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

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

HI1000/4 V.1.04
ユーザーズマニュアル
（H8SX,H8Sファミリ用リアルタイムOS）

ルネサスマイクロコンピュータ開発環境システム

R0R41600TRW01J

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　2006.06

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、HI1000/4(以下、本製品と略します。)の使用方法について説明します。
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。

■表記上の注意事項

HEW	弊社統合開発環境ツール「High-performance Embedded Workshop」を HEW と省略して表記します。
"H'",. "0x", "D'"	“H'”と"0x"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 ***nnnnz***_tmrdrv.c と表記していますが、H8SX/1650 用のタイマドライバファイル名は”1650z_tmrdrv.src”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、「7.4.4 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。

■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://japan.renesas.com/homepage.jsp

目次

図目次

表目次

目次

1.　本マニュアルの構成

本マニュアルは、以下の章から構成されています。

●「2. 概要」
本製品の概要について解説しています。

●「3. カーネル入門」
カーネルに関する基本的な項目を解説しています。

●「4. カーネルの機能」
カーネルの機能全般について解説しています。

●「5. サービスコール」
サービスコールの仕様について解説しています。

●「6. アプリケーションプログラムの作成」
タスクやハンドラの作成方法について解説しています。

●「7. コンフィギュレーション」
コンフィギュレータの位置付け、機能、使用方法について解説しています。

●「8. HEW ワークスペースとプロジェクト」
各サンプルプログラムの解説と、それらを用いてロードモジュールを生成する方法を解説しています。

●「9. 作業領域サイズの算出」
スタックサイズの算出方法について解説しています。

●「10. システムダウン時の情報」
システムダウンなど、異常発生時の対処方法について解説しています。

●「11. 各種一覧」
サービスコールやエラーコードなどの一覧を掲載しています。

●「12. 付録」
デバイスに依存した定義ファイルを掲載しています。

2. 概要

2.1 特長

2.1.1 H8 マイコンを幅広くサポート

HI1000/4 V.1.04 は、H8SX、H8S ファミリ、および IC カードマイコン AE5 シリーズ CPU で動作するμITRON4.0 仕様に準拠したマイコン用のリアルタイム・マルチタスク OS です。
HI1000/4は、各H8マイコンの以下に示す割込み制御モード、CPU動作モードに対応しています。

(1) H8SX ファミリ、IC カードマイコン AE5 シリーズ

- 割込み制御モード：０、２
- CPU 動作モード：アドバンストモード

(2) H8S ファミリ

- 割込み制御モード：０、１、２、３
- CPU動作モード：ノーマルモード、アドバンストモード

2.1.2 豊富なリアルタイム・マルチタスクサービス

HI1000/4 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
- タスクの起動、終了、優先度変更などのタスク管理機能
- タスクの実行中断、再開、自タスクの遅延などのタスク付属同期機能
- 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
- システム構成管理機能

2.1.3 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。

カーネルライブラリとアプリケーションを一括してリンクすることにより､ユーザシステムで使用するサービスコールのみが組み込まれます。

2.1.4 コンフィギュレータ

カーネルの構築作業を GUI 画面上で容易に行えるコンフィギュレータを装備しています。

2.1.5 サンプルプログラム

以下のようなサンプルを提供しています。

- システムダウンルーチン
- H8S™、H8SX™、AE5™マイコン内蔵タイマ用のタイマドライバ
- 各種ハンドラおよびルーチン
- CPU 初期化ルーチン
- コンフィギュレータ設定ファイル
- ロードモジュールを生成する HEW ワークスペース

2.1.6 デバッギングエクステンション (オプション製品)

弊社統合開発環境ツールの「HEW」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。

2.2 動 作 環 境

表 2-1に、動作環境を示します。

表2-1 動作環境

項目	内容
対象マイコン	H8S™、H8SX™、AE5™マイコン全般
ホストコンピュータ	Windows® 98, Windows® Millennium Edition(Windows® Me), WindowsNT® 4.0,Windows® 2000, Windows® XP が動作している PC
サンプル HEW ワークスペース、プロジェクト	HEW Ver.4.0 以降 （H8SX,H8S,H8 ファミリ C/C++コンパイラパッケージ V6.01 Release01 以降）

3. カーネル入門

3.1 カーネルの動作原理

カーネルとは、リアルタイム OS の中核となるプログラムのことです。

カーネルは、1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せることのできるソフトウェアです。では 1 個のマイクロコンピュータをどのようにして複数あるように見せかけるのでしょうか？

それは、図 3-1に示すようにそれぞれのタスクを時分割で動作させるからです。つまり実行するタスクを一定時間ごとに切り替えて、複数のタスクが同時に実行しているように見せるのです。

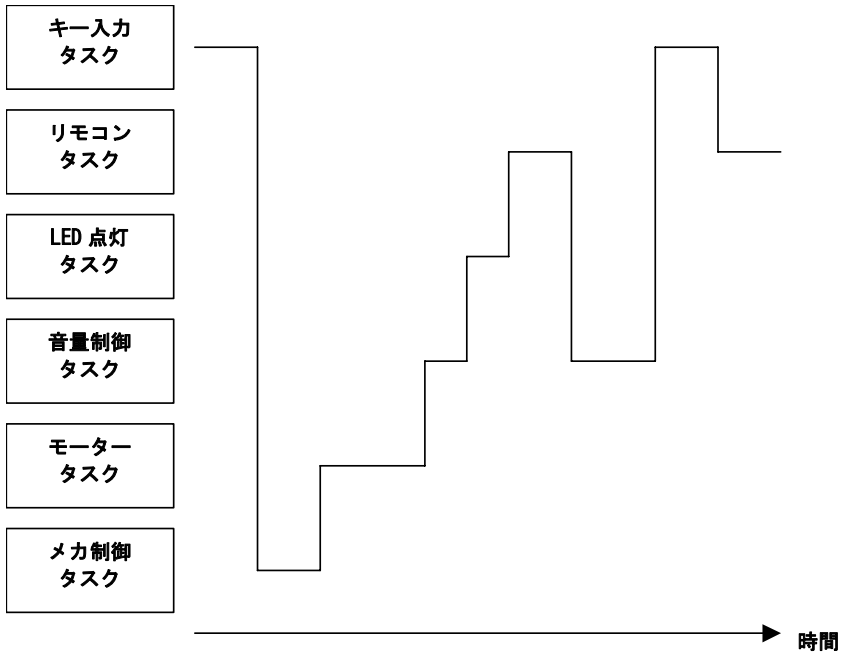


図3-1　タスクの時分割動作

このタスクを切り替えることを「ディスパッチ」と呼ぶこともあります。

タスク切り替え(ディスパッチ)は、以下の要因で発生します。

- タスクが自分自身で切り替えを要求する
- 割込みなど、タスクから見て外部の要因で切り替わる

言い換えると、一定時間毎にタスクが切り替わる(タイムシェアリング)わけではありません。このようなスケジューリングを一般に「イベントドリブン」または「イベント駆動型」と呼びます。

タスク切り替えが発生し、再度そのタスクを実行するときには、中断していたところから再開します(図 3-2)。

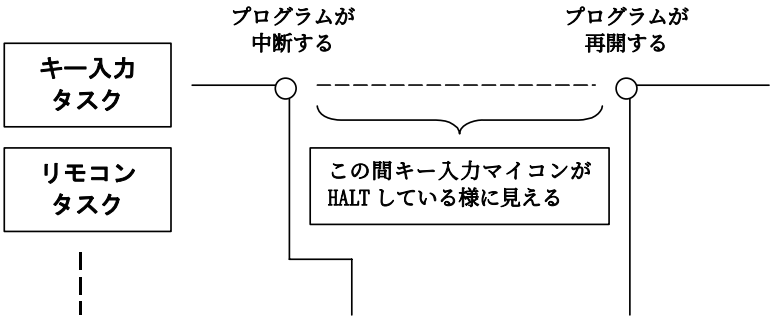


図3-2　タスクの中断と再開

図 3-2においてキー入力タスクは、他のタスクに実行制御が移っている間、プログラマから見ればプログラムが中断しそのマイコンが HALT しているようにみえます。

カーネルは、中断した時点のレジスタ内容を復帰することにより、タスクを中断した時点の状態から再開させます。すなわちタスクの切り替えとは、現在実行中のタスクのレジスタの内容をそのタスクを管理するメモリ領域に退避し、切り替えるタスクのレジスタ内容を復帰することです(図 3-3)。

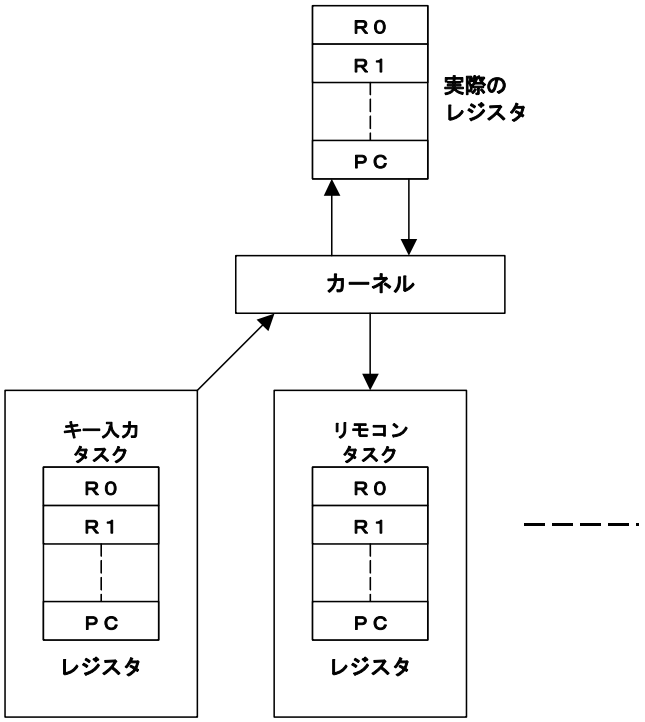


図3-3　タスクの切り替え

タスクが実行するには、レジスタだけでなく、スタック領域も必要です。スタック領域も各タスク毎に別の領域を使用します。

3.2 サービスコール

プログラマは、カーネルの機能をプログラム中でどのように使用するのでしょうか?

これにはカーネルの機能をプログラムから何らかの形で呼び出すことが必要です。カーネルの機能を呼び出すことを「サービスコール」といいます。サービスコールを呼び出すことで、タスクを起動したり、イベントを待ったり、といったことをカーネルに要求することができます。

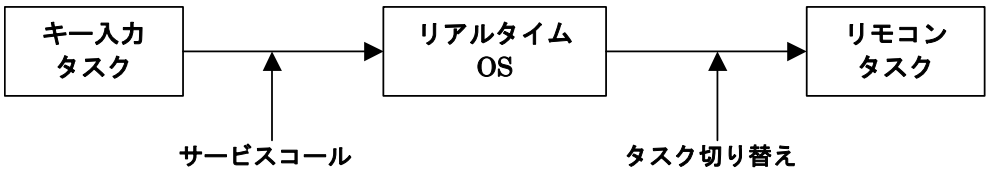


図3-4 サービスコール

実際には、サービスコールは以下のようにC言語関数もしくは、アセンブリ言語関数として呼び出します。

C言語関数

```
act_tsk(1);
```

アセンブリ言語関数

```
MOV.W  #TSKID,R0
JSR    @_act_tsk
```

3.3 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番号によって識別されます。

```
act_tsk(1);   /* タスクIDが1のタスクを起動する */
```

通常ID番号は、オブジェクトを生成するときに指定します。

この他に、コンフィギュレータでオブジェクトを生成するときには、ID番号を自動的に割り当てることもできます。この場合、コンフィギュレータがID番号を自動的に割り当てて、その結果が以下のように定義されたヘッダファイル(kernel_id.h, kernel_id.inc)を出力します。

```
#define ID_TASK1  1
```

この定義を用いると、先の例は以下のように記述できます。

```
act_tsk(ID_TASK1);   /* タスクIDが"ID_TASK1"のタスクを起動する */
```

3.4 タスク

本節ではタスクをカーネルがどのように管理しているかを説明します。

3.4.1 タスクの状態

カーネルは、タスクを実行するべきか否かを、タスクの状態を管理することにより制御しています。例えば、図3-5にキー入力タスクの実行制御と状態の関係を示します。キー入力が発生した場合はそ

のタスクを実行しなければなりません。すなわち、キー入力タスクが実行状態となります。またキー入力を待っているときはタスクを実行する必要はありません。すなわち、キー入力タスクは待ち状態になっています。

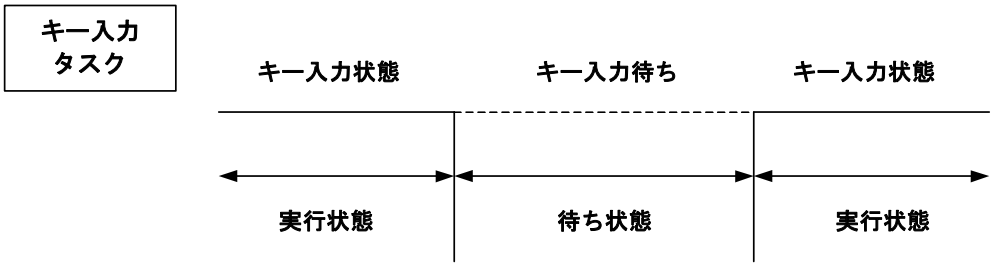


図3-5　タスクの状態

μITRON4.0 仕様では、タスクは図 3-6に示す７つの状態を遷移します。

本カーネルの状態遷移の詳細は、図 4-1タスクの状態遷移と図 4-2共有スタック機能使用時のタスク状態遷移を参照してください。

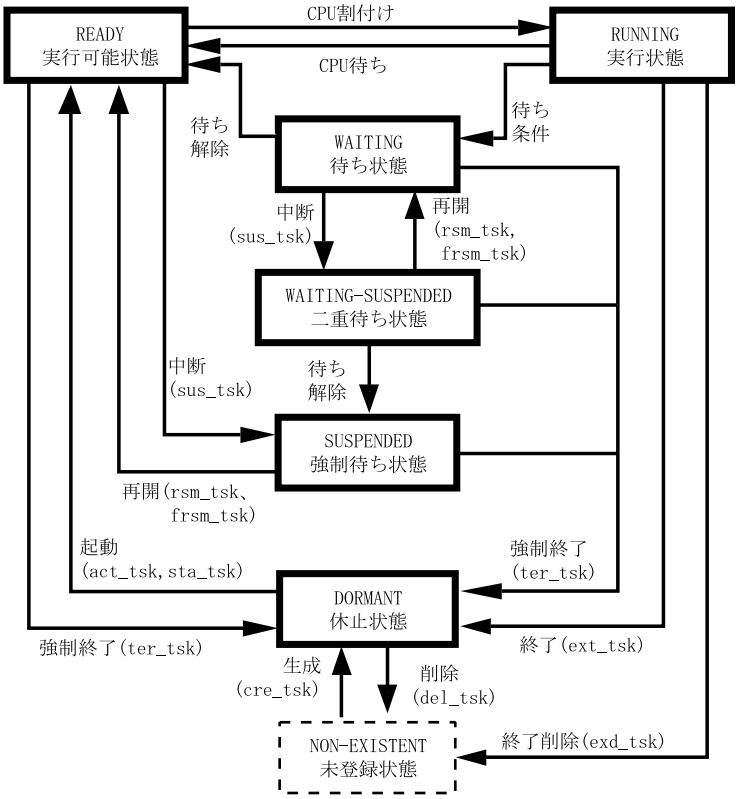


図3-6　タスクの状態遷移図

(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。

3.4.2　タスクのスケジューリング（優先度とレディキュー）

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。

図 3-7にレディキューの構造を示します。レディキューは優先度ごとに管理され、カーネルはタスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。

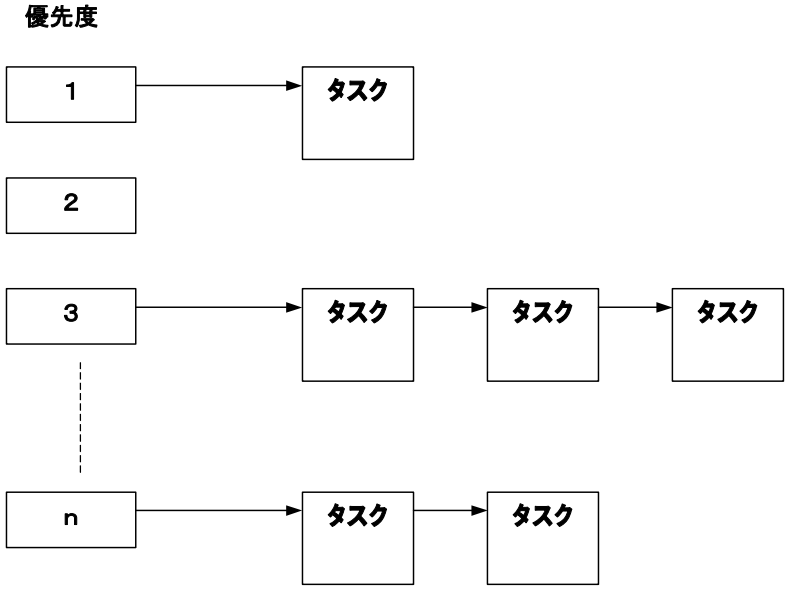


図3-7　レディーキュー(実行待ち状態)

4. カーネルの機能

4.1 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、割付け順序の制御や、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を用いて行います。

(5) メモリプール管理機能

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) システム構成管理機能

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。

4.2 処理の単位

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) 割込みハンドラ

割込みが発生したときに実行されます。

(3) CPU 例外ハンドラ

CPU 例外が発生したときに実行されます。

(4) タイムイベントハンドラ（周期ハンドラ）

指定した周期になったときに実行されます。

4.3 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。

4.3.1 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 4-1に示します。

表4-1 タスクコンテキストと非タスクコンテキスト

項　目	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	4.3.2、4.3.3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）
- 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

4.3.2 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。

4.3.3 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、システム構築ファイルに定義した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。

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（E_CTX エラーの検出を省略しています）。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
- 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。

4.4 処理の優先順位

HI1000/4 では、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（周期ハンドラ）、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替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と同じになります。

CPU 例外ハンドラの優先順位は、ディスパッチャの優先順位よりも高く、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。

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。

(a)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。

(b)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カーネル割込みマスクレベル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。

(c)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。

4.5 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、システム構築時に使用する最大番号を指定することができます。

4.6 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、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 4-2、表 4-3にタスクを操作するサービスコールを示します。

表4-2 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	act_tsk、 iact_tsk	タスクの起動	6	chg_pri	タスク優先度の変更
2	can_act	タスク起動要求のキャンセル	7	get_pri	タスク優先度の参照
3	sta_tsk、 ista_tsk	タスクの起動 (起動コード指定)	8	ref_tsk、 iref_tsk	タスクの状態参照
4	ext_tsk	自タスクの終了	9	ref_tst、 iref_tst	タスクの状態参照（簡易版）
5	ter_tsk	タスクの強制終了			

表4-3 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	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	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	強制待ち状態からの強制再開
4	can_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			

4.6.1 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 4-1に示す 6 つの状態を遷移します。

(1) 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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

(6) 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。

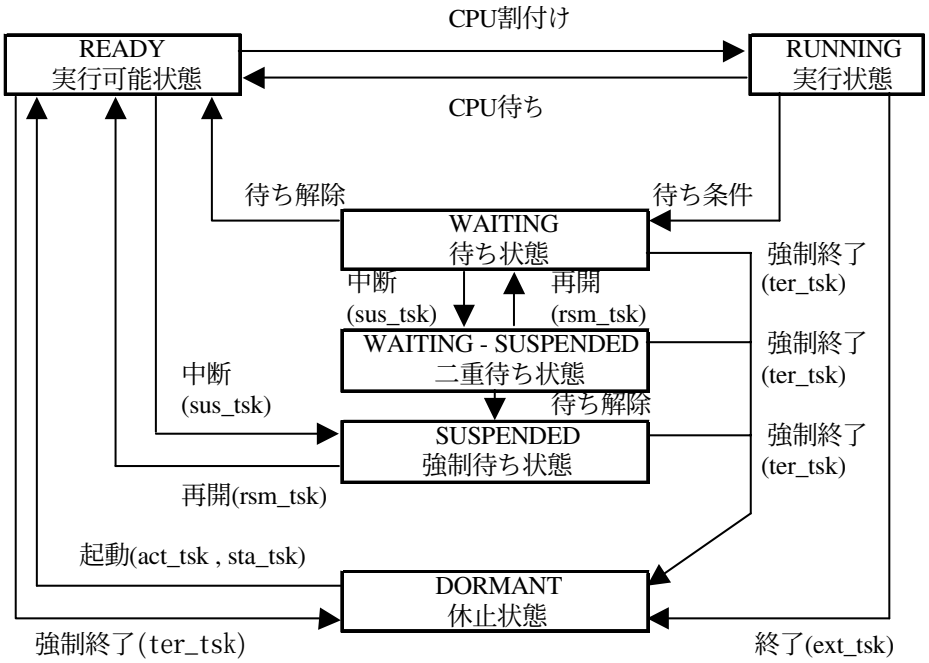


図4-1 タスクの状態遷移

4.6.2 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
- システム構築時にタスク初期状態に実行可能状態を指定する

4.6.3 タスクのスケジューリング

(1) スケジューリング

タスクのスケジューリングとは、実行可能状態のタスクを CPU に割り付けて実行させる順序を決定することです。実行可能状態のタスクの中から、1 つのタスクが選ばれ、実行状態になります。実行可能状態のタスクがない場合、カーネルはアイドリング状態となり、割込みマスクを解除して無限ループを実行することで、割込みからタスクが起床されることを待ち続けます。実行可能状態のタスクが 2 つ以上ある場合、レディキューと呼ばれる CPU 割付け待ち行列によりタスクの実行順序が決められます。

レディキューは最大タスク優先度（CFG_MAXTSKPRI）の数だけ存在し、FCFS(First Come First Service)で管理されます。タスク優先度は、値の小さい方が高い優先順位になります。

非タスクコンテキスト実行中は、タスクコンテキストに戻るまでスケジューリングは遅延されます。

(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替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。

4.6.4 タスクの終了

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

- ext_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。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 以外で、対象タスクに共有スタック待ちタスクが存在する場合は、先に共有スタック待ちタスクの待ちを解除し、自分が共有スタック待ち状態になります(タスク切り替えが発生します)。

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、タスクの終了処理で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処理を行います。

4.6.5 タスクのスタック

本カーネルではタスクスタックはシステム構築時に静的に割り付けられます。なお、複数のタスクで 1 つのスタックを共有（共有スタック機能）することもできます。

4.6.6　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容量を削減します。

共有スタックの定義は、コンフィギュレータで行います。スタックを共有するタスク群では、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。

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場合に解放され、共有スタック待ちが存在すれば、待ちの先頭タスクが占有し実行可能状態となります。

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。

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、同時に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となります。

図 4-2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。

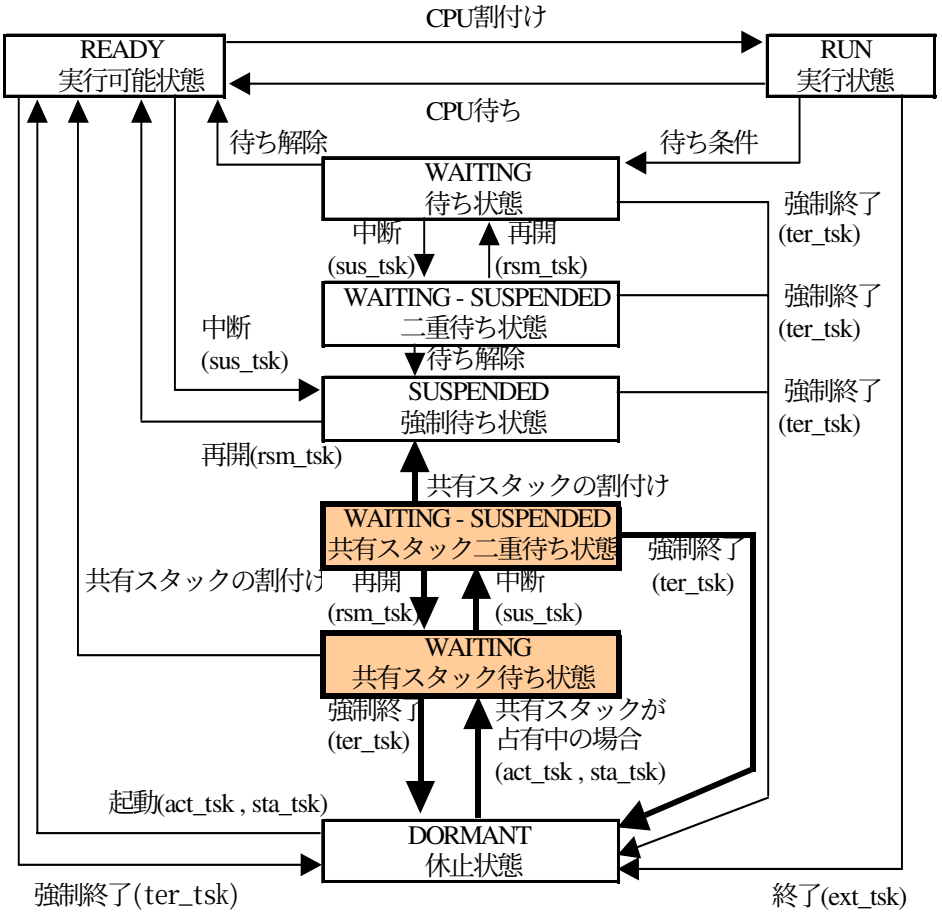


図4-2　共有スタック機能使用時のタスク状態遷移

4.6.7 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 4-4に排他制御方法を示しますので、参考にしてください。

表4-4 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。

4.7　同期・通信

本カーネルでは、同期・通信のための機能として、以下の機能を持っています。これらは、タスクとは独立したオブジェクトです。

- セマフォ　　：複数個の資源の排他制御
- イベントフラグ　：複数事象の待ち合わせ、タスクの一斉動作
- データキュー　　：データ送受信（1 ワードデータの実体の受渡し）
- メールボックス　：データ通信（データアドレスの受渡し）
- ミューテックス　：単一の資源の排他制御(優先度逆転の回避機構あり)

4.7.1　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 4-5にセマフォを操作するサービスコールを示します。

表4-5　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	sig_sem、 isig_sem	セマフォ資源の返却	4	twai_sem	セマフォ資源の獲得 (タイムアウト有)
2	wai_sem	セマフォ資源の獲得	5	ref_sem、 iref_sem	セマフォの状態参照
3	pol_sem、 ipol_sem	セマフォ資源の獲得 (ポーリング)			

セマフォは、システム構築時に初期登録することで生成されます。

図 4-3にセマフォの動作例を示します。

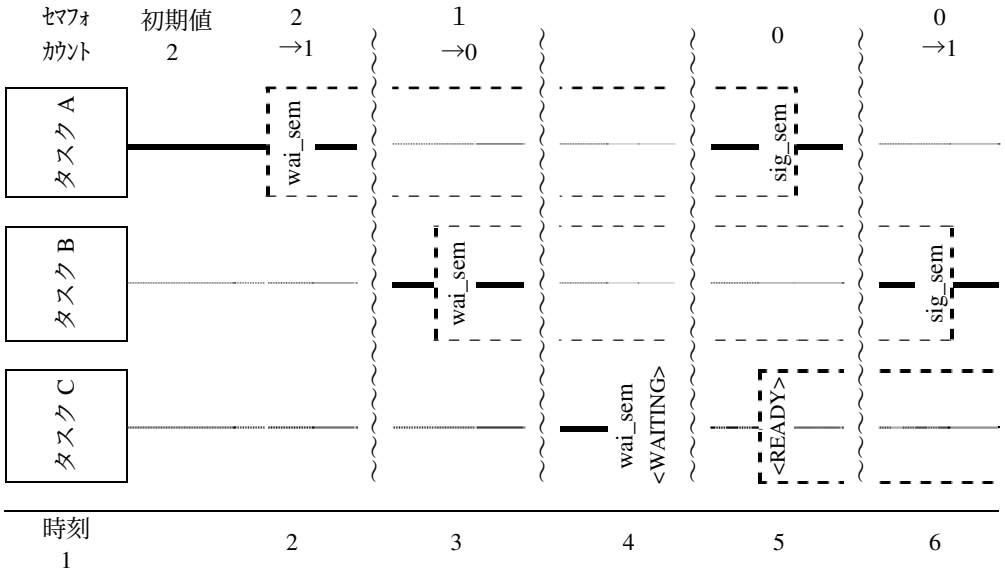


図4-3　セマフォの動作例

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. セマフォカウントの初期値が2のセマフォを初期登録します。
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。

4.7.2　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 4-6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4-6　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	set_flg、 iset_flg	イベントフラグのセット	4	pol_flg、 ipol_flg	イベントフラグ待ち (ポーリング)
2	clr_flg、 iclr_flg	イベントフラグのクリア	5	twai_flg	イベントフラグ待ち (タイムアウト有)
3	wai_flg	イベントフラグ待ち	6	ref_flg、 iref_flg	イベントフラグの状態参照

イベントフラグは、システム構築時に初期登録することで生成されます。

図 4-4にイベントフラグの動作例を示します。

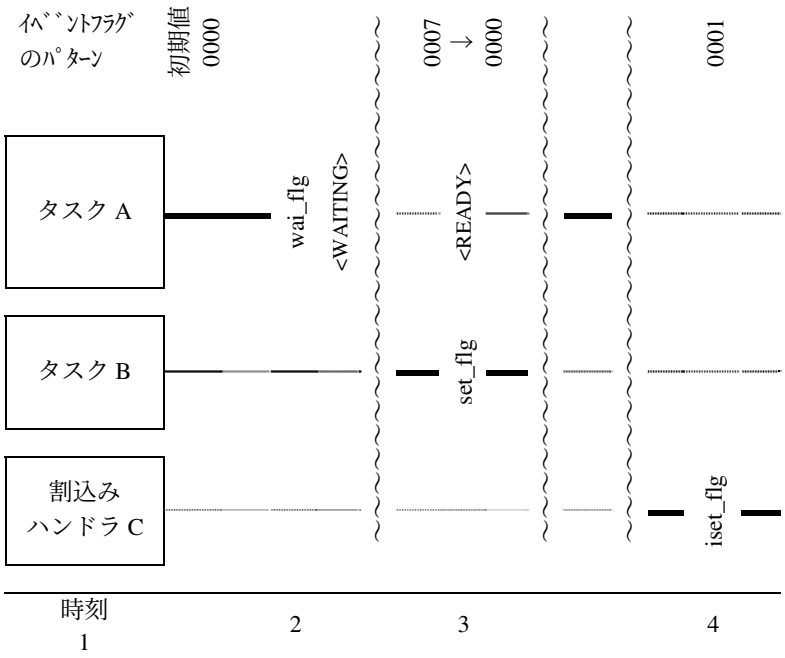


図4-4　イベントフラグの動作例

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. TA_CLR(待ち解除時にフラグパターンを0クリアする)属性のイベントフラグを初期登録します。
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。

4.7.3 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。

表 4-7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4-7　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	snd_dtq	データキューへの送信	5	rcv_dtq	データキューからの受信
2	psnd_dtq、 ipsnd_dtq	データキューへの送信 (ポーリング)	6	prcv_dtq	データキューからの受信 (ポーリング)
3	tsnd_dtq	データキューへの送信 (タイムアウト有)	7	trcv_dtq	データキューからの受信 (タイムアウト有)
4	fsnd_dtq、 ifsnd_dtq	データキューへの強制送信	8	ref_dtq、 iref_dtq	データキューの状態参照

データキューは、システム構築時に初期登録することで生成されます。

図 4-5にデータキューの動作例を示します。

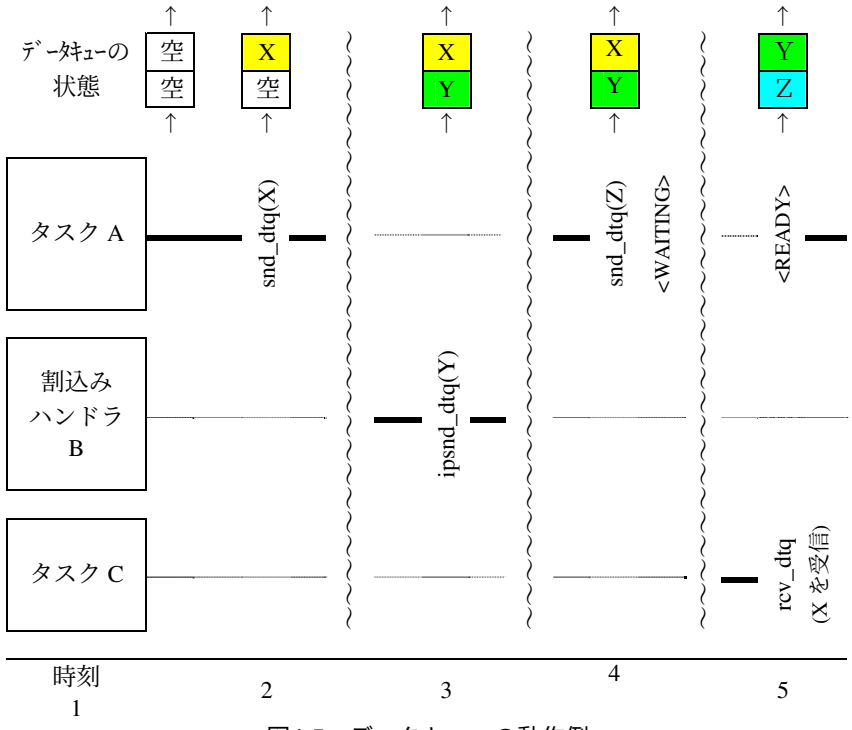


図4-5　データキューの動作例

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. 容量が2ワードのデータキューを初期登録します。
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B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。

4.7.4　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 4-8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4-8　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	snd_mbx、 isnd_mbx	メールボックスへの送信	4	trcv_mbx	メールボックスからの受信 (タイムアウト有)
2	rcv_mbx	メールボックスからの受信	5	ref_mbx、 iref_mbx	メールボックスの状態参照
3	prcv_mbx、 iprcv_mbx	メールボックスからの受信 (ポーリング)			

メールボックスは、システム構築時に初期登録することで生成されます。

図 4-6にメールボックスの動作例を示します。

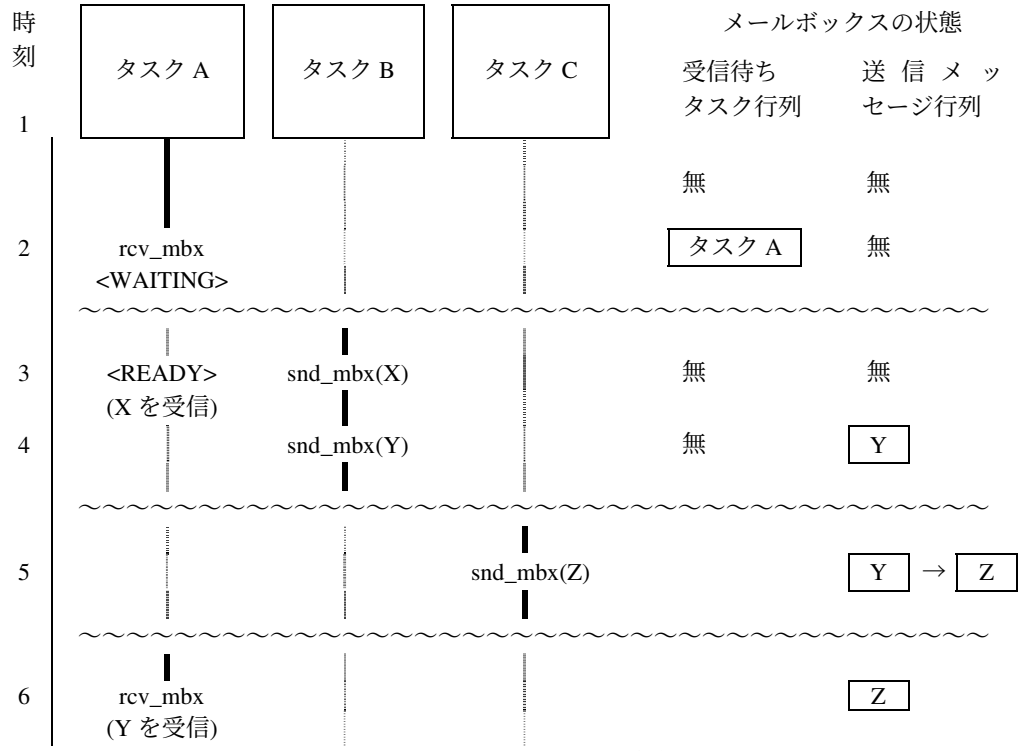


図4-6　メールボックスの動作例

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. 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)属性のメールボックスを初期登録します。
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Y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。

4.7.5 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されます。

表 4-9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4-9　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	loc_mtx	ミューテックスのロック	4	unl_mtx	ミューテックスのロック解除
2	ploc_mtx	ミューテックスのロック(ポーリング)	5	ref_mtx	ミューテックスの状態参照
3	tloc_mtx	ミューテックスのロック(タイムアウト有)			

ミューテックスは、システム構築時に初期登録することで生成されます。

図 4-7にミューテックスの動作例を示します。

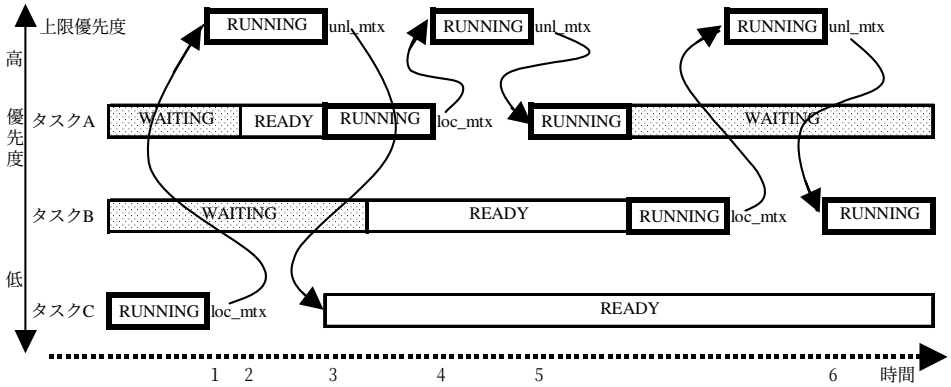


図4-7　ミューテックスの動作例

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。

4.8 メモリプール

メモリプールを使用することにより、空きメモリを効率的に使用することができます。
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。

4.8.1 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、メモリプールの初期登録時に指定します。
表 4-10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4-10 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	get_mpf	固定長メモリブロックの獲得	4	rel_mpf	固定長メモリブロックの返却
2	pget_mpf、 ipget_mpf	固定長メモリブロックの獲得 (ポーリング)	5	ref_mpf、 iref_mpf	固定長メモリプールの 状態参照
3	tget_mpf	固定長メモリブロックの 獲得(タイムアウト有)			

固定長メモリプールは、システム構築時に初期登録することで生成されます。

図 4-8に固定長メモリプールの動作例を示します。

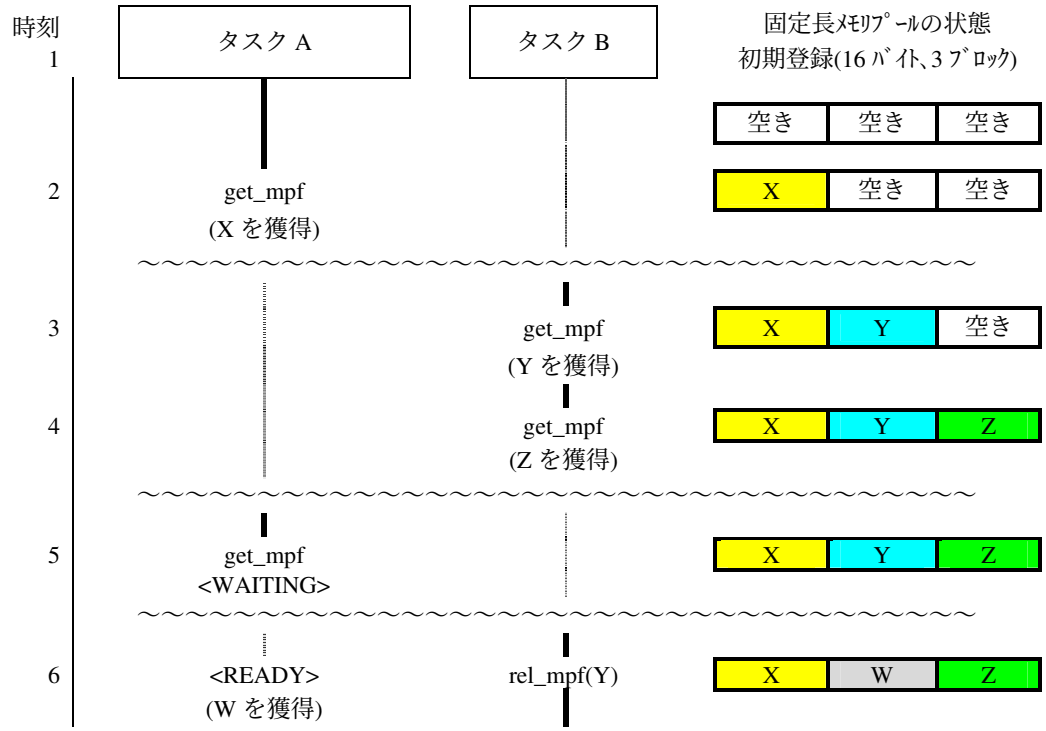


図4-8　固定長メモリプールの動作例

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. 16バイトのブロックが3つの固定長メモリプールを初期登録します。
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A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Wが割り当てられます。

4.8.2 可変長メモリプール

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。

表 4-11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4-11 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	get_mpl	可変長メモリブロックの獲得	4	rel_mpl	可変長メモリブロックの返却
2	pget_mpl、 ipget_mpl	可変長メモリブロックの獲得 (ポーリング)	5	ref_mpl、 iref_mpl	可変長メモリプールの 状態参照
3	tget_mpl	可変長メモリブロックの 獲得(タイムアウト有)			

可変長メモリプールは、システム構築時に初期登録することで生成されます。

図 4-9に可変長メモリプールの動作例を示します。

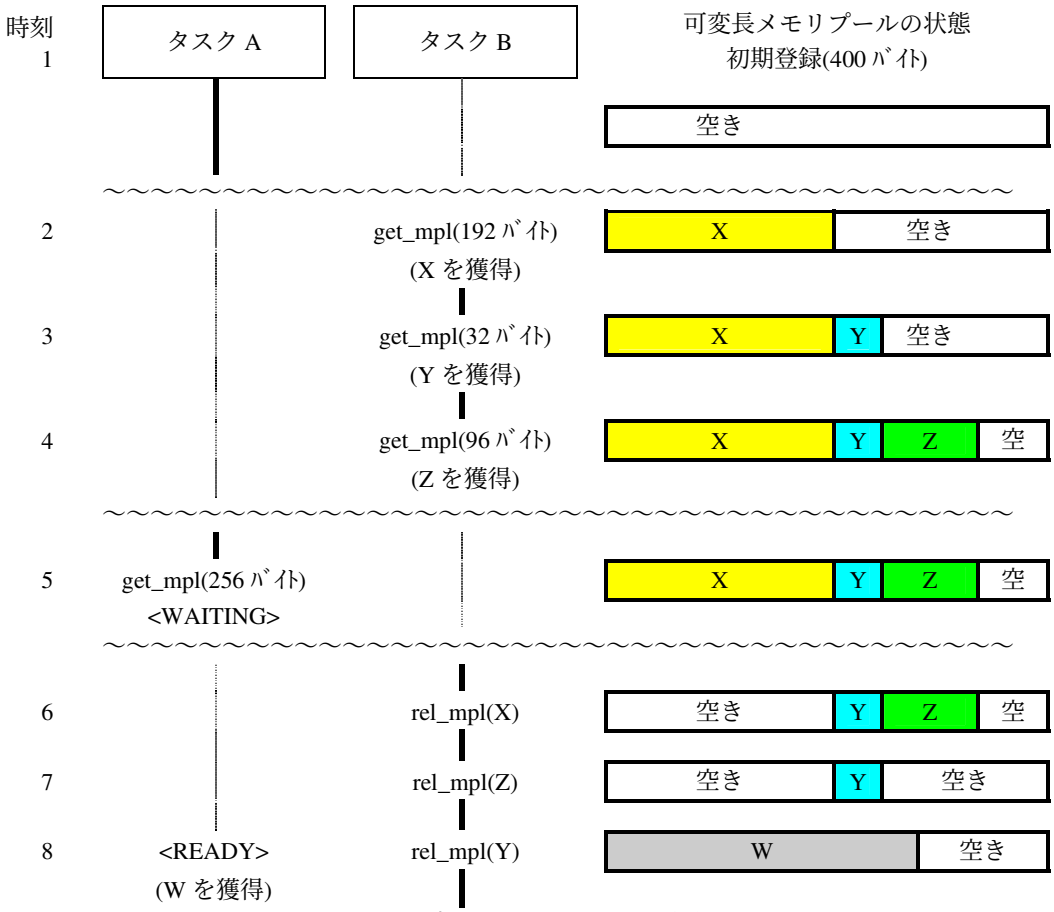


図4-9　可変長メモリプールの動作例

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. 400バイトの可変長メモリプールを初期登録します。
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。

4.9 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）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、システム時刻の周期は、システム構築でタイムティック周期の分子 (TIC_NUME) 、およびタイムティック周期の分母 (TIC_DENO)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、いくつかの準備が必要です。「6.8 タイマドライバ」を参照してください。

表 4-12にシステム時刻を操作するサービスコールを示します。HI1000/4 では、システム時刻を更新する機構をシステムで独自に用意しています。

表4-12　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	set_tim、iset_tim	システム時刻の設定	2	get_tim、iget_tim	システム時刻の参照

4.9.1 周期ハンドラ

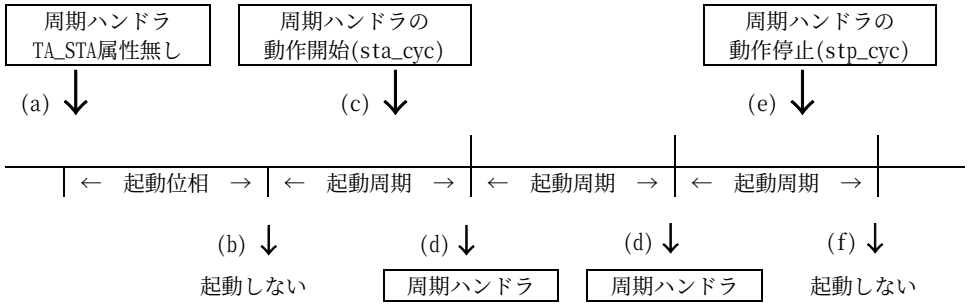
周期ハンドラは、指定した起動位相経過後、起動周期ごとに起動されるタイムイベントハンドラです。表 4-13に周期ハンドラを操作するサービスコールを示します。

表4-13　周期ハンドラを操作するサービスコール

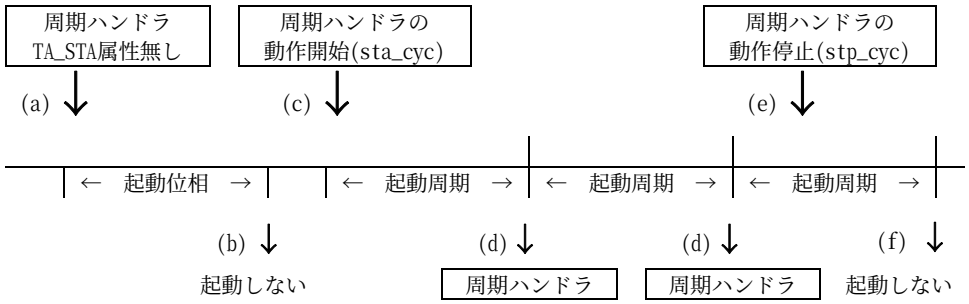
項番	ｻｰﾋﾞｽｺｰﾙ	操作内容	項番	ｻｰﾋﾞｽｺｰﾙ	操作内容
1	sta_cyc、ista_cyc	周期ハンドラの動作開始	3	ref_cyc、iref_cyc	周期ハンドラの状態参照
2	stp_cyc、istp_cyc	周期ハンドラの動作停止			

周期ハンドラは、システム構築時に初期登録することで生成されます。

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、周期ハンドラの初期登録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 4-10に周期ハンドラの動作例を示します。



(1) 起動位相を保存する場合



(2) 起動位相を保存しない場合

図4-10 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）を初期登録します。

(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、周期ハンドラ初期登録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。

4.9.2 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) 周期時間に達した全ての周期ハンドラの起動と実行
(c) 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)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。

(3) タイムティック周期の設定

タイムティック周期の分子（TIC_NUME）を 1 以外に設定したシステムにおいて、周期ハンドラの周期時間、起動位相に TIC_NUME 以下の値を設定した場合、周期ハンドラの起動時間にキャリーが発生し、同一タイムティック処理で複数回周期ハンドラが起動されるため、(1)に示す弊害が発生する可能性があります。

具体的には、タイムティック周期の分子（TIC_NUME）を 10 に設定したシステムにおいて、周期ハンドラの周期時間に 5 を設定した場合、1 回のタイムティック処理で同一周期ハンドラが 2 回起動されることになり、(1)に示す弊害が発生する可能性があります。

そのため、タイムアウト時間、周期時間、起動位相時間は、タイムティック周期の分子（TIC_NUME）の倍数の値を設定することを推奨します。

4.10 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 4-14に示すサービスコールをサポートしています。

表4-14 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	タスクの優先順位の回転	8	sns_loc	CPU ロック状態の参照
2	get_tid、 iget_tid	実行状態のタスク ID の参照	9	sns_dsp	ディスパッチ禁止状態の参照
3	loc_cpu、 iloc_cpu	CPU ロック状態への移行	10	sns_dpn	ディスパッチ保留状態の参照
4	unl_cpu、 iunl_cpu	CPU ロック状態の解除	11	vsta_knl、 ivsta_knl	カーネルの起動
5	dis_dsp	ディスパッチの禁止	12	vsys_dwn、 ivsys_dwn	システムダウン
6	ena_dsp	ディスパッチの許可	13	ivbgn_int	割込みハンドラ開始のトレース取得
7	sns_ctx	コンテキストの参照	14	ivend_int	割込みハンドラ終了のトレース取得

4.10.1 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 または 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、6.10節を参照してください。

4.10.2 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、システム実行中のサービスコール発行履歴をトレースバッファに保存する機能です。基本的には 1 回のサービスコール発行により、発行時とリターン時の 2 つの情報が取得されます。この情報を「イベント」と呼びます。

サービスコールトレース機能を使用するには、システム構築時にコンフィギュレータでサービスコールトレース機能の組み込みを選択する必要があります。さらにトレース機能の選択およびトレース取得数の設定を行うことで、ユーザ環境に応じた設定が可能となります。

トレース機能を組み込んだ場合、システム起動後、初期化ルーチン以降のすべてのイベントが取得されます。トレース機能の選択でターゲットトレースもしくはターゲットトレース(短縮)を選択した場合は、トレースデータはトレースバッファに格納されます。トレースバッファはリングバッファ構造となっており、古い情報は順次新しい情報に書き替えられます。

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。

(1) 履歴情報の保存場所

履歴情報は、ターゲットメモリまたはデバッガ（E6000H 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「ツールトレース」と呼びます。後者は、E6000H エミュレータ、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長があります。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ビューで行います。
サービスコールトレース機能のタイプには以下の 4 種類があります。

- ターゲットトレース

  全てのサービスコールトレース情報（属性、タスクID、イベント情報、パラメータ、発行元EXR,CCR,PC）をシステム内に確保したトレースバッファ領域に取得します。

- ツールトレース

  全てのサービスコールトレース情報（属性、タスクID、イベント情報、パラメータ、発行元EXR,CCR,PC）をツール（シミュレータ・デバッガまたはエミュレータ）のトレースメモリに取得します。

- ターゲットトレース(短縮)

  必要最小限のサービスコールトレース情報（属性、タスクID、イベント情報）をシステム内に確保したトレースバッファ領域に取得します。

- ツールトレース(短縮)

  必要最小限のサービスコールトレース情報（属性、タスクID、イベント情報）をツール（シミュレータ・デバッガまたはエミュレータ）のトレースメモリに取得します。

(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

以下のサービスコールは取得されません。

vsta_knl、ivsta_knl、vsys_dwn、ivsys_dwn

また、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。

4.11　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：一般不当命令による例外です。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。

例外・割込みが発生すると、ベクタテーブルに登録された CPU 例外ハンドラ、割込みハンドラが直接起動されます。

表 4-15に割込み管理機能のサービスコールを、表 4-16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4-15　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	chg_ims、 ichg_ims	割込みマスクの変更	2	get_ims、 iget_ims	割込みマスクの参照

表4-16　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
1	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、システム構築時にベクタテーブルに登録します。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。

4.11.1　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 4-11に示すようになります。

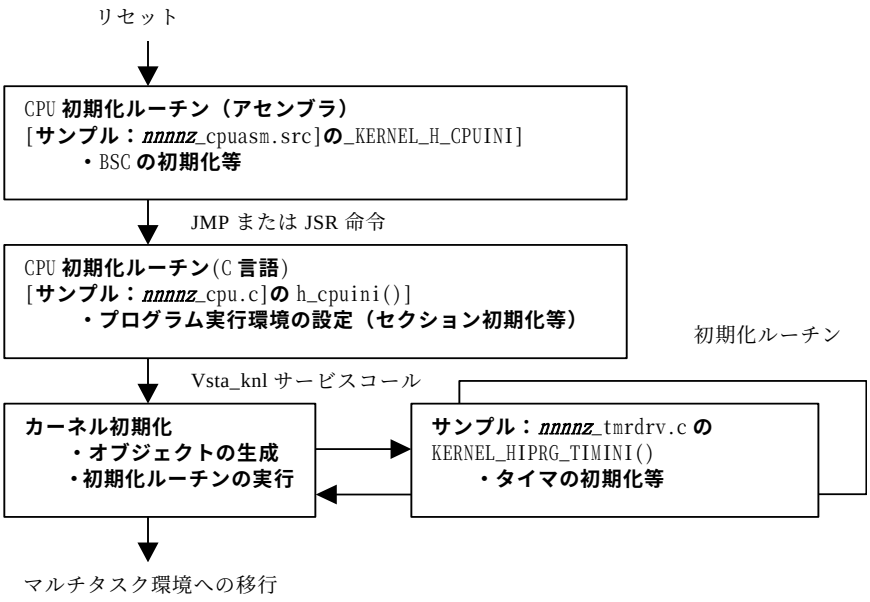


図4-11　リセットからカーネル起動までの流れ

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、『H8S,H8/300 シリーズ C/C++コンパイラパッケージ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)　割込みマスクレベルをカーネル割込みマスクレベルに設定
(b)　カーネル作業領域の初期化
(c)　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
(d)　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
(e)　マルチタスク環境に移行します。

CPU初期化ルーチンの作成については、「6.9 CPU初期化ルーチン」を参照してください。

4.11.2 割 込 み

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。

(1) 割込みマスクの変更

割込みマスクを変更する方法には以下があります。

(a) chg_imsサービスコール

(b) CCRレジスタ、EXRレジスタの割込みマスクビットを直接変更（Cコンパイラの組込み関数set_imask_ccr( )、set_ccr( )、set_imask_exr( )、set_exr( )）

割込みマスクを 0 から 0 以外に変更する場合、および 0 以外から 0 に変更する場合には、必ず(a)の方法を使用してください。その他の場合には、(a)、(b)どちらの方法でも構いません。

また、(a)の方法で割込みマスクを 0 から x に変更した後に、(b)の方法で割込みマスクを x 以外に変更した場合は、(a)で割込みマスクを 0 に戻す前に、割込みマスクを x に戻さなければなりません。

また、割込みハンドラ実行中は割込みマスクを自レベルよりも低いレベルにしてはなりません。

(2) カーネル割込みマスクレベル

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、システム構築時に設定します。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。

4.11.3 CPU 例外

CPU 例外ハンドラ（TRAPA 命令例外含む）は、非タスクコンテキストとして動作します。スタックは例外発生元のものを使用します。CPU 例外ハンドラは、非タスクコンテキストであり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
- iget_tid
- ivsta_knl
- ivsys_dwn

5. サービスコール

5.1 概 要

各サービスコールの機能は、表 5-1のように分類されます。

表5-1 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、再開など
3	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
4	拡張同期・通信機能	ミューテックス
5	メモリプール管理機能	動的なメモリの割り付け
6	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、周期ハンドラの管理など
7	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
8	割込み管理機能	割込みマスクの変更と参照など
9	システム構成管理機能	バージョン情報の参照など

5.2 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すことができます。

5.2.1 ヘッダファイル

サービスコールを使用するにはいくつかのヘッダファイルをインクルードする必要があります。ヘッダファイルについては、「6.1 ヘッダファイル」を参照してください。

5.2.2 C 言語 API

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。

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

また、パラメータのデータ型とサイズを以下に示します。これらは C 言語用ヘッダファイルで定義しています。

```
typedef char            B;          /* 符号付き 8 ビット整数                          */
typedef short           H;          /* 符号付き 16 ビット整数                         */
typedef long            W;          /* 符号付き 32 ビット整数                         */
typedef unsigned char   UB;         /* 符号無し 8 ビット整数                          */
typedef unsigned short  UH;         /* 符号無し 16 ビット整数                         */
typedef unsigned long   UW;         /* 符号無し 32 ビット整数                         */
typedef char            VB;         /* データタイプが定まらない 8 ビットの値          */
typedef short           VH;         /* データタイプが定まらない 16 ビットの値         */
typedef long            VW;         /* データタイプが定まらない 32 ビットの値         */
typedef void            *VP;        /* データタイプが定まらないものへのポインタ       */
typedef void            (*FP)();    /* プログラムの起動番地                           */
typedef H               INT;        /* 符号付き 16 ビット整数                         */
typedef UH              UINT;       /* 符号無し 16 ビット整数                         */
typedef H               BOOL;       /* 真偽値(TRUE または FALSE)                      */
typedef H               FN;         /* 機能コード(符号付き整数)                       */
typedef H               ER;         /* エラーコード(符号付き整数)                     */
typedef H               ID;         /* オブジェクトの ID 番号(符号付き整数)           */
typedef UH              ATR;        /* オブジェクト属性(符号無し整数)                 */
typedef UH              STAT;       /* オブジェクトの状態(符号無し整数)               */
typedef UH              MODE;       /* サービスコールの動作モード(符号無し整数)       */
typedef H               PRI;        /* 優先度(符号付き整数)                           */
typedef UW              SIZE;       /* メモリ領域のサイズ(符号無し整数)               */
typedef W               TMO;        /* タイムアウト指定(符号付き整数)                 */
typedef UW              RELTIM;     /* 相対時間(符号無し整数)                         */
typedef W               VP_INT;     /* データタイプが定まらないものへのポインタ
                                                       または符号付き整数       */
typedef H               ER_BOOL;    /* エラーコードまたは真偽値                       */
typedef H               ER_ID;      /* エラーコードまたは ID 番号                     */
typedef H               ER_UINT;    /* エラーコードまたは符号無し整数                 */
typedef UH              FLGPTN;     /* イベントフラグのビットパターン(符号無し整数)   */
typedef UH              INHNO;      /* 割込みハンドラ番号                             */
typedef UH              IMASK;      /* 割込みマスク                                   */
typedef UH              EXCNO;      /* CPU 例外ハンドラ番号                           */
```

5.2.3 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5-1のようにして呼び出します。

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場合のパラメータについては、個々のサービスコールの説明を参照してください。

```
   .INCLUDE "kernel.inc"      ←標準ヘッダ"kernel.inc"をインクルードしてください。
; ……
   .import  _act_tsk
;
TSKID: .assign 1
;
Task_a:
   MOV.W  #TSKID,R0           ←パラメータの設定を行います。
   JSR    @_act_tsk           ←サービスコールを呼び出します。
; ……                        ←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 jsr によってスタックに退避されたアドレスにリターンしま
   .end                        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5-1 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

5.2.4 引数格納レジスタ数

HI1000/4 のサービスコール（関数）は、引数用レジスタ数（キーワード_ _regparam3 の設定）が 3 であることを前提にしています。そのため、サービスコールは、キーワード「_ _regparam3」を指定してください。**サンプル提供のヘッダファイルは、サービスコールの原型宣言で、引数用レジスタ数を 3 に指定しています。**

サービスコールの引数用レジスタ数を 3 に指定しないでサービスコールを発行した場合の動作は保証されません。

5.2.5 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 H8S,H8/300H C/C++コンパイラに準じていますが、その詳細を表 5-2に示します。

表5-2 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	ER3～ER6、SBR(H8SX のみ)	保証されます。
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	ER0～ER2	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。

【注】 * キーワード「_ _regparam3」を指定した場合のレジスタ保証規則です。
タスク、ハンドラ作成時のレジスタ保証については、「6. 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。

5.2.6　サービスコールの返値とエラーコード

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

5.2.7　パラメータとリターンパラメータ

リターンパラメータを持つサービスコールでは、パラメータに指定した領域にリターンパラメータが格納されます。第 1 引数がリターンパラメータであるサービスコールでは、リターンパラメータは ER0 レジスタで渡されます。しかし、サービスコール処理終了後は、R0 にエラーコードが返るためリターンパラメータ ER0 は破壊されます。C 言語でプログラムを記述する場合は意識する必要はありませんが、アセンブラでプログラムを記述する際は、サービスコール発行前にリターンパラメータを格納する領域を保証されるレジスタに保持しておき、サービスコール終了後に保持しておいたレジスタの指す領域からリターン値を得るようにプログラミングする必要があります。

5.2.8　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)　非タスクコンテキスト専用サービスコールは、"i"で始まるサービスコールです。

(b)　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は、"sns"で始まるサービスコールです。

(c)　それ以外の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。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（E_CTX エラーの検出を省略しています）。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
- 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（E_CTX エラーの検出を省略しています）。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_ctx
- sns_loc
- sns_dsp
- sns_dpn
- iget_tid
- ivsta_knl
- 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。

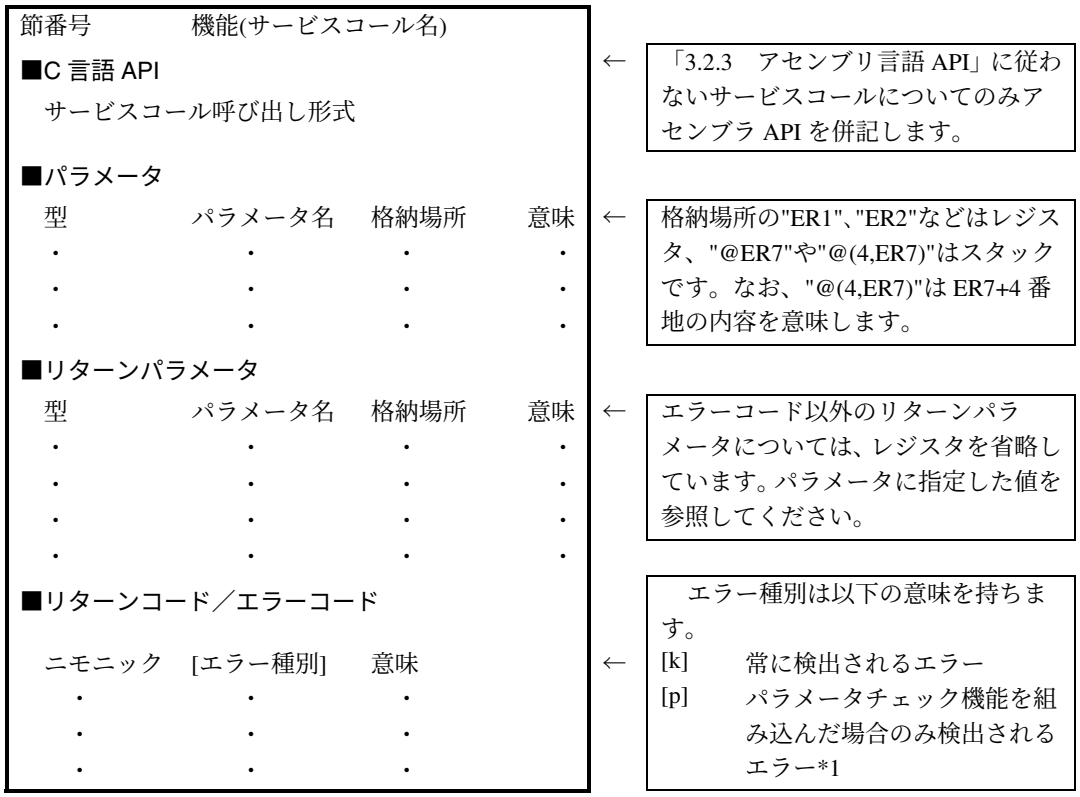
- vsta_knl、ivsta_knl
- vsys_dwn、ivsys_dwn

5.2.9 μITRON4.0 仕様範囲外のサービスコール

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。

5.3　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 5-2の形式で説明します。



・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;             +2        2       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
```

*1：**パラメータチェック機能を組み込まない場合に、[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しません。**

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5-2　サービスコールの説明形式

5.4 タスク管理機能

表 5-3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5-3 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	act_tsk	[S]	タスクの起動	○		○	○	○		
	iact_tsk	[S]			○	○	○	○		
2	can_act	[S]	タスク起動要求のキャンセル	○		○	○	○		
3	sta_tsk		タスクの起動(起動コード指定)	○		○	○	○		
	ista_tsk				○	○	○	○		
4	ext_tsk	[S]	自タスクの終了	○		○	○	○	○	
5	ter_tsk	[S]	タスクの強制終了	○		○	○	○		
6	chg_pri	[S]	タスク優先度の変更	○		○	○	○		
7	get_pri	[S]	タスク優先度の参照	○		○	○	○		
8	ref_tsk		タスクの状態参照	○		○	○	○		
	iref_tsk				○	○	○	○		
9	ref_tst		タスクの状態参照（簡易版）	○		○	○	○		
	iref_tst				○	○	○	○		

【注】 *1 "[S]"はスタンダードプロファイルの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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-4にタスク管理の仕様を示します。

表5-4 タスク管理の仕様

項番	項目	内容
1	タスク ID	1～CFG_MAXTSKID (最大 255)
2	タスク優先度	1～CFG_MAXTSKPRI (最大 31)
3	タスク起動要求キューイング数の最大値	255 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
5	タスクスタック	共有スタック機能あり
6	共有スタック待ち行列(共有スタック機能使用時)	FIFO(First-In First-Out)

5.4.1 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	R0	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	[p]	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＞H'ff)
E_CTX	[p]	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。

tskid=TSK_SELF（0）の指定により、自タスクの指定になります。

タスクには、タスクの初期登録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。

共有スタック機能を使用時には、tskid で示されたタスクのスタックが解放（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 255 回まで記憶されます。

5.4.2　タスクの起動要求のキャンセル(can_act)

■C 言語 API

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

■パラメータ

ID	tskid	R0	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)
E_NOEXS	[p]	未登録(tskid のタスクが生成されていない)
E_CTX	[p]	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。

5.4.3　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	R0	タスク ID
VP_INT	stacd	ER1	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(tskid のタスクが休止状態でない、または自タスク指定)
E_CTX	[p]	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。

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。

共有スタック機能を使用時には、tskid で示されたタスクのスタックが解放（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。

5.4.4　自タスクの終了(ext_tsk)

■C 言語 API

```
void ext_tsk();
```

■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。

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　　[p]　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。ただし、共有スタック待ちタスクが存在する場合は、共有スタック待ちタスクの待ちを解除し、自タスクは共有スタック待ち状態になります。

また、タスクがミューテックスをロックしている場合は、ロックしているミューテックスのロックを解除します。さらに、そのミューテックスにロック解除待ちタスクが存在する場合は、ロック待ちタスクの待ちを解除し、そのタスクを実行可能状態に移行します。

ext_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 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てスタック待ち状態を解除し、そのタスクを実行可能状態に移行します。

ext_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。

5.4.5 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	R0	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(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	[p]	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。

起動要求がキューイングされている場合には、対象タスクを実行可能状態に移行します。

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、その共有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てスタック待ち状態を解除し、そのタスクを実行可能状態に移行します。

また、タスクがミューテックスをロックしている場合は、ロックしているミューテックスのロックを解除します。さらに、そのミューテックスにロック解除待ちタスクが存在する場合は、ロック待ちタスクの待ちを解除し、そのタスクを実行可能状態に移行します。

5.4.6　タスク優先度の変更(chg_pri)

■C 言語 API

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

■パラメータ

ID	tskid	R0	タスク ID
PRI	tskpri	E0	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)
E_NOEXS	[p]	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]	オブジェクト状態エラー(タスクが休止状態である)
E_CTX	[p]	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、タスク初期登録時に指定したタスクの起動時優先度に戻します。

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定義時に指定したタスクの起動時優先度になります。

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場合、または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。

5.4.7　タスク優先度の参照(get_pri)

■C 言語 API

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

■パラメータ

ID	tskid	R0	タスク ID
PRI	*p_tskpri	ER1	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	--	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 0)
E_ID	[p]	不正 ID 番号(tskid＜0、tskid＞CFG_MAXTSKID)
E_NOEXS	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(タスクが休止状態である)
E_CTX	[p]	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 で示されたタスクの現在優先度を参照し、tskpri 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。

5.4.8 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	R0	タスク ID
T_RTSK	*pk_rtsk	ER1	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	--	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    2    タスク状態
           PRI    tskpri;       +2    2    タスクの現在優先度
           PRI    tskbpri;      +4    2    タスクのベース優先度
           STAT   tskwait;      +6    2    待ち要因
           ID     wobjid;       +8    2    待ちオブジェクト ID
           TMO    lefttmo;      +10   4    タイムアウトするまでの時間
           UINT   actcnt;       +14   2    起動要求キューイング数
           UINT   wupcnt;       +16   2    起床要求キューイング数
           UINT   suscnt;       +18   2    強制待ち要求ネスト数
} T_RTSK;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtsk が 0)
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	[p]	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

現在のタスク状態です。tskstat には、次の値を返します。

- TTS_RUN （H'0001） 実行状態
- TTS_RDY （H'0002） 実行可能状態
- TTS_WAI （H'0004） 待ち状態
- TTS_SUS （H'0008） 強制待ち状態
- TTS_WAS （H'000c） 二重待ち状態
- TTS_DMT （H'0010） 休止状態
- TTS_STK （H'4000） 共有スタック解放待ち状態

タスクが強制待ち状態でかつ共有スタック解放待ち状態の場合は TTS_SUS｜TTS_STK（H'4008）を返します。また、tsksts の値が H'0000 で返る場合があります。これはカーネル実行中を表します。この場合、pk_rtsk で返る他の情報は不定となります。

♦ tskpri

タスクの現在優先度です。タスクが休止状態の場合は、タスクの起動時優先度を返します。

♦ 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、タスクの起動時優先度を返します。

♦ 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'0001） slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY （H'0002） dly_tsk サービスコールによる待ち
- TTW_SEM （H'0004） wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG （H'0008） wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ （H'0010） snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ （H'0020） rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX （H'0040） rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX （H'0080） loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_MPF （H'2000） get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL （H'4000） get_mpl、tget_mpl 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。

5.4.9　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	R0	タスク ID
T_RTST	*pk_rtst	ER1	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	--	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     2    タスク状態
           STAT   tskwait;      +2     2    待ち要因
} T_RTST;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtst が 0)
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	[p]	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最低限の状態を参照し、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

現在のタスク状態です。tskstat には、次の値を返します。

- TTS_RUN　（H'0001）　実行状態
- TTS_RDY　（H'0002）　実行可能状態
- TTS_WAI　（H'0004）　待ち状態
- TTS_SUS　（H'0008）　強制待ち状態
- TTS_WAS　（H'000c）　二重待ち状態
- TTS_DMT　（H'0010）　休止状態
- TTS_STK　（H'4000）　共有スタック解放待ち状態
-

タスクが強制待ち状態でかつ共有スタック解放待ち状態の場合は TTS_SUS｜TTS_STK（H’4008）を返します。
また、tsksts の値が H’0000 で返る場合があります。これはカーネル実行中を表します。この場合、

pk_rtst で返る他の情報は不定となります。

♦ 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'0001） slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY （H'0002） dly_tsk サービスコールによる待ち
- TTW_SEM （H'0004） wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG （H'0008） wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ （H'0010） snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ （H'0020） rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX （H'0040） rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX （H'0080） loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_MPF （H'2000） get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL （H'4000） get_mpl、tget_mpl サービスコールによる待ち

5.5 タスク付属同期機能

表 5-5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5-5 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	○		○	○	○		
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	○		○	○	○		
7	rsm_tsk	[S]	強制待ち状態からの再開	○		○	○	○		
8	frsm_tsk	[S]	強制待ち状態からの強制再開	○		○	○	○		
9	dly_tsk	[S]	自タスクの遅延	○		○		○		

【注】 *1 "[S]"はスタンダードプロファイルの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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-6にタスク付属同期の仕様を示します。

表5-6 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	255 回
2	タスク強制待ち要求ネスト数の最大値	1 回

5.5.1 起床待ち(slp_tsk、tslp_tsk)

■C 言語 API

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

■パラメータ

《tslp_tsk》

TMO	tmout	ER0	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_NOSPT	[p]	未サポート機能(時間管理機能未使用(tslp_tsk))
E_PAR	[p]	パラメータエラー(tmout≦-2)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、または CPU ロック状態からの tsl_tsk の呼び出しで tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	タイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

■機能

自タスクを起床待ち状態に移行します。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 を返します。

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。

タイムティック周期時間の分母(CFG_TICDENO)に 1 より大きな値を設定した場合は、tmout に指定可能な値は最大値は H'7fffffff/タイムティック周期時間の分母に制限されます。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。

5.5.2　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	R0	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(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＞H'ff)
E_CTX	[p]	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

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 255 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。

5.5.3　タスク起床要求のキャンセル(can_wup)

■C 言語 API

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

■パラメータ

ID	tskid	R0	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)
E_NOEXS	[p]	未登録(tskid のタスクが生成されていない)
E_OBJ	[k]	オブジェクト状態エラー(tskid のタスクが休止状態である)
E_CTX	[p]	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。

5.5.4 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	R0	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー（tskid のタスクが待ち状態でない、または自タスク ID を指定）
E_CTX	[p]	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 を返します。

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 または frsm_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 を返します。

なお、本サービスコールで共有スタック待ち状態を解除することはできません。

5.5.5 強制待ち状態への移行(sus_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	R0	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(tskid のタスクが休止状態である)
E_CTX	[p]	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)
	[k]	(タスクコンテキストかつディスパッチ禁止状態、またはタスクコンテキストかつ CPU ロック状態で 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(既に強制待ち状態である)

■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 または frsm_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要求はネストしません。

5.5.6 強制待ち状態からの再開(rsm_tsk)
強制待ち状態からの強制再開(frsm_tsk)

■C 言語 API

```
ER ercd = rsm_tsk(ID tskid);
ER ercd = frsm_tsk(ID tskid);
```

■パラメータ

ID	tskid	R0	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	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	[k]	オブジェクト状態エラー(tskid のタスクが強制待ち状態でない)
E_CTX	[p]	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 で示されたタスクの強制待ち状態を解除します。二重待ち状態の場合は、待ち状態に移行します。

HI1000/4 では、強制待ち要求がネストしないため、rsm_tsk、frsm_tsk サービスコールとも同じ処理を行います。

5.5.7　自タスクの遅延 (dly_tsk)

■C 言語 API

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

■パラメータ

RELTIM	dlytim	ER0	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_NOSPT	[p]	未サポート機能 (時間管理機能未使用)
E_CTX	[p]	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)
	[k]	（ディスパッチ禁止状態、または CPU ロック状態からの呼び出し）
E_RLWAI	[k]	待ち状態の強制解除 (待ち状態の間に rel_wai を受付)

■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します。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母となります。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。

5.6 同期･通信(セマフォ)機能

表 5-7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5-7 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	sig_sem [S]	セマフォ資源の返却	○		○	○	○		
	isig_sem [S]			○	○	○	○		
2	wai_sem [S]	セマフォ資源の獲得	○		○		○		
3	pol_sem [S]	同上(ポーリング)	○		○	○	○		
	ipol_sem			○	○	○	○		
4	twai_sem [S]	同上(タイムアウト有)	○		○		○		
5	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-8にセマフォの仕様を示します。

表5-8 セマフォの仕様

項番	項目	内容
1	セマフォ ID	1～CFG_MAXSEMID(最大 255)
2	最大資源数	1～65535
3	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO

5.6.1 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	R0	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	[p]	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＞最大資源数)
E_CTX	[p]	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、そのセマフォの資源数を 1 増やします。

セマフォの資源数の最大値は、セマフォ初期登録時に指定した最大資源数です。

5.6.2 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	R0	セマフォ ID
《twai_sem》			
TMO	tmout	ER1	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(twai_sem))
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	[p]	未登録(semid のセマフォが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(wai_sem、twai_sem) twai_sem は tmout が TMO_POL(0)以外の場合)
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)
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 1 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、FIFO で管理されます。

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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

5.6.3 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	R0	セマフォ ID
T_RSEM	*pk_rsem	ER1	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	--	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;       +2     2    セマフォの現在の資源数
} T_RSEM;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rsem が 0)
E_ID	[p]	不正 ID 番号(semid≦0、semid＞CFG_MAXSEMID)
E_NOEXS	[p]	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)、セマフォの現在の資源数(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。

5.7 同期･通信(イベントフラグ)機能

表 5-9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5-9 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	set_flg [S]	イベントフラグのセット	○		○	○	○		
	iset_flg [S]			○	○	○	○		
2	clr_flg [S]	イベントフラグのクリア	○		○	○	○		
	iclr_flg			○	○	○	○		
3	wai_flg [S]	イベントフラグ待ち	○		○		○		
4	pol_flg [S]	同上(ポーリング)	○		○	○	○		
	ipol_flg [S]			○	○	○	○		
5	twai_flg [S]	同上(タイムアウト有)	○		○		○		
6	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-10にイベントフラグの仕様を示します。

表5-10 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 255)
2	イベントフラグのビット数	16 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_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 ： 待ちタスクを解除する時にビットパターンをクリアする

5.7.1 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	R0	イベントフラグ ID
FLGPTN	setptn	E0	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	[p]	未登録(flgid のイベントフラグが存在しない)
E_CTX	[p]	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、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します。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状態を解除します。

5.7.2　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	R0	イベントフラグ ID
FLGPTN	clrptn	E0	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	[p]	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。

5.7.3 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	R0	イベントフラグ ID
FLGPTN	waiptn	E0	待ちビットパターン
MODE	wfmode	R1	待ちモード
FLGPTN	*p_flgptn	ER2	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	@ER7	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	--	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(twai_flg))
E_PAR	[p]	パラメータエラー(waiptn=0、wfmode が不正、tmout≦-2、p_flgptn が 0)
E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[p]	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	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(wai_flg、twai_flg) twai_flg は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

■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'0000)　AND 待ち
- TWF_ORW　　(H'0001)　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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

対象イベントフラグに TA_CLR 属性が指定された場合には、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。

5.7.4 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	R0	イベントフラグ ID
T_RFLG	*pk_rflg	ER1	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	--	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;       +2     2    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 0)
E_ID	[p]	不正 ID 番号(flgid≦0、flgid＞CFG_MAXFLGID)
E_NOEXS	[p]	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。

5.8 同期･通信(データキュー)機能

表 5-11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5-11 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	snd_dtq	[S]	データキューへの送信	○		○		○		
2	psnd_dtq	[S]	同上(ポーリング)	○		○	○	○		
	ipsnd_dtq	[S]			○	○	○	○		
3	tsnd_dtq	[S]	同上(タイムアウト有)	○		○		○		
4	fsnd_dtq	[S]	データキューへの強制送信	○		○	○	○		
	ifsnd_dtq	[S]			○	○	○	○		
5	rcv_dtq	[S]	データキューからの受信	○		○		○		
6	prcv_dtq	[S]	同上(ポーリング)	○		○	○	○		
7	trcv_dtq	[S]	同上(タイムアウト有)	○		○		○		
8	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-12にデータキューの仕様を示します。

表5-12 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 255)
2	データキュー領域の容量（データの個数）	0～65535
3	データサイズ	32 ビット
4	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO

5.8.1 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	R0	データキューID
VP_INT	data	ER1	データキューへ送信するデータ
《tsnd_dtq》			
TMO	tmout	ER2	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(tsnd_dtq))
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_NOEXS	[p]	未登録(dtqid のデータキューが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(snd_dtq、tsnd_dtq) tsnd_dtq は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)
E_ILUSE	[k]	サービスコール不正使用(データキュー領域の容量が 0 のデータキューに対する fsnd_dtq、ifsnd_dtq の発行)

■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。

対象データキューに受信待ちタスクが存在する場合には、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。

対象データキューに既に送信待ちタスクが存在する場合、snd_dtq、tsnd_dtq サービスコールでは、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、psnd_dtq、ipsnd_dtq 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、FIFO で管理されます。

受信待ちタスクも送信待ちタスクも存在しない場合は、データをデータキューに格納します。この

結果、データの数が＋1 されます。

データの数＝データキュー領域の容量の場合は、呼び出しタスクは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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

fsnd_dtq、ifsnd_dtq では、対象データキューに受信待ちタスクが存在する場合や、受信待ちタスクは存在しないが、データキューに空きがある場合は、snd_dtq、isnd_dtq と同じ処理を行います。

ただし、対象データキューに送信待ちタスクが存在する場合や、送信待ちタスクは存在しないが、データキューに空きがない場合には、データキューの先頭（最古）のデータを抹消し、その領域にデータを送信します。

5.8.2 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	R0	データキューID
VP_INT	*p_data	ER1	受信したデータを返す領域の先頭アドレス
《trcv_dtq》			
TMO	tmout	ER2	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	--	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(trcv_dtq))
E_PAR	[p]	パラメータエラー(p_data が 0、tmout≦-2)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_NOEXS	[p]	未登録(dtqid のデータキューが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(rcv_dtq、trcv_dtq) trcv_dtq は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

■機能

dtqid で示されたデータキューからデータを受信し、data 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、データの数は－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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

5.8.3 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	R0	データキューID
T_RDTQ	*pk_rdtq	ER1	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	--	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     2    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 0)
E_ID	[p]	不正 ID 番号(dtqid≦0、dtqid＞CFG_MAXDTQID)
E_NOEXS	[p]	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。

5.9 同期･通信(メールボックス)機能

表 5-13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5-13 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	snd_mbx [S]	メールボックスへの送信	○		○	○	○		
	isnd_mbx			○	○	○	○		
2	rcv_mbx [S]	メールボックスからの受信	○		○		○		
3	prcv_mbx [S]	同上(ポーリング)	○		○	○	○		
	iprcv_mbx			○	○	○	○		
4	trcv_mbx [S]	同上(タイムアウト有)	○		○		○		
5	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-14にメールボックスの仕様を示します。

表5-14 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 255)
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

5.9.1 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	R0	メールボックス ID
T_MSG	*pk_msg	ER1	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 0)
	[k]	(msgpri≦0、msgpri＞CFG_MAXMSGPRI、メッセージ先頭 4 バイトが 0 以外)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＜0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[p]	未登録(mbxid のメールボックスが存在しない)
E_CTX	[p]	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、初期登録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 5-3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 5-4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5-3　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5-4　優先度付きメッセージの形式例

5.9.2 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	R0	メールボックス ID
T_MSG	**ppk_msg	ER1	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	ER2	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	--	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能がtrcv_mbx は未使用(trcv_mbx))
E_PAR	[p]	パラメータエラー(ppk_msg が 0、tmout≦-2)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[p]	未登録(mbxid のメールボックスが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(rcv_mbx、trcv_mbx) trcv_mbx は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

■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、初期登録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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

5.9.3 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	R0	メールボックス ID
T_RMBX	*pk_rmbx	ER1	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	--	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;     +2     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 0)
E_ID	[p]	不正 ID 番号(mbxid≦0、mbxid＞CFG_MAXMBXID)
E_NOEXS	[p]	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。

5.10　拡張同期・通信(ミューテックス)機能

表 5-15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5-15　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	loc_mtx	ミューテックスのロック	○		○		○		
2	ploc_mtx	同上(ポーリング)	○		○	○	○		
3	tloc_mtx	同上(タイムアウト有)	○		○		○		
4	unl_mtx	ミューテックスのロック解除	○		○	○	○		
5	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-16にミューテックス機能の仕様を示します。

表5-16　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 255)
2	サポート属性	TA_CEILING（優先度上限プロトコル）

HI1000/4 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。

5.10.1 ミューテックスのロック(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	R0	ミューテックス ID
《tloc_mtx》			
TMO	tmout	ER1	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(tloc_mtx))
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	[p]	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	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(loc_mtx、tloc_mtx) tloc_mtx は tmout が TMO_POL(0)以外の場合)
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)
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します。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。

タイムティック周期時間の分母(CFG_TICDENO)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。

5.10.2　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	R0	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	[p]	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	[p]	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。

5.10.3　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	R0	ミューテックス ID
T_RMTX	*pk_rmtx	ER1	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	--	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 0)
E_ID	[p]	不正 ID 番号(mtxid≦0、mtxid＞CFG_MAXMTXID)
E_NOEXS	[p]	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 TSK_NOME（0）を返します。

対象ミューテックスに待ちタスクが無い場合は、wtskid に TSK_NONE（0）を返します。

5.11　メモリプール管理(固定長メモリプール)機能

表 5-17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5-17　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	get_mpf [S]	固定長メモリブロックの獲得	○		○		○		
2	pget_mpf [S]	同上(ポーリング)	○		○	○	○		
	ipget_mpf			○	○	○	○		
3	tget_mpf [S]	同上(タイムアウト有)	○		○		○		
4	rel_mpf [S]	固定長メモリブロックの返却	○		○	○	○		
5	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-18に固定長メモリプールの仕様を示します。

表5-18　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 255)
2	固定長メモリプール個数	1～65535
3	固定長メモリプールサイズ	2～65530
4	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO

5.11.1 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	R0	固定長メモリプール ID
VP	*p_blk	ER1	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	ER2	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	--	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(tget_mpf))
E_PAR	[p]	パラメータエラー(p_blk が 0、tmout≦-2)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[p]	未登録(mpfid の固定長メモリプールが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(get_mpf、tget_mpf) tget_mpf は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

■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、FIFO で管理されます。

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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

5.11.2　固定長メモリブロックの返却(rel_mpf)

■C 言語 API

```
ER ercd = rel_mpf(ID mpfid, VP blk);
```

■パラメータ

ID	mpfid	R0	固定長メモリプール ID
VP	blk	ER1	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 0 または奇数、 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)
	[k]	(メモリブロックの先頭アドレス以外)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[p]	未登録(mpfid の固定長メモリプールが存在しない)
E_CTX	[p]	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。

5.11.3　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	R0	固定長メモリプール ID
T_RMPF	*pk_rmpf	ER1	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	--	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;     +2     2    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 0)
E_ID	[p]	不正 ID 番号(mpfid≦0、mpfid＞CFG_MAXMPFID)
E_NOEXS	[p]	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。

5.12　メモリプール管理(可変長メモリプール)機能

表 5-19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5-19　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	get_mpl	可変長メモリブロックの獲得	○		○		○		
2	pget_mpl	同上(ポーリング)	○		○	○	○		
	ipget_mpl			○	○	○	○		
3	tget_mpl	同上(タイムアウト有)	○		○		○		
4	rel_mpl	可変長メモリブロックの返却	○		○	○	○		
5	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-20に可変長メモリプールの仕様を示します。

表5-20　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 255)
2	可変長メモリプールサイズ	18～65534
3	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO

可変長メモリプールのフラグメンテーション

ひとつの可変長メモリプールからメモリブロックの獲得と返却を繰り返していると、メモリプール内の空き領域のフラグメンテーションが発生し、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ようになってしまいます。しかし、本カーネルには、フラグメンテーションを整理する機能はありません。

特に、返却されることのないメモリブロックがメモリプールの中央近辺に存在すると、そのメモリブロックによってメモリプールの空き領域が分断されてしまうため、メモリプールサイズに見合う十分な連続空き領域を確保できなくなってしまいます。このような事態をある程度避けるためには、返却される予定のないメモリブロックは返却する可能性のあるメモリブロックの獲得前に、獲得するようにしてください。このようにすることで、そのメモリブロックはメモリプールの端から確保されるようになります。

5.12.1 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	R0	可変長メモリプール ID
UINT	blksz	E0	メモリブロックサイズ（バイト数）
VP	*p_blk	ER1	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	ER2	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	--	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_NOSPT	[p]	未サポート機能(時間管理機能未使用、またはタイムアウト機能が未使用(tget_mpl))
E_PAR	[p]	パラメータエラー(p_blk が 0、blksz が 2 の倍数以外または 0、tmout≦-2、mplsz* -16＜blksz)
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[p]	未登録(mplid の可変長メモリプールが存在しない)
E_CTX	[p]	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)
	[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(get_mpl、tget_mpl) tget_mpl は tmout が TMO_POL(0)以外の場合)
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_RLWAI	[k]	待ち状態の強制解除(待ち状態の間に rel_wai を受付)

【注】 * 可変長メモリプール初期登録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 ＝ blksz + 16

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。

タイムティック周期時間の分母(CFG_TICDENO)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/タイムティック周期時間の分母に制限されます。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。

5.12.2　可変長メモリブロックの返却(rel_mpl)

■C 言語 API

```
ER ercd = rel_mpl(ID mplid, VP blk);
```

■パラメータ

ID	mplid	R0	可変長メモリプール ID
VP	blk	ER1	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 0 または奇数)
	[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	[p]	未登録(mplid の可変長メモリプールが存在しない)
E_CTX	[p]	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 ＝ (獲得時に指定した blksz) + 16

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。

5.12.3　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	R0	可変長メモリプール ID
T_RMPL	*pk_rmpl	ER1	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	--	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;      +2     4    空き領域の合計サイズ（バイト数）
           UINT   fblksz;      +6     2    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 0)
E_ID	[p]	不正 ID 番号(mplid≦0、mplid＞CFG_MAXMPLID)
E_NOEXS	[p]	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を返します。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。

5.13　時間管理機能(システム時刻管理)

表 5-21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5-21　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			○	○	○	○		

【注】 *1　"[S]"はスタンダードプロファイルの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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

なお、HI1000/4 では、システム時刻を更新する機能を独自に用意しています。

表 5-22にシステム時刻管理の仕様を示します。

表5-22　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	ユーザ指定のタイムティック更新時間[ms]
4	システム時刻初期値(初期起動時)	H'000000000000

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。

[タイムティック周期 ≦ 1 の場合]

最大値＝H'ffffffffffff／タイムティック周期の分母（TIC_DENO）

[タイムティック周期 ＞ 1 の場合]

最大値＝H'ffffffffffff

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 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

5.13.1 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	ER0	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;        +2     4    システムの現在時刻（下位）
} SYSTIM;
```

■エラーコード

E_NOSPT	[p]	未サポート機能(時間管理機能未使用)
E_PAR	[p]	パラメータエラー(p_systim が 0)

■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。

タイムティック周期時間の分母(CFG_TICDENO)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'ffffffffffff/タイムティック周期時間の分母となります。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。

5.13.2　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	ER0	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	--	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;        +2     4    システムの現在時刻（下位）
} SYSTIM;
```

■エラーコード

E_NOSPT	[p]	未サポート機能(時間管理機能未使用)
E_PAR	[p]	パラメータエラー(p_systim が 0)

■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。

5.14　時間管理機能(周期ハンドラ)

表 5-23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5-23　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	sta_cyc　[S]	周期ハンドラの動作開始	○		○	○	○		
	ista_cyc			○	○	○	○		
2	stp_cyc　[S]	周期ハンドラの動作停止	○		○	○	○		
	istp_cyc			○	○	○	○		
3	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

表 5-24に周期ハンドラの仕様を示します。

表5-24　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 254)
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

5.14.1　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	R0	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_NOSPT	[p]	未サポート機能(時間管理機能未使用)
E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[p]	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します。

周期ハンドラ属性に TA_PHS 属性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 属性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行います。

TA_PHS 属性が指定されている場合は、周期ハンドラの初期登録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。

5.14.2　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	R0	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_NOSPT	[p]	未サポート機能(時間管理機能未使用)
E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[p]	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します。

5.14.3　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	R0	周期ハンドラ ID
T_RCYC	*pk_rcyc	ER1	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	--	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     2    周期ハンドラの動作状態
           RELTIM lefttim;      +2     4    周期ハンドラを次に起動するまでの時間
} T_RCYC;
```

■エラーコード

E_NOSPT	[p]	未サポート機能(時間管理機能未使用)
E_PAR	[p]	パラメータエラー(pk_rcyc が 0)
E_ID	[p]	不正 ID 番号(cycid≦0、cycid＞CFG_MAXCYCID)
E_NOEXS	[p]	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)、周期ハンドラを次に起動するまでの時間(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'0000）周期ハンドラが動作していない
- TCYC_STA（H'0001）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。

5.15　システム状態管理機能

表 5-25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5-25　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	ivbgn_int		割込みハンドラの開始をトレースに取得		○	○	○	○		
14	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び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能
"C"は CPU 例外ハンドラから呼び出し可能

5.15.1　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	R0	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)
E_CTX	[p]	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

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。

5.15.2　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	ER0	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	--	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 0)

■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。

5.15.3 CPU ロック状態への移行 (loc_cpu、iloc_cpu)

■C 言語 API

```
ER ercd = loc_cpu();
ER ercd = iloc_cpu();
```

■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) 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。
(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
- 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。

(1) unl_cpu、iunl_cpuサービスコールの呼び出し
(2) ext_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 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。

5.15.4 CPU ロック状態の解除(unl_cpu、iunl_cpu)

■C 言語 API

```
ER ercd = unl_cpu();
ER ercd = iunl_cpu();
```

■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_CTX	[p]	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

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。

5.15.5　ディスパッチの禁止(dis_dsp)

■C 言語 API

```
ER ercd = dis_dsp();
```

■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_CTX	[p]	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

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サービスコールの呼び出し

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。

ディスパッチ禁止状態とディスパッチ許可解除状態の間の遷移は、dis_dsp、ena_dsp、ext_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。

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。

5.15.6　ディスパッチの許可(ena_dsp)

■C 言語 API

```
ER ercd = ena_dsp();
```

■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_CTX	[p]	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

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。

5.15.7　コンテキストの参照(sns_ctx)

■C 言語 API

```
BOOL state = sns_ctx();
```

■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。

5.15.8　CPU ロック状態の参照(sns_loc)

■C 言語 API

```
BOOL state = sns_loc();
```

■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。

5.15.9　ディスパッチ禁止状態の参照(sns_dsp)

■C 言語 API

```
BOOL state = sns_dsp();
```

■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。

5.15.10 ディスパッチ保留状態の参照 (sns_dpn)

■C 言語 API

```
BOOL state = sns_dpn();
```

■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 を返し、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。

5.15.11　カーネルの起動(vsta_knl、ivsta_knl)

■C 言語 API

```
void vsta_knl();
void ivsta_knl();
```

■アセンブリ言語 API

シンボル"_vsta_knl"に分岐

シンボル"_ivsta_knl"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で発行してください。

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。

なお、本サービスコールは HI1000/4 の独自機能です。

5.15.12　システムダウン(vsys_dwn、ivsys_dwn)

■C 言語 API

```
void vsys_dwn(H type, H inf1, B inf2, B inf3, H inf4, UW inf5);
void ivsys_dwn(H type, H inf1, B inf2, B inf3, H inf4, UW inf5);
```

■パラメータ

H	type	R0	エラー種別
H	inf1	E0	システムダウン情報１
B	inf2	R1L	システムダウン情報２
B	inf3	R1H	システムダウン情報３
H	inf4	E1	システムダウン情報４
UW	inf5	ER2	システムダウン情報５

■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(H'1～H'7fff)を設定してください。それ以外の値はシステム用に予約されています。

ユーザが vsys_dwnサービスコールを発行する場合のシステムダウン情報１～システムダウン情報５については、ユーザが定義してください。システムで発生したシステムダウンのシステムダウン情報については「10　システムダウン時の情報」を参照してください。

システムダウン情報にアクセスする場合は、エラー種別でシステムでのシステムダウンなのか、ユーザが呼び出したシステムダウンなのかを判断してください。

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。

なお、本サービスコールは HI1000/4 の独自機能です。

5.15.13 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	R0	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、CPU のベクタ番号です。

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。

トレース機能を組み込んでいない場合は、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 HI1000/4 の独自機能です。

E6000H の RTOS デバッグ機能を使用する場合は、必ず全ての割込みハンドラの先頭で呼び出す必要があります。

5.15.14 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	R0	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、CPU のベクタ番号です。

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。

トレース機能を組み込んでいない場合は、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 HI1000/4 の独自機能です。

E6000H の RTOS デバッグ機能を使用する場合は、必ず全ての割込みハンドラの最後で呼び出す必要があります。

5.16　割込み管理機能

表 5-26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5-26　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	chg_ims	割込みマスクの変更	○		○		○		
	ichg_ims			○	○		○		
2	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

5.16.1 割込み制御モードと割込みマスクレベル値

表 5-27に各割込み制御モードの割込みマスクレベルを示します。割込み制御モード、および CCR 値、EXR 値、受け付ける割込みに関する詳細は、当該ハードウェアマニュアルを参照してください。

表5-27 割込み制御モードと割込みマスクレベル

割込み制御モード	割込みﾏｽｸﾚﾍﾞﾙ値 (imask)	CCR 値		EXR 値			受け付ける割込み
		I	UI	I2	I1	I0	
モード 0	1	1	−	−	−	−	NMI のみ
	0	0	−	−	−	−	すべて
モード 1	3	1	1	−	−	−	NMI のみ
	2	1	0	−	−	−	コントロールレベル 1
	1	0	1	−	−	−	すべて
	0	0	0	−	−	−	すべて
モード 2	7	−	−	1	1	1	NMI のみ
	6	−	−	1	1	0	プライオリティレベル 7
	5	−	−	1	0	1	プライオリティレベル 6〜7
	4	−	−	1	0	0	プライオリティレベル 5〜7
	3	−	−	0	1	1	プライオリティレベル 4〜7
	2	−	−	0	1	0	プライオリティレベル 3〜7
	1	−	−	0	0	1	プライオリティレベル 2〜7
	0	−	−	0	0	0	すべて
モード 3	8	1	1	1	1	1	NMI のみ
	7	1	0	−	−	−	コントロールレベル 1
	6	0	0	1	1	0	プライオリティレベル 7／コントロールレベル 0、1
	5	0	0	1	0	1	プライオリティレベル 6〜7／コントロールレベル 0、1
	4	0	0	1	0	0	プライオリティレベル 5〜7／コントロールレベル 0、1
	3	0	0	0	1	1	プライオリティレベル 4〜7／コントロールレベル 0、1
	2	0	0	0	1	0	プライオリティレベル 3〜7／コントロールレベル 0、1
	1	0	0	0	0	1	プライオリティレベル 2〜7／コントロールレベル 0、1
	0	0	0	0	0	0	すべて

【注】 "ー"は不定値です

5.16.2　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	R0	割込みマスク値
			CFG_INTMD=0：CR_IMS0～CR_IMS1(H'0~H'1)
			CFG_INTMD=1：CR_IMS0～CR_IMS3(H'0~H'3)
			CFG_INTMD=2：CR_IMS0～CR_IMS7(H'0~H'7)
			CFG_INTMD=3：CR_IMS0～CR_IMS8(H'0~H'8)

■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 に範囲外の値を指定)
E_CTX	[k]	コンテキストエラー(CPU ロック状態から発行できない)

■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、割込み制御モード（CFG_INTMD）に応じて、CR_IMSn(n：0～8)を指定します。

割込みマスクは、CCR、EXR に直接値を設定して割込みの禁止・解除を行います。

割込みマスク値（imask）と CCR、EXR 値の対応は、表 5-27を参照してください。

本サービスコールにより割込みをマスクした場合は、非タスクコンテキストとなります。したがって、待ち状態へ移行するサービスコール、タスクコンテキスト部専用サービスコールは発行できません。

本サービスコールによってタスク実行状態から非タスクコンテキストに遷移した場合、タスク実行状態に戻るには、本サービスコールを用いて割込みマスクを解除する必要があります。非タスクコンテキストとして実行中にサービスコールを発行してタスク切り替えが必要になっても、本サービスコールによって割込みマスクを CR_IMS0(H'0)に戻す(タスク実行状態に戻る)まで遅延されます。

【注】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、本サービスコールを用いてカーネル割込みマスクレベル以下に下げる場合を除き、サービスコールは発行できません。発行した場合、システムの正常な動作は保証されません。

5.16.3　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	ER0	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
IMASK	*p_imask	--	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 0)

■機能

現在の割込みマスクレベルを返します。

割込み制御モード（CFG_INTMD）によって、割込みマスクレベルとして使用できる値の範囲と内容が異なります。

5.17　システム構成管理機能

表 5-28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5-28　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	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び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から呼び出し可能、"L"は CPU ロック状態から呼び出し可能

"C"は CPU 例外ハンドラから呼び出し可能

5.17.1　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	ER0	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	--	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    カーネルのメーカコード
           UH     prid;       +2    2    カーネルの識別番号
           UH     spver;      +4    2    ITRON 仕様のバージョン番号
           UH     prver;      +6    2    カーネルのバージョン番号
           UH     prno[4];    +8    8    カーネル製品の管理情報
}  T_RVER;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rver が 0)

■機能

使用しているカーネルのバージョン情報を参照し、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、この製品を作ったメーカを表します。
本カーネルの maker の値は、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。

- HI1000/4　　H’0011

(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’401（μITRON仕様 Ver.4.01.00）です。

(4)　prver

prver は、製品のバージョン番号を表します。
本カーネルの prver の値は、Ver.1.04.00 を意味する H'0140 です。

(5) prno

prno は、製品管理情報や製品番号などを表します。

カーネルの prno[0]から prno[3]の値は H'0000 です。

6. アプリケーションプログラムの作成

6.1 ヘッダファイル

6.1.1 標準ヘッダファイル

本カーネルでは、以下の標準ヘッダファイルを提供しています。

(1) itron.h（C 言語用）

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) hihead¥itron.inc（アセンブリ言語用）

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。

(3) kernel.h と kernel_macro.h（C 言語用）

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。

(4) kernel.inc と kernel_macro.inc（アセンブリ言語用）

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 とコンフィギュレータが出力する kernel_macro.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。

「5 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、「5 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6-1 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	kernel.inc	TMIN_MPRI	メッセージ優先度の最小値（常に 1）
3		TKERNEL_MAKER	カーネルのメーカコード
4		TKERNEL_PRID	カーネルの識別番号
5		TKERNEL_SPVER	ITRON 仕様のバージョン番号
6		TKERNEL_PRVER	カーネルのバージョン番号
7		TMAX_ACTCNT	タスクの起動要求キューイング数の最大値（常に 255）
8		TMAX_WUPCNT	タスクの起床要求キューイング数の最大値（常に 255）
9		TMAX_SUSCNT	タスクの強制待ち要求ネスト数の最大値（常に 1）
10		TBIT_FLGPTN	イベントフラグのビット数（常に 16）
11		TMAX_MAXSEM	セマフォの最大資源数の最大値（常に 65535）

12	kernel_macro.h	TIC_NUME	タイムティックの周期の分子
13	kernel_macro.inc	TIC_DENO	タイムティックの周期の分母
14		TMAX_TPRI	タスク優先度の最大値
15		TMAX_MPRI	メッセージ優先度の最大値
16		VTKNL_LVL	カーネル割込みマスクレベル
17		VTIM_LVL	タイマ割込みマスクレベル

6.2 CPU リソースの扱い

6.2.1 VBR、SBR(H8SX のみ)レジスタ

H8SX を使用する場合、VBR および SBR はカーネル起動前に、アプリケーションで初期化してください。カーネル起動後 VBR を変更する場合は、変更する前に、割込みベクタテーブルを VBR に設定するアドレスにコピー等により作成する必要があります。

SBR の初期化が必要となるのは HEW で SBR オプションを指定した場合です。初期化で設定する値は、SBR オプションに指定した値と同じ値でなければなりません。

6.3 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。

6.4 タ ス ク

(1) 記述方法

タスクは、図 6-1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 サービスコールを用いて終了してください。ext_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"          kernel.h は、itron.h とコンフィギュレータが出力する
#include "kernel_id.h"       kernel_macro.h をインクルードしています。

#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)
{                            ←act_tsk で起動された場合には、パラメータとしてタスク初期登録時
                             に指定した exinf が渡され、sta_tsk で起動された場合は sta_tsk で
  ext_tsk();                 指定された stacd が渡されます。
                             ←タスクを終了する場合は、ext_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6-1 タスクの C 言語記述例

(2) コンテキストレジスタ

タスクのレジスタ内容は、タスクスイッチや割込みが発生しても、次に実行されるときには元の値が保証されるようになっています。保証対象のレジスタをコンテキストレジスタと呼びます。コンテキストレジスタ以外の CPU レジスタの内容を変更してはなりません。

どのレジスタがコンテキストレジスタとなるかは、使用するライブラリによって変わります。詳細は「8.1 提供カーネルライブラリ」を参照してください。表 6-2に、タスクのコンテキストレジスタとタスク起動時の初期化内容を示します。タスクをアセンブリ言語で記述する場合も、表 6-2を参考にしてください。

なお、サービスコールの前後では一部のコンテキストレジスタは保証されません。詳細は、「5.2.5 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。

表6-2 タスクのコンテキストレジスタと初期化内容

項番	項目	初期化の内容
1	プログラムカウンタ(PC)	タスク登録時に指定したタスク起動アドレス
2	コンディションコードレジスタ(CCR)	割込みマスク解放(0)
3	エクステンドレジスタ(EXR)	割込みマスク解放(0)
4	スタックポインタ(ER7)	タスク登録時に指定したタスクスタック領域へのポインタ
5	ER0(ｱｾﾝﾌﾞﾗ記述)／第 1 引数(C 記述)	act_tsk の場合は、タスクの拡張情報(exinf)、sta_tsk の場合は指定した起動コード(stacd)
6	汎用レジスタ(ER1～ER6) 積和演算レジスタ(MACH、MACL)	不定 (MACH、MACL は機能選択時)
7	ショートアドレスベースレジスタ(SBR) (H8SX のみ)	カーネルリセット時の SBR を保持します
8	タスクベース優先度	タスク登録時に指定したタスク起動優先度
9	タスク現在優先度	タスク登録時に指定したタスク起動優先度
10	タスク起床要求キューイング数	0
11	タスク強制待ち要求ネスト数	0

(3) タスクの生成方法

タスクは、システム構築時にコンフィギュレータに初期登録することで生成されます。詳細については「7 コンフィギュレーション」を参照してください。

6.5 割込みハンドラ

6.5.1 割込みハンドラの作成

割込みが発生すると、制御はカーネルの介入なしに割込みハンドラに渡されます。割込みハンドラでは、割込み発生時レジスタを保証しなければなりません。

割込みハンドラは通常以下の手順で作成してください。

(a) ivbgn_int サービスコールの発行(サービスコールトレース表示、E6000H の RTOS デバッグ機能を使用する場合に必要です。)

(b) 割込みハンドラで使用するレジスタの退避

- スタックポインタの保存

–割込みハンドラ専用スタック領域にスタックポインタを変更
(割込みハンドラでスタックを使用しない場合は不要)
–レジスタの内容保存

(c) 割込み処理

(d) 割込みハンドラで使用したレジスタの復帰

- レジスタの内容復帰
- スタックポインタの変更(割込みハンドラでスタックを使用しない場合は不要)

(e) ivend_int サービスコールの発行(サービスコールトレース表示、E6000H の RTOS デバッグ機能を使用する場合に必要です。)

(f) ret_int ルーチンの呼び出し(カーネル割込みマスクレベル以下の場合)、または RTE 命令実行
(カーネル割込みマスクレベルより高い場合)

C コンパイラの割込み関数の作成機能(#pragma interrupt)を用いることにより、割込みハンドラを C 言語で記述できます。

#pragma interrupt を用いて割込みハンドラとなる関数を宣言します。図 6-2では、Inhhdr 関数を割込みハンドラとして宣言しています。

割込みの仕様として「スタック切り替え指定」と「割込み関数終了指定」を行います。

「スタック切り替え指定」は、割込みハンドラ開始時に切り替えるスタック領域の指定であり、初期スタックポインタを“ sp=＜アドレス＞”で指定します。スタック領域は、割込みレベル毎に固有の領域を指定してください。

「割込み関数終了指定」は、割込みハンドラ終了時の復帰方法の指定であり、割込みハンドラ終了時に ret_int ルーチンの呼び出し、または RTE 命令を実行する必要があります。

カーネル割込みマスクレベル以下の割込みハンドラの場合、”sy=$ret_int”と指定します。これにより、割込みハンドラ終了時、”jmp @ret_int”の命令が実行されます。

また、カーネル割込みマスクレベルより高い割込みハンドラの場合は、何も記述する必要はありません。

詳細については、『H8S,H8/300 シリーズ C/C++ コンパイラ、アセンブラ、最適化リンケージエディタユーザーズマニュアル』を参照してください。

図 6-2に割込みハンドラの記述例を示します。

(1) 記述方法

割込みハンドラは、図 6-2に示すように割込み関数として記述します。

```
#include "kernel.h"

extern VP int_stk001;                          ←コンフィギュレータで確保した割込みスタックを指定します。

static const VP p_stk=(VP)&int_stk001;         ←スタックポインタの初期値を const 型として定義します。

                                               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。

#pragma                                        ←#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。割込み仕様として以下を指定してください。
  interrupt(Inhhdr(sp=p_stk,sy=$ret_int))
                                               ・スタック切り替え指定（sp=p_stk）

                                                 NMI 割込みハンドラの場合はスタック切り替えを行わないで下さい。

                                               ・割込み関数終了指定（sy=$ret_int）

                                                 カーネル割込みマスクレベル以下のレベルの割込みハンドラは、"sy=$ret_int"を指定してください。カーネル割込みマスクレベルよりも高いレベル(NMI を含む)の割込みハンドラは RTE 命令で終了する必要があるので、割込み関数終了指定は、記述しないでください。

void Inhhdr(void)                              割込みハンドラは、void 型の関数として記述します。
{
    //ivbgn_int(88); // call ivbgn_int(88)    ←割込み処理の開始(ivbgn_int)と終了(ivend_int)を指定します。E6000H の RTOS デバッグ機能を使用する場合は必須となります。
    /* 割込みハンドラの処理 */
    //ivend_int(88); // call ivend_int(88)
}
```

図6-2　割込みハンドラの C 言語記述例

(2) 割込みハンドラ処理条件

割込みハンドラでは、全レジスタ内容を保証しなければなりません。表 6-3に割込みハンドラ処理条件を示します。

表6-3 割込みハンドラ処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態で起動されます。
2	使用できるレジスタ	ER0～ER6、MACH、MACL(MACH、MACL は機能選択時)が使用できます。割込みハンドラ処理終了前に起動時の値に戻してください。
3	スタックポインタ	割込み発生元に制御を戻すときは起動時と同じ値にしてください。
4	使用できるサービスコール	非タスクコンテキストから発行可能なサービスコール。 ただし、カーネル割込みマスクレベルより高い割込みハンドラおよび NMI 割込みハンドラからサービスコールを発行することはできません。
5	使用できるスタック領域	コンフィギュレータで確保し、起動時に確保した割込みスタックに切り替えてください。 同一割込みレベルの割込みハンドラでは、スタックを共有することができます。
6	終了方法	ret_int ルーチンにより処理を終了します。 終了時は、スタックポインタを起動時と同じ状態にしてください。 カーネル割込みマスクレベルより高い割込み、および NMI 割込みハンドラは RTE 命令で終了してください。

割込みハンドラを作成する場合、以下の項目に注意してください。

(3) 割込みマスクレベルの保証

カーネルでは、4 種類の割込み制御モードを使用することができます。
割込みのマスクレベルは CCR レジスタ、EXR レジスタの割込み制御ビットで表されます。
割込みハンドラでは、各割込み制御モードによって割込みマスクレベルに対する処理が異なります。

(a) 割込み制御モード 0

CCR レジスタの I ビットのみで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビットがセットされます。割込みハンドラでは、CCR の I ビットをクリアしないでください。I ビットをクリアした場合、システムの動作は保証されません。

(b) 割込み制御モード 1

CCR レジスタの I ビット、UI ビットで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビット、UI ビットがセットされます。

コントロールレベル 0 の割込みハンドラでは、CCR の UI ビットをクリアして割込みマスクレベルを変更し、コントロールレベル 1 の割込みを受け付けるように設定してください。コントロールレベル 0 の割込みハンドラで CCR レジスタの I ビットをクリアした場合、システムの動作は保証されません。また、コントロールレベル 1 の割込みハンドラでは、割込みマスクレベルを変更しないでください。

(c) 割込み制御モード 2

EXR レジスタの I0～I2 ビットで制御するモードで、CCR レジスタの I ビット、UI ビットは無視

されます。割込みハンドラが起動されると、EXR レジスタの I0～I2 ビットが発生した割込みのレベルでマスクレベルがセットされます。

(d) 割込み制御モード 3

CCR レジスタの I ビット、UI ビットと EXR レジスタの I0～I2 ビットで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビット、UI ビット、EXR レジスタの I0～I2 ビットがセットされます。

コントロールレベル 0 の割込みハンドラでは、CCR の I ビット、UI ビットをクリアして割込みマスクレベルを変更し、優先度レベルの高い割込みを受け付けるようにしてください。割込みハンドラでは、EXR レジスタを変更しないでください。EXR レジスタを変更した場合、システムの動作は保証されません。

(4) カーネル割込みマスクレベル

カーネルには、内部に持つ情報に矛盾が生じるのを防ぐため、割込みをマスクして実行する部分(クリティカルセクション)があります。カーネルのクリティカルセクション実行中に発生した割込みは、通常はクリティカルセクションが終わるまで受け付けが遅延されます。しかし、カーネル割込みマスクレベルより高い割込みについては、クリティカルセクション実行中にも即座に受け付けられます。

【注】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ハンドラからは、サービスコールは発行できません。発行した場合、システムの正常な動作は保証されません。
また、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ハンドラからの復帰は RTE 命令を実行してください。

【注】割込み制御モードを 3 で、カーネル割込みマスクレベルとしてレベル 7 を使用した場合、コントロールレベル 1 の割込みハンドラからはサービスコールを発行できません。

(5) 割込みの注意事項

- ユーザは、割込みハンドラを自由に作成することができますが、割込みハンドラの実行時間が長すぎるとシステム全体のスループットを低下させることになります。システムの応答性に大きな影響を与えますので、注意して作成してください。
- コンフィギュレータへの定義により、カーネルの割込みマスクレベルを決めることができます。この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では、サービスコールを発行することはできません。また、NMI(Non Maskable Interrupt)の割込みハンドラでも、サービスコールは発行できません。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発行した場合、システムの正常な動作は保証されません。
- カーネル割込みマスクレベル以下の割込みハンドラからの復帰は、ret_int ルーチンを呼び出してください。ret_int ルーチン以外を使用した場合、システムの正常な動作は保証されません。

6.5.2 割込みハンドラの登録

割込みハンドラの登録は、該当するベクタテーブルに割込みハンドラの先頭アドレスを設定します。割込みハンドラは、割込み発生時にカーネルの介入を受けずに直接制御が移ります。

割込み発生要因に関する詳細は当該ハードウェアマニュアルを参照してください。

割込みハンドラ登録の詳細については、「7 コンフィギュレーション」を参照してください。

6.5.3 未定義割込みハンドラ

未定義割込みハンドラは、システムで予期せぬ割込みが発生すると実行するプログラムです。

サンプル提供の未定義割込みハンドラのプログラムは、カーネルの未定義割込み処理(_KERNEL_H_ilint)をサブルーチンコールしシステムダウンとなります。

システムダウン時の未定義割込みの情報については、「10 システムダウン時の情報 システムダウン時の情報」および当該ハードウェアマニュアルを参照してください。

未定義割込みハンドラは、割込みハンドラと同様に作成することができます。

未定義割込みハンドラの登録の詳細については、「7 コンフィギュレーション」を参照してください。

6.6 CPU 例外ハンドラ(TRAPA 例外を含む)

6.6.1 CPU 例外ハンドラの作成

(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、割込みハンドラと同様に#pragma interrupt で記述しますが、CPU 例外ハンドラは再入の可能性があるため、専用のスタックを持つことができません。そのため、#pragma interrupt でスタック切り替え指定(sp=)を行わないでください。CPU 例外ハンドラは例外発生元のスタックで動作します。

(2) CPU 例外ハンドラ処理条件

表 6-4にレジスタ CPU 例外ハンドラ処理条件を示します。

表6-4 CPU 例外ハンドラ処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態で起動されます。CCR.I ビットが ON の状態で起動されます。
2	使用できるレジスタ	ER0～ER6、MACH、MACL(MACH、MACL は機能選択時)が使用できます。CPU 例外ハンドラ処理終了前に起動時の値に戻してください。
3	スタックポインタ	起動時は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。
4	使用できるサービスコール	CPU 例外ハンドラから発行可能な SVC は以下です。 sns_ctx、sns_loc、sns_dsp、sns_dpn、(i)get_tid、(i)vsta_knl、(i)vsys_dwn ただし、カーネル割込みマスクレベルより高い状態からサービスコールを発行することはできません。
5	使用できるスタック領域	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。例外発生分のスタック領域を確保する必要があります。
6	終了方法	ret_int ルーチンにより処理を終了します。 ただし、カーネル割込みマスクレベルより高い状態からは、RTE 命令で終了してください。

6.6.2 CPU 例外ハンドラの登録

CPU 例外ハンドラの登録は、割込みハンドラ同様、該当するベクタテーブルに CPU 例外ハンドラの先頭アドレスを設定します。CPU 例外ハンドラは、例外発生時にカーネルの介入を受けずに直接制御が移ります。

例外発生要因に関する詳細は当該ハードウェアマニュアルを参照してください。

CPU 例外ハンドラ登録の詳細については、「7 コンフィギュレーション」を参照してください。

6.7 周期ハンドラ、初期化ルーチン

6.7.1 周期ハンドラ、初期化ルーチンの作成

(1) 記述方法

周期ハンドラ、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 6-3に周期ハンドラの C 言語記述例、図 6-4に初期化ルーチンの 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 HDR(VP_INT exinf)          ←パラメータとして初期登録時に指定した exinf が渡されます。
{
  /* 周期ハンドラ */
}
```

図6-3 周期ハンドラの C 言語記述例

```
#include "kernel.h"

void INI(void)
{
  /* 初期化ルーチン */
}
```

図6-4 初期化ルーチンの C 言語記述例

【注】タイマ初期化ルーチンは、コンフィギュレータで時間管理機能を選択する事によって自動的に組み込まれます。タイマ初期化ルーチンはユーザが作成することができますが、タイマ初期化ルーチンのシンボル名称は、_KERNEL_HIPRG_TIMINI で固定となっています。
デバッグデーモン初期化ルーチンは、コンフィギュレータでオブジェクト操作機能を選択する事によって自動的に組み込まれます。

(2)　　周期ハンドラ、初期化ルーチン処理条件

表 6-5に周期ハンドラ、初期化ルーチン処理条件を示します。

表6-5　周期ハンドラ、初期化ルーチン処理条件

項番	項目	内容
1	割込みマスク	[周期ハンドラ] タイマ割込みマスクレベルで起動されます。 [初期化ルーチン] カーネル割込みマスクレベルで起動されます。
2	使用できるレジスタ	ER0～ER6、MACH、MACL(MACH、MACL は機能選択時)が使用できます。 [周期ハンドラ] ER0（アセンブラ記述）／第 1 引数（C 記述）に周期ハンドラ拡張情報が格納されています。
3	スタックポインタ	[周期ハンドラ] タイマ割込みハンドラのスタックを使用します。 周期ハンドラ終了前に、起動時の値に戻してください。 [初期化ルーチン] カーネルスタックを使用します。
4	使用できるサービスコール	非タスクコンテキストから発行可能なサービスコール。
5	使用できるスタック領域	[周期ハンドラ] システム構築時にタイマ割込みハンドラスタックに、周期ハンドラで使用するサイズを加算して確保してください。 [初期化ルーチン] カーネルスタックで処理されます。 タイマ初期化ルーチンで使用するスタックサイズについては、C.10 (3)タイマ初期化ルーチンを参照してください。
6	終了方法	RTS 命令により処理を終了します。 終了時は、スタックポインタを起動時と同じ状態にしてください。

6.7.2　　周期ハンドラ、初期化ルーチンの登録

周期ハンドラ、初期化ルーチンは、システム構築時に初期登録します。詳細については、「7 コンフィギュレーション」を参照してください。

6.8 タイマドライバ

カーネルの時間管理機能を使用するには、タイマドライバを作成する必要があります。

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。タイマ割込み処理ルーチンは、割込み要因をクリアし、カーネルのタイマ割込みハンドラ _KERNEL_H_timsys( )を呼び出します。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。

6.8.1 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、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
- TIC_DENO：タイムティック周期の分母
- VTIM_LVL：タイマ割込みレベル

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 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で時間管理機能を選択した場合は、初期化ルーチンとしてタイマ初期化ルーチン(_KERNEL_HIPRG_TIMINI)が自動的に登録されます。

(2) タイマドライバを作成する

以下の 2 つのルーチンを作成します。

- タイマ初期化ルーチン：_KERNEL_HIPRG_TIMINI（シンボル名称固定）
- タイマ割込みルーチン：_KERNEL_H_TIM（シンボル名称固定）

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。「6.7 周期ハンドラ、初期化ルーチン」を参照してください。

タイマ初期化ルーチンでは、kernel_macro.h, kernel_macro.inc 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の初期化を行ってください。

また、kernel_macro.h, kernel_macro.inc に定義されている VTIM_LVL により割込みコントローラに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、タイマ割込みルーチン(_KERNEL_H_TIM)が割込みハンドラとして起動されます。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 6-5のように記述します。タイマ割込み処理ルーチンは、割込みハンドラとして動作します。タイマ割込みルーチンの処理条件は表 6-3を参照してください。

```
#include "kernel.h"
#include "1650tmrdrv.h"                          ←タイマのヘッダファイルをインクルードします。

extern KERNEL_HI_TIM_SP;
void KERNEL_H_timsys(void);
const VP P_intstkxx = &KERNEL_HI_TIM_SP;         ←スタックポインタの初期値をconst 型として定義します。
                                                 【注】タイマ割込みスタックのシンボル名称はシステムで固定となっています。

#pragma interrupt(KERNEL_H_TIM(sp=P_intstkxx,    ←#pragma interrupt により、タイマ割込みルーチンを割込み関数として宣言します。また、以下を指定してください。
                               sy=$KERNEL_H_timsys))
void KERNEL_H_TIM(void)
{                                                ・スタック切り替え指定
       //unsigned char tsr = TPU0.TSR.BIT.TGFA;  （sp=P_intstkxx）
       // dummy read                             ・タイマ割込みルーチン終了指定
       TPU0.TSR.BIT.TGFA;  // dummy read        （sy=$KERNEL_H_timsys）
       TPU0.TSR.BIT.TGFA = 0;                    関数名は固定です。
       // status register(TGFA flag) clear
       //ivbgn_int(88);     // call ivbgn_int(88) ←割込み処理の開始(ivbgn_int)と終了(ivend_int)を指定します。E6000H の RTOS デバッグ機能を使用する場合は必須となります。
      // タイマ割込みルーチン  //
       //ivend_int(88);     // call ivend_int(88)
}
```

図6-5　タイマ割込みルーチンの記述例

(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。

6.8.2 サンプルタイマドライバ

サンプルタイマドライバは、以下に構成されています。

- ソースプログラムファイル（ファイル名 : ***nnnnz***_tmrdrv.c）
  - タイマ初期化ルーチン（関数 : _KERNEL_HIPRG_TIMINI）
  - タイマ割込みハンドラ（関数 : _KERNEL_H_TIM）

表 6-6に、各製品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6-6 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	製品名	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	HI1000/4	H8SX/1650	TPU	周辺クロック(Pφ)=35MHz
2		H8SX/1525	TPU	周辺クロック(Pφ)=35MHz
3		AE57	TMR1	周辺クロック(Pφ)=20MHz
4		H8S/2655	TPU	周辺クロック(Pφ)=20MHz
5		H8S/2148	FRT	周辺クロック(Pφ)=20MHz

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、タイマ割込みレベルを変更する場合は、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)、タイマ割込みレベル(CFG_TIMINTLVL)の設定を変更してください。

6.9 CPU 初期化ルーチン

6.9.1 CPU 初期化ルーチンの作成

CPU 初期化ルーチンは、CPU リセットによって初めに実行されるプログラムです。「4.11.1 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_cpuini.c(C 言語記述)の内容も参考にしてください。CPU 初期化ルーチンのシンボル名称(_KERNEL_H_CPUINI)はシステムで固定となっています。

CPU 初期化ルーチンでは、必要に応じて VBR レジスタ、SBR レジスタの初期化などを行ってください。

6.9.2 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（ベクタ番号 0）に登録します。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。

- ベクタ番号 0：パワーオンリセット

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 6-6のように作成してください。

```
    .SECTION C_hivct, DATA, LOCATE=0    ←リセットベクタのセクション名を C_hivct とし、
    .IMPORT   _KERNEL_H_CPUINI            配置アドレスを 0 番地とします。
:
    .DATA.L  _KERNEL_H_CPUINI           ←ベクタ番号 0(パワーオンリセット)のプログラムアド
    .END                                  レスを_KERNEL_H_CPUINI とします。
```

図6-6 リセットベクタの作成例

(2) コンフィギュレータで設定

コンフィギュレータでは、自動的に割込み番号 0 に対して CPU 初期化ルーチンのアドレス（KERNEL_H_CPUINI）が定義されます。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（ベクタテーブル形式で非分割を指定した場合）または C_hiresvct（ベクタテーブル形式で分割を指定した場合）をリンク時に 0 番地に配置してください。

6.10 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   vsys_dwn (H type,H inf1,B inf2,B inf3,H inf4,UW inf5)
```

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、「10 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。

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、対策を行ってください。

7. コンフィギュレーション

7.1 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、「7.4 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。

7.1.1 リンク方式

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

(1) リンク

HI1000/4 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、アプリケーションを 1 つのロードモジュールにリンクします。
ロードモジュールの生成を図 7-1に示します。

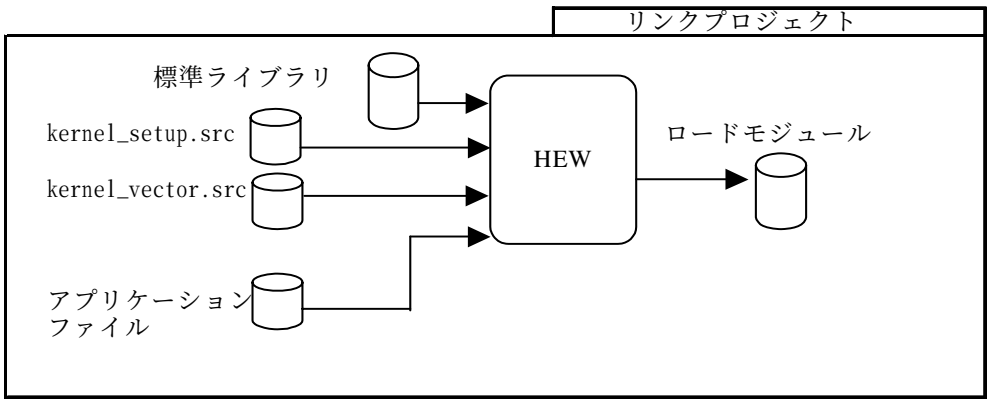


図7-1 ロードモジュールの生成

7.1.2 コンフィギュレータの出力ファイル

コンフィギュレータが出力するファイルは、表 7-1の通りです。

表7-1 コンフィギュレータの出力ファイル

項番	ファイル名	内容
1	kernel_setup.src	セットアップファイル
2	kernel_id.h	ID 自動割付け結果ヘッダファイル（C 言語用）
	kernel_id.inc	ID 自動割付け結果ヘッダファイル（アセンブリ言語用）
3	kernel_macro.h	カーネル構成定数定義ヘッダファイル（C 言語用）
	kernel_macro.inc	カーネル構成定数定義ヘッダファイル（アセンブリ言語用）
4	kernel_sysini.src	システム初期化ルーチン登録ファイル
5	kernel_vector.src	ベクタテーブル生成情報定義ファイル

(1) セットアップファイル

カーネルライブラリとリンクすることで、必要な機能だけが組み込まれたカーネルを生成します。また、TCB(タスク管理ブロック)などのカーネルのテーブルなどが生成されます。カーネルは、この情報を随時参照・更新しながら動作します。

(2) ID 番号自動割付け結果ヘッダファイル(C 言語用、アセンブリ言語用)

コンフィギュレータでは、ID 番号の自動割付けに対応しており、その結果が kernel_id.h として出力されます。例えば、コンフィギュレータでタスクの生成時にタスク ID として"ID_MainTask"を指定した場合、コンフィギュレータは以下のような内容の kernel_id.h を出力します。

```
#define     ID_MainTask     1
```

(3) カーネル構成定数定義ヘッダファイル(C 言語用、アセンブリ言語用)

kernel_macro.h, kernel_macro.inc は、hihead¥kernel.h, kernel.inc（「6.1.1　標準ヘッダファイル」参照）からインクルードされています。

(4) システム初期化ルーチン登録ファイル

コンフィギュレータは、登録した初期化ルーチンの呼び出しを行う初期化ルーチンソースファイル kernel_sysini.src を生成します。

(5) ベクタテーブル生成情報定義ファイル

コンフィギュレータは、割込み、CPU 例外ハンドラを登録したベクタテーブルソースファイル kernel_vector.src を生成します。

7.2 ディレクトリ構成

出荷時のディレクトリ構成は、表 7-2のようになっています。提供ファイルの詳細は、製品添付のリリースノートを参照してください。以下 H8SX/1650 アドバンストモードを例に説明します。

表7-2 出荷時のディレクトリ構成

項番	ディレクトリ、ファイル名 *1	内容
1	kernel¥hihead	標準ヘッダファイルディレクトリ
2	kernel ¥hilib¥	カーネルライブラリ格納ディレクトリ *3
3	kernel¥samples¥h8sx¥***nnnnz***smp¥***nnnnz***smp.hws	サンプルワークスペース *2
4	kernel¥samples¥h8sx¥***nnnnz***smp¥***nnnnz***smp¥***nnnnz***smp.hwp	サンプルプロジェクト*2
5	kernel¥samples¥h8sx¥***nnnnz***smp¥src	サンプルファイル、システム構築用ファイルのディレクトリ
6	kernel¥samples¥h8sx¥ ***nnnnz***smp¥src¥***nnnnz***.hcf	サンプルコンフィギュレータ設定ファイル(HCF ファイル)
7	kernel¥samples¥h8sx¥***nnnnz***smp¥***nnnnz***smp¥obj	オブジェクト、ロードモジュール出力ディレクトリ(アドバンスト、ミドルモード用)

【注】 *1 ***nnnn*** は、デバイス名を示します。***z*** は、CPU 動作モードを示します（a:アドバンストモード、n:ノーマルモード）H8SX/1650 アドバンストモードの例では、***nnnnz*** が 1650a になります。

*2 サンプルプロジェクトファイルは、あらかじめサンプルワークスペースに登録済みのため、通常はユーザが直接サンプルプロジェクトファイルを操作する必要はありません。

*3 カーネルライブラリのサブディレクトリ構成については、「表 8-1 提供カーネルライブラリディレクトリ一覧」を参照してください。

7.3 作 業 手 順

通常は、以下の手順で作業します。

- kernel¥samples¥h8sx¥ ***nnnnz***smp¥src¥***nnnnz***.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。
- kernel¥samples¥h8sx¥ ***nnnnz***smp¥src¥***nnnnz***.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¥samples¥h8sx¥***nnnnz***smp¥src としてください。
- コンフィギュレータを終了します。
- ***nnnnz***smp.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(***nnnnz***smp など)を選択します。なお、使用しないプロジェクトは削除して構いません。
- 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。その結果、kernel¥samples¥h8sx¥***nnnnz***smp¥***nnnnz***smp¥obj フォルダに、ロードモジュールファイルが生成されます。

【注】 ***nnnn*** は、デバイス名を示します。***z*** は、CPU 動作モードを示します（a:アドバンストモード、n:ノーマルモード）。

7.4　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。

7.4.1　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、いくつかのヘッダ、アセンブラソースファイル(表 7-1参照)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 7-2のようになります。

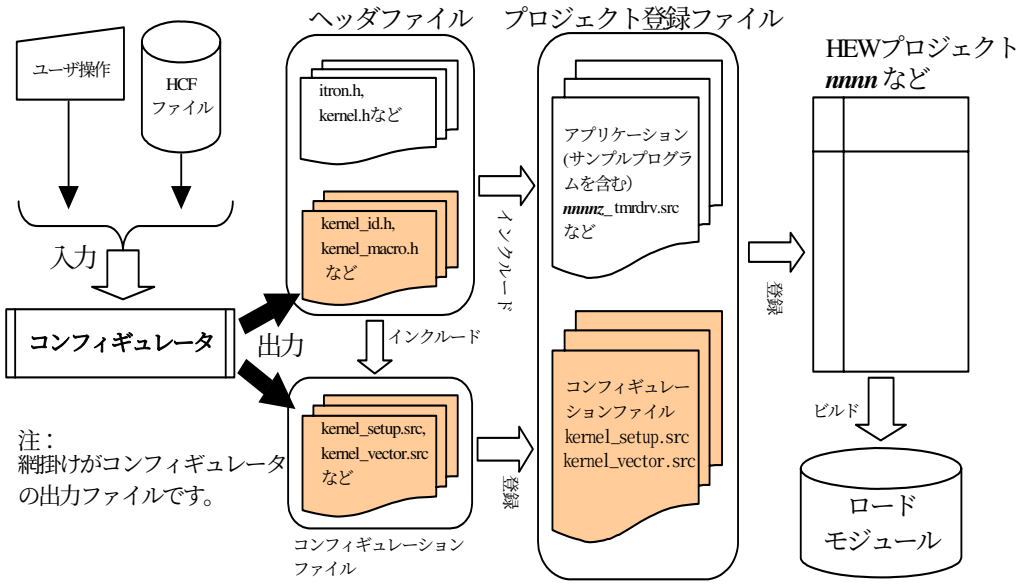


図7-2　システム構築におけるコンフィギュレータの位置付け

7.4.2 コンフィギュレータの構成

Windows の[ｽﾀｰﾄ]メニューから[すべてのﾌﾟﾛｸﾞﾗﾑ]を選択し、[HI1000-4]プログラムフォルダの[HI1000-4 ｺﾝﾌｨｷﾞｭﾚｰﾀ]を選択すると、コンフィギュレータが起動します。

図 7-3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。

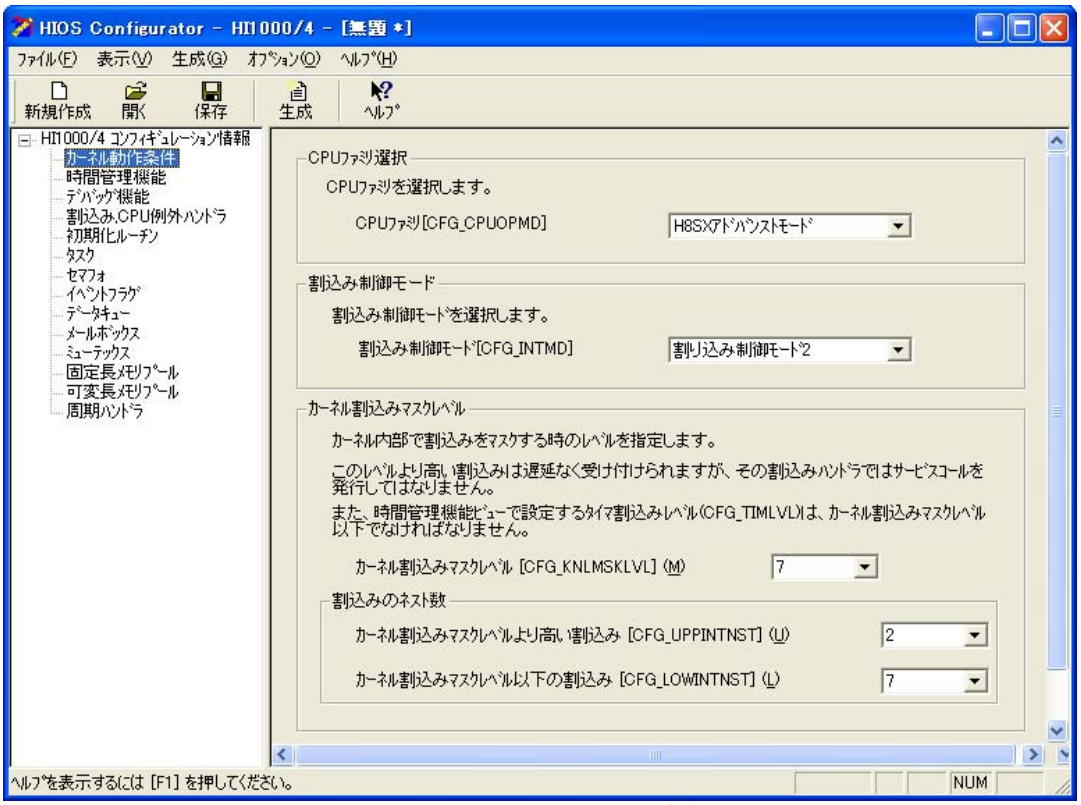


図7-3　コンフィギュレータ概観

7.4.3 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]メニューから[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7-4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、使用するデバイス、CPU ファミリに応じて kernel¥samples¥***mmmm***¥***nnnn***zsmp¥src フォルダに生成する必要があります。

【注】 ***mmmm*** は、CPU ファミリ名を示します。***nnnn*** は、デバイス名を示します。

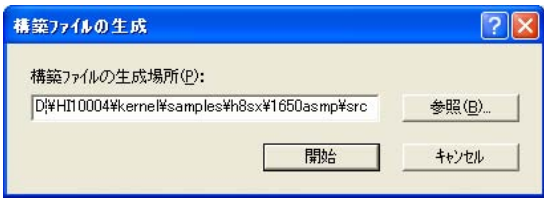


図7-4 フォルダ選択ダイアログ

これにより、表 7-1に示すファイルが指定したフォルダに生成されます。

【注】コンフィギュレーションファイルを生成するフォルダに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。

7.4.4 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。 表 7-3に、各設定項目を示します。

ID 番号の自動割付けで、ID 番号として名前を指定すると、コンフィギュレータが自動的に ID 番号を付与し、その結果を kernel_id.h に出力します。なお、名前を指定する場合は、他の名前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7-3　コンフィギュレータの設定項目

ビュー	項番	設定項目	名称
		内容	
1. カーネル動作条件ビュー	1.1	CPU ファミリ	CFG_CPUOPMD
		CPU ファミリを指定します。以下の CPU ファミリを指定できます。 – H8SX アドバンストモード – AE-5 アドバンストモード – H8S/2600 アドバンストモード – H8S/2600 ノーマルモード – H8S/2000 アドバンストモード – H8S/2000 ノーマルモード 詳細は使用する CPU のハードウェアマニュアルを参照してください。	
	1.2	割込み制御モード	割込み制御モード
		割込み制御モードを指定します。CPU ファミリの選択により、指定できる割込み制御モードは変わります。以下の割込み制御モードを指定できます。 – CPU が H8SX・AE-5 の時:割込み制御モード 0・2 – CPU が H8S/2000・2600 の時:割込み制御モード 0～3 詳細は使用する CPU のハードウェアマニュアルを参照してください。	
	1.3	カーネル割込みマスクレベル	CFG_KNLMSKLVL
		カーネル内部で割込みをマスクするときのマスクレベルを指定します。 指定できる値は、割込み制御モード(CFG_INTMD)によって変わります。 – 割込み制御モード 0 の場合、1 を指定します。 – 割込み制御モード 1 の場合、1～3 の値を指定します。 – 割込み制御モード 2 の場合、1～7 の値を指定します。 – 割込み制御モード 3 の場合、1～8 の値を指定します。 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では、サービスコールを発行してはなりません。 カーネル割込みマスクレベルは、kernel_macro.h, kernel_macro.inc に出力されます。	
	1.4	カーネル割込みマスクレベルより高い割込みネスト数	CFG_UPPINTNST
		カーネル割込みマスクレベル(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。	
	1.5	カーネル割込みマスクレベル以下の割込みネスト数	CFG_LOWINTNST
		カーネル割込みマスクレベル(CFG_KNLMSKLVL)以下の割込みの最大ネスト数を指定してください。	

表7-3　コンフィギュレータの設定項目（続き）

<table>
<tr><td rowspan="2">ビュー</td><td rowspan="2">項番</td><td>設定項目</td><td>名称</td></tr>
<tr><td colspan="2">内容</td></tr>
<tr><td rowspan="14">2. 時間管理機能ビュー</td><td rowspan="2">2.1</td><td>時間管理機能の使用</td><td>CFG_TIMUSE</td></tr>
<tr><td colspan="2">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、タイマドライバを作成する必要があります。詳細は、「6.8 タイマドライバ」を参照してください。</td></tr>
<tr><td rowspan="2">2.2</td><td>タイマ割込み番号</td><td>CFG_TIMINTNO</td></tr>
<tr><td colspan="2">タイマ割込みハンドラの割込みベクタ番号を指定します。</td></tr>
<tr><td rowspan="2">2.3</td><td>タイマ割込みレベル</td><td>CFG_TIMINTLVL</td></tr>
<tr><td colspan="2">タイマ割込みハンドラの割込みレベルを指定します。
タイマ割込みレベルは、kernel_macro.h, kernel_macro.inc に出力されます。タイマ初期化ハンドラでは、その値をデバイスの割込みコントローラに設定する必要があります。</td></tr>
<tr><td rowspan="2">2.4</td><td>タイマ割込みハンドラのスタックサイズ</td><td>CFG_TMRSTKSZ</td></tr>
<tr><td colspan="2">タイマ割込みハンドラのスタックサイズを指定します。「9.8 タイマ割込みのスタック」にしたがって、算出したサイズを指定してください。</td></tr>
<tr><td rowspan="2">2.5</td><td>タイムティック周期の分子(TIC_NUME)と分母(TIC_DENO)</td><td>CFG_TICNUME、
CFG_TICDENO</td></tr>
<tr><td colspan="2">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 の範囲の整数値を指定できます
TIC_NUME と TIC_DENO は、kernel_macro.h、kernel_macro.inc に出力されます。タイマドライバは、kernel_macro.h、kernel_macro.inc の情報にしたがって、タイムティックの周期を設定しなければなりません。</td></tr>
<tr><td rowspan="2">2.6</td><td>タイムアウト付き同期通信機能</td><td>CFG_TOUTUSE</td></tr>
<tr><td colspan="2">以下のタイムアウト付き同期通信機能を使用する場合は、CFG_ TOUTUSE をチェックしてください。
<table>
<tr><td>オブジェクト</td><td>サービスコール</td><td>オブジェクト</td><td>サービスコール</td></tr>
<tr><td>セマフォ</td><td>twai_sem</td><td>メールボックス</td><td>trcv_mbx</td></tr>
<tr><td>イベントフラグ</td><td>twai_flg</td><td>ミューテックス</td><td>tloc_mtx</td></tr>
<tr><td rowspan="2">データキュー</td><td rowspan="2">tsnd_dtq、
trcv_dtq</td><td>固定長メモリプール</td><td>tget_mpf</td></tr>
<tr><td>可変長メモリプール</td><td>tget_mpl</td></tr>
</table></td></tr>
</table>

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
3. デバッグ機能ビュー	3.1	オブジェクト操作機能	CFG_ACTION
		デバッギングエクステンションでオブジェクト操作機能を使用する場合、CFG_ACTION をチェックしてください。 チェックした場合、初期化ルーチンとして、デバッグデーモン初期化ルーチンが自動的に組み込まれます。	
	3.2	サービスコールトレース機能	CFG_TRACE
		サービスコールトレース機能を使用する場合 CFG_TRACE をチェックしてください。	
	3.3	サービスコールトレース機能のタイプ	CFG_TRCTYPE
		トレース機能のタイプを選択します。詳細は「4.10.2 サービスコールトレース機能」を参照してください。 – ターゲットトレース – ツールトレース – ターゲットトレース(短縮) – ツールトレース(短縮)	
	3.4	トレース取得数	CFG_TRCCNT
		トレースで取得する情報の数を指定します。	
	3.5	オブジェクト状態取得数	CFG_TRCOBJCNT
		トレース取得時に、オブジェクト履歴として取得する最大数を指定してください。 オブジェクト状態取得数には 0～32 まで指定できます。 ターゲットトレース(短縮)、ツールトレース(短縮)を選択した場合、オブジェクト履歴数は 0 が設定されます。	
	3.6	バッファサイズ	CFG_TRCBUFSZ
		ターゲットトレース、ターゲットトレース(短縮)を使用する場合のトレースバッファサイズを表示します。	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
4. 割込み、CPU 例外ハンドラビュー	4.1	最大割込みベクタ番号	CFG_MAXVCTNO
		最大の割込みハンドラのベクタ番号を設定します。 ベクタ番号は、使用するデバイスのハードウェアマニュアルを参照してください。	
	4.2	ベクタテーブル形式	CFG_VCTFMT
		ベクタテーブルの形式を指定します。 – 分割：0番地からベクタテーブルを配置する場合に選択します。 – 非分割：0 番地、VBR 指定の番地にベクタを分割して配置する場合に選択します。	
	4.3	割込みハンドラ情報の設定	-
		登録するベクタを選択し、割込みハンドラの情報として、4.3.1 から 4.3.2 までを設定します。	
	4.3.1	アドレス	-
		割込みハンドラの先頭アドレスを設定します。	
	4.3.2	記述言語	-
		高級言語(TA_HLNG)、アセンブリ言語(TA_ASM)を選択します。	
	4.4	割込みハンドラ用スタック情報の設定	-
		割込みハンドラのスタック情報として、4.4.1 から 4.4.2 までを設定します。	
	4.4.1	割込みハンドラのスタック名称（スタックポインタ）	-
		使用する割込みハンドラのスタック名称（スタックポインタ）を設定します。	
	4.4.2	割込みハンドラのスタックサイズ	-
		使用する割込みハンドラのスタック領域を設定します。 スタック領域のサイズの算出方法は、「9.7 割込みハンドラのスタック」を参照してください。	
5. 初期化ルーチンビュー	5.1	初期化ルーチンアドレス	-
		初期化ルーチンの先頭アドレスを設定します。	
	5.2	初期化ルーチンスタックサイズ	-
		初期化ルーチンで使用するスタックサイズを指定します。初期化ルーチンでは、カーネルスタックを使用するため、ここで指定したサイズは、カーネルスタックサイズに影響します。詳細は「9.9 カーネルのスタック」を参照してください。	
	5.2	記述言語	-
		高級言語(TA_HLNG)、アセンブリ言語(TA_ASM)を選択します。	
6. タスクビュー	6.1	最大タスク ID	CFG_MAXTSKID
		最大タスク ID の値を指定します。1～255 の範囲を指定できます。 使用可能なタスクの ID の範囲は 1～CFG_MAXTSKID となります。	
	6.2	最大タスク優先度	CFG_MAXTSKPRI
		最大タスク優先度の値を指定します。1～31 の範囲を指定できます。 使用可能なタスク優先度の範囲は、1～CFG_MAXTSKPRI となります。	
	6.3	スタック数	CFG_NUMTSKSTK
		タスクスタックの数を指定します。指定した数だけ、タスクスタックが生成されます。 生成されたスタックの名称(スタックポインタ)を、タスクの登録で設定してください。	
	6.3.1	スタック領域の設定	-
		タスクが使用するスタック領域（スタックポインタ）、サイズを定義します。	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
6. タスクビュー	6.4	タスク情報の設定	-
		該当タスクの情報として、6.4.1 から 6.4.7 までを設定します。	
	6.4.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なタスクの ID 番号の範囲は 1～CFG_MAXTSKID となります。	
	6.4.2	タスクアドレス	-
		該当タスク ID のタスクの先頭アドレスを定義します。	
	6.4.3	タスク起動時の優先度	-
		該当タスク ID のタスク起動時の優先度を定義します。 定義できる値は 1～CFG_MAXTSKPRI の範囲内です。	
	6.4.4	属性	-
		生成後、起動(TA_ACT)をチェックすると、起動時にタスクは実行可能状態に設定されます。タスクの起動は、タスクを登録した順序に関係なく、ID 順で起動されます。チェックした場合、タスク起動時にはタスクの拡張情報が渡されます。	
	6.4.5	記述言語	-
		高級言語(TA_HLNG)、またはアセンブリ言語(TA_ASM)を選択します。	
	6.4.6	タスクスタック	-
		該当タスク ID のタスクが使用するスタック領域を選択します。 該当タスク ID のタスクが他のタスクのスタックと共有する場合には、スタック領域の定義で同じスタックを指定してください。	
	6.4.7	拡張情報	-
		該当タスク ID のタスクの拡張情報を設定します。 拡張情報はタスクの ID 毎に設定が可能です。拡張情報には、対象のオブジェクトに関する情報を入れるなどの目的で確保したメモリの領域をパケットとして、その先頭アドレスを設定します。	
7. セマフォビュー	7.1	最大セマフォ ID	CFG_MAXSEMID
		最大セマフォ ID の値を指定します。0～255 の範囲を指定できます。 使用可能なセマフォの ID の範囲は 1～CFG_MAXSEMID となります。 セマフォを使用しない場合は 0 を指定してください。	
	7.2	セマフォ情報の設定	-
		該当セマフォの情報として、7.2.1 から 7.2.3 までを設定します。	
	7.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なセマフォの ID 番号の範囲は 1～CFG_MAXSEMID となります。	
	7.2.2	セマフォ資源数の最大値	-
		該当セマフォ ID の最大資源数を設定します。1～65535 の範囲が指定できます。	
	7.2.3	セマフォ資源数の初期値	-
		該当セマフォ ID の資源数の初期値を設定します。0～セマフォ資源数の最大値の範囲が指定できます。	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
8. イベントフラグビュー	8.1	最大イベントフラグ ID	CFG_MAXFLGID
		最大イベントフラグ ID の値を指定します。0〜255 の範囲を指定できます。 使用可能なイベントフラグの ID の範囲は 1〜CFG_MAXFLGID となります。 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。	
	8.2	イベントフラグ情報の設定	-
		該当イベントフラグの情報として、8.2.1 から 8.2.4 までを設定します。	
	8.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なイベントフラグの ID 番号の範囲は 1〜CFG_ MAXFLGID となります。	
	8.2.2	属性	-
		該当イベントフラグ ID の属性を設定します。 – TA_WMUL : 複数タスクの待ちを許す。 – TA_CLR : 待ちタスクを解除する時にビットパターンをクリアする。	
	8.2.3	待ちタスクキューの並び方	-
		該当イベントフラグ ID の待ちタスクキューの並び方は、以下に設定されます。 – TA_TFIFO : 待ちタスクのキューイングは FIFO	
	8.2.4	初期ビットパターン	-
		該当イベントフラグ ID のビットパターンの初期値を設定します。16 ビット範囲のビットパターンが指定できます。	
9. データキュービュー	9.1	最大データキューID	CFG_MAXDTQID
		最大データキューID の値を指定します。0〜255 の範囲を指定できます。 使用可能なデータキューの ID の範囲は 1〜CFG_MAXDTQID となります。 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。	
	9.2	データキュー情報の設定	-
		該当データキューの情報として、9.2.1 から 9.2.3 までを設定します。	
	9.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なデータキューの ID 番号の範囲は 1〜CFG_ MAXDTQID となります。	
	9.2.1	データ数	-
		該当データキューID のデータの数を指定します。0〜65535 の範囲が指定できます。	
	9.2.2	待ちタスクキューの並び方	-
		該当データキューID の待ちタスクキューの並び方は、以下に設定されます。 – TA_TFIFO : 待ちタスクのキューイングは FIFO	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
10. メールボックスビュー	10.1	最大メールボックス ID	CFG_MAXMBXID
		最大メールボックス ID の値を指定します。0〜255 の範囲を指定できます。 使用可能なメールボックスの ID の範囲は 1〜CFG_MAXMBXID となります。 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。	
	10.2	最大メッセージの最大優先度	CFG_MAXMSGPRI
		メッセージの最大優先度を指定します。1〜255 の範囲を指定できます。 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。	
	10.3	メールボックス情報の設定	-
		該当メールボックスの情報として、10.3.1 から 10.3.4 までを設定します。	
	10.3.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なメールボックスの ID 番号の範囲は 1〜CFG_ MAXMBXID となります。	
	10.3.2	メッセージの最大優先度	-
		メッセージの最大優先度を指定します。1〜255 の範囲を指定できます。 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。	
	10.3.3	待ちタスクキューの並び方	-
		該当メールボックス ID の待ちタスクキューの並び方を設定します。 – 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	
	10.3.4	メッセージキューの並び方	-
		該当メールボックス ID のメッセージキューの並び方を設定します。 – 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	
11. ミューテックスビュー	11.1	最大ミューテックス ID	CFG_MAXMTXID
		使用するミューテックスの数を指定します。0〜255 の範囲を指定できます。 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。 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。	
	11.2	ミューテックス情報の設定	-
		該当ミューテックスの情報として、11.2.1 から 11.2.3 までを設定します。	
	11.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能なミューテックスの ID 番号の範囲は 1〜CFG_ MAXMTXID となります。	
	11.2.2	属性	-
		優先度上限プロトコル(TA_CELING)に設定されます。	
	11.2.3	上限優先度の最大値	-
		該当ミューテックス ID の上限優先度の最大値を指定します。 指定できる最大値は CFG_MAXTSKPRI(最大タスク優先度)までとなります。	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
12. 固定長メモリプールビュー	12.1	最大固定長メモリプール ID	CFG_MAXMPFID
		最大固定長メモリプール ID の値を指定します。0～255 の範囲を指定できます。 使用可能な固定長メモリプールの ID の範囲は 1～CFG_MAXMPFID となります。 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。	
	12.2	固定長メモリプール情報の設定	-
		該当固定長メモリプールの情報として、12.2.1 から 12.2.4 までを設定します。	
	12.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能な固定長メモリプールの ID 番号の範囲は 1～CFG_MAXMPFID となります。	
	12.2.2	獲得可能数	-
		該当固定長メモリプール ID で獲得可能なメモリブロック数を指定してください。 1～65535 までの範囲が指定できます。	
	12.2.3	サイズ	-
		該当固定長メモリプール ID の固定長メモリ 1 ブロックのサイズを指定します。 2～65530 までの範囲が指定できます。	
	12.2.4	待ちタスクキューの並び方	-
		該当固定長メモリプール ID の待ちタスクキューの並び方は、以下に設定されます。 – TA_TFIFO : 待ちタスクのキューイングは FIFO	
13. 可変長メモリプールビュー	13.1	最大可変長メモリプール ID	CFG_MAXMPLID
		最大可変長メモリプール ID の値を指定します。0～255 の範囲を指定できます。 使用可能な可変長メモリプールの ID の範囲は 1～CFG_MAXMPLID となります。 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。	
	13.2	可変長メモリプール情報の設定	-
		該当可変長メモリプールの情報として、13.2.1 から 13.2.3 までを設定します。	
	13.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能な可変長メモリプールの ID 番号の範囲は 1～CFG_MAXMPLID となります。	
	13.2.2	可変長メモリプール領域サイズ	-
		該当可変長メモリプール ID の可変長メモリのサイズを指定します。 18～65534 までの範囲が指定できます。	
	13.2.3	待ちタスクキューの並び方	-
		該当可変長メモリプール ID の待ちタスクキューの並び方は、以下に設定されます。 – TA_TFIFO : 待ちタスクのキューイングは FIFO	

表7-3　コンフィギュレータの設定項目（続き）

ビュー	項番	設定項目	名称
		内容	
14. 周期ハンドラビュー	14.1	最大周期ハンドラ ID	CFG_MAXCYCID
		最大周期ハンドラ ID の値を指定します。0～254 の範囲を指定できます。 使用可能な周期ハンドラの ID の範囲は 1～CFG_MAXCYCID となります。	
	14.2	周期ハンドラ情報の設定	-
		該当周期ハンドラの情報として、14.2.1 から 14.2.7 までを設定します。	
	14.2.1	ID 番号/ ID 名称	-
		ID 番号、または ID 名称を指定します。 使用可能な周期ハンドラの ID 番号の範囲は 1～CFG_MAXCYCID となります。	
	14.2.2	周期ハンドラアドレス	-
		該当周期ハンドラ ID の周期ハンドラの先頭アドレスを設定します。	
	14.2.3	拡張情報	-
		該当周期ハンドラ ID の周期ハンドラの拡張情報を設定します。 拡張情報は周期ハンドラの ID 毎に設定が可能です。拡張情報には、対象のオブジェクトに関する情報を入れるなどの目的で確保したメモリの領域をパケットとして、その先頭アドレスを設定します。	
	14.2.4	起動情報の起動周期	-
		該当周期ハンドラ ID の周期ハンドラの起動周期を設定します。 0x1～0x7fffffff までの範囲が指定できます。	
	14.2.5	起動情報の起動位相	-
		該当周期ハンドラ ID の周期ハンドラの起動位相を設定します。 0x0～周期ハンドラの起動周期までの範囲が指定できます。	
	14.2.6	属性	-
		該当周期ハンドラ ID の周期ハンドラ属性を指定します。 – TA_STA：生成後、動作状態へ設定する。 – TA_PHS：起動位相を保存する。	
	14.2.7	記述言語	-
		高級言語(TA_HLNG)、またはアセンブリ言語(TA_ASM)を選択します。	

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(a)～(c)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、CPU ファミリの各種デバイスに対応したワークスペースファイル ***nnnnz***smp.hws を提供しています。***nnnnz***smp.hws をダブルクリックすると、HEW が起動して ***nnnnz***smp.hws が開きます。

nnnnzsmp.hws には、各種デバイスに対応したサンプルのプロジェクトがあらかじめ登録されています。

以下に提供しているサンプルプロジェクトを示します。

- 1650asmp：H8SX/1650 アドバンストモード用プロジェクト
- 1525asmp：H8SX/1525 アドバンストモード用プロジェクト
- 2655asmp：H8S/2655 アドバンストモード用プロジェクト
- 2655nsmp：H8S/2655 ノーマルモード用プロジェクト
- 2148asmp：H8S/2148 アドバンストモード用プロジェクト
- 2148nsmp：H8S/2148 ノーマルモード用プロジェクト
- ae57smp：AE57 用プロジェクト

使用するデバイス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。

サンプルプロジェクトをダブルクリックすると、図 8-1のように HEW のワークスペースウィンドウが開きます。

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、最適化リンケージエディタ、オブジェクトコンバートを一連の流れで実行し、ロードモジュールを生成します。

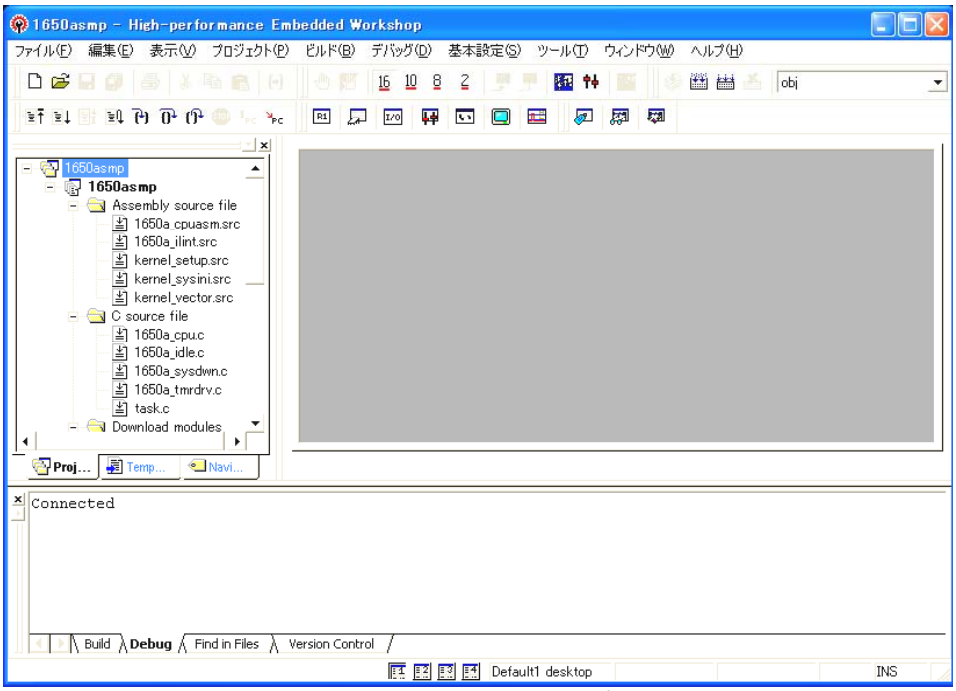


図8-1　プロジェクトの選択

8.1 提供カーネルライブラリ

提供カーネルライブラリのディレクトリを、アプリケーションが使用する CPU ファミリ、機能の有無で選択します。表 8-1に示す提供カーネルライブラリのディレクトリ一覧から、以下の(a)～(c)の条件に応じて、ディレクトリを選択してください。

(a) CPUファミリは、使用するCPUファミリに応じて選択します。

(b) h8sx_advの場合、MACレジスタの有無はタスクのコンテキストにMACレジスタを含むかどうかを選択します。

(c) E6000Hデバッグの有無は、E6000HエミュレータのRTOSデバッグ機能（タスクの時間計測等）を使用するかどうかを選択します。

表8-1 提供カーネルライブラリのディレクトリ一覧

項番	CPU ファミリ	ディレクトリ名*	機能の有無	
			MAC レジスタ	E6000H RTOS デバッグ
1	h8sx_adv	mac_dbg	有り(mac)	有り(dbg)
2		mac_nodbg		無し(nodbg)
3		nomac_dbg	無し(nomac)	有り(dbg)
4		nomac_nodbg		無し(nodbg)
5	ae57	dbg	無し	有り(dbg)
6		nodbg		無し(nodbg)
7	h8s26_adv	dbg	有り	有り(dbg)
8		nodbg		無し(nodbg)
9	h8s26_nor	dbg	有り	有り(dbg)
10		nodbg		無し(nodbg)
11	h8s20_adv	dbg	無し	有り(dbg)
12		nodbg		無し(nodbg)
13	h8s20_nor	dbg	無し	有り(dbg)
14		nodbg		無し(nodbg)

【注】* ディレクトリ名は、***yyy_zzz*** の形式で示します。***yyy*** は、MAC レジスタを示します(mac：有り、nomac：無し)。***zzz*** は、E6000H の RTOS デバッグ機能の使用を示します(dbg：有り、nodbg:無し)。

次にカーネルライブラリファイルをサービスコールのパラメータチェック機能、共有スタック機能の有無で選択します。

表 8-2に示すカーネルライブラリファイル一覧から、以下の(a)～(b)の条件に応じて、使用するカーネルライブラリを選択してください。

(a) パラメータチェックは、サービスコールのパラメータをチェックする機能を使用する場合に選択します。また、構築情報の設定のチェックも行います。

(b) 共有スタックは、タスク間でスタック領域を共有する機能を使用する場合に選択します。

【注】 アプリケーションで共有スタック機能を使用している場合には、必ず共有スタック機能ありのライブラリを選択してください。間違ったライブラリを選択した場合、システムの正常な動作は保証されません

表8-2　カーネルライブラリファイル一覧

項番	ファイル名*	機能の有無	
		パラメータチェック	共有スタック
1	hiknl_nn.lib	無し(n)	無し(n)
2	hiknl_ns.lib	無し(n)	有り(s)
3	hiknl_pn.lib	有り(p)	無し(n)
4	hiknl_ps.lib	有り(p)	有り(s)

【注】*　ファイル名は、hiknl_***xy***.lib の形式で示します。***x*** は、パラメータチェック機能の有無(p：有り、n：無し)を示します。***y*** は、共有スタック機能の有無(s：有り、n：無し)を示します。

8.2 セクション構成

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 8-3に提供ファイルのセクション一覧を示します。

HI1000/4 の表 8-3に示す全てのセクションは、必ずビッグエンディアン空間に配置してください。

また、サービスコールのパラメータ、リターンパラメータに使用する領域もビッグエンディアン空間に配置してください。

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。

表8-3　セクション一覧

項番	ディレクトリ、ファイル名 *1	セクション名	内容
1	カーネルライブラリ	P_hiknl	カーネルプログラム
2	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥kernel_setup.src （コンフィギュレータ出力ファイル）	C_hisetup	コンフィギュレーション情報
		B_histack	タスクスタック領域
		B_hiintstk	割込みスタック領域
		B_hitrc	トレースバッファ
		B_hidtq	データキュー領域
		B_himpf	固定長メモリプール領域
		B_himpl	可変長メモリプール領域
		B_hidbginf	E6000H RTOS デバッグ用領域
		B_hiwrk	カーネル作業領域
3	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥kernel_sysini.src	P_hiknl	カーネル初期化ルーチン
4	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥kernel_vector.src （コンフィギュレータ出力ファイル）	C_hiresvct *2 C_hivct	ベクタテーブル
5	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_sysdwn.c	P_hisysdwn	システムダウンルーチン
6	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_ilint.src	P_hiknl	未定義割り込み詳細情報取得処理
7	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_idle.src	P_hiidle	カーネルアイドリング
8	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_cpu.c kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_cpuasm.src	P_hicpuini	CPU 初期化ルーチン
9	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src ¥***nnnnz***_tmrdrv.c	P_hitmrdrv	タイマドライバ
10	アプリケーションファイル	任意	―

【注】*1 ディレクトリ、ファイル名の ***mmmm*** はファミリ名を、***nnnn*** はデバイス名を示します。
z は、CPU 動作モードを示します（a:アドバンストモード、n:ノーマルモード）。

*2　C_hiresvct は、コンフィギュレータでベクタテーブル形式に分割を指定したときのみ出力されます。

8.3 ロードモジュールの生成

使用するデバイスに対応したサンプルワークスペース ***nnnnz***smp.hws を開いてください。

8.3.1 プロジェクトへの登録

表 8-4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8-4 プロジェクトに登録するソースプログラムファイル

項番	ディレクトリ、ファイル名 *	内容	備考
1	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥kernel_setup.src	セットアップファイル	必須
2	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥kernel_sysini.src	初期化ルーチン	必須
3	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥kernel_vector.src	割込みベクタテーブル	必須
4	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_sysdwn.c	システムダウンルーチン	必須
5	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_ilint.src	未定義割込み詳細情報取得処理	必須
6	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_idle.src	システムアイドリングルーチン	必須
7	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_cpu.c kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_cpuasm.src	CPU 初期化ルーチン	ﾘｾｯﾄで実行する場合は必須
8	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥***nnnnz***_tmrdrv.c	タイマドライバ	
9	kernel¥samples¥***mmmm***¥***nnnnz***smp¥src¥task.c	サンプルタスク	
10	アプリケーションファイル	－	

【注*1 ディレクトリ、ファイル名の ***mmmm*** はファミリ名を、***nnnn*** はデバイス名を示します。
z は、CPU 動作モードを示します（a:アドバンストモード、n:ノーマルモード）。

8.3.2 CPU、コンパイラ、アセンブラのオプションの設定

H8S,H8/300 Standard Toolchain で CPU、コンパイラ、アセンブラのオプションを設定してください。

(1) CPU タブ

図 8-2に CPU タブを示します。

図8-2 CPU タブ

CPU タブでは以下を設定します。

- [CPU 種別]：使用する CPU 種別を指定してください。
- [Multiple/Divide]：乗算器/除算器の使用/未使用を指定してください。
- [スタック計算時のサイズ]：スタックのアドレス計算に「ラージ(4 バイト)」、または「ミディアム(2 バイト)」を指定してください。

サンプルワークスペースは、「ラージ(4 バイト)」を選択しています。「ミディアム(2 バイト)」を選択する場合は、スタックアドレス計算を 2 バイトで行うため、スタックを配置する際は 2 バイトの境界以内に配置してください。

(2)　　コンパイラタブ、アセンブラタブ

図 8-3にコンパイラタブを示します。

図8-3　コンパイラタブ

コンパイラタブのソースカテゴリでインクルードファイルのディレクトリを設定します。

- [ソース]の[インクルードディレクトリ]：
  「$(WORKSPDIR)¥..¥..¥..¥hihead」（ヘッダファイル格納ディレクトリ）、
  「$(WORKSPDIR)¥..¥..¥..¥hisys」（システムファイル格納ディレクトリ）、
  「$((WORKSPDIR)¥src)（サンプルソースファイル格納ディレクトリ）を指定してください。

同様にアセンブリタブのソースカテゴリでインクルードファイルのディレクトリを設定します。

- [ソース]の[インクルードディレクトリ]：
  「$(WORKSPDIR)¥..¥..¥..¥hihead」（ヘッダファイル格納ディレクトリ）、
  「$(WORKSPDIR)¥..¥..¥..¥hisys」（システムファイル格納ディレクトリ）、
  「$((WORKSPDIR)¥src)（サンプルソースファイル格納ディレクトリ）を指定してください。

8.3.3 最適化リンケージエディタのオプション設定

H8S,H8/300 Standard Toolchain の最適化リンカタブで最適化リンケージエディタのオプションを設定してください。

(1) 入力カテゴリ

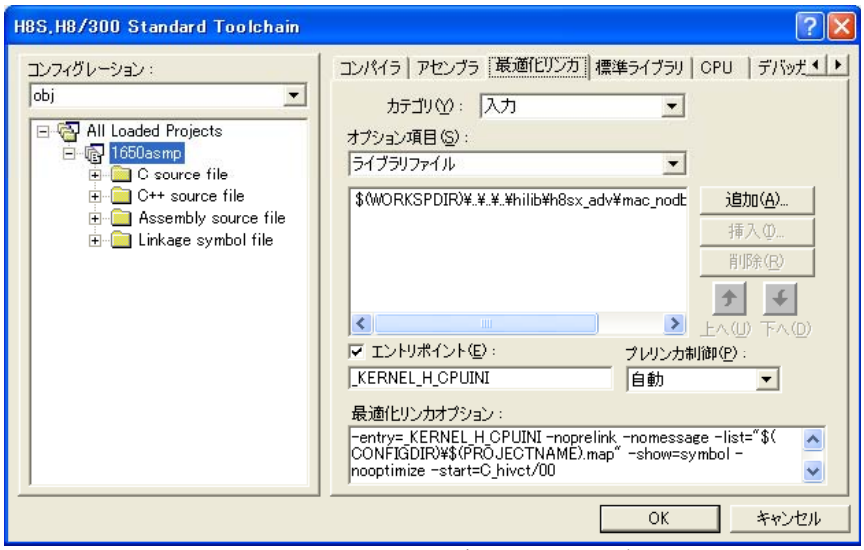


図8-4 Link/Library タブの Input カテゴリ

最適化リンカタブの入力カテゴリにライブラリファイル等を設定します。

- [入力]の[ライブラリファイル]：表 8-1、表 8-2から使用するカーネルライブラリを指定してください。
- [エントリポイント]： CPU 初期化ルーチン「_KERNEL_H_CPUINI」を指定してください。

(2) セクションカテゴリ

最適化リンカタブのセクションカテゴリで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のメモリ構成に合わせて、入力ファイルに含まれる各セクションを指定し、配置するアドレスを指定してください。

HI1000/4 のセクションは、表 8-3のセクション一覧を参照ください。

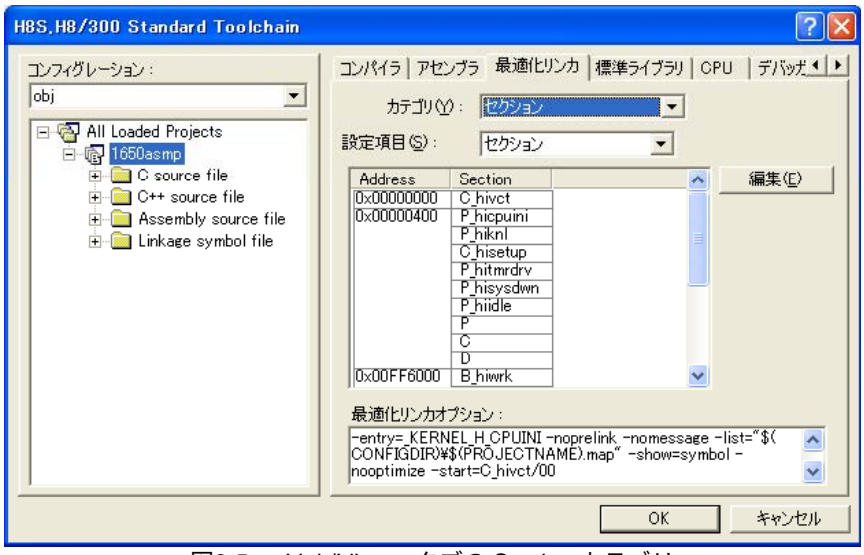


図8-5　Link/Library タブの Section カテゴリ

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、または存在しない場合、最適化リンケージエディタはウォーニングを表示しますが、リンケージは正常に行われます。

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。

セクションの配置について注意事項を以下に示します。

(a)　割込みベクタテーブル(セクション名：C_hiresvct、C_hivct)

コンフィギュレータでベクタテーブル形式を非分割にした場合は、割込みベクタテーブル（セクション名：C_hivct）を 0 番地に配置し、VBR レジスタを 0 固定としてください。

コンフィギュレータでベクタテーブル形式を分割にした場合は、リセットベクタ（セクション名：C_hiresvct）を 0 番地に配置し、割込みベクタテーブル（セクション名：C_hivct）を、VBR レジスタに設定したアドレスに配置してください。

•

(b)　カーネル(セクション名：P_hiknl)

カーネルは偶数番地から配置してください。また、カーネルのセクションを H’xxxx0000～H’xxxxFFFF 番地の間に配置してください。配置する番地の上位アドレス “ xxxx ” は同一にしてください。

(c)　カーネル作業領域(セクション名：B_hiwrk)

カーネル作業領域は偶数番地から配置してください。また、セクションを H’ xxxx0000～H’ xxxxFFFF 番地の間に配置してください。配置する番地の上位アドレス “ xxxx ” は同一にしてください。

(d)　コンフィギュレーション情報(セクション名：C_hisetup)

コンフィギュレーション情報は偶数番地から配置してください。また、セクションを H’ xxxx0000～H’ xxxxFFFF 番地の間に配置してください。配置する番地の上位アドレス“ xxxx ”は同一にしてください。

(e)　E6000H RTOS デバッグ用領域(セクション名：B_hidbginf)

E6000H RTOS デバッグ用領域は、E6000H エミュレータでタスクのデバッグ用機能を使用するとき、に使用するセクションです。この領域は、I/O 領域に設定します。設定する I/O 領域のアドレスは、使用する E6000H のユーザーズマニュアルを参照してください。

8.3.4　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 8-6のように HEW の[ビルド]メニューから[ビルド]または[すべてをビルド]を選択してください。

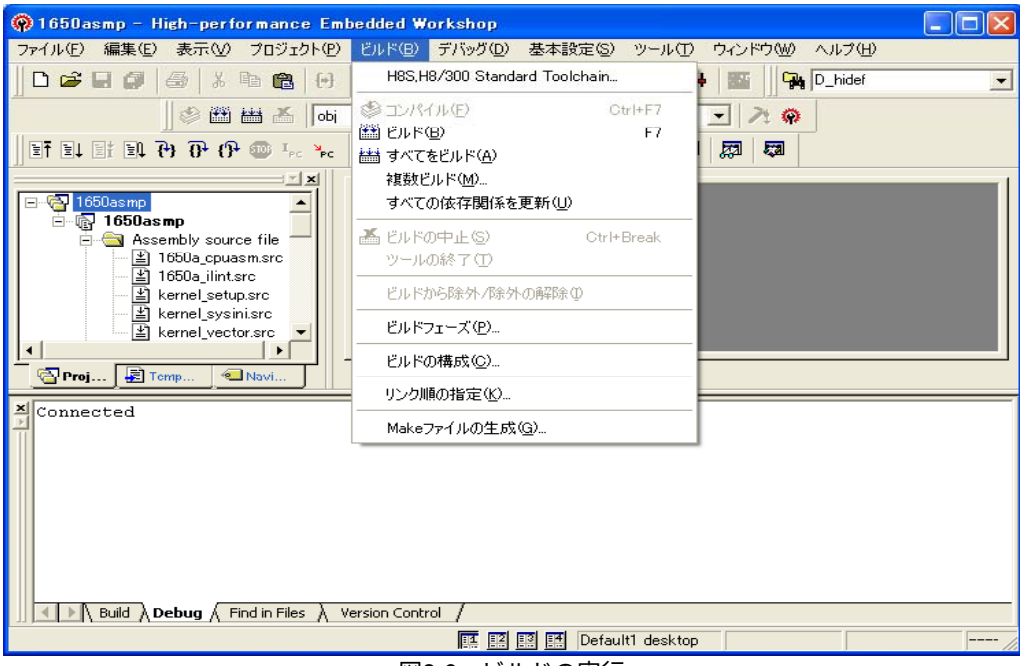


図8-6　ビルドの実行

9.　作業領域サイズの算出

9.1　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 9-1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9-1　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	スタック領域	B_histack	product¥sample¥***nnnnz***smp¥kernel_setup.src
2	割込みスタック領域	B_hiintstk	
3	トレースバッファ領域	B_hitrc	
4	データキュー領域	B_hidtq	
5	固定長メモリプール領域	B_himpf	
6	可変長メモリプール領域	B_himpl	
7	カーネル作業領域	B_hiwrk	product¥sample¥setup.inc
8	デバッグ情報管理領域	B_hidbginf	
9	アプリケーションで使用する作業領域	任意	任意

それぞれのセクションのサイズは、アセンブルリストから知ることができます。

(1)　スタック領域（セクション名 B_histack）

コンフィギュレータで定義、確保するスタック領域です。

(2)　割込みハンドラスタック領域（セクション名 B_hiintstk）

コンフィギュレータで定義、確保する割込みハンドラのスタック領域です。

(3)　トレースバッファ領域（セクション名 B_hitrc）

トレース機能を選択した場合に、トレースバッファ領域が確保されます。

(4)　データキュー領域（セクション名 B_hidtq）

データキューに割り当てられた領域です。

(5)　固定長メモリプール領域（セクション名 B_himpf）

固定長メモリプールとして使用される領域です。

(6)　可変長メモリプール領域（セクション名 B_himpl）

可変長メモリプールとして使用される領域です。

(7)　カーネル作業領域（セクション名 B_hiwrk）

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）、イベントフラグ管理ブロック（FLGCB）、カーネルのスタック領域などが含まれます。

(8)　デバッグ情報管理領域（セクション名 B_hidbginf）

デバッグ情報を出力するための領域です。

(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。

9.2 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り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

割込み発生時には、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を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フローする可能性があります。なお、NMI は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替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。

(3) 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。タイマ初期化ルーチンで使用するスタックサイズについては、「9.10 (3)タイマ初期化ルーチン」を参照してください。

(4) 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、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。

9.3　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 9-1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。

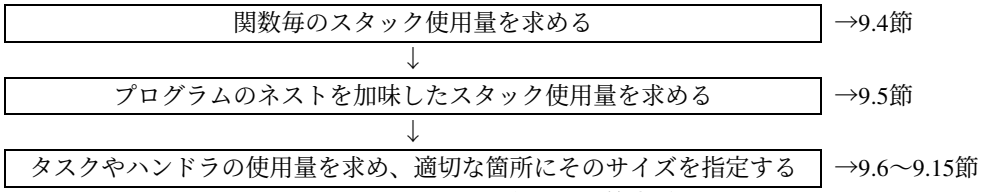


図9-1　スタック使用量の算出手順

9.4 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び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「STACK FRAME INFORMATION」等から知ることができます。

以下に例を示します。

```
************ SOURCE LISTING ************
  Line Pi 0----+----1----+----2----+----3----+----4----+-----…
FILE NAME: E:¥TASK.C
    1     extern int h(char, char *, double );
    2     int
    3     h(char a, register char *b, double c)
    4     {
    5         char *d;
    6         d= &a;
    7         h(*d,b,c);
    8         {
    9             register int i;
   10             i= *d;
   11             return i;
   12         }
   13     }

******* STACK FRAME INFORMATION ********
FILE NAME: E:¥TASK.C
Function (File E:¥TASK  , Line     3): h
  Optimize Option Specified : No Allocation Information Available

Parameter Area Size     : 0x00000000 Byte(s)
Linkage Area Size       : 0x00000008 Byte(s)
Local Variable Size     : 0x00000000 Byte(s)
Temporary Size          : 0x00000000 Byte(s)
Register Save Area Size : 0x0000000c Byte(s)
Total Frame Size        : 0x00000014 Byte(s)        (1)
(以下省略)
```

図9-2　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)で示されるサイズで、図9-2の例では20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『H8S,H8/300シリーズ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ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。

9.5　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
- CPU 例外ハンドラ（TRAPA 命令例外含む）

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。

そして、このネストケース毎に9.4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、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 9-2に示すスタックサイズをさらに加算してください。

表9-2　CPU 例外ハンドラの加算サイズ

項　目	計算式	使用量 (バイト)	備　考
CPU 例外ハンドラが独自に使用するスタックサイズ	ユーザ使用サイズ		
CPU 例外ハンドラ用スタックサイズ	8		
合　計			

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 9-3のプログラムネスト状況を想定します。

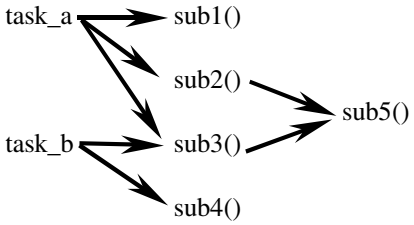


図9-3　プログラムネスト状況

個々の関数のスタック使用量は、表 9-3の通りであるとします。

表9-3　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
2	task_b	40	タスク B の開始関数
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	sub5	80	共通サブルーチン

呼び出し経路を考慮したタスク A､B の使用量は、表 9-4のようになります。

表9-4　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>→sub5<80>	144
	task_a<56 バイト>→sub3<24 バイト>→sub5<80>	160 [最大]
タスク B	task_b<40 バイト>→sub3<24 バイト>→sub5<80>	144 [最大]
	task_b<40 バイト>→sub4<12 バイト>	52

9.6　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、9.5節に従って求めたサイズを表 9-5に代入することで算出します。ここで算出したサイズを、コンフィギュレータのタスクスタックサイズに指定してください。

複数のタスクで同じスタックを共有する場合は､それらのタスク毎に表 9-5でスタックサイズを算出し、最も大きい値をコンフィギュレータに指定してください。

なお、共有スタックを使用した場合の実際に確保されるサイズは、ここで求めたサイズにスタック管理のためにカーネルが使用する 8 バイトを加えた値となります。

表9-5　個々のタスクのスタック使用量

項　目	計算式	使用量 (バイト)	備　考
9.4節、9.5節で求めたサイズ	ユーザ使用サイズ		
OS が使用するスタックサイズ	58(MAC レジスタ使用時) 50(MAC レジスタ未使用)	58 or 50	常に必要です
多重割込み加算値	10 × LOWINTNST *1 ＋ 10 × UPPINTNST *2		割込み制御モード 2,3 使用時
	8 × LOWINTNST *1 ＋ 8 × UPPINTNST *2		割込み制御モード 0,1 使用時
サービスコールトレース用 スタックサイズ	6		サービスコールトレース機能を使用する場合必要
未定義割込み用スタックサイズ *3	アドバンストモード：10 ノーマルモード：8		
合　計			

【注】 *1 カーネル割込みマスクレベル以下の割込みのネスト数
*2 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*3 未定義割込みが発生する場合必要 s

(2)　スタック領域の確保方法

タスクのスタック領域は、コンフィギュレータへの設定によって確保されます。スタック領域全体のサイズは、以下の式で算出されます。

全体サイズ＝Σ((コンフィギュレータに設定した各スタックサイズ)＋8*)

【注】 *共有スタック機能を使用した場合のみ８バイト加算されます。

9.7 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、9.5節に従って求めたサイズを表 9-6に代入することで算出します。

割込みハンドラ用スタック領域は、ハンドラ毎にスタック領域を確保してください。

表9-6 個々の割込みハンドラのスタック使用量

項 目	計算式	使用量 (バイト)	備 考
9.4節､9.5節で求めたサイズ	ユーザ使用サイズ		
OS が使用するスタックサイズ	50		サービスコールを発行する場合
多重割込み加算値	10 × LOWINTNST *1 ＋ 10 × UPPINTNST *2		割込み制御モード 2,3 使用時
	8 × LOWINTNST *1 ＋ 8 × UPPINTNST *2		割込み制御モード 0,1 使用時
サービスコールトレース用スタックサイズ	6		サービスコールトレース機能を使用する場合必要
未定義割込み用スタックサイズ *3	アドバンストモード：10 ノーマルモード：8		
合 計			

【注】 *1 カーネル割込みマスクレベル以下で、かつ自割込みレベルより高い割込みのネスト数

*2 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数

*3 未定義割込みが発生する場合必要

(2) スタック領域の確保方法

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、「6.5 割込みハンドラ」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。

9.8 タイマ割込みのスタック

(1) タイマ割込みハンドラのスタック使用量

タイマ割込みのスタック使用量を表 9-7に代入することで算出します。

表9-7 タイマ割込みのスタック使用量

項 目	計算式	使用量 (バイト)	備 考
9.4節、9.5節で求めたサイズ	ユーザ使用サイズ		
必須分	アドバンストモード：52 ノーマルモード：50	52 or 50	常に必要です
多重割込み加算値	10 × LOWINTNST *1 ＋ 10 × UPPINTNST *2		割込み制御モード 2,3 使用時
	8 × LOWINTNST *1 ＋ 8 × UPPINTNST *2		割込み制御モード 0,1 使用時
未定義割込み用スタックサイズ *3	アドバンストモード：10 ノーマルモード：8		
周期ハンドラが独自に使用するスタックサイズ *4	ユーザ使用サイズ		
	50		サービスコールを発行する場合
	6		サービスコールトレース機能を使用する場合必要
合 計			

【注】 *1 カーネル割込みマスクレベル以下でかつタイマ割込みレベルより高い割込みのネスト数
*2 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*3 未定義割込みが発生する場合必要
*4 周期ハンドラを複数使用する場合は、ハンドラ単位にスタックサイズを算出し、その中から、最大のスタックサイズを加算してください。
周期ハンドラを C 言語で記述した場合、周期ハンドラが独自に使用するスタックサイズは、コンパイルリストに出力される関数のフレームサイズから求めてください。

(2) スタック領域の確保方法

タイマのスタック領域は、コンフィギュレータで CFG_TMRSTKSIZ に設定することで自動的に確保されます。

(3) タイマ割込みハンドラスタック名称

タイマ割込みハンドラのスタック名称はシステムで固定（KERNEL_HI_TIM_SP）となっており、そのアドレスはタイマハンドラスタックのボトムとなっています。

9.9 カーネルのスタック

(1) カーネルのスタック使用量

カーネルのスタック使用量は、表 9-8に代入することで算出します。

表9-8 カーネルのスタック使用量

項 目	計算式	使用量 (バイト)	備 考
OS が使用するスタックサイズ	アドバンストモード：20 ノーマルモード：16	20 or 16	常に必要です
多重割込み加算値	10 × LOWINTNST *1 ＋ 10 × UPPINTNST *2		割込み制御モード 2,3 使用時
	8 × LOWINTNST *1 ＋ 8 × UPPINTNST *2		割込み制御モード 0,1 使用時
未定義割込み用スタックサイズ *3	アドバンストモード：10 ノーマルモード：8		
合 計			

【注】 *1 カーネル割込みマスクレベル以下の割込みのネスト数
*2 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*3 未定義割込みが発生する場合必要

(2) スタック領域の確保方法

カーネルのスタック領域は、コンフィギュレータで自動的に必要サイズが確保されます。確保されるサイズは、以下のいずれかの大きい方の値となります。

- 表 9-8で求めたスタックサイズ
- 「9.10 初期化ルーチンのスタック」の表 9-9で求めた全初期化ルーチンの中で最大のスタックサイズ＋4

9.10 初期化ルーチンのスタック

(1) 初期化ルーチンのスタック使用量

初期化ルーチンのスタック使用量は、表 9-9に代入することで算出します。

表9-9 初期化ルーチンのスタック使用量

項 目	計算式	使用量 (バイト)	備 考
9.4節、9.5節で求めたサイズ	ユーザ使用サイズ		
OS が使用するスタックサイズ	50	50	サービスコールを発行する場合
多重割込み加算値	10 × UPPINTNST *1		割込み制御モード 2,3 使用時
	8 × UPPINTNST *1		割込み制御モード 0,1 使用時
未定義割込み用スタックサイズ *2	アドバンストモード：10 ノーマルモード：8		
合 計			

【注】 *1 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*2 未定義割込みが発生する場合必要

(2) スタック領域の確保方法

コンフィギュレータで初期化ルーチン定義する際は、ここで求めたスタックサイズを指定してください。

初期化ルーチンはカーネルスタックを使用します。カーネルスタックは、9.9節に記載の通り、初期化ルーチンのスタックサイズを考慮した上で、コンフィギュレータが自動的に必要サイズを確保します。

(3) タイマ初期化ルーチン

タイマ初期化ルーチンの名称はシステムで固定（KERNEL_HIPRG_TIMINI）となっています。また、タイマ初期化ルーチンで使用するスタックサイズが「9.9 カーネルのスタック」の(2)で確保されるスタックサイズよりも大きい場合は、タイマ初期化ルーチン内でスタックを切り替えてご使用ください。

9.11　トレース機能のスタック

(1)　　トレース機能のスタック使用量

本スタック領域は、トレース機能を使用する場合にのみ必要です。
トレース機能が使用するスタック使用量を表 9-10に代入することで算出します。

表9-10　トレース機能のスタック使用量

項　目	計算式	使用量 (バイト)	備　考
OS が使用するスタックサイズ	26	26	ターゲットトレース､ツールトレースを使用する場合
	20	20	ターゲットトレース(短縮)、ツールトレース(短縮)を使用する場合
多重割込み加算値	10 × UPPINTNST *1		割込み制御モード 2,3 使用時
	8 × UPPINTNST *1		割込み制御モード 0,1 使用時
未定義割込み用スタックサイズ *2	アドバンストモード：10 ノーマルモード：8		
合　計			

【注】　*1 カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*2 未定義割込みが発生する場合必要

(2)　　スタック領域の確保方法

トレース機能のスタック領域は、コンフィギュレータでトレース機能を選択することで自動的に必要サイズが確保されます。

9.12 トレースバッファ領域

(1) トレースバッファ領域の使用量

トレースバッファ領域の使用量は、表 9-11に代入することで算出します。
本領域は、ターゲットトレース、ターゲットトレース(短縮)を使用する場合にのみ必要です。

表9-11 トレースバッファ領域の使用量

項 目	計算式	使用量(バイト)	備 考
トレースバッファ管理領域	16	16	
トレースエントリ情報領域	30×トレース取得数(TRCCNT)＋オブジェクト状態取得数(TRCOBJCNT)×4		ターゲットトレースを使用する場合
	6×トレース取得数(TRCCNT)		ターゲットトレース(短縮)を使用する場合
合 計			

(2) トレースバッファ領域の確保方法

トレースバッファ領域はコンフィギュレータでトレース情報の最大数とオブジェクト状態取得数を定義することで確保されます。

9.13 データキュー領域

(1) データキュー領域の使用量

データキュー領域の使用量は、表 9-12に代入することで算出します。
データキュー領域全体の使用量は、データキューID 毎に求めたサイズの総和になります。

表9-12 データキュー領域の使用量

内 訳	計算式	使用量(バイト)	備 考
データキュー領域	最大データキューカウント(MAXDTQCNT)× 4		
合 計			

(2) データキュー領域の確保方法

データキュー領域はコンフィギュレータでデータキューID 毎に最大データキューカウントを設定することで確保されます。

9.14　固定長メモリプール領域

(1)　　固定長メモリプール領域の使用量

固定長メモリプール領域の使用量は、表 9-13に代入することで算出します。

固定長メモリプール領域全体の使用量は、固定長メモリプール ID 毎に求めたサイズの総和になります。

表9-13　固定長メモリプール領域の使用量

項　目	計算式	使用量 (バイト)	備　考
固定長メモリプール領域サイズ	固定長メモリブロック数 × (固定長メモリブロックサイズ ＋ 4)		メモリブロック毎に 管理領域(4 バイト)が必要
合　計			

(2)　　固定長メモリプール領域の確保方法

固定長メモリプール領域はコンフィギュレータで固定長メモリプール ID 毎の固定長メモリブロック数と固定長メモリブロックサイズを設定することで確保されます。

9.15　可変長メモリプール領域

(1)　　可変長メモリプール領域の使用量

可変長メモリプール領域の使用量は、表 9-14に代入することで算出します。

可変長メモリプール領域全体の使用量は、可変長メモリプール ID 毎に求めたサイズの総和になります。

表9-14　可変長メモリプール領域の使用量

項　目	計算式	使用量 (バイト)	備　考
可変長メモリプール領域サイズ	可変長メモリプールサイズ＋ (16 × n *)		メモリブロック獲得毎に 管理領域(16 バイト)が必要
合　計			

【注】 * n：可変長メモリブロックを獲得する最大数

(2)　　可変長メモリプール領域の確保方法

可変長メモリプール領域はコンフィギュレータで上表により求めた可変長メモリプール ID 毎の可変長メモリプール領域サイズを設定することで確保されます。

9.16　カーネル作業領域の使用量

(1)　　カーネル作業領域の使用量

カーネル作業領域の使用量は、表 9-15に代入することで算出します。
本表を使用して、カーネルの使用する作業領域を算出します。

表9-15　カーネル作業領域の使用量

項　目	計算式	使用量 (ﾊﾞｲﾄ)	備　考
システム管理テーブル (_KERNEL_HI_SYSMT)	14 ＋4×最大タスク優先度 (CFG_MAXTSKPRI)		必須です
タスク管理ブロック (_KERNEL_HI_TCB)	18×最大タスク ID (CFG_MAXTSKID)		必須です
タスク管理ブロック 2 (_KERNEL_HI_TCB2)	8×最大タスク ID (CFG_MAXTSKID)		タイムアウト機能付サービスコールを使用する場合必要
タスク管理ブロック 3 (_KERNEL_HI_TCB3)	2×最大タスク ID (CFG_MAXTSKID)		必須です
イベントフラグ管理ブロック (_KERNEL_HI_FLGCB)	6×最大イベントフラグ ID(CFG_MAXFLGID)		イベントフラグを使用する場合必要
セマフォ管理ブロック (_KERNEL_HI_SEMCB)	6×最大セマフォ ID (CFG_MAXSEMID)		セマフォを使用する場合必要
メールボックス管理ブロック (_KERNEL_HI_MBXCB)	8×最大メールボックス ID (CFG_MAXMBXID)		メールボックスを使用する場合必要
データキュー管理ブロック 1 (_KERNEL_HI_DTQCB1)	14×最大データキューID (CFG_MAXDTQID)		データキューを使用する場合必要
データキュー管理ブロック 2 (_KERNEL_HI_DTQCB2)	1×最大データキューID (CFG_MAXDTQID)		データキューを使用する場合必要
ミューテックス管理ブロック (_KERNEL_HI_MTXCB)	10×最大ミューテックス ID (CFG_MAXMTXID)		ミューテックスを使用する場合必要
タスクロックミューテックス管理ブロック(_KERNEL_HI_TLMXCB)	4×最大タスク ID (CFG_MAXTSKID)		ミューテックスを使用する場合必要
固定長メモリプール管理ブロック (_KERNEL_HI_MPFCB)	6×最大固定長メモリプール ID (CFG_MAXMPFID)		固定長メモリプールを使用する場合必要
可変長メモリプール管理ブロック (_KERNEL_HI_MPLCB)	20×最大可変長メモリプール ID (CFG_MAXMPLID)		可変長メモリプールを使用する場合必要
周期ハンドラ管理ブロック (_KERNEL_HI_CYHCB)	24×最大周期ハンドラ ID(CFG_MAXCYCID)		周期ハンドラを使用する場合必要
タイマ管理ブロック*1 *2 (_KERNEL_HI_TIMCB, _KERNEL_HI_TIMCB2, _KERNEL_HI_TIMCB3)	10＋ 4＋ 14 (10=TIMCB、4=TIMCB2、14=TIMCB3)		TIMCB：タイマドライバを使用する場合必要 TIMCB2：タイムアウト機能付サービスコールを使用する場合必要 TIMCB3：周期ハンドラを使用する場合必要
トレースバッファ管理ブロック*3 (_KERNEL_TBACB)	4		サービスコールトレース機能を使用する場合必要

項　目	計算式	使用量 (ﾊﾞｲﾄ)	備　考
TX トレース管理ブロック (_KERNEL_TX_TRCCB)	4＋オブジェクト状態取得数 (TRCOBJCNT)×4		ターゲットトレース、またはツールトレース機能を使用する場合必要
	4		ツールトレース(短縮)を使用する場合必要
合　計			

【注】 *1 コンフィギュレータの時間管理機能で”タイムアウト機能を使用する[CFG_TOUTUSE]”チェックボックスをチェックしない場合、タイムアウト機能が使用する領域(TCB2、TIMCB2 領域)は確保されません。

*2 コンフィギュレータの時間管理機能で”時間管理機能を使用する[CFG_TIMUSE]”チェックボックスをチェックしない場合、タイマ管理ブロック(TIMCB、TIMCB2、TIMCB3 領域)およびタイマ関連管理ブロック(TCB2、CYHCB)は確保されません。

*3 コンフィギュレータのデバッグ機能で”サービスコールトレース機能を組み込む[CFG_TRACE]”チェックボックスをチェックしない場合、トレースバッファ管理ブロックは確保されません。

10. 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 10-1に示す情報が渡されます。

表10-1 システムダウンルーチンに渡される情報

システムダウン要因	ｴﾗｰ種別 H type (R0)	パケット内容				
		ｼｽﾃﾑﾀﾞｳﾝ 情報 1 H inf1 (E0)	ｼｽﾃﾑﾀﾞｳﾝ 情報 2 B inf2 (R1L)	ｼｽﾃﾑﾀﾞｳﾝ 情報 3 B inf3 (R1H)	ｼｽﾃﾑﾀﾞｳﾝ 情報 4 H inf4 (E1)	ｼｽﾃﾑﾀﾞｳﾝ 情報 5 UW inf5 (ER2)
コンフィギュレーション情報エラー	H'fffb	エラー番号	H'00	H'00	H'0000	H'00000000
非タスクコンテキストからの ext_tsk サービスコール発行	H'fffd	E_CTX	発生時の CCR	発生時の EXR	0	発生時の PC
タスク実行状態、CPU ロック状態からの ret_int ルーチンの呼び出し	H'fffe	H'0000	tskid	H'00	H'0000	H'00000000
未定義割込みの発生	H'ffff	割込み ベクタ番号	発生時の CCR	発生時の EXR	tskid *	発生時の PC

【注】* 非タスクコンテキストからの未定義割込み発生時、または CPU ロック状態からの未定義割込み発生時は H'0000 が渡されます。

表 10-2にコンフィギュレーション情報エラーの不正内容およびエラー番号を示します。

表10-2 コンフィギュレーション情報不正内容一覧

項番	不正項目		エラー番号
1	アドレス不正	カーネル用スタックポインタ(_KERNEL_HI_OS_SP)が 0 または奇数	H'0101
		タイマ割込み用スタックポインタ(_KERNEL_HI_TIM_SP)が 0 または奇数	H'0102
		カーネル作業領域の先頭アドレス(セクション名「B_hiknl」)が 0 または奇数	H'0103
		TIMCB 領域(_KERNEL_HI_TIMCB)の先頭アドレスが 0 または奇数	H'0104
		TIMCB2 領域(_KERNEL_HI_TIMCB2)の先頭アドレスが 0 または奇数	H'0105
		TCB 領域(_KERNEL_HI_TCB)の先頭アドレスが 0 または奇数	H'0106
		TCB2 領域(_KERNEL_HI_TCB2)の先頭アドレスが 0 または奇数	H'0107
		FLGCB 領域(_KERNEL_HI_FLGCB)の先頭アドレスが 0 または奇数	H'0108
		SEMCB 領域(_KERNEL_HI_SEMCB)の先頭アドレスが 0 または奇数	H'0109
		MBXCB 領域(_KERNEL_HI_MBXCB)の先頭アドレスが 0 または奇数	H'010A
		MPFCB 領域(_KERNEL_HI_MPFCB)の先頭アドレスが 0 または奇数	H'010B
		MPLCB 領域(_KERNEL_HI_MPLCB)の先頭アドレスが 0 または奇数	H'010C
		トレーススタックポインタ(_KERNEL_HI_TRC_SP)が 0 または奇数	H'010D
		トレース管理領域(_KERNEL_TBACB)の先頭アドレスが 0 または奇数	H'010E
		TIMCB3 領域(_KERNEL_HI_TIMCB3)の先頭アドレスが 0 または奇数	H'010F
		CYHCB 領域(_KERNEL_HI_CYHCB)の先頭アドレスが 0 または奇数	H'0110
		TCB3 領域(_KERNEL_HI_TCB3)の先頭アドレスが 0 または奇数	H'0111
		DTQCB1 領域(_KERNEL_HI_DTQCB1)の先頭アドレスが 0 または奇数	H'0112
		MTXCB 領域(_KERNEL_HI_MTXCB)の先頭アドレスが 0 または奇数	H'0113
		TLMXCB 領域(_KERNEL_HI_TLMXCB)の先頭アドレスが 0 または奇数	H'0114
2	ルーチンアドレス不正	初期化ルーチンの先頭アドレス(_KERNEL_HIPRG_SYSINI)が奇数	H'0201
		タイマ初期化設定ルーチンの先頭アドレス(_KERNEL_HIPRG_TIMINI)が奇数	H'0202
3	設定値不正(範囲外)	割込み制御モード(CFG_INTMD)が 4 以上	H'0301
		カーネル割込みマスクレベル(CFG_KNLMSKLVL)が 9 以上	H'0302
		最大タスク優先度(CFG_MAXTSKPRI)が 32 以上	H'0303
		最大タスク ID(CFG_MAXTSKID)が 256 以上	H'0304
		最大イベントフラグ ID(CFG_MAXFLGID)が 256 以上	H'0305
		最大セマフォ ID(CFG_MAXSEMID)が 256 以上	H'0306
		最大メールボックス ID(CFG_MAXMBXID)が 256 以上	H'0307
		最大固定長メモリプール ID(CFG_MAXMPFID)が 256 以上	H'0308
		最大可変長メモリプール ID(CFG_MAXMPLID)が 256 以上	H'0309
		最大周期ハンドラ ID(CFG_MAXCYCID)が 256 以上	H'030A
		最大データキューID(CFG_MAXDTQID)が 256 以上	H'030B
		最大ミューテックス ID(CFG_MAXMTXID)が 256 以上	H'030C
		最大メッセージ優先度(CFG_MAXMSGPRI)が 256 以上	H'030D

項番	不正項目		エラー番号
4	テーブルアドレス不正(0か奇数)	タスク定義テーブル(_KERNEL_HI_TDT)の先頭アドレスが 0 または奇数	H'0401
		固定長メモリプール定義テーブル(_KERNEL_HI_MPFDT)の先頭アドレスが 0 または奇数	H'0402
		可変長メモリプール定義テーブル(_KERNEL_HI_MPLDT)の先頭アドレスが 0 または奇数	H'0403
		未定義割込みハンドラ(_KERNEL_HI_ILT)の先頭アドレスが 0 または奇数	H'0404
		トレースバッファ情報テーブル(_KERNEL_INITRC)の先頭アドレスが 0 または奇数	H'0405
		周期ハンドラ定義テーブル(_KERNEL_HI_CYCDT)の先頭アドレスが 0 または奇数	H'0406
		セマフォ定義テーブル(_KERNEL_HI_SEMDT)の先頭アドレスが 0 または奇数	H'0407
		イベントフラグ定義テーブル(_KERNEL_HI_FLGDT)の先頭アドレスが 0 または奇数	H'0408
		メールボックス定義テーブル(_KERNEL_HI_MBXDT)の先頭アドレスが 0 または奇数	H'0409
		データキュー定義テーブル(_KERNEL_HI_DTQDT)の先頭アドレスが 0 または奇数	H'040a
		時間定義テーブル(_KERNEL_HI_TIMDT)の先頭アドレスが 0 または奇数	H'040b
5	項目不正	タスク起動時の優先度が 0 または最大タスク優先度(CFG_MAXTSKPRI)より大きい値	H'0501
		タスク先頭アドレスが 0 または奇数	H'0502
		タスクスタックポインタが 0 または奇数	H'0503
		固定長メモリブロックサイズ(BLFLEN)が 0、奇数または 65530 バイト以上	H'0504
		固定長メモリプール領域の先頭アドレスが 0 または奇数	H'0505
		可変長メモリプールサイズが 0、奇数または 16 バイト以下	H'0506
		可変長メモリプール領域の先頭アドレスが 0 または奇数	H'0507
		トレースバッファアドレス(TRACE BUFFER ADDRESS)が 0 または奇数	H'0508
		周期ハンドラの先頭アドレスが 0 または奇数	H'0509
		周期ハンドラの起動周期が 0 または H'80000000 以上	H'050A
		周期ハンドラの起動位相が起動周期より大きい値	H'050B
		セマフォの資源数の初期値が最大資源数より大きい値	H'050C
		イベントフラグ属性が不正	H'050D
		メッセージの優先度が最大メッセージ優先度(CFG_MAXMSGPRI)より大きい	H'050E
		データキュー領域の先頭アドレスが最終アドレスより大きい値	H'050F
		ミューテックスの上限優先度が最大タスク優先度(CFG_MAXTSKPRI)より大きい値	H'0510
		タイムティックの周期の分母が 100 より大きい値	H'0511
		タイムティックの周期の分母または分子のどちらかが 1 でない	H'0512
6	領域の配置不正	セクション名 P_hiknl の領域が H'xxxx0000～H'xxxxFFFF 番地の範囲に配置されていない（xxxx は同一）	H'0601
		セクション名 C_hisetup の領域が H'xxxx0000～H'xxxxFFFF 番地の範囲に配置されていない（xxxx は同一）	H'0602
		セクション名 B_hiwrk の領域が H'xxxx0000～H'xxxxFFFF 番地の範囲に配置されていない（xxxx は同一）	H'0603

11. 各種一覧

11.1 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	act_tsk	ER ercd = act_tsk(ID tskid);	タスクの起動
	iact_tsk	ER ercd = iact_tsk(ID tskid);	
2	can_act	ER_UINT actcnt = can_act(ID tskid);	タスク起動要求のキャンセル
3	sta_tsk	ER ercd = sta_tsk(ID tskid, VP_INT stacd);	タスクの起動(起動コード指定)
	ista_tsk	ER ercd = ista_tsk(ID tskid, VP_INT stacd);	
4	ext_tsk	void ext_tsk();	自タスクの終了
5	ter_tsk	ER ercd = ter_tsk(ID tskid);	タスクの強制終了
6	chg_pri	ER ercd = chg_pri(ID tskid, PRI tskpri);	タスク優先度の変更
7	get_pri	ER ercd = get_pri(ID tskid, PRI *p_tskpri);	タスク優先度の参照
8	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);	
9	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);	
タスク付属同期機能			
10	slp_tsk	ER ercd = slp_tsk();	起床待ち
11	tslp_tsk	ER ercd = tslp_tsk(TMO tmout);	同上(タイムアウト有)
12	wup_tsk	ER ercd = wup_tsk(ID tskid);	タスクの起床
	iwup_tsk	ER ercd = iwup_tsk(ID tskid);	
13	can_wup	ER_UINT wupcnt = can_wup(ID tskid);	タスク起床要求のキャンセル
14	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);	
15	sus_tsk	ER ercd = sus_tsk(ID tskid);	強制待ち状態への移行
16	rsm_tsk	ER ercd = rsm_tsk(ID tskid);	強制待ち状態からの再開
17	frsm_tsk	ER ercd = frsm_tsk(ID tskid);	強制待ち状態からの強制再開
18	dly_tsk	ER ercd = dly_tsk(RELTIM dlytim);	自タスクの遅延
同期･通信(セマフォ)機能			
19	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);	
20	wai_sem	ER ercd = wai_sem(ID semid);	セマフォ資源の獲得
21	pol_sem	ER ercd = pol_sem(ID semid);	同上(ポーリング)
	ipol_sem	ER ercd = ipol_sem(ID semid);	
22	twai_sem	ER ercd = twai_sem(ID semid, TMO tmout);	同上(タイムアウト有)
23	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);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
同期･通信(イベントフラグ)機能			
24	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);	
25	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);	
26	wai_flg	ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);	イベントフラグ待ち
27	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);	
28	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)
29	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			
30	snd_dtq	ER ercd = snd_dtq(ID dtqid, VP_INT data);	データキューへの送信
31	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);	
32	tsnd_dtq	ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);	同上(タイムアウト有)
33	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);	
34	rcv_dtq	ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data);	データキューからの受信
35	prcv_dtq	ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data);	同上(ポーリング)
36	trcv_dtq	ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);	同上(タイムアウト有)
37	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			
38	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);	
39	rcv_mbx	ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);	メールボックスからの受信
40	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);	
41	trcv_mbx	ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);	同上(タイムアウト有)
42	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			
43	loc_mtx	ER ercd = loc_mtx(ID mtxid);	ミューテックスのロック
44	ploc_mtx	ER ercd = ploc_mtx(ID mtxid);	同上(ポーリング)
45	tloc_mtx	ER ercd = tloc_mtx(ID mtxid, TMO tmout);	同上(タイムアウト有)
46	unl_mtx	ER ercd = unl_mtx(ID mtxid);	ミューテックスのロック解除
47	ref_mtx	ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx);	ミューテックスの状態参照

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
メモリプール管理(固定長メモリプール)機能			
48	get_mpf	ER ercd = get_mpf(ID mpfid, VP *p_blk);	固定長メモリブロックの獲得
49	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);	
50	tget_mpf	ER ercd = tget_mpf(ID mpfid, VP *p_blk, TMO tmout);	同上(タイムアウト有)
51	rel_mpf	ER ercd = rel_mpf(ID mpfid, VP blk);	固定長メモリブロックの返却
52	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			
53	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
54	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);	
55	tget_mpl	ER ercd =tget_mpl(ID mplid, UINT blksz, VP *p_blk, TMO tmout);	同上(タイムアウト有)
56	rel_mpl	ER ercd = rel_mpl(ID mplid, VP blk);	可変長メモリブロックの返却
57	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)			
58	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);	
59	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);	
時間管理機能(周期ハンドラ)			
60	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);	
61	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);	
62	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);	
システム状態管理機能			
63	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);	
64	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);	
65	loc_cpu	ER ercd = loc_cpu();	CPU ロック状態への移行
	iloc_cpu	ER ercd = iloc_cpu();	
66	unl_cpu	ER ercd = unl_cpu();	CPU ロック状態の解除
	iunl_cpu	ER ercd = iunl_cpu();	
67	dis_dsp	ER ercd = dis_dsp();	ディスパッチの禁止
68	ena_dsp	ER ercd = ena_dsp();	ディスパッチの許可
69	sns_ctx	BOOL state = sns_ctx();	コンテキストの参照
70	sns_loc	BOOL state = sns_loc();	CPU ロック状態の参照
71	sns_dsp	BOOL state = sns_dsp();	ディスパッチ禁止状態の参照
72	sns_dpn	BOOL state = sns_dpn();	ディスパッチ保留状態の参照
73	vsta_knl	void vsta_knl();	カーネルの起動
	ivsta_knl	void ivsta_knl();	
74	vsys_dwn	void vsys_dwn(H type,H inf1,B inf2,B inf3,H inf4,UW inf5);	システムダウン
	ivsys_dwn	void ivsys_dwn(H type,H inf1,B inf2,B inf3,H inf4,UW inf5);	

No	ｻｰﾋﾞｽｺｰﾙ	C 言語 API	機能説明
75	ivbgn_int	void ivbgn_int(UINT dintno);	割込ハンドラ開始のトレース取得
76	ivend_int	void ivend_int(UINT dintno);	割込ハンドラ終了のトレース取得
割込み管理機能			
77	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);	
78	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);	
システム構成管理機能			
79	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);	

11.2 サービスコールエラーコード一覧

表11-1 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'0000	(D'0)	[k]	正常終了
E_NOSPT	H'fff7	(-D'9)	[p]	未サポート機能（機能が未登録）
E_PAR	H'ffef	(-D'17)	[p]/[k]	パラメータエラー
E_ID	H'ffee	(-D'18)	[p]	不正 ID 番号
E_CTX	H'ffe7	(-D'25)	[p]/[k]	コンテキストエラー
E_ILUSE	H'ffe4	(-D'28)	[k]	サービスコール不正使用
E_OBJ	H'ffd7	(-D'41)	[k]	オブジェクト状態エラー
E_NOEXS	H'ffd6	(-D'42)	[p]	オブジェクトが存在しない
E_QOVR	H'ffd5	(-D'43)	[k]	キューイングオーバフロー
E_RLWAI	H'ffcf	(-D'49)	[k]	待ち状態の強制解除
E_TMOUT	H'ffce	(-D'50)	[k]	ポーリング失敗またはタイムアウト

【注】* [p]は、パラメータチェック機能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。

12. 付録

12.1 許可割込み要因の設定

HI1000/4 では、H8SX/1650 用の割込み要因の設定を行う定義ファイルを提供します。

割込み要因の設定定義ファイル名は、「1650_intdef.h」です。本定義ファイルを使用して、H8SX/1650 の割込み要因単位に割込みの許可、禁止が行えます。

割込み要因を設定する場合は、「1650_intdef.h」ファイルをインクルードしてください。

表12-1　割込み要因の設定一覧

項番	割込み要因の設定	機　能
1	dis_int	割込み要因の禁止
2	ena_int	割込み要因の許可

12.1.1　割込み要因の禁止(dis_int)

■C 言語 API

```
dis_int(<interrupt_id>);
```

■パラメータ

```
<interrupt_id>    割込み要因の名称、または値
```

■機能

<interrupt_id>で指定された割込み要因に対応する割込み許可ビットを無効にします。割込み要因の名称(<interrupt_id>)、値は、表 12-2を参照してください。

割込み要因の名称(<interrupt_id>)に、変数や他の#define 文の定義を指定することはできません。

12.1.2　割込み要因の許可(ena_int)

■C 言語 API

```
ena_int(<interrupt_id>);
```

■パラメータ

```
<interrupt_id>    割込み要因の名称、または値
```

■機能

<interrupt_id>で指定された割込み要因に対応する割込み許可ビットを有効にします。割込み要因の名称(<interrupt_id>)、値は、表 12-2を参照してください。

割込み要因の名称(<interrupt_id>)に変数、他の#define 文の定義を指定することはできません。

表12-2 割込み要因の名称、値の一覧

No.	割込み要因の名称	値	No.	割込み要因の名称	値
1	IRQ0_ID	0	36	TPU4_TCIEV_ID	35
2	IRQ1_ID	1	37	TPU4_TCIEU_ID	36
3	IRQ2_ID	2	38	TPU5_TGIEA_ID	37
4	IRQ3_ID	3	39	TPU5_TGIEB_ID	38
5	IRQ4_ID	4	40	TPU5_TCIEV_ID	39
6	IRQ5_ID	5	41	TPU5_TCIEU_ID	40
7	IRQ6_ID	6	42	TMR0_CMIEA_ID	41
8	IRQ7_ID	7	43	TMR0_CMIEB_ID	42
9	IRQ8_ID	8	44	TMR0_OVIE_ID	43
10	IRQ9_ID	9	45	TMR1_CMIEA_ID	44
11	IRQ10_ID	10	46	TMR1_CMIEB_ID	45
12	IRQ11_ID	11	47	TMR1_OVIE_ID	46
13	SWDTIE_ID*1	12	48	TMR2_CMIEA_ID	47
14	WDT_ID	13	49	TMR2_CMIEB_ID	48
15	ADIE_ID	14	50	TMR2_OVIE_ID	49
16	TPU0_TGIEA_ID	15	51	TMR3_CMIEA_ID	50
17	TPU0_TGIEB_ID	16	52	TMR3_CMIEB_ID	51
18	TPU0_TGIEC_ID	17	53	TMR3_OVIE_ID	52
19	TPU0_TGIED_ID	18	54	SCI0_TEIE_ID	53
20	TPU0_TCIEV_ID	19	55	SCI0_MPIE_ID	54
21	TPU1_TGIEA_ID	20	56	SCI0_RIE_ID	55
22	TPU1_TGIEB_ID	21	57	SCI0_TIE_ID	56
23	TPU1_TCIEV_ID	22	58	SCI1_TEIE_ID	57
24	TPU1_TCIEU_ID	23	59	SCI1_MPIE_ID	58
25	TPU2_TGIEA_ID	24	60	SCI1_RIE_ID	59
26	TPU2_TGIEB_ID	25	61	SCI1_TIE_ID	60
27	TPU2_TCIEV_ID	26	62	SCI2_TEIE_ID	61
28	TPU2_TCIEU_ID	27	63	SCI2_MPIE_ID	62
29	TPU3_TGIEA_ID	28	64	SCI2_RIE_ID	63
30	TPU3_TGIEB_ID	29	65	SCI2_TIE_ID	64
31	TPU3_TGIEC_ID	30	66	SCI4_TEIE_ID	65
32	TPU3_TGIED_ID	31	67	SCI4_MPIE_ID	66
33	TPU3_TCIEV_ID	32	68	SCI4_RIE_ID	67
34	TPU4_TGIEA_ID	33	69	SCI4_TIE_ID	68
35	TPU4_TGIEB_ID	34			

【注】*1 **SWDTIE_ID は提供している H8SX/1650 のサンプルには存在しません。**

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
HI1000/4 V.1.04（H8SX,H8Sファミリ用リアルタイムOS）

発行年月日　2004 年 1 月 14 日　Rev.1.00
　　2006 年 6 月 28 日　Rev.2.00
発　　行　株式会社ルネサス テクノロジ 営業統括部
　　〒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

株式会社ルネサス テクノロジ　営業統括部　〒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

HI1000/4 V.1.04
ユーザーズマニュアル
（H8SX,H8S ファミリ用リアルタイム OS）

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

RJJ10B0108-0200