お客様各位

カタログ等資料中の旧社名の扱いについて

2010 年 4 月 1 日を以って NEC エレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010 年 4 月 1 日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

RENESAS

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

RENESAS

HI7000/4 シリーズ
(HI7000/4 V.2.02, HI7700/4 V.2.02, HI7750/4 V.2.02)

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

ルネサス エレクトロニクス
www.renesas.com

Rev.7.00　2006.01

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ (http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

【商標等について】

- TRON は、"The Real-time Operating system Nucleus" の略称です。ITRON は、"Industrial TRON"の略称です。μITRON は、"Micro Industrial TRON" の略称です。
- TRON、ITRON、およびμITRON は、コンピュータの仕様に対する名称であり、特定の商品ないし商品群を指すものではありません。
- μITRON4.0 仕様は、(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は、(社)トロン協会ホームページ(http://www.assoc.tron.org/）から入手が可能です。
- μITRON 仕様の著作権は(社)トロン協会に属しています。
- Microsoft® Windows® 98 operating system, Microsoft® Windows® Millennium Edition(Windows® Me) operating system, Microsoft® Windows NT® operating system, Microsoft® Windows® 2000 operating system, Microsoft® Windows® XP operating system は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- SuperH™ は、(株)ルネサステクノロジの商標です。
- その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

はじめに

本マニュアルでは、μITRON4.0 仕様に準拠した機器組込み用リアルタイム・マルチタスク OS（Operating System）である HI7000/4、HI7700/4、HI7750/4（以下 HI7000/4 シリーズと略します）の使用方法について説明します。

HI7000/4 シリーズをご使用になる前に本マニュアルを良く読んで理解してください。また、下記の関連マニュアルもお読みの上、理解してください。

〈マニュアルの構成〉

本マニュアルは、以下の章と付録から構成されています。

- 第１章では、HI7000/4 シリーズの概説を記載しています。
- 第２章では、HI7000/4 シリーズのカーネルの機能概略を説明しています。
- 第３章では、HI7000/4 シリーズのカーネルのサービスコールの仕様を説明しています。
- 第４章では、アプリケーションプログラムの作成方法について説明しています。
- 第５章では、コンフィギュレータを用いたシステム構築の方法を説明しています。
- 付録では、サービスコールとエラーのリファレンス、作業領域サイズの算出方法、タイマドライバ、HI7700/4 のオプショナルな機能(最適化タイマドライバ、DSP スタンバイ制御機能)、FPU に関する注意事項、新バージョンでの変更点について説明しています。

〈関連マニュアル〉

本カーネルに関連する以下の製品のマニュアルも、あわせて参照してください。

- 製品添付のリリースノート
- 「SuperH™ RISC engine C/C++ コンパイラパッケージ」に添付のマニュアル
- 使用する SuperH™マイコンのハードウェアマニュアル、プログラミングマニュアル

〈ホームページ〉

以下の弊社ホームページにて、各種サポート情報をお知らせしておりますので、あわせてご利用ください。

http://www.renesas.com/jp/tools

〈製品名の略記〉

HI7000/4 シリーズ	HI7000/4, HI7700/4, HI7750/4 を統合した略記です。
DX	「デバッギングエクステンション」の略記です。
HEW	統合開発環境ツールである「High-performance Embedded Workshop」の略記です。

〈本マニュアルで使用する記号などの意味〉

H'と D'	"H'"は 16 進数、"D'"は 10 進数を意味するプリフィックスです。プリフィックスの無い場合は 10 進数です。
nnnn	サンプルファイル名に使用している CPU 品種を意味する数字です。例えば、タイマドライバのファイル名は ***nnnn***_tmrdrv.c と表記していますが、SH7708 用のタイマドライバファイル名は"7708_tmrdrv.c"となります。
???	"???"は、HI7700/4, HI7750/4 のファイル名に使っています。ビッグエンディアン用は"???"が"big"に、リトルエンディアン用は"???"が"little"となります。
CFG_MAXTSKID	"CFG_"で始まる文字列は、コンフィギュレータでの設定項目です。詳細は、「5.4.6 コンフィギュレータの設定項目」およびコンフィギュレータのオンラインヘルプを参照してください。

なお、日本語環境でないホストマシンでは、フォルダ区切り記号「¥」が「＼」と表示される場合があります。読み替えてご使用ください。

目次

1. 概説 1
1.1 概要 1
1.2 特長 1
1.3 動作環境 2
1.4 インストール 2
1.5 本マニュアルの対象製品 3
2. カーネル 5
2.1 概要 5
2.2 カーネルの機能 5
2.3 処理の単位と優先順位 6
2.4 システムの状態 7
2.4.1 タスクコンテキストと非タスクコンテキスト 7
2.4.2 ディスパッチ禁止/許可状態 7
2.4.3 CPU ロック/ロック解除状態 8
2.5 オブジェクト 8
2.6 タスク 9
2.6.1 タスクの状態と遷移 10
2.6.2 タスクの生成 11
2.6.3 タスクの起動 11
2.6.4 タスクのスケジューリング 12
2.6.5 タスクの終了と削除 13
2.6.6 タスクのスタック 13
2.6.7 共有スタック機能 14
2.6.8 タスク実行モード 14
2.6.9 排他制御 15
2.6.10 タスク付属イベントフラグ 16
2.7 タスク例外処理 17
2.8 セマフォ 18
2.9 イベントフラグ 20
2.10 データキュー 22
2.11 メールボックス 24
2.12 ミューテックス 26
2.13 メッセージバッファ 28
2.14 固定長メモリプール 30
2.15 可変長メモリプール 32
2.15.1 概要 32

2.15.2　空き領域の断片化とその対策 34
2.15.3　可変長メモリプールの管理 36
2.16　時間管理 37
2.16.1　周期ハンドラ 38
2.16.2　アラームハンドラ 40
2.16.3　オーバーランハンドラ 41
2.16.4　注意事項 42
2.17　システム状態管理 43
2.17.1　システムダウン 43
2.17.2　サービスコールトレース機能 44
2.18　割込み管理・システム構成管理 45
2.18.1　リセットとカーネルの起動 46
2.18.2　割込みハンドラ 47
2.18.3　割込みの禁止 48
2.18.4　カーネル割込みマスクレベル(CFG_KNLMSKLVL) 49
2.18.5　CPU 例外 49
2.19　サービスコール管理 50
2.20　キャッシュサポート（HI7700/4、HI7750/4） 51
2.21　カーネルのアイドリング 52
2.22　プリフェッチ機能(HI7700/4, HI7750/4) 52
2.23　最適化タイマドライバ機能(HI7700/4) 52
2.24　DSP スタンバイ制御機能(HI7700/4) 52
3.　サービスコール 53
3.1　概要 53
3.2　サービスコールインタフェース 54
3.2.1　C 言語 API 54
3.2.2　アセンブリ言語 API 55
3.2.3　サービスコール呼び出し前後のレジスタ保証規則 56
3.2.4　サービスコールの返値とエラコード 57
3.2.5　システム状態とサービスコール 57
3.2.6　μITRON4.0 仕様範囲外のサービスコール 59
3.3　サービスコールの説明形式 60
3.4　タスク管理機能 61
3.4.1　タスクの生成 63
＜ダイナミックスタック使用＞
(cre_tsk, icre_tsk)
(acre_tsk, iacre_tsk：ID 番号自動割付け)
＜スタティックスタック使用＞
(vscr_tsk, ivcsr_tsk)
3.4.2　タスクの削除(del_tsk) 66
3.4.3　タスクの起動(act_tsk, iact_tsk) 67
3.4.4　タスクの起動要求のキャンセル(can_act, ican_act) 68
3.4.5　タスクの起動(起動コード指定)(sta_tsk, ista_tsk) 69
3.4.6　自タスクの終了(ext_tsk) 70

自タスクの終了と削除(exd_tsk)
3.4.7 タスクの強制終了(ter_tsk) 71
3.4.8 タスク優先度の変更(chg_pri, ichg_pri) 72
3.4.9 タスク優先度の参照(get_pri, iget_pri) 73
3.4.10 タスクの状態参照(ref_tsk, iref_tsk) 74
3.4.11 タスクの状態参照(簡易版)(ref_tst, iref_tst) 77
3.4.12 タスク実行モードの変更(vchg_tmd) 79
3.5 タスク付属同期機能 80
3.5.1 起床待ち(slp_tsk, tslp_tsk) 81
3.5.2 タスクの起床(wup_tsk, iwup_tsk) 82
3.5.3 タスク起床要求のキャンセル(can_wup, ican_wup) 83
3.5.4 待ち状態の強制解除(rel_wai, irel_wai) 84
3.5.5 強制待ち状態への移行(sus_tsk, isus_tsk) 85
3.5.6 強制待ち状態からの再開(rsm_tsk, irsm_tsk) 86
強制待ち状態からの強制再開(frsm_tsk, ifrsm_tsk)
3.5.7 タスク遅延(dly_tsk) 87
3.5.8 タスク付属イベントフラグのセット(vset_tfl, ivset_tfl) 88
3.5.9 タスク付属イベントフラグのクリア(vclr_tfl, ivclr_tfl) 89
3.5.10 タスク付属イベントフラグ待ち(vwai_tfl, vpol_tfl, vtwai_tfl) 90
3.6 タスク例外処理機能 91
3.6.1 タスク例外処理ルーチンの定義(def_tex, idef_tex) 92
3.6.2 タスク例外処理の要求(ras_tex, iras_tex) 94
3.6.3 タスク例外処理の禁止(dis_tex) 95
3.6.4 タスク例外処理の許可(ena_tex) 96
3.6.5 タスク例外禁止状態の参照(sns_tex) 97
3.6.6 タスク例外処理の状態参照(ref_tex, iref_tex) 98
3.7 同期・通信(セマフォ)機能 99
3.7.1 セマフォの生成(cre_sem, icre_sem) 100
(acre_sem, iacre_sem：ID 番号自動割付け)
3.7.2 セマフォの削除(del_sem) 102
3.7.3 セマフォ資源の返却(sig_sem, isig_sem) 103
3.7.4 セマフォ資源の獲得(wai_sem, pol_sem, ipol_sem, twai_sem) 104
3.7.5 セマフォの状態参照(ref_sem, iref_sem) 105
3.8 同期・通信(イベントフラグ)機能 106
3.8.1 イベントフラグの生成(cre_flg, icre_flg) 107
(acre_flg, iacre_flg：ID 番号自動割付け)
3.8.2 イベントフラグの削除(del_flg) 109
3.8.3 イベントフラグのセット(set_flg, iset_flg) 110
3.8.4 イベントフラグのクリア(clr_flg, iclr_flg) 111
3.8.5 イベントフラグ待ち(wai_flg, pol_flg, ipol_flg, twai_flg) 112
3.8.6 イベントフラグの状態参照(ref_flg, iref_flg) 114
3.9 同期・通信(データキュー)機能 115
3.9.1 データキューの生成(cre_dtq, icre_dtq) 116
(acre_dtq, iacre_dtq：ID 番号自動割付け)
3.9.2 データキューの削除(del_dtq) 118
3.9.3 データキューへの送信(snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq, fsnd_dtq,

ifsnd_dtq) 119
3.9.4 データキューからの受信(rcv_dtq, prcv_dtq, trcv_dtq) 121
3.9.5 データキューの状態参照(ref_dtq, iref_dtq) 123
3.10 同期･通信(メールボックス)機能 124
3.10.1 メールボックスの生成(cre_mbx, icre_mbx) 125
(acre_mbx, iacre_mbx：ID 番号自動割付け)
3.10.2 メールボックスの削除(del_mbx) 127
3.10.3 メールボックスへの送信(snd_mbx, isnd_mbx) 128
3.10.4 メールボックスからの受信(rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx) 130
3.10.5 メールボックスの状態参照(ref_mbx, iref_mbx) 132
3.11 拡張同期・通信(ミューテックス)機能 133
3.11.1 ミューテックスの生成(cre_mtx) 134
(acre_mtx：ID 番号自動割付け)
3.11.2 ミューテックスの削除(del_mtx) 136
3.11.3 ミューテックス資源の獲得(loc_mtx, ploc_mtx, tloc_mtx) 137
3.11.4 ミューテックスのロック解除(unl_mtx) 138
3.11.5 ミューテックスの状態参照(ref_mtx) 139
3.12 拡張同期・通信(メッセージバッファ)機能 140
3.12.1 メッセージバッファの生成(cre_mbf, icre_mbf) 141
(acre_mbf, iacre_mbf：ID 番号自動割付け)
3.12.2 メッセージバッファの削除(del_mbf) 143
3.12.3 メッセージバッファへの送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf) 144
3.12.4 メッセージバッファからの受信(rcv_mbf, prcv_mbf, trcv_mbf) 146
3.12.5 メッセージバッファの状態参照(ref_mbf, iref_mbf) 148
3.13 メモリプール管理(固定長メモリプール)機能 149
3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf) 150
(acre_mpf, iacre_mpf：ID 番号自動割付け)
3.13.2 固定長メモリプールの削除(del_mpf) 153
3.13.3 固定長メモリブロックの獲得(get_mpf, pget_mpf, ipget_mpf, tget_mpf) 154
3.13.4 固定長メモリブロックの返却(rel_mpf, irel_mpf) 156
3.13.5 固定長メモリプールの状態参照(ref_mpf, iref_mpf) 157
3.14 メモリプール管理(可変長メモリプール)機能 158
3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl) 159
(acre_mpl, iacre_mpl：ID 番号自動割付け)
3.14.2 可変長メモリプールの削除(del_mpl) 163
3.14.3 可変長メモリブロックの獲得(get_mpl, pget_mpl, ipget_mpl, tget_mpl) 164
3.14.4 可変長メモリブロックの返却(rel_mpl, irel_mpl) 166
3.14.5 可変長メモリプールの状態参照(ref_mpl, iref_mpl) 167
3.15 時間管理機能(システム時刻管理) 168
3.15.1 システム時刻の設定(set_tim, iset_tim) 169
3.15.2 システム時刻の参照(get_tim, iget_tim) 170
3.15.3 タイムティックの供給(isig_tim) 171
3.16 時間管理機能(周期ハンドラ) 172
3.16.1 周期ハンドラの生成(cre_cyc, icre_cyc) 173
(acre_cyc, iacre_cyc：ID 番号自動割付け)
3.16.2 周期ハンドラの削除(del_cyc) 175

RENESAS

3.16.3　周期ハンドラの動作開始(sta_cyc, ista_cyc) 176
3.16.4　周期ハンドラの動作停止(stp_cyc, istp_cyc) 177
3.16.5　周期ハンドラの状態参照(ref_cyc, iref_cyc) 178
3.17　時間管理機能(アラームハンドラ) 179
3.17.1　アラームハンドラの生成(cre_alm, icre_alm) 180
(acre_alm, iacre_alm：ID 番号自動割付け)
3.17.2　アラームハンドラの削除(del_alm) 182
3.17.3　アラームハンドラの動作開始(sta_alm, ista_alm) 183
3.17.4　アラームハンドラの動作停止(stp_alm, istp_alm) 184
3.17.5　アラームハンドラの状態参照(ref_alm, iref_alm) 185
3.18　時間管理機能(オーバーランハンドラ) 186
3.18.1　オーバーランハンドラの定義(def_ovr) 187
3.18.2　オーバーランハンドラの動作開始(sta_ovr, ista_ovr) 188
3.18.3　オーバーランハンドラの動作停止(stp_ovr, istp_ovr) 189
3.18.4　オーバーランハンドラの状態参照(ref_ovr, iref_ovr) 190
3.19　システム状態管理機能 191
3.19.1　タスクの優先順位の回転(rot_rdq, irot_rdq) 192
3.19.2　実行状態のタスク ID の参照(get_tid, iget_tid) 193
3.19.3　CPU ロック状態への移行(loc_cpu, iloc_cpu) 194
3.19.4　CPU ロック状態の解除(unl_cpu, iunl_cpu) 195
3.19.5　ディスパッチの禁止(dis_dsp) 196
3.19.6　ディスパッチの許可(ena_dsp) 197
3.19.7　コンテキストの参照(sns_ctx) 198
3.19.8　CPU ロック状態の参照(sns_loc) 199
3.19.9　ディスパッチ禁止状態の参照(sns_dsp) 200
3.19.10　ディスパッチ保留状態の参照(sns_dpn) 201
3.19.11　カーネルの起動(vsta_knl, ivsta_knl) 202
3.19.12　システムダウン(vsys_dwn, ivsys_dwn) 203
3.19.13　トレースの取得(vget_trc, ivget_trc) 204
3.19.14　割込みハンドラの開始をトレースに取得(ivbgn_int) 205
3.19.15　割込みハンドラの終了をトレースに取得(ivend_int) 206
3.20　割込み管理機能 207
3.20.1　割込みハンドラの定義(def_inh, idef_inh) 208
3.20.2　割込みマスクの変更(chg_ims, ichg_ims) 211
3.20.3　割込みマスクの参照(get_ims, iget_ims) 212
3.21　サービスコール管理機能 213
3.21.1　拡張サービスコールの定義(def_svc, idef_svc) 214
3.21.2　サービスコールの呼び出し(cal_svc, ical_svc) 215
3.22　システム構成管理機能 216
3.22.1　CPU 例外ハンドラの定義(def_exc, idef_exc) 217
3.22.2　CPU 例外(TRAPA 命令例外)ハンドラ定義(vdef_trp, ivdef_trp) 219
3.22.3　コンフィグレーション情報の参照(ref_cfg, iref_cfg) 221
3.22.4　バージョン情報の参照(ref_ver, iref_ver) 223
3.23　キャッシュサポート機能(HI7700/4: SH-3, SH3-DSP 用) 225
3.23.1　キャッシュの初期化(vini_cac, ivini_cac) 226
3.23.2　キャッシュのクリア(vclr_cac, ivclr_cac) 228

3.23.3 キャッシュのフラッシュ(vfls_cac, ivfls_cac) 229
3.23.4 キャッシュの無効化(vinv_cac, ivinv_cac) 230
3.24 キャッシュサポート機能(HI7750/4:SH-4 用) 231
3.24.1 キャッシュの初期化(vini_cac, ivini_cac) 232
3.24.2 オペランドキャッシュのクリア(vclr_cac, ivclr_cac) 233
3.24.3 オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac) 234
3.24.4 オペランドキャッシュの無効化(vinv_cac, ivinv_cac) 235
3.25 キャッシュサポート機能(HI7700/4:SH4AL-DSP(拡張機能なし), HI7750/4:SH-4A(拡張機能なし)用) 236
3.25.1 キャッシュの初期化(vini_cac, ivini_cac) 237
3.25.2 命令・オペランドキャッシュのクリア(vclr_cac, ivclr_cac) 238
3.25.3 オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac) 240
3.25.4 命令・オペランドキャッシュの無効化(vinv_cac, ivinv_cac) 242
3.26 キャッシュサポート機能(HI7700/4：SH4AL-DSP, 拡張機能あり, HI7750/4：SH-4A, 拡張機能あり用) 244
3.26.1 キャッシュの初期化(vini_cac, ivini_cac) 245
3.26.2 命令・オペランドキャッシュのクリア(vclr_cac, ivclr_cac) 247
3.26.3 オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac) 249
3.26.4 命令・オペランドキャッシュの無効化(vinv_cac, ivinv_cac) 251

4. アプリケーションプログラムの作成 253

4.1 ヘッダファイル 253
4.1.1 C/C++言語用 253
4.1.2 アセンブリ言語用 255
4.2 CPU リソースの扱い等 256
4.2.1 SR レジスタ 256
4.2.2 キャッシュロック機能(SH-3, SH3-DSP) 257
4.2.3 VBR レジスタ 257
4.2.4 MMU(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A) 257
4.2.5 SR.BL=1 期間中の NMI の受付け（SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A） 257
4.2.6 割込みのネスト(SH-3, SH3-DSP, SH4AL-DSP,. SH-4, SH-4A) 257
4.2.7 32 ビットアドレス拡張モード(SH-4A) 257
4.2.8 TBR レジスタ(SH-2A, SH2A-FPU) 258
4.2.9 レジスタバンク(SH-2A, SH2A-FPU) 258
4.3 SH2A-FPU, SH-4, SH-4A を使用する場合 259
4.4 システム予約 259
4.4.1 予約名 259
4.4.2 予約トラップ(HI7000/4 のみ) 259
4.5 タスク 260
4.6 タスク例外処理ルーチン 265
4.7 拡張サービスコールルーチン 269
4.8 割込みハンドラ 270
4.8.1 通常の割込みハンドラ 270
4.8.2 ダイレクト割込みハンドラ（HI7000/4） 276

RENESAS

4.9 CPU 例外ハンドラ(TRAPA 例外を含む) ... 284
4.10 タイムイベントハンドラ、初期化ルーチン ... 288
4.11 CPU 初期化ルーチン ... 293
4.11.1 CPU 初期化ルーチンの作成 ... 293
4.11.2 HI7000/4 の CPU 初期化ルーチンの登録 ... 293
4.11.3 HI7700/4, HI7750/4 での CPU 初期化ルーチンの登録 ... 293
4.12 システムダウンルーチン ... 294
4.13 DSP を使用する場合(HI7000/4, HI7700/4) ... 295
4.13.1 DSR レジスタの初期化について ... 295
4.13.2 ハンドラ等での DSP の使用 ... 296

5. コンフィギュレーション ... 297

5.1 前知識 ... 297
5.1.1 一括リンクと分割リンク ... 297
5.2 フォルダ構成 ... 299
5.2.1 hihead フォルダ ... 299
5.2.2 hisys フォルダ ... 299
5.2.3 hilib フォルダ ... 299
5.2.4 knl フォルダ ... 299
5.2.5 samples¥sh***nnnn*** フォルダ ... 299
5.3 作業手順 ... 303
5.4 コンフィギュレータ ... 303
5.4.1 概要 ... 303
5.4.2 コンフィギュレータの構成 ... 304
5.4.3 ファイル操作 ... 304
5.4.4 コンフィギュレーションファイル ... 305
5.4.5 分割リンク ... 307
5.4.6 コンフィギュレータの設定項目 ... 308
5.5 最適化タイマドライバを使用する場合(HI7700/4 のみ) ... 315
5.6 DSP スタンバイ制御機能を使用する場合(HI7700/4 のみ) ... 315
5.7 SH4AL-DSP(HI7700/4)または SH-4A(HI7750/4)でキャッシュサポートサービスコールを使用する場合 ... 315
5.8 HEW ワークスペースとプロジェクト ... 316
5.9 カーネルライブラリ ... 317
5.9.1 HI7000/4 ... 317
5.9.2 HI7700/4 ... 318
5.9.3 HI7750/4 ... 319
5.10 セクション構成 ... 320
5.11 各プロジェクト共通の設定 ... 322
5.11.1 コンパイラ、アセンブラの CPU オプション ... 322
5.11.2 コンパイラの GBR オプション(コンパイラパッケージ V7.1 以降) ... 322
5.11.3 コンパイラの PACK オプションと#pragma pack について(コンパイラパッケージ V8 以降) ... 322
5.11.4 C/C++コンパイラ、アセンブラのインクルードディレクトリ ... 322
5.11.5 SH2A-FPU, SH-4, SH-4A 使用時 ... 322

5.11.6 コンパイラの TBR オプション(コンパイラパッケージ V9 以降) 323
5.12 一括リンクでのビルド(mix プロジェクト) 324
5.12.1 プロジェクトへの登録 324
5.12.2 エンディアンの指定 324
5.12.3 最適化リンケージエディタの設定 325
5.12.4 ビルドの実行 327
5.13 分割リンク：カーネル側のビルド(def プロジェクト) 328
5.13.1 プロジェクトへの登録 328
5.13.2 エンディアンの指定（HI7700/4、HI7750/4） 328
5.13.3 最適化リンケージエディタの設定 329
5.13.4 ビルドの実行 331
5.14 分割リンク：カーネル環境側のビルド(cfg プロジェクト) 332
5.14.1 プロジェクトへの登録 332
5.14.2 エンディアンの指定（HI7700/4、HI7750/4） 332
5.14.3 最適化リンケージエディタの設定 332
5.14.4 ビルドの実行 333
5.15 アプリケーションロードモジュールの作成 334

付録 A サービスコール一覧 335

付録 B エラー一覧 343

B.1 サービスコールエラーコード一覧 343
B.2 システムダウン時の情報 344
B.3 コンパイル時のエラー 345

付録 C 作業領域サイズの算出 347

C.1 作業領域の内訳 347
C.2 スタックの分類 348
C.3 スタック使用量の算出手順 349
C.4 関数単体のスタック使用量 350
C.5 プログラムネストを加味したスタック使用量 351
C.6 タスクのスタック 353
C.7 割込みハンドラのスタック 355
C.8 タイムイベントハンドラ、タイマ割込みルーチンのスタック 357
C.9 初期化ルーチンのスタック 358
C.10 タイマ初期化ルーチンのスタック 358

付録 D タイマドライバ 359

D.1 概要 359
D.2 標準タイマドライバ 359

付録 E 最適化タイマドライバ(HI7700/4) 363

E.1 概要 363
E.2 動作 363
E.3 利用可能なマイコン 364

E.4　ハードウェア初期化処理 365
E.5　標準タイマドライバとの相違 365
E.6　組み込み方法 366
E.7　使用するカーネルライブラリ 367

付録 F　DSP スタンバイ制御機能(HI7700/4) 369

F.1　概要 369
F.2　利用可能なマイコン 370
F.3　各プログラム起動時のモジュールスタンバイ状態 370
F.4　TA_COP0 属性変更サービスコール(vchg_cop) 371
F.5　組み込み方法 372
F.6　使用するカーネルライブラリ 373
F.7　注意事項 374

付録 G　SH2A-FPU, SH-4, SH-4A における FPU に関する注意 375

G.1　タスク、タスク例外処理ルーチン 375
G.2　非タスクコンテキスト(通常の割込みハンドラ、ダイレクト割込みハンドラ、CPU 例外ハンドラ、タイムイベントハンドラ、初期化ルーチン) 376
G.3　拡張サービスコールルーチン 379
G.4　参考情報 381

付録 H　HI7000/4 V.2 の新機能 385

H.1　SH-2A, SH2A-FPU のサポート 385
H.2　タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.00 Release 00) 386
H.3　固定長メモリプールの管理方式(V.2.00 Release 00) 387
H.4　ダイレクト割込みハンドラ(V.2.00 Release 00, V.2.02 Release 00) 387
H.5　サイズを算出するマクロ(V.2.00 Release 00) 388
H.6　最大ベクタ番号の拡張(V.2.00 Release 00) 388
H.7　ID 名称(V.2.00 Release 00) 388
H.8　SH-2 リトルエンディアンのサポート(V.2.00 Release 01) 388
H.9　可変長メモリプールの改善(V.2.01 Release 00) 389
H.10　DSR レジスタの初期値(V.2.01 Release 00) 389
H.11　タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00) 389
H.12　ベクタ番号 16～31 の扱い(V.2.01 Release 00, V.2.02 Release 00) 390
H.13　構造体アライメントに関する制限事項の解除(V.2.01 Release 00) 390
H.14　コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00) 390

付録 I　HI7700/4 V.2 の新機能 391

I.1　SH4AL-DSP(拡張機能あり)のサポート(V.2.01 Release 00) 391
I.2　タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00) 391

I.3 固定長メモリプールの管理方式(V.2.01 Release 00) 391
I.4 可変長メモリプールの改善(V.2.01 Release 00) 392
I.5 サイズを算出するマクロ(V.2.01 Release 00) 393
I.6 DSR レジスタの初期値(V.2.01 Release 00) 393
I.7 タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00) 393
I.8 最大例外コード(CFG_MAXVCTNO)の拡張(V.2.01 Release 00) 393
I.9 TRAPA #16～31 の扱い(V.2.01 Release 00) 394
I.10 構造体アライメントに関する制限事項の解除(V.2.01 Release 00) 394
I.11 ID 名称(V.2.01 Release 00) 394
I.12 コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00) 394

付録 J HI7750/4 V.2 の新機能 395

J.1 SH-4A(拡張機能あり)のサポート(V.2.01 Release 00) 395
J.2 タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00) 395
J.3 固定長メモリプールの管理方式(V.2.01 Release 00) 395
J.4 可変長メモリプールの改善(V.2.01 Release 00) 396
J.5 サイズを算出するマクロ(V.2.01 Release 00) 397
J.6 タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00) 397
J.7 最大例外コード(CFG_MAXVCTNO)の拡張(V.2.01 Release 00) 397
J.8 TRAPA #16～31 の扱い(V.2.01 Release 00) 397
J.9 構造体アライメントに関する制限事項の解除(V.2.01 Release 00) 398
J.10 ID 名称(V.2.01 Release 00) 398
J.11 コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00) 398

図目次

図2.1　タスクの状態遷移 10
図2.2　共有スタック機能使用時のタスク状態遷移 14
図2.3　タスク付属イベントフラグの動作例 16
図2.4　タスク例外処理の動作例 17
図2.5　セマフォの動作例 19
図2.6　イベントフラグの動作例 21
図2.7　データキューの動作例 23
図2.8　メールボックスの動作例 25
図2.9　ミューテックスの動作例 27
図2.10　メッセージバッファ動作例 29
図2.11　固定長メモリプールの動作例 31
図2.12　可変長メモリプールの動作例 33
図2.13　空き領域の断片化 34
図2.14　可変長メモリプールの例 35
図2.15　周期ハンドラの動作例 39
図2.16　アラームハンドラの動作例 40
図2.17　オーバーランハンドラの動作例 41
図2.18　リセットからカーネル起動までの流れ 46
図2.19　拡張サービスコールの概要 50
図2.20　キャッシュサポートの概要 51
図3.1　アセンブリ言語プログラムからのサービスコール呼び出し例 55
図3.2　TBRを使用してサービスコールを呼び出す場合の記述例(SH-2A, SH2A-FPU) 55
図3.3　サービスコールの説明形式 60
図3.4　メッセージの形式例 129
図3.5　優先度付きメッセージの形式例 129
図4.1　タスクのC言語記述例 260
図4.2　タスク例外処理ルーチンのC言語記述例 265
図4.3　拡張サービスコールルーチンのC言語記述例 269
図4.4　通常の割込みハンドラのC言語記述例 270
図4.5　IRL割込みハンドラのC言語記述例 274
図4.6　ダイレクト割込みハンドラのC言語記述例 276
図4.7　ダイレクト割込みハンドラのアセンブリ言語記述例 279
図4.8　IRLダイレクト割込みハンドラの本体処理のC言語記述例 281
図4.9　IRLダイレクト割込みハンドラインタフェースルーチンの記述例 282
図4.10　周期ハンドラ、アラームハンドラ、初期化ルーチンのC言語記述例 288
図4.11　オーバーランハンドラのC言語記述例 288
図4.12　リセットベクタの作成例(HI7000/4) 293
図4.13　DSPを使用するハンドラ等のC言語記述例 296
図5.1　一括リンク方式によるロードモジュールの生成 297
図5.2　分割リンク方式によるロードモジュールの生成 298
図5.3　kernelフォルダ以下の構成 299
図5.4　sh***nnnn***フォルダ以下の構成(HEW3以上用) 301
図5.5　sh***nnnn***フォルダ以下の構成(HEW1.2用) 302
図5.6　システム構築におけるコンフィギュレータの位置付け 303
図5.7　コンフィギュレータ概観 304

図5.8　フォルダ選択ダイアログ 304
図5.9　プロジェクトの選択 317
図5.10　エンディアンの選択 324
図5.11　[Input]カテゴリの[Library files] 325
図5.12　[Input]カテゴリの[Defines] 325
図5.13　[Section]カテゴリ 326
図5.14　ビルドの実行 327
図5.15　[Input]カテゴリの[Library files] 329
図5.16　[Input]カテゴリの[Defines] 329
図5.17　[Section]カテゴリ 330
図5.18　[Input]カテゴリの[Defines] 332
図5.19　[Section]カテゴリ 333
図C.1　スタック使用量の算出手順 349
図C.2　コンパイルリストとスタックの使用量 350
図C.3　プログラムネスト状況 351
図D.1　タイマ割込みルーチンのC言語記述例 360
図E.1　動作例 363
図E.2　最適化タイマドライバによる効果イメージ 364
図F.1　動作概要 369
図G.1　タスクにおけるFPSCR初期化例 375
図G.2　割込みハンドラにおけるFPSCR初期化・FPUレジスタ保証例(SH-4, SH-4A) 377
図G.3　拡張サービスコールルーチンにおけるFPSCRの初期化・保証例 379

表目次

表1.1　動作環境 2
表2.1　タスクコンテキストと非タスクコンテキスト 7
表2.2　タスクを操作するサービスコール（タスク管理機能） 9
表2.3　タスクを操作するサービスコール（タスク付属同期機能） 9
表2.4　タスクの生成方法 11
表2.5　排他制御方法 15
表2.6　タスク付属イベントフラグを操作するサービスコール 16
表2.7　タスク例外処理を操作するサービスコール 17
表2.8　セマフォを操作するサービスコール 18
表2.9　イベントフラグを操作するサービスコール 20
表2.10　データキューを操作するサービスコール 22
表2.11　メールボックスを操作するサービスコール 24
表2.12　ミューテックスを操作するサービスコール 26
表2.13　メッセージバッファを操作するサービスコール 28
表2.14　固定長メモリプールを操作するサービスコール 30
表2.15　可変長メモリプールを操作するサービスコール 32
表2.16　微小ブロックの扱い 34
表2.17　システム時刻を操作するサービスコール 37
表2.18　周期ハンドラを操作するサービスコール 38
表2.19　アラームハンドラを操作するサービスコール 40
表2.20　オーバーランハンドラを操作するサービスコール 41
表2.21　システム管理のサービスコール 43
表2.22　割込み管理機能のサービスコール 45
表2.23　システム構成管理機能のサービスコール 45
表2.24　２つの割込みハンドラの違い 47
表2.25　拡張サービスコールを操作するサービスコール 50
表2.26　キャッシュを操作するサービスコール 51
表3.1　サービスコールの機能分類 53
表3.2　基本データ型 54
表3.3　サービスコール発行前後のレジスタ保証 56
表3.4　タスク管理サービスコール 61
表3.5　タスク管理の仕様 62
表3.6　タスク生成時に行われる処理 64
表3.7　タスク起動時に行われる処理 67
表3.8　タスク終了時に行われる処理 70
表3.9　タスク付属同期サービスコール 80
表3.10　タスク付属同期の仕様 80
表3.11　タスク例外処理サービスコール 91
表3.12　タスク例外の仕様 91
表3.13　同期･通信（セマフォ）サービスコール 99
表3.14　セマフォの仕様 99
表3.15　同期･通信（イベントフラグ）サービスコール 106
表3.16　イベントフラグの仕様 106
表3.17　同期･通信（データキュー）サービスコール 115
表3.18　データキューの仕様 115

表3.19　同期･通信（メールボックス）サービスコール.. 124
表3.20　メールボックスの仕様.. 124
表3.21　同期･通信（ミューテックス）サービスコール.. 133
表3.22　ミューテックスの仕様.. 133
表3.23　同期･通信（メッセージバッファ）サービスコール... 140
表3.24　メッセージバッファの仕様.. 140
表3.25　メモリプール管理（固定長メモリプール）サービスコール...................................... 149
表3.26　固定長メモリプールの仕様.. 149
表3.27　メモリプール管理（可変長メモリプール）サービスコール...................................... 158
表3.28　可変長メモリプールの仕様.. 158
表3.29　システム時刻管理サービスコール 168
表3.30　システム時刻管理の仕様.. 168
表3.31　周期ハンドラサービスコール 172
表3.32　周期ハンドラの仕様.. 172
表3.33　アラームハンドラサービスコール 179
表3.34　アラームハンドラの仕様.. 179
表3.35　オーバーランハンドラサービスコール 186
表3.36　オーバーランハンドラの仕様.. 186
表3.37　システム状態管理サービスコール 191
表3.38　割込み管理サービスコール 207
表3.39　割込み管理の仕様.. 207
表3.40　サービスコール管理サービスコール 213
表3.41　サービスコール管理の仕様.. 213
表3.42　システム構成管理サービスコール 216
表3.43　システム構成管理の仕様.. 216
表3.44　キャッシュサポート機能のライブラリ 225
表3.45　キャッシュサポートサービスコール(SH-3, SH3-DSP用).. 225
表3.46　キャッシュサポート機能のライブラリ 231
表3.47　キャッシュサポートサービスコール(SH-4用).. 231
表3.48　キャッシュサポート機能のライブラリ 236
表3.49　キャッシュサポートサービスコール(SH4AL-DSP(拡張機能なし), SH-4A(拡張機能なし)用).. 236
表3.50　キャッシュ属性.. 237
表3.51　キャッシュサポート機能のライブラリ 244
表3.52　キャッシュサポートサービスコール(SH4AL-DSP(拡張機能あり), SH-4A(拡張機能あり)用).. 244
表3.53　キャッシュ属性.. 245
表4.1　定数、マクロ 253
表4.2　タスクのレジスタ使用規約(HI7000/4) 261
表4.3　タスクのレジスタ使用規約(HI7700/4) 262
表4.4　タスクのレジスタ使用規約(HI7750/4) 263
表4.5　タスク管理情報の初期化内容 264
表4.6　タスク例外処理ルーチンのレジスタ使用規約(HI7000/4).. 266
表4.7　タスク例外処理ルーチンのレジスタ使用規約(HI7700/4).. 267
表4.8　タスク例外処理ルーチンのレジスタ使用規約(HI7750/4).. 268
表4.9　通常の割込みハンドラのレジスタ使用規約(HI7000/4).. 271
表4.10　通常の割込みハンドラのレジスタ使用規約(HI7700/4).. 272

表4.11　通常の割込みハンドラのレジスタ使用規約(HI7750/4) 273
表4.12　“tn=”指定と“resbank”指定 (HI7000/4) 277
表4.13　ダイレクト割込みハンドラのレジスタ使用規約(HI7000/4) 278
表4.14　ダイレクト割込みハンドラの終了命令(HI7000/4) 280
表4.15　CPU例外ハンドラのレジスタ使用規約(HI7000/4) 284
表4.16　CPU例外ハンドラのレジスタ使用規約(HI7700/4) 285
表4.17　CPU例外ハンドラのレジスタ使用規約(HI7750/4) 285
表4.18　タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7000/4) 289
表4.19　タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7700/4) 290
表4.20　タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7750/4) 291
表4.21　DSRレジスタの初期値 295
表5.1　HI7000/4 V.2.02における***nnnn***の対応 300
表5.2　HI7700/4 V.2.02における***nnnn***の対応 300
表5.3　HI7750/4 V.2.02における***nnnn***の対応 300
表5.4　コンフィギュレーションファイル 305
表5.5　コンフィギュレータの設定項目 308
表5.6　カーネルライブラリのリンク優先順位(HI7000/4) 317
表5.7　カーネルライブラリのリンク優先順位(HI7700/4) 318
表5.7　カーネルライブラリのリンク優先順位(HI7700/4) (続き) 319
表5.8　カーネルライブラリのリンク優先順位(HI7750/4) 319
表5.9　セクション一覧 321
表5.10　CPUオプション 322
表5.11　プロジェクトに登録するソースプログラムファイル(mix) 324
表5.12　プロジェクトに登録するソースプログラムファイル(def) 328
表5.13　プロジェクトに登録するソースプログラムファイル(cfg) 332
表B.1　サービスコールエラーコード一覧 343
表B.2　システムダウンルーチンに渡される情報 344
表B.3　最適化タイマドライバに関するエラー 345
表B.4　DSPスタンバイ制御機能に関するエラー 345
表C.1　作業領域の内訳 347
表C.2　呼び出しルーチン、ハンドラの加算サイズ 351
表C.3　個々の関数のスタック使用量 352
表C.4　呼び出し経路を考慮した使用量 352
表C.5　個々のタスクのスタック使用量 353
表C.6　個々の割込みハンドラのスタック使用量 355
表C.7　タイムイベントハンドラ、タイマ割込みルーチンのスタック使用量 357
表C.8　個々の初期化ルーチンのスタック使用量 358
表C.9　タイマ初期化ルーチンのスタック使用量 358
表D. 1　サンプルタイマドライバファイル一覧とクロックソース 362
表E.1　標準タイマドライバと最適化タイマドライバの相違 365
表E.2　タイマドライバの組み込み方法 366
表F.1　各プログラム起動時のモジュールスタンバイ状態 370
表F.2　DSPスタンバイ制御機能の組み込み方法 372
表F.3　各マイコンでの指定値 373
表G.1　TA_COP1, TA_COP2属性の指定 376
表G.2　呼び出し元に必要なTA_COP1, TA_COP2属性 380
表G.3　タスクやハンドラなどの起動時の状態 381

表G.4　コンパイラによるFPSCR.PRビットの扱い 383
表G.5　コンパイラによるFPSCR.DNビットの扱い 383
表G.6　コンパイラによるFPSCR.RMビットの扱い 383
表H.1　ID名称に関する機能追加 388
表I.1　ID名称に関する機能追加 394
表J.1　ID名称に関する機能追加 398

1. 概説

1.1 概要

マイクロコンピュータ応用分野の広がりに伴い、OS(Operating System)の役割と重要性が高まってきています。このなかで、産業用システムに用いられる OS としてリアルタイム OS があります。

HI7000/4 シリーズは、μITRON4.0 仕様に準拠した SH マイコン用のリアルタイム・マルチタスク OS です。

1.2 特長

HI7000/4 シリーズの特長を以下に示します。

(1) 豊富なリアルタイム・マルチタスクサービス

HI7000/4 シリーズのカーネルは、μITRON4.0 仕様に準拠しています。

- 優先度に基づくタスクスケジューリング
- タスクの生成、削除、起動、終了などのタスク管理機能
- タスクの実行中断、再開、タスク付属イベントフラグなどのタスク付属同期機能
- タスク例外処理ルーチンの定義、タスク例外処理の要求、タスク例外処理の許可・禁止などのタスク例外処理機能
- セマフォ、イベントフラグ、データキュー、メールボックスによるタスク間の同期・通信機能
- ミューテックス、メッセージバッファによるタスク間の拡張同期・通信機能
- メモリブロックの獲得、返却などのメモリプール管理機能
- システムクロックの設定､参照、周期ハンドラ、アラームハンドラ、オーバーランハンドラなどの時間管理機能
- システム状態管理機能
- 割込み管理機能
- 拡張サービスコールの定義、呼び出しなどのサービスコール管理機能
- CPU 例外ハンドラの定義などのシステム構成管理機能
- DSP, FPU のサポート (SH-2E の FPU はサポートしていません)
- キャッシュサポート機能(HI7700/4, HI7750/4 のみ)
- 省電力化のための最適化タイマドライバと DSP スタンバイ制御機能(HI7700/4 のみ)

(2) 機能選択可能でコンパクトなカーネル

ユーザシステムに必要となる ROM/RAM を最小にできるように、カーネルのプログラムサイズおよび作業領域の小型化を図っています。

また、ユーザプログラムで使用するカーネル機能を選択することで、ユーザシステム用に最適化されたカーネルを構築することができます。

(3) サンプルプログラム

サンプルプログラムとして、以下のソースプログラムを提供しています。必要に応じてユーザシステム用に修正するだけで容易にシステムを構築できます。

- システムダウンルーチン
- 各種 SuperH™マイコン内蔵タイマ用のタイマドライバ

- CPU 初期化ルーチン
- セクション初期化、情報定義ファイル

(4) コンフィギュレータ

カーネルの構築作業を容易にするコンフィギュレータを装備しています。

(5) デバッギングエクステンション(オプション製品)

HEW3 以降にマルチタスクデバッグ機能を追加するデバッギングエクステンションを用意しています。デバッギングエクステンションでは、以下のような機能をサポートしています。

- タスクなどのオブジェクトの状態参照
- タスクの起動やイベントフラグのセットといった、オブジェクトに対する操作
- サービスコール履歴の表示

なお、デバッギングエクステンションは、当社ホームページより無償でダウンロードできます。

1.3 動作環境

表 1.1に、動作環境を示します。

表1.1 動作環境

項番	製品名	構成部品	動作環境
1	HI7000/4	カーネル	SH-1, SH-2, SH2-DSP, SH-2A, SH2A-FPU を搭載した SuperH™マイコン全般
2		サンプルプログラム	SH-1, SH-2, SH2-DSP, SH-2A, SH2A-FPU を搭載した各種 SuperH™マイコン
3		サンプル HEW ワークスペース、プロジェクト	HEW Ver.1.2 以降（SuperH™ RISC Engine C/C++コンパイラパッケージ V6.0C 以降）
4		コンフィギュレータ	Windows® 98, Windows® Millennium Edition(Windows® Me), WindowsNT® 4.0, Windows® 2000, Windows® XP
5	HI7700/4	カーネル	SH-3, SH3-DSP, SH4AL-DSP を搭載した SuperH™マイコン全般
6		サンプルプログラム	SH-3, SH3-DSP, SH4AL-DSP を搭載した各種 SuperH™マイコン
7		サンプル HEW ワークスペース、プロジェクト	HEW Ver.1.2 以降（SuperH™ RISC Engine C/C++コンパイラパッケージ V6.0C 以降）
8		コンフィギュレータ	Windows® 98, Windows® Millennium Edition(Windows® Me), WindowsNT® 4.0, Windows® 2000, Windows® XP
9	HI7750/4	カーネル	SH-4, SH-4A を搭載した SuperH™マイコン全般
10		サンプルプログラム	SH-4, SH-4A を搭載した各種 SuperH™マイコン
11		サンプル HEW ワークスペース、プロジェクト	HEW Ver.1.2 以降（SuperH™ RISC Engine C/C++コンパイラパッケージ V6.0C 以降）
12		コンフィギュレータ	Windows® 98, Windows® Millennium Edition(Windows® Me), WindowsNT® 4.0, Windows® 2000, Windows® XP

1.4 インストール

インストール方法は、製品添付のリリースノートを参照してください。

1.5 本マニュアルの対象製品

本マニュアルは、以下の製品用に作成されています。

(1) HI7000/4：V.2.02.Release 00以降

(2) HI7700/4：V.2.02 Release 00以降

(3) HI7750/4：V.2.02.Release 00以降

2. カーネル

2.1 概要

リアルタイム・マルチタスク処理を行う核となる部分を「カーネル」と呼びます。
以下にユーザが利用するカーネルの 3 つの役割を示します。

(1) 各事象への対応

非同期に発生する事象（イベント）を認識し、その事象を処理する仕事（タスク）を直ちに実行します。

(2) タスクのスケジューリング

優先度に応じて、タスクの実行順序を決定します。

(3) サービスコールの実行

タスクからの各種処理要求（サービスコール）を受け付け、その処理を行います。

2.2 カーネルの機能

カーネル機能のほとんどは、アプリケーションプログラムからサービスコール呼び出しという形で利用することができます。

(1) タスク管理機能

タスクは実行時に CPU が割り付けられます。カーネルは、割付け順序を制御したり、タスクの起動、終了などタスクの状態を管理します。また、共有スタック機能により複数のタスクでスタックを共有することができます。

(2) タスク付属同期機能

タスクの実行中断、再開、タスク付属イベントフラグなど、タスクの基本的な同期処理を行います。

(3) 同期・通信機能

タスク間の同期・通信処理をイベントフラグ、セマフォ、データキュー、およびメールボックスを用いて行います。

(4) 拡張同期・通信機能

タスク間の同期・通信処理をミューテックス、メッセージバッファを用いて行います。

(5) メモリプール管理機能

ユーザシステム内の未使用メモリをメモリプールとして管理します。タスクは必要に応じてメモリプールからメモリブロックを動的に獲得、返却します。
メモリプールには、固定長メモリプール、可変長メモリプールがあります。

(6) 時間管理機能

時間の管理を行います。また、タスク実行制御のため、時間監視を行います。

(7) システム状態管理機能

コンテキストやシステムの状態の変更、参照を行います。

(8) 割込み管理機能

外部割込みが発生すると割込みハンドラが起動され、割込み処理やタスクへの割込み発生の連絡が行われます。

(9) サービスコール管理機能

拡張サービスコールの定義、呼び出しなどを行います。

(10) システム構成管理機能

CPU 例外ハンドラの定義、カーネルのバージョン番号の参照などシステム構成管理を行います。

(11) DSP, FPU のサポート

マルチタスク環境で、DSP や FPU を使用することができます。つまり、DSP, FPU の各命令で使用する専用レジスタを、タスクごとに持つことができます。

2.3 処理の単位と優先順位

アプリケーションは、以下の処理単位によって実行制御されます。

(1) タスク

マルチタスクの制御対象となる単位です。

(2) タスク例外処理ルーチン

タスクに対してタスク例外処理が要求（ras_tex サービスコール）されると、タスク例外処理ルーチンが実行されます。

(3) 割込みハンドラ

割込みが発生したときに実行されます。

(4) CPU 例外ハンドラ

CPU 例外が発生したときに実行されます。

(5) タイムイベントハンドラ（周期ハンドラ、アラームハンドラ、オーバーランハンドラ）

指定した周期や時刻になったときに実行されます。

(6) 拡張サービスコール

拡張サービスコールは、リンクしないモジュールを呼び出すための機能です。拡張サービスコールを発行すると、対応する拡張サービスコールルーチンが呼び出されます。

また、各処理単位は以下の優先順位で処理されます。

(1) 割込みハンドラ、タイムイベントハンドラ、CPU 例外ハンドラ
(2) ディスパッチャ (カーネルの処理の一部)
(3) タスク

なお、ディスパッチャとは、実行するタスクを切り換えるカーネルの処理のことです。

割込みハンドラは、割込みレベルが高いほど優先順位が高くなります。

タイムイベントハンドラの優先順位は、タイマ割込みレベル(CFG_TIMINTLVL)と同じとなります。

CPU 例外ハンドラの優先順位は、CPU 例外が発生した処理の優先順位と、ディスパッチャの優先順位のいずれよりも高く、かつ CPU 例外が発生した処理よりも高い優先順位を持つ他のいずれの処理よりも低くなります。

タスクの間の優先順位は、タスクに付与された優先度に従います。

拡張サービスコールルーチンの優先順位は、拡張サービスコールを呼び出した処理の優先順位よりも高く、かつ拡張サービスコールルーチンを呼び出した処理よりも高い優先順位を持つ他のいずれの

RENESAS

処理よりも低くなります。

タスク例外処理ルーチンの優先順位は、当該タスクよりも高くかつ他の優先度の高いタスクよりも低くなります。

また以下のサービスコールを呼び出した場合は、一時的に上記に該当しない優先順位を作り出すことができます。

- dis_dsp を呼び出すと、その処理の優先順位は上記の(1)と(2)の間となります。この状態は、ena_dsp を呼び出すことによって元に戻ります。
- loc_cpu, iloc_cpu を呼び出すと、その処理の優先順位は割込みレベルが CFG_KNLMSKLVL である割込みハンドラと同じになります。この状態は、unl_cpu, iunl_cpu を呼び出すことによって元に戻ります。
- chg_ims によって割込みマスクレベルを 0 以外に変更している間の優先順位は、そのレベルの割込みハンドラと同じとなります。

2.4 システムの状態

システムの状態は、以下の 3 つの直行する状態によって区別されます。

(1)タスクコンテキスト/非タスクコンテキスト

(2)ディスパッチ禁止/許可状態

(3)CPU ロック/ロック解除状態

システムの挙動や呼び出し可能なサービスコールは、これらの状態によって決まります。

2.4.1 タスクコンテキストと非タスクコンテキスト

システムは、「タスクコンテキスト」か「非タスクコンテキスト」のいずれかのコンテキスト状態で実行します。タスクコンテキストと非タスクコンテキストの違いを、表 2.1に示します。

表2.1 タスクコンテキストと非タスクコンテキスト

	タスクコンテキスト	非タスクコンテキスト
呼び出し可能なサービスコール	タスクコンテキストから呼び出しできるもの	非タスクコンテキストから呼び出しできるもの
タスクスケジューリング	2.4.2, 2.4.3項参照	発生しない

非タスクコンテキストで実行される処理には、以下があります。

- 割込みハンドラ
- CPU 例外ハンドラ
- タイムイベントハンドラ（周期ハンドラ、アラームハンドラ、オーバーランハンドラ）
- chg_ims サービスコールで割込みマスクを 0 以外に変更して実行する部分

また、これらの状態から起動される拡張サービスコールルーチンも、非タスクコンテキストで実行されます。

2.4.2 ディスパッチ禁止/許可状態

システムは、ディスパッチ禁止状態か許可状態かのいずれかの状態をとります。ディスパッチ禁止状態では、タスクスケジューリングは行われません。また、待ち状態に遷移するサービスコールは呼び出しできません。

ディスパッチ禁止状態へは dis_dsp、許可状態へは ena_dsp サービスコールによって遷移します。また、sns_dsp サービスコールでディスパッチ禁止状態かどうかを知ることができます。

2.4.3 CPU ロック/ロック解除状態

システムは、CPU ロック状態か CPU ロック解除状態かのいずれかの状態をとります。CPU ロック状態では、割込みの受け付けが禁止され、タスクスケジューリングも行われません。ただし、コンフィギュレーションで指定するカーネル割込みマスクレベル(CFG_KNLMSKLVL)より高いレベルの割込みは受け付けられます。また、待ち状態に遷移するサービスコールは呼び出しできません。

CPU ロック状態へは loc_cpu, iloc_cpu、解除状態へは unl_cpu, iunl_cpu サービスコールによって遷移します。また、sns_loc サービスコールで CPU ロック状態かどうかを知ることができます。

また、CPU ロック状態では「3.2.5 システム状態とサービスコール」に記載のように発行可能なサービスコールが制限されます。

2.5 オブジェクト

タスクやセマフォなど、サービスコールによって操作する対象を「オブジェクト」と呼びます。オブジェクトは ID 番号やそのオブジェクト番号によって識別されます。

ほとんどのオブジェクトは、コンフィギュレーション時に使用する最大番号を指定することができます。

2.6 タスク

リアルタイム・マルチタスクシステムでは、アプリケーションを独立して並列に処理可能な単位に分割して作成します。この分割したプログラムをタスクと呼びます。

タスクは、タスク間で必要な連絡を、サービスコールを使用して行います。カーネルはサービスコールを介して、外部装置や計算機内部から非同期に発生した事象（イベント）を処理することができます。

表 2.2, 表 2.3にタスクを操作するサービスコールを示します。

表2.2　タスクを操作するサービスコール（タスク管理機能）

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_tsk, icre_tsk	タスクの生成 (ダイナミックスタック使用)	9	exd_tsk	自タスクの終了と削除
2	vscr_tsk, ivscr_tsk	タスクの生成 (スタティックスタック使用)	10	ter_tsk	他タスク強制終了
3	acre_tsk, iacre_tsk	タスクの生成 (ID 番号自動割付け)	11	chg_pri, ichg_pri	タスク優先度の変更
4	del_tsk	タスクの削除	12	get_pri, iget_pri	タスク優先度の参照
5	act_tsk, iact_tsk	タスクの起動	13	ref_tsk, iref_tsk	タスクの状態参照
6	can_act, ican_act	タスク起動要求のキャンセル	14	ref_tst, iref_tst	タスクの状態参照 (簡易版)
7	sta_tsk, ista_tsk	タスクの起動 (起動コード指定)	15	vchg_tmd	タスク実行モードの変更
8	ext_tsk	自タスク終了			

表2.3　タスクを操作するサービスコール（タスク付属同期機能）

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	slp_tsk	起床待ち	6	sus_tsk, isus_tsk	強制待ち状態への移行
2	tslp_tsk	起床待ち(タイムアウト有)	7	rsm_tsk, irsm_tsk	強制待ち状態からの再開
3	wup_tsk, iwup_tsk	タスクの起床	8	frsm_tsk, ifrsm_tsk	強制待ち状態からの強制再開
4	can_wup, ican_wup	タスク起床要求のキャンセル	9	dly_tsk	自タスクの遅延
5	rel_wai, irel_wai	待ち状態の強制解除			

2.6.1 タスクの状態と遷移

タスクは、外部や内部の事象により、図 2.1に示す 7 つの状態を遷移します。

(1) 未登録状態(NON-EXISTENT)

カーネルに登録されていない仮想的な状態です。

(2) 休止状態(DORMANT)

カーネルに登録された後、まだ起動されていない状態、または終了後の状態です。

(3) 実行可能状態(READY)

実行するための準備がすべて整った状態ですが、他の高い優先度のタスクが実行中のため実行はできない状態です。

(4) 実行状態(RUNNING)

現在、CPU 上で実行している状態です。
カーネルは実行可能状態のタスクの中で最も高い優先度のタスクを実行状態にします。

(5) 待ち状態(WAITING)

tslp_tsk サービスコールなどを呼び出した場合、条件が満たされない場合は待ち状態になります。待ち状態は、wup_tsk サービスコールなど、待ち状態になった要因に対応するサービスコールが呼び出されると解除され、実行可能状態に遷移します。

(6) 強制待ち状態(SUSPENDED)

タスクの実行が(sus_tsk サービスコール)により強制的に中断させられた状態です。

(7) 二重待ち状態(WAITING-SUSPENDED)

待ち状態と強制待ち状態が重なった状態です。

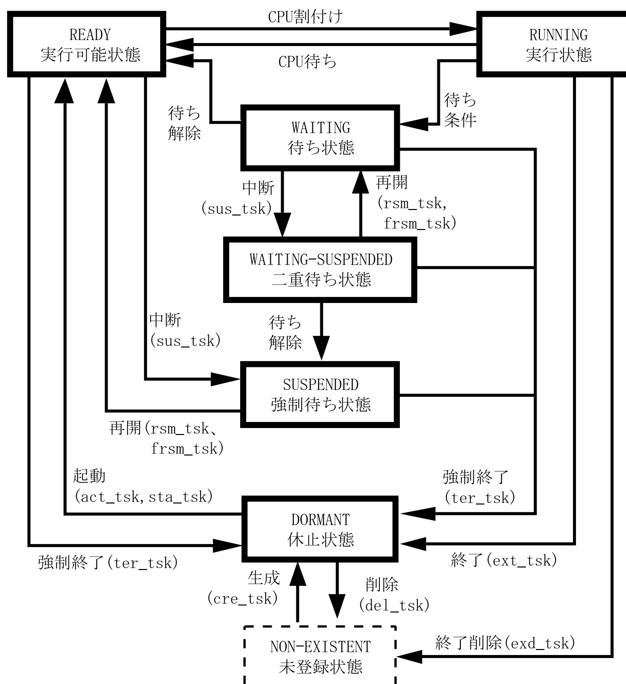


図2.1 タスクの状態遷移

RENESAS

2.6.2 タスクの生成

タスクを未登録状態から休止状態にすることを「タスクを生成する」といいます。

タスクの生成方法は、生成するタスクが使用するスタックの種類によって分かれます。スタックについては、「2.6.6 タスクのスタック」を参照してください。表 2.4に、タスクの生成方法を示します。

表2.4 タスクの生成方法

生成方法	使用するスタック		
	スタティック スタック	ダイナミック スタック	アプリケーションで確保した スタック
cre_tsk, acre_tsk サービスコール	×	○	○
vscr_tsk サービスコール	○	×	×
コンフィギュレータで生成	○	○	○

2.6.3 タスクの起動

休止状態のタスクを実行可能状態にすることを「タスクを起動する」といいます。タスクを起動するには、以下の方法があります。

- 目的のタスクに対して act_tsk, sta_tsk サービスコールを呼び出す
- タスクの生成時、タスク属性に TA_ACT を指定する

カーネルは、タスクを起動する際に以下の処理を行います。

- タスクのベース優先度および現在優先度の初期化
- 起床要求キューイング数のクリア
- 強制待ち要求ネスト数のクリア
- 保留例外要因のクリア
- タスク例外処理禁止状態への設定
- タスク付属イベントフラグのフラグパターンを 0 に設定

タスクには、以下のパラメータが渡ります。

(1) タスク生成時にTA_ACT属性を指定して起動した場合、およびact_tskサービスコールによって起動した場合
タスク生成時に指定した拡張情報(exinf)が渡ります。

(2) sta_tskサービスコールによって起動した場合
sta_tskサービスコールで指定したタスク起動コード(stacd)が渡ります。

2.6.4 タスクのスケジューリング

(1) スケジューリング

各タスクには、処理の優先順位を意味する「タスク優先度」が付与されます。タスク優先度は、値が小さいほど高い優先順位となり、1 が最高の優先順位です。

カーネルは、実行可能状態にあるタスクの中で最も高い優先度のタスクを実行状態にします。

複数のタスクに同じ優先度を与えることもできます。カーネルは、実行可能状態にあるタスクの中で最も高い優先度のタスクが複数存在する場合には、最も先に実行可能状態になったタスクを実行状態にします。カーネルはこれを実現するために、レディキューと呼ぶ実行可能状態のタスクの実行待ち行列を持っています。

なお、非タスクコンテキスト実行中は、非タスクコンテキストの処理が終了するまでタスクは実行されません。

(2) ラウンドロビンスケジューリング

標準のスケジューリングの他に、ラウンドロビンスケジューリングがあります。ラウンドロビンスケジューリングとは、一定時間ごとにレディキューを回転させ同一優先度を持つタスクの CPU 割り付け時間を平均化するスケジューリングです。

ラウンドロビンスケジューリングは、レディキューを操作する rot_rdq サービスコールを使用することで実現することができます。

- **rot_rdq サービスコールによるラウンドロビンスケジューリング**

  一定周期でrot_rdqサービスコールを呼び出すことで、一定時間毎に実行中のタスクと同じ優先度を持つ別のタスクに切り換えることができます。

(3) スケジューリングの制限

dis_dsp サービスコールによってディスパッチ禁止状態に遷移すると、タスクのスケジューリングが行われなくなります。タスクのスケジューリングは、ena_dsp サービスコールによってディスパッチ許可状態に遷移したときに行われます。

また、loc_cpu サービスコールによって CPU ロック状態に遷移すると、タスクのスケジューリングが行われなくなるだけでなく、割込みも禁止（カーネル管理外の割込みを除く）されます。タスクのスケジューリングと割込みの受付は、unl_cpu サービスコールによって CPU ロック解除状態に遷移したときに行われます。

2.6.5 タスクの終了と削除

タスクの終了とは、起動されたタスクの処理を終了し、休止状態または未登録状態になることをいいます。

- ext_tsk サービスコールを呼び出す
- exd_tsk サービスコールを呼び出す
- 目的のタスクに対して ter_tsk サービスコールを呼び出す

タスクは一度終了すると、次に起動がかけられたとき、または act_tsk サービスコールによる起動要求キューイング数が 0 以外のときに初期状態から実行されます。

そのタスクを以降使用しないとき、またはそのタスクの ID に別のタスクを割り付けたいときには、exd_tsk サービスコールを呼び出して未登録状態にしてください。未登録状態に移行した場合は、そのタスクの ID に別のタスクを割り付ける（その ID で別のタスクを生成する）ことができます。

タスクは終了前に、獲得していた資源を解放しなければなりません。ただし、ミューテックスに関しては、タスクの終了と削除処理でミューテックスのロックが解除されます。

カーネルは、タスクを終了する際とタスクを削除する際に以下の処理を行います。

＜タスクを終了する際に行う処理＞

- 自タスクがロックしているミューテックスのロック解除
- 上限プロセッサ時間の設定解除

＜タスクを削除する際に行う処理＞

- タスクのスタック領域の開放

2.6.6 タスクのスタック

タスクスタックの確保方法として以下の 3 種類があります。

- スタティックスタック

  コンフィギュレータで、タスク毎に使用するスタックを静的に確保します。スタティックスタックを使用するタスクは、複数のタスクで1つのスタックを共有する（共有スタック機能）こともできます。

- ダイナミックスタック

  カーネルが、コンフィギュレータで確保したダイナミックスタック領域(CFG_TSKSTKSZ)から、指定したサイズを割り付けるスタックです。ダイナミックスタックを使用する場合、共有スタック機能は使えません。

- アプリケーションでスタック領域を確保

  アプリケーションでスタック領域を確保し、タスク生成時にそのアドレスを指定します。共有スタック機能は使えません。

コンフィギュレータでは、スタティックスタックを使用する最大のタスク ID(CFG_STSTKID)と、全体の最大タスク ID(CFG_MAXTSKID)を指定します。1 から CFG_STSTKID のタスク ID のタスクはスタティックスタックを使用します。一方、CFG_STSTKID+1 から CFG_MAXTSKID のタスク ID のタスクはダイナミックスタックまたはアプリケーションで確保したスタックを使用します。

CFG_STSTKID に 0 を指定するとスタティックスタックを全く使わないという意味に、CFG_STSTKID と CFG_MAXTSKID に同じ値を指定すると全てのタスクがスタティックスタックを使用するという意味になります。

2.6.7　共有スタック機能

共有スタック機能は、複数のタスクでスタックを共有する機能で、スタック全体の使用サイズを削減できます。共有スタック機能は、μITRON4.0 仕様の範囲外です。

スタックを共有するかは、スタティックスタックの確保とともにコンフィギュレータで定義します。スタックを共有するタスク群では、同時には 1 タスクのみがスタックを占有して実行する権利が与えられます。同一スタックを使用するように定義されているタスクが複数起動された場合は、先に起動されたタスクがスタックを占有し、他のタスクは共有スタック待ち状態となります。共有スタック待ちのタスクは、優先度に関係なく FIFO(First-In First-Out)で管理され、起動要求が行われた順番につながれます。

共有スタックは占有しているタスクが休止状態となったときに解放されます。この時、共有スタック待ちのタスクが存在すれば、待ちの先頭タスクがスタックを占有して実行可能状態となります。

図 2.2に共有スタック機能を使用するタスクの状態遷移を示します。

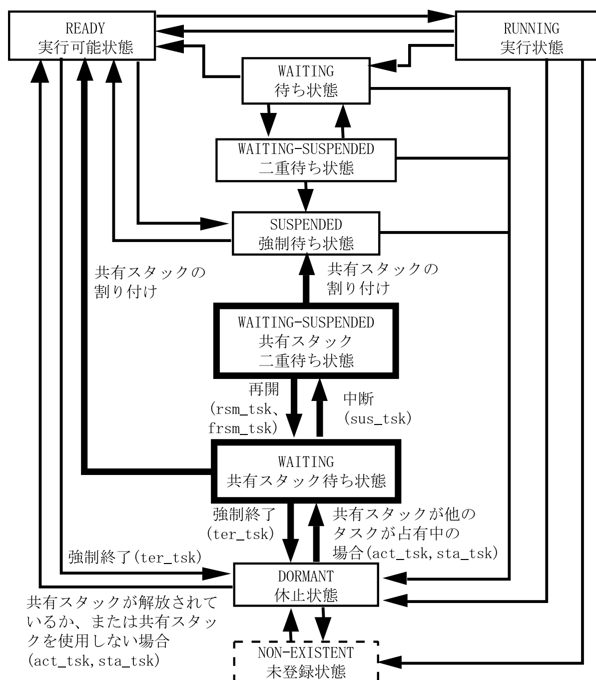


図2.2　共有スタック機能使用時のタスク状態遷移

2.6.8　タスク実行モード

タスクは、確保した資源を解放する前に、他タスクからの強制終了要求(ter_tsk サービスコール)によって意図しないタイミングで休止状態になる可能性があります。また、タスクは sus_tsk サービスコールによって予期しないタイミングで実行が中断する可能性があります。

vchg_tmd サービスコールを用いることで、ter_tsk, sus_tsk サービスコールによる要求をマスクすることができます。

2.6.9 排他制御

タスク実行中には、他のプログラムとは排他的に実行させる必要が生じる場合があります。例えば、タスク A と割込みハンドラ B が同じグローバル変数を参照･変更する場合には、参照・変更の一連の手続きを互いに排他的に実行する必要があります。排他実行の対象は、この例の割込みハンドラだけでなく、特定の他タスクや不特定の他タスクの場合もあります。

表 2.5に排他制御方法を示しますので、参考にしてください。

表2.5　排他制御方法

項番	排他制御方法	禁止される割込み	タスクスケジューリング
1	loc_cpu サービスコールで CPU ロック状態に遷移する	カーネル割込みマスクレベル (CFG_KNLMSKLVL)まで	発生しない
2	chg_ims サービスコールで割込みをマスク(タスクコンテキストから呼び出した場合は非タスクコンテキストに遷移する)	指定した割込みレベル以下	発生しない
3	dis_dsp サービスコールでディスパッチ禁止状態に遷移する	一切禁止されない	発生しない
4	セマフォによる排他制御	一切禁止されない	スケジューリングは発生するが、同じセマフォを使用するタスク群の中では、同時に実行可能状態となるタスク数はセマフォの初期カウント値までに制限される。
5	ミューテックスによる排他制御	一切禁止されない	スケジューリングは発生するが、同時に同じミューテックスを使用するタスクは同時には実行状態にはならない。さらに、同じミューテックスを使用するタスクの間では、優先度の逆転は発生しない。

2.6.10　タスク付属イベントフラグ

タスク付属イベントフラグはタスクに付与されたビットパターンで、タスク付属イベントフラグの指定したいずれかのビットのセット（事象の発生）を待つことができます。

表 2.6にタスク付属イベントフラグを操作するサービスコールを示します。

表2.6　タスク付属イベントフラグを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	vset_tfl, ivset_tfl	**タスク付属イベントフラグのセット**	4	vpol_tfl	**タスク付属イベントフラグ待ち（ポーリング）**
2	vclr_tfl, ivclr_tfl	**タスク付属イベントフラグのクリア**	5	vtwai_tfl	**タスク付属イベントフラグ待ち（タイムアウト有）**
3	vwai_tfl	**タスク付属イベントフラグ待ち**			

図 2.3にタスク付属イベントフラグの動作例を示します。

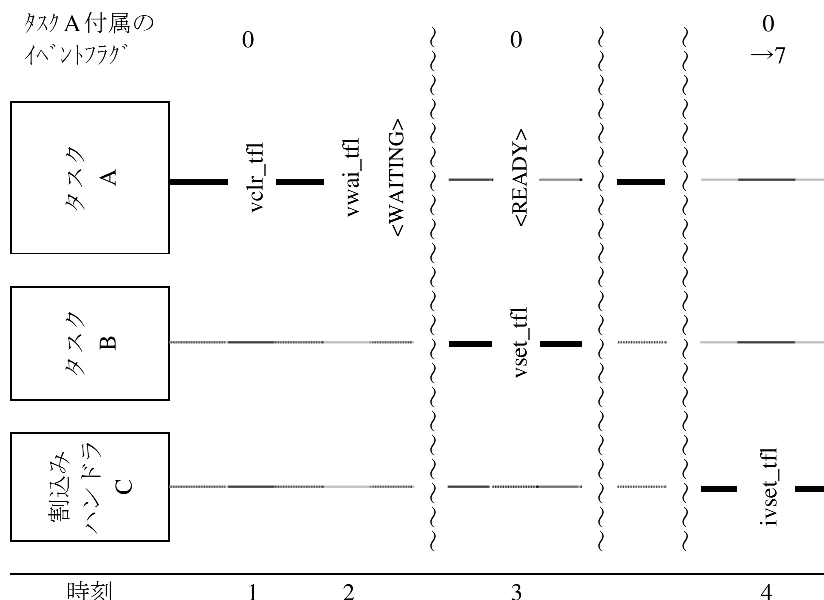


図2.3　タスク付属イベントフラグの動作例

図の解説

太実線は、実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAがvclr_tflを発行して自分のイベントフラグを全ビットクリアします。
2. タスクAは、イベントを待つためにvwai_tfl(待ちパターン=H'ffffffff)を発行します。
3. タスクBがタスクAに対してvset_tfl(セットパターン=1)を発行します。このセットパターンはタスクAの待ちパターンに含まれるので、タスクAの待ち状態は解除されます。また、待ち解除と同時にタスクA付属のイベントフラグは0クリアされます。
4. 割込みハンドラCがivset_tfl(セットパターン=7)でタスクAのイベントフラグをセットしますが、タスクAはイベント待ちではないので、単にタスクAのイベントフラグにvset_tflで指定したパターンがORされます。

RENESAS

2.7 タスク例外処理

タスク例外処理機能は、タスクに発生した例外事象の処理を、タスク本体の処理とは非同期に行うための機能で、一般に「シグナル」と呼ばれている機能に類似した機能です。

表 2.7にタスク例外処理を操作するサービスコールを示します。

表2.7 タスク例外処理を操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	def_tex, idef_tex	**タスク例外処理ルーチンの定義**	4	ena_tex	**タスク例外処理の許可**
2	ras_tex, iras_tex	**タスク例外処理の要求**	5	sns_tex	**タスク例外処理禁止状態の参照**
3	dis_tex	**タスク例外処理の禁止**	6	ref_tex, iref_tex	**タスク例外処理の状態参照**

タスク例外処理ルーチンは、コンフィギュレータで定義することもできます。

図 2.4にタスク例外処理の動作例を示します。

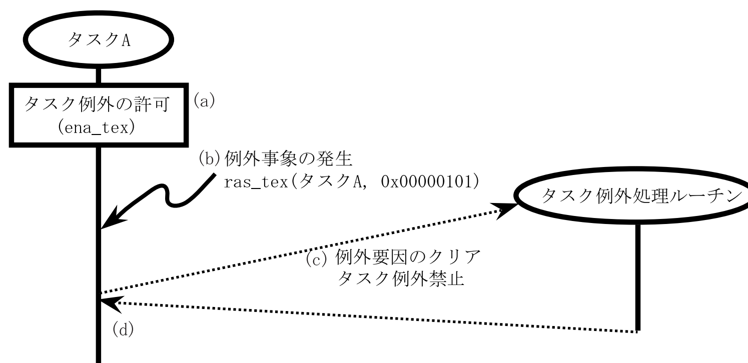


図2.4 タスク例外処理の動作例

図の解説（アルファベットは、その順に実行することを意味します）

(a) タスクAが、タスク例外を許可します。

(b) タスクA実行中にras_texサービスコールによってタスクAに例外事象（例外要因(00000101)）が要求されました。

(c) タスクAにスケジュールされたときに、タスクAの本体を実行する前にタスク例外処理ルーチンが起動されます。このとき、タスク例外禁止状態となり、タスク例外要因もクリアされます。

(d) タスク例外処理ルーチンからリターンすると、タスク例外処理ルーチンを起動する前に実行していたタスク本体の処理を継続します。

2.8 セマフォ

タスク実行のために必要となる各種要素を資源と呼びます。資源には、各種 I/O や共有するメモリが考えられます。

セマフォは、このような資源の有無や数をカウンタで表現することで排他制御や同期機能を提供するオブジェクトです。タスクは、資源を排他的にアクセスしたい区間を、wai_sem～sig_sem サービスコールで区切るように作成します。通常、セマフォのカウンタは使用可能な資源の数に対応させて使用します。

表 2.8にセマフォを操作するサービスコールを示します。

表2.8　セマフォを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_sem, icre_sem	セマフォの生成	5	wai_sem	セマフォ資源の獲得
2	acre_sem, iacre_sem	セマフォの生成 (ID 番号自動割付け)	6	pol_sem, ipol_sem	セマフォ資源の獲得 (ポーリング)
3	del_sem	セマフォ削除	7	twai_sem	セマフォ資源の獲得 (タイムアウト有)
4	sig_sem, isig_sem	セマフォ資源返却	8	ref_sem, iref_sem	セマフォの状態参照

セマフォは、コンフィギュレータで生成することもできます。

図 2.5にセマフォの動作例を示します。

RENESAS

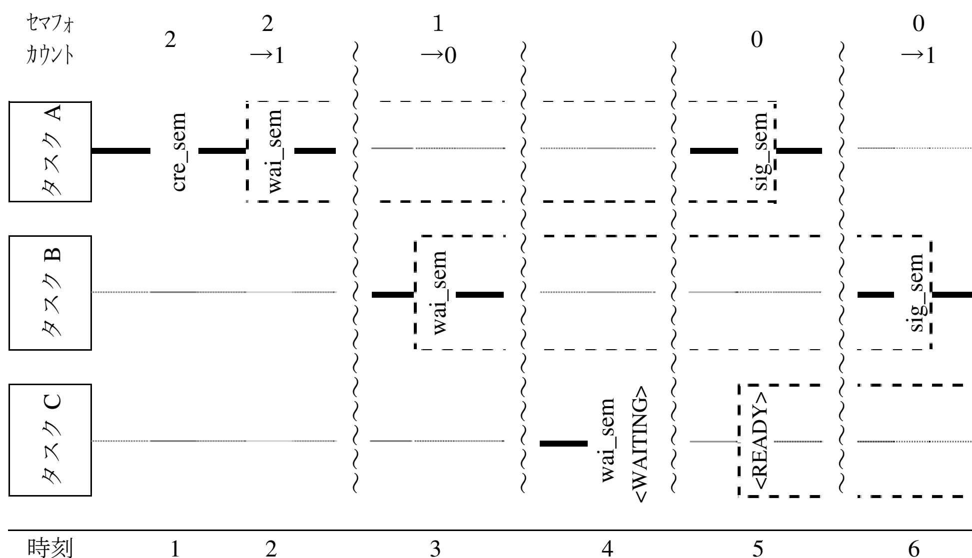


図2.5　セマフォの動作例

図の解説

太実線は実際に実行した部分、点線の枠はそれぞれのタスクが資源を排他的にアクセス可能な区間を示しています。以下、各時刻での説明を補足します。

1. タスクAがcre_semでカウント初期値2のセマフォを生成します。
2. タスクAがwai_semでセマフォの獲得要求を行うと、セマフォカウントが1減ります。タスクAは実行を継続します。
3. 同様にタスクBがwai_semを発行します。
4. タスクCがwai_semを発行しますが、セマフォカウントは0のためタスクCはセマフォを獲得できずに待ち状態になります。
5. タスクAがsig_semを発行してセマフォを返却すると、セマフォ獲得を待っていたタスクCにセマフォが割付けられ、タスクCの待ち状態は解除されます。
6. タスクBがsig_semを発行します。セマフォ獲得を待っているタスクは存在しないので、セマフォカウントが1増えます。

2.9 イベントフラグ

イベントフラグは、事象に対応したビットの集合で、1 つの事象が 1 ビットで表わされます。タスクは、イベントフラグの指定したビットのセット（事象の発生）を待つことができます。1 つのイベントフラグには、複数のタスクが事象の発生を待つことができます。

表 2.9にイベントフラグを操作するサービスコールを示します。

表2.9 イベントフラグを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_flg, icre_flg	イベントフラグの生成	6	wai_flg	イベントフラグ待ち
2	acre_flg, iacre_flg	イベントフラグの生成 (ID 番号自動割付け)	7	pol_flg, ipol_flg	イベントフラグの待ち(ポーリング)
3	del_flg	イベントフラグの削除	8	twai_flg	イベントフラグ待ち(タイムアウト有)
4	set_flg, iset_flg	イベントフラグのセット	9	ref_flg, iref_flg	イベントフラグの状態参照
5	clr_flg, iclr_flg	イベントフラグのクリア			

イベントフラグは、コンフィギュレータで生成することもできます。

図 2.6にイベントフラグの動作例を示します。

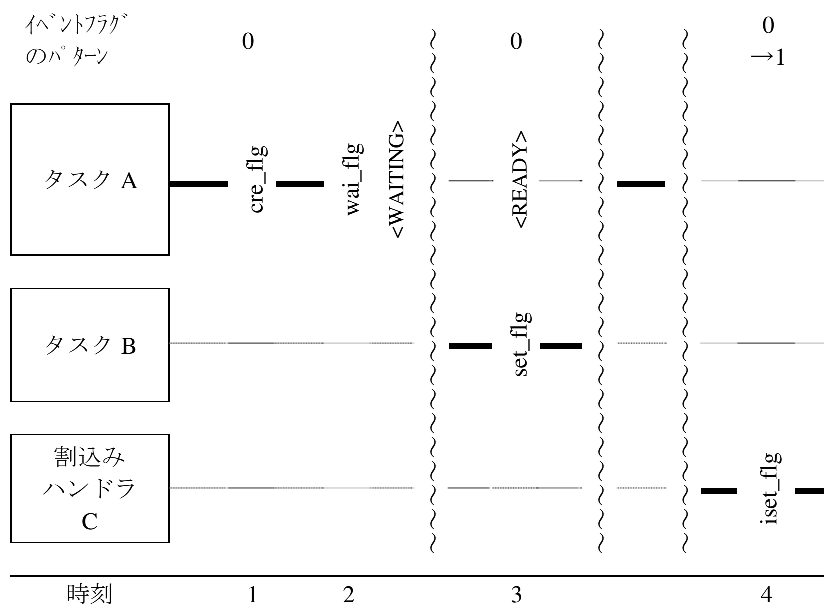


図2.6　イベントフラグの動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAがcre_flgでイベントフラグを生成します。TA_CLR(待ち解除時にイベントフラグを0クリアする)属性を指定し、初期パターンを0とします。
2. タスクAは、イベントを待つためにwai_flg(待ちパターン=3, AND待ち)を発行します。
3. タスクBがset_flg(セットパターン=7)を発行します。タスクAが待っていた全ビットがセットされたため、タスクAの待ち状態は解除されます。また、TA_CLR属性が指定されているので、待ち解除と同時にイベントフラグが0クリアされます。
4. 割込みハンドラCがiset_flg(セットパターン=1)でイベントフラグをセットしますが、イベント待ちのタスクは存在しないので、単にイベントフラグにiset_flgで指定したパターンがORされます。

2.10　データキュー

データキューを使用することにより、タスク間で 1 ワード(4 バイト)データの受渡しを行うことができます。データキューをでは 1 ワードデータそのものがコピーされるため、高速にデータの受渡しができます。データとしてポインタを指定することもできます。

データキューは、コンフィギュレータで指定したデータキュー用領域(CFG_DTQSZ)から割り付けられます。

表 2.10にデータキューを操作するサービスコールを示します。

表2.10　データキューを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_dtq, icre_dtq	データキューの生成	7	fsnd_dtq, ifsnd_dtq	データキューへの強制送信
2	acre_dtq, iacre_dtq	データキューの生成 (ID 番号自動割付け)	8	rcv_dtq	データキューからの受信
3	del_dtq	データキューの削除	9	prcv_dtq	データキューからの受信(ポーリング)
4	snd_dtq	データキューへの送信	10	trcv_dtq	データキューからの受信(タイムアウト有)
5	psnd_dtq, ipsnd_dtq	データキューへの送信 (ポーリング)	11	ref_dtq, iref_dtq	データキューの状態参照
6	tsnd_dtq	データキューへの送信 (タイムアウト有)			

データキューは、コンフィギュレータで生成することもできます。

図 2.7にデータキューの動作例を示します。

RENESAS

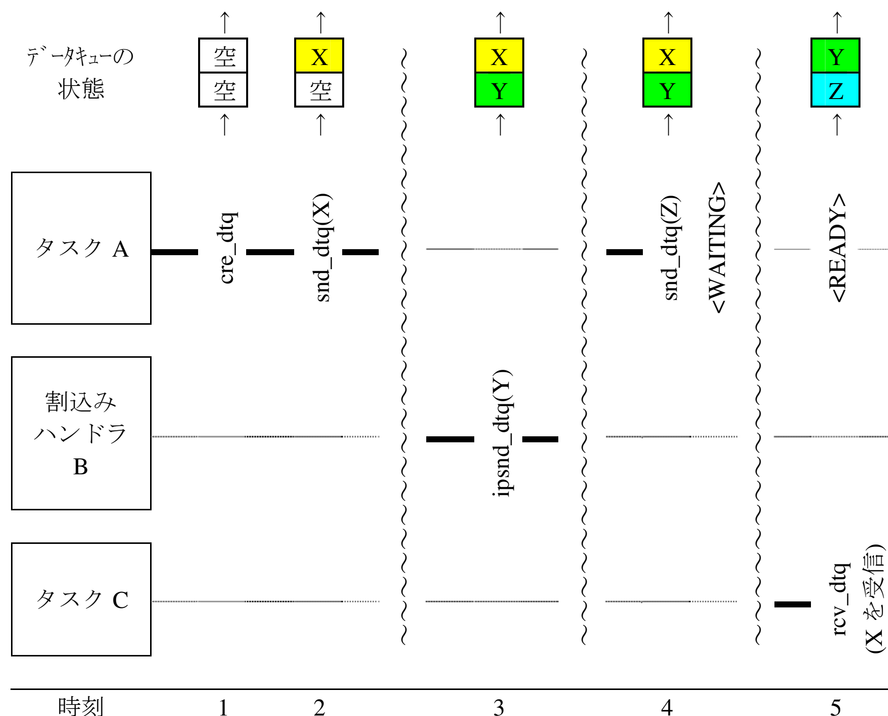


図2.7　データキューの動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAがcre_dtqで容量が2ワードのデータキューを生成します。
2. タスクAがsnd_dtqでデータXを送信します。データXはデータキューにコピーされ、タスクAは実行を継続します。
3. 割込みハンドラがipsnd_dtqでデータYを送信します。
4. タスクAがデータZを送信しようとしますが、データキューに空きが無いため待ち状態になります。
5. タスクCがrcv_dtqでデータキューからデータを受信します。タスクCには、最も先に送信されたデータXがコピーされます。同時にデータキューに空きができたので、タスクAが送信しようとしていたデータZがデータキューにコピーされ、タスクAの待ち状態は解除されます。

2.11　メールボックス

メールボックスを使用することにより、タスク間でメッセージと呼ばれるデータの受け渡しを行うことができます。メールボックスを用いた通信は、メッセージの先頭アドレスの受け渡しによって実現されているため、メッセージサイズに依存せずに高速に行われます。

表 2.11にメールボックスを操作するサービスコールを示します。

表2.11　メールボックスを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_mbx, icre_mbx	**メールボックスの生成**	5	rcv_mbx	**メールボックスからの受信**
2	acre_mbx, iacre_mbx	**メールボックスの生成** (ID **番号自動割付け**)	6	prcv_mbx, iprcv_mbx	**メールボックスからの受信** (**ポーリング**)
3	del_mbx	**メールボックスの削除**	7	trcv_mbx	**メールボックスからの受信** (**タイムアウト有**)
4	snd_mbx, isnd_mbx	**メールボックスへの送信**	8	ref_mbx, iref_mbx	**メールボックスの状態参照**

メールボックスは、コンフィギュレータで生成することもできます。

図 2.8にメールボックスの動作例を示します。

RENESAS

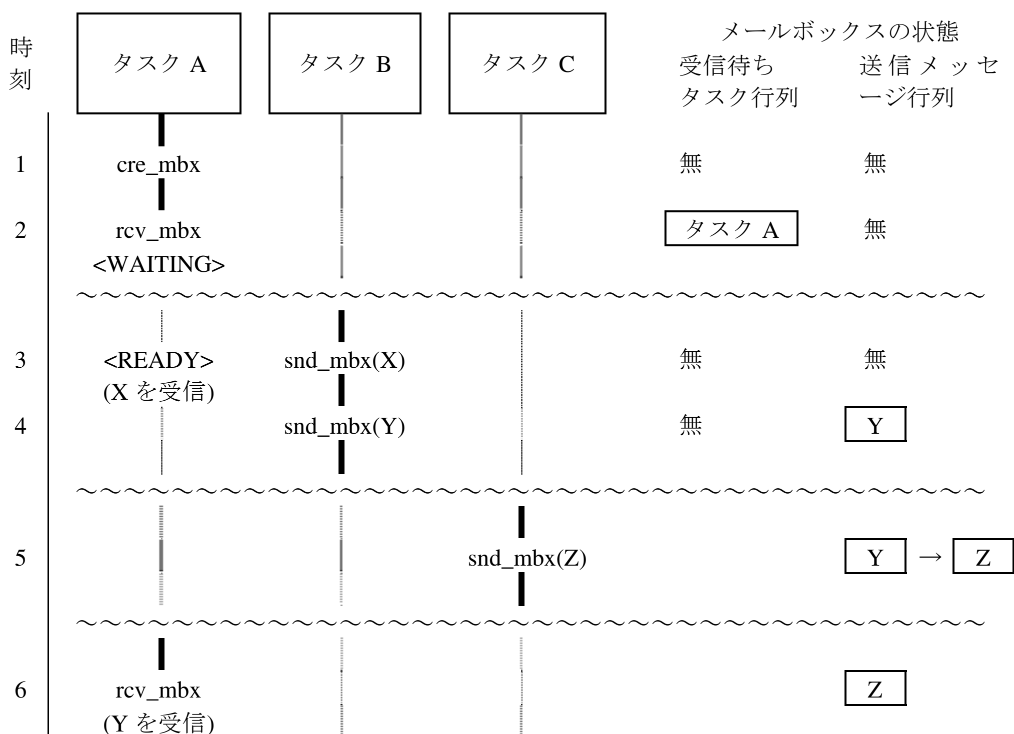


図2.8 メールボックスの動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAがcre_mbxでメールボックスを生成します。属性として、TA_TFIFO(受信待ちタスク行列はFIFO順)とTA_MFIFO(送信メッセージ行列はFIFO順)を指定します。
2. タスクAがrcv_mbxでメッセージを受信しようとしますが、メールボックスにメッセージが無いので待ち状態になります。
3. タスクBがsnd_mbxでメッセージXを送信すると、受信を待っていたタスクAの待ち状態が解除され、タスクAにメッセージXのアドレスが渡されます。
4. タスクBがsnd_mbxでメッセージYを送信します。受信待ちタスクは存在しないので、メッセージXはメッセージ行列につながれます。
5. タスクCがsnd_mbxでメッセージZを送信します。TA_MFIFO属性に従い、メールボックスにはY→Zの順にメッセージがつながれます。
6. タスクAがrcv_mbxを発行します。タスクAには、メッセージ行列先頭のメッセージYのアドレスが渡されます。

2.12　ミューテックス

ミューテックスは排他制御を行うためのオブジェクトで、優先度逆転現象を回避するための優先度上限プロトコルをサポートしています。このプロトコルでは、ミューテックスを獲得しているタスクは、タスク優先度がそのミューテックスに設定された上限優先度まで引き上げられて実行します。

表 2.12にミューテックスを操作するサービスコールを示します。

表2.12　ミューテックスを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_mtx	ミューテックスの生成	5	ploc_mtx	ミューテックスのロック(ポーリング)
2	acre_mtx	ミューテックスの生成(ID 番号自動割付け)	6	tloc_mtx	ミューテックスのロック(タイムアウト有)
3	del_mtx	ミューテックスの削除	7	unl_mtx	ミューテックスのロック解除
4	loc_mtx	ミューテックスのロック	8	ref_mtx	ミューテックスの状態参照

ミューテックスは、コンフィギュレータで生成することもできます。

図 2.9にミューテックスの動作例を示します。

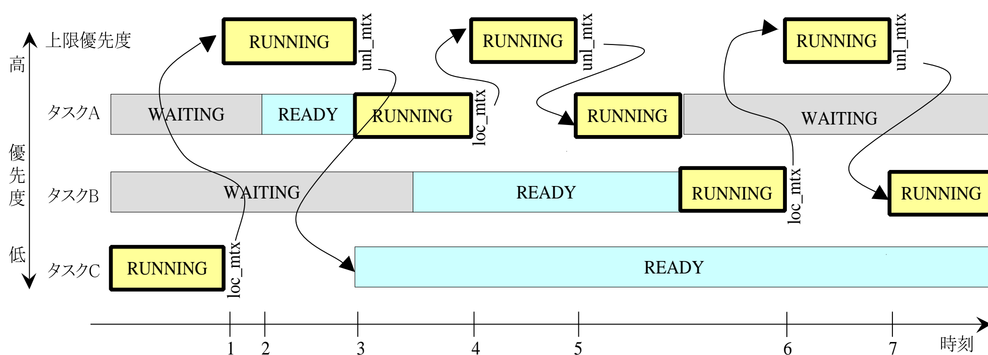


図2.9　ミューテックスの動作例

図の解説

この図では、優先度がタスク A, B, C の順に高くなっています。また、ミューテックスの上限優先度は、ミューテックスをロックするタスクの優先度よりも高く設定しなければなりません。以下、各時刻での説明を補足します。

1. タスクCがloc_mtxでミューテックスをロックすると、優先度がミューテックスの上限優先度に引き上げられます。
2. タスクCが上限優先度で実行中にタスクAがREADY状態になりました。タスクAの優先度は本来はタスクCよりも高いですが、タスクCはタスクAよりも高い上限優先度で実行しているため、タスクAはRUNNING状態にはなりません。すなわちタスクCは、ミューテックスをロックしている間は、本来の優先度がより高いタスクAが実行可能になっても、タスクAに邪魔されずに処理を継続することができます。
3. タスクCがunl_mtxでミューテックスのロックを解除すると、タスクCは元の優先度に戻ります。この結果、優先度の高いタスクAがRUNNING状態になります。
4. タスクAがloc_mtxを発行すると、タスクAの優先度は上限優先度に引き上げられます。
5. タスクAがunl_mtxを発行すると、タスクAの優先度は元に戻ります。
6. タスクBがloc_mtxを発行すると、タスクBの優先度は上限優先度に引き上げられます。
7. タスクBがunl_mtxを発行すると、タスクBの優先度は元に戻ります。

2.13　メッセージバッファ

メッセージバッファを使用することにより、タスク間でメッセージと呼ばれるデータの受け渡しを行うことができます。メッセージバッファを用いた通信はメッセージそのもののコピーによって実現されているため、メッセージ送信後は相手が受信したかどうかに関わらず、直ちにメッセージ領域を再利用することができます。

メッセージバッファは、コンフィギュレータで指定したメッセージバッファ用領域(CFG_MBFSZ)から割り付けられます。

表 2.13にメッセージバッファを操作するサービスコールを示します。

表2.13　メッセージバッファを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_mbf, icre_mbf	メッセージバッファの生成	6	tsnd_mbf	メッセージバッファへ送信 (タイムアウト有)
2	acre_mbf, iacre_mbf	メッセージバッファの生成 (ID 番号の自動割付け)	7	rcv_mbf	メッセージバッファから受信
3	del_mbf	メッセージバッファ削除	8	prcv_mbf	メッセージバッファから受信 (ポーリング)
4	snd_mbf	メッセージバッファへ送信	9	trcv_mbf	メッセージバッファから受信 (タイムアウト有)
5	psnd_mbf, ipsnd_mbf	メッセージバッファへ送信 (ポーリング)	10	ref_mbf, iref_mbf	メッセージバッファ状態参照

メッセージバッファは、コンフィギュレータで生成することもできます。

図 2.10にメッセージバッファの動作例を示します。

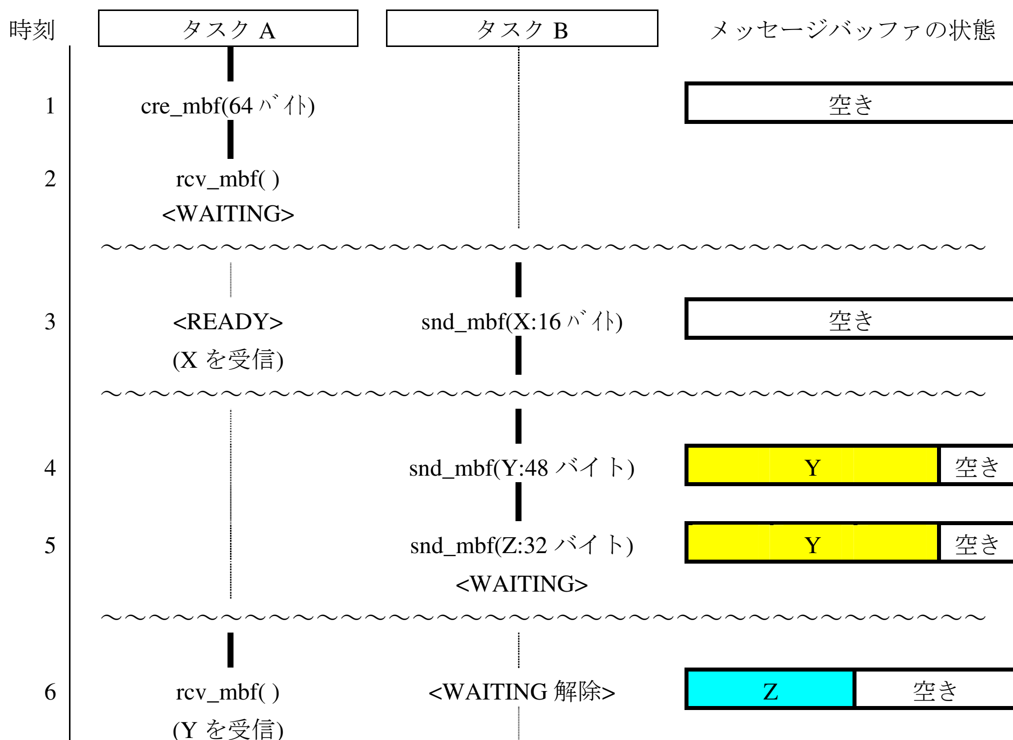


図2.10　メッセージバッファ動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAがcre_mbfでバッファサイズが64バイトで、扱えるメッセージの最大サイズが48バイトのメッセージバッファを生成します。
2. タスクAはメッセージを受信するために、48バイトのメモリを用意してrcv_mbfを発行します。この時点ではメッセージバッファにメッセージが無いので、タスクAはメッセージ受信待ち状態になります。
3. タスクBがsnd_mbfで16バイトのメッセージを送信すると、受信を待っていたタスクAの待ち状態が解除され、タスクAが用意したメモリにXがコピーされます。また、タスクAは受信したメッセージサイズ16をリターンパラメータとして受け取ります。
4. タスクBがsnd_mbfで48バイトのメッセージYを送信します。受信待ちタスクは存在しないので、メッセージYはメッセージバッファにコピーされます。なお、メッセージをメッセージバッファにコピーする際、カーネルはメッセージバッファ領域を4バイト消費しますが、図中ではこの表記はしていません。
5. タスクBがsnd_mbfで32バイトのメッセージZを送信しようとしますが、メッセージバッファの空きが足りないのでタスクBは送信待ち状態になります。
6. タスクAが48バイトのメモリを用意してrcv_mbfを発行すると、メッセージバッファに蓄えられていたメッセージYがタスクAが用意したメモリにコピーされます。また、タスクAは受信したメッセージサイズ48をリターンパラメータとして受け取ります。また、メッセージYをタスクAに渡したことによってメッセージバッファに空きが生じたので、タスクBの送信待ち状態は解除され、メッセージZがメッセージバッファにコピーされます。

2.14　固定長メモリプール

タスクは、固定長メモリプールからメモリプール毎に決められた固定長のメモリブロックを獲得して使用することができます。メモリブロックのサイズは、メモリプールの生成時に指定します。

各固定長メモリプールは、コンフィギュレータで指定した固定長メモリプール用領域(CFG_MPFSZ)から割り付けられます。

また、アプリケーション側で確保した領域を固定長メモリプールとして使用することもできます。この場合、生成時にメモリプール領域のアドレスを指定します。

固定長メモリプールの管理方式(CFG_MPFMANAGE)として、コンフィギュレータで以下のいずれかを選択できます。

(1)　従来方式(CFG_MPFMANAGE非選択時)
固定長メモリプール内のメモリブロックに隣接して、メモリブロックを管理するためのカーネル管理テーブルを配置する管理方式です。従来バージョン(HI7000/4 V1.0.05以前、HI7700/4 V.1.03 Release 02以前、HI7750/4 V1.1.00以前)互換の管理方式です。

(2)　拡張方式(CFG_MPFMANAGE選択時)
メモリブロックを管理するためのカーネル管理テーブルを固定長メモリプール領域とは別の場所に配置する方式です。この方式では、固定長メモリプール生成時に、その管理テーブル領域のアドレスを指定します。管理テーブル領域は、アプリケーションで確保する必要があります。

表 2.14に固定長メモリプールを操作するサービスコールを示します。

表2.14　固定長メモリプールを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_mpf, icre_mpf	**固定長メモリプールの生成**	5	pget_mpf, ipget_mpf	**固定長メモリブロックの獲得(ポーリング)**
2	acre_mpf, iacre_mpf	**固定長メモリプールの生成** (ID 番号自動割付け)	6	tget_mpf	**固定長メモリブロックの獲得(タイムアウト有)**
3	del_mpf	**固定長メモリプールの削除**	7	rel_mpf, irel_mpf	**固定長メモリブロックの返却**
4	get_mpf	**固定長メモリブロックの獲得**	8	ref_mpf, iref_mpf	**固定長メモリプールの状態参照**

固定長メモリプールは、コンフィギュレータで生成することもできます。

図 2.11に固定長メモリプールの動作例を示します。

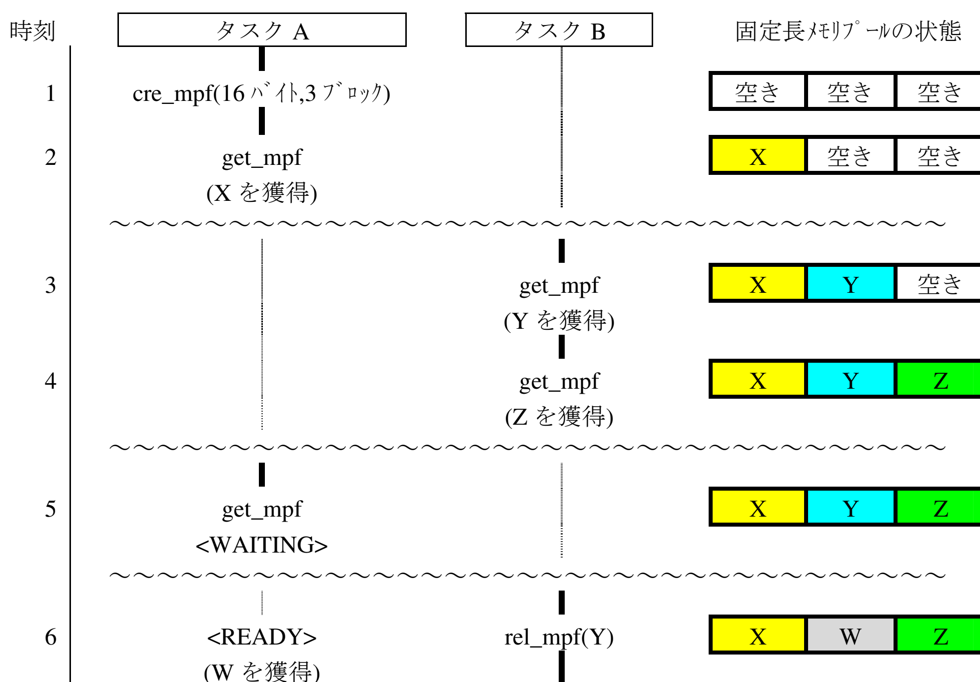


図2.11　固定長メモリプールの動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAが、cre_mpfで16バイトのブロックが3つの固定長メモリプールを生成します。
2. タスクAがget_mpfでブロックXを獲得します。
3. タスクBがget_mpfでブロックYを獲得します。
4. タスクBがget_mpfでブロックZを獲得します。
5. タスクAがget_mpfでブロックを獲得しようとしますが、空きブロックが無いのでタスクBはブロック獲得待ち状態になります。
6. タスクBがrel_mpfでブロックYを返却します。これにより空きブロックが生じたので、タスクAの待ち状態は解除され、タスクAにブロックYが割り当てられます。

2.15　可変長メモリプール

2.15.1　概要

タスクは、可変長メモリプールから任意のサイズのメモリブロックを獲得して使用することができます。

可変長メモリプールは固定長メモリプールよりも柔軟性に富む反面、獲得・返却処理のオーバヘッドが大きいという欠点を持っています。また、可変長メモリプールでは、空き領域が分断する可能性があります。つまり、空き領域の合計が十分にあっても連続した空き領域が存在しないために、メモリブロックを獲得できない場合があります。

各可変長メモリプールは、コンフィギュレータで指定した可変長メモリプール用領域(CFG_MPLSZ)から割り付けられます。

また、アプリケーション側で確保した領域を可変長メモリプールとして使用することもできます。この場合、生成時にメモリプール領域のアドレスを指定します。

可変長メモリプールの管理方式(CFG_NEWMPL)として、コンフィギュレータで以下のいずれかを選択できます。

(1)　従来方式(CFG_NEWMPL非選択時)
以下のバージョン互換の管理方式です。
・HI7000/4 V.2.00 Release 02以前
・HI7700/4 V.1.03 Release 02以前
・HI7750/4 V1.1.00以前

(2)　新方式(CFG_NEWMPL選択時)
従来方式に対して、以下が改善されます。
・特に多くのメモリブロックを扱うメモリプールで、獲得/返却が高速化されます。
・空き領域の断片化を軽減するVTA_UNFRAGMENT属性を使用できます。
なお、CFG_NEWMPL選択時は、従来バージョンに対してT_CMPL構造体に新メンバが追加になります。詳細は、「3.14.1　可変長メモリプールの生成(cre_mpl, icre_mpl)」を参照してください。

表 2.15に可変長メモリプールを操作するサービスコールを示します。

表2.15　可変長メモリプールを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_mpl, icre_mpl	可変長メモリプールの生成	5	pget_mpl, ipget_mpl	可変長メモリブロックの獲得(ポーリング)
2	acre_mpl, iacre_mpl	可変長メモリプールの生成 (ID 番号自動割付け)	6	tget_mpl	可変長メモリブロックの 獲得(タイムアウト有)
3	del_mpl	可変長メモリプールの削除	7	rel_mpl, irel_mpl	可変長メモリブロックの 返却
4	get_mpl	可変長メモリブロックの獲得	8	ref_mpl, iref_mpl	可変長メモリプールの 状態参照

可変長メモリプールは、コンフィギュレータで生成することもできます。
図 2.12に可変長メモリプールの動作例を示します。

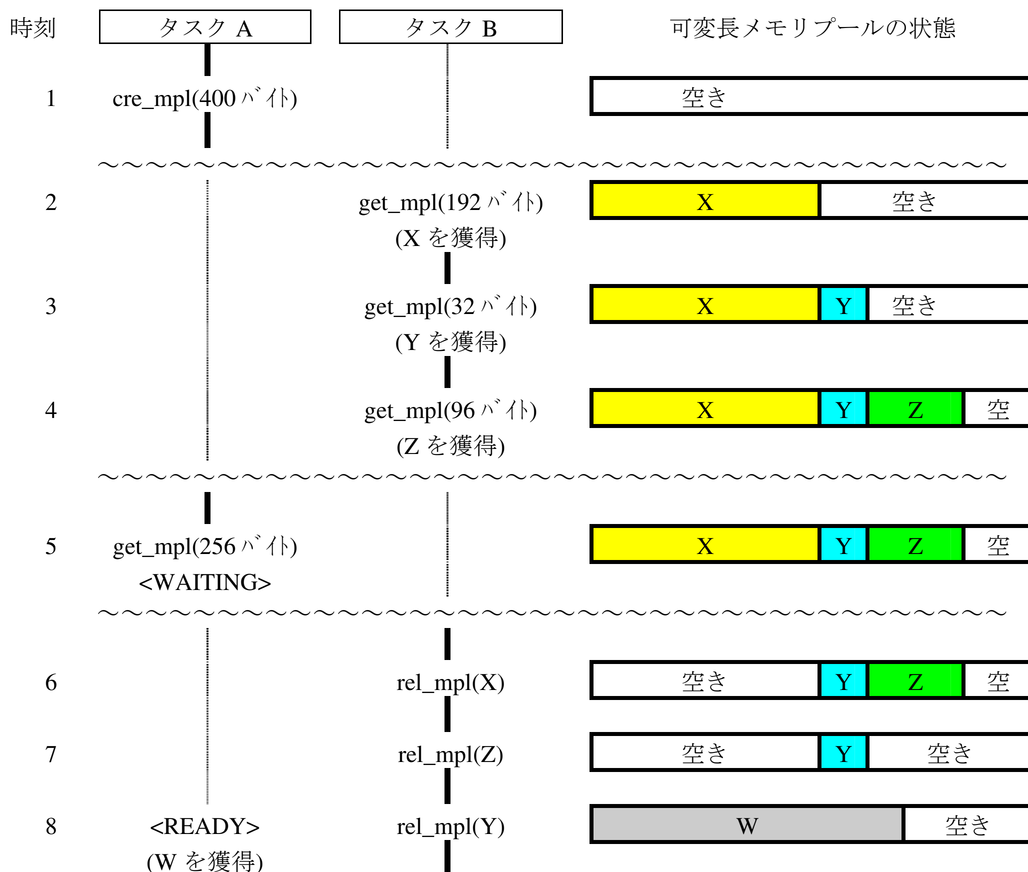


図2.12　可変長メモリプールの動作例

図の解説

太実線は実際に実行したことを示しています。以下、各時刻での説明を補足します。

1. タスクAが、cre_mplで400バイトの可変長メモリプールを生成します。
2. タスクBがget_mplで192バイトのブロックXを獲得します。なお、カーネルはブロックを割付ける際にメモリプール領域を16バイト消費しますが、図中ではこの表記はしていません。
3. タスクBがget_mplで32バイトのブロックYを獲得します。
4. タスクBがget_mplで96バイトのブロックZを獲得します。
5. タスクAがget_mplで256バイトのブロックを獲得しようとしますが、空きが無いのでタスクBはブロック獲得待ち状態になります。
6. タスクBがrel_mplで192バイトのブロックXを返却します。まだタスクAが要求する256バイトの連続した空き領域が無いので、タスクAは待ち状態のままです。
7. タスクBがrel_mplで96バイトのブロックZを返却します。空き領域の合計はタスクAの要求する256バイト以上存在しますが、256バイトの連続した空きが無いので、タスクAは待ち状態のままです。
8. タスクBがrel_mplで32バイトのブロックYを返却します。この結果、256バイト以上の空きが生じたので、タスクAの待ち状態は解除され、タスクAに256バイトのブロックWが割付けられます。

2.15.2　空き領域の断片化とその対策

可変長メモリプールからメモリブロックの獲得と解放(返却)を繰り返していると、空き領域の断片化が発生し、空き領域のトータルサイズは十分でも、連続した空き領域が存在しない、つまり大きなメモリブロックを獲得できない状況になることがあります(図 2.13)。

図2.13　空き領域の断片化

コンフィギュレータの CFG_NEWMPL を選択すると、この問題がある程度軽減されます。さらに、可変長メモリプールの属性に VTA_UNFRAGMENT を指定すると、より断片化を軽減できる場合があります。断片化度合いは、一般には VTA_UNFRAGMENT を指定したほうが良くなる傾向がありますが、メモリプールの利用方法に依存します。

CFG_NEWMPL を選択した場合、可変長メモリプール生成時に指定する T_CMPL 構造体に、VTA_UNFRAGMENT 属性用のパラメータ(最小ブロックサイズ、セクタ数、管理テーブルアドレス)が追加されます。

VTA_UNFRAGMENT 属性を指定すると、可変長メモリプールがセクタ方式で管理されます。セクタ方式では、「最小ブロックサイズ」×8 バイトまでのブロックを「微小なブロック」として扱います。また、獲得要求サイズは表 2.16に示すように切り上げて扱います。

微小なブロックの獲得要求があると、カーネルはその切上げ後のサイズのブロックで構成されるセクタを生成します。セクタのサイズは常に minblksz×32 です。つまり、要求サイズに応じて、セクタ内のブロック数が異なります。

表2.16　微小ブロックの扱い

獲得要求サイズ (blksz) *	切上げ後のサイズ *	セクタ内のブロック数
0＜blksz≦minblksz	minblksz	32
minblksz＜blksz≦minblksz×2	minblksz×2	16
minblksz×2＜blksz≦minblksz×4	minblksz×4	8
minblksz×4＜blksz≦minblksz×8	minblksz×8	4

【注】blksz：要求サイズ、minblksz：最小ブロックサイズ

そして、カーネルはセクタ内のメモリブロックを割り当てます。セクタ内の残りのブロックは、以降のそのブロックサイズ以下のメモリブロック獲得要求のために予約されることになります。
このように、微小なブロックを連続して使用されるようにすることで、比較的大きなサイズの連続空き領域が維持されやすくなっています.

図 2.14に、「最小ブロックサイズ」が 32 の可変長メモリプールの例を示します。
最初に 32 バイトのメモリブロック獲得要求があると、32×32＝1024 バイトのセクタ[A]を確保し、そのセクタ内の 32 バイトの領域[A-1]を割り付けます(図 2.14 (1))。次に 16 バイトの獲得要求があると、A の中の 32 バイトの領域(A-2)を割り付けます(図 2.14 (2))。
さらに、今度は 36 バイトのメモリブロック獲得要求があったとします。この場合、セクタ A の各ブロックは 32 バイトなので、この要求には対応できません。そこで、要求サイズの 36 を最小ブロックサイズで切上げた 64 バイトのブロックが 16 個で構成される新たなセクタ B(64×16=1024 バイト)を確保し、そのセクタ内の 64 バイトの領域[B-1]を割り付けます(図 2.14 (3))。

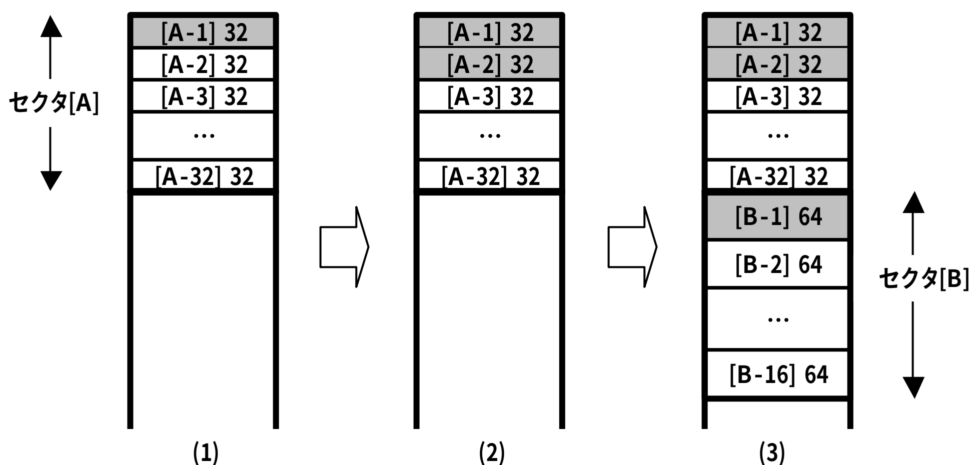


図2.14　可変長メモリプールの例

最大セクタ数を使いきってしまっている場合、または新たなセクタを生成するだけの連続空き領域がない場合には、セクタを生成せずに要求サイズのメモリブロックを獲得します。つまり、この場合には、断片化が発生しやすくなります。さらに要求サイズの連続空き領域も無い場合は、より大きなブロックサイズを持つセクタから獲得します。
セクタ内の全てのブロックが解放されると、セクタそのものも解放されます。
また、微小でないサイズの要求(minblksz×8 を超える要求サイズ)では、常にセクタを生成せずに要求サイズのメモリブロックを獲得します。

2.15.3 可変長メモリプールの管理

カーネルは、割り付けたメモリブロックを管理するために、可変長メモリプール内に管理テーブルを生成します。つまり、可変長メモリプールは、アプリケーションが獲得するメモリブロックの領域以外に OS の管理テーブルのためにも使用されます。このことを考慮して、可変長メモリプールサイズを決定する必要があります。

(1) CFG_NEWMPL 非選択時

メモリブロックの獲得時に 16 バイトの管理テーブルを生成します。この管理テーブルは、メモリブロック返却時に解放されます。

(2) CFG_NEWMPL 選択時

VTA_UNFRAGMENT 属性の指定がある場合は、セクタの生成時に 32 バイトの管理テーブルを生成します。この管理テーブルは、セクタ解放時に解放されます。

また、VTA_UNFRAGMENT 属性の指定がある場合でセクタ外のメモリブロックの獲得時、および VTA_UNFRAGMENT 属性の指定がない場合のメモリブロックの獲得時にも、32 バイトの管理テーブルを生成します。この管理テーブルは、メモリブロック返却時に解放されます。

2.16　時間管理

カーネルは、時間に関する以下のような機能をサポートしています。

- システム時刻の参照・設定
- タイムイベントハンドラ（周期ハンドラ、アラームハンドラ、オーバーランハンドラ）の実行制御
- タイムアウトなどの時間によるタスクの実行制御

カーネルは、システム時刻と呼ぶカウンタによってこれらの機能を実現しています。サービスコールで使用する時間パラメータの単位時間は 1ms です。一方、タイムティックを供給する周期は、コンフィギュレータで CFG_TICNUME(タイムティック周期の分子)および CFG_TICDENO(タイムティック周期の分母)を指定することで、1ms 以外にも設定できます。

時間管理機能を使用するには、タイマドライバを組み込む必要があります。タイマドライバには、標準タイマドライバと最適化タイマドライバの 2 種類があります。ただし、最適化タイマドライバは HI7700/4 でのみサポートしている機能です。詳細は、「付録 D　タイマドライバ」を参照してください。

表 2.17にシステム時刻を操作するサービスコールを示します。なお、isig_tim はコンフィギュレータの設定(CFG_TIMUSE)によって自動的に実行されるようになります。

表2.17　システム時刻を操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	isig_tim	**タイムティックの供給**	3	get_tim, iget_tim	**システム時刻の参照**
2	set_tim, iset_tim	**システム時刻の設定**			

2.16.1 周期ハンドラ

周期ハンドラは、指定した起動位相経過後、起動周期ごとに起動されるタイムイベントハンドラです。表 2.18に周期ハンドラを操作するサービスコールを示します。

表2.18 周期ハンドラを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_cyc, icre_cyc	周期ハンドラの生成	4	sta_cyc, ista_cyc	周期ハンドラの動作開始
2	acre_cyc, iacre_cyc	周期ハンドラの生成 (ID 番号自動割付け)	5	stp_cyc, istp_cyc	周期ハンドラの動作停止
3	del_cyc	周期ハンドラの削除	6	ref_cyc, iref_cyc	周期ハンドラの状態参照

周期ハンドラは、コンフィギュレータで生成することもできます。

周期ハンドラの起動には、起動位相を保存する方法と起動位相を保存しない方法があります。起動位相を保存する場合は、周期ハンドラの生成時点を基準に周期ハンドラを起動します。起動位相を保存しない場合は、周期ハンドラの動作開始時点を基準に周期ハンドラを起動します。図 2.15に周期ハンドラの動作例を示します。

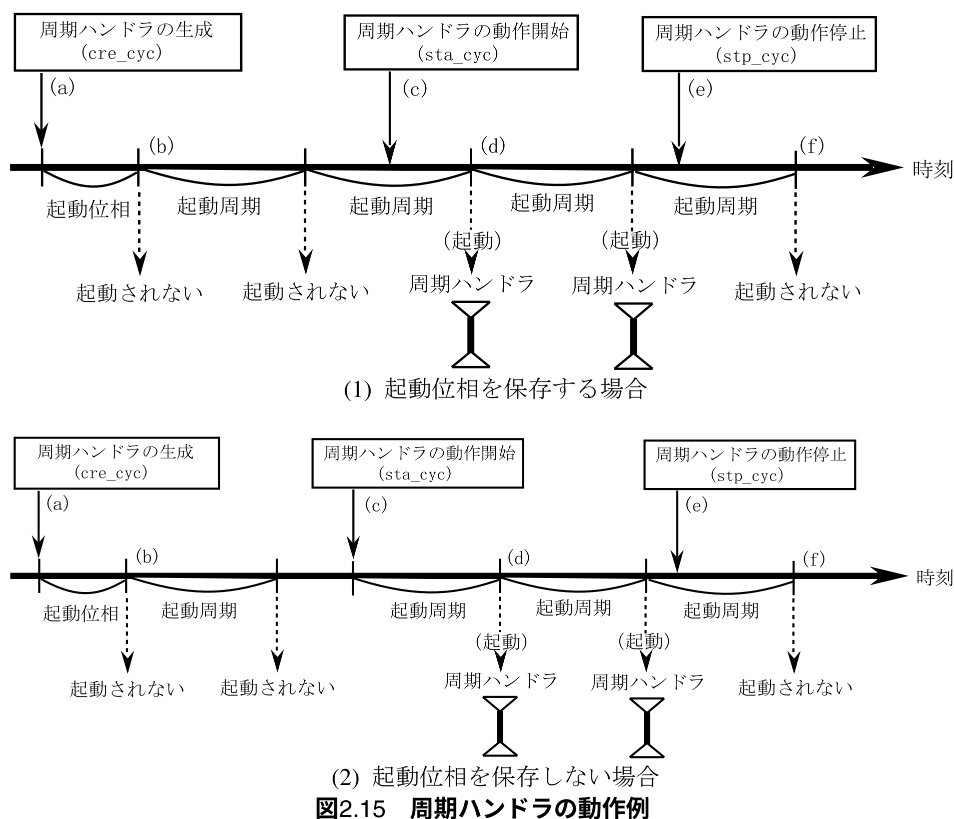


図2.15　周期ハンドラの動作例

図の解説

(a)　周期ハンドラ（TA_STA属性なし）を生成します。

(b)　周期ハンドラの動作が開始されていないため、起動位相、起動周期時間が経過しても周期ハンドラは起動されません。

(c)　sta_cycサービスコールによって周期ハンドラの動作が開始されます。

(d)　起動位相を保存する場合(1)は、周期ハンドラ生成時点からの起動周期で周期ハンドラが起動されます。起動位相を保存しない場合(2)は、sta_cycサービスコール呼び出し時点からの起動周期で周期ハンドラが起動されます。

(e)　stp_cycサービスコールによって周期ハンドラの動作が停止されます。

(f)　周期ハンドラの動作が停止されているため、周期時間が経過しても周期ハンドラは起動されません。

2.16.2　アラームハンドラ

アラームハンドラは、指定した時刻になると 1 度だけ起動されるタイムイベントハンドラです。
アラームハンドラを用いることにより、時刻に依存した処理を行うことができます。
表 2.19にアラームハンドラを操作するサービスコールを示します。

表2.19　アラームハンドラを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	cre_alm, icre_alm	アラームハンドラの生成	4	sta_alm, ista_alm	アラームハンドラの動作開始
2	acre_alm, iacre_alm	アラームハンドラの生成 (ID 番号の自動割付け)	5	stp_alm, istp_alm	アラームハンドラの動作停止
3	del_alm	アラームハンドラの削除	6	ref_alm, iref_alm	アラームハンドラの状態参照

アラームハンドラは、コンフィギュレータで生成することもできます。
図 2.16にアラームハンドラの動作例を示します。

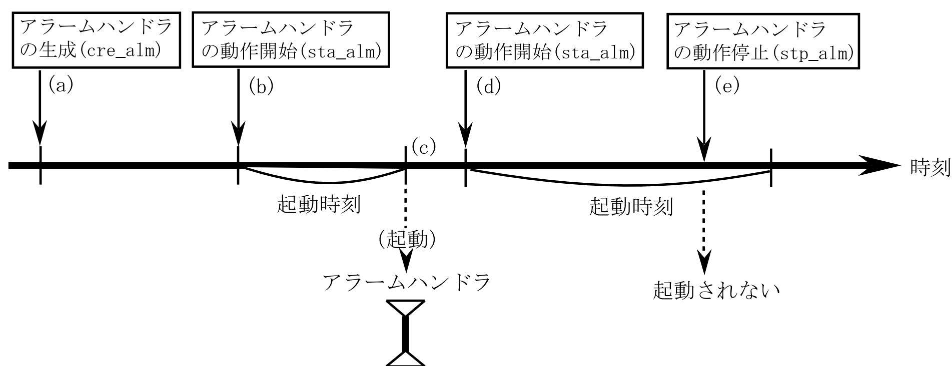


図2.16　アラームハンドラの動作例

図の解説

(a)　アラームハンドラを生成します。
(b)　sta_almサービスコールによってアラームハンドラの動作が開始されます。
(c)　指定した起動時刻が経過したため、アラームハンドラが起動されます。
(d)　sta_almサービスコールを別の起動時刻を指定して呼び出すと、再びアラームハンドラの動作が開始されます。
(e)　起動時刻になる前にstp_almサービスコールが呼び出されたため、アラームハンドラは起動されません。

2.16.3 オーバーランハンドラ

オーバーランハンドラは、各タスクに設定された時間を超えてプロセッサを使用した場合に起動されるタイムイベントハンドラで、システムにひとつだけ定義できます。

表 2.20にオーバーランハンドラを操作するサービスコールを示します。

表2.20 オーバーランハンドラを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	def_ovr	**オーバーランハンドラの定義**	3	stp_ovr, istp_ovr	**オーバーランハンドラの動作停止**
2	sta_ovr, ista_ovr	**オーバーランハンドラの動作開始**	4	ref_ovr, iref_ovr	**オーバーランハンドラの状態参照**

オーバーランハンドラは、コンフィギュレータで定義することもできます。

図 2.17にオーバーランハンドラの動作例を示します。

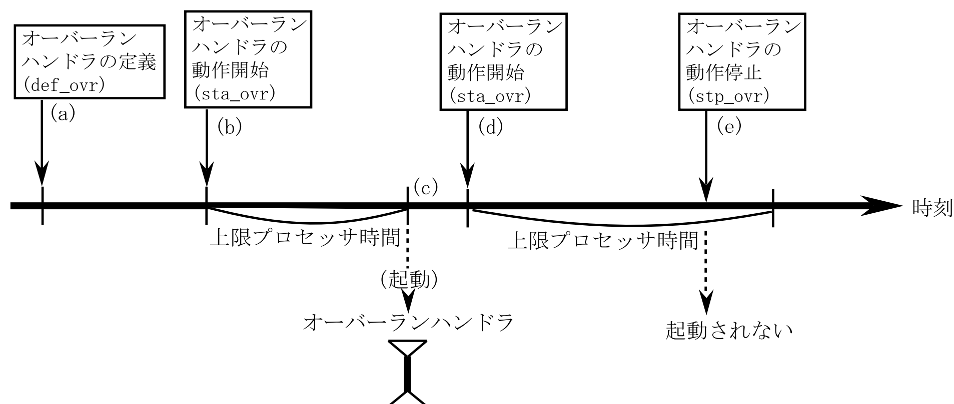


図2.17 オーバーランハンドラの動作例

図の解説

(a) オーバーランハンドラを定義します。

(b) sta_ovrサービスコールで、タスクに対して上限プロセッサ時間を設定します。この時点からオーバーランハンドラの動作が開始されます。

(c) タスクが使用した累計プロセッサ時間が指定した上限プロセッサ時間を超えると、オーバーランハンドラが起動されます。

(d) 次にsta_ovrサービスコールで上限プロセッサ時間を変更すると、再びオーバーランハンドラの動作が開始されます。

(e) タスクが使用した累計プロセッサ時間が、指定した上限プロセッサ時間を超える前にstp_ovrサービスコールが呼び出されると、オーバーランハンドラの動作が停止します。その後、上限プロセッサ時間に達しても、オーバーランハンドラは起動されません。

2.16.4 注意事項

(1) 多用による弊害

タイマ割込み発生時には、カーネルは以下の処理を行います。

(a) システム時刻の更新

(b) 起動時刻に達した全てのアラームハンドラの起動と実行

(c) 周期時間に達した全ての周期ハンドラの起動と実行

(d) タスクが使用した累計プロセッサ時間が、指定した上限プロセッサ時間を超えた場合のオーバーランハンドラの起動と実行

(e) tslp_tskなどのタイムアウト付きサービスコールによるタスクのタイムアウト処理

これらの処理は全て、タイマ割込みレベルをマスクした状態で行われます。

上記のうち、(b),(c),(e)は、複数のタスクやハンドラに対する処理が重複する可能性があるため、このような場合カーネルの処理時間が極端に長くなります。これは、以下のような弊害をもたらします。

- 割込みに対するレスポンスの悪化
- システム時刻の遅れ

これを避けるために、以下を遵守してください。

- タイムイベントハンドラの処理は、可能な限り短くしてください。
- タイムイベントハンドラの周期、タイムアウト付きサービスコールで指定するタイムアウト値は、なるべく大きな値にしてください。極端な例としては、ある周期ハンドラの周期時間が 1ms で、そのハンドラ処理時間が 1ms 以上かかるような場合、永久にその周期ハンドラだけが実行されることになり、事実上ハングアップします。

(2) 時間の管理方法

サービスコールで指定する時間パラメータは、すべて相対時間で指定します。つまり、相対時間に 1ms が指定されると、サービスコールが呼び出されてから 1ms が経過した次のタイムティックでイベント処理が行われます。相対時間に 0ms が指定されると、サービスコールが呼び出された後の最初のタイムティックでイベント処理が行われます。

また、set_tim サービスコールでシステム時刻を変更しても、それ以前に行った時間管理要求に関するイベントまでの実際の相対時間は変化しません。

2.17 システム状態管理

システム状態管理のための機能として、表 2.21に示すサービスコールをサポートしています。

表2.21　システム管理のサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	rot_rdq, irot_rdq	タスク優先順位の回転	9	sns_dsp	ディスパッチ禁止状態の参照
2	get_tid, iget_tid	実行状態のタスク ID の参照	10	sns_dpn	ディスパッチ放流状態の参照
3	loc_cpu, iloc_cpu	CPU ロック状態への移行	11	vsta_knl, ivsta_knl	カーネルの起動
4	unl_cpu, iunl_cpu	CPU ロック状態の解除	12	vsys_dwn, ivsys_dwn	システムダウン
5	dis_dsp	ディスパッチの禁止	13	vget_trc, ivget_trc	ユーザイベントのトレース取得
6	ena_dsp	ディスパッチの許可	14	ivbgn_int	割込みハンドラの開始をトレース取得
7	sns_ctx	コンテキストの参照	15	ivend_int	割込みハンドラの終了をトレース取得
8	sns_loc	CPU ロック状態の参照			

2.17.1 システムダウン

システムに異常が発生した場合、システムダウンとなります。システムダウン時に起動されるプログラムをシステムダウンルーチンと呼びます。

システムダウンの要因は、以下の 3 種類に分類できます。

- アプリケーションからの vsys_dwn, ivsys_dwn サービスコール呼び出し
- カーネル内部で異常を検出
- 未定義の割込み、例外が発生

システムダウンルーチンは、必ずユーザが作成しなければなりません。詳細は、「4.12 システムダウンルーチン」を参照してください。

2.17.2 サービスコールトレース機能

サービスコールトレース機能は、サービスコールの履歴を保存する機能で、その結果はデバッギングエクステンションを用いてグラフィカルに参照することができます。

サービスコールトレース機能により、以下のタイミングでタスク ID、PC、サービスコールパラメータなどが自動的に取得されます。

- サービスコール発行とリターン
- タスクの実行開始と実行終了
- タスク例外処理ルーチンの実行開始と実行終了

また、割込みハンドラの先頭に ivbgn_int、最後に ivend_int サービスコール記述すると、割込みハンドラの実行開始/実行終了も取得できます。

さらに、vget_trc, ivget_trc サービスコールを使用すれば、任意のタイミングで任意の情報を取得することもできます。

(1) 履歴情報の保存場所

履歴情報の保存場所としてターゲットメモリまたはデバッガ(E6000 エミュレータ, シミュレータなど)のどちらかを選択できます。前者を「ターゲットトレース」、後者を「エミュレータトレース(ツールトレース)」と呼びます。

エミュレータトレース(ツールトレース)は、E6000 エミュレータやシミュレータなど、利用できる環境は限定されますが、ターゲット上にサービスコールトレースのための領域をほとんど必要としないという特徴があります。エミュレータトレース(ツールトレース)を利用可能な環境は、デバッギングエクステンションのマニュアルまたはオンラインヘルプを参照してください。

(2) サービスコールトレース機能の使用準備

サービスコールトレース機能の設定は、コンフィギュレータのデバッグ機能ページで行います。

また、リンク時にはトレース用のセクション(ターゲットトレースの場合は B_hitrcbuf, エミュレータトレースの場合は B_hitrceml)を適切なアドレスに配置してください。

(3) 注意事項

(a) パフォーマンスの低下

サービスコールトレース機能を使用すると、若干カーネルのパフォーマンスが低下します。

(b) 取得されないサービスコール等

非タスクコンテキスト用の ixxx_yyy サービスコールとタスクコンテキスト用の xxx_yyy サービスコールが存在するものは、両方とも xxx_yyy サービスコールとして取得されます。

また、以下のサービスコールは取得されません。

isig_tim, cal_svc, ical_svc, vsta_knl, ivsta_knl, vsys_dwn, ivsys_dwn, vini_cac, ivini_cac

また、HI7700/4 においては、以下のサービスコールも取得されません。

vchg_cop

また、HI7750/4 においては、以下のサービスコールも取得されません。

vfls_cac, ivfls_cac, vclr_cac, ivclr_cac, vinv_cac, ivinv_cac

2.18　割込み管理・システム構成管理

本カーネルでは、割込み・例外を以下のように分類しています。

- リセット：CPU リセットです。リセットによって実行されるプログラムを「CPU 初期化ルーチン」と呼びます。
- 割込み：外部割込み端子や周辺モジュールからの割込みによって発生します。割込み発生時には、割込みハンドラが実行されます。
- CPU 例外：アドレスエラーやゼロ除算などの例外です。CPU 例外には TRAPA 命令によって発生するトラップ例外も含みます。CPU 例外発生時には、CPU 例外ハンドラが実行されます。

表 2.22に割込み管理機能のサービスコールを、表 2.23にシステム構成管理機能のサービスコールを示します。

表2.22　割込み管理機能のサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	def_inh, idef_inh	割込みハンドラの定義	3	get_ims, iget_ims	割込みマスクの参照
2	chg_ims, ichg_ims	割込みマスクの変更			

表2.23　システム構成管理機能のサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	def_exc, idef_exc	CPU 例外ハンドラの定義	3	ref_cfg, iref_cfg	コンフィギュレーション情報の参照
2	vdef_trp, ivdef_trp	CPU 例外ハンドラの定義 (TRAPA 命令用)	4	ref_ver, iref_ver	バージョン情報の参照

割込みハンドラ、CPU 例外ハンドラ(TRAPA 命令用を含む)は、コンフィギュレータでも定義できます。なお、未定義の割込み、例外が発生すると、システムダウンとなります。

2.18.1　リセットとカーネルの起動

CPU リセットからカーネル起動までの手順は、一般には図 2.18に示すようになります。

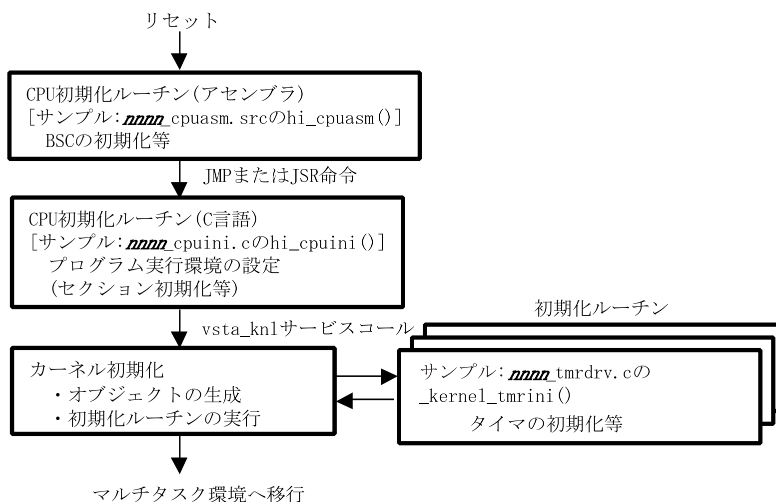


図2.18　リセットからカーネル起動までの流れ

CPU 初期化ルーチンでは、カーネルを含めたソフトウェア全体が動作するのに必要な処理を行います。具体的には、メモリを正しくアクセスできるようにするための BSC(バスステートコントローラ)の設定がこれに該当します。特に C 言語プログラムはスタックをアクセスしますので、C 言語プログラムが実行する前にスタックをアクセス可能にする必要があります。

また、セクションの初期化などの C 言語プログラムの実行環境設定もここで行います。これについては、『SuperH™ RISC engine C/C++コンパイラユーザーズマニュアル』も参照してください。

その後、vsta_knl または ivsta_knl サービスコールを呼び出すことでカーネルを起動します。いったんカーネルを起動すると、呼び出し元には戻りません。

vsta_knl, ivsta_knl サービスコールでは、以下のような処理を行います。

(a)　全ての割込みを禁止(HI7000/4ではSR.IMASK=15、HI7700/4およびHI7750/4ではSR.BL=1)
(b)　VBRの初期化
(c)　HI7700/4およびHI7750/4では、SR.BL=0かつSR.IMASK=15に設定
(d)　カーネル作業領域の初期化
(e)　SR.IMASKをカーネル割込みマスクレベルに設定。HI7700/4およびHI7750/4では、同時にSR.BL=0に設定
(f)　コンフィギュレータに初期登録するように指定されたオブジェクトの生成
(g)　コンフィギュレータに設定された初期化ルーチンの呼び出し
(h)　マルチタスク環境に移行します。この時点で、全ての最初のタスクが実行がされます。

なお、vsta_knl, ivsta_knl サービスコールを呼び出す場合は、通常はカーネルとリンクしなければなりませんが、vsta_knl, ivsta_knl サービスコールのアドレスは P_hireset セクションの先頭アドレスと同じですので、P_hireset セクションの配置アドレスが分かっていれば、カーネルとリンクしなくても P_hireset セクションの先頭アドレスに分岐することで、vsta_knl, ivsta_knl サービスコールを発行できます。

CPU 初期化ルーチンの作成については、「4.11 CPU 初期化ルーチン」を参照してください。

2.18.2 割込みハンドラ

(1) 概要

割込みが発生すると、カーネルの割込み出入口処理を介して割込みハンドラが起動されます。

本カーネルでは、より高速な割込み応答のために、カーネル実行中にマスクする割込みレベルをコンフィギュレーション時に定義できるようになっています。このレベルのことを「カーネル割込みマスクレベル(CFG_KNLMSKLVL)」と呼びます。カーネル割込みマスクレベルより高いレベルの割込みハンドラは、カーネル実行中であっても直ちに受け付けられます。ただし、カーネル割込みマスクレベルより高いレベルの割込みハンドラから、サービスコールを呼び出してはなりません。

割込みハンドラは、非タスクコンテキストで実行します。割込みハンドラ実行中に呼び出したサービスコールによって優先度の高いタスクが実行可能状態となっても、その時点ではタスクスケジューリングは行われず、割込みハンドラの処理が終了するまで遅延されます。

割込みハンドラでは、プロセッサの割込みマスクレベル(SR レジスタの IMASK ビット)を自割込みレベルよりもさげてはなりません。

(2) ダイレクト割込みハンドラ(HI7000/4 のみ)

HI7000/4 では、カーネルの割込み出入口処理を介さずに起動される「ダイレクト割込みハンドラ」も使用することができます。ダイレクト割込みハンドラは、CPU の割込みベクタに直接登録されるため、通常の割込みハンドラよりも割込み応答が高速です。カーネル割込みマスクレベルよりも高い割込みハンドラは、ダイレクト割込みハンドラとして定義しなければなりません。

２つの割込みハンドラの違いを、表 2.24に示します。

表2.24　２つの割込みハンドラの違い

項番	項目	通常の割込みハンドラ	ダイレクト割込みハンドラ
1	割込み発生時の挙動	カーネルの割込み処理を介して起動される	カーネルを介さずに起動される(CPU の割込みベクタに登録される)
2	サービスコールの発行	非タスクコンテキスト用を利用可能。ただし、CFG_KNLMSKLVL より高いレベルの割込みハンドラはサービスコール発行禁止。	
3	割込みレベル	CFG_KNLMSKLVL より高いレベルの割込みは、ダイレクト割込みハンドラとして記述する必要があります	
4	定義方法	・def_inh, idef_inh サービスコール ・コンフィギュレータで定義	
5	ハンドラ記述方法	「4.8 割込みハンドラ」を参照してください。	

(3) レジスタバンク(SH-2A, SH2A-FPU)

「4.2.9 レジスタバンク(SH-2A, SH2A-FPU)」を参照してください。

2.18.3　割込みの禁止

割込みを禁止するには、以下の方法があります。

(a)　loc_cpu, iloc_cpuサービスコール：SRレジスタのIMASKビットをカーネル割込みマスクレベルに変更
(b)　chg_ims, ichg_imsサービスコール：SRレジスタのIMASKビットを変更
(c)　サービスコールを使わずにSRレジスタのIMASKビットを変更する
(d)　サービスコールを使わずにSRレジスタのBLビットを変更する(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4Aのみ)

なお、SR レジスタの IMASK ビットがカーネル割込みマスクレベル(CFG_KNLMSKLVL)よりも高いとき、および SR の BL ビットが 1 の時にはサービスコールを発行してはなりません。

(a)　loc_cpu, iloc_cpu サービスコール

この方法では、CPU ロック状態に遷移します。CPU ロック状態では、カーネル割込みマスクレベル以下の割込みが禁止されます。

なお、CPU ロック状態の間は「3.2.5 システム状態とサービスコール」に記載のように発行可能なサービスコールが制限されます。

CPU ロック状態を解除するには、unl_cpu または iunl_cpu を発行してください。なお、割込みハンドラ、CPU 例外ハンドラ、タイムイベントハンドラで CPU ロック状態に遷移させた場合は、そのハンドラ内で CPU ロック状態を解除しなければなりません。

(b)　chg_ims, ichg_ims サービスコール

この方法では、SR レジスタの IMASK ビットが指定した値に変更されます。即ち、指定したレベル以下の割込みが禁止されます。

また、CPU ロック状態の時には、chg_ims, ichg_ims はエラーE_CTX となります。

chg_ims, ichg_ims によって SR レジスタの IMASK ビットを 0 以外に変更している間は、非タスクコンテキストと扱われます。

割込み禁止を解除するには、ichg_ims で SR レジスタの IMASK ビットを変更前の値に戻してください。

ただし、タスクコンテキストで chg_ims(変更後の値[A])によって割込みを禁止した場合は、ichg_ims で SR の IMASK ビットを元に戻す時には SR の IMASK ビットが[A]となっていなければなりません。そうでない場合、正常な動作は保証されません。以下に、悪い例を示します。

```
/* この時点ではタスクコンテキストで、SR の IMASK ビットは 0 */
chg_ims(SR_IMS05);  /* (1)割込みを禁止する：SR レジスタの IMASK ビットを 5 に変更 */
set_imask(8);       /* (2)SR の IMASK ビットを 8 に変更 */
ichg_ims(SR_IMS00); /* (3)割込み禁止を解除する */
                    /* この時点の SR レジスタの IMASK ビットは(1)で変更したときの値 */
                    /* の 5 でなければならないが、8 になっている。 */
```

この例を改善するには、以下のように(3)の前に SR の IMASK ビットを 5 に戻す処理を追加します。

```
/* この時点ではタスクコンテキストで、SR の IMASK ビットは 0 */
chg_ims(SR_IMS05);  /* (1)割込みを禁止する：SR レジスタの IMASK ビットを 5 に変更 */
```

```
set_imask(8);       /* (2)SR の IMASK ビットを 8 に変更 */
set_imask(5);       /* (追加)SR の IMASK ビットを 5 に戻す */
ichg_ims(SR_IMS00); /* (3)割込み禁止を解除する */
```

(c) **サービスコールを使わずに** SR **レジスタの** IMASK **ビットを変更する**

LDC 命令によって SR レジスタの IMASK ビットを変更します。C 言語では、コンパイラがサポートしている組み込み関数 set_imask()または set_cr()を使用します。

タスクコンテキストでは、カーネル割込みマスクレベルよりも高い値にのみ変更できます。それ以外の値に変更した場合、正常な動作は保証されません。一方、非タスクコンテキストでは、この制限はありません。

割込み禁止を解除するには、SR レジスタの IMASK ビットを変更前の値に戻してください。

(d) SR **レジスタの** BL **ビットを変更する**(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A **のみ**)

LDC 命令によって SR レジスタの BL ビットを 1 にします。C 言語では、コンパイラがサポートしている組み込み関数 set_cr()を使用します。

割込み禁止を解除するには、SR レジスタの BL ビットを元の値に変更してください。

2.18.4 カーネル割込みマスクレベル(CFG_KNLMSKLVL)

カーネル割込みマスクレベル(CFG_KNLMSKLVL)はカーネル実行中に禁止する割込みレベルで、コンフィギュレータで設定します。カーネル割込みマスクレベルよりも高いレベルの割込みは、カーネル実行中でも即座に受け付けられますが、そのハンドラからはサービスコールを呼び出してはなりません。

また、割込みマスクをカーネル割込みマスクレベルより高く設定している間は、chg_ims サービスコールで割込みマスクをカーネル割込みマスクレベル以下に変更する場合を除いて、サービスコールを呼び出してはなりません。

HI7000/4 では、カーネル割込みマスクレベルよりも高い割込みハンドラは、ダイレクト割込みハンドラとして定義しなければなりません。

2.18.5 CPU 例外

CPU 例外ハンドラ（TRAPA 命令例外含む）は、例外が発生したときと同じコンテキストで動作します。スタックも例外発生元のものを使用します。CPU 例外ハンドラは、ディスパッチャよりも優先順位が高いため、CPU 例外ハンドラ実行中はタスク切り替えは発生しません。

CPU 例外ハンドラでは、タスク実行モードなどの状態は、例外発生時の状態を引き継ぎます。

CPU 例外ハンドラが動作するコンテキストは、例外発生時のコンテキストになりますが、CPU 例外ハンドラから呼び出し可能なサービスコールは、以下に示すサービスコールに限られます。

- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- sns_tex
- get_tid, iget_tid
- ras_tex, iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

2.19 サービスコール管理

サービスコール処理ルーチンを作成し、拡張サービスコールルーチンとしてカーネルに登録することができます。システム共通の処理を拡張サービスコールルーチンとして作成しておけば、その処理ルーチンとリンクしなくとも呼び出すことができます。

表 2.25に拡張サービスコールを操作するサービスコールを示します。

表2.25 拡張サービスコールを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	def_svc, idef_svc	**拡張サービスコールの定義**	2	cal_svc, ical_svc	**サービスコールの呼び出し**

図 2.19に拡張サービスコールの概要を示します。

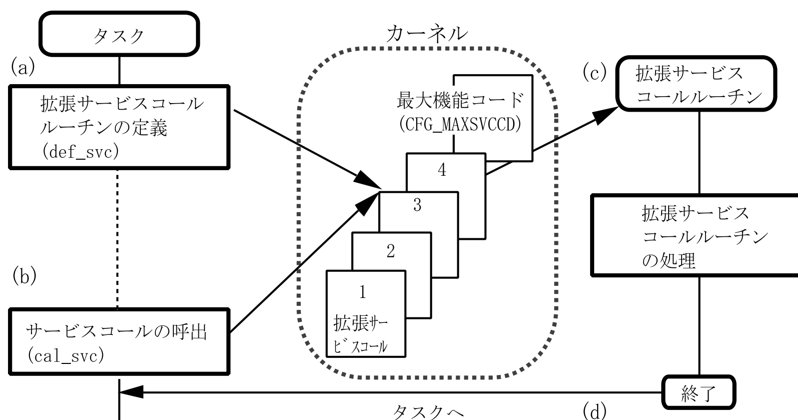


図2.19 拡張サービスコールの概要

図の解説

(a) 拡張サービスコールを定義します。

(b) 定義した拡張サービスコールルーチンを呼び出すため、cal_svcサービスコールを呼び出します。

(c) タスクが呼び出したcal_svcサービスコールによって、拡張サービスコールルーチンが起動されます。

(d) 拡張サービスコールルーチン終了後、呼び出し元のタスクへ復帰します。

拡張サービスコールルーチンでは、呼び出し元のタスク実行モードを引き継ぎます。呼び出し元タスクが強制終了要求（ter_tsk サービスコール）、強制待ち要求（sus_tsk サービスコール）をマスクしていなければ、拡張サービスコールルーチン実行中でも、それらのサービスコールが呼び出されると、ただちに受け付けられます。

RENESAS

2.20　キャッシュサポート（HI7700/4、HI7750/4）

キャッシュサポート機能は、アプリケーションからメモリ内容の転送（DMA 転送）を行う時など、キャッシュとメモリとの間のコヒーレンシをとる必要があるときに使用します。

表 2.26にキャッシュを操作するサービスコールを示します。

表2.26　キャッシュを操作するサービスコール

項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	vini_cac, ivini_cac	**キャッシュ初期化**	3	vfls_cac, ivfls_cac	**キャッシュフラッシュ**
2	vclr_cac, ivclr_cac	**キャッシュクリア**	4	vinv_cac, ivinv_cac	**キャッシュ無効化**

図 2.20にキャッシュサポートの概要を示します。

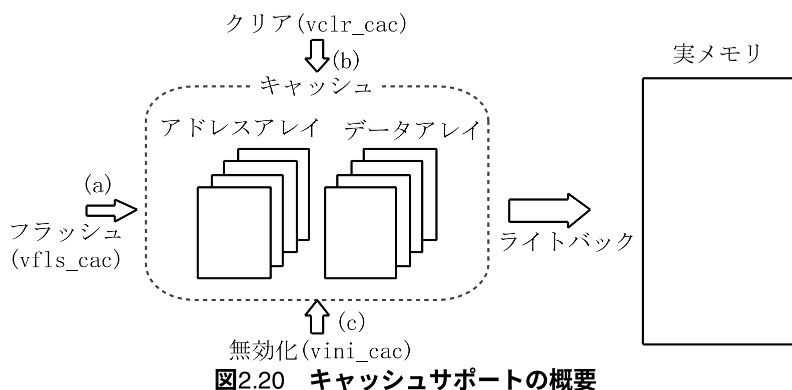


図2.20　キャッシュサポートの概要

(1)　フラッシュ

キャッシュの内容をメモリに書き出すことで、キャッシュとメモリとの間のコヒーレンシをとります。DMA など他のバスマスタが CPU によって更新したメモリ内容を読み出す場合には、キャッシュをフラッシュする必要があります。

なお、キャッシュをライトスルーモードで使用している場合は、フラッシュの必要性はありません。

(2)　クリア

キャッシュの内容をメモリに書き出し、さらにキャッシュ内容を無効化します。

(3)　無効化

DMA など他のバスマスタによって更新されたメモリ内容を CPU から読み出す場合、確実にメモリから読み込むためにはキャッシュを無効化する必要があります。

ライトバックモードで使用時にキャッシュを無効化する場合、まだメモリに書き出されていないデータがキャッシュに存在してもライトバックは発生しません。メモリに書き出されていないデータは単に捨てられる（メモリに書き出されない）ため、使い方を誤るとシステムが正しく動作しなくなるので注意が必要です。

(4)　初期化

キャッシュサポート機能を使用するには、その前に vini_cac, ivini_cac サービスコールを実行してお

く必要があります。vini_cac, ivini_cac サービスコールは、カーネル起動前にも発行できます。

2.21　カーネルのアイドリング

実行可能状態のタスクが存在しなくなると、カーネルは後述のプリフェッチ機能の処理を行った後に内部で無限ループとなり、割込みが発生するのを待ちます。

CPU の省電力モードを利用するには、通常は最低優先度のタスクで省電力モードに移行するようにします。

2.22　プリフェッチ機能(HI7700/4, HI7750/4)

プリフェッチ機能はアイドリング後に発生する割込みの高速応答を目的とした機能で、カーネルがシステムアイドリングに入った時に PREF(キャッシュへのプリフェッチ)命令を実行します。

PREF 命令で取り込む領域の範囲はコンフィギュレータで設定します。SH-3, SH3-DSP の PREF 命令は命令/データ混合型キャッシュに取り込む動作なので、指定可能な領域はプログラム領域でもデータ領域でも構いません。一方、SH4L-DSP, SH-4, SH-4A の PREF 命令はオペランドキャッシュに取り込む動作なので、指定できる領域はデータ領域です。

2.23　最適化タイマドライバ機能(HI7700/4)

HI7700/4 は、主に省電力化を目的とした最適化タイマドライバ機能をサポートしています。詳細は、「付録 E　最適化タイマドライバ(HI7700/4)」を参照してください。

2.24　DSP スタンバイ制御機能(HI7700/4)

HI7700/4 は、省電力化を目的とした DSP スタンバイ制御機能をサポートしています。詳細は、「付録 F DSP スタンバイ制御機能(HI7700/4)」を参照してください。

3. サービスコール

3.1 概要

各サービスコールの機能は、表 3.1のように分類されます。

表3.1 サービスコールの機能分類

項番	分類	機能
1	タスク管理機能	タスクの起動、終了など
2	タスク付属同期機能	タスク実行の中断、再開、タスク付属イベントフラグなど
3	タスク例外処理機能	タスク例外処理ルーチンの登録、タスク例外の要求、許可、禁止など
4	同期・通信機能	セマフォ、イベントフラグ、データキュー、メールボックス
5	拡張同期・通信機能	ミューテックス、メッセージバッファ
6	メモリプール管理機能	動的なメモリの割り付け
7	時間管理機能	カーネルへのタイマ刻みの通知、システム時刻の設定と参照、タイムイベントハンドラの定義など
8	システム状態管理機能	CPU ロック状態、ディスパッチ禁止状態への移行など
9	割込み管理機能	割込みハンドラの定義、割込みマスクの変更と参照など
10	サービスコール管理機能	拡張サービスコールの定義、呼び出し
11	システム構成管理機能	CPU 例外ハンドラの定義、コンフィギュレーション情報の参照など
12	キャッシュサポート機能（HI7700/4、HI7750/4）	キャッシュのフラッシュ、クリア、無効化

3.2 サービスコールインタフェース

サービスコールは、C 言語およびアセンブリ言語プログラムから呼び出せます。

3.2.1 C 言語 API

(1) ヘッダファイル

ヘッダファイルとして kernel.h をインクルードしてください。kernel.h は hihead フォルダにあります。ヘッダファイルについては、「4.1 ヘッダファイル」を参照してください。

(2) サービスコールの呼び出し形式

すべてのサービスコールは、以下のように C 言語の関数呼出の形式で記述します。

```
#include "kernel.h"
  /* ... */
   ercd = act_tsk(1);
```

(3) 基本データ型

HI7000/4 シリーズで使用する基本データ型の詳細を、表 3.2に示します。

表3.2　基本データ型

No.	データ型	定義内容	No.	データ型	定義内容
1	B	符号付き 8 ビット整数	21	PRI	符号付き 16 ビット整数
2	H	符号付き 16 ビット整数	22	SIZE	符号無し 32 ビット整数
3	W	符号付き 32 ビット整数	23	TMO	符号付き 32 ビット整数
4	UB	符号無し 8 ビット整数	24	RELTIM	符号無し 32 ビット整数
5	UH	符号無し 16 ビット整数	25	SYSTIM	以下のメンバから構成される構造体 上位：符号無し 16 ビット整数 下位：符号無し 32 ビット整数
6	UW	符号無し 32 ビット整数			
7	VB	符号付き 8 ビット整数 *			
8	VH	符号付き 16 ビット整数 *	26	VP_INT	符号付き 32 ビット整数 *
9	VW	符号付き 32 ビット整数 *	27	ER_BOOL	符号付き 32 ビット整数
10	VP	void 型へのポインタ	28	ER_ID	符号付き 32 ビット整数
11	FP	void 型関数へのポインタ	29	ER_UINT	符号付き 32 ビット整数
12	INT	符号付き 32 ビット整数	30	TEXPTN	符号無し 32 ビット整数
13	UINT	符号無し 32 ビット整数	31	FLGPTN	符号無し 32 ビット整数
14	BOOL	符号付き 32 ビット整数	32	RDVPTN	符号無し 32 ビット整数
15	FN	符号付き 32 ビット整数	33	RDVNO	符号無し 32 ビット整数
16	ER	符号付き 32 ビット整数	34	OVRTIM	符号無し 32 ビット整数
17	ID	符号付き 16 ビット整数	35	INHNO	符号無し 32 ビット整数
18	ATR	符号無し 32 ビット整数	36	EXCNO	符号無し 32 ビット整数
19	STAT	符号無し 32 ビット整数	37	IMASK	符号無し 32 ビット整数
20	MODE	符号無し 32 ビット整数			

【注】* これらのデータタイプの変数の値を参照する場合や代入する場合には、明示的に型変換(キャスト)を行う必要があります。

3.2.2 アセンブリ言語 API

アセンブリ言語プログラムからサービスコールを呼び出すには、一部のサービスコールを除いて図3.1のようにして呼び出します。なお、オフセットは#OFF_XXX_XXX で指定します。XXX_XXX にはサービスコールと同じ名称を大文字で指定してください。

```
 .INCLUDE "kernel.inc"          →標準ヘッダ"kernel.inc"をインクルードしてください。
 .IMPORT __kernel_cnfgtbl       ←カーネルのサービスコールエントリテーブルを外部参照します。
_task:
 ; ・・・・・・
 MOV.L  #OFF_ACT_TSK,R0          ←サービスコールに対応したオフセット値を設定します。
 MOV.L  #__kernel_cnfgtbl,R1
 MOV.L  @(R0,R1),R0              ←発行するサービスコールのエントリアドレスを求めます。
 MOV    #1,R4                    ←パラメータの設定を行います。
 JSR    @R0                      ←サービスコールを呼び出します。
 nop
 ; ・・・・・・                  ←呼び出し元に戻らないサービスコールを除き、サービスコール終
                                 了後は PR レジスタのアドレスにリターンします。この例では、こ
                                 の位置にリターンします。リターンしたときには、R0 に正常終了
                                 (E_OK)またはエラーコードが設定されます。
```

図3.1　アセンブリ言語プログラムからのサービスコール呼び出し例

HI7000/4 で SH-2A または SH2A-FPU を使用する場合は、コンフィギュレータで CFG_TBR として「サービスコール呼び出し専用に使用」を選択することができます。この場合、サービスコールは TBR レジスタを使った呼び出しとなります。TBR レジスタを使用してサービスコールを呼び出すには、一部のサービスコールを除いて図 3.2のようにして呼び出します。なお、ディスプレースメントには INDEX_XXX_XXX を指定します。XXX_XXX にはサービスコールと同じ名称を大文字で指定してください。

```
 .INCLUDE "kernel.inc"              →標準ヘッダ"kernel.inc"をインクルードしてください。
_task:
; ・・・・・・
 MOV    #1,R4                        ←パラメータの設定を行います。
 JSR/N @@(INDEX_ACT_TSK,TBR)         ←サービスコールを呼び出します。
 NOP                                 ←呼び出し元に戻らないサービスコールを除き、サービスコー
; ・・・・・・                       ル終了後は PR レジスタのアドレスにリターンします。この例
                                     では、この位置にリターンします。リターンしたときには、
                                     R0 に正常終了(E_OK)またはエラーコードが設定されます。
```

図3.2　TBR を使用してサービスコールを呼び出す場合の記述例(SH-2A, SH2A-FPU)

3.2.3 サービスコール呼び出し前後のレジスタ保証規則

サービスコール呼び出し前後において、レジスタの値が同一であることを保証するレジスタと保証しないレジスタがあります。この規則は、基本的には当社製 C コンパイラに準じていますが、その詳細を表 3.3に示します。

表3.3　サービスコール発行前後のレジスタ保証

項番	レジスタ	レジスタ保証
1	SR, R8～R15, PR, GBR, MACH, MACL	保証されます。ただし、chg_ims, ichg_ims, loc_cpu, iloc_cpu, unl_cpu, iunl_cpu の場合は、SR.IMASK は更新されます。
2	R0	正常終了(E_OK)またはエラーコードが設定されます。
3	R1～R7	リターンパラメータとして明示している場合以外は保証されません。
4	[SH2-DSP, SH3-DSP, SH4AL-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	以下に示す状態から発行した場合のみ保証されます。 ・TA_COP0 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態
5	[SH-4, SH-4A] FPSCR	保証されます。
6	[SH-4, SH-4A] FR0～FR11 (DR0～DR10, FV0～FV8)	保証されません。
7	[SH-4, SH-4A] FR12～FR15 (DR12～DR14, FV12), FPUL	以下に示す状態から発行した場合のみ保証されます。 (1)FPSCR.FR=0 の状態で発行した場合 ・TA_COP1 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態 (2)FPSCR.FR=1 の状態で発行した場合 ・TA_COP2 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態
8	[SH-4, SH-4A] XF0～XF15 (XD0～XD14, XMTRX),	以下に示す状態から発行した場合のみ保証されます。 (1)FPSCR.FR=0 の状態で発行した場合 ・TA_COP2 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態 (2)FPSCR.FR=1 の状態で発行した場合 ・TA_COP1 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態
9	[SH-2A, SH2A-FPU] TBR	CFG_TBR を「サービスコール呼び出し専用に使用」または「タスクコンテキストと扱う」と設定していた場合は保証されます。 「4.2.8 TBR レジスタ(SH-2A, SH2A-FPU)」も参照してください。
10	[SH2A-FPU] FR0～FR11	保証されません。
11	[SH2A-FPU] FPSCR, FPUL FR12～FR15	以下に示す状態から発行した場合のみ保証されます。 ・TA_COP1 属性のタスクまたはタスク例外処理ルーチン ・【注】に記載した状態

【注】　非タスクコンテキスト、またはディスパッチ禁止状態

RENESAS

タスク、ハンドラ作成時のレジスタ保証については、「4. アプリケーションプログラムの作成」を参照してください。

3.2.4 サービスコールの返値とエラコード

リターンコードを持つサービスコールでは、正の値または0(E_OK)が正常終了、負の値がエラーコードを意味します。ただし、BOOL型の返値を持つサービスコールはこの限りではありません。正常終了時の返値の意味はサービスコール毎に異なりますが、多くのサービスコールの正常終了時は、E_OKのみが返ります。

エラーコードは、下位8ビットのメインエラーコードとそれを除いた上位ビットのサブエラーコードから構成されていますが、本カーネルではサブエラーコードは常に-1です。

なお、標準ヘッダitron.hには、以下のマクロが定義されています。本カーネルでは、SERCDマクロは常に-1を返すことになります。

- ER mercd = MERCD(ER ercd);　エラーコードからメインエラーコードを取り出す
- ER sercd = SERCD(ER ercd);　エラーコードからサブエラーコードを取り出す

また、コンフィギュレータで選択しなかったサービスコールを発行した場合は、サービスコールの型に関わらず、リターン値にエラーE_NOSPTが返ります。

3.2.5 システム状態とサービスコール

サービスコールを呼び出せるかどうかは、システムの状態（「2.4 システムの状態」参照）に依存します。

(1) タスクコンテキストと非タスクコンテキスト

サービスコールは、タスクコンテキスト専用サービスコールと非タスクコンテキスト専用サービスコール、すべてのコンテキストから呼び出し可能なサービスコールの3つに分類できます。

(a) 基本的に、”i”で始まるサービスコールは、非タスクコンテキスト専用です。

(b) 以下のサービスコールは、すべてのコンテキストから呼び出し可能です。

- sns_tex
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl *
- vsys_dwn, ivsys_dwn *
- vini_cac, ivini_cac *
- vclr_cac, ivclr_cac *
- vfls_cac, ivfls_cac *
- vinv_cac, ivinv_cac *

＊ これらは”i”で始まる名称ですが、例外的にタスクコンテキストからも発行できます。

(c) (a),(b)以外のサービスコールはタスクコンテキスト専用となります。

上記に示した仕様と異なるコンテキストからサービスコールを呼び出した場合の動作は保証されません。ただし、非タスクコンテキストから待ち状態に遷移するサービスコールを呼び出すなど、矛盾が生じる特別な場合については、E_CTXエラーが返ります。

(2) ディスパッチ禁止/許可状態

すべてのサービスコールは、ディスパッチ許可状態から呼び出せます。一方、待ち状態に遷移するサービスコールなどの一部のサービスコールは、ディスパッチ禁止状態からは呼び出せません。これらのサービスコールをディスパッチ禁止状態から呼び出すと、E_CTX エラーが返ります。

(3) CPU ロック/ロック解除状態

すべてのサービスコールは CPU ロック解除状態から呼び出せます。一方、CPU ロック状態から呼び出せるサービスコールは以下のものに限定されています。これら以外のサービスコールを CPU ロック状態から呼び出した場合の動作は保証されません。ただし、待ち状態に遷移するサービスコールを呼び出した場合は、E_CTX エラーが返ります。

- ext_tsk
- exd_tsk
- 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

(4) CPU 例外ハンドラ

CPU 例外ハンドラから呼び出し可能なサービスコールは、以下のものに限定されています。これら以外のサービスコールを CPU 例外ハンドラから呼び出した場合の動作は保証されません。ただし、待ちに入るサービスコールを呼び出した場合は、E_CTX エラーが返ります。

- sns_tex
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- get_tid, iget_tid
- ras_tex, iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

(5) カーネル起動前

カーネル起動(vsta_knl サービスコール)前であっても、以下のサービスコールを呼び出すことができます。

- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn
- vini_cac, ivini_cac
- HI7700/4：vclr_cac, ivclr_cac
- HI7700/4：vfls_cac, ivfls_cac
- HI7700/4：vinv_cac, ivinv_cac

3.2.6 μITRON4.0 仕様範囲外のサービスコール

vscr_tsk、ivbgn_int サービスコールのように"v"または"iv"で始まる名称のサービスコールは、μITRON4.0 仕様の範囲外サービスコールです。

また、以下の"ixxx_yyy"('i'で始まる名称)のサービスコールは、μITRON4.0 仕様でタスクコンテキスト専用として用意されている"xxx_yyy"のサービスコールを非タスクコンテキストから呼び出せるようにしたもので、μITRON4.0 仕様の範囲外です。

icre_tsk, iacre_tsk, ista_tsk, ichg_pri, iget_pri, iref_tsk, iref_tst, isus_tsk, irsm_tsk, frsm_tsk, idef_tex, iref_tex, 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, iprcv_mbx, iref_mbx, icre_mbf, iacre_mbf, ipsnd_mbf, iref_mbf, icre_mpf, iacre_mpf, ipget_mpf, iref_mpf, icre_mpl, iacre_mpl, ipget_mpl, iref_mpl, iset_tim, iget_tim, icre_cyc, iacre_cyc, ista_cyc, istp_cyc, iref_cyc, iacre_alm, iacre_alm, ista_alm, istp_alm, iref_alm, ista_ovr, istp_ovr, iref_ovr, idef_inh, ichg_ims, iget_ims, idef_svc, ical_svc, idef_exc, iref_cfg, iref_ver

3.3 サービスコールの説明形式

以降の節では、サービスコールについての詳細を図 3.3の形式で説明します。

節番号　　　機能(サービスコール名)

■C 言語 API

サービスコール呼出し形式

←「3.2.2 アセンブリ言語 API」に従わないサービスコールについてのみアセンブラ API を併記します。

■パラメータ

型	パラメータ名	格納場所	意味
・	・	・	・
・	・	・	・
・	・	・	・

← 格納場所の"R4", "R5"などはレジスタ、"@R15"や"@(4,R15)"はスタックです。なお、"@(4,R15)"は R15+4 番地の内容を意味します。

■リターンパラメータ

型	パラメータ名	格納場所	意味
・	・	・	・
・	・	・	・
・	・	・	・

■リターンコード／エラーコード

ニモニック	[エラー種別]	意味
・	・	・
・	・	・
・	・	・

← エラー種別は以下の意味を持ちます。コンフィギュレータで CFG_PARCHK をチェックしない場合、[p]に分類されるエラーが発生する状況が生じた場合の動作は保証されません。

[k]　常に検出されるエラー
[p]　パラメータチェック機能付きのカーネル(コンフィギュレータで CFG_PARCHK をチェック)の場合のみ検出されるエラー

・パケットの構造

パケットを扱うサービスコールでは、パケット構造を以下のように表記します。

```
typedef struct t_rsem {
    ID      wtskid;          +0    2        待ちタスク ID
    UINT    semcnt;          +4    4        現在のセマフォカウント値
} T_RSEM;
```

↑ C 言語構造体　↑ パケット先頭からのオフセット　↑ メンバのサイズ　↑ メンバの説明

・属性の説明

属性の説明では、以下の記号を使用します。

[A] : A の指定を省略可能であることを示します。
(A || B) : A または B を選択することを示します。

図3.3　サービスコールの説明形式

3.4　タスク管理機能

表 3.4にタスク管理でサポートしているサービスコール一覧を示します。

表3.4　タスク管理サービスコール

項番	サービスコール　*1	機　能	呼び出し可能なシステム状態　*2						
			T	N	E	D	U	L	C
1	cre_tsk [s]	タスクの生成(ダイナミックスタック使用)	○		○	○	○		
	icre_tsk			○	○	○	○		
2	vscr_tsk [s]	タスクの生成(スタティックスタック使用)	○		○	○	○		
	ivscr_tsk			○	○	○	○		
3	acre_tsk	タスクの生成(ID 番号自動割付け)	○		○	○	○		
	iacre_tsk			○	○	○	○		
4	del_tsk	タスクの削除	○		○	○	○		
5	act_tsk [S]	タスクの起動	○		○	○	○		
	iact_tsk [S]			○	○	○	○		
6	can_act [S]	タスク起動要求のキャンセル	○		○	○	○		
	ican_act			○	○	○	○		
7	sta_tsk	タスクの起動(起動コード指定)	○		○	○	○		
	ista_tsk			○	○	○	○		
8	ext_tsk [S]	自タスクの終了	○		○	○	○	○	
9	exd_tsk [S]	自タスクの終了と削除	○		○	○	○	○	
10	ter_tsk [S]	タスクの強制終了	○		○	○	○		
11	chg_pri [S]	タスク優先度の変更	○		○	○	○		
	ichg_pri			○	○	○	○		
12	get_pri [S]	タスク優先度の参照	○		○	○	○		
	iget_pri			○	○	○	○		
13	ref_tsk	タスクの状態参照	○		○	○	○		
	iref_tsk			○	○	○	○		
14	ref_tst	タスクの状態参照(簡易版)	○		○	○	○		
	iref_tst			○	○	○	○		
15	vchg_tmd	タスク実行モードの変更	○		○	○	○		

【注】 *1　"[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2　それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.5にタスク管理の仕様を示します。

表3.5　タスク管理の仕様

項番	項目	内容
1	タスク ID	1～CFG_MAXTSKID (最大 1023)
2	タスク優先度	1～CFG_MAXTSKPRI(最大 255) *
3	タスク起動要求カウントの最大値	15 回
4	タスク属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述 TA_ACT : タスク生成後にタスクを実行可能状態へ [HI7000/4, HI7700/4] TA_COP0 : DSP を使用 [HI7000/4]　TA_COP1 : FPU を使用 [HI7750/4]　TA_COP1 : FPU バンク 0 を使用 TA_COP2 : FPU バンク 1 を使用

【注】　kernel_macro.h に定義される TMAX_TPRI と同じ値です。

3.4.1 タスクの生成
<ダイナミックスタック使用>

(cre_tsk, icre_tsk)

(acre_tsk, iacre_tsk：ID 番号自動割付け)

<スタティックスタック使用>

(vscr_tsk, ivcsr_tsk)

■C 言語 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);
ER ercd = vscr_tsk(ID tskid, T_CTSK *pk_ctsk);
ER ercd = ivscr_tsk(ID tskid, T_CTSK *pk_ctsk);
```

■パラメータ

《cre_tsk、icre_tsk、vscr_tsk、ivscr_tsk》

ID	tskid	R4	タスク ID
T_CTSK	*pk_ctsk	R5	タスク生成情報を格納したパケットへのポインタ

《acre_tsk、iacre_tsk》

T_CTSK	*pk_ctsk	R4	タスク生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_tsk、icre_tsk、vscr_tsk、ivscr_tsk》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_tsk、iacre_tsk》

ER_ID	tskid	R0	生成したタスク ID(正の値)またはエラーコード

■パケットの構造

```
typedef   struct  t_ctsk {
          ATR     tskatr;      0    4   タスク属性
          VP_INT  exinf;      +4    4   拡張情報
          FP      task;       +8    4   タスク起動アドレス
          PRI     itskpri;    +12   2   初期タスク優先度
          SIZE    stksz;      +16   4   タスクスタックサイズ
          VP      stk;        +20   4   タスクスタック領域の先頭アドレス
} T_CTSK;
```

■エラーコード

E_NOMEM	[k]	メモリ不足(タスクスタック領域を確保できない)
E_RSATR	[p]	予約属性(tskatr が不正)
E_PAR	[p]	パラメータエラー(pk_ctsk が 4 の倍数以外、task が奇数、stksz が 4 の倍数以外、stksz=0、stksz≧0x80000000、itskpri≦0、itskpri＞CFG_MAXTSKPRI, stk が NULL 以外で 4 の倍数以外)
E_ID	[p]	不正 ID 番号(tskid≦0、tskid＞CFG_MAXTSKID(cre_tsk, icre_tsk)、tskid≦CFG_STSTKID(cre_tsk, icre_tsk)、tskid＞CFG_STSTKID(vscr_tsk, ivscr_tsk)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態でない、または自タスク指定)
E_NOID	[k]	空き ID なし

■機能

cre_tsk, icre_tsk, acre_tsk, iacre_tsk サービスコールはダイナミックスタックを使用するタスクを、vscr_tsk, ivscr_tsk サービスコールはスタティックスタックを使用するタスクを生成します。生成したタスクは、TA_ACT 属性の指定がない場合は休止状態へ、TA_ACT 属性の指定がある場合は実行可能状態に移行します。

タスク生成時に行われる処理は、表 3.6の通りです。

表3.6　タスク生成時に行われる処理

項番	処理内容
1	タスク起動要求キューイング数をクリアする。
2	タスク例外処理ルーチンを定義されていない状態にする。
3	上限プロセッサ時間が設定されていない状態にする。
4	スタックを割り付ける。（cre_tsk, acre_tsk の場合）

以下に各パラメータの意味を示します。

(1)　tskid

vscr_tsk, ivscr_tsk サービスコールの場合、スタティックスタックを使用するタスクを生成します。tskid には 1 ～CFG_STSTKID の値を指定します。

cre_tsk, icre_tsk サービスコールの場合、ダイナミックスタックを使用するタスクを生成します。tskid には CFG_STSTKID+1～CFG_MAXTSKID の値を指定します。

acre_tsk, iacre_tsk サービスコールの場合、未登録のタスク ID を検索してその ID を持つタスクを pk_ctsk で示された内容で生成し、その ID をリターンパラメータとして返します。acre_tsk, iacre_tsk サービスコールで生成するのは、ダイナミックスタック使用タスクです。検索するタスク ID 範囲は、CFG_STSTKID+1～CFG_MAXTSKID となります。

(2)　tskatr

tskatr には属性として、タスクを記述した言語およびコプロセッサの使用を指定します。

```
tskatr := ( (TA_HLNG || TA_ASM) [ |TA_ACT] [ |TA_COP0] [ |TA_COP1] [ |TA_COP2] )
```

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述
- TA_ACT（H'00000002）タスク生成後にタスクを実行可能状態へ

- TA_COP0（H'00000100）DSP を使用（HI7000/4, HI7700/4）
- TA_COP1（H'00000200）FPU を使用（HI7000/4）、FPU のバンク 0 を使用（HI7750/4）*
- TA_COP2（H'00000400）FPU のバンク 1 を使用（HI7750/4）*

* FPSCR の初期値は、H’00040001(バンク 0)です。

TA_ACT 属性を指定した場合、対象タスクには拡張情報(exinf)がパラメータとして渡ります。

TA_COPn 属性の指定により、当該コプロセッサのレジスタもタスクコンテキストとして保存されるようになります。なお、TA_COPn 属性はμITRON4.0 仕様の範囲外の属性です。

(3) exinf

exinf は、ユーザが生成するタスクに関する情報を設定するなどの目的で自由に使用できます。

(4) task

task には、タスクの起動アドレスを指定します。

(5) itskpri

itskpri には、タスク起動時の優先度の初期値として 1～CFG_MAXTSKPRI の値を指定します。

(6) stksz

stksz は、cre_tsk, icre_tsk, acre_tsk, iacre_tsk サービスコールでのみ有効なパラメータで、生成するタスクのスタックサイズを指定します。スタックサイズには 4 の倍数の値を指定します。

スタティックスタックを使用するタスクを生成する vscr_tsk, ivscr_tsk では、stksz は意味を持ちません。カーネルの処理では無視されます。

(7) stk

stk は、cre_tsk, icre_tsk, acre_tsk, iacre_tsk サービスコールでのみ有効なパラメータです。

stk に NULL を指定した場合は、スタックはコンフィギュレータで指定したダイナミックスタック領域(CFG_TSKSTKSZ)から割り付けられます。生成に成功すると、ダイナミックスタック領域の空きは以下の式で計算されるサイズだけ減少します。

- 減少サイズ ＝ stksz＋16

生成するタスクのスタックアドレスを指定することもできます。この場合、stksz で指定したサイズのスタック領域を確保し、その先頭アドレスを指定してください。

なお、vscr_tsk, ivscr_tsk サービスコールは HI7000/4 シリーズの独自機能です。

3.4.2 タスクの削除(del_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid≦0、tskid＞CFG_MAXTSKID)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態でない、または自タスク指定)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

tskid で示されたタスクを削除します。削除されたタスクは、未登録状態へ移行します。

tskid で示されたタスクがダイナミックスタックを使用していた場合は、そのタスクのスタックは解放され、ダイナミックスタック領域の空きは以下の式で算出されるサイズだけ増加します。

- 増加サイズ ＝ (生成時に指定した stksz）＋16

3.4.3 タスクの起動(act_tsk, iact_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_QOVR	[k]	キューイングのオーバフロー(actcnt＞15)

■機能

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。タスク起動時に行われる処理は、表 3.7の通りです。

表3.7　タスク起動時に行われる処理

項番	処理内容
1	タスクのベース優先度と現在優先度をを初期化する。
2	起床要求キューイング数をクリアする。
3	強制待ち要求ネスト数をクリアする。
4	保留例外要因をクリアする。
5	タスク例外処理禁止状態にする。
6	タスク付属イベントフラグのフラグパターンをクリアする。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

対象タスクには、タスク生成時に指定したタスクの拡張情報がパラメータとして渡ります。

tskid で示されたタスクがスタティックスタックを使用するタスク(tskid が 1～CFG_STSTKID)で、そのスタックが解放（どのタスクも使用していない）されている場合は、tskid で示されたタスクが共有スタックを占有し、実行可能状態に移行します。そのスタックを他のタスクが使用している場合は、スタック領域を使用できないために tskid で示されたタスクは共有スタック待ち状態になり、共有スタックの待ち行列につながれます。共有スタックの待ち行列は、FIFO で管理されます。

対象タスクが休止状態でない場合には、本サービスコールによるタスクの起動要求は、最大 15 回まで記憶されます。

3.4.4 タスクの起動要求のキャンセル(can_act, ican_act)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER_UINT	actcnt	R0	キューイングされていた起動要求の回数(正の値または 0) またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid<0、tskid>CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが生成されていない)

■機能

tskid で示されたタスクにキューイングされていた起動要求回数を求め、その結果をリターンパラメータとして返し、同時にその起動要求を全て無効にします。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

休止状態のタスクを対象として呼び出すこともできます。その場合のリターンパラメータは 0 となります。

3.4.5 タスクの起動(起動コード指定)(sta_tsk, ista_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID
VP_INT	stacd	R5	タスク起動コード

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid≦0、tskid＞CFG_MAXTSKID)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態でない、または自タスク指定)

■機能

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。この時、タスク起動時に行うべき処理(表 3.7参照)を行います。

起動したタスクには、パラメータとして stacd で示されたタスク起動コードが渡されます。

tskid で示されたタスクがスタティックスタックを使用するタスクで、そのスタックが解放（どのタスクも使用していない）されている場合は、tskid で示されたタスクが共有スタックを占有し、実行可能状態に移行します。そのスタックを他のタスクが使用している場合は、スタック領域を使用できないために tskid で示されたタスクは共有スタック待ち状態になり、共有スタックの待ち行列につながれます。共有スタックの待ち行列は、FIFO で管理されます。

3.4.6 自タスクの終了(ext_tsk) 自タスクの終了と削除(exd_tsk)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

サービスコールの呼び出し元には戻りません。

ただし、本サービスコールをシステム構築時に組み込まずに呼び出した場合には、R0 に E_RSFN が設定されてリターンします。

また、以下のエラーが発生するとシステムダウンとなります。

E_CTX　[k]　コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

ext_tsk サービスコールは、自タスクを正常終了します。タスクの状態は、実行状態から休止状態へ移行します。起動要求がキューイングされている場合は、自タスクをいったん終了させた後に再起動します。

タスク終了時に行われる処理は、表 3.8の通りです。

表3.8　タスク終了時に行われる処理

項番	処理内容
1	タスクがロックしていたミューテックスをロック解除する。
2	上限プロセッサ時間を解除する。

exd_tsk サービスコールは、自タスクを正常終了し、さらに削除します。タスクの状態は、実行状態から未登録状態へ移行します。

ext_tsk, exd_tsk サービスコールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

ext_tsk, exd_tsk サービスコールを呼び出したタスクが他のタスクとスタックを共有しており、そのスタックの待ち行列に他のタスクがつながれている場合、先頭のタスクをスタックの待ち行列から外し、スタック待ち状態を解除します。その際、スタック待ち解除されたタスクに関して、タスクを起動する際に行うべき処理を行い(表 3.7参照)、そのタスクは実行可能状態に移行します。

ダイナミックスタックを使用しているタスクが exd_tsk サービスコールを呼び出した場合、そのタスクのスタックは解放され、ダイナミックスタック領域の空きは以下の式で算出されるサイズだけ増加します。

- 増加サイズ ＝ (生成時に指定した stksz) ＋16

ext_tsk, exd_tsk サービスコールは、ディスパッチ禁止状態および CPU ロック状態からも呼び出せます。この場合、ディスパッチ禁止状態および CPU ロック状態は解除されます。

なお、タスク開始関数からのリターンした場合は、ext_tsk サービスコールと同じ動作となります。

3.4.7 タスクの強制終了(ter_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid≦0、tskid＞CFG_MAXTSKID)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態)
E_ILUSE	[k]	サービスコール不正使用(対象タスクが自タスク)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

tskid で示された他タスクを強制的に終了させます。終了させた他タスクは休止状態へ移行します。この時、表 3.8に示す処理が行われます。

起動要求がキューイングされている場合には、タスクを起動する際に行うべき処理を行い、対象タスクを実行可能状態に移行します。

本サービスコールによる強制終了要求は、次の場合には遅延されます。

- tskid で示されたタスクが vchg_tmd サービスコールにより強制終了をマスクしている場合

本サービスコールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

tskid で示されたタスクが他のタスクとスタックを共有しており、そのスタックの待ち行列に他のタスクがつながれている場合、先頭のタスクをスタックの待ち行列から外し、スタック待ち状態を解除します。その際、スタック待ち解除されたタスクに関して、タスクを起動する際に行うべき処理を行い(表 3.7参照)、そのタスクは実行可能状態に移行します。

3.4.8 タスク優先度の変更(chg_pri, ichg_pri)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID
PRI	tskpri	R5	タスクのベース優先度

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tskpri＜0、tskpri＞CFG_MAXTSKPRI)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_ILUSE	[k]	サービスコール不正使用(上限優先度の違反)
E_OBJ	[k]	オブジェクト状態不正(タスクが休止状態である)

■機能

tskid で示されたタスクのベース優先度を、tskpri で示された値に変更します。それに伴い、タスクの現在優先度も変更します。tskid=TSK_SELF（0）の指定により、自タスク指定となります。

tskpri=TPRI_INI（0）の指定により、タスク生成時に指定した初期タスク優先度に戻します。

変更したタスク優先度は、タスクが終了、または本サービスコールを呼び出すまで有効です。タスクが休止状態になると終了前のタスク優先度は無効になり、次に起動されたときにはタスク生成時に指定した初期タスク優先度になります。

tskid で示されたタスクが何らかの待ち状態で待ちのオブジェクトの属性が TA_TPRI の場合、本サービスコールによって待ち行列が変更され、その結果待ち行列につながれていたタスクの待ち状態が解除されることがあります。

対象タスクが TA_CEILING 属性のミューテックスをロックしているかロックを待っている場合で、tskpri に指定されたベース優先度が、それらのミューテックスのいずれかの上限優先度よりも高い場合には、E_ILUSE を返します。

3.4.9 タスク優先度の参照(get_pri, iget_pri)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID
PRI	*p_tskpri	R5	対象タスクの現在優先度を返す領域へのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
PRI	*p_tskpri	R5	対象タスクの現在優先度へのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_tskpri が 2 の倍数で無い)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(タスクが休止状態である)

■機能

tskid で示されたタスクの現在優先度を参照し、p_tskpri の指す領域に返します。
tskid=TSK_SELF（0）の指定により、自タスク指定となります。

3.4.10 タスクの状態参照(ref_tsk, iref_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID
T_RTSK	*pk_rtsk	R5	タスク状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RTSK	*pk_rtsk	R5	タスク状態を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct  t_rtsk {
          STAT    tskstat;    +0     4    タスク状態
          PRI     tskpri;     +4     2    タスクの現在優先度
          PRI     tskbpri;    +6     2    タスクのベース優先度
          STAT    tskwait;    +8     4    待ち要因
          ID      wobjid;     +12    2    待ちオブジェクト ID
          TMO     lefttmo;    +16    4    タイムアウトするまでの時間
          UINT    actcnt;     +20    4    起動要求キューイング数
          UINT    wupcnt;     +24    4    起床要求キューイング数
          UINT    suscnt;     +28    4    強制待ち要求ネスト数
          UINT    tskmode;    +32    4    タスクの実行モード
          UINT    tflptn;     +36    4    現在のタスク付属イベントフラグの値
} T_RTSK;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtsk が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)

■機能

tskid で示されたタスクの状態を参照し、pk_rtsk が指す領域に返します。
tskid=TSK_SELF（0）の指定により自タスクの指定になります。

pk_rtsk の指す領域には、以下の値を返します。なお、*のデータはタスクが休止状態の場合は無効です。また、機能が組み込まれていないものに対する情報は不定となります。

♦ tskstat

現在のタスクの状態です。tskstat には、次の値を返します。

- TTS_RUN（H'00000001）実行状態
- TTS_RDY（H'00000002）実行可能状態
- TTS_WAI（H'00000004）待ち状態
- TTS_SUS（H'00000008）強制待ち状態
- TTS_WAS（H'0000000c）二重待ち状態
- TTS_DMT（H'00000010）休止状態
- TTS_STK（H'40000000）共有スタック解放待ち状態

♦ tskpri

タスクの現在優先度です。タスクが休止状態の場合は、タスクの初期優先度を返します。

♦ tskbpri

タスクのベース優先度です。タスクが休止状態の場合は、タスクの初期優先度を返します。

♦ tskwait *

tskstat が TTS_WAI、TTS_WAS のときに有効で、次の値を返します。

- TTW_SLP（H'00000001）slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY（H'00000002）dly_tsk サービスコールによる待ち
- TTW_SEM（H'00000004）wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG（H'00000008）wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ（H'00000010）snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ（H'00000020）rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX（H'00000040）rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX（H'00000080）loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_SMBF（H'00000100）snd_mbf、tsnd_mbf サービスコールによる待ち
- TTW_RMBF（H'00000200）rcv_mbf、trcv_mbf サービスコールによる待ち
- TTW_MPF（H'00002000）get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL（H'00004000）get_mpl、tget_mpl サービスコールによる待ち
- TTW_TFL（H'00008000）vwai_tfl、vtwai_tfl サービスコールによる待ち

♦ wobjid *

tskstat が TTS_WAI、TTS_WAS のときに有効で、待ち対象のオブジェクト ID を返します。

♦ lefttmo *

対象タスクがタイムアウトするまでの時間を返します。対象タスクが dly_tsk サービスコールによる待ち状態の場合は、この値は不定値となります。

♦ actcnt *

現在の起動要求キューイング数を返します。

♦ wupcnt *

現在の起床要求キューイング数を返します。

♦ suscnt *

現在の強制待ち要求ネスト数を返します。

♦ tskmode *

vchg_tmd サービスコールで設定したタスク実行モードと、それによって遅延されている要求があるかを返します。tskmode には次の値を返します。

- ECM_SUS（H'00000001）強制待ち要求がマスクされている
- ECM_TER（H'00000002）強制終了要求がマスクされている
- PND_SUS（H'00000004）強制待ち要求が遅延されている
- PND_TER（H'00000008）強制終了要求が遅延されている

♦ tflptn *

現在のタスク付属イベントフラグの値を返します。ただし、タスク付属イベントフラグ機能を組み込んでいない場合には、この値は不定値を返します。

tskmode および tflptn は、μITRON4.0 仕様の範囲外のメンバです。

RENESAS

3.4.11 タスクの状態参照(簡易版)(ref_tst, iref_tst)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID
T_RTST	*pk_rtst	R5	タスク状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RTST	*pk_rtst	R5	タスク状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rtst {
           STAT    tskstat;     +0     4    タスク状態
           STAT    tskwait;     +4     4    待ち要因
} T_RTST;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtsk が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)

■機能

tskid で示されたタスクについて、タスク状態と待ち要因を参照し、pk_rtst が指す領域に返します。tskid=TSK_SELF（0）の指定により自タスクの指定になります。

pk_rtst の指す領域には、以下の値を返します。なお、*のデータはタスクが休止状態の場合は無効です。また、機能が組み込まれていないものに対する情報参照値は不定となります。

♦ tskstat

現在のタスクの状態です。tskstat には、次の値を返します。

- TTS_RUN（H'00000001）実行状態
- TTS_RDY（H'00000002）実行可能状態
- TTS_WAI（H'00000004）待ち状態
- TTS_SUS（H'00000008）強制待ち状態
- TTS_WAS（H'0000000c）二重待ち状態
- TTS_DMT（H'00000010）休止状態
- TTS_STK（H'40000000）共有スタック解放待ち状態

♦ tskwait *

tskstat が TTS_WAI、TTS_WAS のときに有効で、次の値を返します。

- TTW_SLP（H'00000001）slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY（H'00000002）dly_tsk サービスコールによる待ち
- TTW_SEM（H'00000004）wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG（H'00000008）wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ（H'00000010）snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ（H'00000020）rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX（H'00000040）rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX（H'00000080）loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_SMBF（H'00000100）snd_mbf、tsnd_mbf サービスコールによる待ち
- TTW_RMBF（H'00000200）rcv_mbf、trcv_mbf サービスコールによる待ち
- TTW_MPF（H'00002000）get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL（H'00004000）get_mpl、tget_mpl サービスコールによる待ち
- TTW_TFL（H'00008000）vwai_tfl、vtwai_tfl サービスコールによる待ち

3.4.12　タスク実行モードの変更(vchg_tmd)

■C 言語 API

```
ER ercd = vchg_tmd(UINT tmd);
```

■パラメータ

UINT	tmd	R4	変更するタスク実行モード

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tmd が不正)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

自タスクのタスク実行モードを変更します。tmd にはタスク実行モードとして他タスクからの要求に対するマスクを設定できます。

- ECM_SUS（H'00000001）強制待ち要求マスクのセット
- ECM_TER（H'00000002）強制終了要求マスクのセット

強制待ち要求マスクをセットすると、sus_tsk, isus_tsk サービスコールが呼び出されても、vchg_tmd サービスコールによってマスクを解除（tmd=0 を指定）するまでその要求は遅延されます。

強制終了要求マスクをセットすると、ter_tsk サービスコールが呼び出されても、vchg_tmd サービスコールによってマスクを解除（tmd=0 を指定）するまでその要求は遅延されます。

タスク実行モードは、タスクコンテキストとして動作する拡張サービスコールルーチン、タスク例外処理ルーチンでは、呼び出し元タスクの状態を引き継ぎます。

強制待ち要求、強制終了要求の遅延状況は、ref_tsk, iref_tsk サービスコールで参照できます。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.5 タスク付属同期機能

表 3.9にタスク付属同期でサポートしているサービスコール一覧を示します。

表3.9 タスク付属同期サービスコール

項番	サービスコール *1		機能	呼び出し可能な状態 *2						
				T	N	E	D	U	L	C
1	slp_tsk	[S]	起床待ち	○		○		○		
2	tslp_tsk	[S]	同上(タイムアウト有)	○		○		○		
3	wup_tsk	[S]	タスクの起床	○		○	○	○		
	iwup_tsk	[S]			○	○	○	○		
4	can_wup	[S]	タスク起床要求のキャンセル	○		○	○	○		
	ican_wup				○	○	○	○		
5	rel_wai	[S]	待ち状態の強制解除	○		○	○	○		
	irel_wai	[S]			○	○	○	○		
6	sus_tsk	[S]	強制待ち状態への移行	○		○	○	○		
	isus_tsk				○	○	○	○		
7	rsm_tsk	[S]	強制待ち状態からの再開	○		○	○	○		
	irsm_tsk				○	○	○	○		
8	frsm_tsk	[S]	強制待ち状態からの強制再開	○		○	○	○		
	ifrsm_tsk				○	○	○	○		
9	dly_tsk	[S]	自タスクの遅延	○		○		○		
10	vset_tfl		タスク付属イベントフラグのセット	○		○	○	○		
	ivset_ftl				○	○	○	○		
11	vclr_tfl		タスク付属イベントフラグのクリア	○		○	○	○		
	ivclr_tfl				○	○	○	○		
12	vwai_tfl		タスク付属イベントフラグ待ち	○		○		○		
13	vpol_tfl		同上(ポーリング)	○		○	○	○		
14	vtwai_tfl		同上(タイムアウト有)	○		○		○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.10にタスク付属同期の仕様を示します。

表3.10 タスク付属同期の仕様

項番	項目	内容
1	タスク起床要求カウントの最大値	15 回
2	タスク強制待ち要求ネスト数の最大値	15 回
3	タスク付属イベントフラグビット数	32 ビット(下位 16 ビットは将来拡張用)
4	タスク付属イベントフラグ初期値	タスク起動時に 0 に初期化される
5	タスク付属イベントフラグ待ち条件	OR 待ち

3.5.1 起床待ち(slp_tsk, tslp_tsk)

■C 言語 API

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

■パラメータ

《tslp_tsk》

TMO	tmout	R4	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tmout≦-2)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_TMOUT	[k]	タイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

自タスクを起床待ち状態に移行させせます。ただし，自タスクに対する起床要求がキューイングされている場合は、起床要求カウントを 1 減らしてそのまま実行を継続します。

起床待ち状態は、wup_tsk, iwup_tsk サービスコールによって解除されます。

tslp_tsk サービスコールでは、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち状態のまま tmout 時間が経過すると、待ち状態は解除され、エラーコードとして E_TMOUT が返ります。

tmout=TMO_POL（0）を指定した場合、起床要求カウントが正なら起床要求カウントを 1 減らして実行を継続し、0 ならエラーコードとして E_TMOUT を返します。

tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。この場合、slp_tsk サービスコールと同じ動作となります。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は､tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

3.5.2 タスクの起床(wup_tsk, iwup_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid ＜ 0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である)
E_QOVR	[k]	キューイングのオーバフロー(wupcnt＞15)

■機能

slp_tsk、または tslp_tsk サービスコールの呼び出しにより待ち状態になっているタスクの待ち状態を解除します。

対象タスクが slp_tsk、または tslp_tsk サービスコールによる待ち状態でない場合には、本サービスコールによる起床要求は、最大 15 回まで記憶されます。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

3.5.3 タスク起床要求のキャンセル(can_wup, ican_wup)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER_UINT	wupcnt	R0	キューイングされていた起床要求の回数(正の値または 0) またはエラーコード

■エラーコード

E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが生成されていない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である)

■機能

tskid で示されたタスクにキューイングされていた起床要求回数を求め、その結果をリターンパラメータとして返し、同時にその起床要求を全て無効にします。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

3.5.4　待ち状態の強制解除(rel_wai, irel_wai)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid≦0、tskid＞CFG_MAXTSKID)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正（tskid のタスクが待ち状態でない、または自タスク指定）

■機能

tskid で示されるタスクが何らかの待ち状態（強制待ち状態および共有スタック解放待ち状態は含まれません）の場合、それを強制的に解除します。本サービスコールにより待ち状態を解除したタスクには、エラーコードとして E_RLWAI が返ります。

二重待ち状態のタスクに対して本サービスコールを呼び出すと、対象タスクは強制待ち状態へ移行します。その後 rsm_tsk, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールが呼び出され、強制待ち状態が解除されると、対象タスクにはエラーコードとして E_RLWAI が返されます。

なお、強制待ち状態を解除するには、rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk を使用してください。また、共有スタック待ち状態を解除するサービスコールはありません。

3.5.5 強制待ち状態への移行(sus_tsk, isus_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である)
E_CTX	[k]	コンテキストエラー(タスクコンテキストかつディスパッチ禁止状態で tskid=TSK_SELF(0)または自タスク ID を指定)
E_QOVR	[k]	キューイングのオーバフロー(suscnt＞15)

■機能

tskid で示されたタスクの実行を中断させ、強制待ち状態へ移行します。tskid で示されたタスクが待ち状態にある場合は、二重待ち状態へ移行します。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

強制待ち状態は、rsm_tsk, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールの呼び出しにより解除されます。

本サービスコールによる強制待ちの要求はネストします。強制待ち要求のネスト数は最大 15 回まで記憶されます。

本サービスコールによる強制待ちの要求は、次の場合には遅延されます。

(1) tskidで示されたタスクがvchg_tmdサービスコールにより、強制待ち要求をマスクしている場合は、vchg_tmdにより強制待ちを解除（tmd=0を指定）した時点で強制待ち状態に移行します。

(2) tskidで示されたタスクがdis_dspサービスコールを呼び出してシステム状態がディスパッチ禁止状態になっていた場合は、タスク実行状態に戻った時点で強制待ち状態に移行します。

遅延されている強制待ち要求も、rsm_tsk, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールの呼び出しによってその要求を解除できます。したがって強制待ち状態への移行は、遅延解除時の強制待ちの要求ネスト数が 0 でない場合に行います。

3.5.6 強制待ち状態からの再開(rsm_tsk, irsm_tsk) 強制待ち状態からの強制再開(frsm_tsk, ifrsm_tsk)

■C 言語 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);
```

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid<0、tskid>CFG_MAXTSKID)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクトの状態不正(tskid のタスクが休止状態である、tskid のタスクが強制待ち状態でない、または自タスク指定)

■機能

tskid で示されたタスクの強制待ち状態を解除します。

具体的には、rsm_tsk, irsm_tsk サービスコールは、tskid で示されたタスクの強制待ち要求ネスト数を 1 減算し、その結果が 0 になった場合に強制待ち状態を解除します。frsm_tsk, ifrsm_tsk サービスコールは、強制待ち要求ネスト数を 0 にし、強制待ち状態を解除します。二重待ち状態の場合は、待ち状態に移行させます。

3.5.7 タスク遅延(dly_tsk)

■C 言語 API

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

■パラメータ

RELTIM	dlytim	R4	遅延時間

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

自タスクの状態を実行状態から時間経過待ち状態へ移行し、dlytim で指定された時間が経過するのを待ちます。dlytim で指定された時間が経過した時点で、自タスクの状態を実行可能状態に移行します。dlytim＝0 を指定した場合にも、自タスクを待ち状態に移行させます。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、dlytim に指定可能な最大値は H'ffffffff/CFG_TICDENO となります。これより大きな値を指定した場合の動作は保証されません。

本サービスコールは、tslp_tsk サービスコールとは異なり、dlytim 時間だけ実行を遅延して終了した場合に正常終了します。また、遅延時間中に wup_tsk, iwup_tsk サービスコールが実行されても、待ち状態は解除されません。遅延時間が経過する前に待ち状態を解除するのは、rel_wai, irel_wai または ter_tsk サービスコールが呼び出された場合に限られます。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.5.8 タスク付属イベントフラグのセット(vset_tfl, ivset_tfl)

■C 言語 API

```
ER ercd = vset_tfl(ID tskid, UINT setptn);
ER ercd = ivset_tfl(ID tskid, UINT setptn);
```

■パラメータ

ID	tskid	R4	タスク ID
UINT	setptn	R5	セットするビットパターン

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが生成されていない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である)

■機能

tskid で示されたタスクのタスク付属イベントフラグを、setptn で示された値との論理和（OR）に更新します。ただし、タスク付属イベントフラグの下位 16 ビットは将来拡張用ですので、setptn の下位 16 ビットは 0 としてください。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

タスク付属イベントフラグ値の更新の結果、対象タスクの待ちビットパターンとの論理和が 0 以外になると、そのタスクの待ち状態を解除します。このとき、タスク付属イベントフラグは 0 クリアされます。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.5.9 タスク付属イベントフラグのクリア(vclr_tfl, ivclr_tfl)

■C 言語 API

```
ER ercd = vclr_tfl(ID tskid, UINT clrptn);
ER ercd = ivclr_tfl(ID tskid, UINT clrptn);
```

■パラメータ

ID	tskid	R4	タスク ID
UINT	clrptn	R5	クリアするビットパターン

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストでtskid=TSK_SELF(0)を指定)
E_NOEXS	[p]	未登録(tskid のタスクが生成されていない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である)

■機能

tskid で示されたタスクのタスク付属イベントフラグを、clrptn との論理積（AND）に更新します。ただし、タスク付属イベントフラグの下位 16 ビットは将来拡張用ですので、clrptn の下位 16 ビットは H'ffff としてください。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.5.10 タスク付属イベントフラグ待ち(vwai_tfl, vpol_tfl, vtwai_tfl)

■C 言語 API

```
ER ercd = vwai_tfl(UINT waiptn, UINT *p_tflptn);
ER ercd = vpol_tfl(UINT waiptn, UINT *p_tflptn);
ER ercd = vtwai_tfl(UINT waiptn, UINT *p_tflptn, TMO tmout);
```

■パラメータ

UINT	waiptn	R4	待ちビットパターン
UINT	*p_tflptn	R5	待ち解除時のビットパターンを返す領域へのポインタ
《vtwai_tfl》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
UINT	*p_tflptn	R5	待ち解除時のビットパターンへのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_tflptn が 4 の倍数以外、waiptn=0、tmout≦-2)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_TMOUT	[k]	タイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

タスク付属イベントフラグの waiptn で示されたビットのいずれかがセットされるのを待ちます。p_tflptn の指す領域には、待ち解除される時のタスク付属イベントフラグのビットパターンを返します。また、待ち解除されるときには、タスク付属イベントフラグは 0 にクリアされます。

本サービスコール呼び出し時にすでに waiptn で示されたビットのいずれかがセットされていた場合は、本サービスコールは直ちに終了します。いずれのビットもセットされていない場合は、vwai_tfl, vtwai_tfl サービスコールの場合は待ち状態に移行し、vpol_tfl サービスコールの場合は直ちにエラーE_TMOUT で終了します。待ち状態となった場合は、その後 vset_tfl サービスコールによって待っているビットがセットされたときに待ち状態が解除されます。

なお、タスク起動時のタスク付属イベントフラグの値は 0 です。

vtwai_tfl サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が成立しないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、vpol_tfl サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、vwai_tfl サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.6 タスク例外処理機能

表 3.11にタスク例外処理でサポートしているサービスコール一覧を示します。

表3.11 タスク例外処理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_tex [s]	タスク例外処理ルーチンの定義	○		○	○	○		
	idef_tex			○	○	○	○		
2	ras_tex [S]	タスク例外処理の要求	○		○	○	○		○
	iras_tex [S]			○	○	○	○		○
3	dis_tex [S]	タスク例外処理の禁止	○		○	○	○		
4	ena_tex [S]	タスク例外処理の許可	○		○	○	○		
5	sns_tex [S]	タスク例外処理禁止状態の参照	○	○	○	○	○	○	○
6	ref_tex	タスク例外処理の状態参照	○		○	○	○		
	iref_tex			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.12にタスク例外の仕様を示します。

表3.12 タスク例外の仕様

項番	項目	内容
1	例外要因	32 ビット
2	タスク起動時の状態	・タスク例外処理禁止状態 ・保留例外要因なし
3	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述 [HI7000/4, HI7700/4] TA_COP0 : DSP を使用 [HI7000/4] TA_COP1 : FPU を使用 [HI7750/4] TA_COP1 : FPU バンク 0 を使用 TA_COP2 : FPU バンク 1 を使用

タスク例外処理ルーチンは、以下に示す条件が揃ったときに、タスクコンテキストとして起動されます。

- タスク例外処理許可状態
- 保留例外要因が 0 以外
- タスク実行状態
- 非タスクコンテキストまたは CPU 例外ハンドラが実行されていない

タスク例外処理ルーチンからリターンすると、タスク例外処理ルーチンを起動する前に実行していた処理を継続します。この時、このタスクはタスク例外許可状態に移行します。ここで、保留例外要因が 0 でない場合には、再びタスク例外処理ルーチンが起動されます。

3.6.1 タスク例外処理ルーチンの定義(def_tex, idef_tex)

■C 言語 API

```
ER ercd = def_tex(ID tskid, T_DTEX *pk_dtex);
ER ercd = idef_tex(ID tskid, T_DTEX *pk_dtex);
```

■パラメータ

ID	tskid	R4	タスク ID
T_DTEX	*pk_dtex	R5	タスク例外処理ルーチン定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef    struct  t_dtex {
           ATR     texatr;     0     4    タスク例外処理ルーチン属性
           FP      texrtn;    +4     4    タスク例外処理ルーチン起動アドレス
} T_DTEX;
```

■エラーコード

E_RSATR	[p]	予約属性(texatr が不正)
E_PAR	[p]	パラメータエラー(pk_dtex が 4 の倍数以外、texrtn が奇数)
E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)

■機能

tskid で示されたタスク例外処理ルーチンを pk_dtex で示された内容で定義します。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

texatr には属性として、タスク例外処理ルーチンを記述した言語を指定します。

```
texatr := ( (TA_HLNG || TA_ASM) [ |TA_COP0] [ |TA_COP1] [ |TA_COP2] )
```

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述
- TA_COP0（H'00000100）DSP を使用（HI7000/4, HI7700/4）
- TA_COP1（H'00000200）FPU を使用（HI7000/4）、FPU のバンク 0 を使用（HI7750/4）*
- TA_COP2（H'00000400）FPU のバンク 1 を使用（HI7750/4）*

* FPSCR の初期値は、H'00040001(バンク 0)です。

TA_COPn 属性の指定により、当該コプロセッサのレジスタもタスク例外処理ルーチンのコンテキストとして保存されるようになります。なお、TA_COPn 属性はμITRON4.0 仕様の範囲外の属性です。

texrtn には、タスク例外処理ルーチンの先頭アドレスを指定します。

def_tex, idef_tex サービスコールでは、pk_dtex＝NULL（0）と指定した場合には tskid のタスク例外処理ルーチンの定義を解除します。この時、タスクの保留例外要因を 0 クリアし、タスクをタスク例外処理禁止状態に移行します。

すでにタスク例外処理ルーチンが定義されていた場合は、以前の定義を解除し新しい定義に置き換えます。この時、保留例外要因のクリアとタスク例外処理の禁止は行いません。

3.6.2 タスク例外処理の要求(ras_tex, iras_tex)

■C 言語 API

```
ER ercd = ras_tex(ID tskid, TEXPTN rasptn);
ER ercd = iras_tex(ID tskid, TEXPTN rasptn);
```

■パラメータ

ID	tskid	R4	タスク ID
TEXPTN	rasptn	R5	要求するタスク例外処理のタスク例外要因

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(rasptn=0)
E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である、またはタスク例外処理ルーチンが定義されていない)

■機能

tskid で示されたタスクに対して、rasptn で指定されるタスク例外要因によって、タスク例外処理を要求します。つまり、対象タスクの保留例外要因を、rasptn で示された値との論理和（OR）に更新します。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

このサービスコールにより、タスク例外処理ルーチンを起動する条件が揃った場合には、タスク例外処理ルーチンを起動する処理を行います。

本サービスコールは、CPU 例外ハンドラからも呼び出せます。

RENESAS

3.6.3　タスク例外処理の禁止(dis_tex)

■C 言語 API

```
ER ercd = dis_tex(void);
```

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_OBJ	[k]	オブジェクト状態不正(自タスクにタスク例外処理ルーチンが定義されていない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

自タスクをタスク例外処理禁止状態に移行させます。

3.6.4 タスク例外処理の許可(ena_tex)

■C 言語 API

```
ER ercd = ena_tex(void);
```

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_OBJ	[k]	オブジェクト状態不正(自タスクにタスク例外処理ルーチンが定義されていない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

自タスクを、タスク例外許可状態に移行させます。

このサービスコールにより、タスク例外処理ルーチンを起動する条件が揃った場合には、タスク例外処理ルーチンを起動する処理を行います。

3.6.5 タスク例外禁止状態の参照(sns_tex)

■C 言語 API

```
BOOL state = sns_tex(void);
```

■パラメータ

なし

■リターンパラメータ

BOOL	state	R0	タスク例外禁止状態

■エラーコード

なし

■機能

実行状態のタスクが、タスク例外禁止状態の場合に TRUE、タスク例外許可状態の場合に FALSE を返します。実行状態のタスクとは、タスクコンテキストから呼び出した場合は自タスクであり、非タスクコンテキストから呼び出した場合は非タスクコンテキストに移行する直前に実行していたタスクです。非タスクコンテキストから呼び出された場合で、実行状態のタスクがないときには TRUE を返します。

タスク例外処理ルーチンが定義されていないタスクは、タスク例外処理禁止状態に保たれているため、実行状態のタスクにタスク例外処理ルーチンが定義されていない場合には、このサービスコールは TRUE を返します。

本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

3.6.6 タスク例外処理の状態参照(ref_tex, iref_tex)

■C 言語 API

```
ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex);
ER ercd = iref_tex(ID tskid, T_RTEX *pk_rtex);
```

■パラメータ

ID	tskid	R4	タスク ID
T_RTEX	*pk_rtex	R5	タスク例外処理状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RTEX	*pk_rtex	R5	タスク例外処理状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rtex {
           STAT    texstat;     0     4    タスク例外処理の状態
           TEXPTN  pndptn;     +4     4    保留例外要因
} T_RTEX;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtex が 4 の倍数以外)
E_ID	[p]	ID 範囲外(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(tskid のタスクが休止状態である、またはタスク例外処理ルーチンが定義されていない)

■機能

tskid で示されたタスクのタスク例外処理に関する状態を参照し、pk_rtex が指す領域に返します。

texstat には、対象タスクがタスク例外許可状態かタスク例外処理禁止状態かによって次のいずれかの値を返します。

- TTEX_ENA（H'00000000）タスク例外許可状態
- TTEX_DIS（H'00000001）タスク例外禁止状態

pndptn には、対象タスクの保留例外要因を返します。処理されていない例外処理要求がないときには、pndptn には 0 を返します。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

RENESAS

3.7 同期･通信(セマフォ)機能

表 3.13にセマフォでサポートしているサービスコール一覧を示します。

表3.13 同期･通信（セマフォ）サービスコール

項番	サービスコール *1		機 能	呼び出し可能な状態 *2						
				T	N	E	D	U	L	C
1	cre_sem	[s]	セマフォの生成	○		○	○	○		
	icre_sem				○	○	○	○		
2	acre_sem		セマフォの生成	○		○	○	○		
	iacre_sem		(ID 番号自動割付け)		○	○	○	○		
3	del_sem		セマフォの削除	○		○	○	○		
4	sig_sem	[S]	セマフォ資源の返却	○		○	○	○		
	isig_sem	[S]			○	○	○	○		
5	wai_sem	[S]	セマフォ資源の獲得	○		○		○		
6	pol_sem	[S]	同上(ポーリング)	○		○	○	○		
	ipol_sem				○	○	○	○		
7	twai_sem	[S]	同上(タイムアウト有)	○		○		○		
8	ref_sem		セマフォの状態参照	○		○	○	○		
	iref_sem				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.14にセマフォの仕様を示します。

表3.14 セマフォの仕様

項番	項目	内容
1	セマフォ ID	1～CFG_MAXSEMID (最大 1023)
2	セマフォカウンタ最大値	65535
3	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順

3.7.1 セマフォの生成(cre_sem, icre_sem)(acre_sem, iacre_sem：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_sem、icre_sem》

ID	semid	R4	セマフォ ID
T_CSEM	*pk_csem	R5	セマフォ生成情報を格納したパケットへのポインタ

《acre_sem、iacre_sem》

T_CSEM	*pk_csem	R4	セマフォ生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_sem、icre_sem》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_sem、iacre_sem》

ER_ID	semid	R0	生成したセマフォの ID 番号（正の値）またはエラーコード

■パケットの構造

```
typedef   struct  t_csem {
          ATR     sematr;     +0    4   セマフォ属性
          UINT    isemcnt;    +4    4   セマフォの資源数の初期値
          UINT    maxsem;     +8    4   セマフォの最大資源数
} T_CSEM;
```

■エラーコード

E_RSATR	[p]	属性不正(sematr が不正)
E_PAR	[p]	パラメータエラー(pk_csem が 4 の倍数以外、maxsem=0、maxsem＞H’ffff、isemcnt＞maxsem)
E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_OBJ	[k]	オブジェクト状態不正(semid のセマフォが存在)
E_NOID	[k]	空き ID なし

RENESAS

■機能

cre_sem, icre_sem サービスコールは、semid で示された ID を持つセマフォを、pk_csem で示された内容で生成します。

acre_sem, iacre_sem サービスコールは、未登録のセマフォ ID を検索して、その ID を持つセマフォを pk_csem で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のセマフォ ID を検索する範囲は 1～CFG_MAXSEMID です。

sematr には属性として、セマフォ資源獲得待ちタスクが待ち行列に並ぶ際の並び方を指定します。

```
sematr := (TA_TFIFO || TA_TPRI)
```

- TA_TFIFO（H'00000000）待ちタスクのキューイングは FIFO
- TA_TPRI（H'00000001）待ちタスクのキューイングは優先度順

isemcnt には、生成するセマフォの初期値を指定します。指定できる値の範囲は、0 から maxsem です。

maxsem には、生成するセマフォの最大資源数を指定します。指定できる値の範囲は、1 から 65,535 です。

なお、セマフォはコンフィギュレータで静的に生成することもできます。

3.7.2 セマフォの削除(del_sem)

■C 言語 API

```
ER ercd = del_sem(ID semid);
```

■パラメータ

ID	semid	R4	セマフォ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_NOEXS	[k]	未登録(semid のセマフォが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

semid で示されたセマフォを削除します。

semid で示されたセマフォで資源獲得を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

3.7.3 セマフォ資源の返却(sig_sem, isig_sem)

■C 言語 API

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

■パラメータ

ID	semid	R4	セマフォ ID

■リターンパラメータ

ER	ercd	R0	正常終了(E_OK)またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_NOEXS	[k]	未登録(semid のセマフォが存在しない)
E_QOVR	[k]	キューイングオーバフロー(semcnt＞maxsem*1)

*1 maxsem:セマフォ生成時に指定した最大セマフォ資源数

■機能

semid で示されたセマフォに資源をひとつ返却します。対象セマフォで待っているタスクがあれば、セマフォの待ち行列先頭タスクに資源を割り付けて待ち状態を解除します。セマフォに対して待っているタスクがなければ、そのセマフォのカウント値を 1 増やします。

セマフォのカウント値の最大値は、セマフォ生成時に指定した maxsem です。

3.7.4 セマフォ資源の獲得(wai_sem, pol_sem, ipol_sem, twai_sem)

■C 言語 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);
```

■パラメータ

ID	semid	R4	セマフォ ID
《twai_sem》			
TMO	tmout	R5	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tmout≦-2)
E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_NOEXS	[k]	未登録(semid のセマフォが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(semid のセマフォが削除された)
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト

■機能

semid で指定されるセマフォから、資源をひとつ獲得します。

対象セマフォの資源数が 1 以上の場合には、セマフォの資源数から 1 を減じ、実行を継続します。資源数が 0 の場合には、wai_sem, twai_sem サービスコールでは呼び出しタスクはそのセマフォの待ち行列につながれ、pol_sem, ipol_sem サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

twai_sem サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、pol_sem サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。この場合、wai_sem サービスコールと同じ動作となります。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.7.5 セマフォの状態参照(ref_sem, iref_sem)

■C 言語 API

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

■パラメータ

ID	semid	R4	セマフォ ID
T_RSEM	*pk_rsem	R5	セマフォ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RSEM	*pk_rsem	R5	セマフォ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct  t_rsem {
          ID      wtskid;    +0    2   待ちタスク ID
          UINT    semcnt;    +4    4   現在のセマフォカウント値
} T_RSEM;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rsem が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_NOEXS	[k]	未登録(semid のセマフォが存在しない)

■機能

semid で示されたセマフォの状態を参照します。

pk_rsem が指す領域に、待ち行列の先頭タスク ID(wtskid)、現在のセマフォカウント値(semcnt)を返します。

対象セマフォの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE（0）を返します。

3.8 同期･通信(イベントフラグ)機能

表 3.15にイベントフラグでサポートしているサービスコール一覧を示します。

表3.15 同期･通信（イベントフラグ）サービスコール

項番	サービスコール *1		機 能	呼び出し可能なシステム状態 *2						
				T	N	E	D	U	L	C
1	cre_flg	[s]	イベントフラグの生成	○		○	○	○		
	icre_flg				○	○	○	○		
2	acre_flg		イベントフラグの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_flg				○	○	○	○		
3	del_flg		イベントフラグの削除	○		○	○	○		
4	set_flg	[S]	イベントフラグのセット	○		○	○	○		
	iset_flg	[S]			○	○	○	○		
5	clr_flg	[S]	イベントフラグのクリア	○		○	○	○		
	iclr_flg				○	○	○	○		
6	wai_flg	[S]	イベントフラグ待ち	○		○		○		
7	pol_flg	[S]	同上(ポーリング)	○		○	○	○		
	ipol_flg	[S]			○	○	○	○		
8	twai_flg	[S]	同上(タイムアウト有)	○		○		○		
9	ref_flg		イベントフラグの状態参照	○		○	○	○		
	iref_flg				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.16にイベントフラグの仕様を示します。

表3.16 イベントフラグの仕様

項番	項目	内容
1	イベントフラグ ID	1～CFG_MAXFLGID(最大 1023)
2	イベントフラグのビット数	32 ビット
3	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順 TA_WSGL : 複数タスクの待ちを許さない TA_WMUL : 複数タスクの待ちを許す TA_CLR : 待ち解除時にイベントフラグを 0 クリア

RENESAS

3.8.1 イベントフラグの生成(cre_flg, icre_flg)
(acre_flg, iacre_flg：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_flg、icre_flg》

ID	flgid	R4	イベントフラグ ID
T_CFLG	*pk_cflg	R5	イベントフラグ生成情報を格納したパケットへのポインタ

《acre_flg、iacre_flg》

T_CFLG	*pk_cflg	R4	イベントフラグ生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_flg、icre_flg》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_flg、iacre_flg》

ER_ID	flgid	R0	生成したイベントフラグの ID 番号（正の値） またはエラーコード

■パケットの構造

```
typedef    struct  t_cflg {
           ATR     flgatr;      +0     4    イベントフラグ属性
           FLGPTN  iflgptn;     +4     4    イベントフラグの初期値
} T_CFLG;
```

■エラーコード

E_RSATR	[p]	属性不正(flgatr が不正)
E_PAR	[p]	パラメータエラー(pk_cflg が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_OBJ	[k]	オブジェクト状態不正(flgid のイベントフラグが存在)
E_NOID	[k]	空き ID なし

■機能

cre_flg, icre_flg サービスコールは、flgid で示された ID を持つイベントフラグを、pk_cflg で示された内容で生成します。

acre_flg, iacre_flg サービスコールは、未登録のイベントフラグ ID を検索してその ID を持つイベントフラグを pk_cflg で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のイベントフラグ ID を検索する範囲は 1～CFG_MAXFLGID です。

flgatr には属性として、イベントフラグ待ちタスクが待ち行列に並ぶ際の並び方と、生成するイベントフラグに待つことのできるタスク数を指定します。

```
flgatr := ((TA_TFIFO || TA_TPRI) | (TA_WSGL || TA_WMUL) | [TA_CLR])
```

- TA_TFIFO（H'00000000）待ちタスクのキューイングは FIFO
- TA_TPRI（H'00000001）待ちタスクのキューイングは優先度順
- TA_WSGL（H'00000000）複数タスクの待ちを許さない
- TA_WMUL（H'00000002）複数タスクの待ちを許す
- TA_CLR（H'00000004）待ち解除時にイベントフラグを 0 クリア

TA_WSGL 属性のイベントフラグでは、待つことのできるタスクは 1 つになります。この場合は、TA_TFIFO と TA_TPRI のどちらの属性を指定してもイベントフラグの動作は同じになります。これに対して、TA_WMUL 属性のイベントフラグでは、複数のタスクが待ち状態に遷移することができます。TA_CLR 属性が指定された場合は、タスクをイベントフラグ待ち状態から解除する時に、イベントフラグのビットパターンのすべてのビットをクリアします。

iflgptn には、生成するイベントフラグの初期値を指定します。

なお、イベントフラグはコンフィギュレータで静的に生成することもできます。

3.8.2 イベントフラグの削除(del_flg)

■C 言語 API

```
ER ercd = del_flg(ID flgid);
```

■パラメータ

ID	flgid	R4	イベントフラグ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[k]	未登録(flgid のイベントフラグが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

flgid で示されたイベントフラグを削除します。

flgid で示されたイベントフラグで条件成立を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

3.8.3 イベントフラグのセット(set_flg, iset_flg)

■C 言語 API

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

■パラメータ

ID	flgid	R4	イベントフラグ ID
FLGPTN	setptn	R5	セットするビットパターン

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(flgid≦0、flgid>CFG_MAXFLGID)
E_NOEXS	[k]	未登録(flgid のイベントフラグが存在しない)

■機能

flgid で示されたイベントフラグを、setptn で示された値との論理和（OR）に更新します。

イベントフラグ値の更新の結果、そのイベントフラグで待っているタスクの待ち解除条件を満たせば、そのタスクの待ち状態を解除します。なお、待ち解除条件は待ち行列の順に評価されます。この時、対象のイベントフラグ属性に TA_CLR 属性が指定されている場合には、イベントフラグのビットパターンのすべてのビットをクリアし、サービスコールの処理を終了します。

イベントフラグに TA_WMUL 属性が指定されており、TA_CLR 属性が指定されていない場合、set_flg の 1 回の呼び出しで複数のタスクが待ち解除される可能性があります。待ち解除されるタスクが複数ある場合には、イベントフラグの待ち行列につながれていた順序で待ち解除されます。

3.8.4 イベントフラグのクリア(clr_flg, iclr_flg)

■C 言語 API

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

■パラメータ

ID	flgid	R4	イベントフラグ ID
FLGPTN	clrptn	R5	クリアするビットパターン

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[k]	未登録(flgid のイベントフラグが存在しない)

■機能

flgid で示されたイベントフラグを、clrptn で示された値との論理積（AND）に更新します。

3.8.5 イベントフラグ待ち(wai_flg, pol_flg, ipol_flg, twai_flg)

■C 言語 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);
```

■パラメータ

ID	flgid	R4	イベントフラグ ID
FLGPTN	waiptn	R5	待ちビットパターン
MODE	wfmode	R6	待ちモード
FLGPTN	*p_flgptn	R7	待ち解除時のビットパターンを返す領域へのポインタ
《twai_flg》			
TMO	tmout	@R15	タイムアウト値

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
FLGPTN	*p_flgptn	R7	待ち解除時のビットパターンへのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_flgptn が 4 の倍数以外、waiptn=0、wfmode が不正、tmout≦-2)
E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[k]	未登録(flgid のイベントフラグが存在しない)
E_ILUSE	[k]	サービスコール不正使用(TA_WSGL 属性のイベントフラグに待ちタスクが存在)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(flgid のイベントフラグが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待ちます。p_flgptn の指す領域には、待ち解除される時のイベントフラグのビットパターンを返します。

対象イベントフラグが TA_WSGL 属性で、すでに他のタスクが待っている場合は、本サービスコールはエラーE_ILUSE となります。

本サービスコール呼び出し時にすでに待ち解除条件が成立している場合は、本サービスコールは直ちに終了します。待ち解除条件が成立していない場合は、wai_flg, twai_flg サービスコールの場合はイベント待ち行列につながれ、pol_flg, ipol_flg サービスコールの場合は直ちにエラーE_TMOUT で終了します。

wfmode には、次のような指定を行います。

```
wfmode := ( (TWF_ANDW || TWF_ORW) )
```

- TWF_ANDW（H'00000000）AND 待ち
- TWF_ORW（H'00000001）OR 待ち

TWF_ANDW では、waiptn で指定したビットの全てがセットされるのを待ちます。TWF_ORW では、flgid で示されたイベントフラグのうち waiptn で指定したビットのいずれかがセットされるのを待ちます。

twai_flg サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、pol_flg サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。この場合、wai_flg サービスコールと同じ動作となります。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.8.6 イベントフラグの状態参照(ref_flg, iref_flg)

■C 言語 API

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

■パラメータ

ID	flgid	R4	イベントフラグ ID
T_RFLG	*pk_rflg	R5	イベントフラグ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RFLG	*pk_rflg	R5	イベントフラグ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rflg {
           ID      wtskid;      +0     2    待ちタスク ID
           FLGPTN  flgptn;      +4     4    イベントフラグのビットパターン
} T_RFLG;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rflg が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[k]	未登録(flgid のイベントフラグが存在しない)

■機能

flgid で示されたイベントフラグの状態を参照します。

pk_rflg の指す領域に、待ち行列の先頭タスク ID(wtskid)、現在のイベントフラグのビットパターン(flgptn)を返します。

対象イベントフラグの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE（0）を返します。

RENESAS

3.9 同期・通信(データキュー)機能

表 3.17にデータキューでサポートしているサービスコール一覧を示します。

表3.17 同期・通信（データキュー）サービスコール

項番	サービスコール *1		機 能	呼び出し可能なシステム状態 *2						
				T	N	E	D	U	L	C
1	cre_dtq	[s]	データキューの生成	○		○	○	○		
	icre_dtq				○	○	○	○		
2	acre_dtq		データキューの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_dtq				○	○	○	○		
3	del_dtq		データキューの削除	○		○	○	○		
4	snd_dtq	[S]	データキューへの送信	○		○		○		
5	psnd_dtq	[S]	同上(ポーリング)	○		○	○	○		
	ipsnd_dtq	[S]			○	○	○	○		
6	tsnd_dtq	[S]	同上(タイムアウト有)	○		○		○		
7	fsnd_dtq	[S]	データキューへの強制送信	○		○	○	○		
	ifsnd_dtq	[S]			○	○	○	○		
8	rcv_dtq	[S]	データキューからの受信	○		○		○		
9	prcv_dtq	[S]	同上(ポーリング)	○		○	○	○		
10	trcv_dtq	[S]	同上(タイムアウト有)	○		○		○		
11	ref_dtq		データキューの状態参照	○		○	○	○		
	iref_dtq				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.18にデータキューの仕様を示します。

表3.18 データキューの仕様

項番	項目	内容
1	データキューID	1～CFG_MAXDTQID (最大 1023)
2	1 ワード	32 ビット
3	サポート属性	TA_TFIFO： 送信待ちタスクのキューイングは FIFO TA_TPRI： 送信待ちタスクのキューイングは優先度順

3.9.1 データキューの生成(cre_dtq, icre_dtq)
(acre_dtq, iacre_dtq：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_dtq、icre_dtq》

ID	dtqid	R4	データキューID
T_CDTQ	*pk_cdtq	R5	データキュー生成情報を格納したパケットへのポインタ

《acre_dtq、iacre_dtq》

T_CDTQ	*pk_cdtq	R4	データキュー生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_dtq、icre_dtq》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_dtq、iacre_dtq》

ER_ID	dtqid	R0	生成したデータキューの ID 番号（正の値） またはエラーコード

■パケットの構造

```
typedef   struct  t_cdtq {
          ATR     dtqatr;     +0    4   データキュー属性
          UINT    dtqcnt;     +4    4   データキュー領域の容量（データの個数）
          VP      dtq;        +8    4   データキュー領域の先頭アドレス
} T_CDTQ;
```

■エラーコード

E_NOMEM	[k]	メモリ不足(データキュー領域が確保できない)
E_RSATR	[p]	属性不正(dtqatr が不正)
E_PAR	[p]	パラメータエラー(pk_cdtq が 4 の倍数以外)
	[k]	(dtqcnt×データキューサイズ(4 バイト)が 32 ビット範囲を超えている)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_OBJ	[k]	オブジェクト状態不正(dtqid のデータキューが存在)
E_NOID	[k]	空き ID なし

■機能

cre_dtq, icre_dtq サービスコールは、dtqid で示された ID を持つデータキューを、pk_cdtq で示された内容で生成します。

acre_dtq, iacre_dtq サービスコールは、未登録のデータキューID を検索してその ID を持つデータキューを pk_cdtq で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のデータキューID を検索する範囲は 1～CFG_MAXDTQID です。

dtqatr には属性として、データキュー送信待ちタスクが待ち行列に並ぶ際の並び方を指定します。

```
dtqatr := (TA_TFIFO‖TA_TPRI)
```

- TA_TFIFO（H'00000000）送信待ちタスクのキューイングは FIFO
- TA_TPRI（H'00000001）送信待ちタスクのキューイングは優先度順

なお、データキューの受信待ち行列は、常に FIFO 順となります。また、データキューに送信されるデータは優先度を持たず、データキュー中のデータの順序も FIFO で管理されます。

dtqcnt には、データキュー領域に格納できるデータの個数を指定します。dtqcnt に 0 を指定することも可能です。その場合、データ送信タスクとデータ受信タスクは、完全に同期した動作になります。

データキューは、コンフィギュレータで指定したデータキュー用領域(CFG_DTQSZ)から割り付けられます。生成に成功すると、データキュー用領域の空きは以下の式で計算されるサイズだけ減少します。

- 減少サイズ ＝ dtqcnt×4＋16

dtq は、将来拡張用であり、本カーネルでは NULL を指定する必要があります。NULL 以外を指定した場合の動作は保証されません。

なお、データキューはコンフィギュレータで静的に生成することもできます。

3.9.2 データキューの削除(del_dtq)

■C 言語 API

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

■パラメータ

ID	dtqid	R4	データキューID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_NOEXS	[k]	未登録(dtqid のデータキューが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

dtqid で示されたデータキューを削除します。

dtqid で示されたデータキューで送信待ち、受信待ちのタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

削除により、データキュー用領域の空きは以下の式で計算されるサイズだけ増加します。

- 増加サイズ ＝ (生成時に指定した dtqcnt)×4＋16

3.9.3　データキューへの送信(snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq, fsnd_dtq, ifsnd_dtq)

■C 言語 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);
```

■パラメータ

ID	dtqid	R4	データキューID
VP_INT	data	R5	データキューへ送信するデータ
《tsnd_dtq》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tmout≦-2)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_ILUSE	[k]	サービスコール不正使用(dtqcnt が 0 のデータキューに対する fsnd_dtq, ifsnd_dtq の発行)
E_NOEXS	[k]	未登録(dtqid のデータキューが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(dtqid のデータキューが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

dtqid で示されたデータキューに対して、data で示されたデータ（4 バイト）を送信します。

なお、fsnd_dtq, ifsnd_dtq は、dtqcnt=0 で生成したデータキューを指定した場合は常に E_ILUSE エラーとなります。

(1)　対象データキューに受信待ちタスクが存在する場合

データキューには格納せずに受信待ち行列の先頭タスクにデータを渡し、そのタスクの待ち状態を解除します。

(2)　対象データキューに受信待ちタスクが存在しない場合

(a)　データキューに空きがある場合

data をデータキューの末尾に格納します。データキューカウントは＋1 されます。

(b) データキューに空きがない場合

- snd_dtq, tsnd_dtq **の場合**

  呼び出しタスクはデータキューの空き領域を待つための待ち行列（送信待ち行列）につながれます。
  tsnd_dtqサービスコールの場合、tmoutには待ち時間を指定します。
  tmoutに正の値を指定した場合、待ち条件が満たされないままtmout時間が経過すると、エラーコードとしてE_TMOUTを返します。tmout=TMO_POL（0）を指定した場合、psnd_dtqサービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、snd_dtqサービスコールと同じ処理を行います。
  CFG_TICDENO(タイムティック周期時間の分母)に1より大きな値を設定した場合は、tmoutに指定可能な最大値はH'7fffffff/CFG_TICDENOに制限されます。これより大きな値を指定した場合の動作は保証されません。
  時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

- psnd_dtq, ipsnd_dtq **の場合**

  直ちにエラーE_TMOUTで終了します。

- fsnd_dtq, ifsnd_dtq **の場合**

  送信待ちタスクが存在するかどうかに関わらず、データキューの先頭のデータ(最も古いデータ)を削除した後、dataをデータキューの末尾に格納します。

3.9.4 データキューからの受信(rcv_dtq, prcv_dtq, trcv_dtq)

■C 言語 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);
```

■パラメータ

ID	dtqid	R4	データキューID
VP_INT	*p_data	R5	受信したデータを返す領域の先頭アドレス
《trcv_dtq》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
VP_INT	*p_data	R5	受信したデータを格納した領域へのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_data が 4 の倍数以外、tmout≦-2)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_NOEXS	[k]	未登録(dtqid のデータキューが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(dtqid のデータキューが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

dtqid で示されたデータキューからデータを受信し、p_data の指す領域に格納します。

データキューにデータがあれば、その先頭のデータ（最古のメッセージ）を受信します。データキュー内のデータを受信することで、データキューカウントは－1 されます。この結果、送信待ち行列のタスクに対してもデータの格納が可能であれば、待ち行列の順にデータの送信処理を行います。

データキューにデータが存在せず、データ送信待ちタスクが存在する場合（このような状況が起るのは、データキュー領域の容量が 0 の場合のみです）、データ送信待ち行列先頭タスクのデータを受信します。この結果、そのデータ送信待ちタスクの待ち状態は解除されます。

データキューにデータがなく、データ送信待ちタスクも存在しない場合、rcv_dtq, trcv_dtq サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_dtq サービスコールでは直ちにエラーE_TMOUT で終了します。受信待ち行列は、FIFO で管理されます。

trcv_dtq サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、prcv_dtq サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、rcv_dtq サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.9.5 データキューの状態参照(ref_dtq, iref_dtq)

■C 言語 API

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

■パラメータ

ID	dtqid	R4	データキューID
T_RDTQ	*pk_rdtq	R5	データキュー状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RDTQ	*pk_rdtq	R5	データキュー状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rdtq {
           ID      stskid;      0    2    送信待ちタスク ID
           ID      rtskid;     +2    2    受信待ちタスク ID
           UINT    sdtqcnt;    +4    4    データキューに入っているデータの数
} T_RDTQ;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rdtq が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_NOEXS	[k]	未登録(dtqid のデータキューが存在しない)

■機能

dtqid で示されたデータキューの状態を参照し、pk_rdtq が指す領域に送信待ちタスク ID(stskid)、受信待ちタスク ID(rtskid)、データキューに入っているデータの数(sdtqcnt)を返します。

送信待ちタスク、受信待ちタスクが無い場合は、待ちタスク ID として TSK_NONE（0）を返します。

3.10 同期･通信(メールボックス)機能

表 3.19にメールボックスでサポートしているサービスコール一覧を示します。

表3.19 同期･通信（メールボックス）サービスコール

項番	サービスコール *1		機 能	呼び出し可能な状態 *2						
				T	N	E	D	U	L	C
1	cre_mbx	[s]	メールボックスの生成	○		○	○	○		
	icre_mbx				○	○	○	○		
2	acre_mbx		メールボックスの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mbx				○	○	○	○		
3	del_mbx		メールボックスの削除	○		○	○	○		
4	snd_mbx	[S]	メールボックスへの送信	○		○	○	○		
	isnd_mbx				○	○	○	○		
5	rcv_mbx	[S]	メールボックスからの受信	○		○		○		
6	prcv_mbx	[S]	同上(ポーリング)	○		○	○	○		
	iprcv_mbx				○	○	○	○		
7	trcv_mbx	[S]	同上(タイムアウト有)	○		○		○		
8	ref_mbx		メールボックスの状態参照	○		○	○	○		
	iref_mbx				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.20にメールボックスの仕様を示します。

表3.20 メールボックスの仕様

項番	項目	内容
1	メールボックス ID	1～CFG_MAXMBXID (最大 1023)
2	メッセージ優先度	1～CFG_MAXMSGPRI * (最大 255)
3	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順 TA_MFIFO : メッセージのキューイングは FIFO TA_MPRI : メッセージのキューイングは優先度順

【注】 * kernel_macro.h に定義される TMAX_MPRI と同じ値です。

3.10.1 メールボックスの生成(cre_mbx, icre_mbx)
(acre_mbx, iacre_mbx：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_mbx、icre_mbx》

ID	mbxid	R4	メールボックス ID
T_CMBX	*pk_cmbx	R5	メールボックス生成情報を格納したパケットへのポインタ

《acre_mbx、iacre_mbx》

T_CMBX	*pk_cmbx	R4	メールボックス生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_mbx、icre_mbx》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_mbx、iacre_mbx》

ER_ID	mbxid	R0	生成したメールボックスの ID 番号（正の値） またはエラーコード

■パケットの構造

```
typedef    struct  t_cmbx {
           ATR     mbxatr;     +0    4    メールボックス属性
           PRI     maxmpri;    +4    2    メッセージ優先度の最大値
           VP      mprihd;     +8    4    優先度別メッセージキューヘッダの先頭アドレス
} T_CMBX;
```

■エラーコード

E_RSATR	[p]	属性不正(mbxatr が不正)
E_PAR	[p]	パラメータエラー(pk_cmbx が 4 の倍数以外、maxmpri≦0、 maxmpri＞CFG_MAXMSGPRI)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_OBJ	[k]	オブジェクト状態不正(mbxid のメールボックスが存在)
E_NOID	[k]	空き ID なし

■機能

cre_mbx, icre_mbx サービスコールは、mbxid で示された ID を持つメールボックスを、pk_cmbx で示された内容で生成します。

acre_mbx, iacre_mbx サービスコールは、未登録のメールボックス ID を検索してその ID を持つメールボックスを pk_cmbx で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のメールボックス ID を検索する範囲は 1～CFG_MAXMBXID です。

mbxatr には属性として、受信待ちタスクおよびメッセージが待ち行列に並ぶ際の並び方を指定します。

```
mbxatr := ( (TA_TFIFO‖TA_TPRI) | (TA_MFIFO‖TA_MPRI) )
```

- TA_TFIFO（H'00000000）受信待ちタスクのキューイングは FIFO
- TA_TPRI（H'00000001）受信待ちタスクのキューイングは優先度順
- TA_MFIFO（H'00000000）メッセージのキューイングは FIFO
- TA_MPRI（H'00000002）メッセージのキューイングは優先度順

mprihd は、mbxatr に TA_MPRI を指定した場合は必ず NULL を指定してください。μITRON4.0 仕様では、NULL 以外を指定することで、mprihd で指定された領域にメッセージキューヘッダ領域を生成する仕様になっていますが、本カーネルは NULL 以外を指定した場合の機能はサポートしていません。**NULL 以外の指定を行った場合の動作は保証されません。**なお、TA_MPRI の指定がない場合は、mprihd は意味を持たず、単に無視されます。

なお、メールボックスはコンフィギュレータで静的に生成することもできます。

RENESAS

3.10.2 メールボックスの削除(del_mbx)

■C 言語 API

```
ER ercd = del_mbx(ID mbxid);
```

■パラメータ

ID	mbxid	R4	メールボックス ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[k]	未登録(mbxid のメールボックスが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mbxid で示されたメールボックスを削除します。

mbxid で示されたメールボックスでメッセージを待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。また、メールボックス内にメッセージが存在する場合でもエラーにはなりませんが、メッセージ領域については何ら処理を行いません。たとえば、メモリプールから獲得したメモリブロックをメッセージとして使用していた場合でも、カーネルが自動的にメッセージ領域をメモリプールに返却するわけではありません。

3.10.3 メールボックスへの送信(snd_mbx, isnd_mbx)

■C 言語 API

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

■パラメータ

ID	mbxid	R4	メールボックス ID
T_MSG	*pk_msg	R5	送信メッセージの先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

《メールボックスのメッセージヘッダ》

```
typedef    struct  t_msg {
           VP      msghead;    +0     4    カーネル管理領域
} T_MSG;
```

《メールボックスの優先度付きメッセージヘッダ》

```
typedef    struct  t_msg_pri {
           T_MSG   msgque;     +0     4    メッセージヘッダ
           PRI     msgpri;     +4     2    メッセージ優先度
} T_MSG_PRI;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_msg が 4 の倍数以外、メッセージ先頭 4 バイトが 0 以外)
	[k]	(msgpri≦0、msgpri＞CFG_MAXMSGPRI)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＜0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[k]	未登録(mbxid のメールボックスが存在しない)

■機能

mbxid で示されたメールボックスに pk_msg で示されたメッセージを送信します。

すでに対象メールボックスにメッセージの受信を待つタスクが存在していれば、待ち行列先頭のタスクに送信したメッセージが渡され、そのタスクの待ち状態が解除されます。

メッセージの受信を待つタスクが存在しない場合は、メッセージをメッセージ待ち行列につなぎます。待ち行列は、生成時に指定した属性にしたがって管理されます。

TA_MFIFO 属性のメールボックスにメッセージを送る場合は、図 3.4に示すように先頭に T_MSG 構造体を付加した形式で、メッセージを作成してください。

TA_MPRI 属性のメールボックスにメッセージを送る場合は、図 3.5に示すように先頭に T_MSG_PRI 構造体を付加した形式で、メッセージを作成してください。

TA_MFIFO, TA_MPRI いずれの属性の場合もメッセージは RAM 領域に作成し、送信前に T_MSG 領域を 0 にしてください。

T_MSG の領域はカーネルが使用するため、送信後は書き換えてはなりません。メッセージ送信後、メッセージが受信される前にこの領域を書き換えた場合の動作は保証されません。

```
typedef struct  user_msg {
    T_MSG    t_msg;      /* T_MSG 構造体                              */
    B        data[8];    /* ユーザメッセージデータ構造の例(任意の構造) */
} USER_MSG;
```

図3.4　メッセージの形式例

```
typedef struct  user_msg {
    T_MSG_PRI  t_msg;    /* T_MSG_PRI 構造体                          */
    B          data[8];  /* ユーザメッセージデータ構造の例(任意の構造) */
} USER_MSG;
```

図3.5　優先度付きメッセージの形式例

3.10.4 メールボックスからの受信(rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx)

■C 言語 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);
```

■パラメータ

ID	mbxid	R4	メールボックス ID
T_MSG	**ppk_msg	R5	受信メッセージ先頭アドレスを返す領域へのポインタ
《trcv_mbx》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了 (E_OK) またはエラーコード
T_MSG	**ppk_msg	R5	受信メッセージ先頭アドレスを格納した領域へのポインタ

■パケットの構造

《メールボックスのメッセージヘッダ》

```
typedef    struct  t_msg {
           VP      msghead;     +0     4    カーネル管理領域
} T_MSG;
```

《メールボックスの優先度付きメッセージヘッダ》

```
typedef    struct  t_msg_pri {
           T_MSG   msgque;      +0     4    メッセージヘッダ
           PRI     msgpri;      +4     2    メッセージ優先度
} T_MSG_PRI;
```

■エラーコード

E_PAR	[p]	パラメータエラー(ppk_msg が 4 の倍数以外、tmout≦-2)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[k]	未登録(mbxid のメールボックスが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mbxid のメールボックスが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

RENESAS

■機能

mbxid で示されたメールボックスからメッセージを受信し、受信したメッセージの先頭アドレスを pk_msg に返します。

メールボックスにメッセージが存在しない場合は、rcv_mbx, trcv_mbx サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_mbx, iprcv_mbx サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

trcv_mbx サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、prcv_mbx サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、rcv_mbx サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.10.5　メールボックスの状態参照(ref_mbx, iref_mbx)

■C 言語 API

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

■パラメータ

ID	mbxid	R4	メールボックス ID
T_RMBX	*pk_rmbx	R5	メールボックス状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RMBX	*pk_rmbx	R5	メールボックス状態を格納したパケットへのポインタ

■パケットの構造

(1)T_RMBX

```
typedef     struct  t_rmbx{
            ID      wtskid;     +0    2   待ちタスク ID
            T_MSG   *pk_msg;    +4    4   次に受信されるメッセージの先頭アドレス
} T_RMBX;
```

(2)T_MSG

《メールボックスのメッセージヘッダ》

```
typedef     struct  t_msg {
            VP      msghead;    +0    4   カーネル管理領域
} T_MSG;
```

《メールボックスの優先度付きメッセージヘッダ》

```
typedef     struct  t_msg_pri {
            T_MSG   msgque;     +0    4   メッセージヘッダ
            PRI     msgpri;     +4    2   メッセージ優先度
} T_MSG_PRI;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmbx が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[k]	未登録(mbxid のメールボックスが存在しない)

■機能

mbxid で示されたメールボックスの状態を参照します。pk_rmbx が示す領域に待ちタスク ID(wtskid)、次に受信されるメッセージの先頭アドレス(pk_msg)を返します。対象メールボックスの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE（0）を返します。次に受信されるメッセージが無い場合は、メッセージの先頭アドレスとして NULL（0）を返します。

3.11 拡張同期・通信(ミューテックス)機能

表 3.21にミューテックスでサポートしているサービスコール一覧を示します。

表3.21 同期·通信（ミューテックス）サービスコール

項番	サービスコール *1	機 能	呼び出し可能な状態 *2						
			T	N	E	D	U	L	C
1	cre_mtx	ミューテックスの生成	○		○	○	○		
2	acre_mtx	ミューテックスの生成 (ID 番号自動割付け)	○		○	○	○		
3	del_mtx	ミューテックスの削除	○		○	○	○		
4	loc_mtx	ミューテックスのロック	○		○		○		
5	ploc_mtx	同上(ポーリング)	○		○	○	○		
6	tloc_mtx	同上(タイムアウト有)	○		○		○		
7	unl_mtx	ミューテックスのロック解除	○		○	○	○		
8	ref_mtx	ミューテックスの状態参照	○		○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.22にミューテックス機能の仕様を示します。

表3.22 ミューテックスの仕様

項番	項目	内容
1	ミューテックス ID	1〜CFG_MAXMTXID (最大 1023)
2	サポート属性	TA_CEILING（優先度上限プロトコル）

【注】* HI7000/4 シリーズのミューテックスにおける TA_CEILING 属性(優先度上限プロトコル)では、簡略化した優先度制御規則を採用しています。簡略化した優先度制御規則では、タスクの優先度を高くする制御はすべて行われますが、タスクの優先度を低くする制御は、タスクがロックしていたミューテックスが無くなったとき(複数のミューテックスをロックしていた場合は、それら全てを解放したとき)にのみ行われます。

3.11.1 ミューテックスの生成(cre_mtx)

(acre_mtx：ID 番号自動割付け)

■C 言語 API

```
ER ercd = cre_mtx(ID mtxid, T_CMTX *pk_cmtx);
ER_ID mtxid = acre_mtx(T_CMTX *pk_cmtx);
```

■パラメータ

《cre_mtx》

ID	mtxid	R4	ミューテックス ID
T_CMTX	*pk_cmtx	R5	ミューテックス生成情報を格納したパケットへのポインタ

《acre_mtx》

T_CMTX	*pk_cmtx	R4	ミューテックス生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_mtx》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_mtx》

ER_ID	mtxid	R0	生成したミューテックスの ID 番号（正の値） またはエラーコード

■パケットの構造

```
typedef   struct  t_cmtx {
          ATR     mtxatr;     +0    4    ミューテックス属性
          PRI     ceilpri;    +4    2    ミューテックスの上限優先度
} T_CMTX;
```

■エラーコード

E_RSATR	[p]	属性不正(mtxatr が不正)
E_PAR	[p]	パラメータエラー(pk_cmtx が 4 の倍数以外、ceilpri≦0、ceilpri＞CFG_MAXTSKPRI)
E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_OBJ	[k]	オブジェクト状態不正(mtxid のミューテックスが存在)
E_NOID	[k]	空き ID なし

■機能

cre_mtx サービスコールは、mtxid で示された ID を持つミューテックスを、pk_cmtx で示された内容で生成します。

acre_mtx サービスコールは､未登録のミューテックス ID を検索してその ID を持つミューテックスを pk_cmtx で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のミューテックス ID を検索する範囲は 1～CFG_MAXMTXID です。

mtxatr には属性としては、優先度上限プロトコル（TA_CEILING）のみを指定できます。

```
mtxatr := (TA_CEILING)
```

- TA_CEILING（H'00000003）優先度上限プロトコル

ceilpri には、生成するミューテックスの上限優先度を指定します。指定できる値の範囲は、1～CFG_MAXTSKPRI です。

なお、ミューテックスはコンフィギュレータで静的に生成することもできます。

3.11.2 ミューテックスの削除(del_mtx)

■C 言語 API

```
ER ercd = del_mtx(ID mtxid);
```

■パラメータ

ID	mtxid	R4	ミューテックス ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_NOEXS	[k]	未登録(mtxid のミューテックスが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mtxid で示されたミューテックスを削除します。

mtxid で示されたミューテックスにロック待ちタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

対象ミューテックスがロックされていた場合には、それをロックしているタスクのロックを解除します。その結果、そのタスクがロックしているミューテックスがなくなった場合のみ、タスクの現在優先度をベース優先度に戻します。

削除されたミューテックスをロックしているタスクには、ミューテックスが削除されたことは通知されません。後でミューテックスをロック解除しようとした時点でエラーが返されます。

RENESAS

3.11.3 ミューテックス資源の獲得(loc_mtx, ploc_mtx, tloc_mtx)

■C 言語 API

```
ER ercd = loc_mtx(ID mtxid);
ER ercd = ploc_mtx(ID mtxid);
ER ercd = tloc_mtx(ID mtxid, TMO tmout);
```

■パラメータ

ID	mtxid	R4	ミューテックス ID
《tloc_mtx》			
TMO	tmout	R5	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tmout≦-2)
E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_NOEXS	[k]	未登録(mtxid のミューテックスが存在しない)
E_ILUSE	[k]	サービスコール不正使用(呼び出しタスクは既に mtxid のミューテックスをロック済み、呼び出しタスクの bpri ＞ 対象ミューテックスの ceilpri)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mtxid のミューテックスが削除された)
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト

■機能

mtxid で指定されるミューテックスをロックします。

対象ミューテックスがロックされていない場合には、自タスクがミューテックスをロックした状態にして、サービスコールの処理を終了します。その際、自タスクの現在優先度はミューテックスの上限優先度まで引き上げられます。

対象ミューテックスがロックされている場合には、自タスクを待ち行列につなぎ、ミューテックスのロック待ち状態に移行させます。待ち行列は、優先度順に管理されます。

tloc_mtx サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、ploc_mtx サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。この場合、loc_mtx サービスコールと同じ動作となります。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.11.4 ミューテックスのロック解除(unl_mtx)

■C 言語 API

```
ER ercd = unl_mtx(ID mtxid);
```

■パラメータ

ID	mtxid	R4	ミューテックス ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_NOEXS	[k]	未登録(mtxid のミューテックスが存在しない)
E_ILUSE	[k]	サービスコール不正使用(対象ミューテックスをロックしていない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mtxid で示されたミューテックスのロックを解除します。対象ミューテックスに対してロックを待っているタスクがあれば、ミューテックスの待ち行列先頭タスクを待ち解除し、待ち解除されたタスクがミューテックスをロックした状態にします。その際、ロックするタスクの現在優先度はミューテックスの上限優先度まで引き上げられます。ミューテックスに対して待っているタスクがなければ、そのミューテックスをロックされていない状態にします。

本カーネルの TA_CEILING 属性は、簡略化した優先度上限プロトコルを採用しています。つまり、本サービスコールによって呼び出しタスクがロックしているミューテックスが全て無くなったときのみ、現在優先度をベース優先度に戻します。呼び出しタスクがまだ他のミューテックスをロックしている場合、本サービスコールでは現在優先度は変化しません。

3.11.5 ミューテックスの状態参照(ref_mtx)

■C 言語 API

```
ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

■パラメータ

ID	mtxid	R4	ミューテックス ID
T_RMTX	*pk_rmtx	R5	ミューテックス状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RMTX	*pk_rmtx	R5	ミューテックス状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rmtx {
           ID      htskid;      +0     2    ミューテックスをロックしているタスク ID
           ID      wtskid;      +2     2    ミューテックスの待ち行列の先頭タスク ID
} T_RMTX;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmtx が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_NOEXS	[k]	未登録(mtxid のミューテックスが存在しない)

■機能

mtxid で示されたミューテックスの状態を参照します。

pk_rmtx が指す領域に、ミューテックスをロックしているタスク ID(htskid)、ミューテックスの待ち行列の先頭タスク ID(wtskid)を返します。

対象ミューテックスをロックしているタスクが存在しない場合は、htskid には TSK_NONE（0）が返ります。

対象ミューテックスに待ちタスクが無い場合は、wtskid には TSK_NONE（0）が返ります。

3.12 拡張同期・通信(メッセージバッファ)機能

表 3.23にメッセージバッファでサポートしているサービスコール一覧を示します。

表3.23 同期·通信（メッセージバッファ）サービスコール

項番	サービスコール *1	機 能	呼び出し可能な状態 *2						
			T	N	E	D	U	L	C
1	cre_mbf	メッセージバッファの生成	○		○	○	○		
	icre_mbf			○	○	○	○		
2	acre_mbf	メッセージバッファの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mbf			○	○	○	○		
3	del_mbf	メッセージバッファの削除	○		○	○	○		
4	snd_mbf	メッセージバッファへの送信	○		○		○		
5	psnd_mbf	同上(ポーリング)	○		○	○	○		
	ipsnd_mbf			○	○	○	○		
6	tsnd_mbf	同上(タイムアウト有)	○		○		○		
7	rcv_mbf	メッセージバッファからの受信	○		○		○		
8	prcv_mbf	同上(ポーリング)	○		○	○	○		
9	trcv_mbf	同上(タイムアウト有)	○		○		○		
10	ref_mbf	メッセージバッファの状態参照	○		○	○	○		
	iref_mbf			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.24にメッセージバッファ機能の仕様を示します。

表3.24 メッセージバッファの仕様

項番	項目	内容
1	メッセージバッファ ID	1～CFG_MAXMBFID (最大 1023)
2	サポート属性	TA_TFIFO : 送信待ちタスクのキューイングは FIFO TA_TPRI : 送信待ちタスクのキューイングは優先度順

3.12.1 メッセージバッファの生成(cre_mbf, icre_mbf)
(acre_mbf, iacre_mbf：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_mbf、icre_mbf》

ID	mbfid	R4	メッセージバッファ ID
T_CMBF	*pk_cmbf	R5	メッセージバッファ生成情報を格納したパケットへのポインタ

《acre_mbf、iacre_mbf》

T_CMBF	*pk_cmbf	R4	メッセージバッファ生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_mbf、icre_mbf》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_mbf、iacre_mbf》

ER_ID	mbfid	R0	生成したメッセージバッファの ID 番号（正の値）またはエラーコード

■パケットの構造

```
typedef   struct  t_cmbf {
          ATR     mbfatr;     +0    4   メッセージバッファ属性
          UINT    maxmsz;     +4    4   メッセージの最大サイズ（バイト数）
          SIZE    mbfsz;      +8    4   メッセージバッファ領域のサイズ（バイト数）
          VP      mbf;        +12   4   メッセージバッファ領域の先頭アドレス
} T_CMBF;
```

■エラーコード

E_NOMEM	[k]	メモリ不足(メッセージバッファ領域が確保できない)
E_RSATR	[p]	属性不正(mbfatr が不正)
E_PAR	[p]	パラメータエラー(pk_cmbf が 4 の倍数以外、mbfsz が 4 の倍数以外、maxmsz=0、maxmsz≧H'80000000、mbfsz が 0 以外で maxmsz+4＞mbfsz)
E_ID	[p]	不正 ID 番号(mbfid≦0、mbfid＞CFG_MAXMBFID)
E_OBJ	[k]	オブジェクト状態不正(mbfid のメッセージバッファが存在)
E_NOID	[k]	空き ID なし

■機能

cre_mbf, icre_mbf サービスコールは、mbfid で示された ID を持つメッセージバッファを、pk_cmbf で示された内容で生成します。

acre_mbf, iacre_mbf サービスコールは、未登録のメッセージバッファ ID を検索してその ID を持つメッセージバッファを pk_cmbf で示された内容で生成し、その ID をリターンパラメータとして返します。検索するメッセージバッファ ID の範囲は 1～CFG_MAXMBFID です。

mbfatr には属性として、メッセージ送信待ちタスクが待ち行列に並ぶ際の並び方を指定します。

```
mbfatr := (TA_TFIFO || TA_TPRI)
```

- TA_TFIFO（H'00000000）送信待ちタスクのキューイングは FIFO
- TA_TPRI（H'00000001）送信待ちタスクのキューイングは優先度順

メッセージ受信待ちタスクの待ち行列、およびメッセージ行列は、mbfatr に関わらず FIFO（First-In First-Out）となります。

メッセージバッファは、コンフィギュレータで指定したメッセージバッファ用領域(CFG_MBFSZ)から割り付けられます。生成に成功すると、メッセージバッファ用領域の空きは以下の式で計算されるサイズだけ減少します。

- 減少サイズ ＝ mbfsz＋16

mbfsz には、生成するメッセージバッファのサイズを指定します。指定するサイズは 4 の倍数でかつ最小バッファ（8 バイト）サイズ以上でなければなりません。このサイズを計算する際、1 つのメッセージを格納するごとにカーネルが管理のためにメッセージバッファを 4 バイト消費することを考慮してください。

また、mbfsz=0 でメッセージバッファを生成することもできます。mbfsz=0 で生成したメッセージバッファではバッファにメッセージを蓄えておくことができないため、メッセージ送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまりメッセージ送信側と受信側が完全に同期した動作となります。また、mbfsz=0 のメッセージバッファでは、メッセージバッファを介したコピー動作を伴わないというメリットもあります。

maxmsz には、生成するメッセージバッファで扱うことのできるメッセージの最大長を指定します。

mbf には、必ず NULL を指定してください。μITRON4.0 仕様では、NULL 以外を指定することで、mbf で指定された領域にメッセージバッファを生成する仕様になっていますが、本カーネルは NULL 以外を指定した場合の機能はサポートしていません。**NULL 以外の指定を行った場合の動作は保証されません。**

なお、メッセージバッファはコンフィギュレータで静的に生成することもできます。

3.12.2 メッセージバッファの削除(del_mbf)

■C 言語 API

```
ER ercd = del_mbf(ID mbfid);
```

■パラメータ

ID	mbfid	R4	メッセージバッファ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mbfid≦0、mbfid＞CFG_MAXMBFID)
E_NOEXS	[k]	未登録(mbfid のメッセージバッファが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mbfid で示されたメッセージバッファを削除します。

mbfid で示されたメッセージバッファにおいて、メッセージ受信またはメッセージ送信を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。また、メッセージバッファ内にメッセージが格納されていた場合でもエラーにはなりませんが、格納されていたメッセージは全て破棄されます。

削除により、メッセージバッファ用領域の空きは以下の式で計算されるサイズだけ増加します。

- 増加サイズ ＝ (生成時に指定した mbfsz) ＋ 16

3.12.3 メッセージバッファへの送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

■C 言語 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);
```

■パラメータ

ID	mbfid	R4	メッセージバッファ ID
VP	msg	R5	送信メッセージの先頭アドレス
UINT	msgsz	R6	送信メッセージのサイズ(バイト数)
《tsnd_mbf》			
TMO	tmout	R7	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了 (E_OK) またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(msg が 4 の倍数以外、msgsz=0、tmout≦-2)
	[k]	(msgsz＞maxmsz[*1])
E_ID	[p]	不正 ID 番号(mbfid≦0、mbfid＞CFG_MAXMBFID)
E_NOEXS	[k]	未登録(mbfid のメッセージバッファが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mbfid のメッセージバッファが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

*1 maxmsz：メッセージバッファ生成時に指定したメッセージの最大長

■機能

mbfid で示されたメッセージバッファに対して、msg で示されたメッセージを送信します。送信するサイズは msgsz で示されたバイト数です。

対象メッセージバッファに受信待ちタスクが存在する場合には、メッセージバッファには格納せずに受信待ち行列の先頭タスクにメッセージを渡し、そのタスクの待ち状態を解除します。

対象メッセージバッファに既に送信待ちタスクが存在する場合、snd_mbf, tsnd_mbf サービスコールではメッセージバッファの空き領域を待つための待ち行列（送信待ち行列）につながれ、psnd_mbf, ipsnd_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。送信待ち行列は、生成時に指定した属性にしたがって管理されます。

受信待ちタスクも送信待ちタスクも存在しない場合は、メッセージをメッセージバッファに格納します。この結果、メッセージバッファの空きサイズは、以下の式で算出されるサイズだけ減少します。

- 減少サイズ ＝ msgsz＋4

このサイズだけの空きがメッセージバッファに存在しない場合（バッファサイズが 0 の場合も含む）は、呼び出しタスクは送信待ち行列につながれます。

ipsnd_mbf は、非タスクコンテキストからも発行可能です。対象のメッセージバッファに TA_TPRI 属性が指定されている場合は、非タスクコンテキストはタスクよりも優先順位が高いため、バッファに空きがあれば、先に送信を待っているタスクが存在しても、バッファにメッセージがコピーされます。

tsnd_mbf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、psnd_mbf サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、snd_mbf サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.12.4 メッセージバッファからの受信(rcv_mbf, prcv_mbf, trcv_mbf)

■C 言語 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);
```

■パラメータ

ID	mbfid	R4	メッセージバッファ ID
VP	msg	R5	送信メッセージを返す領域への先頭アドレス
《trcv_mbf》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER_UINT	msgsz	R0	受信メッセージのサイズ（バイト数、正の値） またはエラーコード
VP	msg	R5	送信メッセージを格納した領域の先頭アドレス

■エラーコード

E_PAR	[p]	パラメータエラー(msg が 4 の倍数以外、tmout≦-2)
E_ID	[p]	不正 ID 番号(mbfid≦0、mbfid＞CFG_MAXMBFID)
E_NOEXS	[k]	未登録(mbfid のメッセージバッファが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mbfid のメッセージバッファが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

mbfid で示されたメッセージバッファからメッセージを受信し、受信したメッセージを msg の指す領域に格納します。また、受信したメッセージサイズをリターンパラメータとして返します。

メッセージバッファにメッセージがあれば、メッセージ行列先頭のメッセージ(最古のメッセージ)を受信します。メッセージバッファ内のメッセージを受信することで、メッセージバッファの空きサイズは以下の式で算出されるサイズだけ増加します。

- 増加サイズ ＝ msgsz＋4

この結果、空きサイズがメッセージ送信待ち行列先頭のタスクが送信しようとしていたメッセージサイズよりも大きくなると、そのメッセージがメッセージバッファに格納され、そのタスクの待ち状態が解除されます。送信待ち行列の以降のタスクに対してもメッセージの格納が可能であれば、待ち行列の順に同様の処理を行います。

メッセージバッファにメッセージが存在せず、メッセージ送信待ちタスクが存在する場合、メッセージ送信待ち行列先頭タスクのメッセージを受信します。この結果、そのメッセージ送信待ちタスクの待ち状態は解除されます。

メッセージバッファにメッセージがなく、メッセージ送信待ちタスクも存在しない場合、rcv_mbf, trcv_mbf サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。受信待ち行列は、FIFO で管理されます。

RENESAS

msg の指す領域として、cre_mbf, icre_mbf, acre_mbf, iacre_mbf サービスコールで指定した maxmsz 分の RAM 領域が必要です。

trcv_mbf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、prcv_mbf サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、rcv_mbf サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は､tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.12.5 メッセージバッファの状態参照(ref_mbf, iref_mbf)

■C 言語 API

```
ER ercd = ref_mbf(ID mbfid, T_RMBF *pk_rmbf);
ER ercd = iref_mbf(ID mbfid, T_RMBF *pk_rmbf);
```

■パラメータ

ID	mbfid	R4	メッセージバッファ ID
T_RMBF	*pk_rmbf	R5	メッセージバッファ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RMBF	*pk_rmbf	R5	メッセージバッファ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rmbf {
           ID      stskid;     +0     2    送信待ち行列の先頭タスク ID
           ID      rtskid;     +2     2    受信待ち行列の先頭タスク ID
           UINT    smsgcnt;    +4     4    メッセージバッファに入っているメッセージの数
           SIZE    fmbfsz;     +8     4    空きバッファのサイズ（バイト数）
} T_RMBF;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmbf が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(mbfid≦0、mbfid＞CFG_MAXMBFID)
E_NOEXS	[k]	未登録(mbfid のメッセージバッファが存在しない)

■機能

mbfid で示されたメッセージバッファの状態を参照し、pk_rmbf が指す領域に送信待ちタスク ID (stskid)、受信待ちタスク ID(rtskid)、メッセージバッファに入っているメッセージの数(smsgcnt)、および空きバッファサイズ(fmbfsz)を返します。

受信待ちタスク、送信待ちタスクが無い場合は、待ちタスク ID として TSK_NONE（0）を返します。

3.13 メモリプール管理(固定長メモリプール)機能

表 3.25に固定長メモリプールでサポートしているサービスコール一覧を示します。

表3.25 メモリプール管理（固定長メモリプール）サービスコール

項番	サービスコール *1		機 能	呼び出し可能な状態 *2						
				T	N	E	D	U	L	C
1	cre_mpf	[s]	固定長メモリプールの生成	○		○	○	○		
	icre_mpf				○	○	○	○		
2	acre_mpf		固定長メモリプールの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mpf				○	○	○	○		
3	del_mpf		固定長メモリプールの削除	○		○	○	○		
4	get_mpf	[S]	固定長メモリブロックの獲得	○		○		○		
5	pget_mpf	[S]	同上(ポーリング)	○		○	○	○		
	ipget_mpf				○	○	○	○		
6	tget_mpf	[S]	同上(タイムアウト有)	○		○		○		
7	rel_mpf	[S]	固定長メモリブロックの返却	○		○	○	○		
	irel_mpf				○	○	○	○		
8	ref_mpf		固定長メモリプールの状態参照	○		○	○	○		
	iref_mpf				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.26に固定長メモリプールの仕様を示します。

表3.26 固定長メモリプールの仕様

項番	項目	内容
1	固定長メモリプール ID	1～CFG_MAXMPFID(最大 1023)
2	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順
3	管理方式	メモリプール領域内に管理情報を置くかどうかを、コンフィギュレータの CFG_MPFMANAGE で選択できます。

3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf) (acre_mpf, iacre_mpf：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_mpf、icre_mpf》

ID	mpfid	R4	固定長メモリプール ID
T_CMPF	*pk_cmpf	R5	固定長メモリプール生成情報を格納したパケットへのポインタ

《acre_mpf、iacre_mpf》

T_CMPF	*pk_cmpf	R4	固定長メモリプール生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_mpf、icre_mpf》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_mpf、iacre_mpf》

ER_ID	mpfid	R0	生成した固定長メモリプールの ID 番号（正の値）またはエラーコード

■パケットの構造

（1） CFG_MPFMANAGE をチェックしない場合

```
typedef    struct t_cmpf {
           ATR     mpfatr;     +0     4    固定長メモリプール属性
           UINT    blkcnt;     +4     4    メモリプール全体のブロック数
           UINT    blksz;      +8     4    固定長メモリブロックサイズ（バイト数）
           VP      mpf;        +12    4    固定長メモリプール領域の先頭アドレス
} T_CMPF;
```

（2） CFG_MPFMANAGE をチェックした場合

```
typedef    struct t_cmpf {
           ATR     mpfatr;     +0     4    固定長メモリプール属性
           UINT    blkcnt;     +4     4    メモリプール全体のブロック数
           UINT    blksz;      +8     4    固定長メモリブロックサイズ（バイト数）
           VP      mpf;        +12    4    固定長メモリプール領域の先頭アドレス
           VP      mpfmb;      +16    4    固定長メモリブロック管理領域の先頭アドレス
} T_CMPF;
```

■エラーコード

E_NOMEM	[k]	メモリ不足(メモリプール用の領域が確保できない)
E_RSATR	[p]	属性不正(mpfatr が不正)
E_PAR	[p]	パラメータエラー(pk_cmpf が 4 の倍数以外、blkcnt=0、 blksz が 4 の倍数以外、blksz=0、 mpf が NULL 以外で 4 の倍数以外、 mpfmb が 4 の倍数以外(CFG_MPFMANAGE チェック有りの場合のみ)
	[k]	TSZ_MPF(blkcnt, blksz)が 32 ビット範囲を超えている))
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_OBJ	[k]	オブジェクト状態不正(mpfid の固定長メモリプールが存在)
E_NOID	[k]	空き ID なし

■機能

cre_mpf, icre_mpf サービスコールは、mpfid で示された ID を持つ固定長メモリプールを、pk_cmpf で示された内容で生成します。

acre_mpf, iacre_mpf サービスコールは、未登録の固定長メモリプール ID を検索してその ID を持つ固定長メモリプールを pk_cmpf で示された内容で生成し、その ID をリターンパラメータとして返します。未登録の固定長メモリプール ID を検索する範囲は 1～CFG_MAXMPFID です。

mpfatr には属性として、メモリブロック獲得を待つ待ち行列に並ぶ際の並び方を指定します。

```
mpfatr := (TA_TFIFO‖TA_TPRI)
```

- TA_TFIFO（H'00000000）メモリブロック獲得待ちタスクのキューイングはFIFO
- TA_TPRI（H'00000001）メモリブロック獲得待ちタスクのキューイングは優先度順

blkcnt には、生成するメモリプールの総ブロック数を指定します。

blksz には、メモリブロックのサイズを指定します。指定するサイズは 4 の倍数でなければなりません。

mpf に NULL を指定した場合は、カーネルが自動的に固定長メモリプールを確保します。その際、固定長メモリプールは、コンフィギュレータで指定した固定長メモリプール用領域(CFG_MPFSZ)から割り付けられます。生成に成功すると、固定長メモリプール用領域の空きは以下の式で計算されるサイズだけ減少します。

(1) CFG_MPFMANAGEをチェックしない場合
減少サイズ ＝ （blksz+4）×blkcnt＋16

(2) CFG_MPFMANAGEをチェックした場合
減少サイズ ＝ blksz×blkcnt＋16

mpf に生成する固定長メモリプールのアドレスを指定することもできます。この場合、固定長メモリプールとして TSZ_MPF(blkcnt, blksz)で算出されるサイズの領域を確保し、そのアドレスを mpf に指定してください。なお、TSZ_MPF()マクロの定義内容は、CFG_MPFMANAGE のチェックの有無によって異なります。

CFG_MPFMANAGE をチェックした場合は、mpfmb に固定長メモリブロック管理領域の先頭アドレスを指定しなければなりません。具体的には、VTSZ_MPFMB(blkcnt, blksz)で算出されるサイズの領域を確保し、そのアドレスを mpfmb に指定しなければなりません。

mpfmb は、μITRON4.0 仕様の範囲外のメンバです。

なお、固定長メモリプールはコンフィギュレータで静的に生成することもできます。

3.13.2 固定長メモリプールの削除(del_mpf)

■C 言語 API

```
ER ercd = del_mpf(ID mpfid);
```

■パラメータ

ID	mpfid	R4	固定長メモリプール ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mpfid≦ 0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[k]	未登録(mpfid の固定長メモリプールが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mpfid で示された固定長メモリプールを削除します。

mpfid で示された固定長メモリプールにおいて、メモリ獲得を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

固定長メモリプール用領域(CFG_MPFSZ)から割り付けられた固定長メモリプール(生成時に mpf に NULL を指定)を削除すると、固定長メモリプール用領域の空きは以下の式で計算されるサイズだけ増加します。

(1) CFG_MPFMANAGEをチェックしない場合
増加サイズ ＝ ((生成時に指定したblksz) + 4)×(生成時に指定したblkcnt)＋16

(2) CFG_MPFMANAGEをチェックした場合
増加サイズ ＝ (生成時に指定したblksz)×(生成時に指定したblkcnt)＋16

なお、すでに獲得済みのブロックがあっても、カーネルはそれに関して何も処理しません。

3.13.3 固定長メモリブロックの獲得(get_mpf, pget_mpf, ipget_mpf, tget_mpf)

■C 言語 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);
```

■パラメータ

ID	mpfid	R4	固定長メモリプール ID
VP	*p_blk	R5	メモリブロック先頭アドレスを返す領域へのポインタ
《tget_mpf》			
TMO	tmout	R6	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
VP	*p_blk	R5	メモリブロック先頭アドレスを格納した領域へのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_blk が 4 の倍数以外、tmout≦-2)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[k]	未登録(mpfid の固定長メモリプールが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mpfid の固定長メモリプールが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

■機能

mpfid で示される固定長メモリプールからひとつのメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを p_blk の指す領域に返します。

既にメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが対象となる固定長メモリプールに空きブロックが存在しない場合は、get_mpf, tget_mpf サービスコールでは呼び出しタスクはそのメモリプールのメモリ獲得の待ち行列につながれ、pget_mpf, ipget_mpf サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

tget_mpf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、pget_mpf サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合は、タイムアウト監視を行いません。したがって、get_mpf サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.13.4 固定長メモリブロックの返却(rel_mpf, irel_mpf)

■C 言語 API

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

■パラメータ

ID	mpfid	R4	固定長メモリプール ID
VP	blk	R5	メモリブロックの先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(blk が 4 の倍数以外)
	[k]	(メモリブロックの先頭アドレス以外、またはすでに返却した blk を指定)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[k]	未登録(mpfid の固定長メモリプールが存在しない)

■機能

mpfid で示された固定長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpf、pget_mpf, ipget_mpf または tget_mpf サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

対象固定長メモリプールでメモリブロックの獲得を待っているタスクがある場合、本サービスコールで返却したブロックを待ち行列先頭のタスクに割り付け、待ち状態を解除します。

RENESAS

3.13.5 固定長メモリプールの状態参照(ref_mpf, iref_mpf)

■C 言語 API

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

■パラメータ

ID	mpfid	R4	固定長メモリプール ID
T_RMPF	*pk_rmpf	R5	固定長メモリプール状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RMPF	*pk_rmpf	R5	固定長メモリプール状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rmpf {
           ID      wtskid;     +0     2    待ちタスク ID
           UINT    fblkcnt;    +4     4    空き領域のブロック数
} T_RMPF;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmpf が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[k]	未登録(mpfid の固定長メモリプールが存在しない)

■機能

mpfid で示された固定長メモリプールの状態を参照します。
pk_rmpf の指す領域に待ちタスク ID(wtskid)、空き領域のブロック数(fblkcnt)を返します。
対象メモリプールの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE (0) を返します。

3.14 メモリプール管理(可変長メモリプール)機能

表 3.27に可変長メモリプールでサポートしているサービスコール一覧を示します。

表3.27 メモリプール管理（可変長メモリプール）サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	cre_mpl	可変長メモリプールの生成	○		○	○	○		
	icre_mpl			○	○	○	○		
2	acre_mpl	可変長メモリプールの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mpl			○	○	○	○		
3	del_mpl	可変長メモリプールの削除	○		○	○	○		
4	get_mpl	可変長メモリブロックの獲得	○		○		○		
5	pget_mpl	同上(ポーリング)	○		○	○	○		
	ipget_mpl			○	○	○	○		
6	tget_mpl	同上(タイムアウト有)	○		○		○		
7	rel_mpl	可変長メモリブロックの返却	○		○	○	○		
	irel_mpl			○	○	○	○		
8	ref_mpl	可変長メモリプールの状態参照	○		○	○	○		
	iref_mpl			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.28に可変長メモリプールの仕様を示します。

表3.28 可変長メモリプールの仕様

項番	項目	内容
1	可変長メモリプール ID	1～CFG_MAXMPLID(最大 1023)
2	管理方式	コンフィギュレータの CFG_NEWMPL を選択すると、以下が改善されます。 (1) 特に多くのメモリブロックを扱うメモリプールで、獲得/返却処理が高速化されます。 (2) 空き領域の断片化を軽減する VTA_UNFRAGMENT 属性を使用できます。
3	サポート属性	・TA_TFIFO : 待ちタスクのキューイングは FIFO ・VTA_UNFRAGMENT：セクタ管理方式 (空き領域の断片化が発生しにくい方式、CFG_NEWMPL 選択時のみ指定可能)

可変長メモリプールでは、空き領域が断片化する問題があります。VTA_UNFRAGMENT 属性は、これを軽減する機能です。「2.15.2 空き領域の断片化とその対策」も参照してください。

RENESAS

3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)
(acre_mpl, iacre_mpl：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_mpl、icre_mpl》

ID	mplid	R4	可変長メモリプール ID
T_CMPL	*pk_cmpl	R5	可変長メモリプール生成情報を格納したパケットへのポインタ

《acre_mpl、iacre_mpl》

T_CMPL	*pk_cmpl	R4	可変長メモリプール生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_mpl、icre_mpl》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_mpl、iacre_mpl》

ER_ID	mplid	R0	生成した可変長メモリプールの ID 番号（正の値）またはエラーコード

■パケットの構造

(1)CFG_NEWMPL をチェックしない場合

```
typedef    struct  t_cmpl {
           ATR     mplatr;    +0    4   可変長メモリプール属性
           SIZE    mplsz;     +4    4   メモリプール全体のサイズ（バイト数）
           VP      mpl;       +8    4   可変長メモリプール領域の先頭アドレス
}T_CMPL;
```

(2)CFG_NEWMPL をチェックした場合

```
typedef    struct  t_cmpl {
           ATR     mplatr;    +0    4   可変長メモリプール属性
           SIZE    mplsz;     +4    4   メモリプール全体のサイズ（バイト数）
           VP      mpl;       +8    4   可変長メモリプール領域の先頭アドレス
           VP      mplmb;     +12   4   可変長メモリプール管理領域の先頭アドレス
           UINT    minblksz;  +16   4   最小ブロックサイズ
           UINT    sctnum;    +20   4   最大セクタ数
}T_CMPL;
```

■エラーコード

E_NOMEM	[k]	メモリ不足(メモリプール用の領域が確保できない)
E_RSATR	[p]	属性不正(mplatr が不正)
E_PAR	[p]	パラメータエラー
		pk_cmpl が 4 の倍数以外、
		mplsz が 4 の倍数以外、
		mplsz＜TSZ_MPL(1,4)
		mplsz≧H'80000000、
		mpl が NULL 以外で 4 の倍数以外
		VTA_UNFRAGMENT 属性の場合で、minblksz が 0
		VTA_UNFRAGMENT 属性の場合で、sctnum=0
		VTA_UNFRAGMENT 属性の場合で、mplsz<minblksz*32
		VTA_UNFRAGMENT 属性の場合で、mplmb が 4 の倍数以外
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_OBJ	[k]	オブジェクト状態不正(mplid の可変長メモリプールが存在)
E_NOID	[k]	空き ID なし

RENESAS

■機能

cre_mpl, icre_mpl サービスコールは、mplid で示された ID を持つ可変長メモリプールを、pk_cmpl で示された内容で生成します。

acre_mpl, iacre_mpl サービスコールは、未登録の可変長メモリプール ID を検索してその ID を持つ可変長メモリプールを pk_cmpl で示された内容で生成し、その ID をリターンパラメータとして返します。検索する可変長メモリプール ID の範囲は 1～CFG_MAXMPLID です。

(1) mplatr

mplatr には、以下の論理和を指定してください。

(a) メモリブロック獲得を待つ待ち行列に並ぶ際の並び方

TA_TFIFO のみを指定できます。

- TA_TFIFO(H'00000000)：メモリ獲得待ちタスクのキューイングは FIFO 順

(b) 管理方式

CFG_NEWMPL を選択していた場合は、VTA_UNFRAGMENT を指定できます。

- VTA_UNFRAGMENT(H'80000000)：セクタ管理方式(空き領域の断片化が発生しにくい方式)

VTA_UNFRAGMENT 属性は、微小なメモリブロックを大量に獲得するメモリプールに適した属性で、微小なブロックをできるだけ連続して配置することで、大きなサイズの連続空き領域が維持されやすくします。

VTA_UNFRAGMENT 属性を指定した場合のみ、mplmb, minblksz, sctnum が有効になります。sctnum に mplsz/(minblksz×32)よりも大きい値を指定した場合は、mplsz/(minblksz×32)として扱います。

詳細は、「2.15.2 空き領域の断片化とその対策」を参照してください。

(2) mplsz

mplsz には、生成する可変長メモリプール領域のサイズを指定します。「2.15.3 可変長メモリプールの管理」も参照してください。

なお、mplsz に指定すべきサイズの目安を知るために、以下のマクロが用意されています。

SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得するのに必要な可変長メモリプール領域のサイズ(目安のバイト数)

このマクロは、VTA_UNFRAGMENT 属性の指定が無い前提で計算します。また、CFG_NEWMPL の選択有無によって計算式が異なります。

(3) mpl

mpl には、可変長メモリプールとして使用する空き領域の先頭アドレスを指定します。mpl から mplsz バイトを可変長メモリプールとして使用します。

mpl に NULL を指定すると、カーネルは可変長メモリプール用領域(CFG_MPLSZ)から mplsz バイトのメモリプール領域を割り付けます。これにより、可変長メモリプール用領域の空きは以下の式で計算されるサイズだけ減少します。

- 減少サイズ ＝ mplsz ＋ 16

(4) mplmb

これはμITRON 仕様外の項目です。

mplmb は、VTA_UNFRAGMENT 属性を指定した場合のみ有効です。その他の場合は単に無視されます。

以下のマクロで算出されるサイズの領域を確保し、その先頭アドレスを mplmb に指定してください。

VTSZ_MPLMB(最大セクタ数)

(5) minblksz と sctnum

これらはμITRON 仕様外の項目です。

これらは VTA_UNFRAGMENT 属性を指定した場合のみ有効です。詳細は、前述の VTA_UNFRAGMENT 属性の説明を参照してください。

なお、可変長メモリプールはコンフィギュレータで静的に生成することもできます。

■補足

通常、獲得するメモリブロックのアドレスのアライメントは 4 です。

メモリブロックのアドレスを、キャッシュラインサイズ(16 または 32)のアドレスにアライメントするには、以下のようにしてください。ここでは、アライメント数を N と表記します。

(1) CFG_NEWMPL 選択時、VTA_UNFRAGMENT 属性なし

- アプリケーション側で、N バイト境界アドレスにメモリプール領域を確保し、メモリプール生成時にそのアドレスを指定してください。
- 獲得するメモリブロックのサイズは、全て N の倍数としてください。

(2) CFG_NEWMPL 選択時、VTA_UNFRAGMENT 属性あり

- アプリケーション側で、N バイト境界のアドレスにメモリプール領域を確保し、メモリプール生成時にそのアドレスを指定してください。
- 最小ブロックサイズは、N としてください。
- 獲得するメモリブロックのサイズは、全て N の倍数としてください。

(3) CFG_NEWMPL 非選択時

1. N=16の場合

- アプリケーション側で、16 バイト境界アドレスにメモリプール領域を確保し、メモリプール生成時にそのアドレスを指定してください。
- 獲得するメモリブロックのサイズは、全て 16 の倍数としてください。

2. N=32の場合

- アプリケーション側で、「32 バイト境界アドレス-16」のアドレスにメモリプール領域をに確保し、メモリプール生成時にそのアドレスを指定してください。
- 獲得するメモリブロックのサイズは、全て「(N の倍数)+16」としてください。

3.14.2　可変長メモリプールの削除(del_mpl)

■C 言語 API

```
ER ercd = del_mpl(ID mplid);
```

■パラメータ

ID	mplid	R4	可変長メモリプール ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[k]	未登録(mplid の可変長メモリプールが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

mplid で示された可変長メモリプールを削除します。

mplid で示された可変長メモリプールにおいて、メモリ獲得を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

可変長メモリプール用領域(CFG_MPLSZ)から割り付けられた可変長メモリプール(生成時に mpl に NULL を指定)を削除すると、可変長メモリプールプール用領域の空きは以下の式で計算されるサイズだけ増加します。

- 増加サイズ ＝ (生成時に指定した mplsz) ＋ 16

なお、すでに獲得済みのブロックがあっても、カーネルはそれに関して何も処理しません。

3.14.3 可変長メモリブロックの獲得(get_mpl, pget_mpl, ipget_mpl, tget_mpl)

■C 言語 API

```
ER ercd = get_mpl(ID mplid, UINT blksz, VP *p_blk);
ER ercd = pget_mpl(ID mplid, UINT blksz, VP *p_blk);
ER ercd = ipget_mpl(ID mplid, UINT blksz, VP *p_blk);
ER ercd = tget_mpl(ID mplid, UINT blksz, VP *p_blk, TMO tmout);
```

■パラメータ

ID	mplid	R4	可変長メモリプール ID
UINT	blksz	R5	メモリブロックサイズ（バイト数）
VP	*p_blk	R6	メモリブロックの先頭アドレスを返す領域へのポインタ
《tget_mpl》			
TMO	tmout	R7	タイムアウト指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
VP	*p_blk	R6	メモリブロックの先頭アドレスを格納した領域のポインタ

■エラーコード

E_PAR	[p]	パラメータエラー(p_blk が 4 の倍数以外、blksz が 4 の倍数以外または 0、tmout≦-2)
	[k]	(mplsz[*1]-16＜blksz)
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[k]	未登録(mplid の可変長メモリプールが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_DLT	[k]	待ちオブジェクト削除(mplid の可変長メモリプールが削除された)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)

*1 **可変長メモリプール生成時に指定したメモリプールサイズ**

■機能

mplid で示される可変長メモリプールから、blksz で示されるサイズ（バイト数）のメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを p_blk の指す領域に返します。

メモリブロックの獲得により、可変長メモリプールの空きサイズが減少します。詳細は、「2.15.3 可変長メモリプールの管理」を参照してください。

既にメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが上記サイズの連続した空き領域が存在しない場合は、get_mpl, tget_mpl サービスコールでは呼び出しタスクはそのメモリプールのメモリ獲得の待ち行列につながれ、pget_mpl, ipget_mpl サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、FIFO で管理されます。

tget_mpl サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL（0）を指定した場合、pget_mpl サービスコールと同じ処理を行います。tmout=TMO_FEVR（-1）を指定した場合、タイムアウト監視を行いません。したがって、get_mpl サービスコールと同じ処理を行います。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、tmout に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.14.4 可変長メモリブロックの返却(rel_mpl, irel_mpl)

■C 言語 API

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

■パラメータ

ID	mplid	R4	可変長メモリプール ID
VP	blk	R5	メモリブロックの先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(blk が 4 の倍数以外)
	[k]	(メモリブロックの先頭アドレス以外、またはすでに返却した blk を指定)
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[k]	未登録(mplid の可変長メモリプールが存在しない)

■機能

mplid で示された可変長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpl、pget_mpl, ipget_mpl または tget_mpl サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

メモリブロックの返却により、可変長メモリプールの空きサイズが増加します。詳細は、「2.15.3 可変長メモリプールの管理」を参照してください。

この結果、対象可変長メモリプールでメモリブロックの獲得待ち行列の先頭タスクが要求するだけの連続空き領域ができると、そのタスクにメモリブロックを割り付けて待ち状態を解除します。待ち行列の以降のタスクに対してもメモリブロックを割り付け可能であれば、待ち行列の順に同様の処理を行います。

3.14.5 可変長メモリプールの状態参照(ref_mpl, iref_mpl)

■C 言語 API

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

■パラメータ

ID	mplid	R4	可変長メモリプール ID
T_RMPL	*pk_rmpl	R5	可変長メモリプール状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RMPL	*pk_rmpl	R5	可変長メモリプール状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rmpl {
           ID      wtskid;      +0     2    待ちタスク ID
           SIZE    fmplsz;      +4     4    空き領域の合計サイズ（バイト数）
           UINT    fblksz;      +8     4    獲得可能な最大メモリブロックサイズ（バイト数）
} T_RMPL;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmpl が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[k]	未登録(mplid の可変長メモリプールが存在しない)

■機能

mplid で示された可変長メモリプールの状態を参照します。

pk_rmpl が指す領域に待ちタスク ID(wtskid)、現在の空き領域の合計サイズ(fmplsz)、獲得可能な最大メモリブロックのサイズ(fblksz)を返します。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。1 回の get_mpl、pget_mpl, ipget_mpl または tget_mpl サービスコールで、fblksz までのブロックを即座に獲得できます。

対象メモリプールの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE (0) を返します。

3.15 時間管理機能(システム時刻管理)

表 3.29にシステム時刻管理でサポートしているサービスコール一覧を示します。

表3.29 システム時刻管理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	set_tim [S]	システム時刻の設定	○		○	○	○		
	iset_tim			○	○	○	○		
2	get_tim [S]	システム時刻の参照	○		○	○	○		
	iget_tim			○	○	○	○		
3	isig_tim [S]	タイムティックの供給	(CFG_TIMUSE のチェックにより自動的に実行されるようになります)						

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.30にシステム時刻管理の仕様を示します。

表3.30 システム時刻管理の仕様

項番	項目	内容
1	システム時刻値	符号なし 48 ビット
2	システム時刻の単位	1[ms]
3	システム時刻の更新周期	CFG_TICNUME/CFG_TICDENO[ms]*
4	システム時刻初期値(初期起動時)	H'000000000000

【注】* kernel_macro.h に定義される TIC_NUME, TIC_DENO は、それぞれ CFG_TICNUME, CFG_TICDENO と同じ値です。

システム時刻は SYSTIM 型の構造体によって符号無し 48bit 整数として表現されますが、その最大値は以下のようになります。

[CFG_TICNUME/CFG_TICDENO ≦ 1 の場合]
最大値＝H'7fffffffffff/CFG_TICDENO
[CFG_TICNUME/CFG_TICDENO ＞ 1 の場合]
最大値＝H'7fffffffffff

タイマ割込み(isig_tim)によってシステム時刻を更新する際に上記最大値を超える場合には、システム時刻は 0 に戻ります。

また、set_tim, iset_tim サービスコールで、上記最大値を超える値を指定した場合の動作は保証されません。

RENESAS

3.15.1 システム時刻の設定(set_tim, iset_tim)

■C 言語 API

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

■パラメータ

SYSTIM	*p_systim	R4	設定するシステム時刻を示すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef    struct  systim {
           UH      utime;       0      2    システムの現在時刻（上位）
           UW      ltime;       +4     4    システムの現在時刻（下位）
} SYSTIM;
```

■エラーコード

E_PAR	[p]	パラメータエラー（p_systim が 4 の倍数以外）

■機能

システムが保持しているシステム時刻の現在の値を、p_systim で示される値に設定します。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、指定可能な最大値は H'7fffffffffff/CFG_TICDENO となります。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.15.2 システム時刻の参照(get_tim, iget_tim)

■C 言語 API

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

■パラメータ

SYSTIM	*p_systim	R4	システムの現在時刻を返すパケットの先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
SYSTIM	*p_systim	R4	システムの現在時刻を格納したパケットの先頭アドレス

■パケットの構造

```
typedef  struct  systim {
         UH      utime;      0     2   システムの現在時刻（上位）
         UW      ltime;      +4    4   システムの現在時刻（下位）
} SYSTIM;
```

■エラーコード

E_PAR [p] パラメータエラー(p_systim が 4 の倍数以外)

■機能

システム時刻の現在値を読み出し、その結果を p_systim の指す領域に返します。

3.15.3 タイムティックの供給(isig_tim)

■機能

システム時刻を更新します。

CFG_TIMUSE を選択すると、CFG_TICDENO/CFG_TICNUME[ms]で計算される周期で、自動的に isig_tim サービスコール処理が実行されるようにコンフィギュレーションされます。つまり、本機能はサービスコールではありませんでの、アプリケーションから呼び出すことはできません。

タイムティックの供給時には、カーネルは時間に関する次のような処理を行います。

(1) システム時刻の更新（+1）

(2) タイムイベントハンドラの起動

(3) tslp_tskサービスコールなどのタイムアウト付きサービスコールで待ち状態になっているタスクのタイムアウト処理

カーネルの時間に関する機能を使用するには、タイマドライバの組み込みが必要です。詳細は、「付録 D タイマドライバ」を参照してください。

3.16 時間管理機能(周期ハンドラ)

表 3.31に周期ハンドラでサポートしているサービスコール一覧を示します。

表3.31 周期ハンドラサービスコール

項番	サービスコール *1		機 能	呼び出し可能なシステム状態 *2						
				T	N	E	D	U	L	C
1	cre_cyc	[s]	周期ハンドラの生成	○		○	○	○		
	icre_cyc				○	○	○	○		
2	acre_cyc		周期ハンドラの生成	○		○	○	○		
	iacre_cyc		(ID 番号自動割付け)		○	○	○	○		
3	del_cyc		周期ハンドラの削除	○		○	○	○		
4	sta_cyc	[S]	周期ハンドラの動作開始	○		○	○	○		
	ista_cyc				○	○	○	○		
5	stp_cyc	[S]	周期ハンドラの動作停止	○		○	○	○		
	istp_cyc				○	○	○	○		
6	ref_cyc		周期ハンドラの状態参照	○		○	○	○		
	iref_cyc				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.32に周期ハンドラの仕様を示します。

表3.32 周期ハンドラの仕様

項番	項目	内容
1	周期ハンドラ ID	1～CFG_MAXCYCID (最大 14)
2	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述 TA_STA : 周期ハンドラの動作開始 TA_PHS : 起動位相の保存

RENESAS

3.16.1 周期ハンドラの生成(cre_cyc, icre_cyc)
(acre_cyc, iacre_cyc：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_cyc、icre_cyc》

ID	cycid	R4	周期ハンドラ ID
T_CCYC	*pk_ccyc	R5	周期ハンドラ生成情報を格納したパケットへのポインタ

《acre_cyc、iacre_cyc》

T_CCYC	*pk_ccyc	R4	周期ハンドラ生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_cyc、icre_cyc》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_cyc、iacre_cyc》

ER_ID	cycid	R0	生成した周期ハンドラの ID 番号（正の値） またはエラーコード

■パケットの構造

```
typedef    struct  t_ccyc {
           ATR     cycatr;     0     4    周期ハンドラ属性
           VP_INT  exinf;      +4    4    拡張情報
           FP      cychdr;     +8    4    周期ハンドラアドレス
           RELTIM  cyctim;     +12   4    周期ハンドラの起動周期
           RELTIM  cycphs;     +16   4    周期ハンドラの起動位相
} T_CCYC;
```

■エラーコード

E_RSATR	[p]	属性不正(cycatr が不正)
E_PAR	[p]	パラメータエラー(pk_ccyc が 4 の倍数以外、cyctim=0、cycphs＞cyctim、cychdr が奇数)
E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_OBJ	[k]	オブジェクト状態不正(cycid の周期ハンドラが存在)
E_NOID	[k]	空き ID なし

■機能

cre_cyc, icre_cyc サービスコールは、cycid で示された周期ハンドラを pk_ccyc で示された内容で生成します。

acre_cyc, iacre_cyc サービスコールは、未登録の周期ハンドラ ID を検索してその ID を持つ周期ハンドラを pk_ccyc で示された内容で定義し、その ID をリターンパラメータとして返します。検索する周期ハンドラ ID の範囲は 1～CFG_MAXCYCID です。

周期ハンドラは、一定周期で起動される非タスクコンテキストのタイムイベントハンドラです。

cycatr には属性として、ハンドラを記述した言語や起動属性を指定します。

```
cycatr := ((TA_HLNG || TA_ASM) | [TA_STA] | [TA_PHS])
```

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述
- TA_STA（H'00000002）周期ハンドラの動作開始
- TA_PHS（H'00000004）起動位相の保存

TA_STA が指定された場合には、周期ハンドラを生成した後に、周期ハンドラを動作している状態にします。指定されていない場合は、sta_cyc, ista_cyc サービスコールが呼び出されるまで周期ハンドラは動作しません。TA_PHS が指定された場合は、周期ハンドラの動作を開始する時に、周期ハンドラの起動位相を保存して、次に起動すべき時刻を決定します。指定されていない場合は、sta_cyc, ista_cyc サービスコールが呼び出された時刻を基準として、周期ハンドラを次に起動する時刻を決定します。

exinf は、周期ハンドラを起動するときに、パラメータとして渡す拡張情報を指定します。周期ハンドラに関する情報を設定するなどの目的でユーザが自由に使用できます。

cychdr には、周期ハンドラの先頭アドレスを指定します。

cyctim には、周期ハンドラの起動周期を指定します。

cycphs には、周期ハンドラの起動位相を指定します。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、cyctim, cycphs に指定可能な最大値は H'7fffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

周期ハンドラを生成するサービスコールが呼び出されてから、指定した cycphs(起動位相)以上の時間が経過した時が、周期ハンドラを一回目に起動すべき時刻となります。その後は cyctim(起動周期)が経過する毎に、周期ハンドラが起動されます。

なお、周期ハンドラはコンフィギュレータで静的に生成することもできます。

3.16.2 周期ハンドラの削除(del_cyc)

■C 言語 API

```
ER ercd = del_cyc(ID cycid);
```

■パラメータ

ID	cycid	R4	周期ハンドラ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[k]	未登録(cycid の周期ハンドラが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

cycid で示された周期ハンドラを削除します。

3.16.3 周期ハンドラの動作開始(sta_cyc, ista_cyc)

■C 言語 API

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

■パラメータ

ID	cycid	R4	周期ハンドラ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(cycid≦ 0、cycid＞CFG_MAXCYCID)
E_NOEXS	[k]	未登録(cycid の周期ハンドラが存在しない)

■機能

cycid で示された周期ハンドラを、動作している状態に移行させます。

周期ハンドラ属性に TA_PHS が指定されていない場合には、このサービスコールが呼び出された時刻を基準として、その時刻から起動周期が経過する毎に、周期ハンドラが起動されます。

TA_PHS が指定されていない動作している状態の周期ハンドラが指定された場合は、周期ハンドラを次に起動する時刻の再設定のみを行います。

TA_PHS が指定されている場合は、周期ハンドラ生成時点の時刻を基準に起動するため、時刻の設定は行いません。

3.16.4 周期ハンドラの動作停止(stp_cyc, istp_cyc)

■C 言語 API

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

■パラメータ

ID	cycid	R4	周期ハンドラ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[k]	未登録(cycid の周期ハンドラが存在しない)

■機能

cycid で示された周期ハンドラを、動作していない状態に移行させます。

3.16.5 周期ハンドラの状態参照(ref_cyc, iref_cyc)

■C 言語 API

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

■パラメータ

ID	cycid	R4	周期ハンドラ ID
T_RCYC	*pk_rcyc	R5	周期ハンドラの状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RCYC	*pk_rcyc	R5	周期ハンドラの状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rcyc {
           STAT    cycstat;     +0     4    周期ハンドラの動作状態
           RELTIM  lefttim;     +4     4    周期ハンドラ起動までの残り時間
} T_RCYC;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rcyc が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[k]	未登録(cycid の周期ハンドラが存在しない)

■機能

cycid で示された周期ハンドラの状態を参照し、pk_rcyc が指す領域に周期ハンドラの動作状態(cycstat)、周期ハンドラ起動までの残り時間(lefttim)を返します。

cycstat には、対象周期ハンドラの動作状態を返します。

- TCYC_STP（H'00000000）周期ハンドラが動作していない
- TCYC_STA（H'00000001）周期ハンドラが動作している

lefttim には、対象周期ハンドラを次に起動する時刻までの相対時間を返します。対象周期ハンドラが動作していない場合、lefttim は不定値となります。

RENESAS

3.17　時間管理機能(アラームハンドラ)

表 3.33にアラームハンドラでサポートしているサービスコール一覧を示します。

表3.33　アラームハンドラサービスコール

項番	サービスコール　*1	機　能	呼び出し可能なシステム状態　*2						
			T	N	E	D	U	L	C
1	cre_alm	アラームハンドラの生成	○		○	○	○		
	icre_alm			○	○	○	○		
2	acre_alm	アラームハンドラの生成	○		○	○	○		
	iacre_alm	(ID 番号自動割付け)		○	○	○	○		
3	del_alm	アラームハンドラの削除	○		○	○	○		
4	sta_alm	アラームハンドラの動作開始	○		○	○	○		
	ista_alm			○	○	○	○		
5	stp_alm	アラームハンドラの動作停止	○		○	○	○		
	istp_alm			○	○	○	○		
6	ref_alm	アラームハンドラの状態参照	○		○	○	○		
	iref_alm			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.34にアラームハンドラの仕様を示します。

表3.34　アラームハンドラの仕様

項番	項目	内容
1	アラームハンドラ指定番号	1〜CFG_MAXALMID (最大 15)
2	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述

3.17.1 アラームハンドラの生成(cre_alm, icre_alm)
(acre_alm, iacre_alm：ID 番号自動割付け)

■C 言語 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);
```

■パラメータ

《cre_alm、icre_alm》

ID	almid	R4	アラームハンドラ ID
T_CALM	*pk_calm	R5	アラームハンドラ生成情報を格納したパケットへのポインタ

《acre_alm、iacre_alm》

T_CALM	*pk_calm	R4	アラームハンドラ生成情報を格納したパケットへのポインタ

■リターンパラメータ

《cre_alm、icre_alm》

ER	ercd	R0	正常終了（E_OK）またはエラーコード

《acre_alm、iacre_alm》

ER_ID	almid	R0	生成したアラームハンドラの ID 番号（正の値）またはエラーコード

■パケットの構造

```
typedef   struct  t_calm {
          ATR     almatr;      0      4    アラームハンドラ属性
          VP_INT  exinf;      +4      4    拡張情報
          FP      almhdr;     +8      4    アラームハンドラアドレス
} T_CALM;
```

■エラーコード

E_RSATR	[p]	属性不正(almatr が不正)
E_PAR	[p]	パラメータエラー(pk_calm が 4 の倍数以外、almhdr が奇数)
E_ID	[p]	不正 ID 番号(almid≦0、almid＞CFG_MAXALMID)
E_OBJ	[k]	オブジェクト状態不正(almid のアラームハンドラが存在)
E_NOID	[k]	空き ID なし

■機能

cre_alm, icre_alm サービスコールは、almid で示されたアラームハンドラを pk_calm で示された内容で生成します。

acre_alm, iacre_alm サービスコールは、未登録のアラームハンドラ ID を検索してその ID を持つアラームハンドラを pk_calm で示された内容で定義し、その ID をリターンパラメータとして返します。検索するアラームハンドラ ID の範囲は 1～CFG_MAXALMID です。

アラームハンドラは、指定した時刻に一度だけ起動される非タスクコンテキストのタイムイベントハンドラです。

almatr には属性として、ハンドラを記述した言語を指定します。

```
almatr := (TA_HLNG || TA_ASM)
```

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述

exinf は、アラームハンドラを起動するときに、パラメータとして渡す拡張情報を指定します。アラームハンドラに関する情報を設定するなどの目的でユーザが自由に使用できます。

almhdr には、アラームハンドラの先頭アドレスを指定します。

アラームハンドラの生成直後は、アラームハンドラの起動時刻は設定されていません。アラームハンドラは停止状態となっています。

なお、アラームハンドラはコンフィギュレータで静的に生成することもできます。

3.17.2　アラームハンドラの削除(del_alm)

■C 言語 API

```
ER ercd = del_alm(ID almid);
```

■パラメータ

ID	almid	R4	アラームハンドラ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(almid≦0、almid＞CFG_MAXALMID)
E_NOEXS	[k]	未登録(almid の可変長メモリプールが存在しない)
E_CTX	[k]	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能

almid で示されたアラームハンドラを削除します。

3.17.3 アラームハンドラの動作開始(sta_alm, ista_alm)

■C 言語 API

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

■パラメータ

ID	almid	R4	アラームハンドラ ID
RELTIM	almtim	R5	アラームハンドラの起動時刻

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(almid≦0、almid＞CFG_MAXALMID)
E_NOEXS	[k]	未登録(almid のアラームハンドラが存在しない)

■機能

almid で示されたアラームハンドラの起動時刻を、サービスコールが呼び出された時刻から almtim で指定された相対時間後に設定し、アラームハンドラの動作を開始します。

すでに動作しているアラームハンドラが指定された場合は、以前の起動時刻の設定を解除し、新しい起動時刻を設定します。

almtim に 0 が指定された場合は、次のタイムティックでアラームハンドラが起動されます。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、almtim に指定可能な最大値は H'ffffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.17.4 アラームハンドラの動作停止(stp_alm, istp_alm)

■C 言語 API

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

■パラメータ

ID	almid	R4	アラームハンドラ ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(almid≦0、almid＞CFG_MAXALMID)
E_NOEXS	[k]	未登録(almid のアラームハンドラが存在しない)

■機能

almid で示されたアラームハンドラの起動時刻の設定を解除し、アラームハンドラの動作を停止します。

3.17.5 アラームハンドラの状態参照(ref_alm, iref_alm)

■C 言語 API

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

■パラメータ

ID	almid	R4	アラームハンドラ ID
T_RALM	*pk_ralm	R5	アラームハンドラ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RALM	*pk_ralm	R5	アラームハンドラ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct  t_ralm {
          STAT    almstat;     +0     4    アラームハンドラの動作状態
          RELTIM  lefttim;     +4     4    アラームハンドラ起動までの残り時間
} T_RALM;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_ralm が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(almid≦0、almid＞CFG_MAXALMID)
E_NOEXS	[k]	未登録(almid のアラームハンドラが存在しない)

■機能

almid で示されたアラームハンドラの状態を参照し、pk_ralm が指す領域にアラームハンドラの動作状態(almstat)、アラームハンドラ起動までの残り時間(lefttim)を返します。

almstat には、対象アラームハンドラの動作状態を返します。

- TALM_STP（H'00000000）アラームハンドラが動作していない
- TALM_STA（H'00000001）アラームハンドラが動作している

lefttim には、対象アラームハンドラ起動までの相対時間を返します。対象アラームハンドラが動作していない場合、lefttim は不定値となります。

3.18 時間管理機能(オーバーランハンドラ)

表 3.35にオーバーランハンドラでサポートしているサービスコール一覧を示します。

表3.35 オーバーランハンドラサービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_ovr	オーバーランハンドラの定義	○		○	○	○		
2	sta_ovr	オーバーランハンドラの動作開始	○		○	○	○		
	ista_ovr			○	○	○	○		
3	stp_ovr	オーバーランハンドラの動作停止	○		○	○	○		
	istp_ovr			○	○	○	○		
4	ref_ovr	オーバーランハンドラの状態参照	○		○	○	○		
	iref_ovr			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

オーバーランハンドラは、システムに 1 つだけ定義できるタイムイベントハンドラです。

タスクが使用したプロセッサ時間には、タスクとタスクが呼び出したサービスコール、そのタスクの実行中に起動された割込みハンドラの各実行時間が含まれます。

タスクが実行状態以外の間は、使用プロセッサ時間はカウントされません。

表 3.37にオーバーランハンドラの仕様を示します。

表3.36 オーバーランハンドラの仕様

項番	項目	内容
1	プロセッサ時間の単位(OVRTIM)	システム時刻と同じ(1[ms])
2	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述

RENESAS

3.18.1 オーバーランハンドラの定義(def_ovr)

■C 言語 API

```
ER ercd = def_ovr(T_DOVR *pk_dovr);
```

■パラメータ

T_DOVR	*pk_dovr	R4	オーバーランハンドラ定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef    struct  t_dovr {
           ATR     ovratr;    0     4   オーバーランハンドラ属性
           FP      ovrhdr;    +4    4   オーバーランハンドラアドレス
} T_DOVR;
```

■エラーコード

E_RSATR	[p]	属性不正(ovratr が不正)
E_PAR	[p]	パラメータエラー(pk_dovr が 4 の倍数以外、ovrhdr が奇数)

■機能

pk_dovr で示された内容でオーバーランハンドラを定義します。

オーバーランハンドラは、タスクが設定された時間を超えてプロセッサを使用した場合に起動される非タスクコンテキストのタイムイベントハンドラです。

ovratr には属性として、ハンドラを記述した言語を指定します。

ovratr := (TA_HLNG || TA_ASM)

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述

ovrhdr には、オーバーランハンドラの先頭アドレスを指定します。

def_ovr サービスコールでは、pk_dovr=NULL(0)が指定された場合は、オーバーランハンドラの定義を解除します。

また、すでにオーバーランハンドラが定義されている状態で、本サービスコールが呼び出された場合は、以前の定義を解除し、新しい定義に置き換えます。

なお、オーバーランハンドラはコンフィギュレータで静的に生成することもできます。

3.18.2 オーバーランハンドラの動作開始(sta_ovr, ista_ovr)

■C 言語 API

```
ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim);
ER ercd = ista_ovr(ID tskid, OVRTIM ovrtim);
```

■パラメータ

ID	tskid	R4	タスク ID
OVRTIM	ovrtim	R5	上限プロセッサ時間

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、 非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(オーバーランハンドラが定義されていない)

■機能

tskid で示されたタスクに対して、オーバーランハンドラの動作を開始します。

tskid=TSK_SELF（0）の指定により自タスクの指定になります。

対象タスクの上限プロセッサ時間を ovrtim で指定される時間に設定し、使用プロセッサ時間を 0 クリアします。すでに動作しているオーバーランハンドラが指定された場合は、以前の上限プロセッサ時間の設定を解除し、新しい上限プロセッサ時間を設定します。

使用プロセッサ時間が上限プロセッサ時間を超えたときに、オーバーランハンドラが起動されます。

CFG_TICDENO(タイムティック周期時間の分母)に 1 より大きな値を設定した場合は、ovrtim に指定可能な最大値は H'ffffffff/CFG_TICDENO に制限されます。これより大きな値を指定した場合の動作は保証されません。

ovrtim に 0 が指定された場合は、対象タスクがプロセッサを使用してから、1 回目のタイムティックでオーバーランハンドラが起動されます。

時間の管理方法については、「2.16.4(2)時間の管理方法」を参照してください。

3.18.3　オーバーランハンドラの動作停止(stp_ovr, istp_ovr)

■C 言語 API

```
ER ercd = stp_ovr(ID tskid);
ER ercd = istp_ovr(ID tskid);
```

■パラメータ

ID	tskid	R4	タスク ID

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(オーバーランハンドラが定義されていない)

■機能

tskid で示されたタスクに対して、上限プロセッサ時間の設定を解除し、オーバーランハンドラの動作を停止します。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

3.18.4 オーバーランハンドラの状態参照(ref_ovr, iref_ovr)

■C 言語 API

```
ER ercd = ref_ovr(ID tskid, T_ROVR *pk_rovr);
ER ercd = iref_ovr(ID tskid, T_ROVR *pk_rovr);
```

■パラメータ

ID	tskid	R4	タスク ID
T_ROVR	*pk_rovr	R5	オーバーランハンドラの状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_ROVR	*pk_rovr	R5	オーバーランハンドラの状態を格納したパケットのへのポインタ

■パケットの構造

```
typedef   struct t_rovr {
          STAT    ovrstat;    +0    4   オーバーランハンドラの動作状態
          OVRTIM  leftotm;    +4    4   残りのプロセッサ時間
} T_ROVR;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rovr が 4 の倍数以外)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID、非タスクコンテキストで tskid=TSK_SELF(0)を指定)
E_NOEXS	[k]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態不正(オーバーランハンドラが定義されていない)

■機能

tskid で示されたタスクのオーバーランハンドラに関する状態を参照し、pk_rovr が指す領域にオーバーランハンドラの動作状態(ovrstat)、残りのプロセッサ時間(leftotm)を返します。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

ovrstat には、対象オーバーランハンドラの動作状態として、上限プロセッサ時間の設定状態を返します。

- TOVR_STP（H'00000000）上限プロセッサ時間が設定されていない
- TOVR_STA（H'00000001）上限プロセッサ時間が設定されている

leftotm には、対象タスクを原因としてオーバーランハンドラが起動されるまでの残りプロセッサ時間を返します。対象タスクに上限プロセッサ時間が設定されていない場合、leftotm は不定値となります。

3.19 システム状態管理機能

表 3.37にシステム状態管理でサポートしているサービスコール一覧を示します。

表3.37 システム状態管理サービスコール

項番	サービスコール *1		機 能	呼び出し可能なシステム状態 *2						
				T	N	E	D	U	L	C
1	rot_rdq	[S]	タスクの優先順位の回転	○		○	○	○		
	irot_rdq	[S]			○	○	○	○		
2	get_tid	[S]	実行状態のタスク ID の参照	○		○	○	○		○
	iget_tid	[S]			○	○	○	○		○
3	loc_cpu	[S]	CPU ロック状態への移行	○		○	○	○	○	
	iloc_cpu	[S]			○	○	○	○	○	
4	unl_cpu	[S]	CPU ロック状態の解除	○		○	○	○	○	
	iunl_cpu	[S]			○	○	○	○	○	
5	dis_dsp	[S]	ディスパッチの禁止	○		○	○	○		
6	ena_dsp	[S]	ディスパッチの許可	○		○	○	○		
7	sns_ctx	[S]	コンテキストの参照	○	○	○	○	○	○	○
8	sns_loc	[S]	CPU ロック状態の参照	○	○	○	○	○	○	○
9	sns_dsp	[S]	ディスパッチ禁止状態の参照	○	○	○	○	○	○	○
10	sns_dpn	[S]	ディスパッチ保留状態の参照	○	○	○	○	○	○	○
11	vsta_knl	[s]	カーネルの起動	○	○	○	○	○	○	○
	ivsta_knl	[s]		○	○	○	○	○	○	○
12	vsys_dwn	[s]	システムダウン	○	○	○	○	○	○	○
	ivsys_dwn	[s]		○	○	○	○	○	○	○
13	vget_trc		トレースの取得	○		○	○	○		
	ivget_trc				○	○	○	○		
14	ivbgn_int		割込みハンドラの開始をトレースに取得		○	○	○	○		
15	ivend_int		割込みハンドラの終了をトレースに取得		○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

3.19.1 タスクの優先順位の回転(rot_rdq, irot_rdq)

■C 言語 API

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

■パラメータ

PRI	tskpri	R4	タスク優先度

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(tskpri＜0、tskpri＞CFG_MAXTSKPRI、非タスクコンテキストで tskpri=TPRI_SELF(0)を指定)

■機能

tskpri で示された優先度のレディキューにつながれている先頭タスクをレディキューの最後尾につなぎかえ（レディキューを回転）、次につながれているタスクに実行を切り替えます。

tskpri=TPRI_SELF（0）を指定すると、自タスクのベース優先度のレディキューを回転します。ミューテックス機能を使用しない場合は、ベース優先度と現在優先度は同じですが、ミューテックスをロック中は一般には現在優先度とベース優先度は一致しないため、TPRI_SELF を指定しても現在の自タスクが属する優先度のレディキューを回転することはできません。

3.19.2 実行状態のタスク ID の参照(get_tid, iget_tid)

■C 言語 API

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

■パラメータ

ID	*p_tskid	R4	タスク ID を返す領域へのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
ID	*p_tskid	R4	タスク ID へのポインタ

■エラーコード

E_PAR [p] パラメータエラー(p_tskid が奇数)

■機能

実行状態のタスクの ID を求め、その結果を p_tskid の指す領域に返します。
具体的には、タスクコンテキストから呼び出された場合は自タスクの ID を返し、非タスクコンテキストから呼び出された場合はその時実行していたタスクの ID を返します。実行状態のタスクがない場合は、TSK_NONE（0）を返します。
本サービスコールは、CPU 例外ハンドラからも呼び出せます。

3.19.3 CPU ロック状態への移行(loc_cpu, iloc_cpu)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

システム状態を CPU ロック状態とし、割込みとタスクのディスパッチを禁止します。
CPU ロック状態の特長を以下に示します。

(1) CPUロック状態の間は、タスクのスケジューリングは行われません。
(2) コンフィギュレータで定義したカーネル割込みマスクレベル(CFG_KNLMSKLVL)以下の割り込みが禁止されます。
(3) CPUロック状態から呼び出し可能なサービスコールは、以下のサービスコールのみです。その他のサービスコールが呼び出された場合の動作は保証されません。

- ext_tsk
- exd_tsk
- 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

CPU ロック状態は、以下の操作で解除されます。

(a) unl_cpu, iunl_cpuサービスコールの呼び出し
(b) ext_tsk, exd_tskサービスコールの呼び出し

CPU ロック状態と CPU ロック解除状態の間の遷移は、loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, ext_tsk, exd_tsk サービスコールによってのみ発生します。カーネル割込みマスクレベル(CFG_KNLMSKLVL)以下の割込みハンドラ、タイムイベントハンドラ、初期化ルーチン、タスク例外処理ルーチンの終了時には、必ず CPU ロック解除状態でなければなりません。CPU ロック状態の場合、動作は保証されません。なお、これらのハンドラ開始時は、常に CPU ロック解除状態です。

CPU 例外ハンドラで CPU ロック/ロック解除状態を変更した場合、必ず終了前に元に状態に戻さなくてはなりません。戻さない場合、動作は保証されません。

すでに CPU ロック状態のときに、再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

3.19.4 CPU ロック状態の解除(unl_cpu, iunl_cpu)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了 (E_OK)

■エラーコード

なし

■機能

loc_cpu, iloc_cpu サービスコールによって設定されていた CPU ロック状態を解除します。ディスパッチ許可状態から unl_cpu サービスコールを発行した場合、タスクのスケジューリングが行われます。

割込みハンドラ内で iloc_cpu を呼び出し、CPU ロック状態に移行した場合は、割込みハンドラからリターンする前に必ず iunl_cpu を呼び出し、CPU ロック状態を解除してください。

CPU ロック状態とディスパッチ禁止状態は、独立して管理されます。そのため、unl_cpu, iunl_cpu サービスコールでは、ena_dsp サービスコールによるディスパッチ禁止状態は解除されません。

CPU ロック解除状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

3.19.5 ディスパッチの禁止(dis_dsp)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

システム状態をディスパッチ禁止状態にします。ディスパッチ禁止状態の特長を、以下に示します。

(1) タスクのスケジューリングが行われなくなるため、自タスク以外のタスクが実行状態に移行することはなくなります。
(2) 割込みは受け付けられます。
(3) 待ち状態になるサービスコールを呼び出せません。

ディスパッチ禁止状態の間に以下の操作を行うと、システム状態はタスク実行状態に戻ります。

(1) ena_dspサービスコールの呼び出し
(2) ext_tsk, exd_tskサービスコールの呼び出し

ディスパッチ禁止状態とディスパッチ許可状態の間の遷移は、dis_dsp, ena_dsp, ext_tsk, exd_tsk サービスコールによってのみ発生します。

CPU 例外ハンドラでディスパッチ禁止/許可状態を変更した場合、必ず終了前に元に状態に戻さなくてはなりません。戻さない場合、動作は保証されません。

本サービスコールによってディスパッチ禁止状態の間は、ref_tsk サービスコールで自タスクの状態を参照しても実行状態とは見えない場合があるので、注意してください。

すでにディスパッチ禁止状態のときに再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

RENESAS

3.19.6 ディスパッチの許可(ena_dsp)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了 (E_OK)

■エラーコード

なし

■機能

dis_dsp サービスコールによって設定されていたディスパッチ禁止状態を解除します。それにより、システムがタスク実行状態になった場合は、タスクのスケジューリングが行われます。

タスク実行状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

3.19.7 コンテキストの参照(sns_ctx)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

BOOL	state	R0	コンテキスト

■機能

非タスクコンテキストから呼び出された場合に TRUE、タスクコンテキストから呼び出された場合に FALSE を返します。

本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

3.19.8 CPU ロック状態の参照(sns_loc)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

BOOL	state	R0	CPU ロック状態

■機能

システムが CPU ロック状態の場合に TRUE、CPU ロック解除状態の場合に FALSE を返します。
本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

3.19.9 ディスパッチ禁止状態の参照(sns_dsp)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

BOOL	state	R0	ディスパッチ禁止状態

■機能

システムがディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返します。

本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

RENESAS

3.19.10　ディスパッチ保留状態の参照(sns_dpn)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

BOOL	state	R0	ディスパッチ保留状態

■機能

システムがディスパッチ保留状態の場合に TRUE、そうでない場合に FALSE を返します。

具体的には、以下の全ての条件が満足される場合に FALSE が返し、その他の場合には TRUE を返します。

(1)　ディスパッチ禁止状態でない
(2)　CPUロック状態でない
(3)　タスクまたはタスク例外処理ルーチンである
(4)　chg_imsサービスコールによって割込みをマスクしている状態ではない

本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

3.19.11 カーネルの起動(vsta_knl, ivsta_knl)

■C 言語 API

```
void vsta_knl(void);
void ivsta_knl(void);
```

■アセンブリ言語 API

シンボル"__kernel_reset"に分岐

■パラメータ

なし

■リターンパラメータ

本サービスコールの呼び出し元には戻りません。

■機能

カーネルを起動します。
すでにカーネルを起動済みの場合は、それまでのマルチタスク環境は全て破棄されます。
本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。また、カーネル起動前であっても呼び出せます。
本サービスコールは、すべての割込みをマスクした状態(SR.IMASK=15)で発行してください。
HI7700/4, HI7750/4 においては、本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。
本サービスコールを呼び出すアプリケーションは、カーネルとリンクする必要があります。
「2.18.1 リセットとカーネルの起動」も参照してください。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.19.12　システムダウン(vsys_dwn, ivsys_dwn)

■C 言語 API

```
void vsys_dwn(W type, ER ercd, VW inf1, VW inf2);
void ivsys_dwn(W type, ER ercd, VW inf1, VW inf2);
```

■パラメータ

W	type	R4	エラー種別
ER	ercd	R5	エラーコード
VW	inf1	R6	システム異常情報 1
VW	inf2	R7	システム異常情報 2

■リターンパラメータ

本サービスコール呼び出し元には戻りません。

■機能

システムダウンルーチンに制御を渡します。

type には、エラー種別として発生したエラーに対応した値(1～H'7fffffff)を設定してください。0 以下の値はシステム用に予約されています。

カーネル内で異常を検出した場合にも、システムダウンルーチンが呼び出されます。

本サービスコールは、CPU ロック状態および CPU 例外ハンドラからも呼び出せます。

HI7700/4, HI7750/4 においては、本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.19.13 トレースの取得(vget_trc, ivget_trc)

■C 言語 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);
```

■パラメータ

VW	para1	R4	パラメータ 1
VW	para2	R5	パラメータ 2
VW	para3	R6	パラメータ 3
VW	para4	R7	パラメータ 4

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

ユーザ任意の情報をトレース取得します。
para1～para4 は、取得される情報を識別するために、ユーザが自由に使用できます。
取得したトレース情報は、デバッギングエクステンション(DX)でトレース表示されます。
コンフィギュレータで CFG_TRACE をチェックしていない場合は、本サービスコールは何も処理せずに終了します。
なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.19.14 割込みハンドラの開始をトレースに取得(ivbgn_int)

■C 言語 API

```
ER ercd = ivbgn_int(UINT dintno);
```

■パラメータ

UINT	dintno	R4	割込みハンドラ番号

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

dintno で示された割込みハンドラ番号に対する割込みハンドラの処理開始を、トレース取得します。

割込みハンドラ番号は、HI7000/4 では CPU のベクタ番号、HI7700/4 および HI7750/4 では CPU の例外コード(INTEVT2 が存在する CPU では INTEVT2 コード、そうでない CPU では INTEVT コード)です。

本サービスコールは、割込みハンドラの先頭で呼び出すようにしてください。また、必ず ivend_int サービスコールとセットで使用してください。

割込みハンドラ以外から呼び出ししてもエラーにはなりませんが、この場合デバッギングエクステンションによるトレース表示が不正になる可能性があります。

コンフィギュレータで CFG_TRACE をチェックしていない場合は、本サービスコールは何も処理せずに終了します。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.19.15 割込みハンドラの終了をトレースに取得(ivend_int)

■C 言語 API

```
ER ercd = ivend_int(UINT dintno);
```

■パラメータ

UINT	dintno	R4	割込みハンドラ番号

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

dintno で示された割込みハンドラ番号に対する割込みハンドラの処理終了をトレース情報に取得します。

割込みハンドラ番号は、HI7000/4 では CPU のベクタ番号、HI7700/4 および HI7750/4 では CPU の例外コード(INTEVT2 が存在する CPU では INTEVT2 コード、そうでない CPU では INTEVT コード)です。

本サービスコールは、割込みハンドラの最後で呼び出すようにしてください。また、必ず ivbgn_int サービスコールとセットで使用してください。

割込みハンドラ以外から呼び出ししてもエラーにはなりませんが、この場合デバッギングエクステンションによるトレース表示が不正になる可能性があります。

コンフィギュレータで CFG_TRACE をチェックしていない場合は、本サービスコールは何も処理せずに終了します。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.20 割込み管理機能

表 3.38に割込み管理でサポートしているサービスコール一覧を示します。

表3.38 割込み管理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_inh	割込みハンドラの定義	○		○	○	○		
	idef_inh			○	○	○	○		
2	chg_ims	割込みマスクの変更	○		○	○	○		
	ichg_ims			○	○	○	○		
3	get_ims	割込みマスクの参照	○		○	○	○		
	iget_ims			○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.39に割込み管理の仕様を示します。

表3.39 割込み管理の仕様

項番	項目	内容
1	割込みハンドラ番号	[HI7000/4] 0～CFG_MAXVCTNO (最大 511) [HI7700/4, HI7750/4] 0～CFG_MAXVCTNO (最大 H'fe0)
2	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述

3.20.1　割込みハンドラの定義(def_inh, idef_inh)

■C 言語 API

```
ER ercd = def_inh(INHNO inhno, T_DINH *pk_dinh);
ER ercd = idef_inh(INHNO inhno, T_DINH *pk_dinh);
```

■パラメータ

INHNO	inhno	R4	割込みハンドラ番号
T_DINH	*pk_dinh	R5	割込みハンドラ定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef   struct  t_dinh {
          ATR     inhatr;     0     4   ハンドラ属性
          FP      inthdr;     +4    4   ハンドラアドレス
          UINT    inhsr;      +8    4   起動時の SR(HI7000/4 では無視されます)
} T_DINH;
```

■エラーコード

E_RSATR　[p]　属性不正(inhatr が不正)

E_PAR　[p]　パラメータエラー(pk_dinh が 4 の倍数以外、inthdr が奇数)

〈HI7000/4〉

不正番号指定(inhno=25,26、inhno＞CFG_MAXVCTNO)

〈HI7700/4、HI7750/4〉

不正番号指定(inhno が H'20 の倍数以外、inhno=H'160、inhno＞CFG_MAXVCTNO)

■機能

inhno で示された割込みハンドラ番号に対するハンドラを、pk_dinh で示された内容で定義します。

割込みハンドラ番号は、HI7000/4 では CPU のベクタ番号、HI7700/4 および HI7750/4 では CPU の例外コード(INTEVT2 が存在する CPU では INTEVT2 コード、そうでない CPU では INTEVT コード)です。

HI7000/4 では、割込みハンドラ番号 0～3（パワーオンリセット、マニュアルリセット）には定義できません（指定しても無視されます）。

HI7700/4, HI7750/4 では、割込みハンドラ番号 0、H'20（パワーオン、マニュアルリセット）には定義できません（指定しても無視されます）。

inhatr には属性として、以下を指定します。

```
inhatr := (TA_HLNG‖TA_ASM) [ |VTA_DIRECT ] [|VTA_REGBANK]
```

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述
- VTA_DIRECT（H'80000000）ダイレクト割込みハンドラ（HI7000/4 のみ）
- VTA_REGBANK（H'40000000）レジスタバンクを使用する通常割込みハンドラ（HI7000/4 のみ）

HI7000/4 の VTA_DIRECT は、ダイレクト割込みハンドラを定義する場合に指定してください。ダイレクト割込みハンドラの割込み応答は、通常の割込みハンドラよりも高速です。カーネル割込みマスクレベル(CFG_KNLMSKLVL)より高い割込みレベルの割込みハンドラを定義する場合は、必ず VTA_DIRECT を指定しなければなりません。また、CPU 例外およびトラップの割込みハンドラ番号に VTA_DIRECT を指定してはなりません。

コンフィギュレータの CFG_DIRECT を選択している場合は、VTA_DIRECT 指定のないハンドラを定義することはできません。この場合、エラーE_RSATR が返ります。

HI7000/4 の VTA_REGBANK は、使用している CPU コアが SH-2A または SH2A-FPU で、かつコンフィギュレータの CFG_REGBANK に SELECT を指定した場合のみ有効で、レジスタバンクを使用するように設定された割込みに対する通常割込みハンドラを定義する場合に指定します。その他の場合は、意味を持ちません。なお、VTA_DIRECT が指定されている場合は、VTA_REGBANK 指定は意味を持たず、単に無視されます。

コンフィギュレータの CFG_BANKLVLxx(xx は割込みレベルで、01～15)を ON にした割込みレベルの通常割込みハンドラを定義する場合は必ず VTA_REGBANK を指定し、CFG_BANKLVLxx を OFF にした割込みレベルの通常割込みハンドラを定義する場合は必ず VTA_REGBANK を指定しないようしなければなりません。

また、VTA_REGBANK は、割込みハンドラ番号が 14 または 64 以上の場合のみ意味があります。なお、この範囲は「レジスタバンクを使用可能な割込みハンドラ番号」の範囲を意味しており、NMI および UBC はここには含まれません。この範囲以外の割込みハンドラ番号を指定した場合は、VTA_REGBANK 指定は意味を持たず、単に無視されます。

使用している CPU コアが SH-2A、SH2A-FPU 以外の場合、または CFG_REGBANK が SELECT 以外の場合は、通常割込みハンドラのレジスタバンクの使用について以下のように扱います。

(1) 使用しているCPUコアがSH-2AまたはSH2A-FPUで、かつCFG_REGBANKがALLの場合
割込みハンドラ番号14および64以上は、全てレジスタバンクを使用すると扱い、それ以外(具体的にはNMIとUBC)はすべてレジスタバンクを使用しないと扱います。

(2) 使用しているCPUコアがSH-2A、SH2A-FPU以外の場合、またはCFG_REGBANKがNOTUSEまたはNOBANKの場合
全割込みでレジスタバンクを使用しないと扱います。

HI7700/4、HI7750/4 では、inhsr には割込みハンドラ起動時のステータスレジスタ（SR）の値を指定します。inhsr は、SR の構成と同じビット位置で指定します。なお、割込みマスクビットには、必ず当該割込みレベル以上の値を指定してください。当該割込みレベルより低い値を指定すると、システムの動作は異常になります。inhsr の設定については、「4.2.1 SR レジスタ」および「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」も参照してください。

実際の割込みハンドラ起動時の SR は、以下のようになります。

- 割込みマスクビット
  inhsrで指定した通りになります。
- 割込みマスク以外のビット
  inhsrで指定した通りになります。

HI7000/4 では、inhsr の指定は意味を持たず，単に無視されます。実際の起動時の SR は、CPU の割込み処理によって決まります。

pk_dinh＝NULL（0）と指定した場合には、inhno の定義を解除します。

なお、割込みハンドラはコンフィギュレータで静的に定義することもできます。

inhsr は、μITRON4.0 仕様の範囲外のメンバです。

3.20.2 割込みマスクの変更(chg_ims, ichg_ims)

■C 言語 API

```
ER ercd = chg_ims(IMASK imask);
ER ercd = ichg_ims(IMASK imask);
```

■パラメータ

IMASK	imask	R4	割込みマスク値

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(imask に SR_IMS00～SR_IMS15 以外の値を指定)

■機能

現在の割込みマスクを imask で指定した値に変更します。

imask には、以下の指定ができます。

- SR_IMSnn（H'0000000m）割込みマスクレベルを nn に変更
  - nn : 0～15 を 10 進数 2 桁で表現した文字列（“00”, “01”, “02”, ... , “15”）
  - m : nn を 16 進数に変換した文字

以下のケースで割込みマスクレベルを変更する場合は、必ず本サービスコールを使用してください。それ以外のケース以外で割込みマスクレベルを変更する場合は，SR レジスタを直接変更しても構いません。

(1)　タスクコンテキストで割込みマスクレベルを0から0以外に変更する場合

(2)　(1)の後、割込みマスクレベルを0に戻す場合

これに違反した場合の動作は保証されません。

割込みマスクレベルをカーネル割込みマスクレベル(CFG_KNLMSKLVL)より高いレベルに変更している間は、本サービスコールでカーネル割込みマスクレベル以下に割込みマスクレベルを下げる場合を除き、サービスコールを発行してはなりません。これに違反した場合の動作は保証されません。

また、タスクコンテキストから割込みをマスクした場合は、割込みマスクレベルを 0 に戻すまでは非タスクコンテキストとして扱われます。

「4.2.1 SR レジスタ」も参照してください。

3.20.3 割込みマスクの参照(get_ims, iget_ims)

■C 言語 API

```
ER ercd = get_ims(IMASK *p_imask);
ER ercd = iget_ims(IMASK *p_imask);
```

■パラメータ

IMASK	*p_imask	R4	割込みマスクレベルを返す領域の先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
UINT	*p_imask	R4	割込みマスクレベルを格納した領域の先頭アドレス

■エラーコード

E_PAR [p] パラメータエラー(p_imask が 4 の倍数以外)

■機能

現在の CPU ステータスレジスタ(SR)の割込みマスクビット(IMASK ビット)を参照し、割込みマスクレベルを p_imask の指す領域に返します。p_imask の指す領域に返る値は、chg_ims サービスコールで用いるパラメータ imask と同じフォーマットです。

3.21　サービスコール管理機能

表 3.40にサービスコール管理でサポートしているサービスコール一覧を示します。

表3.40　サービスコール管理サービスコール

項番	サービスコール　*1	機　能	呼び出し可能なシステム状態　*2						
			T	N	E	D	U	L	C
1	def_svc	拡張サービスコールの定義	○		○	○	○		
	idef_svc			○	○	○	○		
2	cal_svc	サービスコールの呼び出し	○		○	○	○		
	ical_svc			○	○	○	○		

【注】 *1　"[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2　それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.41にサービスコール管理の仕様を示します。

表3.41　サービスコール管理の仕様

項番	項目	内容
1	拡張サービスコールの機能コード	1～CFG_MAXSVCCD (最大 1023)
2	渡せるパラメータ	VP_INT 型で、0～4 個
3	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述

3.21.1 拡張サービスコールの定義(def_svc, idef_svc)

■C 言語 API

```
ER ercd = def_svc(FN fncd, T_DSVC *pk_dsvc);
ER ercd = idef_svc(FN fncd, T_DSVC *pk_dsvc);
```

■パラメータ

FN	fncd	R4	拡張サービスコールの機能コード
T_DSVC	*pk_dsvc	R5	拡張サービスコールルーチン定義情報の先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef    struct  t_dsvc {
           ATR     svcatr;     0      4    拡張サービスコールルーチン属性
           FP      svcrtn;     +4     4    拡張サービスコールルーチンドレス
} T_DSVC;
```

■エラーコード

E_RSATR	[p]	属性不正(svcatr が不正)
E_PAR	[p]	パラメータエラー(pk_dsvc が 4 の倍数以外、svcrtn が奇数、fncd≦0、fncd＞CFG_MAXSVCCD)

■機能

def_svc, idef_svc は、fncd で示された拡張機能コードに対する拡張サービスコールルーチンを、pk_dsvc で示された内容で定義します。

svcatr には属性として、ルーチンを記述した言語を指定します。

svcatr := ((TA_HLNG || TA_ASM))

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述

svcrtn には、拡張サービスコールルーチンの先頭アドレスを指定します。

pk_dsvc＝NULL（0）と指定した場合には、fncd の拡張サービスコールルーチンの定義を解除します。

拡張サービスコールルーチンは、呼び出し元の状態を引き継ぎます。

3.21.2 サービスコールの呼び出し(cal_svc, ical_svc)

■C 言語 API

```
ER_UINT ercd = cal_svc(FN fncd, ...);
ER_UINT ercd = ical_svc(FN fncd, ...);
```

■パラメータ

FN	fncd	@R15	拡張サービスコールの機能コード

"..."で示した部分には、以下のように VP_INT 型の引数を 0 から 4 個記述できます。5 つ以上指定しても、拡張サービスコールルーチンには 4 つ目の引数までしか渡りません。

VP_INT	par1	@(4,R15)	パラメータ 1
VP_INT	par2	@(8,R15)	パラメータ 2
VP_INT	par3	@(12,R15)	パラメータ 3
VP_INT	par4	@(16,R15)	パラメータ 4

■リターンパラメータ

ER_UINT	ercd	R0	サービスコールからのリターン値

■エラーコード

E_RSFN [p] 予約機能コード(fncd が不正あるいは使用できない)

■機能

fncd で指定された機能コードに対応する拡張サービスコールルーチンを実行します。

パラメータは、VP_INT 型で 0〜4 個指定できます。呼び出される拡張サービスコールルーチンでは、par 1 〜par4 がそれぞれ R4〜R7 に格納されて渡されます。

詳細は、「4.7 拡張サービスコールルーチン」を参照してください。

3.22　システム構成管理機能

表 3.42にシステム構成管理でサポートしているサービスコール一覧を示します。

表3.42　システム構成管理サービスコール

項番	サービスコール　*1	機　能	呼び出し可能なシステム状態　*2						
			T	N	E	D	U	L	C
1	def_exc	CPU 例外ハンドラの定義	○		○	○	○		
	idef_exc			○	○	○	○		
2	vdef_trp	CPU 例外ハンドラの定義 (TRAPA 命令例外)	○		○	○	○		
	ivdef_trp			○	○	○	○		
3	ref_cfg	コンフィギュレーション情報の参照	○		○	○	○		
	iref_cfg			○	○	○	○		
4	ref_ver	バージョン情報の参照	○		○	○	○		
	iref_ver			○	○	○	○		

【注】 *1　"[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2　それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

表 3.43にシステム構成管理の仕様を示します。

表3.43　システム構成管理の仕様

項番	項目	内容
1	CPU 例外ハンドラ番号	[HI7000/4] 0〜CFG_MAXVCTNO (最大 511) [HI7700/4, HI7750/4] 0〜CFG_MAXVCTNO (最大 H'fe0)
2	トラップ番号	0〜CFG_MAXTRPNO (最大 255)
3	サポート属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述
4	制限事項(HI7700/4, HI7750/4)	HI7700/4, HI7750/4 では、例外コードが H'20 の倍数以外の例外を扱うことはできません。これらの例外が発生しないようにしてください。

3.22.1 CPU 例外ハンドラの定義(def_exc, idef_exc)

■C 言語 API

```
ER ercd = def_exc(EXCNO excno, T_DEXC *pk_dexc);
ER ercd = idef_exc(EXCNO excno, T_DEXC *pk_dexc);
```

■パラメータ

EXCNO	excno	R4	CPU 例外ハンドラ番号
T_DEXC	*pk_dexc	R5	CPU 例外ハンドラ定義情報の先頭アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef   struct  t_dexc {
          ATR     excatr;     0      4    ハンドラ属性
          FP      exchdr;     +4     4    ハンドラアドレス
          UINT    excsr;      +8     4    起動時の SR(HI7000/4 では無視されます)
} T_DEXC;
```

■エラーコード

E_RSATR　[p]　属性不正(excatr が不正)

E_PAR　[p]　パラメータエラー(pk_dexc が 4 の倍数以外、exchdr が奇数)

〈HI7000/4〉

不正番号指定(excno=25,26、excno＞CFG_MAXVCTNO)

〈HI7700/4、HI7750/4〉

不正番号指定(excno が H'20 の倍数以外、excno=H'160、excno＞CFG_MAXVCTNO)

■機能

excno で示された CPU 例外ハンドラ番号に対するハンドラを、pk_dexc で示された内容で定義します。

CPU 例外ハンドラ番号は、HI7000/4 では CPU のベクタ番号、HI7700/4 および HI7750/4 では CPU の例外コード(EXPEVT コード)です。

HI7700/4, HI7750/4 では、EXPEVT コードが H'20 の倍数以外の例外コードに対する CPU 例外ハンドラは定義できません。SH3-DSP のリピートループによる TLB 例外(例外コード H'070 および H'0D0)など、H'20 の倍数以外のコードの例外が発生した場合の動作は保証されません。

excatr には属性として、ハンドラを記述した言語を指定します。

excatr := (TA_HLNG || TA_ASM)

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述

HI7700/4、HI7750/4 の場合、excsr には、CPU 例外ハンドラ起動時のステータスレジスタ（SR）の値を指定します。excsr は、SR の構成と同じビット位置で指定します。excsr の設定については、「4.2.1 SR レジスタ」および「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」も参照してください。

実際の CPU 例外ハンドラ起動時の SR は、以下のようになります。

- 割込みマスクビット
  例外発生前と同じです。
- CPU 例外マスク以外のビット
  excsrで指定した通りになります。

HI7000/4 では、excsr の指定は意味を持たず，単に無視されます。実際の起動時の SR は、CPU による例外処理の通りとなります。

pk_dexc＝NULL（0）と指定した場合には、excno の定義を解除します。

CPU 例外ハンドラから呼び出し可能なサービスコールは、以下のサービスコールのみです。その他のサービスコールが呼び出された場合の動作は保証されません。

- sns_tex
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- get_tid, iget_tid
- ras_tex, iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

CPU 例外ハンドラはコンフィギュレータで静的に定義することもできます。

なお、TRAPA 命令に対する CPU 例外ハンドラを定義するには、本サービスコールではなく vdef_trp, ivdef_trp サービスコールを使用してください。

excsr は、μITRON4.0 仕様の範囲外のメンバです。

RENESAS

3.22.2 CPU 例外(TRAPA 命令例外)ハンドラ定義(vdef_trp, ivdef_trp)

■C 言語 API

```
ER ercd = vdef_trp(UINT dtrpno, T_DTRP *pk_dtrp);
ER ercd = ivdef_trp(UINT dtrpno, T_DTRP *pk_dtrp);
```

■パラメータ

UINT	dtrpno	R4	トラップ番号
T_DTRP	*pk_dtrp	R5	CPU 例外(TRAPA 命令例外)ハンドラ定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■パケットの構造

```
typedef   struct  t_dtrp {
          ATR     trpatr;     0     4   CPU 例外(TRAPA 命令例外)ハンドラ属性
          FP      trphdr;     +4    4   CPU 例外(TRAPA 命令例外)ハンドラアドレス
          UINT    trpsr;      +8    4   CPU 例外(TRAPA 命令例外)ハンドラ起動時 SR
} T_DTRP;
```

■エラーコード

E_RSATR　[p]　属性不正(trpatr が不正)

E_PAR　[p]　パラメータエラー(pk_dtrp が 4 の倍数以外、trphdr が奇数)、

〈HI7000/4〉

不正番号指定(trpno=25,26、trpno＞CFG_MAXVCTNO)

〈HI7700/4、HI7750/4〉

不正番号指定(trpno＞CFG_MAXTRPNO)

■機能

dtrpno で示されたトラップ番号に対する CPU 例外(TRAPA 命令例外)ハンドラを pk_dtrp で示された内容で定義します。

trpatr には属性として、CPU 例外(TRAPA 命令例外)ハンドラを記述した言語を指定します。

trpatr := (TA_HLNG || TA_ASM)

- TA_HLNG（H'00000000）高級言語記述
- TA_ASM（H'00000001）アセンブラ記述

trphdr には、CPU 例外(TRAPA 命令例外)ハンドラの先頭アドレスを指定します。

HI7700/4、HI7750/4 の場合、trpsr には、CPU 例外ハンドラ起動時のステータスレジスタ（SR）の値を指定します。trpsr は、SR の構成と同じビット位置で指定します。trpsr の設定については、「4.2.1 SR レジスタ」および「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」も参照してください。

実際の CPU 例外ハンドラ起動時の SR は、以下のようになります。

- 割込みマスクビット
  例外発生前と同じです。
- CPU 例外マスク以外のビット
  excsrで指定した通りになります。

HI7000/4 では、trpsr の指定は意味を持たず，単に無視されます。実際の起動時の SR は、CPU による例外処理の通りとなります。

pk_dtrp＝NULL（0）と指定した場合には、dtrpno の定義を解除します。

CPU 例外(TRAPA 命令例外)ハンドラから呼び出し可能なサービスコールは、以下のサービスコールのみです。その他のサービスコールが呼び出された場合の動作は保証されません。

- sns_tex
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- get_tid, iget_tid
- ras_tex, iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

なお、CPU 例外(TRAPA 命令例外)ハンドラはコンフィギュレータで静的に定義することもできます。

なお、本サービスコールは HI7000/4 シリーズの独自機能です。

3.22.3 コンフィグレーション情報の参照(ref_cfg, iref_cfg)

■C 言語 API

```
ER ercd = ref_cfg(T_RCFG *pk_rcfg);
```

■パラメータ

T_RCFG	*pk_rcfg	R4	コンフィグレーション情報を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RCFG	*pk_rcfg	R4	コンフィグレーション情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct  t_rcfg {
           ID      maxtskid;     +0     2   最大タスク ID
           ID      ststkid;      +2     2   最大スタティックスタック使用タスク ID
           ID      maxsemid;     +4     2   最大セマフォ ID
           ID      maxflgid;     +6     2   最大イベントフラグ ID
           ID      maxdtqid;     +8     2   最大データキューID
           ID      maxmbxid;     +10    2   最大メールボックス ID
           ID      maxmtxid;     +12    2   最大ミューテックス ID
           ID      maxmbfid;     +14    2   最大メッセージバッファ ID
           ID      maxmplid;     +16    2   最大可変長メモリプール ID
           ID      maxmpfid;     +18    2   最大固定長メモリプール ID
           ID      maxcycid;     +20    2   最大周期ハンドラ ID
           ID      maxalmid;     +22    2   最大アラームハンドラ ID
           FN      maxs_fncd;    +24    4   最大拡張サービスコール機能コード
}  T_RCFG;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rcfg が 4 の倍数以外)

■機能

pk_rcfg で示された領域に、システムコンフィグレーション情報を返します。
pk_rcfg の指すパケットには、次の情報を返します。括弧内は、対応するコンフィギュレータでの設定項目です。

- maxtskid最大タスク ID（CFG_MAXTSKID）
- ststkidスタティックスタックを使用する最大タスク ID（CFG_STSTKID）
- maxsemid最大セマフォ ID（CFG_MAXSEMID）
- maxflgid最大イベントフラグ ID（CFG_MAXFLGID）
- maxdtqid最大データキューID（CFG_MAXDTQID）
- maxmbxid最大メールボックス ID（CFG_MAXMBXID）
- maxmtxid最大ミューテックス ID（CFG_MAXDTQID）
- maxmbfid最大メッセージバッファ ID（CFG_MAXMBFID）
- maxmplid最大可変長メモリプール ID（CFG_MAXMPLID）
- maxmpfid最大固定長メモリプール ID（CFG_MAXMPFID）
- maxcycid最大周期ハンドラ ID（CFG_MAXCYCID）
- maxalmid最大アラームハンドラ ID（CFG_MAXALMID）
- maxs_fncd最大拡張サービスコール機能コード（CFG_MAXSVCCD）

なお、CFG_ACTION をチェックしている場合は、maxcycid には CFG_MAXCYCID + 1 の値が返ります。

T_RCFG 構造体のメンバはすべて、μITRON4.0 仕様の範囲外です。なお、μITRON4.0 仕様では、T_RCFG 構造体の内容は規定されていません。

RENESAS

3.22.4 バージョン情報の参照(ref_ver, iref_ver)

■C 言語 API

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

■パラメータ

T_RVER	*pk_rver	R4	バージョン情報を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード
T_RVER	*pk_rver	R4	バージョン情報を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct  t_rver {
          UH      maker;      0     2   メーカー
          UH      prid;       +2    2   形式番号
          UH      spver;      +4    2   仕様書バージョン
          UH      prver;      +6    2   製品バージョン
          UH      prno[4];    +8    8   製品管理情報
} T_RVER;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rver が奇数)

■機能

現在実行中のカーネルのバージョンに関する情報を読み出し、その結果を pk_rver の指す領域に返します。

pk_rver の指すパケットには、次の情報を返します。

(1) maker

maker は、このカーネルを作ったメーカを表します。本カーネルでは、ルネサスを意味する H'0115 です。

(2) prid

prid は、OS や VLSI の種類を区別する番号を表します。以下に各カーネルの id を示します。

- HI7000/4：H'0010
- HI7700/4：H'000F
- HI7750/4：H'000E

(3) spver

spver は、カーネルの準拠する仕様を表しており、ビット対応に意味を持っています。

- ビット 15～12：MAGIC（TRON 仕様のシリーズを区別する番号）
  本カーネルでは、H'5（μITRON仕様）です。
- ビット 11～0：SpecVer（製品の元となった TRON 仕様書のバージョン番号）
  本カーネルでは、H'400（μITRON仕様 Ver.4.00.00）です。

(4) prver

prver は、カーネルのバージョン番号を表します。

prver の値は、製品添付のリリースノートを参照してください。本マニュアル作成時点の各製品の prver は、以下の通りです。

- HI7000/4 V.2.02.00：H'0220
- HI7700/4 V.2.02.00：H'0220
- HI7750/4 V.2.02.00：H'0220

(5) prno

prno は、製品管理情報や製品番号などを表します。

本カーネルの prno[0]から prno[3]の値は H'0000 です。

3.23 キャッシュサポート機能(HI7700/4: SH-3, SH3-DSP 用)

キャッシュサポート機能は、HI7700/4、HI7750/4 のみでサポートしている独自機能です。キャッシュサポート機能は、表 3.44のようにマイコンによって使用するライブラリが異なります。また、機能・API も一部異なります。

表3.44 キャッシュサポート機能のライブラリ

項番	カーネル	マイコン	操作対象キャッシュ	ライブラリ
1	**HI7700/4**	**SH-3, SH3-DSP**	**命令・オペランド混在キャッシュ**	**7708_cache_???.lib**
2		SH4AL-DSP(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4al_cache_???.lib
3		SH4AL-DSP(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib
4	HI7750/4	SH-4	オペランドキャッシュ	7750_cache_???.lib
5		SH-4A(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4a_cache_???.lib
6		SH-4A(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib

本節では、SH-3, SH3-DSP 用(7708_cache_???.lib)について解説します。
表 3.45に 7708_cache_???.lib でサポートしているサービスコール一覧を示します。

表3.45 キャッシュサポートサービスコール(SH-3, SH3-DSP 用)

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	vini_cac	キャッシュの初期化	○	○	○	○	○		
	ivini_cac		○	○	○	○	○		
2	vclr_cac	キャッシュのクリア	○	○	○	○	○		
	ivclr_cac		○	○	○	○	○		
3	vfls_cac	キャッシュのフラッシュ	○	○	○	○	○		
	ivfls_cac		○	○	○	○	○		
4	vinv_cac	キャッシュの無効化	○	○	○	○	○		
	ivinv_cac		○	○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

3.23.1 キャッシュの初期化(vini_cac, ivini_cac)

■C 言語 API

```
void vini_cac(UW ccr_data, UW entnum, UW waynum);
void ivini_cac(UW ccr_data, UW entnum, UW waynum);
```

■パラメータ

UW	ccr_data	R4	CPU のキャッシュ制御レジスタに設定する値
UW	entnum	R5	キャッシュのエントリ数
UW	waynum	R6	キャッシュのウェイ数

■リターンパラメータ

なし

■エラーコード

なし

■機能

キャッシュを初期化します。
キャッシュ機能を使用する場合には、その前に本サービスコールを呼び出す必要があります。
ccr_data には、CCR レジスタに設定する値を指定します。
entnum には、キャッシュのウェイあたりのエントリ数を指定します。
waynum には、使用するキャッシュのウェイ数を指定します。
entnum, waynum は使用する CPU の仕様や RAM モードに合わせて設定してください。異なる設定をした場合に動作は保証されません。
以下に、HI7700/4 のキャッシュサポートサービスコールが対応しているキャッシュと、vini_cac, ivini_cac で指定が必要なパラメータを示します。

キャッシュサイズ	代表的なマイコン	条件	パラメータ		
			ccr_data	entnum	waynum
8kB	SH7708 シリーズ, SH7709	内蔵 RAM モード未使用	デバイス仕様に合わせて指定してください。	128	4
		内蔵 RAM モード使用		128	2
16kB	SH7706, SH7709S, SH7727, SH7641, SH7660	−		256	4
32kB	SH7290, SH7294, SH7300, SH7705, SH7710	32kB モードを使用		512	4
		16kB モードを使用		256	4

本サービスコールは､使い方を誤るとキャッシュとメモリのコヒーレンシが保たれなくなる可能性があります。キャッシュ内容がメモリに書き出されていない可能性がある場合で、かつその内容をメモリに書き出す必要がある場合は、vfls_cac, ivfls_cac サービスコールでキャッシュの内容をフラッシュし、その後本サービスコールを発行してください。このとき、ccr_data には CCR.CF ビットが 1 となる値を指定してください。

また、SH7290 などの CCR3 レジスタの設定が必要な場合は、必ず本サービスコール発行前にアプリケーション側で CCR3 レジスタを設定してください。CCR3 レジスタの設定から vini_cac, ivini_cac からリターンするまでの期間では、キャッシュをアクセスしないようにしてください。例えば、キャッシュディスエーブル状態で CCR3 レジスタの設定と vini_cac, ivini_cac を行ってください。

同様に、SH7709S や SH7290 などの CCR2 レジスタの設定が必要な場合は、必ず本サービスコール発行前にアプリケーション側で CCR2 レジスタを設定してください。CCR2 レジスタの設定から vini_cac, ivini_cac からリターンするまでの期間では、キャッシュをアクセスしないようにしてください。例えば、キャッシュディスエーブル状態で CCR2 レジスタの設定と vini_cac, ivini_cac を行ってください。

本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。

3.23.2 キャッシュのクリア(vclr_cac, ivclr_cac)

■C 言語 API

```
ER ercd = vclr_cac(VP clradr1, VP clradr2);
ER ercd = ivclr_cac(VP clradr1, VP clradr2);
```

■パラメータ

VP	clradr1	R4	クリア先頭アドレス
VP	clradr2	R5	クリア最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR [p] パラメータエラー(clradr1＞clradr2)

■機能

キャッシュをクリアします。

clradr1 から clradr2 までのアドレスがキャッシュされていれば、それを無効にするとともに、その中でメモリに書き出されていないキャッシュ内容をメモリに書き出します。

clradr1, clradr2 には物理アドレス(0～H'1bffffff)を指定してください。それ以外のアドレスを指定した場合の動作は保証されません。

全てのキャッシュ内容を対象とする場合は、 clradr1=0, clradr2=H'1bffffff を指定してください。

SH-3,. SH3-DSP の場合、キャッシュラインサイズは 16 バイトのため、clradr1 の下位 4 ビットは全て 0 に、clradr2 の下位 4 ビットは全て 1 に補正して扱います。

vini_cac, ivini_cac サービルコールを発行する前に本サービスコールを呼び出した場合の動作は保証されません。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをクリアすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.23.3 キャッシュのフラッシュ(vfls_cac, ivfls_cac)

■C 言語 API

```
ER ercd = vfls_cac(VP flsadr1, VP flsadr2);
ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	R4	フラッシュ先頭アドレス
VP	flsadr2	R5	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR [p] パラメータエラー(flsadr1＞flsadr2)

■機能

キャッシュをフラッシュします。

flsadr1 から flsadr2 までのアドレスがキャッシュされていれば、その中でメモリに書き出されていないキャッシュ内容をメモリに書き出します。

flsadr1, flsadr2 には物理アドレス(0～H'1bffffff)を指定してください。それ以外のアドレスを指定した場合の動作は保証されません。

全てのキャッシュ内容を対象とする場合は、flsadr1=0, flsadr2=H'1bffffff を指定してください。

SH-3, SH3-DSP の場合、キャッシュラインサイズは 16 バイトのため、clradr1 の下位 4 ビットは全て 0 に、clradr2 の下位 4 ビットは全て 1 に補正して扱います。

vini_cac, ivini_cac サービルコールを発行する前に本サービスコールを呼び出した場合の動作は保証されません。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをフラッシュすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.23.4 キャッシュの無効化(vinv_cac, ivinv_cac)

■C 言語 API

```
ER ercd = vinv_cac(void);
ER ercd = ivinv_cac(void);
```

■パラメータ

なし

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

キャッシュ内容を無効にします。キャッシュに登録されていてメモリに書き出されていない内容があってもその内容は破棄されますので、注意してください。

本サービスコールは、キャッシュ制御レジスタ(CCR)のキャッシュフラッシュビット(CF)に 1 を設定）することで、キャッシュされている内容をすべて無効にします。

vini_cac, ivini_cac サービルコールを発行する前に本サービスコールを呼び出した場合の動作は保証されません。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュを無効化することができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

RENESAS

3.24 キャッシュサポート機能(HI7750/4:SH-4 用)

キャッシュサポート機能は、HI7700/4、HI7750/4 のみでサポートしている独自機能です。キャッシュサポート機能は、表 3.46のように使用するマイコン毎に使用するライブラリが異なります。また、機能・API も一部異なります。

表3.46 キャッシュサポート機能のライブラリ

項番	カーネル	マイコン	操作対象キャッシュ	ライブラリ
1	HI7700/4	SH-3, SH3-DSP	命令・オペランド混在キャッシュ	7708_cache_???.lib
2		SH4AL-DSP(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4al_cache_???.lib
3		SH4AL-DSP(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib
4	**HI7750/4**	**SH-4**	**オペランドキャッシュ**	**7750_cache_???.lib**
5		SH-4A(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4a_cache_???.lib
6		SH-4A(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib

本節では、SH-4 用(7750_cache_???.lib)について解説します。
表 3.47に 7750_cache_???.lib でサポートしているサービスコール一覧を示します。

表3.47 キャッシュサポートサービスコール(SH-4 用)

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	vini_cac	キャッシュの初期化	○	○	○	○	○		
	ivini_cac		○	○	○	○	○		
2	vclr_cac	オペランドキャッシュのクリア	○	○	○	○	○		
	ivclr_cac		○	○	○	○	○		
3	vfls_cac	オペランドキャッシュのフラッシュ	○	○	○	○	○		
	ivfls_cac		○	○	○	○	○		
4	vinv_cac	オペランドキャッシュの無効化	○	○	○	○	○		
	ivinv_cac		○	○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

3.24.1 キャッシュの初期化(vini_cac, ivini_cac)

■C 言語 API

```
void vini_cac(UW ccr_data);
void ivini_cac(UW ccr_data);
```

■パラメータ

UW	ccr_data	R4	CPU のキャッシュ制御レジスタに設定する値

■リターンパラメータ

なし

■エラーコード

なし

■機能

キャッシュを初期化します。
キャッシュ機能を使用する場合には、その前に本サービスコールを呼び出す必要があります。
ccr_data には、CCR レジスタに設定する値を指定します。

本サービスコールは、使い方を誤るとキャッシュとメモリのコヒーレンシが保たれなくなる可能性があります。キャッシュ内容がメモリに書き出されていない可能性がある場合で、かつその内容をメモリに書き出す必要がある場合は、vfls_cac, ivfls_cac サービスコールでキャッシュの内容をフラッシュし、その後本サービスコールを発行してください。このとき、ccr_data には CCR.CF ビットが 1 となる値を指定してください。
本サービスコールはカーネル起動前にも呼び出せます。
本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。

RENESAS

3.24.2 オペランドキャッシュのクリア(vclr_cac, ivclr_cac)

■C 言語 API

```
ER ercd = vclr_cac(VP clradr1, VP clradr2);
ER ercd = ivclr_cac(VP clradr1, VP clradr2);
```

■パラメータ

VP	clradr1	R4	クリア先頭アドレス
VP	clradr2	R5	クリア最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR [p] パラメータエラー(clradr1＞clradr2)

■機能

オペランドキャッシュをクリアします。

clradr1 から clradr2 までのアドレスがオペランドキャッシュに登録されていれば、それを無効にするとともに、その中でメモリに書き出されていないキャッシュ内容をメモリに書き出します。

clradr1, clradr2 には論理アドレスを指定してください。また、clradr1, clradr2 で決まる範囲には、以下のアドレスが含まれないようにしてください。これに違反した場合の動作は保証されません。

- 対応する物理アドレスがエリア 7
- P2, P4 領域

全てのオペランドキャッシュ内容を対象とする場合は、clradr1=H'80000000, clradr2=H'9bffffff を指定してください。

SH-4 の場合、キャッシュラインサイズは 32 バイトのため、clradr1 の下位 5 ビットは全て 0 に、clradr2 の下位 5 ビットは全て 1 に補正して扱います。

本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをクリアすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.24.3 オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac)

■C 言語 API

```
ER ercd = vfls_cac(VP flsadr1, VP flsadr2);
ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	R4	フラッシュ先頭アドレス
VP	flsadr2	R5	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR　[p]　パラメータエラー(flsadr1＞flsadr2)

■機能

オペランドキャッシュをフラッシュします。

flsadr1 から flsadr2 までのアドレスがオペランドキャッシュに登録されていれば、その中でメモリに書き出されていないキャッシュ内容をメモリに書き出します。

flsadr1, flsadr2 には論理アドレスを指定してください。また、flsadr1, flsadr2 で決まる範囲には、以下のアドレスが含まれないようにしてください。これに違反した場合の動作は保証されません。

- 対応する物理アドレスがエリア 7
- P2, P4 領域

全てのオペランドキャッシュ内容を対象とする場合は、flsadr1=H'80000000, flsadr2=H'9bffffff を指定してください。

SH-4 の場合、キャッシュラインサイズは 32 バイトのため、clradr1 の下位 5 ビットは全て 0 に、clradr2 の下位 5 ビットは全て 1 に補正して扱います。

本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをフラッシュすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.24.4 オペランドキャッシュの無効化(vinv_cac, ivinv_cac)

■C 言語 API

```
ER ercd = vinv_cac(VP invadr1, VP invadr2);
ER ercd = ivinv_cac(VP invadr1, VP invadr2);
```

■パラメータ

VP	invadr1	R4	無効化する先頭アドレス
VP	invadr2	R5	無効化する最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR　[p]　パラメータエラー(invadr1＞invadr2)

■機能

オペランドキャッシュ内容を無効にします。オペランドキャッシュに登録されていてメモリに書き出されていない内容があってもその内容は破棄されますので、注意してください。

invadr1 から invadr2 までのアドレスがキャッシュされていれば、それを無効にします。

invadr1, invadr2 には論理アドレスを指定してください。また、invadr1, invadr2 で決まる範囲には、以下のアドレスが含まれないようにしてください。これに違反した場合の動作は保証されません。

- 対応する物理アドレスがエリア 7
- P2, P4 領域

全てのオペランドキャッシュ内容を対象とする場合は、invadr1=H'80000000, invadr2=H'9bffffff を指定してください。

SH-4 の場合、キャッシュラインサイズは 32 バイトのため、invadr1 の下位 5 ビットは全て 0 に、invadr2 の下位 5 ビットは全て 1 に補正して扱います。

本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュを無効化することができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.25 キャッシュサポート機能(HI7700/4:SH4AL-DSP(拡張機能なし), HI7750/4:SH-4A(拡張機能なし)用)

キャッシュサポート機能は、HI7700/4、HI7750/4 のみでサポートしている独自機能です。キャッシュサポート機能は、表 3.48のように使用するマイコン毎に使用するライブラリが異なります。また、機能・API も一部異なります。

表3.48 キャッシュサポート機能のライブラリ

項番	カーネル	マイコン	操作対象キャッシュ	ライブラリ
1	**HI7700/4**	SH-3, SH3-DSP	命令・オペランド混在キャッシュ	7708_cache_???.lib
2		**SH4AL-DSP(拡張機能なし)**	**命令キャッシュ、オペランドキャッシュ**	**sh4al_cache_???.lib**
3		SH4AL-DSP(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib
4	**HI7750/4**	SH-4	オペランドキャッシュ	7750_cache_???.lib
5		**SH-4A(拡張機能なし)**	**命令キャッシュ、オペランドキャッシュ**	**sh4a_cache_???.lib**
6		SH-4A(拡張機能あり)	命令キャッシュ、オペランドキャッシュ	shx2_cache_???.lib

SH4AL-DSP(拡張機能なし)用(sh4al_cache_???.lib)と SH-4A(拡張機能なし)用(sh4a_cache_???.lib)については共通仕様であり、本節ではこの仕様について解説します。

なお、関連項目として、以下も必ず参照してください。

- 「5.7 SH4AL-DSP(HI7700/4)または SH-4A(HI7750/4)でキャッシュサポートサービスコールを使用する場合」
- 「5.11.1 コンパイラ、アセンブラの CPU オプション」

表 3.49に sh4al_cache_???.lib および sh4a_cache_???.lib でサポートしているサービスコール一覧を示します。

表3.49 キャッシュサポートサービスコール(SH4AL-DSP(拡張機能なし), SH-4A(拡張機能なし)用)

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	vini_cac	キャッシュの初期化	○	○	○	○	○		
	ivini_cac		○	○	○	○	○		
2	vclr_cac	命令・オペランドキャッシュのクリア	○	○	○	○	○		
	ivclr_cac		○	○	○	○	○		
3	vfls_cac	オペランドキャッシュのフラッシュ	○	○	○	○	○		
	ivfls_cac		○	○	○	○	○		
4	vinv_cac	命令・オペランドキャッシュの無効化	○	○	○	○	○		
	ivinv_cac		○	○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

3.25.1 キャッシュの初期化(vini_cac, ivini_cac)

■C 言語 API

```
ER ercd = vini_cac(ATR cacatr);
ER ercd = ivini_cac(ATR cacatr);
```

■パラメータ

ATR	cacatr	R4	キャッシュ属性

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

キャッシュを初期化します。具体的には、指定された cacatr に基づいて、表 3.50に示すようにプロセッサの CCR レジスタと RAMCR レジスタを設定します。

cacatr には、表 3.50の各項目の論理和を指定できますが、cacatr に指定された値のエラーチェックは一切行いません。
また、本サービスコールは、cacatr の指定内容に関わらず、CCR.ICI, OCI ビットに 1 を書き込みます。即ち、本サービスコール呼び出し以前のキャッシュの内容は全て破棄されます。

表3.50 キャッシュ属性

属性	値	意味	CCR,RAMCR 設定
TCAC_IC_ENABLE	H'00000100	命令キャッシュを Enable にする。 指定しない場合は Disable にする。	指定時：CCR.ICE=1 非指定時：CCR.ICE=0
TCAC_OC_ENABLE	H'00000001	オペランドキャッシュを Enable にする 指定しない場合は Disable にする。	指定時：CCR.OCE=1 非指定時 CCR.OCE=0
TCAC_IC_2WAY	H'00800000	命令キャッシュを 2WAY にする。 指定しない場合は 4WAY にする。	指定時：RAMCR.IC2W=1 非指定時：RAMCR.IC2W=0
TCAC_OC_2WAY	H'00400000	オペランドキャッシュを 2WAY にする。指定しない場合は 4WAY にする。	指定時：RAMCR.OC2W=1 非指定時：RAMCR.OC2W=0
TCAC_P1_CB	H'00000004	P1 領域へのライトをライトバックと扱う。指定しない場合は、ライトスルーと扱う	指定時：CCR.CB=1 非指定時：CCR.CB=0
TCAC_P0_WT	H'00000002	P0/U0/P3 領域へのライトをライトスルーと扱う。指定しない場合はライトバックと扱う。	指定時：CCR.WT=1 非指定時：CCR.WT=0

なお、本サービスコールでは CCR, RAMCR 以外のレジスタは一切変更しません。
本サービスコールはカーネル起動前にも呼び出せます。
本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。

3.25.2　命令・オペランドキャッシュのクリア(vclr_cac, ivclr_cac)

■C 言語 API

```
ER ercd = vclr_cac(VP clradr1, VP clradr2, MODE mode);
ER ercd = ivclr_cac(VP clradr1, VP clradr2, MODE mode);
```

■パラメータ

VP	clradr1	R4	クリア先頭アドレス
VP	clradr2	R5	クリア最終アドレス
MODE	mode	R6	対象キャッシュ指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)clradr1＞clradr2 (2)mode が不正
E_OBJ	[k]	mode によって決まる対象キャッシュが Disable

■機能

キャッシュをクリアします。即ち、キャッシュ内容を無効化すると共に、オペランドキャッシュにメモリに書き戻していないデータがあれば、それをメモリに書き戻します。
対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)

clradr1 は 32 の倍数に切り捨て、clradr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定
mode で決まるキャッシュに対し、論理アドレスが clradr1～clradr2 のエントリをクリアします。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)は、そのエントリがダーティ(メモリに書き出されていない)であれば、クリアする前にメモリへのライトバックを行います。

本サービスコールは、clradr1～clradr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBP 命令
- mode=TC_EXCLUDE_IC の場合：OCBP 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

この処理は、SR.BL=0, SR.I=15 の状態で行います。ただし、1 エントリを処理する毎に、SR を呼び出し時の値に戻します。つまり、呼び出し時の SR によって、割込みが受け付けられる可能性があります。

本サービスコールでは、clradr1, clradr2 については、エラーコード欄に記載した基本的なエラーチ

ェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（２）対象キャッシュの全エントリを対象とする

clradr1=0, clradr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリをクリアします。この場合、本サービスコールは以下のように処理します。

(1) modeがTC_FULLまたはTC_EXCLUDE_OCの場合は、CCR.ICI=1を設定することで、命令キャッシュの全エントリを無効化します。この処理は、SR.BL=1の状態で行います。

(2) modeがTC_FULLまたはTC_EXCLUDE_ICの場合は、オペランドキャッシュのメモリ割付キャッシュを操作し、全エントリについてダーティ(U=1)なエントリの内容をメモリにライトバックさせた後、全エントリを無効化(V=0)します。この処理は、SR.BL=1の状態で行います。ただし、1エントリを処理する毎に、SRを呼び出し時の値に戻します。つまり、呼び出し時のSRによって、割込みが受け付けられる可能性があります。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをクリアすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.25.3 オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac)

■C 言語 API

```
ER ercd = vfls_cac(VP flsadr1, VP flsadr2);
ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	R4	フラッシュ先頭アドレス
VP	flsadr2	R5	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(flsadr1＞flsadr2)
E_OBJ	[k]	オペランドキャッシュが Disable

■機能

オペランドキャッシュをフラッシュします。すなわち、メモリに書き出されていない内容をメモリに書き出します(ライトバック)。

flsadr1 は 32 の倍数に切り捨て、flsadr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定

論理アドレスが flsadr1～flsadr2 に対応する物理アドレスを含むオペランドキャッシュエントリをフラッシュします。すなわち、該当するオペランドキャッシュエントリがメモリに書き出されていない場合、メモリに書き出します。

本サービスコールでは、flsadr1～flsadr2 に対して、OCBWB 命令を繰り返して発行します。この処理は、SR.BL=0, SR.I=15 の状態で行います。ただし、1 エントリを処理する毎に、SR を呼び出し時の値に戻します。つまり、呼び出し時の SR によって、割込みが受け付けられる可能性があります。

本サービスコールでは、flsadr1, flsadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（2）全エントリを対象とする

flsadr1=0, flsadr2=H'fffffff を指定すると、オペランドキャッシュの全エントリをクリアします。この場合、本サービスコールは以下のように処理します。

- オペランドキャッシュのメモリ割付キャッシュを操作し、全エントリについてダーティ(U=1)なエントリの内容をメモリにライトバックさせます。この処理は、SR.BL=1 の状態で行います。ただし、1 エントリを処理する毎に、SR を呼び出し時の値に戻します。つまり、呼び出し時の SR によって、割込みが受け付けられる可能性があります。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをフラッシュすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.25.4 命令・オペランドキャッシュの無効化(vinv_cac, ivinv_cac)

■C 言語 API

```
ER ercd = vinv_cac(VP invadr1, VP invadr2, MODE mode);
ER ercd = ivinv_cac(VP invadr1, VP invadr2, MODE mode);
```

■パラメータ

VP	invadr1	R4	無効化する先頭アドレス
VP	invadr2	R5	無効化する最終アドレス
MODE	mode	R6	対象キャッシュ指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR [p] パラメータエラー
(1)invadr1＞invadr2
(2)mode が不正

E_OBJ [k] mode によって決まる対象キャッシュが Disable

■機能

キャッシュを無効化します。オペランドキャッシュにメモリに書き戻していないデータがあっても、それは破棄されます。
対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)

invadr1 は 32 の倍数に切り捨て、invadr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定
mode で決まるキャッシュに対し、論理アドレスが invadr1～invadr2 に対応する物理アドレスを含むエントリを無効化します。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)で、そのエントリがダーティ(メモリに書き出されていない)であっても、メモリへのライトバックは行いません。即ち、そのデータは消失することになります。

本サービスコールは、invadr1～invadr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBI 命令
- mode=TC_EXCLUDE_IC の場合：OCBI 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

この処理は、SR.BL=0, SR.I=15 の状態で行います。ただし、1 エントリを処理する毎に、SR を呼び出し時の値に戻します。つまり、呼び出し時の SR によって、割込みが受け付けられる可能性があります。

本サービスコールでは、invadr1, invadr2 については、エラーコード欄に記載した基本的なエラーチ

ェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（2）対象キャッシュの全エントリを対象とする

invadr1=0, invadr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリを無効化します。この場合、本サービスコールは mode に応じて CCR レジスタの以下のビットを操作します。この処理は、SR.BL=1 の状態で行います。

- mode=TC_FULL の場合：CCR.ICI と CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_IC の場合：CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_OC の場合：CCR.ICI に 1 を設定します。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュを無効化することができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.26 キャッシュサポート機能(HI7700/4：SH4AL-DSP，拡張機能あり，HI7750/4：SH-4A，拡張機能あり用)

キャッシュサポート機能は、HI7700/4、HI7750/4 のみでサポートしている独自機能です。キャッシュサポート機能は、表 3.51のように使用するマイコン毎に使用するライブラリが異なります。また、機能・API も一部異なります。

表3.51 キャッシュサポート機能のライブラリ

項番	カーネル	マイコン	操作対象キャッシュ	ライブラリ
1	**HI7700/4**	SH-3, SH3-DSP	命令・オペランド混在キャッシュ	7708_cache_???.lib
2		SH4AL-DSP(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4al_cache_???.lib
3		**SH4AL-DSP(拡張機能あり)**	**命令キャッシュ、オペランドキャッシュ**	**shx2_cache_???.lib**
4	**HI7750/4**	SH-4	オペランドキャッシュ	7750_cache_???.lib
5		SH-4A(拡張機能なし)	命令キャッシュ、オペランドキャッシュ	sh4a_cache_???.lib
6		**SH-4A(拡張機能あり)**	**命令キャッシュ、オペランドキャッシュ**	**shx2_cache_???.lib**

SH4AL-DSP(拡張機能あり)用(shx2_cache_???.lib)と SH-4A(拡張機能あり)用(shx2_cache_???.lib)については共通仕様であり、本節ではこの仕様について解説します。

なお、関連項目として、以下も必ず参照してください。

- 「5.7 SH4AL-DSP(HI7700/4)または SH-4A(HI7750/4)でキャッシュサポートサービスコールを使用する場合」
- 「5.11.1 コンパイラ、アセンブラの CPU オプション」

表 3.52に shx2_cache_???.lib および shx2_cache_???.lib でサポートしているサービスコール一覧を示します。

表3.52 キャッシュサポートサービスコール(SH4AL-DSP(拡張機能あり), SH-4A(拡張機能あり)用)

項番	サービスコール *1	機能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	vini_cac	キャッシュの初期化	○	○	○	○	○		
	ivini_cac		○	○	○	○	○		
2	vclr_cac	命令・オペランドキャッシュのクリア	○	○	○	○	○		
	ivclr_cac		○	○	○	○	○		
3	vfls_cac	オペランドキャッシュのフラッシュ	○	○	○	○	○		
	ivfls_cac		○	○	○	○	○		
4	vinv_cac	命令・オペランドキャッシュの無効化	○	○	○	○	○		
	ivinv_cac		○	○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルのサービスコール、"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼出し可能、"N"は非タスクコンテキストから呼出し可能
"E"はディスパッチ許可状態から呼出し可能、"D"はディスパッチ禁止状態から呼出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼出し可能
"C"は CPU 例外ハンドラから呼出し可能

3.26.1 キャッシュの初期化(vini_cac, ivini_cac)

■C 言語 API

```
ER ercd = vini_cac(ATR cacatr);
ER ercd = ivini_cac(ATR cacatr);
```

■パラメータ

ATR	cacatr	R4	キャッシュ属性

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）

■エラーコード

なし

■機能

キャッシュを初期化します。具体的には、指定された cacatr に基づいて、表 3.50に示すようにプロセッサの CCR レジスタと RAMCR レジスタを設定します。

cacatr には、表 3.50の各項目の論理和を指定できますが、cacatr に指定された値のエラーチェックは一切行いません。

また、本サービスコールは、cacatr の指定内容に関わらず、CCR.ICI, OCI ビットに 1 を書き込みます。即ち、本サービスコール呼び出し以前のキャッシュの内容は全て破棄されます。

表3.53　キャッシュ属性

属性	値	意味	CCR,RAMCR 設定
TCAC_IC_ENABLE	H'00000100	命令キャッシュを Enable にする。 指定しない場合は Disable にする。	指定時：CCR.ICE=1 非指定時：CCR.ICE=0
TCAC_OC_ENABLE	H'00000001	オペランドキャッシュを Enable にする 指定しない場合は Disable にする。	指定時：CCR.OCE=1 非指定時 CCR.OCE=0
TCAC_IC_2WAY	H'00800000	命令キャッシュを 2WAY にする。 指定しない場合は 4WAY にする。	指定時：RAMCR.IC2W=1 非指定時：RAMCR.IC2W=0
TCAC_OC_2WAY	H'00400000	オペランドキャッシュを 2WAY にする。指定しない場合は 4WAY にする。	指定時：RAMCR.OC2W=1 非指定時：RAMCR.OC2W=0
TCAC_P1_CB	H'00000004	P1 領域へのライトをライトバックと扱う。指定しない場合は、ライトスルーと扱う	指定時：CCR.CB=1 非指定時：CCR.CB=0
TCAC_P0_WT	H'00000002	P0/U0/P3 領域へのライトをライトスルーと扱う。指定しない場合はライトバックと扱う。	指定時：CCR.WT=1 非指定時：CCR.WT=0
TCAC_IC_WPD	H'00200000	命令キャッシュのウェイ予測を行わない。指定しない場合はウェイ予測を行う。	指定時：CCR.ICWPD=1 非指定時：CCR.ICWPD=0
TCAC_L2_ENABLE	H'00010000	2 次キャッシュを Enable にする。 指定しない場合は Disable にする。	指定時：RAMCR.L2E=1 非指定時：RAMCR.L2E=0
TCAC_L2_FC	H'00020000	2 次キャッシュ強制コヒーレンシモードにする。指定しない場合は、強制コヒーレンシモードにしない。	指定時：RAMCR.L2FC=1 非指定時：RAMCR.L2FC=0

2 次キャッシュを搭載していないマイコンを使用する場合は、TCAC_L2_ENABLE, TCAC_L2_FC を指定してはなりません。

なお、本サービスコールでは CCR, RAMCR 以外のレジスタは一切変更しません。
本サービスコールはカーネル起動前にも呼び出せます。
本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。

3.26.2 命令・オペランドキャッシュのクリア(vclr_cac, ivclr_cac)

■C 言語 API

```
ER ercd = vclr_cac(VP clradr1, VP clradr2, MODE mode);
ER ercd = ivclr_cac(VP clradr1, VP clradr2, MODE mode);
```

■パラメータ

VP	clradr1	R4	クリア先頭アドレス
VP	clradr2	R5	クリア最終アドレス
MODE	mode	R6	対象キャッシュ指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)clradr1＞clradr2 (2)mode が不正
E_OBJ	[k]	mode によって決まる対象キャッシュが Disable

■機能

キャッシュをクリアします。即ち、キャッシュ内容を無効化すると共に、オペランドキャッシュにメモリに書き戻していないデータがあれば、それをメモリに書き戻します。
対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)

clradr1 は 32 の倍数に切り捨て、clradr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが clradr1～clradr2 のエントリをクリアします。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)は、そのエントリがダーティ(メモリに書き出されていない)であれば、クリアする前にメモリへのライトバックを行います。

本サービスコールは、clradr1～clradr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBP 命令
- mode=TC_EXCLUDE_IC の場合：OCBP 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。

本サービスコールでは、clradr1, clradr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（２）対象キャッシュの全エントリを対象とする

clradr1=0, clradr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリをクリアします。この場合、本サービスコールは以下のように処理します。

(1) modeがTC_FULLまたはTC_EXCLUDE_OCの場合は、CCR.ICI=1を設定することで、命令キャッシュの全エントリを無効化します。この処理は、SR.BL=1の状態で行います。

(2) modeがTC_FULLまたはTC_EXCLUDE_ICの場合は、オペランドキャッシュのメモリ割付キャッシュの全エントリについてOCBP命令を実行することで、ダーティ(U=1)なエントリの内容をメモリにライトバックさせるとともに無効化(V=0)します。この処理を実行するときは、SRレジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをクリアすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.26.3　オペランドキャッシュのフラッシュ(vfls_cac, ivfls_cac)

■C 言語 API

```
ER ercd = vfls_cac(VP flsadr1, VP flsadr2);
ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	R4	フラッシュ先頭アドレス
VP	flsadr2	R5	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー(flsadr1＞flsadr2)
E_OBJ	[k]	オペランドキャッシュが Disable

■機能

オペランドキャッシュをフラッシュします。すなわち、メモリに書き出されていない内容をメモリに書き出します(ライトバック)。

flsadr1 は 32 の倍数に切り捨て、flsadr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定

論理アドレスが flsadr1～flsadr2 に対応する物理アドレスを含むオペランドキャッシュエントリをフラッシュします。すなわち、該当するオペランドキャッシュエントリがメモリに書き出されていない場合、メモリに書き出します。

本サービスコールは、flsadr1～flsadr2 に対して、OCBWB 命令を繰り返して発行します。この処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。

本サービスコールでは、flsadr1, flsadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（2）全エントリを対象とする

flsadr1=0, flsadr2=H'fffffff を指定すると、オペランドキャッシュの全エントリをクリアします。この場合、本サービスコールは以下のように処理します。

- オペランドキャッシュのメモリ割付キャッシュを操作の全エントリについて OCBWB 命令を実行することで、ダーティ(U=1)なエントリの内容をメモリにライトバックさせます。この処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。

本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュをフラッシュすることができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

3.26.4 命令・オペランドキャッシュの無効化(vinv_cac, ivinv_cac)

■C 言語 API

```
ER ercd = vinv_cac(VP invadr1, VP invadr2, MODE mode);
ER ercd = ivinv_cac(VP invadr1, VP invadr2, MODE mode);
```

■パラメータ

VP	invadr1	R4	無効化する先頭アドレス
VP	invadr2	R5	無効化する最終アドレス
MODE	mode	R6	対象キャッシュ指定

■リターンパラメータ

ER	ercd	R0	正常終了（E_OK）またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)invadr1＞invadr2 (2)mode が不正
E_OBJ	[k]	mode によって決まる対象キャッシュが Disable

■機能

キャッシュを無効化します。オペランドキャッシュにメモリに書き戻していないデータがあっても、それは破棄されます。
対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)

invadr1 は 32 の倍数に切り捨て、invadr2 は 32 の倍数-1 に 1 切り上げて扱います。

（1）アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが invadr1～invadr2 に対応する物理アドレスを含むエントリを無効化します。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)で、そのエントリがダーティ(メモリに書き出されていない)であっても、メモリへのライトバックは行いません。即ち、そのデータは消失することになります。

本サービスコールは、invadr1～invadr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBI 命令
- mode=TC_EXCLUDE_IC の場合：OCBI 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。

本サービスコールでは、invadr1, invadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが XY メモリ

本サービスコールの処理時間は、指定領域のサイズに比例します。

（2）対象キャッシュの全エントリを対象とする
invadr1=0, invadr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリを無効化します。この場合、本サービスコールは mode に応じて CCR レジスタの以下のビットを操作します。この処理は、SR.BL=1 の状態で行います。
- mode=TC_FULL の場合：CCR.ICI と CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_IC の場合：CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_OC の場合：CCR.ICI に 1 を設定します。

HI7750/4 では、本サービスコールはカーネル起動前にも呼び出せます。
本サービスコールは例外ブロック状態(SR.BL=1)からも呼び出せます。これにより、例外、割込み発生とは不可分に、キャッシュを無効化することができます。SR.BL が 1 の状態で本サービスコールを呼び出した場合は、リターン後に必ず SR.BL ビットを 0 に戻してください。

4. アプリケーションプログラムの作成

4.1 ヘッダファイル

4.1.1 C/C++言語用

(1) itron.h

itron.h は、ITRON 仕様共通定義が記述された C/C++言語用ヘッダファイルです。
itron.h は hihead フォルダにあります。

(2) kernel.h と kernel_macro.h

kernel.h は、μITRON4.0 カーネル仕様の定義が記述された C/C++言語用ヘッダファイルです。
kernel.h は、itron.h とコンフィギュレータが出力する kernel_macro.h をインクルードしています。
kernel.h は hihead フォルダにあります。
「3. サービスコール」では、これらのヘッダファイルで定義しているデータタイプ、定数、マクロなどを使用して説明していますが、「3. サービスコール」には現われない定数、マクロもあります。これを、表 6.1 に示します。詳細は、各ヘッダファイルの内容を参照してください。

表4.1 定数、マクロ

項番	ファイル	定数、マクロ	説明
1	kernel.h	TMIN_TPRI	タスク優先度の最小値（常に 1）
2		TMIN_MPRI	メッセージ優先度の最小値（常に 1）
3		TKERNEL_MAKER	カーネルのメーカーコード ref_ver で返る maker と同じ値です。
4		TKERNEL_PRID	カーネルの識別番号 ref_ver で返る prid と同じ値です。
5		TKERNEL_SPVER	準拠する ITRON 仕様のバージョン番号 ref_ver で返る spver と同じ値です。
6		TKERNEL_PRVER	カーネルのバージョン番号 ref_ver で返る prver と同じ値です。
7		TMAX_ACTCNT	タスクの起動要求キューイング数の最大値（常に 15）
8		TMAX_WUPCNT	タスクの起床要求キューイング数の最大値（常に 15）
9		TMAX_SUSCNT	タスクの強制待ち要求ネスト数の最大値（常に 15）
10		TBIT_TEXPTN	タスク例外要因のビット数（常に 32）
11		TBIT_FLGPTN	イベントフラグのビット数（常に 32）
12		SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz);	サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールのサイズ（バイト数）
13		SIZE size =VTSZ_MPFMB(UINT blkcnt, UINT blksz);	サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールの管理領域のサイズ（バイト数）

表 4.1　定数、マクロ(続き)

項番	ファイル	定数、マクロ	説明
14	kernel_macro.h	SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz); *1	サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる可変長メモリプールのサイズ（目安のバイト数）
15		SIZE mplsz = VTSZ_MPLMB(UINT sctnum); *2	VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）
16		TMAX_MAXSEM	セマフォの最大資源数の最大値（常に 65535）
17		TIC_NUME	タイムティックの周期の分子
18		TIC_DENO	タイムティックの周期の分母
19		TMAX_TPRI	タスク優先度の最大値
20		TMAX_MPRI	メッセージ優先度の最大値
21		VTCFG_TBR *3	CFG_TBR の設定を示します。 0：「カーネル管理外」 1：「サービスコール呼び出し専用に使用」 2：「タスクコンテキストと扱う」
22		VTCFG_REGBANK *3	CFG_REGBANK の設定を示します。 0：レジスタバンクを使用しない(NOTUSE) 1：NMI,UBC を除く全割込みでレジスタバンクを使用(ALL) 2：レジスタバンクを使用する割込みレベルを選択する(SELECT) 3：レジスタバンクを持たないマイコン(NOBANK)
23		VTCFG_BANKLVLnn *3 (nn : “01”～15”)	割込みレベル nn について、レジスタバンクを使用するかどうかを示します。 0：レジスタバンクを使用しない 1：レジスタバンクを使用する CFG_REGBANK が NOBANK 以外の場合のみ、定義されます。
24		VTCFG_MPFMANAGE	CFG_MPFMANAGE の設定を示します。 0：プール領域内にカーネル管理情報を配置する(従来方式) 1：プール領域内にカーネル管理情報を配置しない(拡張方式)
25		VTCFG_NEWMPL	CFG_NEWMPL の設定を示します。 0：従来方式 1：新方式(断片化軽減、高速化)

【注】 *1　CFG_NEWMPL 選択有無によって、定義内容が異なります。
*2　CFG_NEWMPL 選択時のみ有効
*3　HI7000/4 のみ

(3) ID 名称ヘッダファイル(kernel_id.h, kernel_id_sys.h)

コンフィギュレータでは、各オブジェクトに対して ID 名称を設定することができます。コンフィギュレータは、指定された ID 名称を C 言語用の ID 名称ヘッダファイルに出力します。

ID 名称ヘッダファイルをインクルードすることで、以下のようにコンフィギュレータで指定した ID 名称を ID 番号として使用することができます。

```
wup_tsk(ID_main);
```

詳細は、「5.4.4(1) kernel_id.h, kernel_id_sys.h」を参照してください。

4.1.2 アセンブリ言語用

(1) itron.inc

itron.inc は、ITRON 仕様共通定義が記述されたアセンブリ言語用ヘッダファイルです。
itron.inc は hihead フォルダにあります。

(2) hihead¥kernel.inc

kernel.inc は、μITRON4.0 カーネル仕様の定義が記述されたアセンブリ言語用ヘッダファイルです。
kernel.inc は、itron.inc をインクルードしています。
kernel.inc は hihead フォルダにあります。

4.2 CPU リソースの扱い等

4.2.1 SR レジスタ

(1) SR.IMASK (割込みマスク)ビット

「2.18.3 割込みの禁止」を参照してください。

(2) SR.MD (処理モード)ビット (SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A)

アプリケーションは、SR.MD を 0 にしてはなりません。また、以下の箇所で指定する SR の MD ビットを必ず 1 にしてください。

- def_inh, idef_inh, def_exc, idef_exc, vdef_trp, ivdef_trp サービスコール
- コンフィギュレータでの割込みハンドラ、CPU 例外ハンドラ(TRAPA を含む)の定義

(3) SR.RB (レジスタバンク)ビット(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A)

アプリケーションは、バンク 1 の汎用レジスタを変更してはなりません。また、SR.RB を 1 にしてはなりません。以下の箇所で指定する SR の RB ビットを必ず 0 にしてください。

- def_inh, idef_inh, def_exc, idef_exc, vdef_trp, ivdef_trp サービスコール
- コンフィギュレータでの割込みハンドラ、CPU 例外ハンドラ(TRAPA を含む)の定義

(4) SR.BL (例外ブロック)ビット(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A)

例外ブロック状態では、特に断りのあるものを除いてサービスコールを発行してはなりません。

割込みを禁止する目的で使用する場合は、「2.18.3 割込みの禁止」を参照してください。

(5) SR.DSP (DSP 演算モード)ビット(SH3-DSP, SH4AL-DSP)

SH3-DSP および SH4AL-DSP を使用する場合は、以下を行ってください。

(a) コンフィギュレータで、CFG_DSPをチェックしてください。これにより、タスク起動時のSR.DSPビットが1になります。

(b) 割込みハンドラ、CPU例外ハンドラ(TRAPAを含む)
これらのハンドラでDSP演算を行う場合は、ハンドラ定義時に指定する「ハンドラのSR」のDSPビットを1にしてください。なお、ハンドラの定義方法には、以下があります。

- def_inh, idef_inh, def_exc, idef_exc, vdef_trp, ivdef_trp サービスコール
- コンフィギュレータでの割込みハンドラ、CPU 例外ハンドラ(TRAPA を含む)の定義

ただし、SH-3, SH3-DSPでキャッシュロック機能を使用する場合は「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」を参照してください。

(6) SR.CL (キャッシュロック)ビット(SH-3)

SR レジスタに CL ビットを持つマイコンでキャッシュロック機能を使用する場合は、「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」を参照してください。

(7) SR.FD (FPU ディスエーブル)ビット(SH-4, SH-4A)

アプリケーションは、SR.FD を 1 にしてはなりません。つまり、SH-4 の FPU 抑止例外を利用することはできません。また、以下の箇所で指定する SR の FD ビットを必ず 0 にしてください。

- def_inh, idef_inh, def_exc, idef_exc, vdef_trp, ivdef_trp サービスコール
- コンフィギュレータでの割込みハンドラ、CPU 例外ハンドラ(TRAPA を含む)の定義

また、「付録 G SH2A-FPU, SH-4, SH-4A における FPU に関する注意」も参照してください。

4.2.2 キャッシュロック機能(SH-3, SH3-DSP)

キャッシュロック機能をサポートしているマイコンでキャッシュロック機能を使用する場合は、以下を行ってください。なお、カーネルはキャッシュロックに関する制御は行いません。キャッシュロックの制御はアプリケーション側で行ってください。

(1) キャッシュ制御レジスタ 2(CCR2)にロックイネーブルビット(LE)があるマイコン

このタイプのマイコンでは、CCR2.LE=1 の場合にキャッシュロック機能が Enable となります。キャッシュロック機能の Enable/Disable 制御は、CCR2.LE ビットで行ってください。

(2) キャッシュ制御レジスタ 2(CCR2)にロックイネーブルビット(LE)がないマイコン

このタイプのマイコンでは、SR.DSP=1 または SR.CL=1 の場合にキャッシュロック機能が Enable となります。すなわち、キャッシュロック状態を維持するには、タスクやハンドラなどが実行するときには常に SR.DSP または SR.CL が 1 になっている必要があります。そのためには、以下を行う必要があります。

(a) コンフィギュレータで、CFG_DSPまたはCFG_CACLOCをチェックしてください。これにより、タスク起動時およびカーネル実行中のSR.DSPビットまたはCLビットが1となります。

(b) すべての割込みハンドラ、CPU例外ハンドラ(TRAPA含む)について、定義時に指定する「ハンドラのSR」のDSPビットまたはCLビットを1にしてください。

(c) アプリケーションでSR.DSPビットまたはCLビットををクリアしないでください。

これらが行われない場合、意図しないタイミングで SR.DSP ビットまたは CL ビットが 0 となり、キャッシュロックが解除される場合があります。

4.2.3 VBR レジスタ

VBR はカーネル起動時に初期化されます。アプリケーションで VBR を変更してはなりません。

4.2.4 MMU(SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A)

本カーネルは、一切 MMU を制御しません。

4.2.5 SR.BL=1 期間中の NMI の受付け (SH-3, SH3-DSP, SH4AL-DSP, SH-4, SH-4A)

これらのマイコンの中には、SR.BL=1 の時に発生した NMI を SR.BL=0 になるまで保留するか、即時に検出するかを、割込みコントローラで設定できるものがあります。**即時に検出する設定にした場合、NMI 割込みハンドラ終了後にシステムの正常動作に復帰することは保証されません。**

4.2.6 割込みのネスト(SH-3, SH3-DSP, SH4AL-DSP,. SH-4, SH-4A)

NMI を含めて 64 回以上割込みがネストした場合、システムの正常な動作は保証されません。割込みが 64 回以上ネストしないようにしてください。

4.2.7 32 ビットアドレス拡張モード(SH-4A)

32 ビットアドレス拡張モードを使用する場合、カーネルを起動する前に、P1, P2 領域の振る舞いが 29 ビットアドレスモードと同じになるように PMB を設定してください。

4.2.8 TBR レジスタ(SH-2A, SH2A-FPU)

TBR レジスタをどのように使用するかは、コンフィギュレータの CFG_TBR によって指定します。

(1) 「カーネル管理外」

カーネルは一切制御を行いません。

(2) 「サービスコール呼び出し専用に使用」

サービスコール呼び出し専用に TBR を使用します。サービスコール呼び出しが高速になります。
TBR は、カーネルが初期化します。アプリケーションで **TBR を変更した場合は、システムの正常な動作は保証されません。**
コンパイラの TBR オプション、および#pragma tbr は使用してはなりません。

(3) 「タスクコンテキストと扱う」

タスク内で TBR を任意の値に設定して使用することができます。タスク起動時の TBR は不定です。タスク以外のハンドラ等で TBR を変更する場合は、そのハンドラ内で TBR を保証しなければなりません。

4.2.9 レジスタバンク(SH-2A, SH2A-FPU)

SH-2A および SH2A-FPU では、割込み応答を高速化するためのレジスタバンクをサポートしています。レジスタバンクをどのように使用するかは、コンフィギュレータの[割込み情報の変更]ダイアログボックスで設定します。

(1) 使用するマイコンがレジスタバンクをサポートしており、NMI および UBC を除く全割込みでレジスタバンクを使用する場合

CFG_REGBANK で[NMI, UBC を除く全割込みでレジスタバンクを使用(ALL)]を選択し、CFG_IBNR_ADR に IBNR レジスタアドレスを指定してください。この場合、カーネルは初期化(vsta_knl)時に IBNR レジスタを 0x4000 に初期化します。

(2) 使用するマイコンがレジスタバンクをサポートしており、割込みレベルごとにレジスタバンクを使用するかどうかを選択する場合

CFG_REGBANK で[レジスタバンクを使用する割込みレベルを選択する(SELECT)]を選択し、CFG_BANKLVL01～CFG_BANKLVL15 のうちレジスタバンクを使用する割込みレベルについてチェックしてください。また CFG_IBNR_ADR に IBNR レジスタアドレスを指定してください。この場合、カーネルは初期化(vsta_knl)時に IBNR レジスタを 0xC000 に、IBCR レジスタを CFG_BANKLVL01～CFG_BANKLVL15 の設定に従って初期化します。

(3) 使用するマイコンがレジスタバンクをサポートしているが、レジスタバンクを一切使用しない場合

CFG_REGBANK で[レジスタバンクを使用しない(NOTUSE)]を選択し、CFG_IBNR_ADR に IBNR レジスタアドレスを指定してください。この場合、カーネルは初期化(vsta_knl)時に IBNR レジスタを 0 に初期化します。

(4) 使用するマイコンがレジスタバンクをサポートしていない場合

SH-2A 以外のマイコンなど、レジスタバンクをサポートしていないマイコンを使用する場合は、CFG_REGBANK で[レジスタバンクを持たないマイコン(NOBANK)]を選択してください。

なお、CFG_REGBANK の設定に応じて割込みハンドラの記述方法が変わります。詳細については、「4.8 割込みハンドラ」を参照してください。

4.3 SH2A-FPU, SH-4, SH-4A を使用する場合

FPU 機能を使用しない場合でも、必ず「付録 G SH2A-FPU, SH-4, SH-4A における FPU に関する注意」を参照してください。

4.4 システム予約

4.4.1 予約名

C 言語レベルでの"_kernel_", "_KERNEL_", "hi_"で始まる外部定義名はカーネル用に予約されているため、アプリケーションでは使用しないでください。

4.4.2 予約トラップ(HI7000/4 のみ)

HI7000/4 では、TRAPA #25, 26 のトラップ命令はカーネルで使用しているため、アプリケーションでは使用できません。

4.5 タスク

(1) 記述方法

タスクは、図 4.1に示すように通常の C 言語関数として記述します。また、サンプル提供の task.c も参考にしてください。タスクを終了する場合は、ext_tsk または exd_tsk サービスコールを用いて終了してください。ext_tsk, exd_tsk サービスコールを発行せずにタスク関数からリターンした場合は、ext_tsk サービスコールを発行した場合と同じ動作となります。

```
#include "kernel.h"
#pragma noregsave(Task)      ←タスク関数はレジスタを保証する必要はないので、#pragma
                               noregsave を指定することができます。
void Task(VP_INT exinf)      ←タスク生成時に TA_ACT 属性が指定されて起動された場合、および
                               act_tsk で起動された場合には、パラメータとしてタスク生成時に指
{                              定された exinf が渡され、sta_tsk で起動された場合は sta_tsk で
    /* タスクの処理 */          指定された stacd が渡されます。
    ext_tsk();               ←タスクを終了する場合は、ext_tsk または exd_tsk
                               サービスコールを呼び出してください。
}                            ←関数の終了で、ext_tsk が自動的に呼び出されます。
```

図4.1 タスクの C 言語記述例

(2) レジスタ使用規約

表 4.2, 表 4.3, 表 4.4に、それぞれ HI7000/4, HI7700/4, HI7750/4 におけるタスクのレジスタ使用規約を示します。

RENESAS

表4.2　タスクのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスクのアドレス
2	SR	○	*3	H'00000000
3	R0～R3	○		不定
4	R4	○		TA_ACT 属性または act_tsk で起動された場合はタスク生成時に指定した exinf、sta_tsk で起動された場合は sta_tsk で指定された stacd
5	R5～R14, MACH, MACL, GBR	○		不定
6	R15	○	○	タスクのスタック領域最終アドレス
7	PR	○	○	不定
8	[SH2-DSP] DSR	*4		TA_COP0 属性が指定されている場合は 0、その他の場合は不定
9	[SH2-DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	*4		不定
10	[SH-2A, SH2A-FPU]TBR	*5		不定
11	[SH2A-FPU] FPSCR,	*6		H'00040001
12	[SH2A-FPU] FPUL FR0～FR15	*6		不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　タスク開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CPU ロック状態である場合を除き、IMASK=0 でなければなりません。

*4　TA_COP0 属性の場合のみ

*5　CFG_TBR の設定に依存します。

(1)「カーネル管理外」：カーネルは TBR を一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。

*6　TA_COP1 属性の場合のみ

表4.3　タスクのレジスタ使用規約(HI7700/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスクのアドレス
2	SR	○	*3	CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は H'40001000，いずれもチェックしない場合は H'40000000
3	R0_BANK0～R3_BANK0	○		不定
4	R4_BANK0	○		TA_ACT 属性または act_tsk で起動された場合はタスク生成時に指定した exinf、sta_tsk で起動された場合は sta_tsk で指定された stacd
5	R5_BANK0～R7_BANK0, R8～R14, MACH, MACL, GBR	○		不定
6	R15	○	○	タスクのスタック領域最終アドレス
7	PR	○	○	不定
8	[SH3-DSP, SH4AL-DSP] DSR	*4		TA_COP0 属性が指定されている場合は 0、その他の場合は不定
9	[SH3-DSP, SH4AL-DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	*4		不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　タスク開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は、DSP/CL=1 でなければなりません。CPU ロック状態である場合を除き、IMASK=0 でななければなりません。また、MD=1, BL=0, RB=0 でなければなりません。

*4　TA_COP0 属性の場合のみ

表4.4　タスクのレジスタ使用規約(HI7750/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスクのアドレス
2	SR	○	*3	H'40000000
3	R0_BANK0～R3_BANK0	○		不定
4	R4_BANK0	○		TA_ACT 属性または act_tsk で起動された場合はタスク生成時に指定した exinf、sta_tsk で起動された場合は sta_tsk で指定された stacd
5	R5_BANK0～R7_BANK0, R8～R14, MACH, MACL, GBR	○		不定
6	R15	○	○	タスクのスタック領域最終アドレス
7	PR	○	○	不定
8	[SH-4, SH-4A] FPSCR,	○		H'00040001
9	[SH-4, SH-4A] FPUL	*4		不定
10	[SH-4, SH-4A] FR0_BANK0～FR15_BANK0	*5		不定
11	[SH-4, SH-4A] FR0_BANK1～FR15_BANK1	*6		不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　タスク開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CPU ロック状態である場合を除き、IMASK=0 でななければなりません。また、MD=1, BL=0, RB=0, FD=0 でなければなりません。

*4　TA_COP1,TA_COP2 属性の少なくとも一方が指定された場合のみ

*5　TA_COP1 属性の場合のみ

*6　TA_COP2 属性の場合のみ

(3) タスク管理情報の初期値

タスク起動時には表 4.5に示すようにタスク管理情報が初期化されます。

表4.5 タスク管理情報の初期化内容

項番	項目	初期化の内容
1	**タスクベース優先度**	**タスク生成時に指定したタスク起動優先度**(itskpri)
2	**タスク現在優先度**	**タスク生成時に指定したタスク起動優先度**(itskpri)
3	**タスク起床要求キューイング数**	0
4	**タスク強制待ち要求ネスト数**	0
5	**タスク付属イベントフラグ**	0
6	**タスク例外処理**	**禁止状態**
7	**保留例外要因**	0
8	**タスク実行モード**	0

(4) タスクの生成方法

タスクは、以下の方法で生成します。

- **ダイナミックスタックを使用するタスク、またはアプリケーションで確保したスタックを使用するタスク**

- cre_tsk, icre_tsk, acre_tsk, iacre_tsk サービスコール
- コンフィギュレータでの初期登録

- **スタティックスタックを使用するタスク**

- vscr_tsk サービスコール
- コンフィギュレータでの初期登録

RENESAS

4.6　タスク例外処理ルーチン

(1) 記述方法

タスク例外処理ルーチンは、図 4.2に示すように通常の C 言語関数として記述します。

```
#include "kernel.h"
#pragma noregsave(Texrtn)                   ←タスク例外処理ルーチン関数はレジスタを保証す
                                              る必要はないので、#pragma noregsave を指定
                                              することができます。
void Texrtn(TEXPTN texptn,VP_INT exinf)    ←パラメータとして、例外要因と拡張情報が
{                                             渡されます。
    /* タスク例外処理ルーチンの処理 */
}
```

図4.2　タスク例外処理ルーチンの C 言語記述例

(2) レジスタ使用規約

表 4.6, 表 4.7, 表 4.8に、それぞれ HI7000/4, HI7700/4, HI7750/4 におけるタスク例外処理ルーチンのレジスタ使用規約を示します。

表4.6　タスク例外処理ルーチンのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスク例外処理ルーチンのアドレス
2	SR	○	*3	0
3	R0～R3	○		不定
4	R4	○		例外要因パターン
5	R5	○		タスク例外処理ルーチンの拡張情報
6	R6～R14, MACH, MACL, GBR	○		不定
7	R15	○	○	タスクのスタック領域を指しています。
8	PR	○	○	不定
9	[SH2-DSP] DSR	*4		TA_COP0 属性が指定されている場合は 0、その他の場合は不定
10	[SH2-DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	*4		不定
11	[SH-2A, SH2A-FPU]TBR	*5	*5	*5
12	[SH2A-FPU] FPSCR,	*6		H'00040001
13	[SH2A-FPU] FPUL, FR0～FR15	*6		不定

【注】 *1 使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2 タスク例外処理ルーチン開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3 CPU ロック状態である場合を除き、IMASK=0 でなければなりません。

*4 TA_COP0 属性の場合のみ

*5 CFG_TBR の設定に依存します。

(1)「カーネル管理外」：カーネルは TBR を一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。終了時には、起動時の値に戻さなければなりません。初期値はタスク例外処理ルーチン起動直前の元のタスクの TBR と同じです。

*6 TA_COP1 属性の場合のみ

RENESAS

表4.7　タスク例外処理ルーチンのレジスタ使用規約(HI7700/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスク例外処理ルーチンのアドレス
2	SR	○	*3	CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は H'40001000，いずれもチェックしない場合は H'40000000
3	R0_BANK0～R3_BANK0	○		不定
4	R4_BANK0	○		例外要因パターン
5	R5_BANK0	○		タスク例外処理ルーチンの拡張情報
6	R6_BANK0, R7_BANK0, R8～R14, MACH, MACL, GBR	○		不定
7	R15	○	○	タスクのスタック領域を指しています。
8	PR	○	○	不定
9	[SH3-DSP, SH4AL-DSP] DSR	*4		TA_COP0 属性が指定されている場合は 0、その他の場合は不定
10	[SH3-DSP, SH4AL-DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	*4		不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　タスク例外処理ルーチン開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は、DSP/CL=1 でなければなりません。CPU ロック状態である場合を除き、IMASK=0 でななければなりません。また、MD=1, BL=0, RB=0 でなければなりません。

*4　TA_COP0 属性の場合のみ

表4.8　タスク例外処理ルーチンのレジスタ使用規約(HI7750/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		タスク例外処理ルーチンのアドレス
2	SR	○	*3	H'40000000
3	R0_BANK0～R3_BANK0	○		不定
4	R4_BANK0	○		例外要因パターン
5	R5_BANK0	○		タスク例外処理ルーチンの拡張情報
6	R6_BANK0, R7_BANK0, R8～R14, MACH, MACL, GBR	○		不定
7	R15	○	○	タスクのスタック領域を指しています。
8	PR	○	○	不定
9	[SH-4, SH-4A] FPSCR,	○		H'00040001
10	[SH-4, SH-4A] FPUL	*4		不定
11	[SH-4, SH-4A] FR0_BANK0～FR15_BANK0	*5		不定
12	[SH-4, SH-4A] FR0_BANK1～FR15_BANK1	*6		不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　タスク例外処理ルーチン開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CPU ロック状態である場合を除き、IMASK=0 でななければなりません。また、MD=1, BL=0, RB=0, FD=0 でなければなりません。

*4　TA_COP1,TA_COP2 属性の少なくとも一方が指定された場合のみ

*5　TA_COP1 属性の場合のみ

*6　TA_COP2 属性の場合のみ

(3)　タスク例外処理の定義

タスク例外処理は、以下の方法で生成します。

- def_tex, idef_tex サービスコール
- コンフィギュレータでの定義

4.7 拡張サービスコールルーチン

(1) 記述方法

拡張サービスコールルーチンは、図 4.3に示すように通常の C 言語関数として記述します。

```
#include "kernel.h"
ER_UINT Svcrtn(VP_INT par1, VP_INT par2) ←拡張サービスコールルーチンには、cal_svc で指定
{                                           したパラメータが渡されます。
    /* 拡張ｻｰﾋﾞｽｺｰﾙﾙｰﾁﾝの処理 */              ←発行元にリターン値を返します。
    return E_OK;
}
```

図4.3 拡張サービスコールルーチンの C 言語記述例

(2) レジスタ使用規約

拡張サービスコールルーチンは、cal_svc, ical_svc サービスコールによって、単なる関数呼出しを行った場合と同様に呼び出されます。したがって、拡張サービスコールルーチンのレジスタ使用規約は通常の C 言語関数と同じです。詳細は、『SuperH™ RISC engine C/C++コンパイラ ユーザーズマニュアル』を参照してください。

R4～R7 には、cal_svc で指定した第 1～第 4 パラメータが設定されます。

拡張サービスコールルーチンで DSP, FPU のレジスタを使用できるかは、cal_svc, ical_svc 発行元に依存しますので、注意してください。

(3) 拡張サービスコールルーチンの定義方法

拡張サービスコールルーチンは、以下の方法で生成します。

- def_svc, idef_svc サービスコール
- コンフィギュレータでの初期定義

4.8 割込みハンドラ

割込みハンドラには、通常の割込みハンドラと HI7000/4 専用のダイレクト割込みハンドラがあります。ダイレクト割込みハンドラの割込み応答は、通常の割込みハンドラよりも高速です。

4.8.1 通常の割込みハンドラ

通常の割込みハンドラは、割込みが発生するとカーネルの出入口処理ルーチンを介して起動されます。

なお、HI7000/4 ではカーネル割込みマスクレベル(CFG_KNLMSKLVL)よりも高いレベルの割込み(NMI を含む)は、ダイレクト割込みハンドラとして記述・定義する必要があります。**これらのハンドラを通常の割込みハンドラとして記述・定義した場合、システムの正常な動作は保証されません。**

(1) 記述方法

通常の割込みハンドラは、図 4.4のように通常の C 言語関数として記述します。

```
#include "kernel.h"
#pragma noregsave(Inh)      ←SH-2A, SH2A-FPU でバンク割込みを使用する場合(CFG_REGBANK
                              をチェック)のみ、レジスタを保証する必要はないので、#pragma
                              noregsave を指定することができます。
void Inh(void)              ←割込みハンドラは、void 型の関数として記述します。
{
    /* 割込みハンドラの処理 */
}
```

図4.4 通常の割込みハンドラの C 言語記述例

(2) レジスタ使用規約

表 4.9, 表 4.10, 表 4.11に、それぞれ HI7000/4, HI7700/4, HI7750/4 における通常の割込みハンドラのレジスタ使用規約を示します。

表4.9　通常の割込みハンドラのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		割込みハンドラのアドレス
2	SR	○	*3	IMASK：割込みレベル その他のビット：割込み発生前と同じ
3	R0〜R7	○		不定
4	R8〜R14, MACH, MACL, GBR	○	○ *4	不定
5	R15	○	○	割込みハンドラスタックを指しています。 割込み発生時には、カーネルの出入口処理でスタックを割込みハンドラ用のスタックに切り替えます。すべての割込みハンドラは、同じ割込みハンドラスタックを使用します。 割込みハンドラスタックのサイズは、CFG_IRQSTKSZ に指定します。割込みはネストする可能性があるので、割込みハンドラスタックのサイズは割込みのネストを考慮として算出する必要があります。詳しくは、「C.7 割込みハンドラのスタック」を参照してください。
6	PR	○	○	不定
7	[SH2-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	不定
8	[SH-2A, SH2A-FPU]TBR	*5	*5	*5
9	[SH2A-FPU] FPSCR,	○	○	不定
10	[SH2A-FPU] FR0〜FR11	○	○	不定
11	[SH2A-FPU] FPUL, FR12〜FR15	○	○	不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　割込みハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　IMASK ビットは、自割込みレベルよりも下げてはなりません。

*4　SH-2A または SH2A-FPU において、CFG_REGBANK をチェックした場合のみ不要です。

*5　CFG_TBR の設定に依存します。

(1)「カーネル管理外」：カーネルは TBR を一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。終了時には、起動時の値に戻さなければなりません。初期値は不定です。

表4.10　通常の割込みハンドラのレジスタ使用規約(HI7700/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		割込みハンドラのアドレス
2	SR	○	*3	ハンドラ定義時に指定した値
3	R0_BANK0～R7_BANK0	○		不定
4	R8～R14, MACH, MACL, GBR	○	○	不定
5	R15	○	○	割込みハンドラスタックを指しています。 割込み発生時には、カーネルの出入口処理でスタックを割込みハンドラ用のスタックに切り替えます。すべての割込みハンドラは、同じ割込みハンドラスタックを使用します。 割込みハンドラスタックのサイズは、CFG_IRQSTKSZ に指定します。割込みはネストする可能性があるので、割込みハンドラスタックのサイズは割込みのネストを考慮として算出する必要があります。詳しくは、「C.7 割込みハンドラのスタック」を参照してください。
6	PR	○	○	不定
7	[SH3-DSP, SH4AL-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　割込みハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は、DSP/CL=1 でなければなりません。また、MD=1 でなければなりません。また、IMASK ビットは、自割込みレベルよりも下げてはなりません。

RENESAS

表4.11　通常の割込みハンドラのレジスタ使用規約(HI7750/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		割込みハンドラのアドレス
2	SR	○	*3	ハンドラ定義時に指定した値
3	R0_BANK0～R7_BANK0	○		不定
4	R8～R14, MACH, MACL, GBR	○	○	不定
5	R15	○	○	割込みハンドラスタックを指しています。 割込み発生時には、カーネルの出入口処理でスタックを割込みハンドラ用のスタックに切り替えます。すべての割込みハンドラは、同じ割込みハンドラスタックを使用します。 割込みハンドラスタックのサイズは、CFG_IRQSTKSZ に指定します。割込みはネストする可能性があるので、割込みハンドラスタックのサイズは割込みのネストを考慮として算出する必要があります。詳しくは、「C.7 割込みハンドラのスタック」を参照してください。
6	PR	○	○	不定
7	[SH-4, SH-4A] FPSCR,	○		不定
8	[SH-4, SH-4A] FPUL	○	○	不定
9	[SH-4, SH-4A] FR0_BANK0～FR15_BANK0	○	○	不定
10	[SH-4, SH-4A] FR0_BANK1～FR15_BANK1	○	○	不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　割込みハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　MD=1, FD=0 でなければなりません。また、IMASK ビットは、自割込みレベルよりも下げてはなりません。

(3)　DSP を使用する場合

DSP を使用する場合は「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」を参照してください。

(4) IRL 割込みを使用する場合

IRL 割込みは、1 ベクタに 2 つの異なるレベルの割込み要因が割当てられています。この 2 つの要因を共に使用する場合は、通常の割込みハンドラの記述方法を図 4.5のように変更する必要があります。

```
#include "kernel.h"
#define I_HILEVEL 15                      ←高いほうのレベル
void vec071_handler14(void)               ←IRL14 の処理
{
   /* IRL14 割り込み処理  */
}

void vec071_handler15(void)               ←IRL15 の処理
{
   /* IRL15 割り込み処理  */
}

void vec071(void)                         ←vec071()を通常の割込みハンドラとして定義
{
   if((get_imask()) == I_HILEVEL)
      vec071_handler15();
   else
      vec071_handler14();
}
```

図4.5 IRL 割込みハンドラの C 言語記述例

(5) 通常の割込みハンドラの定義方法

通常の割込みハンドラは、以下の方法で定義します。

- def_inh, idef_inh サービスコール
- コンフィギュレータでの定義

(6) HI7700/4, HI7750/4 での NMI に関する注意事項

NMI の再入を許容するかどうかを、以下のように制御することができます。

- NMI の再入を許容しないようにする

  NMI割込みハンドラを定義する際に指定する「ハンドラ起動時のSR」のBLビットを1にしてください。そして、NMI割込みハンドラではSR.BLをクリアしないで下さい。NMI割込みハンドラはSR.BLが1のまま実行することになるので、ハンドラで例外(TLBミスを含む)を発生させないようにしてください。
- NMI の再入を許容するようにする

  NMI割込みハンドラを定義する際に指定する「ハンドラ起動時のSR」のBLビットを0にしたり、ハンドラでSR.BLをクリアしても構いません。ただしこの場合、NMIは再入するので、割込みハンドラ用スタックサイズはNMIの再入を許容しない場合よりも多く確保する必要があります。

4.8.2 ダイレクト割込みハンドラ（HI7000/4）

HI7000/4 のダイレクト割込みハンドラは、割込みが発生するとカーネルの介入なしに CPU の処理だけで起動されます。

また、カーネル割込みマスクレベル(CFG_KNLMSKLVL)よりも高いレベル（NMI を含む）の割込みハンドラは、ダイレクト割込みハンドラとして記述・定義する必要があります。**これらのハンドラを通常の割込みハンドラとして記述・定義した場合、システムの正常な動作は保証されません。**

(1) 記述方法

ダイレクト割込みハンドラは、図 4.6に示すように割込み関数として記述します。

```
#include "kernel.h"
#define stksz 512                                                       (1)
VW  stk[stksz / sizeof(VW)];
static const VP  p_stk=(VP)&stk[stksz/sizeof(VW)];                      (2)
#pragma interrupt(DirectInh(sp=p_stk,tn=25,resbank))                    (3)
void DirectInh(void)                                                    (4)
{
   /* 割込みハンドラの処理 */
}
```

図4.6 ダイレクト割込みハンドラの C 言語記述例

■図 4.6の解説

(1) 割込みハンドラで使用するスタックを確保します。NMI以外の割込みは、必ずスタックを切り替えなければなりません。これは、割り込まれる側のスタックのオーバフローを避けるためです。同じ割込みレベルのハンドラは、スタックを共有できます。ただし、NMI割込みハンドラは専用のスタックを持つことはできません。

(2) スタックポインタの初期値をconst型として定義します。

(3) #pragma interruptにより、割込みハンドラを割込み関数として宣言します。割込み関数仕様として、以下を指定してください。

  (a) "sp="指定(スタック切り替え)
  スタックを切り替える場合は、(2)の変数を指定してください。

  (b) "tn="指定(トラップリターン指定)
  割込みマスクレベル(CFG_KNLMSKLVL)以下の割込みレベルのハンドラでは、"tn="指定が必要です。詳細は、表4.12を参照してください。

  (c) "resbank"指定(バンク割込み)
  SH-2AおよびSH2A-FPUにおいて、レジスタバンクを使用する割込みの場合は指定が必要です。詳細は、表4.12を参照してください。

(4) 割込み関数は、void型の関数として記述します。

RENESAS

表4.12　“tn=”指定と“resbank”指定 (HI7000/4)

CFG_REGBANK	レジスタバンクについて	割込みレベル	"tn="指定	"resbank"指定
NOBANK または NOTUSE	−	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	なし	なし
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	"tn=25"または "tn=26" *2	
ALL または SELECT	レジスタバンクを使用しない割込み要因 *1	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	なし	なし
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	"tn=26"	
	レジスタバンクを使用する割込み要因 *1	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	なし	"resbank"
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	"tn=25"	

【注】 *1　NMI と UBC は、常に「レジスタバンクを使用しない割込み要因」です。その他の割込み要因については、CFG_REGBANK が ALL の場合は、すべて「レジスタバンクを使用する割込み要因」です。CFG_REGBANK が SELECT の場合は､割込みレベルに対応する CFG_BANKLVL??が選択されている場合は「レジスタバンクを使用する割込み要因」、そうでない場合は「レジスタバンクを使用しない割込み要因」です。

*2　どちらでも同じ動作となります。

(2) レジスタ使用規約

表4.13にレジスタ使用規約を、図4.7にアセンブリ言語記述例を示します。

表4.13 ダイレクト割込みハンドラのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		割込みハンドラのアドレス
2	SR	○	*3	IMASK：割込みレベル。 その他のビット：割込み発生前と同じ
3	R0～R14, MACH, MACL, GBR	○	○*4	不定
4	R15	○	○	割込まれたプログラムのスタックを指しています。 [NMI以外の場合] 割込み前に実行していたプログラムのスタックオーバフローを防ぐため、必ず割込み前のスタックからハンドラ専用のスタックに切り替えてください。これを怠ると、割込みハンドラは割込み前に実行していたタスクのスタックを使うことになるため、そのタスクのスタックがオーバフローする可能性があります。 同じ割込みレベルのハンドラは同時に実行することは無いため、スタックを共有できます。この場合、最も多くスタックを使用するハンドラのサイズを確保してください。なお、割込みハンドラでは割込み前のスタックを4バイトだけ使用することが許されます。 [NMIの場合] NMI割込みハンドラは再入する可能性があるため、スタック切り換えは行わないで下さい。NMI割込みハンドラは、NMI発生時のスタックを使用することになるので、NMI割込みハンドラが使用するサイズを、タスクや割込みハンドラなどのスタックに加算する必要があります
5	PR	○	○*4	不定
6	[SH2-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	不定
7	[SH-2A, SH2A-FPU]TBR	*5	*5	*5
8	[SH2A-FPU] FPSCR, FPUL, FR0～FR15	○	○	不定

【注】 *1 使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2 ダイレクト割込みハンドラ開始関数が終了(RTEまたはTRAPA命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3 IMASKビットは、自割込みレベルよりも下げてはなりません。

*4 SH-2AまたはSH2A-FPUにおいて、レジスタバンクを使用する割込みの場合のみ不要です。

*5 CFG_TBRの設定に依存します。

(1)「カーネル管理外」：カーネルはTBRを一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。終了時には、起動時の値に戻さなければなりません。初期値は不定です。

```
    .INCLUDE "kernel.inc"
    .SECTION B_hiirqstk, DATA, ALIGN=4   ←割込みハンドラスタックを確保
    .RES.L  128
_Stk:                                   ←スタックボトムにシンボル"_Stk"を付与
    .SECTION P_ISR, CODE, ALIGN=4
    .EXPORT  DirectInh
_DirectInh:                             ←割込みハンドラの開始アドレス
    MOV.L    R0,@-R15                   ←割り込まれたスタックに R0 を退避
    MOV.L    #_Stk,R0                   ←割込みハンドラスタックにスタックポインタを退避
    MOV.L    R15,@-R0
    MOV.L    R0,R15                     ←スタックポインタを割込みハンドラスタックに変更
    MOV.L    R1,@-R15                   ←ハンドラで使用するレジスタ(DSP も含む)をスタッ
;     .....                               クに退避
; 割込みハンドラの処理
;     .....                             ←ハンドラの処理終了後、レジスタを復帰
    MOV.L    @R15+,R1
    MOV.L    @R15+,R15                  ←スタックポインタを復帰
    MOV.L    @R15+,R0                   ←R0 を復帰
    <終了命令>                           ←表 4.14を参照してください
    .POOL
    .END
```

図4.7　ダイレクト割込みハンドラのアセンブリ言語記述例

表4.14　ダイレクト割込みハンドラの終了命令(HI7000/4)

CFG_REGBANK	レジスタバンクについて *1	割込みレベル	終了命令
NOBANK または NOTUSE	—	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	RTE
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	TRAPA #D'25 または TRAPA #D'26 *2
ALL または SELECT	レジスタバンクを使用しない割込み要因	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	RTE
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	TRAPA #D'26
	レジスタバンクを使用する割込み要因	カーネル割込みマスクレベル (CFG_KNLMSKLVL)より高い	RESBANK RTE *3
		カーネル割込みマスクレベル (CFG_KNLMSKLVL)以下	TRAPA #D'25

【注】 *1　NMI と UBC は、常に「レジスタバンクを使用しない割込み要因」です。その他の割込み要因については、CFG_REGBANK が ALL の場合は、すべて「レジスタバンクを使用する割込み要因」です。CFG_REGBANK が SELECT の場合は、割込みレベルに対応する CFG_BANKLVL??が選択されている場合は「レジスタバンクを使用する割込み要因」、そうでない場合は「レジスタバンクを使用しない割込み要因」です。

*2　どちらでも同じ動作となります。

*3　この 2 つの命令をこの順に連続して記述してください。

(3)　DSP を使用する場合

DSP を使用する場合は「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」を参照してください。

(4)　ダイレクト割込みハンドラの定義方法

ダイレクト割込みハンドラは、以下の方法で定義します。

- def_inh, idef_inh サービスコール
- コンフィギュレータでの定義

(5) IRL 割込みを使用する場合

IRL 割込みは、１ベクタに２つの異なるレベルの割込み要因が割当てられています。この２つの要因を共に使用する場合は、ダイレクト割込みハンドラの記述方法を以下のように変更する必要があります。

(a) C言語で記述する場合

ダイレクト割込みハンドラの本体処理は図 4.8のように通常のＣ関数で記述し、割込み発生時にその関数を呼び出すインタフェースルーチンを図 4.9のようにアセンブラで作成してください。コンフィギュレータでダイレクト割込みハンドラを初期登録する場合は、ハンドラアドレスとしてこのインタフェースルーチンのアドレスを指定してください。

```
#include "kernel.h"
void vec071_handler14(void)              ←通常の void 型関数として記述
{
   /* IRL14 割込み処理  */
}

void vec071_handler15(void)              ←通常の void 型関数として記述
{
   /* IRL15 割込み処理  */
}
```

図4.8 IRL ダイレクト割込みハンドラの本体処理の C 言語記述例

```
    .EXPORT _Vec071                         ←インタフェースルーチンの外部定義
    .IMPORT _Vec071_Handler14               ←IRL14 の割込みハンドラ本体の外部参照
    .IMPORT _Vec071_Handler15               ←IRL15 の割込みハンドラ本体の外部参照
    .SECTION B_hiirqstk,DATA,ALIGN=4
    .RES.L   128                            ←IRL15 用割込みハンドラスタックを確保
_Stk15:                                     ←IRL15 スタックボトムにシンボル″_Stk15″を付与
    .RES.L   128                            ←IRL14 用割込みハンドラスタックを確保
_Stk14:                                     ←IRL14 スタックボトムにシンボル″_Stk14″を付与

    .SECTION P_ISR,CODE,ALIGN=4
I_HILEVEL   .equ    H'F0>>1                 ←(高レベル側の SR ビット位置)>>1
I_BITMASK   .equ    H'F0                    ←SR.i フィールドのビットマスク
_Vec071:                                    ←インタフェースルーチン
    MOV.L  R0,@-R15                         ←割り込まれたスタックに R0 を退避
    STC     SR,R0                           ←割込みレベルを判定し、切り換えるスタックと本体処
    AND     #I_BITMASK,R0                     理のアドレスを決定する
    SHLR    R0
    CMP/EQ  #I_HILEVEL,R0
    BF      IRL14
IRL15:                                      ←IRL15 の場合の処理
    MOV.L   #_Stk15,R0                      ←IRL15 用のスタックにスタックポインタを退避
    MOV.L   R15,@-R0
    MOV     R0,R15                          ←スタックポインタを IRL15 用スタックに変更
    MOV.L   #_Vec071_Handler15,R0           ←R0=IRL15 の本体処理のアドレス
CONTINUE:
    MOV.L   R1,@-R15                        ←本体処理で使用するレジスタを C 言語関数呼び出し
    MOV.L   R2,@-R15                          規約に従って退避
    MOV.L   R3,@-R15                          ハンドラで DSP を使用する場合は、DSP レジスタも
    MOV.L   R4,@-R15                          退避必要
    MOV.L   R5,@-R15
    STS.L   PR,@^R!5
    MOV.L   R6,@-R15
    STS.L   MACL,@-R15
    MOV.L   R7,@-R15
    STS.L   MACH,@-R15
```

図4.9　IRL ダイレクト割込みハンドラインタフェースルーチンの記述例

RENESAS

```
    JSR     @R0                     ←本体処理をコール
    NOP
    LDS.L   @R15+,MACH              ←レジスタを復帰
    MOV.L   @R15+,R7
    LDS.L   @R15+,MACH
    MOV.L   @R15+,R6
    LDS.L   @R15+,PR
    MOV.L   @R15+,R5
    MOV.L   @R15+,R4
    MOV.L   @R15+,R3
    MOV.L   @R15+,R2
    MOV.L   @R15+,R1
    MOV.L   @R15+,R15               ←スタックポインタを復帰
    MOV.L   @R15+,R0                ←R0 を復帰

    TRAPA   #D'25                   ←カーネル割込みマスクレベル以下の割込みハンドラは
                                      TRAPA #25 で終了
;    RTE                            ←カーネル割込みマスクレベルより高い割込みハンドラは
;    NOP                              RTE で終了

IRL14:                              ←IRL14 の場合の処理
    MOV.L   #_Stk14,R0              ←IRL14 用のスタックにスタックポインタを退避
    MOV.L   R15,@-R0
    MOV     R0,R15                  ←スタックポインタを IRL14 用スタックに変更
    BRA     CONTINUE
    MOV.L   #_Vec071_Handler14,R0   ←R0=IRL14 の本体
    .POOL
    .END
```

図 4.9 IRL ダイレクト割込みハンドラインタフェースルーチンの記述例(続き)

(b) アセンブリ言語で記述する場合

(a)のインタフェースルーチンのリストを参考に、割込みレベルに応じたスタックに切り替えた後に必要な処理を行い、終了時に元のスタックに戻してください。

4.9 CPU 例外ハンドラ(TRAPA 例外を含む)

例外が発生するとカーネルの出入口処理ルーチンを介して CPU 例外ハンドラが起動されます。

(1) 記述方法

CPU 例外ハンドラ（TRAPA 命令例外含む）の記述方法は、通常の割込みハンドラと全く同じです。「4.8.1 通常の割込みハンドラ」を参照してください。

(2) レジスタ使用規約

表 4.15, 表 4.16, 表 4.17に、それぞれ HI7000/4, HI7700/4, HI7750/4 における CPU 例外ハンドラのレジスタ使用規約を示します。

表4.15 CPU 例外ハンドラのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		CPU 例外ハンドラのアドレス
2	SR	○		CPU 例外発生前と同じ
3	R0〜R7	○		不定
4	R8〜R14, MACH, MACL, GBR	○	○	CPU 例外発生時と同じ
5	R15	○	○	例外発生元のプログラムのスタックを指しています。CPU 例外ハンドラは再入の可能性があるため、例外を発生させたプログラムのスタックを使用して動作します。CPU 例外ハンドラ専用のスタックを持つことはできません
6	PR	○	○	不定
7	[SH2-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	CPU 例外発生時と同じ
8	[SH-2A, SH2A-FPU]TBR	*3	*3	*3
9	[SH2A-FPU] FPSCR,	○	○	CPU 例外発生時と同じ
10	[SH2A-FPU] FR0〜FR11	○	○	CPU 例外発生時と同じ
11	[SH2A-FPU] FPUL, FR12〜FR15	○	○	CPU 例外発生時と同じ

【注】 *1 使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2 CPU 例外ハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3 CFG_TBR の設定に依存します。

(1)「カーネル管理外」：カーネルは TBR を一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。終了時には、起動時の値に戻さなければなりません。初期値は CPU 例外発生時と同じです。

RENESAS

表4.16　CPU 例外ハンドラのレジスタ使用規約(HI7700/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		CPU 例外ハンドラのアドレス
2	SR	○	*3	IMASK ビット：CPU 例外発生前と同じ その他のビット：ハンドラ定義時に指定した値
3	R0_BANK0～R7_BANK0	○		不定
4	R8～R14, MACH, MACL, GBR	○	○	CPU 例外発生時と同じ
5	R15	○	○	例外発生元のプログラムのスタックを指しています。CPU 例外ハンドラは再入の可能性があるため、例外を発生させたプログラムのスタックを使用して動作します。CPU 例外ハンドラ専用のスタックを持つことはできません
6	PR	○	○	不定
7	[SH3-DSP, SH4AL-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	CPU 例外発生時と同じ

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　CPU 例外ハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は、DSP/CL=1 でなければなりません。また、MD=1 でなければなりません。

表4.17　CPU 例外ハンドラのレジスタ使用規約(HI7750/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		CPU 例外ハンドラのアドレス
2	SR	○	*3	IMASK ビット：CPU 例外発生前と同じ その他のビット：ハンドラ定義時に指定した値
3	R0_BANK0～R7_BANK0	○		不定
4	R8～R14, MACH, MACL, GBR	○	○	CPU 例外発生時と同じ
5	R15	○	○	例外発生元のプログラムのスタックを指しています。CPU 例外ハンドラは再入の可能性があるため、例外を発生させたプログラムのスタックを使用して動作します。CPU 例外ハンドラ専用のスタックを持つことはできません
6	PR	○	○	不定
7	[SH-4, SH-4A] FPSCR,	○		不定
8	[SH-4, SH-4A] FPUL	○	○	CPU 例外発生時と同じ
9	[SH-4, SH-4A] FR0_BANK0～FR15_BANK0	○	○	CPU 例外発生時と同じ
10	[SH-4, SH-4A] FR0_BANK1～FR15_BANK1	○	○	CPU 例外発生時と同じ

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　割込みハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　MD=1, FD=0 でなければなりません。

(3) CPU 例外ハンドラ起動時のスタックの状態

CPU 例外が発生すると、カーネルはスタックに各種レジスタを保存した後、CPU 例外ハンドラを起動します。CPU 例外ハンドラ終了時には、カーネルはこれらのレジスタをスタックから復帰します。

(a) HI7000/4

	スタック内容	オフセット
スタックポインタ(R15)→	例外発生時の R7	0
	例外発生時の R6	+4
	例外発生時の R5	+8
	例外発生時の R4	+12
	例外発生時の R3	+16
	例外発生時の R2	+20
	例外発生時の R1	+24
	例外発生時の R0	+28
	例外発生時の PR	+32
	例外発生時の PC	+36
	例外発生時の SR	+40
CPU 例外発生前のスタックポインタ→	例外発生前のスタック	+44

(b) HI7700/4

	スタック内容	オフセット
スタックポインタ(R15)→	例外発生時の R7_BANK0	0
	例外発生時の R6_BANK0	+4
	例外発生時の R5_BANK0	+8
	例外発生時の R4_BANK0	+12
	例外発生時の R3_BANK0	+16
	例外発生時の R2_BANK0	+20
	例外発生時の R1_BANK0	+24
	例外発生時の R0_BANK0	+28
	例外発生時の PR	+32
	例外発生時の PC(SPC)	+36
	例外発生時の SR(SSR)	+40
CPU 例外発生前のスタックポインタ→	例外発生前のスタック	+44

(c) HI7750/4

スタックポインタ(R15)→	例外発生時の R7_BANK0	0
	例外発生時の R6_BANK0	+4
	例外発生時の R5_BANK0	+8
	例外発生時の R4_BANK0	+12
	例外発生時の R3_BANK0	+16
	例外発生時の R2_BANK0	+20
	例外発生時の R1_BANK0	+24
	例外発生時の R0_BANK0	+28
	例外発生時の PR	+32
	例外発生時の FPSCR	+36
	例外発生時の PC(SPC)	+40
	例外発生時の SR(SSR)	+44
CPU 例外発生前のスタックポインタ→	例外発生前のスタック	+48

(4) DSP を使用する場合

DSP を使用する場合は「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」を参照してください。

(5) CPU 例外ハンドラの定義方法

CPU 例外ハンドラは、以下の方法で定義します。

- def_exc, idef_exc サービスコール（TRAPA 以外の CPU 例外ハンドラ）
- vdef_trp, ivdef_trp サービスコール（TRAPA の CPU 例外ハンドラ）
- コンフィギュレータでの定義

4.10 タイムイベントハンドラ、初期化ルーチン

(1) 記述方法

タイムイベントハンドラ、初期化ルーチンは、通常の C 言語関数として記述します。図 4.10に周期ハンドラ、アラームハンドラ、初期化ルーチン、図 4.11にオーバーランハンドラの C 言語記述例を示します。これらのハンドラは非タスクコンテキストで動作します。

```
#include "kernel.h"
void Handler(VP_INT exinf)    ←パラメータとして生成時に指定した exinf が渡されます。
{
    /* ハンドラの処理 */
}
```

図4.10 周期ハンドラ、アラームハンドラ、初期化ルーチンの C 言語記述例

```
#include "kernel.h"
void Ovrhdr(ID tskid, VP_INT exinf)   ←パラメータとして起動の原因となった tskid と、そのタ
{                                      スクの exinf が渡されます。
    /* ハンドラの処理 */
}
```

図4.11 オーバーランハンドラの C 言語記述例

(2) レジスタ使用規約

表 4.18, 表 4.19, 表 4.20に、それぞれ HI7000/4, HI7700/4, HI7750/4 におけるタイムイベントハンドラ、初期化ルーチンのレジスタ使用規約を示します。

表4.18　タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7000/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		ハンドラ・ルーチンのアドレス
2	SR	○	*3	(1)IMASK [タイムイベントハンドラ] タイマ割込みレベル (CFG_TIMINTLVL) [初期化ルーチン] カーネル割込みマスクレベル (CFG_KNLMSKLVL) (2)その他のビット：不定
3	R0〜R3	○		不定
4	R4	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] ハンドラ・ルーチンの拡張情報 [オーバーランハンドラ] 対象のタスク ID
5	R5	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] 不定 [オーバーランハンドラ] 対象タスクの拡張情報
6	R6〜R7	○		不定
7	R8〜R14, MACH, MACL, GBR	○	○	不定
8	R15	○	○	適切なスタックを指しています。
9	PR	○	○	不定
10	[SH2-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	不定
11	[SH-2A, SH2A-FPU]TBR	*4	*4	*4
12	[SH2A-FPU] FPSCR,	○	○	不定
13	[SH2A-FPU] FR0〜FR11	○	○	不定
14	[SH2A-FPU] FPUL, FR12〜FR15	○	○	不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　ハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　IMASK ビットは起動時よりも下げてはならず、終了時には起動時の値に戻さなければなりません。

*4　CFG_TBR の設定に依存します。

(1)「カーネル管理外」：カーネルは TBR を一切操作しません。

(2)「サービスコール呼び出し専用に使用」の場合：変更禁止です。

(3)「タスクコンテキストと扱う」の場合：使用可能です。終了時には、起動時の値に戻さなければなりません。初期値は不定です。

表4.19　タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7700/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		ハンドラ・ルーチンのアドレス
2	SR	○	*3	(1)MD=1, BL=0, RB=0 (2)IMASK [タイムイベントハンドラ] タイマ割込みレベル (CFG_TIMINTNO) [初期化ルーチン] カーネル割込みマスクレベル (CFG_KNLMSKLVL) (3)DSP/CL CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は 1、そうでない場合は 0 (4)その他のビット：不定
3	R0～R3	○		不定
4	R4	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] ハンドラ・ルーチンの拡張情報 [オーバーランハンドラ] 対象のタスク ID
5	R5	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] 不定 [オーバーランハンドラ] 対象タスクの拡張情報
6	R6～R7	○		不定
7	R8～R14, MACH, MACL, GBR	○	○	不定
8	R15	○	○	適切なスタックを指しています。
9	PR	○	○	不定
10	[SH3-DSP, SH4AL-DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○	○	不定

【注】 *1　使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2　ハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3　CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合は、DSP/CL=1 でなければなりません。また、MD=1, BL=0, RB=0 でなければなりません。また、IMASK ビットは起動時よりも下げてはならず、終了時には起動時の値に戻さなければなりません。

表4.20 タイムイベントハンドラ、初期化ルーチンのレジスタ使用規約(HI7750/4)

No	レジスタ	使用可能 *1	終了条件 *2	初期値
1	PC	○		ハンドラ・ルーチンのアドレス
2	SR	○	*3	(1)MD=1, BL=0, RB=0, FD=0 (2)IMASK [タイムイベントハンドラ] タイマ割込みレベル (CFG_TIMINTNO) [初期化ルーチン] カーネル割込みマスクレベル (CFG_KNLMSKLVL) (3)その他のビット：不定
3	R0～R3	○		不定
4	R4	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] ハンドラ・ルーチンの拡張情報 [オーバーランハンドラ] 対象のタスク ID
5	R5	○		[周期ﾊﾝﾄﾞﾗ,ｱﾗｰﾑﾊﾝﾄﾞﾗ,初期化ﾙｰﾁﾝ] 不定 [オーバーランハンドラ] 対象タスクの拡張情報
6	R6～R7	○		不定
7	R8～R14, MACH, MACL, GBR	○	○	不定
8	R15	○	○	適切なスタックを指しています。
9	PR	○	○	不定
10	[SH-4, SH-4A] FPSCR,	○		不定
11	[SH-4, SH-4A] FPUL	○	○	不定
12	[SH-4, SH-4A] FR0_BANK0～FR15_BANK0	○	○	不定
13	[SH-4, SH-4A] FR0_BANK1～FR15_BANK1	○	○	不定

【注】 *1 使用可能なレジスタです。なお、サービスコールの前後では一部のレジスタは保証されません。詳細は、「3.2.3 サービスコール呼び出し前後のレジスタ保証規則」を参照してください。

*2 ハンドラ開始関数からリターン(RTS 命令)する時点では、これらのレジスタは起動時の状態でなければなりません。

*3 MD=1, BL=0, RB=0, FD=0 でなければなりません。また、IMASK ビットは起動時よりも下げてはならず、終了時には起動時の値に戻さなければなりません。

(3) DSP を使用する場合

DSP を使用する場合は「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」を参照してください。

(4) 定義方法

(a) 周期ハンドラ

- cre_cyc, icre_cyc, acre_cyc, iacre_cyc サービスコール
- コンフィギュレータでの初期登録

(b) アラームハンドラ

- cre_alm, icre_alm, acre_alm, iacre_alm サービスコール
- コンフィギュレータでの初期登録

(c) オーバーランハンドラ

- def_ovr サービスコール
- コンフィギュレータでの初期登録

(d) 初期化ルーチン

- コンフィギュレータでの初期登録

4.11 CPU 初期化ルーチン

4.11.1 CPU 初期化ルーチンの作成

CPU 初期化ルーチンは、CPU リセットによって最初に実行されるプログラムです。「2.18.1 リセットとカーネルの起動」も参照してください。また、サンプル提供の ***nnnn***_cpuasm.src(アセンブラ記述)と ***nnnn***_cpuini.c(C 言語記述)の内容も参考にしてください。

4.11.2 HI7000/4 の CPU 初期化ルーチンの登録

CPU 初期化ルーチンは、開始アドレスをリセットベクタ（0 番地）に登録し、初期スタックポインタをリセットベクタに登録する必要があります。また、CPU 初期化ルーチンの最後では、通常はカーネルを起動します。

具体的には、リセットベクタは以下のいずれかの方法で作成してください。

(1) リセットベクタをユーザ自身で作成

リセットベクタは、図 4.12のように作成してください。

```
_INIT_SP   .EQU  H'FFFA0000              ←リセット時の初期スタックポインタを定義します。
                                           通常は、内蔵 RAM の最終アドレスとしてください。
    .SECTION C_ResVec, DATA, LOCATE=0    ←リセットベクタのセクション名を C_ResVec とし、
    .IMPORT   _hi_cpuasm                   配置アドレスを 0 番地とします。
_ResVec:
    .DATA.L  _hi_cpuasm, _INIT_SP
    .DATA.L  _hi_cpuasm, _INIT_SP        ←ベクタ番号 0(パワーオンリセット)、2（マニュアル
    .END                                   リセット）のプログラムアドレスを_hi_cpuasm、ス
                                           タックポインタを_INIT_SP とします。
```

図4.12 リセットベクタの作成例(HI7000/4)

(2) コンフィギュレータで設定

コンフィギュレータで、ベクタ番号 0、1 に対してそれぞれ CPU 初期化ルーチンのアドレスとリセット時のスタックポインタを割込みハンドラのアドレスとして定義してください。そして、コンフィギュレータが生成したベクタテーブルのセクション C_hivct を、リンク時に 0 番地に配置してください。

4.11.3 HI7700/4, HI7750/4 での CPU 初期化ルーチンの登録

CPU 初期化ルーチンは、リンク時にリセットベクタアドレス H'a0000000 番地に配置する必要があります。

4.12 システムダウンルーチン

システムダウンルーチンは、以下の C 言語関数として作成します。この名前は固定です。

```
void    _kernel_sysdwn(W type, ER ercd, VW inf1, VW inf2)
```

システムダウンルーチンに渡されるパラメータの意味は、「B.2 システムダウン時の情報」を参照してください。

システムダウンルーチンは、必ず作成してカーネルと結合しなければなりません。

C 言語によるシステムダウンルーチンの記述例は、サンプル提供の ***nnnn***_sysdwn.c を参照してください。

システムダウンルーチンでは異常内容に応じた処理を行なうことができますが、カーネル内部でシステムダウンとなった場合（エラー種別が負）は、サービスコールなどカーネルの機能は使用できません。また、システムダウンルーチンからはリターンしないでください。

デバッグ時には、システムダウンではその時の状態を保持して無限ループさせ、システムダウン要因の解析、対策を行なってください。

RENESAS

4.13 DSP を使用する場合(HI7000/4, HI7700/4)

4.13.1 DSR レジスタの初期化について

各プログラムでの DSR レジスタの初期値は、表 4.21に示すようになっています。

表4.21 DSR レジスタの初期値

プログラム	DSR レジスタの初期値
タスク(TA_COP0 属性なし)	不定
タスク(TA_COP0 属性あり)	0
タスク例外処理ルーチン(TA_COP0 属性なし)	不定
タスク例外処理ルーチン(TA_COP0 属性あり)	0
割込みハンドラ	不定
CPU 例外ハンドラ	不定
タイムイベントハンドラ	不定
初期化ルーチン	不定

DSP 演算を行う場合は、事前に DSR レジスタを適切に初期化してください。どのような値に初期化すべきかは、使用するマイコンのハードウェアマニュアルを参照してください。

なお、DSR レジスタの初期化は、上記のプログラム毎に必要で、DSP 演算を行う前に一度だけ実施すれば良いです。

以下に、タスクで DSR レジスタを初期化する例を示します。

```
#include "kernel.h"
#pragma inline_asm(SetDSR)  // DSR レジスタを設定するインラインアセンブラ関数
static void SetDSR(UW dsr)
{
  lds  r4,dsr
}

void task(VP_INT exinf)
{
  SetDSR(0);  // DSR レジスタの初期化

 // DSP 演算
}
```

4.13.2 ハンドラ等での DSP の使用

以下のプログラムで DSP を使用する場合は、図 4.13のように DSP レジスタを退避、復帰する処理が必要です。また、コンパイル時にはオブジェクト形式指定オプション code =asmcode を用いてコンパイルする必要があります。

- 通常の割込みハンドラ
- ダイレクト割込みハンドラ(HI7000/4)
- CPU 例外ハンドラ
- タイムイベントハンドラ
- 初期化ルーチン
- タイマ割込みルーチン

HI7700/4 で DSP スタンバイ制御機能を使用する場合は、「F.3 各プログラム起動時のモジュールスタンバイ状態」も参照してください。

```
#include "kernel.h"
#include “shdsp.h”               ←ヘッダファイル”shdsp.h”をインクルードします。
#pragma inline_asm(SetDSR)       ←DSR レジスタを設定するインラインアセンブラ関数
static void SetDSR(UW dsr)
{
  lds  r4,dsr
}
void HadlerMain(VP_INT exinf)   ←ハンドラの本体処理
{
    /* ハンドラの処理 */
}

void Handler(VP_INT exinf)       ←ハンドラの開始関数では、図に書いた処理のみを記述し、
{                                  ハンドラの本来の処理は、HandlerMain( )に記述します。
    T_DSP area;                  ←DSP レジスタを保存する領域を確保します。
    IniDSP(&area);               ←DSP レジスタを保存します。
    SetDSR(0);                   ←DSR レジスタの初期化
    HandlerMain(exinf);          ←本体処理を行う HandlerMain( )をコールします。
    EndDSP(&area);               ←DSP レジスタを復帰します。
}
```

図4.13 DSP を使用するハンドラ等の C 言語記述例

5. コンフィギュレーション

5.1 前知識

システムを作成するには、あらかじめ本節に記載の内容を十分に理解してください。

また、実際のコンフィギュレーション作業には、以下のツールを使用します。

- 本製品添付のコンフィギュレータ
- HEW

HEW については、HEW のマニュアルやオンラインヘルプなどを用いて、あらかじめ操作方法を習得しておいてください。また、コンフィギュレータの操作方法については、「5.4 コンフィギュレータ」およびオンラインヘルプを参照してください。

5.1.1 一括リンクと分割リンク

システムのロードモジュールファイルを生成するには、以下の作業が必要になります。

- アプリケーションを作成する
- コンフィギュレータを用いて、コンフィギュレーションファイルを生成する
- HEW を用いてビルドを行い、ロードモジュールを生成する

これら 3 つの作業は、ロードモジュールをどのように生成するかによって若干異なってきます。このため、まずロードモジュールの生成方法を理解・決定しなければなりません。

ロードモジュールの生成方法は、「一括リンク方式」と「分割リンク方式」の 2 種類があります。

(1) 一括リンク方式

一括リンク方式はカーネルとすべてのコンフィギュレーションファイルを 1 つのロードモジュール（これを「一括ロードモジュール」と呼びます）にする方式です。アプリケーションは、一括ロードモジュールに含めることも、別ロードモジュール（これを「アプリケーションロードモジュール」と呼びます）にすることもできます。

一括リンク方式によるロードモジュールの生成を図 5.1に示します。

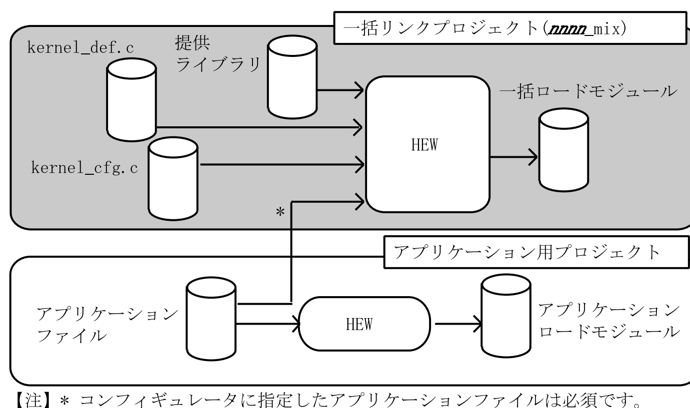


図5.1 一括リンク方式によるロードモジュールの生成

(2) 分割リンク方式

分割リンク方式は、カーネルのコード部分とデータ部分を別々のロードモジュールにする方式です。

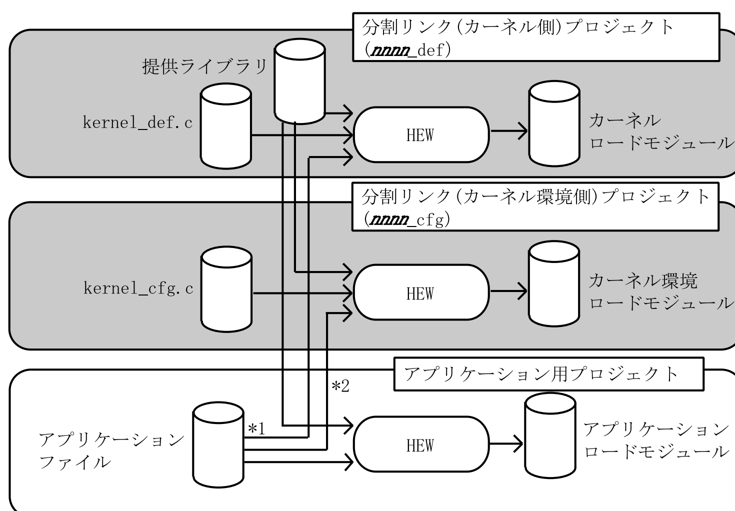
前者は「カーネルロードモジュール」と呼ばれ、カーネルライブラリとコンフィギュレータ出力ファイルの一部(kernel_def.c)をリンクすることで生成します。また、カーネルロードモジュールのリンク単位を「カーネル側」と呼びます。

一方、後者は「カーネル環境ロードモジュール」と呼ばれ、コンフィギュレータ出力ファイルの一部(kernel_cfg.c)をリンクすることで生成します。また、カーネル環境ロードモジュールのリンク単位を「カーネル環境側」と呼びます・

分割リンク方式では、カーネルロードモジュールを ROM 化した後にもカーネルロードモジュールを更新せずに、タスクの最大数(CFG_MAXTSKID)などといったコンフィギュレータパラメータを変更してカーネル環境ロードモジュールを再生成することができます。

アプリケーションは、カーネルロードモジュール、カーネル環境ロードモジュールに含めることも、別のアプリケーションロードモジュールにすることもできます。

分割リンク方式によるロードモジュールの生成を図 5.2に示します。



【注】*1 コンフィギュレータでカーネル側に指定したアプリケーションファイルは必須です。
*2 コンフィギュレータでカーネル環境側に指定したアプリケーションファイルは必須です。

図5.2 分割リンク方式によるロードモジュールの生成

RENESAS

5.2 フォルダ構成

カーネルは、インストーラで指定したフォルダの下の kernel フォルダに、図 5.3に示す構成でインストールされます。

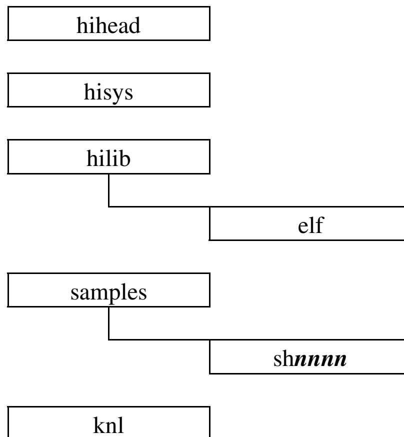


図5.3 kernel フォルダ以下の構成

5.2.1 hihead フォルダ

itron.h, kernel.h などアプリケーションで使用するヘッダファイルが格納されています。

5.2.2 hisys フォルダ

コンフィギュレータ出力ファイルをコンパイルするときに使用するシステム定義ファイルが格納されています。格納されているファイルを変更してはなりません。

5.2.3 hilib フォルダ

カーネルライブラリが格納されたフォルダを含みます。

hilib¥elf フォルダには、ELF フォーマットのカーネルライブラリが格納されています。このライブラリの生成に使用したコンパイラバージョンについては、製品添付のリリースノートを参照してください。

5.2.4 knl フォルダ

このフォルダは、カーネルソースコード付きの契約時のみ提供されます。詳細は、製品添付のリリースノートを参照してください。

5.2.5 samples¥sh*nnnn* フォルダ

nnnn に対応する各種マイコン用のサンプルが格納されたフォルダです。ロードモジュールを生成する HEW ワークスペースファイルも含まれます。

表 5.1～表 5.3に、それぞれ HI7000/4 V.2.02、HI7700/4 V.2.02、HI7750/4 V.2.02 における ***nnnn*** の対応を示します。最新の情報は製品添付のリリースノートを参照してください。

表5.1　HI7000/4 V.2.02 における *nnnn* の対応

nnnn	CPU コア	対象マイコン	作成時に使用したHEW(コンパイラパッケージ)バージョン
7011	SH-2	SH7011, SH7018	1.2 (6.0C)
703x	SH-1	SH7020, SH7021, SH7032, SH7034	1.2 (6.0C)
704x	SH-2	SH7040, SH7041, SH7042, SH7043, SH7044, SH7045, SH7014, SH7016, SH7017	1.2 (6.0C)
7046	SH-2	SH7046, SH7047, SH7048, SH7049, SH7144, SH7145, SH7148	1.2 (6.0C)
7050	SH-2	SH7050, SH7051	1.2 (6.0C)
7052	SH-2	SH7052, SH7053, SH7054	1.2 (6.0C)
7065	SH2-DSP	SH7065	1.2 (6.0C)
7604	SH-2	SH7604	1.2 (6.0C)
7615	SH2-DSP	SH7615, SH7616	1.2 (6.0C)
7618	SH-2	SH7618	4.00.02 (9.00 Release 03)
72060	SH-2A	SH72060	4.00.02 (9.00 Release 03)

表5.2　HI7700/4 V.2.02 における *nnnn* の対応

nnnn	CPU コア	対象マイコン	作成時に使用したHEW(コンパイラパッケージ)バージョン
7707	SH-3	SH7707	1.2 (6.0C)
7708	SH-3	SH7708, SH7708R, SH7708S	1.2 (6.0C)
7709	SH-3	SH7709	1.2 (6.0C)
7709a	SH-3	SH7709A, SH7709S, SH7706	1.2 (6.0C)
7729	SH3-DSP	SH7729, SH7729R, SH7727	1.2 (6.0C)
7290	SH3-DSP	SH7290, SH7294, SH7300	3.0.01 (8.00 Release 00)
7641	SH3-DSP	SH7641	3.0.01 (8.00 Release 00)
7318	SH4AL-DSP(拡張機能なし)	SH7318	3.0.01 (8.00 Release 00)
7343	SH4AL-DSP(拡張機能あり)	SH7343	4.00.02 (9.00 Release 03)

表5.3　HI7750/4 V.2.02 における *nnnn* の対応

nnnn	CPU コア	対象マイコン	作成時に使用したHEW(コンパイラパッケージ)バージョン
7750	SH-4	SH7750, SH7750S, SH7750R	1.2 (6.0C)
7751	SH-4	SH7751, SH7751R	1.2 (6.0C)
7760	SH-4	SH7760	3.0.01 (8.00 Release 00)
7770	SH-4A(拡張機能なし)	SH7770	3.0.01 (8.00 Release 00)
7785	SH-4A(拡張機能あり)	SH7785	4.00.02 (9.00 Release 03)

sh*nnnn* フォルダ以下の構成は、対応する HEW バージョンによって 2 種類あります。

(1) HEW3 以上用

sh*nnnn* フォルダ以下は、図 5.4に示す構成になっています。

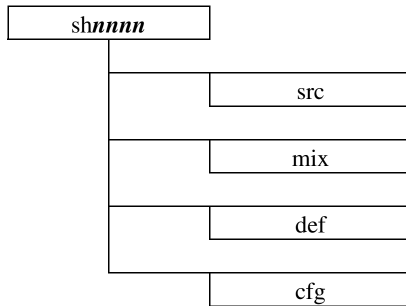


図5.4 sh*nnnn* フォルダ以下の構成(HEW3 以上用)

(a) サンプル HEW ワークスペースファイル(sh*nnnn*.hws)

sh*nnnn* フォルダには、このワークスペースファイルがあります。このワークスペースには、以下の３つのプロジェクトが登録されています。

- 一括リンク用(mix¥mix.hwp)
- 分割リンク・カーネル側用(def¥def.hwp)
- 分割リンク・カーネル環境側用(cfg¥cfg.hwp)

(b) sh*nnnn*¥src フォルダ

以下のソースプログラムが格納されています。

- サンプルタスク(task.c)
- サンプルタイマドライバ(*nnnn*_tmrdrv.c, *nnnn*_tmrdrv.h, *nnnn*_tmrdef.h)
- サンプル CPU 初期化ルーチン(*nnnn*_cpuasm.src, *nnnn*_cpuini.c)
- サンプルシステムダウンルーチン(*nnnn*_sysdwn.c)
- サンプルコンフィギュレータ設定ファイル(*nnnn*.hcf)とその出力ファイル(表 5.4参照)
- サンプルセクション初期化処理(*nnnn*_sct.src, *nnnn*_inisct.c : 出荷時の構成では使用していません)
- HI7000/4 のみ：割込み・CPU 例外処理(kernel_exp.src)
- HI7700/4、HI7750/4 のみ：割込み・CPU 例外入口処理(*nnnn*_expent.src)
- HI7700/4、HI7750/4 のみ：未定義割込み・CPU 例外処理(*nnnn*_intdwn.src)
- kernel_def.c、kernel_cfg.c
- HI7700/4 のみ：DSP スタンバイ制御機能設定ファイル(kernel_def_dspstby_set.h)
- HI7700/4 のみ：最適化タイマ機能設定ファイル(kernel_def_opttmr_set.h)
- HI7700/4, HI7750/4 の SH4AL-DSP, SH-4A 用のみ：キャッシュサポートサービスコール設定ファイル(kernel_cfg_cac_set.h)

(c) sh*nnnn*¥mix フォルダ

一括リンク用の HEW プロジェクトファイル mix.hwp が格納されています。
この下に、mix.hwp で生成するオブジェクトファイルを格納するためのフォルダがあります。

(d) sh*nnnn*¥def フォルダ

分割リンクのカーネル側の HEW プロジェクトファイル def.hwp が格納されています。
この下に、def.hwp で生成するオブジェクトファイルを格納するためのフォルダがあります。

(e) sh*nnnn*¥cfg フォルダ

分割リンクのカーネル環境側の HEW プロジェクトファイル cfg.hwp が格納されています。
この下に、cfg.hwp で生成するオブジェクトファイルを格納するためのフォルダがあります。

(2) HEW1.2 用

sh***nnnn*** フォルダ以下は、図 5.5に示す構成になっています。

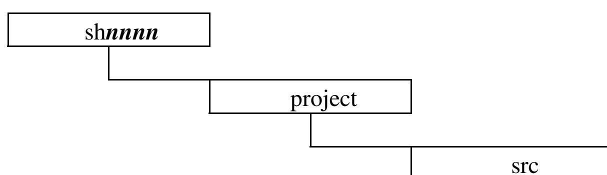


図5.5 sh*nnnn* フォルダ以下の構成(HEW1.2 用)

(a) サンプル HEW ワークスペースファイル(sh*nnnn*.hws)

sh***nnnn*** フォルダには、このワークスペースファイルがあります。このワークスペースには、以下の 3 つのプロジェクトが登録されています。

- 一括リンク用(project¥mix.hwp)
- 分割リンク・カーネル側用(project¥def.hwp)
- 分割リンク・カーネル環境側用(project¥cfg.hwp)

(b) sh*nnnn*¥project フォルダ

このフォルダには、sh***nnnn***.hws に登録されている 3 つのプロジェクトファイルが格納されています。また、この下にそれぞれのプロジェクトで生成するオブジェクトファイルを格納するためのフォルダがあります。

(c) sh*nnnn*¥project¥src フォルダ

以下のソースプログラムが格納されています。

- サンプルタスク(task.c)
- サンプルタイマドライバ(***nnnn***_tmrdrv.c, ***nnnn***_tmrdrv.h, ***nnnn***_tmrdef.h)
- サンプル CPU 初期化ルーチン(***nnnn***_cpuasm.src, ***nnnn***_cpuini.c)
- サンプルシステムダウンルーチン(***nnnn***_sysdwn.c)
- サンプルコンフィギュレータ設定ファイル(***nnnn***.hcf)とその出力ファイル(表 5.4参照)
- サンプルセクション初期化処理(***nnnn***_sct.src, ***nnnn***_inisct.c : 出荷時の構成では使用していません)
- HI7000/4 のみ：割込み・CPU 例外処理(kernel_exp.src)
- HI7700/4、HI7750/4 のみ：割込み・CPU 例外入口処理(***nnnn***_expent.src)
- HI7700/4、HI7750/4 のみ：未定義割込み・CPU 例外処理(***nnnn***_intdwn.src)
- kernel_def.c、kernel_cfg.c
- HI7700/4 のみ：DSP スタンバイ制御機能設定ファイル(kernel_def_dspstby_set.h)
- HI7700/4 のみ：最適化タイマ機能設定ファイル(kernel_def_opttmr_set.h)

5.3 作業手順

通常は、以下の手順で作業します。

- ***nnnn***.hcf をダブルクリックしてコンフィギュレータを起動します。
- コンフィギュレータで必要な設定を行います。
- ***nnnn***.hcf を保存し、さらに構築ファイルを生成します。生成フォルダは kernel_def.c, kernel_cfg.c が存在するフォルダとしてください。
- コンフィギュレータを終了します。
- サンプルの HEW ワークスペースファイル(sh***nnnn***.hws)をダブルクリックして HEW を起動します。そして、使用するプロジェクトを選択します。
- HEW にアプリケーションファイルの追加や、C コンパイラ、リンカなどのオプション設定を行い、ビルドを実行します。

5.4 コンフィギュレータ

本節では、コンフィギュレータの基本的な操作および設定項目について解説します。操作方法の詳細は、コンフィギュレータのオンラインヘルプを参照してください。

5.4.1 概要

コンフィギュレータは、カーネルの動作パラメータを設定するためのツールです。コンフィギュレータは、設定された内容にしたがっていくつかの C ソースファイルを生成します。生成されたファイルおよびアプリケーションをビルド（コンパイル・リンク）することで、システム（ロードモジュール）を作成します。

システム構築におけるコンフィギュレータの位置付けは、図 5.6のようになります。

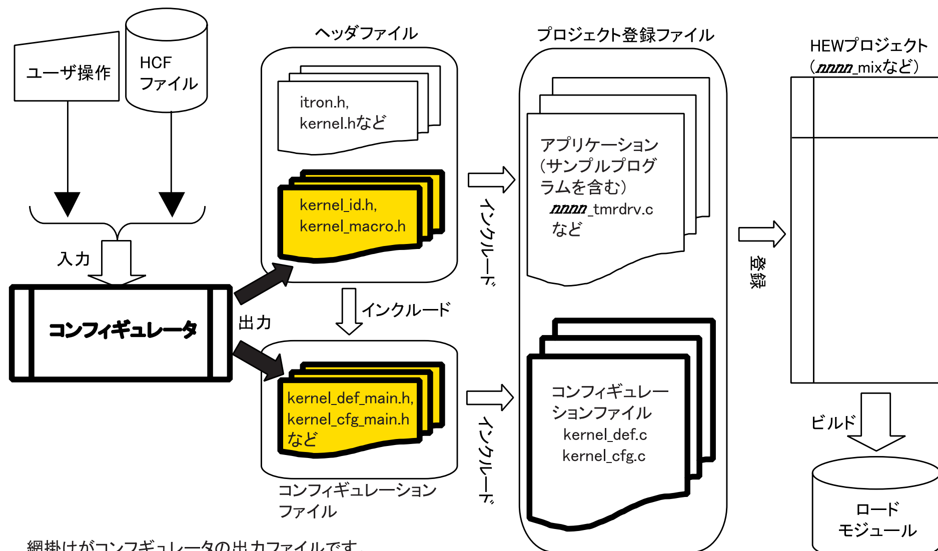


図 5.6 システム構築におけるコンフィギュレータの位置付け

5.4.2 コンフィギュレータの構成

図 5.7に、コンフィギュレータの概観を示します。

コンフィギュレータのウィンドウは、構築情報入力パート一覧ウィンドウ（左側）と構築情報入力ウィンドウ（右側）で構成されています。構築情報入力ウィンドウにより各種情報を入力後、「構築ファイルの生成」コマンド（メニューおよびツールボタン）により、構築ファイルを生成します。

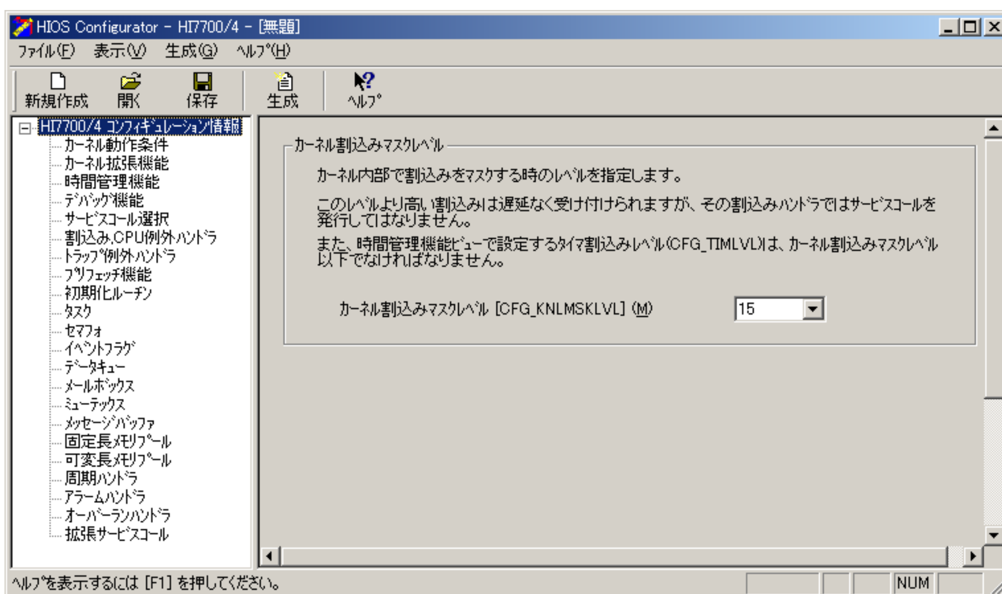


図5.7 コンフィギュレータ概観

5.4.3 ファイル操作

(1) コンフィギュレータ設定ファイル(HCF ファイル)

コンフィギュレータに設定されている情報は、HCF ファイルとして保存できます。

(2) コンフィギュレーションファイルの生成

[生成->構築ファイル生成]を選択するか、またはツールボタンの[生成]を押すと、図 5.8に示すダイアログが開きますので、ファイルを生成するフォルダを指定してください。

なお、コンフィギュレーションファイルは、kernel_def.c, kernel_cfg.c があるフォルダに生成する必要があります。

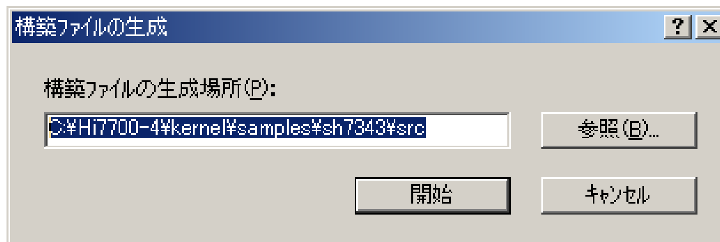


図5.8 フォルダ選択ダイアログ

RENESAS

これにより、表5.4に示すファイルが指定したフォルダに生成されます。このとき、同名のファイルが存在しても無条件に上書きされますので、注意してください。

5.4.4　コンフィギュレーションファイル

コンフィギュレータが出力するコンフィギュレーションファイルは、表5.4の通りです。

表5.4　コンフィギュレーションファイル

項番	分類	ファイル名	リンク単位
1	ヘッダファイル	kernel_id.h	カーネル環境側
2		kernel_id_sys.h	カーネル側
3		kernel_macro.h	カーネル側
4	システム定義ファイル	kernel_def_main.h	カーネル側
5		kernel_def_inidata.def	カーネル側
6		kernel_def_vct.inc	カーネル側(HI7000/4 のみ)
7		kernel_cfg_main.h	カーネル環境側
8		kernel_cfg_inidata.def	カーネル環境側

「カーネル側」と「カーネル環境側」は、以下の項目に関連します。

- ID 番号の自動割当について ... 「(1) kernel_id.h, kernel_id_sys.h」参照
- 分割リンクについて ... 「5.4.5 分割リンク」参照

以下に、各生成ファイルについて説明します。

(1) kernel_id.h, kernel_id_sys.h

タスクなど、ID 番号で識別されるオブジェクトは、生成時に ID 名称を指定することができます。指定した ID 名称は、以下の形式でカーネル側の情報は kernel_id_sys.h、カーネル環境側の情報は kernel_id.h に出力されます。

```
#define ID_MainTask 1
```

また、カーネル環境側については、コンフィギュレータが ID 番号を自動的に割り当てることもできます。生成ダイアログボックスで ID 番号として「自動割当」を選択すると、コンフィギュレータが空いている ID 番号を自動的に割り当てます。

(2) kernel_macro.h

kernel_macro.h は、kernel.h（「4.1 ヘッダファイル」参照）からインクルードされています。

(3) カーネル側のシステム定義ファイル(kernel_def_main.h, kernel_def_inidata.def, kernel_def_vct.inc)

kernel_def_main.h と kernel_def_inidata.def は、kernel_def.c からインクルードされています。また、kernel_def_vct.inc は HI7000/4 でのみ生成されるファイルで、標準提供の *nnnn*_expent.src と *nnnn*_intdwn.src からインクルードされています。

(4) カーネル環境側の構築ファイル(kernel_cfg_main.h, kernel_cfg_inidata.def)

kernel_cfg_main.h と kernel_cfg_inidata.def は、kernel_cfg.c からインクルードされています。

なお、HI7000/4 シリーズ間では、これらのファイルの内容は異なります。異なる HI7000/4 シリーズの環境でコンパイルを行うと、コンパイル時にエラーとなります。例えば、HI7750/4 のコンフィギュレータで生成したファイルを HI7000/4 または HI7700/4 の環境でコンパイルした場合のエラーメッセージは、以下のようになります。

```
Unmatch HIOS(This file is designed for HI7750/4.)
```

5.4.5 分割リンク

(1) 設定項目のリンク単位とカーネルロックモード

コンフィギュレータの全ての設定項目は「カーネル側」と「カーネル環境側」に分類されています。カーネル側の設定項目とカーネル環境側の設定項目は、それぞれ表 5.4のカーネル側のファイルとカーネル環境側のファイルに分割して出力されます。

分割リンクをする場合は、カーネルロードモジュールを更新せずに、カーネル環境ロードモジュールだけを更新するケースがほとんどです。これをサポートするために、コンフィギュレータは「カーネルロックモード」と呼ぶ操作モードを持っています。カーネルロックモードでは、カーネル側ファイルに展開されるパラメータの編集は制限され、カーネル側ファイルの出力も行いません。カーネルロックモードにするには、[生成->ｶｰﾈﾙﾛｯｸﾓｰﾄﾞ]を選択します。

どの項目がカーネル側なのかは、カーネルロックモードにすることで確認することもできますが、詳細はコンフィギュレータのヘルプを参照してください。

なお、タスクなどのオブジェクト生成については、以下のように設定・確認することができます。

- 生成ダイアログボックスには、[カーネルライブラリとリンク]チェックボックスがあります。これをチェックするとカーネル側となります。
- 一覧リストボックスで旗印のアイコンがあるものがカーネル側です。

タスクのアドレスなどアプリケーションのシンボルを指定する場合は、そのシンボルの実体を定められたリンク単位(カーネル側/カーネル環境側のロードモジュール)に含めなければなりません。

(2) kernel_id.h

kernel_id.h はカーネル環境側のファイルなので、分割リンクをする場合はカーネル側のアプリケーションで kernel_id.h をインクルードしないように注意してください。

5.4.6　コンフィギュレータの設定項目

コンフィギュレータに設定する項目の多くは、カーネルの動作に影響を与えます。そのような設定項目には、"CFG_"で始まる名称が付与されており、コンフィギュレータ画面に表示されるのはもちろん、本マニュアル中でも利用しています。なお、全ての設定項目に名称が付与されているわけではありません。

表 5.5に、各設定項目を示します。なお、ここでは以下の記号を用います。

- [L] : カーネル側の項目です。カーネルロックモードでは変更できません
- [B] : [カーネルライブラリとリンク]チェックボックスによって、カーネル側とカーネル環境側のいずれかを指定できる項目です。
- [BA] : : [カーネルライブラリとリンク]チェックボックスによって、カーネル側とカーネル環境側のいずれかを指定できる項目で、ID 名称を設定できる項目です。カーネル環境側へ設定する場合は、ID 番号の自動割当に対応しています。なお、ID 名称を指定する場合は、他の ID 名称および関数名などの外部定義名と重複しないようにしてください。

表5.5　コンフィギュレータの設定項目

1．カーネル動作条件ページ			
1	**カーネル割込みマスクレベル**	CFG_KNLMSKLVL	[L]
	カーネルは、SR.IMASK をここで指定する値に設定して実行する部分があります。CFG_KNLMSKLVL より高いレベルの割込みハンドラでは、サービスコールを発行してはなりません。また、HI7000/4 では CFG_KNLMSKLVL より高い割込みは、ダイレクト割込みハンドラとして記述する必要があります。		
2	**カーネル割込みマスクレベルより高い割込みネスト数(HI7000/4 のみ)**	CFG_UPPINTNST	
	CFG_KNLMSKLVL より高い割込みの最大ネスト数を指定してください。		
3	**カーネル割込みマスクレベル以下の割込みネスト数(HI7000/4 のみ)**	CFG_LOWINTNST	
	CFG_KNLMSKLVL 以下の割込みの最大ネスト数を指定してください。		
4	**TBR レジスタの利用形態(HI7000/4 のみ)**	CFG_TBR	[L]
	SH-2A の TBR レジスタの利用方法として、以下のいずれかから選択します。 (1) **カーネル管理外** (2)**サービスコール呼び出し専用に使用** (3)**タスクコンテキストとして扱う** **TBR を持たないマイコンを使用する場合は、必ず「カーネル管理外」を選択してください。** **詳細は、「4.2.8 TBR レジスタ(SH-2A, SH2A-FPU)」を参照してください。**		
2．カーネル拡張機能ページ			
1	**サービスコールのパラメータチェック**	CFG_PARCHK	[L]
	これをチェックすると、サービスコールの静的なパラメータエラーがチェックされます。チェックされるエラーは、「3. サービスコール」の各サービスコールの説明中で"[p]"と表記したエラーです。		
2	**SH3-DSP, SH4AL-DSP を使用(HI7700/4 のみ)**	CFG_DSP	[L]
	SR に DSP ビットを持つプロセッサ(SH7729 など)を使用する場合は必ずチェックしてください。また、「4.2.1 SR レジスタ」および「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」も参照してください。		
3	**キャッシュロック機能をもつプロセッサを使用(HI7700/4 のみ)**	CFG_CACLOC	[L]
	「4.2.2 キャッシュロック機能(SH-3, SH3-DSP)」を参照してください。		

表 5.5　コンフィギュレータの設定項目(続き)

3．時間管理機能ページ			
1	**カーネルの時間管理機能の使用**	CFG_TIMUSE	[L]
	tslp_tsk や周期ハンドラなど、時間パラメータを持つサービスコールを使用する場合は、CFG_TIMUSE をチェックしてください。この場合、タイマドライバを組み込む必要があります。詳細は、「付録 D タイマドライバ」を参照してください。		
2	**タイマの割込み番号**	CFG_TIMINTNO	[L]
	HI7000/4 ではタイマ割込みのベクタ番号を、HI7700/4 および HI7750/4 では INTEVT コードを指定してください。		
3	**タイマの割込みレベル**	CFG_TIMINTLVL	[L]
	CFG_TIMINTLVL は、kernel_macro.h に TIM_LVL として出力されます。タイマドライバは、kernel_macro.h のこの情報にしたがって、タイマ割込みレベルを設定しなければなりません。 HI7700/4 で最適化タイマドライバを使用する場合は、必ず CFG_KNLMSKLVL と同じ値にしなければなりません。		
4	**タイムイベントハンドラのスタックサイズ**	CFG_TMRSTKSZ	[L]
	「C.8 タイムイベントハンドラ」にしたがって算出したサイズを指定してください。		
5	**タイムティック周期の分子(TIC_NUME)と分母(TIC_DENO)**	CFG_TICNUME, CFG_TICDENO	[L]
	タイムティック周期時間は、TIC_NUME/TIC_DENO [ms] となります。TIC_NUME, TIC_DENO の少なくとも一方は 1 でなければなりません。なお、TIC_DENO に 1 より大きな値を指定した場合は、TMO, RELTIM, OVRTIM 型のパラメータに指定可能な最大値は、(その型で表現できる値)/TIC_DENO に制限されます。 TIC_NUME には 1～65535、TIC_DENO には 1～100 の範囲を整数値を指定できます。 TIC_NUME と TIC_DENO は、kernel_macro.h に出力されます。標準タイマドライバは、kernel_macro.h のこの情報にしたがって、タイマティックの周期を設定しなければなりません。		
4．デバッグ機能ページ			
1	**デバッギングエクステンションのオブジェクト操作機能**	CFG_ACTION	[L]
	これをチェックすると、デバッギングエクステンションで「タスクを起動する」といったオブジェクト操作機能（DX のダイアログタイトルに"[Action]"と表示されます）を使用する場合に、チェックしてください。 なお、CFG_ACTION をチェックした場合は、必ず以下の設定を行わなければなりません。 ・時間管理機能ページで CFG_TIMUSE をチェックする。 ・サービスコール機能ページで cre_cyc サービスコールを選択する。		
2	**サービスコールトレース機能の組み込み**	CFG_TRACE	[L]
	デバッギングエクステンションでサービスコールトレース機能を使用する場合はチェックしてください。また、「2.17.2 サービスコールトレース機能」も参照してください。		
3	**サービスコールトレース機能のタイプ**	CFG_TRCTYPE	[L]
	ターゲットトレースかエミュレータトレース(ツールトレース)を選択できます。エミュレータトレース(ツールトレース)を利用可能な環境は、デバッギングエクステンションのマニュアルまたはオンラインヘルプを参照してください。		
4	**ターゲットトレース選択時のバッファサイズ**	CFG_TRCBUFSZ	
	ターゲットトレース使用時のバッファサイズをバイト単位で指定します。		

表 5.5　コンフィギュレータの設定項目(続き)

5．サービスコール選択ページ			
1	組み込むサービスコールを選択します。組み込まれていないサービスコールを発行した場合は、エラーE_NOSPT が返ります。 なお、"i"で始まるサービスコールの選択項目はありません。"i"のないサービスコールを選択することで、自動的に対応する"i"で始まるサービスコールも選択された意味になります。 また。各オブジェクトを使用する場合は、必ずそのオブジェクトを生成・定義するサービスコール(下表参照)を選択してください。		

オブジェクト	サービスコール	オブジェクト	サービスコール
タスク(ﾀﾞｲﾅﾐｯｸｽﾀｯｸ使用)	cre_tsk	固定長メモリプール	cre_mpf
タスク(ｽﾀﾃｨｯｸｽﾀｯｸ使用)	vscr_tsk	可変長メモリプール	cre_mpl
タスク例外処理ルーチン	def_tex	周期ハンドラ	cre_cyc
セマフォ	cre_sem	アラームハンドラ	cre_alm
イベントフラグ	cre_flg	オーバーランハンドラ	def_ovr
データキュー	cre_dtq	拡張ｻｰﾋﾞｽｺｰﾙﾙｰﾁﾝ	def_svc
メッセージバッファ	cre_mbf		

なお、HI7700/4 の vchg_cop はサービスコール選択ページでは選択することはできず、DSP スタンバイ制御機能を組み込むことで自動的に組み込まれます。詳細は、「付録 F DSP スタンバイ制御機能(HI7700/4)」を参照してください。

6．割込み,CPU 例外ハンドラページ			
1	最大割込み番号	CFG_MAXVCTNO	[L]
	システムで使用する割込み・例外要因の中で、最大の番号を指定してください。指定する番号は、HI7000/4 ではプロセッサのベクタ番号、HI7700/4 および HI7750/4 ではプロセッサの割込み・例外コードです。指定できる最大値は、HI7000/4 では 511、HI7700/4 および HI7750/4 では 0x3fe0 です。		
2	割込みハンドラスタックサイズ	CFG_IRQSTKSZ	
	スタックサイズの算出方法は、「C.7 割込みハンドラのスタック」を参照してください。		
3	ダイレクト割込みハンドラのみ使用する(HI7000/4 のみ)	CFG_DIRECT	[L]
	通常の割込みハンドラ、CPU 例外ハンドラ、トラップ例外ハンドラを一切使用しない場合は、これをチェックしてください。		
4	ハンドラの定義情報を RAM に配置し、def_inh,def_exc,vdef_trp を組み込む(HI7000/4 のみ)	CFG_VCTRAM	[L]
	割込みハンドラ、CPU 例外ハンドラ、トラップ例外ハンドラの定義情報を ROM にするか RAM に配置するかを選択できます。前者は後者に対して以下の点が異なります。 ・RAM 消費量を後者より小さくできます。 ・def_inh,def_exc,vdef_trp は使用できません。 ・コンフィギュレータでハンドラを定義する時、「カーネルとリンク」のチェックを外すことはできません。		
5	レジスタバンク使用方法(HI7000/4 のみ)	CFG_REGBANK	[L]
	「4.2.9 レジスタバンク(SH-2A, SH2A-FPU)」を参照してください。		
6	IBNR レジスタアドレス(HI7000/4 のみ)	CFG_IBNR_ADR	[L]
	「4.2.9 レジスタバンク(SH-2A, SH2A-FPU)」を参照してください。		

RENESAS

表 5.5　コンフィギュレータの設定項目(続き)

7	割込みハンドラ, CPU 例外ハンドラの定義	−	[B]
	設定する内容は def_inh, def_exc サービスコールと同様です。 HI7000/4 では、vdef_trp サービスコールとも同等です。 HI7700/4 および HI7750/4 では、カーネル環境側に定義する場合は、def_inh または def_exc サービスコールを組み込む必要があります。 HI7000/4 では、CFG_VCTRAM に応じて def_inh, def_exc, vdef_trp サービスコールが組み込まれるかどうかが自動的に決まります。サービスコール選択ページでこれらのサービスコールの選択を行うことはできません。		
７．トラップ例外ハンドラページ (HI7700/4, HI7750/4 のみ)			
1	最大トラップ番号	CFG_MAXTRPNO	[L]
	システムで使用する最大のトラップ番号を指定してください。指定できる最大値は 255 です。		
2	CPU 例外ハンドラ(TRAPA 用)の定義	−	[B]
	設定する内容は vdef_trp サービスコールと同様です。 カーネル環境側に定義する場合は、vdef_trp サービスコールを組み込む必要があります。		
８．プリフェッチ機能ページ(HI7700/4,HI7750/4 のみ)			
1	プリフェッチするアドレスと範囲	−	[B]
	カーネルがアイドリング時にプリフェッチする領域の先頭アドレスとサイズを指定します。		
９．初期化ルーチンページ			
1	初期化ルーチンの登録	−	[B]
	初期化ルーチンのアドレス、拡張情報、スタックサイズなどを設定します。		
１０．タスクページ			
1	最大タスク ID	CFG_MAXTSKID	
	スタティックスタックを使用する最大タスク ID	CFG_STSTKID	[L]
	1〜CFG_STSTKID の ID のタスクがスタティックスタックを、CFG_STSTKID+1〜CFG_MAXTSKID の ID のタスクがダイナミックスタックを使用します。 CFG_MAXTSKID に指定できる最大値は 1023、CFG_STSTKID に指定できる最大値は CFG_MAXTSKID です。 CFG_MAXTSKID と CFG_STSTKID に応じ、以下のようにサービスコールを組み込む必要があります。 • CFG_STSTKID=0 の場合：全てのタスクがダイナミックスタックを使用するという意味になります。cre_tsk サービスコールを組み込む必要があります。 • CFG_MAXTSKID=CFG_STSTKID の場合：全てのタスクがスタティックスタックを使用するという意味になります。vscr_tsk サービスコールを組み込む必要があります。 • CFG_MAXTSKID>CFG_STSTKID の場合：cre_tsk, vscr_tsk サービスコールを組み込む必要があります。		
2	スタティックスタックの定義	−	[L]
	CFG_STSTKID に 0 以外を設定した場合、タスク ID が 1〜CFG_STSTKID のタスクが使用するスタティックスタック領域を定義します。		
3	最大タスク優先度(TMAX_TPRI)	CFG_MAXTSKPRI	[L]
	使用可能なタスク優先度の範囲は、1〜CFG_MAXTSKPRI となります。指定できる最大値は 255 です。CFG_MAXTSKPRI と同じ値が TMAX_TPRI として kernel_macro.h に出力されます。		
4	ダイナミックスタック領域サイズ	CFG_TSKSTKSZ	
	タスク ID が CFG_STSTKID+1〜CFG_MAXTSKID のタスクの生成時は、指定したサイズの領域からスタック領域が割り当てられます。		

表 5.5　コンフィギュレータの設定項目(続き)

5	**タスクの生成**	−	[BA]
	設定する内容は cre_tsk, vscr_tsk **サービスコールと同様です。** **スタティックスタックを使用するタスクの生成では、そのタスクが使用するスタックを項番** 2 **で定義したスタティックスタックの中から選びます。同じスタックが指定されたタスク同士は、そのスタックを共有することになります。** **属性に「生成後、起動**(TA_ACT)**」を指定する場合は、タスクの生成順によってタスクの実行順序が変わる可能性があります。タスクの生成順は、ウィンドウに表示されている順になるので、必要に応じてポップアップメニューの**[上へ]**または**[下へ]**でタスクを移動させてください。** **なお、**ID **番号の自動割当てが可能なタスクは、ダイナミックスタックを使用するタスクで、カーネル環境側に生成するものだけです。**		
6	**タスク例外処理ルーチンの定義**	−	[B]
	項番 5 **で生成したタスクに対してタスク例外処理ルーチンを定義できます。もちろん、定義しなくても構いません。分割リンク時は、タスクとタスク例外処理ルーチンは同じリンク単位**(**カーネル側またはカーネル環境側**)**にしてください。** **タスク例外処理ルーチンを定義する場合は、**def_tex **サービスコールを組み込む必要があります。**		
11．セマフォページ			
1	**最大セマフォ** ID	CFG_MAXSEMID	
	使用可能なセマフォ ID **の範囲は、**1〜CFG_MAXSEMID **となります。指定できる最大値は** 1023 **です。セマフォを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_sem **サービスコールを組み込む必要があります。**		
2	**セマフォの生成**	−	[BA]
	設定する内容は cre_sem **サービスコールと同様です。**		
12．イベントフラグページ			
1	**最大イベントフラグ** ID	CFG_MAXFLGID	
	使用可能なイベントフラグ ID **の範囲は、**1〜CFG_MAXFLGID **となります。指定できる最大値は** 1023 **です。イベントフラグを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_flg **サービスコールを組み込む必要があります。**		
2	**イベントフラグの生成**	−	[BA]
	設定する内容は cre_flg **サービスコールと同様です。**		
13．データキューページ			
1	**最大データキュー**ID	CFG_MAXDTQID	
	使用可能なデータキューID **の範囲は、**1〜CFG_MAXDTQID **となります。指定できる最大値は** 1023 **です。データキューを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_dtq **サービスコールを組み込む必要があります。**		
2	**データキュー用領域サイズ**	CFG_DTQSZ	
	データキューの生成時は、指定したサイズの領域からデータキューが割り当てられます。		
3	**データキューの生成**	−	[BA]
	設定する内容は cre_dtq **サービスコールと同様です。**		

表 5.5　コンフィギュレータの設定項目(続き)

１４．メールボックスページ			
1	**最大メールボックス ID**	CFG_MAXMBXID	
	使用可能なメールボックス ID の範囲は、1～CFG_MAXMBXID となります。指定できる最大値は 1023 です。メールボックスを使用しない場合は 0 を指定してください。 **1 以上の値を指定する場合は、cre_mbx サービスコールを組み込む必要があります。**		
2	**最大メッセージ優先度(TMAX_MPRI)**	CFG_MAXMSGPRI	[L]
	使用可能なメッセージ優先度の範囲は、1～CFG_MAXMSGPRI となります。CFG_MAXMSGPRI と同じ値が TMAX_MPRI として kernel_macro.h に出力されます。		
3	**メールボックスの生成**	－	[BA]
	設定する内容は cre_mbx サービスコールと同様です。		
１５．ミューテックスページ			
1	**最大ミューテックス ID**	CFG_MAXMTXID	
	使用可能なミューテックス ID の範囲は、1～CFG_MAXMTXID となります。指定できる最大値は 1023 です。ミューテックスを使用しない場合は 0 を指定してください。 **1 以上の値を指定する場合は、cre_mtx サービスコールを組み込む必要があります。**		
2	**ミューテックスの生成**	－	[BA]
	設定する内容は cre_mtx サービスコールと同様です。		
１６．メッセージバッファページ			
1	**最大メッセージバッファ ID**	CFG_MAXMBFID	
	使用可能なメッセージバッファ ID の範囲は、1～CFG_MAXMBFID となります。指定できる最大値は 1023 です。メッセージバッファを使用しない場合は 0 を指定してください。 **1 以上の値を指定する場合は、cre_mbf サービスコールを組み込む必要があります。**		
2	**メッセージバッファ用領域サイズ**	CFG_MBFSZ	
	メッセージバッファの生成時は、指定したサイズの領域からメッセージバッファが割り当てられます。		
3	**メッセージバッファの生成**	－	[BA]
	設定する内容は cre_mbf サービスコールと同様です。		
１７．固定長メモリプールページ			
1	**最大固定長メモリプール ID**	CFG_MAXMPFID	
	使用可能な固定長メモリプール ID の範囲は、1～CFG_MAXMPFID となります。指定できる最大値は 1023 です。固定長メモリプールを使用しない場合は 0 を指定してください。 **1 以上の値を指定する場合は、cre_mpf サービスコールを組み込む必要があります。**		
2	**固定長メモリプール用領域サイズ**	CFG_MPFSZ	
	固定長メモリプールの生成時は、指定したサイズの領域から固定長メモリプールが割り当てられます		
3	**固定長メモリプールの管理方式**	CFG_MPFMANAGE	[L]
	固定長メモリプール領域内にカーネル管理情報を置きたくない場合はチェックしてください。 **「2.14 固定長メモリプール」も参照してください。**		
4	**固定長メモリプールの生成**	－	[BA]
	設定する内容は、cre_mpf サービスコールと同様です。		

表 5.5　コンフィギュレータの設定項目(続き)

１８．可変長メモリプールページ			
1	**最大可変長メモリプール** ID	CFG_MAXMPLID	
	使用可能な可変長メモリプール ID **の範囲は、**1〜CFG_MAXMPLID **となります。指定できる最大値は** 1023 **です。可変長メモリプールを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_mpl **サービスコールを組み込む必要があります。**		
2	**可変長メモリプール用領域サイズ**	CFG_MPLSZ	
	可変長メモリプールの生成時は、指定したサイズの領域から可変長メモリプールが割り当てられます		
3	**可変長メモリプールの管理方式**	CFG_NEWMPL	[L]
	CFG_NEWMPL **を選択しない場合は、従来バージョン**(HI7000/4 V.2.00 Release 02 **以前、**HI7700/4 V.1.03 Release.02 **以前、**HI7750/4 V1.1.00 **以前**)**と同じ管理方式となります。** CFG_NEWMPL **を選択すると、以下が改善されます。** (1) **特に多くのメモリブロックを扱うメモリプールで、獲得**/**返却処理が高速化されます。** (2) **空き領域の断片化を軽減する** VTA_UNFRAGMENT **属性を使用できます。** **なお、**CFG_NEWMPL **選択時は、従来バージョンに対して** T_CMPL **構造体に新メンバが追加になります。詳細は、「**3.14.1 **可変長メモリプールの生成**(cre_mpl, icre_mpl)**」を参照してください。**		
4	**可変長メモリプールの生成**	−	[BA]
	設定する内容は、cre_mpl **サービスコールと同様です。**		
１９．周期ハンドラページ			
1	**最大周期ハンドラ** ID	CFG_MAXCYCID	
	使用可能な周期ハンドラ ID **の範囲は、**1〜CFG_MAXCYCID **となります。指定できる最大値は** 14 **です。周期ハンドラを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_cyc **サービスコールを組み込む必要があります。**		
2	**周期ハンドラの生成**	−	[BA]
	設定する内容は cre_cyc **サービスコールと同様です。**		
２０．アラームハンドラページ			
1	**最大アラームハンドラ** ID	CFG_MAXALMID	
	使用可能なアラームハンドラ ID **の範囲は、**1〜CFG_MAXALMID **となります。指定できる最大値は** 15 **です。アラームハンドラを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**cre_alm **サービスコールを組み込む必要があります。**		
2	**アラームハンドラの生成**	−	[BA]
	設定する内容は cre_alm **サービスコールと同様です。**3 **章のサービスコールの説明も参照してください**		
２１．オーバーランハンドラページ			
1	**オーバーランハンドラの定義**	−	[B]
	設定する内容は def_ovr **サービスコールと同様です。定義する場合は、**def_ovr **サービスコールを組み込む必要があります。**		
２２．拡張サービスコールページ			
1	**拡張サービスコールの最大機能コード**	CFG_MAXSVCCD	
	使用可能な機能コードの範囲は、1〜CFG_MAXSVCCD **となります。指定できる最大値は** 1023 **です。拡張サービスコールを使用しない場合は** 0 **を指定してください。** 1 **以上の値を指定する場合は、**def_svc **サービスコールを組み込む必要があります。**		
2	**拡張サービスコールルーチンの定義**	−	[B]
	設定する内容は def_svc **サービスコールと同様です。**		

5.5 最適化タイマドライバを使用する場合(HI7700/4 のみ)

「付録 E 最適化タイマドライバ(HI7700/4)」を参照してください。

5.6 DSP スタンバイ制御機能を使用する場合(HI7700/4 のみ)

「付録 F DSP スタンバイ制御機能(HI7700/4)」を参照してください。

5.7 SH4AL-DSP(HI7700/4)または SH-4A(HI7750/4)でキャッシュサポートサービスコールを使用する場合

kernel_cfg_cac_set.h に、使用するマイコンが搭載している命令キャッシュとオペランドキャッシュのサイズを定義してください。ただし、搭載されているキャッシュが 2 ウェイ構成の場合は、実際のキャッシュ容量の倍の値を指定してください。なお、キャッシュが 4 ウェイ構成の場合で、2 ウェイモードで使用する場合は、倍にしてはなりません。

このファイルは、kernel_cfg.c からインクルードされます。以下に例を示します。

```
/***************************************************************************
 * Cache size information
 *  Please define the specified Cache size in CPU to be used.
 *    (1) IC_SIZE : Instruction Cache size [Bytes]
 *    (2) OC_SIZE : Operand Cache size [Bytes]
 ***************************************************************************/
#define IC_SIZE  32768UL
#define OC_SIZE  32768UL
```

なお、kernel_cfg_cac_set.h は、CPU コアが SH4AL-DSP, SH-4A のマイコン用のサンプルフォルダのみ(本マニュアル作成時点では、HI7700/4 V.2.02 Release 00 の sh7318 フォルダと sh7343 フォルダ、HI7750/4 の V.2.02 Release 00 の sh7770 フォルダと sh7785 フォルダ)に格納されています。同様に、これらのフォルダ内の kernel_cfg.c のみ、kernel_cfg_cac_set.h をインクルードするようになっています。

5.8 HEW ワークスペースとプロジェクト

ロードモジュールの生成は HEW のビルド機能を用いて行います。HEW のビルド機能では、以下の手順でロードモジュールを生成します。

(a) ロードモジュールの生成に必要なファイルをプロジェクトに登録する。

(b) Cコンパイラ、アセンブラ、最適化リンケージエディタのオプションを設定する。

(c) ビルドを実行する。

本製品では、サンプルワークスペースファイル(sh***nnnn***.hws)を提供しています。サンプルワークスペースファイルをダブルクリックすると、HEW が起動します。

sh***nnnn***.hws には、当該マイコン用の以下のサンプルプロジェクトがあらかじめ登録されています。

(1) HEW3以降用

- mix：一括リンク用
- def：分割リンク・カーネル側用
- def：分割リンク・カーネル環境側用

(2) HEW1.2用

- ***nnnn***_mix：一括リンク用(***nnnn*** に該当するマイコン用)
- ***nnnn***_def：分割リンク・カーネル側用(***nnnn*** に該当するマイコン用)
- ***nnnn***_cfg：分割リンク・カーネル環境側用(***nnnn*** に該当するマイコン用)

使用するマイコン用のプロジェクトを選択し、以降の説明を参考に設定を変更してください。以降の節では、特に留意すべき項目についてのみ解説します。

サンプルプロジェクトを選択するには、図 5.9のように HEW のワークスペースウィンドウでプロジェクトを選び、ポップアップメニューで [アクティブプロジェクトに設定]を選択してください。

ビルドを実行することで、プロジェクトに登録してあるファイルのコンパイル、アセンブル、リンクを一連の流れで実行し、ロードモジュールを生成します。

本節以降では以下の環境を前提に説明します。

- HEW：コンパイラパッケージ V.9.00 Release 03 に付属の HEW4
- カーネル：HI7700/4 V.2.02 Release 00

HEW のバージョンや使用するカーネルによって、画面が本節に掲載のものと異なる場合があります。

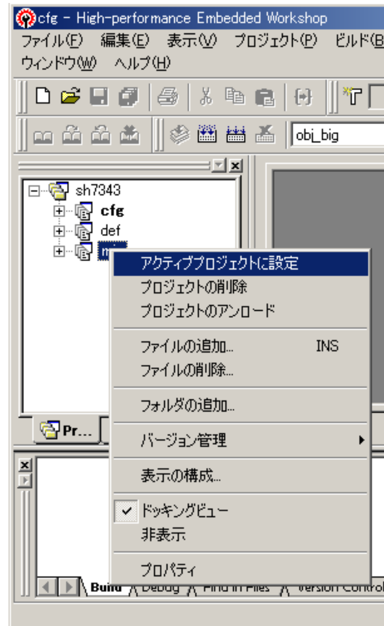


図5.9　プロジェクトの選択

5.9　カーネルライブラリ

カーネルライブラリは hilib¥elf フォルダに格納されており、一括リンク時(mix)、または分割リンク時のカーネル側(def)でリンクします。カーネルライブラリの指定は、後述の節での各プロジェクトの設定の中で行います。リンク時には、使用する機能に応じて複数のカーネルライブラリを優先順位をつけて指定する必要があります。なお、HEW ではライブラリ指定画面で上方に指定するほど優先になります(図 5.10参照)。

5.9.1　HI7000/4

表 5.6に、HI7000/4 のカーネルライブラリのリンク優先順位を示します。

表5.6　カーネルライブラリのリンク優先順位(HI7000/4)

項番	使用する CPU コア	リンク優先順位
1	SH-1, SH-2, SH-2A(FPU 無し)	ビッグエンディアン：hiknl.lib リトルエンディアン：hiknl_little.lib
2	SH2-DSP	(1) dsp_knl.lib (2) hiknl.lib
3	SH2A-FPU	(1) fpu_knl.lib (2) hiknl.lib

なお、以下のカーネルライブラリはデバッガサポート用であり、通常は使用しません。

- hiexpand.lib, hiexpand_little.lib
- dsp_expand.lib
- fpu_expand.lib

5.9.2 HI7700/4

表 5.7に、HI7700/4 のカーネルライブラリのリンク優先順位を示します。

表5.7 カーネルライブラリのリンク優先順位(HI7700/4)

項番	使用するCPU コア	DSP スタンバイ制御機能	最適化タイマドライバ機能	リンク優先順位
1	SH-3	(使用不可)	未使用	(1) 7708_cache_???.lib (2) hiknl_???.lib
2			使用	(1) opttmr_???.lib (2) 7708_cache_???.lib (3) hiknl_???.lib
3	SH3-DSP	未使用	未使用	(1) dsp_knl_???.lib (2) 7708_cache_???.lib (3) hiknl_???.lib
4			使用	(1) dsp_knl_???.lib (2) opttmr_???.lib (3) 7708_cache_???.lib (4) hiknl_???.lib
5		使用	未使用	(1) dspstby_???.lib (2) dsp_knl_???.lib (3) 7708_cache_???.lib (4) hiknl_???.lib
6			使用	(1) dspstby_???.lib (2) dsp_knl_???.lib (3) opttmr_???.lib (4) 7708_cache_???.lib (5) hiknl_???.lib
7	SH4AL-DSP (拡張機能なし)	未使用	未使用	(1) sh4al_dsp_knl_???.lib (2) sh4al_cache_???.lib (3) hiknl_???.lib
8			使用	(1) sh4al_dsp_knl_???.lib (2) sh4al_opttmr_???.lib (3) sh4al_cache_???.lib (4) hiknl_???.lib
9		使用	未使用	(1) sh4al_dspstby_???.lib (2) sh4al_dsp_knl_???.lib (3) sh4al_cache_???.lib (4) hiknl_???.lib
10			使用	(1) sh4al_dspstby_???.lib (2) sh4al_dsp_knl_???.lib (3) sh4al_opttmr_???.lib (4) sh4al_cache_???.lib (5) hiknl_???.lib

表 5.7　カーネルライブラリのリンク優先順位(HI7700/4) (続き)

項番	使用する CPU コア	DSP スタンバイ制御機能	最適化タイマドライバ機能	リンク優先順位
11	SH4AL-DSP (拡張機能有り)	未使用	未使用	(1) sh4al_dsp_knl_???.lib (2) shx2_cache_???.lib (3) hiknl_???.lib
12			使用	(1) sh4al_dsp_knl_???.lib (2) sh4al_opttmr_???.lib (3) shx2_cache_???.lib (4) hiknl_???.lib
13		使用	未使用	(1) sh4al_dspstby_???.lib (2) sh4al_dsp_knl_???.lib (3) shx2_cache_???.lib (4) hiknl_???.lib
14			使用	(1) sh4al_dspstby_???.lib (2) sh4al_dsp_knl_???.lib (3) sh4al_opttmr_???.lib (4) shx2_cache_???.lib (5) hiknl_???.lib

なお、以下のカーネルライブラリはデバッガサポート用であり、通常は使用しません。

- hiexpand_???.lib
- dsp_expand_???.lib
- sh4al_dsp_expand_???.lib

備考：7708_cache_???.lib は、SH7708 専用ではなく、SH-3, SH3-DSP 共用です。

5.9.3　HI7750/4

表 5.8に、HI7750/4 のカーネルライブラリのリンク優先順位を示します。

表5.8　カーネルライブラリのリンク優先順位(HI7750/4)

項番	使用する CPU コア	リンク優先順位
1	SH-4	(1) 7750_cache_???.lib (2) hiknl_???.lib
2	SH-4A(拡張機能なし)	(1) sh4a_cache_???.lib (2) hiknl_???.lib
3	SH-4A(拡張機能あり)	(1) shx2_cache_???.lib (2) hiknl_???.lib

なお、以下のカーネルライブラリはデバッガサポート用であり、通常は使用しません。

- hiexpand_???.lib

備考：7750_cache_???.lib は、SH7750 専用ではなく、SH-4 共用です。

5.10 セクション構成

リンク方式にかかわらず、各モジュールの配置アドレスはセクション単位にリンク時に決定します。
ここでは、セクションについて説明します。
表 5.9に提供ファイルのセクション一覧を示します。
各セクション名の先頭 1 文字は、セクションの属性を表しています。

(1) P 属性

プログラムセクションです。ROM に配置できます。

(2) C 属性

定数セクションです。ROM に配置できます。

(3) B 属性

未初期化データセクションです。RAM に配置しなければなりません。

(4) D 属性と R 属性

D 属性は初期化データセクションで、ROM に配置できます。D 属性のセクションを ROM に配置する場合、D 属性セクションの内容を変数として扱えるようにする為に、プログラム実行前に D 属性セクションの内容を RAM にコピーする必要があります。具体的には、以下の作業が必要になります。

(a) 最適化リンケージエディタのROM化支援機能を用いて、D属性セクションと同じ大きさのR属性セクションを生成する。R属性のセクションはRAMに配置する。

(b) D属性セクションの内容をR属性セクションにコピーするプログラムを作成し、プログラム開始時（通常はCPU初期化ルーチン）に実行する。

RENESAS

表5.9　セクション一覧

項番	セクション名	ファイル名	内容	非ｷｬｯｼｬﾌﾞﾙ領域への配置
1	P_hiknl	カーネルライブラリ *1	カーネルプログラム	不要
2	P_hireset		カーネルの初期化処理	不要
3	P_hiknl_P2 *4		キャッシュ操作の一部	必要
4	P_hiexpent	***nnnn***_expent.src *2 kernel_exp.src *3	カーネルの割込み･例外出入口処理ﾙｰﾁﾝ	不要
5	P_hiintdwn	***nnnn***_intdwn.src *2 kernel_exp.src *3	カーネルの未定義割込み/例外解析処理	不要
6	C_hidef	kernel_def.c	カーネル側の定義情報	不要
7	C_hivct		CPU ベクタテーブル, 割込みハンドラ,CPU 例外ハンドラ登録テーブル	不要 *7
8	C_hitrp *2		TRAPA 用 CPU 例外ﾊﾝﾄﾞﾗ登録テーブル	不要
9	C_hibase		サービスコールｲﾝﾀﾌｪｰｽ情報	不要
10	B_hidef *5		カーネル作業領域	不要
11	C_hisysmt	kernel_cfg.c	カーネル環境側の構築情報	不要
12	C_hicfg		C_hisysmt 以外のカーネル環境側の構築情報	不要
13	B_hivct *6		CPU ベクタテーブル	不要
14	B_hiwrk		カーネル作業領域	不要
15	B_hicfg *5		カーネル作業領域	不要
16	B_himpl		可変長/固定長メモリプール用領域	不要
17	B_hidystk		ダイナミックスタック	不要
18	B_histstk		スタティックスタック	不要
19	B_hiirqstk		割込みハンドラ､タイムイベントハンドラスタック	不要
20	B_hitrceml		DX ｴﾐｭﾚｰﾀﾄﾚｰｽ(ﾂｰﾙﾄﾚｰｽ)領域	不要
21	B_hitrcbuf		DX ターゲットトレースバッファ	不要
22	P_hisysdwn	***nnnn***_sysdwn.c	サンプルシステムダウンルーチン	不要
23	P_hicpuasm	***nnnn***_cpuasm.src	サンプル CPU 初期化ルーチン	*8
24	P_hicpuini	***nnnn***_cpuini.c	サンプル CPU 初期化サブルーチン	不要
25	P_hitmrdrv	***nnnn***_tmrdrv.c	サンプルタイマドライバ	不要
26	任意	アプリケーション	—	

【注】*1　「5.9 カーネルライブラリ」参照

*2　HI7700/4, HI7750/4 のみ

*3　HI7000/4 のみ

*4　HI7700/4 の sh4al_cache_???.lib と shx2_cache_???.lib、HI7750/4 の sh4a_cache_???.lib と shx2_cache_???.lib のみ

*5　コンフィギュレータで CFG_MPFMANAGE をチェックして固定長メモリプールを生成した場合、および CFG_NEWMPL をチェックして VTA_UNFRAGMENT 属性の可変長メモリプールを生成した場合。

*6　HI7000/4 で CFG_VCTRAM を選択した場合のみ

*7　HI7000/4 の場合、コンフィギュレータでリセットベクタを指定した場合、0 番地に配置してください。「4.11.2 HI7000/4 の CPU 初期化ルーチンの登録」も参照してください。

*8　HI7700/4, HI7750/4 では、CPU のリセットベクタ(P2 領域の H'a0000000)に配置してください。

5.11　各プロジェクト共通の設定

5.11.1　コンパイラ、アセンブラの CPU オプション

コンパイラ、アセンブラの CPU オプションは、使用するマイコンに応じた設定を行ってください。特に、以下のファイルは必ず表 5.10の通りに指定してください。

表5.10　CPU オプション

カーネル	使用 CPU コア	対象ソースファイル	CPU オプション
HI7000/4	SH-2A, SH2A-FPU	(1) kernel_def.c(mix, def プロジェクトに含まれています) (2) kernel_cfg.c(mix, cfg プロジェクトに含まれています) (3) kernel_exp.src (4)CFG_TBR が「サービスコール呼び出し専用に使用」の場合は、サービスコールを使用する全ファイル	SH2A または SH2AFPU
HI7700/4	SH4AL-DSP	(1) kernel_def.c(mix, def プロジェクトに含まれています) (2) kernel_cfg.c(mix, cfg プロジェクトに含まれています) (3) キャッシュサポートサービスコールを使用するファイル	SH4ALDSP
HI7750/4	SH-4A		SH4A

5.11.2　コンパイラの GBR オプション(コンパイラパッケージ V7.1 以降)

いくつかのデバイス用のサンプルタイマドライバ(***nnnn***_tmrdrv.c)では、コンパイラの GBR 組み込み関数を使用しています。これらのサンプルタイマドライバを使用する場合は、"gbr=user"を設定してください。以下に、本マニュアル作成時点の各カーネルバージョンで、GBR 組み込み関数を使用しているサンプルタイマドライバを示します。

- HI7000/4 V.2.02.00：SH72060, SH7618 用以外全て
- HI7700/4 V.2.02.00：SH7290, SH7318, SH7641, SH7343 用以外全て
- HI7750/4 V.2.02.00：SH7760, SH7770, SH7785 用以外全て

5.11.3　コンパイラの PACK オプションと#pragma pack について(コンパイラパッケージ V8 以降)

カーネルに渡す構造体パラメータは、メンバの境界調整数と同じでなければなりません。これを保証するために、標準ヘッダファイル(itron.h, kernel.h)では、構造体の型定義に#pragma pack 4 を指定しています。

5.11.4　C/C++コンパイラ、アセンブラのインクルードディレクトリ

標準ヘッダファイルは hihead フォルダに格納されているため、通常は hihead フォルダをインクルードディレクトリに指定してください。

また、kernel_def.c と kernel_cfg.c は、hisys フォルダをインクルードディレクトリに指定してください。

5.11.5　SH2A-FPU, SH-4, SH-4A 使用時

「付録 G SH2A-FPU, SH-4, SH-4A における FPU に関する注意」にもコンパイラオプションに関する記載があります。FPU 機能を使用しない場合も、必ず参照してください。

RENESAS

5.11.6 コンパイラの TBR オプション(コンパイラパッケージ V9 以降)

TBR オプションは、SH-2A および SH2A-FPU 専用のオプションです。

CFG_TBR を「サービスコール呼び出し専用に使用」に設定した場合は、TBR はカーネルが初期化し、以降変更してはなりません。このため、TBR オプションを指定してはなりません。同様に#pragma tbr も使用してはなりません。

5.12 一括リンクでのビルド(mix プロジェクト)

5.12.1 プロジェクトへの登録

表 5.11に、プロジェクトに登録するソースプログラムファイルを示します。サンプルのプロジェクトでは、これらのファイルがあらかじめ登録されています。

表5.11 プロジェクトに登録するソースプログラムファイル(mix)

項番	ファイル名	内容	備考
1	kernel_def.c *1	カーネル側構築ファイル	必須
2	kernel_cfg.c *1	カーネル環境側構築ファイル	必須
3	***nnnn***_sysdwn.c *1	システムダウンルーチン	必須
4	***nnnn***_expent.src *1 *2	割込み・例外出入口処理ルーチン	必須
5	***nnnn***_intdwn.src *1 *2	未定義割込み詳細情報取得処理	必須
6	kernel_exp.src *1 *3	割込み・例外処理ルーチン	必須
7	***nnnn***_cpuasm.src *1 ***nnnn***_cpuini.c *1	CPU 初期化ルーチン	リセットで実行する場合は必須
8	***nnnn***_tmrdrv.c *1	標準タイマドライバ	HI7700/4 で最適化タイマドライバを使用する場合は、プロジェクトに含めないでください。
9	task.c *1	サンプルタスク	
10	アプリケーションファイル	−	

【注】 *1 sh***nnnn***¥src フォルダまたは sh***nnnn***¥project¥src フォルダにあります。
*2 HI7700/4, HI7750/4 のみ
*3 HI7000/4 のみ

5.12.2 エンディアンの指定

リトルエンディアンをサポートしているマイコン用のサンプルプロジェクトでは、エンディアンの指定を HEW のビルドコンフィギュレーションで選択するように設定しています。ビルドコンフィギュレーションは、図 5.10のように選択してください。

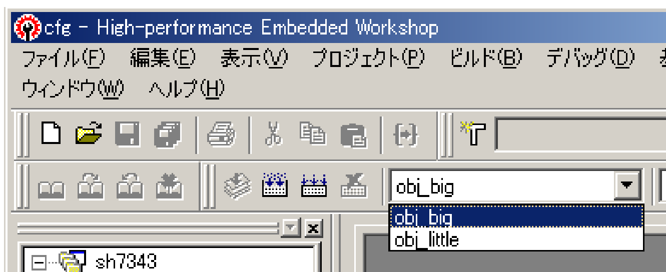


図5.10 エンディアンの選択

5.12.3 最適化リンケージエディタの設定

(1) [Input]カテゴリの[Library files]

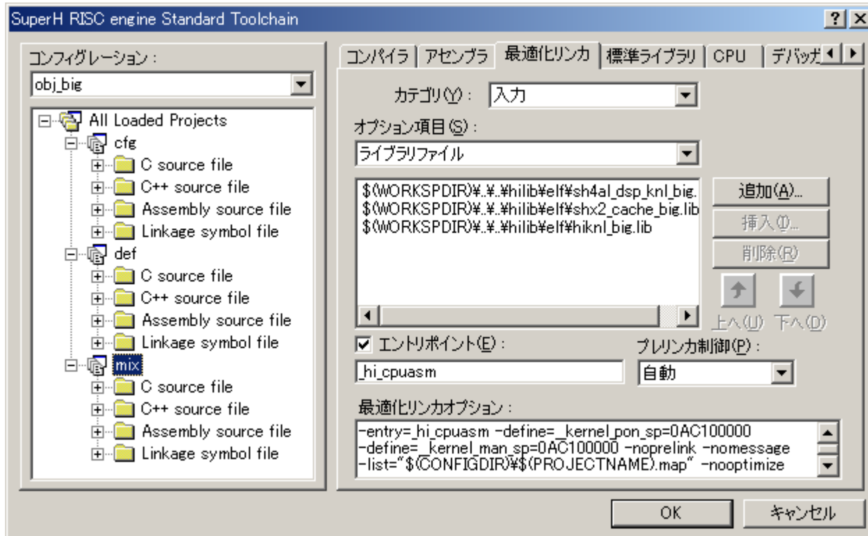


図5.11 [Input]カテゴリの[Library files]

「5.9 カーネルライブラリ」を参考に、必要なライブラリを指定してください。
アプリケーションのライブラリなどを使用する場合は、そのライブラリも指定してください。

(2) [Input]カテゴリの[Defines]

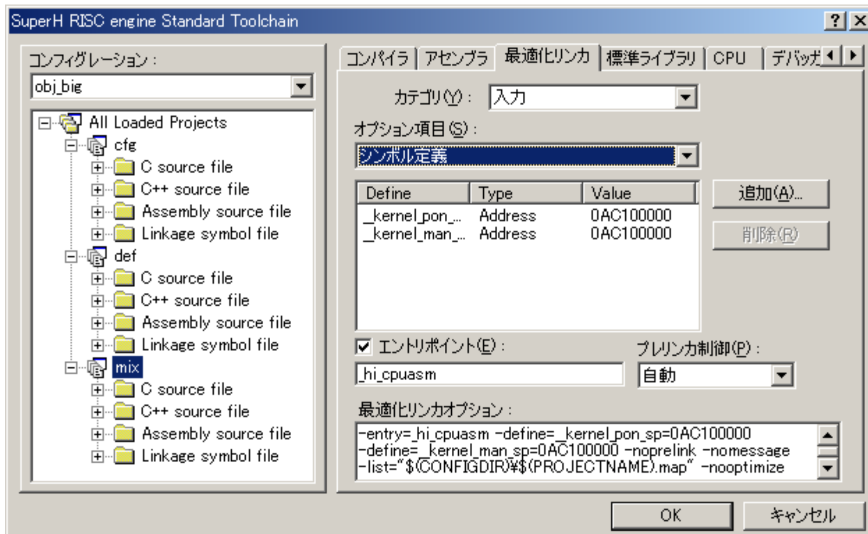


図5.12 [Input]カテゴリの[Defines]

HI7700/4, HI7750/4 のサンプルの ***nnnn***_cpuasm.src では、以下のシンボルを外部参照しているので、ここで定義してください。

- __kernel_pon_sp： パワーオンリセット時に使用する初期スタックポインタ
- __kernel_man_sp： マニュアルリセット時に使用する初期スタックポインタ

(3) [Section]カテゴリ

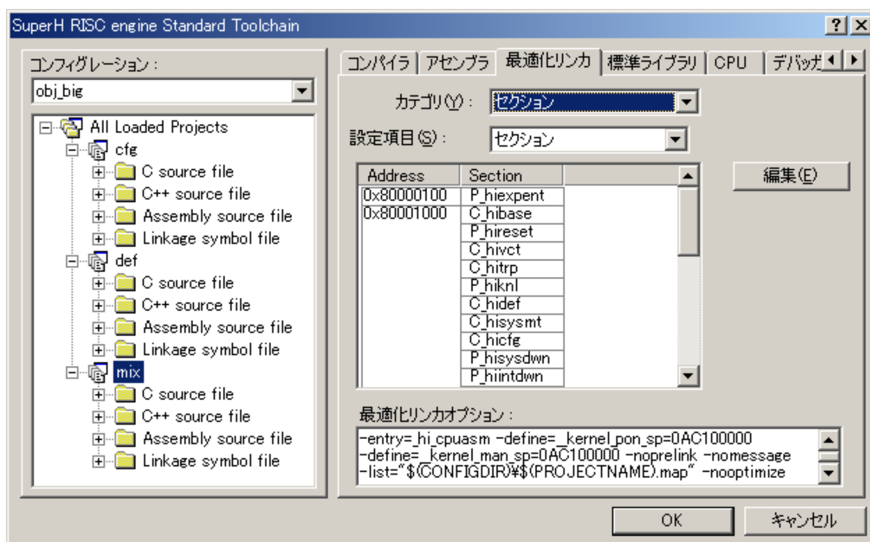


図5.13 [Section]カテゴリ

各セクションの配置アドレスを指定します。
ターゲットハードウェアにあわせて、入力ファイルに含まれる各セクションを指定し、配置アドレスを指定してください。

基本的には、入力ファイル中に含まれるすべてのセクションを漏れなく指定してください。指定されていないセクションが入力ファイル中に存在する場合、最適化リンケージエディタは最後に指定したセクションに続いてこれらのセクションを自動的に配置するため、ユーザが意図しない配置となる可能性があり、誤動作の原因となります。なおこの場合、最適化リンケージエディタは以下のようなウォーニングを出力します(P_Task1 セクションが指定されていない場合の例)。

```
L1120 (W) Section address is not assigned to "P_Task1"
```

逆に、指定したセクションが入力ファイル中に存在しない場合、最適化リンケージエディタは以下のようなウォーニングを表示します(C セクションが存在しない場合)が、結合は正常に行われます。

```
L1100 (W) Cannot find "C" specified in option "start"
```

提供時の設定では、P、C、D、B、R などのセクションについて、このウォーニングが表示される場合があります。これは、結合したアプリケーションオブジェクト内でそれらのセクションを使用していないためです。生成されるロードモジュールに何ら問題はありません。

♦ **HI7000/4 のメモリ配置**

ベクタテーブル（C_hivct セクション）は必ず指定してください。C_hivct セクションはコンフィギュレータでリセットベクタを定義した場合は、必ず 0 番地に配置してください。

♦ **HI7700/4、HI7750/4 のメモリ配置**

CPU 初期化ルーチンのセクション(P_hicpuasm)は、リセットアドレスである H'a0000000 番地に配置してください。

RENESAS

5.12.4　ビルドの実行

プロジェクトへのアプリケーションファイルの登録、コンパイル、アセンブル、最適化リンケージエディタのオプション設定後、ビルドを実行することでロードモジュールを生成します。

ビルドを実行するには、図 5.14のように HEW で[ビルド->ビルド]または[ビルド->すべてをビルド]を選択してください。

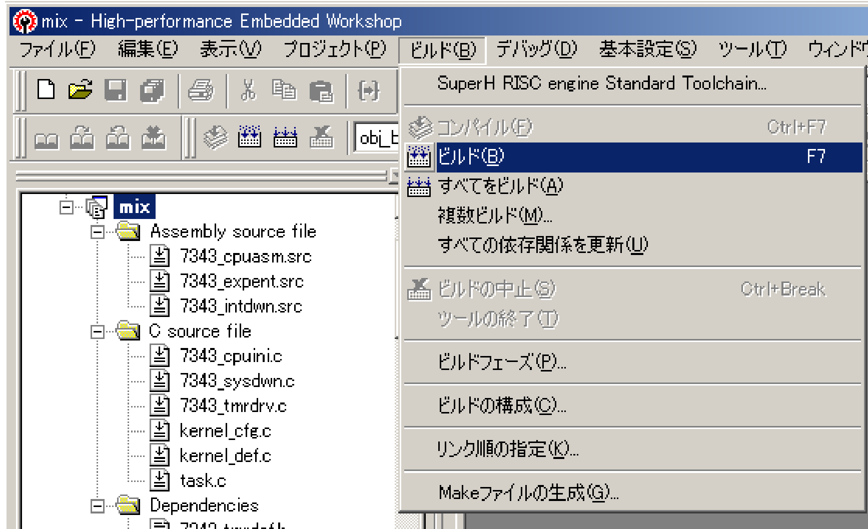


図5.14　ビルドの実行

5.13　分割リンク：カーネル側のビルド(def プロジェクト)

5.13.1　プロジェクトへの登録

表 5.12に、プロジェクトに登録するソースプログラムファイルを示します。サンプルのプロジェクトでは、これらのファイルがあらかじめ登録されています。

表5.12　プロジェクトに登録するソースプログラムファイル(def)

項番	ファイル名	内容	備考
1	kernel_def.c *1	カーネル側構築ファイル	必須
2	***nnnn***_sysdwn.c *1	システムダウンルーチン	必須
3	***nnnn***_expent.src *1 *2	割込み・例外出入口処理ルーチン	必須
4	***nnnn***_intdwn.src *1 *2	未定義割込み詳細情報取得処理	必須
5	kernel_exp.src *1 *3	割込み・例外処理ルーチン	必須
6	***nnnn***_cpuasm.src *1 ***nnnn***_cpuini.c *1	CPU 初期化ルーチン	リセットで実行する場合は必須
7	***nnnn***_tmrdrv.c *1	標準タイマドライバ	HI7700/4 で最適化タイマドライバを使用する場合は、プロジェクトに含めないでください。
8	アプリケーションファイル	−	

【注】 *1　sh***nnnn***¥src フォルダまたは sh***nnnn***¥project¥src フォルダにあります。
*2　HI7700/4, HI7750/4 のみ
*3　HI7000/4 のみ

5.13.2　エンディアンの指定（HI7700/4、HI7750/4）

提供のサンプルプロジェクトでは、エンディアンの指定を前述の図 5.10のように HEW のビルドコンフィギュレーションで選択するように設定しています。

5.13.3 最適化リンケージエディタの設定

(1) [Input]カテゴリの[Library files]

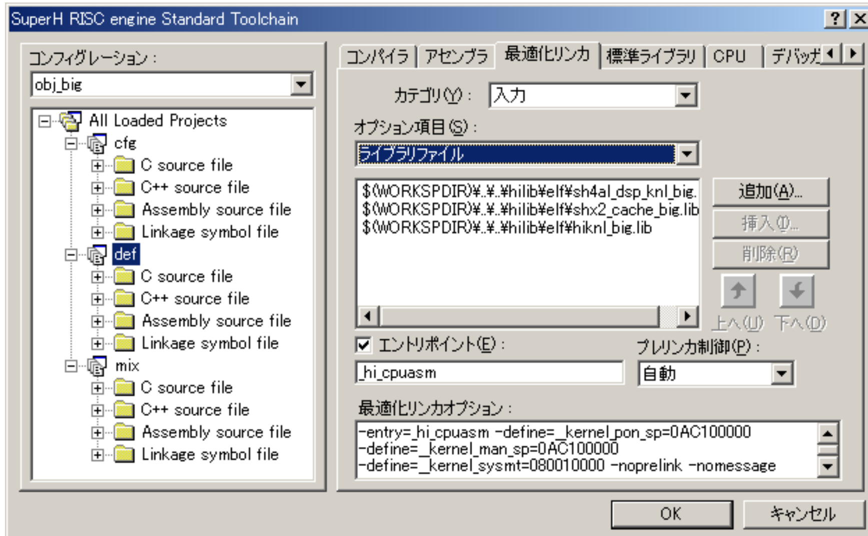


図5.15 [Input]カテゴリの[Library files]

「5.9 カーネルライブラリ」を参考に、必要なライブラリを指定してください。
アプリケーションのライブラリなどを使用する場合は、そのライブラリも指定してください。

(2) [Input]カテゴリの[Defines]

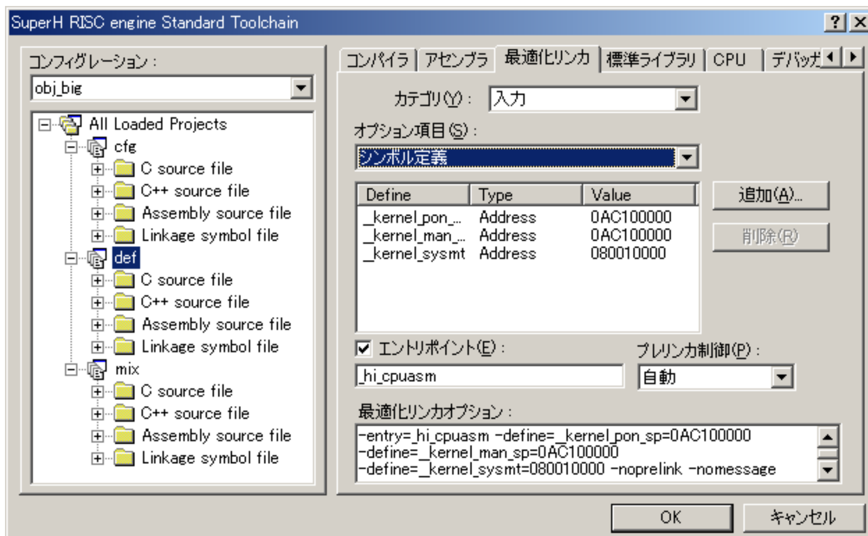


図5.16 [Input]カテゴリの[Defines]

以下のシンボルアドレスを定義します。

- __kernel_sysmt：カーネル環境情報

カーネル環境情報はカーネル環境ロードモジュール(cfg)に含まれるため、ここでカーネル環境情報

の配置アドレスを強制的に定義する必要があります。__kernel_sysmt は、カーネル環境側構築ファイル(kernel_cfg.c)中の C_hisysmt セクションの先頭アドレスです。カーネル環境側ロードモジュール生成時には、C_hisysmt セクションをここで定義したアドレスに配置しなければなりません。

また、HI7700/4, HI7750/4 のサンプルの ***nnnn***_cpuasm.src では、以下のシンボルを外部参照しているので、ここで定義してください。

- __kernel_pon_sp : パワーオンリセット時に使用する初期スタックポインタ
- __kernel_man_sp : マニュアルリセット時に使用する初期スタックポインタ

(3) [Section]カテゴリ

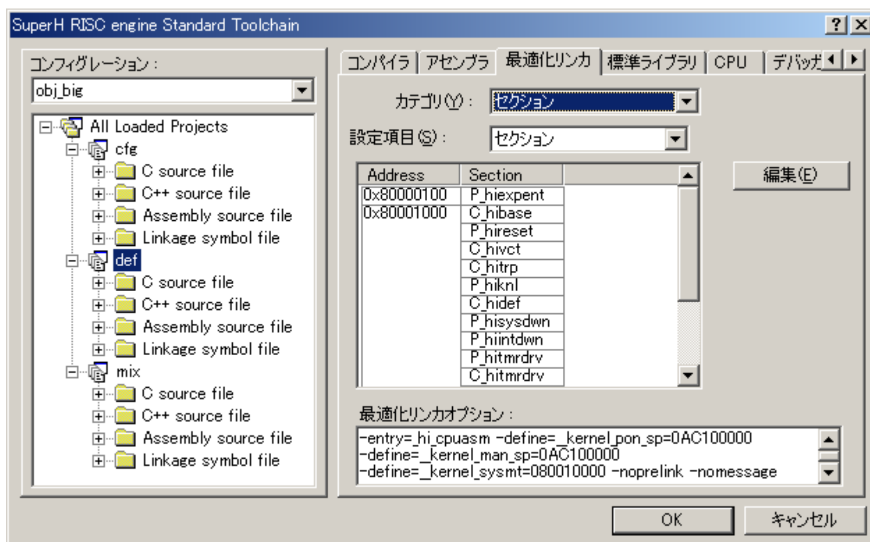


図5.17 [Section]カテゴリ

各セクションの配置アドレスを指定します。

ターゲットハードウェアにあわせて、入力ファイルに含まれる各セクションを指定し、配置アドレスを指定してください。

カーネル環境ロードモジュール(**cfg**)生成時には、__kernel_cnfgtbl(サービスコールインタフェース情報 C_hibase セクションの先頭アドレス)を指定する必要があります。この定義アドレスと、ここでの C_hibase セクションの配置アドレスは同じでなければなりません。

基本的には、入力ファイル中に含まれるすべてのセクションを漏れなく指定してください。指定されていないセクションが入力ファイル中に存在する場合、最適化リンケージエディタは最後に指定したセクションに続いてこれらのセクションを自動的に配置するため、ユーザが意図しない配置となる可能性があり、誤動作の原因となります。なおこの場合、最適化リンケージエディタは以下のようなウォーニングを出力します(P_Task1 セクションが指定されていない場合の例)。

```
L1120 (W) Section address is not assigned to "P_Task1"
```

逆に、指定したセクションが入力ファイル中に存在しない場合、最適化リンケージエディタは以下のようなウォーニングを表示します(C セクションが存在しない場合)が、結合は正常に行われます。

```
L1100 (W) Cannot find "C" specified in option "start"
```

提供時の設定では、P、C、D、B、R などのセクションについて、このウォーニングが表示される場合があります。これは、結合したアプリケーションオブジェクト内でそれらのセクションを使用していないためです。生成されるロードモジュールに何ら問題はありません。

♦ HI7000/4 のメモリ配置

ベクタテーブル（C_hivct セクション）は必ず指定してください。C_hivct セクションはコンフィギュレータでリセットベクタを定義した場合は、必ず 0 番地に配置してください。

♦ HI7700/4、HI7750/4 のメモリ配置

CPU 初期化ルーチンのセクション(P_hicpuasm)は、リセットアドレスである H'a0000000 番地に配置してください。

5.13.4 ビルドの実行

プロジェクトへのアプリケーションファイルの登録、コンパイル、アセンブル、最適化リンケージエディタのオプション設定後、ビルドを実行することでロードモジュールを生成します。

ビルドを実行するには、図 5.14のように HEW で[ビルド->ビルド]または[ビルド->すべてをビルド]を選択してください。

5.14　分割リンク：カーネル環境側のビルド(cfg プロジェクト)

5.14.1　プロジェクトへの登録

表 5.13に、プロジェクトに登録するソースプログラムファイルを示します。サンプルのプロジェクトでは、これらのファイルがあらかじめ登録されています。

表5.13　プロジェクトに登録するソースプログラムファイル(cfg)

項番	ファイル名	内容	備考
1	kernel_cfg.c *1	カーネル環境側構築ファイル	必須
2	task.c *1	サンプルタスク	
3	アプリケーションファイル	－	

【注】 *1　sh*nnnn*¥src フォルダまたは sh*nnnn*¥project¥src フォルダにあります。

5.14.2　エンディアンの指定（HI7700/4、HI7750/4）

提供のサンプルプロジェクトでは、エンディアンの指定を前述の図 5.10のように HEW のビルドコンフィギュレーションで選択するように設定しています。

5.14.3　最適化リンケージエディタの設定

(1)　[Input]カテゴリの[Defines]

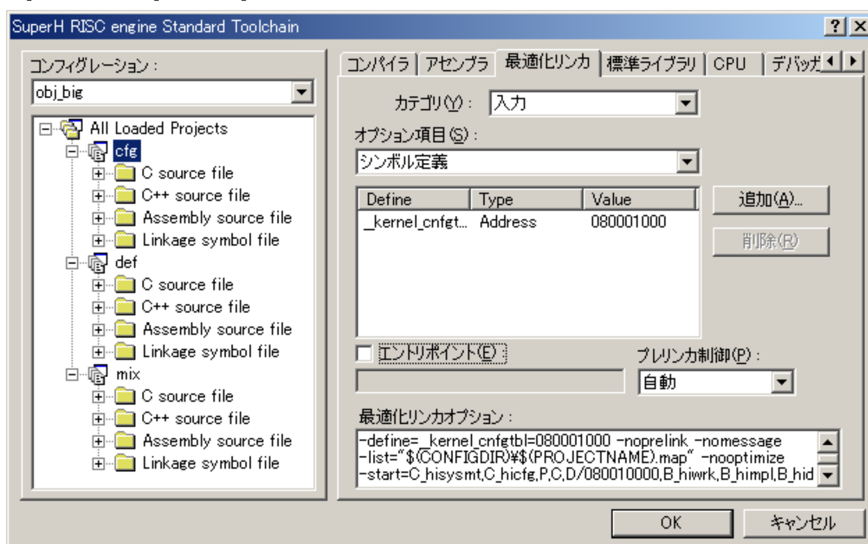


図5.18　[Input]カテゴリの[Defines]

以下のシンボルアドレスを定義します。

- __kernel_cnfgtbl：サービスコールインタフェース情報

サービスコールインタフェース情報はカーネルロードモジュール(def)に含まれるため、ここでサービスコールインタフェース情報の配置アドレスを強制的に定義する必要があります。__kernel_cnfgtbl は、カーネル側構築ファイル(kernel_def.c)中の C_hibase セクションの先頭アドレスです。

RENESAS

(2)　[Section]カテゴリ

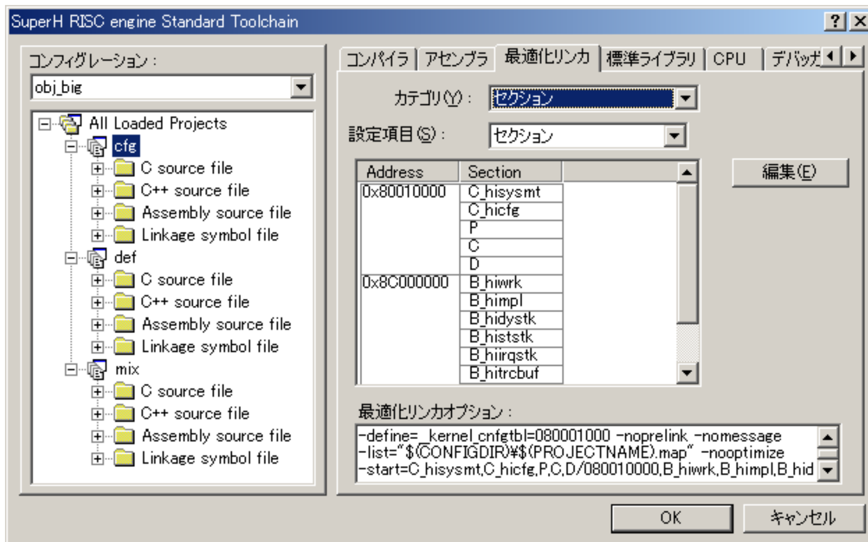


図5.19　[Section]カテゴリ

各セクションの配置アドレスを指定します。

ターゲットハードウェアにあわせて、入力ファイルに含まれる各セクションを指定し、配置アドレスを指定してください。

カーネルロードモジュール(def)生成時には、__kernel_sysmt（カーネル環境情報 C_hisysmt セクションの先頭アドレス）のアドレスを定義する必要があります。この定義アドレスと、ここでの C_hisysmt セクションの配置アドレスは、同じでなければなりません。

基本的には、入力ファイル中に含まれるすべてのセクションを漏れなく指定してください。指定されていないセクションが入力ファイル中に存在する場合、最適化リンケージエディタは最後に指定したセクションに続いてこれらのセクションを自動的に配置するため、ユーザが意図しない配置となる可能性があり、誤動作の原因となります。なおこの場合、最適化リンケージエディタは以下のようなウォーニングを出力します(P_Task1 セクションが指定されていない場合の例)。

```
L1120 (W) Section address is not assigned to "P_Task1"
```

逆に、指定したセクションが入力ファイル中に存在しない場合、最適化リンケージエディタは以下のようなウォーニングを表示します(C セクションが存在しない場合)が、結合は正常に行われます。

```
L1100 (W) Cannot find "C" specified in option "start"
```

提供時の設定では、P、C、D、B、R などのセクションについて、このウォーニングが表示される場合があります。これは、結合したアプリケーションオブジェクト内でそれらのセクションを使用していないためです。生成されるロードモジュールに何ら問題はありません。

5.14.4　ビルドの実行

プロジェクトへのアプリケーションファイルの登録、コンパイル、アセンブル、最適化リンケージエディタのオプション設定後、ビルドを実行することでロードモジュールを生成します。

ビルドを実行するには、図 5.14のように HEW で[ビルド->ビルド]または[ビルド->すべてをビルド]を選択してください。

5.15 アプリケーションロードモジュールの作成

コンフィギュレータに指定したアプリケーションの外部名(タスクのシンボルなど)を持たないアプリケーションファイルは、単独でロードモジュールを生成することができます。逆に、すべてのアプリケーションファイルを一括ロードモジュール、カーネルロードモジュール、カーネル環境ロードモジュールに含めてもかまいません。その場合、アプリケーションロードモジュールは必要ありません。

アプリケーションだけのロードモジュールを作成するには、HEW のマニュアルおよびオンラインヘルプを参考に、サンプルワークスペース hios.hws に新規プロジェクトを追加するか、または新規にワークスペースを作成してください。

リンク時には、以下に注意してください。

(1) __kernel_cnfgtbl アドレスの定義

以下のシンボルアドレスを定義します。

- __kernel_cnfgtbl : サービスコールインタフェース情報

サービスコールインタフェース情報はカーネルを含むロードモジュール(mix または def)に含まれるため、ここでサービスコールインタフェース情報の配置アドレスを強制的に定義する必要があります。__kernel_cnfgtbl は、カーネル側構築ファイル(kernel_def.c)中の C_hibase セクションの先頭アドレスです。

付録A　サービスコール一覧

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
タスク管理機能			
1	cre_tsk	ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk);	タスクの生成
	icre_tsk	ER ercd = icre_tsk(ID tskid, T_CTSK *pk_ctsk);	(ダイナミックスタック使用)
2	vscr_tsk	ER ercd = vscr_tsk(ID tskid, T_CTSK *pk_ctsk);	タスクの生成
	ivscr_tsk	ER ercd = ivscr_tsk(ID tskid, T_CTSK *pk_ctsk);	(スタティックスタック使用)
3	acre_tsk	ER_ID tskid = acre_tsk(T_CTSK *pk_ctsk);	タスクの生成
	iacre_tsk	ER_ID tskid = iacre_tsk(T_CTSK *pk_ctsk);	(ID 番号自動割付け)
4	del_tsk	ER ercd = del_tsk(ID tskid);	タスクの削除
5	act_tsk	ER ercd = act_tsk(ID tskid);	タスクの起動
	iact_tsk	ER ercd = iact_tsk(ID tskid);	
6	can_act	ER_UINT actcnt = can_act(ID tskid);	タスク起動要求のキャンセル
	ican_act	ER_UINT actcnt = ican_act(ID tskid);	
7	sta_tsk	ER ercd = sta_tsk(ID tskid, VP_INT stacd);	タスクの起動
	ista_tsk	ER ercd = ista_tsk(ID tskid, VP_INT stacd);	(起動コード指定)
8	ext_tsk	void ext_tsk(void);	自タスクの終了
9	exd_tsk	void exd_tsk(void);	自タスクの終了と削除
10	ter_tsk	ER ercd = ter_tsk(ID tskid);	タスクの強制終了
11	chg_pri	ER ercd = chg_pri(ID tskid, PRI tskpri);	タスク優先度の変更
	ichg_pri	ER ercd = ichg_pri(ID tskid, PRI tskpri);	
12	get_pri	ER ercd = get_pri(ID tskid, PRI *p_tskpri);	タスク優先度の参照
	iget_pri	ER ercd = iget_pri(ID tskid, PRI *p_tskpri);	
13	ref_tsk	ER ercd = ref_tsk(ID tskid, T_RTSK *pk_rtsk);	タスクの状態参照
	iref_tsk	ER ercd = iref_tsk(ID tskid, T_RTSK *pk_rtsk);	
14	ref_tst	ER ercd = ref_tst(ID tskid, T_RTST *pk_rtst);	タスクの状態参照(簡易版)
	iref_tst	ER ercd = iref_tst(ID tskid, T_RTST *pk_rtst);	
15	vchg_tmd	ER ercd = vchg_tmd(UINT tmd);	タスク実行モードの変更
タスク付属同期機能			
16	slp_tsk	ER ercd = slp_tsk(void);	起床待ち
17	tslp_tsk	ER ercd = tslp_tsk(TMO tmout);	同上(タイムアウト有)
18	wup_tsk	ER ercd = wup_tsk(ID tskid);	タスクの起床
	iwup_tsk	ER ercd = iwup_tsk(ID tskid);	
19	can_wup	ER_UINT wupcnt = can_wup(ID tskid);	タスク起床要求のキャンセル
	ican_wup	ER_UINT wupcnt = ican_wup(ID tskid);	
20	rel_wai	ER ercd = rel_wai(ID tskid);	待ち状態の強制解除
	irel_wai	ER ercd = irel_wai(ID tskid);	
21	sus_tsk	ER ercd = sus_tsk(ID tskid);	強制待ち状態への移行
	isus_tsk	ER ercd = isus_tsk(ID tskid);	
22	rsm_tsk	ER ercd = rsm_tsk(ID tskid);	強制待ち状態からの再開
	irsm_tsk	ER ercd = irsm_tsk(ID tskid);	
23	frsm_tsk	ER ercd = frsm_tsk(ID tskid);	強制待ち状態からの強制再開
	ifrsm_tsk	ER ercd = ifrsm_tsk(ID tskid);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
24	dly_tsk	ER ercd = dly_tsk(RELTIM dlytim);	タスクの遅延
25	vset_tfl	ER ercd = vset_tfl(ID tskid, UINT setptn);	タスク付属イベントフラグのセット
	ivset_tfl	ER ercd = ivset_tfl(ID tskid, UINT setptn);	
26	vclr_tfl	ER ercd = vclr_tfl(ID tskid, UINT clrptn);	タスク付属イベントフラグのクリア
	ivclr_tfl	ER ercd = ivclr_tfl(ID tskid, UINT clrptn);	
27	vwai_tfl	ER ercd = vwai_tfl(UINT waiptn, UINT *p_tflptn);	タスク付属イベントフラグ待ち
28	vpol_tfl	ER ercd = vpol_tfl(UINT waiptn, UINT *p_tflptn);	同上(ポーリング)
29	vtwai_tfl	ER ercd = vtwai_tfl(UINT waiptn, UINT *p_tflptn, TMO tmout);	同上(タイムアウト有)
タスク例外管理機能			
30	def_tex	ER ercd = def_tex (ID tskid, T_DTEX *pk_dtex);	タスク例外処理ルーチンの定義
	idef_tex	ER ercd = idef_tex (ID tskid, T_DTEX *pk_dtex);	
31	ras_tex	ER ercd = ras_tex(ID tskid, TEXPTN rasptn);	タスク例外処理の要求
	iras_tex	ER ercd = iras_tex(ID tskid, TEXPTN rasptn);	
32	dis_tex	ER ercd = dis_tex(void);	タスク例外処理の禁止
33	ena_tex	ER ercd = ena_tex(void);	タスク例外処理の許可
34	sns_tex	BOOL state =sns_tex(void);	タスク例外禁止状態の参照
35	ref_tex	ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex);	タスク例外処理の状態参照
	iref_tex	ER ercd = iref_tex(ID tskid, T_RTEX *pk_rtex);	
同期・通信(セマフォ)機能			
36	cre_sem	ER ercd = cre_sem(ID semid, T_CSEM *pk_csem);	セマフォの生成
	icre_sem	ER ercd = icre_sem(ID semid, T_CSEM *pk_csem);	
37	acre_sem	ER_ID semid = acre_sem(T_CSEM *pk_csem);	セマフォの生成 (ID 番号自動割付け)
	iacre_sem	ER_ID semid = iacre_sem(T_CSEM *pk_csem);	
38	del_sem	ER ercd = del_sem(ID semid);	セマフォの削除
39	sig_sem	ER ercd = sig_sem(ID semid);	セマフォ資源の返却
	isig_sem	ER ercd = isig_sem(ID semid);	
40	wai_sem	ER ercd = wai_sem(ID semid);	セマフォ資源の獲得
41	pol_sem	ER ercd = pol_sem(ID semid);	同上(ポーリング)
	ipol_sem	ER ercd = ipol_sem(ID semid);	
42	twai_sem	ER ercd = twai_sem(ID semid, TMO tmout);	同上(タイムアウト有)
43	ref_sem	ER ercd = ref_sem(ID semid, T_RSEM *pk_rsem);	セマフォの状態参照
	iref_sem	ER ercd = iref_sem(ID semid, T_RSEM *pk_rsem);	
同期・通信(イベントフラグ)機能			
44	cre_flg	ER ercd = cre_flg(ID flgid, T_CFLG *pk_cflg);	イベントフラグの生成
	icre_flg	ER ercd = icre_flg(ID flgid, T_CFLG *pk_cflg);	
45	acre_flg	ER_ID flgid = acre_flg(T_CFLG *pk_cflg);	イベントフラグの生成 (ID 番号自動割付け)
	iacre_flg	ER_ID flgid = iacre_flg(T_CFLG *pk_cflg);	
46	del_flg	ER ercd = del_flg(ID flgid);	イベントフラグの削除
47	set_flg	ER ercd = set_flg(ID flgid, FLGPTN setptn);	イベントフラグのセット
	iset_flg	ER ercd = iset_flg(ID flgid, FLGPTN setptn);	
48	clr_flg	ER ercd = clr_flg(ID flgid, FLGPTN clrptn);	イベントフラグのクリア
	iclr_flg	ER ercd = iclr_flg(ID flgid, FLGPTN clrptn);	
49	wai_flg	ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);	イベントフラグ待ち

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
50	pol_flg	ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);	同上(ポーリング)
	ipol_flg	ER ercd = ipol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);	
51	twai_flg	ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);	同上(タイムアウト有)
52	ref_flg	ER ercd = ref_flg(ID flgid, T_RFLG *pk_rflg);	イベントフラグの状態参照
	iref_flg	ER ercd = iref_flg(ID flgid, T_RFLG *pk_rflg);	
同期・通信(データキュー)機能			
53	cre_dtq	ER ercd = cre_dtq(ID dtqid, T_CDTQ *pk_cdtq);	データキューの生成
	icre_dtq	ER ercd = icre_dtq(ID dtqid, T_CDTQ *pk_cdtq);	
54	acre_dtq	ER_ID dtqid = acre_dtq(T_CDTQ *pk_cdtq);	データキューの生成 (ID 番号自動割付け)
	iacre_dtq	ER_ID dtqid = iacre_dtq(T_CDTQ *pk_cdtq);	
55	del_dtq	ER ercd = del_dtq(ID dtqid);	データキューの削除
56	snd_dtq	ER ercd = snd_dtq(ID dtqid, VP_INT data);	データキューへの送信
57	psnd_dtq	ER ercd = psnd_dtq(ID dtqid, VP_INT data);	同上(ポーリング)
	ipsnd_dtq	ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);	
58	tsnd_dtq	ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);	同上(タイムアウト有)
59	fsnd_dtq	ER ercd = fsnd_dtq(ID dtqid, VP_INT data);	データキューへの強制送信
	ifsnd_dtq	ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);	
60	rcv_dtq	ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data);	データキューからの受信
61	prcv_dtq	ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data);	同上(ポーリング)
62	trcv_dtq	ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);	同上(タイムアウト有)
63	ref_dtq	ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq);	データキューの状態参照
	iref_dtq	ER ercd = iref_dtq(ID dtqid, T_RDTQ *pk_rdtq);	
同期・通信(メールボックス)機能			
64	cre_mbx	ER ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx);	メールボックスの生成
	icre_mbx	ER ercd = icre_mbx(ID mbxid, T_CMBX *pk_cmbx);	
65	acre_mbx	ER_ID mbxid = acre_mbx(T_CMBX *pk_cmbx);	メールボックスの生成 (ID 番号自動割付け)
	iacre_mbx	ER_ID mbxid = iacre_mbx(T_CMBX *pk_cmbx);	
66	del_mbx	ER ercd = del_mbx(ID mbxid);	メールボックスの削除
67	snd_mbx	ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg);	メールボックスへの送信
	isnd_mbx	ER ercd = isnd_mbx(ID mbxid, T_MSG *pk_msg);	
68	rcv_mbx	ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);	メールボックスからの受信
69	prcv_mbx	ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg);	同上(ポーリング)
	iprcv_mbx	ER ercd = iprcv_mbx(ID mbxid, T_MSG **ppk_msg);	
70	trcv_mbx	ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);	同上(タイムアウト有)
71	ref_mbx	ER ercd = ref_mbx(ID mbxid, T_RMBX *pk_rmbx);	メールボックスの状態参照
	iref_mbx	ER ercd = iref_mbx(ID mbxid, T_RMBX *pk_rmbx);	
拡張同期・通信(ミューテックス)機能			
72	cre_mtx	ER ercd = cre_mtx(ID mtxid, T_CMTX *pk_cmtx);	ミューテックスの生成
73	acre_mtx	ER_ID mtxid = acre_mtx(T_CMTX *pk_cmtx);	ミューテックスの生成 (ID 番号自動割付け)
74	del_mtx	ER ercd = del_mtx(ID mtxid);	ミューテックスの削除
75	loc_mtx	ER ercd = loc_mtx(ID mtxid);	ミューテックスのロック
76	ploc_mtx	ER ercd = ploc_mtx(ID mtxid);	同上(ポーリング)

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
77	tloc_mtx	ER ercd = tloc_mtx(ID mtxid, TMO tmout);	同上(タイムアウト有)
78	unl_mtx	ER ercd = unl_mtx(ID mtxid);	ミューテックスのロック解除
79	ref_mtx	ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx);	ミューテックスの状態参照
拡張同期・通信(メッセージバッファ)機能			
80	cre_mbf	ER ercd = cre_mbf(ID mbfid, T_CMBF *pk_cmbf);	メッセージバッファの生成
	icre_mbf	ER ercd = icre_mbf(ID mbfid, T_CMBF *pk_cmbf);	
81	acre_mbf	ER_ID mbfid = acre_mbf(T_CMBF *pk_cmbf);	メッセージバッファの生成
	iacre_mbf	ER_ID mbfid = iacre_mbf(T_CMBF *pk_cmbf);	(ID 番号自動割付け)
82	del_mbf	ER ercd = del_mbf(ID mbfid);	メッセージバッファの削除
83	snd_mbf	ER ercd = snd_mbf(ID mbfid, VP msg, UINT msgsz);	メッセージバッファへの送信
84	psnd_mbf	ER ercd = psnd_mbf(ID mbfid, VP msg, UINT msgsz);	同上(ポーリング)
	ipsnd_mbf	ER ercd = ipsnd_mbf(ID mbfid, VP msg, UINT msgsz);	
85	tsnd_mbf	ER ercd = tsnd_mbf(ID mbfid, VP msg, UINT msgsz, TMO tmout);	同上(タイムアウト有)
86	rcv_mbf	ER_UINT msgsz = rcv_mbf(ID mbfid, VP msg);	メッセージバッファからの受信
87	prcv_mbf	ER_UINT msgsz = prcv_mbf(ID mbfid, VP msg);	同上(ポーリング)
88	trcv_mbf	ER_UINT msgsz = trcv_mbf(ID mbfid, VP msg, TMO tmout);	同上(タイムアウト有)
89	ref_mbf	ER ercd = ref_mbf(ID mbfid, T_RMBF *pk_rmbf);	メッセージバッファの状態参照
	iref_mbf	ER ercd = iref_mbf(ID mbfid, T_RMBF *pk_rmbf);	
メモリプール管理(固定長メモリプール)機能			
90	cre_mpf	ER ercd = cre_mpf(ID mpfid, T_CMPF *pk_cmpf);	固定長メモリプールの生成
	icre_mpf	ER ercd = icre_mpf(ID mpfid, T_CMPF *pk_cmpf);	
91	acre_mpf	ER_ID mpfid = acre_mpf(T_CMPF *pk_cmpf);	固定長メモリプールの生成
	iacre_mpf	ER_ID mpfid = iacre_mpf(T_CMPF *pk_cmpf);	(ID 番号自動割付け)
92	del_mpf	ER ercd = del_mpf(ID mpfid);	固定長メモリプールの削除
93	get_mpf	ER ercd = get_mpf(ID mpfid, VP *p_blk);	固定長メモリブロックの獲得
94	pget_mpf	ER ercd = pget_mpf(ID mpfid, VP *p_blk);	同上(ポーリング)
	ipget_mpf	ER ercd = ipget_mpf(ID mpfid, VP *p_blk);	
95	tget_mpf	ER ercd = tget_mpf(ID mpfid, VP *p_blk, TMO tmout);	同上(タイムアウト有)
96	rel_mpf	ER ercd = rel_mpf(ID mpfid, VP blk);	固定長メモリブロックの返却
	irel_mpf	ER ercd = irel_mpf(ID mpfid, VP blk);	
97	ref_mpf	ER ercd = ref_mpf(ID mpfid, T_RMPF *pk_rmpf);	固定長メモリプールの状態参照
	iref_mpf	ER ercd = iref_mpf(ID mpfid, T_RMPF *pk_rmpf);	
メモリプール管理(可変長メモリプール)機能			
98	cre_mpl	ER ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl);	可変長メモリプールの生成
	icre_mpl	ER ercd = icre_mpl(ID mplid, T_CMPL *pk_cmpl);	
99	acre_mpl	ER_ID mplid = acre_mpl(T_CMPL *pk_cmpl);	可変長メモリプールの生成
	iacre_mpl	ER_ID mplid = iacre_mpl(T_CMPL *pk_cmpl);	(ID 番号自動割付け)
100	del_mpl	ER ercd = del_mpl(ID mplid);	可変長メモリプールの削除
101	get_mpl	ER ercd = get_mpl(ID mplid, UINT blksz, VP *p_blk);	可変長メモリブロックの獲得
102	pget_mpl	ER ercd = pget_mpl(ID mplid, UINT blksz, VP *p_blk);	同上(ポーリング)
	ipget_mpl	ER ercd = ipget_mpl(ID mplid, UINT blksz, VP *p_blk);	
103	tget_mpl	ER ercd = tget_mpl(ID mplid, UINT blksz, VP *p_blk, TMO tmout);	同上(タイムアウト有)
104	rel_mpl	ER ercd = rel_mpl(ID mplid, VP blk);	可変長メモリブロックの返却
	irel_mpl	ER ercd = irel_mpl(ID mplid, VP blk);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
105	ref_mpl	ER ercd = ref_mpl(ID mplid, T_RMPL *pk_rmpl);	可変長メモリプールの状態参照
	iref_mpl	ER ercd = iref_mpl(ID mplid, T_RMPL *pk_rmpl);	
時間管理機能(システム時刻管理)			
106	set_tim	ER ercd = set_tim(SYSTIM *p_systim);	システム時刻の設定
	iset_tim	ER ercd = iset_tim(SYSTIM *p_systim);	
107	get_tim	ER ercd = get_tim(SYSTIM *p_systim);	システム時刻の参照
	iget_tim	ER ercd = iget_tim(SYSTIM *p_systim);	
108	isig_tim	CFG_TIMUSE を選択することで自動的に組み込まれます。	タイムティックの供給
時間管理機能(周期ハンドラ)			
109	cre_cyc	ER ercd = cre_cyc(ID cycid, T_CCYC *pk_ccyc);	周期ハンドラの生成
	icre_cyc	ER ercd = icre_cyc(ID cycid, T_CCYC *pk_ccyc);	
110	acre_cyc	ER_ID cycid = acre_cyc(T_CCYC *pk_ccyc);	周期ハンドラの生成
	iacre_cyc	ER_ID cycid = iacre_cyc(T_CCYC *pk_ccyc);	
111	del_cyc	ER ercd = del_cyc(ID cycid);	周期ハンドラの削除
112	sta_cyc	ER ercd = sta_cyc(ID cycid);	周期ハンドラの動作開始
	ista_cyc	ER ercd = ista_cyc(ID cycid);	
113	stp_cyc	ER ercd = stp_cyc(ID cycid);	周期ハンドラの動作停止
	istp_cyc	ER ercd = istp_cyc(ID cycid);	
114	ref_cyc	ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc);	周期ハンドラの状態参照
	iref_cyc	ER ercd = iref_cyc(ID cycid, T_RCYC *pk_rcyc);	
時間管理機能(アラームハンドラ)			
115	cre_alm	ER ercd = cre_alm(ID almid, T_CALM *pk_calm);	アラームハンドラの生成
	icre_alm	ER ercd = icre_alm(ID almid, T_CALM *pk_calm);	
116	acre_alm	ER_ID almid = acre_alm(T_CALM *pk_calm);	アラームハンドラの生成
	iacre_alm	ER_ID almid = iacre_alm(T_CALM *pk_calm);	
117	del_alm	ER ercd = del_alm(ID almid);	アラームハンドラの削除
118	sta_alm	ER ercd = sta_alm(ID almid, RELTIM almtim);	アラームハンドラの動作開始
	ista_alm	ER ercd = ista_alm(ID almid, RELTIM almtim);	
119	stp_alm	ER ercd = stp_alm(ID almid);	アラームハンドラの動作停止
	istp_alm	ER ercd = istp_alm(ID almid);	
120	ref_alm	ER ercd = ref_alm(ID almid, T_RALM *pk_ralm);	アラームハンドラの状態参照
	iref_alm	ER ercd = iref_alm(ID almid, T_RALM *pk_ralm);	
時間管理機能(オーバーランハンドラ)			
121	def_ovr	ER ercd = def_ovr(T_DOVR *pk_dovr);	オーバーランハンドラの定義
122	sta_ovr	ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim);	オーバーランハンドラの動作開始
	ista_ovr	ER ercd = ista_ovr(ID tskid, OVRTIM ovrtim);	
123	stp_ovr	ER ercd = stp_ovr(ID tskid);	オーバーランハンドラの動作停止
	istp_ovr	ER ercd = istp_ovr(ID tskid);	
124	ref_ovr	ER ercd = ref_ovr(ID tskid, T_ROVR *pk_rovr);	オーバーランハンドラの状態参照
	iref_ovr	ER ercd = iref_ovr(ID tskid, T_ROVR *pk_rovr);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
システム状態管理機能			
125	rot_rdq	ER ercd = rot_rdq(PRI tskpri);	タスクの優先順位の回転
	irot_rdq	ER ercd = irot_rdq(PRI tskpri);	
126	get_tid	ER ercd = get_tid(ID *p_tskid);	実行状態のタスク ID の参照
	iget_tid	ER ercd = iget_tid(ID *p_tskid);	
127	loc_cpu	ER ercd = loc_cpu(void);	CPU ロック状態への移行
	iloc_cpu	ER ercd = iloc_cpu(void);	
128	unl_cpu	ER ercd = unl_cpu(void);	CPU ロック状態の解除
	iunl_cpu	ER ercd = iunl_cpu(void);	
129	dis_dsp	ER ercd = dis_dsp(void);	ディスパッチの禁止
130	ena_dsp	ER ercd = ena_dsp(void);	ディスパッチの許可
131	sns_ctx	BOOL state = sns_ctx(void);	コンテキストの参照
132	sns_loc	BOOL state = sns_loc(void);	CPU ロック状態の参照
133	sns_dsp	BOOL state = sns_dsp(void);	ディスパッチ禁止状態の参照
134	sns_dpn	BOOL state = sns_dpn(void);	ディスパッチ保留状態の参照
135	vsta_knl	void vsta_knl(void);	カーネルの起動
	ivsta_knl	void ivsta_knl(void);	
136	vsys_dwn	void vsys_dwn(W type, ER ercd, VW inf1, VW inf2);	システムダウン
	ivsys_dwn	void ivsys_dwn(W type, ER ercd, VW inf1, VW inf2);	
137	vget_trc	ER ercd = vget_trc(VW para1, VW para2, VW para3, VW para4);	トレースの取得
	ivget_trc	ER ercd = ivget_trc(VW para1, VW para2, VW para3, VW para4);	
138	ivbgn_int	ER ercd = ivbgn_int(UINT dintno);	割込みハンドラの開始をトレースに取得
139	ivend_int	ER ercd = ivend_int(UINT dintno);	割込みハンドラの終了をトレースに取得
割込み管理機能			
140	def_inh	ER ercd = def_inh(INHNO inhno, T_DINH *pk_dinh);	割込みハンドラの定義
	idef_inh	ER ercd = idef_inh(INHNO inhno, T_DINH *pk_dinh);	
141	chg_ims	ER ercd = chg_ims(IMASK imask);	割込みマスクの変更
	ichg_ims	ER ercd = ichg_ims(IMASK imask);	
142	get_ims	ER ercd = get_ims(IMASK *p_imask);	割込みマスクの参照
	iget_ims	ER ercd = iget_ims(IMASK *p_imask);	
サービスコール管理機能			
143	def_svc	ER ercd = def_svc(FN fncd, T_DSVC *pk_dsvc);	拡張サービスコールルーチンの定義
	idef_svc	ER ercd = idef_svc(FN fncd, T_DSVC *pk_dsvc);	
144	cal_svc	ER_UINT ercd = cal_svc(FN fncd, ...);	拡張サービスコールの呼出し
	ical_svc	ER_UINT ercd = ical_svc(FN fncd, ...);	
システム構成管理機能			
145	def_exc	ER ercd = def_exc(EXCNO excno, T_DEXC *pk_dexc);	CPU 例外ハンドラの定義
	idef_exc	ER ercd = idef_exc(EXCNO excno, T_DEXC *pk_dexc);	
146	vdef_trp	ER ercd = vdef_trp(UINT dtrpno, T_DTRP *pk_dtrp);	CPU 例外ハンドラの定義 (TRAP 例外)
	ivdef_trp	ER ercd = ivdef_trp(UINT dtrpno, T_DTRP *pk_dtrp);	
147	ref_cfg	ER ercd = ref_cfg(T_RCFG *pk_rcfg);	コンフィギュレーション情報の参照
	iref_cfg	ER ercd = iref_cfg(T_RCFG *pk_rcfg);	
148	ref_ver	ER ercd = ref_ver(T_RVER *pk_rver);	バージョン情報の参照
	iref_ver	ER ercd = iref_ver(T_RVER *pk_rver);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
キャッシュサポート機能 [HI7700/4:SH-3, SH3-DSP **用**]			
149	vini_cac	void vini_cac(UW ccr_data, UW entnum, UW waynum);	**キャッシュの初期化**
	ivini_cac	void ivini_cac(UW ccr_data, UW entnum, UW waynum);	
150	vclr_cac	ER ercd = vclr_cac(VP clradr1, VP clradr2);	**キャッシュのクリア**
	ivclr_cac	ER ercd = ivclr_cac(VP clradr1, VP clradr2);	
151	vfls_cac	ER ercd = vfls_cac(VP flsadr1, VP flsadr2);	**キャッシュのフラッシュ**
	ivfls_cac	ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);	
152	vinv_cac	ER ercd = vinv_cac(void);	**キャッシュの無効化**
	ivinv_cac	ER ercd = ivinv_cac(void);	
キャッシュサポート機能 [HI7750/4:SH-4 **用**]			
153	vini_cac	void vini_cac(UW ccr_data);	**キャッシュの初期化**
	ivini_cac	void vini_cac(UW ccr_data);	
154	vclr_cac	ER ercd = vclr_cac(VP clradr1, VP clradr2);	**オペランドキャッシュのクリア**
	ivclr_cac	ER ercd = ivclr_cac(VP clradr1, VP clradr2);	
155	vfls_cac	ER ercd = vfls_cac(VP flsadr1, VP flsadr2);	**オペランドキャッシュのフラッシュ**
	ivfls_cac	ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);	
156	vinv_cac	ER ercd = vinv_cac(VP invadr1, VP invadr2);	**オペランドキャッシュの無効化**
	ivinv_cac	ER ercd = ivinv_cac(VP invadr1, VP invadr2);	
キャッシュサポート機能 [HI7700/4:SH4AL-DSP(**拡張機能なし**)**用**, HI7750/4:SH-4A(**拡張機能なし**)**用**]			
157	vini_cac	ER vini_cac(ATR cacatr);	**キャッシュの初期化**
	ivini_cac	ER ivini_cac(ATR cacatr);	
158	vclr_cac	ER ercd = vclr_cac(VP clradr1, VP clradr2, MODE mode);	**命令・オペランドキャッシュのクリア**
	ivclr_cac	ER ercd = ivclr_cac(VP clradr1, VP clradr2, MODE mode);	
159	vfls_cac	ER ercd = vfls_cac(VP flsadr1, VP flsadr2);	**オペランドキャッシュのフラッシュ**
	ivfls_cac	ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);	
160	vinv_cac	ER ercd = vinv_cac(VP invadr1, VP invadr2, MODE mode);	**命令・オペランドキャッシュの無効化**
	ivinv_cac	ER ercd = ivinv_cac(VP invadr1, VP invadr2, MODE mode);	
キャッシュサポート機能 [HI7700/4:SH4AL-DSP(**拡張機能あり**)**用**, HI7750/4:SH-4A(**拡張機能あり**)**用**]			
161	vini_cac	ER vini_cac(ATR cacatr);	**キャッシュの初期化**
	ivini_cac	ER ivini_cac(ATR cacatr);	
162	vclr_cac	ER ercd = vclr_cac(VP clradr1, VP clradr2, MODE mode);	**命令・オペランドキャッシュのクリア**
	ivclr_cac	ER ercd = ivclr_cac(VP clradr1, VP clradr2, MODE mode);	
163	vfls_cac	ER ercd = vfls_cac(VP flsadr1, VP flsadr2);	**オペランドキャッシュのフラッシュ**
	ivfls_cac	ER ercd = ivfls_cac(VP flsadr1, VP flsadr2);	
164	vinv_cac	ER ercd = vinv_cac(VP invadr1, VP invadr2, MODE mode);	**命令・オペランドキャッシュの無効化**
	ivinv_cac	ER ercd = ivinv_cac(VP invadr1, VP invadr2, MODE mode);	
DSP **スタンバイ制御機能** [HI7700/4]			
165	vchg_cop	ER_UINT oldatr = vchg_cop(ATR newatr);	TA_COP0 **属性の変更**

RENESAS

付録B　エラー一覧

B.1　サービスコールエラーコード一覧

表 B.1　サービスコールエラーコード一覧

エラーコード (ニーモニック)	エラーコード値		エラーチェック種別 *	エラー内容
E_OK	H'00000000	(D'0)	[k]	正常終了
E_NOSPT	H'fffffff7	(-D'9)	[p]	未サポート機能（機能が未登録）
E_RSFN	H'fffffff6	(-D'10)	[p]	サービスコールが組み込まれていない
E_RSATR	H'fffffff5	(-D'11)	[p]	予約属性（属性が不正）
E_PAR	H'ffffffef	(-D'17)	[p]/[k]	パラメータエラー
E_ID	H'ffffffee	(-D'18)	[p]	不正 ID 番号
E_CTX	H'ffffffe7	(-D'25)	[k]	コンテキストエラー
E_ILUSE	H'ffffffe4	(-D'28)	[k]	サービスコール不正使用
E_NOMEM	H'ffffffdf	(-D'33)	[k]	メモリ不足
E_NOID	H'ffffffde	(-D'34)	[k]	ID 番号不足
E_OBJ	H'ffffffd7	(-D'41)	[k]	オブジェクトの状態不正
E_NOEXS	H'ffffffd6	(-D'42)	[k]	オブジェクトが存在しない
E_QOVR	H'ffffffd5	(-D'43)	[k]	キューイングまたはネストのオーバフロー
E_RLWAI	H'ffffffcf	(-D'49)	[k]	待ち状態強制解除
E_TMOUT	H'ffffffce	(-D'50)	[k]	ポーリング失敗またはタイムアウト
E_DLT	H'ffffffcd	(-D'51)	[k]	待ちオブジェクトが削除された

【注】 [p]は、パラメータチェック機能(CFG_PARCHK)を選択した場合にのみチェックされます。[k]は、常にチェックされます。

B.2 システムダウン時の情報

システムダウンになると、システムダウンルーチンが呼び出されます。システムダウンルーチンには、表 B.2 に示す情報が渡されます。

表 B.2 システムダウンルーチンに渡される情報

項番	システムダウン要因	システムダウンルーチンに渡されるパラメータ			
		エラー種別 W type (R4)	エラーコード ER ercd (R5)	システムダウン情報 1 VW inf1 (R6)	システムダウン情報 2 VW inf2 (R7)
1	vsys_dwn, ivsys_dwn サービスコール	1〜H'7fffffff	vsys_dwn, ivsys_dwn サービスコールのパラメータ		
2	コンフィギュレータでの初期登録情報に誤りがあった場合	0	エラーコード	0:カーネル側 1:カーネル環境側	カーネル側またはカーネル環境側の何番目の登録でエラーとなったかを示す *2
3	非タスクコンテキストからの ext_tsk サービスコールでコンテキストエラーが発生	H'ffffffff(-1)	E_CTX (h'ffffffe7)	ext_tsk を呼び出したアドレス	不定
4	非タスクコンテキストからの exd_tsk サービスコールでコンテキストエラーが発生	H'fffffffe(-2)	E_CTX (h'ffffffe7)	exd_tsk を呼び出したアドレス	不定
5	未定義割込み、例外が発生	H'fffffff0(-16)	例外コードまたはベクタ番号 *1	例外発生時の PC	例外発生時の SR

【注】 *1 HI7700/4、HI7750/4 の場合は、割込み・例外・無条件トラップ発生時に、CPU が割込み事象レジスタ（INTEVT または INTEVT2）または例外事象レジスタ（EXPEVT）に設定する情報です。

*2 初期登録は、カーネル側が先にカーネル環境側が後に処理されます。カーネル側およびカーネル環境側での初期登録の処理順は以下の通りで、それぞれの中の順序は各ページでのリストに表示されている順序です。ただし、カーネル側の(1)〜(3)でシステムダウンが発生することはありえないため、初期登録のカウントには含めません。

(1)割込み,CPU 例外ハンドラページでの割込みハンドラの初期登録
(2)割込み,CPU 例外ハンドラページでの CPU 例外ハンドラの初期登録
(3)トラップ例外ハンドラページでのトラップ用 CPU 例外ハンドラの初期登録
(4)タスクページでのタスクおよびタスク例外処理ルーチンの初期登録
(5)セマフォページでのセマフォの初期登録
(6)イベントフラグページでのイベントフラグの初期登録
(7)データキューページでのデータキューの初期登録
(8)メールボックスページでのメールボックスの初期登録
(9)ミューテックスページでのミューテックスの初期登録
(10)メッセージバッファページでのメッセージバッファの初期登録
(11)固定長メモリプールページでの固定長メモリプールの初期登録
(12)可変長メモリプールページでの可変長メモリプールの初期登録
(13)周期ハンドラページでの周期ハンドラの初期登録
(14)アラームハンドラページでのアラームハンドラの初期登録
(15)オーバーランハンドラページでのオーバーランハンドラの初期登録
(16)拡張サービスコールページでの拡張サービスコールの初期登録

B.3 コンパイル時のエラー

B.3.1 異なる HI7000/4 シリーズ環境でのエラー

コンフィギュレータの生成ファイルを使って異なる HI7000/4 シリーズ環境で kernel_def.c および kernel_cfg.c をコンパイルすると、コンパイル時に以下のエラーメッセージが出力されます。下線部は、コンフィギュレータ生成ファイルの対象 OS を示します。

```
Unmatch HIOS(This file is designed for HI7750/4.)
```

B.3.2 最適化タイマドライバに関するエラー(HI7700/4)

最適化タイマドライバの定義ファイルkernel_def_opttmr_set.hの設定、およびコンフィギュレータの時間管理機能ページの設定について、kernel_def.cおよびkernel_cfg.cのコンパイル時に表B.3に示すエラーチェックが行われます。

表 B.3 最適化タイマドライバに関するエラー

エラーメッセージ	意味
#error directive: "Illegal CFG_TIMUSE"	コンフィギュレータで CFG_TIMUSE がチェックされていません。CFG_TIMUSE をチェックしてください。
#error directive: "Illegal hi_longticrate"	hi_longticrate が不正です。hi_longticrate には、2～0xff の整数定数を指定してください。
#error directive: "Illegal hi_pclock"	hi_pclock が不正です。hi_pclock には、0 以外の整数定数を指定してください。
#error directive: "Illegal CFG_TIMINTNO"	CFG_TIMINTNO が不正です。CFG_TIMINTNO には、0x400 を指定してください。
#error directive: "Illegal CFG_TIMINTLVL"	CFG_TIMINTLVL が不正です。CFG_TIMINTLVL には、CFG_KNLKMSKLVL(カーネル割込みマスクレベル)と同じ値を指定してください。 なお、このエラーは kernel_def.c のコンパイル時には検出されません。

B.3.3 DSP スタンバイ制御機能に関するエラー(HI7700/4)

DSPスタンバイ制御機能の定義ファイルkernel_def_dspstby_set.hの設定について、kernel_def.cおよびkernel_cfg.cのコンパイル時に表B.4に示すエラーチェックが行われます。

表 B.4 DSP スタンバイ制御機能に関するエラー

エラーメッセージ	意味
#error directive: "Illegal hi_cop_stby_adr"	hi_cop_stby_adr が不正です。hi_cop_stby_adr には、0 以外の整数定数を指定してください。
#error directive: "Illegal hi_cop_stby_bit"	hi_cop_stby_bit が不正です。hi_cop_stby_bit には、SH3-DSP の場合は 1～0xff の整数定数、SH4AL-DSP の場合は 1～0xffffffff の整数定数を指定してください。

付録C　作業領域サイズの算出

C.1　作業領域の内訳

各種作業領域は、柔軟なメモリ配置を可能とする為に表 C.1 に示す各種セクションに分割されています。リンク時に、これらのセクションを最適なアドレスに配置してください。

表 C.1　作業領域の内訳

項番	領域種別	セクション名	確保しているファイル
1	カーネル作業領域	B_hiwrk	kernel_cfg.c
2	CPU ベクタテーブル(HI7000/4 のみ)	B_hivct	
3	スタティックスタック領域	B_histstk	
4	ダイナミックスタック領域	B_hidystk	
5	割込みハンドラ,タイムイベントハンドタのスタック領域	B_hiirqstk	
6	メモリプール用領域	B_himpl	
7	DX 用ターゲットトレースバッファ領域	B_hitrcbuf	
8	メモリプール管理テーブル *	B_hicfg	
9	DX 用エミュレータトレース(ツールトレース)領域	B_hitrceml	
10	メモリプール管理テーブル *	B_hidef	kernel_def.c
11	アプリケーションで使用する作業領域	任意	任意

【注】　コンフィギュレータで CFG_MPFMANAGE をチェックして固定長メモリプールを生成した場合、および CFG_NEWMPL をチェックして VTA_UNFRAGMENT 属性の可変長メモリプールを生成した場合。

それぞれのセクションのサイズは、コンパイルリストから知ることができます。

(1)　カーネル作業領域（セクション名 B_hiwrk, B_hidef, B_hicfg）

カーネルが動作するために使用する領域で、タスク管理ブロック（TCB）、メッセージバッファ領域、カーネルのスタック領域などが含まれます。CFG_MAXTSKID などのオブジェクトの最大 ID、CFG_DTQSZ, CFG_MBFSZ などによってサイズが決まります。

(2)　CPU ベクタテーブル(HI7000/4 のみ)（セクション名 B_hivct）

コンフィギュレータで CFG_VCTRAM を選択した場合、CPU のベクタテーブルがこのセクションで生成されます。

(3)　スタティックスタック領域（セクション名 B_histstk）

コンフィギュレータで静的に定義、確保するスタティックスタック領域です。

(4)　ダイナミックスタック領域（セクション名 B_hidystk）

タスク生成時に動的に割り付けられるタスクスタック領域です。CFG_TSKSTKSZ によってサイズが決まります。

(5)　割込みハンドラ、タイムイベントハンドラのスタック領域（セクション名 B_hiirqstk）

割込みハンドラおよびタイムイベントハンドラが使用するスタック領域です。CFG_IRQSTKSZ と CFG_TMRSTKSZ によってサイズが決まります。

(6) メモリプール用領域（セクション名 B_himpl）

可変長メモリプール、固定長メモリプールとして使用される領域です。CFG_MPLSZ と CFG_MPFSZ によってサイズが決まります。

(7) DX 用トレースバッファ領域（セクション名 B_hitrcbuf）

CFG_TRCTYPE に「ターゲットトレース」を選択した場合のみ、CFG_TRCBUFSZ バイトの領域が確保されます。

(8) DX 用エミュレータトレース(ツールトレース)トレース領域（セクション名 B_hitrceml）

CFG_TRCTYPE に「エミュレータトレース(ツールトレース)」を選択した場合のみ、14 バイトが確保されます。

(9) アプリケーションで使用する作業領域

アプリケーションで使用する各種変数領域です。

C.2 スタックの分類

タスクやハンドラは、スタック領域としてそれぞれ専用の連続した領域を必要とします。ユーザは、タスクやハンドラの実行にどれだけのスタック使用量が必要かを見極め、個々のタスクやハンドラに割当てる必要があります。スタックのオーバフローはシステムの異常動作を引き起こすので、本節を参考にして十分なスタック領域を確保してください。スタックには、以下の種類があります。

(1) タスクスタック

タスクは、タスク ID 毎に異なるスタックを使用します。カーネルは、スケジューリング時にタスクのスタックを切り替えます。タスク例外処理ルーチンも、同じタスクのスタックを引き継いで使用します。タスクのスタックには、「スタティックスタック」と「ダイナミックスタック」があり、さらにアプリケーションで確保した領域をスタックとして使用することもできます。

タスクのスタックは、カーネルが切り替えます。したがってタスクではスタックを切り替えてはなりません。

(2) 割込みハンドラスタック

通常の割込み発生時には、カーネルが割込みハンドラ用のスタックに切り替えます。したがって割込みハンドラではスタックを切り替えてはなりません。

なお、タイムイベントハンドラは割込みハンドラのスタックを使用します。タイムイベントハンドラで、スタックを切り替えてはなりません。

(3) ダイレクト割込みハンドラスタック(HI7000/4)

HI7000/4 のダイレクト割込みでは、割込みハンドラでスタックを確保し、ハンドラの先頭でそのスタック切り替えてください。切り替えない場合には、割込みハンドラは割込み発生前に実行していたタスクのスタックを使うことになるため、そのタスクのスタックがオーバーフローする可能性があります。なお、NMI はダイレクト割込みハンドラとして定義する必要がありますが、NMI は再入する可能性があるため、NMI 割込みハンドラではスタック切り換えを行ってはなりません。NMI 割込みハンドラは NMI 発生前のスタックを使用することになるため、タスクやハンドラのスタックは NMI 割込みハンドラが消費するサイズを加味して確保する必要があります。

(4) カーネルスタック

カーネルが使用するスタックです。初期化ルーチンもカーネルスタックを使用します。初期化ルーチン内で、スタックを切り替えてはなりません。

(5)　カーネル起動前のスタック

CPU 初期化ルーチンなどカーネル起動前に実行されるプログラムのスタックは、カーネルの管理外です。このため、任意の領域をスタックとして使用することができます。パワーオンリセット時のスタックポインタは、HI7000/4 ではリセットベクタに、HI7700/4 と HI7750/4 では CPU 初期化ルーチンの先頭で設定する必要があります。

内蔵 RAM を持つマイコンでは、通常はリセット時のスタックを内蔵 RAM にしてください。内蔵 RAM を持たないマイコンでは、リセット直後の BSC(バスステートコントローラ)の状態ではリセット時のスタック（実装されている RAM）にアクセスできない場合があります。このような場合は、メモリが使用できるように BSC の設定が完了するまで、プログラムでスタックを使用しないのはもちろん、割込みや例外も起こさないようにする必要があります。これは、割込みや例外が発生するとスタックへのレジスタ保存が行われるためです。

C.3　スタック使用量の算出手順

タスクやハンドラのスタック使用量は、図 C.1 に従って算出し、それぞれ適切な部分にそのサイズを指定してください。

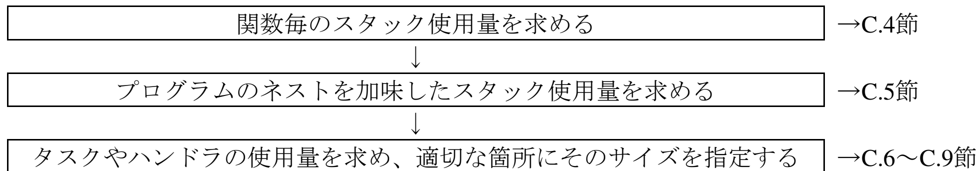


図 C.1　スタック使用量の算出手順

C.4　関数単体のスタック使用量

(1)　C 言語関数の場合

C 言語関数の場合は、関数起動時にスタックフレームをスタック上に確保します。スタックフレームは、関数の局所変数や関数呼出し時の引数領域として利用されます。このスタックフレームサイズは、C コンパイラが出力するコンパイルリスト上の「frame size」等から知ることができます。

以下に例を示します。

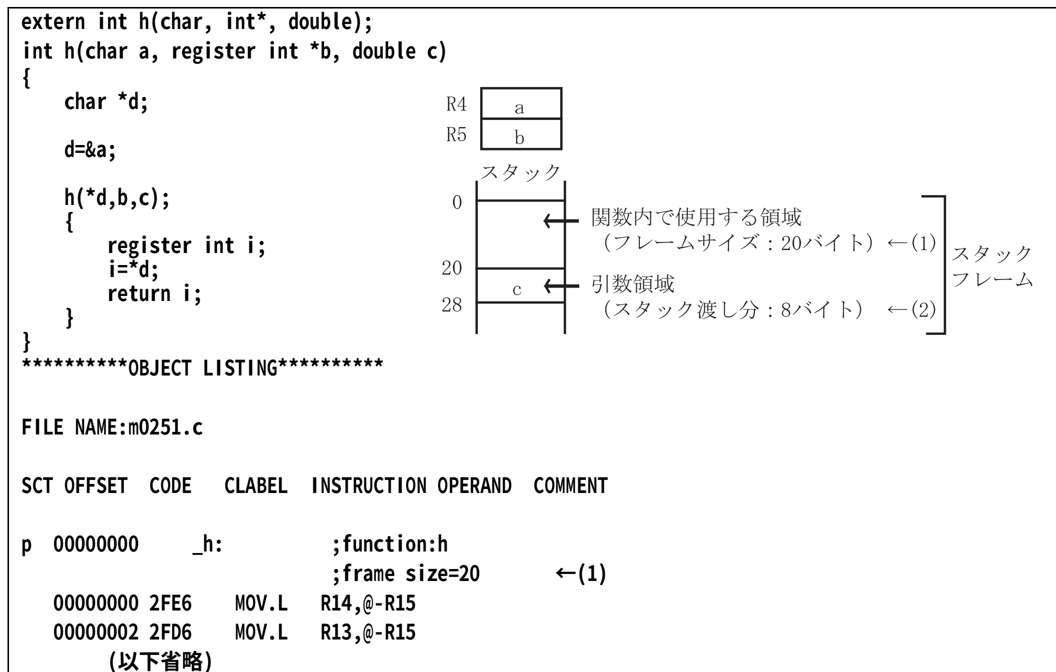


図 C.2　コンパイルリストとスタックの使用量

関数の使用するスタック領域サイズは(1) + (2)で示されるサイズで、図 C.2 の例では 28 バイトとなります。スタック上の引数領域に割り付けられる引数等、詳細については『SuperH™ RISC engine C/C++ コンパイラユーザーズマニュアル』を参照してください。

(2)　アセンブリ言語関数の場合

スタックプッシュ、ポップ（プリデクリメント、ポストインクリメントレジスタ間接形式）命令の使用状況を調査して、スタック使用量を算出してください。引数をスタックにプッシュして関数を呼び出す場合は、そのサイズも加算してください。

RENESAS

C.5 プログラムネストを加味したスタック使用量

以下のプログラム開始関数を基点に、プログラムネストを加味したスタック使用量を算出します。

- タスク
- 割込みハンドラ
- タイムイベントハンドラ
- 初期化ルーチン

プログラムのネストには、これらの開始関数から呼び出されるすべての関数はもちろん、以下のプログラムの呼び出しも含みます。

- 拡張サービスコールルーチン
- タスク例外処理ルーチン
- CPU 例外ハンドラ（TRAPA 命令例外含む）

そして、このネストのケース毎に C.4 節に従って求めた各々の関数が使用するサイズの総和を算出してください。その際、タスク例外処理ルーチン、CPU 例外ハンドラ（TRAPA 命令例外含む）がネストする場合は、ネストのたびに表 C.2 に示すサイズをさらに加算してください。

表 C.2 呼び出しルーチン、ハンドラの加算サイズ

項番	種別		加算サイズ（バイト）		
			HI7000/4	HI7700/4	HI7750/4
1	タスク例外処理ルーチン		144	144 *2	152
2		TA_COP0 属性あり *1	56	56	−
3		TA_COP1 属性あり	72	−	64
4		TA_COP2 属性あり	−	−	64
5		CFG_NEWMPL 選択あり	28	28	28
6	CPU 例外ハンドラ（TRAPA 命令例外含む）		44	44	48

【注】 *1 HI7700/4 では、vchg_cop サービスコールで TA_COP0 属性を動的に設定または解除することができます。vchg_cop サービスコールで TA_COP0 属性を設定する場合は、この加算値が必要です。

*2 最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、168 となります。

例

HI7700/4 におけるタスクのプログラムネストを加味したスタック使用量の算出例を示します。なお、ここでは図 C.3 のプログラムネスト状況を想定します。

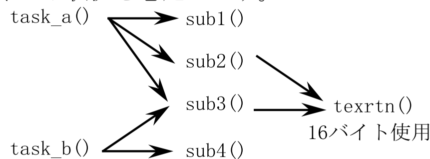


図 C.3 プログラムネスト状況

個々の関数のスタック使用量は、表 C.3 の通りであるとします。

表 C.3　個々の関数のスタック使用量

項番	関数名	使用量(バイト)	備考
1	task_a	56	タスク A の開始関数, TA_COP0 属性なし
2	task_b	40	タスク B の開始関数, TA_COP0 属性なし
3	sub1	88	task_a のサブルーチン
4	sub2	8	task_a のサブルーチン
5	sub3	24	共通サブルーチン
6	sub4	12	task_b のサブルーチン
7	texrtn	144+ 16 = 160	タスク例外処理ルーチン開始関数、TA_COP0 属性なし、CFG_NEWMPL 選択無し

呼び出し経路を考慮したタスク A, B の使用量は、表 C.4 のようになります。

表 C.4　呼び出し経路を考慮した使用量

タスク	呼び出し経路	使用量(バイト)
タスク A	task_a<56 バイト>→sub1<88 バイト>	144
	task_a<56 バイト>→sub2<8 バイト>→texrtn<160>	224
	task_a<56 バイト>→sub3<24 バイト>→texrtn<160>	240 [最大]
タスク B	task_b<40 バイト>→sub3<24 バイト>→texrtn<160>	224 [最大]
	task_b<56 バイト>→sub4<12 バイト>	68

RENESAS

C.6 タスクのスタック

(1) 個々のタスクのスタック使用量

個々のタスクのスタック使用量は、C.5 節に従って求めたサイズを表 C.5 に代入することで算出します。

ダイナミックスタックを使用するタスクの場合は、タスク生成時(cre_tsk, acre_tsk サービスコール、コンフィギュレータでの生成)に表 C.5 によって算出した値をスタックサイズとして指定してください。

スタティックスタックを使用するタスクの場合は、コンフィギュレータで表 C.5 によって算出したサイズをスタティックスタックとして確保してください。

表 C.5 個々のタスクのスタック使用量

項番	項目	使用量（バイト）		
		HI7000/4	HI7700/4	HI7750/4
1	C.4節、C.5節で求めたサイズ			
2	必須分	140	184 *4	196
3	TA_COP0 属性 *3	56	56	−
4	TA_COP1 属性	72	−	64
5	TA_COP2 属性	−	−	64
6	スタティックスタックを使用	8	8	8
7	CFG_TRACE 選択あり	24	24	24
8	CFG_NEWMPL 選択あり	28	28	28
9	多重割込み加算値	*1	−	−
10	NMI 使用時の加算値	*2	−	−
合計				

【注】 *1 (1) CFG_REGBANK が NOBANK または NOTUSE または SELECT の場合

(1.1)CFG_DIRECT が選択ありの場合

12×CFG_UPPINTNST +20×CFG_LOWINTNST

(1.2) CFG_DIRECT 選択なしの場合

12×CFG_UPPINTNST + 24×CFG_LOWINTNST + 20

ただし、CFG_LOWINTNST=0 の場合は、下線部を 0 として計算してください。

(2)CFG_REGBANK が ALL の場合

8×CFG_UPPINTNST + 16×CFG_LOWINTNST

ただし、UBC 割込みを使用する場合は以下のサイズを加算してください。

- UBC 割込みをダイレクト割込みとして使用する場合：4 バイト
- UBC 割込みを通常割込みとして使用する場合：28 バイト

*2 (C.4 節,C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 8)×NMI のネスト数

なお、「NMI のネスト数」は、ネストしない場合を 1 と数えてください。

*3 HI7700/4 では、vchg_cop サービスコールで TA_COP0 属性を動的に設定または解除することができます。vchg_cop サービスコールで TA_COP0 属性を設定する場合は、この加算値が必要です

*4 最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、208 となります。

(2) スタック領域の確保方法

ダイナミックスタック領域は、コンフィギュレータで CFG_TSKSTKSZ にサイズを指定することで

自動的に確保されます。CFG_TSKSTKSZ には、次式の値以上のサイズを指定してください。

CFG_TSKSTKSZ = Σ(ダイナミックスタックを使用するタスクのスタック使用量 + 16) + 28

C.7 割込みハンドラのスタック

(1) 個々の割込みハンドラのスタック使用量

個々の割込みハンドラのスタック使用量は、C.5 節に従って求めたサイズを表 C.6 に代入することで算出します。なお、NMI 割込みハンドラのスタック使用量は、表 C.6 の項番 1 のみで算出してください。この値は、C.6～C.9 節での計算に使用します。

表 C.6 個々の割込みハンドラのスタック使用量

項番	項目	使用量（バイト） HI7000/4 ダイレクト割込みハンドラ	使用量（バイト） HI7000/4 通常の割込みハンドラ	使用量（バイト） HI7700/4	使用量（バイト） HI7750/4
1	C.4節、C.5節で求めたサイズ				
2	割込みハンドラからサービスコールを呼び出す	140	140	140 *3	144
3	CFG_TRACE 選択あり	24	24	24	24
4	CFG_NEWMPL 選択あり	28	28	28	28
5	多重割込み加算値	*1	—	—	—
6	NMI 使用時の加算値	*2	—	—	—
合計					

【注】 *1 ここでは、以下の記号を使用します。

uppintnst：CFG_KNLMSKLVL より高く、かつ自割込みレベルより高いダイレクト割込みのネスト数

lowintnst：CFG_KNLMSKLVL 以下で、かつ自割込みレベルより高い割込みのネスト数

(1) CFG_REGBANK が NOBANK または NOTUSE または SELECT の場合

(1.1)CFG_DIRECT が選択ありの場合

12×uppintnst +20×lowintnst

(1.2) CFG_DIRECT 選択なしの場合

12×uppintnst + 24×lowintnst + 20

ただし、lowintnst =0 の場合は、下線部を 0 として計算してください。

(2)CFG_REGBANK が ALL の場合

8×uppintnst+ 16×lowintnst

ただし、UBC 割込みを使用する場合は以下のサイズを加算してください。

- UBC 割込みをダイレクト割込みとして使用する場合：4 バイト
- UBC 割込みを通常割込みとして使用する場合：28 バイト

*2 (C.4 節,C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 8)×NMI のネスト数

なお、「NMI のネスト数」は、ネストしない場合を 1 と数えてください。

*3 最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、164 となります。

(2) スタック領域の確保方法

(a) ダイレクト割込みハンドラ（HI7000/4）の場合

同一割込みレベルのハンドラが同時に実行することは無いので、同一割込みレベルの割込みハンドラの中でスタックを最も多く使用するハンドラの使用量を、その割込みレベルのハンドラ用スタックとして確保してください。そして、割込みハンドラの先頭でそのスタックに切り替えてください。割込みハンドラでスタックを切り替える方法は、「4.8.2 ダイレクト割込みハンドラ（HI7000/4）」を参照してください。もちろん、同一割込みレベルのハンドラでスタックを共有せずに、別々のスタックにしても構いません。

ただし、NMI は再入する可能性があるため、NMI 割込みハンドラ専用のスタックを持つことはできません。NMI 割込みハンドラは NMI 発生時点のスタックを使用するので、C.6～C.9 節で NMI 割込みハンドラが使用するサイズを加算します。

(b) 通常の割込みハンドラの場合

通常の割込みハンドラはすべて、同じ割込みハンドラスタックを使用します。割込みハンドラスタックは、コンフィギュレータで CFG_IRQSTKSZ にサイズを指定することで自動的に確保されます。CFG_IRQSTKSZ には、次式の値以上のサイズを指定してください。

- HI7000/4 の場合

  (1) CFG_DIRECTが選択ありの場合

  この場合は、通常割込みハンドラは存在しないため、割込みハンドラスタックは確保されません。

  (2) CFG_DIRECTが選択なしの場合

  (a)CFG_REGBANKがNOBANKまたはNOTUSEまたはSELECTの場合

  CFG_IRQSTKSZ = Σ(各割込みレベルで最も多く使用するハンドラの使用量) + 4
  ＋12×CFG_UPPINTNST + 24×(CFG_LOWINTNST - 1) + 20
  ＋(C.4節,C.5節にしたがって求めたNMI割込みハンドラの使用量+ 8)×NMIネスト数

  ただし、CFG_LOWINTNST<=1の場合は、下線部を0として計算してください。

  (b)CFG_REGBANKがALLの場合

  CFG_IRQSTKSZ = Σ(各割込みレベルで最も多く使用するハンドラの使用量) + 4
  ＋8×CFG_UPPINTNST+16×(CFG_LOWINTNST-1)
  ＋(C.4節,C.5節にしたがって求めたNMI割込みハンドラの使用量+ 8)×NMIネスト数

  ただし、CFG_LOWINTNST<=1の場合は、下線部を0として計算してください。

  また、UBC割込みを使用する場合は以下のサイズを加算してください。

  ・UBC割込みをダイレクト割込みとして使用する場合：4バイト
  ・UBC割込みを通常割込みとして使用する場合：28バイト

- HI7700/4 の場合

  CFG_IRQSTKSZ = Σ(各割込みレベルで最も多く使用するハンドラの使用量) + 4
  ＋44×(NMIを除くシステム内の全割込みレベル数－1)
  ＋(C.4節,C.5節にしたがって求めたNMI割込みハンドラの使用量 ＋44)×NMIネスト数

- HI7750/4 の場合

  CFG_IRQSTKSZ = Σ(各割込みレベルで最も多く使用するハンドラの使用量) + 4
  ＋48×(NMIを除くシステム内の全割込みレベル数－1)
  ＋(C.4節,C.5節にしたがって求めたNMI割込みハンドラの使用量 ＋48)×NMIネスト数

なお、「NMI ネスト数」はネストしない場合を 1 と数えてください。

RENESAS

C.8 タイムイベントハンドラ、タイマ割込みルーチンのスタック

個々のタイムイベントハンドラおよびタイマ割込みルーチン(_kernel_tmrint())のスタック使用量を C.4節、C.5節に従って算出し、表 C.7 に代入してください。そして、算出される値の中で最大値を CFG_TMRSTKSZ に指定してください。

なお、CFG_ACTION を選択した場合は、表 C.7 を以下の条件で計算してください。

- C.4節、C.5節で求めたサイズ：32
- サービスコールを呼び出す：はい

タイムイベントハンドラを全く使用せず、CFG_ACTION も選択してしない場合は、表 C.7 を以下の条件で計算してください。

- C.4節、C.5節で求めたサイズ：0
- サービスコールを呼び出す：いいえ

表 C.7　タイムイベントハンドラ、タイマ割込みルーチンのスタック使用量

項番	項目	使用量（バイト）		
		HI7000/4	HI7700/4	HI7750/4
1	C.4節、C.5節で求めたサイズ			
2	必須分	(1)CFG_REGBANK 選択なしの場合 144 (2)CFG_REGBANK 選択ありの場合 68	140　*3	144
3	CFG_NEWMPL 選択あり	28	28	28
4	サービスコールを呼び出す	140	140	144
5	CFG_TRACE 選択あり	24	－	－
6	多重割込み加算値	*1	－	－
7	NMI 使用時の加算値	*2	－	－
	合計			

【注】 *1　ここでは、以下の記号を使用します。

uppintnst：CFG_KNLMSKLVL より高く、かつ CFG_TIMINTLV より高いダイレクト割込みのネスト数

lowintnst：CFG_KNLMSKLVL 以下で、かつ CFG_TIMINTLV より高い割込みのネスト数

(1) CFG_REGBANK が NOBANK または NOTUSE または SELECT の場合

(1.1)CFG_DIRECT が選択ありの場合

12×uppintnst +20×lowintnst

(1.2) CFG_DIRECT 選択なしの場合

12×uppintnst + <u>24×lowintnst + 20</u>

ただし、lowintnst =0 の場合は、下線部を 0 として計算してください。

(2)CFG_REGBANK が ALL の場合

8×uppintnst+ 16×lowintnst

ただし、UBC 割込みを使用する場合は以下のサイズを加算してください。

・UBC 割込みをダイレクト割込みとして使用する場合：4 バイト

・UBC 割込みを通常割込みとして使用する場合：28 バイト

*2　(C.4 節,C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 8)×NMI ネスト数

なお、「NMI ネスト数」は、ネストしない場合を 1 と数えてください。

*3　最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、164 となります。

C.9 初期化ルーチンのスタック

個々の初期化ルーチンのスタック使用量は、C.5 節に従って求めたサイズを表 C.8 に代入することで算出します。コンフィギュレータで各初期化ルーチンを定義する際には、ここで算出したサイズを指定してください。

実際には、初期化ルーチンはシリアルに実行するので、各初期化ルーチンのスタック使用量の最大サイズが確保されます。

表 C.8 個々の初期化ルーチンのスタック使用量

項番	項目		使用量（バイト）		
			HI7000/4	HI7700/4	HI7750/4
1	C.4節、C.5節で求めたサイズ				
2	サービスコールを呼び出す		140	140 *2	144
3		CFG_TRACE 選択あり	24	24	24
4		CFG_NEWMPL 選択あり	28	28	28
5	NMI 使用時の加算値		*1	–	–
合計					

【注】 *1 (C.4 節,C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 8)×NMI ネスト数
なお、「NMI ネスト数」は、ネストしない場合を 1 と数えてください。
*2 最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、164 となります。

C.10 タイマ初期化ルーチンのスタック

タイマ初期化ルーチンで(_kernel_tmrini())は、使用できるスタックサイズの上限が決まっています。上限値は、以下の通りです。

- HI7000/4：252 バイト
- HI7700/4：208 バイト
- HI7750/4：204 バイト

表 C.9 を元に算出されるサイズが上限値を超える場合は、そのサイズのスタック領域を別途確保し、切り替えて使用してください。

表 C.9 タイマ初期化ルーチンのスタック使用量

項番	項目		使用量（バイト）		
			HI7000/4	HI7700/4	HI7750/4
1	C.4節、C.5節で求めたサイズ				
2	サービスコールを呼び出す		140	140 *2	144
3		CFG_TRACE 選択あり	24	24	24
4		CFG_NEWMPL 選択あり	28	28	28
5	NMI 使用時の加算値		*1	–	–
合計					

【注】 *1 (C.4 節,C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 8)×NMI ネスト数
なお、「NMI ネスト数」は、ネストしない場合を 1 と数えてください。
*2 最適化タイマドライバまたは DSP スタンバイ制御機能を組み込んでいる場合は、164 となります。

付録D　タイマドライバ

D.1　概要

カーネルの時間管理機能(CFG_TIMUSE を選択)を使用するには、タイマドライバが必要です。
タイマドライバには、標準タイマドライバと最適化タイマドライバの 2 種類があります。

- 標準タイマドライバ
  標準タイマドライバは、ユーザが作成してカーネルとリンクする必要があります。HI7000/4 シリーズでは、いくつかのマイコン用のサンプルを提供しています。

- 最適化タイマドライバ
  最適化タイマドライバはHI7700/4でのみサポートされています。標準タイマドライバに対して、割込み発生回数が削減されるという特徴があります。最適化タイマドライバはカーネルに内蔵されており、ユーザが作成することはできません。

最適化タイマドライバを使用する場合は、「付録 E 最適化タイマドライバ(HI7700/4)」を参照してください。以降では、標準タイマドライバの仕様について解説します。

D.2　標準タイマドライバ

標準タイマドライバは、タイマ初期化ルーチンとタイマ割込みルーチンから構成されています。タイマ割込みルーチンは、カーネルのタイマ割込みハンドラ_kernel_isig_tim( )から呼び出され、割込み要因をクリアします。また、タイマ初期化ルーチンは、初期化ルーチンとして実行します。

D.2.1　時間管理機能の組み込み方法

カーネルの時間管理機能を使用するには、以下の操作が必要です。

(1)　コンフィギュレータで「時間管理機能ページ」を設定する

ここでは、以下のカーネル構成定数を定義します。

- TIC_NUME：タイムティック周期の分子(CFG_TICNUME)
- TIC_DENO：タイムティック周期の分母(CFG_TICDENO)
- TIM_LVL：タイマ割込みレベル(CFG_TIMINTLVL)

タイムティックが供給される周期時間は、TIC_NUME/TIC_DENO[ms]となります。これによって、サービスコールで指定する時間パラメータの精度が決まります。例えば、tslp_tsk(10)を行った場合、タイムアウトするまでの時間は TIC_NUME=3, TIC_DENO=1 なら 12ms～15ms となりますが、TIC_NUME=1, TIC_DENO=2 なら 10～10.5ms となります。

なお、TIC_NUME, TIC_DENO の少なくとも一方は 1 を指定しなければなりません。

また、TIC_DENO に 1 より大きな値を指定した場合は、TMO, RELTIM, OVRTIM 型のパラメータに指定可能な最大値は、(その型で表現できる最大値)/TIC_DENO に制限されます。

コンフィギュレータの時間管理機能ページを設定することで、自動的に以下が行われます。

- カーネル側の初期化ルーチンとして、_kernel_tmrini()が登録される。
- タイマ割込みハンドラとして、カーネルの isig_tim 処理モジュールが定義される。

(2) タイマドライバを作成する

以下の 2 つの C 言語関数を作成します。これらの名前は固定です。

- タイマ初期化ルーチン：_kernel_tmrini()
- タイマ割込みルーチン：_kernel_tmrint()

タイマ初期化ルーチンは、初期化ルーチンとして作成します。「4.10 タイムイベントハンドラ、初期化ルーチン」を参照してください。なお、タイマ初期化ルーチンは CFG_TIMUSE をチェックすることで自動的に初期化ルーチンとして登録されます。

タイマ初期化ルーチンでは、kernel_macro.h に定義されている TIC_NUME(タイムティック周期の分子)と TIC_DENO(タイムティック周期の分母)にしたがってタイマデバイスのカウンタレジスタなどを初期化し、同じく kernel_macro.h に定義されている TIM_LVL(タイマ割込みレベル)にしたがってタイマ割込みレベルを設定してください。

タイマ割込みが発生すると、まずカーネルの isig_tim 処理モジュールが割込みハンドラとして起動されます。そして、isig_tim 処理モジュールはタイマ割込みルーチン_kernel_tmrint()を呼び出します。タイマ割込みルーチンでは、割込み要因フラグをクリアしてください。

タイマ割込みルーチンは、図 D.1 のように通常の C 言語関数として記述します。タイマ割込みルーチンは、割込みハンドラの一部として動作します。

```
#inlcude "kernel.h"
void _kernel_tmrint(void)     ←関数名は固定です。
{
    /* ルーチンの処理 */
}
```

図 D.1　タイマ割込みルーチンの C 言語記述例

タイマ割込みルーチンのレジスタ仕様規約は、通常の割込みハンドラと同じです。「4.8.1 通常の割込みハンドラ」を参照してください。

タイマ初期化ルーチン、タイマ割込みルーチンで DSP を使用する場合は、「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」を参照してください。

タイマ初期化ルーチン、タイマ割込みルーチンで FPU を使用する場合は、「G.2 非タスクコンテキスト(通常の割込みハンドラ、ダイレクト割込みハンドラ、CPU 例外ハンドラ、タイムイベントハンドラ、初期化ルーチン)」を参照してください。

(3) タイマドライバをリンクする

タイマドライバは、カーネルとリンクする必要があります。

D.2.2　サンプルタイマドライバ

サンプルタイマドライバは、ヘッダファイルとソースファイルの２つから構成されています。

(1)　デバイスヘッダファイル（ファイル名 ：***nnnn***_tmrdrv.h）
　レジスタのI/Oアドレスや初期値など、タイマデバイスに依存した情報を定義しています。通常は、変更する必要はありません。

(2)　ヘッダファイル（ファイル名 ：***nnnn***_tmrdef.h）
　タイマドライバの動作を決定付ける情報(クロック周波数など)を定義しています。

(3)　ソースプログラムファイル（ファイル名 ：***nnnn***_tmrdrv.c）
- タイマ初期化ルーチン（関数 ：void _kernel_tmrini(void)）
- タイマ割込みルーチン（関数 ：void _kernel_tmrint(void)）

タイマ割込み周期時間(T)は、以下の式で表されます。

T＝｛（1／PCLOCK）×DIV｝×N

ここで、それぞれの記号の意味は以下の通りです。

- PCLOCK：タイマモジュールへ供給されるクロック周波数
- DIV：タイマモジュールのレジスタ設定による分周比
- N：周期時間に相当するクロックカウント数

一般にタイマモジュールのカウンタレジスタに n を設定すると、n+1 回のカウントが行われるため、

n = N - 1

となります。

HI7000/4 シリーズで提供するサンプルタイマドライバでは、上記の計算方法にしたがって n を算出し、タイマモジュールのレジスタに設定します。サンプルタイマドライバでは、上式中の記号を以下のように対応させています。

T：CFG_TICNUME/CFG_TICDENO [ミリ秒]
PCLOCK：　***nnnn***_tmrdef.h で定義している PCLOCK [Hz]
DIV：HI7000/4 では、***nnnn***_tmrdrv.c 中で定義しています。
　HI7700/4, HI7750/4 では、PCLOCK に応じて自動選択するようになっています。
nnnnnn_tmrdrv.c で、上記計算式を COUNTER に定義しています。

なお、COUNTER の計算結果がタイマカウンタレジスタのサイズを超える場合、周期は期待した通りになりません。COUNTER の計算結果がレジスタのサイズを超えないかを必ず確認してください。

また、COUNTER の計算結果が整数とならない場合、整数に丸められますので、周期 T は期待した時間よりもわずかに短くなることになります。

表 D.1 に、本マニュアル作成時点の各製品の最新バージョンで提供しているサンプルタイマドライバファイルと、サンプルタイマドライバが想定しているタイマモジュールへのクロックソースを示します。その他の詳細な設定は、ヘッダファイルの内容を参照してください。

表 D. 1　サンプルタイマドライバファイル一覧とクロックソース

項番	カーネル	*nnnn*	対象マイコンと 対象内蔵タイマモジュール		想定するタイマモジュールのクロックソース
1	HI7000/4 V.2.02	7011	SH7011, SH7018	CMT	システムクロック(φ)=20MHz
2		703x	SH7020, SH7021, SH7032, SH7034	ITU	システムクロック (φ)=20MHz
3		704x	SH7040, SH7041, SH7042, SH7043, SH7044, SH7045, SH7014, SH7016, SH7017	CMT	システムクロック(φ)=28MHz
4		7046	SH7046, SH7047, SH7048, SH7049, SH7144, SH7145, SH7148	CMT	システムクロック(φ)=40MHz
5		7050	SH7050, SH7051	CMT	システムクロック(φ)=20MHz
6		7052	SH7052, SH7053, SH7054	CMT	周辺クロック(φ)=20MHz
7		7065	SH7065	CMT	周辺クロック(Pφ)=30MHz
8		7604	SH7604	FRT	システムクロック(φ)=28.7MHz
9		7615	SH7615, SH7616	FRT	周辺クロック(Pφ)=30MHz
10		7618	SH7618	CMT	周辺クロック(Pφ)=12.5MHz
11		72060	SH72060	CMT	周辺クロック(Pφ)=24MHz
12	HI7700/4 V.2.02	7707	SH7707	TMU	周辺クロック(Pφ)=30MHz
13		7708	SH7708, SH7708R, SH7708S	TMU	周辺クロック(Pφ)=15MHz
14		7709	SH7709	TMU	周辺クロック(Pφ)=30MHz
15		7709a	SH7709A, SH7709S, SH7706	TMU	周辺クロック(Pφ)=33.34MHz
16		7729	SH7729, SH7729R, SH7727	TMU	周辺クロック(Pφ)=33.34MHz
17		7290	SH7290, SH7294, SH7300	TMU	周辺クロック(Pφ)=19.8MHz
18		7641	SH7641	CMT	周辺クロック(Pφ)=25 MHz
19		7318	SH7318	TMU	周辺クロック(Pφ)=27MHz
20		7343	SH7343	TMU	周辺クロック(Pφ)=27MHz
21	HI7750/4 V.2.02	7750	SH7750, SH7750S, SH7750R	TMU	周辺クロック(Pφ)=50MHz
22		7751	SH7751, SH7751R	TMU	周辺クロック(Pφ)=41.6MHz
23		7760	SH7760	TMU	周辺クロック(Pφ)= 33.34MHz
24		7770	SH7770	TMU	周辺クロック(Pφ)= 50MHz
25		7785	SH7785	TMU	周辺クロック(Pφ)= 50MHz

サンプルタイマドライバの各種動作条件は、ヘッダファイルに定義されています。必要に応じて、ヘッダファイルを修正してください。ただし、タイマ割込み周期時間を変更する場合は、ヘッダファイルの設定を変更するのではなく、コンフィギュレータでタイムティックの周期(CFG_TICNUME, CFG_TICDENO)の設定を変更してください。

付録E　最適化タイマドライバ(HI7700/4)

E.1　概要

従来の標準タイマドライバは、サービスコールの時間精度(CFG_TICNUME／CFG_TICDENO[ms])と同じ周期でタイマ割込みを発生させていました。最適化タイマドライバを使用すると、従来と同じサービスコールの時間精度を維持したまま、割込み頻度が低減されます。これによって、以下の効果が見込まれます。

・マイコンのスリープモード中に発生するタイマ割込み頻度が減少し、省電力効果が高まります。

・タイマ割込み頻度が減少するため、タイマ割込み処理による CPU 占有率が低減され、システムのスループットが向上します。あるいは、低減分だけより省電力モードの時間を延ばすことができます。

最適化タイマドライバは、標準タイマドライバとは異なり、カーネル内部に組み込まれています。ユーザが独自に最適化タイマドライバを作成することはできません。

また、本機能は特に既存アプリケーションプログラムの修正を必要としません。

E.2　動作

図 E.1 に、時間精度(CFG_TICNUME／CFG_TICDENO)が 1[ms]の場合の標準タイマドライバを使用した場合と最適化タイマドライバを使用した場合の動作例を示します。

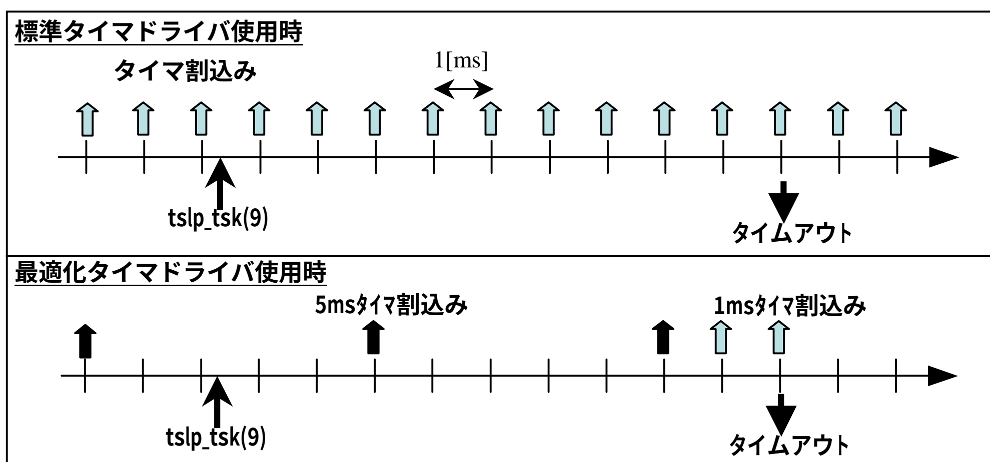


図 E.1　動作例

図 E.1 に示すように、最適化タイマドライバでは 1[ms]と 5[ms]の２つの TMU タイマチャネルを使用します。1[ms]を「高精度周期」、5[ms]を「粗精度周期」と呼びます。高精度周期は、コンフィギュレータで指定する CFG_TICNUME/CFG_TICDENO[ms]で算出される時間です。また、粗精度周期は、高精度周期の整数倍の時間で、ユーザが静的に設定することができます。

最適化タイマドライバ使用時、カーネルは適時以下の状況を調査します。

- タイムアウト指定のサービスコール(txxx_yyy)による待ちタスク
- dly_tsk サービスコールによる待ちタスク
- 周期ハンドラ
- アラームハンドラ

そして、高精度周期割込みが必要かどうかを判断し、その結果に応じて高精度周期割込みのイネーブル／ディスエーブルを制御します。

一方、粗精度周期割込みは常にイネーブルです。

また、図 E.1 には表現されていませんが、オーバーランハンドラの監視用にさらにもう 1 つの TMU チャネルを使用します。このチャネルのタイマ割込みは、タスクが上限プロセッサ時間を使いきったときにのみ発生します。

図 E.2 に、本機能による効果イメージを示します。本機能を使用することでタイマ割込み頻度が低減するため、標準タイマドライバ使用時に比べて以下の効果があります。

- 早くスリープモードに移行できる。(タイマ割込み処理による CPU 消費量が少ない)
- タイマ割込みによるスリープモードの解除の頻度が少ない

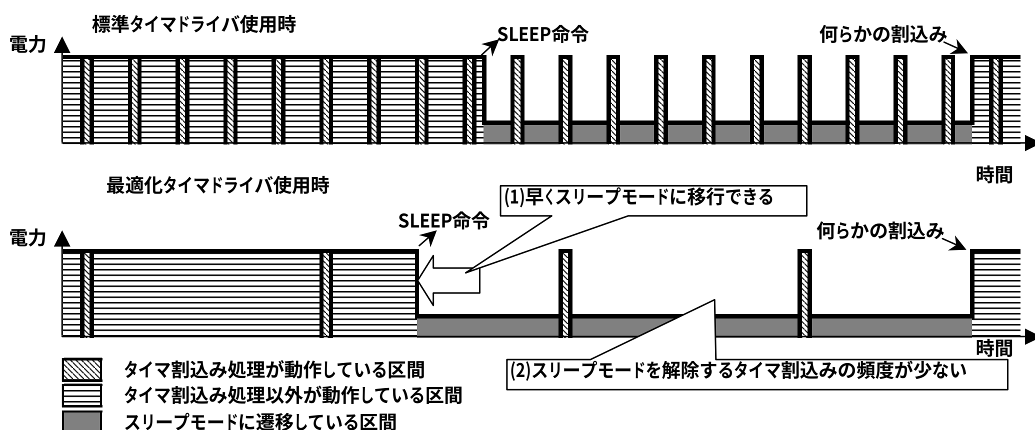


図 E.2 最適化タイマドライバによる効果イメージ

E.3 利用可能なマイコン

最適化タイマドライバはマイコン内蔵の TMU を使用しますが、TMU を内蔵するすべてのマイコンで利用できるわけではありません。以下に、本マニュアル作成時点で最適化タイマドライバを利用可能なマイコンを示します。最新の情報については、製品添付のリリースノートを参照してください。

SH7630, SH7290, SH7294, SH7300, SH7660, SH7318, SH7343

RENESAS

E.4 ハードウェア初期化処理

カーネル起動時に、以下の最適化タイマドライバの初期化処理が行われます。

(1) TMUのモジュールスタンバイ状態を解除します。

(2) チャネル0,1,2の初期設定を行います。

(3) 割込みコントローラに対して、チャネル0,1,2の割込みレベルを設定します。

ただし、def_ovr サービスコールが組み込まれていない場合は、(2), (3)のチャネル 2 に関する設定は行いません。

なお、最適化タイマドライバが、TMU をモジュールスタンバイに移行させることはありません。

E.5 標準タイマドライバとの相違

表 E.1 に、標準タイマドライバを使用した場合と最適化タイマドライバを使用した場合の相違を示します。

表 E.1 標準タイマドライバと最適化タイマドライバの相違

	標準タイマドライバ	最適化タイマドライバ
サービスコールの時間精度	CFG_TICNUME／CFG_TICDENO[ms]	同左。 (ただし、オーバーランハンドラについては、常に 1[ms])
必要なハードウェアタイマ	任意のタイマ 1 チャネル(サンプル標準タイマドライバでは TMU のチャネル 0 のみを使用)	TMU チャネル 0,1,2。ただし、def_ovr を組み込まない場合は、チャネル 2 は未使用。
タイマ割込みレベル	1〜カーネル割込みマスクレベル (CFG_KNLMSKLVL)の間のユーザ任意の値を CFG_TIMINTLVL に設定します。	カーネル割込みマスクレベルに限定
ユーザによるドライバ作成	可能	不可能

E.6 組み込み方法

E.6.1 概要

本節では、標準的な構築方法に対する追加の情報を説明します。標準的な構築方法については、「5. コンフィギュレーション」を参照してください。

表 E.2 にタイマドライバの組み込み方法の概要を示します。

表 E.2 タイマドライバの組み込み方法

項目		タイマドライバを組み込まない	標準タイマドライバを組み込む	最適化タイマドライバを組み込む
定義ファイルの作成(E.6.2節)		不要	不要	必要
コンフィギュレータ	CFG_TIMUSE	チェックしない	チェックする	チェックする
	E.6.3記載の留意事項	不要	不要	必要
kernel_sys.h の修正(E.6.4節)		不要	不要	必要
リンクするタイマドライバ		なし	標準ﾀｲﾏﾄﾞﾗｲﾊﾞ	なし(最適化ﾀｲﾏﾄﾞﾗｲﾊﾞはｶｰﾈﾙﾗｲﾌﾞﾗﾘ内にあります)

E.6.2 定義ファイル kernel_def_opttmr_set.h の作成

kernel_def_opttmr_set.h では、以下のような最適化タイマドライバの設定情報を定義します。

- hi_longticrate ... 粗精度周期÷高精度周期の比
- hi_pclock … 使用するマイコンの周辺クロック(Pφ)

このファイルは、kernel_def.c, kernel_cfg.c からインクルードされます。

(1) hi_longticrate

書式：#define hi_longticrate <設定値>

粗精度周期÷高精度周期で計算される値を整数定数で指定してください。指定可能な範囲は、2～0xff です。これ以外を指定した場合には、kernel_def.c, kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal hi_longticrate"
```

効果的な設定値は、システムでの時間管理機能の利用状況(頻度や指定する時間)に大きく依存します。一般的には 5～50 程度が適切です。

(2) hi_pclock

書式：#define hi_pclock <設定値>

使用するマイコンの周辺クロック(Pφ)値を、0 以外の Hz 単位の整数定数を指定してください。これ以外を指定した場合には、kernel_def.c, kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal hi_pclock"
```

E.6.3 コンフィギュレータでの留意事項

コンフィギュレータの[時間管理機能]ページでの設定項目は、以下のように設定してください。

(1) タイマ割込み番号(CFG_TIMINTNO)

0x400(TMU のチャネル 0 割込み)を指定してください。これ以外を指定すると、kernel_def.c, kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal CFG_TIMINTNO"
```

最適化タイマドライバでは、以下の割込みを使用します。

・0x400 ... TMU のチャネル 0
・0x420 ... TMU のチャネル 1
・0x440 ... TMU のチャネル 2

これらの割込み番号には、割込みハンドラを定義しないでください。

ただし、オーバーランハンドラを使用しない場合(def_ovr サービスコールが選択されていない場合)には、0x440(チャネル 2)は未定義となります。この場合、チャネル 2 はユーザ側で利用可能です。

(2) タイマ割込みレベル(CFG_TIMINTLVL)

カーネル割込みマスクレベル(CFG_KNLMSKLVL)と同じ値を指定してください。これ以外を指定すると、kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal CFG_TIMINTLVL"
```

(3) タイムティック周期(CFG_TICNUME, CFG_TICDENO)

タイムティック周期が高精度周期時間になります。

E.6.4 kernel_sys.h の修正

hisys¥kernel_sys.h の先頭付近に以下のステートメントを追加してください。これにより、前述の定義ファイルの設定内容が取り込まれるようになります。

```
#ifndef _HIOS_KERNEL_SYS_H
#define _HIOS_KERNEL_SYS_H
#define OPTTMR    /* ←追加 */
```

E.7 使用するカーネルライブラリ

使用するカーネルライブラリについては、「5.9 カーネルライブラリ」を参照してください。

付録F　DSP スタンバイ制御機能(HI7700/4)

F.1　概要

本機能は、タスクおよびタスク例外処理ルーチンの TA_COP0 属性(DSP 演算を行う)にハードウェアモジュール(DSP, X/Y メモリ)を関連付け、TA_COP0 属性の指定が無いタスクおよびタスク例外処理ルーチンを実行する際に、カーネルが自動的に関連するハードウェアモジュールをマイコンが持つモジュールスタンバイ状態にすることで省電力化を図ります(図 F.1 (b))。TA_COP0 属性に関連付けるハードウェアモジュールとして、以下のいずれかを静的に選択することができます。

- DSP のみ
- X/Y メモリのみ
- DSP + X/Y メモリ

また、TA_COP0 属性を動的に変更する vchg_cop サービスコールをサポートしています。アプリケーションの修正は必要になりますが、vchg_cop を活用すればさらにこれらのハードウェアリソースをモジュールスタンバイにする期間を長くすることが可能となります(図 F.1 (c))。

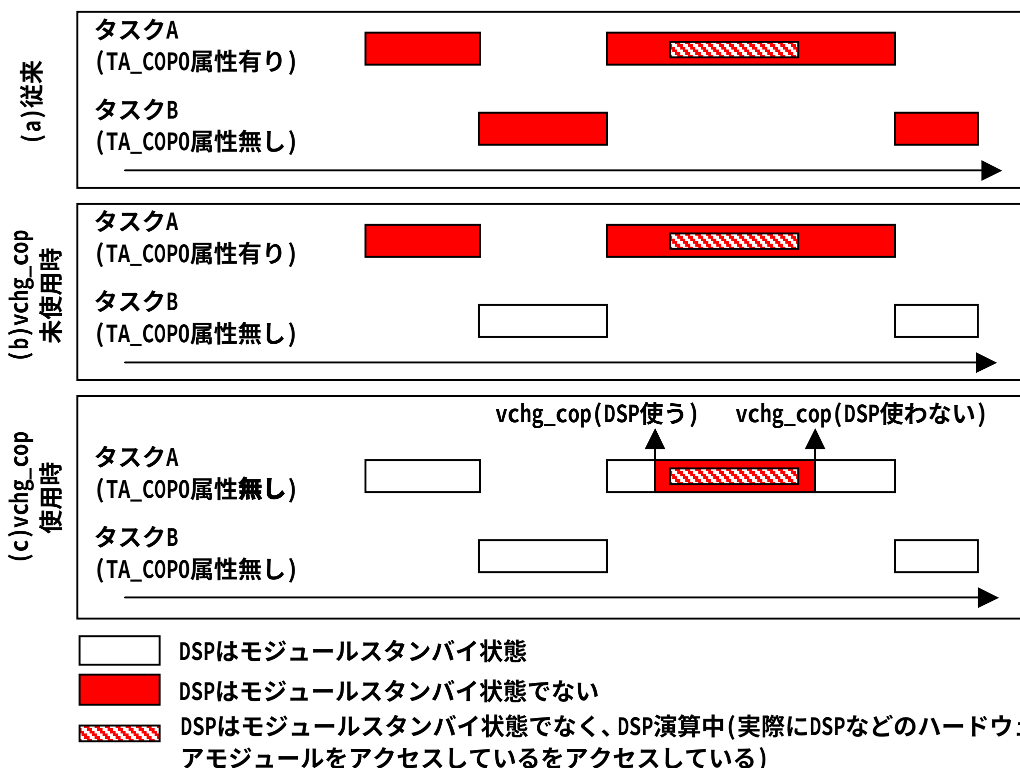


図 F.1　動作概要

F.2　利用可能なマイコン

本機能は、DSP を内蔵し、かつ DSP, X/Y メモリのいずれかあるいは両方のモジュールスタンバイ機能を持つマイコンでのみ使用できます。本マニュアル作成時点では、以下のマイコンがこの仕様を満たしています。

SH7727, SH7729R, SH7290, SH7294, SH7300, SH7641, SH7660, SH7710, SH7318 , SH7343

F.3　各プログラム起動時のモジュールスタンバイ状態

本機能を組み込んだ場合、TA_COP0 属性に関連付けられたモジュール(DSP, X/Y メモリ)の各プログラム開始時のモジュールスタンバイ状態は、表 F.1 に示すようになります。

表 F.1 で「不定」となっているプログラムでは、基本的には TA_COP0 属性に関連付けられたモジュールをアクセスしないように作成することが推奨されます。そのモジュールを使用するには、プログラムの起動時にアプリケーション側でそれらのモジュールスタンバイ状態を保存してからスタンバイを解除し、終了前にモジュールスタンバイ状態を元に戻す必要があります。

表 F.1　各プログラム起動時のモジュールスタンバイ状態

プログラム	モジュールスタンバイ状態
タスク	TA_COP0 属性がある場合：非スタンバイ TA_COP0 属性が無い場合：スタンバイ
タスク例外処理ルーチン	TA_COP0 属性がある場合：非スタンバイ TA_COP0 属性が無い場合：スタンバイ
拡張サービスコールルーチン	呼び出し前と同じ
割込みハンドラ	不定
CPU 例外ハンドラ	不定
タイムイベントハンドラ	不定
初期化ルーチン	不定

また、カーネルアイドリング時(実行可能状態のタスクが存在しないとき)には、TA_COP0 属性に関連付けられたモジュールはスタンバイ状態になります。

F.4　TA_COP0 属性変更サービスコール(vchg_cop)

■C 言語 API

```
ER_UINT oldatr = vchg_cop(ATR newatr);
```

■パラメータ

ATR	newatr	R4	変更後の属性

■リターンパラメータ

ER_UINT	oldatr	R0	変更前の属性(正の値または 0)またはエラーコード

■エラーコード

E_RSATR	[p]	予約属性(newatr が不正)
E_CTX	[k]	非タスクコンテキストからの発行

■機能

発行元のタスクの TA_COP0 属性を、newatr に変更します。
newatr には、以下を指定できます。

```
newatr := TA_COP0 | TA_NULL
```

- TA_COP0（H'00000100）DSP を使用
- TA_NULL（H'00000000）DSP を使用しない

oldatr には、本サービスコール発行前に TA_COP0 属性が指定されていたかどうかが返ります。
TA_COP0 ... 変更前は、TA_COP0 属性が指定されていた
TA_NULL ... 変更前は、TA_COP0 属性が指定されていなかった

発行元がタスクの場合、タスクが終了するとタスクの属性はタスク生成時の状態に戻ります。
発行元がタスク例外処理ルーチンの場合は、タスク例外処理ルーチンの TA_COP0 属性が変更されます。この場合、タスク本体の TA_COP0 属性は変更されません。また、タスク例外処理ルーチンが終了すると、タスク例外処理ルーチンの属性はタスク例外処理ルーチン定義時の状態に戻ります。

TA_COP0 属性が無い状態で TA_COP0 を指定した場合は、DSR レジスタが 0 に初期化されます。その他の DSP レジスタの値は不定です。

本サービスコールは、頻繁に発行して木目細かに ON/OFF するとそれだけオーバヘッドも増えるため、ある程度まとまった DSP 演算の単位で ON/OFF する使い方が推奨されます。
なお、本サービスコールは HI7700/4 独自機能です。また、本サービスコールはサービスコールトレース機能の対象外です。

F.5　組み込み方法

F.5.1　概要

本節では、標準的な構築方法に対する追加の情報を説明します。標準的な構築方法については、「5. コンフィギュレーション」を参照してください。

表 F.2 に DSP スタンバイ制御機能の組み込み方法の概要を示します。

表 F.2　DSP スタンバイ制御機能の組み込み方法

項目	DSP ｽﾀﾝﾊﾞｲ制御機能を組み込まない	DSP ｽﾀﾝﾊﾞｲ制御機能を組み込む
定義ファイルの作成(F.5.2節)	不要	必要
kernel_sys.h の修正(F.5.3節)	不要	必要

なお、vchg_cop サービスコールを組み込むかどうかの選択項目はありません。DSP スタンバイ制御機能を組み込んだ場合は、常に vchg_cop サービスコールも組み込まれます。

F.5.2　定義ファイル kernel_def_dspstby_set.h の作成

kernel_def_dspstby_set.h では、以下のような情報を定義します。

- hi_cop_stby_adr ... モジュールスタンバイ制御レジスタのアドレス
- hi_cop_stby_bit ... モジュールスタンバイ制御レジスタのビット位置

カーネルは、TA_COP0 属性のタスクおよびタスク例外処理ルーチンを実行するときに、アドレス hi_cop_stby_adr に対して、SH3-DSP の場合はアクセスサイズ 8 ビットで、SH4AL-DSP の場合はアクセスサイズ 32 ビットで、hi_cop_stby_bit で示されるビットを 0 にします。すなわち、hi_cop_stby_bit で指定されたモジュールを非スタンバイ状態にします。逆に、カーネルは TA_COP0 属性でないタスクおよびタスク例外処理ルーチンを実行するときに、アドレス hi_cop_stby_adr に対して、SH3-DSP の場合はアクセスサイズ 8 ビットで、SH4AL-DSP の場合はアクセスサイズ 32 ビットで、hi_cop_stby_bit で示されるビットを 1 にします。すなわち、hi_cop_stby_bit で指定されたモジュールをスタンバイ状態にします。

kernel_def_dspstbyr_set.h は、kernel_def.c, kernel_cfg.c からインクルードされます。

(1)　hi_cop_stby_adr

書式：#define hi_cop_stby_adr　<設定値>

DSP、X/Y メモリのモジュールスタンバイを制御するレジスタのアドレスを 0 以外の整数定数で指定してください。これ以外を指定した場合には、kernel_def.c, kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal hi_cop_stby_adr"
```

(2) hi_cop_stby_bit

書式：#define hi_cop_stby_bit <設定値>

hi_cop_stby_adr で指定したレジスタで、本機能の制御対象とするモジュールに該当するビットを整数定数で指定してください。指定可能な範囲は、SH3-DSP マイコンを使用する場合は 1～0xff、SH4AL-DSP マイコンを使用する場合は 1～0xffffffff です。これ以外を指定した場合には、kernel_def.c, kernel_cfg.c のコンパイル時に、以下のコンパイルエラーメッセージが表示されます。

```
#error directive: "Illegal hi_cop_stby_bit"
```

この指定によって、TA_COP0 属性に関連付けるハードウェアモジュールが決まります。

表 F.3 に、本マニュアル作成時点で存在する各マイコンにおける具体的な指定値を示します。

表 F.3 各マイコンでの指定値

マイコン	hi_cop_stby_adr	hi_cop_stby_bit		
		DSP のみ	X/Y メモリのみ	DSP + X/Y メモリ
SH7290, SH7294, SH7300, SH7641, SH7660, SH7710	0xa415ff88	0x10	0x1	0x11
SH7727, SH7729R	0xffffff88	(指定不可)	0x80	(指定不可)
SH7729	使用不可			
SH7318, SH7343	0xa4150030	(指定不可)	0x4000000	(指定不可)

F.5.3 kernel_sys.h の修正

hisys¥kernel_sys.h の先頭付近に以下のステートメントを追加してください。これにより、前述の定義ファイルの設定内容が取り込まれるようになります。

```
#ifndef _HIOS_KERNEL_SYS_H
#define _HIOS_KERNEL_SYS_H
#define DSPSTBY     /* ←追加 */
```

F.6 使用するカーネルライブラリ

使用するカーネルライブラリについては、「5.9 カーネルライブラリ」を参照してください。

F.7　注意事項

X/Y メモリは比較的電力を多く消費するため、X/Y メモリを本機能の対象に含めるとそれだけ省電力効果も高まりますが、以下のように DMA 転送に関する注意事項があります。

TA_COP0 属性のタスク A が X/Y メモリをソース、あるいはデスティネーションとする DMA 転送をスタートさせた場合、その転送中に TA_COP0 属性でない別のタスク B にスイッチすると、その時点でカーネルは X/Y メモリをモジュールスタンバイにするため、DMA 転送が正常に行われなくなります。

これは、X/Y メモリのスタンバイ制御がタスクの実行と同期して行われることに対し、DMA による X/Y メモリへのアクセスはソフトウェア動作とは非同期に行われるためです。

この問題への対処としては、以下が考えられます。

(1)X/Y メモリを本機能の制御対象にしない。

(2)X/Y メモリに対する DMA 転送は行わない。

(3)DMA 転送が終了するまでタスクスイッチを起こさないようにする。

付録G　SH2A-FPU, SH-4, SH-4A における FPU に関する注意

G.1　タスク、タスク例外処理ルーチン

G.1.1　FPSCR の初期化

タスクおよびタスク例外処理ルーチン起動時の FPSCR の値は、以下のようになっています。

- SH-4, SH-4A：H'00040001(FR=0, SZ=0, PR=0, DN=1, RM=B'01)
- SH2A-FPU : TA_COP1 属性有りの場合は H'00040001(SZ=0, PR=0, DN=1, RM=B'01)、TA_COP1 属性無しの場合は不定

浮動小数点演算を行う場合で、以下のいずれかのコンパイルオプションを指定している場合は、タスクおよびタスク例外処理ルーチンのエントリ関数の先頭で FPSCR を初期化する必要があります。

具体的には、「G.4.3 コンパイラによる扱い」に記載の「関数開始時点でコンパイラが想定している値」に FPSCR を初期化する必要があります。

- fpu=double
- denormalize=on
- round=nearest

図 G.1 に、下記の条件におけるタスクの FPSCR 初期化例を以下に示します。

[コンパイラオプション]

- CPU = SH4A
- FPU = DOUBLE
- Denormalize = ON
- Round = Nearest

```
#include <machine.h>   /* 組み込み関数 set_fpscr()を使用するためにインクルード */
#define INI_FPSCR 0x00080000 /* FPSCR 初期値(FR=0, SZ=0, PR=1, DN=0, SZ=0, RM=B’00) */
#pragma noregsave(Task)
void Task(VP_INT exinf)
{
    set_fpscr(INI_FPSCR);    /* タスクの先頭で FPSCR を初期化 */
    /* タスクの処理 */
    ext_tsk();
}
```

図 G.1　タスクにおける FPSCR 初期化例

G.1.2　TA_COP1, TA_COP2 属性

TA_COP1, TA_COP2 属性は、表 G.1 に示すように指定してください。

表 G.1　TA_COP1, TA_COP2 属性の指定

条件	SH2A-FPU	SH-4, SH-4A
マトリックス演算などを行う場合(両方の FPU レジスタバンクを使用する場合)	−	TA_COP1|TA_COP2
通常の浮動小数点演算を行う場合(FPU レジスタバンク 0 のみを使用)	TA_COP1	TA_COP1
浮動小数点演算は行わないが、コンパイラオプションとして fpu=mix を指定した場合	TA_COP1 *	(不要)
浮動小数点演算は行わなず、コンパイラオプションとして fpu=mix 以外を指定した場合	(不要)	(不要)

【注】　この使用形態は推奨しません。

G.2　非タスクコンテキスト(通常の割込みハンドラ、ダイレクト割込みハンドラ、CPU 例外ハンドラ、タイムイベントハンドラ、初期化ルーチン)

G.2.1　概要

(1)　FPU レジスタの保証

浮動小数点演算を行う場合は、これらのハンドラで全 FPU レジスタを明示的に保証しなければなりません。

(2)　FPSCR の保証

SH2A-FPU の場合は、コンパイラオプションとして、fpu=mix かつ fpscr=aggressive を指定した場合は、浮動小数点演算を行うかどうかに関わらず、これらのハンドラで FPSCR を明示的に保証しなければなりません。

SH-4, SH-4A では、FPSCR の明示的な保証は不要です。

(3)　FPSCR の初期化

各ハンドラ起動時の FPSCR は、CPU 例外ハンドラは CPU 例外発生前と同じ、その他のハンドラは不定です。

これらのハンドラで浮動小数点演算を行う場合は、ハンドラのエントリ関数の先頭で、「G.4.3 コンパイラによる扱い」に記載の「関数開始時点でコンパイラが想定している値」に FPSCR を初期化する必要があります。

以降、この方法について説明します。

G.2.2 SH-4, SH-4A の場合

HI7750/4 では、これらを行うための以下のマクロを提供しています。これらのマクロは、sh4fpu.h で定義されています。これらのマクロではインラインアセンブルを使用しているので、コンパイル時にはオブジェクト形式指定オプション code =asmcode を用いてコンパイルする必要があります。

(1) void IniFPU_ONEBANK(T_FPU_ONEBANK *pk_save, UW ini_fpscr)

現在のレジスタバンクのみを使用するためのマクロです。ハンドラ関数の先頭で使用します。pk_save に、現在の FPU レジスタを退避し、FPSCR を ini_fpscr に初期化します。

(2) void EndFPU_ONEBANK(T_FPU_ONEBANK *pk_save)

現在のレジスタバンクのみを使用するためのマクロです。ハンドラ関数の最後で使用します。pk_save から FPU レジスタを復帰します。

(3) void IniFPU_ALLBANK(T_FPU_ALLBANK *pk_save, UW ini_fpscr)

両方のレジスタバンクを使用するためのマクロです。ハンドラ関数の先頭で使用します。pk_save に、現在の FPU レジスタを退避し、FPSCR を ini_fpscr に初期化します。

(4) void EndFPU_ALLBANK(T_FPU_ALLBANK *pk_save)

両方のレジスタバンクを使用するためのマクロです。ハンドラ関数の最後で使用します。pk_save から FPU レジスタを復帰します。

図 G.2 に割込みハンドラにおける FPSCR 初期化・FPU レジスタ保証例を示します。

```
#include "sh4fpu.h"             /* ヘッダファイル"sh4fpu.h"をインクルード */
#define INI_FPSCR 0x00040001  /* FPSCR 初期値(FR=0, PR=0, DN=1, SZ=0, RM=B'01) */
void HandlerMain(void)         /* ハンドラの本体処理を行う関数 */
{
    /* ハンドラの処理 */
}

void Handler(void)              /* ハンドラのエントリ関数 */
{
    T_FPU_ONEBANK area;         /* FPU レジスタの保存領域 */
    IniFPU_ONEBANK(&area, INI_FPSCR);  /* FPU レジスタの保存、FPSCR の初期化 */
    HandlerMain();              /* 本体処理を行う HandlerMain()をコール */
    EndFPU_OneBank(&area);       /* FPU レジスタの復帰 */
}
```

図 G.2 割込みハンドラにおける FPSCR 初期化・FPU レジスタ保証例(SH-4, SH-4A)

G.2.3 SH2A-FPU の場合

HI7000/4 では、これらを行うための以下のマクロを提供しています。これらのマクロは、sh2afpu.h で定義されています。これらのマクロではインラインアセンブルを使用しているので、コンパイル時にはオブジェクト形式指定オプション code =asmcode を用いてコンパイルする必要があります。

(1) void IniFPU (VT_FPU *pk_save, UW ini_fpscr)

ハンドラ関数の先頭で使用します。
pk_save に、FPSCR を含む現在の FPU レジスタを退避し、FPSCR を ini_fpscr に初期化します。

(2) void EndFPU(VT_FPU *pk_save)

現在のレジスタバンクのみを使用するためのマクロです。ハンドラ関数の最後で使用します。
pk_save から FPSCR を含む FPU レジスタを復帰します。

記述例は、図 G.2 を参考にしてください。相違点は、使用するヘッダファイルとマクロの名称だけです。

G.3 拡張サービスコールルーチン

拡張サービスコールの発行は、コンパイラは関数呼び出しと扱います。

G.3.1 コンパイラオプションについて

拡張サービスコールの発行は、コンパイラは関数呼び出しと扱うため、基本的には拡張サービスコールルーチンとその呼び出し側は、以下のコンパイラオプションを同じにしてください。

- fpu オプション
- fpscr オプション
- denormalize オプション
- round オプション

呼び出し側と拡張サービスコールルーチンのコンパイラオプションが異なる場合は、以下の事項に留意してください。

(1) FPSCR の初期化

拡張サービスコールルーチン起動時の FPSCR は、SH-4, SH-4A の FPSCR.FR(レジスタバンク)ビットは呼び出し前と同じ、その他のビットは呼び出し元のコンパイラオプションで決まります。

拡張サービスコールルーチンで浮動小数点演算を行う場合は、拡張サービスコールルーチンのエントリ関数の先頭で、「G.4.3 コンパイラによる扱い」に記載の「関数開始時点でコンパイラが想定している値」に FPSCR を初期化する必要があります。

(2) FPSCR の保証

拡張サービスコールルーチンのコンパイラオプションが fpu=mix かつ fpscr=aggressive の場合は、拡張サービスコールルーチンで明示的に FPSCR を保証する必要があります。

図 G.3 に、拡張サービスコールルーチンにおける FPSCR の初期化・保証例を示します。

```
#include <machine.h>   /* 組み込み関数 set_fpscr()を使用するためにインクルード */
#define INI_FPSCR 0x00080000  /* FPSCR 初期値(FR=0, PR=1, DN=0, SZ=0, RM=B'00) */
void ExtendedSVCRoutine(VP_INT par1)
{
    UW old_fpscr;
    old_fpscr = get_fpscr(); /* FPSCR を退避 */
    set_fpscr(INI_FPSCR);    /* FPSCR を初期化 */
    /* 拡張サービスコールルーチンの処理 */
    set_fpscr(old_fpscr);    /* FPSCR の復帰 */
}
```

図 G.3 拡張サービスコールルーチンにおける FPSCR の初期化・保証例

G.3.2 タスクコンテキストから呼び出された場合

表 G.2 に、呼び出し元のタスクまたはタスク例外処理ルーチンに必要な TA_COP1, TA_COP2 属性を示します。

表 G.2 呼び出し元に必要な TA_COP1, TA_COP2 属性

条件	SH2A-FPU	SH-4, SH-4A
マトリックス演算などを行う場合(両方の FPU レジスタバンクを使用する場合)	ー	TA_COP1|TA_COP2
通常の浮動小数点演算を行う場合(FPU レジスタバンク 0 のみを使用)	TA_COP1	TA_COP1
浮動小数点演算は行わないが、コンパイラオプションとして fpu=mix を指定した場合	TA_COP1 *	(不要)
浮動小数点演算は行わなず、コンパイラオプションとして fpu=mix 以外を指定した場合	(不要)	(不要)

【注】 この使用形態は推奨しません。

G.3.3 非タスクコンテキストから呼び出された場合

浮動小数点演算を行う場合は、呼び出し元の割込みハンドラなどで全 FPU レジスタを保証する必要があります。「G.2 非タスクコンテキスト(通常の割込みハンドラ、ダイレクト割込みハンドラ、CPU 例外ハンドラ、タイムイベントハンドラ、初期化ルーチン)」を参照してください。

G.4 参考情報

G.4.1 タスクやハンドラなどの起動時の状態

表 G.3 に、タスクやハンドラなどの起動時の状態を示します。

表 G.3 タスクやハンドラなどの起動時の状態

起動時の状態	タスク, タスク例外処理ルーチン	拡張サービスコールルーチン	割込みハンドラ, タイムイベントハンドラ,初期化ルーチン	CPU 例外ハンドラ(TRAPA 含む)
FPSCR の値	H'00040001	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
精度モード (FPSCR.PR)	単精度(0)	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
非正規化モード (FPSCR.DN) *1	ゼロと扱う(1)	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
丸めモード (FPSCR.RM)	ゼロに丸める (B'01)	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
転送サイズモード (FPSCR.SZ)	32 ビット(0)	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
FPU レジスタバンク (FPSCR.FR) *2	バンク 0(0)	拡張サービスコール発行前と同じ	不定	例外発生前と同じ
FPSCR のその他のビット	0	拡張サービスコール発行前と同じ	不定	例外発生前と同じ

【注】 *1 SH2A-FPU では常に B'01

*2 SH-4, SH-4A のみ

G.4.2 FPSCR 構造

31–22	21	20	19	18	17–12	11–7	6–2	1–0
予約	FR	SZ	PR	DN	Cause	Enable	Flag	RM

bit		意味		
21	FR	FPU レジスタバンク *1	0	バンク 0
			1	バンク 1
20	SZ	転送サイズモード	0	FMOV のデータサイズは 32 ビット
			1	FMOV のデータサイズは 32 ビットペア(64 ビット)
19	PR	精度モード	0	単精度
			1	倍精度
18	DN	非正規化モード *2	0	非正規化数を 0 と扱う
			1	非正規化数を非正規化数と扱う
17-12	Cause	FPU 例外要因		
11-7	Enable	FPU 例外イネーブル		
6-2	Flag	FPU 例外フラグ		
1,0	RM	丸めモード	B'00	近傍への丸め
			B'01	0 方向への丸め

【注】 *1 SH-4, SH-4A のみ

*2 SH2A-FPU では常に B'01

G.4.3 コンパイラによる扱い

本節では、コンパイラによるFPUの扱いについて解説します。なお、FPUオプションにSingleまたはDoubleが指定された場合は、コンパイラがFPSCRを変更するオブジェクトコードを生成することはありません。

(1) FPSCR.PR(精度モード)

表 G.4 コンパイラによる FPSCR.PR ビットの扱い

コンパイラオプション		関数開始時点でコンパイラが想定している精度モード(FPSCR.PR ビット) *1	関数終了時の精度モード *2	備考
FPU オプション	FPSCR オプション			
Single	(指定不可)	単精度(0)	単精度(0)	コンパイラは PR ビットを変更するコード生成を行いません。
Double	(指定不可)	倍精度(1)	倍精度(1)	
指定無し (Mix)	Safe	単精度(0)	単精度(0)	
	Aggressive	単精度(0)	不定	

【注】 *1 コンパイラは、関数先頭でこの精度モードになっている前提でコード生成を行います。
*2 コンパイラは、関数終了時にこの精度モードとなるようにコード生成を行います。

(2) FPSCR.DN(非正規化モード)(SH-4, SH-4A のみ)

表 G.5 コンパイラによる FPSCR.DN ビットの扱い

コンパイラオプション	コンパイラが想定している非正規化モード(FPSCR.DN ビット) *1	備考
Denormalize オプション		
OFF	非正規化数を 0 と扱う(1)	コンパイラは DN ビットを変更するコード生成を行いません
ON	非正規化数を非正規化数と扱う(0)	

【注】 *1 コンパイラは、関数先頭でこの非正規化モードになっている前提でコード生成を行います。

(3) FPSCR.RM(丸めモード)

表 G.6 コンパイラによる FPSCR.RM ビットの扱い

コンパイラオプション	コンパイラが想定している丸めモード (FPSCR.RM ビット) *1	備考
Round オプション		
Zero	0 で丸める(B'01)	コンパイラは RM ビットを変更するコード生成を行いません
Nearest	近傍に丸める(B'00)	

【注】 *1 コンパイラは、関数先頭でこの丸めモードになっている前提でコード生成を行います。

(4) FPSCR.SZ(転送サイズモード)

コンパイラは、常にSZ=0(FMOV命令の転送サイズは32ビット)を仮定しています。また、SZビットを変更するコード生成は行いません。

(5) FPSCR.FR(FPU レジスタバンク)(SH-4, SH-4A のみ)

コンパイラは、FRビットを変更するコード生成は行いません。
ただし、組み込み関数st_ext(), ld_ext()では、関数内部で一時的にFRビットを変更します。リターン時のFRビットは、呼び出し時と同じです。

付録H　HI7000/4 V.2 の新機能

H.1　SH-2A, SH2A-FPU のサポート

H.1.1　SH2A-FPU の FPU(V.2.00 Release 00)

タスクおよびタスク例外処理ルーチンの属性として TA_COP1 属性をサポートしました。TA_COP1 属性を指定することで、そのタスクおよびタスク例外処理ルーチンで FPU レジスタがコンテキストレジスタとして追加され、マルチタスク環境で FPU を使用することが可能となります。

コンフィギュレータでは、タスクの生成およびタスク例外処理ルーチンの定義のダイアログボックスで TA_COP1 属性を選択可能になっています。

■関連ページ

- p.63：「3.4.1　タスクの生成」
- p.92：「3.6.1　タスク例外処理ルーチンの定義(def_tex, idef_tex)」
- p. 322：「5.11.1　コンパイラ、アセンブラの CPU オプション」
- p. 375：「付録 G SH2A-FPU, SH-4, SH-4A における FPU に関する注意」

H.1.2　TBR レジスタ(V.2.00 Release 00)

TBR レジスタの利用形態として、以下の３つの中から選択できます。

- カーネル管理外
- サービスコール呼び出し専用に使用
- タスクコンテキストと扱う

これは、コンフィギュレータの[カーネル動作条件]ページの CFG_TBR によって選択します。

■関連ページ

- p.258：「4.2.8 TBR レジスタ(SH-2A, SH2A-FPU)」
- p.308：「5.4.6　コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 1.4
- p. 322：「5.11.1　コンパイラ、アセンブラの CPU オプション」

H.1.3 レジスタバンク(V.2.00 Release 00, V.2.01 Release 00, V.2.02 Release 00)

V.2.00 Release 00 にて、コンフィギュレータの CFG_REGBANK チェックボックスによって、NMI,UBC を除く全割込みについて、全てレジスタバンクを使用するか全てレジスタバンクを使用しないかを設定できるようにしました。

さらに、V.2.02 Release 00 では、割込みレベル毎にレジスタバンクを使用するかを選択できるように、CFG_REGBANK をコンボボックス形式に変更しました。

また、V.2.01 Release 00 にて、コンフィギュレータの[割込み,CPU 例外ハンドラ]ページに、レジスタバンクに関する IBNR レジスタアドレスを設定するための CFG_IBNR_ADR を追加しました。従来バージョンのカーネルでは、IBNR レジスタが固定アドレス (H'fffe080e) として処理していましたが、マイコン品種によって IBNR レジスタが異なることがあるため、CFG_IBNR_ADR を追加しました。

■関連ページ

- p.258 : 「4.2.9 レジスタバンク(SH-2A, SH2A-FPU)」
- p. 270 : 「4.8 割込みハンドラ」
- p. 308 : 「5.4.6 コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 6.5 と項番 6.6

H.2 タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.00 Release 00)

従来バージョンでは、タスク生成時のスタックアドレス、固定長メモリプール生成時のメモリプールアドレス、可変長メモリプール生成時のメモリプールアドレスはそれぞれ指定できず、決められた領域からカーネルが割り当てる方式のみサポートしていました。

V2 ではこれに加え、ユーザがそれぞれの領域を確保し、生成時にそのアドレスを指定可能としました。

これによって、例えば特定のタスクのスタックだけ高速な内蔵メモリを使う、というように、ユーザが使用目的に応じて個々のタスクスタック、個々のメモリプールを任意のアドレスに割り当てることができるようになりました。

■関連ページ

- p.11 : 「2.6.2 タスクの生成」
- p. 63 : 「3.4.1 タスクの生成」
- p. 150 : 「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 159 : 「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」

RENESAS

H.3　固定長メモリプールの管理方式(V.2.00 Release 00)

固定長メモリプールの各メモリブロックを管理するためのカーネルの管理テーブルは、従来バージョンでは固定長メモリプール領域内に存在していました。

V2 では、このカーネル管理テーブル領域をユーザが確保し、そのアドレスを固定長メモリプール生成時に指定可能としました。この場合、メモリプール内に管理テーブルは配置されなくなります。

「H.2 タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能」に記載のメモリプールアドレスの指定機能と組み合わせると、例えば以下のように特定のオフセットアドレスを持つメモリブロックを獲得できるようになります。

■固定長メモリプール生成時の指定

- 固定長メモリプールのアドレス＝H'0c000000
- 固定長メモリブロックのサイズ＝H'1000(4kB)
- 固定長メモリブロック数＝4

この例の場合、獲得されるメモリブロックは以下のメモリブロック[A]～[D]のいずれかとなり、いずれも 4KB 境界のアドレスとなります。

アドレス	
H'0C000000→	メモリブロック[A]
H'0C001000→	メモリブロック[B]
H'0C002000→	メモリブロック[C]
H'0C003000→	メモリブロック[D]

新方式では、固定長メモリプール生成時に指定する T_CMPF 構造体の新メンバとして、管理テーブルアドレスを意味する mpfmb が追加されています。

なお、固定長メモリプールの管理方式は、従来方式か新方式かをコンフィギュレータの[固定長メモリプール]ページの CFG_MPFMANAGE で選択します。両方式を同時に使用することはできません。

■関連ページ

- p. 150：「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 17.3

H.4　ダイレクト割込みハンドラ(V.2.00 Release 00, V.2.02 Release 00)

従来バージョンでは、ダイレクト割込みハンドラからサービスコールを呼び出してはならない仕様でしたが、V.2.00 Release 00 ではこれを可能にしました。ただし、カーネル割込みマスクレベルよりも高い割込みレベルのダイレクト割込みハンドラはサービスコールを呼び出してはならない、という仕様は従来と同じです。

また、V.2.02 Release 00 にて、def_inh, idef_inh サービスコールでダイレクト割込みハンドラを定義できるように改善しました。

H.5　サイズを算出するマクロ(V.2.00 Release 00)

以下のマクロを追加しました。

(1)　SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールのサイズ（バイト数）

(2)　SIZE size = VTSZ_MPFMB(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールの管理領域のサイズ（バイト数）

(3)　SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる可変長メモリプールのサイズ（目安のバイト数）

■関連ページ

- p253：「4.1 ヘッダファイル」の「表 4.1　定数、マクロ」

H.6　最大ベクタ番号の拡張(V.2.00 Release 00)

コンフィギュレータで指定する最大ベクタ番号(CFG_MAXVCTNO)に指定可能な値を、255 から 511 に拡張しました。

■関連ページ

- p.207：「3.20 割込み管理機能」の「表 3.39　割込み管理の仕様」
- p. 216：「3.22 システム構成管理機能」の「表 3.43　システム構成管理の仕様」

H.7　ID 名称(V.2.00 Release 00)

従来バージョンのコンフィギュレータでは、ID 番号を自動割当する場合のみ ID 名称の設定が可能でしたが、V2 では全て ID 名称を設定可能としました。表 H.1 に、従来バージョンからの ID 名称に関する機能追加について示します。

表 H.1　ID 名称に関する機能追加

バージョン	カーネル側(kernel_id_sys.h)		カーネル環境側(kernel_id.h)	
	ID 番号自動割当	ID 名称の設定	ID 番号自動割当	ID 名称の設定
従来バージョン	不可	不可	可能	ID 番号を自動割り当てと指定したもののみ可能
V.2.00 以降		可能		可能

■関連ページ

- p. 255：「4.1.1(3) ID 名称ヘッダファイル」
- p. 305：「5.4.4 (1) kernel_id.h, kernel_id_sys.h」

H.8　SH-2 リトルエンディアンのサポート(V.2.00 Release 01)

SH-2 のリトルエンディアンモードをサポートしました。

H.9 可変長メモリプールの改善(V.2.01 Release 00)

コンフィギュレータに新設された CFG_NEWMPL を選択すると、以下が改善されます。

(1) 獲得/返却の高速化

特に多くのブロックを扱う使い方をする場合、CFG_NEWMPL 非選択時よりも処理が高速になります。

(2) 空き領域断片化の軽減

全般に、CFG_NEWMPL 非選択時よりも空き領域が断片化しにくくなります。また、さらに空き領域の断片化を軽減する VTA_UNFRAGMENT 属性をサポートしました。断片化度合いは、一般には VTA_UNFRAGMENT を指定したほうが良くなる傾向がありますが、メモリプールの利用方法に依存します。

VTA_UNFRAGMENT 属性の追加に伴い、以下のマクロを追加しました。

```
SIZE mplsz = VTSZ_MPLMB(UINT sctnum)
```

VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）

CFG_NEWMPL を選択した場合は、可変長メモリプール生成時に指定する T_CMPL 構造体の新メンバとして、管理テーブルアドレスを意味する「mpfmb」、最小ブロックサイズ「minblksz」、セクタ数「sctnum」が追加されます。ただし、これらは VTA_UNFRAGMENT 属性の指定が無い場合は、単に無視されます。

なお、CFG_NEWMPL を選択すると、全般にスタック使用サイズが増加します。

■関連ページ

- p. 34：「2.15.2 空き領域の断片化とその対策」
- p. 159：「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 18.3
- p. 347：付録 C 作業領域サイズの算出」

H.10 DSR レジスタの初期値(V.2.01 Release 00)

TA_COP0 属性が指定されたタスクおよびタスク例外処理ルーチンの DSR レジスタの初期値を、不定値から 0 に変更しました。

■関連ページ

- p.295：「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」

H.11 タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00)

タスク例外処理ルーチンの SR レジスタの初期値を、以下のように変更しました。

【変更前】起動直前のタスクと同じ

【変更後】H'00000000

H.12　ベクタ番号 16～31 の扱い(V.2.01 Release 00, V.2.02 Release 00)

V.2.01 Release 00 にて、25 を除くこれらのベクタ番号に対してハンドラを定義できるようにしましたが、さらに V.2.02 Release 00 にてベクタ番号 26 はカーネルが使用する仕様に変更したためにハンドラを定義できないこととしました。

H.13　構造体アライメントに関する制限事項の解除(V.2.01 Release 00)

コンパイラの pack オプションおよび#pragma pack に関し、従来マニュアルにて以下の注意を記載していましたが、この制限を解除しました。

カーネルで定義されている構造体型の変数を使用するソースプログラムは、必ず"pack=4"を指定してください。また、カーネルで定義されている構造体型の変数を#pragma pack 1 宣言しないでくださまた、kernel_def.c, kernel_cfg.c も、コンパイルオプションとして必ず"pack=4"を指定してください。

H.14　コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00)

メニューバーに[オプション]メニューを追加しました。[オプション]メニューには[前回使用したファイルを開く] コマンドがあります。これを選択すると、コンフィギュレータは起動時に前回使用したファイルを自動的に開きます。

付録I　HI7700/4 V.2 の新機能

I.1　SH4AL-DSP(拡張機能あり)のサポート(V.2.01 Release 00)

SH4AL-DSP(拡張機能あり)用のキャッシュサポートライブラリ(shx2_cache_???.lib)を追加しました。

I.2　タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00)

従来バージョンでは、タスク生成時のスタックアドレス、固定長メモリプール生成時のメモリプールアドレス、可変長メモリプール生成時のメモリプールアドレスはそれぞれ指定できず、決められた領域からカーネルが割り当てる方式のみサポートしていました。

V2 ではこれに加え、ユーザがそれぞれの領域を確保し、生成時にそのアドレスを指定可能としました。

これによって、例えば特定のタスクのスタックだけ高速な内蔵メモリを使う、というように、ユーザが使用目的に応じて個々のタスクスタック、個々のメモリプールを任意のアドレスに割り当てることができるようになりました。

■関連ページ

- p.11：「2.6.2 タスクの生成」
- p. 63：「3.4.1 タスクの生成」
- p. 150：「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 159：「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」

I.3　固定長メモリプールの管理方式(V.2.01 Release 00)

固定長メモリプールの各メモリブロックを管理するためのカーネルの管理テーブルは、従来バージョンでは固定長メモリプール領域内に存在していました。

V2 では、このカーネル管理テーブル領域をユーザが確保し、そのアドレスを固定長メモリプール生成時に指定可能としました。この場合、メモリプール内に管理テーブルは配置されなくなります。

「I.2 タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00)」に記載のメモリプールアドレスの指定機能と組み合わせると、例えば以下のように特定のオフセットアドレスを持つメモリブロックを獲得できるようになります。

■固定長メモリプール生成時の指定

- 固定長メモリプールのアドレス＝H'0c000000
- 固定長メモリブロックのサイズ＝H'1000(4kB)
- 固定長メモリブロック数＝4

この例の場合、獲得されるメモリブロックは以下のメモリブロック[A]～[D]のいずれかとなり、いずれも 4KB 境界のアドレスとなります。

アドレス	
H'0C000000→	メモリブロック[A]
H'0C001000→	メモリブロック[B]
H'0C002000→	メモリブロック[C]
H'0C003000→	メモリブロック[D]

新方式では、固定長メモリプール生成時に指定する T_CMPF 構造体の新メンバとして、管理テーブルアドレスを意味する mpfmb が追加されています。

なお、固定長メモリプールの管理方式は、従来方式か新方式かをコンフィギュレータの[固定長メモリプール]ページの CFG_MPFMANAGE で選択します。両方式を同時に使用することはできません。

■関連ページ

- p. 150：「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5 コンフィギュレータの設定項目」の項番 17.3

I.4 可変長メモリプールの改善(V.2.01 Release 00)

コンフィギュレータに新設された CFG_NEWMPL を選択すると、以下が改善されます。

(1) 獲得/返却の高速化

特に多くのブロックを扱う使い方をする場合、CFG_NEWMPL 非選択時よりも処理が高速になります。

(2) 空き領域断片化の軽減

全般に、CFG_NEWMPL 非選択時よりも空き領域が断片化しにくくなります。また、さらに空き領域の断片化を軽減する VTA_UNFRAGMENT 属性をサポートしました。断片化度合いは、一般には VTA_UNFRAGMENT を指定したほうが良くなる傾向がありますが、メモリプールの利用方法に依存します。

VTA_UNFRAGMENT 属性の追加に伴い、以下のマクロを追加しました。

```
SIZE mplsz = VTSZ_MPLMB(UINT sctnum)
```

VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）

CFG_NEWMPL を選択した場合は、可変長メモリプール生成時に指定する T_CMPL 構造体の新メンバとして、管理テーブルアドレスを意味する「mpfmb」、最小ブロックサイズ「minblksz」、セクタ数「sctnum」が追加されます。ただし、これらは VTA_UNFRAGMENT 属性の指定が無い場合は、単に無視されます。

なお、CFG_NEWMPL を選択すると、全般にスタック使用サイズが増加します。

■関連ページ

- p. 34：「2.15.2 空き領域の断片化とその対策」
- p. 159：「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5 コンフィギュレータの設定項目」の項番 18.3
- p. 347：付録 C 作業領域サイズの算出」

I.5 サイズを算出するマクロ(V.2.01 Release 00)

以下のマクロを追加しました。

(1) SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールのサイズ（バイト数）

(2) SIZE size = VTSZ_MPFMB(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールの管理領域のサイズ（バイト数）

(3) SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる可変長メモリプールのサイズ（目安のバイト数）

(4) SIZE mplsz = VTSZ_MPLMB(UINT sctnum)

VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）

■関連ページ

- p.253：「4.1 ヘッダファイル」の「表 4.1　定数、マクロ」

I.6 DSR レジスタの初期値(V.2.01 Release 00)

TA_COP0 属性が指定されたタスクおよびタスク例外処理ルーチンの DSR レジスタの初期値を、不定値から 0 に変更しました。なお、本変更は.1.03 Release 02 でも実施されています。

また、vchg_cop サービスコールで TA_COP0 なしから TA_COP0 ありに変更した場合、従来バージョンではカーネルは DSR レジスタを初期化しませんでしたが、これを 0 に初期化するようにしました。

■関連ページ

- p.295：「4.13 DSP を使用する場合(HI7000/4, HI7700/4)」

I.7 タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00)

タスク例外処理ルーチンの SR レジスタの初期値を、以下のように変更しました。

【変更前】起動直前のタスクと同じ

【変更後】CFG_DSP, CFG_CACLOC の少なくとも一方をチェックした場合はH'40001000, いずれもチェックしない場合はH'40000000

I.8 最大例外コード(CFG_MAXVCTNO)の拡張(V.2.01 Release 00)

コンフィギュレータで指定する最大例外コード(CFG_MAXVCTNO)に指定可能な値を、0xfe0 から 0x3fe0 に拡張しました。

I.9 TRAPA #16～31 の扱い(V.2.01 Release 00)

これらの番号に対して、トラップ例外ハンドラを定義できるようにしました。

I.10 構造体アライメントに関する制限事項の解除(V.2.01 Release 00)

コンパイラの pack オプションおよび#pragma pack に関し、従来マニュアルにて以下の注意を記載していましたが、この制限を解除しました。

> カーネルで定義されている構造体型の変数を使用するソースプログラムは、必ず"pack=4"を指定してください。また、カーネルで定義されている構造体型の変数を#pragma pack 1 宣言しないでくださ
> また、kernel_def.c, kernel_cfg.c も、コンパイルオプションとして必ず"pack=4"を指定してください。

I.11 ID 名称(V.2.01 Release 00)

従来バージョンのコンフィギュレータでは、ID 番号を自動割当する場合のみ ID 名称の設定が可能でしたが、V2 では全て ID 名称を設定可能としました。表 I.1 に、従来バージョンからの ID 名称に関する機能追加について示します。

表 I.1 ID 名称に関する機能追加

<table>
<tr><th rowspan="2">バージョン</th><th colspan="2">カーネル側(kernel_id_sys.h)</th><th colspan="2">カーネル環境側(kernel_id.h)</th></tr>
<tr><th>ID 番号自動割当</th><th>ID 名称の設定</th><th>ID 番号自動割当</th><th>ID 名称の設定</th></tr>
<tr><td>従来バージョン</td><td rowspan="2">不可</td><td>不可</td><td rowspan="2">可能</td><td>ID 番号を自動割り当てと指定したもののみ可能</td></tr>
<tr><td>V.2.01 以降</td><td>可能</td><td>可能</td></tr>
</table>

■関連ページ

- p. 255：「4.1.1(3) ID 名称ヘッダファイル」
- p. 305：「5.4.4 (1) kernel_id.h, kernel_id_sys.h」

I.12 コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00)

メニューバーに[オプション]メニューを追加しました。[オプション]メニューには[前回使用したファイルを開く] コマンドがあります。これを選択すると、コンフィギュレータは起動時に前回使用したファイルを自動的に開きます。

付録J　HI7750/4 V.2 の新機能

J.1　SH-4A(拡張機能あり)のサポート(V.2.01 Release 00)

SH-4A(拡張機能あり)用のキャッシュサポートライブラリ(shx2_cache_???.lib)を追加しました。

J.2　タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00)

従来バージョンでは、タスク生成時のスタックアドレス、固定長メモリプール生成時のメモリプールアドレス、可変長メモリプール生成時のメモリプールアドレスはそれぞれ指定できず、決められた領域からカーネルが割り当てる方式のみサポートしていました。

V2 ではこれに加え、ユーザがそれぞれの領域を確保し、生成時にそのアドレスを指定可能としました。

これによって、例えば特定のタスクのスタックだけ高速な内蔵メモリを使う、というように、ユーザが使用目的に応じて個々のタスクスタック、個々のメモリプールを任意のアドレスに割り当てることができるようになりました。

■関連ページ

- p.11：「2.6.2 タスクの生成」
- p. 63：「3.4.1 タスクの生成」
- p. 150：「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 159：「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」

J.3　固定長メモリプールの管理方式(V.2.01 Release 00)

固定長メモリプールの各メモリブロックを管理するためのカーネルの管理テーブルは、従来バージョンでは固定長メモリプール領域内に存在していました。

V2 では、このカーネル管理テーブル領域をユーザが確保し、そのアドレスを固定長メモリプール生成時に指定可能としました。この場合、メモリプール内に管理テーブルは配置されなくなります。

「J.2 タスクスタック、固定長メモリプール、可変長メモリプールのアドレス指定機能(V.2.01 Release 00)」に記載のメモリプールアドレスの指定機能と組み合わせると、例えば以下のように特定のオフセットアドレスを持つメモリブロックを獲得できるようになります。

■固定長メモリプール生成時の指定

- 固定長メモリプールのアドレス＝H'0c000000
- 固定長メモリブロックのサイズ＝H'1000(4kB)
- 固定長メモリブロック数＝4

この例の場合、獲得されるメモリブロックは以下のメモリブロック[A]～[D]のいずれかとなり、いずれも 4KB 境界のアドレスとなります。

アドレス	
H'0C000000→	メモリブロック[A]
H'0C001000→	メモリブロック[B]
H'0C002000→	メモリブロック[C]
H'0C003000→	メモリブロック[D]

新方式では、固定長メモリプール生成時に指定する T_CMPF 構造体の新メンバとして、管理テーブルアドレスを意味する mpfmb が追加されています。

なお、固定長メモリプールの管理方式は、従来方式か新方式かをコンフィギュレータの[固定長メモリプール]ページの CFG_MPFMANAGE で選択します。両方式を同時に使用することはできません。

■関連ページ

- p. 150：「3.13.1 固定長メモリプールの生成(cre_mpf, icre_mpf)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 17.3

J.4　可変長メモリプールの改善(V.2.01 Release 00)

コンフィギュレータに新設された CFG_NEWMPL を選択すると、以下が改善されます。

(1)　獲得/返却の高速化

特に多くのブロックを扱う使い方をする場合、CFG_NEWMPL 非選択時よりも処理が高速になります。

(2)　空き領域断片化の軽減

全般に、CFG_NEWMPL 非選択時よりも空き領域が断片化しにくくなります。また、さらに空き領域の断片化を軽減する VTA_UNFRAGMENT 属性をサポートしました。断片化度合いは、一般には VTA_UNFRAGMENT を指定したほうが良くなる傾向がありますが、メモリプールの利用方法に依存します。

VTA_UNFRAGMENT 属性の追加に伴い、以下のマクロを追加しました。

```
SIZE mplsz = VTSZ_MPLMB(UINT sctnum)
```

VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）

CFG_NEWMPL を選択した場合は、可変長メモリプール生成時に指定する T_CMPL 構造体の新メンバとして、管理テーブルアドレスを意味する「mpfmb」、最小ブロックサイズ「minblksz」、セクタ数「sctnum」が追加されます。ただし、これらは VTA_UNFRAGMENT 属性の指定が無い場合は、単に無視されます。

なお、CFG_NEWMPL を選択すると、全般にスタック使用サイズが増加します。

■関連ページ

- p. 34：「2.15.2 空き領域の断片化とその対策」
- p. 159：「3.14.1 可変長メモリプールの生成(cre_mpl, icre_mpl)」
- p. 308：「5.4.6 コンフィギュレータの設定項目」の「表 5.5　コンフィギュレータの設定項目」の項番 18.3
- p. 347：付録 C 作業領域サイズの算出」

J.5 サイズを算出するマクロ(V.2.01 Release 00)

以下のマクロを追加しました。

(1) SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールのサイズ（バイト数）

(2) SIZE size = VTSZ_MPFMB(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる固定長メモリプールの管理領域のサイズ（バイト数）

(3) SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)

サイズが blksz バイトのメモリブロックを blkcnt 個獲得できる可変長メモリプールのサイズ（目安のバイト数）

(4) SIZE mplsz = VTSZ_MPLMB(UINT sctnum)

VTA_UNFRAGMENT 属性の可変長メモリプールの管理領域のサイズ（バイト数）

■関連ページ

• p253：「4.1 ヘッダファイル」の「表 4.1 定数、マクロ」

J.6 タスク例外処理ルーチンの SR レジスタの初期値(V.2.01 Release 00)

タスク例外処理ルーチンの SR レジスタの初期値を、以下のように変更しました。
【変更前】起動直前のタスクと同じ
【変更後】H'40000000

J.7 最大例外コード(CFG_MAXVCTNO)の拡張(V.2.01 Release 00)

コンフィギュレータで指定する最大例外コード(CFG_MAXVCTNO)に指定可能な値を、0xfe0 から 0x3fe0 に拡張しました。

J.8 TRAPA #16～31 の扱い(V.2.01 Release 00)

これらの番号に対して、トラップ例外ハンドラを定義できるようにしました。

J.9 構造体アライメントに関する制限事項の解除(V.2.01 Release 00)

コンパイラの pack オプションおよび#pragma pack に関し、従来マニュアルにて以下の注意を記載していましたが、この制限を解除しました。

カーネルで定義されている構造体型の変数を使用するソースプログラムは、必ず"pack=4"を指定してください。また、カーネルで定義されている構造体型の変数を#pragma pack 1 宣言しないでくださまた、kernel_def.c, kernel_cfg.c も、コンパイルオプションとして必ず"pack=4"を指定してください。

J.10 ID 名称(V.2.01 Release 00)

従来バージョンのコンフィギュレータでは、ID 番号を自動割当する場合のみ ID 名称の設定が可能でしたが、V2 では全て ID 名称を設定可能としました。表 J.1 に、従来バージョンからの ID 名称に関する機能追加について示します。

表 J.1 ID 名称に関する機能追加

バージョン	カーネル側(kernel_id_sys.h)		カーネル環境側(kernel_id.h)	
	ID 番号自動割当	ID 名称の設定	ID 番号自動割当	ID 名称の設定
従来バージョン	不可	不可	可能	ID 番号を自動割り当てと指定したもののみ可能
V.2.01 以降		可能		可能

■関連ページ

- p. 255 : 「4.1.1(3) ID 名称ヘッダファイル」
- p. 305 : 「5.4.4 (1) kernel_id.h, kernel_id_sys.h」

J.11 コンフィギュレータの[前回使用したファイルを開く]コマンド(V.2.02 Release 00)

メニューバーに[オプション]メニューを追加しました。[オプション]メニューには[前回使用したファイルを開く] コマンドがあります。これを選択すると、コンフィギュレータは起動時に前回使用したファイルを自動的に開きます。

ルネサスマイクロコンピュータ開発環境システム
ユーザーズマニュアル
HI7000/4シリーズ（HI7000/4 V.2.02, HI7700/4 V.2.02, HI7750/4 V.2.02）

発行年月日　2006 年 1 月 4 日　Rev.7.00
発　　　行　株式会社ルネサス テクノロジ 営業企画統括部
　　　　　　〒100-0004 東京都千代田区大手町 2-6-2
編　　　集　株式会社ルネサスソリューションズ
　　　　　　グローバルストラテジックコミュニケーション本部
　　　　　　カスタマサポート部

© 2004. Renesas Technology Corp., All rights reserved. Printed in Japan.

株式会社ルネサス テクノロジ　営業企画統括部　〒100-0004 東京都千代田区大手町2-6-2 日本ビル

営業お問合せ窓口
株式会社ルネサス販売

http://www.renesas.com

本社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
京浜支社	〒212-0058	川崎市幸区鹿島田890-12 (新川崎三井ビル)	(044) 549-1662
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
東北支社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア13F)	(022) 221-1351
いわき支店	〒970-8026	いわき市平小太郎町4-9 (平小太郎ビル)	(0246) 22-3222
茨城支店	〒312-0034	ひたちなか市堀口832-2 (日立システムプラザ勝田1F)	(029) 271-9411
新潟支店	〒950-0087	新潟市東大通1-4-2 (新潟三井物産ビル3F)	(025) 241-4361
松本支社	〒390-0815	松本市深志1-2-11 (昭和ビル7F)	(0263) 33-6622
中部支社	〒460-0008	名古屋市中区栄4-2-29 (名古屋広小路プレイス)	(052) 249-3330
関西支社	〒541-0044	大阪市中央区伏見町4-1-1 (明治安田生命大阪御堂筋ビル)	(06) 6233-9500
北陸支社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル8F)	(076) 233-5980
広島支店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング8F)	(082) 244-2570
鳥取支店	〒680-0822	鳥取市今町2-251 (日本生命鳥取駅前ビル)	(0857) 21-1915
九州支社	〒812-0011	福岡市博多区博多駅前2-17-1 (ヒロカネビル本館5F)	(092) 481-7695

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：コンタクトセンタ　E-Mail: csc@renesas.com

HI7000/4 シリーズ
(HI7000/4 V.2.02, HI7700/4 V.2.02, HI7750/4 V.2.02)
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10B0063-0700