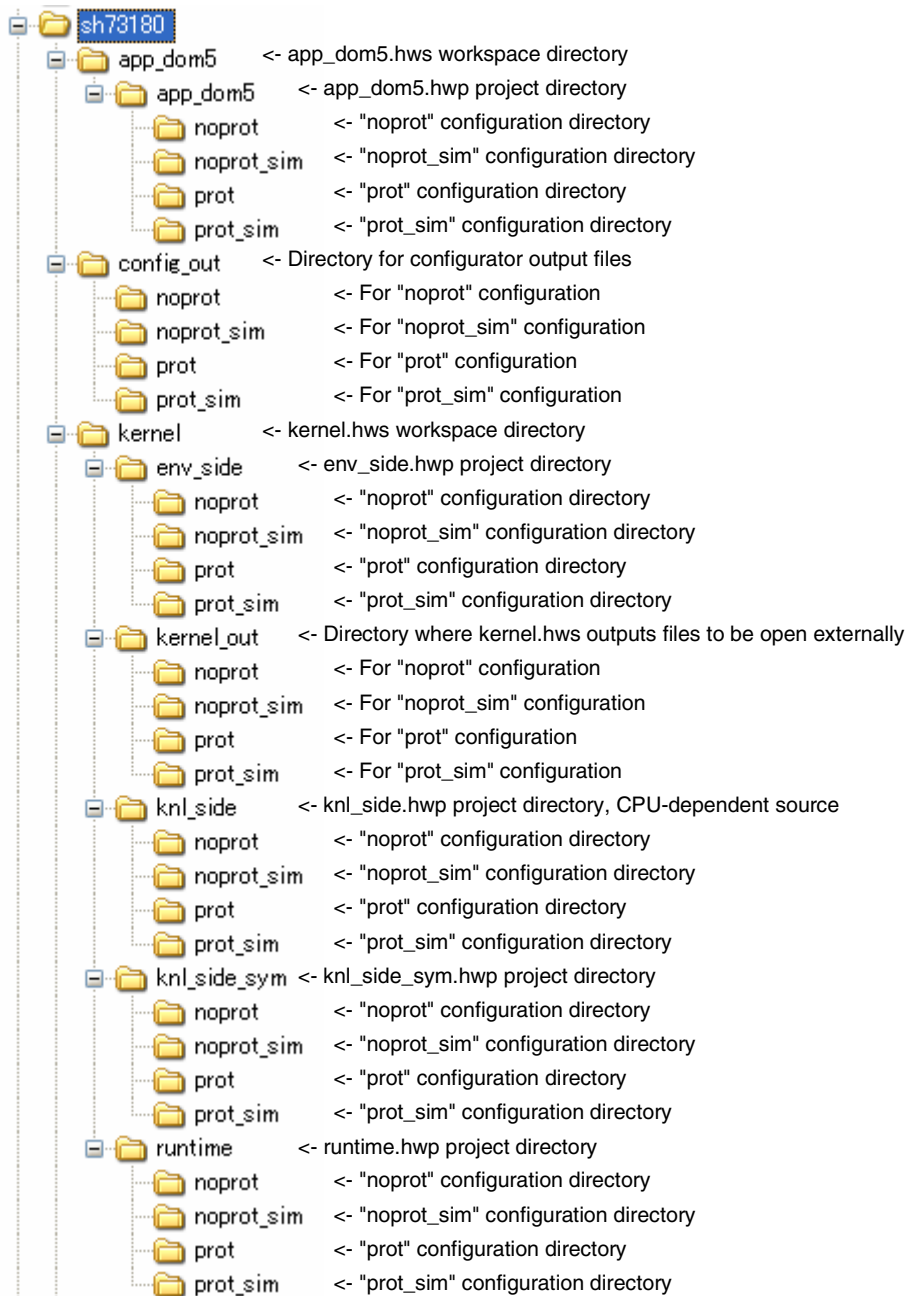


Figure 11.8 Directory Structure under `samples\shnnnn\`

kernel\knl_side\ contains the following sample source codes for the respective microcomputer.

- reset.src, resetprg.c, init_mmu.c: CPU reset processing
- tmrdrv.c: Standard timer driver

11.9.3 HEW Build Configuration and Directories for Configurator Files

For all projects, a set of HEW configurations of the same names are provided.

Since the configurator settings are related to the HEW configuration, the configurator file (with extension hcf) and its output file for each HEW configuration are also stored in the directory with the same name as the respective HEW configuration under *shnnnn*\config_out\. In each project, the configurator output directory for each configuration is specified by using HEW placeholder "\$ (CONFIGNAME)" (configuration name).

Table 11.12 shows the differences between configurations.

Table 11.12 HEW Configurations

Configuration	Memory Object Protection	Simulated I/O Window and Monitor	Memory Map * ¹
noprot	Not used	Not used	Memory map suitable for download to RAM on the board
noprot_sim * ²	Not used	Used	Memory map suitable for storing in ROM
prot	Used	Not used	Memory map suitable for download to RAM on the board
prot_sim * ²	Used	Used	Memory map suitable for storing in ROM

Notes: *¹ For details, refer to section 11.15, Memory Allocation.

*² This configuration is not provided for **sh7780** (for the SH7780 microcomputer) in V.1.00, Release 00.

In the "prot_sim" and "noprot_sim" configurations, which are configurations for use with the simulator, "-def = SIM" is specified as a compiler option. samples\stdlib\lowsrc.c is compiled according to this condition as shown in table 11.13.

Table 11.3 Difference Due to "-def = SIM" Option

"-def = SIM"	Support of stdio.h
Specified	Only "stdin", "stdout", and "stderr" are supported. They are used as input/output interface with the simulated I/O window of the simulator.
Not specified	Not supported.

In common header `samples\include\sim_printf.h`, `sim_printf()` used in each domain is defined as `printf()` when "-def = SIM" is specified, or defined as a null statement when "-def = SIM" is not specified.

11.9.4 Moving HEW Workspaces

HEW workspaces must be moved with keeping the directory structure under the kernel installation directory.

In the sample workspaces, the locations of the files, such as include files or library files, and directories are specified in relative paths (HEW placeholders) within the range under the kernel installation directory.

11.9.5 Option Settings for Build

(1) Common Options for All Projects

Refer to the following compiler user's manual and make the same settings for necessary options for all projects.

Section 9.4.3, Important Information on Program Development, in SuperHTM RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor (Compiler Package V.9.00) User's Manual

(2) Include Paths

The following paths must be specified as include paths. At shipment, they are specified with the paths relative to placeholder `$(WORKSPDIR)` (workspace directory).

- `include\`: Contains the standard headers of the kernel, such as `itron.h`.
- `samples\include\`: Contains the common headers used in the sample system.
- `config_out\$(CONFIGNAME)`: Contains the configurator output files for the respective HEW configuration.

(3) Reentrant Library

When a standard library is created as a reentrant library, compiler option "-def = _REENTRANT" must be specified for the source codes that use the library functions. In this sample system, this option is specified for all files.

(4) Configurations for Simulator ("prot_sim" and "noprot_sim")

Compiler option "-def = _SIM" is specified for the configurations for simulator use. For details, refer to section 11.9.3, HEW Build Configuration and Directories for Configurator Files.

(5) Endian

Big or little endian is specified at shipment. Modify it as necessary.

The file names of the kernel library and cache support objects to be specified at knl_side linkage differ according to endian. When modifying endian, change these file names together.

(6) Other Options

For the other option settings, refer to the settings in the sample project.

11.10 knl_side.hwp Project in kernel.hws

11.10.1 Overview

This project generates the following files.

(1) knl_side.abs

This file is generated in kernel\knl_side\\$(CONFIGNAME).

knl_side.abs contains the kernel library and cache support object files provided as part of the HI7300/PX and the standard library functions.

(2) knl_side.fsy

This file is generated in kernel\knl_side\\$(CONFIGNAME).

knl_side.fsy is an assembly-language source file, which defines the symbol addresses that are used in knl_side.abs and open to the other linkage units. This file is assembled in the knl_side_sym.hwp project and the resultant object file is linked to the other linkage units.

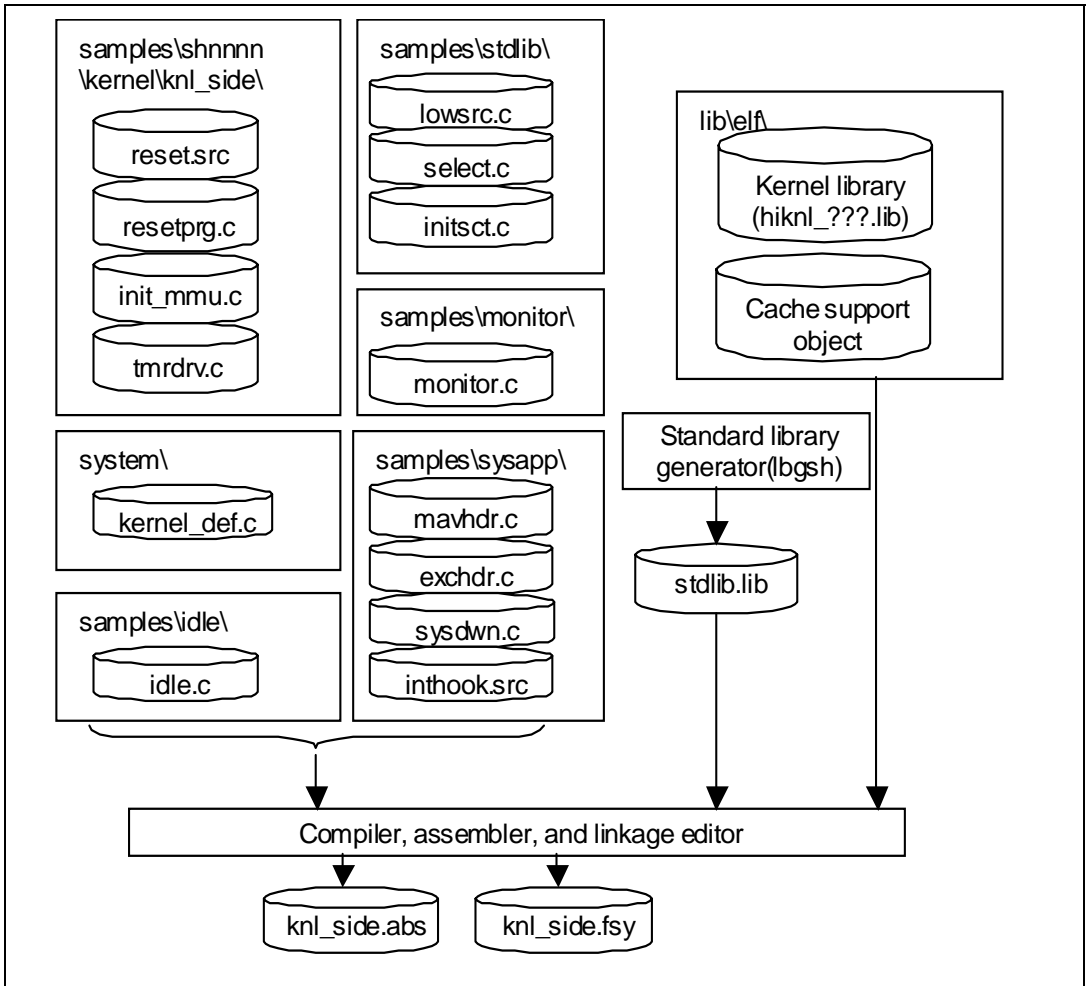


Figure 11.9 Overview of knl_side.hwp

11.10.2 Source Files to Be Registered in Project

The following source files should be registered in the project.

- (1) system\kernel_def.c: Always necessary

One of the configurator output files. It contains the kernel-side information.

- (2) samples\sysapp\sysdwn.c: Always necessary

A system down routine. It is always necessary.

- (3) samples\sysapp\mavhdr.c

A memory access violation handler. It is necessary when the memory object protection function is selected in the configurator. If this file is compiled when the memory object protection function is not selected, an empty object file is generated.

(4) `samples\sysapp\inthook.src`

An interrupt and exception hook routine. It is necessary when use of an interrupt and CPU exception hook is selected in the configurator.

(5) `samples\sysapp\exchdr.c`

A CPU exception handler. In this sample, this CPU exception handler is defined for exception codes H'0E0, H'100, H'180, and H'1A0 in the configurator.

(6) `samples\shnnnn\kernel\knl_side\reset.src`, `resetprg.c`, and `init_mmu.c`

Sample files for CPU reset processing.

(7) `samples\shnnnn\kernel\knl_side\tmrdrv.c`

A sample file for the standard timer driver. It is necessary when use of the standard timer driver is selected in the configurator. If this file is compiled when use of the optimized timer driver is selected, an empty object file is generated.

(8) `samples\stdlib\lowsr.c`, `select.c`, and `initsct.c`

These files must be registered when the standard library is used.

(9) `samples\monitor\monitor.c`

This file must be registered when the monitor is used in the simulator.

(10) `samples\idle\idle.c`

An idling task file.

In addition, the symbols and sections of the objects for which [Kernel side] is specified in the configurator must also be linked in this project.

11.10.3 Standard Library Generator Settings

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "knl_side" project in the left pane, then select the [Standard Library] tab.

Select [Standard Library] for [Category:] and the category list shown in figure 11.10 appears. Here, select the library functions that should be included in the library file output by the standard library generator; that is, specify the functions selected in `samples\stdlib\select.c`. If all the necessary functions are not specified here, an error is generated at linkage because the library functions referred to in `select.c` cannot be found.

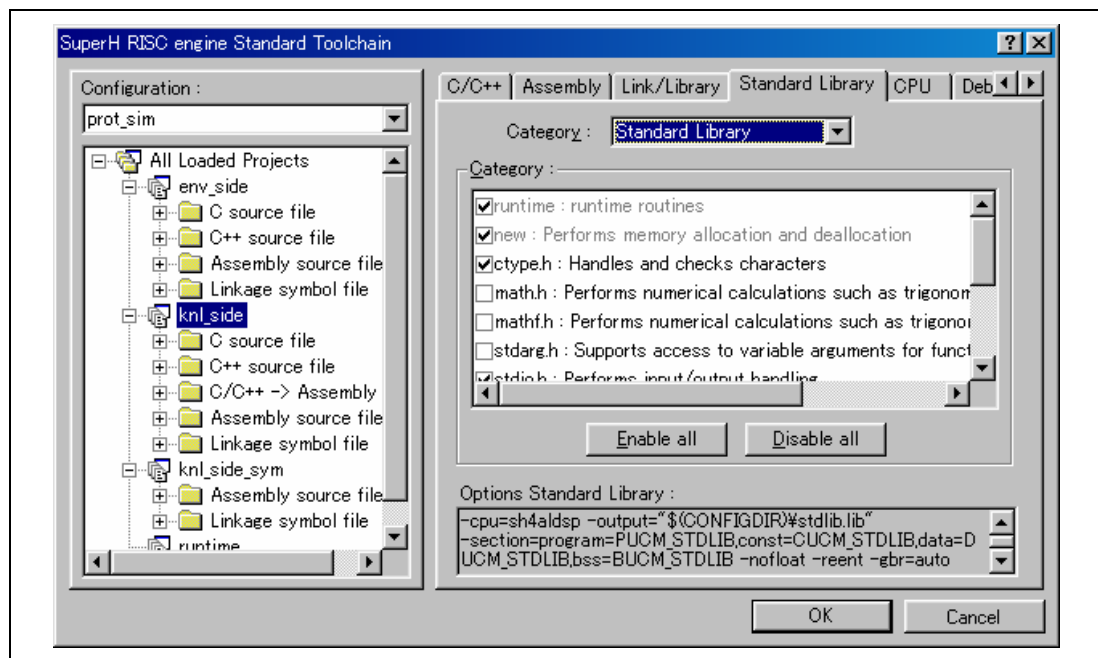


Figure 11.10 Standard Library Generator Settings ([Standard Library] Category)

Next, select [Object] for [Category:] and the object setting items shown in figure 11.11 appears. In usual operation, select [Generate reentrant library].

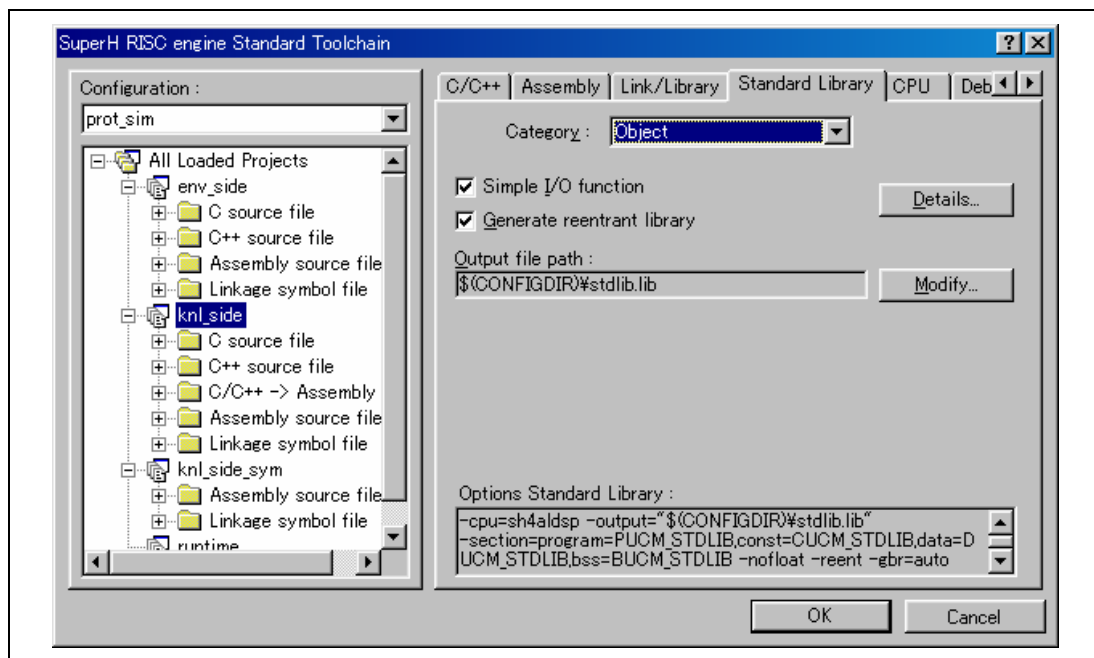


Figure 11.11 Standard Library Generator Settings ([Object] Category)

Then, click the [Details...] button to open the dialog box shown in figure 11.12.

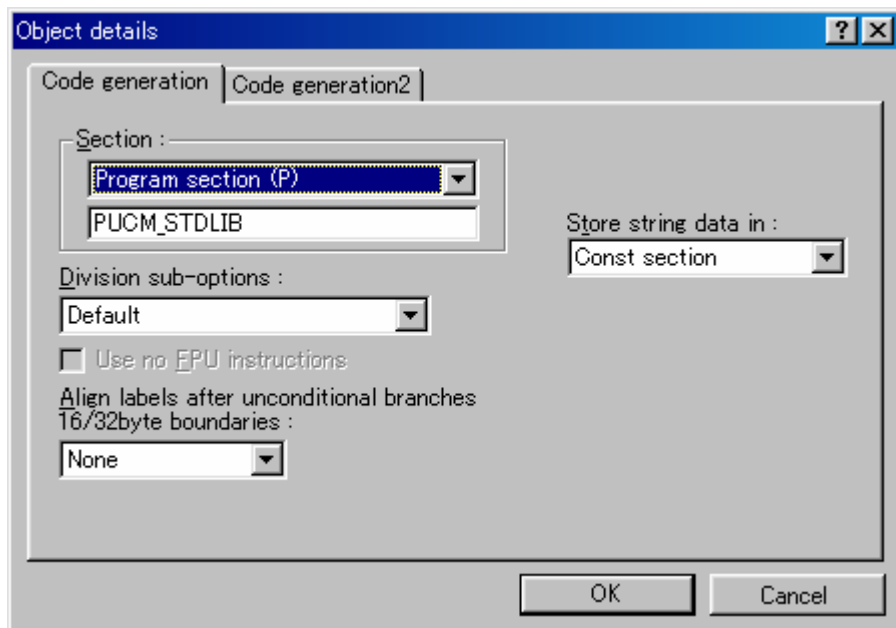


Figure 11.12 Standard Library Generator Settings ([Object details] Dialog Box)

In the [Section:] group box, specify the section names for the standard library functions as shown below. These names are also specified by #pragma section statements in the source files in samples\stdlib\. When changing the section names specified in this dialog box, also modify the respective names in the source files.

- [Program section (P)]: PUCM_STDLIB
- [Constant section (C)]: CUCM_STDLIB
- [Initialized data section (D)]: DUCM_STDLIB
- [Uninitialized data section (B)]: BUCM_STDLIB

11.10.4 Linkage Editor Settings

(1) Specifying Kernel Library

Be sure to link either one of the following kernel library files, which are stored in lib\elf\.

- \$(WORKSPDIR)\..\..\lib\elf\hiknl_big.lib: For big endian.
- \$(WORKSPDIR)\..\..\lib\elf\hiknl_little.lib: For little endian.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "knl_side" project in the left pane, then select the [Link/Library] tab.

Select [Input] for [Category:] and [Library files] for [Show entries for:], then specify the kernel library as shown in figure 11.13.

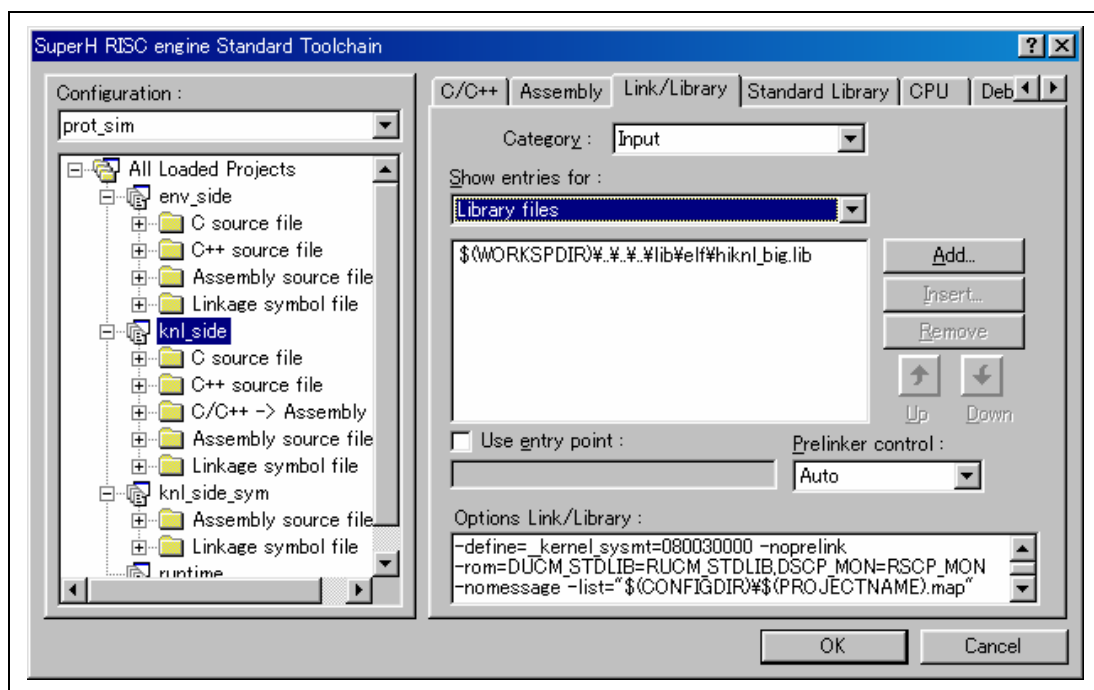


Figure 11.13 Linkage Editor Settings (Specifying Kernel Library)

(2) Specifying Cache Support Object File

To embed the cache support functions in the system, a cache support object file must be input through the linkage editor. Cache support object files are stored in lib\elf\.

- \$(WORKSPDIR)\..\..\lib\elf\cache_sh4a_big.rel: For the SH-4A/SH4AL-DSP in big endian.
- \$(WORKSPDIR)\..\..\lib\elf\cache_sh4a_little.rel: For the SH-4A/SH4AL-DSP in little endian.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "kn1_side" project in the left pane, then select the [Link/Library] tab.

Select [Input] for [Category:] and [Relocatable files and object files] for [Show entries for:], and specify the cache support object file as shown in figure 11.14.

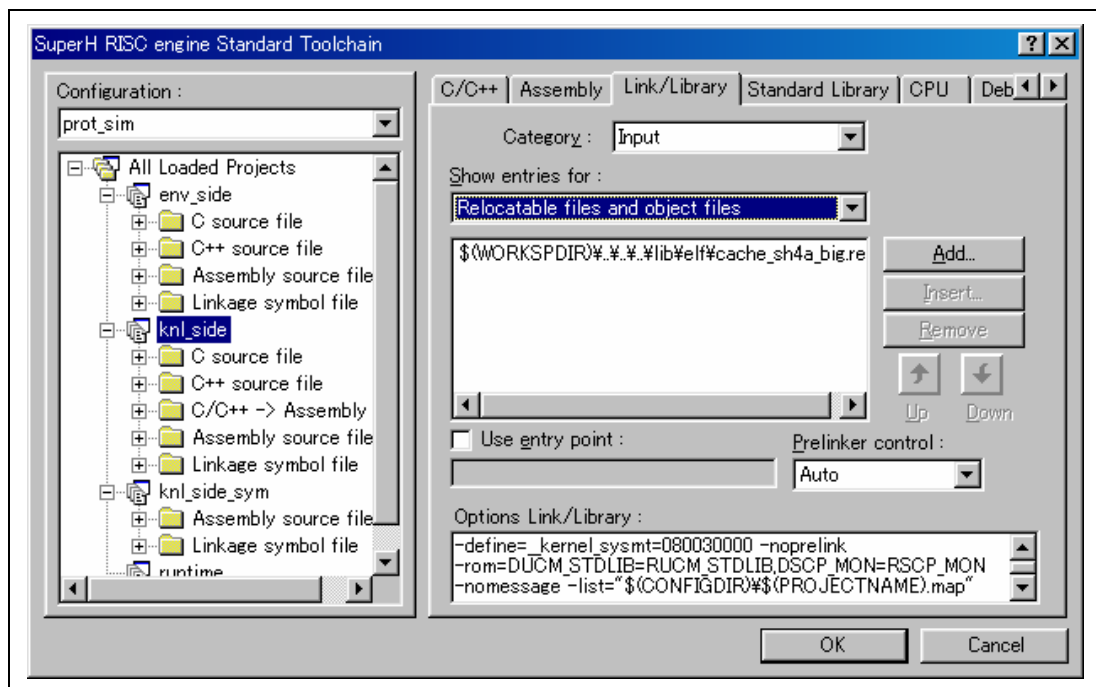


Figure 11.14 Linkage Editor Settings (Specifying Cache Support Object File)

(3) Settings for Initialized Data Sections

If an object to be linked includes an initialized data section (D section), [ROM to RAM mapped sections] (generation of R sections) must be specified in the linkage editor.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "knl_side" project in the left pane, then select the [Link/Library] tab.

Select [Output] for [Category:] and [ROM to RAM mapped sections] for [Show entries for:].

The following initialized data sections are included in the configuration at shipment.

- DSCP_MON (monitor) (only for the "prot_sim" and "noprot_sim" configurations)
- DUCM_STDLIB (standard library)

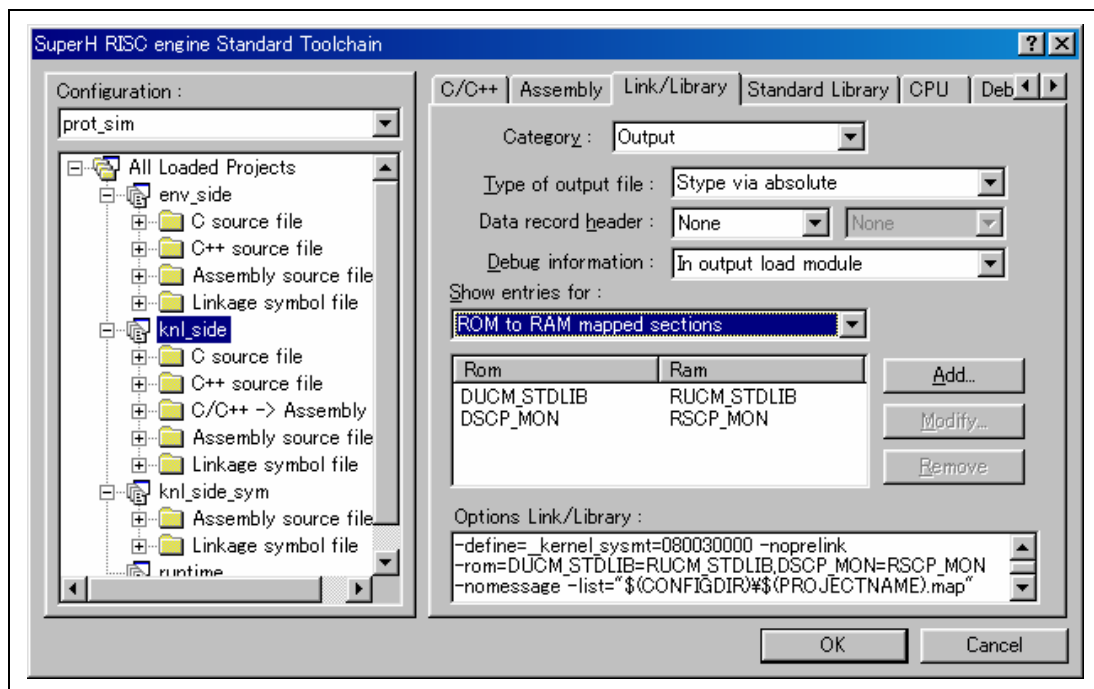


Figure 11.15 Linkage Editor Settings (Initialized Data Sections)

(4) Output of Symbol Address File (knl_side.fsy)

Specify the sections that include symbols to be open to the linkage units other than knl_side; that is, the following sections should be specified.

- PUC_hiintfc: Kernel service calls (always necessary)
- PSCP_hicac: Cache support functions
- PUCM_STDLIB: Standard library functions and section initialization function (_INITSCT())

The linkage editor generates assembly-language source file knl_side.fsy, which includes the addresses for the externally defined symbols used in the sections specified here, and outputs it in the directory where knl_side.abs is output (\$(CONFIGDIR) in usual operation). This file is used in knl_side_sym.hwp.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "knl_side" project in the left pane, then select the [Link/Library] tab. Select [Section] for [Category:] and [Symbol file] for [Show entries for:].

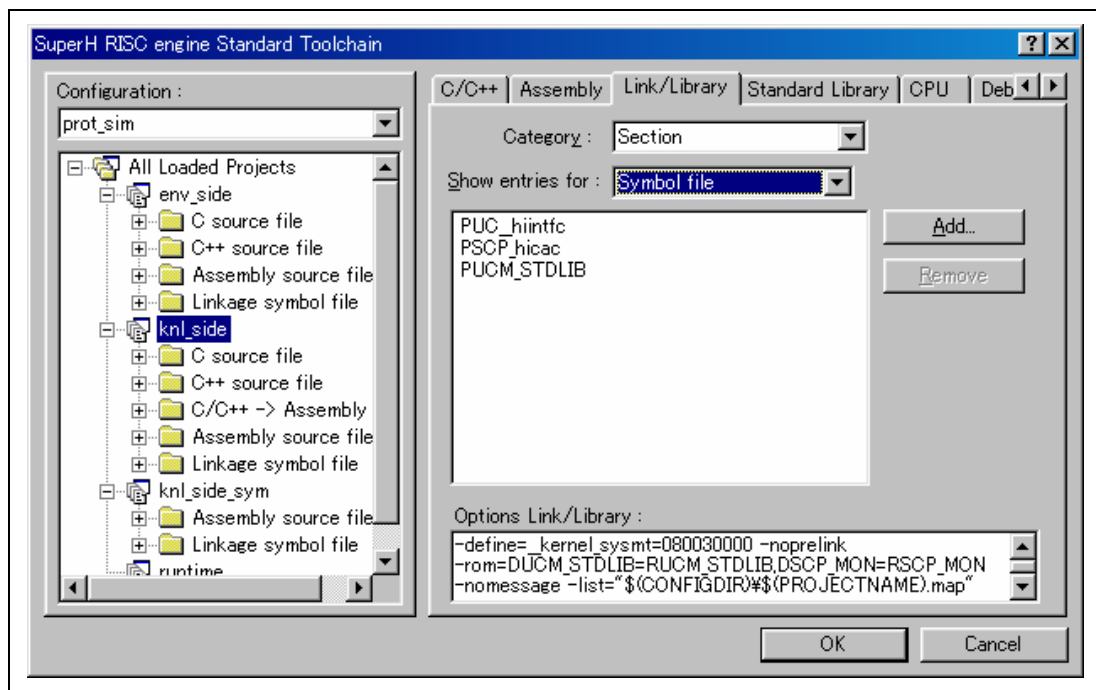


Figure 11.16 Linkage Editor Settings (Symbol Address File Output)

(5) Defining __kernel_sysmt Address

__kernel_sysmt is the kernel information table generated on the kernel environment side (env_side), and it indicates the start address of the PSCP_hisysmt section. The address where the PSCP_hisysmt is to be allocated must be determined in advance while the system is designed, and the determined address must be specified here.

Reference: Section 11.15.4 (2) (a), Section Block at Logical Address H'80030000 (CSCP_hisysmt, etc.)

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "knl_side" project in the left pane, then select the [Link/Library] tab.

Select [Input] for [Category:] and [Defines] for [Show entries for:].

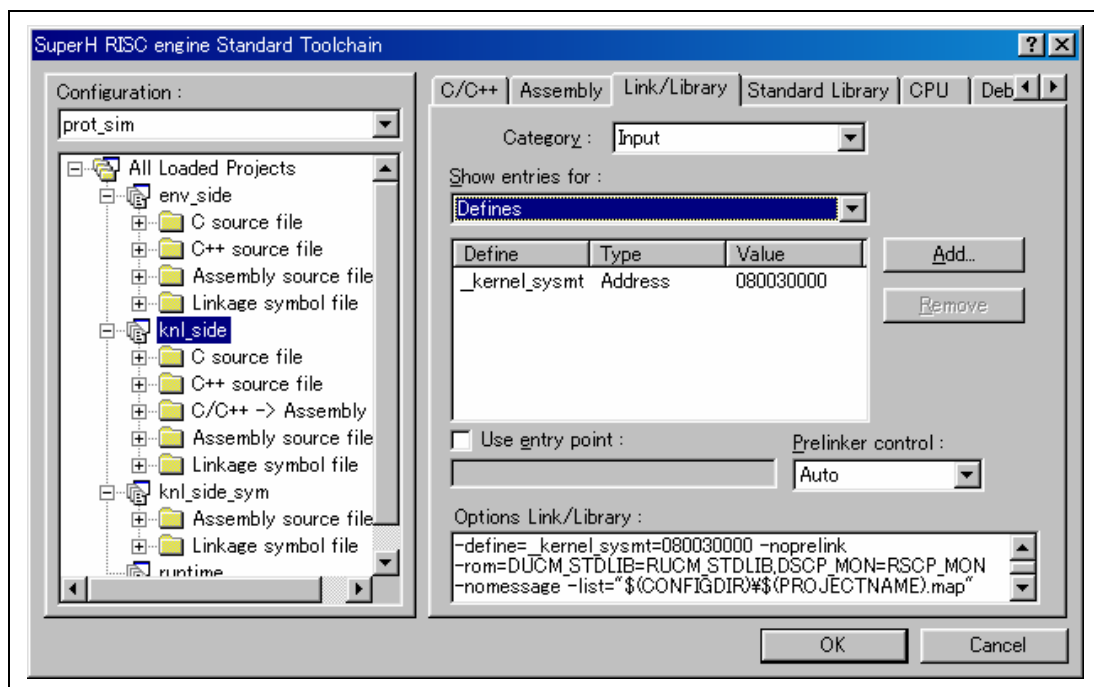


Figure 11.17 Linkage Editor Settings (__kernel_sysmt Address Definition)

(6) Allocating Sections

For section allocation, refer to section 11.15, Memory Allocation.

11.10.5 Build Execution

When a build is executed, the knl_side.abs and knl_side.fsy files are created in \$(WORKSPDIR)\knl_side\\$(CONFIGNAME)\.

11.11 knl_side_sym.hwp Project in kernel.hws

This project creates object file knl_side_sym.obj for the symbol file that has been generated in knl_side.hwp and outputs it in the directory having the same name as the configuration name under kernel_out\ . This object file is used to determine the addresses of the symbol references in knl_side when linkage units other than knl_side are linked.

This project performs the following processing.

- (1) knl_side_sym.obj that was generated at step (3) in the last build is deleted by DelFile.bat in the project directory.
- (2) knl_side.fsy that has been generated in \$(WORKSPDIR)\knl_side\\$(CONFIGNAME)\ through knl_side.hwp is copied to a file (knl_side_sym.fsy) in the project directory by CopyFile.bat which is also in the project directory.
- (3) The copied knl_side_sym.fsy file is assembled and resultant object file knl_side_sym.obj is output in \$(WORKSPDIR)\kernel_out\\$(CONFIGNAME)\.

11.12 runtime.hwp Project in kernel.hws

11.12.1 Overview

This project uses the standard library generator to generate library file runtime.lib, which contains runtime routines only, and outputs it in the directory having the same name as the configuration name under kernel_out. This project does not have source files.

runtime.lib is used to link runtime routines when linkage units other than knl_side are linked.

11.12.2 Standard Library Generator Settings

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "runtime" project in the left pane, then select the [Standard Library] tab. Select [Standard Library] for [Category:], and select only the runtime routines as shown in figure 11.18.

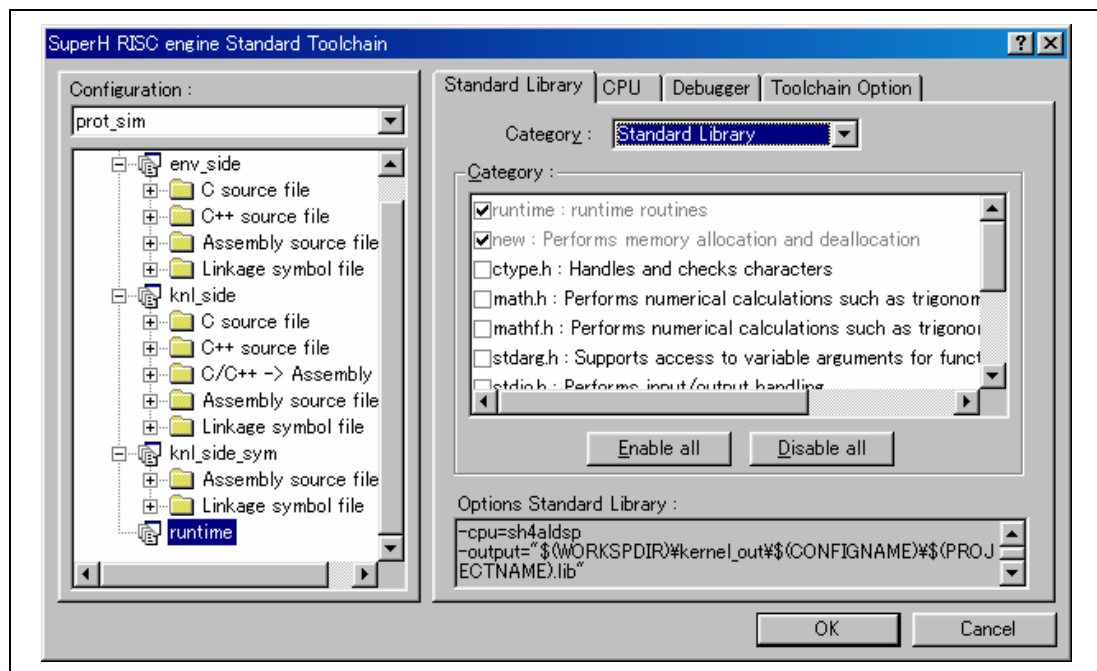


Figure 11.18 Standard Library Generator Settings ([Standard Library] Category)

Next, select [Object] for [Category:] and click the [Details...] button to open the dialog box shown in figure 11.19.

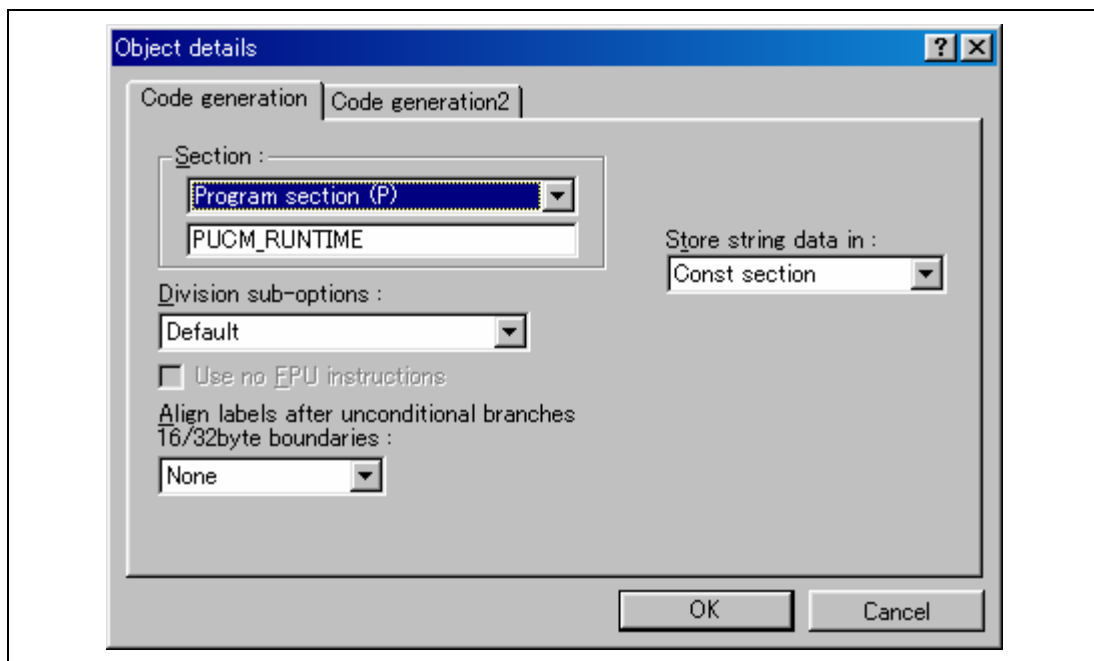


Figure 11.19 Standard Library Generator Settings ([Object details] Dialog Box)

In the [Section:] group box, specify the section names for the runtime routines as shown below.

- [Program section (P)]: PUCM_RUNTIME
- [Constant section (C)]: CUCM_RUNTIME
- [Initialized data section (D)]: DUCM_RUNTIME
- [Uninitialized data section (B)]: BUCM_RUNTIME

Then, select [Object] for [Category:], click the [Modify...] button, and specify the output file path as shown in figure 11.20 so that the runtime.lib file is output in \$(WORKSPDIR)\kernel_out\\$(CONFIGNAME)\.

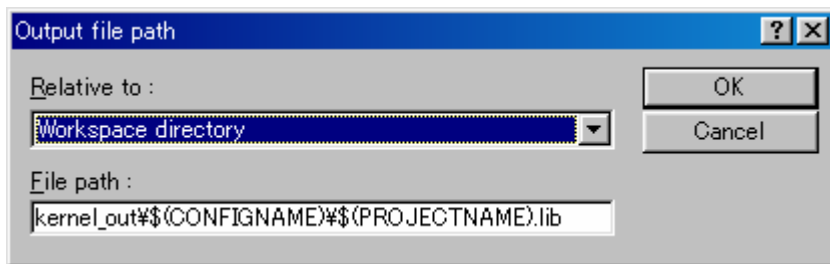


Figure 11.20 Standard Library Generator Settings ([Output file path] Dialog Box)

11.12.3 Build Execution

When a build is executed, `$(WORKSPDIR)\kernel_out\$(CONFIGNAME)\runtime.lib` is created.

11.12.4 Notes on Section Initialization

Refer to section 11.7.9, Runtime Routines.

11.13 env_side.hwp Project in kernel.hws

11.13.1 Overview

This project generates env_side.abs in kernel\env_side\\$(CONFIGNAME).\

env_side.abs contains domains 1 to 4. In addition, knl_side_sym.obj to determine symbol references in knl_side, runtime.lib to link runtime routines, and init_runtime.c to initialize runtime routines are linked through this project.

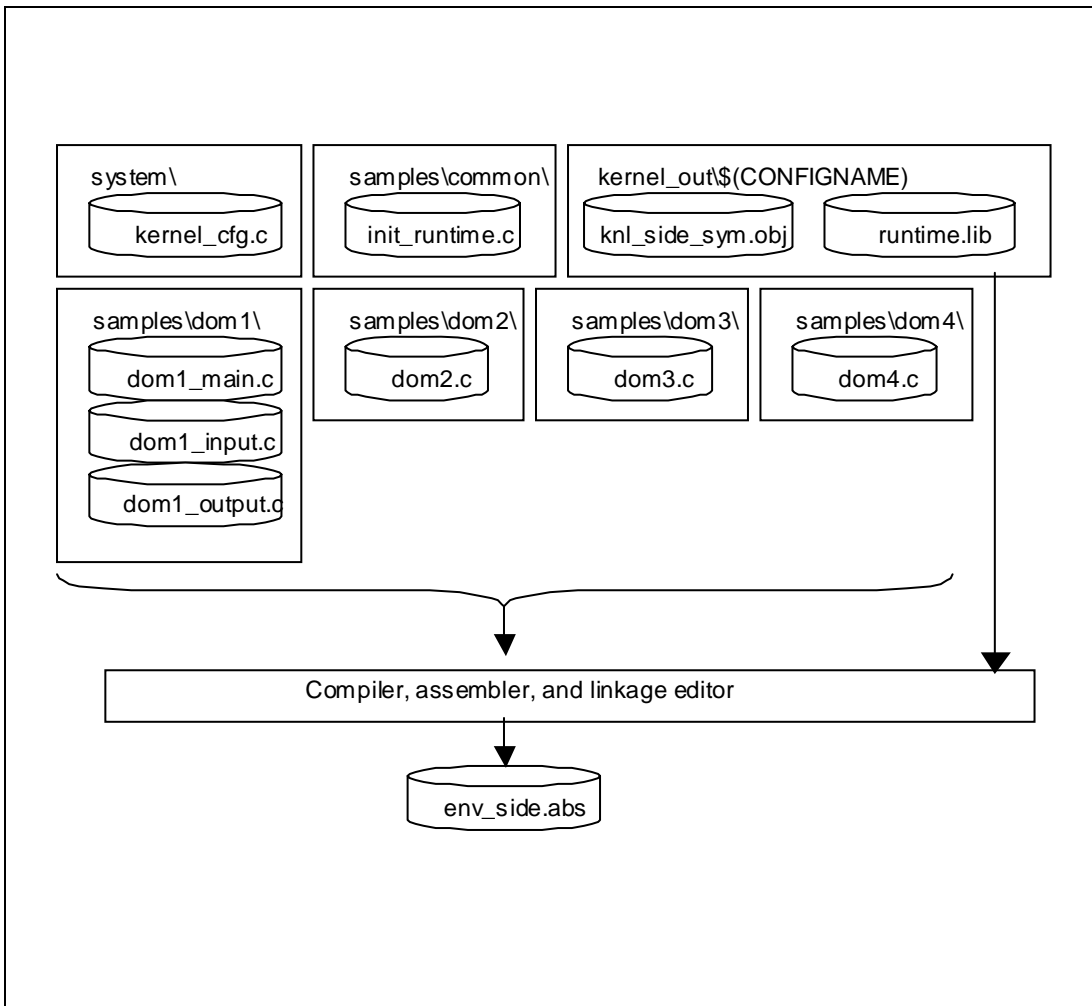


Figure 11.21 Build Phases for env_side.hwp

11.13.2 Source Files to Be Registered in Project

The following source files should be registered in the project.

- (1) `system\kernel_cfg.c`: Always necessary

One of the configurator output files. It contains the kernel environment-side information.

- (2) `samples\common\init_runtime.c`

A function for initializing runtime routines (`runtime.lib`).

(3) samples\dom1\dom1_main.c, dom1_input.c, and dom1_output.c

Sample files for domain 1.

(4) samples\dom2\dom2.c

A sample file for domain 2.

(5) samples\dom3\dom3.c

A sample file for domain 3.

(6) samples\dom4\dom4.c

A sample file for domain 4.

In addition, the symbols and sections of the objects for which [Kernel side] is not specified in the configurator must also be linked in this project.

11.13.3 Standard Library Generator Settings

runtime.lib generated through runtime.hwp in kernel.hws must be specified here.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "env_side" project in the left pane, then select the [Standard Library] tab.

Select [Mode] for [Category:], and specify kenrel_out\\$(CONFIGNAME)\runtime.lib as shown in figure 11.22.

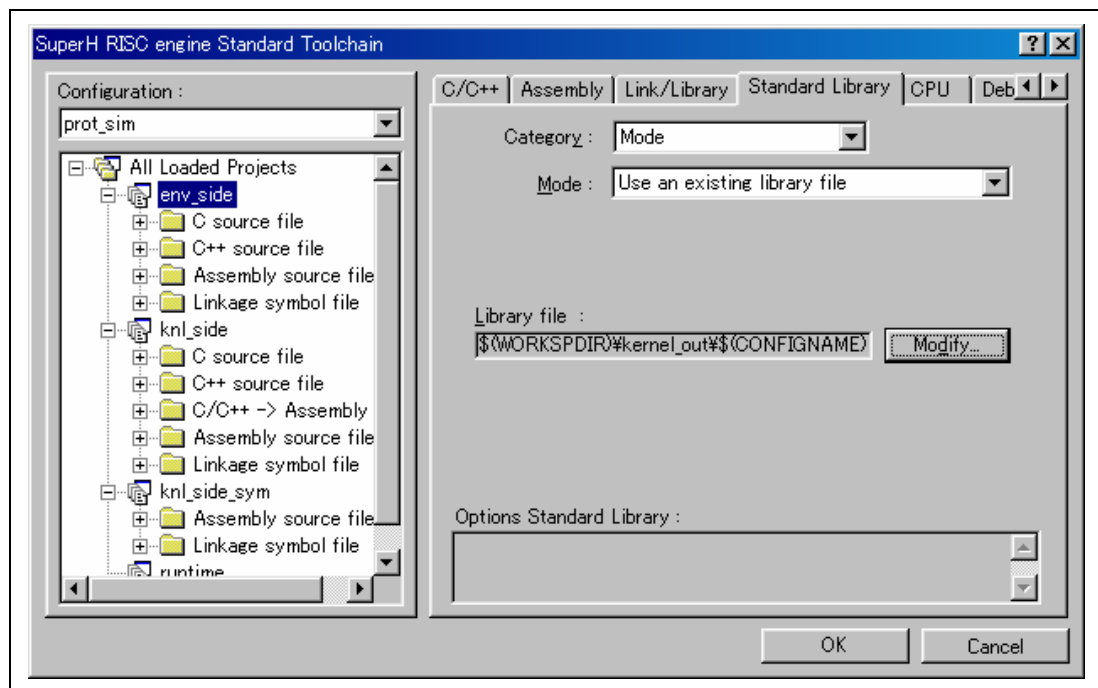


Figure 11.22 Standard Library Generator Settings (Specifying runtime.lib)

11.13.4 Linkage Editor Settings

(1) Specifying knl_side.sym.obj

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "env_side" project in the left pane, then select the [Link/Library] tab .

Select [Input] for [Category:] and [Relocatable files and object files] for [Show entries for:], then specify \$(WORKSPDIR)\kernel_out\\$(CONFIGNAME)\knl_side_sym.obj as shown in figure 11.23.

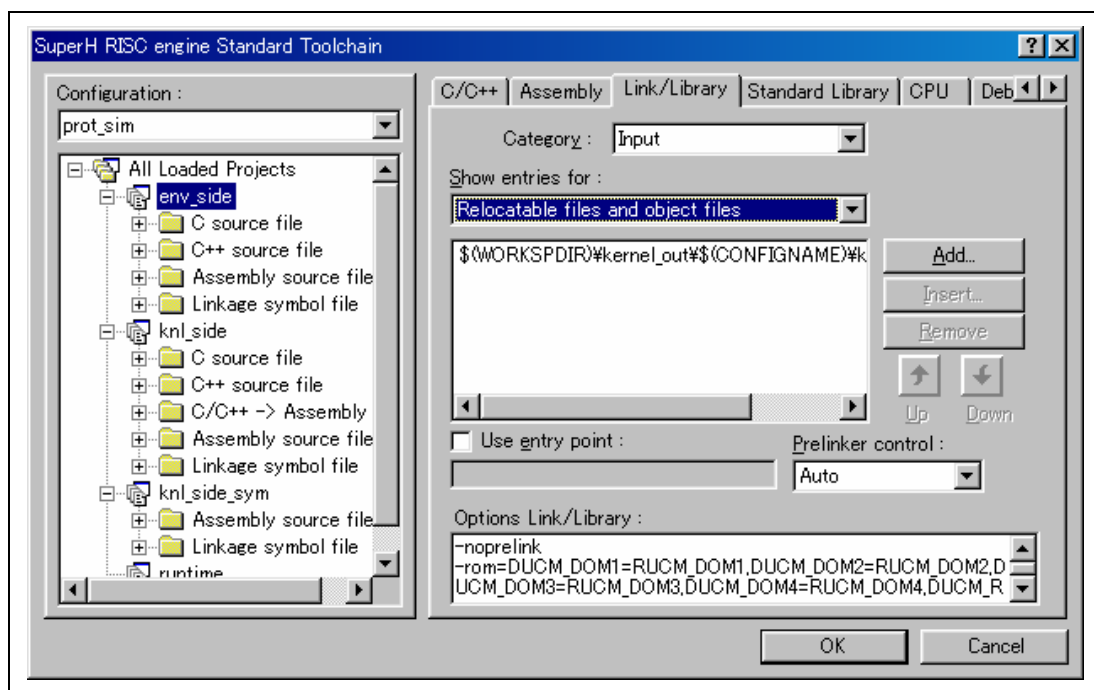


Figure 11.23 Linkage Editor Settings (Specifying knl_side_sym.obj)

(2) Settings for Initialized Data Sections

If an object to be linked includes an initialized data section (D section), [ROM to RAM mapped sections] (generation of R sections) must be specified in the linkage editor.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "env_side" project in the left pane, then select the [Link/Library] tab.

Select [Output] for [Category:] and [ROM to RAM mapped sections] for [Show entries for:].

The following initialized data sections are included in the configuration at shipment.

- DUCM_DOM1 (domain 1)
- DUCM_DOM2 (domain 2)
- DUCM_DOM3 (domain 3)
- DUCM_DOM4 (domain 4)
- DUCM_RUNTIME (runtime.lib)

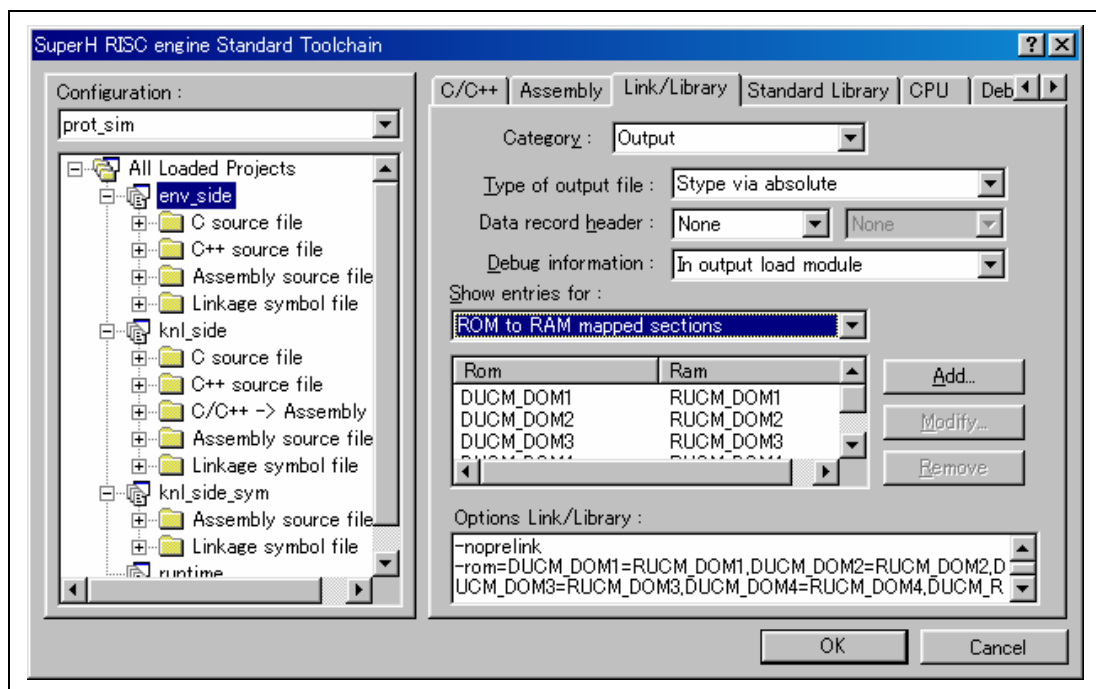


Figure 11.24 Linkage Editor Settings (Initialized Data Sections)

(3) Allocating Sections

For section allocation, refer to section 11.15, Memory Allocation.

11.13.5 Build Execution

When a build is executed, `$(WORKSPDIR)\env_side\$(CONFIGNAME)\env_side.abs` is created.

11.14 app_dom5.hwp Project in app_dom5.hws

11.14.1 Overview

This workspace shows an example of creation of a load module consisting of applications only.

This project generates app_dom5.abs.

app_dom5.abs contains domain 5. In addition, knl_side_sym.obj to determine symbol references in knl_side, runtime.lib to link runtime routines, and init_runtime.c to initialize runtime routines are linked through this project.

11.14.2 Source Files to Be Registered in Project

The following source files should be registered in the project.

(1) samples\common\init_runtime.c

A function for initializing runtime routines (runtime.lib).

(2) samples\dom5\dom5.c

A sample file for domain 5.

11.14.3 Standard Library Generator Settings

runtime.lib generated through runtime.hwp in kernel.hws must be specified here.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "app_dom5" project in the left pane, then select the [Standard Library] tab.

Select [Mode] for [Category:], and specify

\$(WORKSPDIR)\..\kernel\kernel_out\\$(CONFIGNAME)\runtime.lib as shown in figure 11.25.

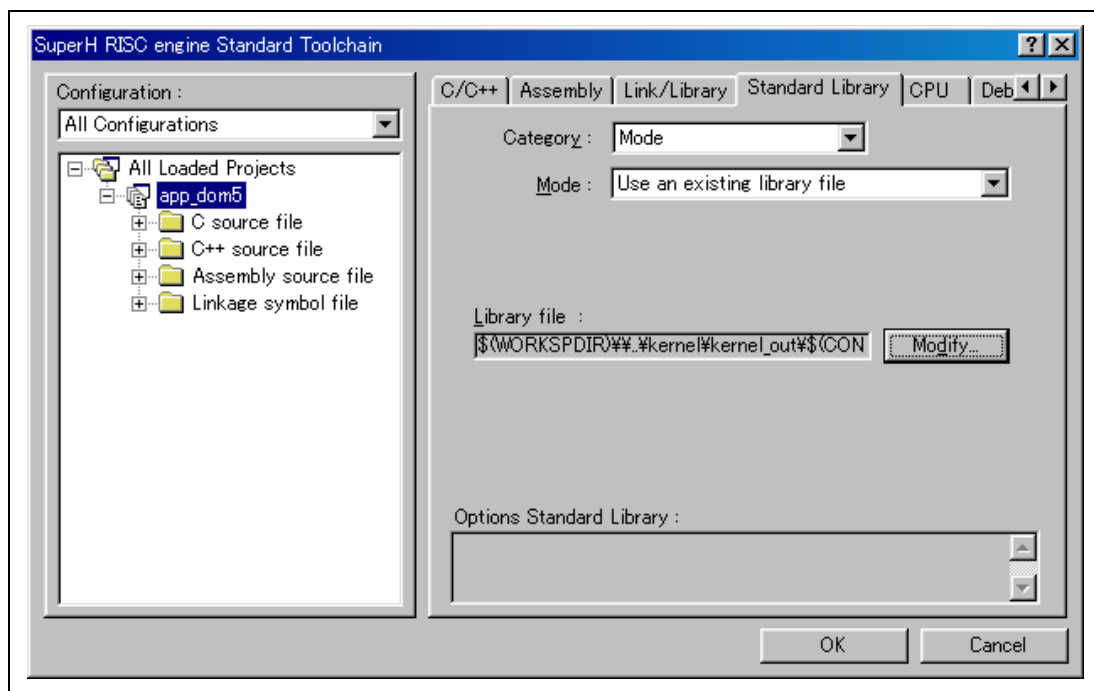


Figure 11.25 Standard Library Generator Settings (Specifying runtime.lib)

11.14.4 Linkage Editor Settings

(1) Specifying knl_side.sym.obj

Specify symbol file knl_side_sym.obj generated in knl_side_sym.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the "app_dom5" project in the left pane, then select the [Link/Library] tab.

Select [Input] for [Category:] and [Relocatable files and object files] for [Show entries for:], then specify \$(WORKSPDIR)\..\kernel\kernel_out\$(CONFIGNAME)\knl_side_sym.obj as shown in figure 11.26.

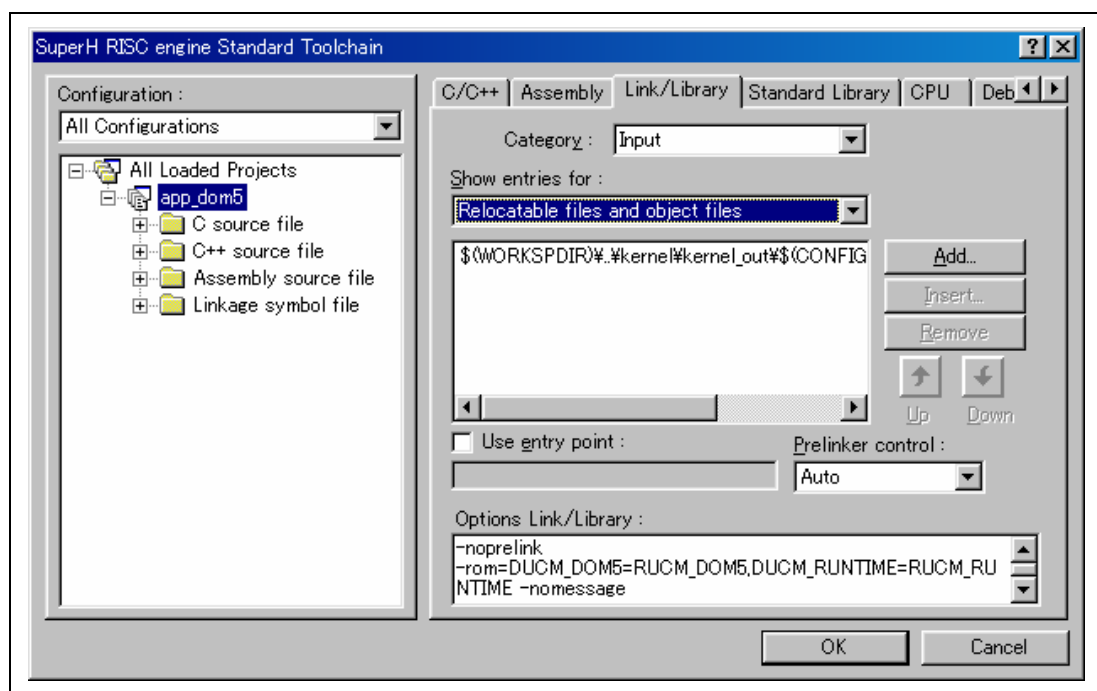


Figure 11.26 Linkage Editor Settings (Specifying knl_side_sym.obj)

(2) Settings for Initialized Data Sections

If an object to be linked includes an initialized data section (D section), [ROM to RAM mapped sections] (generation of R sections) must be specified in the linkage editor.

Select [Option -> SuperH RISC engine Standard Toolchain...] from the HEW menu to open the [SuperH RISC engine Standard Toolchain] dialog box.

Select the [Link/Library] tab.

Select [Output] for [Category:] and [ROM to RAM mapped sections] for [Show entries for:].

The following initialized data sections are included in the configuration at shipment.

- DUCM_DOM5 (domain 5)
- DUCM_RUNTIME (runtime.lib)

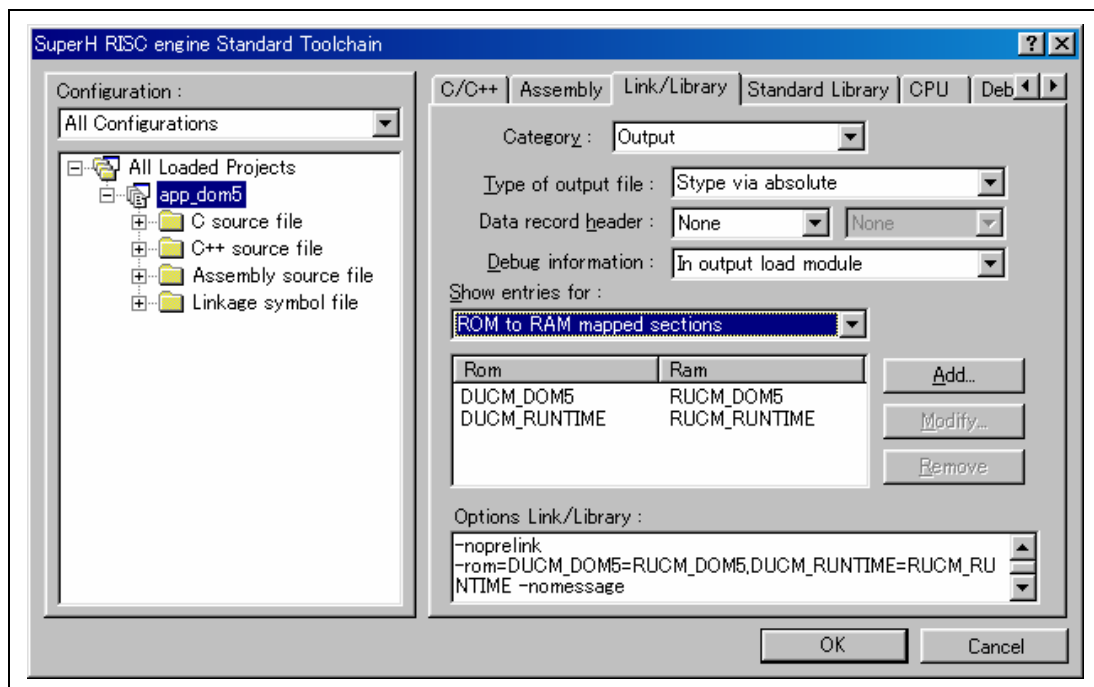


Figure 11.27 Linkage Editor Settings (Initialized Data Sections)

(3) Allocating Sections

For section allocation, refer to section 11.15, Memory Allocation.

11.14.5 Build Execution

When a build is executed, `$(WORKSPDIR)\app_dom5$(CONFIGNAME)\app_dom5.abs` is created.

11.15 Memory Allocation

11.15.1 Overview

This sample system consists of multiple load modules. The user must consider appropriate address allocation to every section in each load module, and the determined addresses must be specified through the linkage editor.

Through preliminary linkage without specifying section addresses, the size of each section can be obtained from the linkage map.

Although logical addresses must be used to specify the sections through the linkage editor, the physical memory areas to be allocated must also be considered. Generally, the area allocated to the P, C, D sections must be separated from the area for the B and R sections. The P, C, and D sections can be stored in ROM.

Reference: Section 5, Logical Address Space

11.15.2 Sections

Sections are named according to the following rules.

PSCP_hiknl

(1) First character

P: Program section

C: Constant section

B: Uninitialized data section

D: Initialized data section (ROM section)

R: Initialized data section (RAM section, which is generated with [ROM to RAM mapped sections] specified in the linkage editor)

(2) Second character

S: Never accessed in the user mode

U: Can be accessed in the user mode

(3) Third character

C: Cacheable access enabled

D: Cacheable access disabled

- (4) Fourth character (valid only when the memory object protection function is selected)
- (a) When the memory object protection function is used
 - P: Must be allocated to an MMU non-mapped area that cannot be accessed in the user mode.
 - M: Must be allocated to an MMU mapped area.
 - _ : Must be allocated to an area that can be accessed in the user mode, in either an MMU mapped area or an MMU non-mapped area.
 - (b) When the memory object protection function is unused
 - P: Must be allocated to an area that cannot be accessed in the user mode.
 - M: Must be allocated to an appropriate area (accessible or inaccessible in the user mode) according to the program that accesses the section.
 - _ : Must be allocated to an area that can be accessed in the user mode.

(1) Kernel Library (knl_sde.hwp)

Table 11.14 Sections for hiknl_big.lib and hiknl_little.lib

Section Name	Description
PSCP_hireset	vsta_knl
PSDP_hiknl	Program for manipulating registers such as MMUCR
PSCP_hiexp	Program generally executed when an interrupt or a TLB miss occurs
PSCP_hicom	Program executed for service calls in common
PUC__hiintfc	Entry function for service calls. This section must be readable from all user domains.
PSCP_hiknl	Others

(2) Cache Support Object (knl_sde.hwp)

Table 11.15 Sections for cache_sh4a_big.rel and cache_sh4a_little.rel

Section Name	Description
PSDP_hicac	Program for manipulating elements such as the CCR register.
PSCP_hicac	Other programs
BSCP_hicac	Management table

(3) system\kernel_def.c (knl_sde.hwp)

Table 11.16 Sections for kernel_def.c

Section Name	Description
PSCP_hidef	Functions such as those process initial object definitions
CSCP_hidef	-
BSCP_hidef	-
BSCP_himpp_<ID>	Protected memory pools (for which [Kernel side] is selected) <ID> indicates the ID name when available, or the ID number in decimal when the ID name is not available

(4) system\kernel_cfg.c (env_side.hwp)**Table 11.17 Sections for kernel_cfg.c**

Section Name	Description
PSCP_hicfg	Functions such as those process initial object definitions
CSCP_hidef	-
CSCP_hisysmt	Configuration information table on the kernel environment side
BSCP_hintskstk	Stack area for the non-task context
BUCM_hisyspl	System pool
BSCP_hirespl	Resource pool
BSCP_hiwrk	Kernel work area
BSCP_hitrbuf	Buffer for target trace
BSCP_hitooltrc	Area for tool trace
BSCP_hictxid	Always four bytes are allocated
BSCP_hicfg	-
BUCM_himpp_<ID>	Protected memory pools (for which [Kernel side] is not selected) <ID> indicates the ID name when available, or the ID number in decimal when the ID name is not available

(5) Sample Programs

Table 11.18 Sections for System Application

Linkage Unit	Section Name	File
knl_side	PSCP_hiknl	samples\sysapp\mavhdr.c samples\sysapp\sysdwn.c samples\sysapp\exchdr.c samples\sysapp\inthook.src
	BSCP_hiknl	samples\sysapp\sysdwn.c

Table 11.19 Sections for Target-Dependent Part

Linkage Unit	Section Name	File
knl_side	PSDP_RESET *	samples\shnnnn\kernel\knl_side\reset.src
	PSDP_RESET2	samples\shnnnn\kernel\knl_side\resetprg.c samples\shnnnn\kernel\knl_side\init_mmu.c
	PSCP_hiknl	samples\shnnnn\kernel\knl_side\tmrdrv.c
	CSCP_hiknl	samples\shnnnn\kernel\knl_side\tmrdrv.c

Note: * When it is stored in ROM, this section must be allocated at H'A0000000, which is the CPU reset address.

Table 11.20 Sections for Standard Library

Linkage Unit	Section Name	File
knl_side	PUCM_STDLIB	stdlib.lib (generated by standard library generator)
	CUCM_STDLIB	samples\stdlib\lowsr.c
	BUCM_STDLIB	samples\stdlib\initsct.c
	DUCM_STDLIB	
	RUCM_STDLIB	samples\stdlib\lowsr.c

Table 11.21 Sections for Monitor

Linkage Unit	Section Name	File
knl_side	PSCP_MON	samples\monitor\monitor.c
	CSCP_MON	
	BSCP_MON	
	DSCP_MON	
	RSCP_MON	

Table 11.22 Section for Idling Task

Linkage Unit	Section Name	File
knl_side	PSCP_IDLE	samples\idle\idle.c

Table 11.23 Sections for Domain 1

Linkage Unit	Section Name	File
env_side	PUCM_DOM1	samples\dom1\dom1_main.c
	CUCM_DOM1	samples\dom1\dom1_input.c
	BUCM_DOM1	samples\dom1\dom1_output.c
	DUCM_DOM1	
	RUCM_DOM1	

Table 11.24 Sections for Domain 2

Linkage Unit	Section Name	File
env_side	PUCM_DOM2	samples\dom2\dom2.c
	CUCM_DOM2	
	BUCM_DOM2	
	DUCM_DOM2	
	RUCM_DOM2	

Table 11.25 Sections for Domain 3

Linkage Unit	Section Name	File
env_side	PUCM_DOM3	samples\dom3\dom3.c
	CUCM_DOM3	
	BUCM_DOM3	
	DUCM_DOM3	
	RUCM_DOM3	

Table 11.26 Sections for Domain 4

Linkage Unit	Section Name	File
env_side	PUCM_DOM4	samples\dom4\dom4.c
	CUCM_DOM4	
	BUCM_DOM4	
	DUCM_DOM4	
	RUCM_DOM4	

Table 11.27 Sections for Domain 5

Linkage Unit	Section Name	File
app_dom5	PUCM_DOM5	samples\dom5\dom5.c
	CUCM_DOM5	
	BUCM_DOM5	
	DUCM_DOM5	
	RUCM_DOM5	

Table 11.28 Sections for runtime.lib

Linkage Unit	Section Name	File
Other than knl_side	PUCM_RUNTIIME	runtime.lib (generated by standard library generator)
	CUCM_RUNTIIME	samples\common\init_runtime.c
	BUCM_RUNTIIME	
	DUCM_RUNTIIME	
	RUCM_RUNTIIME	

Table 11.29 Section for knl_side_sym.obj

Linkage Unit	Section Name	File
Other than knl_side	P *	samples\shnnnn\kernel_out\\$(CONFIGNAME)\knl_side_sym.obj

Note: * The size of this section is 0 byte. Since no program accesses this section, it can be allocated to any address at linkage.

11.15.3 Notes

(1) When Memory Object Protection Function Is Used

Note the following regarding section allocation for static memory objects, system pool (BUCM_hisyspl section), and protected memory pools.

- The start address of a section must be aligned on a boundary of the specified page size for static memory objects, or a boundary of the default page size (4 Kbytes) for the system pool and protected memory pools.
- No data must be stored in the area between the end address of the section and the next page boundary.
- The sections must be allocated in MMU mapped areas.

(2) When Memory Object Protection Function Is Not Used

The system pool (BUCM_hisyspl section) must be allocated on a 32-byte boundary; otherwise, when a variable-size memory pool is allocated in the system pool, memory blocks are not aligned on the desired boundaries even if the VTA_ALIGN16 or VTA_ALIGN32 attribute is specified. In addition, the system pool must be allocated to an area that can be accessed in the user mode.

11.15.4 Memory Map and Static Memory Objects

The memory map for each sample differs depending whether the configuration assumes the use of the simulator: "prot_sim" and "noprot_sim" assume the use of the simulator, and "prot" and "noprot" do not.

Table 11.30 Memory Map in Each Configuration

Configuration	Physical Memory for Section Allocation *	
	P, C, and D Sections	B and R Sections
"prot_sim", "noprot_sim"	From address 0	From address H'0C000000 (an address for allocation to an external RAM area)
"prot", "noprot"	An address for allocation to an external RAM area	An address for allocation to an external RAM area
Note: * For exact addresses, refer to the settings of each sample project.		

In "prot_sim" and "noprot_sim", the system can be started by clicking [Reset Go] in the simulator.

In "prot" and "noprot", the memory map is determined assuming that the application is executed after it is downloaded to the external RAM on the target board.

Note that the on-chip memory is not used in any of these configurations.

The approximate memory sizes used by the sample system are as follows.

- P, C, and D sections: About 310 Kbytes
- B and R sections: About 440 Kbytes

The following describes the memory map of "prot_sim" and "noprot_sim" for the SH73180 as an example.

Figure 11.28 shows the P, C, and D section allocation, and figure 11.29 shows the B and D section allocation.

Physical address	Logical address	[knl_side]	[env_side]	[app_dom5]	User-mode access not allowed	Non-cacheable	MMU mapped area	Static memory object
H'00000000	H'A0000000	PSDP_RESET PSDP_RESET2 PSDP_hiknl PSDP_hicac						
H'00001000	H'80001000	PSCP_hiexp PSCP_hicom PSCP_hiknl PSCP_hireset PSCP_hidef CSCP_hiknl CSCP_hidef PSCP_hicac PSCP_MON CSCP_MON DSCP_MON PSCP_IDLE						
H'00020000	H'00020000	PUC_hiintfc PUCM_STDLIB CUCM_STDLIB DUCM_STDLIB						
H'00030000	H'80030000		CSCP_hisysmt PSCP_hicfg CSCP_hicfg P *					
H'00040000	H'00040000		PUCM_RUNTIME CUCM_RUNTIME DUCM_RUNTIME					
H'00042000	H'00042000		PUCM_DOM1 CUCM_DOM1 DUCM_DOM1					
H'00044000	H'00044000		PUCM_DOM2 CUCM_DOM2 DUCM_DOM2					
H'00046000	H'00046000		PUCM_DOM3 CUCM_DOM3 DUCM_DOM3					
H'00048000	H'00048000		PUCM_DOM4 CUCM_DOM4 DUCM_DOM4					
H'0004A000	H'0004A000			PUCM_DOM5 CUCM_DOM5 DUCM_DOM5 P * PUCM_RUNTIME CUCM_RUNTIME DUCM_RUNTIME				
H'0004C000	H'0004C000							

Note: Symbol file knl_side_sym.obj uses a P section with a size of 0 byte.

Figure 11.28 Allocation of P, C, and D Sections

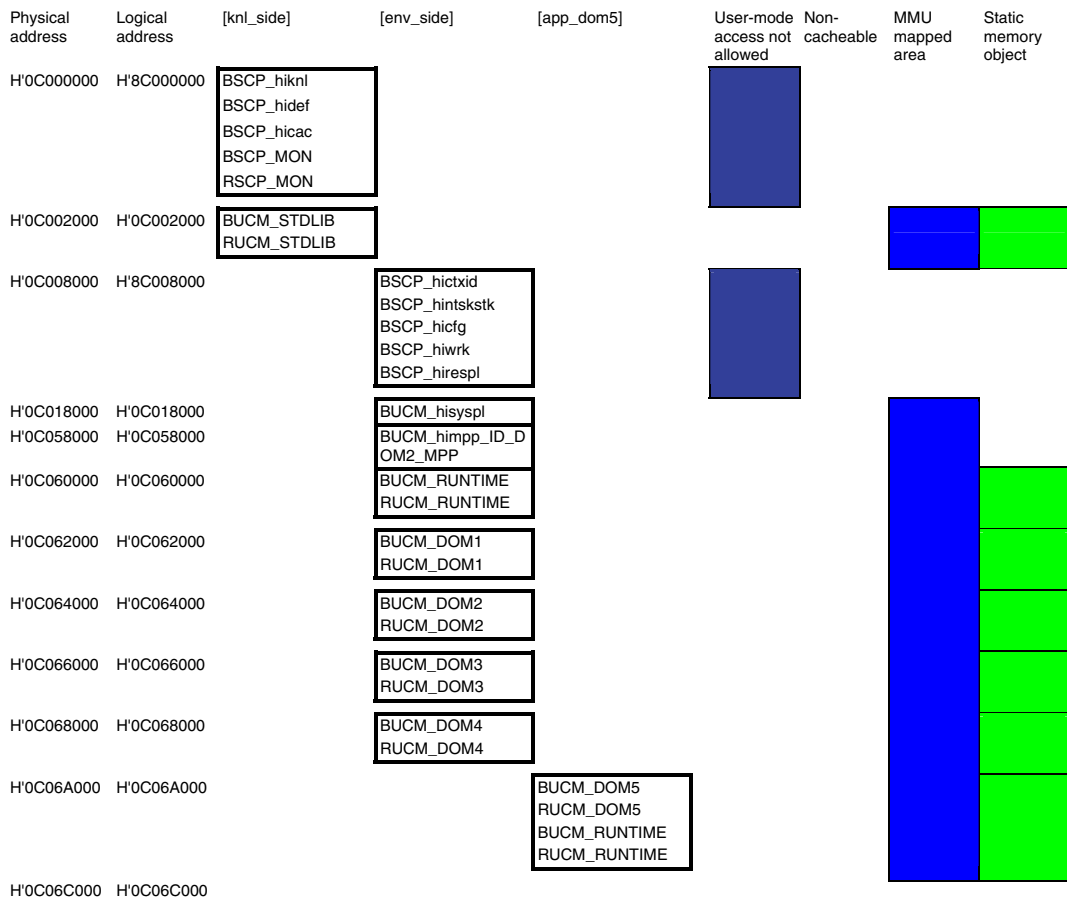


Figure 11.29 Allocation of B and R Sections

The following describes section allocation for each linkage unit in detail.

(1) kn1_side

(a) Section block at logical address H'A0000000 (such as PSDP_RESET)

PSDP_RESET (reset.src) is a program section that must be executed immediately after a CPU reset, and so it is allocated at the CPU reset address (H'A0000000). The other sections can be allocated in any order.

(b) Section block at logical address H'80001000 (such as PSCP_hiexp)

This block contains the sections that are read only in the privileged mode. These sections can be allocated in any order.

(c) Section block at logical address H'00020000 (such as PUC__hiintfc)

This block contains the sections that can be read from any user domain.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section PUC__hiintfc to section DUCM_STDLIB
- Page size: 64 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_SRO (can be read from any domain)

(d) Section block at logical address H'8C000000 (such as BSCP_hiknl)

This block contains the sections that are read and written to only in the privileged mode. These sections can be allocated in any order.

(e) Logical address H'0C002000 (such as BUCM_STDLIB)

This block contains the sections that can be read or written to from any user domain.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section BUCM_STDLIB to section RUCM_STDLIB
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_SRW (can be read and written to from any domain)

(2) env_side

(a) Section block at logical address H'80030000 (such as CSCP_hisysmt)

This block contains the sections that are read only in the privileged mode.

CSCP_hisysmt is the section for the kernel environment-side information table (`__kernel_sysmt`). The start address of this section must be determined during system design in advance, and the determined address must be specified at linkage of `knl_side`. To ensure that the start address remains unchanged even if the sizes of individual sections are changed during development of the system, it is recommended that CSCP_hisysmt section be allocated at the head of the section block.

The other sections can be allocated in any order.

(b) Section block at logical address H'00040000 (such as PUCM_RUNTIME)

This block contains the sections that can be read from any domain in `env_side`.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section PUCM_RUNTIME to section DUCM_RUNTIME
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_SRO (can be read from any user domain)
- (c) Section block at logical address H'00042000 (such as PUCM_DOM1)
- This block contains the sections that are read only from domain 1.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.
- Address range: Section PUCM_DOM1 to section DUCM_DOM1
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_PRO(ID_DOM1) (can be read only from domain 1)
- (d) Section block at logical address H'00044000 (such as PUCM_DOM2)
- This block contains the sections that are read only from domain 2.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.
- Address range: Section PUCM_DOM2 to section DUCM_DOM2
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_PRO(ID_DOM2)) (can be read only from domain 2)
- (e) Section block at logical address H'00046000 (such as PUCM_DOM3)
- This block contains the sections that are read only from domain 3.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.
- Address range: Section PUCM_DOM3 to section DUCM_DOM3
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_PRO(ID_DOM3)) (can be read only in domain 3)
- (f) Section block at logical address H'00048000 (such as PUCM_DOM4)
- This block contains the sections that are read only from domain 4.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section PUCM_DOM4 to section DUCM_DOM4
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_PRO(ID_DOM4)) (can be read only from domain 4)
- (g) Section block at logical address H'8C008000 (such as BSCP_hictxid)
- This block contains the sections that are read and written to only in the privileged mode. These sections can be allocated in any order.
- (h) Section block at logical address H'0C0180000 (BUCM_hisyspl)
- This block contains the system pool section.
- (i) Section block at logical address H'0C0580000 (BUCM_himpp_ID_DOM2MPP)
- This block contains the section for the protected memory pool registered through the configurator.
- This sample specifies H'8000 as the protected memory pool size.
- The "noprot_sim" and "noprot" configurations do not have this section.
- (j) Section block at logical address H'0C060000 (such as BUCM_RUNTIME)
- This block contains the sections that can be read and written to from any domain in env_side.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.
- Address range: Section BUCM_RUNTIME to section RUCM_RUNTIME
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_SRW (can be read and written to from any domain)
- (k) Section block at logical address H'0C062000 (such as BUCM_DOM1)
- This block contains the sections that are read and written to only from domain 1.
- When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.
- Address range: Section BUCM_DOM1 to section RUCM_DOM1
 - Page size: 4 Kbytes
 - Cache attribute: Copy-back
 - Access permission vector: TACT_PRW(ID_DOM1) (can be read and written to only from domain 1)
- (l) Section block at logical address H'0C064000 (such as BUCM_DOM2)
- This block contains the sections that are read and written to only from domain 2.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section BUCM_DOM2 to RUCM_DOM2
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_PRW(ID_DOM2) (can be read and written to only from domain 2)

(m) Section block at logical address H'0C066000 (such as BUCM_DOM3)

This block contains the sections that are read and written to only from domain 3.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section BUCM_DOM3 to RUCM_DOM3
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_PRW(ID_DOM3) (can be read and written to only from domain 3)

(n) Section block at logical address H'0C068000 (such as BUCM_DOM4)

This block contains the sections that are read and written to only from domain 4.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the order of the sections specified through the linkage editor should match the settings in the configurator.

- Address range: Section BUCM_DOM4 to RUCM_DOM4
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_PRW(ID_DOM4) (can be read and written to only from domain 4)

(3) app_dom5

app_dom5 is a load module consisting of domain 5 only. Accordingly, all of its sections including the runtime routines are accessed only from domain 5.

(a) Section block at logical address H'0004A000 (such as PUCM_DOM5)

This block contains the sections that are read only from domain 5.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the section addresses specified through the linkage editor and the resultant end address should match the settings in the configurator.

- Address range: H'2000 (8192) bytes starting from H'0004A000
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_PRO(ID_DOM5) (can be read only from domain 5)

(b) Section block at logical address H'0C06A000 (such as BUCM_DOM5)

This block contains the sections that are read and written to only from domain 5.

When the memory object protection function is selected, this block is registered as a static memory object having the following attributes through the configurator. Note that the section addresses specified through the linkage editor and the resultant end address should match the settings in the configurator.

- Address range: H'2000 (8192) bytes starting from H'0C06A000
- Page size: 4 Kbytes
- Cache attribute: Copy-back
- Access permission vector: TACT_PRW(ID_DOM5) (can be read and written to only from domain 5)

11.16 Execution on Simulator

11.16.1 Debugging Session

Debugging sessions for execution on the simulator are provided for the following projects.

- knl_side.hwp in kernel.hws
- envl_side.hwp in kernel.hws
- app_dom5.hwp in app_dom5.hws

The debugging session name depends on the microprocessor type (*shnnnn*); for example, "sim_sh4aldsp-cyc_env_side".

In the debugging sessions for knl_side.hwp and env_side.hwp, only knl_side.abs and env_side are downloaded and domain 5 is not downloaded.

In the debugging session for app_dom5.hwp, app_dom5.abs is downloaded in addition to knl_side.abs and env_side.abs. Therefore, the main task of domain 5 can be initiated through the monitor in the case of app_dom5.hwp.

Note: V.1.01 Release 00 of the HI7300/PX provides only debugging sessions for *sh73180* (SH73180 microcomputer).

Almost the same settings are made for these three debugging sessions. Note the following settings.

- (1) Mapping has been specified
- (2) I/O simulation: Enabled (I/O system call address = 4)
- (3) Timer simulation: Enabled
- (4) At interrupt or exception occurrence: Execution continues (if execution stop is specified, simulation stops when a TLB miss exception occurs).

These settings are made through the HEW command file (extension "hdc") in the workspace directory. Each session is set up so that this command file is automatically executed when the simulator is connected and when a program is downloaded.

In addition to the above settings, this command file contains the processing for disabling the MMU because a program cannot be downloaded if a TLB miss occurs during the downloading process when the MMU is enabled.

11.16.2 Execution

To execute a program, select [Debug] -> [Download] -> [All Download Modules] in each debugging session, reset the CPU, and then execute it.

11.16.3 Monitor Startup

In the "prot_sim" and "noprot_sim" configurations, the monitor can be used.

Click the trigger button indicated as [Monitor] to start the monitor (figure 11.30); monitor prompt "MON>" appears in the [Simulated I/O] window.

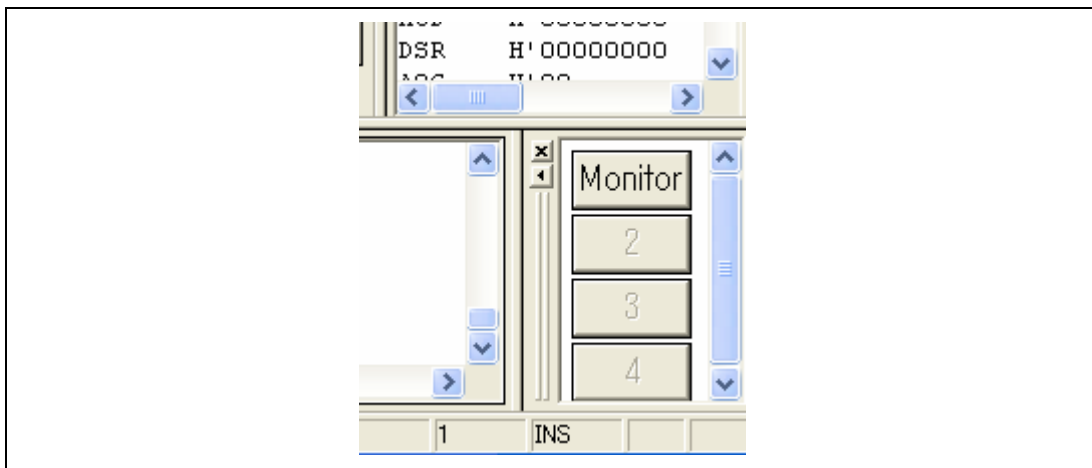


Figure 11.30 Trigger Button for Starting the Monitor

11.16.4 Detection of Illegal Access by Domain 4

Domain 4 is a sample application for intentionally making illegal access.

When the memory object protection function is selected ("prot_sim" configuration), the MMU or CPU detects this illegal access and the memory access violation handler (sysapp\mavhdr.c) sends a request for task exception processing to the main task in domain 4. As a result, the program line after the access violation is not executed and the task exception processing routine is executed before that line. The task exception processing routine displays an access violation message in the simulated I/O window, and restarts the task to generate access violation again.

This procedure for access violation exception can be monitored by executing domain 4 with the "prot_sim" configuration.

When domain 4 is executed with the "noprot_sim" configuration, in which the memory object protection function is not selected, domain 4 makes an illegal access in the same way as in the "prot_sim" configuration. However, since the memory object protection function is not used, only the access violations that can be detected by the CPU (access to addresses larger than H'80000000 in the user mode) are detected.

For details, refer to the source code of domain 4 and set breakpoints at appropriate locations to confirm the actual behavior.

11.16.5 Execution of Domain 5

A task is created for domain 5 by the configurator, but it is not initiated.

To initiate it, enter the act command through the monitor as follows.

```
MON> act 5 (RET)
```

The task ID for domain 5 is specified for automatic assignment with an ID name "ID_DOM5_MAIN" through the configurator. The actual ID number is output by the configurator to kernel_id.h in samples\shnnnn\config_out\\$(CONFIGNAME)\ as "ID_DOM5_MAIN". At shipment, this task ID number is 5.

Section 12 Calculation of Stack Size

12.1 Stack Types

This kernel has the following types of stacks.

(1) Task stack

Each task has a different stack.

A task associated with a user domain has two stacks. One stack is used by the task itself and the other stack is a system stack. The system stack is used by extended service call routines and trap routines called by the task. It is also used by the kernel for saving task context.

A task associated with the kernel domain has only one stack. This stack is used not only by the task itself but also by extended service call routines and trap routines called by the task. It is also used by the kernel for saving task context.

(2) Non-task context stack

This is a stack used, as its name shows, for execution of non-task context. The kernel switches the stack pointer to the non-task context stack when transiting from the task context state to the non-task context state.

There is only one non-task context stack in the system.

12.2 Overview of Calculation Procedure for Stack Size

The stack size used by each task and handler is calculated by tracking back the stack size used from the nested subroutines (functions) of the start function. This size is called the "independent stack size".

The actual required size is the value obtained by adding the stack size used by the kernel to the "independent stack size".

12.3 Stack Size Used by Each Task

The stack size of each task is specified at task creation. Calculate the value to be specified with reference to the subsequent sections.

12.3.1 Task Associated with User Domain

A task associated with a user domain has a system stack in addition to a stack used by the task.

(1) Stack used by task

$$\begin{aligned}\text{Stack size} &= (\text{Independent stack size used by task}) \\ &+ (\text{Independent stack size used by task exception processing routine})\end{aligned}$$

When the task exception processing routine is nested at initiation, calculate the total size with programming nesting added.

(2) System stack

$$\begin{aligned}\text{Stack size} &= (\text{Context saving stack size of task}) \\ &+ (\text{Independent stack size used by extended service call routine or trap routine that is called}) \\ &+ (\text{Context saving stack size of extended service call routine, trap routine, or task exception processing routine})\end{aligned}$$

The context saving stack size is the size for saving the registers necessary for program execution. The context saving stack size of a task is necessary in all cases. The context saving stack size of an extended service call routine, trap routine, or task exception processing routine is necessary each time it is initiated.

Table 12.1 shows the context saving stack sizes.

Table 12.1 Context Saving Stack Size

Classification	Size	Necessary Condition
Task	180	Always necessary
	56	For the TA_COP0 attribute
	72	For the TA_COP1 attribute
	64	For the TA_COP2 attribute
Extended service call routine, trap routine, or task exception processing routine	112	Always necessary
	56	For the TA_COP0 attribute
	72	For the TA_COP1 attribute
	64	For the TA_COP2 attribute

When an extended service call routine, trap routine, or task exception processing routine is nested, add its size considering programming nesting.

(3) Calculation example

An example of calculation is shown with figure 12.1 used as an example. Numeric values enclosed in square brackets in the figure are independent stack sizes. It is assumed that the TA_COP0, TA_COP1, and TA_COP2 attributes are not specified in this example.

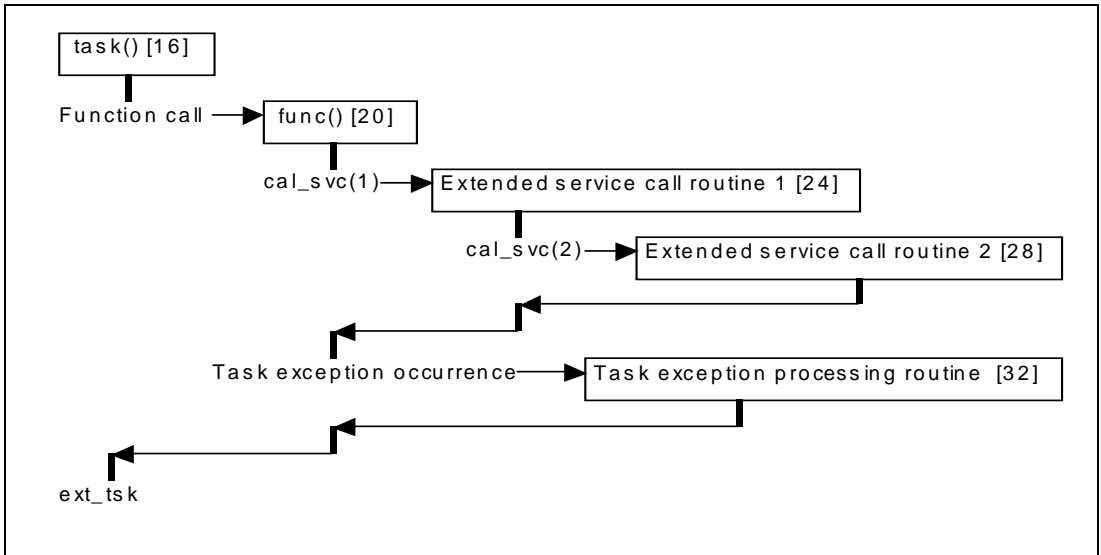


Figure 12.1 Calculation Example of Stack Size Used by Task

(a) Stack used by task

$$\begin{aligned}\text{Stack size} &= (\text{Independent stack size used by task}) \\ &\quad + (\text{Independent stack size used by task exception processing routine}) \\ &= (16 + 20) + 32 \\ &= 68\end{aligned}$$

(b) System stack

— When extended service call routine 2 is nested

$$\begin{aligned}\text{Stack size} &= (\text{Context saving stack size of task}) \\ &\quad + (\text{Independent stack size used by extended service call routine or trap} \\ &\quad \quad \text{routine that is called}) \\ &\quad + (\text{Context saving stack size of extended service call routine, trap routine,} \\ &\quad \quad \text{or task exception processing routine}) \\ &= 180 \\ &\quad + (24 + 28) \\ &\quad + (112 + 112) \\ &= 456\end{aligned}$$

- When task exception processing routine is nested
 - Stack size = (Context saving stack size of task)
 - + (Independent stack size used by extended service call routine or trap routine that is called)
 - + (Context saving stack size of extended service call routine, trap routine, or task exception processing routine)
 - = 180
 - + 0
 - + 112
 - = 292

Accordingly, the stack size used by the system stack is 456.

12.3.2 Task Associated with Kernel Domain

A task associated with the kernel domain has only one stack.

The stack size used is the sum of the calculated results of "section 12.3.1 (1) Stack used by task" and "section 12.3.1 (2) System stack".

12.4 Calculation of Non-Task Context Stack Size

There is only one non-task context stack in the system, and its size is specified by CFG_NTSKSTKSZ.

The value set in CFG_NTSKSTKSZ must be equal to or greater than the larger size of size 1 or size 2 shown below.

- Size 1
 - = 256
 - + (Maximum stack size used by each initialization routine) (1)
 - + (Stack size used by NMI interrupt handler) × (NMI nest count) (2)
- Size 2
 - = 256
 - + (Maximum stack size used by CPU exception handler occurring during task context execution) (3)
 - + Σ (Maximum stack size used by interrupt handler at each interrupt level) (4)
 - + (Stack size used by NMI interrupt handler) × (NMI nest count) (2)

(1) For each initialization routine, calculate the stack size used by referring to section 12.4.1, Stack Size Used by Each Initialization Routine and Timer Initialization Routine of Standard Timer Driver, and obtain the maximum stack size within all initialization routines.

- (2) For calculation of the stack size used by the NMI interrupt handler, refer to section 12.4.3, Stack Size Used by NMI Interrupt Handler. For the NMI nest count, set the NMI nest count to a possible number if the setting to accept the NMI interrupt is made in the interrupt controller when the BL bit in SR is 1. In other cases, set the NMI nest count to 1.
- (3) For a CPU exception that may occur during execution of task context, calculate the stack size used by referring to section 12.4.4, Stack Size Used by Each CPU Exception Handler, and obtain the maximum stack size among all CPU exception handlers.
- (4) Calculate the stack size used by individual handlers according to section 12.4.2, Stack Size Used by Each Interrupt Handler, Time Event Handler, and Timer Interrupt Routine of Standard Timer Driver, and obtain the maximum stack size within handlers for each interrupt level. Then, add the stack sizes for all interrupt levels.

Reference: Section 8.4.4, Notes on NMI

12.4.1 Stack Size Used by Each Initialization Routine and Timer Initialization Routine of Standard Timer Driver

Stack size = (Independent stack size used by initialization routine)
 + 216 (Required only when service call is issued)
 + (Independent stack size used by extended service call routine or trap routine that is called)
 + (Context saving stack size of extended service call routine or trap routine)
 + (Stack size used by CPU exception handler)
 + (140 whenever CPU exception occurs)

The context saving stack size is in accordance with table 12.1.

When an extended service call routine or trap routine is called, add its size considering programming nesting.

The stack size used by a CPU exception handler is required when a CPU exception occurs during execution of an initialization routine whose used stack size is to be calculated. For the calculation method, refer to section 12.4.4, Stack Size Used by Each CPU Exception Handler.

Note that the initialization routine (`_kernel_tmrini()`) of the standard timer driver must also be calculated by using this equation.

12.4.2 Stack Size Used by Each Interrupt Handler, Time Event Handler, and Timer Interrupt Routine of Standard Timer Driver

Stack size = (Independent stack size used by interrupt handler)
+ 76
+ 216 (Required only when service call is issued)
+ (Independent stack size used by extended service call routine or trap routine that is called)
+ (Context saving stack size of extended service call routine or trap routine)
+ (Stack size used by CPU exception handler)
+ (140 whenever CPU exception occurs)

The context saving stack size is in accordance with table 12.1.

The stack size used by a CPU exception handler is required when a CPU exception occurs during execution of a handler or routine whose used stack size is to be calculated. For the calculation method, refer to section 12.4.4, Stack Size Used by Each CPU Exception Handler.

When an extended service call routine or trap routine is called, add its size considering programming nesting.

Note that the time event handler, and timer interrupt routine (`_kernel_tmrint()`) of the standard timer driver must be calculated by this equation as an interrupt handler whose interrupt level is `CFG_KNLLVL`. However, 160 must be added to the calculation.

12.4.3 Stack Size Used by NMI Interrupt Handler

Stack size = (Independent stack size used by NMI interrupt handler)
+ (Stack size used by CPU exception handler)
+ (140 whenever CPU exception occurs)

The stack size used by a CPU exception handler is required when a CPU exception occurs during execution of the NMI interrupt handler. For the calculation method, refer to section 12.4.4, Stack Size Used by Each CPU Exception Handler. When a CPU exception handler is nested, add its size considering programming nesting.

12.4.4 Stack Size Used by Each CPU Exception Handler

Stack size = (Independent stack size used by CPU exception handler)
+ (Stack size used by CPU exception handler)
+ 216 (Required only when service call is issued)
+ (Stack size used by additional CPU exception handler)
+ (140 whenever additional CPU exception occurs)

An additional CPU exception is the CPU exception that occurs during execution of a CPU exception handler whose used stack size is to be calculated. When a CPU exception handler is nested, add its size with regard to programming nesting.

Section 13 Estimation of Resource Pool Size

13.1 Overview

The resource pool size is specified by CFG_RESPOOLSZ.

The resource pool is mainly used to dynamically allocate the kernel internal management table.

Memory can be recycled by acquiring memory when required and releasing it when no longer required. Compared to allocating memory statically, the amount of memory used can be reduced with this method.

On the other hand, since the state of resource pool usage depends on the system status, it is generally difficult to calculate the minimum required size accurately.

If the resource pool is insufficient during system operation, the system enters an erroneous state, in which, for example, service calls (functions) result in E_NOMEM errors. Roughly estimate the required size using the following procedure, and specify a sufficient size in CFG_RESPOOLSZ.

1. Determine the timing at which the resource pool is used and its requested size → Section 13.2, Requested Timing and Size
2. Estimate the required size taking into consideration the algorithm described below. To be specific, make an estimation with reference to section 13.3, Calculation.

Reference: Section 4.31, Controlling Memory Fragmentation

In this section, the following symbols are used.

ROUND_UP(a, b): a is rounded up to a multiple of b

13.2 Requested Timing and Size

13.2.1 When Kernel is Started (vsta_knl)

(1) Management of static memory objects

For all static memory objects, the total sum of the sizes calculated by VTSZ_MEMMB (page size, static memory object size) is used. The contents of this macro are shown below.

```
VTSZ_MEMMB(page size, static memory object size)
    =(ROUND_UP(static memory object size, page size)+CFG_PAGESZ(4096)×12)+4
```

This area is never released.

13.2.2 When Object is Created

The resource pool is requested when the following objects are created. The area used at this time is released when that object is deleted.

(1) Task

(a) System stack allocated from resource pool

When the kernel is specified to allocate the system stack area at task creation, (system stack size + 4) bytes of the resource pool are requested. If the configurator creates the task, the stack area can only be allocated by the kernel.

(b) Management of stack area allocated from system pool

If the kernel is specified to allocate the stack area when creating a task associated with a user domain, the stack area is allocated from the system pool. However, the resource pool is used for the size of VTSZ_SPLALCMB at maximum for managing the stack area. If the configurator creates the task, only the kernel can be specified to allocate the stack area.

The contents of this macro are shown below.

- When CFG_PROTMEM is selected
VTSZ_SPLALCMB=60
- When CFG_PROTMEM is not selected
VTSZ_SPLALCMB=36

(2) Data queue

If the data count is not 0 when creating a data queue, the resource pool is requested for the size of TSZ_DTQMB (data count).

The contents of this macro are shown below.

```
TSZ_DTQMB(data count) = ((data count) × 4) + 4
```

The area used is released when the data queue is deleted.

Note that the macro TSZ_DTQ (data count) has the same definition as the above macro. TSZ_DTQMB is a macro defined by the μ TRON 4.0 protection function extension, while TSZ_DTQ is a macro defined by the original μ TRON 4.0 specification.

(3) Mailbox

If the condition of "TA_MPRI attribute with the Max. message priority > 1" is satisfied, the resource pool is requested for the size of TSZ_MBXMB (number of messages, Max. message priority).

The contents of this macro are shown below.

```
TSZ_MBXMB(number of messages, Max. message priority)=((Max. message priority)×8)+4
```

The area used is released when the mailbox is deleted.

(4) Message buffer

If the buffer size is not 0 when creating a message buffer, the resource pool is requested for the size of (buffer size + 4).

The area used is released when the message buffer is deleted.

(5) Fixed-size memory pool

(a) Management of fixed-size memory blocks

To manage the fixed-size memory blocks, the resource pool is requested for the size of TSZ_MPFMB (number of blocks, block size). The contents of this macro are shown below.

```
TSZ_MPFMB(number of blocks, block size)=((number of blocks)×4)+4
```

(b) When allocating fixed-size memory pool from system pool

If the kernel is specified to allocate the pool area when creating a fixed-size memory pool, the pool area is allocated from the system pool. However, the resource pool is used for the size of VTSZ_SPLALCMB at maximum for managing the pool area. If the configurator creates the fixed-size memory pool, only the kernel can be specified to allocate the pool area. The contents of this macro are shown below.

- When CFG_PROTMEM is selected
VTSZ_SPLALCMB=60
- When CFG_PROTMEM is not selected
VTSZ_SPLALCMB=36

(6) Variable-size memory pool

(a) When allocating variable-size memory pool from system pool

If the kernel is specified to allocate the pool area when creating a variable-size memory pool, the pool area is allocated from the system pool. However, the resource pool is used for the size of VTSZ_SPLALCMB at maximum for managing the pool area. If the configurator creates the variable-size memory pool, only the kernel can be specified to allocate the pool area. The contents of this macro are shown below.

- When CFG_PROTMEM is selected
$$\text{VTSZ_SPLALCMB} = 60$$
- When CFG_PROTMEM is not selected
$$\text{VTSZ_SPLALCMB} = 36$$

(b) Sector management

If the VTA_UNFRAGMENT attribute is specified, the resource pool is requested for the size of VTSZ_SCTMB (Max. sector count). The contents of this macro are shown below.

$$\text{VTSZ_SCTMB}(\text{Max. sector count}) = (20 \times (\text{Max. sector count}) + 72) + 4$$

(7) Protected mailbox

If the condition of "TA_MPRI attribute with the Max. message priority > 1" is satisfied when creating a protected mailbox, the resource pool is requested for the size of TSZ_MBPMB (number of messages, Max. message priority).

The contents of this macro are shown below.

$$\text{TSZ_MBPMB}(\text{number of messages, Max. message priority}) = ((\text{Max. message priority}) \times 8) + 4$$

The area used is released when the protected mailbox is deleted.

13.2.3 Sizes Used and Released at Other Timings

(1) Mailbox: snd_mbx, isnd_mbx

If a message is queued in a mailbox when there is no task waiting to receive a message, the resource pool is requested for the size of VTSZ_MSGMB. The contents of this macro are shown below.

$$\text{VTSZ_MSGMB} = 20$$

This area is released when the message is received or the relevant mailbox is deleted.

(2) Variable-size memory pool: `get_mpl`, `pget_mpl`, `ipget_mpl`, `tget_mpl`

When acquiring a memory block, the resource pool is requested for the size of `VTSZ_BLKMB` at maximum. The contents of this macro are shown below.

```
VTSZ_BLKMB=36
```

This area is released when the block is released or the relevant variable-size memory pool is deleted.

(3) Protected memory pool: `pget_mpp`

When acquiring a protected memory block, the resource pool is requested for the size of `VTSZ_MPPBLKMB` at maximum. The contents of this macro are shown below.

```
VTSZ_MPPBLKMB=60
```

This area is released when the block is released.

(4) Protected mailbox: `snd_mbp`

If a message is queued in a protected mailbox when there is no task waiting to receive a message, the resource pool is used for the size of `VTSZ_MSGMB`. The contents of this macro are shown below.

```
VTSZ_MSGMB=20
```

This area is released when the message is received or the relevant protected mailbox is deleted.

13.3 Calculation

The resource pool is managed by sectors with the "Min. block size" as 20 bytes.

(1) Request of 160 bytes or less (allocated as a sector)

The requested size is calculated by the following equation.

$$SZ_RESSCT = \Sigma ((ROUND_UP (Num[n]/Cnt[n], 1) \times 696)$$

- n: 1, 2, 4, or 8
- Num[n]: Number of simultaneous requests for a size of (20 × n) bytes or less (see the table below)
- Cnt[n]: Number of blocks in the sector (see the table below)

n	Num[n]	Cnt[n]
1	20 bytes or less	32
2	21 to 40 bytes	16
4	41 to 80 bytes	8
8	81 to 160 bytes	4

(2) Request exceeding 160 bytes

Simultaneous requests are calculated with the following equation.

$$SZ_RESLARGE = \Sigma (\text{Requested size} + 32)$$

(3) Total required size

VTSZ_RPLMB bytes are used for management of the resource pool. The contents of this macro are shown below.

VTSZ_RPLMB=32

This area is never released.

The total required size is calculated with the following equation.

$$\text{Total required size} = \text{VTSZ_RPLMB} + \text{SZ_RESSCT} + \text{SZ_RESLARGE}$$

Section 14 Estimation of System Pool Size

14.1 Overview

The system pool size is specified by CFG_SYSPoolsSZ.

The system pool size is estimated using the following procedure.

1. Determine the timing at which the system pool is used and its requested size → Section 14.2, Requested Timing and Size
2. Estimate the required size taking into consideration the algorithm described below.

Reference: Section 4.31, Controlling Memory Fragmentation

14.2 Requested Timing and Size

The system pool is used in the following cases. This means that the system pool size can be set to 0 as long as the areas below are allocated on the application side.

(1) When task is created

If the kernel is specified to allocate the stack area when creating a task associated with a user domain, the stack area of the specified stack size is allocated from the system pool. If the configurator creates the task, only the kernel can be specified to allocate the stack area.

This area is released when the task is deleted.

(2) When fixed-size memory pool is created

If the kernel is specified to allocate the pool area when creating a fixed-size memory pool, a fixed-size memory pool area with the size of TSZ_MPF (number of blocks, block size) bytes is allocated from the system pool.

This area is released when the fixed-size memory pool is deleted.

The contents of the TSZ_MPF macro are shown below.

```
TSZ_MPF(number of blocks, block size)=(block size) × (number of blocks)
```

(3) When variable-size memory pool is created

If the kernel is specified to allocate the pool area when creating a variable-size memory pool, a memory pool area with the specified memory pool size is allocated from the system pool.

This area is released when the variable-size memory pool is deleted.

Section 15 Notes on FPU

15.1 Meaning of "Using FPU"

From the kernel viewpoint, "using FPU" stands for accessing the FPU registers from the application.

The FPU registers are accessed if "cpu = sh4a" is specified as a compiler option and also either of the following is satisfied.

- (1) The fpu option of the compiler is not specified. In the HEW, [Mix] is selected for [Floating-point operation mode] in the [CPU] tab of the [SuperH RISC engine Standard Toolchain] dialog box that is opened by selecting [Option] -> [SuperH RISC engine Standard Toolchain...].
- (2) The floating-point data type is used.

Particularly in case (1), the FPU is assumed to be used even when no floating-point data is handled. For this reason, specifying (1) is not recommended strongly.

15.2 FPU Usage in Each Application

15.2.1 Task, Task Exception Processing Routine, Extended Service Call Routine, or Trap Routine

- (1) TA_COP1 or TA_COP2 attribute

In a task, task exception processing routine, extended service call routine, or trap routine, the FPU can be used only when TA_COP1 or TA_COP1 and TA_COP2 are specified as the attribute. Specify the TA_COP1 or TA_COP2 attribute as shown in table 15.1.

Table 15.1 Specification of TA_COP1 or TA_COP2 Attribute

Condition	Attribute
When matrix calculation is performed (both FPU register banks are used)	TA_COP1 TA_COP2
When normal floating-point operation is performed (only FPU register bank 0 is used)	TA_COP1
When no floating-point operation is performed	(Not required)

(2) Initial FPSCR value

The initial FPSCR value is specified at creation or definition of a task, task exception processing routine, extended service call routine, or trap routine. Make sure that this specification does not conflict with the compiler option settings. The relationship is shown in table 15.2.

Table 15.2 Relationship between Initial FPSCR Value Specified at Creation/Definition and Compiler Option Settings

Compiler Options			Initial FPSCR Value (inifpscr) That Should be Specified at Creation/Definition
fpu	denormalize	round	
No specification or single*	on*	zero*	H'00040001 (FR = 0, PR = 0, DN = 1, RM = 1)
		nearest	H'00040000 (FR = 0, PR = 0, DN = 1, RM = 0)
	off	zero	H'00000001 (FR = 0, PR = 0, DN = 0, RM = 1)
		nearest	H'00000000 (FR = 0, PR = 0, DN = 0, RM = 0)
double	on	zero	H'000C0001 (FR = 0, PR = 1, DN = 1, RM = 1)
		nearest	H'000C0000 (FR = 0, PR = 1, DN = 1, RM = 0)
	off	zero	H'00080001 (FR = 0, PR = 1, DN = 0, RM = 1)
		nearest	H'00080000 (FR = 0, PR = 1, DN = 0, RM = 0)

Note: Default setting of the compiler

15.2.2 Other Applications

Other applications cannot use the FPU.

To use the FPU, the FPU registers must be allocated on the application side.

Section 16 System Down Handling

16.1 Information during System Down

The system down routine is called when the system goes down. Information listed in table 16.1 is passed to the system down routine.

Table 16.1 Information Passed to System Down Routine

		Parameters Passed to System Down Routine				
Cause of System		Error Type	Information 1	Information 2	Information 3	
No.	Down	ER type	VW inf1	VW inf2	VW inf3	Countermeasure
		(R4)	(R5)	(R6)	(R7)	
1	vsys_dwn, ivsys_dwn 1 to H'7ffffff		Parameter of vsys_dwn or ivsys_dwn			
2	vsta_knl System pool cannot be created	0	1	Undefined	Undefined	Refer to section 16.2.
3	vsta_knl		2	E_PAR	Undefined	
4	Static memory object cannot be created			E_NOMEM	Undefined	
5	ext_tsk Called by non-task context	−1	E_CTX	Address calling ext_tsk	Undefined	
6	exd_tsk Called by non-task context	−2	E_CTX	Address calling exd_tsk	Undefined	
7	Undefined CPU exception occurred	−H'10	EXPEVT code	VT_EXC *pk_exc	Undefined	
8	Undefined interrupt occurred	−H'11	INTEVT code	Undefined	Undefined	
9	vsta_knl Initialization related to CFG_ACTION failed	−H'20	Undefined	Undefined	Undefined	
10	Memory access violation occurred* (detected by MMU)	−H'80	VT_MAV *pk_mav	VT_EXC *pk_exc	Undefined	Remove the cause of memory access violation based on pk_mav and pk_exc.

Note: Realized by the sample memory access violation handler (samples\sysapp\mavhdr.c).

16.2 Error at Kernel Start (vsta_knl)

16.2.1 System Down Occurrence

(1) System pool cannot be created

When the memory object protection function is installed, the system pool section must be located in an MMU mapped area whose start address is at the CFG_PAGESZ (4 Kbytes) boundary. Otherwise it must be located in an area that is accessible in the user mode, and whose start address is a multiple of four. If this is not satisfied, the system goes down.

(2) Static memory object cannot be created

System down occurs on detection of one of the following errors regarding the static memory object.

- Start address of the static memory object is not the boundary address of the page size of the static memory object (No. 3 in table 16.1)
- Resource pool is insufficient (No. 4 in table 16.1)

16.2.2 When Object Specified in Configurator Cannot be Created

Objects created or defined in the configurator are actually created or defined by issuing service calls from the "initialization routine" output by the configurator.

There are two types of initialization routines, as shown below.

(1) kernel_def_inireg.h

Object created or defined with [Kernel Side] selected

(2) kernel_cfg_inireg.h

Object created or defined without [Kernel Side] selected

In these files, the following kind of statement is used to create an object.

```
if (_CRE_MPL(ID_DOM2_MPL) != E_OK)
    while(1);
```

`_CRE_MPL()` is a macro that issues a service call to create a variable-size memory pool.

Table 16.2 shows the service calls and macros used.

Table 16.2 Service Calls Used by Initialization Routines

Object Type	Service Call Used	Macro
Interrupt handler	idef_inh	_DEF_INH
CPU exception handler	idef_inh	_DEF_INH
Overrun handler	def_ovr	_DEF_OVR
Cyclic handler	icre_cyc	_CRE_CYC
Alarm handler	icre_alm	_CRE_ALM
Extended service call routine	idef_svc	_DEF_SVC
Trap routine	ivdef_trp	_DEF_TRP
Semaphore	icre_sem	_CRE_SEM
Event flag	icre_flg	_CRE_FLG
Data queue	icre_dtq	_CRE_DTQ
Mailbox	icre_mbx	_CRE_MBX
Mutex	cre_mtx	_CRE_MTX
Message buffer	icre_mbf	_CRE_MBF
Fixed-size memory pool	When memory object protection function is used: icra_mpf When memory object protection function is not used: icre_mpf	_CRE_MPF
Variable-size memory pool	When memory object protection function is used: ivcra_mpl When memory object protection function is not used: icre_mpl	_CRE_MPL
Protected memory pool	icre_mpp	_CRE_MPP
Protected mailbox	icre_mbp	_CRE_MBP
Task	icre_tsk	_CRE_TSK
Task exception processing routine	idef_tex	_DEF_TEX

When these service calls result in an error, an infinite loop is entered immediately after the issuance of that service call. Determine the corresponding object created by the configurator from the location of the infinite loop, and confirm the configurator settings for that object.

Reference: Section 6.22.12, Start Kernel (vsta_knl, ivsta_knl)

Section 17 Reference Listing

17.1 Service Call Reference

(1) Task Management

1	ER cre_tsk(ID, T_CTSK *);	Create task
	ER icre_tsk(ID, T_CTSK *);	Create task (non-task context)
2	ER_ID acre_tsk(T_CTSK *);	Create task and assign task ID automatically
	ER_ID iacre_tsk(T_CTSK *);	Create task and assign task ID automatically (non-task context)
3	ER del_tsk(ID);	Delete task
4	ER act_tsk(ID);	Initiate task
	ER iact_tsk(ID);	Initiate task (non-task context)
5	ER_UINT can_act(ID);	Cancel task initiation request
	ER_UINT ican_act(ID);	Cancel task initiation request (non-task context)
6	ER sta_tsk(ID, VP_INT);	Initiate task and specify start code
	ER ista_tsk(ID, VP_INT);	Initiate task and specify start code (non-task context)
7	void ext_tsk(void);	Exit current task
8	void exd_tsk(void);	Exit and delete current task
9	ER ter_tsk(ID);	Forcibly terminate task
10	ER chg_pri(ID, PRI);	Change task priority
	ER ichg_pri(ID, PRI);	Change task priority (non-task context)
11	ER get_pri(ID, PRI *);	Refer to task priority
	ER iget_pri(ID, PRI *);	Refer to task priority (non-task context)
12	ER ref_tsk(ID, T_RTsk *);	Refer to task state
	ER iref_tsk(ID, T_RTsk *);	Refer to task state (non-task context)
13	ER ref_tst(ID, T_RTST *);	Refer to task state (simple version)
	ER iref_tst(ID, T_RTST *);	Refer to task state (simple version, non-task context)
14	ER vchg_tmd(MODE);	Change task execution mode

(2) Task Synchronization

15	ER slp_tsk(void);	Shift current task to the WAITING state
16	ER tslp_tsk(TMO);	Shift current task to the WAITING state with timeout function

17	ER wup_tsk(ID);	Wake up task
	ER iwup_tsk(ID);	Wake up task (non-task context)
18	ER_UINT can_wup(ID);	Cancel wakeup request
	ER_UINT ican_wup(ID);	Cancel wakeup request (non-task context)
19	ER rel_wai(ID);	Cancel the WAITING state forcibly
	ER irel_wai(ID);	Cancel the WAITING state forcibly (non-task context)
20	ER sus_tsk(ID);	Shift to the SUSPENDED state
	ER isus_tsk(ID);	Shift to the SUSPENDED state (non-task context)
21	ER rsm_tsk(ID);	Resume the execution of a task in the SUSPENDED state
	ER irsm_tsk(ID);	Resume the execution of a task in the SUSPENDED state (non-task context)
22	ER frsm_tsk(ID);	Forcibly resume the execution of a task in the SUSPENDED state
	ER ifrsm_tsk(ID);	Forcibly resume the execution of a task in the SUSPENDED state (non-task context)
23	ER dly_tsk(RELTIM);	Delay the current task
24	ER vset_tfl(ID, FLGPTN);	Set the task event flag
	ER ivset_tfl(ID, FLGPTN);	Set the task event flag (non-task context)
25	ER vclr_tfl(ID, FLGPTN);	Clear the task event flag
	ER ivclr_tfl(ID, FLGPTN);	Clear the task event flag (non-task context)
26	ER vwai_tfl(FLGPTN, FLGPTN *);	Wait for the task event flag
27	ER vpol_tfl(FLGPTN, FLGPTN *);	Poll and wait for the task event flag
28	ER vtwai_tfl(FLGPTN, FLGPTN *, TMO);	Wait for the task event flag with timeout function

(3) Task Exception Processing Functions

29	ER def_tex(ID, T_DTEX *);	Define the task exception processing routine
	ER ideo_def_tex(ID, T_DTEX *);	Define the task exception processing routine (non-task context)
30	ER ras_tex(ID, TEXPTN);	Request the task exception processing
	ER iras_tex(ID, TEXPTN);	Request the task exception processing (non-task context)
31	ER dis_tex(void);	Disable the task exception processing
32	ER ena_tex(void);	Enable the task exception processing
33	BOOL sns_tex(void);	Refer to the task exception processing disabled state

34	ER ref_tex(ID, T_RTEX *);	Refer to the task exception processing state
	ER iref_tex(ID, T_RTEX *);	Refer to the task exception processing state (non-task context)
(4) Synchronization and Communication (Semaphore)		
35	ER cre_sem(ID, T_CSEM *);	Create semaphore
	ER icre_sem(ID, T_CSEM *);	Create semaphore (non-task context)
36	ER_ID acre_sem(T_CSEM *);	Create semaphore and assign semaphore ID automatically
	ER_ID iacre_sem(T_CSEM *);	Create semaphore and assign semaphore ID automatically (non-task context)
37	ER del_sem(ID);	Delete semaphore
38	ER sig_sem(ID);	Return semaphore resource
	ER isig_sem(ID);	Return semaphore resource (non-task context)
39	ER wai_sem(ID);	Wait for semaphore resource
40	ER pol_sem(ID);	Polls and waits for semaphore resource
	ER ipol_sem(ID);	Polls and waits for semaphore resource (non-task context)
41	ER twai_sem(ID, TMO);	Wait for semaphore resource with timeout function
42	ER ref_sem(ID, T_RSEM *);	Refer to semaphore state
	ER iref_sem(ID, T_RSEM *);	Refer to semaphore state (non-task context)
(5) Synchronization and Communication (Event Flag)		
43	ER cre_flg(ID, T_CFLG *);	Create event flag
	ER icre_flg(ID, T_CFLG *);	Create event flag (non-task context)
44	ER_ID acre_flg(T_CFLG *);	Create event flag and assign event flag ID automatically
	ER_ID iacre_flg(T_CFLG *);	Create event flag and assign event flag ID automatically (non-task context)
45	ER del_flg(ID);	Delete event flag
46	ER set_flg(ID, FLGPTN);	Set event flag
	ER iset_flg(ID, FLGPTN);	Set event flag (non-task context)
47	ER clr_flg(ID, FLGPTN);	Clear event flag
	ER iclr_flg(ID, FLGPTN);	Clear event flag (non-task context)
48	ER wai_flg(ID, FLGPTN, MODE, FLGPTN *);	Wait for event flag
49	ER pol_flg(ID, FLGPTN, MODE, FLGPTN *);	Poll and wait for event flag

ER ipol_flg(ID, FLGPTN, MODE, FLGPTN *);	Poll and wait for event flag (non-task context)
50 ER twai_flg(ID, FLGPTN, MODE, FLGPTN *, TMO);	Wait for event flag with timeout function
51 ER ref_flg(ID, T_RFLG *);	Refer to event flag state
ER iref_flg(ID, T_RFLG *);	Refer to event flag state (non-task context)

(6) Synchronization and Communication (Data Queue)

52 ER cre_dtq(ID, T_CDTQ *);	Create data queue
ER icre_dtq(ID, T_CDTQ *);	Create data queue (non-task context)
53 ER_ID acre_dtq(T_CDTQ *);	Create data queue and assign data queue ID automatically
ER_ID iacre_dtq(T_CDTQ *);	Create data queue and assign data queue ID automatically (non-task context)
54 ER del_dtq(ID);	Delete data queue
55 ER snd_dtq(ID, VP_INT);	Send data to data queue
56 ER psnd_dtq(ID, VP_INT);	Poll and send data to data queue
ER ipsnd_dtq(ID, VP_INT);	Poll and send data to data queue (non-task context)
57 ER tsnd_dtq(ID, VP_INT, TMO);	Send data to data queue with timeout function
58 ER fsnd_dtq(ID, VP_INT);	Forcibly send data to data queue
ER ifsnd_dtq(ID, VP_INT);	Forcibly send data to data queue (non-task context)
59 ER rcv_dtq(ID, VP_INT *);	Receive data from data queue
60 ER prcv_dtq(ID, VP_INT *);	Poll and receive data from data queue
61 ER trcv_dtq(ID, VP_INT *, TMO);	Receive data from data queue with timeout function
62 ER ref_dtq(ID, T_RDTQ *);	Refer to data queue state
ER iref_dtq(ID, T_RDTQ *);	Refer to data queue state (non-task context)

(7) Synchronization and Communication (Mailbox)

63 ER cre_mbx(ID , T_CMBX *);	Create mailbox
ER icre_mbx(ID , T_CMBX *);	Create mailbox (non-task context)
64 ER_ID acre_mbx(T_CMBX *);	Create mailbox and assign mailbox ID automatically
ER_ID iacre_mbx(T_CMBX *);	Create mailbox and assign mailbox ID automatically (non-task context)
65 ER del_mbx(ID);	Delete mailbox
66 ER snd_mbx(ID, T_MSG *);	Send message to mailbox

	ER isnd_mbx(ID, T_MSG *);	Send message to mailbox (non-task context)
67	ER rcv_mbx(ID, T_MSG **);	Receive message from mailbox
68	ER prcv_mbx(ID, T_MSG **);	Poll and receive message from mailbox
	ER iprcv_mbx(ID, T_MSG **);	Poll and receive message from mailbox (non-task context)
69	ER trcv_mbx(ID, T_MSG **, TMO);	Receive message from mailbox with timeout function
70	ER ref_mbx(ID, T_RMBX *);	Refer to mailbox state
	ER iref_mbx(ID, T_RMBX *);	Refer to mailbox state (non-task context)

(8) Synchronization and Communication (Mutex)

71	ER cre_mtx(ID, T_CMTX *);	Create mutex
72	ER_ID acre_mtx(T_CMTX *);	Create mutex and assign mutex ID automatically
73	ER del_mtx(ID);	Delete mutex
74	ER loc_mtx(ID);	Lock mutex
75	ER ploc_mtx(ID);	Poll and lock mutex
76	ER tlloc_mtx(ID, TMO);	Lock mutex with timeout function
77	ER unl_mtx(ID);	Unlock mutex
78	ER ref_mtx(ID, T_RMTX *);	Refer to mutex state

(9) Extended Synchronization and Communication (Message Buffer)

79	ER cre_mbf(ID, T_CMBF *);	Create message buffer
	ER icre_mbf(ID, T_CMBF *);	Create message buffer (non-task context)
80	ER_ID acre_mbf(T_CMBF *);	Create message buffer and assign message buffer ID automatically
	ER_ID iacre_mbf(T_CMBF *);	Create message buffer and assign message buffer ID automatically (non-task context)
81	ER del_mbf(ID);	Delete message buffer
82	ER snd_mbf(ID, VP, UINT);	Send message to message buffer
83	ER psnd_mbf(ID, VP, UINT);	Poll and send message to message buffer
	ER ipsnd_mbf(ID, VP, UINT);	Poll and send message to message buffer (non-task context)
84	ER tsnd_mbf(ID, VP, UINT, TMO);	Send message to message buffer with timeout function
85	ER_UINT rcv_mbf(ID, VP);	Receive message from message buffer
86	ER_UINT prcv_mbf(ID, VP);	Poll and receive message from message buffer

87	ER_UINT trcv_mbf(ID, VP, TMO);	Receive message from message buffer with timeout function
88	ER ref_mbf(ID, T_RMBF *);	Refer to message buffer state
	ER iref_mbf(ID, T_RMBF *);	Refer to message buffer state (non-task context)

(10) Fixed-Size Memory Pool Management

89	ER cre_mpf(ID, T_CMPF *);	Create fixed-size memory pool
	ER icre_mpf(ID, T_CMPF *);	Create fixed-size memory pool (non-task context)
90	ER_ID acre_mpf(T_CMPF *);	Create fixed-size memory pool and assign fixed-size memory pool ID automatically
	ER_ID iacre_mpf(T_CMPF *);	Create fixed-size memory pool and assign fixed-size memory pool ID automatically (non-task context)
91	ER del_mpf(ID);	Delete fixed-size memory pool
92	ER get_mpf(ID, VP *);	Get fixed-size memory block
93	ER pget_mpf(ID, VP *);	Poll and get fixed-size memory block
	ER ipget_mpf(ID, VP *);	Poll and get fixed-size memory block (non-task context)
94	ER tget_mpf(ID, VP *, TMO);	Get fixed-size memory block with timeout function
95	ER rel_mpf(ID, VP);	Release fixed-size memory block
	ER irel_mpf(ID, VP);	Release fixed-size memory block (non-task context)
96	ER ref_mpf(ID, T_RMPF *);	Refer to fixed-size memory pool state
	ER iref_mpf(ID, T_RMPF *);	Refer to fixed-size memory pool state (non-task context)

(11) Variable-Size Memory Pool Management

97	ER cre_mpl(ID, T_CMPL *);	Create variable-size memory pool
	ER icre_mpl(ID, T_CMPL *);	Create variable-size memory pool (non-task context)
98	ER_ID acre_mpl(T_CMPL *);	Create variable-size memory pool and assign variable-size memory pool ID automatically
	ER_ID iacre_mpl(T_CMPL *);	Create variable-size memory pool and assign variable-size memory pool ID automatically (non-task context)
99	ER del_mpl(ID);	Delete variable-size memory pool
100	ER get_mpl(ID, UINT, VP *);	Get variable-size memory block

101	ER pget_mpl(ID, UINT, VP *); ER ipget_mpl(ID, UINT, VP *);	Poll and get variable-size memory block Poll and get variable-size memory block (non-task context)
102	ER tget_mpl(ID, UINT, VP *, TMO);	Get variable-size memory block with timeout function
103	ER rel_mpl(ID, VP); ER irel_mpl(ID, VP);	Release variable-size memory block Release variable-size memory block (non-task context)
104	ER ref_mpl(ID, T_RMPL *); ER iref_mpl(ID, T_RMPL *);	Refer to variable-size memory pool state Refer to variable-size memory pool state (non-task context)

(12) Time Management (System Clock)

105	ER set_tim(SYSTIM *); ER iset_tim(SYSTIM *);	Set system clock Set system clock (non-task context)
106	ER get_tim(SYSTIM *); ER iget_tim(SYSTIM *);	Get system clock Get system clock (non-task context)

(13) Time Management (Cyclic Handler)

107	ER cre_cyc(ID, T_CCYC *); ER icre_cyc(ID, T_CCYC *);	Create cyclic handler Create cyclic handler (non-task context)
108	ER_ID acre_cyc(T_CCYC *); ER_ID iacre_cyc(T_CCYC *);	Create cyclic handler and assign cyclic handler ID automatically Create cyclic handler and assign cyclic handler ID automatically (non-task context)
109	ER del_cyc(ID);	Delete cyclic handler
110	ER sta_cyc(ID); ER ista_cyc(ID);	Start cyclic handler operation Start cyclic handler operation (non-task context)
111	ER stp_cyc(ID); ER istp_cyc(ID);	Stop cyclic handler operation Stop cyclic handler operation (non-task context)
112	ER ref_cyc(ID, T_RCYC *); ER iref_cyc(ID, T_RCYC *);	Refer to the cyclic handler state Refer to the cyclic handler state (non-task context)

(14) Time Management (Alarm Handler)

113	ER cre_alm(ID, T_CALM *); ER icre_alm(ID, T_CALM *);	Create alarm handler Create alarm handler (non-task context)
114	ER_ID acre_alm(T_CALM *);	Create alarm handler and assign alarm

ER_ID iacre_alm(T_CALM *);	handler ID automatically
115 ER del_alm(ID);	Create alarm handler and assign alarm handler ID automatically (non-task context)
116 ER sta_alm(ID, RELTIM);	Delete alarm handler
ER ista_alm(ID, RELTIM);	Start alarm handler operation
117 ER stp_alm(ID);	Start alarm handler operation (non-task context)
ER istp_alm(ID);	Stop alarm handler operation
118 ER ref_alm(ID, T_RALM *);	Stop alarm handler operation (non-task context)
ER iref_alm(ID, T_RALM *);	Refer to the alarm handler state
	Refer to the alarm handler state (non-task context)
(15) Time Management (Overrun Handler)	
119 ER def_ovr(T_DOVR *);	Define overrun handler
120 ER sta_ovr(ID, OVRTIM);	Start overrun handler operation
ER ista_ovr(ID, OVRTIM);	Start overrun handler operation (non-task context)
121 ER stp_ovr(ID);	Stop overrun handler operation
ER istp_ovr(ID);	Stop overrun handler operation (non-task context)
122 ER ref_ovr(ID, T_ROVR *);	Refer to overrun handler state
ER iref_ovr(ID, T_ROVR *);	Refer to overrun handler state (non-task context)
(16) System State Management	
123 ER rot_rdq(PRI);	Rotate ready queue
ER irot_rdq(PRI);	Rotate ready queue (non-task context)
124 ER get_tid(ID *);	Get task ID in RUNNING state
ER iget_tid(ID *);	Get task ID in RUNNING state (non-task context)
125 ER get_did(ID *);	Get domain ID of the task in RUNNING state
ER iget_did(ID *);	Get domain ID of the task in RUNNING state (non-task context)
126 ER loc_cpu(void);	Lock CPU
ER iloc_cpu(void);	Lock CPU (non-task context)
127 ER unl_cpu(void);	Unlock CPU
ER iunl_cpu(void);	Unlock CPU (non-task context)

128	ER dis_dsp(void);	Disable task dispatch
129	ER ena_dsp(void);	Enable task dispatch
130	BOOL sns_ctx(void);	Refer to task context
131	BOOL sns_loc(void);	Refer to CPU-locked state
132	BOOL sns_dsp(void);	Refer to dispatch-disabled state
133	BOOL sns_dpn(void);	Refer to dispatch-pended state
134	void vsta_knl(void);	Start kernel
	void ivsta_knl(void);	Start kernel (non-task context)
135	void vsys_dwn(ER, VW, VW, VW);	Terminate the system
	void ivsys_dwn(ER, VW, VW, VW);	Terminate the system (non-task context)
136	ER vget_trc(VW, VW, VW, VW);	Acquire trace information
	ER ivget_trc(VW, VW, VW, VW);	Acquire trace information (non-task context)
137	ER_UINT vchg_cop(ATR);	Change DSP (TA_COP0) attribute

(17) Interrupt Management

138	ER def_inh(INHNO, T_DINH *);	Define interrupt handler
	ER ndef_inh(INHNO, T_DINH *);	Define interrupt handler (non-task context)
139	ER chg_ims(IMASK);	Change interrupt mask
	ER ichg_ims(IMASK);	Change interrupt mask (non-task context)
140	ER get_ims(IMASK *);	Refer to interrupt mask
	ER iget_ims(IMASK *);	Refer to interrupt mask (non-task context)

(18) Extended Service Call and Trap Management

141	ER def_svc(FN, T_DSVC *);	Define extended service call
	ER ndef_svc(FN, T_DSVC *);	Define extended service call (non-task context)
142	ER_UINT cal_svc(FN, VP_INT, VP_INT, VP_INT, VP_INT);	Issue extended service call
	ER_UINT ical_svc(FN, VP_INT, VP_INT, VP_INT, VP_INT);	Issue extended service call (non-task context)
143	ER vdef_trp(UINT, VT_DTRP *);	Define trap routine
	ER ndef_trp(UINT, VT_DTRP *);	Define trap routine (non-task context)

(19) System Configuration Management

144	ER def_exc(EXCNO, T_DEXC *);	Define CPU exception handler
	ER ndef_exc(EXCNO, T_DEXC *);	Define CPU exception handler (non-task context)
145	ER ref_cfg(T_RCFG *);	Refer to configuration information

ER iref_cfg(T_RCFG *);	Refer to configuration information (non-task context)
146 ER ref_ver(T_RVER *);	Refer to version information
ER iref_ver(T_RVER *);	Refer to version information (non-task context)
(20) Memory Object Management Function	
147 ER sac_mem(VP, ACVCT *);	Change access permission vector for memory object
148 ER prb_mem(VP, SIZE, ID, MODE);	Check access right for memory area
149 ER ref_mem(VP, T_RMEM *);	Refer to memory object state
150 ER vloc_tlb(VP);	Lock TLB entry
151 ER vunl_tlb(VP);	Unlock TLB entry
(21) Protected Memory Pool Management	
152 ER_UINT pget_mpp(ID, UINT, VP *);	Poll and get protected memory block
153 ER rel_mpp(ID, VP);	Release protected memory block
154 ER ref_mpp(ID, T_RMPP *);	Refer to protected memory pool state
(22) Protected Mailbox Management	
155 ER cre_mbp(ID, T_CMBP *);	Create protected mailbox
ER icre_mbp(ID, T_CMBP *);	Create protected mailbox (non-task context)
156 ER_ID acre_mbp(T_CMBP *);	Create protected mailbox and assign mailbox ID automatically
ER_ID iacre_mbp(T_CMBP *);	Create protected mailbox and assign mailbox ID automatically (non-task context)
157 ER del_mbp(ID);	Delete protected mailbox
158 ER snd_mbp(ID, VP, PRI);	Send message to protected mailbox
159 ER_UINT rcv_mbp(ID, VP *);	Receive message from protected mailbox
160 ER_UINT prcv_mbp(ID, VP *);	Poll and receive message from protected mailbox
161 ER_UINT trcv_mbp(ID, VP *, TMO);	Receive message from protected mailbox with timeout function
162 ER ref_mbp(ID, T_RMBP *);	Refer to protected mailbox state
ER iref_mbp(ID, T_RMBP *);	Refer to protected mailbox state (non-task context)
(23) System Memory Management	
163 ER vref_syp(VT_RSYP *);	Refer to system pool state

164 ER vref_rsp(VT_RRSP *);

Refer to resource pool state

(24) Performance Management

165 ER_UINT vchg_ppc(ID, MODE);

Start, stop, or initialize performance measurement

ER_UINT ivchg_ppc(ID, MODE);

Start, stop, or initialize performance measurement (non-task context)

166 ER_UINT vref_ppc(ID, VT_RPPC *);

Refer to performance measurement result

ER_UINT ivref_ppc(ID, VT_RPPC *);

Refer to performance measurement result (non-task context)

17.2 Service Call Error Code List

Table 17.1 List of Service Call Error Codes

Error Code (Mnemonic)	Error Code		Description
E_OK	H'00000000	(D'0)	Normal termination
E_NOSPT	H'ffffff7	(-D'9)	Unsupported function
E_RSFN	H'ffffff6	(-D'10)	Reserved function code
E_RSATR	H'ffffff5	(-D'11)	Reserved attribute (invalid attribute)
E_PAR	H'fffffef	(-D'17)	Parameter error
E_ID	H'fffffee	(-D'18)	Invalid ID number
E_CTX	H'fffffe7	(-D'25)	Context error
E_MACV	H'fffffe6	(-D'26)	Memory access violation
E_ILUSE	H'fffffe4	(-D'28)	Illegal use of service call
E_NOMEM	H'fffffdf	(-D'33)	Insufficient memory
E_NOID	H'fffffde	(-D'34)	No ID available
E_OBJ	H'fffffd7	(-D'41)	Invalid object
E_NOEXS	H'fffffd6	(-D'42)	Undefined object
E_QOVR	H'fffffd5	(-D'43)	Queuing or nesting overflow
E_RLWAI	H'fffffcf	(-D'49)	WAITING state was forcibly cancelled or an attempt was made to shift to the WAITING state in WAITING-disabled state.
E_TMOUT	H'fffffce	(-D'50)	Polling failed or timeout
E_DLT	H'fffffcd	(-D'51)	Waiting object deleted

**Renesas Microcomputer Development Environment System
User's Manual
HI7300/PX V.1.01**

Publication Date: Rev.2.00, July 25, 2006
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Customer Support Department
Global Strategic Communication Div.
Renesas Solutions Corp.

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



RENESAS SALES OFFICES

<http://www.renesas.com>

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

Renesas Technology America, Inc.

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

Renesas Technology Europe Limited

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology (Shanghai) Co., Ltd.

Unit 204, 205, AZIACenter, No.1233 Lujiazui Ring Rd, Pudong District, Shanghai, China 200120
Tel: <86> (21) 5877-1818, Fax: <86> (21) 6887-7898

Renesas Technology Hong Kong Ltd.

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

Renesas Technology Taiwan Co., Ltd.

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

Renesas Technology Singapore Pte. Ltd.

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

Renesas Technology Korea Co., Ltd.

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

Renesas Technology Malaysia Sdn. Bhd

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

HI7300/PX V.1.01 User's Manual



Renesas Electronics Corporation

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

REJ10J1198-0200