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

RENESAS

HI7300/PX V.1.01

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

R0R50730PRW01J

誤記に関するお詫び：
本資料 P.322 「7.4.1 キャッシュの初期化(shx2_vini_cac)」の「■機能」に誤記があり、訂正いたしました。

ルネサス エレクトロニクス
www.renesas.com

Rev.3.00 2006.07

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 仕様およびμITRON4.0 仕様 保護機能拡張は、(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 仕様およびμITRON4.0 仕様 保護機能拡張の仕様書は、(社)トロン協会ホームページ(http://www.assoc.tron.org/)から入手が可能です。
- μITRON 仕様の著作権は(社)トロン協会に属しています。
- Microsoft® Windows® 98 operating system, Microsoft® Windows® Millennium Edition(Windows® Me) operating system, Microsoft® Windows NT® operating system, Microsoft® Windows® 2000 operating system, Microsoft® Windows® XP operating system は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- SuperH™ は、(株)ルネサステクノロジの商標です。
- その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

はじめに

本マニュアルでは、HI7300/PX(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。**
shnnnn	**サンプルファイルは、例えば** SH73180 **用は**"sh73180"**ディレクトリというように、マイコン名称のディレクトリに格納しています。これらを総称して"*shnnnn*"と記載しています。**
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、「**10.6 CFG **名」を参照してください。**
samples¥	'¥'**はディレクトリ区切り記号です。** '¥'**で終わる文字列は、ディレクトリ名を示します。基本的には、カーネルのインストールディレクトリを起点とした相対パス形式で表記していますが、表記が長くなる場合や文脈から冗長と思われる部分は、そのサブディレクトリを基点とした表記としている箇所もあります。**

■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

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」**
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. 論理アドレス空間の扱い」**
論理アドレスの扱いについて解説しています。

● **「6. 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。

● **「7. キャッシュサポート関数」**
キャッシュサポート関数の仕様について解説しています。

● **「8. アプリケーション作成手順」**
タスクやハンドラの作成方法について解説しています。

● **「9. 標準タイマドライバ」**
標準タイマドライバの作成方法について解説しています。

● **「10. コンフィギュレータ」**
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。

● **「11. ビルド」**
各サンプルプログラムの解説と、それ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。

● **「12. スタック使用サイズの算出」**
スタックサイズの算出方法について解説しています。

● **「13. リソースプールサイズの見積り」**
リソースプールに必要なサイズの見積もり方法について解説しています。

● **「14. システムプールサイズの見積り」**
システムプールに必要なサイズの見積もり方法について解説しています。

● **「15. FPU に関する注意事項」**
FPU に関する注意事項を解説しています。

● **「16. システムダウン等の対処」**
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。

● 「17. 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。

2. 概要

2.1 特長

2.1.1 メモリオブジェクト保護機能

(1) デバッグ効率の向上

メモリプロテクションの無いシステムでは、一般にはポインタ不正などでメモリ内容が破壊されても、実際に不具合現象となってはじめて気付きます。つまり、バグの原因を特定するには、エミュレータのトレース機能などを元に解析するしかなく、それには多くの時間を費やす必要がありました。メモリプロテクション機能をサポートした本製品では、不正メモリアクセスの時点でそれが検出されるので、デバッグ効率が飛躍的に高まります。

(2) 高信頼性システムの実現

万一、機器出荷後にシステム基幹部分が破壊されようとしても、それはプロテクトされます。その結果、機器の動作を継続することができます。

(3) 低オーバヘッド

メモリオブジェクト保護機能は、マイコン内蔵の MMU(Memory Management Unit)を利用することで実現しています。一般に、MMU 使用時には TLB ミスオーバヘッドが発生しますが、本製品ではこの時間を約 1 マイクロ秒(注)という高性能を達成しています。さらには、指定したメモリ領域について TLB ミスを回避する vloc_tlb サービスコールもサポートしています。

【注】メモリ容量が 64M バイトの場合の、400MHz 動作・キャッシュヒット時の平均値

2.1.2 業界標準のμITRON 仕様に準拠

本製品は、業界標準のμITRON4.0 仕様に準拠しています。ランデブ以外のほぼ全てのサービスコールをサポートしています。

また、メモリオブジェクト保護機能は、μITRON4.0 保護機能拡張に準拠しています。なお、μITRON4.0 保護機能拡張のメモリオブジェクト以外のカーネルオブジェクトの保護機能はサポートしていません。

2.1.3 DSP, FPU のサポート

マルチタスクで DSP, FPU を使用することができます。

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

以下のようなサンプルを提供しています。

- 各種サービスコールの使用例を兼ねたユーザドメイン
- カーネルオブジェクトの状態などを参照できる簡易モニタ(シミュレータ専用)
- システムダウンルーチン
- メモリアクセス違反ハンドラ
- CPU 例外ハンドラ
- 各種マイコン用のリセットプログラム、標準タイマドライバ
- 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

項目	内容
対象マイコン	CPU **コアとして、**SH4AL-DSP, SH-4A **を搭載したマイコン**
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
コンパイラ	**弊社製** C/C++ Comipler Package for SuperH RISC engine V.9.00Release 03 **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。

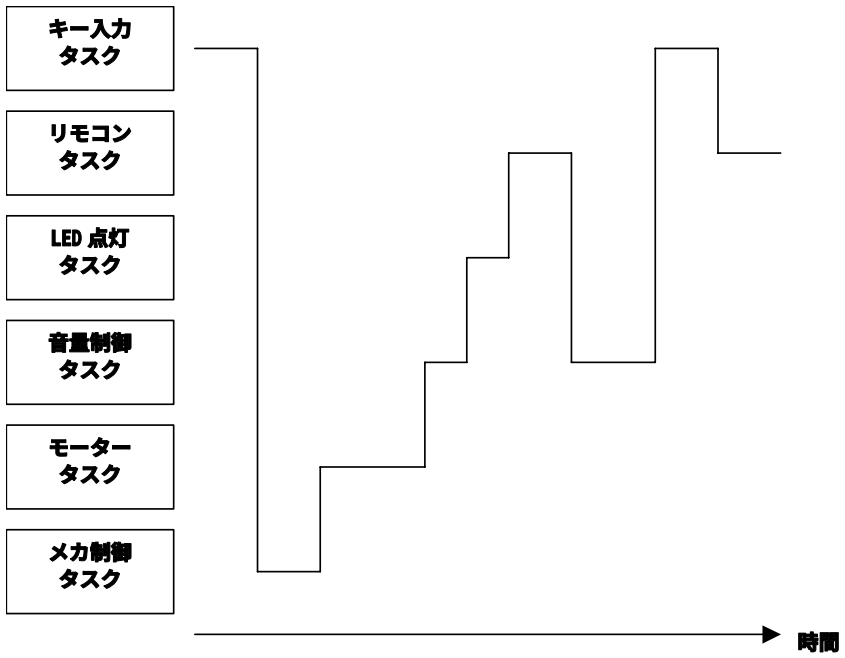


図3.1　タスクの時分割動作

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)。

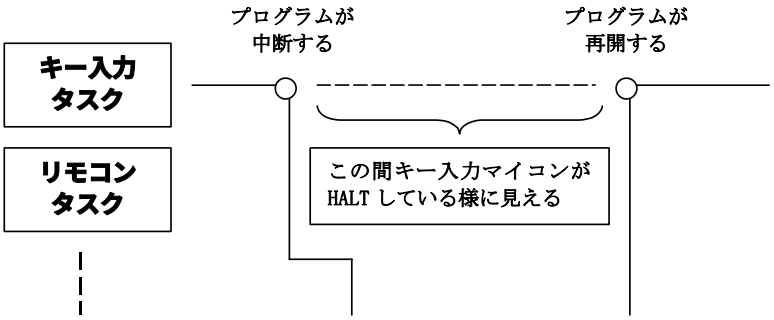


図3.2　タスクの中断と再開

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)。

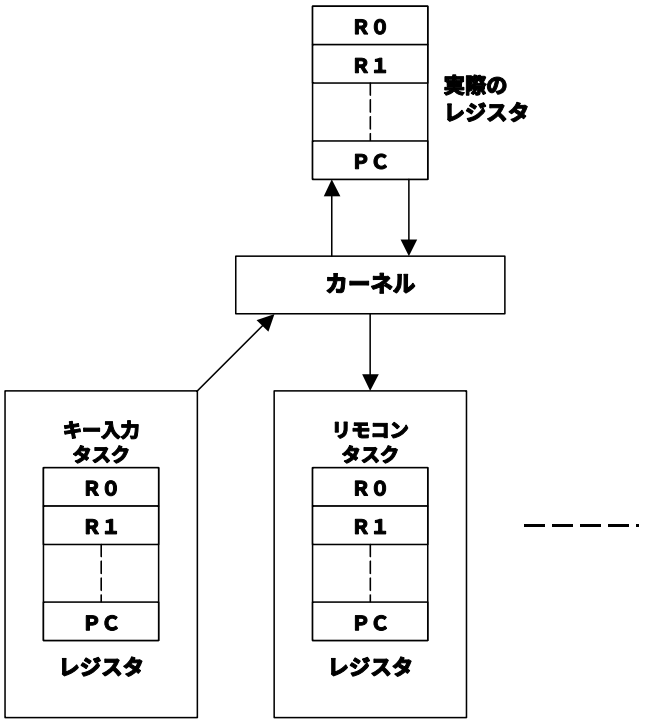


図3.3　タスクの切り替え

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 言語関数として呼び出します。

```
act_tsk(1);
```

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_sys.h)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 */
```

また、acre_tsk など"acre"で始まるサービスコールを使用すれば、カーネルが ID 番号を割り当て、割り当てられた ID 番号が返ります。

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。

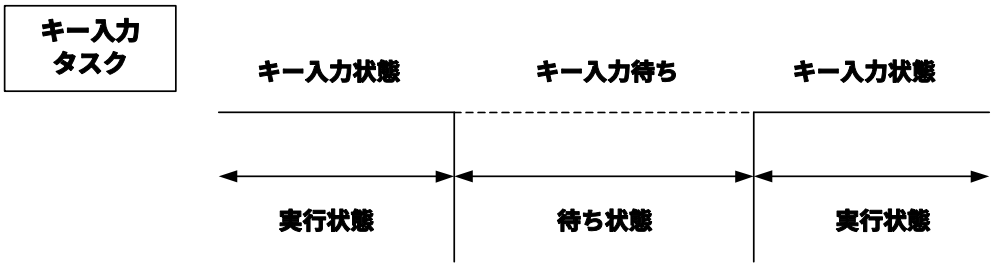


図3.5 タスクの状態

タスクは、図 3.6に示す 7 つの状態を遷移します。

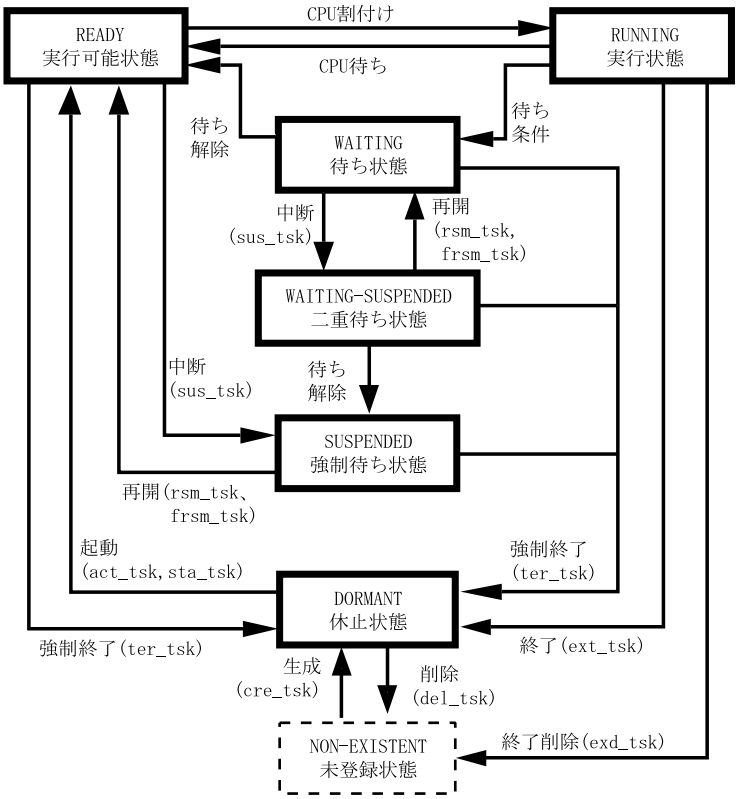


図3.6　タスクの状態遷移図

(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。

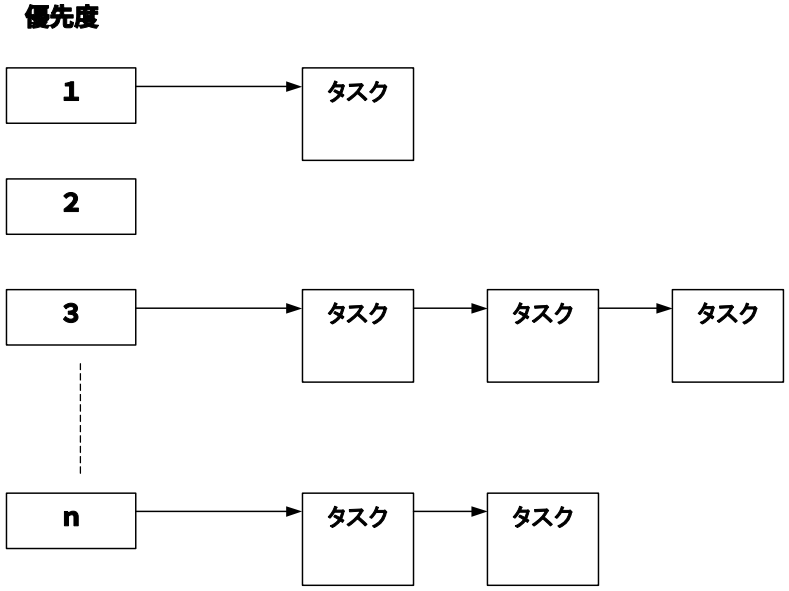


図3.7　レディーキュー(実行待ち状態)

4. カーネルの機能

4.1 アプリケーションの種類

ユーザが作成するアプリケーションには、以下のものがあります。

(1) タスク

マルチタスクの制御対象となる単位です。

(2) タスク例外処理ルーチン

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

(3) 割込みハンドラ

割込みが発生したときに実行されます。

(4) CPU 例外ハンドラ

CPU 例外が発生したときに実行されます。

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

指定した周期や時刻になったときに実行されます。

(6) 拡張サービスコールルーチン

ユーザが作成したプログラムを拡張サービスコールルーチンとしてカーネルに登録できます。登録された拡張サービスコールルーチンは、拡張サービスコール(cal_svc)によって呼び出されます。

(7) トラップルーチン

ユーザが作成したプログラムをトラップルーチンとしてカーネルに登録できます。登録されたトラップルーチンは、TRAPA 命令の実行によって呼び出されます。

(8) 初期化ルーチン

カーネル起動時に一度だけ実行されるプログラムです。

この他に、特殊なアプリケーションとして、以下のものがあります。これらはいずれも、本カーネル仕様としてアプリケーションのシンボル名が固定で決められています。

(9) システムダウンルーチン

システムダウンの場合に呼び出されるプログラムです。

(10)割込み・CPU 例外フックルーチン

割込みおよび CPU 例外が発生した場合、通常はカーネルがこれを受け、適切なプログラム(割込みハンドラ、CPU 例外ハンドラ、トラップルーチン、カーネルのサービスコール処理)を呼び出しますが、本ルーチンはそれらを呼び出す前にフックされて呼び出されます。デバッグ用途のプログラムです。

(11)メモリアクセス違反ハンドラ(メモリオブジェクト保護機能使用時のみ)

MMU 対象領域への不正アクセスが行われた時に呼び出されるプログラムです。

(12)標準タイマドライバ

標準タイマドライバは、タイマ初期化ルーチンとタイマ割込みルーチンで構成されます。最適化タイマドライバを使用する場合は不要です。

4.2 システムの状態

4.2.1 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。この区別によって利用できるサービスコールが異なります。

タスクとタスク例外処理ルーチン、およびそれらから呼び出された拡張サービスコールルーチンとトラップルーチンは、タスクコンテキストで実行します。その他のアプリケーションおよびカーネルは、非タスクコンテキストで実行します。

非タスクコンテキストは、タスクコンテキストよりも高い優先順位を持ちます。タスクコンテキストの処理は、全ての非タスクコンテキストの処理が終了した後に実行されます。例えば、タスク実行中に割込みが発生すると、その時点で割込みハンドラが起動され、タスクの実行は一時中断されます。

なお、非タスクコンテキストかどうかは、sns_ctx で確認することができます。

4.2.2 ディスパッチ禁止状態、CPU ロック状態、ディスパッチ保留状態

システムは、ディスパッチ保留状態かそうでないか、いずれかの状態をとります。

ディスパッチ保留状態では、現在実行中のタスクよりも優先度の高いタスクがあっても、タスクスイッチが発生しません。また、タスクコンテキストの場合は待ち状態に遷移するサービスコールを呼び出すと、E_CTX エラーが返ります。

カーネルは、タスクの状態やレディキューなどのすべての管理情報をディスパッチ保留状態に関係なく更新しますが、タスクスイッチ処理だけを行わなくなります。

以下のいずれかを満たす場合がディスパッチ保留状態です。ディスパッチ保留状態かどうかは、sns_dpn で確認することができます。

- 非タスクコンテキスト実行中
- ディスパッチ禁止状態
- CPU ロック状態
- 実行中のタスクが chg_ims によって SR.IMASK を 0 以外に変更している

(1) ディスパッチ禁止状態

dis_dsp によってディスパッチ禁止状態にすることができます。ディスパッチ禁止状態は ena_dsp によって解除されます。

ディスパッチ禁止状態かどうかは、sns_dsp で確認することができます。

(2) CPU ロック状態

loc_cpu, iloc_cpu によって、CPU ロック状態にすることができます。CPU ロック状態では、CFG_KNLLVL 以下のレベルの割込みがマスクされます。CPU ロック状態は unl_cpu, iunl_cpu によって解除されます。

CPU ロック状態では、利用可能なサービスコールが制限されます。

CPU ロック状態かどうかは、sns_loc で確認することができます。

■関連ページ　CPU ロック状態で利用できるサービスコール→「6.4.3 CPU ロック状態」

4.3 保護ドメイン

全てのアプリケーションは、いずれかの保護ドメインに所属します。保護ドメインには、ユーザドメインとカーネルドメインの 2 種類があります。

ユーザドメインは 1～31 のドメイン ID によって識別されます。カーネルドメインは TDOM_KERNEL(-1)のドメイン ID を持ち、システムにひとつだけ存在します。

カーネルドメインに所属するプログラムは、CPU の特権モード(SR.MD=1)で実行します。一方、ユーザドメインに属するプログラムは、CPU のユーザモード(SR.MD=0)で実行します。

ユーザモードでは、以下の制限があります。

(1) CPUの特権命令を実行できません。実行しようとした場合はCPU例外が発生します。
(2) アクセスできるメモリ範囲に制限があります。制限の内容は、メモリオブジェクト保護機能の有無によって異なります。

なお、メモリ保護機能を使用しない場合は、カーネルドメインとユーザドメインの種類の区別のみ意味を持ち、ドメイン ID による区別は意味を持ちません。

各アプリケーションが所属するドメインを、表 4.1に示します。

表4.1 アプリケーションの所属ドメイン

アプリケーション	所属ドメイン	所属ドメインの指定方法
タスク	任意のドメインに所属可能	タスク生成時に指定
タスク例外処理ルーチン	対象タスクと同じドメインに所属	－
割込みハンドラ	カーネルドメイン	－
CPU 例外ハンドラ	カーネルドメイン	－
タイムイベントハンドラ	カーネルドメイン	－
拡張サービスコールルーチン	カーネルドメイン	－
トラップルーチン	カーネルドメイン	－
初期化ルーチン	カーネルドメイン	－
システムダウンルーチン	カーネルドメイン	－
割込みフックルーチン	カーネルドメイン	－
メモリアクセス違反ハンドラ	カーネルドメイン	－
標準タイマドライバ	カーネルドメイン	－

4.4 タスク管理機能

4.4.1 タスクの生成

タスクを生成(未登録状態→休止状態)するには、以下の方法があります。

- cre_tsk, icre_tsk サービスコール
  指定されたID番号でタスクを生成します。
- acre_tsk, iacre_tsk サービスコール
  カーネルが自動的にID番号を割り当てて、タスクを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

4.4.2 タスクの所属ドメイン

タスクは、いずれかのドメインに所属します。どのドメインに所属するかは、生成時に指定します。ユーザドメインに所属するタスクは CPU のユーザモード(SR.MD=0)、カーネルドメインに所属するタスクは特権モード(SR.MD=1)で実行します。

4.4.3 タスクの起動

タスクを起動(休止状態→実行可能状態)するには、以下の方法があります。

- act_tsk, iact_tsk サービスコール
  既にタスクが起動済みの場合に、その要求がキューイングされます。
- sta_tsk, ista_tsk サービスコール
  既にタスクが起動済みの場合はエラーになりますが、その代わりに起動されるタスクに渡すパラメータを指定することができます。
- タスク生成時に TA_ACT 属性を指定
  タスクが生成されると同時に起動されます。

また、以下のサービスコールがあります。

- can_act, ican_act サービスコール
  起動要求のキューイングをキャンセルします。

4.4.4 タスクの終了、削除

- ext_tsk サービスコール
  自タスクを終了し、休止状態とします。
- exd_tsk サービスコール
  自タスクを終了・削除し、未登録状態とします。
- ter_tsk サービスコール
  休止・未登録状態でない他タスクを強制的に終了させ、休止状態とします。
- del_tsk サービスコール
  休止状態の他タスクを削除し、未登録状態とします。

4.4.5 優先度の変更

chg_pri, ichg_pri を用いて、タスクの優先度を変更することができます。優先度の変更によって、前述のレディキューや、「タスク優先度順(TA_TPRI)の待ち行列」の順序も変更されます。

ただし、一般には優先度の変更はシステム全体の振る舞いに影響を与えるため、本サービスコールを使用しないことが推奨されます。

なお、タスクの優先度には「ベース優先度」と「現在優先度」の２つがあります。通常は、この２つは同じですが、ミューテックスをロックしている間だけ異なります。詳細は、以下を参照してください。

■関連ページ　ベース優先度と現在優先度の違い→「4.12 ミューテックス」

4.4.6 タスク実行モード

タスク実行モードは、μITRON 仕様外の本製品独自機能です。

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

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

4.4.7 タスクの状態参照

タスクの状態を参照するサービスコールとして、ref_tsk, iref_tsk があります。このサービスコールでは、現在のタスクの状態や待ち要因などの詳細な情報を得ることができます。

また、このサービスコールの簡易版として ref_tst, iref_tst があります。

4.5 スタック管理方式

スタックは、非タスクコンテキストスタックと各タスクのスタックの２つに分類されます。

■関連ページ　「12 スタック使用サイズの算出」

4.5.1 非タスクコンテキストスタック

非タスクコンテキストが実行するときに使用するスタックで、システムでひとつだけ存在します。サイズは、コンフィギュレータの CFG_NTSKSTKSZ で指定します。

カーネルは、タスクコンテキストから非タスクコンテキストに遷移するときに、使用するスタックを非タスクコンテキストスタックに切り替えます。

4.5.2 タスクのスタック

(1) ユーザドメインに所属するタスク

ユーザドメインに所属するタスクは、タスクが実行するために使用するスタック以外に「システムスタック」を持ちます。システムスタックは、タスクから呼び出された拡張サービスコールルーチンとトラップルーチン、およびカーネルがタスクのコンテキストを保存するために使用するスタックです。システムスタックは、ユーザモードからはアクセスできません。

いずれのスタックも、タスク生成時にそのアドレスを指定するか、またはカーネルが割り付けるように指定することができます。ただし、コンフィギュレータで生成する場合は、スタックアドレスを指定することはできません。

カーネルが割り付ける場合は、スタックはシステムプールから、システムスタックはリソースプールから割り付けられます。

メモリオブジェクト保護機能使用時は、カーネルがシステムプールから割り付けたスタックは、そのタスクのみがリード・ライト可能なひとつのメモリオブジェクトとなります。

(2) カーネルドメインに所属するタスク

カーネルドメインに所属するタスクは、ひとつのスタックを持ちます。このスタックは、ユーザモードからはアクセスできません。

スタックは、タスク生成時にそのアドレスを指定するか、またはカーネルが割り付けるように指定することができます。ただし、コンフィギュレータで生成する場合は、スタックアドレスを指定することはできません。

カーネルが割り付ける場合は、スタックはリソースプールから割り付けられます。

■関連ページ　「6.7.1 タスクの生成(cre_tsk, icre_tsk, acre_tsk, iacre_tsk)」

4.6 タスク付属同期機能

タスク付属同期機能とは、タスクに付随する機能を用いて、タスク間の同期をとるための機能です。

4.6.1 タスク起床による同期処理

slp_tsk, tslp_tsk は、wup_tsk, iwup_tsk で起床されるまで待ち状態に移行します。なお、tslp_tsk では、起床されるまでのタイムアウトを指定することができます。

この関係を用いれば、図 4.1のように同期をとることができます。これは、最も単純な同期方法です。

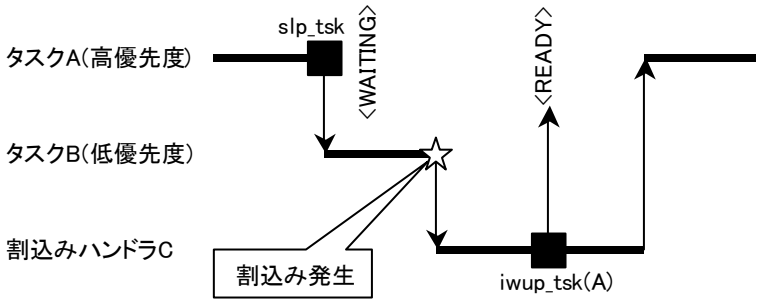


図4.1 タスク起床による同期例

また、slp_tsk, tslp_tsk による待ち状態でないときに行われた起床要求はキューイングされます。起床要求は、can_wup, ican_wup でキャンセルすることができます。

4.6.2 待ち状態の強制解除

rel_wai, irel_wai を用いることで、タスクの待ち状態(WAITING 状態)を強制的に解除することができます。ただし、このサービスコールでは強制待ち状態(SUSPENDED 状態)を解除することはできません。

4.6.3 強制待ち

sus_tsk, isus_tsk を用いることで、他のタスクを強制的に中断(強制待ち状態)させることができます。強制待ち要求はネストして記憶されます。rsm_tsk, irsm_tsk で、この強制待ち要求ネスト数が減算され、ネスト数が 0 になった時に強制待ち状態が解除されます。また、frsm_tsk, ifrsm_tsk では、強制待ち要求ネスト数に関わらず、直ちに強制待ち状態が解除されます。

既に待ち状態になっているタスクに対して sus_tsk, isus_tsk を行った場合は、そのタスクは 2 重待ち状態(WAITING-SUSPENDED)となります。

一般に、強制待ちの機能は主としてデバッグのために用いられます。アプリケーションでは使用しないことを推奨します。

4.6.4 タスク付属イベントフラグ

タスク付属イベントフラグは、μITRON 仕様外の本製品独自機能です。

各タスクは、事象に対応した 32 ビットのタスク付属イベントフラグを持っており、指定したビットがセットされるのを待つ(vwai_tfl, vtwai_tfl)またはポーリング(vpol_tfl)することができます。vtwai_tfl では、タイムアウト時間を指定することができます。

これらのサービスコールでイベントを取得した時には、タスク付属イベントフラグは 0 にクリアされると共に、これらのサービスコールのリターンパラメータにクリア直前のタスク付属イベントフラグの値(取得したイベント)が返ります。

タスクにイベントを通知するには、vset_tfl, ivset_tfl を使用します。このサービスコールは、対象タスクのタスク付属イベントフラグの指定したビットをセットします。

また、vclr_tfl, ivclr_tfl は対象タスクのタスク付属イベントフラグの指定したビットをクリアします。

図 4.2に、タスク付属イベントフラグの動作例を示します。

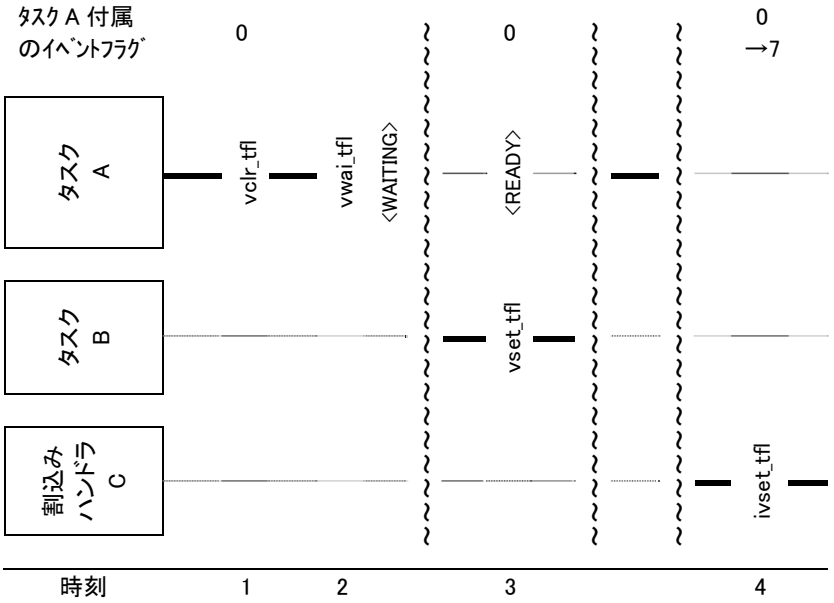


図4.2 タスク付属イベントフラグの動作例

図の解説

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

4.7 タスク例外処理機能

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

タスク例外は、タスク例外処理ルーチンによって処理されます。タスク例外処理ルーチンは、以下の方法で定義できます。

- def_tex, idef_tex サービスコール
- コンフィギュレータで定義

また、以下のサービスコールがあります。

- ras_tex, iras_tex サービスコール：タスク例外処理を要求する
- ena_tex サービスコール：自タスクのタスク例外処理を許可する
- dis_tex サービスコール：自タスクのタスク例外処理を禁止する
- sns_tex サービスコール：自タスクがタスク例外処理禁止状態かどうかを調べる
- ref_tex, iref_tex サービスコール：タスク例外処理の状態を参照する

図 4.3にタスク例外処理の動作例を示します。

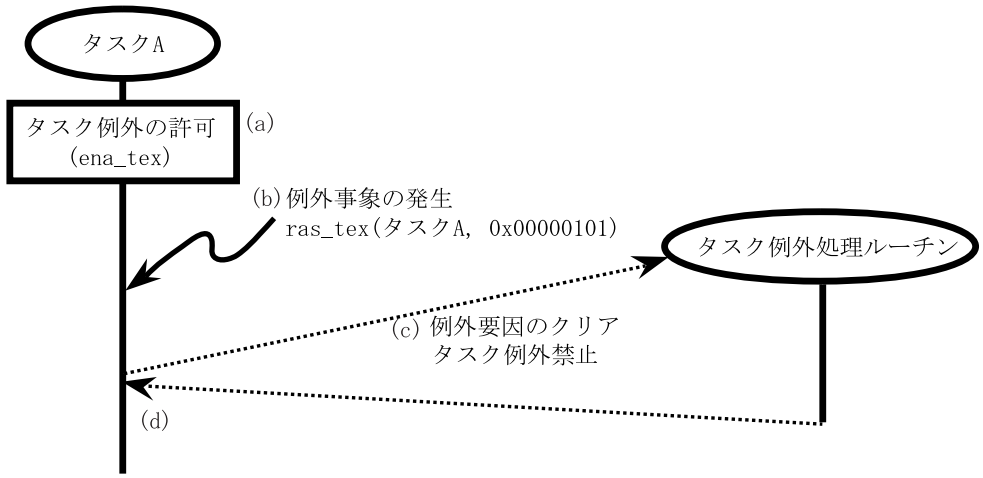


図4.3 タスク例外処理の動作例

図の解説(アルファベットは、その順に実行することを意味します)

(a) タスクAが、ena_texサービスコールでタスク例外を許可します。

(b) タスクA実行中に、ras_texサービスコールによってタスクAに例外事象(例外要因(0x00000101))が要求されました。

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

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

4.8 セマフォ

セマフォは複数のタスクで共有する装置などの資源の競合を防ぐために使用するオブジェクトです。

セマフォは内部にセマフォカウンタと呼ばれる計数値を持っており、そのセマフォカウンタに基づきセマフォの獲得・解放をおこなうことによって資源の競合を防ぎます。

セマフォの初期値を資源の数と同じとし、タスクは資源を使用する前にセマフォを獲得し、資源の使用が終わった時にセマフォを返却する、というようにプログラミングします。

セマフォを生成するには、以下の方法があります。

- cre_sem, icre_sem サービスコール
  指定されたID番号でセマフォを生成します。
- acre_sem, iacre_sem サービスコール
  カーネルが自動的にID番号を割り当てて、セマフォを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、セマフォを操作するサービスコールとして以下があります。

- del_sem サービスコール
  指定されたIDのセマフォを削除します。
- wai_sem, twai_sem サービスコール
  セマフォを獲得します。獲得できない場合(セマフォカウントが0)は待ち状態になります。
  twai_semでは、タイムアウト時間を指定することができます。
- pol_sem, ipol_sem サービスコール
  セマフォを獲得します。獲得できない場合(セマフォカウントが0)はエラーが返ります。
- sig_sem, isig_sem サービスコール
  セマフォを返却します。
- ref_sem, iref_sem サービスコール
  セマフォの状態を参照します。

図 4.4にセマフォの動作例を示します。

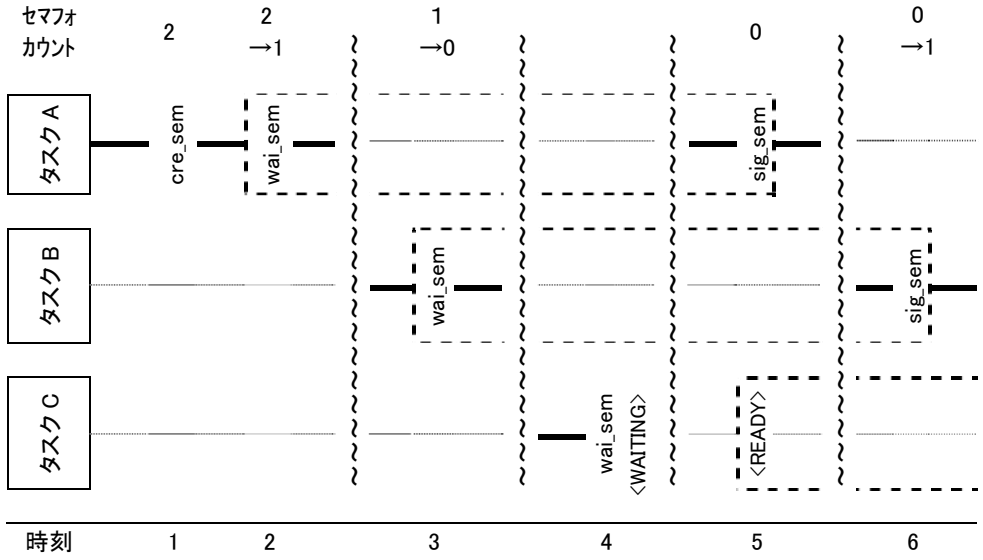


図4.4　セマフォの動作例

図の解説

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。

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

■優先度逆転問題

セマフォを使用した場合は、優先度逆転問題が発生することがあります。

優先度逆転問題とは、図 4.5に示すように優先度の高いタスク(A)がセマフォの獲得要求を行った時に、セマフォを獲得するまでの時間が、セマフォを獲得済みのタスク(C)とは関係のない別のタスク(B)の実行時間に依存してしまう、という問題です。この問題を回避するには、セマフォではなく、ミューテックスを使用してください。

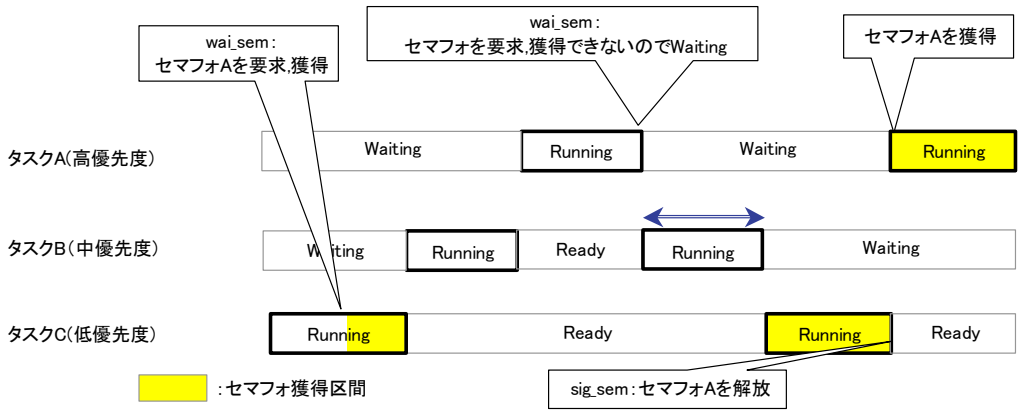


図4.5　優先度逆転問題

低優先度のタスク C がセマフォを解放するまでは、高優先度のタスク A はセマフォを獲得できません。しかし、中優先度のタスク B が実行しているために、セマフォを獲得している低優先度のタスク C が実行できず、それだけタスク C のセマフォ解放が遅れてしまいます(⟷ の区間)。つまり、セマフォを要求していない中優先度のタスク B によって、高優先度のタスク A の実行が待たされてしまいます。

4.9 イベントフラグ

イベントフラグは、事象に対応した 32 ビットのビットパターンを持つオブジェクトです。

タスクは、イベントフラグの指定したビットのいずれかまたは全てがセットされるのを待つことができます。また、1 つのイベントフラグに対して、複数のタスクが事象の発生を待つことを許すかどうかを、生成時に指定することができます。

イベントフラグを生成するには、以下の方法があります。

- cre_flg, icre_flg サービスコール
  指定されたID番号でイベントフラグを生成します。
- acre_flg, iacre_flg サービスコール
  カーネルが自動的にID番号を割り当てて、イベントフラグを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、イベントフラグを操作するサービスコールとして以下があります。

- del_flg サービスコール
  指定されたIDのイベントフラグを削除します。
- wai_flg, twai_flg サービスコール
  イベントフラグの指定したビットがセットされているかを検査します。セットされていなければ、セットされるまで待ち状態になります。twai_flgでは、タイムアウト時間を指定することができます。
- pol_flg, ipol_flg サービスコール
  イベントフラグの指定したビットがセットされているかを検査します。セットされていなければ、エラーが返ります。
- set_flg, iset_flg サービスコール
  イベントフラグの指定したビットをセットします。
- clr_flg, iclr_flg サービスコール
  イベントフラグの指定したビットをクリアします。
- ref_flg, iref_flg サービスコール
  イベントフラグの状態を参照します。

図 4.6にイベントフラグの動作例を示します。

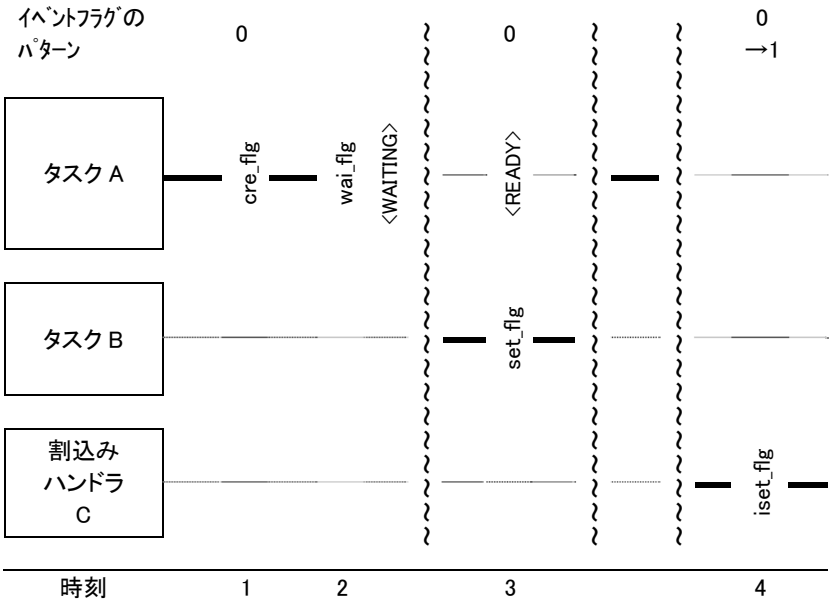


図4.6　イベントフラグの動作例

図の解説

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

4.10 データキュー

データキューは、1 ワード(4 バイト)データの受渡しを行うオブジェクトです。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。

データキューの領域は、リソースプールから割り付けられます。

データキューを生成するには、以下の方法があります。

- cre_dtq, icre_dtq サービスコール
  指定されたID番号でデータキューを生成します。
- acre_dtq, iacre_dtq サービスコール
  カーネルが自動的にID番号を割り当てて、データキューを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、データキューを操作するサービスコールとして以下があります。

- del_dtq サービスコール
  指定されたIDのデータキューを削除します。
- snd_dtq, tsnd_dtq サービスコール
  データを送信します。送信できない場合(データキューにデータが満杯)は、待ち状態になります。tsnd_dtqでは、タイムアウト時間を指定することができます。
- psnd_dtq, ipsnd_dtq サービスコール
  データを送信します。送信できない場合(データキューにデータが満杯)は、エラーが返ります。
- fsnd_dtq, ifsnd_dtq サービスコール
  データを送信します。送信できない場合(データキューにデータが満杯)は、最古のデータを削除して送信します。
- rcv_dtq, trcv_dtq サービスコール
  データを受信します。受信できない場合(データキューにデータが無い)は待ち状態になります。trcv_dtqでは、タイムアウト時間を指定することができます。
- prcv_dtq サービスコール
  データを受信します。受信できない場合(データキューにデータが無い)はエラーが返ります。
- ref_dtq, iref_dtq サービスコール
  データキューの状態を参照します。

図 4.7にデータキューの動作例を示します。

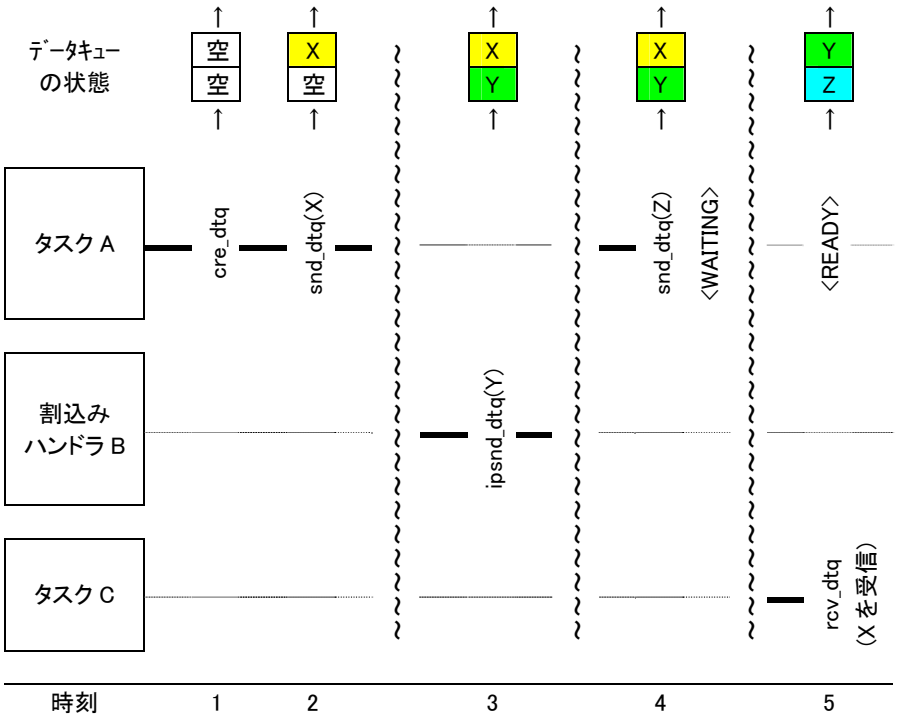


図4.7　データキューの動作例

図の解説

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

4.11 メールボックス

メールボックスは、メッセージの受け渡しを行うオブジェクトです。

メールボックスでは、メッセージの先頭アドレスのみを受け渡すため、メッセージサイズに依存せずに高速に行われます。

メールボックスを生成するには、以下の方法があります。

- cre_mbx, icre_mbx サービスコール
  指定されたID番号でメールボックスを生成します。
- acre_mbx, iacre_mbx サービスコール
  カーネルが自動的にID番号を割り当てて、メールボックスを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、メールボックスを操作するサービスコールとして以下があります。

- del_mbx サービスコール
  指定されたIDのメールボックスを削除します。
- snd_mbx, isnd_mbx サービスコール
  メールボックスにメッセージを送信します。
- rcv_mbx, trcv_mbx サービスコール
  メールボックスからメッセージを受信します。受信できない場合(メールボックスにメッセージが無い場合)は待ち状態になります。trcv_mbxでは、タイムアウト時間を指定することができます。
- prcv_mbx, iprcv_mbx サービスコール
  メールボックスからメッセージを受信します。受信できない場合(メールボックスにメッセージが無い)はエラーが返ります。
- ref_mbx, iref_mbx サービスコール
  メールボックスの状態を参照します。

図 4.8にメールボックスの動作例を示します。

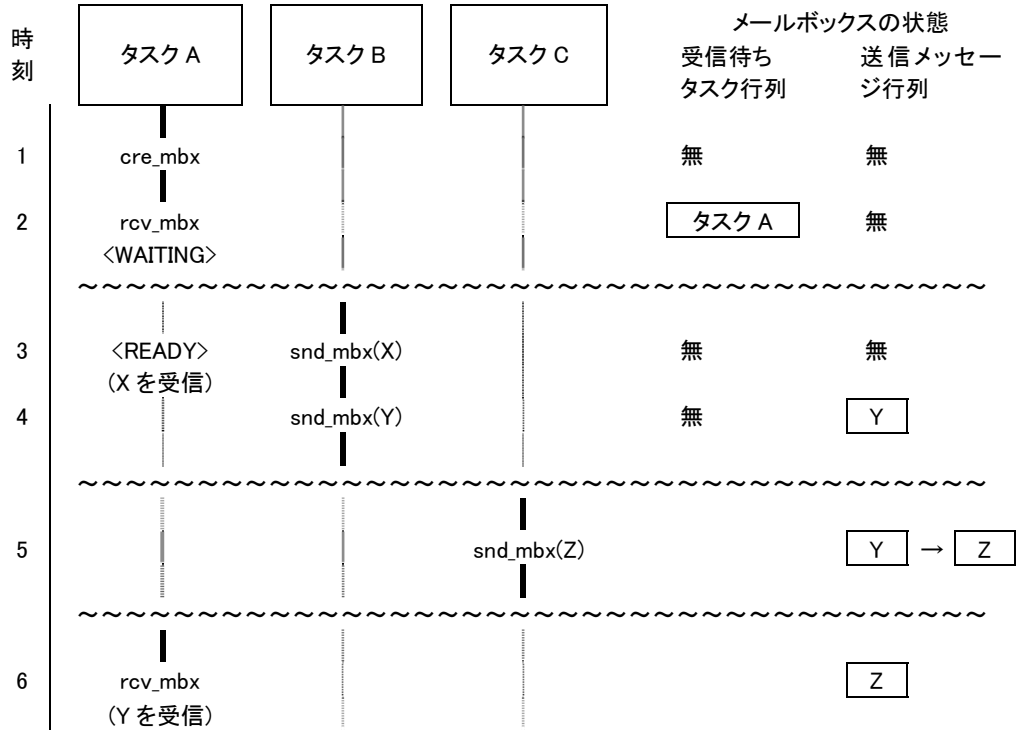


図4.8　メールボックスの動作例

図の解説

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

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

4.12 ミューテックス

ミューテックスは排他制御を行うためのオブジェクトです。セマフォとの主な相違は以下の通りです。

- 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るため、いわゆる「優先度逆転現象」が発生しません。
- 単一資源の排他制御にのみ使用できます。

タスクは資源を使用する前に loc_mtx, tloc_mtx, ploc_mtx を用いてミューテックスをロックし、資源の使用が終わった時に unl_mtx でミューテックスをアンロックするよう、プログラミングします。

ミューテックスをロックすると、そのタスクの優先度はそのミューテックスの上限優先度まで引き上げられ、アンロックすると元の優先度に戻ります。

ミューテックスを生成するには、以下の方法があります。

- cre_mtx サービスコール
  指定されたID番号でミューテックスを生成します。
- acre_mtx サービスコール
  カーネルが自動的にID番号を割り当てて、ミューテックスを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、ミューテックスを操作するサービスコールとして以下があります。

- del_mtx サービスコール
  指定されたミューテックスを削除します。
- loc_mtx, tloc_mtx サービスコール
  ミューテックスをロックします。ロックできない場合(他のタスクがロックしている)は待ち状態になります。tloc_mtxでは、タイムアウト時間を指定することができます。
- ploc_mtx サービスコール
  ミューテックスをロックします。ロックできない場合(他のタスクがロックしている)はエラーが返ります。
- unl_mtx サービスコール
  ミューテックスをアンロックします。
- ref_mtx サービスコール
  ミューテックスの状態を参照します。

図 4.9にミューテックスの動作例を示します。

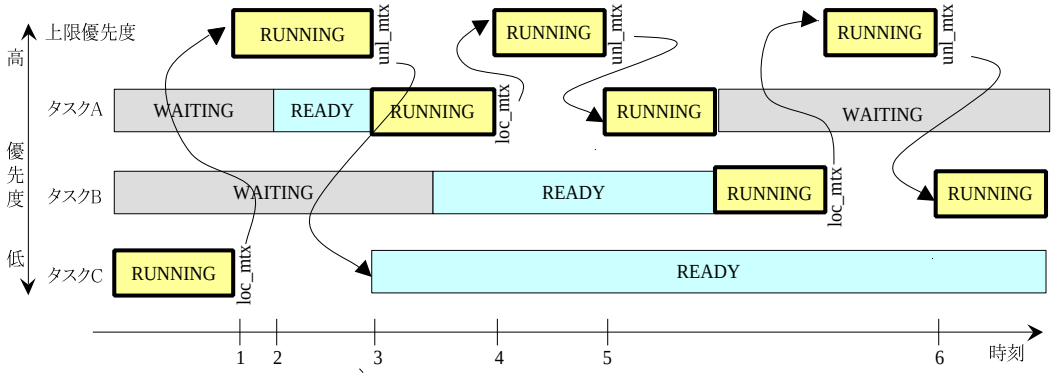


図4.9　ミューテックスの動作例

図の解説

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

■ベース優先度と現在優先度

タスクの優先度には、ベース優先度と現在優先度があります。タスクのスケジューリングは、現在優先度に従って行われます。

ミューテックスをロックしていない時は、両者は常に同じです。

ミューテックスをロックすると、現在優先度のみがそのミューテックスの上限優先度に引き上げられます。

タスクの優先度を変更する chg_pri, ichg_pri では、ミューテックスをロックしていないタスクの場合は、ベース優先度・現在優先度とも変更されますが、ミューテックスをロックしているタスクの場合はベース優先度のみが変更されます。また、ミューテックスロック中またはミューテックスのロックを待っているタスクの場合は、ロック中またはロックを待っているミューテックスのいずれかの上限優先度よりも高い優先度を指定すると、E_ILUSE エラーになります。

get_pri, iget_pri では、現在優先度を取得します。

4.13 メッセージバッファ

メッセージバッファは、コピーによってメッセージの受け渡しを行うオブジェクトです。メッセージ送信後は相手が受信したかどうかに関わらず、直ちに送信したメッセージ領域を再利用することができます。

メッセージバッファの領域は、リソースプールから割り付けられます。

メッセージバッファを生成するには、以下の方法があります。

- cre_mbf, icre_mbf サービスコール
  指定されたID番号でメッセージバッファを生成します。
- acre_mbf, iacre_mbf サービスコール
  カーネルが自動的にID番号を割り当てて、メッセージバッファを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、メッセージバッファを操作するサービスコールとして以下があります。

- del_mbf サービスコール
  指定されたメッセージバッファを削除します。
- snd_mbf, tsnd_mbf サービスコール
  メッセージバッファにメッセージを送信します。送信できない場合(メッセージバッファが満杯、または既に他のタスクが送信待ちになっている)は待ち状態になります。tsnd_mbfでは、タイムアウト時間を指定することができます。
- psnd_mbf, ipsnd_mbf サービスコール
  メッセージバッファにメッセージを送信します。送信できない場合(メッセージバッファが満杯、または既に他のタスクが送信待ちになっている)はエラーが返ります。
- rcv_mbf, trcv_mbf サービスコール
  メッセージバッファからメッセージを受信します。受信できない場合(メッセージバッファにメッセージが無い)は待ち状態になります。trcv_mbfでは、タイムアウト時間を指定することができます。
- prcv_mbf サービスコール
  メッセージバッファからメッセージを受信します。受信できない場合(メッセージバッファにメッセージが無い)はエラーが返ります。
- ref_mbf, iref_mbf サービスコール
  メッセージバッファの状態を参照します。

図 4.10にメッセージバッファの動作例を示します。

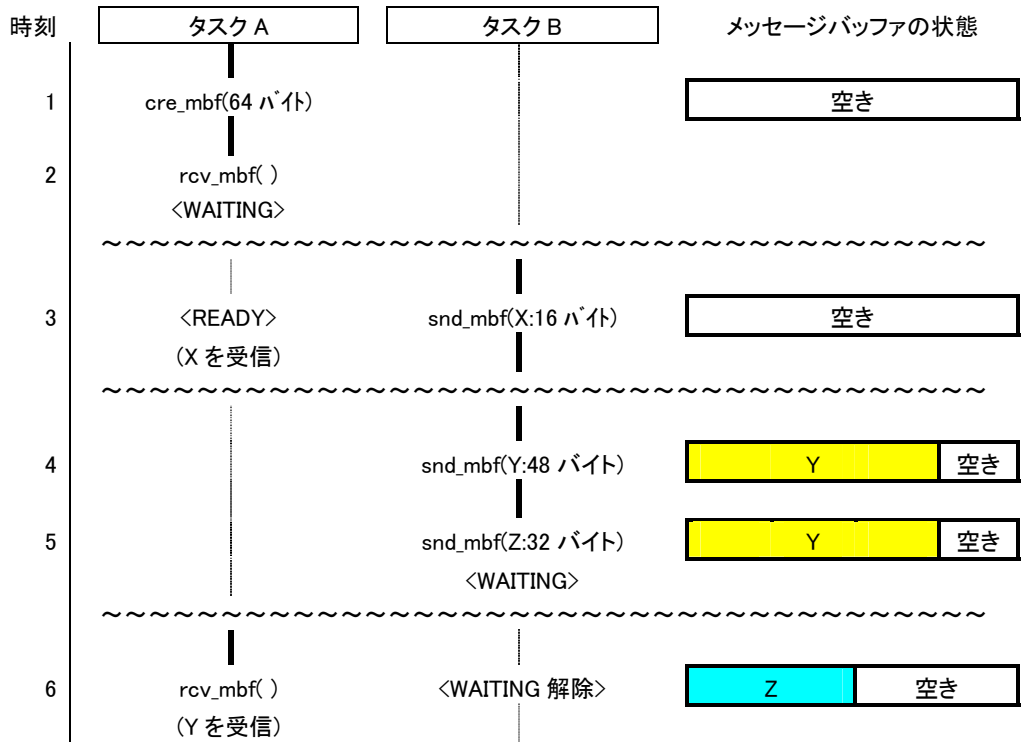


図4.10　メッセージバッファ動作例

図の解説

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

4.14 固定長メモリプール

固定長メモリプールは、固定長のメモリを動的に割り当てるためのオブジェクトです。固定長であるため、可変長メモリプールよりも高速です。

固定長メモリプールを生成するには、以下の方法があります。

- cre_mpf, icre_mpf サービスコール
  指定されたID番号で固定長メモリプールを生成します。
- acre_mpf, iacre_mpf サービスコール
  カーネルが自動的にID番号を割り当てて、固定長メモリプールを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

固定長メモリプールの領域は、生成時にそのアドレスを指定するか、またはカーネルが割り付けるように指定することができます。ただし、コンフィギュレータで生成する場合は、アドレスを指定することはできません。

カーネルが割り付ける場合は、システムプールから割り付けられます。メモリオブジェクト保護機能を使用する場合は、メモリプール領域がひとつのメモリオブジェクトになります。コンフィギュレータで生成する場合はアクセス許可ベクタを指定でき、サービスコールで生成する場合は生成元のドメインのみがリード・ライト可能なアクセス許可ベクタとなります。

また、固定長メモリプールを操作するサービスコールとして以下があります。

- del_mpf サービスコール
  指定された固定長メモリプールを削除します。
- get_mpf, tget_mpf サービスコール
  固定長メモリブロックを獲得します。獲得できない場合(空きメモリが無い場合)は待ち状態になります。tget_mpfは、タイムアウト時間を指定することができます。
- pget_mpf, ipget_mpf サービスコール
  固定長メモリブロックを獲得します。獲得できない場合(空きメモリが無い場合)はエラーが返ります。
- rel_mpf, irelmpf サービスコール
  固定長メモリブロックを返却します。
- ref_mpf, iref_mpf サービスコール
  固定長メモリプールの状態を参照します。

図 4.11に固定長メモリプールの動作例を示します。

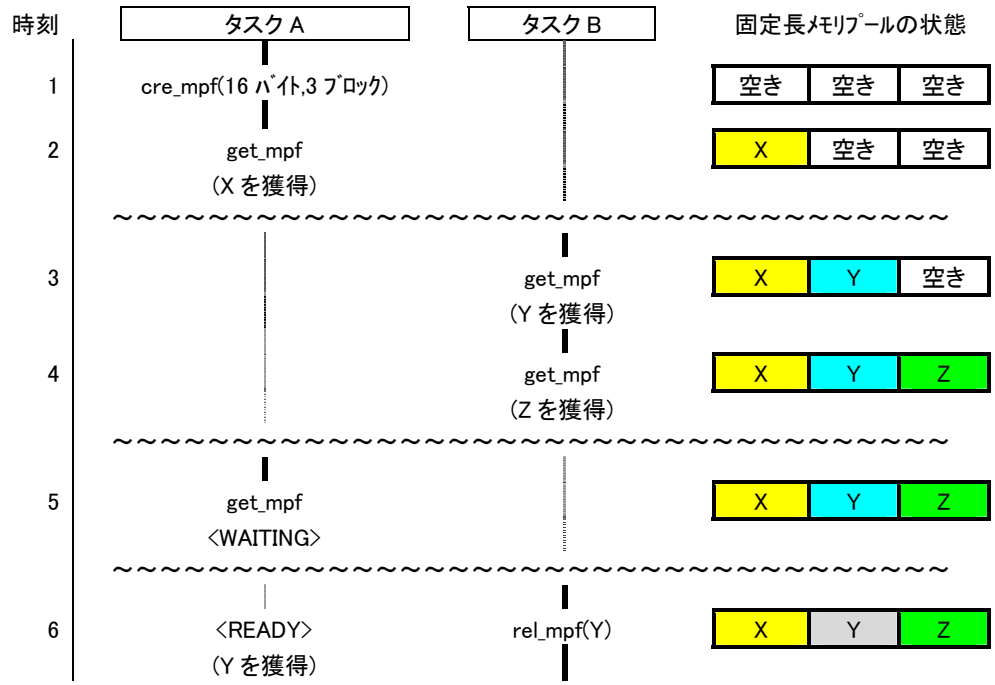


図4.11　固定長メモリプールの動作例

図の解説

1. タスクAが、cre_mpfで16バイトのブロックが3つの固定長メモリプールを生成します。
2. タスクAがget_mpfでブロックXを獲得します。
3. タスクBがget_mpfでブロックYを獲得します。
4. タスクBがget_mpfでブロックZを獲得します。
5. タスクAがget_mpfでブロックを獲得しようとしますが、空きブロックが無いのでタスク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Yが割り当てられます。

4.15 可変長メモリプール

可変長メモリプールは、任意サイズのメモリを動的に割り当てるためのオブジェクトです。
可変長メモリプールを生成するには、以下の方法があります。

- cre_mpl, icre_mpl サービスコール
  指定されたID番号で可変長メモリプールを生成します。
- acre_mpl, iacre_mpl サービスコール
  カーネルが自動的にID番号を割り当てて、可変長メモリプールを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

可変長メモリプールの領域は、生成時にそのアドレスを指定するか、またはカーネルが割り付けるように指定することができます。ただし、コンフィギュレータで生成する場合は、アドレスを指定することはできません。

カーネルが割り付ける場合は、システムプールから割り付けられます。メモリオブジェクト保護機能を使用する場合は、メモリプール領域がひとつのメモリオブジェクトになります。コンフィギュレータで生成する場合はアクセス許可ベクタを指定でき、サービスコールで生成する場合は生成元のドメインのみがリード・ライト可能なアクセス許可ベクタとなります。

また、可変長メモリプールを操作するサービスコールとして以下があります。

- del_mpl サービスコール
  指定された可変長メモリプールを削除します。
- get_mpl, tget_mpl サービスコール
  指定したサイズの可変長メモリブロックを獲得します。獲得できない場合(空きメモリが無い場合、または既に獲得待ちタスクが存在する場合)は待ち状態になります。tget_mplでは、タイムアウト時間を指定することができます。
- pget_mpl, ipget_mpl サービスコール
  可変長メモリブロックを獲得します。獲得できない場合(空きメモリが無い場合、または既に獲得待ちタスクが存在する場合)はエラーが返ります。
- rel_mpl, irel_mpl サービスコール
  可変長メモリブロックを返却します。
- ref_mpl, iref_mpl サービスコール
  可変長メモリプールの状態を参照します。

図 4.12に可変長メモリプールの動作例を示します。

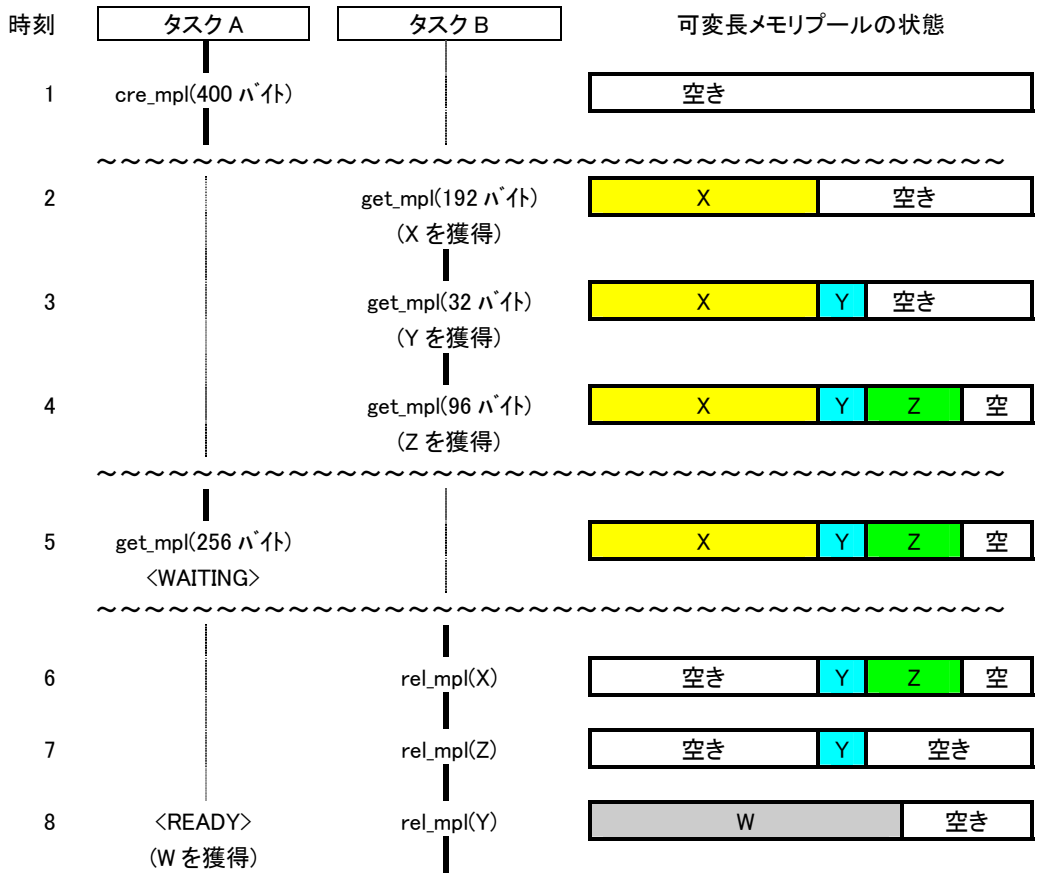


図4.12　可変長メモリプールの動作例

図の解説

1. タスクAが、cre_mplで400バイトの可変長メモリプールを生成します。
2. タスクBがget_mplで192バイトのブロックXを獲得します。なお、カーネルはブロックを割付ける際にメモリプール領域を16バイト消費しますが、図中ではこの表記はしていません。
3. タスクBがget_mplで32バイトのブロックYを獲得します。
4. タスクBがget_mplで96バイトのブロックZを獲得します。
5. タスクAがget_mplで256バイトのブロックを獲得しようとしますが、空きが無いのでタスクA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バイト以上の連続した空きが生じたので、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。

4.15.1 断片化問題

可変長メモリプールでは、メモリの獲得と解放を繰り返すと、いわゆる「断片化現象」が発生します。断片化が発生すると、メモリ獲得時に、空き領域の合計が十分でも必要なサイズの連続空き領域が存在しないためにメモリ獲得できない状況が発生しえます。

本製品では、断片化を発生しにくくする機能もサポートしています。断片化を発生しにくくするには、可変長メモリプールの生成時に VTA_UNFRAGMENT 属性を指定します。

詳細は、以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

4.16 時間管理機能

カーネルは、時間に関する以下のような機能をサポートしています。

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

時間パラメータの単位時間は 1[ms]です。

カーネルは、システム時刻と呼ぶ 48 ビットのカウンタによって時間管理機能を実現しています。

4.16.1 時間の精度

タイムアウトなどの時間パラメータの単位は[ms]ですが、その精度は TIC_NUME／TIC_DENO[ms]となります。この精度で、システム時刻の更新や時間管理が行われます。

また、指定した時間以上が経過してから時間イベント(タイムアウト発生や周期ハンドラ起動など)が発生するようになっています。

図 4.13に、実時刻が 9.2[ms]の時点で tslp_tsk(5)を実行した場合の例を示します。

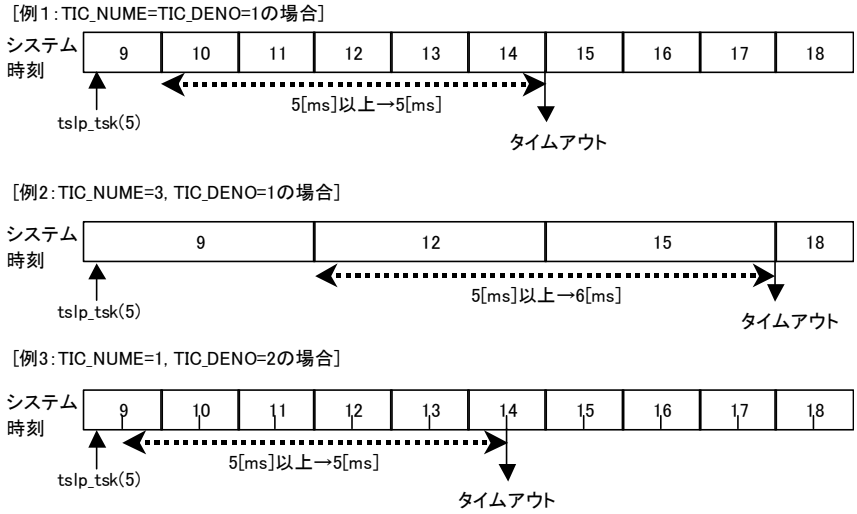


図4.13　時間の精度

4.16.2 システム時刻の設定・参照

get_tim, iget_tim サービスコールで、現在のシステム時刻を取得することができます。

また、set_tim, iset_tim サービスコールで、システム時刻を指定された値に変更することができます。ただし、システム時刻を変更しても、既に時間監視中のイベント(タイムアウトなど)が発生する実時刻は変化しません。

4.16.3 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。

周期ハンドラは、非タスクコンテキストに分類されます。

周期ハンドラは、カーネルドメインに所属します。

周期ハンドラを生成するには、以下の方法があります。

- cre_cyc, icre_cyc サービスコール
  指定されたID番号で周期ハンドラを生成します。TA_STA属性を指定すれば、周期ハンドラの動作を開始することもできます。
- acre_cyc, iacre_cyc サービスコール
  カーネルが自動的にID番号を割り当てて、周期ハンドラを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、周期ハンドラを操作するサービスコールとして以下があります。

- del_cyc サービスコール
  周期ハンドラを削除します。
- sta_cyc, ista_cyc サービスコール
  周期ハンドラの動作を開始します。
- stp_cyc, istp_cyc サービスコール
  周期ハンドラの動作を停止します。
- ref_cyc, iref_cyc サービスコール
  周期ハンドラの状態を参照します。

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

起動位相を保存するかどうかは生成時に指定します。

図 4.14に周期ハンドラの動作例を示します。

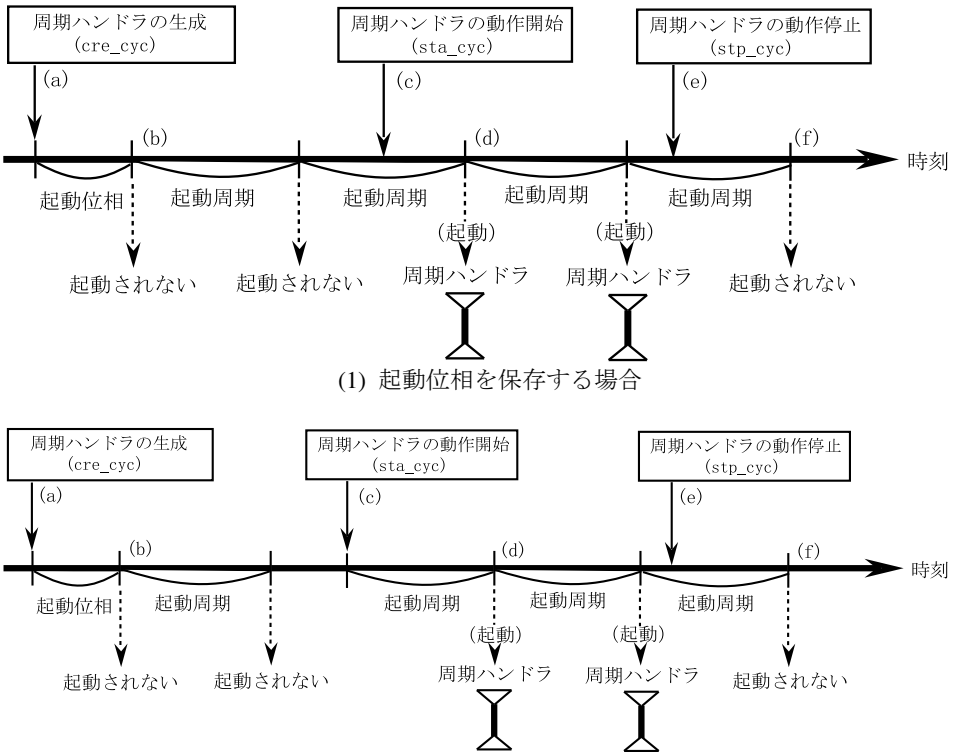


図4.14　周期ハンドラの動作例

図の解説

(a)　周期ハンドラ(TA_STA属性なし)を生成します。

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

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

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

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

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

4.16.4 アラームハンドラ

アラームハンドラは、指定した時刻になると 1 度だけ起動されるタイムイベントハンドラです。
アラームハンドラは、非タスクコンテキストに分類されます。
アラームハンドラは、カーネルドメインに所属します。
アラームハンドラを生成するには、以下の方法があります。

- cre_alm, icre_alm サービスコール
  指定されたID番号でアラームハンドラを生成します。TA_STA属性を指定すれば、周期ハンドラの動作を開始することもできます。
- acre_alm, iacre_alm サービスコール
  カーネルが自動的にID番号を割り当てて、アラームハンドラを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、アラームハンドラを操作するサービスコールとして以下があります。

- del_alm サービスコール
  アラームハンドラを削除します。
- sta_alm, ista_alm サービスコール
  アラームハンドラの動作を開始します。
- stp_alm, istp_alm サービスコール
  アラームハンドラの動作を停止します。
- ref_alm, iref_alm サービスコール
  アラームハンドラの状態を参照します。

図 4.15にアラームハンドラの動作例を示します。

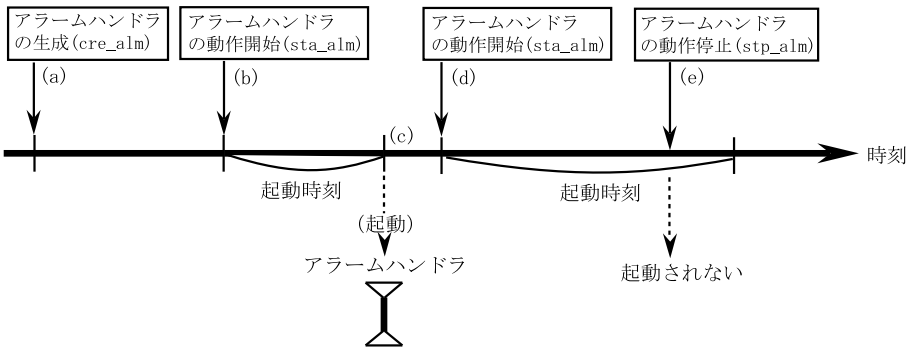


図4.15　アラームハンドラの動作例

図の解説

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

4.16.5 オーバーランハンドラ

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

オーバーランハンドラは、非タスクコンテキストに分類されます。

オーバーランハンドラは、カーネルドメインに所属します。

オーバーランハンドラを生成するには、以下の方法があります。

- def_ovr サービスコール
  オーバーランハンドラを定義します。
- コンフィギュレータで定義

また、オーバーランハンドラを操作するサービスコールとして以下があります。

- sta_ovr, ista_ovr サービスコール
  タスクにオーバーラン時間を設定し、オーバーラン監視を開始します。
- stp_ovr, istp_ovr サービスコール
  タスクのオーバーラン監視を停止します。
- ref_ovr, iref_ovr サービスコール
  オーバーハンドラの状態を参照します。

図 4.16にオーバーランハンドラの動作例を示します。

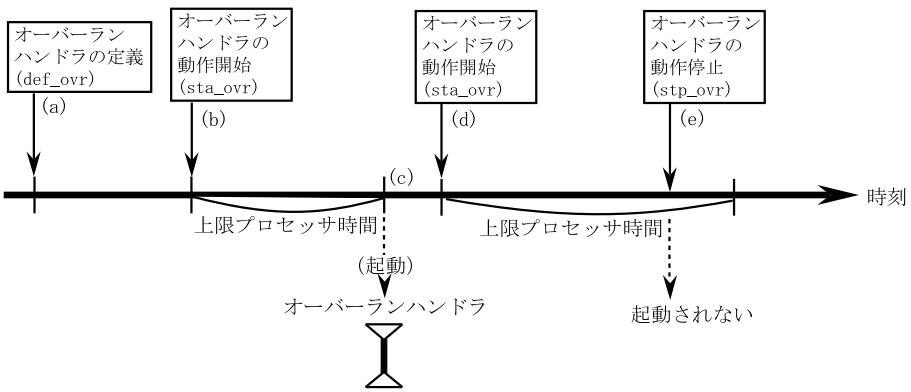


図4.16 オーバーランハンドラの動作例

図の解説

(a) オーバーランハンドラを定義します。

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

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

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

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

4.16.6 タイマドライバ

時間管理機能を実現するためにはタイマドライバが必要です。タイマドライバには、標準タイマドライバと最適化タイマドライバの 2 種類があります。どちらを使用するかは、コンフィギュレータの CFG_OPTTMR で指定します。

標準タイマドライバは、TIC_NUME/TIC_DENO[ms]周期でタイマ割込みを発生させるドライバで、ユーザが作成してカーネルに組み込まなければなりません。本製品では、各種マイコン内蔵タイマ用のサンプルを提供しています。

最適化タイマドライバは、カーネルが提供するドライバで、タイマ割込み頻度を低減できるという特長がありますが、利用できるハードウェアが限定されています。

■関連ページ　・標準タイマドライバの作成方法→「9.標準タイマドライバ」
・最適化タイマドライバ→「4.17 最適化タイマドライバ」

4.16.7 注意事項

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) アラームハンドラの起動と実行
(c) 周期ハンドラの起動と実行
(d) オーバーランハンドラの起動と実行
(e) tslp_tskなどのタイムアウト付きサービスコールによるタスクのタイムアウト処理

これらの処理は全て、CFG_KNLLVL 以下の割込みレベルをマスクした状態で行われます。

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

- 割込みに対するレスポンスの悪化
- システム時刻の遅れ

これを避けるために、以下を遵守してください。

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

4.17 最適化タイマドライバ

4.17.1 概要

標準タイマドライバでは、サービスコールの時間精度(CFG_TICNUME／CFG_TICDENO[ms])と同じ周期でタイマ割込みを発生させます。最適化タイマドライバを使用すると、時間精度を維持したまま、タイマ割込み頻度を低減することができます。

- マイコンのスリープモード中に発生するタイマ割込み頻度が減少し、省電力効果が高まります。
- タイマ割込み頻度が減少するため、タイマ割込み処理による CPU 占有率が低減され、システムのスループットが向上します。あるいは、低減分だけより省電力モードの時間を延ばすことができます。

最適化タイマドライバは、特定のマイコンに内蔵された TMU 用に設計されており、標準タイマドライバとは異なり、カーネル内部に組み込まれています。ユーザが独自に最適化タイマドライバを作成することはできません。

4.17.2 動作

図 4.17に、時間精度(CFG_TICNUME／CFG_TICDENO)が 1[ms]の場合の標準タイマドライバを使用した場合と最適化タイマドライバを使用した場合の動作例を示します。

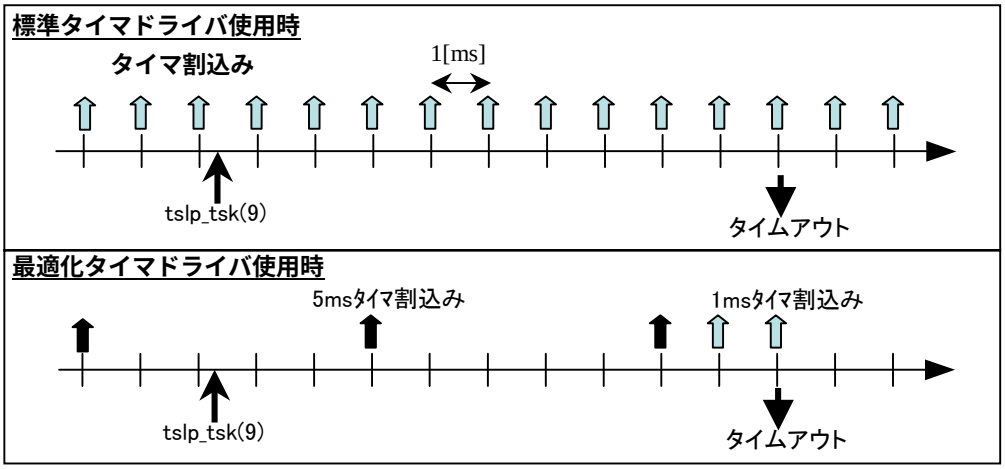


図4.17 動作例

図 4.17に示すように、最適化タイマドライバでは 1[ms]と 5[ms]の２つの TMU タイマチャネルを使用します。1[ms]を「高精度周期」、5[ms]を「粗精度周期」と呼びます。高精度周期は、コンフィギュレータで指定する CFG_TICNUME/CFG_TICDENO[ms]で算出される時間です。また、粗精度周期は、高精度周期の整数倍の時間で、コンフィギュレータの CFG_LONGTICRATE で指定します。

最適化タイマドライバ使用時、カーネルは適時以下の状況を調査します。

- タイムアウト指定のサービスコール(txxx_yyy)による待ちタスク
- dly_tsk サービスコールによる待ちタスク
- 周期ハンドラ
- アラームハンドラ

そして、高精度周期割込みが必要かどうかを判断し、その結果に応じて高精度周期割込みのイネーブル／ディスエーブルを制御します。
一方、粗精度周期割込みは常にイネーブルです。

また、図 4.17には表現されていませんが、オーバーランハンドラの監視用にさらにもう 1 つの TMU チャネルを使用します。このチャネルのタイマ割込みは、タスクが上限プロセッサ時間を使いきったときにのみ発生します。

図 4.18に、本機能による効果イメージを示します。本機能を使用することでタイマ割込み頻度が低減するため、標準タイマドライバ使用時に比べて以下の効果があります。

- 早くスリープモードに移行できる。(タイマ割込み処理による CPU 消費量が少ない)
- タイマ割込みによるスリープモードの解除の頻度が少ない

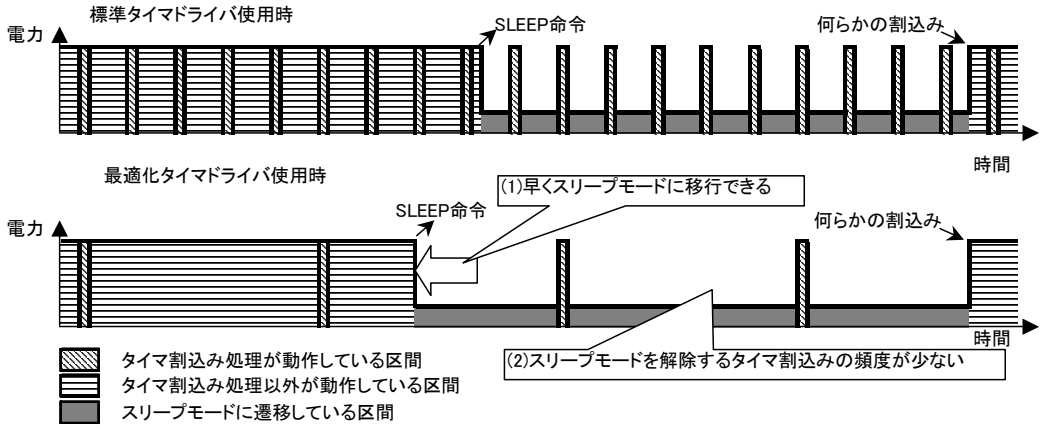


図4.18　最適化タイマドライバによる効果イメージ

4.17.3　利用可能なマイコン

最適化タイマドライバはマイコン内蔵の TMU を使用しますが、TMU を内蔵するすべてのマイコンで利用できるわけではありません。以下に、本マニュアル作成時点で最適化タイマドライバを利用可能なマイコンを示します。

SH73180, SH73230

4.17.4　ハードウェア初期化処理

カーネル起動時に、以下の最適化タイマドライバの初期化処理が行われます。

(1)　TMUのモジュールスタンバイ状態を解除します。
(2)　チャネル0,1,2の初期設定を行います。
(3)　割込みコントローラに対して、チャネル0,1,2の割込みレベルを設定します。

ただし、def_ovr サービスコールが組み込まれていない場合は、(2), (3)のチャネル 2 に関する設定は行いません。

なお、最適化タイマドライバが、TMU をモジュールストップ状態に移行させることはありません。

4.18 割込み管理機能

4.18.1 割込みハンドラ

割込みは、外部割込み端子や周辺モジュールからの要求によって発生します。割込みが発生すると、カーネルを介して割込みハンドラが実行されます。

割込みは、割込みハンドラ番号によって区別されます。割込みハンドラ番号は、CPU の INTEVT コードと同じです。

割込みハンドラは、非タスクコンテキストに分類されます。

割込みハンドラは、カーネルドメインに所属します。

割込みハンドラを定義するには、以下の方法があります。

- def_inh, idef_inh サービスコール
  指定された割込みハンドラ番号に割込みハンドラを定義します。
- コンフィギュレータで定義

割込みハンドラが定義されていない番号の割込みが発生すると、システムダウンになります。

■関連ページ　「8.4 割込みハンドラ」

4.18.2 カーネルレベル(CFG_KNLLVL)

カーネルレベル (CFG_KNLLVL)はカーネル実行中にマスクする割込みレベルで、かつタイマ割込みレベルです。CFG_KNLLVL はコンフィギュレータで設定します。

カーネルは、クリティカルセクションおよびタイマ割込み処理を CFG_KNLLVL 以下のレベルの割込みをマスクした状態で実行します。

また、メモリオブジェクト保護機能使用時は、TLB 関連の例外処理を全割込みをマスクした状態(SR.BL=1)で実行します。

カーネルレベルよりも高いレベルの割込みは、カーネルのクリティカルセクション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、SR.IMASK がカーネルレベルより高い時(カーネルレベルよりも高いレベルの割込みハンドラは、常にこの条件を満たします)は、サービスコールを呼び出してはなりません。呼び出した場合、カーネル内部でマスクレベルが下がることになります。

4.18.3 割込みの禁止

各割込み要因に関係なく割込みを禁止するには、以下の方法があります。

- SR.IMASK を変更する
- SR.BL を 1 にする

(1) SR.IMASK ビットを変更する

SR.IMASK ビットを変更することで、特定レベル以下の割込みを禁止することができます。

以下の事項に注意してください。

- SR.IMASK がカーネルレベル(CFG_KNLLVL)より高い値の時には、サービスコールを呼び出してはなりません。呼び出した場合、システムの正常な動作は保証されません。
- 割込みハンドラの場合は、自割込みレベルより低い値に変更してはなりません。その他の非タスクコンテキストの場合は、起動時より下げてはなりません。

SR.IMASK を変更するには、以下の方法があります。

(a) loc_cpu, iloc_cpu サービスコール

この方法では、CPU ロック状態に遷移します。CPU ロック状態では、SR.IMASK がカーネルレベル(CFG_KNLLVL)となります。

CPU ロック状態を解除するには、unl_cpu または iunl_cpu を発行してください。

以下の事項に注意してください。

- 非タスクコンテキストで CPU ロック状態に遷移させた場合は、そのハンドラ内で CPU ロック状態を解除しなければなりません。
- CPU ロック状態の間は利用できるサービスコールに制限があります。

■関連ページ　CPU ロック状態で利用できるサービスコール→「6.4.3 CPU ロック状態」

(b) chg_ims, ichg_ims サービスコール

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

割込みを許可するには、chg_ims で SR レジスタの IMASK ビットを変更前の値に戻してください。

以下の事項に注意してください。

- この方法では、カーネルレベル(CFG_KNLLVL)より高い値には変更できません。この場合、chg_ims サービスコールは E_PAR エラーが返ります。カーネルレベルより高い値に変更する場合は、「(c) サービスコールを使わずに SR レジスタの IMASK ビットを変更する」に示す方法を使用してください。
- タスクコンテキストで、本方法で IMASK を 0 以外に変更している間は、ディスパッチ保留状態になります。
- CPU ロック状態の時には、chg_ims, ichg_ims はエラーE_CTX となります。

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

LDC 命令によって SR レジスタの IMASK ビットを変更します。即ち、指定したレベル以下の割込みがマスクされます。C 言語では、コンパイラがサポートしている組み込み関数 set_imask()または set_cr()を使用します。

割込みを許可するには、SR レジスタの IMASK ビットを変更前の値に戻してください。

以下の事項に注意してください。

- タスクコンテキストでは、カーネルレベル(CFG_KNLLVL)よりも高い値および 0 にのみ変更できます。この方法でそれ以外の値に変更した場合、正常な動作は保証されません。
- CPU ロック状態の場合、IMASK を CFG_KNLLVL 未満の値に変更してはなりません。変更した場合、システムの正常な動作は保証されません。
- LDC 命令は特権命令なので、この方法はユーザドメインの場合は使用できません。

(2) SR レジスタの BL ビットを変更する

この方法では、全割込みが禁止されます。

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

割込みを許可するには、SR レジスタの BL ビットを 0 に戻してください。

以下の事項に注意してください。

- BL ビットが 1 の時に CPU 例外が発生すると、CPU がリセットします。このため、例外を発生させないようにしなければなりません。特に、メモリオブジェクト保護機能を組み込んでいる場合は、MMU 対象領域へのアクセスは、そのアドレスが vloc_tlb サービスコールによっ

てロックされている場合を除き、TLB ミス例外が発生する可能性があるため、アクセスしてはなりません。
同じ理由から、サービスコールを呼び出してはなりません。
- LDC 命令は特権命令なので、この方法はユーザドメインの場合は使用できません。

■関連ページ　「8.4.4 NMI に関する注意事項」

4.19 CPU 例外

CPU 例外は、プログラムの実行に起因して発生します。CPU 例外が発生すると、カーネルを介して CPU 例外ハンドラが実行されます。

CPU 例外は、CPU 例外ハンドラ番号によって区別されます。CPU 例外ハンドラ番号は、CPU の EXPEVT コードと同じです。

CPU 例外ハンドラは、非タスクコンテキストに分類されます。

CPU 例外ハンドラは、カーネルドメインに所属します。

CPU 例外ハンドラを定義するには、以下の方法があります。

- def_exc, idef_exc サービスコール
  指定されたCPU例外ハンドラ番号にCPU例外ハンドラを定義します。
- コンフィギュレータで定義

CPU 例外ハンドラが定義されていない番号の CPU 例外が発生すると、システムダウンになります。

CPU 例外ハンドラには、CPU 例外発生時点のレジスタ値などを格納したパケットが渡されます。また、このパケットを使って CPU 例外発生時点の各種情報を取り出す以下のマクロを提供しています。

- VSNS_CTX：CPU 例外発生時点が非タスクコンテキストかどうかを返す
- VSNS_LOC：CPU 例外発生時点が CPU ロック状態であったかどうかを返す
- VSNS_DSP：CPU 例外発生時点がディスパッチ禁止状態であったかどうかを返す
- VSNS_DPN：CPU 例外発生時点がディスパッチ保留状態であったかどうかを返す
- VSNS_TEX：CPU 例外発生時点がタスクコンテキストである場合、そのタスクがタスク例外許可状態であったかどうかを返す
- VGET_TID：CPU 例外発生時点の実行状態のタスク ID を返す
- VGET_DID：CPU 例外発生時点の実行状態のタスクが所属するドメイン ID を返す

また、このパケットを書き換えることで CPU 例外復帰後の状態を変更することもできます。

■関連ページ　「8.8 CPU 例外ハンドラ」

4.20 拡張サービスコールとトラップ

拡張サービスコールとトラップは、共にカーネルドメインに所属するプログラム(拡張サービスコールルーチンとトラップルーチン)を呼び出すための仕組みです。

いずれも、ユーザが作成したプログラムをカーネルに定義することで、呼び出すことができるようになります。

■関連ページ　「8.3 拡張サービスコールルーチン、トラップルーチン」

拡張サービスコールルーチンとトラップルーチンのコンテキスト種別は、呼び出し元と同じです。

また、いずれもカーネルドメインに所属します。

4.20.1 拡張サービスコール

拡張サービスコールは、機能コードによって区別されます。機能コード毎に「拡張サービスコールルーチン」を定義し、cal_svc, ical_svc によって指定した機能コードに対応した拡張サービスコールルーチンが呼び出されます。

拡張サービスコールを定義するには、以下の方法があります。

- def_svc, idef_svc サービスコール
  指定された機能コードに拡張サービスコールを定義します。
- コンフィギュレータで定義

4.20.2 トラップ

トラップは、トラップ番号によって区別されます。

0～15 のトラップ番号はカーネル用に予約されているので、アプリケーションで使用できるトラップ番号は 16 以降です。TRAPA 命令を実行すると、そのトラップ番号に対応したトラップルーチンが呼び出されます。

トラップルーチンを定義するには、以下の方法があります。

- vdef_trp, ivdef_trp サービスコール
  指定された番号にトラップを定義します。
- コンフィギュレータで定義

4.21 メモリオブジェクト保護機能

4.21.1 概要

メモリオブジェクト保護機能では、「だれ」が「どのメモリ」に「どのようなアクセス」が可能かを管理します。これによって、以下の機能を実現します。

- 許可のないアクセスの検出
- カーネルのサービスコールで渡されるアドレスパラメータのエラーチェック

本機能により、許可されていないメモリへのアクセスが禁止され、デバッグ効率の向上、およびシステム出荷後の意図しない不正メモリアクセスからのシステム基幹部の保護が可能になります。

「だれ」に該当するのがユーザドメイン ID です。カーネルドメインに所属するプログラムは、管理対象外で常にアクセスできます。

タスクはいずれかのユーザドメインまたはカーネルドメインに所属します。各種ハンドラはカーネルドメインに所属します。

「どのメモリ」に該当するのが「メモリオブジェクト」で、「どのようなアクセス」に該当するのが「リード(命令フェッチを含む)」と「ライト」です。カーネルは、メモリオブジェクト毎にこれを管理します。この管理情報を、「アクセス許可ベクタ」と呼びます。

また、メモリオブジェクトには以下の属性を設定することができます。

- リードオンリーかリードライト可能か
- キャッシュ可否、キャッシュする場合は書き込みモードがコピーバックかライトスルーか

カーネルは、本機能を実現するためにプロセッサに内蔵されている MMU(Memory Management Unit)を使用します。

メモリオブジェクト保護機能を組み込むには、CFG_PROTMEM を選択します。

図 4.19に、メモリオブジェクト保護機能の概念図を示します。

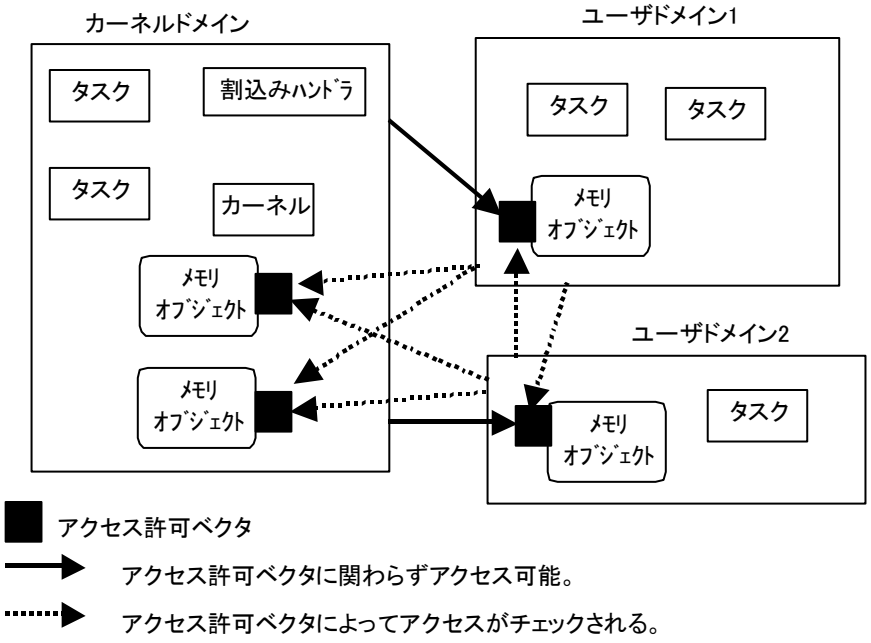


図4.19　メモリオブジェクト保護機能の概念図

4.21.2　メモリオブジェクトの種類

メモリオブジェクトには、表 4.2に示すものがあります。

表4.2　メモリオブジェクトの種類

メモリオブジェクト		生成方法
システムプールから割り付けられるメモリオブジェクト	カーネルがシステムプールから割り付けるユーザドメイン所属タスクのスタック	タスクの生成によって生成される。
	カーネルがシステムプールから割り付ける固定長メモリプール領域	固定長メモリプールの生成によって生成される。
	カーネルがシステムプールから割り付ける可変長メモリプール領域	可変長メモリプールの生成によって生成される。
保護メモリプールから獲得した保護メモリブロック		保護メモリブロックの獲得によって生成される。
保護メールボックスから受信した保護メモリブロック		
コンフィギュレータで定義した静的メモリオブジェクト		－

4.21.3 属性と所属ドメイン

各メモリオブジェクトは、属性を持ちます。本カーネルでは、以下の属性をサポートしています。

(1) リードオンリー(TA_RO)またはリードライト可能(TA_RW)
TA_ROの場合は、メモリオブジェクトへのライトが発生した時に例外が検出されます。これは、後述のアクセス許可ベクタに関係なく検出されます。

(2) キャッシャブル(TA_CACHE), 非キャッシャブル(TA_UNCACHE)

(3) キャッシュに対する書き込みモード：コピーバック(TA_WBACK)またはライトスルー(TA_WTHROUGH)

ただし、キャッシュに関する属性は、キャッシュが Enable の場合のみ意味を持ちます。また、内蔵メモリの場合は、キャッシュに関する属性が無視され、常に非キャッシャブルとなります。

■関連ページ 「5.3.3 内蔵メモリ」

また、メモリオブジェクトは、いずれかのドメインに所属します。所属ドメインには、以下の意味があります。

(1) 保護メモリブロックを保護メモリプールに返却(rel_mpp)する場合は、保護メモリブロックの所属ドメインと返却するタスクの所属ドメインが同じでなければなりません。

(2) 保護メモリブロックを保護メールボックスに送信(snd_mbp)する場合は、保護メモリブロックの所属ドメインと送信するタスクの所属ドメインが同じでなければなりません。

(3) システムプールから割り付けられるメモリオブジェクトのアクセス許可ベクタの初期値は、所属ドメインによって変わります(次節を参照してください)

表 4.3に、各メモリオブジェクトの所属ドメインと属性を示します。なお、属性はサービスコールで動的に変更することはできません。

表4.3 各メモリオブジェクトの所属ドメインと属性

メモリオブジェクト		所属ドメイン	属性
システムプールから割り付けられるメモリオブジェクト	カーネルがシステムプールから割り付けるユーザドメイン所属タスクのスタック	対象タスクが所属するドメイン(タスク属性で指定)	常に以下の属性となります。 • TA_RW • TA_CACHE • TA_WBACK
	カーネルがシステムプールから割り付ける固定長メモリプール領域	(1)コンフィギュレータで生成した場合 カーネルドメイン (2)サービスコールで生成した場合 サービスコールを呼び出したプログラムが所属するドメイン	
	カーネルがシステムプールから割り付ける可変長メモリプール領域		
保護メモリプールから獲得した保護メモリブロック		獲得したタスクが所属するドメイン	コンフィギュレータで保護メモリプール生成時に指定
保護メールボックスから受信した保護メモリブロック		受信したタスクが所属するドメイン	
コンフィギュレータで定義する静的メモリオブジェクト		カーネルドメイン	コンフィギュレータで静的メモリオブジェクト登録時に指定

4.21.4 アクセス許可ベクタ

各メモリオブジェクトは、アクセス許可ベクタを持ちます。アクセス許可ベクタとは、そのメモリオブジェクトに対してどのユーザドメインがどのようなアクセスが可能かを示す情報です。なお、カーネルドメインは、アクセス許可ベクタに関係なくメモリオブジェクトにアクセスできます。

本カーネルでは、表 4.4に示すアクセス許可ベクタをサポートしています。

表4.4 アクセス許可ベクタ

アクセス許可ベクタ	アクセス可能なユーザドメイン	
	ライトアクセス *	リードアクセス
TACT_KERNEL	なし(全ユーザドメイン不可)	なし(全ユーザドメイン不可)
TACT_PRW(domid)	domid のユーザドメインのみ	domid のユーザドメインのみ
TACT_PRO(domid)	なし(全ユーザドメイン不可)	domid のユーザドメインのみ
TACT_SRW	全ユーザドメイン	全ユーザドメイン
TACT_SRO	なし(全ユーザドメイン不可)	全ユーザドメイン
TACT_SRPW(domid)	domid のユーザドメインのみ	全ユーザドメイン

【注】 TA_RO 属性のメモリオブジェクトは、アクセス許可ベクタに関わらず、カーネルドメインも含めてライトはできません。

アクセス許可ベクタは、sac_mem サービスコールで変更することができます。

表 4.5に、各メモリオブジェクトのアクセス許可ベクタの初期値を示します。

表4.5 各メモリオブジェクトのアクセス許可ベクタ

メモリオブジェクト		アクセス許可ベクタの初期値
システムプールから割り付けられるメモリオブジェクト	カーネルがシステムプールから割り付けるユーザドメイン所属タスクのスタック	TACT_PRW(domid) domid は、対象タスクが所属するドメイン
	カーネルがシステムプールから割り付ける固定長メモリプール領域	(1)コンフィギュレータで生成した場合 コンフィギュレータで指定 (2)サービスコールで生成した場合 所属ドメインのみがリード・ライト可能なように設定されます。 所属ドメインがカーネルドメインの場合： TACT_KERNEL 所属ドメインがユーザドメインの場合： TACT_PRW(domid) domid は、所属ドメイン
	カーネルがシステムプールから割り付ける可変長メモリプール領域	
保護メモリプールから獲得した保護メモリブロック		TACT_PRW(domid) domid は獲得タスクの所属ドメイン
保護メールボックスから受信した保護メモリブロック		TACT_PRW(domid) domid は受信タスクの所属ドメイン
コンフィギュレータで定義する静的メモリオブジェクト		コンフィギュレータで指定

4.21.5 ページサイズ

MMU はページ単位にメモリを管理します。

本カーネルでは、基本的にはページサイズは 4kB(CFG_PAGESZ)です。ただし、静的メモリオブジェクトに限っては、ページサイズとして 4kB 以外のサイズを選択できます(ただし、1kB ページは本カーネル仕様としてサポートしていません)。

また、各メモリオブジェクトの先頭アドレスは、ページサイズにアラインされていなければなりません。

静的メモリオブジェクトでは、ページサイズを大きくすることで TLB の消費数が減るため、TLB ミスの確率が小さくなる期待が持てます。しかし一方で、メモリ効率が悪くなる場合があります。

例えば、40kB の領域を静的メモリオブジェクトとして扱いたい場合、ページサイズを 4kB にすると TLB エントリは 10 個必要となりますが、ページサイズを 64kB にするとひとつで済みます。つまり、ページサイズを 64kB にすると、TLB ミスの確率は単純計算では 1/10 となります。しかし、ページサイズを 64kB にした場合は、その領域は 64kB 境界に配置する必要があり、さらに後半の 24kB の空間には他の領域を配置できないので、無駄になります。

各メモリオブジェクトのアラインの扱いを表 4.6に示します。

表4.6 各メモリオブジェクトのアライン

メモリオブジェクト		ページサイズのアライン
システムプールから割り付けられるメモリオブジェクト	カーネルがシステムプールから割り付けるユーザドメイン所属タスクのスタック	カーネルが自動的にアラインします。 ただし、リンク時にはユーザがシステムプールのセクション(BSCP_hisyspl)を 4kB 境界に配置する必要があります。
	カーネルがシステムプールから割り付ける固定長メモリプール領域	
	カーネルがシステムプールから割り付ける可変長メモリプール領域	
保護メモリプールから獲得した保護メモリブロック、保護メールボックスから獲得した保護メモリブロック		カーネルが自動的にアラインします。 ただし、リンク時にはユーザが保護メモリプールのセクションを 4kB 境界に配置する必要があります。
コンフィギュレータで定義する静的メモリオブジェクト		(1)アドレス指定 そのメモリオブジェクトのページサイズ境界の値を指定しなければなりません。シンボルを指定する場合は、リンク時にそのシンボルアドレスがそのメモリオブジェクトのページサイズ境界となるようにしなければなりません。 (2)セクション指定 リンク時にはユーザがそのメモリオブジェクトのセクションをそのメモリオブジェクトのページサイズ境界に配置する必要があります

4.21.6 不正アクセスの検出

以下を参照してください。

■関連ページ 「5.3.1(2) 不正アクセスの検出」

4.21.7 TLB ミスペナルティ

MMU は、TLB(Translation Lookaside Buffer)と呼ぶ各ページの保護情報などを記憶するキャッシュメモリを持っており、カーネルは TLB を管理しています。

しかし TLB はキャッシュなので、システムに必要なすべての情報を記憶しておくことはできません。MMU 対象領域のアクセス発生時に TLB に有効な情報が存在しない場合、プロセッサは TLB ミス例外を発生させます。この時、カーネルはメモリ上に管理している情報を元に TLB を適切に更新し、例外発生時の処理を再開させます。しかし、アプリケーションから見た場合、メモリアクセス時にカーネルの TLB ミス例外処理が実行されることになるため、局所的には多大なオーバヘッドがかかることになります。これを TLB ミスペナルティと呼びます。

TLB ミスペナルティを避ける、もしくは低減する方法として以下があります。

(1) MMU 非対象領域に配置する

アクセス対象領域をメモリオブジェクトではなく MMU 非対象領域に配置すれば、TLB ミスは発生しなくなります。ただし、ユーザドメインから MMU 非対象領域としてアクセス可能な外部メモリは存在しません。唯一、内蔵メモリのみ CFG_IRAMUSAGE を「MMU 非対象領域、全モードからアクセス可能」と設定した場合にユーザドメインからアクセス可能になります。

(2) TLB をロックする

vloc_tlb サービスコールを用いることで、指定したアドレスを含むページを一時的に TLB に常駐させることができます。ただし、同時に常駐可能なページ数は、CFG_MAXLOCPAGE(最大 32)までに制限されています。

(3) 静的メモリオブジェクトのページサイズを大きくする

ページサイズを大きくすると TLB エントリの消費数が減るため、TLB ミスが発生する確率が下がります。

(4) システムで同時に使用するページ数を、マイコンの TLB ページ数以下にする

システムで同時に使用するページ数は、以下の総和となります。

- システムプールサイズ(CFG_SYSPOOLSZ)／4096
- 各静的メモリオブジェクトのサイズ／その静的メモリオブジェクトのページサイズ
- 各保護メモリプールのサイズ／4096

この総和がマイコンの TLB ページ数以下の場合は、最初のアクセスで TLB ミスが発生しますが、その後そのページが TLB からリプレースされることは無いため、その後のアクセスで TLB ミスが発生することはありません。

最初のアクセス時に TLB ミスを発生させたくない場合は、通常動作に移行する前に、初期化の一環として意図的に上記の全ページを一旦アクセスするようにすれば、その後の通常動作時に TLB ミスが発生しないようにすることができます。

(5) MMU を Enable にしない

デバッグが完了してメモリアクセス違反を起こす可能性が無い場合には、MMU が Disable の状態でカーネルを起動(vsta_knl サービスコール)することもできます。こうすることで、当然ながら TLB ミスは一切発生しなくなりますが、「4.21.6 不正アクセスの検出」に記載の MMU によって検出される例外も検出されなくなるので注意してください。

■関連ページ　MMU を Disable にするには　→　「4.21.10 MMU の初期化等」

4.21.8 アクセス許可の確認(prb_mem)

prb_mem サービスコールを使用することで、指定されたドメインが指定されたアドレスに対して指定されたアクセス(リードまたはライト)可能かどうかを確認することができます。

4.21.9 サービスコールのアドレスパラメータのエラーチェック

すべてのサービスコールのアドレスパラメータについて、prb_mem サービスコールと等価のエラーチェックを行うことができます。この機能を組み込むには、CFG_MEMCHK を選択します。

パラメータの正当性が確認済みであれば、この機能を組み込まないことで処理性能を向上させることができます。

4.21.10 MMU の初期化等

(1) MMUCR レジスタ

MMUCR の一部のビットは、vsta_knl(カーネル起動)で初期化されます。その他のビットは、必要に応じて vsta_knl 前にアプリケーションで初期化してください。

MMUCR は、vsta_knl 後に変更してはなりません。

- AT ビット：vsta_knl は初期化しません。vsta_knl 前にアプリケーションで適切に初期化してください。MMU を Enable にするには 1、Disable にするには 0 を設定してください。
- LRUI ビット：vsta_knl は 0 に初期化します。
- URB ビット：vsta_knl は 0 に初期化します。
- URC ビット：vsta_knl は 0 に初期化します。
- SQMD ビット(SH-4A のみ)：vsta_knl は初期化しません。vsta_knl 前にアプリケーションで適切に初期化してください。
- SV ビット：vsta_knl は 1(単一仮想記憶モード)に初期化します
- TI ビット：vsta_knl は 1 を書き込むことにより TLB の全エントリを無効化します
- ME ビット（CPU が拡張機能を持つ SH4AL-DSP, SH-4A の場合）：vsta_knl は 1 に初期化します。

(2) TTB, PTEH, PTEL レジスタ

これらのレジスタは、カーネルが使用します。vsta_knl(カーネル起動)後に変更してはなりません。

(3) PASCR, IRMCR レジスタ

システムの使用形態に応じて、vsta_knl 前にアプリケーションで適切に初期化してください。カーネルはこれらのレジスタを一切アクセスしません。

参考　サンプルシステムでは、samples¥***shnnnn***¥kernel¥knl_side¥init_mmu.c **でこれらを初期化しています**

4.22 保護メモリプール

保護メモリプールは、メモリオブジェクトを動的に割り当てるためのオブジェクトです。メモリオブジェクト保護機能を組み込まない場合は使用できません。

保護メモリプールから獲得した保護メモリブロックは、獲得したドメインに所属し、獲得したドメインのみがアクセスできるメモリオブジェクトになります。また、保護メールボックスに送信できるのは、保護メモリブロックのみです。

保護メモリプールは、コンフィギュレータでのみ生成できます。

保護メモリプールを操作するサービスコールとして以下があります。

- pget_mpp サービスコール
  指定したサイズの保護メモリブロックを獲得します。獲得できない場合(空きメモリが無い場合)はエラーが返ります。獲得されるブロックのサイズは、指定したサイズを CFG_PAGESZ(4096)で切上げた値になります。
- rel_mpp サービスコール
  保護メモリブロックを返却します。返却するタスクの所属ドメインと返却する保護メモリブロックの所属ドメインが同じ場合のみ返却できます。
- ref_mpp サービスコール
  保護メモリプールの状態を参照します。

■注意

保護メモリプールでは、空き領域が断片化する可能性があります。以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

図 4.20に保護メモリプールの動作例を示します。

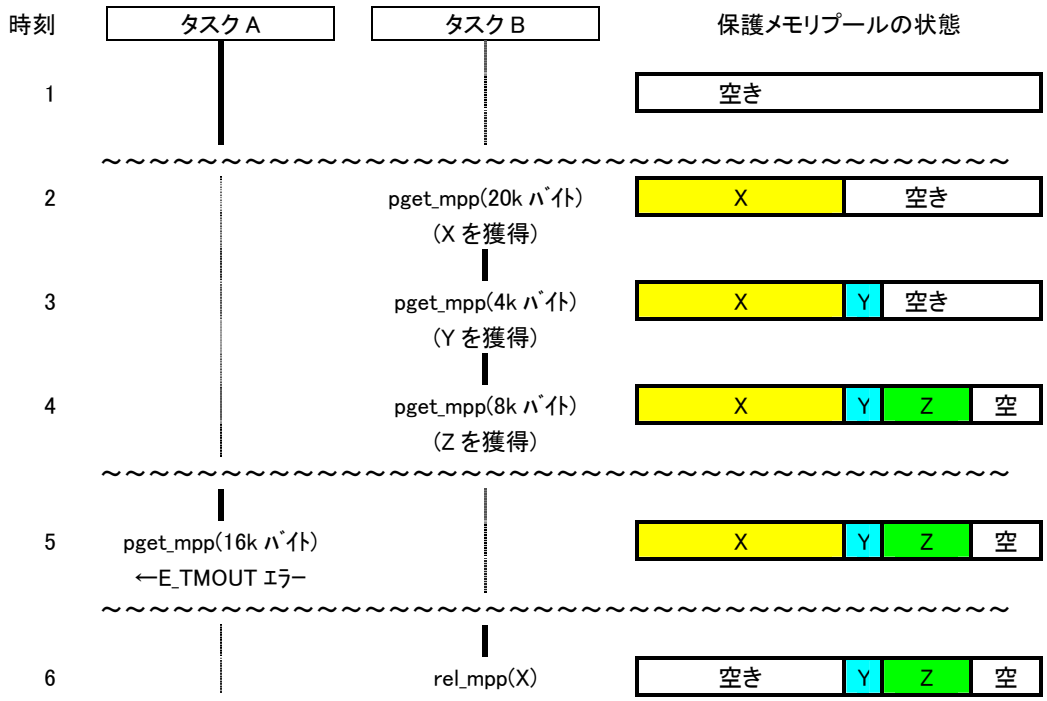


図4.20　保護メモリプールの動作例

図の解説

1. コンフィギュレータで36 k バイトの保護メモリプールを生成しておきます。
2. タスクBがpget_mppで20kバイトの保護メモリブロックXを獲得します。
3. タスクBがpget_mppで4kバイトの保護メモリブロックYを獲得します。
4. タスクBがpget_mppで8kバイトの保護メモリブロックZを獲得します。
5. タスクAがpget_mppで16kバイトの保護メモリブロックを獲得しようとしますが、空きが無いのでエラーE_TMOUTが返ります。
6. タスクBがrel_mppで20kバイトのブロックXを返却します。

4.23 保護メールボックス

保護メールボックスは、ドメイン間でメッセージの受け渡しを行うオブジェクトです。メモリオブジェクト保護機能を組み込まない場合は使用できません。

保護メールボックスでは、メッセージの先頭アドレスのみを受け渡すため、メッセージサイズに依存せずに高速に行われます。

メッセージとして使用できるのは、保護メモリプールから獲得した保護メモリブロックのみです。

保護メールボックスを生成するには、以下の方法があります。

- cre_mbp, icre_mbp サービスコール
  指定されたID番号で保護メールボックスを生成します。
- acre_mbp, iacre_mbp サービスコール
  カーネルが自動的にID番号を割り当てて、保護メールボックスを生成します。
- コンフィギュレータで生成
  ID名称を指定することができます。また、[カーネル側]をチェックしない場合は、コンフィギュレータがID番号を自動的に割り当てることができます。

また、保護メールボックスを操作するサービスコールとして以下があります。

- del_mbp サービスコール
  指定されたIDの保護メールボックスを削除します。
- snd_mbp サービスコール
  保護メールボックスに保護メモリブロックをメッセージとして送信します。送信するタスクの所属ドメインと送信する保護メモリブロックの所属ドメインが同じ場合のみ送信できます。
  保護メールボックスに送信された保護メモリブロックは、アクセス許可ベクタがTACT_KERNELに変更されます。
- rcv_mbp, trcv_mbp サービスコール
  保護メールボックスからメッセージを受信します。受信できない場合(保護メールボックスにメッセージが無い場合)は待ち状態になります。trcv_mbpでは、タイムアウト時間を指定することができます。
  受信したメッセージ(保護メモリブロック)は、所属ドメインが獲得したタスクの所属ドメイン、アクセス許可ベクタが獲得したタスクの所属ドメインのみがリード・ライト可能なように、それぞれ変更されます。
- prcv_mbp サービスコール
  保護メールボックスからメッセージを受信します。受信できない場合(保護メールボックスにメッセージが無い)はエラーが返ります。
  受信したメッセージ(保護メモリブロック)は、所属ドメインが獲得したタスクの所属ドメイン、アクセス許可ベクタが獲得したタスクの所属ドメインのみがリード・ライト可能なように、それぞれ変更されます。
- ref_mbp, iref_mbp サービスコール
  保護メールボックスの状態を参照します。

図 4.21に保護メールボックスの動作例を示します。

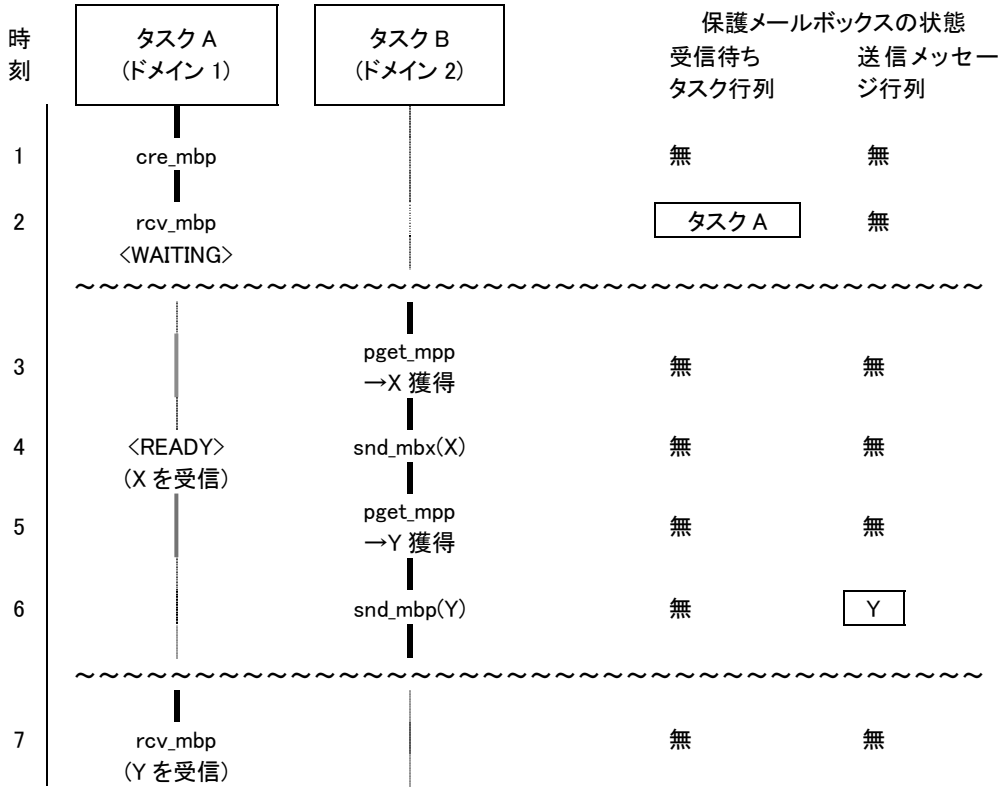


図4.21　保護メールボックスの動作例

図の解説

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

1. タスクAがcre_mbpで保護メールボックスを生成します。属性として、TA_TFIFO(受信待ちタスク行列はFIFO順)とTA_MFIFO(送信メッセージ行列はFIFO順)を指定します。
2. タスクAがrcv_mbpでメッセージを受信しようとしますが、保護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pget_mppで保護メモリブロックXを獲得し、このブロックにメッセージを作成します。
4. タスクBがsnd_mbpで保護メモリブロックXをメッセージとして送信すると、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。
5. タスクBがpget_mppで保護メモリブロックYを獲得し、このブロックにメッセージを作成します。
6. タスクBがsnd_mbpで保護メモリブロックYをメッセージとして送信します。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。
7. タスクAがrcv_mbp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。

4.24 システムメモリ管理機能

4.24.1 システムプール

システムプールとは、以下の領域をカーネルが割り付けるための領域です。

- ユーザドメインに所属するタスクのスタック領域
- 固定長メモリプール領域
- 可変長メモリプール領域

システムプールのサイズは、コンフィギュレータで CFG_SYSPOOLSZ によって指定します。

システムプールの空きが不足している場合は、サービスコールが E_NOMEM エラーとなります。

メモリオブジェクト保護機能を組み込んだ場合は、システムプールから割り付けられた領域はメモリオブジェクトとなります。また、割付サイズは要求サイズを CFG_PAGESZ(4kB)の倍数で切り上げた値となります。割り付けられた領域の先頭アドレスは、CFG_PAGESZ 境界にアライメントされます。

メモリオブジェクト保護機能を組み込まない場合は、割付サイズは要求サイズを 64 の倍数で切り上げた値となります。割り付けられた領域の先頭アドレスは、32 バイト境界にアライメントされます。

vref_syp サービスコールを使えば、現在のシステムプールの状態を確認することができます。

なお、カーネルがシステムプール内に管理テーブルを生成することはありません。

システムプールでは、空き領域が断片化する可能性があります。以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

また、システムプールサイズの算出方法については、以下を参照してください。

■関連ページ　「14. システムプールサイズの見積り」

4.24.2 リソースプール

リソースプールは、カーネル内部で動的に必要となる管理テーブル等を割り付けるための領域です。

リソースプールのサイズは、コンフィギュレータで CFG_RESPOOLSZ によって指定します。

カーネルは、サービスコール要求に応じてリソースプールから領域を獲得します。割付サイズは、要求サイズを20の倍数で切り上げた値となります

リソースプールの空きが不足している場合は、サービスコールが E_NOMEM エラーとなります。

vref_rsp サービスコールを使えば、現在のリソースプールの状態を確認することができます。

リソースプールでは、空き領域が断片化する可能性があります。以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

また、リソースプールサイズを要求するタイミングや算出方法については、以下を参照してください。

■関連ページ　「13. リソースプールサイズの見積り」

4.25 DSP スタンバイ機能

4.25.1 概要

本機能は、TA_COP0 属性の指定が無いプログラムを実行する際に、カーネルが自動的に X/Y メモリをマイコンが持つモジュールストップ状態にすることで省電力化を図ります。

また、TA_COP0 属性を動的に変更する vchg_cop サービスコールをサポートしています。vchg_cop を活用すればさらにこれらのハードウェアリソースをモジュールスタンバイにする期間を長くすることが可能となります。

図 4.22に、本機能の動作概要を示します。

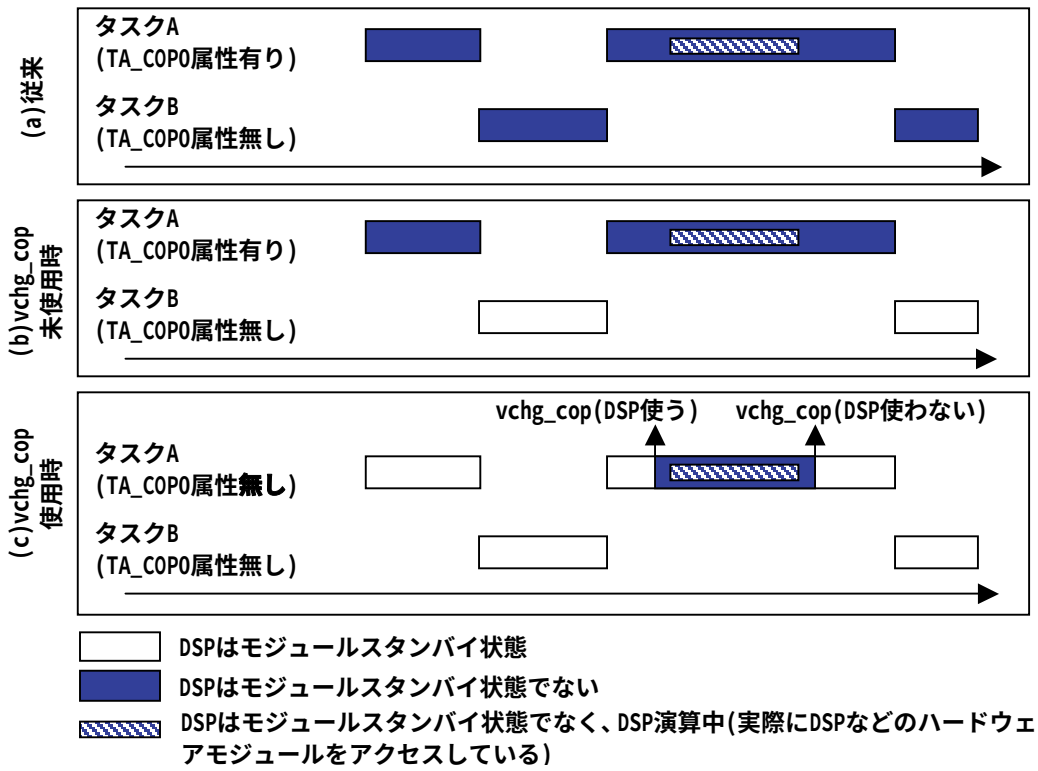


図4.22 動作概要

4.25.2 利用可能なマイコン

本機能は、DSP を内蔵し、以下の条件を満たすマイコンでのみ使用できます。

- X/Y メモリをモジュールストップするレジスタ長が 32 ビット

4.25.3 各プログラム起動時の X/Y メモリのモジュールストップ状態

本機能を組み込んだ場合、各プログラム開始時の X/Y メモリのモジュールストップ状態は、表 4.7 に示すようになります。

表 4.7で「不定」となっているプログラムでは、基本的には TA_COP0 属性に関連付けられた X/Y メモリをアクセスしてはなりません。X/Y メモリを使用するには、プログラムの起動時にアプリケーション側で X/Y メモリのモジュールストップ状態を保存してからモジュールストップを解除し、終了前に元に戻す必要があります。

表4.7　各プログラム起動時のモジュールスタンバイ状態

プログラム	モジュールストップ状態
タスク、タスク例外処理ルーチン、拡張サービスコールルーチン、トラップルーチン	TA_COP0 属性がある場合：非モジュールストップ TA_COP0 属性が無い場合：モジュールストップ
割込みハンドラ	不定(割込み発生前と同じ)
CPU 例外ハンドラ	不定(CPU 例外発生前と同じ)
タイムイベントハンドラ	不定
初期化ルーチン	不定

また、カーネルアイドリング時(実行可能状態のタスクが存在しないとき)は、X/Y メモリはモジュールストップ状態となります。

4.25.4 注意事項

例えば、TA_COP0 属性のタスク A が X/Y メモリをソース、あるいはデスティネーションとする DMA 転送をスタートさせた場合、その転送中に TA_COP0 属性でない別のタスク B にスイッチすると、その時点でカーネルは X/Y メモリをモジュールストップさせるため、DMA 転送が正常に行われなくなります。

あるいは、その転送中に割込みが発生し、割込みハンドラから TA_COP0 属性のない拡張サービスコールが呼び出されると、その時点でカーネルは X/Y メモリをモジュールストップさせるため、DMA 転送が正常に行われなくなります。

これは、X/Y メモリのスタンバイ制御がプログラムの実行と同期して行われることに対し、DMA による X/Y メモリへのアクセスはソフトウェア動作とは非同期に行われるためです。

これが問題となる場合は、本機能を使用しないようにするか、または以下が考えられます。

- X/Y メモリに対する DMA 転送は行わない。
- DMA 転送が終了するまで CPU ロック状態にする。

4.26 パフォーマンス管理機能

(1) 概要

パフォーマンス管理機能は、マイコンが内蔵しているプログラムパフォーマンスカウンタを使用して、実行時間や回数を計測する機能です。ただし、マイコンによってはプログラムパフォーマンスカウンタを内蔵していないものもあります。

本機能では、プログラムパフォーマンスカウンタのカウンタ 0 とカウンタ 1 を使用します。それぞれのカウンタは 32 ビットです。

パフォーマンス管理機能を組み込むには、CFG_PERFORM を選択します。

なお、本機能を使用するには、プログラムパフォーマンスカウンタに関する知識が必要です。詳細は、「SH-4A, SH4AL-DSP プログラムパフォーマンスカウンタ アプリケーションノート」などを参照してください。

(2) 測定項目(CFG_PPC0TYPE, CFG_PPC1TYPE)

どういう項目を測定するかは、コンフィギュレータの CFG_PPC0TYPE, CFG_PPC1TYPE で指定します。前者はカウンタ 0、後者はカウンタ 1 の設定になります。

ここには、それぞれプログラムパフォーマンスカウンタのカウント条件設定レジスタ(CCBR0, CCBR1)の CIT ビット(CIT9～CIT0 の計 10 ビット)への設定値を指定します。入力値は、H'3ff でマスクして扱われます。

例えば、0 を指定すると「経過サイクル数」という意味になります。測定結果が H'100000 で CPU の動作周波数が 266MHz の場合は、H'100000/266MHz=3942 μ sec ということになります。

(3) カウンタの連結

各カウンタは 32 ビットしかないので、すぐにオーバフローしてしまいます。このため、２つのカウンタを連結してひとつの 64 ビットカウンタとしても使用できるようになっています。カウンタを連結するには、CFG_CONNECT を選択してください。

ただし、連結した場合は、カーネルはカウンタの読み出し時に一時的にカウント動作を停止します。つまり、停止による誤差が累積されます。

(4) 測定対象

カーネルは、以下の分類でプログラムパフォーマンスカウンタの累積値を管理します。

- 各タスク(タスクから呼び出された拡張サービスコールルーチン、トラップルーチンを含む)
- カーネルアイドリング
- 上記以外(非タスクコンテキスト、カーネル)

ただし、メモリオブジェクト保護機能を組み込んだ場合のカーネルによる TLB 関連例外処理は、その例外を発生させたコンテキストとして取得されます。

(5) プログラムパフォーマンスカウンタの制御

カーネルは、vsta_knl でプログラムパフォーマンスカウンタのカウンタ 0、１を 0 に初期化し、カウント動作を開始します。その後、カウンタ動作は「(3) カウンタの連結」に示すケースを除いて停止しません。

(6) サービスコール

以下のサービスコールがあります。

- vchg_ppc：パフォーマンス測定の開始・停止・初期化
- vref_ppc：パフォーマンス測定結果の参照

(7) エミュレータ使用時の注意

エミュレータ使用時には、エミュレータ側でプログラムパフォーマンスカウンタを占有してしまうことがあります。カーネルのパフォーマンス機能を使用する場合は、エミュレータのマニュアルやヘルプを参照し、エミュレータがプログラムパフォーマンスカウンタを占有しないように設定してください。

例えば、E10A-USB エミュレータを使用する場合は、HEW の[Configuration]ダイアログボックスの[PPC mode]リストボックス、または PPC_MODE コマンドを用いて、「PPC をユーザに解放する」ように設定してください。

4.27 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とは、サービスコールの呼び出しや割込みの発生など、システムの動作履歴を保存する機能です。パフォーマンス管理機能を組み込んでいる場合は、プログラムパフォーマンスカウンタの値も取得されます。トレース結果は、デバッギングエクステンションを用いて参照することができます。

トレース機能を組み込むには、CFG_TRACE を選択します。

トレース機能の詳細は、デバッギングエクステンションのヘルプを参照してください。

(1) 取得タイミングと取得される情報

トレースは、以下のようなタイミングで取得されます。

- サービスコール呼び出し
- サービスコールからの復帰
- タスクやハンドラの実行開始
- タスクやハンドラの実行終了
- カーネルアイドリングへの遷移

取得される情報には、以下のようなものがあります。

- サービスコールのパラメータ
- サービスコールのエラーコード
- PC(プログラムカウンタ)
- プログラムパフォーマンスカウンタのカウンタ 0, 1(パフォーマンス管理機能使用時のみ)

(2) トレースのタイプ(CFG_TRCTYPE)

履歴の保存場所として、CFG_TRCTYPE でターゲット上の RAM に確保したバッファと、シミュレータやエミュレータのトレースメモリの、いずれかを選択できます。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。前者の場合は、CFG_TRCBUFSZ にバッファのサイズを指定します。

(3) オブジェクト取得数(CFG_TRCOBJCNT)

デバッギングエクステンションでは、ユーザが指定したオブジェクトの状態を、トレースを取得するタイミング毎に取得するように指定することができます。CFG_TRCOBJCNT には、同時に取得可能なオブジェクト数を指定します。

(4) ユーザイベントの取得(vget_trc, ivget_trc)

vget_trc, ivget_trc を使用すれば、ユーザ任意のタイミングで任意の情報を取得することができます。

4.28 その他の機能

(1) タスクの実行待ち行列を回転する (rot_rdq, irot_rdq)

本サービスコールを一定周期で呼び出す(例えば、周期ハンドラから呼び出す)ことで、TSS で必要なラウンドロビンスケジューリングを実現することができます。(図 4.23参照)

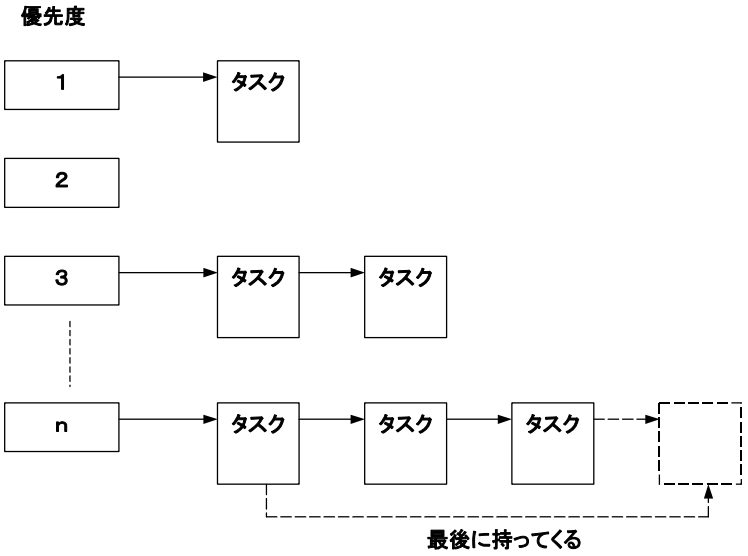


図4.23 rot_rdq サービスコールによるレディキューの操作

(2) 現在実行中のタスク ID を得る (get_tid, iget_tid)

現在実行中のタスク ID 番号を取得します。

(3) 現在実行中のタスクが所属するドメイン ID を得る (get_did, iget_did)

現在実行中のタスクが所属するドメイン ID を取得します。

拡張サービスコールルーチンとトラップルーチンはカーネルドメインに所属しますが、本サービスコールを使用すれば呼び出したタスクが所属するドメイン ID を知ることができます。

(4) コンフィギュレーション情報を参照する (ref_cfg, iref_cfg)

各種オブジェクトの最大 ID などの情報を参照します。

(5) カーネルのバージョン情報を参照する (ref_ver, iref_ver)

カーネルのバージョン情報を参照します。ref_ver で得られる情報の一部は、構成定数で得ることもできます。

■関連ページ 「6.1.5(1) 構成定数」

4.29 カーネルのアイドリング

実行可能状態のタスクが存在しなくなると、カーネルは内部で無限ループとなり、割込みが発生するのを待ちます。

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

4.30 CPU リセットとカーネルの起動

図 4.24に、CPU リセットからカーネル起動(vsta_knl)までの一般的な流れを示します。なお、vsta_knl はリターンすることはありません。

vsta_knl の呼び出しからマルチタスク環境に遷移するまでの流れは、以下を参照してください。

■関連ページ　「6.22.12 カーネルの起動(vsta_knl, ivsta_knl)」

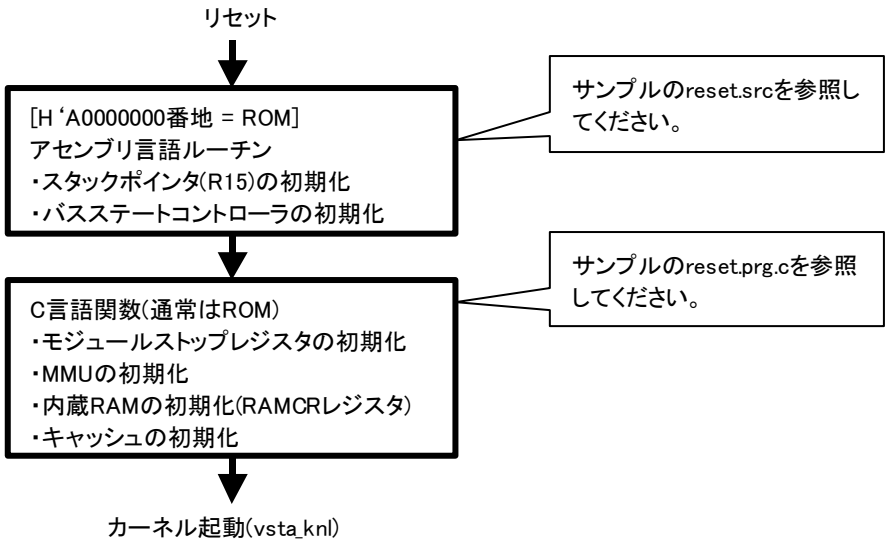


図4.24　CPU リセットからカーネル起動までの流れ

(1) バスステートコントローラの初期化

C 言語関数が実行するには、スタックや各種データセクションに使用する RAM 領域がアクセス可能でなければなりません。これらの領域に SDRAM などの初期化が必要な RAM を使用する場合は、C 言語関数が実行する前にバスステートコントローラ等の初期化を行う必要があります。

バスステートコントローラは、使用するターゲットボードの仕様にしたがって初期化してください。

(2) モジュールストップレジスタの初期化

システムで使用しないモジュールがあれば、モジュールストップにすることで、消費電力を低減することができます。逆に、システムで使用するモジュールは、モジュールストップを解除します。

メモリオブジェクト保護機能を使用する場合は、TLB のモジュールストップをここで解除してください。

(3) MMU の初期化

メモリオブジェクト保護機能を使用しない場合は、MMUCR を 0 に初期化してください。
メモリオブジェクト保護機能を使用する場合は、以下を参照してください。

■関連ページ
- 「6.22.12 カーネルの起動(vsta_knl, ivsta_knl)」
- 「4.21.10 MMU の初期化等」

また、32 ビットアドレス拡張モードを使用する場合は、以下を参照してください。

■関連ページ 「5.6 32 ビットアドレス拡張モード」

(4) RAMCR レジスタの RP, RMD ビットの初期化

これらのビットは、メモリオブジェクト保護機能を使用する場合は vsta_knl で初期化されます。
メモリオブジェクト保護機能を使用しない場合は、カーネルは一切初期化しません。以下を参照してください。

■関連ページ 「5.2.3 内蔵メモリ」

(5) キャッシュと RAMCR レジスタの IC2W, OC2W ビットの初期化

これらは、キャッシュサポート関数の sh4a_vini_cac()などを呼び出すことで初期化します。
以下を参照してください。

(6) 割込みの禁止と CPU 例外の抑止

カーネルによる割込み・CPU 例外のハンドリングは、カーネルを起動して始めてその準備が整います。このため、通常はカーネルを起動するまでは全ての割込みを禁止し、さらに CPU 例外を発生させないようにしてください。
全ての割込みを禁止するには、SR.BL=1 にしてください。CPU のリセット直後はこの状態になります。
メモリオブジェクト保護機能を使用する場合は、通常はカーネル起動前にアプリケーション側で MMU を Enable にする必要がありますが、この場合も CPU 例外(TLB 関連例外)を発生させないようにするために、MMU 対象領域をアクセスしないようにしてください。
なお、SR.BL=1 の時に CPU 例外が発生すると、CPU はリセットベクタに分岐します。

また、セクション初期化と標準ライブラリ関数の初期化については、以下を参照してください。

■関連ページ 「11.7 標準ライブラリ関数と実行時ルーチン」

4.31 メモリの断片化とその対策（VTA_UNFRAGMENT 属性）

以下の領域は、空き領域が断片化することがあります。

- 可変長メモリプール
- 保護メモリプール
- システムプール
- リソースプール

これらの領域からメモリの獲得と返却を繰り返していると、空き領域の断片化が発生し、空き領域のトータルサイズは十分でも、連続した空き領域が存在しない、つまり大きなメモリを獲得できない状況になることがあります(図 4.25)。

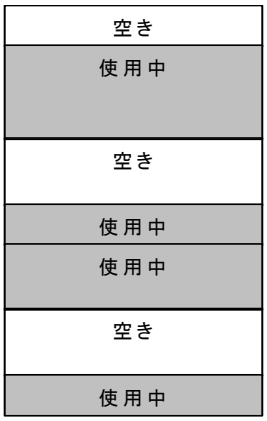


図4.25 空き領域の断片化

本カーネルでは、この問題をある程度抑制するための「セクタ管理方式」をサポートしています。可変長メモリプールおよび保護メモリプールの VTA_UNFRAGMENT 属性がこれに該当します。システムプールとリソースプールも同様の方式を採用しています。

セクタ管理方式は、大容量のメモリプールから多数の微小なブロックと大きなブロックを獲得するような使い方をする場合に、断片化を発生しにくくする管理方式です。断片化度合いは、一般には VTA_UNFRAGMENT を指定したほうが良くなる傾向がありますが、メモリプールの利用方法に依存します。

セクタ管理方式では、「最小ブロックサイズ」×8 バイトまでのブロックを「微小なブロック」として扱います。また、獲得要求サイズは表 4.8に示すように切り上げて扱います。

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

表4.8 微小ブロックの扱い

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

【注】 blksz：**要求サイズ**、minblksz：**最小ブロックサイズ**

そして、カーネルはセクタ内のメモリブロックを割り当てます。セクタ内の残りのブロックは、以降のそのブロックサイズ以下のメモリブロック獲得要求のために予約されることになります。

このように、微小なブロックを連続して使用されるようにすることで、より大きなサイズの連続空き領域が維持されやすくなっています.

図 4.26に、「最小ブロックサイズ」が 32 の可変長メモリプールの例を示します。

最初に 32 バイトのメモリブロック獲得要求があると、32×32＝1024 バイトのセクタ[A]を確保し、そのセクタ内の 32 バイトの領域[A-1]を割り付けます(図 4.26 (1))。次に 16 バイトの獲得要求があると、A の中の 32 バイトの領域(A-2)を割り付けます(図 4.26 (2))。

さらに、今度は 36 バイトのメモリブロック獲得要求があったとします。この場合、セクタ A の各ブロックは 32 バイトなので、この要求には対応できません。そこで、要求サイズの 36 を最小ブロックサイズで切上げた 64 バイトのブロックが 16 個で構成される新たなセクタ B(64×16=1024 バイト)を確保し、そのセクタ内の 64 バイトの領域[B-1]を割り付けます(図 4.26 (3))。

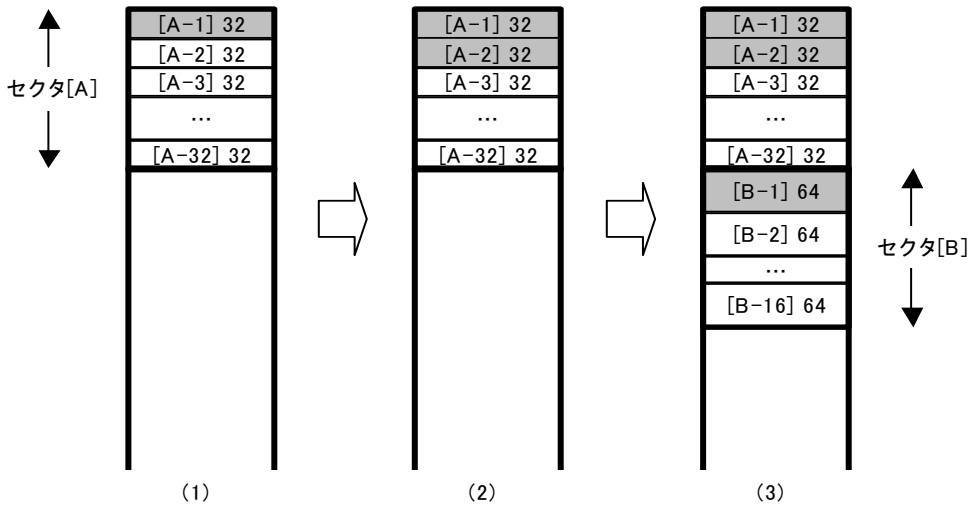


図4.26　可変長メモリプールの例

最小ブロックサイズと最大セクタ数は、プールの種類によって表 4.9に示すように異なります.

最大セクタ数を使いきってしまっている場合、またはセクタサイズに見合う連続空き領域がない場合には、セクタを生成せずに要求サイズのメモリブロックを獲得します。つまり、この場合には、断片化が発生しやすくなります。

セクタ内の全てのブロックが解放されると、セクタそのものも解放されます。

表4.9　各プールの最小ブロックサイズと最大セクタ数

各種プール		最小ブロックサイズ	最大セクタ数
可変長メモリプール		生成時に指定	生成時に指定
保護メモリプール		生成時に指定	生成時に指定
システムプール	メモリオブジェクト保護機能使用時	CFG_PAGESZ(=4kB)	コンフィギュレータで設定 (CFG_SYSPOOLSCTNUM)
	メモリオブジェクト保護機能未使用時	64	
リソースプール		20	オンデマンドに可能な限りセクタを生成しす。

4.32 デバッギングエクステンション

デバッギングエクステンションは、HEW にマルチタスクデバッグ機能を付加するソフトウェアで、弊社ホームページからダウンロードできます。

デバッギングエクステンションには、以下のような機能があります。詳細は、デバッギングエクステンションのマニュアルまたはヘルプを参照してください。

(1) 各種オブジェクトの状態参照

タスクやセマフォなど、各種オブジェクトの状態を表示します。

(2) 各種オブジェクト状態の変更(サービスコールの発行)

タスクを起動したり、イベントフラグをセットしたりすることができます。コンフィギュレータで CFG_ACTION を選択した場合のみ使用できます。

(3) トレース結果の表示

コンフィギュレータで CFG_TRACE を選択した場合のみ使用できます。

5. 論理アドレス空間の扱い

5.1 概要

本カーネルでは、メモリオブジェクト保護機能の使用に関わらず、常に論理アドレス＝物理アドレスと扱います。

また、本カーネルでは、基本的にすべてのコード・データ領域はリンク時のセクション配置によって論理アドレスを決定します。各セクションは、その意味によって適切な論理アドレスに配置しなければなりません。そのためには、本章の理解が必要です。

本章では、実際の外部メモリや内蔵メモリのアドレスがどのような扱いとなるのかを解説します。

5.2 メモリオブジェクト保護機能を使用しない場合

5.2.1 概要

メモリオブジェクト保護機能を使用しない場合は、特権モード(カーネルドメイン)とユーザモード(ユーザドメイン)の区別による保護ができます。

SH4AL-DSP または SH-4A CPU の場合は、表 5.1に示す例外を検出できます。太字が特に保護に関する検出機能です。詳細は、使用するマイコンのマニュアルを参照してください。

表5.1 CPU による保護機能に関する例外検出

例外コード(EXPEVT)	例外発生条件
H'0E0 *2	(1) 命令アドレスエラー：以下のいずれかの場合に検出されます。 **・ユーザモードの時に、H'80000000 以上のアドレスから命令フェッチ *1** ・奇数アドレスから命令フェッチ (2) データアドレスエラー(リード)：以下のいずれかの場合に検出されます。 **・ユーザモードの時に、H'80000000 以上のアドレスからデータリード *1** ・アライメント不正のリードアクセス
H'100 *2	データアドレスエラー(ライト)：以下のいずれかの場合に検出されます。 **・ユーザモードの時に、H'80000000 以上のアドレスへライト *1** ・アライメント不正のライトアクセス
H'180 *2	一般不当命令例外：以下のいずれかの場合に検出されます。 **・ユーザモードの時に、遅延スロット以外にある特権命令をデコード** **・遅延スロット以外にある未定義命令をデコード** ・SR.DSP=0 の時に、DSP 命令を実行(SH4AL-DSP のみ)
H'1A0 *2	スロット不当命令例外：以下のいずれかの場合に検出されます。 **・ユーザモードの時に、遅延スロットにある特権命令をデコード** ・遅延スロットにある未定義命令をデコード ・遅延スロットにある PC を書き換える命令をデコード ・遅延スロットにある PC 相対 MOV 命令、MOVA 命令をデコード
H'800	一般 FPU 抑止例外(SH-4A のみ)： ・SR.FD=1 の時に、遅延スロット以外にある FPU 命令をデコード
H'820	スロット FPU 抑止例外(SH-4A のみ)： ・SR.FD=1 の時に、遅延スロットにある FPU 命令をデコード

【注】 *1 内蔵メモリはアクセスできる場合があります。詳細は「5.2.3 内蔵メモリ」を参照してください。

*2 サンプルシステムでは、これらの例外コードに対して CPU 例外ハンドラ(samples¥sysapp¥cpuexc.c)を定義しています。

5.2.2 外部メモリ

外部メモリの論理アドレス空間は表 5.2に示すように分類されます。

表5.2 外部メモリアドレス空間の扱い

領域(アドレス範囲)	ユーザモードからのアクセス	キャッシュ Enable 時の動作		備考
		リード	ライト	
P0/U0 領域 (0～H'7fffffff)	可能	キャッシャブル	キャッシャブル *1	
P1 領域 (H'80000000～H'9fffffff)	不可	キャッシャブル	キャッシャブル *2	
P2 領域 (H'a0000000～H'bfffffff)	不可	非キャッシャブル	非キャッシャブル	
P3 領域 (H'c0000000～H'dfffffff)	不可	－	－	カーネル仕様として、P3 領域は使用禁止です。
P4 領域 (H'e0000000～H'ffffffff)	不可	非キャッシャブル	非キャッシャブル	マイコン内部リソースにマッピングされる領域です。

【注】 *1 キャッシュへの書き込みモードは、CCR レジスタの WT ビットが 1 の場合(vini_cac で TCAC_P0_WT を指定した場合)はライトスルーモード、そうでない場合はコピーバックモードとなります。

*2 キャッシュへの書き込みモードは、CCR レジスタの CB ビットが 1 の場合(vini_cac で TCAC_P1_CB を指定した場合)はコピーバックモード、そうでない場合はライトスルーモードとなります。

5.2.3 内蔵メモリ

(1) 内蔵メモリが割り当てられている論理アドレスの扱い

内蔵メモリの論理アドレスは、P2 領域または P4 領域にマッピングされていますが、その特性は外部メモリの P2,P4 領域とは異なり、RAMCR レジスタの RMD ビットの設定によって特性が決まります。

表 5.3にその詳細を示しますが、内蔵メモリは常に非キャッシャブルアクセスとなります。RAMCR.RMD は、表 5.3を参考に、アプリケーションが適切に初期化しなければなりません。

表5.3 内蔵メモリ論理(仮想)アドレスの扱い

RAMCR.RMD の値	ユーザモードからのアクセス	キャッシュ Enable 時の動作	
		リード	ライト
0	不可 *	非キャッシャブル	非キャッシャブル
1	可能	非キャッシャブル	非キャッシャブル

【注】SR.DSP=1 の場合は、ユーザドメインでもアクセス可能になります。

(2) キャッシャブルアクセスするには

内蔵メモリをキャッシャブルアクセするには、以下の方法があります。

なお、X/Y メモリ, L メモリなど、キャッシャブルアクセスしてはならない内蔵メモリもあるので注意してください。

(a) 物理アドレスがエリア 1 に割り当てられている内蔵メモリ

一部のマイコンでは、物理アドレスがエリア 1 に割り当てられている内蔵メモリを持っています。このような内蔵メモリでは、エリア 1(P0/U0 領域)またはその P1 領域のシャドウアドレスを用いてアクセスすることができます。この場合の特性は、外部メモリの場合と同じです。

(b) 32 ビットアドレス拡張モードを使用する

32 ビットアドレス拡張モードをサポートしているマイコンでは、PMB を適切に設定することで、P1, P2 領域の論理アドレスに、内蔵メモリの物理アドレスを割り当てることができます。PMB の C, WT ビットによって、キャシュの振る舞いを制御することができます。ただし、P1, P2 領域は、ユーザモードからはアクセスできません。

5.3 メモリオブジェクト保護機能を使用する場合

5.3.1 概要

(1) MMU 対象領域と MMU 非対象領域

全論理アドレス空間は、MMU 対象領域と MMU 非対象領域のいずれかに分類されます。

(a) MMU 非対象領域

MMU 非対象領域は、MMU を介さずにアクセスされる領域です。通常は、特権モードからのみアクセスできます。ただし、内蔵メモリはコンフィギュレータの CFG_IRAMUSAGE の設定によってはユーザモードからもアクセスできます。

(b) MMU 対象領域

MMU 対象領域とは、アクセスが MMU によって検査される領域です。静的メモリオブジェクトなど、メモリオブジェクトとして使用する領域は、MMU 対象領域のアドレスに配置しなければなりません。

(2) 不正アクセスの検出

メモリオブジェクト保護機能使用時は、CPU と MMU による 2 種類の保護機能が働きます。

CPU による保護は、メモリオブジェクト保護機能を使用しない場合と全く同じです。「5.2.1 概要」を参照してください。

MMU は、MMU 対象領域に対する以下の不正アクセスを検出します。

(1) アクセスしたアドレスにメモリオブジェクトが存在しない。
(2) TA_RO属性のメモリオブジェクトに対して、ライトアクセスを行った。
(3) メモリオブジェクトに対して、そのメモリオブジェクトのアクセス許可ベクタでは許可されていないユーザドメインからアクセスした。

これを実現するために、カーネルは以下の CPU 例外をハンドリングします。これらの例外コードに対して CPU 例外ハンドラを定義しても無視されます。

- EXPEVT=H'040：命令 TLB ミス例外、データ TLB ミス例外(リード)
- EXPEVT=H'060：データ TLB ミス例外(ライト)
- EXPEVT=H'0A0：命令 TLB 保護違反例外、データ TLB 保護違反例外(リード)
- EXPEVT=H'0C0：データ TLB 保護違反例外(ライト)

不正アクセスが行われた場合は、メモリアクセス違反ハンドラ(samples¥sysapp¥mavhdr.c)が起動されます。メモリアクセス違反ハンドラは、必ず作成し、組み込まなければなりません。

なお、MMU のハードウェア機能には、以下のものがありますが、本カーネルでは MMU をアクセス保護とキャッシュ制御の目的でのみ使用します。アドレス変換は行いません。

- アドレス変換
- アクセス保護
- キャッシュ制御

■注意	「8.9 メモリアクセス違反ハンドラ」

5.3.2 外部メモリ空間

外部メモリの論理アドレス空間は表 5.4に示すように分類されます。

表5.4 外部メモリアドレス空間の扱い

領域(アドレス範囲)	MMU 対象/ 非対象	ユーザモードからのアクセス	キャッシュ Enable 時の動作		備考
			リード	ライト	
P0/U0 領域 (0～H'7fffffff)	MMU 対象	メモリオブジェクトのアクセス許可ベクタに従う	メモリオブジェクト属性に従う		
P1 領域 (H'80000000～H'9fffffff)	MMU 非対象	不可	キャッシャブル	キャッシャブル *1	
P2 領域 (H'a0000000～H'bfffffff)	MMU 非対象	不可	非キャッシャブル	非キャッシャブル	
P3 領域 (H'c0000000～H'dfffffff)	－	不可	－	－	カーネル仕様として、P3 領域は使用禁止です。
P4 領域 (H'e0000000～H'ffffffff)	MMU 非対象	不可	非キャッシャブル	非キャッシャブル	マイコン内部リソースにマッピングされる領域です。

【注】 *1 キャッシュへの書き込みモードは、CCR レジスタの CB ビットが 1 の場合(vini_cac で TCAC_P1_CB を指定した場合)はコピーバックモード、そうでない場合はライトスルーモードとなります。

5.3.3 内蔵メモリ

(1) 内蔵メモリが割り当てられている論理アドレスの扱い

内蔵メモリの論理アドレスは、コンフィギュレータの設定によってその特性が決まります。カーネルは、コンフィギュレータの設定に従って、vsta_knl で RAMCR の RP,RMD ビットを初期化します。

表 5.5にその詳細を示しますが、内蔵メモリは常に非キャッシャブルアクセスとなります。

表5.5 内蔵メモリ論理(仮想)アドレスの扱いの vsta_knl での RAMCR 初期化

コンフィギュレータ設定 CFG_IRAMUSAGE	MMU 対象/非対象	ユーザモードからのアクセス	キャッシュ Enable 時の動作		vsta_knl での RAMCR 初期化
			リード	ライト	
(1)MMU 非対象領域、全ﾓｰﾄﾞからｱｸｾｽ可能	非対象	可能	非キャッシャブル	非キャッシャブル	RP=0 RMD=1
(2)MMU 非対象領域、ﾕｰｻﾞ(非 DSP)ﾓｰﾄﾞではｱｸｾｽ不可	非対象	不可 *1	非キャッシャブル	非キャッシャブル	RP=0 RMD=0
(3)MMU 対象領域	対象	メモリオブジェクトのアクセス許可ベクタに従う	非キャッシャブル *2	非キャッシャブル *2	RP=1 RMD=1

【注】 *1 SR.DSP=1 の場合は、ユーザドメインでもアクセス可能になります。

*2 メモリオブジェクト属性に関係なく、非キャッシャブルとなります。

(2) キャッシャブルアクセスするには

内蔵メモリをキャッシャブルアクセスするには、以下の方法があります。

なお、X/Y メモリなど、キャッシャブルアクセスしてはならない内蔵メモリもあるので注意してください。

(a) 物理アドレスがエリア 1 に割り当てられている内蔵メモリ

一部のマイコンでは、物理アドレスがエリア 1 に割り当てられている内蔵メモリを持っています。このような内蔵メモリでは、エリア 1(P0/U0 領域)またはその P1 領域のシャドウアドレスを用いてアクセスすることができます。この場合の特性は、外部メモリの場合と同じです。

(b) 32 ビットアドレス拡張モードを使用する

32 ビットアドレス拡張モードをサポートしているマイコンでは、PMB を適切に設定することで、P1, P2 領域の論理アドレスに、内蔵メモリの物理アドレスを割り当てることができます。PMB の C, WT ビットによって、キャシュの振る舞いを制御することができます。ただし、P1, P2 領域は、ユーザモードからはアクセスできません。

5.3.4 注意事項

MMU 対象領域をアクセスするときには、TLB ミス例外が発生する可能性があります。SR レジスタの BL ビットが 1 の時に例外が発生すると、CPU はリセットしてしまうため、BL=1 の時には MMU 対象領域をアクセスしてはなりません。もちろん、アクセスするデータだけでなく、そのプログラムも MMU 対象領域に配置してはなりません。

5.4 P4 領域に割り当てられている内部リソース

表 5.2に示すように、P4 領域はユーザドメインからアクセスすることはできません。

ユーザドメインから P4 領域に割り当てられている内部リソースをアクセスする唯一の方法は、メモリオブジェクト保護機能を使用して、静的メモリオブジェクトとして登録することです。

5.5 物理アドレスがエリア 1 に割り当てられている内部リソース

通常は、物理アドレスの先頭 3 ビットを B'101 として、P2 領域としてアクセスしてください。表 5.2に示すように、P2 領域はユーザドメインからアクセスすることはできません。

ユーザモードからアクセスする唯一の方法は、メモリオブジェクト保護機能を使用して、静的メモリオブジェクトとして登録することです。

5.6 32 ビットアドレス拡張モード

一部のマイコンでは、32 ビットアドレス拡張モードをサポートしています。32 ビットアドレス拡張モードを使用する場合は、カーネル起動前に以下の手順で初期化してください。サンプルの samples¥sh7780¥kernel¥knl_side¥7780¥init_mmu.c も参考にしてください。

1. PMBを設定します。
2. PASCRレジスタのSEビットをONします。
3. 必要なら、MMUをEnableにします。

なお、32 ビットアドレスモードを使用する場合、P1, P2 領域の特性は、PMB の設定に依存することになるので、注意してください。

6. サービスコール

6.1 C 言語 API

6.1.1 呼び出し形式

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);
```

6.1.2 ヘッダファイル

(1) include¥itron.h

基本データ型などが定義されています。

(2) include¥kernel.h

本カーネル仕様に関連する定義が記述されています。kernel.h は、itron.h および表 6.1に示すファイルをインクルードしています。

表6.1 kernel.h からインクルードされるファイル

ファイル名	説明
include¥kernel_api.h	カーネルのサービスコール定義が記述されています。
include¥kernel_tsz.h	メモリサイズに関するマクロが定義されています。
kernel_macro.h	カーネルのマクロ定義が記述されています。このファイルは、コンフィギュレータによって生成されます。

6.1.3 コンフィギュレータが出力するヘッダファイル

コンフィギュレータは、kernel_macro.h、kernel_id.h、kenrel_id_sys.h の 3 つのヘッダファイルを出力します。

(1) kernel_macro.h

kernel_macro.h には、コンフィギュレータの設定情報の一部が define 文として出力されます。内容については、表 6.3を参照してください。

kernel_macro.h は、kernel.h からインクルードされています。アプリケーションから直接インクルードする必要はありません。

なお、kernel_macro.h はコンフィギュレータが「カーネルロックモード」の時には生成されません。

(2) kernel_id.h と kernel_id_sys.h

これらのファイルには、コンフィギュレータで指定した ID 名称に対する define 文が出力されます。

コンフィギュレータの[カーネル側]チェックボックスをチェックして生成したオブジェクトの ID 名称は kernel_id_sys.h に、チェックせずに生成したオブジェクトの ID 名称は kernel_id.h に出力されます。

なお、kernel_id_sys.h はコンフィギュレータが「カーネルロックモード」の時には生成されません。

これらのファイルは、必要に応じてアプリケーションが明示的にインクルードしてください。なお、

カーネル側のファイルでは、kernel_id.h をインクルードしてはなりません。

6.1.4 基本データ型

基本データ型を以下に示します。これらは、itron.h で定義されています。

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

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

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

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

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

typedef INT             BOOL;           /* Bool value                       */

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

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

typedef struct {                            /* system clock                 */
            UH          utime;              /* current date/time (upper)    */
            VH          _Hrsv1;             /* reserved                     */
            UW          ltime;              /* current date/time (lower)    */
        } SYSTIM;

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

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

各サービスコールで使用する構造体型については、各サービスコールの節で説明しています。

6.1.5 定数とマクロ

(1) 構成定数

構成定数は、カーネルの構成情報を得るためのものです。

構成定数は、カーネル仕様として静的に定まっているものと、コンフィギュレータの設定によって決まるものがあります。前者の定義箇所は kernel.h、後者の定義箇所はコンフィギュレータが出力する kernel_macro.h です。

表6.2 kernel.h で定義されている構成定数

項番	マクロ名	定義内容	説明
1	TMIN_TPRI	1	タスク優先度の最小値
2	TMIN_MPRI	1	メッセージ優先度の最小値
3	TKERNEL_MAKER	H'0115	カーネルのメーカーコード ref_ver で返る maker と同じです。
4	TKERNEL_PRID	H'0012	カーネルの識別番号 ref_ver で返る prid と同じです。
5	TKERNEL_SPVER	H'5402	準拠する ITRON 仕様のバージョン番号 ref_ver で返る spver と同じです。
6	TKERNEL_PRVER	H'0100	カーネルのバージョン番号 ref_ver で返る prver と同じです。
7	TKERNEL_PXVER	H'0100	本カーネルが準拠するμITRON4.0 仕様保護機能拡張仕様のバージョン番号 ref_ver で返る pxver と同じです。
8	TBIT_TEXPTN	32	タスク例外要因のビット数
9	TBIT_FLGPTN	32	イベントフラグのビット数
10	TMAX_MAXSEM	65535	セマフォの最大資源数の最大値

表6.3 kernel_macro.h で定義されている構成定数

項番	マクロ名	コンフィギュレータ関連項目	説明
1	TIC_NUME	CFG_TICNUME	タイムティックの周期の分子
2	TIC_DENO	CFG_TICDENO	タイムティックの周期の分母
3	TMAX_TPRI	CFG_MAXTSKPRI	タスク優先度の最大値
4	TMAX_MPRI	CFG_MAXMSGPRI	メッセージ優先度の最大値
5	TMAX_ACTCNT	CFG_MAXACTCNT	タスクの起動要求キューイング数の最大値
6	TMAX_WUPCNT	CFG_MAXWUPCNT	タスクの起床要求キューイング数の最大値
7	TMAX_SUSCNT	CFG_MAXSUSCNT	タスクの強制待ち要求ネスト数の最大値
8	VTCFG_PROTMEM	CFG_PROTMEM	メモリオブジェクト保護機能組込みの有無。 CFG_PROTMEM チェック時は 1、未チェック時は 0
9	VTCFG_PAGESZ	CFG_PAGESZ	デフォルトの MMU ページサイズ メモリオブジェクト保護機能組込み時は常に 4096。非組込み時は常に 0。
10	VTCFG_TMRCLOCK	CFG_TMRCLOCK	タイマデバイスに入力される周波数
11	VTCFG_TIMINTNO	CFG_TIMINTNO	標準タイマドライバ割込み番号
12	VTCFG_KNLLVL	CFG_KNLLVL	カーネル割込みマスクレベル、兼タイマ割込みレベル

(2) エラーコード

サービスコールで返るエラーコードは itron.h で定義されています。

■関連ページ　「17.2 サービスコールエラーコード一覧」

エラーコード用のマクロとして、以下があります。

表6.4　エラーコード操作マクロ(itron.h)

項番	マクロ名	説明
1	ER ercd = MERCD(ER ercd)	ercd のメインエラーコードを返す。
2	ER ercd = SERCD(ER ercd)	ercd のサブエラーコードを返す。 *

【注】 カーネルのサービスコールが返すエラーは、全てサブエラーコードは-1 です。

(3) サイズに関するマクロ

サイズ(バイト数)を求めるマクロとして、以下のものがあります。

- アプリケーション側で指定する領域サイズを求めるためのマクロ(表 6.5)
- メッセージバッファの消費サイズを求めるマクロ(表 6.6)
- リソースプールの消費サイズを求めるマクロ

リソースプールの消費サイズを求めるマクロについては、下記を参照してください。

■関連ページ　「13. リソースプールサイズの見積り」

表6.5　アプリケーション側で指定する領域サイズを求めるためのマクロ(kernel_tsz.h)

構成マクロ	説明
SIZE mbfsz = TSZ_MBFMB (UINT msgcnt, UINT msgsz)	msgsz バイトのメッセージを msgcnt 個格納することのできるメッセージバッファ領域の目安のサイズ。 このマクロは、cre_mbf, acre_mbf で指定する mbfsz の目安を知るためのマクロです。
SIZE mbfsz = TSZ_MBF (UINT msgcnt, UINT msgsz)	TSZ_MBFMB と同じです
SIZE mpfsz = TSZ_MPF (UINT blkcnt, UINT blksz)	blksz バイトのメモリブロックを blkcnt 個で獲得できる固定長メモリプール領域のサイズ。 このマクロは、固定長メモリプール領域をアプリケーション側で確保する場合に、そのサイズを算出するためのマクロです。
SIZE mplsz = TSZ_MPL (UINT blkcnt, UINT blksz)	blksz バイトのメモリブロックを blkcnt 個で獲得できる可変長メモリプール領域の目安のサイズ。 このマクロは、可変長メモリプール領域をアプリケーション側で確保する際、その目安のサイズを算出するためのマクロです。
SIZE mplsz = TSZ_MPP (UINT blkcnt, UINT memsz)	memsz バイトの保護メモリブロックを blkcnt 個で獲得できる保護メモリプール領域の目安のサイズ。

表6.6　メッセージバッファの消費サイズを求めるマクロ(kernel_tsz.h)

構成マクロ	説明
SIZE size = VTSZ_MBFMSGMB(UINT msgsz)	msgsz バイトのメッセージをメッセージバッファに格納する時に消費されるメッセージバッファ領域のサイズ

(4) アライメントチェックマクロ

データのアライメントをチェックするための以下のマクロがあります。

表6.7　アライメントのチェックマクロ (itron.h)

項番	マクロ名	説明
1	BOOL align = ALIGN_VB(VP addr)	addr で指定されたアドレスに対し、VB 型のデータアクセスが可能なアライメントになっている場合は TRUE(1)、そうでない場合は FALSE(0)を返す。
2	BOOL align = ALIGN_VH(VP addr)	addr で指定されたアドレスに対し、VH 型のデータアクセスが可能なアライメントになっている場合は TRUE(1)、そうでない場合は FALSE(0)を返す。
3	BOOL align = ALIGN_VW(VP addr)	addr で指定されたアドレスに対し、VW 型のデータアクセスが可能なアライメントになっている場合は TRUE(1)、そうでない場合は FALSE(0)を返す。
4	BOOL align = ALIGN_VD(VP addr)	addr で指定されたアドレスに対し、VD 型のデータアクセスが可能なアライメントになっている場合は TRUE(1)、そうでない場合は FALSE(0)を返す。
5	BOOL align = ALIGN_VP(VP addr)	addr で指定されたアドレスに対し、VP 型のデータアクセスが可能なアライメントになっている場合は TRUE(1)、そうでない場合は FALSE(0)を返す。

(5) その他の定数・マクロ

その他の定数、マクロについては、各サービスコールの節で説明しています。

6.2 サービスコール呼び出し前後のレジスタ保証規則

呼び出し元に戻るサービスコールについては、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、基本的にはルネサス C コンパイラに準じていますが、その詳細を表 6.8に示します。

表6.8 サービスコール発行前後のレジスタ保証

項番	レジスタ	レジスタ保証
1	SR, R8～R15, PR, GBR, MACH, MACL	保証されます。ただし、chg_ims, ichg_ims, loc_cpu, iloc_cpu, unl_cpu, iunl_cpu の場合は、SR.IMASK は更新されます。
2	R0	正常終了(E_OK)またはエラーコードが設定されます。
3	R1～R7	保証されません。
4	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	以下のいずれかの場合のみ保証されます。 ・TA_COP0 属性を持つプログラムからの呼び出し ・ディスパッチ保留状態からの呼び出し *
5	[FPU] FPR0～FPR11_BANK0	(1)FPSCR.FR=0 の場合 ディスパッチ保留状態から呼び出した場合のみ保証されます。 * (2)FPSCR.FR=1 の場合 以下のいずれかの場合のみ保証されます。 ・TA_COP1|TA_COP2 属性を持つプログラムからの呼び出し ・ディスパッチ保留状態からの呼び出し *
6	[FPU] FPR12～FPR15_BANK0, FPSCR, FPUL	以下のいずれかの場合のみ保証されます。 ・TA_COP1 属性を持つプログラムからの呼び出し ・ディスパッチ保留状態からの呼び出し *
7	[FPU] FPR0～FPR11_BANK1,	(1)FPSCR.FR=0 の場合 以下に示す状態から発行した場合のみ保証されます。 ・TA_COP1|TA_COP2 属性を持つプログラムからの呼び出し ・ディスパッチ保留状態からの呼び出し * (2)FPSCR.FR=1 の場合 ディスパッチ保留状態から呼び出した場合のみ保証されます。 *
8	[FPU] FPR12～FPR15_BANK1,	以下に示す状態から発行した場合のみ保証されます。 ・TA_COP1|TA_COP2 属性を持つプログラムからの呼び出し ・ディスパッチ保留状態からの呼び出し *

【注】 この場合、カーネルのサービスコール処理でこれらのレジスタは一切アクセスしません。

6.3 サービスコールのリターン値とエラーコード

6.3.1 概要

リターン値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 型のリターン値を持つサービスコールはこの限りではありません。正常終了時のリターン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、本カーネルではすべてのエラーコードのサブエラーコードは-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。

- 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

6.3.2 パラメータチェック機能

本カーネルでは、単純なパラメータエラーの検出を省略することができます。例えば、デバッグ完了後にパラメータチェック機能を取り外せば、オーバヘッド・コードサイズを削減することができます。

パラメータチェック機能を取り外すには、コンフィギュレータの CFG_PARCHK を選択しないようにします。

6.3.3 アドレスパラメータのアクセス許可チェック機能

メモリオブジェクト保護機能を組み込んだ場合は、パケットアドレスなどのサービスコールのアドレスパラメータについて、アクセス許可が適切かがチェックされます。しかし、このチェックは prb_mem サービスコールと等価の処理が必要であり、大きなオーバヘッドを伴うため、デバッグが十分でチェックが冗長と判断できる場合には、このチェックを省略することができます。

アドレスパラメータのアクセス許可チェック機能を取り外すには、CFG_MEMCHK を選択しないようにします。

6.3.4 E_NOSPT エラー

組み込まれていないサービスコールを呼び出した場合は、E_NOSPT エラーが返ります。

6.4 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、システムの状態に依存します。

6.4.1 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みに制限されています。

- iras_tex
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

その他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、E_CTX エラーは検出されず、その後の動作は保証されません。

6.4.2 タスクコンテキストと非タスクコンテキスト

(1) 特殊なサービスコール

以下は、タスクコンテキストと非タスクコンテキストのどちらからも呼び出せます。

- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

(2) sns で始まるサービスコール

sns で始まるサービスコールは、タスクコンテキストと非タスクコンテキストのどちらからも呼び出せます。

(3) (1), (2)以外のサービスコール

i で始まるサービスコールは非タスクコンテキスト専用、その他はタスクコンテキスト専用です。
これは、以下の 2 種類に分類されます。

(a) del_tsk など、i を付加した"idel_tsk"が存在しないサービスコール

非タスクコンテキストから呼び出した場合は、E_CTX エラーが返ります。

(b) act_tsk と iact_tsk など、同機能で i で始まるものとそうでないものがあるサービスコール

i のあるものと無いもので、カーネルのサービスコール処理は共通です。このため、i で始まるサービスコールをタスクコンテキストから呼び出した場合、i の無いサービスコールを非タスクコンテキストから呼び出した場合、共に E_CTX エラーは検出されずに正常に処理されます。
ただし、これは本カーネルの実装仕様であり、今後のバージョンでは変更される可能性があります。
アプリケーションでは「i で始まるサービスコールは非タスクコンテキスト専用、その他はタスクコンテキスト専用」というルールを遵守したプログラミングが推奨されます。

6.4.3 CPU ロック状態

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、E_CTX エラーは検出されず、その後の動作は保証されません。ただし、待ち状態に遷移するサービスコールでは、E_CTX エラーを返します。このケースについては、以降の各サービスコールのエラーコードの欄に明記しています。

- ext_tsk(CPU ロック状態は解除されます)
- exd_tsk(CPU ロック状態は解除されます)
- sns_tex
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

6.4.4 ディスパッチ禁止状態

待ち状態に遷移するサービスコールを呼び出すと、E_CTX エラーを返します。このケースについては、以降の各サービスコールのエラーコードの欄に明記しています。

6.4.5 タスクコンテキストで chg_ims で SR.IMASK を 0 以外に変更している場合

待ち状態に遷移するサービスコールを呼び出すと、E_CTX エラーを返します。このケースについては、以降の各サービスコールのエラーコードの欄に明記しています。

6.5 μITRON4.0 仕様範囲外の仕様

vset_tfl サービスコールなどのように"v", "iv", "V"で始まる名称は、μITRON4.0 仕様およびμITRON4.0 仕様保護機能拡張の範囲外の本カーネル独自の仕様です。

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

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

6.6 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 6.1の形式で説明します。

節番号　　機能(サービスコール名)	
■C 言語 API	
サービスコール呼出し形式	
■パラメータ	
型　　パラメータ名　　意味	
■リターンパラメータ	
型　　パラメータ名　　意味	
■パケットの構造	
・・・	←　下記(1)参照
■リターンコード／エラーコード	
ニモニック　[エラー種別]　　意味	←　下記(2)参照
■機能	
機能の説明	
■CFG_MEMCHK によるエラー検出	←　下記(2)参照
・・・	

図6.1　サービスコールの説明形式

(1) パケットの構造

パケットは、以下の形式で表記しています。

```
typedef struct {
    ID      wtskid;     0           2           待ちタスク ID
    UINT    semcnt;     +4          4           現在のセマフォカウント値
} T_RSEM;
                        ↑           ↑           ↑
        パケット先頭からのオフセット  メンバの      メンバの説明
                                    サイズ
```

(2) エラー種別

以下のエラー種別があります。

- [k]：常に検出されるエラーです。
- [p]：コンフィギュレータで CFG_PARCHK を選択した場合のみ検出されるエラーです。
- [m]：コンフィギュレータで、メモリオブジェクト保護機能を選択(CFG_PROTMEM)し、かつ CFG_MEMCHK を選択した場合のみ検出されるエラーです。エラー発生条件は、
- 「■CFG_MEMCHK によるエラー検出」に記載されています。

(3) CFG_MEMCHK によるエラー検出

エラー種別[m]のエラー発生条件の詳細を説明しています。

6.7 タスク管理機能

表6.9 タスク管理サービスコール

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

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

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

表6.10　タスク管理の各種仕様

項番	項目	内容
1	タスク ID	1～CFG_MAXTSKID (最大 32767)
2	タスク優先度	1～CFG_MAXTSKPRI(最大 255) *
3	タスク起動要求カウントの最大値	CFG_MAXACTCNT(最大 32767)
4	ドメイン ID	カーネルドメイン：TDOM_KERNEL(-1) ユーザドメイン：1～31
5	タスク属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述 ・TA_ACT：タスク生成後にタスクを実行可能状態にする ・TA_COP0：DSP を使用 ・TA_COP1：FPU のレジスタバンク 0 を使用 ・TA_COP2：FPU のレジスタバンク 1 を使用 ・TA_DOM(domid)：domid のドメインに所属

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

6.7.1 タスクの生成(cre_tsk, icre_tsk, acre_tsk, iacre_tsk)

■C 言語 API

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

■パラメータ

```
T_CTSK     *pk_ctsk          タスク生成情報を格納したパケットへのポインタ
《cre_tsk、icre_tsk》
ID         tskid    タスク ID
```

■リターンパラメータ

```
《cre_tsk、icre_tsk》
ER         ercd              正常終了(E_OK)またはエラーコード
《acre_tsk、iacre_tsk》
ER_ID      tskid    生成したタスク ID(正の値)またはエラーコード
```

■パケットの構造

```
typedef    struct   {
           ATR      tskatr;      0     4    タスク属性
           VP_INT   exinf;      +4     4    拡張情報
           FP       task;       +8     4    タスク起動アドレス
           PRI      itskpri;    +12    2    初期タスク優先度
           SIZE     stksz;      +16    4    タスクのスタックサイズ
           VP       stk;        +20    4    タスクのスタック領域の先頭アドレス
           SIZE     sstksz;     +24    4    タスクのシステムスタックサイズ
           VP       sstk;       +28    4    タスクのシステムスタック領域の先頭アドレス
           UW       inifpscr;   +32    4    初期 FPSCR 値
} T_CTSK;
```

■エラーコード

```
E_RSATR    [p]     予約属性
                   (1)tskatr の TA_COP0,TA_COP1,TA_COP2,TA_ASM,TA_ACT、上位 8 ビット以外
                    のビットが 0 でない
                   (2)CFG_DSP 未チェックで、tskatr に TA_COP0 を指定
                   (3)CFG_FPU 未チェックで、tskatr に TA_COP1 を指定
                   (4)tskatr に、TA_COP1 を指定せずに TA_COP2 を指定
                   (5)tskatr に、TA_COP0 と TA_COP1 を同時に指定
                   (6)tskatr の上位 8bit が 0～31,H'ff 以外
```

E_PAR	[p]	パラメータエラー (1)itskpri≦0 (2)itskpri＞CFG_MAXTSKPRI (3)ユーザドメイン所属指定で stksz=0 または sstksz=0 (4)カーネルドメイン所属指定で stksz=0 (5)pk_ctsk が 4 バイト境界でない (6)task が奇数 (7)stk!=NULL で 4 バイト境界でない (8)ユーザドメイン所属の指定で sstk!=NULL で 4 バイト境界でない (9)ユーザドメイン所属指定で stk=NULL の場合で、stksz＞CFG_SYSPOOLSZ (10)ユーザドメイン所属指定で sstk=NULL の場合で、 sstksz＞(CFG_RESPOOLSZ-VTSZ_RPLMB) (11)カーネルドメイン所属指定で stk=NULL の場合で、 (stksz+sstksz)＞(CFG_RESPOOLSZ- VTSZ_RPLMB)
E_ID	[p]	不正 ID 番号 (1)tskid≦0 (2)tskid＞CFG_MAXTSKID
E_NOMEM	[k]	メモリ不足 (1)システムプールの空き不足 (2)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_tsk のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが存在する
E_MACV	[m]	メモリアクセス違反

■機能

タスクを生成します。生成したタスクは、TA_ACT 属性の指定がない場合は休止状態へ、TA_ACT 属性の指定がある場合は実行可能状態に移行します。

タスク生成時に行われる処理は、表 6.11の通りです。

表6.11　タスク生成時に行われる処理

項番	処理内容
1	**タスク起動要求キューイング数をクリアする。**
2	**タスク例外処理ルーチンを定義されていない状態にする。**
3	**上限プロセッサ時間が設定されていない状態にする。**
4	**スタックを割り付ける。**
5	**パフォーマンスカウンタの測定データを 0 クリアし、累積を開始する。**

以下に各パラメータの意味を示します。

(1) tskid

cre_tsk, icre_tsk サービスコールの場合、tskid には 1～CFG_MAXTSKID の値を指定します。

acre_tsk, iacre_tsk サービスコールの場合、未登録のタスク ID を検索してその ID を持つタスクを pk_ctsk で示された内容で生成し、その ID をリターンパラメータとして返します。

(2) tskatr

tskatr には、以下の論理和を指定してください。

(a) 記述言語

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(b) タスク起動指定

対象タスクを READY 状態にする場合は TA_ACT を指定してください。TA_ACT を指定しない場合、対象タスクは DORMANT 状態になります。

- TA_ACT(H'00000002)：タスク生成後にタスクを実行可能状態にする

(c) DSP 内蔵マイコンを使用する場合(CFG_DSP をチェックした場合)

DSP を使用する場合は、TA_COP0 を指定してください。

- TA_COP0(H'00000100)：DSP を使用

(d) FPU 内蔵マイコンを使用する場合(CFG_FPU をチェックした場合)

浮動小数点演算など、FPU を使用する場合は、TA_COP1 を指定してください。また、マトリックス演算など FPU の両バンクを使用する場合には､TA_COP1 に加えて TA_COP2 を指定してください。

- TA_COP1(H'00000200)：FPU のレジスタバンク 0(FPR0_BANK0～FPR15_BANK0)と FPUL を使用
- TA_COP2(H'00000400)：FPU のレジスタバンク 1(FPR0_BANK1～FPR15_BANK1)を使用
  TA_COP2を指定する場合は、必ずTA_COP1も指定しなければなりません。そうでない場合、E_RSATRエラーを返します。

また、「(7) inifpscr」も参照してください。

(e) 起動時の SR と所属ドメイン

所属するドメインを指定するために以下の属性を指定(OR)することができます。

TA_DOM(domid)

TA_DOM(domid)では、domid に以下を指定することができます。

(a) 1～31：指定されたdomidのユーザドメインに所属

(b) TDOM_SELF(0)：呼び出し元ドメイン。ただし、タスクコンテキストで実行中の拡張サービスコールルーチンまたはトラップルーチンから発行した場合は､それらを呼び出したタスクが所属するドメイン(その拡張サービスコールルーチンまたはトラップルーチンからget_didを発行して得られるドメインIDと同じです)。また、非タスクコンテキストから呼び出した場合は、カーネルドメインとなります。

(c) TDOM_KERNEL(-1)：カーネルドメインに所属

TA_DOM(domid)の指定が無い場合は、(b)の扱いとなります。

カーネルドメインに所属するタスクは特権モード(SR.MD=1)、ユーザドメインに所属するタスクはユーザモード(SR.MD=0)で実行されます。

●メモリオブジェクト保護機能を組み込まない場合
ユーザドメインIDによる区別は意味を持たず、カーネルドメインとユーザドメインの種類のみ意味があります。
具体的には、カーネルドメインとユーザドメインの相違点は以下の点のみです。
・起動時のSRレジスタのMDビットは、カーネルドメイン所属の場合は1(特権モード)、ユーザドメイン所属の場合は0(ユーザモード)となります。
・使用するスタック(「(6) stksz, stk, sstksz, sstk」を参照)

(3) exinf

exinf は、生成するタスクに関する情報を設定するなどの目的で自由に使用できます。
exinf は act_tsk で起動されたときにタスクにパラメータとして渡されます。

(4) task

task には、タスクの起動アドレスを指定します。

(5) itskpri

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

(6) stksz, stk, sstksz, sstk

これらは、スタックを指定するためのパラメータです。
なお、stksz, sstksz は 4 の倍数に切上げて扱われます。以降の説明では、stksz と sstksz は 4 の倍数に切上げ後の値を意味することとします。

(a) ユーザドメインに所属するタスクの場合

ユーザドメインに所属するタスクは、タスクが実行するために使用するスタック以外に「システムスタック」を持ちます。システムスタックは、タスクから呼び出された拡張サービスコールルーチンとトラップルーチン、およびカーネルがタスクのコンテキストを保存するために使用するスタックです。
スタックの範囲は stk から stksz バイト、システムスタックの範囲は sstk から sstksz バイトとなります。

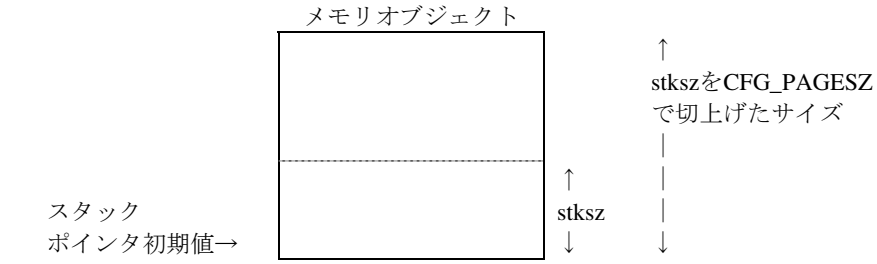
●メモリオブジェクト保護機能を組み込まない場合
スタックは、ユーザモードからアクセス可能な領域でなければなりません。カーネルは、この違反を検出しません。これに違反する場合は、ユーザモードからスタックをアクセスした時にCPU例外が発生します。
システムスタックは、ユーザモードからアクセスできない領域でなければなりません。カーネルは、この違反を検出しません。これに違反する場合は、ユーザモードからシステムスタックをアクセスできてしまうため、スタック破壊の危険性が高まります。

◎メモリオブジェクト保護機能を組み込んだ場合
スタックは、対象タスクが所属するドメインからリード・ライトアクセスアクセス可能な領域でなければなりません。これに違反する場合は、E_MACVエラーを返します。
システムスタックは、ユーザモードからアクセスできず、かつMMU非対象領域でなければなりません。つまり、メモリオブジェクトはシステムスタックとして使用できません。これに違反する場合は、E_MACVエラーを返します。

stk に NULL を指定すると、カーネルはシステムプールからスタックを割り付けます。この場合、カーネルは割り付けたスタックの管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.2(1) タスク」
・システムプールの消費→「14.2(1) タスク生成時」

◎メモリオブジェクト保護機能を組み込んだ場合
カーネルがシステムプールから割り付けるスタック領域は、以下の性質のメモリオブジェクトとなります。
(1)サイズ：stkszをCFG_PAGESZで切上げたサイズ
　スタックサイズはCFG_PAGESZで切上げられます。



(2)ページサイズ：4kB
(3)所属ドメイン：対象タスクが所属するドメイン
(4)メモリ属性：TA_RW|TA_CACHE|TA_WBACK
(5)アクセス許可ベクタ：TACT_PRW(domid)
　　(domidは対象タスクの所属ドメインID)

sstk に NULL を指定すると、カーネルはシステムスタックをリソースプールから割り付けます。詳細は、以下を参照してください。

■関連ページ　「13.2.2(1) タスク」

(b) カーネルドメインに所属するタスクの場合

ユーザドメインに所属するタスクとは異なり、カーネルドメインに所属するタスクはスタックを一つだけ持ちます。sstk は単に無視されます。

スタックは、タスクを実行するため、およびタスクから呼び出された拡張サービスコールルーチンとトラップルーチン、およびカーネルがタスクのコンテキストを保存するために使用されます。

スタックの範囲は stk から(stksz + stksz)バイトとなります。

●メモリオブジェクト保護機能を組み込まない場合
スタックは、ユーザモードからアクセスできない領域でなければなりません。カーネルは、この違反を検出しません。これに違反する場合は、ユーザモードから特権スタックをアクセスできてしまうため、スタック破壊の危険性が高まります。

◎メモリオブジェクト保護機能を組み込んだ場合
スタックは、ユーザモードからアクセスできず、かつMMU非対象領域でなければなりません。つまり、メモリオブジェクトはスタックとして使用できません。これに違反する場合は、E_MACVエラーを返します。

stk に NULL を指定すると、カーネルはリソースプールからスタックを割り付けます。詳細は、以下を参照してください。

■関連ページ　「13.2.2(1) タスク」

(7) inifpscr

inifpscr は、μITRON 仕様外の項目です。

CFG_FPU を選択していた場合で、かつ TA_COP1 属性を指定した場合のみ有効です。その他の場合は単に無視されます。

inifpscr は、起動時の FPSCR 値です。カーネルは、inifpscr で指定された値をそのまま FPSCR に設定します。inifpscr に指定された値に関するエラー検出は行いません。

下記も参照してください。

■関連ページ　「15. FPU に関する注意事項」

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_ctskに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_ctsk
- size=sizeof(T_CTSK)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_ctsk->taskに対する生成対象タスクの所属ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_ctsk->task
- size=1
- domid=生成対象タスクが所属するドメイン
- pmmode=TPM_READ

(3) ユーザドメインに所属するタスクを生成する場合で、stkで指定されたスタック領域に対する、生成対象タスクの所属ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=stk
- size=stksz
- domid=生成対象タスクが所属するドメイン
- pmmode=TPM_READ|TPM_WRITE

(4) ユーザドメインに所属するタスクを生成する場合でsstkで指定されたシステムスタック領域、またはカーネルドメインに所属するタスクでstkで指定されたスタック領域が、「MMU 非対象領域かつユーザモードからアクセス不可」の条件を満たしていない。具体的には、下記以外の領域を指定している。
- CFG_IRAM が「MMU 非対象領域, ユーザ非 DSP モードではアクセス不可」の設定の場合で、コンフィギュレータで設定した内蔵メモリ論理アドレス範囲内
- P1,P2 領域内

6.7.2 タスクの削除(del_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid≦0 (2)tskid＞CFG_MAXTSKID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態でない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

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

削除するタスクの生成時にシステムプールとリソースプールから獲得された領域は、それぞれ解放されます。

6.7.3 タスクの起動(act_tsk, iact_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)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 (1)既に起動要求キューイング数が CFG_MAXACTCNT に達している

■機能

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。
タスク起動時に行われる処理は、表 6.12の通りです。

表6.12 タスク起動時に行われる処理

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

tskid=TSK_SELF(0)の指定により、自タスクの指定になります。
タスクには、タスク生成時に指定したタスクの拡張情報がパラメータとして渡ります。
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 CFG_MAXACTCNT 回まで記憶されます。

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER_UINT	actcnt	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクにキューイングされていた起動要求回数を返し、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。

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
VP_INT	stacd	タスク起動コード

■リターンパラメータ

ER	ercd	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 (1)tskid≦0 (2)tskid＞CFG_MAXTSKID
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態でない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。この時、タスク起動時に行うべき処理(表 6.12参照)を行います。

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

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

■C 言語 API

```
void ext_tsk();
void exd_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。

また、以下のエラーが発生するとシステムダウンとなります。

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し)

■機能

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

タスク終了時に行われる処理は、表 6.13の通りです。

表6.13 タスク終了時に行われる処理

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

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

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

exd_tsk では、タスクの生成時にシステムプールとリソースプールから獲得された領域は、それぞれ解放されます。

ext_tsk, exd_tsk サービスコールは、ディスパッチ禁止状態、CPU ロック状態、chg_ims によって割込みマスクを 0 以外に変更しているときも呼び出せます。この場合、ディスパッチ禁止状態、CPU ロック状態は解除され、割込みマスクも 0 に戻ります。

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

6.7.7 タスクの強制終了(ter_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid≦0 (2)tskid＞CFG_MAXTSKID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)対象タスクが自タスク
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示された他タスクを強制的に終了させます。終了させた他タスクは休止状態へ移行します。この時、表 6.13に示す処理が行われます。

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

本サービスコールによる強制終了要求は、次の場合には遅延されます。

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

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

6.7.8 タスク優先度の変更(chg_pri, ichg_pri)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
PRI	tskpri	タスクのベース優先度

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)tskpri＜0 (2)tskpri＞CFG_MAXTSKPRI
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_ILUSE	[k]	サービスコール不正使用 (1)上限優先度の違反
E_OBJ	[k]	オブジェクト状態不正 (1)タスクが休止状態である
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■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。tskid=TSK_SELF(0)の指定により、自タスク指定となります。

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

対象タスクがミューテックスをロック中、またはミューテックスロック待ち状態の場合、指定した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高い場合は E_ILUSE エラーを返します。

また、対象タスクがミューテックスをロック中で無い場合は、ベース優先度に加えて現在優先度も tskpri に変更します。

変更したベース優先度は、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ベース優先度は無効になり、次に起動されたときにはタスク生成時に指定した初期タスク優先度になります。

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

6.7.9 タスク優先度の参照(get_pri, iget_pri)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
PRI	*p_tskpri	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	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 2 バイト境界でない)
E_ID	[p]	不正 ID 番号
		(1)tskid＜0
		(2)tskid＞CFG_MAXTSKID
		(3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正
		(1)タスクが休止状態である
E_NOEXS	[k]	未登録
		(1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_tskpriに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_tskpri
- size=sizeof(PRI)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.7.10 タスクの状態参照(ref_tsk, iref_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
T_RTSK	*pk_rtsk	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	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 {
           STAT    tskstat;     0      4    タスク状態
           PRI     tskpri;      +4     2    タスクの現在優先度
           PRI     tskbpri;     +6     2    タスクのベース優先度
           STAT    tskwait;     +8     4    待ち要因
           ID      wobjid;      +12    2    待ちオブジェクト ID
           TMO     lefttmo;     +16    4    タイムアウトするまでの時間
           UINT    actcnt;      +20    4    起動要求キューイング数
           UINT    wupcnt;      +24    4    起床要求キューイング数
           UINT    suscnt;      +28    4    強制待ち要求ネスト数
           MODE    tskmode;     +32    4    タスクの実行モード
           FLGPTN  tflptn;      +36    4    現在のタスク付属イベントフラグの値
           ID      domid;       +40    2    所属ドメイン ID
} T_RTSK;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtsk が 4 バイト境界でない)
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

tskid で示されたタスクの状態を参照します。tskid=TSK_SELF(0)の指定により自タスクの指定になります。

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

- tskstat

現在のタスクの状態です。tskstat には、次の値を返します。

- TTS_RUN(H'00000001)実行状態
- TTS_RDY(H'00000002)実行可能状態
- TTS_WAI(H'00000004)待ち状態
- TTS_SUS(H'00000008)強制待ち状態
- TTS_WAS(H'0000000c)二重待ち状態
- TTS_DMT(H'00000010)休止状態

- tskpri

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

- tskbpri

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

- tskwait *

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

- TTW_SLP(H'00000001)slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY(H'00000002)dly_tsk サービスコールによる待ち
- TTW_SEM(H'00000004)wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG(H'00000008)wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ(H'00000010)snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ(H'00000020)rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX(H'00000040)rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX(H'00000080)loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_SMBF(H'00000100)snd_mbf、tsnd_mbf サービスコールによる待ち
- TTW_RMBF(H'00000200)rcv_mbf、trcv_mbf サービスコールによる待ち
- TTW_MPF(H'00002000)get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL(H'00004000)get_mpl、tget_mpl サービスコールによる待ち
- TTW_TFL(H'00008000)vwai_tfl、vtwai_tfl サービスコールによる待ち
- TTW_MBP(H'00020000)rcv_mbp、trcv_mbp サービスコールによる待ち

- wobjid *

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

- lefttmo *

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

- actcnt *

現在の起動要求キューイング数を返します。

♦ wupcnt *

現在の起床要求キューイング数を返します。

♦ suscnt *

現在の強制待ち要求ネスト数を返します。

♦ tskmode *

tskmode は、μITRON 仕様外の項目です。

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

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

♦ tflptn *

tflptn は、μITRON 仕様外の項目です。

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

♦ domid

domid は、μITRON 仕様外の項目です。

domid には、対象タスクが所属するドメイン ID が返ります。

カーネルドメインに所属する場合は、TDOM_KERNEL(-1)が返ります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rtskに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rtsk
- size=sizeof(T_RTSK)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
T_RTST	*pk_rtst	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	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 {
           STAT    tskstat;     0      4    タスク状態
           STAT    tskwait;     +4     4    待ち要因
} T_RTST;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtst が 4 バイト境界でない)
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

tskid で示されたタスクの状態を参照します。tskid=TSK_SELF(0)の指定により自タスクの指定になります。

pk_rtst の各メンバは、ref_tsk で返る pk_rtsk の同名称のメンバと同じです。詳細は、ref_tsk の節を参照してください。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rtskに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rtst
- size=sizeof(T_RTST)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.7.12 タスク実行モードの変更(vchg_tmd)

■C 言語 API

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

■パラメータ

MODE	tmd	変更するタスク実行モード

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)tmd が不正
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し

■機能

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

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

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

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

タスク実行モードは、拡張サービスコールの呼び出しとその復帰、タスク例外処理ルーチンの起動とその復帰時には、一切変更されません。

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

6.8 タスク付属同期機能

表6.14 タスク付属同期サービスコール

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

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

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

表6.15 タスク付属同期機能の仕様

項番	項目	内容
1	タスク起床要求カウントの最大値	CFG_MAXWUPCNT(最大 32767)
2	タスク強制待ち要求ネスト数の最大値	CFG_MAXSUSCNT(最大 32767)
3	タスク付属イベントフラグビット数	32 ビット(下位 16 ビットは将来拡張用)
4	タスク付属イベントフラグ初期値	0
5	タスク付属イベントフラグ待ち条件	OR 待ち

6.8.1 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	タイムアウト指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー
		(1)tmout≦-2
E_CTX	[k]	コンテキストエラー
		(1)ディスパッチ保留状態からの呼び出し
E_RLWAI	[k]	待ち状態強制解除
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	タイムアウト

■機能

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

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

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

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

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

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

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

6.8.2 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	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態である
E_NOEXS	[k]	未登録 (1)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 (1)既に起床要求キューイング数が CFG_MAXWUPCNT に達している

■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 CFG_MAXWUPCNT 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。

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER_UINT	wupcnt	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]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態である
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクにキューイングされていた起床要求回数を返し、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。

6.8.4 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	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid ＜ 0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

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

また、対象タスクが拡張サービスコールルーチンまたはトラップルーチンを実行中の場合は、対象タスクを待ち禁止状態とします。待ち禁止状態は、対象タスクが呼び出している拡張サービスコールルーチンおよびトラップルーチンがすべて終了したときに解除されます。即ち、本サービスコールの対象タスクは、拡張サービスコールルーチンおよびトラップルーチン実行中の間だけ待ち禁止状態となります。

待ち禁止状態では、待ち状態に遷移するサービスコールを発行し、待ち状態に遷移する条件が成立すると、E_RLWAI エラーとなります。

tskid に TSK_SELF(0)を指定すると、自タスクを対象とします。

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, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールが呼び出され、強制待ち状態が解除されると、対象タスクにはエラーコードとして E_RLWAI が返されます。

なお、強制待ち状態を解除するには、rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk を使用してください。

6.8.5 強制待ち状態への移行(sus_tsk, isus_tsk)

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_CTX	[k]	コンテキストエラー (1)ディスパッチ保留状態で、実行状態のタスクを指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態である
E_NOEXS	[k]	未登録 (1)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 (1)既に強制待ち要求ネスト数が CFG_MAXSUSCNT に達している

■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, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールの呼び出しにより解除されます。

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

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

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

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid≦0 (2)tskid＞CFG_MAXTSKID
E_OBJ	[k]	オブジェクトの状態不正 (1)tskid のタスクが休止状態である (2)tskid のタスクが強制待ち状態でない (3)tskid が自タスク
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクの強制待ち状態を解除します。

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

6.8.7 タスク遅延(dly_tsk)

■C 言語 API

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

■パラメータ

RELTIM	dlytim	遅延時間

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)ディスパッチ遅延状態からの呼び出し
E_RLWAI	[k]	待ち状態強制解除 (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした

■機能

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

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

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

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

■C 言語 API

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

■パラメータ

ID	tskid	タスク ID
FLGPTN	setptn	セットするビットパターン

■リターンパラメータ

ER	ercd	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
		(1)tskid ＜ 0
		(2)tskid＞CFG_MAXTSKID
		(3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正
		(1)tskid のタスクが休止状態である
E_NOEXS	[k]	未登録
		(1)tskid のタスクが存在しない

■機能

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

tskid=TSK_SELF(0)の指定により自タスクの指定になります。

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

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

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

ER	ercd	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 (1)tskid ＜ 0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態である
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

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

tskid=TSK_SELF(0)の指定により自タスクの指定になります。

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

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

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

■エラーコード

E_PAR	[p]	パラメータエラー (1)waiptn=0 (2)tmout≦-2 (3)p_tflptn が 4 バイト境界でない
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し (2)タスクコンテキストで、ディスパッチ保留状態からの呼び出し (vwai_tfl, twai_tfl のみ)
E_RLWAI	[k]	待ち状態強制解除(vwai_tfl, vtwai_tfl のみ) (1)待ち状態の間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	タイムアウト
E_MACV	[m]	メモリアクセス違反

■機能

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

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

なお、タスク起動時のタスク付属イベントフラグの値は 0 です。

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_tflptnに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base= p_tflptn
- size=sizeof(T_RTSK)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.9 タスク例外処理機能

表6.16 タスク例外処理サービスコール

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

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

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

表6.17 タスク例外処理機能の仕様

項番	項目	内容
1	タスク例外要因	32 ビット
2	タスク例外処理ルーチン属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述 ・TA_COP0：DSP を使用 ・TA_COP1：FPU のレジスタバンク 0 を使用 ・TA_COP2：FPU のレジスタバンク 1 を使用

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

- タスク例外処理許可状態
- 保留例外要因が 0 以外
- CPU ロック状態でない
- chg_ims によって割込みマスクを 0 以外に変更していない
- タスクは拡張サービスコールルーチンまたはトラップルーチンの実行中でない

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

なお、タスク例外処理ルーチンの起動/終了の前後では、以下の状態は変化しません。

- タスクコンテキスト/非タスクコンテキスト
- ディスパッチ禁止/許可状態
- CPU ロック/ロック解除状態
- ドメイン種別(カーネルドメイン/ユーザドメイン)

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

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct   {
           ATR      texatr;      0     4    タスク例外処理ルーチン属性
           FP       texrtn;     +4     4    タスク例外処理ルーチン起動アドレス
           UW       inifpscr;   +8     4    初期 FPSCR 値
} T_DTEX;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)texatr の TA_COP0,TA_COP1,TA_COP2,TA_ASM 以外のビットが 0 でない (2)CFG_DSP 未チェックで、texatr に TA_COP0 を指定 (3)CFG_FPU 未チェックで、texatr に TA_COP1 を指定 (4)texatr に、TA_COP1 を指定せずに TA_COP2 を指定 (5)texatr に、TA_COP0 と TA_COP1 を同時に指定
E_PAR	[p]	パラメータエラー (1)pk_dtex が 4 バイト境界でない (2)texrtn が奇数
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

タスク例外処理ルーチンを定義します。
以下に各パラメータの意味を示します。

(1) tskid

定義対象のタスク ID を指定します。TSK_SELF(0)の指定により、自タスクが対象になります。
タスク例外処理ルーチンは、対象タスクと同じドメインに所属します。
カーネルドメインに所属するタスクのタスク例外処理ルーチンは特権モード(SR.MD=1)、ユーザドメインに所属するタスクのタスク例外処理ルーチンはユーザモード(SR.MD=0)で実行されます。

(2) texatr

texatr には、以下の論理和を指定してください。

(a) 記述言語

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(b) DSP 内蔵マイコンを使用する場合(CFG_DSP をチェックした場合)

DSP を使用する場合は、TA_COP0 を指定してください。

- TA_COP0(H'00000100)：DSP を使用

(c) FPU 内蔵マイコンを使用する場合(CFG_FPU をチェックした場合)

浮動小数点演算など、FPU を使用する場合は、TA_COP1 を指定してください。また、マトリックス演算など FPU の両バンクを使用する場合には、TA_COP1 に加えて TA_COP2 を指定してください。

- TA_COP1(H'00000200)：FPU のレジスタバンク 0(FPR0_BANK0～FPR15_BANK0)と FPUL を使用
- TA_COP2(H'00000400)：FPU のレジスタバンク 1(FPR0_BANK1～FPR15_BANK1)を使用
  TA_COP2を指定する場合は、必ずTA_COP1も指定しなければなりません。そうでない場合、E_RSATRエラーを返します。

また、「(4) inifpscr」も参照してください。

(3) texrtn

texrtn には、タスク例外処理ルーチンの起動アドレスを指定します。

pk_dtex＝NULL(0)と指定した場合には tskid のタスク例外処理ルーチンの定義を解除します。この時、タスクの保留例外要因を 0 クリアし、タスクをタスク例外処理禁止状態に移行します。
すでにタスク例外処理ルーチンが定義されていた場合は、以前の定義を解除し新しい定義に置き換えます。この時、保留例外要因のクリアとタスク例外処理の禁止は行いません。

(4) inifpscr

inifpscr は、μITRON 仕様外の項目です。

CFG_FPU を選択していた場合で、かつ TA_COP1 属性を指定した場合のみ有効です。その他の場合は単に無視されます。

inifpscr は、起動時の FPSCR 値です。カーネルは、inifpscr で指定された値をそのまま FPSCR に設定します。inifpscr に指定された値に関するエラー検出は行いません。

下記も参照してください。

■関連ページ　「15. FPU に関する注意事項」

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dtex!=NULLの場合で、pk_dtexに対する呼び出し元ドメインのリードアクセス許可が無い。

prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_dtex
- size=sizeof(T_DTEX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dtex!=NULLの場合で、pk_dtex->texrtnに対し、tskidのタスクが所属ドメインのリードアクセス許可が無い。

prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_dtex->texrtn
- size=1
- domid= tskid のタスクの所属ドメイン
- pmmode=TPM_READ

6.9.2 タスク例外処理の要求(ras_tex, iras_tex)

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

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

■機能

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

tskid=TSK_SELF(0)の指定により自タスクの指定になります。

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

6.9.3 タスク例外処理の禁止(dis_tex)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_OBJ	[k]	オブジェクト状態不正 (1)自タスクにタスク例外処理ルーチンが定義されていない

■機能

自タスクをタスク例外処理禁止状態に移行させます。

6.9.4 タスク例外処理の許可(ena_tex)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_OBJ	[k]	オブジェクト状態不正 (1)自タスクにタスク例外処理ルーチンが定義されていない

■機能

自タスクを、タスク例外許可状態に移行させます。

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

6.9.5 タスク例外禁止状態の参照(sns_tex)

■C 言語 API

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

■パラメータ

なし

■リターンパラメータ

BOOL	state	タスク例外禁止状態

■エラーコード

なし

■機能

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

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

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

6.9.6 タスク例外処理の状態参照(ref_tex, iref_tex)

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

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

■パケットの構造

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

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rtex が 4 バイト境界でない)
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)tskid のタスクが休止状態である (2)tskid のタスクにタスク例外処理ルーチンが定義されていない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

tskid で示されたタスク例外処理の状態を参照します。tskid=TSK_SELF(0)の指定により自タスクの指定になります。

pk_rtex が指す領域に、以下の値を返します。

♦ texstat

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

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

♦ pndptn

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rtexに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rtex
- size=sizeof(T_RTEX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.10 同期・通信(セマフォ)機能

表6.18 同期・通信(セマフォ)サービスコール

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

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

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

表6.19 セマフォの仕様

項番	項目	内容
1	セマフォ ID	1～CFG_MAXSEMID (最大 32767)
2	セマフォカウンタ最大値	65535
3	セマフォ属性	・TA_TFIFO：待ちタスクのキューイングは FIFO 順 ・TA_TPRI：待ちタスクのキューイングは現在優先度順

6.10.1 セマフォの生成(cre_sem, icre_sem, acre_sem, iacre_sem)

■C 言語 API

```
ER ercd = cre_sem(ID semid, T_CSEM *pk_csem);
ER ercd = icre_sem(ID semid, T_CSEM *pk_csem);
ER_ID semid = acre_sem(T_CSEM *pk_csem);
ER_ID semid = iacre_sem(T_CSEM *pk_csem);
```

■パラメータ

T_CSEM	*pk_csem	セマフォ生成情報を格納したパケットへのポインタ
《cre_sem、icre_sem》		
ID	semid	セマフォ ID

■リターンパラメータ

《cre_sem、icre_sem》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_sem、iacre_sem》		
ER_ID	semid	生成したセマフォの ID 番号(正の値)またはエラーコード

■パケットの構造

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

■エラーコード

E_RSATR	[p]	予約属性 (1)sematr が不正
E_PAR	[p]	パラメータエラー (1)maxsem=0 または maxsem>H'ffff (2)isemcnt>maxsem (3)pk_csem が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)semid≦0 (2)semid＞CFG_MAXSEMID
E_NOID	[k]	空き ID なし(acre_sem のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)semid のセマフォが存在する
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

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

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

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

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

なお、セマフォはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_csemに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_csem
- size=sizeof(T_CSEM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.10.2 セマフォの削除(del_sem)

■C 言語 API

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

■パラメータ

ID	semid	セマフォ ID

■リターンパラメータ

ER	ercd	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 (1)semid≦0 (2)semid＞CFG_MAXSEMID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)semid のセマフォが存在しない

■機能

semid で示されたセマフォを削除します。

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

6.10.3 セマフォ資源の返却(sig_sem, isig_sem)

■C 言語 API

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

■パラメータ

ID	semid	セマフォ ID

■リターンパラメータ

ER	ercd	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 (1)semid≦0 (2)semid＞CFG_MAXSEMID
E_NOEXS	[k]	未登録 (1)semid のセマフォが存在しない
E_QOVR	[k]	キューイングのオーバフロー (1)既にセマフォカウントがセマフォ生成時に指定した最大セマフォ資源数に達している

■機能

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

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

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

■C 言語 API

```
ER ercd = wai_sem(ID semid);
ER ercd = pol_sem(ID semid);
ER ercd = ipol_sem(ID semid);
ER ercd = twai_sem(ID semid, TMO tmout);
```

■パラメータ

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

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー
		(1)tmout≦-2
E_ID	[p]	不正 ID 番号
		(1)semid≦0
		(2)semid＞CFG_MAXSEMID
E_CTX	[k]	コンテキストエラー(wai_sem, twai_sem のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_NOEXS	[k]	未登録
		(1)semid のセマフォが存在しない
E_RLWAI	[k]	待ち状態強制解除(wai_sem, twai_sem のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除
		(1)semid のセマフォが削除された

■機能

semid で指定されるセマフォから、資源をひとつ獲得します。

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

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

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

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

6.10.5 セマフォの状態参照(ref_sem, iref_sem)

■C 言語 API

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

■パラメータ

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

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rsem が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)semid≦0 (2)semid＞CFG_MAXSEMID
E_NOEXS	[k]	未登録 (1)semid のセマフォが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および現在のセマフォカウント値(semcnt)を返します。対象セマフォに待ちタスクが無い場合は、wtskid に TSK_NONE(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rsemに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rsem
- size=sizeof(T_RSEM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.11 同期・通信(イベントフラグ)機能

表6.20 同期・通信(イベントフラグ)サービスコール

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

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

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

表6.21 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 32767)
2	イベントフラグのビット数	32 ビット
3	イベントフラグ属性	• TA_TFIFO：待ちタスクのキューイングは FIFO 順 • TA_TPRI：待ちタスクのキューイングは現在優先度順 • TA_WSGL：複数タスクの待ちを許さない • TA_WMUL：複数タスクの待ちを許す • TA_CLR：クリア指定

6.11.1 イベントフラグの生成(cre_flg, icre_flg, acre_flg, iacre_flg)

■C 言語 API

```
ER ercd = cre_flg(ID flgid, T_CFLG *pk_cflg);
ER ercd = icre_flg(ID flgid, T_CFLG *pk_cflg);
ER_ID flgid = acre_flg(T_CFLG *pk_cflg);
ER_ID flgid = iacre_flg(T_CFLG *pk_cflg);
```

■パラメータ

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

■リターンパラメータ

《cre_flg、icre_flg》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_flg、iacre_flg》		
ER_ID	flgid	生成したイベントフラグの ID 番号(正の値)またはエラーコード

■パケットの構造

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

■エラーコード

E_RSATR	[p]	予約属性 (1)flgatr が不正
E_PAR	[p]	パラメータエラー (1)pk_cflg が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_NOID	[k]	空き ID なし(acre_flg のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)flgid のイベントフラグが存在する
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

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

- TA_TFIFO(H'00000000)待ちタスクのキューイングは FIFO 順
- TA_TPRI(H'00000001)待ちタスクのキューイングは現在優先度順
- TA_WSGL(H'00000000)複数タスクの待ちを許さない
- TA_WMUL(H'00000002)複数タスクの待ちを許す
- TA_CLR(H'00000004)クリア指定

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

iflgptn には、生成するイベントフラグの初期値を指定します。

なお、イベントフラグはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cflgに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cflg
- size=sizeof(T_CFLG)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.11.2 イベントフラグの削除(del_flg)

■C 言語 API

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

■パラメータ

ID	flgid	イベントフラグ ID

■リターンパラメータ

ER	ercd	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 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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削除します。

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

6.11.3 イベントフラグのセット(set_flg, iset_flg)

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

ER	ercd	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 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_NOEXS	[k]	未登録 (1)flgid のイベントフラグが存在しない

■機能

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

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

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

6.11.4 イベントフラグのクリア(clr_flg, iclr_flg)

■C 言語 API

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

■パラメータ

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

■リターンパラメータ

ER	ercd	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 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_NOEXS	[k]	未登録 (1)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。

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

■C 言語 API

```
ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER ercd = ipol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);
```

■パラメータ

ID	flgid	イベントフラグ ID
FLGPTN	waiptn	待ちビットパターン
MODE	wfmode	待ちモード
FLGPTN	*p_flgptn	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	タイムアウト値

■リターンパラメータ

ER	ercd	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_PAR	[p]	パラメータエラー (1)waiptn=0 (2)wfmode が不正 (3)tmout≦-2 (4)p_flgptn が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_CTX	[k]	コンテキストエラー(wai_flg, twai_flg のみ) (1)ディスパッチ保留状態からの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)対象イベントフラグに TA_WSGL 属性があり、既に待ちタスクが存在している
E_NOEXS	[k]	未登録 (1)flgid のイベントフラグが存在しない
E_RLWAI	[k]	待ち状態強制解除(wai_flg, twai_flg のみ) (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除 (1)flgid のイベントフラグが削除された
E_MACV	[m]	メモリアクセス違反

■機能

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待ちます。

p_flgptn の指す領域には、待ち解除される時のイベントフラグのビットパターンを返します。

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

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

wfmode には、次のような指定を行います。

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

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

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

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_flgptnに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_flgptn
- size=sizeof(FLGPTN)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.11.6 イベントフラグの状態参照(ref_flg, iref_flg)

■C 言語 API

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

■パラメータ

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

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rflg が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)flgid≦0 (2)flgid＞CFG_MAXFLGID
E_NOEXS	[k]	未登録 (1)flgid のイベントフラグが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および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に待ちタスクが無い場合は、wtskid に TSK_NONE(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rflgに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base= pk_rflg
- size=sizeof(T_RFLG)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.12 同期・通信(データキュー)機能

表6.22 同期・通信(データキュー)サービスコール

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

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

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

表6.23 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 32767)
2	1 ワード	32 ビット
3	データキュー属性	・TA_TFIFO：待ちタスクのキューイングは FIFO 順 ・TA_TPRI：待ちタスクのキューイングは現在優先度順

6.12.1 データキューの生成(cre_dtq, icre_dtq, acre_dtq, iacre_dtq)

■C 言語 API

```
ER ercd = cre_dtq(ID dtqid, T_CDTQ *pk_cdtq);
ER ercd = icre_dtq(ID dtqid, T_CDTQ *pk_cdtq);
ER_ID dtqid = acre_dtq(T_CDTQ *pk_cdtq);
ER_ID dtqid = iacre_dtq(T_CDTQ *pk_cdtq);
```

■パラメータ

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

■リターンパラメータ

《cre_dtq、icre_dtq》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_dtq、iacre_dtq》		
ER_ID	dtqid	生成したデータキューの ID 番号(正の値) またはエラーコード

■パケットの構造

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

■エラーコード

E_RSATR	[p]	予約属性 (1)dtqatr が不正
E_PAR	[p]	パラメータエラー (1)dtqcnt!=0 で、TSZ_DTQMB(dtqcnt)>(CFG_RESPOOLSZ- VTSZ_RPLMB) (2)pk_cdtq が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)dtqid≦0 (2)dtqid＞CFG_MAXDTQID
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_dtq のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)dtqid のデータキューが存在する
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

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

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

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

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

dtqmb は、本カーネルでは単に無視されます。プログラムの移植性のためには、dtqmb に NULL を指定してください。

カーネルは、データキュー領域をリソースプールから割り付けます。詳細は、以下を参照してください。

■関連ページ　「13.2.2(2) データキュー」

なお、データキューはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cdtqに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_cdtq
- size=sizeof(T_CDTQ)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.12.2 データキューの削除(del_dtq)

■C 言語 API

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

■パラメータ

ID	dtqid	データキューID

■リターンパラメータ

ER	ercd	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 (1)dtqid≦0 (2)dtqid＞CFG_MAXDTQID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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を削除します。

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

削除によって、リソースプールから割り付けられていたデータキュー領域は解放されます。

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

■C 言語 API

```
ER ercd = snd_dtq(ID dtqid, VP_INT data);
ER ercd = psnd_dtq(ID dtqid, VP_INT data);
ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);
ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
ER ercd = fsnd_dtq(ID dtqid, VP_INT data);
ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);
```

■パラメータ

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

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー
		(1)tmout≦-2
E_ID	[p]	不正 ID 番号
		(1)dtqid≦0
		(2)dtqid＞CFG_MAXDTQID
E_CTX	[k]	コンテキストエラー(snd_dtq, tsnd_dtq のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_ILUSE	[k]	サービスコール不正使用
		(1)dtqcnt が 0 のデータキューに対する fsnd_dtq, ifsnd_dtq の発行
E_NOEXS	[k]	未登録
		(1)dtqid のデータキューが存在しない
E_RLWAI	[k]	待ち状態強制解除(snd_dtq, tsnd_dtq のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除
		(1)dtqid のデータキューが削除された

■機能

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

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、生成時に指定した属性にしたがって管理されます。

受信待ちタスクも送信待ちタスクも存在しない場合は、データをデータキューに格納します。この結果、データキューカウントが＋1 されます。

データキューカウント＝最大データキューカウントの場合は、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。

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

fsnd_dtq, ifsnd_dtq では、対象データキューに送信待ちタスクが存在する場合、および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は、データキューの最古のデータを抹消し、その領域にデータを送信します。それ以外の場合は、それぞれ snd_dtq, isnd_dtq と同じ動作となります。

なお、データ数 0 のデータキューに対して dsnd_dtq, ifsnd_dtq を行うことはできません。この場合、E_ILUSE エラーが返ります。

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

■C 言語 API

```
ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data);
ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data);
ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);
```

■パラメータ

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

■リターンパラメータ

ER	ercd	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_PAR	[p]	パラメータエラー (1)tmout≦-2 (2)p_data が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)dtqid≦0 (2)dtqid＞CFG_MAXDTQID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し (2)タスクコンテキストで、ディスパッチ保留状態からの呼び出し (rcv_dtq, trcv_dtq のみ)
E_NOEXS	[k]	未登録 (1)dtqid のデータキューが存在しない
E_RLWAI	[k]	待ち状態強制解除(rcv_dtq, trcv_dtq のみ) (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除 (1)dtqid のデータキューが削除された
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

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

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_dataに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_data
- size=sizeof(VP_INT)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.12.5 データキューの状態参照(ref_dtq, iref_dtq)

■C 言語 API

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

■パラメータ

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

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rdtq が 4 バイト境界でない)
E_ID	[p]	不正 ID 番号 (1)dtqid≦0 (2)dtqid＞CFG_MAXDTQID
E_NOEXS	[k]	未登録 (1)dtqid のデータキューが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

dtqid で示されたデータキューの状態を参照します。

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)、およびデータキューに入っているデータの数(sdtqcnt)を返します。対象データキューに送信待ちタスクが無い場合は、stskid に TSK_NONE(0)を返します。対象データキューに受信待ちタスクが無い場合は、rtskid に TSK_NONE(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rdtqに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base= pk_rdtq
- size=sizeof(T_RDTQ)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.13 同期・通信(メールボックス)機能

表6.24 同期・通信(メールボックス)サービスコール

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

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

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

表6.25 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 32767)
2	メッセージ優先度	1～CFG_MAXMSGPRI [注] (最大 255)
3	メールボックス属性	・TA_TFIFO：待ちタスクのキューイングは FIFO 順 ・TA_TPRI：待ちタスクのキューイングは現在優先度順 ・TA_MFIFO：メッセージのキューイングは FIFO 順 ・TA_MPRI：メッセージのキューイングは優先度順

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

6.13.1 メールボックスの生成(cre_mbx, icre_mbx, acre_mbx, iacre_mbx)

■C 言語 API

```
ER ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx);
ER ercd = icre_mbx(ID mbxid, T_CMBX *pk_cmbx);
ER_ID mbxid = acre_mbx(T_CMBX *pk_cmbx);
ER_ID mbxid = iacre_mbx(T_CMBX *pk_cmbx);
```

■パラメータ

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

■リターンパラメータ

《cre_mbx、icre_mbx》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mbx、iacre_mbx》		
ER_ID	mbxid	生成したメールボックスの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mbxatr;      0      4    メールボックス属性
           UINT    mbxcnt;      +4     4    格納可能メッセージ数
           PRI     maxmpri;     +8     2    メッセージ優先度の最大値
           VP      mbxmb;       +12    4    メールボックス管理領域の先頭アドレス
} T_CMBX;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)mbxatr が不正
E_PAR	[p]	パラメータエラー (1)maxmpri≦0 (2)maxmpri>CFG_MAXMSGPRI (3)pk_cmbx が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbxid≦0 (2)mbxid＞CFG_MAXMBXID
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_mbx のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)mbxid のメールボックスが存在する
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

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

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

mbxcnt、mbxmb は、本カーネルでは常に無視されます。プログラムの移植性のためには、mbxcnt は適当な値、mbxmb には NULL を指定してください。

TA_MPRI 属性を指定した場合で maxmpri>1 の場合、カーネルはメールボックスを管理するためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　「13.2.2(3) メールボックス」

なお、メールボックスはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmbxに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmbx
- size=sizeof(T_CMBX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.13.2 メールボックスの削除(del_mbx)

■C 言語 API

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

■パラメータ

ID	mbxid	メールボックス ID

■リターンパラメータ

ER	ercd	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 (1)mbxid≦0 (2)mbxid＞CFG_MAXMBXID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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を削除します。

削除によって、生成および送信時にリソースプールから割り付けられていた管理領域は解放されます。

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

6.13.3 メールボックスへの送信(snd_mbx, isnd_mbx)

■C 言語 API

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

■パラメータ

ID	mbxid	メールボックス ID
T_MSG	*pk_msg	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	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 {
           VP      msghead;     0      4    カーネル管理領域
} T_MSG;
```

msghead は、従来製品とのメッセージ形式の互換性のためだけに存在しています。送信時にゼロクリアする必要はありません。

《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   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		パラメータエラー
	[p]	(1)pk_msg が 4 バイト境界でない
	[k]	(2)対象のメールボックスに TA_MPRI 属性が指定されている場合で、 msgpri≦0 または msgpri＞(生成時に指定した最大メッセージ優先度)
E_ID	[p]	不正 ID 番号 (1)mbxid≦0 (2)mbxid＞CFG_MAXMBXID
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_NOEXS	[k]	未登録 (1)mbxid のメールボックスが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

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

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

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

管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.3(1) メールボックス：snd_mbx, isnd_mbx」

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 6.2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 6.3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。

メッセージは RAM 領域に作成してください。

```
typedef struct  {
    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6.2　メッセージの形式例

```
typedef struct {
    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6.3　優先度付きメッセージの形式例

送信されたメッセージは、当然ながら受信側が読み出すことになります。したがって、以下のことに注意してください。

(1)　一般には、メッセージはローカル変数として作成しないでください。

(2)　メモリ保護機能を使用する場合は、受信側がリードアクセス許可されている領域にメッセージを作成する必要があります。通常は、メールボックスはドメイン内のタスク間通信にのみ使用することを推奨します。ドメイン間でメッセージ通信を行う場合は、データキュー(1ワードのみ)、メッセージバッファ、保護メールボックスの利用を検討してください。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1)　pk_msgに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_msg
- size=sizeof(T_MSG_PRI)
  TA_MFIFO属性の場合もこのサイズでチェックすることに注意してください。
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

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

■C 言語 API

```
ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER ercd = iprcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);
```

■パラメータ

ID	mbxid	メールボックス ID
T_MSG	**ppk_msg	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	タイムアウト指定

■リターンパラメータ

ER	ercd	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 {
           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   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
		(1)tmout≦-2
		(2)ppk_msg が 4 バイト境界でない
E_ID	[p]	不正 ID 番号
		(1)mbxid≦0
		(2)mbxid＞CFG_MAXMBXID
E_CTX	[k]	コンテキストエラー(rcv_mbx, trcv_mbx のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_RLWAI	[k]	待ち状態強制解除(rcv_mbx, trcv_mbx のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除
		(1)mbxid のメールボックスが削除された
E_MACV	[m]	メモリアクセス違反

■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 ppk_msg の指す領域に返します。

メッセージを受信すると、メッセージ送信時にカーネルがメッセージ管理のためにリソースプールから確保した管理領域が解放されます。

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

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) ppk_msgに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=ppk_msg
- size=sizeof(T_MSG *)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

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

■C 言語 API

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

■パラメータ

ID	mbxid	メールボックス ID
T_RMBX	*pk_rmbx	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	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{				
	ID	wtskid;	0	2	待ちタスク ID
	T_MSG	*pk_msg;	+4	4	次に受信されるメッセージの先頭アドレス
} T_RMBX;					

(2)T_MSG

《メールボックスのメッセージヘッダ》

typedef	struct {				
	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	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
		(1)pk_rmbx が 4 バイト境界でない
E_ID	[p]	不正 ID 番号
		(1)mbxid≦0
		(2)mbxid＞CFG_MAXMBXID
E_NOEXS	[k]	未登録
		(1)mbxid のメールボックスが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および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に待ちタスクが無い場合は、wtskid に 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、pk_msg に NULL(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmbxに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmbx
- size=sizeof(T_RMBX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.14 拡張同期・通信(ミューテックス)機能

表6.26 同期･通信(ミューテックス)サービスコール

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

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

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

表6.27 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 32767)
2	ミューテックス属性	・TA_CEILING：優先度上限プロトコル

本カーネルでは、TA_CEILING 属性(優先度上限プロトコル)のみをサポートしています。本カーネルが採用している優先度上限プロトコルでは、簡略化した優先度制御を行っています。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。

6.14.1 ミューテックスの生成(cre_mtx, acre_mtx)

■C 言語 API

```
ER ercd = cre_mtx(ID mtxid, T_CMTX *pk_cmtx);
ER_ID mtxid = acre_mtx(T_CMTX *pk_cmtx);
```

■パラメータ

T_CMTX	*pk_cmtx	ミューテックス生成情報を格納したパケットへのポインタ
《cre_mtx》		
ID	mtxid	ミューテックス ID

■リターンパラメータ

《cre_mtx》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mtx》		
ER_ID	mtxid	生成したミューテックスの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mtxatr;     0      4    ミューテックス属性
           PRI     ceilpri;    +4     2    ミューテックスの上限優先度
} T_CMTX;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)mtxatr が不正
E_PAR	[p]	パラメータエラー (1)ceilpri≦0 (2)ceilpri>CFG_MAXTSKPRI (3)pk_cmtx が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mtxid≦0 (2)mtxid＞CFG_MAXMTXID
E_NOID	[k]	空き ID なし(acre_mtx のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)mtxid のミューテックスが存在する
E_MACV	[m]	メモリアクセス違反

■機能

cre_mtx サービスコールは、mtxid で示された ID を持つミューテックスを、pk_cmtx で示された内容で生成します。

acre_mtx サービスコールは、未登録のミューテックス ID を検索してその ID を持つミューテックスを pk_cmtx で示された内容で生成し、その ID をリターンパラメータとして返します。未登録のミューテックス ID を検索する範囲は 1～CFG_MAXMTXID です。

mtxatr には属性としては、優先度上限プロトコル(TA_CEILING)のみを指定できます。

```
mtxatr := (TA_CEILING)
```

- TA_CEILING(H'00000003)優先度上限プロトコル

待ちタスクのキューイングは、常に現在優先度順となります。

ceilpri には、生成するミューテックスの上限優先度を指定します。指定できる値の範囲は、1～CFG_MAXTSKPRI です。

なお、ミューテックスはコンフィギュレータで静的に生成することもできます。

なお、本サービスコールでは非タスクコンテキストから呼び出してはなりませんが、E_CTX エラーは検出されません。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmtxに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_cmtx
- size=sizeof(T_CMTX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.14.2 ミューテックスの削除(del_mtx)

■C 言語 API

```
ER ercd = del_mtx(ID mtxid);
```

■パラメータ

ID	mtxid	ミューテックス ID

■リターンパラメータ

ER	ercd	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 (1)mtxid≦0 (2)mtxid＞CFG_MAXMTXID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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を削除します。

mtxid で示されたミューテックスにロック待ちタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

対象ミューテックスがロックされていた場合には、それをロックしているタスクのロックを解除します。その結果、そのタスクがロックしているミューテックスがなくなった場合のみ、タスクの現在優先度をベース優先度に戻します。

削除されたミューテックスをロックしているタスクには、ミューテックスが削除されたことは通知されません。後でミューテックスをロック解除しようとした時点でエラーが返されます。

6.14.3 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	ミューテックス ID
《tloc_mtx》		
TMO	tmout	タイムアウト指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)tmout≦-2
E_ID	[p]	不正 ID 番号 (1)mtxid≦0 (2)mtxid＞CFG_MAXMTXID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し (2)タスクコンテキストで、ディスパッチ保留状態からの呼び出し （loc_mtx, tloc_mtx のみ）
E_ILUSE	[k]	サービスコール不正使用 (1)呼び出し元タスクは既に対象ミューテックスをロックしている (2)上限優先度違反 （呼び出し元タスクのベース優先度 ＜ 生成時に指定した上限優先度）
E_NOEXS	[k]	未登録 (1)mtxid のミューテックスが存在しない
E_RLWAI	[k]	待ち状態強制解除(loc_mtx, tloc_mtx のみ) (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除 (1)mtxid のミューテックスが削除された

■機能

mtxid で指定されるミューテックスをロックします。

対象ミューテックスがロックされていない場合には、自タスクがミューテックスをロックした状態にして、サービスコールの処理を終了します。その際、自タスクの現在優先度はミューテックスの上限優先度まで引き上げられます。

対象ミューテックスがロックされている場合には、自タスクを待ち行列につなぎ、ミューテックスのロック待ち状態に移行させます。待ち行列は、優先度順に管理されます。

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

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

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

6.14.4 ミューテックスのロック解除(unl_mtx)

■C 言語 API

```
ER ercd = unl_mtx(ID mtxid);
```

■パラメータ

ID	mtxid	ミューテックス ID

■リターンパラメータ

ER	ercd	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 (1)mtxid≦0 (2)mtxid＞CFG_MAXMTXID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)呼び出し元タスクは対象ミューテックスをロックしていない
E_NOEXS	[k]	未登録 (1)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。

このサービスコールを呼び出したことにより、自タスクがロックしているミューテックスが無くなった場合に、そのタスクの現在優先度をベース優先度に戻します。

6.14.5 ミューテックスの状態参照(ref_mtx)

■C 言語 API

```
ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

■パラメータ

ID	mtxid	ミューテックス ID
T_RMTX	*pk_rmtx	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	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 {
           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 (1)pk_rmtx が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mtxid≦0 (2)mtxid＞CFG_MAXMTXID
E_NOEXS	[k]	未登録 (1)mtxid のミューテックスが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および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_NONE(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。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmtxに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmtx
- size=sizeof(T_RMTX)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.15 拡張同期・通信(メッセージバッファ)機能

表6.28 同期･通信(メッセージバッファ)サービスコール

項番	サービスコール *1	機 能	呼び出し可能な状態 *2						
			T	N	E	D	U	L	C
1	cre_mbf	メッセージバッファの生成	○		○	○	○		
	icre_mbf			○	○	○	○		
2	acre_mbf	メッセージバッファの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mbf			○	○	○	○		
3	del_mbf	メッセージバッファの削除	○		○	○	○		
4	snd_mbf	メッセージバッファへの送信	○		○		○		
5	psnd_mbf	同上(ポーリング)	○		○	○	○		
	ipsnd_mbf			○	○	○	○		
6	tsnd_mbf	同上(タイムアウト有)	○		○		○		
7	rcv_mbf	メッセージバッファからの受信	○		○		○		
8	prcv_mbf	同上(ポーリング)	○		○	○	○		
9	trcv_mbf	同上(タイムアウト有)	○		○		○		
10	ref_mbf	メッセージバッファの状態参照	○		○	○	○		
	iref_mbf			○	○	○	○		

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

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

表6.29 メッセージバッファの仕様

項番	項目	内容
1	メッセージバッファ ID	1～CFG_MAXMBFID (最大 32767)
2	メッセージバッファ属性	・TA_TFIFO：送信待ちタスクのキューイングは FIFO 順 ・TA_TPRI：送信待ちタスクのキューイングは現在優先度順

6.15.1 メッセージバッファの生成(cre_mbf, icre_mbf, acre_mbf, iacre_mbf)

■C 言語 API

```
ER ercd = cre_mbf(ID mbfid, T_CMBF *pk_cmbf);
ER ercd = icre_mbf(ID mbfid, T_CMBF *pk_cmbf);
ER_ID mbfid = acre_mbf(T_CMBF *pk_cmbf);
ER_ID mbfid = iacre_mbf(T_CMBF *pk_cmbf);
```

■パラメータ

T_CMBF *pk_cmbf		メッセージバッファ生成情報を格納したパケットへのポインタ
《cre_mbf、icre_mbf》		
ID	mbfid	メッセージバッファ ID

■リターンパラメータ

《cre_mbf、icre_mbf》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mbf、iacre_mbf》		
ER_ID	mbfid	生成したメッセージバッファの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mbfatr;      0      4    メッセージバッファ属性
           UINT    maxmsz;      +4     4    メッセージの最大サイズ(バイト数)
           SIZE    mbfsz;       +8     4    メッセージバッファ領域のサイズ(バイト数)
           VP      mbfmb;       +12    4    メッセージバッファ管理領域の先頭アドレス
} T_CMBF;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)mbfatr が不正
E_PAR	[p]	パラメータエラー (1)mbfsz!=0 で、mbfsz<TSZ_MBFMB(1,maxmsz) または mbfsz＞(CFG_RESPOOLSZ-VTSZ_RPLMB) (2)maxmsz=0 (3)pk_cmbf が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbfid≦0 (2)mbfid＞CFG_MAXMBFID
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_mbf のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)mbfid のメッセージバッファが存在する
E_MACV	[m]	メモリアクセス違反

■機能

cre_mbf, icre_mbf サービスコールは、mbfid で示された ID を持つメッセージバッファを、pk_cmbf で示された内容で生成します。

acre_mbf, iacre_mbf サービスコールは、未登録のメッセージバッファ ID を検索してその ID を持つメッセージバッファを pk_cmbf で示された内容で生成し、その ID をリターンパラメータとして返します。検索するメッセージバッファ ID の範囲は 1～CFG_MAXMBFID です。

mbfatr には属性として、メッセージ送信待ちタスクが待ち行列に並ぶ際の並び方を指定します。

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

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

メッセージ受信待ちタスクの待ち行列、およびメッセージ行列は、mbfatr に関わらず FIFO(First-In First-Out)となります。

maxmsz には､生成するメッセージバッファで扱うことのできるメッセージの最大長を指定します。

mbfsz には、生成するメッセージバッファ領域のサイズを指定します。mbfsz は 4 の倍数に切上げて扱います。なお、mbfsz に指定すべきサイズの目安を知るために、以下のマクロが用意されています。

SIZE mbfsz = TSZ_MBFMB(UINT msgcnt, UINT msgsz)
サイズが msgsz バイトのメッセージを msgcnt 個格納するのに必要なメッセージバッファ領域のサイズ(目安のバイト数)

mbfsz に指定するサイズは TSZ_MBFSZ(1,maxmsz)以上でなければなりません。

ただし、mbfsz=0 でメッセージバッファを生成することもできます。mbfsz=0 で生成したメッセージバッファではバッファにメッセージを蓄えておくことができないため、メッセージ送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまりメッセージ送信側と受信側が完全に同期した動作となります。

mbfsz!=0 の場合、カーネルは、メッセージバッファ領域をリソースプールから割り付けます。詳細は、以下を参照してください。

■関連ページ　「13.2.2(4) メッセージバッファ」

mbfmb は、本カーネルでは単に無視されます。プログラムの移植性のためには、mbfmb に NULL を指定してください。

なお、メッセージバッファはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmbfに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmbf
- size=sizeof(T_CMBF)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.15.2 メッセージバッファの削除(del_mbf)

■C 言語 API

```
ER ercd = del_mbf(ID mbfid);
```

■パラメータ

ID	mbfid	メッセージバッファ ID

■リターンパラメータ

ER	ercd	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 (1)mbfid≦0 (2)mbfid＞CFG_MAXMBFID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)mbfid のメッセージバッファが存在しない

■機能

mbfid で示されたメッセージバッファを削除します。

mbfid で示されたメッセージバッファにおいて、メッセージ受信またはメッセージ送信を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。また、メッセージバッファ内にメッセージが格納されていた場合でもエラーにはなりませんが、格納されていたメッセージは全て破棄されます。

削除によって、リソースプールから割り付けられていたメッセージバッファ領域は解放されます。

6.15.3 メッセージバッファへの送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

■C 言語 API

```
ER ercd = snd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = psnd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = ipsnd_mbf(ID mbfid, VP msg, UINT msgsz);
ER ercd = tsnd_mbf(ID mbfid, VP msg, UINT msgsz, TMO tmout);
```

■パラメータ

ID	mbfid	メッセージバッファ ID
VP	msg	送信メッセージの先頭アドレス
UINT	msgsz	送信メッセージのサイズ(バイト数)
《tsnd_mbf》		
TMO	tmout	タイムアウト指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR		パラメータエラー
	[p]	(1)msgsz=0
		(2)tmout≦-2
		(3)msg が 4 バイト境界でない
	[k]	(4)msgsz＞(生成時に指定したメッセージの最大サイズ)
E_ID	[p]	不正 ID 番号
		(1)mbfid≦0
		(2)mbfid＞CFG_MAXMBFID
E_CTX	[k]	コンテキストエラー(snd_mbf, tsnd_mbf のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_NOEXS	[k]	未登録
		(1)mbfid のメッセージバッファが存在しない
E_RLWAI	[k]	待ち状態強制解除(snd_mbf, tsnd_mbf のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除
		(1)mbfid のメッセージバッファが削除された
E_MACV	[m]	メモリアクセス違反

■機能

mbfid で示されたメッセージバッファに対して、msg で示されたメッセージを送信します。送信するサイズは msgsz で示されたバイト数です。

対象メッセージバッファに受信待ちタスクが存在する場合には、メッセージバッファには格納せずに受信待ち行列の先頭タスクにメッセージを渡し、そのタスクの待ち状態を解除します。

対象メッセージバッファに既に送信待ちタスクが存在する場合、snd_mbf, tsnd_mbf サービスコールではメッセージバッファの空き領域を待つための待ち行列(送信待ち行列)につながれ、psnd_mbf, ipsnd_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。送信待ち行列は、生成時に指定した属性にしたがって管理されます。

受信待ちタスクも送信待ちタスクも存在しない場合は、メッセージをメッセージバッファに格納します。この結果、メッセージバッファの空きサイズは VTSZ_MBFMSGMB(msgsz)バイトだけ減少します。このサイズだけの空きがメッセージバッファに存在しない場合(バッファサイズが 0 の場合も含む)は、呼び出しタスクは送信待ち行列につながれます。

ipsnd_mbf は、非タスクコンテキストからも発行可能です。対象のメッセージバッファに TA_TPRI 属性が指定されている場合は、非タスクコンテキストはタスクよりも優先順位が高いため、バッファに空きがあれば、先に送信を待っているタスクが存在しても、バッファにメッセージがコピーされます。

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) msgに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=msg
- size=msgsz
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.15.4 メッセージバッファからの受信(rcv_mbf, prcv_mbf, trcv_mbf)

■C 言語 API

```
ER_UINT msgsz = rcv_mbf(ID mbfid, VP msg);
ER_UINT msgsz = prcv_mbf(ID mbfid, VP msg);
ER_UINT msgsz = trcv_mbf(ID mbfid, VP msg, TMO tmout);
```

■パラメータ

ID	mbfid	メッセージバッファ ID
VP	msg	送信メッセージを返す領域への先頭アドレス
《trcv_mbf》		
TMO	tmout	タイムアウト指定

■リターンパラメータ

ER_UINT	msgsz	受信メッセージのサイズ(バイト数、正の値) またはエラーコード
VP	msg	送信メッセージを格納した領域の先頭アドレス

■エラーコード

E_PAR	[p]	パラメータエラー (1)tmout≦-2 (2)msg が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbfid≦0 (2)mbfid＞CFG_MAXMBFID
E_CTX	[k]	コンテキストエラー (1)ディスパッチ保留状態からの呼び出し
E_NOEXS	[k]	未登録 (1)mbfid のメッセージバッファが存在しない
E_RLWAI	[k]	待ち状態強制解除(rcv_mbf, trcv_mbf のみ) (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除 (1)mbfid のメッセージバッファが削除された
E_MACV	[m]	メモリアクセス違反

■機能

mbfid で示されたメッセージバッファからメッセージを受信し、受信したメッセージを msg の指す領域に格納します。また、受信したメッセージサイズをリターンパラメータとして返します。msg には、生成時に指定したメッセージ最大長(maxmsz)の空き領域を指定しなければなりません。

メッセージバッファにメッセージがあれば、メッセージ行列先頭のメッセージ(最古のメッセージ)を受信します。メッセージバッファ内のメッセージを受信することで、メッセージバッファの空きサイズは VTSZ_MBFMSGMB(msgsz)だけ増加します。

この結果、空きサイズがメッセージ送信待ち行列先頭のタスクが送信しようとしていたメッセージサイズよりも大きくなると、そのメッセージがメッセージバッファに格納され、そのタスクの待ち状態が解除されます。送信待ち行列の以降のタスクに対してもメッセージの格納が可能であれば、待ち行列の順に同様の処理を行います。

メッセージバッファにメッセージが存在せず、メッセージ送信待ちタスクが存在する場合、メッセージ送信待ち行列先頭タスクのメッセージを受信します。この結果、そのメッセージ送信待ちタスクの待ち状態は解除されます。

メッセージバッファにメッセージがなく、メッセージ送信待ちタスクも存在しない場合、rcv_mbf, trcv_mbf サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列(受信待ち行列)につながれ、prcv_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。受信待ち行列は、FIFO で管理されます。

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) msgに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=msg
- size=生成時に指定した最大メッセージサイズ
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.15.5 メッセージバッファの状態参照(ref_mbf, iref_mbf)

■C 言語 API

```
ER ercd = ref_mbf(ID mbfid, T_RMBF *pk_rmbf);
ER ercd = iref_mbf(ID mbfid, T_RMBF *pk_rmbf);
```

■パラメータ

ID	mbfid	メッセージバッファ ID
T_RMBF	*pk_rmbf	メッセージバッファ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMBF	*pk_rmbf	メッセージバッファ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct {
           ID      stskid;      0      2    送信待ち行列の先頭タスク ID
           ID      rtskid;      +2     2    受信待ち行列の先頭タスク ID
           UINT    smsgcnt;     +4     4    メッセージバッファに入っているメッセージの数
           SIZE    fmbfsz;      +8     4    空きバッファのサイズ(バイト数)
} T_RMBF;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rmbf が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbfid≦0 (2)mbfid＞CFG_MAXMBFID
E_NOEXS	[k]	未登録 (1)mbfid のメッセージバッファが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

mbfid で示されたメッセージバッファの状態を参照します。

pk_rmbf が指す領域に、送信待ちタスク ID (stskid)、受信待ちタスク ID(rtskid)、メッセージバッファに入っているメッセージの数(smsgcnt)、および空きバッファサイズ(fmbfsz)を返します。対象メッセージバッファに受信待ち・送信待ちタスクが共に無い場合は、wtskid に TSK_NONE(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmbfに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmbf
- size=sizeof(T_RMBF)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.16 メモリプール管理(固定長メモリプール)機能

表6.30 メモリプール管理(固定長メモリプール)サービスコール

項番	サービスコール *1	機 能	呼び出し可能な状態 *2						
			T	N	E	D	U	L	C
1	cre_mpf [s]	固定長メモリプールの生成	○		○	○	○		
	icre_mpf			○	○	○	○		
2	icra_mpf	固定長メモリプールの生成(アクセス許可ベクタ指定)	コンフィギュレータが生成する初期登録ルーチン専用です。その他の状態で呼び出した場合の動作は保証されません。						
3	acre_mpf	固定長メモリプールの生成	○		○	○	○		
	iacre_mpf	(ID 番号自動割付け)		○	○	○	○		
4	del_mpf	固定長メモリプールの削除	○		○	○	○		
5	get_mpf [S]	固定長メモリブロックの獲得	○		○		○		
6	pget_mpf [S]	同上(ポーリング)	○		○	○	○		
	ipget_mpf			○	○	○	○		
7	tget_mpf [S]	同上(タイムアウト有)	○		○		○		
8	rel_mpf [S]	固定長メモリブロックの返却	○		○	○	○		
	irel_mpf			○	○	○	○		
9	ref_mpf	固定長メモリプールの状態参照	○		○	○	○		
	iref_mpf			○	○	○	○		

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

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

表6.31 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 32767)
2	固定長メモリプール属性	・TA_TFIFO：待ちタスクのキューイングは FIFO 順 ・TA_TPRI：待ちタスクのキューイングは現在優先度順

6.16.1 固定長メモリプールの生成(cre_mpf, icre_mpf, acre_mpf, iacre_mpf)

■C 言語 API

```
ER ercd = cre_mpf(ID mpfid, T_CMPF *pk_cmpf);
ER ercd = icre_mpf(ID mpfid, T_CMPF *pk_cmpf);
ER_ID mpfid = acre_mpf(T_CMPF *pk_cmpf);
ER_ID mpfid = iacre_mpf(T_CMPF *pk_cmpf);
```

■パラメータ

T_CMPF	*pk_cmpf	固定長メモリプール生成情報を格納したパケットへのポインタ
《cre_mpf、icre_mpf》		
ID	mpfid	固定長メモリプール ID

■リターンパラメータ

《cre_mpf、icre_mpf》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mpf、iacre_mpf》		
ER_ID	mpfid	生成した固定長メモリプールの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mpfatr;      0      4    固定長メモリプール属性
           UINT    blkcnt;      +4     4    メモリプール全体のブロック数
           UINT    blksz;       +8     4    固定長メモリブロックサイズ(バイト数)
           VP      mpf;         +12    4    固定長メモリプール領域の先頭アドレス
           VP      mpfmb;       +16    4    固定長メモリプール管理領域の先頭アドレス
} T_CMPF;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)mpfatr が不正
E_PAR	[p]	パラメータエラー (1)blkcnt=0 (2)blksz=0 (3)mpf=NULL の場合で、TSZ_MPF(blkcnt, blksz)＞CFG_SYSPOOLSZ (4)pk_cmpf が 4 バイト境界でない (5)mpf!=NULL の場合で、mpf が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mpfid≦0 (2)mpfid＞CFG_MAXMPFID
E_NOMEM	[k]	メモリ不足 (1)システムプールの空き不足 (2)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_mpf のみ)

E_OBJ	[k]	オブジェクト状態不正 (1)mpfidの固定長メモリプールが存在する
E_MACV	[m]	メモリアクセス違反

■機能

cre_mpf, icre_mpf サービスコールは、mpfid で示された ID を持つ固定長メモリプールを、pk_cmpf で示された内容で生成します。

acre_mpf, iacre_mpf サービスコールは、未登録の固定長メモリプール ID を検索してその ID を持つ固定長メモリプールを pk_cmpf で示された内容で生成し、その ID をリターンパラメータとして返します。未登録の固定長メモリプール ID を検索する範囲は 1～CFG_MAXMPFID です。

mpfatr には属性として、メモリブロック獲得を待つ待ち行列に並ぶ際の並び方を指定します。

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

- TA_TFIFO(H'00000000)メモリブロック獲得待ちタスクのキューイングは FIFO 順
- TA_TPRI(H'00000001)メモリブロック獲得待ちタスクのキューイングは現在優先度順

blkcnt には、生成するメモリプールの総ブロック数を指定します。

blksz には、メモリブロックのサイズを指定します。blksz は、4 の倍数に切上げて扱われます。

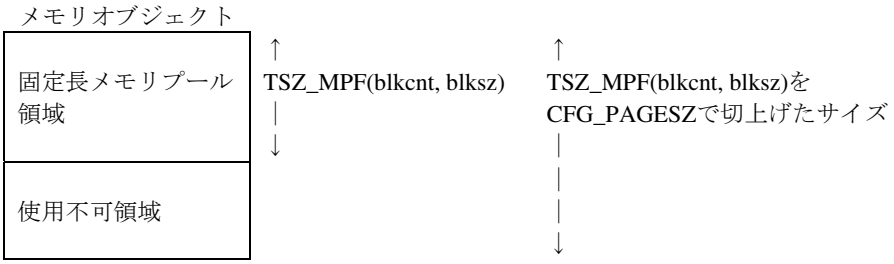
mpf には、固定長メモリプールとして使用する空き領域の先頭アドレスを指定します。mpf から TSZ_MPF(blkcnt, blksz)バイトを固定長メモリプールとして使用します。なお、指定された領域が、どのドメインからアクセスできるかは、カーネルは関知しません。例えば、P1, P2 領域のアドレスを指定した場合、その領域はユーザドメインからはアクセスできませんが、カーネルはこれを関知しません。

◎メモリオブジェクト保護機能を組み込んだ場合
指定範囲は、カーネルドメインからリード・ライトアクセス可能な領域でなければなりません。これに違反する場合は、E_MACVエラーを返します。

mpf に NULL を指定すると、カーネルはシステムプールから TSZ_MPF(blkcnt, blksz)バイトのメモリプール領域を割り付けます。また、カーネルは割り付けたメモリプール領域の管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.2(5) 固定長メモリプール」
・システムプールの消費→「14.2(2) 固定長メモリプール生成時」

◎メモリオブジェクト保護機能を組み込んだ場合
カーネルがシステムプールから割り付けるメモリプール領域は、以下の性質のメモリオブジェクトとなります。
(1)サイズ：TSZ_MPF(blkcnt, blksz)をCFG_PAGESZで切上げたサイズ
　ただし、メモリプールとして使用できるサイズは、TSZ_MPF(blkcnt, blksz)バイトです。
下の図で、使用不可領域は固定長メモリプール領域と同じメモリオブジェクト内なので、アクセス許可の区別はありませんが、メモリプールとしては管理されません。



(2)所属ドメイン：タスクコンテキストから呼び出した場合は、呼び出したタスクが所属するドメインになります。これは、get_didを発行して得られるドメインIDと同じです。非タスクコンテキストから呼び出した場合はカーネルドメインになります
(3)メモリ属性：TA_RW|TA_CACHE|TA_WBACK
(4)アクセス許可ベクタ：所属ドメインのみがリード・ライト可能なように設定されます。
　所属ドメインがカーネルドメインの場合：TACT_KERNEL
　所属ドメインがユーザドメインの場合：TACT_PRW(domid)
　　(domidは所属ドメインID)

mpfmb は、本カーネルでは単に無視されます。プログラムの移植性のためには、mpfmb に NULL を指定してください。

また、mpf に NULL を指定したかどうかに関係なく、生成した固定長メモリプールのメモリブロックを管理するために、リソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.2(5) 固定長メモリプール」

なお、固定長メモリプールはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmpfに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmpf
- size=sizeof(T_CMPF)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_cmpf->mpf != NULLの場合で、mpfからTSZ_MPF(blkcnt, blksz)バイトの領域に対し、カーネルのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmpf->mpf
- size=TSZ_MPF(blkcnt, blksz)
- domid=カーネルドメイン
- pmmode=TPM_READ|TPM_WRITE

6.16.2 固定長メモリプールの生成(アクセス許可ベクタ指定)(icra_mpf)

■C 言語 API

```
ER ercd = icra_mpf(ID mpfid, T_CMPF *pk_cmpf, ACVCT *p_acvct);
```

■パラメータ

ID	mpfid	固定長メモリプール ID
T_CMPF	*pk_cmpf	固定長メモリプール生成情報を格納したパケットへのポインタ
ACVCT	p_acvct	アクセス許可ベクタを格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct{
           ATR     mpfatr;      0      4    固定長メモリプール属性
           UINT    blkcnt;      +4     4    メモリプール全体のブロック数
           UINT    blksz;       +8     4    固定長メモリブロックサイズ(バイト数)
           VP      mpf;         +12    4    固定長メモリプール領域の先頭アドレス
           VP      mpfmb;       +16    4    固定長メモリプール管理領域の先頭アドレス
} T_CMPF;
typedef    struct{
           ACPTN   acptn1;       0     4    ライトアクセスに関するアクセス許可パターン
           ACPTN   acptn2;      +4     4    リードアクセスに関するアクセス許可パターン
} ACVCT;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)TSZ_MPF(blkcnt, blksz)＞CFG_SYSPOOLSZ)
E_NOMEM	[k]	メモリ不足 (1)システムプールの空き不足 (2)リソースプールの空き不足

■機能

mpfid で示された ID を持つ固定長メモリプールを、pk_cmpf, p_acvct で示された内容で生成します。

本サービスコールは、アプリケーションから呼び出してはなりません。本サービスコールは、メモリオブジェクト保護機能を組み込んだ場合で、コンフィギュレータで固定長メモリプールの生成を行っていた場合に、コンフィギュレータが出力する初期登録ルーチンのみから呼び出されます。本サービスコールはこの前提で実装されているため、ほとんどのエラー検出を省略しています。

cre_mpf との主な相違点は、以下の通りです。

(1) pk_cmpf->mpfはNULLのみ許可されます。つまり、メモリプール領域はシステムプールから割り付けられます。メモリプール領域のアドレスを指定することはできません。

(2) p_acvctによって、生成したメモリプール領域のメモリオブジェクトとしてのアクセス許可ベクタを指定できます。ただし、メモリ保護機能を組み込んでいない場合は、p_acvctは無視されます。

(3) E_MACVエラーは検出されません。

6.16.3 固定長メモリプールの削除(del_mpf)

■C 言語 API

```
ER ercd = del_mpf(ID mpfid);
```

■パラメータ

ID	mpfid	固定長メモリプール ID

■リターンパラメータ

ER	ercd	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 (1)mpfid≦0 (2)mpfid＞CFG_MAXMPFID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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を削除します。

mpfid で示された固定長メモリプールにおいて、メモリ獲得を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

削除によって、システムプールから割り付けられたメモリプール領域、およびリソースプールから割り付けられた管理領域が解放されます。

なお、返却されていないブロックがあっても、カーネルはそれに関して何も関知せずに削除処理を行います。

6.16.4 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	固定長メモリプール ID
VP	*p_blk	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	タイムアウト指定

■リターンパラメータ

ER	ercd	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_PAR	[p]	パラメータエラー
		(1)tmout≦-2
		(2)p_blk が 4 バイト境界でない
E_ID	[p]	不正 ID 番号
		(1)mpfid≦0
		(2)mpfid＞CFG_MAXMPFID
E_CTX	[k]	コンテキストエラー(get_mpf, tget_mpf のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_NOEXS	[k]	未登録
		(1)mpfid の固定長メモリプールが存在しない
E_RLWAI	[k]	待ち状態強制解除(get_mpf, tget_mpf のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除
		(1)mpfid の固定長メモリプールが削除された
E_MACV	[m]	メモリアクセス違反

■機能

mpfid で示される固定長メモリプールからひとつのメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを p_blk の指す領域に返します。

既にメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが対象となる固定長メモリプールに空きブロックが存在しない場合は、get_mpf, tget_mpf サービスコールでは呼び出しタスクはそのメモリプールのメモリ獲得の待ち行列につながれ、pget_mpf, ipget_mpf サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

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

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

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_blkに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=p_blk
- size=sizeof(VP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.16.5 固定長メモリブロックの返却(rel_mpf, irel_mpf)

■C 言語 API

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

■パラメータ

ID	mpfid	固定長メモリプール ID
VP	blk	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	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR		パラメータエラー
	[p]	(1)blk が 4 バイト境界でない
	[k]	(2)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
		(1)mpfid≦0
		(2)mpfid＞CFG_MAXMPFID
E_NOEXS	[k]	未登録
		(1)mpfid の固定長メモリプールが存在しない

■機能

mpfid で示された固定長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpf、pget_mpf, ipget_mpf または tget_mpf サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

対象固定長メモリプールでメモリブロックの獲得を待っているタスクがある場合、本サービスコールで返却したブロックを待ち行列先頭のタスクに割り付け、待ち状態を解除します。

6.16.6 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	固定長メモリプール ID
T_RMPF	*pk_rmpf	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	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 {
           ID      wtskid;      0      2    待ちタスク ID
           UINT    fblkcnt;     +4     4    空き領域のブロック数
} T_RMPF;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rmpf が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mpfid≦0 (2)mpfid＞CFG_MAXMPFID
E_NOEXS	[k]	未登録 (1)mpfid の固定長メモリプールが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および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、wtskid に TSK_NONE(0)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmpfに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmpf
- size=sizeof(T_RMPF)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.17 メモリプール管理(可変長メモリプール)機能

表6.32 メモリプール管理(可変長メモリプール)サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	cre_mpl	可変長メモリプールの生成	○		○	○	○		
	icre_mpl			○	○	○	○		
2	ivcra_mpl	可変長メモリプールの生成(アクセス許可ベクタ指定)	コンフィギュレータが生成する初期登録ルーチン専用です。その他の状態で呼び出した場合の動作は保証されません。						
3	acre_mpl	可変長メモリプールの生成	○		○	○	○		
	iacre_mpl	(ID 番号自動割付け)		○	○	○	○		
4	del_mpl	可変長メモリプールの削除	○		○	◯	○		
5	get_mpl	可変長メモリブロックの獲得	○		○		○		
6	pget_mpl	同上(ポーリング)	○		○	○	○		
	ipget_mpl			○	○	○	○		
7	tget_mpl	同上(タイムアウト有)	○		○		○		
8	rel_mpl	可変長メモリブロックの返却	○		○	○	○		
	irel_mpl			○	○	○	○		
9	ref_mpl	可変長メモリプールの状態参照	○		○	○	○		
	iref_mpl			○	○	○	○		

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

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

表6.33 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 32767)
2	可変長メモリプール属性	・TA_TFIFO：メモリ獲得待ちタスクのキューイングは FIFO 順 ・VTA_UNFRAGMENT：セクタ管理方式 (空き領域の断片化が発生しにくい方式) ・VTA_ALIGN16：メモリブロックアドレスを 16 バイト境界に調整 ・VTA_ALIGN32：メモリブロックアドレスを 32 バイト境界に調整

また、以下も参照してください。

■関連ページ 「4.31 メモリの断片化とその対策」

6.17.1 可変長メモリプールの生成(cre_mpl, icre_mpl, acre_mpl, iacre_mpl)

■C 言語 API

```
ER ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl);
ER ercd = icre_mpl(ID mplid, T_CMPL *pk_cmpl);
ER_ID mplid = acre_mpl(T_CMPL *pk_cmpl);
ER_ID mplid = iacre_mpl(T_CMPL *pk_cmpl);
```

■パラメータ

T_CMPL *pk_cmpl		可変長メモリプール生成情報を格納したパケットへのポインタ
《cre_mpl、icre_mpl》		
ID	mplid	可変長メモリプール ID

■リターンパラメータ

《cre_mpl、icre_mpl》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mpl、iacre_mpl》		
ER_ID	mplid	生成した可変長メモリプールの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mplatr;      0      4    可変長メモリプール属性
           SIZE    mplsz;       +4     4    メモリプール全体のサイズ(バイト数)
           VP      mpl;         +8     4    可変長メモリプール領域の先頭アドレス
           VP      mplmb;       +12    4    可変長メモリプール管理領域の先頭アドレス
           UINT    minblksz;    +16    4    最小ブロックサイズ
           UINT    sctnum;      +20    4    最大セクタ数
}T_CMPL;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)mplatr が不正
E_PAR	[p]	パラメータエラー (1)pk_cmpl が 4 バイト境界でない (2)mplsz＜TSZ_MPL(1,4) (3)mpl=NULL の場合で、mplsz＞CFG_SYSPOOLSZ (4)mpl!=NULL の場合で、mpl が 4 バイト境界でない (5)VTA_UNFRAGMENT 属性の場合で、minblksz が 0 (6)VTA_UNFRAGMENT かつ VTA_ALIGN16 属性の場合で、minblksz が 16 の倍数でない (7)VTA_UNFRAGMENT かつ VTA_ALIGN32 属性の場合で、minblksz が 32 の倍数でない (8)VTA_UNFRAGMENT 属性の場合で、かつ VTA_ALIGN16, VTA_ALIGN32 属性のいずれでもない場合で、minblksz が 4 の倍数以外

		(9)VTA_UNFRAGMENT 属性の場合で、sctnum=0
		(10)VTA_UNFRAGMENT 属性の場合で、mplsz<minblksz*32
E_ID	[p]	不正 ID 番号
		(1)mplid≦0
		(2)mplid＞CFG_MAXMPLID
E_NOMEM	[k]	メモリ不足
		(1)システムプール空き不足
		(2)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_mpl のみ)
E_OBJ	[k]	オブジェクト状態不正
		(1)mplid の可変長メモリプールが存在する
E_MACV	[m]	メモリアクセス違反

■機能

cre_mpl, icre_mpl サービスコールは、mplid で示された ID を持つ可変長メモリプールを、pk_cmpl で示された内容で生成します。

acre_mpl, iacre_mpl サービスコールは、未登録の可変長メモリプール ID を検索してその ID を持つ可変長メモリプールを pk_cmpl で示された内容で生成し、その ID をリターンパラメータとして返します。検索する可変長メモリプール ID の範囲は 1～CFG_MAXMPFID です。

(1) mplatr

mplatr には、以下の論理和を指定してください。

(a) メモリブロック獲得を待つ待ち行列に並ぶ際の並び方

TA_TFIFO のみを指定できます。

- TA_TFIFO(H'00000000)：メモリ獲得待ちタスクのキューイングは FIFO 順

(b) 管理方式

VTA_UNFRAGMENT を指定できます。

- VTA_UNFRAGMENT(H'80000000)：セクタ管理方式(空き領域の断片化が発生しにくい方式)

VTA_UNFRAGMENT 属性は、微小なメモリブロックを大量に獲得するメモリプールに適した属性で、微小なブロックをできるだけ連続して配置することで、大きなサイズの連続空き領域が維持されやすくします。

VTA_UNFRAGMENT 属性を指定した場合のみ、minblksz と sctnum が有効になります。sctnum に mplsz/(minblksz×32)よりも大きい値を指定した場合は、mplsz/(minblksz×32)として扱います。

詳細は、以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

また、VTA_UNFRAGMENT 属性を指定した場合は、セクタ管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.2(6) 可変長メモリプール」

(c) メモリブロックのアドレスのアライメント調整

メモリプールから獲得するメモリブロックのアドレスのアライメント調整について、必要なら以下のいずれかを指定できます。

- VTA_ALIGN16(H'00000010)：メモリブロックアドレスを 16 バイト境界に調整
- VTA_ALIGN32(H'00000020)：メモリブロックアドレスを 32 バイト境界に調整

いずれも指定しない場合は、メモリブロックのアドレスは 4 バイト境界になります。

(2) mplsz

mplsz には、生成する可変長メモリプール領域のサイズを指定します。なお、mplsz に指定すべきサイズの目安を知るために、以下のマクロが用意されています。

SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)
サイズが blksz バイトのメモリブロックを blkcnt 個獲得するのに必要な可変長メモリプール領域のサイズ(目安のバイト数)

なお、mplsz は 4 の倍数に切上げて扱われます。以降の説明では、mplsz は 4 の倍数に切上げ後の値を意味することとします。

(3) mpl

mpl には、可変長メモリプールとして使用する空き領域の先頭アドレスを指定します。mpl から mplsz バイトを固定長メモリプールとして使用します。ただし、VTA_ALIGN16 または VTA_ALIGN32 が指定された場合は、実際にメモリプールとして使用されるのは mpl をそれぞれ 16 または 32 の境界に補正したアドレスからになります。この補正の分だけ実際に使用できるメモリプールサイズも減ることになります。

なお、指定された領域が、どのドメインからアクセスできるかは、カーネルは関知しません。例えば、P1, P2 領域のアドレスを指定した場合、その領域はユーザドメインからはアクセスできませんが、カーネルはこれを関知しません。

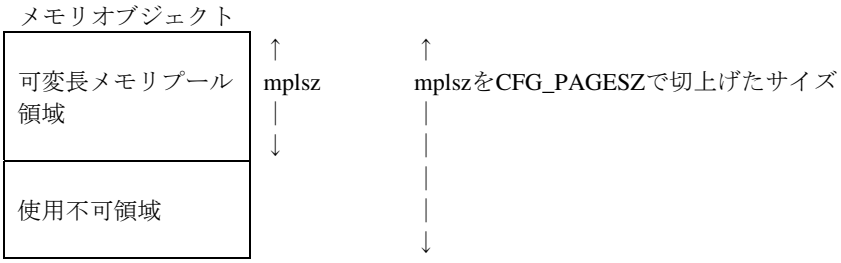
◎メモリオブジェクト保護機能を組み込んだ場合
指定範囲は、カーネルドメインからリード・ライトアクセス可能な領域でなければなりません。これに違反する場合は、E_MACVエラーを返します。

mpl に NULL を指定すると、カーネルはシステムプールから mplsz バイトのメモリプール領域を割り付けます。また、カーネルは割り付けたメモリプール領域の管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.2(6) 可変長メモリプール」
・システムプールの消費→「14.2(3) 可変長メモリプール領域」

◎メモリオブジェクト保護機能を組み込んだ場合
カーネルがシステムプールから割り付けるメモリプール領域は、以下の性質のメモリオブジェクトとなります。

(1)サイズ：mplszをCFG_PAGESZで切上げたサイズ
　ただし、メモリプールとして使用できるサイズは、mplzバイトです。下の図で、使用不可領域は可変長メモリプール領域と同じメモリオブジェクト内なので、アクセス許可の区別はありませんが、メモリプールとしては管理されません。



(2)所属ドメイン：タスクコンテキストから呼び出した場合は、呼び出したタスクが所属するドメインになります。これは、get_didを発行して得られるドメインIDと同じです。非タスクコンテキストから呼び出した場合はカーネルドメインになります
(3)メモリ属性：TA_RW|TA_CACHE| TA_WBACK
(4)アクセス許可ベクタ：所属ドメインのみがリード・ライト可能なように設定されます。
　所属ドメインがカーネルドメインの場合：TACT_KERNEL
　所属ドメインがユーザドメインの場合：TACT_PRW(domid)
　　(domidは対象タスクの所属ドメインID)

(4) minblksz と sctnum

これらはμITRON 仕様外の項目です。
これらは VTA_UNFRAGMENT 属性が指定された場合のみ有効です。詳細は、前述の VTA_UNFRAGMENT 属性の説明を参照してください。

(5) mplmb

mplmb はμITRON 仕様外の項目です。
mplmb は、本カーネルでは単に無視されます。プログラムの移植性のためには、mplmb に NULL を指定してください。

なお、可変長メモリプールはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmplに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmpl
- size=sizeof(T_CMPL)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_cmpl->mpl != NULLの場合で、mplからmplszバイトの領域に対し、カーネルのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmpl->mpl
- size= pk_cmpl->mplsz
- domid=カーネルドメイン
- pmmode=TPM_READ|TPM_WRITE

6.17.2 可変長メモリプールの生成(アクセス許可ベクタ指定)(ivcra_mpl)

■C 言語 API

```
ER ercd = ivcra_mpl (ID mplid, T_CMPL *pk_cmpl, ACVCT *p_acvct);
```

■パラメータ

ID	mplid	可変長メモリプール ID
T_CMPL	*pk_cmpl	可変長メモリプール生成情報を格納したパケットのポインタ
ACVCT	*p_acvct	アクセス許可ベクタ情報を格納したパケットのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mplatr;      0     4    可変長メモリプール属性
           SIZE    mplsz;      +4     4    メモリプールサイズ(バイト数)
           VP      mpl;        +8     4    可変長メモリプール領域の先頭アドレス
           VP      mplmb;      +12    2    可変長メモリプール管理領域の先頭アドレス
           UINT    minblksz;   +16    4    最小ブロックサイズ
           UINT    sctnum;     +20    4    最大セクタ数
} T_CMPL;
typedef    struct{
           ACPTN   acptn1;      0     4    ライトアクセスに関するアクセス許可パターン
           ACPTN   acptn2;     +4     4    リードアクセスに関するアクセス許可パターン
} ACVCT;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)mplsz＞CFG_SYSPOOLSZ
E_NOMEM	[k]	メモリ不足 (1)システムプールの空き不足 (2)リソースプールの空き不足

■機能

mplid で示された ID を持つ可変長メモリプールを、pk_cmpl, p_acvct で示された内容で生成します。

本サービスコールは、アプリケーションから呼び出してはなりません。本サービスコールは、メモリオブジェクト保護機能を組み込んだ場合で、コンフィギュレータで可変長メモリプールの生成を行っていた場合に、コンフィギュレータが出力する初期登録ルーチンのみから呼び出されます。本サービスコールはこの前提で実装されているため、ほとんどのエラー検出を省略しています。

cre_mpl との主な相違点は、以下の通りです。

(1) pk_cmpl->mplはNULLのみ許可されます。つまり、メモリプール領域はシステムプールから割り付けられます。メモリプール領域のアドレスを指定することはできません。

(2) p_acvctによって、生成したメモリプール領域のメモリオブジェクトとしてのアクセス許可ベクタを指定できます。ただし、メモリ保護機能を組み込んでいない場合は、p_acvctは無視されます。

(3) E_MACVエラーは検出されません。

6.17.3 可変長メモリプールの削除(del_mpl)

■C 言語 API

```
ER ercd = del_mpl(ID mplid);
```

■パラメータ

ID	mplid	可変長メモリプール ID

■リターンパラメータ

ER	ercd	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 (1)mplid≦0 (2)mplid＞CFG_MAXMPLID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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を削除します。

mplid で示された可変長メモリプールにおいて、メモリ獲得を待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

削除によって、システムプールから割り付けられたメモリプール領域、およびリソースプールから割り付けられた管理領域が解放されます。

なお、返却されていないブロックがあっても、カーネルはそれに関して何も処理しません。エラーも報告しません。

6.17.4 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	可変長メモリプール ID
UINT	blksz	メモリブロックサイズ(バイト数)
VP	*p_blk	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	タイムアウト指定

■リターンパラメータ

ER	ercd	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_PAR		パラメータエラー
	[p]	(1)tmout≦-2
		(2)blksz = 0
		(3)p_blk が 4 バイト境界でない
	[k]	(4)blksz > (生成時に指定したメモリプールサイズ)
E_ID	[p]	不正 ID 番号
		(1)mplid≦0
		(2)mplid＞CFG_MAXMPLID
E_CTX	[k]	コンテキストエラー(get_mpl, tget_mpl のみ)
		(1)ディスパッチ保留状態からの呼び出し
E_NOMEM	[k]	メモリ不足
		(1)リソースプールの空き不足
E_NOEXS	[k]	未登録
		(1)mplid の可変長メモリプールが存在しない
E_RLWAI	[k]	待ち状態強制解除(get_mpl, tget_mpl のみ)
		(1)待ちの間に rel_wai サービスコールが呼び出された
		(2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除(mplid の可変長メモリプールが削除された)
E_MACV	[m]	メモリアクセス違反

■機能

mplid で示される可変長メモリプールから、blksz (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。

blksz は、表 6.34　blksz の切り上げのように切上げて扱われます。

表6.34　blksz の切り上げ

VTA_ALIGN16 属性	VTA_ALIGN32 属性	VTA_UNFRAGMENT 属性	blksz の切り上げ
×	×	×	4 の倍数に切り上げ
○	×	×	16 の倍数に切り上げ
×	○	×	32 の倍数に切り上げ
×	×	○	(1)blksz≦(最小ブロックサイズ×8)の場合 最少ブロックサイズの２のべき乗倍に切上げ (2)blksz＞(最小ブロックサイズ×8)の場合 4 の倍数に切上げ
○	×	○	(1)blksz≦(最小ブロックサイズ×8)の場合 最少ブロックサイズの２のべき乗倍に切上げ (2)blksz＞(最小ブロックサイズ×8)の場合 16 の倍数に切上げ
×	○	○	(1)blksz≦(最小ブロックサイズ×8)の場合 最少ブロックサイズの２のべき乗倍に切上げ (2)blksz＞(最小ブロックサイズ×8)の場合 32 の倍数に切上げ

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領域はこの切上げ後の blksz だけ減少します。

また、VTA_ALIGN16, VTA_ALIGN32 属性のメモリプールの場合は、それぞれメモリブロックのアドレスは 16, 32 バイト境界のアドレスになります。

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の可変長メモリプールに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。

カーネルは、メモリブロックの管理のためにリソースプールを消費します。リソースプールの空きが不足している場合は、直ちに E_NOMEM エラーが返ります。この処理は、直ちにメモリブロックを獲得できるか、待ち状態に移行するかに関わらず、行われます。

リソースプールの消費については、以下を参照してください。

■関連ページ　「13.2.3(2) 可変長メモリプール：get_mpl, pget_mpl, ipget_mpl, tget_mpl」

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_blkに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_blk
- size=sizeof(VP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.17.5 可変長メモリブロックの返却(rel_mpl, irel_mpl)

■C 言語 API

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

■パラメータ

ID	mplid	可変長メモリプール ID
VP	blk	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	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR		パラメータエラー
	[p]	(1)blk が 4 バイト境界でない
	[k]	(2)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
		(1)mplid≦0
		(2)mplid＞CFG_MAXMPLID
E_NOEXS	[k]	未登録
		(1)mplid の可変長メモリプールが存在しない

■機能

mplid で示された可変長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpl、pget_mpl, ipget_mpl または tget_mpl サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

メモリブロックの返却によって、対象可変長メモリプールに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。

また、メモリブロックの返却により、メモリブロック獲得時にカーネルがメモリブロック管理のためにリソースプールから確保した領域が解放されます。

6.17.6 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	可変長メモリプール ID
T_RMPL	*pk_rmpl	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	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 {
           ID      wtskid;      0      2    待ちタスク ID
           SIZE    fmplsz;      +4     4    空き領域の合計サイズ(バイト数)
           UINT    fblksz;      +8     4    獲得可能な最大メモリブロックサイズ(バイト数)
           SIZE    mplsz;       +12    4    可変長メモリプールのサイズ
} T_RMPL;
```

■エラーコード

E_PAR	[p]	パラメータエラー(pk_rmpl が 4 バイト境界でない)
E_ID	[p]	不正 ID 番号 (1)mplid≦0 (2)mplid＞CFG_MAXMPLID
E_NOEXS	[k]	未登録 (1)mplid の可変長メモリプールが存在しない
E_MACV	[m]	メモリアクセス違反

■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)、および可変長メモリプールのサイズ(mplsz : μITRON 仕様外の項目)を返します。対象メモリプールに待ちタスクが無い場合は、wtskid に TSK_NONE(0)を返します。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。1 回の get_mpl、pget_mpl, ipget_mpl または tget_mpl サービスコールで、fblksz までのブロックを即座に獲得できます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmplに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmpl
- size=sizeof(T_RMPL)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.18 時間管理機能(システム時刻管理)

表6.35 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 大文字の"[S]"はスタンダードプロファイルのサービスコール、小文字の"[s]"はスタンダードプロファイルではありませんが、スタンダードプロファイルの機能を使用するために必要となるサービスコールです。

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

表6.36 システム時刻管理の仕様

項番	項目	内容
1	システム時刻値	符号なし 48 ビット
2	システム時刻の単位	1[ms]
3	システム時刻の更新周期	CFG_TICNUME/CFG_TICDENO[ms]*
4	システム時刻初期値(初期起動時)	H'000000000000

【注】* kernel_macro.h に定義される TIC_NUME, TIC_DENO は、それぞれ CFG_TICNUME, CFG_TICDENO と同じ値です。

システム時刻は SYSTIM 型の構造体によって符号無し 48bit 整数として表現されますが、その最大値は以下のようになります。

[CFG_TICNUME/CFG_TICDENO ≦ 1 の場合]
最大値＝H'ffffffffffff/CFG_TICDENO

[CFG_TICNUME/CFG_TICDENO ＞ 1 の場合]
最大値＝H'ffffffffffff

システム時刻は、タイマ割込みによってインクリメントされます。この時に、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。

6.18.1 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	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	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct{
           UH      utime;      0      2    システムの現在時刻(上位)
           UW      ltime;      +4     4    システムの現在時刻(下位)
} SYSTIM;
```

■エラーコード

E_PAR	[p]	パラメータエラー(p_systim が 4 バイト境界でない)
E_MACV	[m]	メモリアクセス違反

■機能

システムが保持しているシステム時刻の現在の値を、p_systim で示される値に設定します。

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

なお、システム時刻を変更しても、それまでに要求されていた時間イベント(タイムアウトや周期ハンドラの起動など)が発生するまでの実時間は変わりません。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_systimに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=p_systim
- size=sizeof(SYSTIM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.18.2 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	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	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{
           UH      utime;      0      2    システムの現在時刻(上位)
           UW      ltime;      +4     4    システムの現在時刻(下位)
} SYSTIM;
```

■エラーコード

E_PAR	[p]	パラメータエラー(p_systim が 4 バイト境界でない)
E_MACV	[m]	メモリアクセス違反

■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。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_systimに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_systim
- size=sizeof(SYSTIM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.19 時間管理機能(周期ハンドラ)

表6.37 周期ハンドラサービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	cre_cyc [s]	周期ハンドラの生成	○		○	○	○		
	icre_cyc			○	○	○	○		
2	acre_cyc	周期ハンドラの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_cyc			○	○	○	○		
3	del_cyc	周期ハンドラの削除	○		○	○	○		
4	sta_cyc [S]	周期ハンドラの動作開始	○		○	○	○		
	ista_cyc			○	○	○	○		
5	stp_cyc [S]	周期ハンドラの動作停止	○		○	○	○		
	istp_cyc			○	○	○	○		
6	ref_cyc	周期ハンドラの状態参照	○		○	○	○		
	iref_cyc			○	○	○	○		

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

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

表6.38 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	周期ハンドラ属性	・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

6.19.1 周期ハンドラの生成(cre_cyc, icre_cyc, acre_cyc, iacre_cyc)

■C 言語 API

```
ER ercd = cre_cyc(ID cycid, T_CCYC *pk_ccyc);
ER ercd = icre_cyc(ID cycid, T_CCYC *pk_ccyc);
ER_ID cycid = acre_cyc(T_CCYC *pk_ccyc);
ER_ID cycid = iacre_cyc(T_CCYC *pk_ccyc);
```

■パラメータ

T_CCYC	*pk_ccyc	周期ハンドラ生成情報を格納したパケットへのポインタ
《cre_cyc、icre_cyc》		
ID	cycid	周期ハンドラ ID

■リターンパラメータ

《cre_cyc、icre_cyc》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_cyc、iacre_cyc》		
ER_ID	cycid	生成した周期ハンドラの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     cycatr;     0      4    周期ハンドラ属性
           VP_INT  exinf;      +4     4    拡張情報
           FP      cychdr;     +8     4    周期ハンドラアドレス
           RELTIM  cyctim;     +12    4    周期ハンドラの起動周期
           RELTIM  cycphs;     +16    4    周期ハンドラの起動位相
} T_CCYC;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)cycatr が不正
E_PAR	[p]	パラメータエラー (1)cyctim=0 (2)cycphs>cyctim (3)pk_ccyc が 4 バイト境界でない (4)cychdr が奇数
E_ID	[p]	不正 ID 番号 (1)cycid ≦ 0 (2)cycid＞CFG_MAXCYCID
E_NOID	[k]	空き ID なし(acre_cyc のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)cycid の周期ハンドラが存在する
E_MACV	[m]	メモリアクセス違反

■機能

周期ハンドラを生成します。周期ハンドラは、一定周期間隔で実行される非タスクコンテキストのタイムイベントハンドラです。周期ハンドラはカーネルドメインに所属し、特権モードで実行されます。

以下に各パラメータの意味を示します。

(1) cycid

cre_cyc, icre_cyc サービスコールの場合、cycid には 1～CFG_MAXCYCID の値を指定します。

acre_cyc, iacre_cyc サービスコールの場合、未登録の周期ハンドラ ID を検索してその ID を持つ周期ハンドラを pk_ccyc で示された内容で生成し、その ID をリターンパラメータとして返します。

(2) cycatr

cycatr には、以下の論理和を指定してください。

なお、周期ハンドラは常にカーネルドメインに所属します。

PX 仕様では、カーネルドメインへの所属の指定として、cycatr に TA_DOM(TDOM_KERNEL)を指定する仕様となっているので、プログラムの移植性のためには TA_DOM(TDOM_KERNEL)を指定してください。なお、本カーネルは TA_DOM()の指定は単に無視します。TA_DOM()でユーザドメイン所属の指定を行ってもエラーにはならず、常にカーネルドメイン所属と扱います。

(a) 記述言語

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(b) 周期ハンドラの動作開始

ハンドラを直ちに動作開始にするには、TA_STA を指定してください。この場合、周期ハンドラを生成した後に、周期ハンドラを動作している状態にします。指定されていない場合は、sta_cyc, ista_cyc サービスコールが呼び出されるまで周期ハンドラは動作しません。

- TA_STA(H'00000002)：周期ハンドラの動作開始

(c) 起動位相の保存

TA_PHS が指定された場合は、周期ハンドラの動作を開始する時に、周期ハンドラの起動位相を保存して、次に起動すべき時刻を決定します。

TA_PHS 属性が指定されていない場合は、一旦 stp_cyc, istp_cyc サービスコールで動作が停止されると、次回の sta_cyc, ista_cyc サービスコール発行時点から cyctim の周期で起動します。つまり、最初の起動時刻は sta_cyc, ista_cyc 発行時点から cyctim 後となります。

一方、TA_PHS 属性を指定した場合は、一旦 stp_cyc, istp_cyc サービスコールで動作が停止されても、カーネル内部で「起動すべき」時刻を継続して管理します。その後、sta_cyc, ista_cyc が発行されると、継続管理している時刻に達したときに周期ハンドラが起動されます。

- TA_PHS(H'00000004)：起動位相の保存

(3) exinf

exinf は、生成する周期ハンドラに関する情報を設定するなどの目的で自由に使用できます。

exinf は、周期ハンドラにパラメータとして渡されます。

(4) cychdr

cychdr には、周期ハンドラの先頭アドレスを指定します。

(5) cyctim と cycphs

cyctim には周期ハンドラの起動周期を、cycphs には周期ハンドラの起動位相を指定します。

cycphs は、TA_STA 属性と TA_PHS 属性の少なくとも一方の指定がある場合のみ有効です。この場合、周期ハンドラを生成するサービスコールが呼び出されてから、指定した cycphs(起動位相)以上の時間が経過した時が、最初に起動すべき時刻となります。その時点で周期ハンドラが起動済み(TA_STA 属性、または stp_cyc, istp_cyc サービスコール)の場合は、周期ハンドラが実行されます。

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

なお、周期ハンドラはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_ccycに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_ccyc
- size=sizeof(T_CCYC)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_ccyc->cychdrに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_ccyc->cychdr
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.19.2 周期ハンドラの削除(del_cyc)

■C 言語 API

```
ER ercd = del_cyc(ID cycid);
```

■パラメータ

ID	cycid	周期ハンドラ ID

■リターンパラメータ

ER	ercd	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 (1)cycid ≦ 0 (2)cycid＞CFG_MAXCYCID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)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削除します。

6.19.3 周期ハンドラの動作開始(sta_cyc, ista_cyc)

■C 言語 API

```
ER ercd = sta_cyc(ID cycid);
ER ercd = ista_cyc(ID cycid);
```

■パラメータ

ID	cycid	周期ハンドラ ID

■リターンパラメータ

ER	ercd	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 (1)cycid ≦ 0 (2)cycid＞CFG_MAXCYCID
E_NOEXS	[k]	未登録 (1)cycid の周期ハンドラが存在しない

■機能

cycid で示された周期ハンドラを、動作している状態に移行させます。

周期ハンドラ属性に TA_PHS が指定されていない場合には、このサービスコールが呼び出された時刻を基準として、その時刻から起動周期が経過する毎に、周期ハンドラが起動されます。

TA_PHS が指定されていない動作している状態の周期ハンドラが指定された場合は、周期ハンドラを次に起動する時刻の再設定のみを行います。

TA_PHS が指定されている場合は、周期ハンドラ生成時点の時刻を基準に起動するため、時刻の設定は行いません。

6.19.4 周期ハンドラの動作停止(stp_cyc, istp_cyc)

■C 言語 API

```
ER ercd = stp_cyc(ID cycid);
ER ercd = istp_cyc(ID cycid);
```

■パラメータ

ID	cycid	周期ハンドラ ID

■リターンパラメータ

ER	ercd	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 (1)cycid ≦ 0 (2)cycid＞CFG_MAXCYCID
E_NOEXS	[k]	未登録 (1)cycid の周期ハンドラが存在しない

■機能

cycid で示された周期ハンドラを、動作していない状態に移行させます。

6.19.5 周期ハンドラの状態参照(ref_cyc, iref_cyc)

■C 言語 API

```
ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc);
ER ercd = iref_cyc(ID cycid, T_RCYC *pk_rcyc);
```

■パラメータ

ID	cycid	周期ハンドラ ID
T_RCYC	*pk_rcyc	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	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 {
           STAT    cycstat;     0      4    周期ハンドラの動作状態
           RELTIM  lefttim;     +4     4    周期ハンドラ起動までの残り時間
} T_RCYC;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rcyc が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)cycid ≦ 0 (2)cycid＞CFG_MAXCYCID
E_NOEXS	[k]	未登録 (1)cycid の周期ハンドラが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

cycid で示された周期ハンドラの状態を参照します。

pk_rcyc が指す領域に周期ハンドラの動作状態(cycstat)、および周期ハンドラ起動までの残り時間(lefttim)を返します。

cycstat には、対象周期ハンドラの動作状態を返します。

- TCYC_STP(H'00000000)周期ハンドラが動作していない
- TCYC_STA(H'00000001)周期ハンドラが動作している

lefttim には、対象周期ハンドラを次に起動する時刻までの相対時間を返します。対象周期ハンドラが動作していない場合、lefttim は不定値となります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rcycに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rcyc
- size=sizeof(T_RCYC)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.20 時間管理機能(アラームハンドラ)

表6.39 アラームハンドラサービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	cre_alm	アラームハンドラの生成	○		○	○	○		
	icre_alm			○	○	○	○		
2	acre_alm	アラームハンドラの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_alm			○	○	○	○		
3	del_alm	アラームハンドラの削除	○		○	○	○		
4	sta_alm	アラームハンドラの動作開始	○		○	○	○		
	ista_alm			○	○	○	○		
5	stp_alm	アラームハンドラの動作停止	○		○	○	○		
	istp_alm			○	○	○	○		
6	ref_alm	アラームハンドラの状態参照	○		○	○	○		
	iref_alm			○	○	○	○		

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

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

表6.40 アラームハンドラの仕様

項番	項目	内容
1	アラームハンドラ ID	1～CFG_MAXALMID (最大 255)
2	アラームハンドラ属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述

6.20.1 アラームハンドラの生成(cre_alm, icre_alm, acre_alm, iacre_alm)

■C 言語 API

```
ER ercd = cre_alm(ID almid, T_CALM *pk_calm);
ER ercd = icre_alm(ID almid, T_CALM *pk_calm);
ER_ID almid = acre_alm(T_CALM *pk_calm);
ER_ID almid = iacre_alm(T_CALM *pk_calm);
```

■パラメータ

T_CALM	*pk_calm	アラームハンドラ生成情報を格納したパケットへのポインタ
《cre_alm、icre_alm》		
ID	almid	アラームハンドラ ID

■リターンパラメータ

《cre_alm、icre_alm》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_alm、iacre_alm》		
ER_ID	almid	生成したアラームハンドラの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     almatr;      0      4    アラームハンドラ属性
           VP_INT  exinf;       +4     4    拡張情報
           FP      almhdr;      +8     4    アラームハンドラアドレス
} T_CALM;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)almatr が不正
E_PAR	[p]	パラメータエラー (1)pk_calm が 4 バイト境界でない (2)almhdr が奇数
E_ID	[p]	不正 ID 番号 (1)almid ≦ 0 (2)almid＞CFG_MAXALMID
E_NOID	[k]	空き ID なし(acre_alm のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)almid のアラームハンドラが存在する
E_MACV	[m]	メモリアクセス違反

■機能

アラームハンドラを生成します。アラームハンドラは、指定された時刻に一度だけ実行される非タスクコンテキストのタイムイベントハンドラです。アラームハンドラはカーネルドメインに所属し、特権モードで実行されます。

以下に各パラメータの意味を示します。

(1) almid

cre_alm, icre_alm サービスコールの場合、almid には 1～CFG_MAXALMID の値を指定します。
acre_alm, iacre_alm サービスコールの場合、未登録のアラームハンドラ ID を検索してその ID を持つアラームハンドラを pk_calm で示された内容で生成し、その ID をリターンパラメータとして返します。

(2) almatr

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

なお、アラームハンドラは常にカーネルドメインに所属します。
PX 仕様では、カーネルドメインへの所属の指定として、almatr に TA_DOM(TDOM_KERNEL)を指定する仕様となっているので、プログラムの移植性のためには TA_DOM(TDOM_KERNEL)を指定してください。なお、本カーネルは TA_DOM()の指定は単に無視します。TA_DOM()でユーザドメイン所属の指定を行ってもエラーにはならず、常にカーネルドメイン所属と扱います。

(3) exinf

exinf は、生成するアラームハンドラに関する情報を設定するなどの目的で自由に使用できます。
exinf は、アラームハンドラにパラメータとして渡されます。

アラームハンドラの生成直後は、アラームハンドラの起動時刻は設定されていません。アラームハンドラは停止状態となっています。

なお、アラームハンドラはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_calmに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_calm
- size=sizeof(T_CALM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_calm->almhdrに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_calm->almhdr
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.20.2 アラームハンドラの削除(del_alm)

■C 言語 API

```
ER ercd = del_alm(ID almid);
```

■パラメータ

ID	almid	アラームハンドラ ID

■リターンパラメータ

ER	ercd	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 (1)almid ≦ 0 (2)almid＞CFG_MAXALMID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)almid のアラームハンドラが存在しない

■機能

almid で示されたアラームハンドラを削除します。

6.20.3 アラームハンドラの動作開始(sta_alm, ista_alm)

■C 言語 API

```
ER ercd = sta_alm(ID almid, RELTIM almtim);
ER ercd = ista_alm(ID almid, RELTIM almtim);
```

■パラメータ

ID	almid	アラームハンドラ ID
RELTIM	almtim	アラームハンドラの起動時刻

■リターンパラメータ

ER	ercd	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 (1)almid ≦ 0 (2)almid＞CFG_MAXALMID
E_NOEXS	[k]	未登録 (1)almid のアラームハンドラが存在しない

■機能

almid で示されたアラームハンドラの起動時刻を、サービスコールが呼び出された時刻から almtim で指定された相対時間後に設定し、アラームハンドラの動作を開始します。

すでに動作しているアラームハンドラが指定された場合は、以前の起動時刻の設定を解除し、新しい起動時刻を設定します。

almtim に 0 が指定された場合は、次のタイムティックでアラームハンドラが起動されます。

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

6.20.4 アラームハンドラの動作停止(stp_alm, istp_alm)

■C 言語 API

```
ER ercd = stp_alm(ID almid);
ER ercd = istp_alm(ID almid);
```

■パラメータ

ID	almid	アラームハンドラ ID

■リターンパラメータ

ER	ercd	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 (1)almid ≦ 0 (2)almid＞CFG_MAXALMID
E_NOEXS	[k]	未登録 (1)almid のアラームハンドラが存在しない

■機能

almid で示されたアラームハンドラの起動時刻の設定を解除し、アラームハンドラの動作を停止します。

6.20.5 アラームハンドラの状態参照(ref_alm, iref_alm)

■C 言語 API

```
ER ercd = ref_alm(ID almid, T_RALM *pk_ralm);
ER ercd = iref_alm(ID almid, T_RALM *pk_ralm);
```

■パラメータ

ID	almid	アラームハンドラ ID
T_RALM	*pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RALM	*pk_ralm	アラームハンドラ状態を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct {
           STAT    almstat;     0      4    アラームハンドラの動作状態
           RELTIM  lefttim;     +4     4    アラームハンドラ起動までの残り時間
} T_RALM;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_ralm が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)almid ≦ 0 (2)almid＞CFG_MAXALMID
E_NOEXS	[k]	未登録 (1)almid のアラームハンドラが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

almid で示されたアラームハンドラの状態を参照します。

pk_ralm が指す領域にアラームハンドラの動作状態(almstat)、およびアラームハンドラ起動までの残り時間(lefttim)を返します。

almstat には、対象アラームハンドラの動作状態を返します。

- TALM_STP(H'00000000)アラームハンドラが動作していない
- TALM_STA(H'00000001)アラームハンドラが動作している

lefttim には、対象アラームハンドラ起動までの相対時間を返します。対象アラームハンドラが動作していない場合、lefttim は不定値となります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_ralmに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_ralm
- size=sizeof(T_RALM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.21 時間管理機能(オーバーランハンドラ)

表6.41 オーバーランハンドラサービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_ovr	オーバーランハンドラの定義	○		○	○	○		
2	sta_ovr	オーバーランハンドラの動作開始	○		○	○	○		
	ista_ovr			○	○	○	○		
3	stp_ovr	オーバーランハンドラの動作停止	○		○	○	○		
	istp_ovr			○	○	○	○		
4	ref_ovr	オーバーランハンドラの状態参照	○		○	○	○		
	iref_ovr			○	○	○	○		

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

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

表6.42 オーバーランハンドラの仕様

項番	項目	内容
1	プロセッサ時間の単位(OVRTIM)	システム時刻と同じ(1[ms])
2	オーバーランハンドラ属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述

オーバーランハンドラは、システムに 1 つだけ定義できるタイムイベントハンドラです。

タスクが使用したプロセッサ時間には、タスクとタスクが呼び出したサービスコール、そのタスクの実行中に起動された割込みハンドラの各実行時間が含まれます。

タスクが実行状態以外の間は、使用プロセッサ時間はカウントされません。

6.21.1 オーバーランハンドラの定義(def_ovr)

■C 言語 API

```
ER ercd = def_ovr(T_DOVR *pk_dovr);
```

■パラメータ

T_DOVR	*pk_dovr	オーバーランハンドラ定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     ovratr;      0      4    オーバーランハンドラ属性
           FP      ovrhdr;      +4     4    オーバーランハンドラアドレス
} T_DOVR;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)ovratr の TA_ASM 以外のビットが 0 でない
E_PAR	[p]	パラメータエラー (1)pk_dovr が 4 バイト境界でない (2)ovrhdr が奇数
E_MACV	[m]	メモリアクセス違反

■機能

オーバーランハンドラを定義します。オーバーランハンドラは、タスクが設定された時間を越えてプロセッサを使用した場合に実行される非タスクコンテキストのタイムイベントハンドラです。オーバーランハンドラはカーネルドメインに所属し、特権モードで実行されます。

以下に各パラメータの意味を示します。

(1) ovratr

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(2) ovrhdr

ovrhdr には、オーバーランハンドラの先頭アドレスを指定します。

pk_dovr=NULL(0)が指定された場合は、オーバーランハンドラの定義を解除します。

また、すでにオーバーランハンドラが定義されている状態で、本サービスコールが呼び出された場合は、以前の定義を解除し、新しい定義に置き換えます。

本サービスコールでは非タスクコンテキストから呼び出してはなりませんが、E_CTX エラーは検出されません。

なお、オーバーランハンドラはコンフィギュレータで静的に定義することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dovrに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dovr
- size=sizeof(T_DOVR)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dovr->ovrhdrに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dovr->ovrhdr
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.21.2 オーバーランハンドラの動作開始(sta_ovr, ista_ovr)

■C 言語 API

```
ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim);
ER ercd = ista_ovr(ID tskid, OVRTIM ovrtim);
```

■パラメータ

ID	tskid	タスク ID
OVRTIM	ovrtim	上限プロセッサ時間

■リターンパラメータ

ER	ercd	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)オーバーランハンドラが定義されていない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクに対して、オーバーランハンドラの動作を開始します。

tskid=TSK_SELF(0)の指定により自タスクの指定になります。

対象タスクの上限プロセッサ時間を ovrtim で指定される時間に設定し、使用プロセッサ時間を 0 クリアします。すでに動作しているオーバーランハンドラが指定された場合は、以前の上限プロセッサ時間の設定を解除し、新しい上限プロセッサ時間を設定します。

使用プロセッサ時間が上限プロセッサ時間を超えたときに、オーバーランハンドラが起動されます。

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

ovrtim に 0 が指定された場合は、対象タスクがプロセッサを使用してから、1 回目のタイムティックでオーバーランハンドラが起動されます。

6.21.3 オーバーランハンドラの動作停止(stp_ovr, istp_ovr)

■C 言語 API

```
ER ercd = stp_ovr(ID tskid);
ER ercd = istp_ovr(ID tskid);
```

■パラメータ

ID	tskid	タスク ID

■リターンパラメータ

ER	ercd	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 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)オーバーランハンドラが定義されていない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない

■機能

tskid で示されたタスクに対して、上限プロセッサ時間の設定を解除し、オーバーランハンドラの動作を停止します。

tskid=TSK_SELF(0)の指定により、自タスクの指定になります。

6.21.4 オーバーランハンドラの状態参照(ref_ovr, iref_ovr)

■C 言語 API

```
ER ercd = ref_ovr(ID tskid, T_ROVR *pk_rovr);
ER ercd = iref_ovr(ID tskid, T_ROVR *pk_rovr);
```

■パラメータ

ID	tskid	タスク ID
T_ROVR	*pk_rovr	オーバーランハンドラの状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_ROVR	*pk_rovr	オーバーランハンドラの状態を格納したパケットのへのポインタ

■パケットの構造

```
typedef    struct {
           STAT    ovrstat;     0      4    オーバーランハンドラの動作状態
           OVRTIM  leftotm;     +4     4    残りのプロセッサ時間
} T_ROVR;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rovr が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)tskid＜0 (2)tskid＞CFG_MAXTSKID (3)非タスクコンテキストで tskid=TSK_SELF(0)を指定
E_OBJ	[k]	オブジェクト状態不正 (1)オーバーランハンドラが定義されていない
E_NOEXS	[k]	未登録 (1)tskid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

tskid で示されたタスクのオーバーランハンドラの状態を参照します。tskid=TSK_SELF(0)の指定により、自タスクの指定になります。

pk_rovr が指す領域にオーバーランハンドラの動作状態(ovrstat)、および残りのプロセッサ時間(leftotm)を返します。

ovrstat には、対象オーバーランハンドラの動作状態として、上限プロセッサ時間の設定状態を返します。

- TOVR_STP(H'00000000)上限プロセッサ時間が設定されていない
- TOVR_STA(H'00000001)上限プロセッサ時間が設定されている

leftotm には、対象タスクを原因としてオーバーランハンドラが起動されるまでの残りプロセッサ時間を返します。対象タスクに上限プロセッサ時間が設定されていない場合、leftotm は不定値となります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rovrに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rovr
- size=sizeof(T_ROVR)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.22 システム状態管理機能

表6.43 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	get_did		実行状態のタスクの所属ドメイン ID の参照	○		○	○	○		
	iget_did				◯	◯	◯	◯		
4	loc_cpu	[S]	CPU ロック状態への移行	○		○	○	○	○	
	iloc_cpu	[S]			○	○	○	○	○	
5	unl_cpu	[S]	CPU ロック状態の解除	○		○	○	○	○	
	iunl_cpu	[S]			○	○	○	○	○	
6	dis_dsp	[S]	ディスパッチの禁止	○		○	○	○		
7	ena_dsp	[S]	ディスパッチの許可	○		○	○	○		
8	sns_ctx	[S]	コンテキストの参照	○	○	○	○	○	○	
9	sns_loc	[S]	CPU ロック状態の参照	○	○	○	○	○	○	
10	sns_dsp	[S]	ディスパッチ禁止状態の参照	○	○	○	○	○	○	
11	sns_dpn	[S]	ディスパッチ保留状態の参照	○	○	○	○	○	○	
12	vsta_knl	[s]	カーネルの起動	○	○	○	○	○	○	○
	ivsta_knl	[s]		○	○	○	○	○	○	○
13	vsys_dwn	[s]	システムダウン	○	○	○	○	○	○	○
	ivsys_dwn	[s]		○	○	○	○	○	○	○
14	vget_trc		トレースの取得	○		○	○	○		
	ivget_trc				○	○	○	○		
15	ivbgn_int		割込みハンドラの開始をトレースに取得		○	○	○	○		
16	ivend_int		割込みハンドラの終了をトレースに取得		○	○	○	○		
17	vchg_cop		DSP(TA_COP0)属性の変更	○		○	○	○		

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

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

6.22.1 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	タスク優先度

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)tskpri＜0 (2)tskpri＞CFG_MAXTSKPRI (3)非タスクコンテキストからの呼び出しで、tskpri=TPRI_SELF(0)を指定

■機能

tskpri で示された優先度のレディキューにつながれている先頭タスクをレディキューの最後尾につなぎかえ(レディキューを回転)、次につながれているタスクに実行を切り替えます。

tskpri=TPRI_SELF(0)を指定すると、自タスクのベース優先度のレディキューを回転します。ミューテックス機能を使用しない場合は、ベース優先度と現在優先度は同じですが、ミューテックスをロック中は一般には現在優先度とベース優先度は一致しないため、TPRI_SELF を指定しても現在の自タスクが属する優先度のレディキューを回転することはできません。

6.22.2 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	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	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 (1)p_tskid が 2 バイト境界でない
E_MACV	[m]	メモリアクセス違反

■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。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_tskidに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_tskid
- size=sizeof(ID)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.22.3 実行状態のタスクの所属ドメイン ID の参照(get_did, iget_did)

■C 言語 API

```
ER ercd = get_did(ID *p_domid);
ER ercd = iget_did(ID *p_domid);
```

■パラメータ

ID	*p_domid	ドメイン ID を返す領域へのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
ID	*p_domid	ドメイン ID へのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー (1)p_domid が 2 バイト境界でない
E_MACV	[m]	メモリアクセス違反

■機能

実行状態のタスクの所属ドメインの ID 番号を、p_domid の指す領域に返します。
具体的には、呼び出し元の処理単位に応じて、以下のような振る舞いとなります。

(1) タスクまたはタスク例外処理ルーチンから本サービスコールを発行した場合は、そのタスクが所属するドメインのID番号を返します。
(2) タスクまたはタスク例外処理ルーチンから呼び出された拡張サービスコールルーチンまたはトラップルーチンから本サービスコールを発行した場合は、それらを呼び出す前に実行していたタスクが所属するドメインのID番号が返ります。
(3) 非タスクコンテキストから発行した場合は、その時点で実行していたタスクが所属するドメインのID番号を返します。ただし、実行状態のタスクが存在しない場合はTDOM_NONE(-2)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。
(1) p_domidに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=p_domid
- size=sizeof(ID)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.22.4 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	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)タスクコンテキストから、chg_ims で割込みをマスクしている状態で呼び出した

■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) タスクのスケジューリングは行われません。
(2) タスク例外処理ルーチンは起動されません。
(3) コンフィギュレータで定義したカーネル割込みマスクレベル(CFG_KNLMSKLVL)以下の割込みが禁止されます。
(4) 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。ただし、待ち状態に遷移するサービスコールを発行した場合およびchg_imsは、E_CTXエラーが返ります。

- ext_tsk(自動的に CPU ロック状態が解除されます)
- exd_tsk(自動的に CPU ロック状態が解除されます)
- sns_tex
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

CPU ロック状態は、以下の操作で解除されます。

(a) unl_cpu, iunl_cpuサービスコールの呼び出し
(b) ext_tsk, exd_tskサービスコールの呼び出し

また、割込みハンドラ、タイムイベントハンドラ、CPU 例外ハンドラ、初期化ルーチンの終了時には、それぞれのハンドラ起動時の CPU ロック/ロック解除状態に戻ります。

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。

6.22.5 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	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)タスクコンテキストから、chg_ims で割込みをマスクしている状態で呼び出した

■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。

なお、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。unl_cpu, iunl_cpu サービスコールでは、dis_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。

6.22.6 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	正常終了(E_OK)

■エラーコード

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し

■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は行われません。
(2) タスクコンテキストから待ち状態になるサービスコールを呼び出すと、E_CTXエラーとなります。

ディスパッチ禁止状態の間に以下の操作を行うと、ディスパッチ許可状態に戻ります。

(1) ena_dspサービスコールの呼び出し
(2) ext_tsk, exd_tskサービスコールの呼び出し

ディスパッチ禁止状態とディスパッチ許可状態の間の遷移は、dis_dsp, ena_dsp, ext_tsk, exd_tsk サービスコールによってのみ発生します。

本サービスコールによってディスパッチ禁止状態の間は、ref_tsk サービスコールで自タスクの状態を参照しても実行状態とは見えない場合があるので、注意してください。

すでにディスパッチ禁止状態のときに再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

6.22.7 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	正常終了(E_OK)

■エラーコード

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し

■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。

6.22.8 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	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。

6.22.9 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	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。

6.22.10 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	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。

6.22.11 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	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全て満たされる場合に 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) タスクコンテキスト実行中である
(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

6.22.12 カーネルの起動(vsta_knl, ivsta_knl)

■C 言語 API

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

■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。また、呼び出し元には戻りません。

本サービスコールは、SR.MD=1 かつ SR.BL=1 の状態で呼び出す必要があります。また、本サービスコールを呼び出すアプリケーションは、カーネルライブラリとリンクする必要があります。

なお、メモリオブジェクト保護機能を使用する場合でも、カーネルは MMU を Enable にする初期化は行いません。MMU を Enable にするかどうかは、カーネル起動前にアプリケーションによる MMUCR.AT の初期化で決まります

また、MMU を Enable にする場合は、MMU を Enable にしてから vsta_knl を呼び出すまでの間に MMU 対象領域をアクセスしてはなりません。これは、MMU 対象領域をアクセスすると MMU 関連の例外が発生しますが、まだカーネルが起動されていないために、カーネルがこの例外をハンドリングできないためです。

以下に、本サービスコールでの初期化処理の詳細を示します。

(1) SR.BL=1に設定(全割込みを禁止)

(2) スタックポインタ(R15)を非タスクコンテキストスタック(BSCP_hintskstkセクション)に設定

(3) VBRレジスタの初期化
割込みハンドラ起動時のSR.IMASKビットの扱いは、カーネル起動時点のCPUOPMレジスタのINTMUビットで決まります。即ち、カーネル起動前にこのビットを必要に応じて初期化してください。

(4) RAMCRレジスタの初期化
メモリオブジェクト保護機能を使用(CFG_PROTMEMを選択)する場合で、かつCFG_IRAMを選択していた場合のみ、CFG_IRAMUSAGEの設定に従ってRAMCRレジスタのRP,RMDビットを以下のように初期化します。その他のビットは初期化しません。

CFG_IRAMUSAGE の設定	RAMCR.RP 初期設定値	RAMCR.RMD 初期設定値
MMU 非対象領域, 全モードからアクセス可能	0	1
MMU 非対象領域, ユーザ非 DSP モードのみアクセス不可	0	0
MMU 対象領域	1	1

(5) MMUCRレジスタの初期化
メモリオブジェクト保護機能を使用(CFG_PROTMEMを選択)する場合のみ、MMUCRレジスタのLRUI, URB, URCビットを0に初期化します。その他のビットは初期化しません。
MMUをEnableにするには、カーネル起動前にMMUCR.AT=1に初期化してください。逆に、MMUCR.AT=0の状態でカーネルを起動すると、MMUがDisableのままとなるため、不正メモリアクセスが検出されない変わりに、TLBミスが発生しなくなります。

(6) 各種カーネル管理情報の初期化

(7) コンフィギュレータで登録された静的メモリオブジェクトの初期化

(8) CFG_OPTTMRが選択されていた場合は、タイマの初期化

(9) SR.BL=0, IMASK=15に設定

(10) 各種オブジェクトの生成・定義処理
コンフィギュレータで登録された各種オブジェクトを生成・定義します。
これらのオブジェクトの生成・定義は、コンフィギュレータが出力する初期登録ルーチンから対応するサービスコールを発行することで実現されています。
コンフィギュレータでの設定内容に誤りがある場合は、このサービスコールがエラーとなる場合があります。この場合、初期登録ルーチンではエラーが発生したサービスコールを呼び出した場所で無限ループするようになっています。
初期登録ルーチンは、kernel_def_inireg.hとkernel_cfg_inireg.hの２つがあります。前者は、コンフィギュレータで[カーネル側]をチェックしたオブジェクト、後者は[カーネル側]をチェックしていないオブジェクトの生成・定義を行うルーチンです。
初期登録ルーチンは、まずkernel_def_inireg.hが呼び出され、その後kernel_cfg_inireg.hが呼び出されます。両者の中での各オブジェクトの生成・定義順序は以下のように決まっています。また、各オブジェクトの種類の中での生成・定義順序は、タスクと周期ハンドラはコンフィギュレータの画面の上から順に生成され、その他のオブジェクトは順不同です。

順序	オブジェクト
1	**割込み・CPU 例外ハンドラ**
2	**オーバーランハンドラ**
3	**周期ハンドラ**
4	**アラームハンドラ**
5	**拡張サービスコールルーチン**
6	**トラップルーチン**
7	**セマフォ**
8	**イベントフラグ**
9	**データキュー**
10	**メールボックス**
11	**ミューテックス**
12	**メッセージバッファ**
13	**固定長メモリプール**
14	**可変長メモリプール**
15	**保護メモリプール**
16	**保護メールボックス**
17	**タスク、タスク例外処理ルーチン**

(11) タイマ初期化ルーチン(_kernekl_tmrini())の呼び出し(CFG_OPTTMR選択無しの場合のみ)

(12) 初期化ルーチンの呼び出し
コンフィギュレータで登録された初期登録ルーチンを呼び出します。
呼び出し順序は、まず[カーネル側]がチェックされたルーチンをコンフィギュレータの画面の順に呼び出し、次に[カーネル側]がチェックされていないルーチンをコンフィギュレータの画面の順に呼び出します。

(13) プログラムパフォーマンスカウンタの初期化(CFG_PERFORM選択時のみ)

上記の初期化処理を実行後、マルチタスク環境へ移行します。

ivsta_knlは、μITRON4.0仕様の命名規約に沿うために用意しているAPIですが、その実体はvsta_knlと同一です。ヘッダファイルでは、ivsta_knl()を vsta_knl()に定義しています。

6.22.13 システムダウン(vsys_dwn, ivsys_dwn)

■C 言語 API

```
void vsys_dwn(ER type, VW inf1, VW inf2, VW inf3);
void ivsys_dwn(ER type, VW inf1, VW inf2, VW inf3);
```

■パラメータ

ER	type	エラー種別
VW	inf1	システム異常情報 1
VW	inf2	システム異常情報 2
VW	inf3	システム異常情報 3

■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。システムダウンルーチンはカーネルドメインに所属し、特権モードで実行されます。

type には、エラー種別として発生したエラーに対応した値(1～H'7fffffff)を設定してください。0 以下の値はシステム用に予約されています。

カーネル内で異常を検出した場合にも、システムダウンルーチンが呼び出されます。

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

ivsys_dwn は、μITRON4.0 仕様の命名規約に沿うために用意している API ですが、その実体は vsys_dwn と同一です。ヘッダファイルでは、ivsys_dwn を vsys_dwn に定義しています。

なお、本サービスコールは、他のサービスコールと同様に TRAPA 命令を使用するため、SR.BL=1 の状態で発行してはなりません。SR.BL=1 の時にシステムダウンさせたい場合は、システムダウンルーチンを直接呼び出してください。

■関連ページ　「8.10 システムダウンルーチン」

6.22.14 トレースの取得(vget_trc, ivget_trc)

■C 言語 API

```
ER ercd = vget_trc(VW para1, VW para2, VW para3, VW para4);
ER ercd = ivget_trc(VW para1, VW para2, VW para3, VW para4);
```

■パラメータ

VW	para1	パラメータ 1
VW	para2	パラメータ 2
VW	para3	パラメータ 3
VW	para4	パラメータ 4

■リターンパラメータ

ER	ercd	正常終了(E_OK)

■エラーコード

なし

■機能

ユーザ任意の情報をトレース取得します。
para1～para4 は、取得される情報を識別するために、ユーザが自由に使用できます。
取得したトレース情報は、デバッギングエクステンションで表示されます。
コンフィギュレータで CFG_TRACE をチェックしていない場合は、本サービスコールは何も処理せずに、常に正常終了します。

ivget_trc は、μITRON4.0 仕様の命名規約に沿うために用意している API ですが、その実体は vget_trc と同一です。ヘッダファイルでは、ivget_trc を vget_trc に定義しています。

6.22.15 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	割込みハンドラ番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)

■エラーコード

なし

■機能

本 API は、HI7000/4 シリーズとの互換性のためにだけ、C 言語インタフェースのみを残してあります。C 言語インタフェースでは、以下のように定義しています。

```
#define ivbgn_int(dintno) E_OK  /* 常に E_OK を返す */
```

本カーネルでは、CFG_TRACE を選択した場合は常に割込みハンドラの開始と終了をトレース取得します。

6.22.16 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	割込みハンドラ番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)

■エラーコード

なし

■機能

本 API は、HI7000/4 シリーズとの互換性のためにだけ、C 言語インタフェースのみを残してあります。C 言語インタフェースでは、以下のように定義しています。

```
#define ivend_int(dintno) E_OK  /* 常に E_OK を返す */
```

本カーネルでは、CFG_TRACE を選択した場合は常に割込みハンドラの開始と終了をトレース取得します。

6.22.17 DSP(TA_COP0)属性の変更(vchg_cop)

■C 言語 API

```
ER_UINT oldatr = vchg_cop(ATR newatr);
```

■パラメータ

ATR	newatr	変更後の属性

■リターンパラメータ

ER_UINT	oldatr	変更前の属性(正の値または 0)またはエラーコード

■エラーコード

E_RSATR	[p]	予約属性 (1)newatr の TA_COP0 以外のビットが 0 でない
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)TA_COP1 または TA_COP2 属性を持つプログラムからの呼び出し

■機能

発行元の現在の TA_COP0 属性を、newatr に変更します。

本サービスコールは、DSP スタンバイ機能の制御粒度を細かくするために用意されているサービスコールです。

本サービスコールは、現時点の呼び出し元の属性を変更します。すなわち、タスクから呼び出した場合はタスクの、タスク例外処理ルーチンから発行した場合はタスク例外処理ルーチンの、タスクコンテキストから呼び出した拡張サービスコールルーチンまたはトラップルーチンから呼び出した場合は拡張サービスコールルーチンまたはトラップルーチンの属性を、それぞれ変更します。

この変更は、今回限りの一時的なものです。

タスクの場合、タスクが終了して次回起動されたときには、タスク生成時に指定された属性となります。

タスク例外処理ルーチンの場合、タスク例外処理ルーチンが終了して次回起動されたときには、タスク例外処理ルーチン定義時に指定された属性となります。

タスクコンテキストから呼び出した拡張サービスコールルーチンおよびトラップルーチンの場合は、現在実行中のタスクのコンテキストでのみ変更が有効で、その後他のタスクから呼び出した場合は、拡張サービスコールルーチンまたはトラップルーチンの定義時に指定した属性となります。

newatr には、以下を指定できます。FPU に関する属性(TA_COP1, TA_COP2)を指定することはできません。

- TA_COP0(H'00000100) : DSP を使用
- TA_NULL(H'00000000) : DSP を使用しない

oldatr には、変更前の属性に応じて、TA_COP0 または TA_NULL が返ります。

6.23 割込み管理機能

表6.44 割込み管理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_inh	割込みハンドラの定義	○		○	○	○		
	idef_inh			○	○	○	○		
2	chg_ims	割込みマスクの変更	○		○	○	○		
	ichg_ims			○	○	○	○		
3	get_ims	割込みマスクの参照	○		○	○	○		
	iget_ims			○	○	○	○		

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

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

表6.45 割込み管理の仕様

項番	項目	内容
1	割込みハンドラ番号	0～CFG_MAXINTNO (最大 H'3fe0)の H'20 の倍数
2	割込みハンドラ属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述

6.23.1 割込みハンドラの定義(def_inh, idef_inh)

■C 言語 API

```
ER ercd = def_inh(INHNO inhno, T_DINH *pk_dinh);
ER ercd = idef_inh(INHNO inhno, T_DINH *pk_dinh);
```

■パラメータ

INHNO	inhno	割込みハンドラ番号
T_DINH	*pk_dinh	割込みハンドラ定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct{
           ATR     inhatr;      0      4    ハンドラ属性
           FP      inthdr;      +4     4    ハンドラアドレス
           UW      inhsr;       +8     4    起動時の SR
} T_DINH;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)inhatr の TA_ASM 以外のビットが 0 でない
E_PAR	[p]	パラメータエラー (1)0x20 の倍数に切り捨て後の inhno が、0, H'20,H'140,H'160, または CFG_MAXINTNO を超える (2)pk_dinh が 4 バイト境界でない (3)inthdr が奇数
E_MACV	[m]	メモリアクセス違反

■機能

割込みハンドラを定義します。割込みハンドラはカーネルドメインに所属し、特権モードで実行されます。

なお、本サービスコールと def_exc の実体は同じものです。即ち、本サービスコールで CPU 例外ハンドラを定義することもできます。逆に、def_exc で割込みハンドラを定義することもできます。

以下に各パラメータの意味を示します。

(1) inhno

割込みハンドラ番号です。割込みハンドラ番号には、CPU の INTEVT コードを指定してください。inhno は H'20 の倍数に切り捨てて扱います。

なお、SH マイコンでは、割込みコード(INTEVT コード)と例外コード(EXPEVT コード)は、共通のコード体系になっています。本サービスコールは、指定された inhno が割込み用のコードか例外用のコードかはチェックしません。例外用のコードを指定した場合は、def_exc として動作します。

なお、INTEVT, EXPEV コードには、ハンドラを定義のできないコードおよび定義しても無意味なコードがあります。表 6.46に、その詳細を示します。

表6.46　特殊な INTEVT, EXPEVT コード

INTEVT, EXPEVT コード	要因	ハンドラ定義	発生時の CPU の動作	
			CFG_PROTMEM チェックなし	CFG_PROTMEM チェックあり
0	パワーオンリセット、H-UDI リセット	不可	リセットベクタ(H'A0000000)に分岐	
H'20	マニュアルリセット	不可		
H'40	TLB ミス例外(リード)	可能	定義されたハンドラを起動 *	カーネルが TLB を更新します。ただし、更新できない場合はメモリアクセス違反ハンドラを起動します。 定義されたハンドラが実行することはないため、定義しても無意味です。
H'60	TLB ミス例外(ライト)			
H'A0	TLB 保護違反例外(リード)			
H'C0	TLB 保護違反例外(ライト)			
H'140	命令 TLB 多重ヒット例外、データ TLB 多重ヒット例外	不可	リセットベクタ(H'A0000000)に分岐 *	リセットベクタ(H'A0000000)に分岐
H'160	TRAPA 命令	不可	トラップ番号に応じて、カーネルのサービスコールまたは、ユーザのトラップルーチンが実行されます。	

【注】 アプリケーション側で MMU を Enable にしない限り、通常はこれらの例外が発生することはありません。

また、以下のコードは指定しないでください。

(1) CFG_TIMINTNO
CFG_OPTTMRをチェックしない場合は、CFG_TIMINTNOが標準タイマドライバで使用する割込み番号になります。この番号に対するハンドラを定義してしまうと、標準タイマドライバが正しく動作できなくなります。

(2) H'400, H'420, H'440
CFG_OPTTMRをチェックした場合は、これらの番号が最適化タイマドライバで使用する割込み番号になります。この番号に対するハンドラを定義してしまうと、最適化タイマドライバが正しく動作できなくなります。
ただし、def_ovrを組み込まない場合は、H'440は最適化タイマドライバでは使用しないので、割込みハンドラを定義してもかまいません。

(2) inhatr

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(3) inhsr

inhsr は、μITRON 仕様外の項目です。

inhsr には割込みハンドラ起動時のステータスレジスタ(SR)の値を指定します。inhsr は、SR の構成と同じビット位置で指定します。ただし、実際の割込みハンドラ起動時の SR は、以下のようになります。つまり、inhsr で意味があるのはブロック(BL)ビットと割込みマスクレベル(I)ビットのみです。その他のビットは無視されます。なお、割込みマスクレベル(I)ビットには、必ず当該割込みレベル以上の値を指定してください。当該割込みレベルより低い値を指定すると、システムの動作は保証されません。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：inhsr の指定に従います
- DSP ビット(SH4AL-DSP)：0
- FPU ディスエーブル(FD)ビット(SH-4A)：1
- 割込みマスクレベル(IMASK)ビット：カーネル起動時の CPUOPM レジスタの INTMU ビットが 0 の場合は inhsr の指定に従います。INTMU ビットが 1 の場合は、発生した割込みレベル値になります。
- その他のビット：不定

pk_dinh=NULL(0)が指定された場合は、割込みハンドラの定義を解除します。

また、すでに割込みハンドラが定義されている状態で、本サービスコールが呼び出された場合は、以前の定義を解除し、新しい定義に置き換えます。

なお、割込みハンドラはコンフィギュレータで静的に定義することもできます。

定義していない番号の割込みが発生すると、システムダウンとなります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dinhに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dinh
- size=sizeof(T_DINH)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dinh->inthdrに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base= pk_dinh->inthdr
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.23.2 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	割込みマスク値

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[p]	パラメータエラー (1)imask に SR_IMS00～SR_IMS15 以外の値を指定
E_CTX	[k]	コンテキストエラー (1)タスクコンテキストで、CPU ロック状態からの呼び出し

■機能

現在の割込みマスクを imask で指定した値に変更します。
imask には、以下の指定ができます。

- SR_IMSnn(H'0000000m)割込みマスクレベルを nn に変更
  - nn : 0～15 を 10 進数 2 桁で表現した文字列("00", "01", "02", ... , "15")
  - m : nn を 16 進数に変換した文字

タスクコンテキストから本サービスコールで割込みマスクを 0 以外に変更している期間の特長を以下に示します。

(1) タスクのスケジューリングは行われません。即ち、ディスパッチ保留状態になります。このため、待ち状態に遷移するサービスコールは発行できません。発行した場合、E_CTX エラーが返ります。
(2) タスク例外処理ルーチンは起動されません。
(3) imaskで指定したレベル以下の割込みが禁止されます。
(4) loc_cpu, unl_cpuは発行できません。発行した場合、E_CTXエラーが返ります。

以下のケースで割込みマスクレベルを変更する場合は、必ず本サービスコールを使用してください。それ以外のケース以外で割込みマスクレベルを変更する場合は，SR レジスタを直接変更しても構いません。

(1) タスクコンテキストで割込みマスクレベルを0から0以外に変更する場合
(2) (1)の後、割込みマスクレベルを0に戻す場合

これに違反した場合の動作は保証されません。

割込みマスクレベルをカーネル割込みマスクレベル(CFG_KNLMSKLVL)より高いレベルに変更している間は、本サービスコールでカーネル割込みマスクレベル以下に割込みマスクレベルを戻す場合を除き、サービスコールを発行してはなりません。これに違反した場合の動作は保証されません。

6.23.3 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	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	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 (1)p_imask が 4 バイト境界でない
E_MACV	[m]	メモリアクセス違反

■機能

現在の CPU ステータスレジスタ(SR)の割込みマスクビット(IMASK ビット)を参照し、割込みマスクレベルを p_imask の指す領域に返します。

p_imask の指す領域に返る値は、chg_ims サービスコールで用いるパラメータ imask と同じフォーマットです。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_imaskに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_imask
- size=sizeof(IMASK)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.24 拡張サービスコール・トラップ管理機能

表6.47 サービスコール管理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_svc	拡張サービスコールの定義	○		○	○	○		
	idef_svc			○	○	○	○		
2	cal_svc	拡張サービスコールの呼び出し	○		○	○	○		
	ical_svc			○	○	○	○		
3	vdef_trp	トラップの定義	○		○	○	○		
	ivdef_trp			○	○	○	○		

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

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

表6.48 サービスコール管理の仕様

項番	項目	内容
1	拡張サービスコールの機能コード	1～CFG_MAXSVCCD (最大 32767)
2	拡張サービスコールルーチン属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述 ・TA_COP0：DSP を使用 ・TA_COP1：FPU のレジスタバンク 0 を使用 ・TA_COP2：FPU のレジスタバンク 1 を使用
3	トラップ番号	16～CFG_MAXTRPNO (最大 255) なお、0～15 はカーネル予約なので使用できません。
4	トラップルーチン属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述 ・TA_COP0：DSP を使用 ・TA_COP1：FPU のレジスタバンク 0 を使用 ・TA_COP2：FPU のレジスタバンク 1 を使用

拡張サービスコールとトラップは、いずれも以下の特長を持っています。

- 拡張サービスコールルーチンとトラップルーチンは特権モードで実行するので、ユーザドメインでは制限されているアクセスが可能。
- プログラムリンクせずに呼び出し可能。
- タスクコンテキストから呼び出された場合、拡張サービスコールルーチンとトラップルーチンの実行中はタスク例外処理ルーチンは起動されない。
- タスクコンテキストから呼び出された場合、拡張サービスコールルーチンとトラップルーチン実行中に rel_wai サービスコールが発行されると、待ち禁止状態に遷移する。

6.24.1 拡張サービスコールの定義(def_svc, idef_svc)

■C 言語 API

```
ER ercd = def_svc(FN fncd, T_DSVC *pk_dsvc);
ER ercd = idef_svc(FN fncd, T_DSVC *pk_dsvc);
```

■パラメータ

FN	fncd	拡張サービスコールの機能コード
T_DSVC	*pk_dsvc	拡張サービスコールルーチン定義情報の先頭アドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     svcatr;     0      4    拡張サービスコールルーチン属性
           FP      svcrtn;     +4     4    拡張サービスコールルーチンアドレス
           UW      inifpscr;   +8     4    初期 FPSCR 値
} T_DSVC;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)svcatr の TA_COP0,TA_COP1,TA_COP2,TA_ASM 以外のビットが0 でない (2)CFG_DSP 未チェックで、svcatr に TA_COP0 を指定 (3)CFG_FPU 未チェックで、svcatr に TA_COP1 を指定 (4)svcatr に、TA_COP1 を指定せずに TA_COP2 を指定 (5)svcatr に、TA_COP0 と TA_COP1 を同時に指定
E_PAR	[p]	パラメータエラー (1)fncd≦0 (2)fncd>CFG_MAXSVCCD (3)pk_dsvc が 4 バイト境界でない (4)svcrtn が奇数
E_MACV	[m]	メモリアクセス違反

■機能

拡張サービスコールルーチンを定義します。拡張サービスコールルーチンは、cal_svc によって呼び出されます。
拡張サービスコールルーチンは、カーネルドメインに所属し、特権モードで実行されます。

拡張サービスコールルーチンの起動/終了の前後では、以下の状態は変化しません。

- タスクコンテキスト/非タスクコンテキスト
- ディスパッチ禁止/許可状態
- CPU ロック/ロック解除状態

タスクが拡張サービスコールーチンを呼び出して実行している間は、そのタスクのタスク例外処理ルーチンは起動されません。また、そのタスクに対して rel_wai が発行されると、そのタスクは待ち禁止状態になります。

以下に各パラメータの意味を示します。

(1) fncd

定義対象の機能コードを指定します。fncd には、1～CFG_MAXSVCCD の値を指定します。

(2) svcatr

svcatr には、以下の(a)～(c)の論理和を指定してください。

(a) 記述言語

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(b) DSP を搭載したマイコンを使用する場合(CFG_DSP をチェックした場合)

DSP を使用する場合は、TA_COP0 を指定してください。

- TA_COP0(H'00000100)：DSP を使用

(c) FPU を搭載したマイコンを使用する場合(CFG_FPU をチェックした場合)

浮動小数点演算など、FPU を使用する場合は、TA_COP1 を指定してください。また、マトリックス演算など FPU の両バンクを使用する場合には、TA_COP1 に加えて TA_COP2 を指定してください。

- TA_COP1(H'00000200)：FPU のレジスタバンク 0(FPR0_BANK0～FPR15_BANK0)と FPUL を使用
- TA_COP2(H'00000400)：FPU のレジスタバンク 1(FPR0_BANK1～FPR15_BANK1)を使用
  TA_COP2を指定する場合は、必ずTA_COP1も指定しなければなりません。そうでない場合、E_RSATRエラーを返します。

また、「(4) inifpscr」も参照してください。

(3) svcrtn

svcrtn には、拡張サービスコールルーチンの起動アドレスを指定します。

(4) inifpscr

inifpscr は、μITRON 仕様外の項目です。

CFG_FPU を選択していた場合で、かつ TA_COP1 属性を指定した場合のみ有効です。その他の場合は単に無視されます。

inifpscr は、起動時の FPSCR 値です。カーネルは、inifpscr で指定された値をそのまま FPSCR に設定します。inifpscr に指定された値に関するエラー検出は行いません。

下記も参照してください。

■関連ページ　「15. FPU に関する注意事項」

なお、拡張サービスコールルーチンはコンフィギュレータで静的に定義することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dsvcに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dsvc
- size=sizeof(T_DSVC)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dsvc->svcrtnに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base= pk_dsvc->svcrtn
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.24.2 拡張サービスコールの呼び出し(cal_svc, ical_svc)

■C 言語 API

```
ER_UINT ercd = cal_svc(FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4);
ER_UINT ercd = ical_svc(FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4);
```

■パラメータ

FN	fncd	拡張サービスコールの機能コード
VP_INT	par1	パラメータ 1
VP_INT	par2	パラメータ 2
VP_INT	par3	パラメータ 3
VP_INT	par4	パラメータ 4

■リターンパラメータ

ER_UINT	ercd	サービスコールからのリターン値

■エラーコード

E_RSFN	[k]	予約機能コード(fncd が不正あるいは使用できない)

■機能

fncd で指定された機能コードに対応する拡張サービスコールルーチンを実行します。
パラメータ par1～par4 は、拡張サービスコールルーチンにパラメータとして渡されます。

6.24.3 トラップルーチン定義(vdef_trp, ivdef_trp)

■C 言語 API

```
ER ercd = vdef_trp(UINT dtrpno, VT_DTRP *pk_dtrp);
ER ercd = ivdef_trp(UINT dtrpno, VT_DTRP *pk_dtrp);
```

■パラメータ

UINT	dtrpno	トラップ番号
VT_DTRP	*pk_dtrp	トラップルーチン定義情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     trpatr;     0      4    トラップルーチン属性
           FP      trprtn;     +4     4    トラップルーチンアドレス
           UW      inifpscr;   +8     4    初期 FPSCR 値
} VT_DTRP;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)trpatr の TA_COP0,TA_COP1,TA_COP2,TA_ASM 以外のビットが 0 でない (2)CFG_DSP 未チェックで、trpatr に TA_COP0 を指定 (3)CFG_FPU 未チェックで、trpatr に TA_COP1 を指定 (4)trpatr に、TA_COP1 を指定せずに TA_COP2 を指定 (5)trpatr に、TA_COP0 と TA_COP1 を同時に指定
E_PAR	[p]	パラメータエラー (1)0≦trpno≦15 (2)trpno>CFG_NAXTRPNO (3)pk_dtrp が 4 バイト境界でない (4)trprtn が奇数
E_MACV	[m]	メモリアクセス違反

■機能

トラップルーチンを pk_dtrp で示された内容で定義します。トラップルーチンは、dtrpno の TRAPA 命令を実行すると呼び出されます。
トラップルーチンはカーネルドメインに所属し、特権モードで実行されます。

トラップルーチンの起動/終了の前後では、以下の状態は変化しません。

- タスクコンテキスト/非タスクコンテキスト
- ディスパッチ禁止/許可状態
- CPU ロック/ロック解除状態

タスクがトラップルーチンを呼び出して実行している間は、そのタスクのタスク例外処理ルーチンは起動されません。また、そのタスクに対して rel_wai が発行されると、そのタスクは待ち禁止状態になります。

以下に、各パラメータの意味を示します。

(1) dtrpno

対象とする TRAPA 番号を指定します。dtrpno には、16～CFG_MAXTRPNO を指定します。0～15 はシステム用に予約されているので指定できません。

(2) trpatr

trpatr には、以下の論理和を指定してください。

(a) 記述言語

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(b) DSP を搭載したマイコンを使用する場合(CFG_DSP をチェックした場合)

DSP を使用する場合は、TA_COP0 を指定してください。

- TA_COP0(H'00000100)：DSP を使用

(c) FPU を搭載したマイコンを使用する場合(CFG_FPU をチェックした場合)

浮動小数点演算など、FPU を使用する場合は、TA_COP1 を指定してください。また、マトリックス演算など FPU の両バンクを使用する場合には、TA_COP1 に加えて TA_COP2 を指定してください。

- TA_COP1(H'00000200)：FPU のレジスタバンク 0(FPR0_BANK0～FPR15_BANK0)と FPUL を使用
- TA_COP2(H'00000400)：FPU のレジスタバンク 1(FPR0_BANK1～FPR15_BANK1)を使用
  TA_COP2を指定する場合は、必ずTA_COP1も指定しなければなりません。そうでない場合、E_RSATRエラーを返します。

また、「(4) inifpscr」も参照してください。

(3) trprtn

trprtn には、トラップルーチンの起動アドレスを指定します。

(4) inifpscr

inifpscr は、μITRON 仕様外の項目です。

CFG_FPU を選択していた場合で、かつ TA_COP1 属性を指定した場合のみ有効です。その他の場合は単に無視されます。

inifpscr は、起動時の FPSCR 値です。カーネルは、inifpscr で指定された値をそのまま FPSCR に設定します。inifpscr に指定された値に関するエラー検出は行いません。

下記も参照してください。

■関連ページ　「15. FPU に関する注意事項」

なお、トラップルーチンはコンフィギュレータで静的に定義することもできます。

定義していない番号の TRAPA 命令を実行すると、R0 レジスタに E_RSFN が返ります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dtrpに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dtrp
- size=sizeof(VT_DTRP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dtrp->trprtnに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base= pk_dtrp->trprtn
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.25 システム構成管理機能

表6.49 システム構成管理サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	def_exc	CPU 例外ハンドラの定義	○		○	○	○		
	idef_exc			○	○	○	○		
2	ref_cfg	コンフィギュレーション情報の参照	○		○	○	○		
	iref_cfg			○	○	○	○		
3	ref_ver	バージョン情報の参照	○		○	○	○		
	iref_ver			○	○	○	○		

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

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

表6.50 システム構成管理の制限値

項番	項目	内容
1	CPU 例外ハンドラ番号	0～CFG_MAXINTNO (最大 H'3fe0)の H'20 の倍数
2	CPU 例外ハンドラ属性	・TA_HLNG：高級言語記述 ・TA_ASM：アセンブリ言語記述

6.25.1 CPU 例外ハンドラの定義(def_exc, idef_exc)

■C 言語 API

```
ER ercd = def_exc(EXCNO excno, T_DEXC *pk_dexc);
ER ercd = idef_exc(EXCNO excno, T_DEXC *pk_dexc);
```

■パラメータ

EXCNO	excno	CPU 例外ハンドラ番号
T_DEXC	*pk_dexc	CPU 例外ハンドラ定義情報の先頭アドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     excatr;      0      4    ハンドラ属性
           FP      exchdr;      +4     4    ハンドラアドレス
           UW      excsr;       +8     4    起動時の SR
} T_DEXC;
```

■エラーコード

E_RSATR	[p]	予約属性 (1)excatr の TA_ASM 以外のビットが 0 でない
E_PAR	[p]	パラメータエラー (1)0x20 の倍数に切り捨て後の excno が、0, H'20,H'140,H'160, または CFG_MAXINTNO を超える (2)pk_dexc が 4 バイト境界でない (3)exchdr が奇数
E_MACV	[m]	メモリアクセス違反

■機能

CPU 例外ハンドラを定義します。CPU 例外ハンドラはカーネルドメインに所属し、特権モードで実行されます。

なお、本サービスコールと def_inh の実体は同じものです。即ち、本サービスコールで割込みハンドラを定義することもできます。逆に、def_inh で CPU 例外みハンドラを定義することもできます。

以下に各パラメータの意味を示します。

(1) excno

CPU 例外ハンドラ番号です。CPU 例外ハンドラ番号には、CPU の EXPEVT コードを指定してください。excno は H'20 の倍数に切り捨てて扱います。

なお、SH マイコンでは、割込みコード(INTEVT コード)と例外コード(EXPEVT コード)は、共通のコード体系になっています。本サービスコールは、指定された excno が例外用のコードか割込み用のコードかはチェックしません。割込み用のコードを指定した場合は、def_inh として動作します。

なお、INTEVT, EXPEV コードには、ハンドラを定義のできないコードおよび定義しても無意味なコードがあります。表 6.51に、その詳細を示します。

表6.51　特殊な INTEVT, EXPEVT コード

INTEVT, EXPEVT コード	要因	ハンドラ定義	発生時の CPU の動作	
			CFG_PROTMEM チェックなし	CFG_PROTMEM チェックあり
0	パワーオンリセット、H-UDI リセット	不可	リセットベクタ(H'A0000000)に分岐	
H'20	マニュアルリセット	不可		
H'40	TLB ミス例外(リード)	可能	定義されたハンドラを起動 *	カーネルが TLB を更新します。ただし、更新できない場合はメモリアクセス違反ハンドラを起動します。
H'60	TLB ミス例外(ライト)			
H'A0	TLB 保護違反例外(リード)			定義されたハンドラが実行することはないため、定義しても無意味です。
H'C0	TLB 保護違反例外(ライト)			
H'140	命令 TLB 多重ヒット例外、データ TLB 多重ヒット例外	不可	リセットベクタ(H'A0000000)に分岐 *	リセットベクタ(H'A0000000)に分岐
H'160	TRAPA 命令	不可	トラップ番号に応じて、カーネルのサービスコールまたは、ユーザのトラップルーチンが実行されます。	

【注】　アプリケーション側で MMU を Enable にしない限り、通常はこれらの例外が発生することはありません。

また、以下のコードは指定しないでください。

(1)　CFG_TIMINTNO
CFG_OPTTMRをチェックしない場合は、CFG_TIMINTNOが標準タイマドライバで使用する割込み番号になります。この番号に対するハンドラを定義してしまうと、標準タイマドライバが正しく動作できなくなります。

(2)　H'400, H'420, H'440
CFG_OPTTMRをチェックした場合は、これらの番号が最適化タイマドライバで使用する割込み番号になります。この番号に対するハンドラを定義してしまうと、最適化タイマドライバが正しく動作できなくなります。
ただし、def_ovrを組み込まない場合は、H'440は最適化タイマドライバでは使用しないので、割込みハンドラを定義してもかまいません。

(2) excatr

以下のいずれかを指定してください。

- TA_HLNG(H'00000000)：高級言語記述
- TA_ASM(H'00000001)：アセンブリ言語記述

(3) excsr

excsr は、μITRON 仕様外の項目です。

excsr には割込みハンドラ起動時のステータスレジスタ(SR)の値を指定します。excsr は、SR の構成と同じビット位置で指定します。ただし、実際の CPU 例外ハンドラ起動時の SR は、以下のようになります。つまり、excsr で意味があるのはブロック(BL)ビットのみです。その他のビットは無視されます。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：excsr に従う
- その他のビット：CPU 例外発生前と同じ

pk_dexc=NULL(0)が指定された場合は、CPU 例外ハンドラの定義を解除します。

また、すでに CPU 例外ハンドラが定義されている状態で、本サービスコールが呼び出された場合は、以前の定義を解除し、新しい定義に置き換えます。

なお、CPU 例外ハンドラはコンフィギュレータで静的に定義することもできます。

定義していない番号の CPU 例外が発生すると、システムダウンとなります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_dexcに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_dexc
- size=sizeof(T_DEXC)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

(2) pk_dexc->exchdrに対するカーネルドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base= pk_dexc->exchdr
- size=1
- domid=カーネルドメイン
- pmmode=TPM_READ

6.25.2 コンフィグレーション情報の参照(ref_cfg, iref_cfg)

■C 言語 API

```
ER ercd = ref_cfg(T_RCFG *pk_rcfg);
ER ercd = iref_cfg(T_RCFG *pk_rcfg);
```

■パラメータ

T_RCFG	*pk_rcfg	コンフィグレーション情報を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RCFG	*pk_rcfg	コンフィグレーション情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct {
           ID      maxtskid;    0      2    最大タスク ID
           ID      rsv;         +2     2    (予約)
           ID      maxsemid;    +4     2    最大セマフォ ID
           ID      maxflgid;    +6     2    最大イベントフラグ ID
           ID      maxdtqid;    +8     2    最大データキューID
           ID      maxmbxid;    +10    2    最大メールボックス ID
           ID      maxmtxid;    +12    2    最大ミューテックス ID
           ID      maxmbfid;    +14    2    最大メッセージバッファ ID
           ID      maxmplid;    +16    2    最大可変長メモリプール ID
           ID      maxmpfid;    +18    2    最大固定長メモリプール ID
           ID      maxcycid;    +20    2    最大周期ハンドラ ID
           ID      maxalmid;    +22    2    最大アラームハンドラ ID
           FN      maxs_fncd;   +24    4    最大拡張サービスコール機能コード
           ID      maxdomid;    +28    2    最大ドメイン ID
           ID      maxmppid;    +30    2    最大保護メモリプール ID
           ID      maxmbpid;    +32    2    最大保護メールボックス ID
} T_RCFG;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rcfg が 4 バイト境界でない
E_MACV	[m]	メモリアクセス違反

■機能

pk_rcfg で示された領域に、システムコンフィグレーション情報を返します。

pk_rcfg の指すパケットには、次の情報を返します。括弧内は、対応するコンフィギュレータでの設定項目です。

- maxtskid： 最大タスク ID(CFG_MAXTSKID)
- maxsemid：最大セマフォ ID(CFG_MAXSEMID)
- maxflgid：最大イベントフラグ ID(CFG_MAXFLGID)
- maxdtqid：最大データキューID(CFG_MAXDTQID)
- maxmbxid：最大メールボックス ID(CFG_MAXMBXID)
- maxmtxid：最大ミューテックス ID(CFG_MAXMTXID)
- maxmbfid：最大メッセージバッファ ID(CFG_MAXMBFID)
- maxmplid：最大可変長メモリプール ID(CFG_MAXMPLID)
- maxmpfid：最大固定長メモリプール ID(CFG_MAXMPFID)
- maxcycid：最大周期ハンドラ ID
  CFG_ACTIONをチェックしてた場合はCFG_MAXCYCID+1、そうでない場合は
  CFG_MAXCYCIDが返ります。なお前者の場合、IDがCFG_MAXCYCID+1の周期ハンドラは、
  デバッギングエクステンション用の周期ハンドラを意味します。
- maxalmid：最大アラームハンドラ ID(CFG_MAXALMID)
- maxs_fncd：最大拡張サービスコール機能コード(CFG_MAXSVCCD)
- maxdomid：最大ドメイン ID(常に 31 が返ります)
- maxmppid：最大保護メモリプール ID(CFG_MAXMPPID)
  メモリオブジェクト保護機能を組み込まない場合は0が返ります。
- maxmbpid：最大保護メールボックス ID(CFG_MAXMBPID)
  メモリオブジェクト保護機能を組み込まない場合は0が返ります。

なお、T_RCFG 構造体のメンバはすべて、μITRON 仕様の範囲外です。なお、μITRON 仕様では、T_RCFG 構造体の内容は規定されていません。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rcfg対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_rcfg
- size=sizeof(T_RCFG)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.25.3 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	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	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 {
           UH      maker;      0      2    メーカー
           UH      prid;       +2     2    形式番号
           UH      spver;      +4     2    仕様書バージョン
           UH      prver;      +6     2    製品バージョン
           UH      prno[4];    +8     8    製品管理情報
           UH      pxver;      +16    2    μITRON4.0 仕様保護機能拡張のバージョン番号
} T_RVER;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rver が 2 バイト境界でない
E_MACV	[m]	メモリアクセス違反

■機能

現在実行中のカーネルのバージョンに関する情報をし、pk_rver の指す領域に返します。
pk_rver の指すパケットには、次の情報を返します。

(1) maker

maker は、このカーネルを作ったメーカを表します。本カーネルの maker は、ルネサスを意味する H'0115 です。

(2) prid

prid は、OS や VLSI の種類を区別する番号を表します。本カーネルの prid は H'12 です。

(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)
  本カーネルのMAGIC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)
  本カーネルのSpecVerは、H'402(Ver.4.02)です。

(4) prver

prver は、カーネルのバージョン番号を表します。
例えば、カーネルのバージョンが V.1.02.13.456 の場合、prver は H'012D となります。

(5) prno

prno は、製品管理情報や製品番号などを表します。
本カーネルの prno[0]から prno[3]の値は、すべて H'0000 です。

(6) pxver

pxver は、カーネルの準拠している μ ITRON4.0 仕様保護機能拡張のバージョンを示します。
本カーネルの pxver は、H'0100(Ver.1.00)です。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rver対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rver
- size=sizeof(T_RVER)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.26 メモリオブジェクト管理機能

表6.52 メモリオブジェクト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	sac_mem	メモリオブジェクトのアクセス許可ベクタの変更	○		○	○	○		
2	prb_mem	メモリ領域に対するアクセス権のチェック	○		○	○	○		
3	ref_mem	メモリオブジェクト状態の参照	○		○	○	○		
4	vloc_tlb	TLB エントリのロック	○		○	○	○		
5	vunl_tlb	TLB エントリのロック解除	○		○	○	○		

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

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

アドレス空間は、MMU 対象領域と MMU 非対象領域に分けられ、全てのメモリオブジェクトは MMU 対象領域にのみ存在できます。詳細は、下記を参照してください。

■関連ページ 「5. 論理アドレス空間の扱い」

メモリオブジェクトへのアクセスは、各メモリオブジェクトに付与された属性とアクセス許可ベクタによって制御されます。詳細は、下記を参照してください。

■関連ページ 「4.21 メモリオブジェクト保護機能」

6.26.1 メモリオブジェクトのアクセス許可ベクタの変更(sac_mem)

■C 言語 API

```
ER ercd = sac_mem(VP base, ACVCT *p_acvct);
```

■パラメータ

VP	base	メモリオブジェクトのアドレス
ACVCT	*p_acvct	メモリオブジェクトのアクセス許可ベクタを格納したパケットアドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct{
           ACPTN   acptn1;      0      4    ライトアクセスに関するアクセス許可パターン
           ACPTN   acptn2;     +4      4    リードアクセスに関するアクセス許可パターン
} ACVCT;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)p_acvct の内容が不正 (2)p_acvct が 4 バイト境界でない
E_CTX	[k]	コンテキストエラー(非タスクコンテキストからの呼び出し)
E_ILUSE	[k]	サービスコール不正使用 (1)base を含むメモリオブジェクトが保護メールボックスにキューイングされている (2)base を含むメモリオブジェクトに、TLB ロック中のページが含まれている。
E_NOEXS	[k]	未登録 (1)base を含むメモリオブジェクトが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

base で示されるアドレスを含むメモリオブジェクトのアクセス許可ベクタを、p_acvct によって指定されるアクセス許可ベクタに変更します。ただし、以下のいずれかのメモリオブジェクトを指定した場合は、エラーE_ILUSE を返します。

- 保護メールボックスにキューイングされている保護メモリブロック
- TLB ロック中のページを含むメモリオブジェクト

p_acvct に指定できるアクセス許可ベクタは表 6.53に示す通りです。これらの組み合わせ(OR)の指定はできません。下記以外の指定の場合は、E_PAR エラーを返します。なお、呼び出し元ドメインにアクセス許可が無い指定であっても、特にエラーにはなりません。また、TA_RO 属性のメモリの場合は、ライトアクセス許可パターンに関わらず、カーネルドメインも含めて全てのプログラムはライトできません。

表6.53　指定可能なアクセス許可ベクタ

アクセス許可ベクタ	acptn1 (ライトアクセス許可パターン)	acptn2 (リードアクセス許可パターン)
TACT_KERNEL	TACP_KERNEL	TACP_KERNEL
TACT_PRW(domid) *	TACP(domid) *	TACP(domid) *
TACT_PRO(domid) *	TACP_KERNEL	TACP(domid) *
TACT_SRW	TACP_SHARED	TACP_SHARED
TACT_SRO	TACP_KERNEL	TACP_SHARED
TACT_SRPW(domid) *	TACP(domid) *	TACP_SHARED

【注】 これらのマクロの引数 domid には、1～31 のみ指定可能です。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_acvct対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=p_acvct
- size=sizeof(ACVCT)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.26.2 メモリ領域に対するアクセス許可のチェック(prb_mem)

■C 言語 API

```
ER ercd = prb_mem(VP base, SIZE size, ID domid, MODE pmmode);
```

■パラメータ

VP	base	メモリ領域の先頭アドレス
SIZE	size	メモリ領域のサイズ(バイト数)
ID	domid	アクセス元のドメイン ID
MODE	pmmode	アクセスモード

■リターンパラメータ

ER	ercd	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 (1)domid < -1 (2)domid > 31
E_PAR	[p]	パラメータエラー (1)pmmode が不正 (2)size が 0 (3)base+size が 32bit を超える
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_OBJ	[k]	オブジェクト不正 (1)base がメモリオブジェクト内のアドレスで、base + size-1 で指定された アドレスがそのメモリオブジェクト外
E_NOEXS	[k]	未登録 (1)base を含むメモリオブジェクトが存在しない
E_MACV	[m]	メモリアクセス違反(検出条件の詳細は「■機能」を参照してください)

■機能

base で示されるアドレスから size で指定されるサイズのメモリ領域が、domid で指定されるドメインに対してアクセスが許可されているかをチェックし、アクセスが許可されていれば E_OK を返します。

domid には以下を指定できます。

(a) 1～31：指定されたdomidのユーザドメイン

(b) TDOM_SELF(0)：呼び出し元ドメイン。ただし、タスクコンテキストで実行中の拡張サービスコールルーチンまたはトラップルーチンから発行した場合は、それらを呼び出したタスクが所属するドメイン(その拡張サービスコールルーチンまたはトラップルーチンからget_didを発行して得られるドメインIDと同じです)。

(c) TDOM_KERNEL(-1)：カーネルドメイン

pmmode には、チェックしたいアクセス種別を指定します。pmmode には、以下のいずれかまたは両方を指定できます。

- TPM_READ(H'00000001)リードアクセスが許可されているかをチェックする
- TPM_WRITE(H'00000002)ライトアクセスが許可されているかをチェックする

具体的には、以下の 3 通りの指定ができます。

(1) TPM_READ
リードアクセスが許可されているかをチェックします。
(2) TPM_WRITE
ライトアクセスが許可されているかをチェックします。本カーネルでは、ライトアクセスが許可されている場合は、常にリードアクセスも許可される仕様のため、(3)の指定と同じ意味になります。
(3) TPM_READ|TPM_WRITE
リード・ライトアクセスが共に許可されているかをチェックします。

本サービスコールは、base で指定されるアドレスがメモリオブジェクト内のアドレスかどうかによって振る舞いが異なります。具体的には、以下の手順でチェックを行います。

(1) base が MMU 対象領域のアドレスの場合

base を含むメモリオブジェクトが存在しない場合は、E_NOEXS を返します。
base を含むメモリオブジェクトが存在するが、指定範囲が単一のメモリオブジェクトの範囲に納まっていない場合には、E_OBJ エラーを返します。
また、以下の場合には E_MACV エラーを返します。

- pmmode に TPM_WRITE の指定があり、対象メモリオブジェクトが TA_RO 属性
- domid がユーザドメインの場合、対象メモリオブジェクトのアクセス許可ベクタに、domid の pmmode で指定されたアクセス権がない

(2) base が MMU 非対象領域のアドレスの場合

この場合、pmmode は無視され、domid はカーネルドメイン/ユーザドメインの種類の区別のみ意味を持ちます。

(a) base が内蔵メモリの論理アドレスの場合

コンフィギュレータの CFG_IRAMUSAGE を「MMU 非対象領域、全モードアクセス可能」または「MMU 非対象領域、ユーザ非 DSP モードではアクセス不可」と設定していた場合で、base がコンフィギュレータの[内蔵メモリ一覧]に指定した各内蔵メモリの範囲内のケースです。なお、「MMU 対象領域」と設定していた場合は、「(1) base が MMU 対象領域のアドレスの場合」のケースとなります。
base+size-1 が、その内蔵メモリの範囲外の場合、E_OBJ エラーを返します。
それ以外の場合は、「MMU 非対象領域、全モードアクセス可能」と設定していた場合は E_OK を返します。「MMU 非対象領域、ユーザ非 DSP モードではアクセス不可」と設定していた場合は、domid がユーザドメインの場合は E_MACV を返します。ただし、このケースでは、ユーザドメイン(SR.MD=0)であっても SR.DSP=1 の場合は内蔵メモリにアクセスすることはできますが、本サービスコールではエラーとなることに注意してください。一方、domid がカーネルドメインの場合は E_OK を返します。

(b) P3,P4 領域の場合

base,size で指定される領域が P3,P4 領域と重なる場合は、E_MACV エラーを返します。
P3 領域は本カーネルでは使用不可の仕様のため、常に E_MACV を返します。
P4 領域は、カーネルドメイン(特権モード)からのみアクセス可能な I/O 領域です。本サービスコールでは、P4 領域は検査対象外という理由で常に E_MACV を返します。

(c) (a),(b)以外の場合(P1,P2 領域)

この領域には、ユーザドメインからはアクセスできないため、domid がユーザドメインの場合は、E_MACV エラーを返します。domid がカーネルドメインの場合は E_OK を返します

6.26.3 メモリオブジェクト状態の参照(ref_mem)

■C 言語 API

```
ER ercd = ref_mem(VP base, T_RMEM *pk_rmem);
```

■パラメータ

VP	base	メモリオブジェクトのアドレス
T_RMEM	*pk_rmem	メモリオブジェクト状態を返すパケットのアドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMEM	*pk_rmem	メモリオブジェクト状態を格納したパケットのアドレス

■パケットの構造

```
typedef    struct {
           ACVCT   acvct;      0      8    メモリオブジェクトのアクセス許可ベクタ
} T_RMEM;
typedef    struct{
           ACPTN   acptn1;      0     4    ライトアクセスに関するアクセス許可パターン
           ACPTN   acptn2;     +4     4    リード・管理・参照アクセスに関するアクセス許可パ
                                           ターン
} ACVCT;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rmem が 4 の倍数でない
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録 (1)base を含むメモリオブジェクトが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

base で示されるアドレスを含むメモリオブジェクトの状態を参照します。
pk_rsem が指す領域に、対象メモリオブジェクトのアクセス許可ベクタ(acvct)を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmem対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmem
- size=sizeof(T_RMEM)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.26.4 TLB エントリのロック(vloc_tlb)

■C 言語 API

```
ER ercd = vloc_tlb(VP base);
```

■パラメータ

VP	base	TLB にロックするページ内のアドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	[k]	パラメータエラー
		(1)既に CFG_MAXLOCPAGE のページがロックされている
E_CTX	[k]	コンテキストエラー
		(1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用
		(1)base を含むメモリオブジェクトが保護メールボックスにキューイングされている
		(2)base を含むページは既にロック済み
		(3)base を含むメモリオブジェクトのロックが許可されていない
E_NOEXS	[k]	未登録
		(1)base を含むメモリオブジェクトが存在しない

■機能

base で示されるアドレスを含むメモリオブジェクトの MMU の 1 ページを TLB に登録すると共に、その TLB エントリをロックします。ただし、以下のいずれかのメモリオブジェクト内のページを指定した場合は、エラーE_ILUSE を返します。

- 保護メールボックスにキューイングされている保護メモリブロック
- 表 6.54に示すメモリ属性とアクセス許可ベクタの組み合わせを持つメモリオブジェクト

表6.54 ロックできないメモリオブジェクト

アクセス許可ベクタ	メモリ属性
TACT_PRO(domid)	TA_RW
TACT_SRO	TA_RW
TACT_SRPW(domid)	TA_RW

指定されたページがロックされていない状態で TLB に登録されている場合は、そのエントリをパージするとともに、TLB の別エントリに登録し直します。
既にロックされている場合は、E_ILUSE エラーとなります。

ロックされたページは、vunl_tlb サービスコールによってロックを解除できます。

本サービスコールにより、システムでロック可能なページ数が 1 減ります。同時にロック可能なページ数は、コンフィギュレータの CFG_MAXLOCPAGE によって定義されます。
TLB にロックされたエントリは、以降 TLB から追い出されることがなくなるため、そのページへのアクセスで TLB ミスヒットすることがなくなります。ただし、一般には TLB にページをロックさせることで、その他のロックされていないページへアクセスした時に TLB ミスヒットする確率が増えることに注意してください。

ロックしたページを含むメモリオブジェクトに対しては、sac_mem, snd_mbp を行うことはできなくなるので、注意してください。

また、ロックしたページを含むメモリオブジェクトに関して、メモリオブジェクトの削除に相当する表 6.55の操作を行う前に、必ずロック解除してください。ロック解除せずにこれらの操作を行った場合、ロック可能なページ数が CFG_MAXLOCPAGE よりも少なくなります。

表6.55　メモリオブジェクト削除操作

base を含むメモリオブジェクト	メモリオブジェクト削除操作
システムプールから獲得したユーザドメイン所属タスクのスタック領域	del_tsk, exd_tsk
システムプールから獲得した固定長メモリプール領域	del_mpf
システムプールから獲得した可変長メモリプール領域	del_mpl
保護メモリプールから獲得した保護メモリブロック、または保護メールボックスから受信した保護メモリブロック	rel_mpp
静的メモリオブジェクト	無し

■関連ページ　「表 10.14　拡張機能を持たない SH4AL-DSP, SH-4A 使用時のページサイズ」

6.26.5 TLB エントリのロック解除(vunl_tlb)

■C 言語 API

```
ER ercd = vunl_tlb(VP base);
```

■パラメータ

VP	base	TLB ロック解除するページ内のアドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)base を含むページはロックされていない
E_NOEXS	[k]	未登録 (1)base を含むメモリオブジェクトが存在しない。

■機能

base で示されるアドレスを含むメモリオブジェクトの MMU の 1 ページの TLB エントリのロックを解除します。

指定されたアドレスを含むページがロックされていない場合は、E_ILUSE エラーとなります。

本サービスコールにより、システムでロック可能なページ数が 1 増えます。

■関連ページ　「表 10.14　拡張機能を持たない SH4AL-DSP, SH-4A 使用時のページサイズ」

6.27 保護メモリプール管理機能

表6.56 保護メモリプール管理機能サービスコール

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	icre_mpp	保護メモリプールの生成	コンフィギュレータが生成する初期登録ルーチン専用です。その他の状態で呼び出した場合の動作は保証されません						
2	pget_mpp	保護メモリブロックの獲得(ポーリング)	○		○	○	○		
3	rel_mpp	保護メモリブロックの返却	○		○	○	○		
4	ref_mpp	保護メモリプールの状態参照	○		○	○	○		

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

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

表6.57 保護メモリプール管理機能の仕様

項番	項目	内容
1	保護メモリプール ID	1～CFG_MAXMPPID(最大 31)
2	可変長メモリプール属性	・VTA_UNFRAGMENT：セクタ管理方式 (空き領域の断片化が発生しにくい方式)

保護メモリプールは、コンフィギュレータで静的に生成します。サービスコールで生成することはできません。

また、以下も参照してください。

■関連ページ 「4.31 メモリの断片化とその対策」

6.27.1 保護メモリプールの生成(icre_mpp)

■C 言語 API

```
ER ercd = icre_mpp(ID mppid, T_CMPP *pk_cmpp);
```

■パラメータ

ID	mppid	保護メモリプール ID
T_CMPP	*pk_cmpp	保護メモリプール生成情報を格納したパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mppatr;      0      4    保護メモリプール属性
           SIZE    mppsz;       +4     4    保護メモリプール領域のサイズ(バイト数)
           VP      mpp;         +8     4    保護メモリプール領域の先頭アドレス
           VP      mppmb;       +12    4    保護メモリプール管理領域の先頭アドレス
           UINT    minblksz;    +16    4    最小ブロックサイズ
           UINT    sctnum;      +20    4    最大セクタ数
} T_CMPP;
```

■エラーコード

E_PAR	[k]	パラメータエラー (1)mpp から mppsz バイトが MMU 対象領域でない。 (2)mpp が CFG_PAGESZ 境界でない (3)VTA_UNFRAGMENT 属性の場合で、sctnum=0

■機能

mpfid で示された ID を持つ保護メモリプールを、pk_cmpp で示された内容で生成します。

本サービスコールは、アプリケーションから呼び出してはなりません。本サービスコールは、コンフィギュレータで保護メモリプールの生成を行っていた場合に、コンフィギュレータが出力する初期登録ルーチンのみから呼び出されます。本サービスコールはこの前提で実装されているため、ほとんどのエラー検出を省略しています。

以下に各パラメータの意味を示します。

(1) mppatr

mematr には、以下の論理和を指定してください。

(a) メモリブロック獲得を待つ待ち行列に並ぶ際の並び方

TA_TFIFO のみを指定できます。

- TA_TFIFO(H'00000000)：メモリ獲得待ちタスクのキューイングは FIFO 順

(b) リードオンリー/リードライト可能、キャッシュに関する設定

TA_RW, TA_RO のいずれかを指定できます。

- TA_RW(H'00000000)：リード・ライト可能なメモリ(RAM)
- TA_RO(H'00000001)：リードオンリーのメモリ(ROM)

(c) キャッシュに関する設定

以下を指定できます。

```
TA_UNCACHE || ( TA_CHCHE | [TA_WBACK || TA_WTHROUGH] )
```

- TA_CACHE(H'00000000)：リード・ライトアクセス時、キャッシュする
- TA_UNCACHE(H'00000002)：リード・ライトアクセス時、キャッシュしない
- TA_WBACK (H'00000000)：ライトアクセスをコピーバック動作とする
- TA_WTHROUGH(H'00000004)：ライトアクセスをライトスルー動作とする

(d) 管理方式

VTA_UNFRAGMENT を指定できます。

- VTA_UNFRAGMENT(H'80000000)：セクタ管理方式(空き領域の断片化が発生しにくい方式)

VTA_UNFRAGMENT 属性は、微小なメモリブロックを大量に獲得するメモリプールに適した属性で、微小なブロックをできるだけ連続して配置することで、大きなサイズの連続空き領域が維持されやすくします。

VTA_UNFRAGMENT 属性を指定した場合のみ、sctnum が有効になります。sctnum は、mppsz/(4096×32)よりも大きい場合は、mppsz/(4096×32)として扱います。

詳細は、以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

(2) mpp と mppsz

mpp には生成する保護メモリプールのアドレスを、mppsz にはサイズを指定してください。

指定するメモリ領域は、MMU 対象領域で、かつ CFG_PAGESZ の境界アドレスでなければなりません。そうでない場合は、E_PAR エラーが返ります。

(3) minblksz と sctnum

これらはμITRON 仕様外の項目です。

これらは VTA_UNFRAGMENT 属性が指定された場合のみ有効ですが、minblksz は無視されて常に CFG_PAGESZ(4096)と扱います。詳細は、前述の VTA_UNFRAGMENT 属性の説明を参照してください。

(4) mppmb

mppmb には、カーネル管理領域のアドレスを指定します。mppmb の実体はコンフィギュレータによって生成されます。なお、mppmb は、MMU 非対象領域でユーザモードからアクセスできない領域でなければなりません。

6.27.2 保護メモリブロックの獲得(ポーリング)(pget_mpp)

■C 言語 API

```
ER_UINT blksz = pget_mpp(ID mppid, UINT memsz, VP *p_blk);
```

■パラメータ

ID	mppid	保護メモリプール ID
UINT	memsz	獲得するメモリブロックのサイズ
VP	*p_blk	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

■リターンパラメータ

ER_UINT	blksz	獲得したメモリブロックのサイズ(正の値)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_PAR		パラメータエラー
	[p]	(1)p_blk が 4 バイト境界でない
	[k]	(2)memsz＞対象の保護メモリプールのサイズ
E_ID	[p]	不正 ID 番号
		(1)mppid≦0
		(2)mppid＞CFG_MAXMPPID
E_CTX	[k]	コンテキストエラー
		(1)非タスクコンテキストからの呼び出し
E_NOMEM	[k]	メモリ不足
		(1)リソースプールの空き不足
E_NOEXS	[k]	未登録
		(1)mppid の保護メモリプールが存在しない
E_TMOUT	[k]	ポーリング失敗
E_MACV	[m]	メモリアクセス違反

■機能

mppid で示される保護メモリプールから、memsz を CFG_PAGESZ で切り上げたサイズ(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、そのブロックサイズを blksz に返します。

獲得したメモリブロックは、以下の性質のメモリオブジェクトと扱われます。

- 所属ドメイン：呼び出したタスクが所属するドメインになります。これは、get_did を発行して得られるドメイン ID と同じです。
- メモリ属性：mppid の保護メモリプールのメモリ属性(コンフィギュレータで指定)
- アクセス許可ベクタ：
  (a) メモリブロックの所属ドメインがカーネルドメインの場合：TACT_KERNEL
  (b) メモリブロックの所属ドメインがユーザドメインの場合：TACT_PRW(domid)
  (domidはメモリブロックの所属ドメインID)
- 先頭アドレス：4kB 境界のアドレス

アクセス許可ベクタを変更したい場合は、sac_mem を使って変更してください。

獲得により、対象の保護メモリプールの空き領域は blksz だけ減少します。対象の保護メモリプールにこのサイズの連続空き領域がない場合は、E_TMOUT エラーが返ります。

また、カーネルは獲得したメモリブロックをメモリオブジェクトとして管理するためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　「13.2.3(3) 保護メモリプール：pget_mpp」

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_blkに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base= p_blk
- size=sizeof(VP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.27.3 保護メモリブロックの返却(rel_mpp)

■C 言語 API

```
ER ercd = rel_mpp(ID mppid, VP blk);
```

■パラメータ

ID	mppid	保護メモリプール ID
VP	blk	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	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR		パラメータエラー
	[p]	(1)blk が CFG_PAGESZ 境界でない
	[k]	(2)blk が対象保護メモリプール領域外のアドレス
E_ID	[p]	不正 ID 番号 (1)mppid≦0 (2)mppid＞CFG_MAXMPPID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_ILUSE	[k]	サービスコール不正使用 (1)blk は、対象の保護メモリプールから獲得した保護メモリブロックの先頭アドレスでない (2)blk で指定された保護メモリブロックの所属ドメインが、実行状態のタスクが所属するドメインと一致しない (3)blk で指定された保護メモリブロックが、保護メールボックスにキューイングされている
E_NOEXS	[k]	未登録 (1)mppid の保護メモリプールが存在しない

■機能

mppid で示された保護メモリプールへ blk で示されたメモリブロックを返却します。

blk には、pget_mpp 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。

実行状態のタスクが所属するドメイン(get_did で得ることができます)と対象のメモリブロックの所属ドメインが同じ場合にのみ返却できます。そうでない場合は、E_ILUSE エラーを返します。以下に例を示します。

(1) メモリブロックの所属ドメインがAで、ドメインBに所属するタスクから発行すると、E_ILUSEエラーが帰ります。

(2) メモリブロックの所属ドメインがAで、同じドメインAに所属するタスクから呼び出された拡張サービスコールルーチンから発行すると、拡張サービスコールルーチンはカーネルドメインですが、「実行状態のタスクが所属するドメイン」はタスクが所属するドメインAのことであり、それはメモリブロックの所属ドメインと同じなので返却可能です。

返却により、保護メモリプールの空き領域は返却したブロックのサイズだけ増加します。また、リソースプールから割り付けられていた管理領域も解放されます。

6.27.4 保護メモリプールの状態参照(ref_mpp)

■C 言語 API

```
ER ercd = ref_mpp(ID mppid, T_RMPP *pk_rmpp);
```

■パラメータ

ID	mppid	保護メモリプール ID
T_RMPP	*pk_rmpp	保護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	正常終了(E_OK)またはエラーコード
T_RMPP	*pk_rmpp	保護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 {
           ID      wtskid;      0      2    待ちタスク ID
           SIZE    fmppsz;      +4     4    空き領域の合計サイズ(バイト数)
           UINT    fblksz;      +8     4    獲得可能な最大メモリブロックサイズ(バイト数)
           SIZE    mppsz;       +12    4    保護メモリプールのサイズ
} T_RMPP;
```

■エラーコード

E_PAR	[p]	パラメータエラー
		(1)pk_rmpp が 4 バイト境界でない
E_ID	[p]	不正 ID 番号
		(1)mppid≦0
		(2)mppid＞CFG_MAXMPPID
E_CTX	[k]	コンテキストエラー
		(1)非タスクコンテキストからの呼び出し
E_NOEXS	[k]	未登録
		(1)mppid の保護メモリプールが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

mppid で示された保護メモリプールの状態を参照します。

pk_rmpp 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(fmppsz)、獲得可能な最大メモリブロックのサイズ(fblksz)、および保護メモリプールのサイズ(mppsz)を返します。mppsz は μITRON 仕様外の項目です。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。1 回の pget_mpp 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。

wtskid は将来拡張用で、本カーネルでは常に 0 を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmppに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmpp
- size=sizeof(T_RMPP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.28 保護メールボックス管理機能

表6.58 保護メールボックス管理機能サービスコール

項番	サービスコール *1	機 能	呼び出し可能なシステム状態 *2						
			T	N	E	D	U	L	C
1	cre_mbp	保護メールボックスの生成	○		○	○	○		
	icre_mbp			◯	○	○	○		
2	acre_mbp	保護メールボックスの生成 (ID 番号自動割付け)	○		○	○	○		
	iacre_mbp			◯	○	○	○		
3	del_mbp	保護メールボックスの削除	○		○	○	○		
4	snd_mbp	保護メールボックスへの送信	○		○	○	○		
5	rcv_mbp	保護メールボックスからの受信	○		○		○		
6	prcv_mbp	同上(ポーリング)	○		○	◯	○		
7	trcv_mbp	同上(タイムアウト)	○		○		○		
8	ref_mbp	保護メールボックスの状態参照	○		○	○	○		
	iref_mbp			◯	○	○	○		

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

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

表6.59 保護メールボックス管理機能の仕様

項番	項目	内容
1	保護メールボックス ID	1～CFG_MAXMBPID (最大 32767)
2	保護メッセージ優先度	1～CFG_MAXMSGPRI゛(最大 255)
3	保護メールボックス属性	・TA_TFIFO：待ちタスクのキューイングは FIFO 順 ・TA_TPRI：待ちタスクのキューイングは現在優先度順 ・TA_MFIFO：メッセージのキューイングは FIFO 順 ・TA_MPRI：メッセージのキューイングは優先度順

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

6.28.1 保護メールボックスの生成(cre_mbp, icre_mbp, acre_mbp, iacre_mbp)

■C 言語 API

```
ER ercd = cre_mbp(ID mbpid, T_CMBP *pk_cmbp);
ER ercd = icre_mbp(ID mbpid, T_CMBP *pk_cmbp);
ER_ID mbpid = acre_mbp(T_CMBP *pk_cmbp);
ER_ID mbpid = iacre_mbp(T_CMBP *pk_cmbp);
```

■パラメータ

T_CMBP	*pk_cmbp	保護メールボックス生成情報を格納したパケットへのポインタ
《cre_mbp, icre_mbp》		
ID	mbpid	保護メールボックス ID

■リターンパラメータ

《cre_mbp, icre_mbp》		
ER	ercd	正常終了(E_OK)またはエラーコード
《acre_mbp, iacre_mbp》		
ER_ID	mbpid	生成した保護メールボックスの ID 番号(正の値) またはエラーコード

■パケットの構造

```
typedef    struct {
           ATR     mbpatr;      0      4    保護メールボックス属性
           UINT    mbpcnt;      +4     4    格納可能メッセージ数
           PRI     maxmpri;     +8     2    メッセージ優先度の最大値
           VP      mbpmb;       +12    4    保護メールボックス管理領域の先頭アドレス
} T_CMBP;
```

■エラーコード

E_RSATR	[p]	予約属性(mbpatr が不正)
E_PAR	[p]	パラメータエラー (1)maxmpri≦0 (2)maxmpri＞CFG_MAXMSGPRI (3)pk_cmbp が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbpid≦0 (2)mbpid＞CFG_MAXMBPID
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_NOID	[k]	空き ID なし(acre_mbp のみ)
E_OBJ	[k]	オブジェクト状態不正 (1)mbpid の保護メールボックスが存在する
E_MACV	[m]	メモリアクセス違反

■機能

cre_mbp, icre_mbp サービスコールは、mbpid で示された ID を持つ保護メールボックスを、pk_cmbp で示された内容で生成します。

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

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

```
mbpatr := ( (TA_TFIFO | TA_TPRI) | (TA_MFIFO | TA_MPRI) )
```

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

mbpcnt、mbpmb は、本カーネルでは常に無視されます。プログラムの移植性のためには、mbpcnt は適当な値、mbpmb には NULL を指定してください。

TA_MPRI 属性を指定した場合で maxmpri>1 の場合、カーネルはメールボックスを管理するためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　「13.2.2(7) 保護メールボックス」

なお、保護メールボックスはコンフィギュレータで静的に生成することもできます。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_cmbpに対する呼び出し元ドメインのリードアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=pk_cmbp
- size=sizeof(T_CMBP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ

6.28.2 保護メールボックスの削除(del_mbp)

■C 言語 API

```
ER ercd = del_mbp(ID mbpid);
```

■パラメータ

ID	mbpid	保護メールボックス ID

■リターンパラメータ

ER	ercd	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 (1)mbpid≦0 (2)mbpid＞CFG_MAXMBPID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_OBJ	[k]	オブジェクト状態不正 (1)対象の保護メールボックスにキューイングされているメッセージが存在する
E_NOEXS	[k]	未登録 (1)mbpid の保護メールボックスが存在しない

■機能

mbpid で示された保護メールボックスを削除します。

mbpid で示された保護メールボックスでメッセージを待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

保護メールボックス内にメッセージが存在する場合は、E_OBJ エラーが返ります。

削除によって、リソースプールから割り付けられていた管理領域は解放されます。

6.28.3 保護メールボックスへの送信(snd_mbp)

■C 言語 API

```
ER ercd = snd_mbp(ID mbpid, VP blk, PRI msgpri);
```

■パラメータ

ID	mbpid	保護メールボックス ID
VP	blk	送信メッセージを格納した保護メモリブロックの先頭アドレス
PRI	msgpri	メッセージの優先度

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR		パラメータエラー
	[p]	(1)blk が CFG_PAGESZ 境界でない
	[k]	(2)対象の保護メールボックスに TA_MPRI 属性が指定されている場合で、 msgpri≦0 または msgpri＞(生成時に指定した最大メッセージ優先度)
E_ID	[p]	不正 ID 番号 (1)mbpid≦0 (2)mbpid＞CFG_MAXMBPID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_NOMEM	[k]	メモリ不足 (1)リソースプールの空き不足
E_ILUSE	[k]	サービスコール不正使用 (1)先頭アドレスが blk の保護メモリブロックが存在しない (2)blk で指定された保護メモリブロックの所属ドメインが、実行状態のタスクが所属するドメインと一致しない (3)blk で指定された保護メモリブロックが、保護メールボックスにキューイングされている (4)blk を含むメモリオブジェクトに、TLB ロック中のページが含まれている。
E_NOEXS	[k]	未登録 (1)mbpid の保護メールボックスが存在しない

■機能

mbpid で示された保護メールボックスに、blk を先頭アドレスとする保護メモリブロックをメッセージとして、msgpri で指定される優先度で送信します。ただし、mbpid の保護メールボックスに TA_MPRI 属性が指定されていない場合は、msgpri は無視されます。

blk に指定できるのは、保護メモリブロック(pget_mpp で獲得したメモリブロック)の先頭アドレスのみです。その他のアドレスを指定した場合は、E_ILUSE エラーを返します。

また、以下のいずれかの保護メモリブロックも送信できません。この場合、エラーE_ILUSE を返します。

- 保護メールボックスにキューイングされている保護メモリブロック
- TLB ロック中のページを含む保護メモリブロック

blk に指定する保護メモリブロックの所属ドメインは、実行状態のタスクが所属するドメイン(get_did で得ることができます)と同じでなければなりません。そうでない場合は、E_ILUSE エラーを返します。以下に例を示します。

(1) メモリブロックの所属ドメインがAで、ドメインBに所属するタスクから発行すると、E_ILUSEエラーが帰ります。

(2) メモリブロックの所属ドメインがAで、同じドメインAに所属するタスクから呼び出された拡張サービスコールルーチンから発行すると、拡張サービスコールルーチンはカーネルドメインですが、「実行状態のタスクが所属するドメイン」はタスクが所属するドメインAのことであり、それはメモリブロックの所属ドメインと同じなので送信可能です。

すでに対象保護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。この時、送信した保護メモリブロックは、以下の性質に変更されます。

- 所属ドメイン：受信タスクの所属ドメインになります。これは、get_did を発行して得られるドメイン ID と同じです。
- アクセス許可ベクタ：
  (a) メモリブロックの所属ドメインがカーネルドメインの場合：TACT_KERNEL
  (b) メモリブロックの所属ドメインがユーザドメインの場合：TACT_PRW(domid)
  (domidはメモリブロックの所属ドメインID)

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。この時、カーネルはメッセージ管理のためにリソースプールを消費します。詳細は、以下を参照してください。

■関連ページ　・リソースプールの消費→「13.2.3(4) 保護メールボックス：snd_mbp」

メッセージの待ち行列は、生成時に指定した属性にしたがって管理されます。この時、送信された保護メモリブロックは、以下の性質に変更されます。

- 所属ドメイン：カーネルドメイン
- アクセス許可ベクタ：TACT_KERNEL

6.28.4 保護メールボックスからの受信(rcv_mbp, prcv_mbp, trcv_mbp)

■C 言語 API

```
ER_UINT blksz = rcv_mbp(ID mbpid, VP *p_blk);
ER_UINT blksz = prcv_mbp(ID mbpid, VP *p_blk);
ER_UINT blksz = trcv_mbp(ID mbpid, VP *p_blk, TMO tmout);
```

■パラメータ

ID	mbpid	保護メールボックス ID
VP	*p_blk	受信メッセージが格納される保護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
《trcv_mbp》		
TMO	tmout	タイムアウト指定

■リターンパラメータ

ER_UINT	blksz	受信した保護メモリブロックのサイズ(正の値)またはエラーコード
VP	*p_blk	受信メッセージが格納された保護メモリブロックの先頭アドレスへのポインタ

■エラーコード

E_PAR	[p]	パラメータエラー (1)tmout≦-2 (2)p_blk が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbpid≦0 (2)mbpid＞CFG_MAXMBPID
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し (2)タスクコンテキストで、ディスパッチ保留状態からの呼び出し (rcv_mbp, trcv_mbp のみ)
E_NOEXS	[k]	未登録 (1)mbpid の保護メールボックスが存在しない
E_RLWAI	[k]	待ち状態強制解除(rcv_mbp, trcv_mbp のみ) (1)待ちの間に rel_wai サービスコールが呼び出された (2)待ち禁止状態で待ち状態に移行しようとした
E_TMOUT	[k]	ポーリング失敗、またはタイムアウト
E_DLT	[k]	待ちオブジェクト削除 (1)mbpid の保護メールボックスが削除された
E_MACV	[m]	メモリアクセス違反

■機能

mbpid で示された保護メールボックスから保護メモリブロックに格納されたメッセージを受信し、その保護メモリブロックの先頭アドレスを p_blk に、サイズを blksz に返します。

保護メールボックスにメッセージが存在しない場合は、rcv_mbp, trcv_mbp 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_mbp サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

メッセージを受信すると、メッセージ送信時にカーネルがメッセージ管理のためにリソースプールから確保した管理領域が解放されます。

受信した保護メモリブロックは、以下の性質に変更されます。

- 所属ドメイン：呼び出し元ドメイン。ただし、タスクコンテキストで実行中の拡張サービスコールルーチンまたはトラップルーチンから発行した場合は、それらを呼び出したタスクが所属するドメイン(その拡張サービスコールまたはトラップルーチンから get_did を発行して得られるドメイン ID と同じです)
- アクセス許可ベクタ：
  (a)カーネルドメインに所属するタスクから受信サービスコール(rcv_mbp, prcv_mbp, trcv_mbp)を発行し
  ていた場合：TACT_KERNEL
  (b)ユーザドメインに所属するタスクから受信サービスコール(rcv_mbp, prcv_mbp, trcv_mbp)を発行して
  た場合、またはそのタスクから呼び出された拡張サービスコールルーチンまたはトラップルーチンから受信サービスコール(rcv_mbp, trcv_mbp)を発行していた場合：
  TACT_PRW(domid)　domidはそのタスクが所属するドメイン

trcv_mbp 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_mbp 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_mbp サービスコールと同じ処理を行います。

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

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) p_blkに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。
- base=p_blk
- size=sizeof(VP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.28.5 保護メールボックスの状態参照(ref_mbp, iref_mbp)

■C 言語 API

```
ER ercd = ref_mbp(ID mbpid, T_RMBP *pk_rmbp);
ER ercd = iref_mbp(ID mbpid, T_RMBP *pk_rmbp);
```

■パラメータ

ID	mbpid	保護メールボックス ID
T_RMBP	*pk_rmbp	保護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	正常終了(E_OK)またはエラーコード
T_RMBP	*pk_rmbp	保護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

```
typedef    struct{
           ID      wtskid;      0      2    待ちタスク ID
           VP      blk;         +4     4    メッセージキューの先頭の保護メモリブロックの先頭
                                             アドレス
           UINT    blksz;       +8     4    メッセージキューの先頭の保護メモリブロックのサイズ
} T_RMBP;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rmbp が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)mbpid≦0 (2)mbpid＞CFG_MAXMBPID
E_NOEXS	[k]	未登録 (1)mbpid の保護メールボックスが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

mbpid で示された保護メールボックスの状態を参照します。

pk_rmbp が示す領域に待ちタスク ID(wtskid)、次に受信される保護メモリブロックの先頭アドレス(pk_msg)、およびその保護メモリブロックのサイズ(blksz)を返します。対象保護メールボックスの待ちタスクが無い場合は、wtskid に 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、blk に NUL(0)、blksz に不定値を返します。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rmbpに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rmbp
- size=sizeof(T_RMBP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.29 システムメモリ管理機能

表6.60 システムメモリ管理機能サービスコール

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	vref_syp	システムプールの状態参照	○		○	○	○		
2	vref_rsp	リソースプールの状態参照	○		○	○	○		

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

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

システムプールとリソースプールについては、以下を参照してください。

■関連ページ 「4.24 システムメモリ管理機能」

また、以下も参照してください。

■関連ページ 「4.31 メモリの断片化とその対策」

6.29.1 システムプールの状態参照(vref_syp)

■C 言語 API

```
ER ercd = vref_syp(VT_RSYP *pk_rsyp);
```

■パラメータ

VT_RSYP	*pk_rsyp	システム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	正常終了(E_OK)またはエラーコード
VT_RSYP	*pk_rsyp	システム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 {
           ID      wtskid;      0      2    待ちタスク ID
           SIZE    freesz;      +4     4    空き領域の合計サイズ(バイト数)
           UINT    fblksz;      +8     4    獲得可能な最大メモリブロックサイズ(バイト数)
           SIZE    sypsz;       +12    4    システムプールのサイズ
} VT_RSYP;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rsyp が 4 バイト境界でない
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_MACV	[m]	メモリアクセス違反

■機能

システムプールの状態を参照します。

pk_rsyp が指す領域に、現在の空き領域の合計サイズ(freesz)、獲得可能な最大メモリブロックのサイズ(fblksz)、およびシステムプールのサイズ(sypsz)を返します。wtskid には、常に NTSK を返します。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rsypに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rsyp
- size=sizeof(VT_RSYP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.29.2 リソースプールの状態参照(vref_rsp)

■C 言語 API

```
ER ercd = vref_rsp(VT_RRSP *pk_rrsp);
```

■パラメータ

VT_RRSP	*pk_rrsp	リソース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	正常終了(E_OK)またはエラーコード
VT_RRSP	*pk_rrsp	リソース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 {
           ID      wtskid;      0      2    待ちタスク ID
           SIZE    freesz;      +4     4    空き領域の合計サイズ(バイト数)
           UINT    fblksz;      +8     4    獲得可能な最大メモリブロックサイズ(バイト数)
           SIZE    rspsz;       +12    4    リソースプールのサイズ
} VT_RRSP;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rrsp が 4 バイト境界でない
E_CTX	[k]	コンテキストエラー (1)非タスクコンテキストからの呼び出し
E_MACV	[m]	メモリアクセス違反

■機能

リソースプールの状態を参照します。

pk_rrsp が指す領域に、現在の空き領域の合計サイズ(freesz)、獲得可能な最大メモリブロックのサイズ(fblksz)、およびリソースプールのサイズ(rspsz)を返します。wtskid には、常に NTSK を返します。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rrspに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rrsp
- size=sizeof(VT_RRSP)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

6.30 パフォーマンス管理機能

表6.61 保護メールボックス機能サービスコール

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	vchg_ppc	パフォーマンス測定の開始・停止・初期化	○		○	○	○		
2	ivchg_ppc			○	○	○	○		
3	vref_ppc	パフォーマンス測定結果の参照	○		○	○	○		
4	ivref_ppc			○	○	○	○		

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

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

パフォーマンス管理機能は、マイコンが内蔵するプログラムパフォーマンスカウンタ(PPC)を用いてタスクなどの実行時間などを測定する機能です。本機能を使用するには、使用するマイコンがプログラムパフォーマンスカウンタを内蔵している必要があります。

■関連ページ 「4.26 パフォーマンス管理機能」

6.30.1 パフォーマンス測定の開始・停止・初期化(vchg_ppc, ivchg_ppc)

■C 言語 API

```
ER_UINT oldmode = vchg_ppc(ID ctxid, MODE mode);
ER_UINT oldmode = ivchg_ppc(ID ctxid, MODE mode);
```

■パラメータ

ID	ctxid	対象コンテキスト
MODE	mode	累積モード

■リターンパラメータ

ER_UINT	oldmode	変更前の累積モード

■エラーコード

E_PAR	[k]	パラメータエラー (1)mode が不正
E_ID	[p]	不正 ID 番号 (1)ctxid<-3 (2)ctxid>CFG_MAXTSKID (3)非タスクコンテキストからの呼び出しで、ctxid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)ctxid のタスクが存在しない

■機能

ctxid で示されるコンテキストのパフォーマンスカウンタ測定を開始/停止/初期化します。
ctxid には、以下を指定できます。

- 1～CFG_MAXTSKID：タスク ID が ctxid のタスク
- TSK_SELF(0) : 呼び出し元タスク。呼び出し元が非タスクコンテキストの場合は E_ID エラー
- -1：カーネルアイドリング
- -2：非タスクコンテキスト＋カーネル
- -3：全プログラム

mode では、カウンタの停止・再開に関する設定を行います。
mode := (VTPPC_STA0||VTPPC_STP0) | (VTPPC_STA1||VTPPC_STP1) [|VTPPC_INI0][|VTPPC_INI1]

- VTPPC_STA0(H'00000001)：カウンタ 0 の累積を再開
- VTPPC_STP0(H'00000000)：カウンタ 0 の累積を停止
- VTPPC_INI0(H'00000004)：カウンタ 0 の累積データを 0 クリア
- VTPPC_STA1(H'00000002)：カウンタ 1 の累積を再開
- VTPPC_STP1(H'00000000)：カウンタ 1 の累積を停止
- VTPPC_INI1(H'00000008)：カウンタ 1 の累積データを 0 クリア

oldmode には、ctxid に-3 以外を指定した場合には変更前の累積モードとして以下が返ります。
oldmode := (VTPPC_STA0|VTPPC_STP0) | (VTPPC_STA1|VTPPC_STP1)

- VTPPC_STA0(H'00000001)：カウンタ 0 の累積を実行
- VTPPC_STP0(H'00000000)：カウンタ 0 の累積は停止
- VTPPC_STA1(H'00000002)：カウンタ 1 の累積を実行中
- VTPPC_STP1(H'00000000)：カウンタ 1 の累積は停止

一方、ctxid に-3 を指定した場合は、odlmode には不定値が返ります。

CFG_CONNECT をチェックしていた場合は、mode でのカウンタ 1 に対する設定は無視されます。また、oldmode でのカウンタ 1 に対する情報は意味を持ちません。

「カーネルアイドリング」および「非タスクコンテキスト＋カーネル」については、vsta_knl 時にカウンタの累積データが 0 クリアされ、累積が開始されます。一方、各タスクについては、タスク生成時にカウンタの累積データが 0 クリアされ、累積が開始されます。

なお、パフォーマンスカウンタそのものを開始・停止・初期化することはできません。

6.30.2 パフォーマンス測定結果の参照(vref_ppc, ivref_ppc)

■C 言語 API

```
ER_UINT sts = vref_ppc(ID ctxid, VT_RPPC *pk_rppc);
ER_UINT sts = ivref_ppc(ID ctxid, VT_RPPC *pk_rppc);
```

■パラメータ

ID	ctxid	対象コンテキスト
VT_RPPC	*pk_rppc	累積値を返すパケットへのポインタ

■リターンパラメータ

ER_UINT	sts	PPC 状態(正の値)またはエラーコード
VT_RPPC	*pk_rppc	累積値を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct {
           UW      ppc0;       0      4    PPC0 カウンタ
           UW      ppc1;       +4     4    PPC1 カウンタ
} VT_RPPC;
```

■エラーコード

E_PAR	[p]	パラメータエラー (1)pk_rppc が 4 バイト境界でない
E_ID	[p]	不正 ID 番号 (1)ctxid<-2 (2)ctxid>CFG_MAXTSKID (3)非タスクコンテキストからの呼び出しで、ctxid=TSK_SELF(0)を指定
E_NOEXS	[k]	未登録 (1)ctxid のタスクが存在しない
E_MACV	[m]	メモリアクセス違反

■機能

各コンテキストのパフォーマンスカウンタ累積値を参照します。

ctxid には、以下を指定できます。

- 1～CFG_MAXTSKID：タスク ID が ctxid のタスク
- TSK_SELF(0)：呼び出し元タスク。発行元が非タスクコンテキストの場合は E_ID エラー
- -1：カーネルアイドリング
- -2：非タスクコンテキスト

pk_rppc には、ctxid で指定したコンテキストのパフォーマンスカウンタ累積値が返ります。

sts には、以下の情報が返ります。1,2 が返るケースでは、pk_rppc の情報は正確でない可能性があります。正確かどうかを判断することはできません。

- 0：パフォーマンスカウンタ 0,1 共にオーバフローなし
- 1：パフォーマンスカウンタ 0 のみオーバフロー(CFG_CONNECT をチェックした場合はありえません)
- 2：パフォーマンスカウンタ 0, 1 共にオーバフロー

■CFG_MEMCHK によるエラー検出

以下の場合に、エラーE_MACV を返します。

(1) pk_rppcに対する呼び出し元ドメインのリード・ライトアクセス許可が無い。
prb_memを以下のパラメータで発行してエラーが返るケースと同じです。

- base=pk_rppc
- size=sizeof(VT_RPPC)
- domid=呼び出し元ドメイン
- pmmode=TPM_READ|TPM_WRITE

7. キャッシュサポート関数

7.1 概要

キャッシュサポート関数は、キャッシュをメモリに書き戻したり、クリアしたりする関数です。

ヘッダファイルは、include¥ディレクトリにあります。キャッシュサポート関数を使用する場合は、ヘッダファイルをインクルードしてください。

キャッシュサポート関数の実体は、リロケータブルオブジェクトとして lib¥elf¥ディレクトリにあります。このリロケータブルオブジェクトは、カーネル側に組み込みます。

V.1.01 Release00 で提供されているキャッシュサポート関数を、表 7.1に示します。

表7.1 キャッシュサポート関数

対象キャッシュハードウェア仕様の概要	対象 CPU (主な搭載マイコン)	ヘッダファイル	リロケータブルオブジェクト
・命令・オペランド分離キャッシュ ・4 ウェイセットアソシアティブ ・仮想アドレスインデックス／物理アドレスタグ ・ラインサイズ：32 バイト	SH4AL-DSP,SH-4A (拡張機能なし) SH73180, SH7780	cache_sh4a.h	・cache_sh4a_big.rel (ビッグエンディアン用) ・cache_sh4a_little.rel (リトルエンディアン用)
・命令・オペランド分離キャッシュ ・4 ウェイセットアソシアティブ ・仮想アドレスインデックス／物理アドレスタグ ・ラインサイズ：32 バイト ・命令キャッシュウェイ予測機能あり	SH4AL-DSP,SH-4A (拡張機能あり) SH7343, SH7785	cache_shx2.h	・cache_shx2_big.rel (ビッグエンディアン用) ・cache_shx2_little.rel (リトルエンディアン用

7.2 注意事項

(1) キャッシュサポート関数は、特権モード(カーネルドメインに所属するプログラム)でのみ呼び出し可能です。ユーザモードから呼び出した場合は、通常はキャッシュサポート関数内で例外が発生します。

(2) キャッシュサポート関数は、使い方を誤るとキャッシュとメモリのコヒーレンシを確保できなくなるなど、その後のシステムの動作に影響を与える可能性があるため、十分注意してください。使用するマイコンのキャッシュの仕様と、本関数の振る舞いを十分理解した上で使用するようにしてください。

7.3 cache_sh4a.h の関数

cache_sh4a.h で提供している関数は、以下の通りです。

- sh4a_vini_cac( )：キャッシュ初期化
- sh4a_vclr_cac( )：キャッシュのクリア
- sh4a_vfls_cac( )：オペランドキャッシュのフラッシュ
- sh4a_vinv_cac( )：キャッシュの無効化

7.3.1 キャッシュの初期化(sh4a_vini_cac)

■C 言語 API

```
ER ercd = sh4a_vini_cac(ATR cacatr, UINT icsize, UINT ocsize);
```

■パラメータ

ATR	cacatr	キャッシュ属性
UINT	icsize	命令キャッシュのサイズ
UINT	ocsize	オペランドキャッシュのサイズ

■リターンパラメータ

ER	ercd	正常終了(E_OK)

■エラーコード

(エラーが返ることはありません)

■機能

キャッシュを初期化します。具体的には、指定された cacatr に基づいて、以下に示すようにプロセッサの CCR レジスタと RAMCR レジスタを設定します。

CCR, RAMCR の更新は、SR.BL=1 の状態で P2 領域に配置された命令によって行います。

cacatr には、以下の各項目の論理和を指定できますが、cacatr に指定された値のエラーチェックは一切行いません。

また、本関数は、cacatr の指定内容に関わらず、CCR.ICI, OCI ビットに 1 を書き込みます。即ち、本関数呼び出し以前のキャッシュの内容は全て破棄されます。

- TCAC_IC_ENABLE(H'00000100)
  これを指定すると命令キャッシュをEnable(CCR.ICE=1)にし、指定しないとDisable (CCR.ICE=0)にします。
- TCAC_OC_ENABLE(H'00000001)
  これを指定するとオペランドキャッシュをEnable(CCR.OCE=1)にし、指定しないとDisable (CCR.OCE=0)にします。
- TCAC_IC_2WAY(H'00800000)
  これを指定すると命令キャッシュを2WAY(RAMCR.IC2W=1)とし、指定しないと4WAY (RAMCR.IC2W=0)とします。
- TCAC_OC_2WAY(H'00400000)
  これを指定するとオペランドキャッシュを2WAY(RAMCR.OC2W=1)とし、指定しないと 4WAY (RAMCR.OC2W=0)とします。
- TCAC_P1_CB(H'00000004)
  これを指定するとP1領域への書き込みモードをコピーバックモード(CCR.CB=1)とし、指定しないとライトスルーモード(CCR.CB=0)とします。
- TCAC_P0_WT(H'00000002)
  これを指定するとP0/U0領域への書き込みモードをライトスルーモード(CCR.WT=1)とし、指定しないとコピーバックモード(CCR.WT=0)とします。

icsize および ocsize には、それぞれ使用するマイコンが搭載している命令キャッシュおよびオペランドキャッシュのサイズ(バイト数)を指定します。指定されたサイズの正当性については検査されません。

また、他のキャッシュサポート関数を使用する前に、必ず本関数を呼び出す必要があります。

7.3.2 キャッシュのクリア(sh4a_vclr_cac)

■C 言語 API

```
ER ercd = sh4a_vclr_cac(VP clradr1, VP clradr2, MODE mode);
```

■パラメータ

VP	clradr1	クリア先頭アドレス
VP	clradr2	クリア最終アドレス
MODE	mode	対象キャッシュ指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)clradr1＞clradr2 (2)mode が不正
E_OBJ	mode によって決まる対象キャッシュが Disable

■機能

キャッシュをクリアします。即ち、キャッシュ内容を無効化すると共に、オペランドキャッシュにメモリに書き戻していないデータがあれば、それをメモリに書き戻します。

対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)
-

また、クリアするアドレス範囲は、clradr1 と clradr2 によって決まります。clradr1 は 32 の倍数に切り捨て、clradr2 は 32 の倍数-1 に切り上げて扱います。

-

(1) アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが clradr1～clradr2 のエントリをクリアします。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)は、そのエントリがダーティ(メモリに書き出されていない)であれば、クリアする前にメモリへのコピーバックを行います。

本関数は、clradr1～clradr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBP 命令
- mode=TC_EXCLUDE_IC の場合：OCBP 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセット

します。

なお、clradr1, clradr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

clradr1=0, clradr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリをクリアします。この場合、本関数は以下のように処理します。

(a) modeがTC_FULLまたはTC_EXCLUDE_OCの場合は、CCR.ICI=1を設定することで、命令キャッシュの全エントリを無効化します。CCRの更新は、SR.BL=1の状態でP2領域に配置された命令によって行います。

(b) (a)の後、modeがTC_FULLまたはTC_EXCLUDE_ICの場合は、オペランドキャッシュのメモリ割付キャッシュの全エントリについて、V=0, U=0を書き込みます。この時、ダーティ(U=1)であったエントリの内容はメモリにコピーバックされます。この処理を実行するときは、SRレジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。SR.BL=1の状態で呼び出してもかまいません。

7.3.3 オペランドキャッシュのフラッシュ(sh4a_vfls_cac)

■C 言語 API

```
ER ercd = sh4a_vfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	フラッシュ先頭アドレス
VP	flsadr2	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー(flsadr1＞flsadr2)
E_OBJ	オペランドキャッシュが Disable

■機能

オペランドキャッシュをフラッシュします。すなわち、メモリに書き出されていない内容をメモリに書き出します(コピーバック)。

フラッシュするアドレス範囲は、flsadr1 と flsadr2 によって決まります。flsadr1 は 32 の倍数に切り捨て、flsadr2 は 32 の倍数-1 に切り上げて扱います。

(1) アドレス範囲を指定

論理アドレスが flsadr1～flsadr2 に対応するオペランドキャッシュエントリをフラッシュします。すなわち、該当するオペランドキャッシュエントリがメモリに書き出されていない場合、メモリに書き出します。

本関数は、flsadr1～flsadr2 に対して、OCBWB 命令を繰り返して発行します。この処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセットします。

なお、flsadr1, flsadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

flsadr1=0, flsadr2=H'ffffffff を指定すると、オペランドキャッシュの全エントリをフラッシュします。この場合、本サービスコールは以下のように処理します。

- オペランドキャッシュのメモリ割付キャッシュの全エントリについて、それを読み出し、有効(V=1)なエントリについて V=1, U=0 を書き込みます。この時、ダーティ(U=1)であったエントリの内容はメモリにコピーバックされます。この読み出しと書き込み処理は、一時的に SR.BL=1 の状態で行います。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。SR.BL=1 の状態で呼び出してもかまいません。

7.3.4 キャッシュの無効化(sh4a_vinv_cac)

■C 言語 API

```
ER ercd = sh4a_vinv_cac(VP invadr1, VP invadr2, MODE mode);
```

■パラメータ

VP	invadr1	無効化する先頭アドレス
VP	invadr2	無効化する最終アドレス
MODE	mode	対象キャッシュ指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)invadr1＞invadr2 (2)mode が不正
E_OBJ	mode によって決まるキャッシュが Disable

■機能

キャッシュを無効化します。

対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)
-

また、無効化するアドレス範囲は、invadr1 と invadr2 によって決まります。invadr1 は 32 の倍数に切り捨て、invadr2 は 32 の倍数-1 に切り上げて扱います。

-

(1) アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが invadr1～invadr2 のエントリを無効化します。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)で、そのエントリがダーティ(メモリに書き出されていない)であっても、メモリへのコピーバックは行いません。即ち、そのデータは消失することになります。

本関数は、invadr1～invadr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBI 命令
- mode=TC_EXCLUDE_IC の場合：OCBI 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセットします。

invadr1, invadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

invadr1=0, invadr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリを無効化します。この場合、本関数は mode に応じて CCR レジスタの以下のビットを操作します。CCR の更新は、SR.BL=1 の状態で P2 領域に配置された命令によって行います。

- mode=TC_FULL の場合：CCR.ICI と CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_IC の場合：CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_OC の場合：CCR.ICI に 1 を設定します。

7.4 cache_shx2.h の関数

cache_shx2.h で提供している関数は、以下の通りです。

- shx2_vini_cac( )：キャッシュ初期化
- shx2_vini_cac( )：キャッシュのクリア
- shx2_vini_cac( )：オペランドキャッシュのフラッシュ
- shx2_vini_cac( )：キャッシュの無効化

7.4.1 キャッシュの初期化(shx2_vini_cac)

■C 言語 API

```
ER ercd = shx2_vini_cac(ATR cacatr, UINT icsize, UINT ocsize);
```

■パラメータ

ATR	cacatr	キャッシュ属性
UINT	icsize	命令キャッシュのサイズ
UINT	ocsize	オペランドキャッシュのサイズ

■リターンパラメータ

ER	ercd	正常終了(E_OK)

■エラーコード

(エラーが返ることはありません)

■機能

キャッシュを初期化します。具体的には、指定された cacatr に基づいて、以下に示すようにプロセッサの CCR レジスタと RAMCR レジスタを設定します。

CCR, RAMCR の更新は、SR.BL=1 の状態で P2 領域に配置された命令によって行います。

cacatr には、以下の各項目の論理和を指定できますが、cacatr に指定された値のエラーチェックは一切行いません。

また、本関数は、cacatr の指定内容に関わらず、CCR.ICI, OCI ビットに 1 を書き込みます。即ち、本関数呼び出し以前のキャッシュの内容は全て破棄されます。

- TCAC_IC_ENABLE(H'00000100)
  これを指定すると命令キャッシュをEnable(CCR.ICE=1)にし、指定しないとDisable (CCR.ICE=0)にします。
- TCAC_OC_ENABLE(H'00000001)
  これを指定するとオペランドキャッシュをEnable(CCR.OCE=1)にし、指定しないとDisable (CCR.OCE=0)にします。
- TCAC_IC_2WAY(H'00800000)
  これを指定すると命令キャッシュを2WAY(RAMCR.IC2W=1)とし、指定しないと4WAY (RAMCR.IC2W=0)とします。
- TCAC_OC_2WAY(H'00400000)
  これを指定するとオペランドキャッシュを2WAY(RAMCR.OC2W=1)とし、指定しないと 4WAY (RAMCR.OC2W=0)とします。
- TCAC_P1_CB(H'00000004)
  これを指定するとP1領域への書き込みモードをコピーバックモード(CCR.CB=1)とし、指定しないとライトスルーモード(CCR.CB=0)とします。
- TCAC_P0_WT(H'00000002)
  これを指定するとP0/U0領域への書き込みモードをライトスルーモード(CCR.WT=1)とし、指定しないとコピーバックモード(CCR.WT=0)とします。
- TCAC_IC_WPD(H'00200000)
  これを指定すると命令キャッシュのウェイ予測を行わない(CCR.ICWPD=1)とし、指定しないとウェイ予測を行う(CCR.ICWPD=0)とします。

- TCAC_L2_ENABLE(H'00010000)
  これを指定すると2次キャッシュをEnable(RAMCR.L2E=1)とし、指定しないとDisable(RAMCR.L2E=0)とします。
- TCAC_L2_FC(H'00020000)
  これを指定すると2次キャッシュ強制コヒーレンシモード(RAMCR.L2FC=1)とし、指定しないと強制コヒーレンシモード(RAMCR.L2FC=0)でないとします。
-

2次キャッシュを搭載していないマイコンを使用する場合は、TCAC_L2_ENABLE, TCAC_L2_FCを指定してはなりません。

icsize および ocsize には、それぞれ使用するマイコンが搭載している命令キャッシュおよびオペランドキャッシュのサイズ(バイト数)を指定します。指定されたサイズの正当性については検査されません。

また、他のキャッシュサポート関数を使用する前に、必ず本関数を呼び出す必要があります。

7.4.2 キャッシュのクリア(shx2_vclr_cac)

■C 言語 API

```
ER ercd = shx2_vclr_cac(VP clradr1, VP clradr2, MODE mode);
```

■パラメータ

VP	clradr1	クリア先頭アドレス
VP	clradr2	クリア最終アドレス
MODE	mode	対象キャッシュ指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)clradr1＞clradr2 (2)mode が不正
E_OBJ	mode によって決まる対象キャッシュが Disable

■機能

キャッシュをクリアします。即ち、キャッシュ内容を無効化すると共に、オペランドキャッシュにメモリに書き戻していないデータがあれば、それをメモリに書き戻します。

対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)
-

また、クリアするアドレス範囲は、clradr1 と clradr2 によって決まります。clradr1 は 32 の倍数に切り捨て、clradr2 は 32 の倍数-1 に切り上げて扱います。

-

(1) アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが clradr1～clradr2 のエントリをクリアします。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)は、そのエントリがダーティ(メモリに書き出されていない)であれば、クリアする前にメモリへのコピーバックを行います。

本関数は、clradr1～clradr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBP 命令
- mode=TC_EXCLUDE_IC の場合：OCBP 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセット

します。

なお、clradr1, clradr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

clradr1=0, clradr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリをクリアします。この場合、本関数は以下のように処理します。

(a) modeがTC_FULLまたはTC_EXCLUDE_OCの場合は、CCR.ICI=1を設定することで、命令キャッシュの全エントリを無効化します。CCRの更新は、SR.BL=1の状態でP2領域に配置された命令によって行います。

(b) (a)の後、modeがTC_FULLまたはTC_EXCLUDE_ICの場合は、オペランドキャッシュのメモリ割付キャッシュの全エントリについてOCBP命令を実行することで、ダーティ(U=1)なエントリの内容をメモリにライトバックさせるとともに無効化(V=0)します。この処理を実行するときは、SRレジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。SR.BL=1の状態で呼び出してもかまいません。

7.4.3 オペランドキャッシュのフラッシュ(shx2_vfls_cac)

■C 言語 API

```
ER ercd = shx2_vfls_cac(VP flsadr1, VP flsadr2);
```

■パラメータ

VP	flsadr1	フラッシュ先頭アドレス
VP	flsadr2	フラッシュ最終アドレス

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー(flsadr1＞flsadr2)
E_OBJ	オペランドキャッシュが Disable

■機能

オペランドキャッシュをフラッシュします。すなわち、メモリに書き出されていない内容をメモリに書き出します(コピーバック)。

フラッシュするアドレス範囲は、flsadr1 と flsadr2 によって決まります。flsadr1 は 32 の倍数に切り捨て、flsadr2 は 32 の倍数-1 に切り上げて扱います。

(1) アドレス範囲を指定

論理アドレスが flsadr1～flsadr2 に対応するオペランドキャッシュエントリをフラッシュします。すなわち、該当するオペランドキャッシュエントリがメモリに書き出されていない場合、メモリに書き出します。

本関数は、flsadr1～flsadr2 に対して、OCBWB 命令を繰り返して発行します。この処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセットします。

なお、flsadr1, flsadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

flsadr1=0, flsadr2=H'ffffffff を指定すると、オペランドキャッシュの全エントリをフラッシュします。この場合、本サービスコールは以下のように処理します。

- オペランドキャッシュのメモリ割付キャッシュの全エントリについて、それを読み出し、有効(V=1)なエントリについて V=1, U=0 を書き込みます。この時、ダーティ(U=1)であったエントリの内容はメモリにコピーバックされます。この読み出しと書き込み処理は、一時的に SR.BL=1 の状態で行います。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。SR.BL=1 の状態で呼び出してもかまいません。

7.4.4 キャッシュの無効化(shx2_vinv_cac)

■C 言語 API

```
ER ercd = shx2_vinv_cac(VP invadr1, VP invadr2, MODE mode);
```

■パラメータ

VP	invadr1	無効化する先頭アドレス
VP	invadr2	無効化する最終アドレス
MODE	mode	対象キャッシュ指定

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)invadr1＞invadr2 (2)mode が不正
E_OBJ	mode によって決まるキャッシュが Disable

■機能

キャッシュを無効化します。

対象のキャッシュは、mode によって決まります。mode には、以下のいずれかを指定できます。

- TC_FULL(H'00000000)：命令キャッシュ・オペランドキャッシュの両方を対象とする。
- TC_EXCLUDE_IC(H'00000001)：命令キャッシュを対象外とする(オペランドキャッシュのみ)
- TC_EXCLUDE_OC(H'00000002)：オペランドキャッシュを対象外とする(命令キャッシュのみ)
-

また、無効化するアドレス範囲は、invadr1 と invadr2 によって決まります。invadr1 は 32 の倍数に切り捨て、invadr2 は 32 の倍数-1 に切り上げて扱います。

-

(1) アドレス範囲を指定

mode で決まるキャッシュに対し、論理アドレスが invadr1～invadr2 のエントリを無効化します。オペランドキャッシュが対象に含まれる場合(mode が TC_FULL または TC_EXCLUDE_IC の場合)で、そのエントリがダーティ(メモリに書き出されていない)であっても、メモリへのコピーバックは行いません。即ち、そのデータは消失することになります。

本関数は、invadr1～invadr2 に対して、以下の命令を繰り返して実行します。

- mode=TC_FULL の場合：ICBI 命令と OCBI 命令
- mode=TC_EXCLUDE_IC の場合：OCBI 命令
- mode=TC_EXCLUDE_OC の場合：ICBI 命令

これらの処理を実行するときは、SR レジスタの値は呼び出し時と同じです。本関数の処理中に割込みを受け付けたくない場合は、割込みをマスクした状態で呼び出してください。ただし、MMU が Enable の場合で MMU 対象領域を指定する場合は、SR.BL=1 の状態で呼び出してはなりません。SR.BL=1 の状態では、上記命令で TLB 関連例外が発生する可能性があり、この場合 CPU がリセットします。

invadr1, invadr2 については、エラーコード欄に記載した基本的なエラーチェックしか行いません。例えば、以下のようなアドレスが含まれないようにしてください。

- P2, P3, P4 領域
- 対応する物理アドレスが制御レジスタ領域
- 対応する物理アドレスが X/Y メモリ
- P0/U0 領域で、メモリオブジェクトでないアドレス

(2) 全エントリを対象とする

invadr1=0, invadr2=H'ffffffff を指定すると、mode で決まる対象キャッシュの全エントリを無効化します。この場合、本関数は mode に応じて CCR レジスタの以下のビットを操作します。CCR の更新は、SR.BL=1 の状態で P2 領域に配置された命令によって行います。

- mode=TC_FULL の場合：CCR.ICI と CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_IC の場合：CCR.OCI に 1 を設定します。
- mode=TC_EXCLUDE_OC の場合：CCR.ICI に 1 を設定します。

8. アプリケーション作成手順

8.1 タスク

8.1.1 記述方法

タスクは、図 8.1に示すように C 言語関数として記述します。タスクを終了する場合は、ext_tsk または exd_tsk サービスコールを用いて終了してください。ext_tsk, exd_tsk サービスコールを発行せずにタスク関数からリターンした場合は、ext_tsk サービスコールを発行した場合と同じ動作となります。

```
#include "kernel.h"
#pragma noregsave(Task)          ←(1)
void Task(VP_INT exinf)          ←(2)
{
    /*処理 */
    ext_tsk();
}
```

図8.1　タスクの記述例

図の解説

(1)　タスク関数からリターンする場合を除き、#pragma noregsaveを指定することで、スタック使用量を抑止することができます。

(2)　exinfには、sta_tskで起動された場合はsta_tskで指定したstacd、act_tskおよびタスク生成時のTA_ACT属性によって起動された場合は、タスクの拡張情報が渡されます。

タスク関数は、図 8.2に示すように無限ループとして記述することもできます。

```
#include "kernel.h"
#pragma noregsave(Task)          ←(1)
void Task(VP_INT exinf)
{
    for(;;) {
        /*処理 */
    }
}
```

図8.2　無限ループタスクの記述例

図の解説

(1)　#pragma noregsaveを指定することで、スタック使用量を抑止することができます。

8.1.2 レジスタ使用規約

表 8.1に、タスクのレジスタ使用規約と初期値を示します。

表8.1 タスクのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	ー	タスクのアドレス
2	SR	○	表 8.2参照
3	R0～R3	◎	不定
4	R4	◎	TA_ACT 属性または act_tsk で起動された場合はタスク生成時に指定した exinf、sta_tsk で起動された場合は sta_tsk で指定された stacd
5	R5	◎	不定
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	◎	不定
9	R15	○	タスクのスタック領域の最終アドレス
10	PR	○	カーネル内のタスク終了処理のアドレス
11	[DSP] DSR	◎*2	0
12	[DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	◎*2	不定
13	[FPU] FPSCR	◎*3	TA_COP1 属性の指定がある場合はタスク生成時に指定した inifpscr、その他の場合は不定
14	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0	◎*3	不定
15	[FPU] FPR0_BANK1～FPR15_BANK1	◎*4	不定

【注】 *1 "◎"は保証せずに使用可能、"○"はタスク関数からリターンするときには起動時の値に戻す必要があります。

*2 TA_COP0 属性の指定がある場合のみ使用可能。

*3 TA_COP1 属性の指定がある場合のみ使用可能。

*4 TA_COP1|TA_COP2 属性の指定がある場合のみ使用可能。

表8.2 タスク起動時の SR

TA_COPn 属性	所属ドメイン種別	起動時の SR(下記以外のビットは不定です)					
		MD	RB	BL	DSP	FD	IMASK
指定無し	カーネルドメイン	1	0	0	0	1	0
	ユーザドメイン	0					
TA_COP0	カーネルドメイン	1			1		
	ユーザドメイン	0					
TA_COP1(|TA_COP2)	カーネルドメイン	1			0	0	
	ユーザドメイン	0					

8.2 タスク例外処理ルーチン

8.2.1 記述方法

タスク例外処理ルーチンは、図 8.3に示すように C 言語関数として記述します。

```
#include "kernel.h"
void Texrtn(TEXPTN texptn,VP_INT exinf)      ←(1)
{
    /*処理 */
}
```

図8.3 タスク例外処理ルーチンの記述例

図の解説

(1) texptnには、タスク例外要因パターン、exinfにはタスクの拡張情報が渡されます。

8.2.2 レジスタ使用規約

表 8.3に、タスク例外処理ルーチンのレジスタ使用規約と初期値を示します。

表8.3 タスク例外処理ルーチンのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	−	タスク例外処理ルーチンのアドレス
2	SR	○	表 8.4参照
3	R0～R3	◎	不定
4	R4	◎	例外要因パターン
5	R5	◎	タスクの拡張情報
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	◎	不定
9	R15	○	タスクのスタック領域
10	PR	○	カーネル内のタスク例外処理ルーチン終了処理のアドレス
11	[DSP] DSR	◎*2	0
12	[DSP] RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	◎*2	不定
13	[FPU] FPSCR	◎*3	TA_COP1 属性の指定がある場合はタスク例外処理ルーチン定義時に指定した inifpscr、その他の場合は不定
14	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0	◎*3	不定
15	[FPU] FPR0_BANK1～FPR15_BANK1	◎*4	不定

【注】 *1 "◎"は保証せずに使用可能、"○"はタスク例外処理ルーチン関数からリターンするときには起動時の値に戻す必要があります。

*2 TA_COP0 属性の指定がある場合のみ使用可能。

*3 TA_COP1 属性の指定がある場合のみ使用可能。

*4 TA_COP1|TA_COP2 属性の指定がある場合のみ使用可能。

表8.4 タスク例外処理ルーチン起動時の SR

TA_COPn 属性	所属ドメイン種別	起動時の SR(下記以外のビットは不定です)					
		MD	RB	BL	DSP	FD	IMASK
指定無し	カーネルドメイン	1	0	0	0	1	0
	ユーザドメイン	0					
TA_COP0	カーネルドメイン	1			1		
	ユーザドメイン	0					
TA_COP1(|TA_COP2)	カーネルドメイン	1			0	0	
	ユーザドメイン	0					

8.3 拡張サービスコールルーチン、トラップルーチン

8.3.1 記述方法

(1) 拡張サービスコールルーチン

拡張サービスコールルーチンは、cal_svc サービスコールによって呼び出されます。
拡張サービスコールルーチンは、図 8.4に示すように C 言語関数として記述します。

```
#include "kernel.h"
ER_UINT Svcrtn(VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4)   ←(1)
{
    /* 処理 */
    return E_OK;
}                                                                    ←(2)
```

図8.4 拡張サービスコールルーチンの記述例(1)

図の解説

(1) 拡張サービスコールルーチンには、cal_svcで指定したVP_INT型の4つのパラメータを受け取ります。

(2) ER_UINT型の値を返します。リターン値は、cal_svcのリターン値となります。

受け取るパラメータが少ない場合は、図 8.5のように記述することもできます。

```
#include "kernel.h"
ER_UINT Svcrtn(VP_INT par1, VP_INT par2)                             ←(1)
{
    /* 処理 */
    return E_OK;
}
```

図8.5 拡張サービスコールルーチンの記述例(2)

図の解説

(1) cal_svcで指定したVW型のパラメータのpar1, par2だけを受け取ります。

(2) トラップルーチン

トラップルーチンは、TRAPA 命令が実行されたときに呼び出されます。
トラップルーチンは、図 8.6に示すように C 言語関数として記述します。

```
#include "kernel.h"
void Trprtn(VT_TRAP *pk_trap)                                    ←(1)
{
    /* 処理 */
    return;
}
```

図8.6　トラップルーチンの記述例

図の解説

(1)　pk_trapは、TRAPA命令実行時点のレジスタ情報が格納されたパケットアドレスです。

図 8.7に、VT_TRAP の定義を示します。
このパケットの内容は、TRAPA 命令実行時点の各レジスタの値です。
カーネルは、トラップルーチン終了時にこのパケットの内容を該当レジスタにリストアします。
なお、このパケットの ctxid, ssr, r15 は書き換えてはなりません。書き換えた場合の動作は保証されません。

```
typedef struct  {
    UW      r0;         /* R0_BANK0 レジスタ */
    UW      r1;         /* R1_BANK0 レジスタ */
    UW      r2;         /* R2_BANK0 レジスタ */
    UW      r3;         /* R3_BANK0 レジスタ */
    UW      r4;         /* R4_BANK0 レジスタ */
    UW      r5;         /* R5_BANK0 レジスタ */
    UW      r6;         /* R6_BANK0 レジスタ */
    UW      r7;         /* R7_BANK0 レジスタ */
    UW      pr;         /* PR レジスタ */
    UW      spc;        /* SPC レジスタ */
    UW      ssr;        /* SSR レジスタ */
    UW      ctxid;      /* ctxid 情報(カーネル内部情報) */
    UW      r15;        /* R15 レジスタ */
} VT_REG0;

typedef VT_REG0 VT_TRAP;
```

図8.7　VT_TRAP 型

8.3.2 レジスタ使用規約

表 8.5に、拡張サービスコールルーチン、トラップルーチンのレジスタ使用規約と初期値を示します。拡張サービスコールルーチンのリターン値は R0 レジスタに設定します。

表8.5 拡張サービスコールルーチン、トラップルーチンのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	−	ルーチンのアドレス
2	SR	○	表 8.6参照
3	R0～R3	◎	不定
4	R4	◎	拡張サービスコールルーチン：par1 トラップルーチン：pk_trap
5	R5	◎	拡張サービスコールルーチン：par2 トラップルーチン：不定
6	R6	◎	拡張サービスコールルーチン：par3 トラップルーチン：不定
7	R7	◎	拡張サービスコールルーチン：par4 トラップルーチン：不定
8	R8～R14, MACH, MACL, GBR	○	不定
9	R15	○	(1)タスクコンテキストから呼び出した場合 呼び出し元タスクの特権スタック領域 (2)非タスクコンテキストから呼び出した場合 呼び出し前のスタック(非タスクスタック)
10	PR	○	カーネル内の復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	◎*2	不定
12	[FPU] FPSCR	◎*3	TA_COP1 属性の指定がある場合は、ルーチン定義時に指定した inifpscr、その他の場合は不定
13	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0	◎*3	不定
14	[FPU] FPR0_BANK1～FPR15_BANK1	◎*4	不定

【注】 *1 "◎"は保証せずに使用可能、"○"はルーチン関数からリターンするときには起動時の値に戻す必要があります。

*2 TA_COP0 属性の指定がある場合のみ使用可能。

*3 TA_COP1 属性の指定がある場合のみ使用可能。

*4 TA_COP1|TA_COP2 属性の指定がある場合のみ使用可能。

表8.6 拡張サービスコールルーチン、トラップルーチンの起動時の SR

TA_COPn 属性	起動時の SR(下記以外のビットは不定です)					
	MD	RB	BL	DSP	FD	IMASK
指定無し	1	0	0	0	1	拡張サービスコール呼び出し前、または TRAPA 命令実行前と同じ
TA_COP0				1	1	
TA_COP1(|TA_COP2)				0	0	

8.4 割込みハンドラ

8.4.1 記述方法

割込みハンドラは、図 8.8のように通常の C 言語関数として記述します。

```
#include "kernel.h"
void IntHandler(void)
{
    /* 処理 */
}
```

図8.8 割込みハンドラの記述例

8.4.2 レジスタ使用規約

表 8.7に、割込みハンドラのレジスタ使用規約と初期値を示します。

表8.7 割込みハンドラのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	−	割込みハンドラのアドレス
2	SR	○	後述の説明を参照してください
3	R0～R3	◎	不定
4	R4	◎	不定
5	R5	◎	不定
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	○	不定
9	R15	○	非タスクスタック
10	PR	○	カーネル内の割込み復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		割込み発生前と同じ
12	[FPU] FPSCR		割込み発生前と同じ
13	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0		割込み発生前と同じ
14	[FPU] FPR0_BANK1～FPR15_BANK1		割込み発生前と同じ

【注】 *1 "◎"は保証せずに使用可能、"○"はハンドラ関数からリターンするときには起動時の値に戻す必要があります。また、無印は使用(アクセス)してはなりません。

割込みハンドラ起動時の SR レジスタの各ビットは、以下のようになります。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：割込みハンドラ定義時に指定した inhsr に従います
- DSP ビット(SH4AL-DSP)：0
- FPU ディスエーブル(FD)ビット(SH-4A)：1
- 割込みマスクレベル(IMASK)ビット：カーネル起動時の CPUOPM レジスタの INTMU ビットが 0 の場合は inhsr の指定に従います。INTMU ビットが 1 の場合は、発生した割込みレベル値になります。
- その他のビット：不定

割込みハンドラでは、SR.BL,IMASK ビットを自割込みレベルが受理されるような設定にしてはなりません。

8.4.3 DSP, FPU について

割込みハンドラでは、DSP, FPU を利用した演算を行ってはなりません。ただし、割込みハンドラから呼び出した拡張サービスコールルーチンおよびトラップルーチンでは、それらのルーチンに指定された属性(TA_COP0, TA_COP1, TA_COP2)に対応するコプロセッサを使用できます。

8.4.4 NMI に関する注意事項

(1) NMI割込みハンドラを定義する際に指定する「ハンドラ起動時のSR」のBLビットを1にしてください。そして、NMI割込みハンドラではSR.BLをクリアしないで下さい。

(2) 割込みコントローラで、SR.BL=1の時にNMIを受理するかどうかを設定することができますが、通常は「受理しない」に設定してください。
「受理する」にした場合は、NMI割込みハンドラが使用するスタックサイズが増えます。また、メモリオブジェクト保護機能使用時は、NMI割込みハンドラでMMU対象領域をアクセスしないようにしてください。MMU対象領域をアクセスしてTLBミスが発生すると、CPUはリセットします。

■関連ページ
- 「12.4.3 NMI 割込みハンドラの使用サイズ」
- 「5. 論理アドレス空間の扱い」
- 「4.18.3(2) SR レジスタの BL ビットを変更する」

8.5 割込み・例外フックルーチン

8.5.1 概要

CPU 例外や割込みが発生したとき、プロセッサは以下のアドレスに制御を移します。

- 一般例外：VBR+H'100
- TLB ミス例外：VBR+H'400
- 割込み：VBR+H'600

カーネルは、vsta_knl によって上記のアドレスがカーネル内部の処理ルーチンを指すように VBR レジスタを初期化します。このルーチンでは、例外や割込みの要因を解析し、対応するハンドラを起動します。

しかし、デバッグなどの目的で、例外や割込みが発生したときに、ハンドラを起動する前にユーザ任意の処理を行いたい場合があります。

コンフィギュレータで CFG_INTHOOK を選択すると、CPU 例外・割込みが発生したときにフックルーチンが呼び出されるようになります。

フックルーチンは、上記の 3 種類に対して、シンボル名が以下のように定められています。

- 一般例外用：__kernel_hook_exp (アセンブリ言語レベルでの表記)
- TLB ミス例外用：__kernel_hook_tlb (アセンブリ言語レベルでの表記)
- 割込み用：__kernel_hook_int (アセンブリ言語レベルでの表記)

なお、サンプルのフックルーチンが samples¥sysapp¥inthook.src にあるので、参考にしてください。

8.5.2 記述方法

フックルーチンは、図 8.9に示すようにアセンブリ言語で記述する必要があります。

```
     .section  PSCP_hiknl, code, align=4  ; セクション名は任意です
     .export  __kernel_hook_exp
__kernel_hook_exp:          ; 一般例外用のフックルーチン先頭のシンボル
       ; TRAPA を含む一般例外(VBR+H'100)の処理
       ; この位置にコードを記述してください。
     rts           ; RTS 命令により、カーネル標準の一般例外処理に復帰します
     nop
     .pool

     .export  __kernel_hook_tlb
__kernel_hook_tlb:          ; TLB ミス例外用のフックルーチン先頭のシンボル
       ; TLB ミス例外(VBR+H'400)の処理
       ; この位置にコードを記述してください。
     rts           ; RTS 命令により、カーネル標準の TLB 例外処理に復帰します
     nop
     .pool

     .export  __kernel_hook_int
__kernel_hook_int:          ;割込み用のフックルーチン先頭のシンボル
       ; 割込み(VBR+H'600)の処理
       ; この位置にコードを記述してください。
     rts           ; RTS 命令により、カーネル標準の割込み処理に復帰します
     nop
     .pool
     .end
```

図8.9　フックルーチンの記述例

8.5.3 レジスタ使用規約

表 8.8に、フックルーチンのレジスタ使用規約と初期値を示します。

表8.8 フックルーチンのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	―	フックルーチンのアドレス
2	SR	○	プロセッサの割込み・例外処理に従います。 特に ・MD ビット：1 ・RB ビット：1 ・BL ビット：1 であることに注意してください。 また、BL ビットを 0 にしてはなりません。
3	R0～R2_BANK1	◎	不定
4	R3～R7_BANK1	○	不定
5	R0～R7_BANK0	○	割込み・例外発生前と同じ
6	R8～R14, MACH, MACL, GBR	○	割込み・例外発生前と同じ
7	R15	○	割込み・例外発生前と同じ
8	PR	○	カーネル内の割込み・例外処理のアドレス
9	SPC, SSR	○	プロセッサの例外処理によって設定された値
10	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		割込み・例外発生前と同じ
11	[FPU] FPSCR		割込み・例外発生前と同じ
12	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0		割込み・例外発生前と同じ
13	[FPU] FPR0_BANK1～FPR15_BANK1		割込み・例外発生前と同じ

【注】 *1 "◎"は保証せずに使用可能、"○"はルーチン関数からリターンするときには起動時の値に戻す必要があります。また、無印は使用(アクセス)してはなりません。

8.5.4 注意事項

(1) フックルーチンで使用(破壊)可能なレジスタは、R0_BANK1～R2_BANK1のみです。
(2) 起動時は、SR.BL=1, SR.RB=1の状態です。SR.BL=0にしてはなりません。
(3) SR.BL=1の時にCPU例外が発生すると、CPUはリセットするため、CPU例外を発生させないようにしてください。
メモリオブジェクト保護機能を組み込んでいる場合は、同じ理由からTLB関連の例外も発生させないようにしなければなりません。即ち、フックルーチンはMMU対象領域をアクセスしてはなりません。
(4) フックルーチンは、SR.BL=1のまま実行するため、再入することはありません。
(5) スタックを使用する場合は、MMU非対象領域にスタック領域をあらかじめ確保して、切り替えて使用してください。

8.6 タイムイベントハンドラ

8.6.1 記述方法

タイムイベントハンドラは、C 言語関数として記述します。図 8.10に周期ハンドラ、アラームハンドラ、図 8.11にオーバーランハンドラの記述例を示します。

```
#include "kernel.h"
void Handler(VP_INT exinf)                    ←(1)
{
    /* 処理 */
}
```

図8.10　周期ハンドラ、アラームハンドラの記述例

図の解説

(1)　exinfには、ハンドラの拡張情報が渡されます。

```
#include "kernel.h"
void Ovrhdr(ID tskid, VP_INT exinf)           ←(1)
{
    /* 処理 */
}
```

図8.11　オーバーランハンドラの記述例

図の解説

(1)　tskidには対象タスクID、exinfにはそのタスクの拡張情報が渡されます。

8.6.2 レジスタ使用規約

表 8.9に、タイムイベントハンドラのレジスタ使用規約と初期値を示します。

表8.9 タイムイベントハンドラのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	−	ハンドラのアドレス
2	SR	○	後述の説明を参照してください
3	R0～R3	◎	不定
4	R4	◎	周期ハンドラ、アラームハンドラ：ハンドラの拡張情報 オーバーランハンドラ：対象のタスク ID
5	R5	◎	周期ハンドラ、アラームハンドラ：不定 オーバーランハンドラ：対照タスクの拡張情報
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	○	不定
9	R15	○	非タスクスタック
10	PR	○	カーネル内の復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		不定
12	[FPU] FPSCR		不定
13	[FPU] FPUL		不定
14	[FPU] FPR0_BANK0～ FPR15_BANK0		不定
15	[FPU] FPR0_BANK1～ FPR15_BANK1		不定

【注】 *1 **"◎"は保証せずに使用可能、"○"はハンドラ関数からリターンするときにはルーチン起動時の値に戻す必要があります。また、無印は使用(アクセス)してはなりません。**

タイムイベントハンドラ起動時の SR レジスタの各ビットは、以下のようになります。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：常に 0
- DSP ビット(SH4AL-DSP)：常に 0
- FPU ディスエーブル(FD)ビット(SH-4A)：常に 1
- 割込みマスクレベル(IMASK)ビット：CFG_KNLMSKLVL
- その他のビット：不定

タイムイベントハンドラでは、SR.BL,IMASK を起動時点でマスクされている割込みが受理されるように変更してはなりません。

8.6.3 DSP, FPU について

タイムイベントハンドラでは、DSP, FPU を利用した演算を行ってはなりません。ただし、タイムイベントハンドラから呼び出した拡張サービスコールルーチンおよびトラップルーチンでは、それらのルーチンに指定された属性(TA_COP0, TA_COP1, TA_COP2)に対応するコプロセッサを使用できます。

8.7 初期化ルーチン

8.7.1 記述方法

コンフィギュレータの[初期化ルーチン]ページで定義した初期化ルーチンは、カーネル起動時にマルチタスク環境に移行する直前に実行されます。

■関連ページ 「6.22.12 カーネルの起動(vsta_knl, ivsta_knl)」

初期化ルーチンは、図 8.12に示すように C 言語関数として記述します。

```
#include "kernel.h"
void InitRoutine(VP_INT exinf)                ←(1)
{
    /* 処理 */
}
```

図8.12 初期化ルーチンの記述例

図の解説

(1) exinfには、初期化ルーチンの拡張情報が渡されます。

8.7.2 レジスタ使用規約

表 8.10に、初期化ルーチンのレジスタ使用規約と初期値を示します。

表8.10 初期化ルーチンのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	−	ルーチンのアドレス
2	SR	○	後述の説明を参照してください
3	R0～R3	◎	不定
4	R4	◎	初期化ルーチンの拡張情報
5	R5	◎	不定
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	○	不定
9	R15	○	非タスクスタック
10	PR	○	カーネル内の復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1		不定
12	[FPU] FPSCR		不定
13	[FPU] FPUL		不定
14	[FPU] FPR0_BANK0～FPR15_BANK0		不定
15	[FPU] FPR0_BANK1～FPR15_BANK1		不定

【注】 *1 "◎"は保証せずに使用可能、"○"はルーチン関数からリターンするときにはルーチン起動時の値に戻す必要があります。また、無印は使用(アクセス)してはなりません。

タイムイベントハンドラ起動時の SR レジスタの各ビットは、以下のようになります。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：常に 0
- DSP ビット(SH4AL-DSP)：常に 0
- FPU ディスエーブル(FD)ビット(SH-4A)：常に 1
- 割込みマスクレベル(IMASK)ビット：15
- その他のビット：不定

初期化ルーチンでは、SR.BL,IMASK を起動時点でマスクされている割込みが受理されるように変更してはなりません。

8.7.3 DSP, FPU について

初期化ルーチンでは、DSP, FPU を利用した演算を行ってはなりません。ただし、初期化ルーチンから呼び出した拡張サービスコールルーチンおよびトラップルーチンでは、それらのルーチンに指定された属性(TA_COP0, TA_COP1, TA_COP2)に対応するコプロセッサを使用できます。

DSP 演算を行う場合は、事前に DSR レジスタを適切に初期化してください。どのような値に初期化すべきかは、使用するマイコンのハードウェアマニュアルを参照してください

8.8 CPU 例外ハンドラ

サンプルの CPU 例外ハンドラが samples¥sysapp¥exchdr.c にあるので、参考にしてください。

8.8.1 記述方法

CPU 例外ハンドラは、図 8.13のように C 言語関数として記述します。

```
#include "kernel.h"
void Exchdr(VT_EXC *pk_exc)            ←(1)
{
    /* 処理 */
}
```

図8.13 CPU 例外ハンドラの記述例

図の解説

(1) pk_excは、CPU例外発生時点のレジスタ情報が格納されたパケットアドレスです。

VT_EXC 型は、図 8.14に示すように VT_EXCINF, VT_REG0, VT_REG1 型から構成されています。図 8.15～図 8.17に、それぞれの定義を示します。

```
typedef struct {
    VT_EXCINF   vt_excinf;
    VT_REG1     vt_reg1;
    VT_REG0     vt_reg0;
} VT_EXC;
```

図8.14 VT_EXC 型

```
typedef struct {
    UW      syssts;     /* 例外発生前の各種システム状態(カーネル内部情報)  */
    ID      tskid;      /* 例外発生前に実行していたタスク ID。
                           タスクが存在しない場合は TSK_NONE(0) */
    ID      domid;      /* 例外発生前に実行していたタスクが所属するドメイン ID。
                           そのようなタスクが存在しない場合は TDOM_NONE(-2) */
    STAT    texstat;    /* tskid のタスクのタスク例外処理状態。
                           tskid=TSK_NONE(0)の場合は、無効(不定値) */
    UW      expevt;     /* EXPEVT レジスタ */
    UW      tra;        /* TRA レジスタ */
    UW      tea;        /* TEA レジスタ */
} VT_EXCINF;
```

図8.15 VT_EXCINF 型

```
typedef struct  {
    UW      r0;         /* R0_BANK0 レジスタ */
    UW      r1;         /* R1_BANK0 レジスタ */
    UW      r2;         /* R2_BANK0 レジスタ */
    UW      r3;         /* R3_BANK0 レジスタ */
    UW      r4;         /* R4_BANK0 レジスタ */
    UW      r5;         /* R5_BANK0 レジスタ */
    UW      r6;         /* R6_BANK0 レジスタ */
    UW      r7;         /* R7_BANK0 レジスタ */
    UW      pr;         /* PR レジスタ */
    UW      spc;        /* SPC レジスタ */
    UW      ssr;        /* SSR レジスタ */
    UW      ctxid;      /* ctxid 情報(カーネル内部情報) */
    UW      r15;        /* R15 レジスタ */
} VT_REG0;
```

図8.16　VT_REG0 型

```
typedef struct {
    UW      r8;         /* R8 レジスタ */
    UW      r9;         /* R9 レジスタ */
    UW      mach;       /* MACH レジスタ */
    UW      r10;        /* R10 レジスタ */
    UW      macl;       /* MACL レジスタ */
    UW      r11;        /* R11 レジスタ */
    UW      gbr;        /* GBR レジスタ */
    UW      r12;        /* R12 レジスタ */
    UW      r13;        /* R13 レジスタ */
    UW      r14;        /* R14 レジスタ */
} VT_REG1;
```

図8.17　VT_REG1 型

vt_reg0, vt_reg1 の内容は、CPU 例外ハンドラ終了時にカーネルによって該当レジスタにリストアされます。例外処理のためにレジスタを書き換えたい場合は、このパケットの内容を書き換えてください。但し、vt_reg0.ssr,ctxid,r15 は書き換えてはなりません。書き換えた場合の動作は保証されません。

なお、FPU レジスタと DSP レジスタは、このパケットには含まれていません。カーネルはこれらのレジスタの保存を行わずにハンドラを起動し、ハンドラ終了時もリストア処理は行いません。例外処理のためにレジスタを書き換えたい場合は、DSP/FPU 例外に限りハンドラでこれらのレジスタを直接書き換えてください。DSP/FPU に起因する例外でない場合は、DSP/FPU レジスタの操作自体できない場合があります。なお、これらのレジスタを書き換えるプログラムは、アセンブラで記述する必要があります。

8.8.2 CPU 例外ハンドラ専用マクロ

CPU 例外ハンドラ専用に、CPU 例外発生時点の各種状態を参照するための以下の C 言語マクロが用意されています。これらはいずれも、パラメータとして CPU 例外ハンドラに渡される pk_exc を指定します。なお、これらのマクロでは指定パラメータに関するエラー検出は行いません。

(1) CPU 例外発生時点のコンテキストの参照：VSNS_CTX マクロ

■C 言語 API

```
BOOL state = VSNS_CTX(VT_EXC *pk_exc);
```

■リターンパラメータ

```
BOOL        state   コンテキスト
```

■機能

リターン値は、sns_ctx サービスコールと同じ仕様です。

即ち、CPU 例外発生時点が非タスクコンテキストの場合に TRUE、タスクコンテキストの場合に FALSE を返します。

(2) CPU 例外時点に実行状態であったタスク ID の参照：VGET_TID マクロ

■C 言語 API

```
ID tskid = VGET_TID(VT_EXC *pk_exc);
```

■リターンパラメータ

```
ID          tskid   タスク ID
```

■機能

リターン値は、iget_tid サービスコールと同じ仕様です。

即ち、CPU 例外が発生した時点で実行状態であったタスクの ID を返します。具体的には、タスクコンテキストで発生した CPU 例外の場合はそのタスク ID を返し、非タスクコンテキストで発生した CPU 例外の場合は、非タスクコンテキストに遷移する前に実行していたタスク ID を返します。そのようなタスクが存在しない場合は、TSK_NONE(0)を返します。

非タスクコンテキストで CPU 例外が発生した場合であっても、非タスクコンテキストに遷移する前に実行していたタスクの ID を返すことに注意してください。CPU 例外が発生したコンテキストが非タスクコンテキストであるかどうかは、VSNS_CTX()で確認してください。

(3) CPU 例外時点に実行状態であったタスクが所属するドメイン ID の参照：VGET_DID マクロ

■C 言語 API

```
ID domid = VGET_DID(VT_EXC *pk_exc);
```

■リターンパラメータ

```
ID          domid    ドメイン ID
```

■機能

リターン値は、iget_did サービスコールと同じ仕様です。

即ち、CPU 例外が発生した時点で実行状態であったタスクが所属するドメイン ID を返します。具体的には、タスクコンテキストで発生した CPU 例外の場合はそのタスクが所属するドメイン ID を返し、非タスクコンテキストで発生した CPU 例外の場合は、非タスクコンテキストに遷移する前に実行していたタスクが所属するドメイン ID を返します。そのようなタスクが存在しない場合は、TDOM_NONE(-2)を返します。

非タスクコンテキストで CPU 例外が発生した場合であっても、非タスクコンテキストに遷移する前に実行していたタスクが所属するドメイン ID を返すことに注意してください。CPU 例外が発生したコンテキストが非タスクコンテキストであるかどうかは、VSNS_CTX()で確認してください。

(4) CPU 例外発生時点の CPU ロック状態の参照：VSNS_LOC マクロ

■C 言語 API

```
BOOL state = VSNS_LOC(VT_EXC *pk_exc);
```

■リターンパラメータ

```
BOOL        state    CPU ロック状態
```

■機能

リターン値は、sns_loc サービスコールと同じ仕様です。

即ち、CPU 例外発生時点が 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。

(5) CPU 例外発生時点のディスパッチ禁止状態の参照：VSNS_DSP マクロ

■C 言語 API

```
BOOL state = VSNS_DSP(VT_EXC *pk_exc);
```

■リターンパラメータ

```
BOOL        state    ディスパッチ禁止状態
```

■機能

リターン値は、sns_dsp サービスコールと同じ仕様です。

即ち、CPU 例外発生時点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。

(6) CPU 例外発生時点のディスパッチ保留状態の参照：VSNS_DPN マクロ

■C 言語 API

```
BOOL state = VSNS_DPN(VT_EXC *pk_exc);
```

■リターンパラメータ

```
BOOL        state   ディスパッチ保留状態
```

■機能

リターン値は、sns_dpn サービスコールと同じ仕様です。

即ち、CPU 例外発生時点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。

(7) CPU 例外時点に実行状態であったタスクのタスク例外禁止状態の参照：VSNS_TEX マクロ

■C 言語 API

```
BOOL state = VSNS_TEX(VT_EXC *pk_exc);
```

■リターンパラメータ

```
BOOL        state   タスク例外禁止状態
```

■機能

リターン値は、sns_ctx サービスコールと同じ仕様です。

即ち、VGET_TID で得られるタスクが TSK_NONE(0)でなく、かつそのタスクがタスク例外許可状態の場合に FALSE、それ以外の場合に TRUE を返します。

8.8.3 レジスタ使用規約

表 8.11に、CPU 例外ハンドラのレジスタ使用規約と初期値を示します。

表8.11 CPU 例外ハンドラのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	ー	ハンドラのアドレス
2	SR	○	後述の説明を参照してください
3	R0～R3	◎	不定
4	R4	◎	pk_exc
5	R5	◎	不定
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	◎	不定
9	R15	○	非タスクスタック
10	PR	○	カーネル内の CPU 例外復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○*2	CPU 例外発生前と同じ
12	[FPU] FPSCR	○*2	CPU 例外発生前と同じ
13	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0	○*2	CPU 例外発生前と同じ
15	[FPU] FPR0_BANK1～FPR15_BANK1	○*2	CPU 例外発生前と同じ

【注】 *1 "◎"は保証せずに使用可能、"○"はハンドラ関数からリターンするときには起動時の値に戻す必要があります。

*2 例外要因を解決したい場合のみ、該当するレジスタを変更してください。

CPU 例外ハンドラ起動時の SR レジスタの各ビットは、以下のようになります。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：CPU 例外ハンドラ定義時に指定した excsr に従います
- その他のビット：CPU 例外発生前と同じ

CPU 例外ハンドラでは、SR.BL,IMASK を起動時点でマスクされている割込みが受理されるように変更してはなりません。

8.8.4 DSP, FPU について

CPU 例外ハンドラでは、例外処理のためにレジスタを書き換える場合を除いて、DSP, FPU を利用した演算を行ってはなりません。

8.9 メモリアクセス違反ハンドラ

8.9.1 概要

メモリアクセス違反ハンドラは、メモリオブジェクト保護機能を使用し、MMU 対象領域に対する不正メモリアクセスが発生した時に起動されます。メモリオブジェクト保護機能を使用する場合、メモリアクセス違反ハンドラは、必ず作成してカーネルに組み込まなければなりません。

なお、サンプルのメモリアクセス違反ハンドラが samples¥sysapp¥mavhdr.c にあるので、参考にしてください。

8.9.2 記述方法

メモリアクセス違反ハンドラは、図 8.18のように C 言語関数として記述します。

```
#include "kernel.h"
void  _kernel_mavhdr (VT_MAV *pk_mav, VT_EXC *pk_exc)       ←(1)
{
    /* 処理 */
}
```

図8.18　メモリアクセス違反ハンドラ記述例

図の解説

(1)　メモリアクセス違反ハンドラの関数名は、"_kernel_mavhdr"と決められています。pk_mavはアクセス違反の種類、pk_excはCPU例外発生時点のレジスタ情報が格納されたパケットアドレスです。

VT_EXC 型の定義は、前述(図 8.14～図 8.17)の通りです。

vt_reg0, vt_reg1 の内容は、メモリアクセス違反ハンドラ終了時にカーネルによって該当レジスタにリストアされます。例外処理のためにレジスタを書き換えたい場合は、このパケットの内容を書き換えてください。但し、vt_reg0.ssr,ctxid,r15 は書き換えてはなりません。書き換えた場合の動作は保証されません。

なお、FPU レジスタと DSP レジスタは、このパケットには含まれていません。カーネルはこれらのレジスタの保存を行わずにハンドラを起動し、ハンドラ終了時もリストア処理は行いません。例外処理のためにレジスタを書き換えたい場合は、DSP/FPU 例外に限りハンドラでこれらのレジスタを直接書き換えてください。DSP/FPU に起因する例外でない場合は、DSP/FPU レジスタの操作自体ができない場合があります。なお、これらのレジスタを書き換えるプログラムは、アセンブラで記述する必要があります。

図 8.19に、VT_MAV 型を示します。

```
typedef struct {
    UW      type;    /* エラー種別 */
    UW      access;  /* リード/ライト種別 */
} VT_MAV;
```

図8.19　VT_MAV 型

type には、以下のいずれかが渡ります。

- E_NOEXS : MMU 対象領域内で、メモリオブジェクトとして存在しないアドレスをアクセス、または特権モードで P3 領域をアクセス
- E_MACV : 存在するメモリオブジェクトにアクセスしたが、アクセス許可がない

access は、リードの場合は TPM_READ(0)、ライトの場合は TPM_WRITE(1)が設定されます。

8.9.3 CPU 例外ハンドラ専用マクロ

メモリアクセス違反ハンドラでも、「8.8.2 CPU 例外ハンドラ専用マクロ」に示すマクロを使用することができます。

8.9.4 レジスタ使用規約

表 8.12に、メモリアクセス違反ハンドラのレジスタ使用規約と初期値を示します。

表8.12 メモリアクセス違反ハンドラのレジスタのレジスタ使用規約と初期値

No	レジスタ	規約*1	初期値
1	PC	ー	ハンドラのアドレス
2	SR	○	後述の説明を参照してください
3	R0～R3	◎	不定
4	R4	◎	pk_mav
5	R5	◎	pk_exc
6	R6	◎	不定
7	R7	◎	不定
8	R8～R14, MACH, MACL, GBR	◎	不定
9	R15	○	非タスクスタック
10	PR	○	カーネル内のメモリアクセス違反復帰処理のアドレス
11	[DSP] DSR, RS, RE, MOD, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1	○*2	CPU 例外発生前と同じ
12	[FPU] FPSCR	○*2	CPU 例外発生前と同じ
13	[FPU] FPUL, FPR0_BANK0～FPR15_BANK0	○*2	CPU 例外発生前と同じ
15	[FPU] FPR0_BANK1～FPR15_BANK1	○*2	CPU 例外発生前と同じ

【注】 *1 "◎"は保証せずに使用可能、"○"はハンドラ関数からリターンするときには起動時の値に戻す必要があります。

*2 例外要因を解決したい場合のみ、該当するレジスタを変更してください。

メモリアクセス違反ハンドラ起動時の SR レジスタの各ビットは、以下のようになります。特に、BL=1 の状態で起動されることに注意してください。BL=1 の時にサービスコールを呼び出してはなりません。

- モード(MD)ビット：常に 1
- レジスタバンク(RB)ビット：常に 0
- ブロック(BL)ビット：常に 1
- その他のビット：CPU 例外発生前と同じ

メモリアクセス違反ハンドラでは、SR.BL,IMASK を起動時点でマスクされている割込みが受理されるように変更してはなりません。

8.9.5 DSP, FPU について

メモリアクセス違反ハンドラでは、例外処理のためにレジスタを書き換える場合を除いて、DSP, FPU を利用した演算を行ってはなりません。

8.10 システムダウンルーチン

8.10.1 概要

システムダウンルーチンは、vsys_dwn サービスコールが呼び出されたときや、カーネル内部で異常を検出した場合に呼び出されます。システムダウンルーチンには、その原因となった各種情報が渡されます。システムダウンルーチンは、必ず作成してカーネルに組み込まなければなりません。

なお、サンプルのシステムダウンルーチンが samples¥sysapp¥sysdwn.c にあるので、参考にしてください。

8.10.2 記述方法

システムダウンルーチンは、図 8.20のように C 言語関数として記述します。

```
#include "kernel.h"
void    _kernel_sysdwn ( ER type, VW inf1, VW inf2, VW inf3)
{
    /* 処理 */
    while(1);
}
```

図8.20 システムダウンルーチンの記述例

システムダウンルーチンの関数名は、"_kernel_sysdwn"と定められています。

システムダウンルーチンからリターンしてはなりません。

8.10.3 レジスタ使用規約

システムダウンルーチンからリターンしてはならないため、システムダウンルーチンでは自由にレジスタを使用できます。

以下のレジスタにパラメータが渡ります。

- R4：type
- R5：inf1
- R6：inf2
- R7：inf3

渡されるパラメータの意味は、以下を参照してください。

■関連ページ 「16.1 システムダウン時の情報」

9. 標準タイマドライバ

9.1 概要

標準タイマドライバは、コンフィギュレータで CFG_OPTTMR を選択しなかった場合は、必ず作成してカーネルに組み込まなければなりません。

本製品では、標準タイマドライバのサンプルを提供しています。サンプルタイマドライバは、samples¥***shnnn***¥kernel¥knl_side¥tmrdrv.c です。表 9.1に、サンプルタイマドライバの一覧を示します。

表9.1　**HI7300/PX V.1.01 で提供されるサンプル標準タイマドライバ**

shnnnn	**対象マイコンと対象内蔵タイマモジュール**	
73180	SH73180	TMU CH0
7343	SH7343	TMU CH0
7780	SH7780	TMU CH3
7785	SH7785	TMU CH3

9.2 関数構成

標準タイマドライバは、タイマ初期化ルーチンとタイマ割込みルーチンから構成されています。タイマ割込みルーチンは、カーネルのタイマ割込みハンドラ_kernel_isig_tim( )から呼び出され、割込み要因をクリアします。また、タイマ初期化ルーチンは、初期化ルーチンとして実行されます。

いずれの関数も、以下のように関数名が定められています。

- タイマ初期化ルーチン：_kernel_tmrini()
- タイマ割込みルーチン：_kernel_tmrint()

9.2.1 タイマ初期化ルーチン(_kernel_tmrini())

タイマ初期化ルーチンでは、コンフィギュレータが kernel_macro.h に出力する表 9.2のマクロを使用し初期化を行います。

表9.2　**タイマ初期化ルーチンで使用するマクロ**

マクロ名	**コンフィギュレータの設定項目**	**説明**
TIC_NUME	**[時間管理機能]ページの** CFG_TICNUME	**タイムティック周期**[msec] = TIC_NUME／TIC_DENO **の分子**
TIC_DENO	**[時間管理機能]ページの** CFGTICDENO	
VTCFG_TIMINTNO	**[時間管理機能]ページの** CFG_TIMINTNO	**タイマ割込み番号**
VTCFG_TMRCLOCK	**[時間管理機能]ページの** CFG_TMRCLOCK	**タイマに供給されるクロック**[Hz]
VTCFG_KNLLVL	**[カーネル]ページの** CFG_KNLLVL	**カーネル割込みマスクレベル、タイマ割込みレベル**

タイマ初期化ルーチンでは、以下を行います。

- タイマ割込みハンドラの定義
- タイマデバイス、割込みコントローラの初期化

図 9.1に、タイマ初期化ルーチンの例を示します。

```
extern void _kernel_isig_tim(void);
void _kernel_tmrini(void)
{
   /*** Interrupt handler definition packet ***/
   const T_DINH dinh                                                    ←(1)
   = { TA_HLNG, &_kernel_isig_tim, (MD_BIT|BL_BIT) | ((VTCFG_KNLLVL)<<4)};

   INT     old_sr;

   /* Define timer interrupt handler */                                ←(1)
   if((def_inh(VTCFG_TIMINTNO, &dinh)) != E_OK) {
      while(1) {

      }
   }

   /* Save current SR */
   old_sr = get_cr();

   /* Set SR.BL=1 */
   set_cr(BL_BIT | old_sr);

   /* Cancel TMU module-stop */                                         ←(2)
   TIMER_MSTOP_CANCEL();

   /* Initialize INTC for TMU */                                        ←(3)
   TIMER_INTC_SET(VTCFG_KNLLVL);

   /* Initialize TMU.CH0 */                                             ←(4)
   TIMER_INITIALIZE();

   /* Restore SR */
   set_cr(old_sr);
}
```

図9.1　**タイマ初期化ルーチンの例**

(1) タイマ割込みハンドラを定義します。以下のように定義しなければなりません。
- 割込み番号：VTCFG_TIMINTNO
- ハンドラ属性：TA_HLNG
- ハンドラアドレス："_kernel_isig_tim"(カーネル内部モジュール)
- 起動時の SR：(MD_BIT|BL_BIT) | ((VTCFG_KNLLVL)<<4)

(2) タイマデバイスのモジュールストップを解除します。TIMER_MSTOP_CANCEL()は、mstop_tmu.hで定義されています。

(3) 割込みコントローラを設定します。タイマ割込みレベルが、VTCFG_KNLLVLとなるように設定しなければなりません。TIMER_INTC_SET()は、intc_tmu.hで定義されています。

(4) タイマデバイスを初期化します。タイマへの入力クロックがVTCFG_TMRCLOCK[Hz]の条件で､割込み周期がTIC_NUME/TIC_DENO[msec]となるように初期化しなければなりません。TIMER_INITIALIZE()は、tmu.hで定義されています。

9.2.2 タイマ割込みルーチン (_kernel_tmrint())

タイマ割込みルーチンでは、割込み要因のクリアを行います。

```
void _kernel_tmrint(void)
{
   TIMER_INTERRUPT();                                                    ←(1)
}
```

図9.2 **タイマ割込みルーチンの例**

(1) 割込要因をクリアします。TIMER_INTERRUPT()は、tmu.hで定義されています。

10. コンフィギュレータ

10.1 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以下のようなファイルを生成します。これら、コンフィギュレータが生成するファイルを纏めて「構築ファイル」と呼びます。

- ID 名称ヘッダファイル
  コンフィギュレータ上で、各オブジェクトに対してユーザが指定した名称をID番号に定義します。アプリケーションからインクルードします。
- システム定義ファイル
  システムの構築情報が出力されます。これらのファイルは、kernel_def.cとkernel_cfg.cという２つのファイルからインクルードされます。この2つのファイルは、system¥にあります。
  kernel_def.cはカーネルライブラリとリンクします。つまり、カーネルライブラリから必要なモジュールを抽出するためのファイルです。

また、コンフィギュレータの設定内容は、拡張子が hcf のファイルに保存できます。このファイルを、「コンフィギュレータ設定ファイル」または「HCF ファイル」と呼びます。
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、以下を参照してください、

■関連ページ　「11.1 ロードモジュールの種類」

10.2 リンク単位・カーネルロックモード・[カーネル側]

コンフィギュレータの全ての設定項目は「カーネル側」と「カーネル環境側」に分類されています。

「カーネル側」はカーネルロードモジュール(knl_side)に含める情報、「カーネル環境側」はカーネル環境ロードモジュール(env_side)に含める情報を意味します。

カーネルロードモジュールを更新したくない場合は、コンフィギュレータを「カーネルロックモード」にします。カーネルロックモードでは、「カーネル側」の情報変更が抑止され、カーネル側の構築ファイルも出力されません。

カーネルロックモードに設定するには、メニューバーから[生成->カーネルロックモード]を選択します。

どの項目がカーネル側なのかは、タスクなどのオブジェクト生成については、以下のように設定・確認することができます。

- 生成ダイアログボックスには、[カーネル側]チェックボックスがあります。これをチェックするとカーネル側となります。
- 一覧リストボックスで旗印のアイコンがあるものが「カーネル側」です。

その他の設定項目については、以降の節を参照してください。

また、以下も参照してください。

■関連ページ　「11.1 ロードモジュールの種類」

なお、アプリケーションの C 言語シンボルやセクション名を指定する場合は、そのシンボルの実体を定められたリンク単位(カーネル側/カーネル環境側のロードモジュール)に含めなければなりません。

10.3 コンフィギュレータが出力する構築ファイル

構築ファイルには、アプリケーション用ヘッダファイルと、システム定義ファイルがあります。システム定義ファイルは、コンフィギュレーション結果を取り込む以下の２つのファイルからのみインクルードされます。

- kernel_def.c：カーネル側
- kernel_cfg.c：カーネル環境側

kernel_def.c および kernel_cfg.c は、system¥ディレクトリに格納されています。

表 10.1に、構築ファイルを示します。カーネルロックモードでは「カーネル側」のファイルは出力されません。

表10.1　コンフィギュレータが出力する構築ファイル

項番	分類	ファイル名	リンク単位	kernel_def.c からインクルード	kernel_cfg.c からインクルード
1	アプリケーション用ヘッダファイル	kernel_macro.h	カーネル側	○(kernel.h からインクルードされます)	
2		kernel_id_sys.h	カーネル側	○	○
3		kernel_id.h	カーネル環境側	×	○
4	システム定義ファイル	kernel_def_main.h	カーネル側	○	○
5		kernel_def_import.h	カーネル側	○	○
6		kernel_def_inireg.h	カーネル側	○	×
7		kernel_def_inirtn.h	カーネル側	○	×
8		kernel_def_attmem.h	カーネル側	○	×
9		kernel_cfg_main.h	カーネル環境側	×	○
10		kernel_cfg_import.h	カーネル環境側	×	○
11		kernel_cfg_inireg.h	カーネル環境側	×	○
12		kernel_cfg_inirtn.h	カーネル環境側	×	○
13		kernel_cfg_attmem.h	カーネル環境側	×	○

10.3.1 アプリケーション用ヘッダファイル

(1) kernel_macro.h(カーネル側)

このファイルは、kernel.h からインクルードされます。

(2) kernel_id_sys.h(カーネル側), kernel_id.h(カーネル環境側)

これらのファイルには、コンフィギュレータで各種オブジェクトを生成するときに指定した ID 名称の定義が、以下の形式で出力されます。

```
#define ID_TASK_A 1   /* ID名称「ID_TASK_A」のID値は1 */
```

アプリケーションから必要に応じてこれらのファイルをインクルードすることで、オブジェクトの ID 値として ID 名称を使用することができます。

ID 名称は、オブジェクト生成時に、[カーネル側]をチェックした場合は kernel_id_sys.h に、チェックしない場合は kernel_id.h に出力されます。なお、[カーネル側]をチェックしない場合は、ID 番号をコンフィギュレータに自動割当させることもできます。

また、ドメイン名称は常に kernel_id_sys.h に出力されます。

kernel_id.h はカーネル環境側のファイルなので、カーネル側にリンクするアプリケーションではインクルードしないようにしてください。

10.3.2 システム定義ファイル

(1) kernel_def_main.h(カーネル側)

サービスコール選択結果など、カーネルライブラリから必要な機能モジュールを選択するための情報が出力されます。

(2) kernel_cfg_main.h(カーネル環境側)

タスクの数、リソースプールのサイズなど、システムの規模に関する情報が出力されます。

(3) kernel_def_import.h(カーネル側), kernel_cfg_import.h(カーネル環境側)

コンフィギュレータで C 言語シンボルを指定した場合、これらのファイルにそのシンボルの外部参照文が出力されます。カーネル側のシンボルは kernel_def_import.h に、カーネル環境側のシンボルは kernel_cfg_import.h に出力されます。

(4) kernel_def_inireg.h(カーネル側), kernel_cfg_inireg.h(カーネル環境側)

コンフィギュレータに設定した各種オブジェクトの生成・定義を行うための「初期登録ルーチン」プログラムが出力されます。カーネル側のオブジェクトは kernel_def_inireg.h、カーネル環境側のオブジェクトは kernel_cfg_inireg.h で生成・定義されます。

以下も参照してください。

■関連ページ　「6.22.12 カーネルの起動(vsta_knl, ivsta_knl)」
「16.2.2 コンフィギュレータで指定したオブジェクトを生成できない場合」

(5) kernel_def_inirtn.h(カーネル側), kernel_cfg_inirrtn.h(カーネル環境側)

初期化ルーチン情報が出力されます。カーネル側の初期化ルーチン情報は kernel_def_inirtn.h に、カーネル環境側の初期化ルーチン情報は kernel_cfg_inirtn.h に出力されます。

(6) kernel_def_attmem.h(カーネル側), kernel_cfg_attmem.h(カーネル環境側)

静的メモリオブジェクト情報が出力されます。カーネル側の静的メモリオブジェクト情報は kernel_def_attmem.h に、カーネル環境側の静的メモリオブジェクト情報は kernel_cfg_attmem.h に出力されます。

10.4 ユーザインタフェース

10.4.1 画面構成

図 10.1に、コンフィギュレータの画面構成を示します。

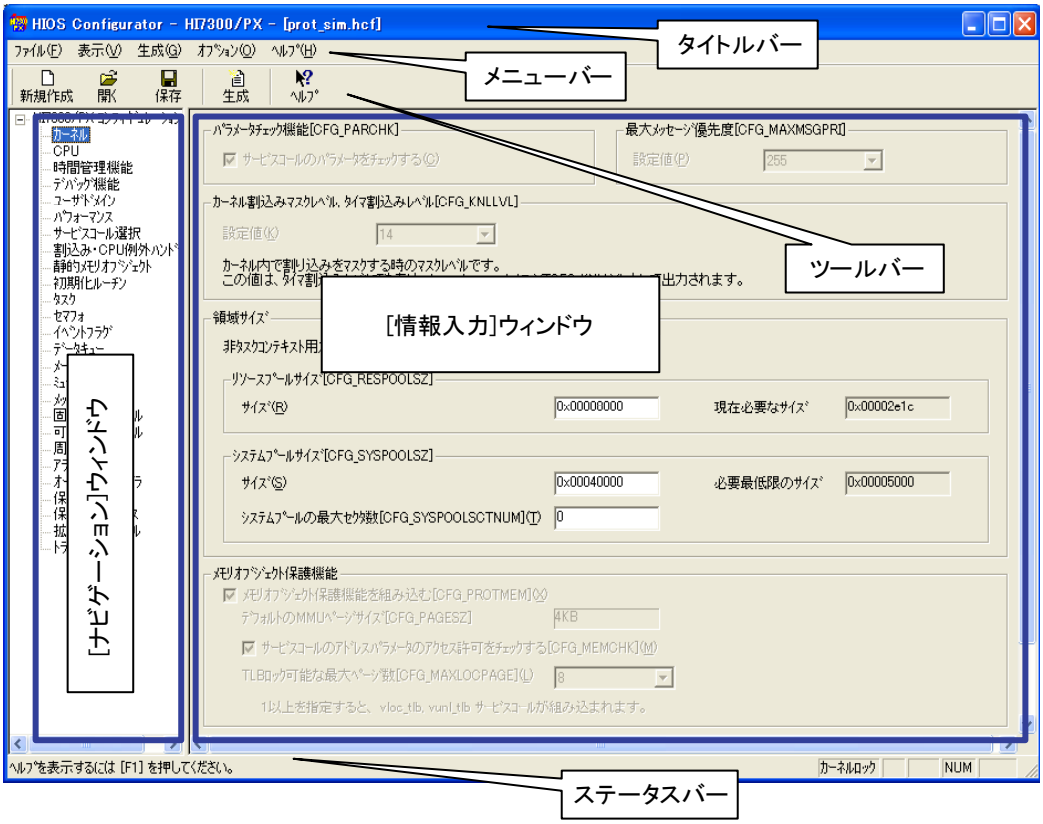


図10.1 コンフィギュレータの画面構成

10.4.2 タイトルバー

ウィンドウ上端のアプリケーション名や文書名が表示される部分です。

タイトルバーには次の要素が含まれます。

(1) アプリケーションのコントロール メニュー ボタン 

(2) アプリケーション名(HIOS Configurator - HI7300/PX -)

(3) HCFファイル名

(4) <最小化>ボタン 

(5) <最大化>／<元のサイズに戻す>ボタン 

(6) <閉じる>ボタン 

10.4.3 メニューバー：[ファイル]メニュー

ユーザが設定した全ての情報は、HCF ファイルに保存することができます。HCF ファイルを読み込むことで前回設定した内容を復元することができます。

[ファイル]メニューは、HCF ファイルの新規作成、開く、保存などの操作を行います。

以下のコマンドがあります。

- [新規作成]
- [開く...]
- [上書き保存]
- [名前を付けて保存...]
- [アプリケーションの終了]

この他に、最近使用した HCF ファイルが最大 4 つまで表示されます。

(1) [新規作成]コマンド([ファイル]メニュー)

新しいコンフィギュレータ設定ファイル(無題.hcf)を作成し、開きます。

ショートカット：

ツールバー： 

キーボード：CTRL+N

(2) [開く...]コマンド([ファイル]メニュー)

既存の HCF ファイルを開きます。

ショートカット：

ツールバー： 

キーボード：CTRL+O

(3) [上書き保存]コマンド([ファイル]メニュー)

作業中の HCF ファイルのファイル名と保存場所を変更しないで保存します。新規作成したコンフィギュレータ設定ファイル(無題.hcf)を初めて保存するときには、[名前を付けて保存]ダイアログボックスが表示されます、適切なファイル名を付けて保存することができます。ファイル名、保存場所を変更して保存したいときは[名前を付けて保存...]コマンドを使います。

ショートカット：

ツールバー： 

キーボード：CTRL+S

(4) [名前を付けて保存...]コマンド([ファイル]メニュー)

作業中の内容を、新しい HCF ファイルに保存します。

(5) [アプリケーションの終了]コマンド([ファイル]メニュー)

コンフィギュレータを終了します。設定内容がまだ保存されていない場合は、保存するかどうか確認するダイアログボックスが表示されます。

10.4.4 メニューバー：[表示]メニュー

[表示]メニューはツールバー、ステータスバーの表示／非表示の操作を行います。
以下のコマンドがあります。

- [ツールバー]
- [ステータスバー]

(1) [ツールバー]コマンド([表示]メニュー)

ツールバーの表示、非表示を切り替えます。ツールバーには、「開く...」などの、最もよく使われるコマンドと同じ機能を持ったツールが含まれています。ツールバーが表示されているときは、「表示」メニューのこのコマンド名の横にチェックマークが表示されます。

(2) [ステータスバー]コマンド([表示]メニュー)

ステータスバーの表示、非表示を切り替えます。ステータスバーには、メニューコマンドやツールバーのボタンを選択したときにコマンドの簡単な説明や、キーボードの特殊キーの ON/OFF 状態などが表示されます。ステータスバーが表示されているときは、このコマンド名の横にチェックマークが表示されます。

10.4.5 メニューバー：[生成]メニュー

[生成]メニューには、以下のコマンドがあります。

- [カーネルロックモード]
- [構築ファイル生成...]

(1) [カーネルロックモード]コマンド([生成]メニュー)

カーネルロックモードの設定／解除を行います。
カーネルロックモード中は、ステータスバーに「カーネルロック」と表示されます。
HCF ファイルには、カーネルロックモードかどうかという情報も保存されます。

■関連ページ　「10.2 リンク単位・カーネルロックモード・[カーネル側]」

(2) [構築ファイル生成]コマンド([生成]メニュー)

構築ファイルを生成します。
本コマンドを選択すると、ファイル生成場所を指定する[構築ファイルの生成]ダイアログボックスが開きます。ファイルの生成場所を指定後、[生成]ボタンをクリックすると構築ファイルが指定された場所に生成されます。

ショートカット：

ツールバー： 

キーボード：CTRL+G

10.4.6 メニューバー：[オプション]メニュー

[オプション]メニューには、以下のコマンドがあります。

- [前回使用したファイルを開く]

(1) [前回使用したファイルを開く]コマンド([オプション]メニュー)

これを選択すると、コンフィギュレータは起動時に前回使用したファイルを自動的に開きます。

10.4.7 メニューバー：[ヘルプ]メニュー

[ヘルプ]メニューには、以下のコマンドがあります。

- [トピックの検索]
- [バージョン情報...]

(1) [トピックの検索]コマンド([ヘルプ]メニュー)

ヘルプファイルを開きます。

(2) [バージョン情報...]コマンド([ヘルプ]メニュー)

コンフィギュレータのバージョンや著作権などを表示します。

10.4.8 ツールバー

ツールバーは、アプリケーションウィンドウ上部のメニューバーのすぐ下に表示されており、頻繁に使用する機能がボタンとして登録されています。

ツールバーの表示と非表示を切り替えるには、[表示]メニューの[ツールバー]コマンドを選択します。

それぞれのボタンは、以下のコマンドに関連付けられています。

[新規作成]コマンド

[開く...]コマンド

[上書き保存]コマンド

[構築ファイル生成]コマンド

コンテキストヘルプモードの起動

10.4.9 ステータスバー

ステータスバーは、メインウィンドウの一番下に表示されます。ステータスバーの表示、非表示を切り替えるには、[表示]メニューの[ステータスバー]コマンドを選択してください。

ステータスバーの左側の部分には、メニューコマンドを選択したときにそれぞれの簡単な説明が表示されます。同様に、ツールバーのボタンにカーソルを合わせた場合も簡単な説明が表示されます。説明を見たあとでそのツールバーのコマンドの実行を中止したいときは、マウスポインタをそのツールバー ボタン以外の位置に移動してマウスボタンを離します。

ステータスバーの右側の部分には、以下の状態が表示されます。

- "カーネルロック"：カーネルロックモード状態
- "カナ"：[カナ]キーが ON の状態
- "CAP"：[Caps Lock]キーが ON の状態
- "NUM"：[Num Lock]キーが ON の状態
- "SCRL"：[Scroll Lock]キーが ON の状態

10.4.10 [ナビゲーション]ウィンドウ

本ウィンドウでは、[情報入力]ウィンドウに表示されるページの選択を行います。マウスまたはキーボードにより一覧内の項目を選択すると、選択された項目に対応したページが[情報入力]ウィンドウに表示されます。

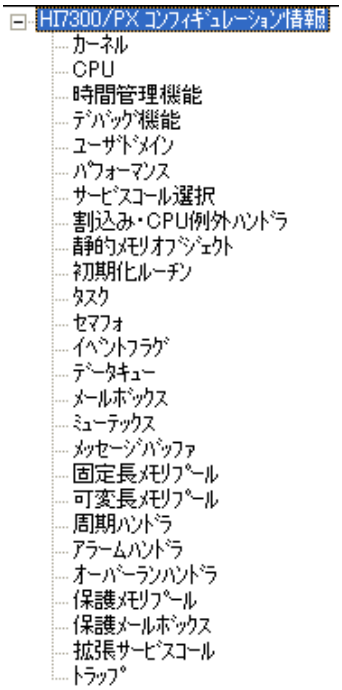


図10.2 ナビゲーションウィンドウ

10.4.11 [情報入力]ウィンドウ

コンフィギュレーション情報を入力するウィンドウです。本ウィンドウには、[ナビゲーション]ウィンドウで選択されたページが表示されます。

各ページの詳細は、以降の節を参照してください。

10.5 ページ構成

表 10.2に、ページ一覧を示します。

表10.2 ページ一覧

No	ページ	設定項目
1	[カーネル]ページ	カーネルの共通的な項目
2	[CPU]ページ	使用するマイコンに関する情報
3	[時間管理機能]ページ	時間管理機能に関する項目
4	[デバッグ機能]ページ	デバッグ機能(デバッギングエクステンション)に関する項目
5	[ユーザドメイン]ページ	ユーザドメインの ID 名称
6	[パフォーマンス]ページ	CPU が内蔵するプログラムパフォーマンスカウンタ(PPC)を用いたパフォーマンス機能に関する項目
7	[サービスコール選択]ページ	組み込むサービスコール
8	[割込み・CPU 例外ハンドラ]ページ	割込み・CPU 例外に関する項目
9	[静的メモリオブジェクト]ページ	静的メモリオブジェクトの登録
10	[初期化ルーチン]ページ	初期化ルーチンの登録
11	[タスク]ページ	タスクに関する項目
12	[セマフォ]ページ	セマフォに関する項目
13	[イベントフラグ]ページ	イベントフラグに関する項目
14	[データキュー]ページ	データキューに関する項目
15	[メールボックス]ページ	メールボックスに関する項目
16	[ミューテックス]ページ	ミューテックスに関する項目
17	[メッセージバッファ]ページ	メッセージバッファに関する項目
18	[固定長メモリプール]ページ	固定長メモリプールに関する項目
19	[可変長メモリプール]ページ	可変長メモリプールに関する項目
20	[周期ハンドラ]ページ	周期ハンドラに関する項目。
21	[アラームハンドラ]ページ	アラームハンドラに関する項目
22	[オーバーランハンドラ]ページ	オーバーランハンドラの定義
23	[保護メモリプール]ページ	保護メモリプールに関する項目
24	[保護メールボックス]ページ	保護メールボックスに関する項目
25	[拡張サービスコール]ページ	拡張サービスコールに関する項目
26	[トラップ]ページ	トラップに関する項目

10.6 CFG 名

コンフィギュレータに設定する項目の多くは、カーネルの動作に影響を与えます。そのような設定項目には、"CFG_"で始まる「CFG 名」が付与されており、コンフィギュレータ画面に表示されるのはもちろん、本マニュアル中でも利用しています。なお、全ての設定項目に名称が付与されているわけではありません。

例えば、[カーネル]ページの CFG_SYSPOOLSZ は、システムプールサイズを意味します。

10.7 各ページ・ダイアログボックスの仕様

10.7.1 [カーネル]ページ

本ページでは、カーネルの共通的な項目を設定します。

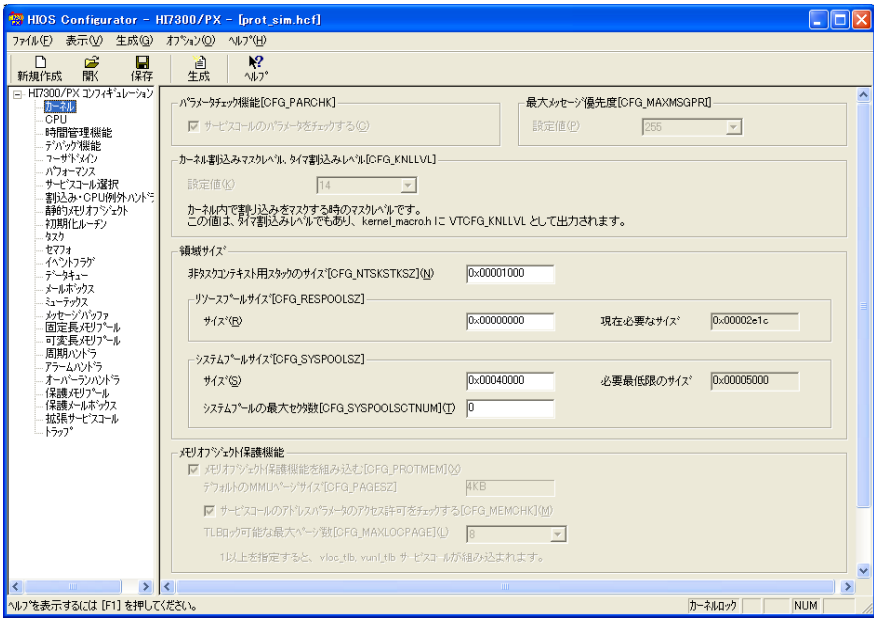


図10.3 [カーネル]ページ

表 10.3に、[カーネル]ページの項目を示します。

表10.3 [カーネル]ページの項目

No	項目	CFG 名	リンク区分
1	パラメータチェック機能	CFG_PARCHK	カーネル側
2	最大メッセージ優先度	CFG_MAXMSGPRI	カーネル側
3	カーネル割込みマスクレベル、タイマ割込みレベル	CFG_KNLLVL	カーネル側
4	非タスクコンテキスト用スタックのサイズ	CFG_NTSKSTKSZ	カーネル環境側
5	リソースプールサイズ	CFG_RESPOOLSZ	カーネル環境側
6	システムプールサイズ	CFG_SYSPOOLSZ	カーネル環境側
7	システムプールの最大セクタ数	CFG_SYSPOOLSCTNUM	カーネル環境側
8	メモリオブジェクト保護機能を組み込む	CFG_PROTMEM	カーネル側
9	デフォルトの MMU ページサイズ	CFG_PAGESZ	カーネル側
10	サービスコールのアドレスパラメータのアクセス許可をチェックする	CFG_MEMCHK	カーネル側
11	TLB ロック可能な最大ページ数	CFG_MAXLOCPAGE	カーネル側

(1) [パラメータチェック機能[CFG_PARCHK]]

これをチェックすると、サービスコールのパラメータエラーがチェックされます。
なお、このチェックを外すと、自動的に後述の CFG_MEMCHK も選択が解除されます。

■関連ページ　「6.3.2 パラメータチェック機能」

(2) [最大メッセージ優先度[CFG_MAXMSGPRI]]

メールボックスおよび保護メールボックスで使用するメッセージ優先度の最大値を、1～255 の中から選択してください。メールボックス、保護メールボックスともに使用しない場合は、この項目は意味を持ちませんので、どの値を選択してもかまいません。
本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TMAX_MPRI 255  /* 255 を選択した場合の例 */
```

(3) [カーネル割込みマスクレベル,タイマ割込みレベル[CFG_KNLLVL]]

カーネルは、クリティカルセクションを実行する場合に、SR レジスタの I ビットを CFG_KNLLVL にします。この値より高いレベルの割込みは、カーネルのクリティカルセクション実行中も遅延されずに受理されますが、そのハンドラからサービスコールを呼び出してはなりません。1～15 を選択できます。
また、CFG_KNLLVL はタイマ割込みレベルでもあります。
本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define VTKNL_LVL 15  /* 15 を選択した場合の例 */
```

[時間管理]ページの CFG_OPTTMR をチェックしない場合、即ち標準タイマドライバを使用する場合は、標準タイマドライバの初期化ルーチンでタイマ割込みレベルが VTKNL_LVL となるように初期化しなければなりません。
最適化タイマドライバを使用する場合は、カーネルがタイマ割込みレベルを CFG_KNLLVL に初期化します。

■関連ページ　標準タイマドライバの作成方法→「9. 標準タイマドライバ」

(4) [領域サイズ]グループ

(a) [非タスクコンテキスト用スタックのサイズ[CFG_NTSKSTKSZ]]

非タスクコンテキスト用のスタックサイズをバイト単位で指定してください。指定可能な範囲は、256～0x20000000 の整数です。指定値は、4 の倍数に切上げられます。

■関連ページ　「12.4 非タスクコンテキストスタックサイズの算出」

(b) [リソースプールサイズ[CFG_RESPOOLSZ]]

リソースプールサイズをバイト単位で指定してください。指定可能な範囲は、256～0x20000000 の整数です。指定値は、4 の倍数に切上げられます。
[現在必要なサイズ]には、コンフィギュレータで生成したリソースプールを使用するオブジェクトのために必要なサイズが表示されます。このサイズ未満の値を指定した場合は、以下のエラーメッセージが表示されます。

「リソースプールサイズは、[現在必要なサイズ]以上のサイズにしてください。」

ただし、リソースプールはシステム稼動中にも消費されるため、このエラーが表示されなければ良いというわけではありません。リソースプールの消費ケースを考慮して、十分なサイズを指定してください。

■関連ページ　・「4.31 メモリの断片化とその対策」
・「13. リソースプールサイズの見積り」

(c) [システムプールサイズ[CFG_SYSPOOLSZ]]

CFG_SYSPOOLSZ には、システムプールサイズをバイト単位で指定してください。指定可能な範囲は、0～0x20000000 の整数です。指定値は、CFG_PROTMEM がチェックされていない場合は 64 の倍数に、チェックされている場合は CFG_PAGESZ(=4kB)で切上げられます。

[必要最低限のサイズ]には、コンフィギュレータで生成したシステムプールを使用するオブジェクトのために最低限必要なサイズが表示されます。このサイズ未満の値を指定した場合は、以下のエラーメッセージが表示されます。

「システムプールサイズは、[必要最低限のサイズ]以上のサイズにしてください。」

[必要最低限のサイズ]に表示される値は、システムプールをセクタ管理しない(CFG_SYSPOOLSCTNUM が 0)の場合のサイズです。CFG_SYSPOOLSCTNUM に 0 以外を設定している場合は、このエラーが出なくてもシステムプールが不足する可能性があります。この場合、カーネル起動時に対象オブジェクトを生成することができず、システム起動に失敗します。

■関連ページ　「16.2.2 コンフィギュレータで指定したオブジェクトを生成できない場合」

また、システムプールを使用するオブジェクトをサービスコールで動的に生成する場合には、システムプールの消費ケースを考慮して、十分なサイズを指定してください。

■関連ページ　・「4.31 メモリの断片化とその対策」
・「14. システムプールサイズの見積り」

(d) [システムプールの最大セクタ数[CFG_SYSPOOLSCTNUM]]

システムプールのセクタ数を指定してください。0 を指定すると、セクタ管理されません。

以下の式で算出される値より大きい値を指定した場合は、実際の最大セクタ数はカーネルによってこの式で算出される値に補正されます。

- CFG_PROTMEM 選択時：CFG_SYSPOOLSZ/(4096×32)
- CFG_PROTMEM 非選択時：CFG_SYSPOOLSZ/(64×32)

(5) [メモリオブジェクト保護機能]グループ

このグループでは、メモリオブジェクト保護機能に関する設定を行います。

■関連ページ　「4.21 メモリオブジェクト保護機能」

(a) [メモリオブジェクト保護機能を組み込む[CFG_PROTMEM]]

メモリオブジェクト保護機能を組み込む場合は、チェックしてください。
本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define VTCFG_PROTMEM  1  /* 組み込む場合は 1,組み込まない場合は 0 */
```

(b) [デフォルトの MMU ページサイズ [CFG_PAGESZ]]

デフォルトのページサイズとは、静的メモリオブジェクト以外のメモリオブジェクトで使用する MMU ページサイズです。4kB(4096)固定で、変更はできません。一方、静的メモリオブジェクトは、個々にページサイズを 4kB 以外のページサイズに選択できます。

CFG_PROTMEM に関係なく、本設定に対し、kernel_macro.h に以下のステートメントが出力されます。ただし、この定義は CFG_PROTMEM がチェックされない場合は意味を持ちません。

```
#define VTCFG_PAGESZ  4096
```

(c) [サービスコールのアドレスパラメータのアクセス許可をチェックする[CFG_MEMCHK]]

これをチェックすると、サービスコールのアドレスパラメータについて、アクセス許可が適切かがチェックされます。しかし、このチェックは大きなオーバヘッドを伴うため、デバッグが十分でチェックが冗長と判断できる場合には、CFG_MEMCHK のチェックを外すことでこのエラーチェックを省略することができます。

なお、これをチェックすると、自動的に CFG_PARCHK も選択されます。

■関連ページ　「6.3.3 アドレスパラメータのアクセス許可チェック機能」

(d) [TLB ロック可能な最大ページ数[CFG_MAXLOCPAGE]]

同時に TLB ロック可能な最大ページ数を 0～32 の範囲から選択してください。

0 以外を選択した場合は、vloc_tlb, vunl_tlb サービスコールが組み込まれます。なお、[サービスコール]ページでは、これらのサービスコールを選択することはできません。

10.7.2 [CPU]ページ

本ページでは、使用するマイコンに関する情報を設定します

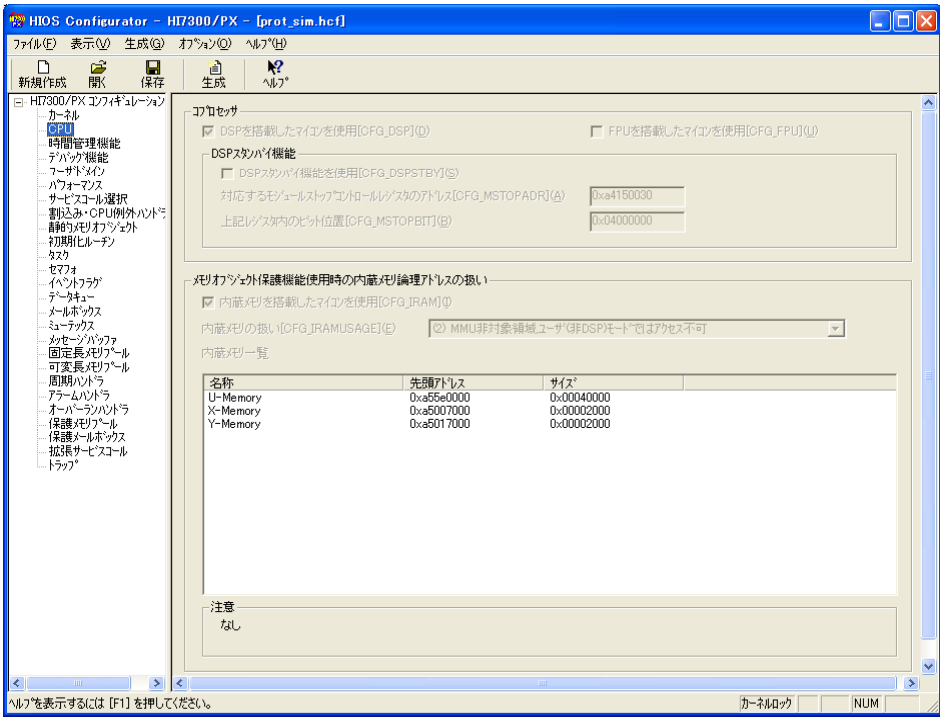


図10.4 [CPU]ページ

表 10.4に、[CPU]ページの項目を示します。

表10.4 [CPU]ページの項目

No	項目	CFG 名	リンク区分
1	DSP を搭載したマイコンを使用	CFG_DSP	カーネル側
2	FPU を搭載したマイコンを使用	CFG_FPU	カーネル側
3	DSP スタンバイ機能を使用	CFG_DSPSTBY	カーネル側
4	対応するモジュールストップコントロールレジスタのアドレス	CFG_MSTOPADR	カーネル側
5	上記レジスタ内のビット位置	CFG_MSTOPBIT	カーネル側
6	内蔵メモリを搭載したマイコンを使用	CFG_IRAM	カーネル側
7	内蔵メモリの扱い	CFG_IRAMUSAGE	カーネル側
8	内蔵メモリー覧	−	カーネル側

(1) [コプロセッサ]グループ

(a) [DSP を搭載したマイコンを使用[CFG_DSP]], [FPU を搭載したマイコンを使用 [CFG_FPU]]

DSP を搭載したマイコンを使用する場合は CFG_DSP を、FPU を搭載したマイコンを使用する場合

は CFG_FPU を、それぞれ選択してください。なお、CFG_DSP. CFG_FPU の両方を同時に選択しないようにしてください。

タスクやハンドラの TA_COP0 属性は、CFG_DSP をチェックした場合のみ使用できます。同様に、TA_COP1, TA_COP2 属性は、CFG_FPU をチェックした場合のみ使用できます。

(b) [DSP スタンバイ機能を使用 [CFG_DSPSTBY]]

DSP スタンバイ機能とは、TA_COP0 属性でないタスクが実行するときに、X/Y メモリなどへのクロック供給を停止(モジュールストップ)することで、消費電力を低減する機能です。

DSP スタンバイ機能を使用する場合は、これをチェックしてください。チェックすると、vchg_cop サービスコールが組み込まれます。なお、[サービスコール]ページでは、vchg_cop サービスコールを選択することはできません。

なお、CFG_DSP をチェックしない場合は、本項目は変更不可となり、意味を持ちません。

■関連ページ　　DSP スタンバイ機能→「4.25 DSP スタンバイ機能」

(c) [対応するモジュールストップコントロールレジスタのアドレス[CFG_MSTOPADR]]、[上記レジスタ内のビット位置[CFG_MSTOPBIT]]

使用するマイコンのハードウェアマニュアルを参照して、DSP スタンバイ機能によってコントロールするモジュールストップコントロールレジスタのアドレスと、ビット位置を指定してください。

[アドレス]に指定できるのは、0xa0000000～0xbfffffff および 0xe0000000～0xffffffff の範囲で、かつ 4 の倍数です。

[ビット位置]は、対象モジュールに対応するビットが 1 となるビットパターンを指定してください。通常は、X/Y メモリのみを対象となるようにしてください。

なお、CFG_DSP をチェックしない場合、または CFG_DSPSTBY をチェックしない場合は、本項目は変更不可となり、意味を持ちません。

(2) [メモリオブジェクト保護機能使用時の内蔵メモリ論理アドレスの扱い]グループ

本グループは、メモリオブジェクト保護機能を使用する場合のみ有効です。[カーネル]ページで CFG_PROTMEM をチェックしない場合は、本グループの項目は変更不可となり、意味を持ちません。

■関連ページ　　「5.3.3 内蔵メモリ」

(a) [内蔵メモリを搭載したマイコンを使用[CFG_IRAM]]

内蔵メモリを搭載したマイコンを使用する場合は、これをチェックしてください。

チェックしない場合は、以降の項目は変更不可となり、意味を持ちません。

(b) [内蔵メモリの扱い[CFG_IRAMUSAGE]]

P2 または P4 領域に配置されている内蔵メモリの論理アドレスをどのように扱うかを、以下の中から選択してください。

(1)　MMU非対象領域、全モードからアクセス可能
(2)　MMU非対象領域、ユーザ(非DSP)モードではアクセス不可
(3)　MMU対象領域

カーネルは、vsta_knl サービスコールで、この設定に応じて RAMCR レジスタの RP, RMD ビットを初期化します。

ただし、CFG_IRAM をチェックしない場合は、カーネルは RAMCR レジスタを初期化しません。

(c) [内蔵メモリ一覧]

ここで指定した内蔵メモリの論理アドレスが、CFG_IRAMUSAGE で指定した扱いとなります。
以下のポップアップメニューがあります。

- [定義] : 内蔵メモリの定義を追加するために、[内蔵メモリの定義]ダイアログボックスを開きます。
- [削除] : 選択された内蔵メモリの定義を削除します。
- [変更] : 選択された内蔵メモリの定義を変更するために、[内蔵メモリの定義情報を変更]ダイアログボックスを開きます。

(d) [注意]

[カーネル]ページの CFG_PROTMEM が選択されていない場合は、本グループの設定項目はすべて無効となります。また、本グループの全ての項目は変更できなくなります。
この場合、[注意]に表 10.5に示すメッセージが表示されます。

表10.5 [メモリオブジェクト保護機能使用時の内蔵メモリ論理アドレスの扱い]グループの[注意]

条件	表示メッセージ
CFG_PROTMEM が未選択	CFG_PROTMEM が選択されていないので、本グループの設定項目は無効です。

10.7.3 [内蔵メモリの定義]ダイアログボックス、[内蔵メモリの定義情報を変更]ダイアログボックス

図10.5 [内蔵メモリの定義]ダイアログボックス

[内蔵メモリの定義]ダイアログボックスは、[CPU]ページのポップアップメニューから[定義]を選択した時に、[内蔵メモリの定義情報を変更]ダイアログボックスは、[CPU]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

ここで定義する内容は、必ず使用するマイコンに搭載されている内蔵メモリと同じ情報を設定してください。

(1) [名称]

内蔵メモリの名称を指定してください。この名称は、[CPU]ページでユーザが各内蔵メモリを区別するためだけに用いられます。コンフィギュレータは、この入力に関して何のチェックも行いません。

(2) [先頭アドレス]

内蔵メモリの論理アドレスを数値で指定してください。指定するアドレスは P2 または P4 領域内でなければなりません。指定された値は、4kB 境界に切り捨てられます。

また、[先頭アドレス]から[サイズ]の範囲は、他の内蔵メモリの範囲と重ならないように注意してください。

(3) [サイズ]

内蔵メモリのサイズを数値で指定します。指定可能なサイズは、4kB～2MB の範囲内で、かつ 4kB の倍数です。

(4) [定義]ボタン([内蔵メモリの定義]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の内蔵メモリの定義を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(5) [OK]ボタン([内蔵メモリの定義情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(6) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.4 [時間管理機能]ページ

本ページでは、時間管理機能に関する項目を設定します。

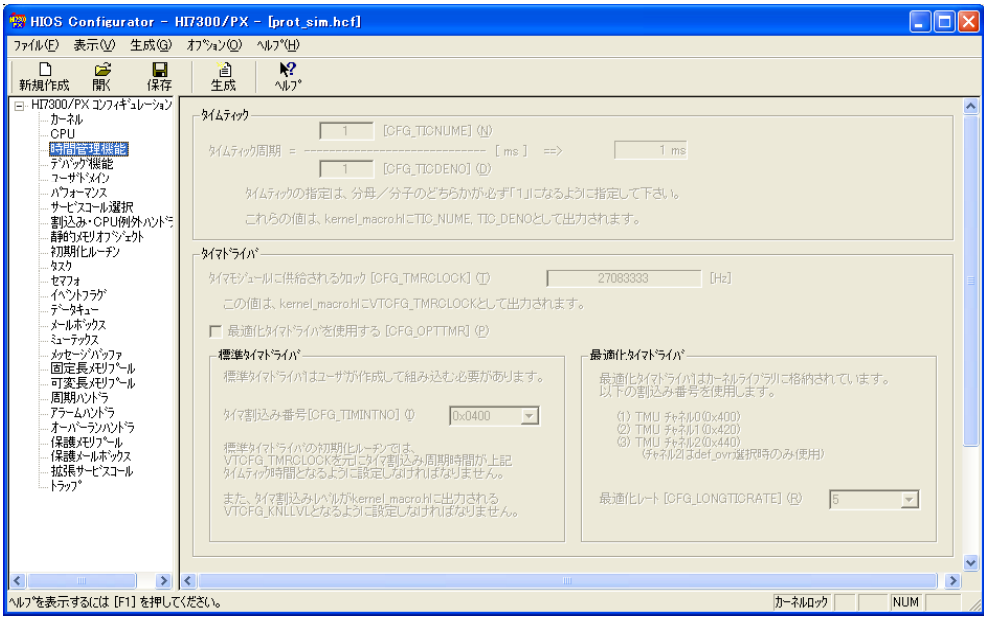


図10.6 [時間管理]ページ

表 10.6に、[時間管理機能]ページの項目を示します。

表10.6 [時間管理機能]ページの項目

No	項目	CFG 名	リンク区分
1	**タイムティック周期の分子**	CFG_TICNUME	**カーネル側**
2	**タイムティック周期の分母**	CFG_TICDENO	**カーネル側**
3	**タイマモジュールに供給されるクロック**	CFG_TMRCLOCK	**カーネル側**
4	**最適化タイマドライバを使用する**	CFG_OPTTMR	**カーネル側**
5	**タイマ割込み番号**	CFG_TIMINTNO	**カーネル側**
6	**最適化レート**	CFG_LONGTICRATE	**カーネル側**

(1) [タイムティック]

タイムティック周期時間は、CFG_TICNUME/CFG_TICDENO[ms]となります。CFG_TICNUME には 1～65535、CFG_TICDENO には 1～100 の整数を指定できますが、少なくとも一方は 1 でなければなりません。

タイムティックの周期時間は、タスクのタイムアウトや周期ハンドラなどの時間の精度を意味します。これを小さくすると、カーネルの時間管理の精度が高まりますが、タイマ割込みの発生頻度が高まり、オーバヘッドが増えます。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TIC_NUME  1  /* CFG_TICNUME に 1 を指定した場合 */
#define TIC_DENO  1  /* CFG_TICDENO に 1 を指定した場合 */
```

(2) [タイマドライバ]グループ

(a) [タイマモジュールに供給されるクロック[CFG_TMRCLOCK]]

使用するタイマモジュールに供給されるクロック周波数を Hz 単位で指定してください。指定できるのは、1～0x40000000(約 1GHz)の整数です。

通常は、内蔵 TMU に供給されるクロック周波数を指定してください。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define VTCFG_TMRCLOCK  10000000  /* 10000000(10MHz)を指定した場合 */
```

CFG_OPTTMR をチェックしない場合、つまり標準タイマドライバを使用する場合は、標準タイマドライバの初期化ルーチンでタイマ割込み周期時間が TIC_NUME/TIC_DENO[ms]となるように初期化しなければなりません。

(b) [最適化タイマドライバを使用する[CFG_OPTTMR]]

タイマドライバとしてカーネルが提供する最適化タイマドライバを使用する場合は、これをチェックしてください。標準タイマドライバを使用する場合は、チェックを外してください。

(c) [標準タイマドライバのタイマ割込み番号[CFG_TIMINTNO]]

標準タイマドライバで使用する割込み番号(INTEVT コード)を選択してください。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define VTCFG_TIMINTNO  0x400  /* 0x400 を選択した場合 */
```

標準タイマドライバの初期化ルーチンでは、この番号に対して idef_inh サービスコールを用いてタイマ割込みハンドラを定義しなければなりません。

なお、CFG_OPTTMR をチェックした場合は、本項目は変更不可となり、意味を持ちません。

(d) [最適化レート[CFG_LONGTICRATE]]

最適化タイマドライバの最適化レートを 2～255 から選択してください。

なお、CFG_OPTTMR をチェックしない場合は、本項目は変更不可となり、意味を持ちません。

■関連ページ　・標準タイマドライバの作成方法→「9. 標準タイマドライバ」
・最適化タイマドライバ→「4.17 最適化タイマドライバ」

10.7.5 [デバッグ機能]ページ

本ページでは、デバッグ機能(デバッギングエクステンション)に関する項目を設定します。

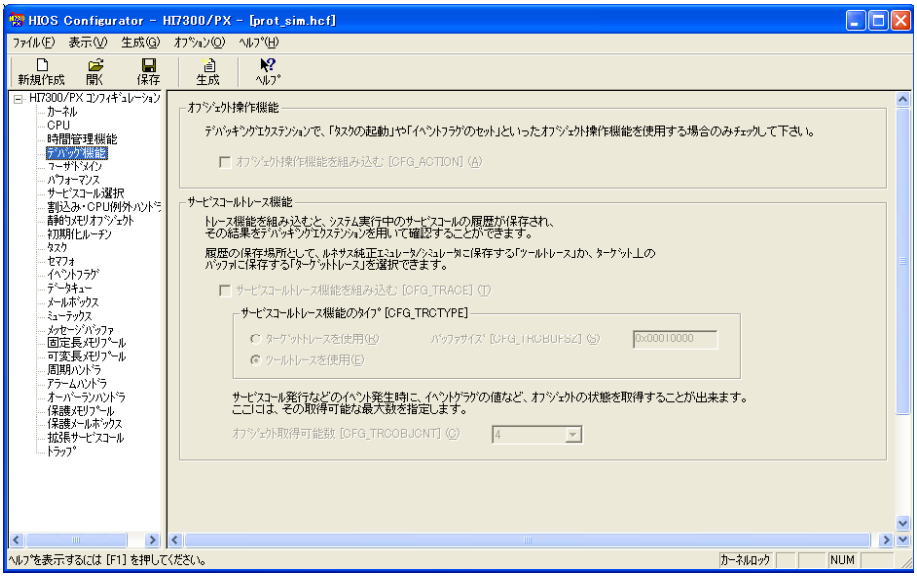


図10.7 [デバッグ機能]ページ

表 10.7に、[デバッグ機能]ページの項目を示します。

表10.7 [デバッグ機能]ページの項目

No	項目	CFG 名	リンク区分
1	オブジェクト操作機能を組み込む	CFG_ACTION	カーネル側
2	サービスコールトレース機能を組み込む	CFG_TRACE	カーネル側
3	サービスコールトレース機能のタイプ	CFG_TRCTYPE	カーネル側
4	バッファサイズ	CFG_TRCBUFSZ	カーネル環境側
5	オブジェクト取得可能数	CFG_TRCOBJCNT	カーネル環境側

(1) [オブジェクト操作機能]グループ

CFG_ACTION をチェックすると、デバッギングエクステンションで「タスクの起動」や「イベントフラグのセット」といったサービスコールを使用した機能を使用できるようになります。

(2) [サービスコールトレース機能]グループ

このグループでは、サービスコールトレース機能に関する設定を行います。下記も参照してください。

■関連ページ　「4.27 サービスコールトレース機能」

(a) [サービスコールトレース機能を組み込む[CFG_TRACE]]

これをチェックすると、サービスコールトレース機能が組み込まれ、vget_trc サービスコールも組み込まれます。なお、[サービスコール]ページで vget_trc サービスコールを選択することはできません。

トレース結果は、デバッギングエクステンションで参照することができます。

チェックしない場合は、以降の項目は変更不可となり、意味を持ちません。

(b) [サービスコールトレース機能のタイプ[CFG_TRCTYPE]]

サービスコールトレース機能のタイプとして、「ターゲットトレース」「ツールトレース」のいずれかを選択してください。

「ターゲットトレース」では、トレース情報を格納するためのトレースバッファがターゲット上に必要となります。

(c) [バッファサイズ[CFG_TRCBUFSZ]]

ターゲットトレースで使用するバッファサイズを指定してください。指定した値は、4 の倍数に切上げられます。指定できるのは、512～0x20000000 の整数です。

なお、CFG_TRCTYPE で「ツールトレース」を選択した場合は、本項目は変更不可となり、意味を持ちません。

(d) [オブジェクト取得可能数[CFG_TRCOBJCNT]]

トレース取得時点のオブジェクト状態を取得することができます。どういったオブジェクト状態を取得するかは、デバッギングエクステンションから設定できます。ここには、同時に取得可能なオブジェクト取得数(最大値)を、[取得しない]または 1～32 の中から選択してください。

10.7.6 [ユーザドメイン]ページ

本ページでは、ユーザドメインの ID 名称を設定します。

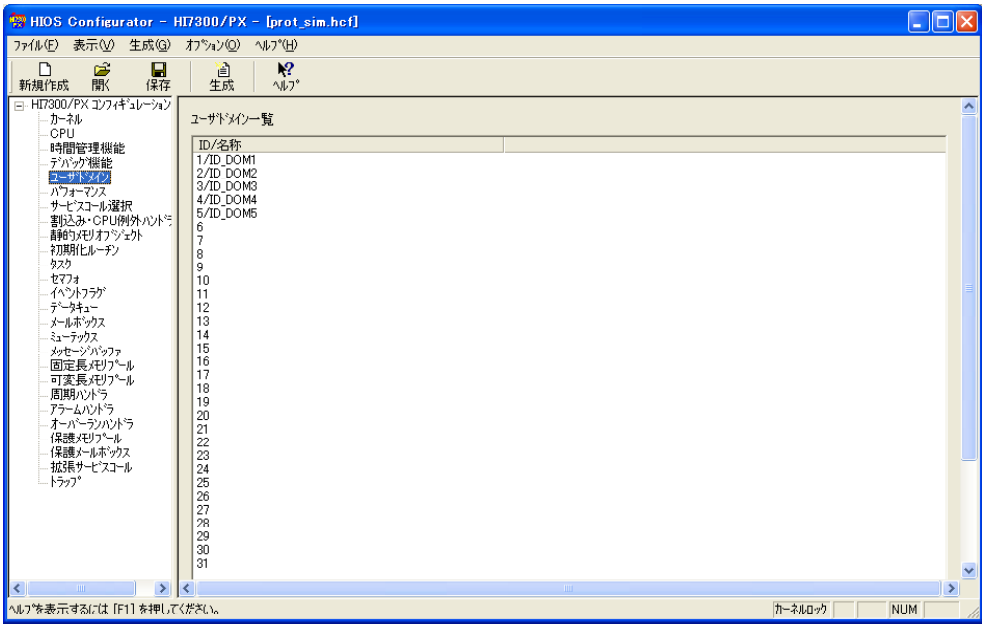


図10.8 [ユーザドメイン]ページ

本カーネルでは、ドメイン ID が 1～31 のユーザドメインが常に存在する仕様になっています。

本ページでは、これらのユーザドメインに対する ID 名称を設定します。ユーザドメインの ID 名称は、常に「カーネル側」の扱いであり、kernel_id_sys.h に出力されます。

ポップアップメニューとして、以下の項目があります。

- [編集]：選択されたドメイン ID に対し、ID 名称を入力するための[ユーザドメイン ID の設定]ダイアログボックスを開きます。
- [削除]：選択されたドメイン ID の ID 名称を削除します。

10.7.7 [ユーザドメイン ID の設定]ダイアログボックス

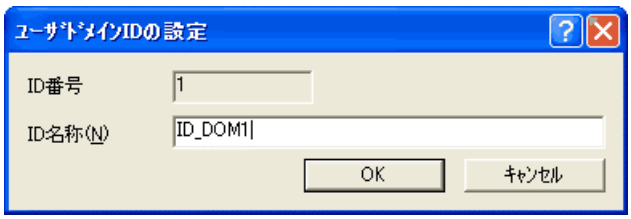


図10.9 [ユーザドメイン ID の設定]ダイアログボックス

[ユーザドメイン]ページのポップアップメニューから[編集]を選択すると、本ダイアログボックスが開きます。

[ID 番号]には、[ユーザドメイン]ページで選択したドメイン ID 番号が表示されます。

[ID 名称]には、付与する ID 名称を入力してください。

[OK]ボタンを押すと、本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。[キャンセル]ボタンを押すと、本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.8 [パフォーマンス]ページ

本ページでは、CPU が内蔵するプログラムパフォーマンスカウンタ(PPC)を用いたパフォーマンス機能に関する項目を設定します。

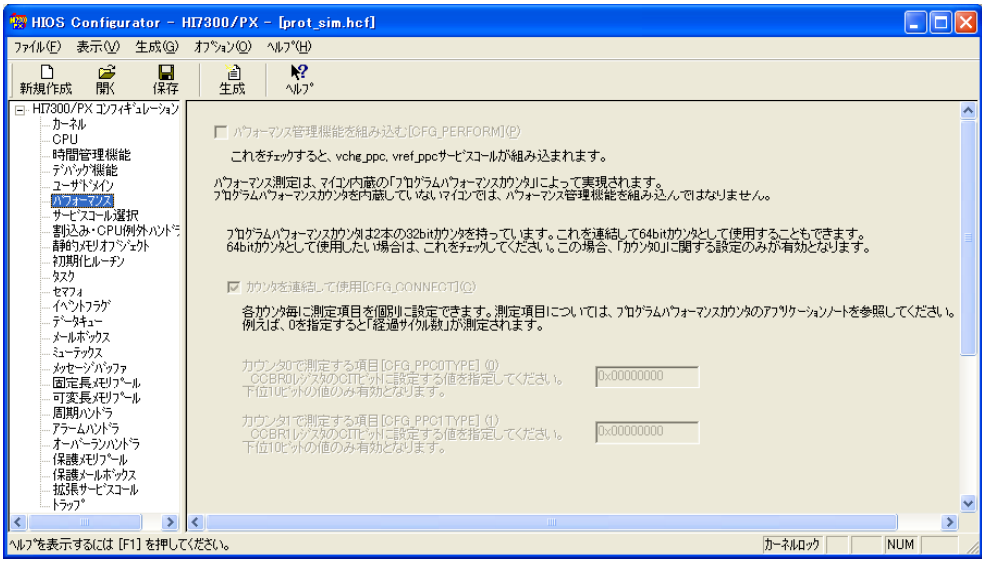


図10.10 [パフォーマンス]ページ

なお、パフォーマンス管理機能を使用するには、プログラムパフォーマンスカウンタに関する知識が必要です。「SH-4A, SH4AL-DSP プログラムパフォーマンスカウンタ アプリケーションノート」を参照してください。

■関連ページ 「4.26 パフォーマンス管理機能」

表 10.8に、[パフォーマンス]ページの項目を示します。

表10.8 [パフォーマンス]ページの項目

No	項目	CFG 名	リンク区分
1	パフォーマンス管理機能を組み込む	CFG_PERFORM	カーネル側
2	カウンタを連結して使用	CFG_CONNECT	カーネル側
3	カウンタ 0 で測定する項目	CFG_PPC0TYPE	カーネル側
4	カウンタ 1 で測定する項目	CFG_PPC1TYPE	カーネル側

(1) [パフォーマンス管理機能を組み込む[CFG_PERFORM]]

パフォーマンス管理機能を使用する場合はチェックしてください。チェックすると、vchg_ppc, vref_ppc サービスコールが組み込まれます。なお、[サービスコール選択]ページで、これらのサービスコールを選択することはできません。

使用するマイコンがプログラムパフォーマンスカウンタを内蔵していない場合は、必ずチェックを

外してください。
　チェックしない場合は、以降の項目は全て変更不可となり、意味を持ちません。

(2) [カウンタを連結して使用[CFG_CONNECT]]

　2 本のプログラムパフォーマンスカウンタを連結して使用する場合は、チェックしてください。
　2 本のプログラムパフォーマンスカウンタは、各々32bit です。例えば、「経過サイクル数」を測定する場合、32bit カウンタでは CPU の内部動作周波数が 400MHz の場合には、約 10 秒でオーバーフローすることになります。オーバフローした場合は、正確な結果を得ることができないため、このような場合にはカウンタを連結して使用するようにしてください。

(3) [カウンタ 0 で測定する項目[CFG_PPC0TYPE]]、[カウンタ 1 で測定する項目[CFG_PPC1TYPE]]

　それぞれ、カウンタ 0, 1 で測定する項目を指定します。指定する値は、それぞれ PPC の CCBR0,CCBR1 レジスタの CIT ビットの値を、bit0 からに割り当てた値です。
　例えば、0 を指定すると、経過サイクル数(CPU サイクル数)となります。詳細は「SH-4A, SH4AL-DSP プログラムパフォーマンスカウンタ アプリケーションノート」を参照してください。

10.7.9 [サービスコール選択]ページ

本ページでは、組み込むサービスコールを選択します。

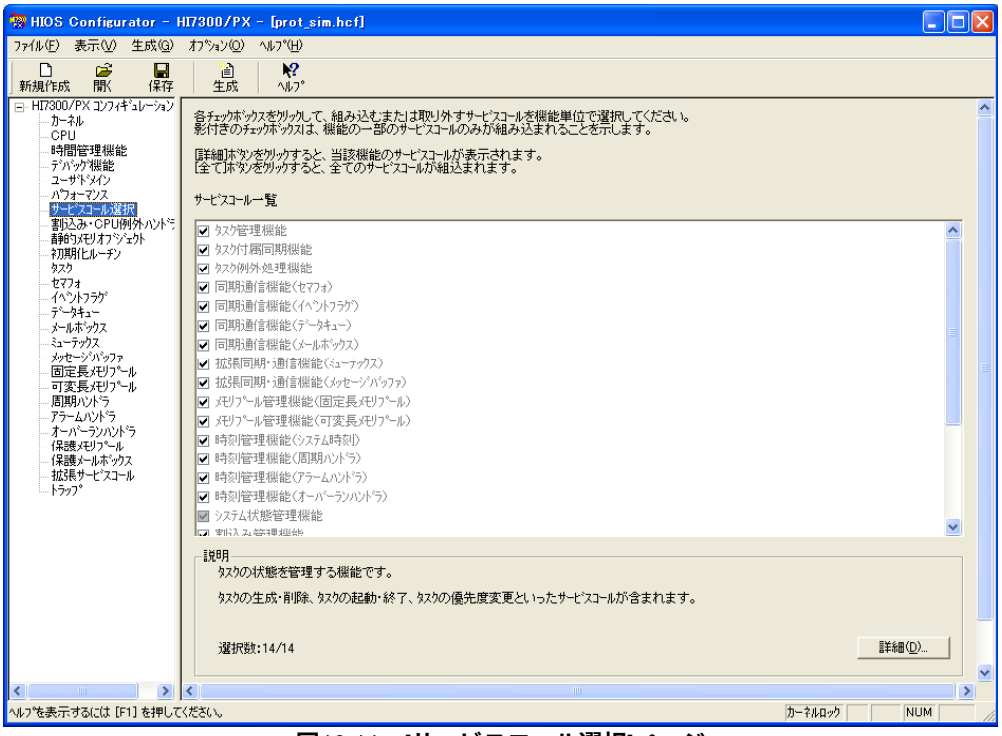


図10.11 [サービスコール選択]ページ

[サービスコール一覧]には、「タスク管理機能」や「同期通信機能(セマフォ)」などのように、機能グループ単位にチェックボックスが割り当てられています。

さらに、いずれかの機能グループを選択してダブルクリックするか、または[詳細]ボタンを押すと、その機能グループに属する個々のサービスコールを選択するためのダイアログボックスが開きます。

また、[全て]ボタンを押すと、全サービスコールが選択されます。

サービスコールによっては、同じ機能でも set_flg と iset_flg のように、"i"で始まる名称のものと"i"のつかない名称のものの 2 つが存在するものがあります。本ページでは、これらはまとめて「set_flg」として選択します。

なお、本ページの設定項目はすべて「カーネル側」です。

(1) 選択できないサービスコール

表 10.9に示すサービスコールは本ページでは選択できません。選択はできませんが、組み込まれるかどうかの表示の確認は可能です。

表10.9　選択できないサービスコール

機能グループ	サービスコール	組み込まれる条件
タスク管理	cre_tsk	常に組み込まれます
	ext_tsk	常に組み込まれます
固定長メモリプール管理	icra_mpf	[カーネル]ページで CFG_PROTMEM を選択し、かつ cre_mpf を選択した場合
可変長メモリプール管理	ivcra_mpl	[カーネル]ページで CFG_PROTMEM を選択し、かつ cre_mpl を選択した場合
システム状態管理	vsta_knl	常に組み込まれます
	vsys_dwn	常に組み込まれます
	vget_trc	[デバッグ機能]ページの CFG_TRACE を選択した場合
	ivbgn_int, ivend_int	これらのサービスコールは HI7000/4 シリーズとの互換性のためだけに API のみが規定されており、その実体は存在しません。
	vchg_cop	[CPU]ページの CFG_DSP, CFG_DSPSTBY を共に選択した場合
割込み管理	def_inh	常に組み込まれます
システム構成管理	def_exc	常に組み込まれます
パフォーマンス管理	vchg_ppc, vref_ppc	[パフォーマンス]ページで CFG_PERFORM を選択した場合
メモリオブジェクト管理	vloc_tlb, vunl_tlb	[カーネル]ページで CFG_PROTMEM を選択し、かつ[カーネル]ページで CFG_MAXLOCPAGE を 0 以外とした場合

(2) オブジェクトの生成に必要なサービスコール

表 10.10に示すオブジェクトに関して、「生成」あるいは「定義」のサービスコールを選択しない場合は、そのオブジェクトを使用することができなくなります。具体的には、コンフィギュレータの各ページでオブジェクトの「生成」や「定義」を行っても、それらは無視されます。これらのオブジェクトを使用する場合は、必ず「生成」「定義」サービスコールを選択してください。

表10.10　各オブジェクトに必要な「生成」「定義」のサービスコール

オブジェクト	「生成」「定義」サービスコール	オブジェクト	「生成」「定義」サービスコール
タスク例外処理ルーチン	def_tex	可変長メモリプール	cre_mpl
セマフォ	cre_sem	周期ハンドラ	cre_cyc
イベントフラグ	cre_flg	アラームハンドラ	cre_alm
データキュー	cre_dtq	オーバーランハンドラ	def_ovr
メールボックス	cre_mbx	拡張サービスコールルーチン	def_svc
ミューテックス	cre_mtx	トラップルーチン	vdef_trp
メッセージバッファ	cre_mbf	保護メモリプール	icre_mpp
固定長メモリプール	cre_mpf	保護メールボックス	cre_mbp

また、表 10.10に示す「生成」「定義」のサービスコールを選択しない場合は、同じ機能グループの全サービスコールは、本ページの設定に関わらず組み込まれなくなります。

なお、表 10.10に示す「生成」「定義」のサービスコールの選択を解除しようとすると、図 10.12のような警告ダイアログが表示されます。

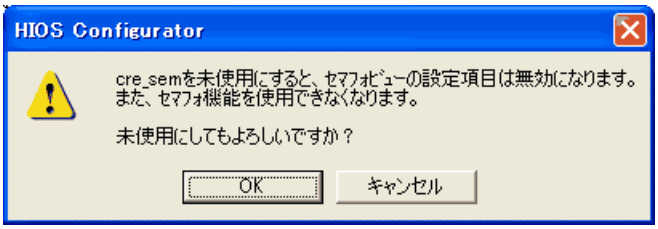


図10.12　[サービスコール選択]ページの警告ダイアログ

(3) メモリオブジェクト保護機能に関するサービスコール

以下の機能グループに属するサービスコールは、[カーネル]ページで CFG_PROTMEM をチェックしない場合には、本ページの設定に関わらず組み込まれなくなります。

- メモリオブジェクト管理機能
- 保護メモリプール管理機能
- 保護メールボックス管理機能

10.7.10 [割込み・CPU 例外ハンドラ]ページ

本ページでは、割込み・CPU 例外に関する項目を設定します。

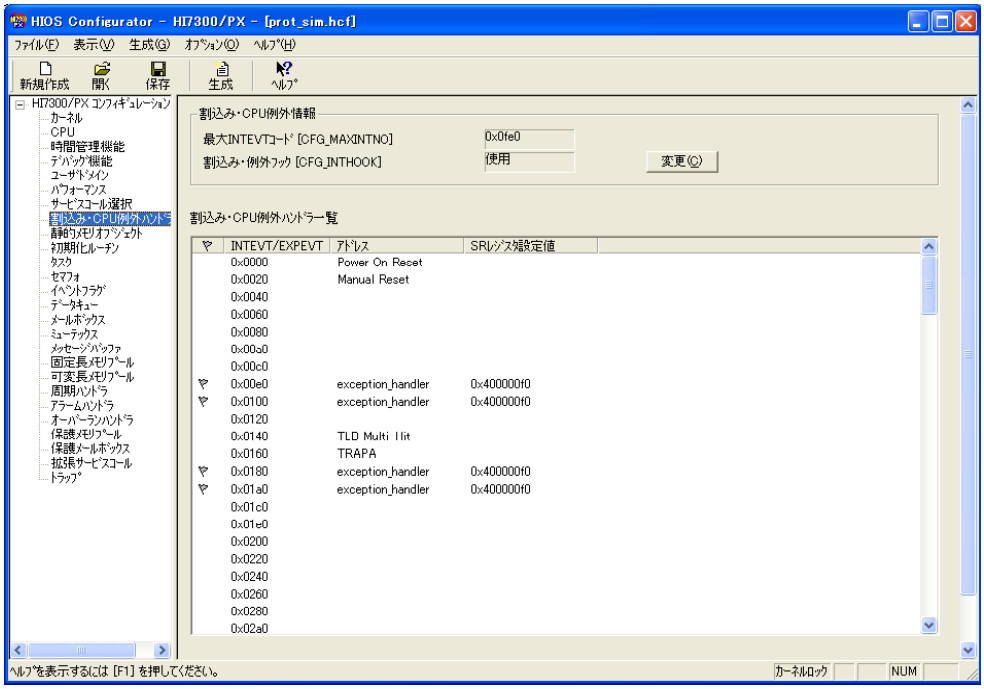


図10.13 [割込み・CPU 例外]ページ

表 10.11に、[割込み・CPU 例外ハンドラ]ページの項目を示します。

表10.11 [割込み・CPU 例外ハンドラ]ページの項目

No	項目	CFG 名	リンク区分
1	最大 INTEVT コード	CFG_MAXINTNO	カーネル環境側
2	割込み・例外フック	CFG_INTHOOK	カーネル側
3	割込み・CPU 例外ハンドラの定義	−	カーネル側/カーネル環境側

(1) [割込み・CPU 例外情報]グループ

[最大 INTEVT コード]には、ターゲットシステムで使用する割込みの中で、最大の INTEVT コードが表示されます。

[割込み・例外フック]には、割込み・例外フックを使用するかどうかが表示されます。

[変更]ボタンを押すと、これらを変更するための[割込み・CPU 例外情報の変更]ダイアログボックスが開きます。

(2) [割込み・CPU 例外ハンドラー覧]グループ

ここには、0～CFG_MAXINTNO までの全 INTEVT/EXPEVT コードに対するハンドラの定義状況が表示されます。旗のアイコンは、そのハンドラが[カーネル側]として定義されていることを示します。

ポップアップメニューとして、以下の項目があります。カーネルロックモードでは、[カーネル側]のハンドラについてはこれらのポップアップメニューは選択できなくなります。

- [定義] : 選択された INTEVT/EXPEVT コードに対し、ハンドラを定義するための[割込み・CPU 例外ハンドラの定義]ダイアログボックスを開きます。
- [解除] : 選択された INTEVT/EXPEVT コードのハンドラの定義を解除します。

(3) 特別な INTEVT/EXPEVT

表 10.12に示す INTEVT/EXPEVT コードは、 [定義]、[解除]できないケースがあります。

表10.12　[定義], [解除]できない INTEVT/EXPEVT

INTEVT/EXPEVT コード	要因	リストの表示	[定義],[解除]できない条件
0	パワーオンリセット	Power On Reset	常に
0x20	マニュアルリセット	Manual Reset	常に
0x140	TLB 多重ヒット	TLB Multi-Hit	常に
0x160	無条件トラップ	TRAPA	常に
[時間管理]ページの CFG_TIMINTNO 選択値	―	Timer(Standard)	[時間管理]ページの CFG_OPTTMR をチェックしない場合
0x400	TMU チャネル 0	Timer CH0(Optimized)	[時間管理]ページの CFG_OPTTMR をチェックした場合
0x420	TMU チャネル 1	Timer CH1(Optimized)	[時間管理]ページの CFG_OPTTMR をチェックした場合
0x440	TMU チャネル 2	Timer CH2(Optimized)	[時間管理]ページの CFG_OPTTMR をチェックし、かつ[サービスコール]ページで def_ovr が選択されている場合

10.7.11 [割込み・CPU 例外情報の変更]ダイアログボックス

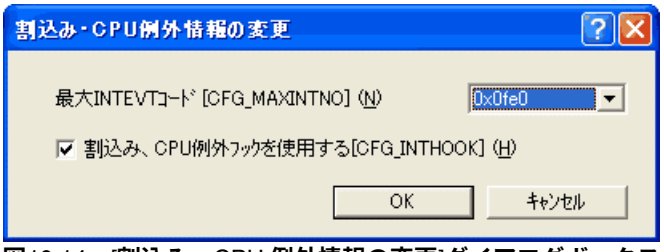


図10.14 [割込み・CPU 例外情報の変更]ダイアログボックス

本ダイアログボックスは、[割込み、CPU 例外ハンドラ]ページの[変更]ボタンを押したときに開きます。

(1) [最大 INTEVT コード[CFG_MAXINTNO]]

ターゲットシステムで使用する割込みの中で、最大の INTEVT コードを選択してください。選択できるのは、0x20 の倍数で、最小は 0x400、最大は 0x3fe0 です。ただし、以下のいずれよりも小さい値は選択できません。

- [時間管理]ページの CFG_TIMINTNO(CFG_OPTTMR チェックしない場合)
- 0x420(CFG_OPTTMR チェックした場合で、def_ovr 未選択の場合)
- 0x440(CFG_OPTTMR チェックした場合で、def_ovr 選択の場合)
- 定義済みのハンドラの INTEVT コード

(2) [割込み、CPU 例外フックを使用する[CFG_INTHOOK]]

割込み、CPU 例外フック機能を使用する場合は、チェックしてください。

■関連ページ 「8.5 割込み・例外フックルーチン」

(3) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(4) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.12 [割込み・CPU 例外ハンドラの定義]ダイアログボックス

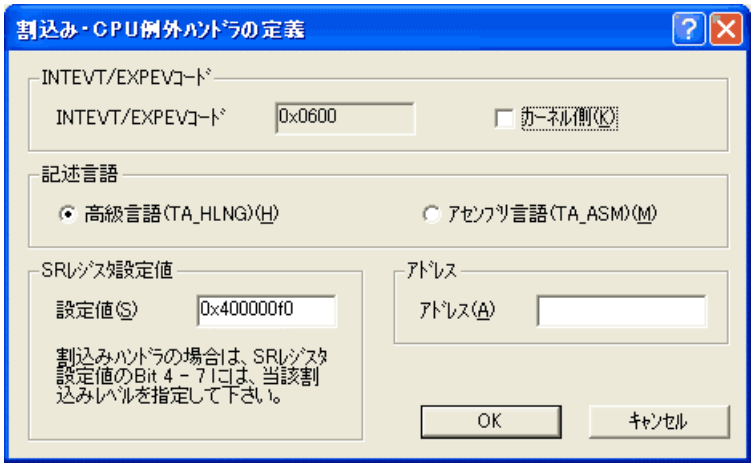


図10.15 [割込み・CPU 例外ハンドラの定義]ダイアログボックス

本ダイアログボックスは、[割込み、CPU 例外ハンドラ]ページのポップアップメニューから[定義]を選択した時に開きます。なお、これらのハンドラは def_inh または def_exc で動的に定義することもできます。

(1) [INTEVT/EXPEVT コード]

[割込み・CPU 例外]ページで選択した INTEVT コードまたは EXPEVT コードが表示されます。

(2) [カーネル側]

定義するハンドラをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(3) [記述言語]

ハンドラが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(4) [SR レジスタ設定値]

以下の関連ページを参考に、ハンドラ起動時の SR レジスタの値を数値で入力してください。

■関連ページ	**・割込みハンドラ起動時の** SR→**「8.4.2 レジスタ使用規約」**
	・CPU 例外ハンドラ起動時の SR→**「8.8.3 レジスタ使用規約」**

(5) [アドレス]

ハンドラの開始アドレスを、C 言語シンボルまたは数値で指定してください。

(6) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(7) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.13 [静的メモリオブジェクト]ページ

本ページでは、静的メモリオブジェクトを登録します。

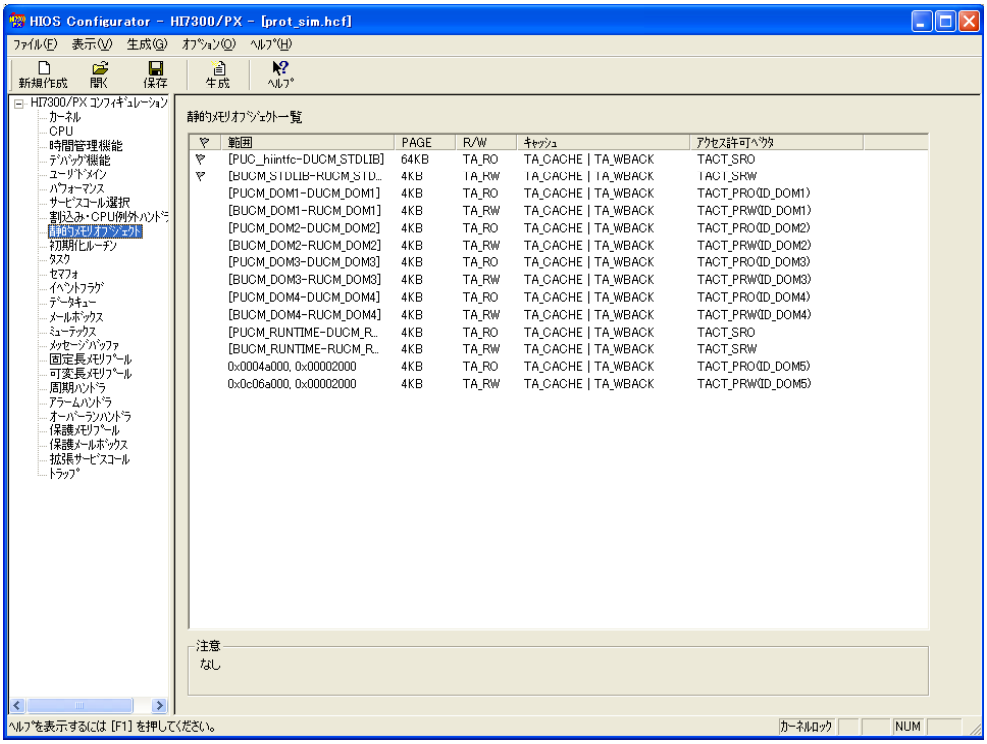


図10.16 [静的メモリオブジェクト]ページ

(1) [静的メモリオブジェクト一覧]

登録済みの静的メモリオブジェクトがリストに表示されます。旗のアイコンは、その静的メモリオブジェクトが[カーネル側]として登録されていることを示します。

以下のポップアップメニューがあります。カーネルロックモードでは、[カーネル側]の静的メモリオブジェクトについては[削除]、[変更]は選択できなくなります。

- [登録]：静的メモリオブジェクトを登録するために、[静的メモリオブジェクトの登録]ダイアログボックスを開きます。
- [削除]：選択された静的メモリオブジェクトを削除します。
- [変更]：選択された静的メモリオブジェクトの設定を変更するために、[静的メモリオブジェクトの登録情報を変更]ダイアログボックスを開きます。

(2) [注意]

[カーネル]ページの CFG_PROTMEM が選択されていない場合は、本ページの設定項目はすべて無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.13に示すメッセージが表示されます。

表10.13　[静的メモリオブジェクト]ページの[注意]

条件	表示メッセージ
CFG_PROTMEM が未選択	CFG_PROTMEM が選択されていないので、本ページの全設定項目は無効です。

10.7.14 [静的メモリオブジェクトの登録]ダイアログボックス、[静的メモリオブジェクトの登録情報を変更]ダイアログボックス

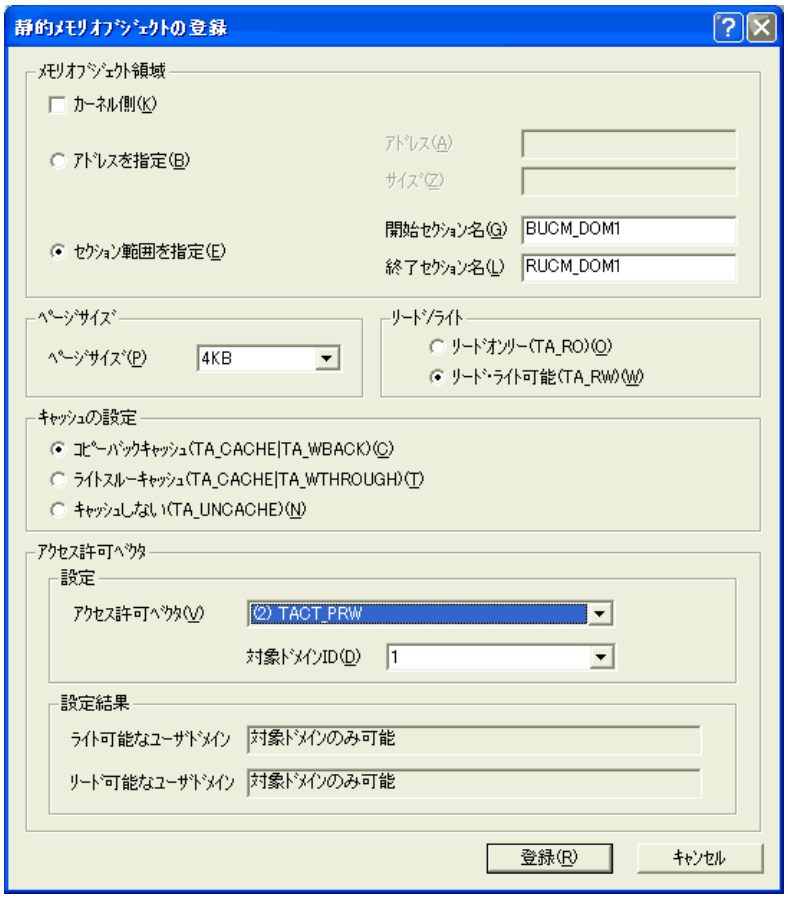


図10.17 [静的メモリオブジェクトの登録]ダイアログボックス

[静的メモリオブジェクトの登録]ダイアログボックスは、[静的メモリオブジェクト]ページのポップアップメニューから[登録]を選択した時に、[静的メモリオブジェクトの登録情報を変更]ダイアログボックスは、[静的メモリオブジェクト]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

静的メモリオブジェクトは、MMU 対象領域にのみ配置でき、その先頭アドレスは[ページサイズ]境界でなければなりません。

■関連ページ　「5.3.1(1) MMU 対象領域と MMU 非対象領域」

また、静的メモリオブジェクトは全てカーネルドメインに所属します。

(1) [メモリオブジェクト領域]グループ

(a) [カーネル側]

登録する静的メモリオブジェクトをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(b) [アドレスを指定]、[セクション範囲を指定]

静的メモリオブジェクトのアドレスを指定する方法として、いずれかを選択してください。
[アドレスを指定]を選択した場合は、[アドレス]と[サイズ]を入力してください。
[セクション範囲を指定]を選択した場合は、[開始セクション名]と[終了セクション名]を入力してください。

(c) [アドレス]、[サイズ]

[アドレス]には、静的メモリオブジェクトの先頭アドレスを数値または C 言語シンボルで指定してください。[アドレス], [サイズ]で決まる範囲は、MMU 対象領域でなければなりませんが、コンフィギュレータは、これを完全には検査しないので、注意してください。
[サイズ]には、静的メモリオブジェクトのサイズ(バイト数)を指定してください。[サイズ]には、1～0x20000000 の整数を指定できます。[サイズ]は、[ページサイズ]の倍数に切り上げられます。

(d) [開始セクション名]、[終了セクション名]

[開始セクション名]から[終了セクション名]までがひとつのメモリオブジェクトとなります。リンク時には、これを考慮して指定したセクションを MMU 対象領域の[ページサイズ]境界に配置しなければなりません。

例：

PUCM_DOM1, CUCM_DOM1, DUCM_DOM1の3つのセクションをひとつの静的メモリオブジェクトにする場合は、以下のようにします。
本ダイアログでは、[開始セクション名]にPCUM_DOM1、[終了セクション名]にDUCM_DOM1を指定します。そして、リンク時には、PUCM_DOM1をMMU対象領域の[ページサイズ]の境界アドレスに配置します。

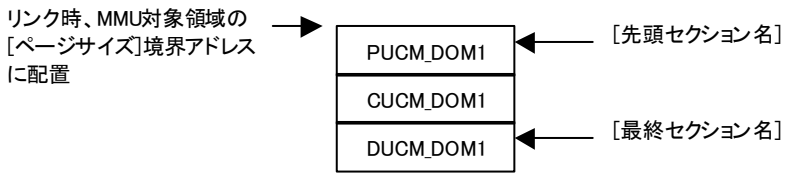


図10.18　静的メモリオブジェクトの[セクション範囲を指定]

■関連ページ　　「11.15 メモリ配置」

(2) [ページサイズ] グループ

ページサイズを選択してください。

選択できるページサイズは、4kB, 8kB, 64kB, 256kB, 1MB, 4MB, 64MB の 7 種類です。ただし、使用する CPU が拡張機能を持たない SH4AL-DSP, SH-4A の場合は、実際に適用されるページサイズは表 10.14に示すようになります。

表10.14　拡張機能を持たない SH4AL-DSP, SH-4A 使用時のページサイズ

選択したページサイズ	実際のページサイズ
4kB	4kB
8kB	
64kB	64kB
256kB	
1MB	1MB
4MB	
64MB	

■関連ページ　「4.21.5 ページサイズ」

(3) [リード/ライト] グループ

リードオンリー(TA_RO)と扱うか、リードライト可能(TA_RW)と扱うかを選択してください。

(4) [キャッシュの設定] グループ

キャッシュ Enable 時の振る舞いを、以下の中から選択してください。

- コピーバックキャッシュ(TA_CACHE|TA_WBACK)
- ライトスルーキャッシュ(TA_CACHE|TA_WTHROUGH)
- キャッシュしない(TA_UNCACHE)

(5) [アクセス許可ベクタ] グループ

[アクセス許可ベクタ]は、以下の中から選択してください。

(1)　TACT_KERNEL
(2)　TACT_PRW(domid)
(3)　TACT_PRO(domid)
(4)　TACT_SRW
(5)　TACT_SRO
(6)　TACT_SRPW(domid)

(2),(3),(6)を選択した場合のみ[対象ドメイン ID]が有効となります。この場合、[対象ドメイン ID]は許可対象とするユーザドメイン ID を選択してください。

[設定結果]には、[リード/ライト]、[アクセス許可ベクタ]、[対象ドメイン ID]の設定によって、どのユーザドメインからライトまたはリード可能かが表示されます。

表 10.15に、[設定結果]の表示内容を示します。

表10.15　[設定結果]の表示内容

設定			[設定結果]の表示	
リード/ライト	アクセス許可ベクタ	対象ドメイン	ライト可能なユーザドメイン	リード可能なユーザドメイン
TA_RO	TACT_KERNEL	無効	全ユーザドメイン不可 (カーネルドメインも不可)	全ユーザドメイン不可
	TACT_PRW(domid)	有効		対象ドメインのみ可能
	TACT_PRO(domid)	有効		対象ドメインのみ可能
	TACT_SRW	無効		全ユーザドメイン可能
	TACT_SRO	無効		全ユーザドメイン可能
	TACT_SRPW(domid)	有効		全ユーザドメイン可能
TA_RW	TACT_KERNEL	無効	全ユーザドメイン不可	全ユーザドメイン不可
	TACT_PRW(domid)	有効	対象ドメインのみ可能	対象ドメインのみ可能
	TACT_PRO(domid)	有効	全ユーザドメイン不可	対象ドメインのみ可能
	TACT_SRW	無効	全ユーザドメイン可能	全ユーザドメイン可能
	TACT_SRO	無効	全ユーザドメイン不可	全ユーザドメイン可能
	TACT_SRPW(domid)	有効	対象ドメインのみ可能	全ユーザドメイン可能

(6) [登録]ボタン([静的メモリオブジェクトの登録]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の静的メモリオブジェクトの登録を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(7) [OK]ボタン([静的メモリオブジェクトの登録情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(8) キャンセルボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

(9) 注意事項

- 静的メモリオブジェクトは MMU 対象領域のみに配置可能ですが、コンフィギュレータおよびカーネルはこれに関するチェックは十分には行っていません。MMU 非対象領域に配置した場合は、そのメモリオブジェクトへのアクセス時には、メモリ属性とアクセス許可ベクタは無視され、メモリオブジェクトとして意味がなくなります。
- 静的メモリオブジェクトの先頭アドレスは、指定した[ページサイズ]境界でなければなりませんが、コンフィギュレータは[アドレス]を値で指定された場合を除き、これに関するチェックを行いません。先頭アドレスが[ページサイズ]境界でない場合は、カーネル起動時にシステムダウンとなります。
- 指定した静的メモリオブジェクトの範囲は、他の静的メモリオブジェクトまたはシステムプールと重複してはなりません。重複とは、論理アドレス空間での重複はもちろん、物理アドレスの重複も含みます。しかし、コンフィギュレータおよびカーネルは、このエラーを検出しません。この場合、システムの正常な動作は保証されません。
-

10.7.15 [初期化ルーチン]ページ

本ページでは、初期化ルーチンを登録します。

図10.19 [初期化ルーチン]ページ

登録済みの初期化ルーチンがリストに表示されます。旗のアイコンは、その初期化ルーチンが「カーネル側」として登録されていることを示します。

初期化ルーチンは、カーネル起動時に、「カーネル側」(旗のアイコンがあるもの)の初期化ルーチンが本リストの上から順に実行され、次に「カーネル環境側」(旗のアイコンが無いもの)の初期化ルーチンが本リストの上から順に実行されます。

以下のポップアップメニューがあります。

- [登録]：初期化ルーチンを登録するために、[初期化ルーチンの登録]ダイアログボックスを開きます。
- [削除]：選択された初期化ルーチンを削除します。
- [変更]：選択された初期化ルーチンの設定を変更するために、[初期化ルーチンの登録情報を変更]ダイアログボックスを開きます。
- [上へ]：選択された初期化ルーチンをひとつ上の初期化ルーチンと順序を入れ替えます。
- [下へ]：選択された初期化ルーチンをひとつ下の初期化ルーチンと順序を入れ替えます。

10.7.16 [初期化ルーチンの登録]ダイアログボックス、[初期化ルーチンの登録情報を変更]ダイアログボックス

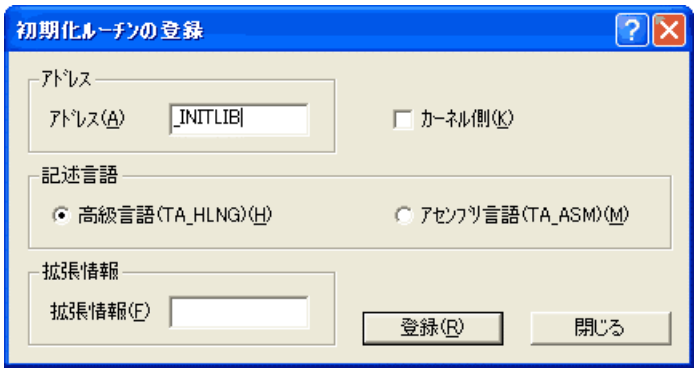


図10.20 [初期化ルーチンの登録]ダイアログボックス

[初期化ルーチンの登録]ダイアログボックスは、[初期化ルーチン]ページのポップアップメニューから[登録]を選択した時に、[初期化ルーチンの登録情報を変更]ダイアログボックスは、[初期化ルーチン]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

(1) [アドレス]

初期化ルーチンのアドレスを、C 言語シンボルまたは数値で指定してください。

(2) [カーネル側]

定義する初期化ルーチンをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(3) [記述言語]

初期化ルーチンが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(4) [拡張情報]

拡張情報は、初期化ルーチンにパラメータとして渡されます。C 言語シンボルまたは数値で指定してください。

(5) [登録]ボタン([初期化ルーチンの登録]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の初期化ルーチンの登録を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(6) [OK]ボタン([初期化ルーチンの登録情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(7) [閉じる]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.17 [タスク]ページ

本ページでは、タスクに関する項目を設定します。

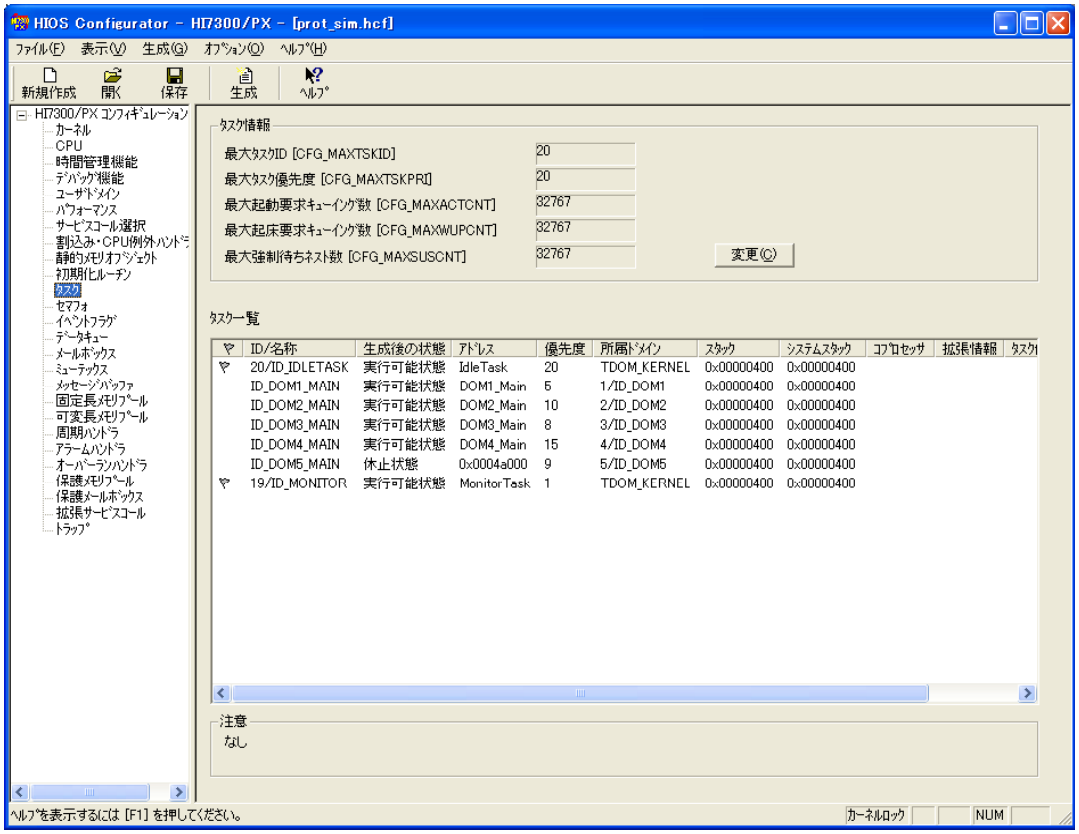


図10.21 [タスク]ページ

表 10.16に、[タスク]ページの項目を示します。

表10.16 [タスク]ページの項目

No	項目	CFG 名	リンク区分
1	**最大タスク ID**	CFG_MAXTSKID	**カーネル環境側**
2	**最大タスク優先度**	CFG_MAXTSKPRI	**カーネル側**
3	**最大起動要求キューイング数**	CFG_MAXACTCNT	**カーネル側**
4	**最大起床要求キューイング数**	CFG_MAXWUPCNT	**カーネル側**
5	**最大強制待ちネスト数**	CFG_MAXSUSCNT	**カーネル側**
6	**タスクの生成・タスク例外処理ルーチンの定義**	−	**カーネル側/カーネル環境側**

(1) [タスク情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[タスク情

報の変更]ダイアログボックスが開きます。

(a) [最大タスク ID[CFG_MAXTSKID]]

使用可能なタスク ID の範囲は、1～CFG_MAXTSKID となります。

(b) [最大タスク優先度[CFG_MAXTSKPRI]]

使用可能なタスク優先度の範囲は、1～[最大タスク優先度]となります。

(c) [最大起動要求キューイング数[CFG_MAXACTCNT]]

タスクに対する起動要求(act_tsk)の最大キューイング数です。

(d) [最大起床要求キューイング数[CFG_MAXWUPCNT]]

タスクに対する起床要求(wup_tsk)の最大キューイング数です。

(e) [最大強制待ちネスト数[CFG_MAXSUSCNT]]

タスクに対する強制待ち要求(sus_tsk)の最大ネスト数です。

(2) [タスク一覧]グループ

ここには、生成済みのタスクが表示されます。旗のアイコンは、そのタスクが[カーネル側]として生成されていることを示します。

タスクは、カーネル起動時に、「カーネル側」(旗のアイコンがあるもの)のタスクが本リストの上から順に生成され、次に「カーネル環境側」(旗のアイコンが無いもの)のタスクが本リストの上から順に生成されます。生成時に[生成後、起動(TA_ACT)]が指定された場合は、この順序に従って READY 状態となることに留意してください。

以下のポップアップメニューがあります。

- [生成]：タスクを生成するために、[タスクの生成]ダイアログボックスを開きます。
- [削除]：選択されたタスクを削除します。
- [変更]：選択されたタスクの設定を変更するために、[タスクの生成情報を変更]ダイアログボックスを開きます。
- [上へ]：選択されたタスクをひとつ上のタスクと順序を入れ替えます。
- [下へ]：選択されたタスクをひとつ下のタスクと順序を入れ替えます。

(3) [注意]

[サービスコール選択]ページの def_tex が選択されていない場合は、本ページでのタスク例外処理ルーチンの定義はすべて無効となります。また、タスク例外処理ルーチンは定義できなくなります。

この場合、[注意]に表 10.17に示すメッセージが表示されます。

表10.17 [タスク]ページの[注意]

条件	表示メッセージ
def_tex が未選択	def_tex が組み込まれない設定になっているので、ここでのタスク例外処理ルーチンの定義指定は無視されます。

10.7.18 [タスク情報の変更]ダイアログボックス

図10.22 [タスク情報の変更]ダイアログボックス

本ダイアログボックスは、[タスク]ページの[変更]ボタンを押したときに開きます。

(1) [最大タスク ID[CFG_MAXTSKID]]

1～CFG_MAXTSKID の範囲のタスク ID を使用できます。指定できるのは、1～32767 の整数です。

[生成するタスクの最大値に自動設定]をチェックすると、[タスク]ページで生成するタスクに応じて、コンフィギュレータが自動計算します。

(2) [最大タスク優先度[CFG_MAXTSKPRI]]

1～CFG_MAXTSKPRI のタスク優先度を使用できます。選択できるのは、1～255 の整数です。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TMAX_TPRI 255    /* 255 を選択した場合 */
```

(3) [最大起動要求キューイング数[CFG_MAXACTCNT]]

act_tsk, iact_tsk サービスコールによる起動要求のキューイングの最大数を指定します。指定できるのは、1～32767 の整数です。act_tsk, iact_tsk サービスコールで、対象タスクが既にこの数だけ起動要求がキューイングされている場合は、act_tsk, iact_tsk サービスコールは E_QOVR エラーとなります。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TMAX_ACTCNT 32767    /* 32767 を指定した場合 */
```

(4) [最大起床要求キューイング数[CFG_MAXWUPCNT]]

wup_tsk, iwup_tsk サービスコールによる起床要求のキューイングの最大数を指定します。指定できるのは、1～32767 の整数です。wup_tsk, iwup_tsk サービスコールで、対象タスクが既にこの数だけ起床要求がキューイングされている場合は、wup_tsk, iwup_tsk サービスコールは E_QOVR エラーとなります。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TMAX_WUPCNT 32767    /* 32767 を指定した場合 */
```

(5) [最大強制待ちネスト数[CFG_MAXSUSCNT]]

sus_tsk, isus_tsk サービスコールによる強制待ち要求のネストの最大数を指定します。指定できるのは、1～32767 の整数です。sus_tsk, isus_tsk サービスコールで、対象タスクが既にこの数だけ強制待ち要求がネストしている場合は、sus_tsk, isus_tsk サービスコールは E_QOVR エラーとなります。

本設定に対し、kernel_macro.h に以下のステートメントが出力されます。

```
#define TMAX_SUSCNT 32767    /* 32767 を指定した場合 */
```

(6) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(7) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.19 [タスクの生成]ダイアログボックス、[タスクの生成情報を変更]ダイアログボックス

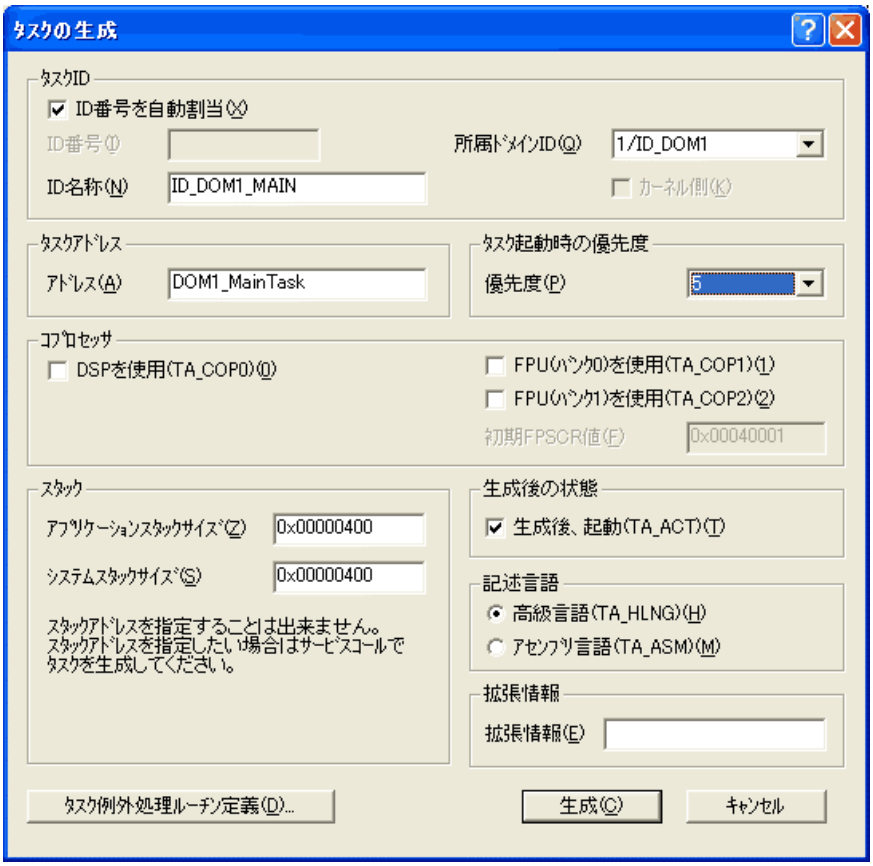


図10.23 [タスクの生成]ダイアログボックス

[タスクの生成]ダイアログボックスは、[タスク]ページのポップアップメニューから[生成]を選択した時に、[タスクの生成情報を変更]ダイアログボックスは、[タスク]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、タスクは cre_tsk, acre_tsk で動的に定義することもできます。

(1) [タスク ID、所属ドメイン]グループ

(a) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(b) [ID 番号]

タスク ID を数値で入力してください。指定できる値は、1～CFG_MAXTSKID の範囲です。ただし、

[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(c) [ID 名称]

ID 名称を指定してください。

[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(d) [所属ドメイン ID]

所属させるドメインを選択してください。TDOM_KERNEL(カーネルドメイン)またはドメイン ID が 1～31 のユーザドメインを選択できます。

(e) [カーネル側]

生成するタスクをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(2) [タスクアドレス]グループ

タスクの開始アドレスを数値または C 言語シンボルで指定してください。

(3) [タスク起動時の優先度] グループ

タスクの起動時の優先度を選択してください。

(4) [コプロセッサ] グループ

(a) [DSP を使用(TA_COP0)]、[FPU(バンク 0)を使用(TA_COP1)]、[FPU(バンク 1)を使用(TA_COP2)]

TA_COP0 は、[CPU]ページで CFG_DSP をチェックした場合のみ有効です。DSP を使用した演算を行う場合にチェックしてください。

TA_COP1, TA_COP2 は、[CPU]ページで CFG_FPU をチェックした場合のみ有効です。通常の FPU 演算を行う場合は TA_COP1 のみをチェックしてください。マトリックス演算など、FPU の両バンクを使用する場合は、両方をチェックしてください。TA_COP1 がチェック無しで TA_COP2 がチェック有り、という設定はできません。

また、TA_COP0 と TA_COP1,TA_COP2 を同時にチェックすることはできません。

(b) [初期 FPSCR 値]

初期 FPSCR 値は、TA_COP1, TA_COP2 のいずれかをチェックをした場合のみ意味を持ちます。0～0xffffffff の整数を指定できます。以下の節を参考に指定してください。

■関連ページ　「15. FPU に関する注意事項」

(5) [スタック] グループ

それぞれ、スタックサイズとシステムスタックサイズ(バイト)を数値で指定してください。指定できるのは、それぞれ 0x20000000 以下の正の整数です。

スタックは、システムプールまたはリソースプールから割り当てられます。指定したサイズに対してシステムプールまたはリソースプールが不足する場合には、その旨がエラーメッセージとして報告

されます。
　なお、サービスコールでタスクを生成する場合は、アプリケーション側で確保した領域をスタックとして使用する指定ができますが、コンフィギュレータではこの指定はできません。
　ここで指定するサイズは、cre_tsk サービスコールで指定する stksz と sstksz に対応します。詳細は、下記を参照してください。

■関連ページ　「6.7.1 タスクの生成(cre_tsk, icre_tsk, acre_tsk, iacre_tsk)」

(6) [生成後の状態] グループ

　生成後に起動したい場合は、TA_ACT をチェックしてください。

(7) [記述言語] グループ

　タスクが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(8) [拡張情報]グループ

　C 言語シンボルまたは数値で指定してください。

(9) [タスク例外処理ルーチン定義]ボタン

　本ダイアログで生成するタスクに対するタスク例外処理ルーチンを定義するために、[タスク例外処理ルーチンの定義]ダイアログボックスを開きます。タスク例外処理ルーチンの定義を解除したい場合も、このボタンを押してください。

(10)[生成]ボタン([タスクの生成]ダイアログボックスのみ)

　本ダイアログボックスの設定内容を有効とします。そして、次のタスクの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(11)[OK]ボタン([タスクの生成情報を変更]ダイアログボックスのみ)

　本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(12)[キャンセル]ボタン

　本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.20 [タスク例外処理ルーチンの定義]ダイアログボックス

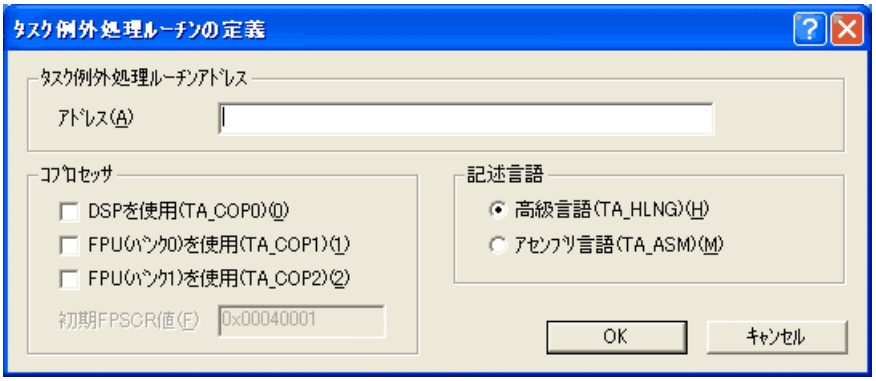


図10.24 [タスク例外処理ルーチンの定義]ダイアログボックス

[タスク例外処理ルーチンの定義]ダイアログボックスは、[タスクの生成]ダイアログボックスまたは[タスクの生成情報を変更]ダイアログボックスの[タスク例外処理ルーチン定義]ボタンを押した時に開きます。

本ダイアログボックスのカーネル側/カーネル環境側の区別は、[タスクの生成]ダイアログボックスまたは[タスクの生成情報を変更]ダイアログボックスの[カーネル側]チェックボックスと同じになります。

なお、タスク例外処理ルーチンは def_tex で動的に定義することもできます。

(1) [アドレス]

タスク例外処理ルーチンの開始アドレスを、数値または C 言語シンボルで指定してください。

空欄にすると、タスク例外処理ルーチンの定義を解除します。

(2) [DSP を使用(TA_COP0)]、[FPU(バンク 0)を使用(TA_COP1)]、[FPU(バンク 1)を使用(TA_COP2)]

TA_COP0 は、[CPU]ページで CFG_DSP をチェックした場合のみ有効です。DSP を使用した演算を行う場合にチェックしてください。

TA_COP1, TA_COP2 は、[CPU]ページで CFG_FPU をチェックした場合のみ有効です。通常の FPU 演算を行う場合は TA_COP1 のみをチェックしてください。マトリックス演算など、FPU の両バンクを使用する場合は、両方をチェックしてください。TA_COP1 がチェック無しで TA_COP2 がチェック有り、という設定はできません。

また、TA_COP0 と TA_COP1,TA_COP2 を同時にチェックすることはできません。

(3) [初期 FPSCR 値]

初期 FPSCR 値は、TA_COP1, TA_COP2 のいずれかをチェックをした場合のみ意味を持ちます。0～0xffffffff の整数を指定できます。以下の節を参考に指定してください。

■関連ページ 「15. FPU に関する注意事項」

(4) [記述言語]

タスク例外処理ルーチンが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(5) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じ、元の[タスクの生成]または[タスクの生成情報を変更]ダイアログボックスに戻ります。

(6) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じ、元の[タスクの生成]または[タスクの生成情報を変更]ダイアログボックスに戻ります。

10.7.21 [セマフォ]ページ

本ページでは、セマフォに関する項目を設定します。

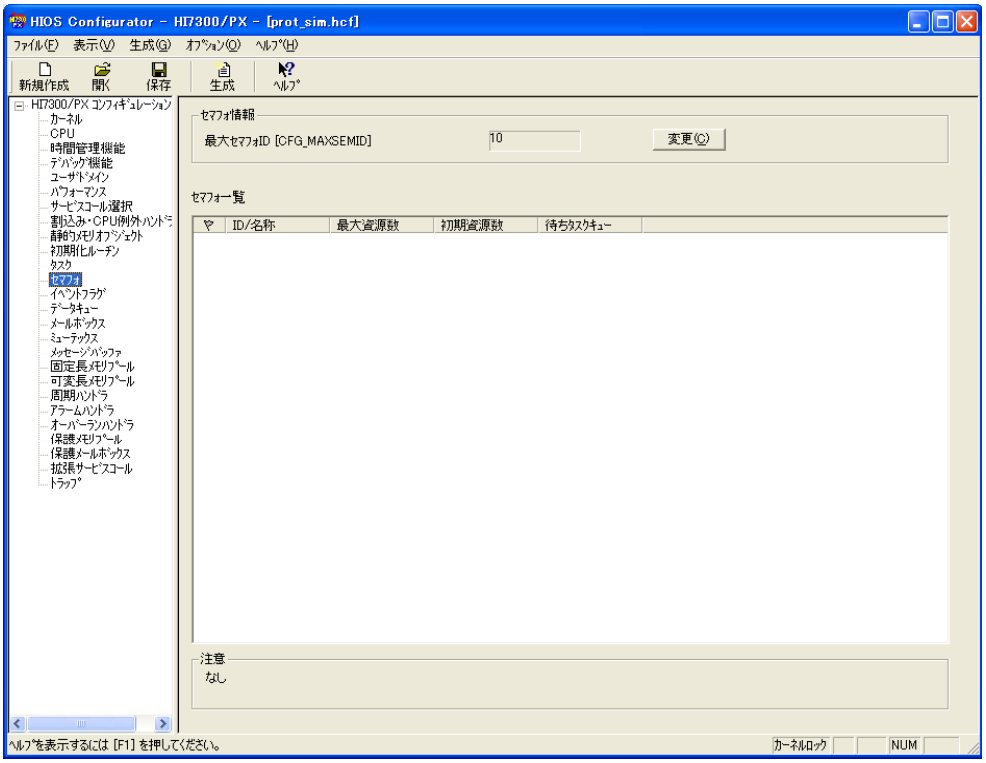


図10.25 [セマフォ]ページ

表 10.18に、[セマフォ]ページの項目を示します。

表10.18 [セマフォ]ページの項目

No	項目	CFG 名	リンク区分
1	最大セマフォ ID	CFG_MAXSEMID	カーネル環境側
2	セマフォの生成	−	カーネル側/カーネル環境側

(1) [セマフォ情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[セマフォ情報の変更]ダイアログボックスが開きます。

(a) 最大セマフォ ID[CFG_MAXSEMID]

使用可能なセマフォ ID の範囲は、1～CFG_MAXSEMID となります。

(2) [セマフォ一覧] グループ

ここには、生成済みのセマフォが表示されます。旗のアイコンは、そのセマフォが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成]：セマフォを生成するために、[セマフォの生成]ダイアログボックスを開きます。
- [削除]：選択されたセマフォを削除します。
- [変更]：選択されたセマフォの設定を変更するために、[セマフォの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_sem が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.19に示すメッセージが表示されます。

表10.19 [セマフォ]ページの[注意]

条件	表示メッセージ
cre_sem が未選択	cre_sem が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.22 [セマフォ情報の変更]ダイアログボックス

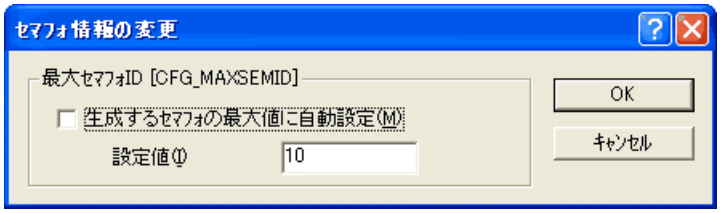


図10.26 [セマフォ情報の変更]ダイアログボックス

本ダイアログボックスは、[セマフォ]ページの[変更]ボタンを押したときに開きます。

(1) [最大セマフォ ID[CFG_MAXSEMID]]

1～CFG_MAXSEMID の範囲のセマフォ ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、セマフォを使用できなくなりますが、セマフォを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するセマフォの最大値に自動設定]をチェックすると、[セマフォ]ページで生成するセマフォに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.23 [セマフォの生成]ダイアログボックス、[セマフォの生成情報を変更]ダイアログボックス

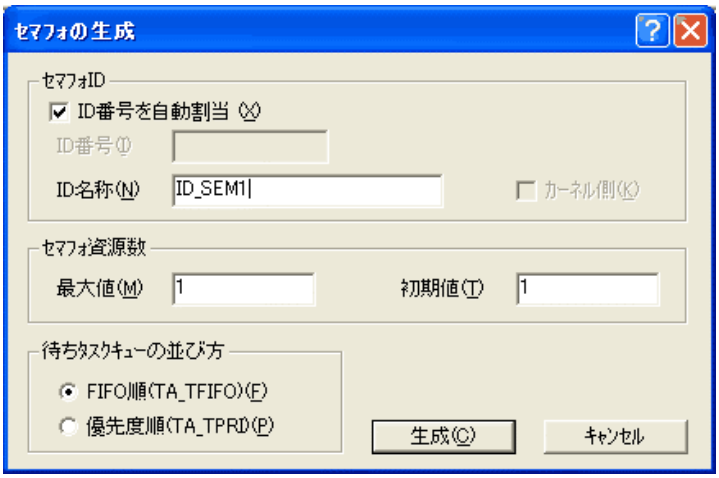


図10.27 [セマフォの生成]ダイアログボックス

[セマフォの生成]ダイアログボックスは、[セマフォ]ページのポップアップメニューから[生成]を選択した時に、[セマフォの生成情報を変更]ダイアログボックスは、[セマフォ]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、セマフォは cre_sem, acre_sem で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

セマフォ ID を数値で入力してください。指定できる値は、1～CFG_MAXSEMID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するセマフォをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [セマフォ資源数]の[最大値]

セマフォの資源数の最大値を指定してください。指定できるのは、1～65535 の整数です。

(6) [セマフォ資源数]の[初期値]

セマフォの資源数の初期値を指定してください。指定できるのは、0～[セマフォ資源数の最大値]の整数です。

(7) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(8) [生成]ボタン([セマフォの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のセマフォの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(9) [OK]ボタン([セマフォの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(10)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.24 [イベントフラグ]ページ

本ページでは、イベントフラグに関する項目を設定します。

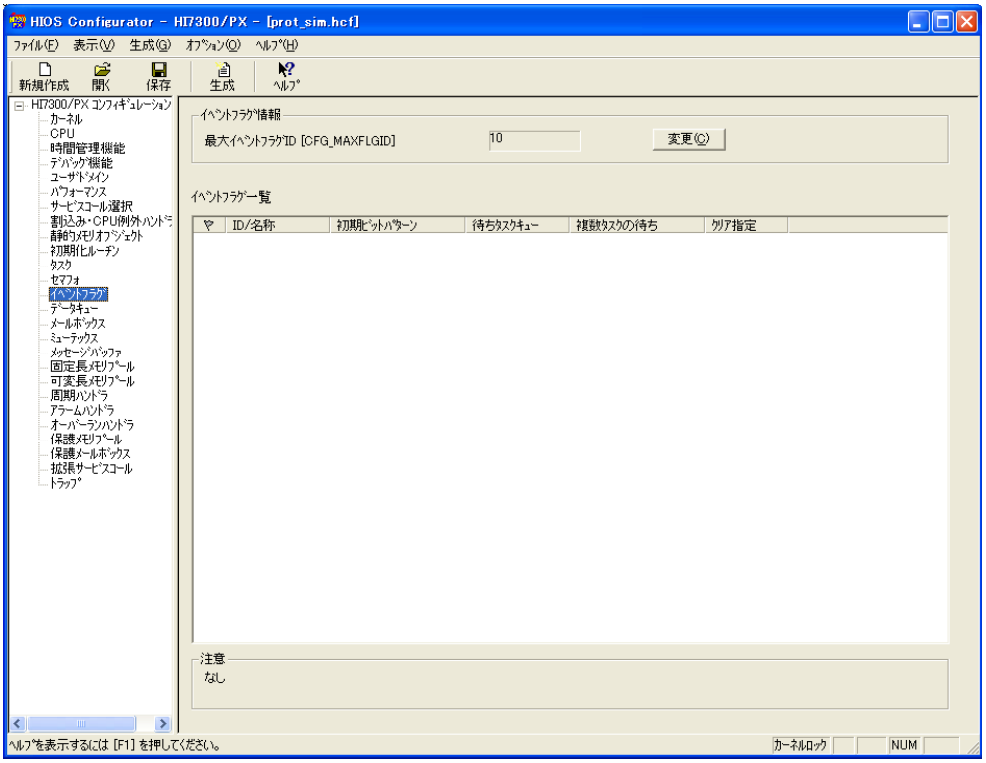


図10.28 [イベントフラグ]ページ

表 10.20に、[イベントフラグ]ページの項目を示します。

表10.20 [イベントフラグ]ページの項目

No	項目	CFG 名	リンク区分
1	**最大イベントフラグ ID**	CFG_MAXFLGID	**カーネル側**
2	**イベントフラグの生成**	−	**カーネル側/カーネル環境側**

(1) [イベントフラグ情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[イベントフラグ情報の変更]ダイアログボックスが開きます。

(a) 最大イベントフラグ ID[CFG_MAXFLGID]

使用可能なイベントフラグ ID の範囲は、1～CFG_MAXFLGID となります。

(2) [イベントフラグ一覧]グループ

ここには、生成済みのイベントフラグが表示されます。旗のアイコンは、そのイベントフラグが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : イベントフラグを生成するために、[イベントフラグの生成]ダイアログボックスを開きます。
- [削除] : 選択されたイベントフラグを削除します。
- [変更] : 選択されたイベントフラグの設定を変更するために、[イベントフラグの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_flg が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.21に示すメッセージが表示されます。

表10.21 [イベントフラグ]ページの[注意]

条件	表示メッセージ
cre_flg が未選択	cre_flg が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.25 [イベントフラグ情報の変更]ダイアログボックス

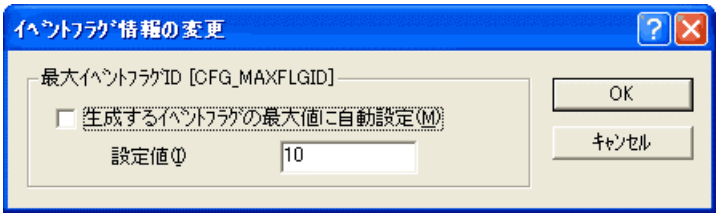


図10.29 [イベントフラグ情報の変更]ダイアログボックス

本ダイアログボックスは、[イベントフラグ]ページの[変更]ボタンを押したときに開きます。

(1) [最大イベントフラグ ID[CFG_MAXFLGID]]

1～CFG_MAXFLGID の範囲のイベントフラグ ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、イベントフラグを使用できなくなりますが、イベントフラグを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するイベントフラグの最大値に自動設定]をチェックすると、[イベントフラグ]ページで生成するイベントフラグに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.26 [イベントフラグの生成]ダイアログボックス、[イベントフラグの生成情報を変更]ダイアログボックス

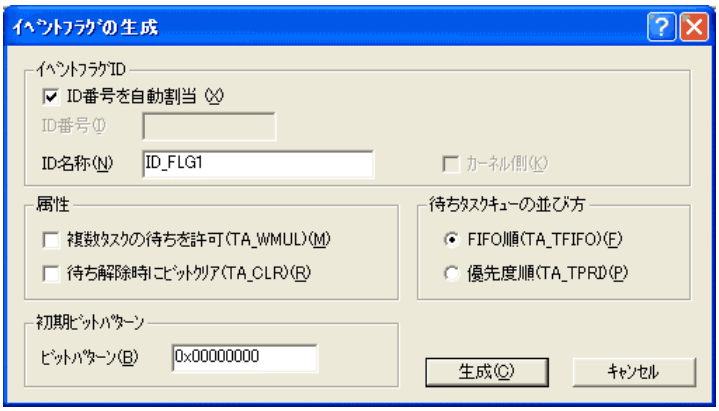


図10.30 [イベントフラグの生成]ダイアログボックス

[イベントフラグの生成]ダイアログボックスは、[イベントフラグ]ページのポップアップメニューから[生成]を選択した時に、[イベントフラグの生成情報を変更]ダイアログボックスは、[イベントフラグ]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、イベントフラグは cre_flg, acre_flg で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

イベントフラグ ID を数値で入力してください。指定できる値は、1～CFG_MAXFLGID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するイベントフラグをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [複数タスクの待ちを許可(TA_WMUL)]

複数のタスクがイベントフラグ待ちになる使い方をする場合はチェックしてください。

(6) [待ち解除時にビットクリア(TA_CLR)]

これをチェックすると、wai_flg, twai_flg, pol_flg サービスコールで E_OK が返る時に、イベントフラグが 0 クリアされます。

(7) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(8) [初期ビットパターン]

イベントフラグの初期値を 0～0xffffffff の整数で指定してください。

(9) [生成]ボタン([イベントフラグの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のイベントフラグの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(10)[OK]ボタン([イベントフラグの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(11)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.27 [データキュー]ページ

本ページでは、データキューに関する項目を設定します。

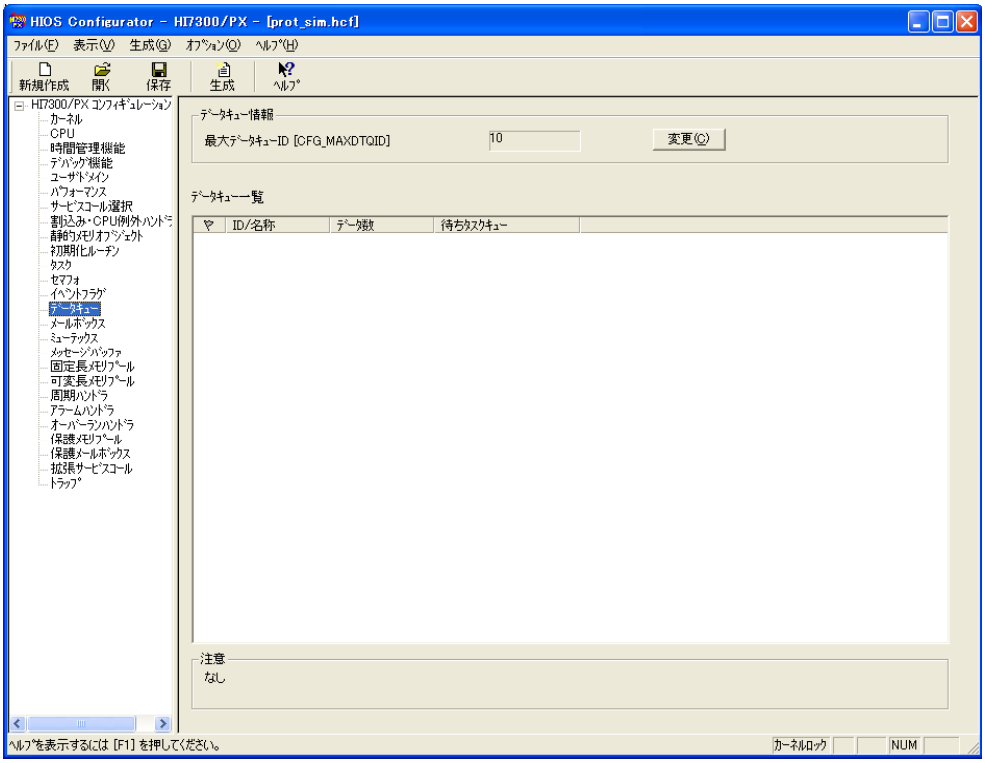


図10.31 [データキュー]ページ

表 10.22に、[データキュー]ページの項目を示します。

表10.22 [データキュー]ページの項目

No	項目	CFG 名	リンク区分
1	最大データキューID	CFG_MAXDTQID	カーネル環境側
2	データキューの生成	−	カーネル側/カーネル環境側

(1) [データキュー情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[データキュー情報の変更]ダイアログボックスが開きます。

(a) [最大データキューID[CFG_MAXDTQID]]

使用可能なデータキューID の範囲は、1～CFG_MAXDTQID となります。

(2) [データキュー一覧]グループ

ここには、生成済みのデータキューが表示されます。旗のアイコンは、そのデータキューが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : データキューを生成するために、[データキューの生成]ダイアログボックスを開きます。
- [削除] : 選択されたデータキューを削除します。
- [変更] : 選択されたデータキューの設定を変更するために、[データキューの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_dtq が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.23に示すメッセージが表示されます。

表10.23 [データキュー]ページの[注意]

条件	表示メッセージ
cre_dtq が未選択	cre_dtq が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.28 [データキュー情報の変更]ダイアログボックス

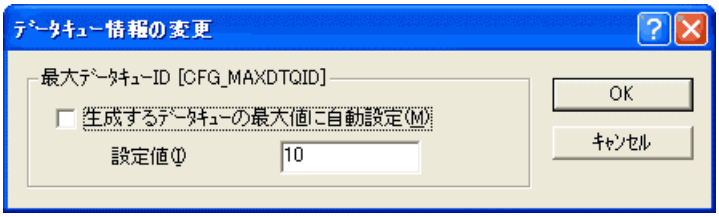


図10.32 [データキュー情報の変更]ダイアログボックス

本ダイアログボックスは、[データキュー]ページの[変更]ボタンを押したときに開きます。

(1) [最大データキューID[CFG_MAXDTQID]]

1～CFG_MAXDTQID の範囲のデータキューID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、データキューを使用できなくなりますが、データキューを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するデータキューの最大値に自動設定]をチェックすると、[データキュー]ページで生成するデータキューに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.29 [データキューの生成]ダイアログボックス、[データキューの生成情報を変更]ダイアログボックス

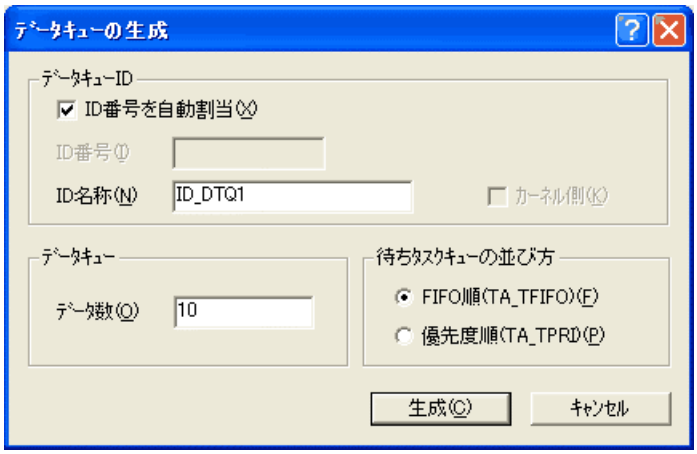


図10.33 [データキューの生成]ダイアログボックス

[データキューの生成]ダイアログボックスは、[データキュー]ページのポップアップメニューから[生成]を選択した時に、[データキューの生成情報を変更]ダイアログボックスは、[データキュー]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、データキューは cre_dtq, acre_dtq で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

データキューID を数値で入力してください。指定できる値は、1～CFG_MAXDTQID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するデータキューをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [データ数]

データキューに格納可能なデータ数を指定してください。0 を指定することもできます。

(6) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(7) [生成]ボタン([データキューの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のデータキューの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(8) [OK]ボタン([データキューの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(9) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.30 [メールボックス]ページ

本ページでは、メールボックスに関する項目を設定します。

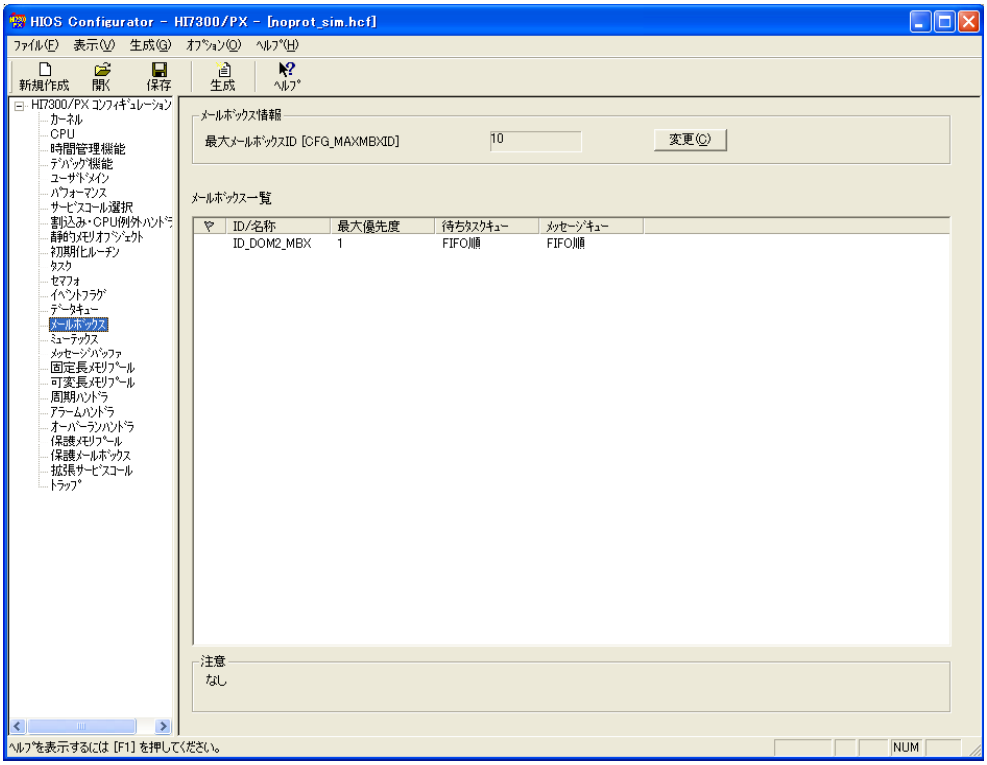


図10.34 [メールボックス]ページ

表 10.24に、[メールボックス]ページの項目を示します。

表10.24 [メールボックス]ページの項目

No	項目	CFG 名	リンク区分
1	最大メールボックス ID	CFG_MAXMBXID	カーネル環境側
2	メールボックスの生成	−	カーネル側/カーネル環境側

(1) [メールボックス情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[メールボックス情報の変更]ダイアログボックスが開きます。

(a) [最大メールボックス ID[CFG_MAXMBXID]]

使用可能なメールボックス ID の範囲は、1～CFG_MAXMBXID となります。

(2) [メールボックス一覧]グループ

ここには、生成済みのメールボックスが表示されます。旗のアイコンは、そのメールボックスが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : メールボックスを生成するために、[メールボックスの生成]ダイアログボックスを開きます。
- [削除] : 選択されたメールボックスを削除します。
- [変更] : 選択されたメールボックスの設定を変更するために、[メールボックスの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_mbx が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.25に示すメッセージが表示されます。

表10.25 [メールボックス]ページの[注意]

条件	表示メッセージ
cre_mbx が未選択	cre_mbx が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.31 [メールボックス情報の変更]ダイアログボックス

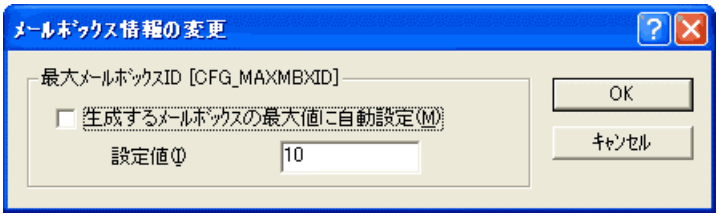


図10.35 [メールボックス情報の変更]ダイアログボックス

本ダイアログボックスは、[メールボックス]ページの[変更]ボタンを押したときに開きます。

(1) [最大メールボックス ID[CFG_MAXMBXID]]

1～CFG_MAXMBXID の範囲のメールボックス ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、メールボックスを使用できなくなりますが、メールボックスを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するメールボックスの最大値に自動設定]をチェックすると、[メールボックス]ページで生成するメールボックスに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.32 [メールボックスの生成]ダイアログボックス、[メールボックスの生成情報を変更]ダイアログボックス

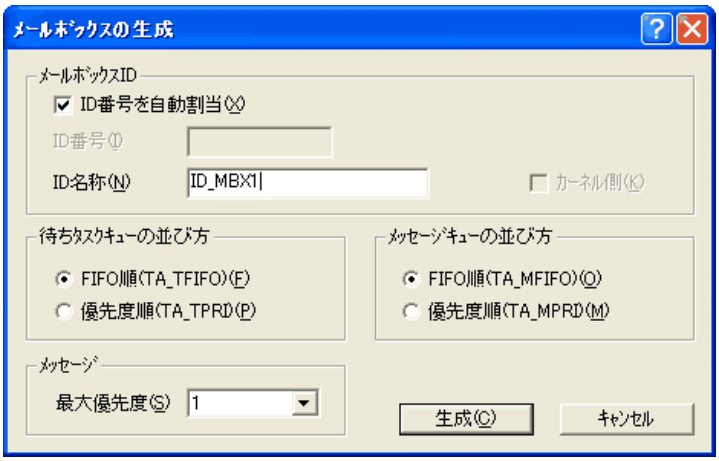


図10.36 [メールボックスの生成]ダイアログボックス

[メールボックスの生成]ダイアログボックスは、[メールボックス]ページのポップアップメニューから[生成]を選択した時に、[メールボックスの生成情報を変更]ダイアログボックスは、[メールボックス]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、メールボックスは cre_mbx, acre_mbx で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

メールボックス ID を数値で入力してください。指定できる値は、1～CFG_MAXMBXID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するメールボックスをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(6) [メッセージキューの並び方]

メッセージキューの並び方として、FIFO 順または優先度順を選択してください。

(7) [最大優先度]

[メッセージキューの並び方]として優先度順を選択した場合は、メッセージの最大優先度を選択してください。選択できるのは、1～CFG_MAXMSGPRI です。

[メッセージキューの並び方]として FIFO 順を選択した場合は、本項目は意味を持ちません。

(8) [生成]ボタン([メールボックスの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のメールボックスの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(9) [OK]ボタン([メールボックスの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(10)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.33 [ミューテックス]ページ

本ページでは、ミューテックスに関する項目を設定します。

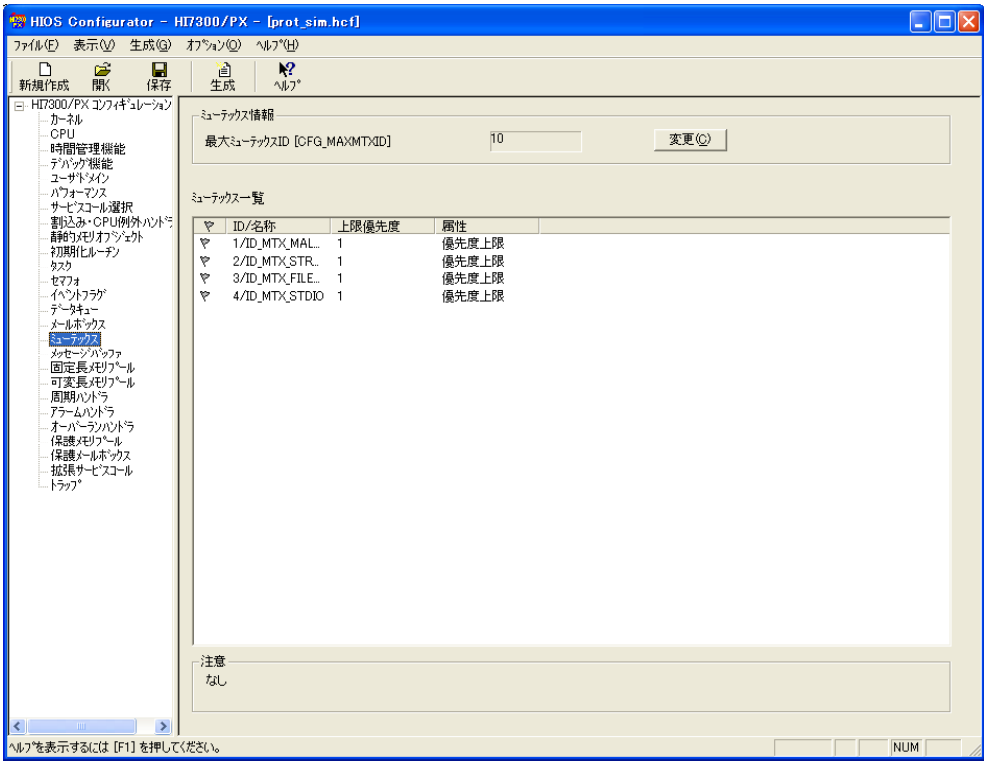


図10.37 [ミューテックス]ページ

表 10.26に、[ミューテックス]ページの項目を示します。

表10.26 [ミューテックス]ページの項目

No	項目	CFG 名	リンク区分
1	最大ミューテックス ID	CFG_MAXMTXID	カーネル環境側
2	ミューテックスの生成	−	カーネル側/カーネル環境側

(1) [ミューテックス情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[ミューテックス情報の変更]ダイアログボックスが開きます。

(a) [最大ミューテックス ID[CFG_MAXMTXID]]

使用可能なミューテックス ID の範囲は、1～CFG_MAXMTXID となります。

(2) [ミューテックス一覧]グループ

ここには、生成済みのミューテックスが表示されます。旗のアイコンは、そのミューテックスが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : ミューテックスを生成するために、[ミューテックスの生成]ダイアログボックスを開きます。
- [削除] : 選択されたミューテックスを削除します。
- [変更] : 選択されたミューテックスの設定を変更するために、[ミューテックスの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_mtx が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.27に示すメッセージが表示されます。

表10.27 [ミューテックス]ページの[注意]

条件	表示メッセージ
cre_mtx が未選択	cre_mtx が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.34 [ミューテックス情報の変更]ダイアログボックス

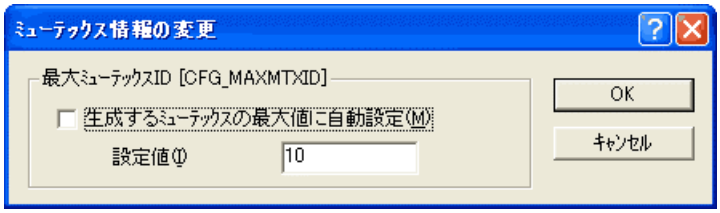


図10.38 [ミューテックス情報の変更]ダイアログボックス

本ダイアログボックスは、[ミューテックス]ページの[変更]ボタンを押したときに開きます。

(1) 最大ミューテックス ID[CFG_MAXMTXID]

1～CFG_MAXMTXID の範囲のミューテックス ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、ミューテックスを使用できなくなりますが、ミューテックスを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するミューテックスの最大値に自動設定]をチェックすると、[ミューテックス]ページで生成するミューテックスに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.35 [ミューテックスの生成]ダイアログボックス、[ミューテックスの生成情報を変更]ダイアログボックス

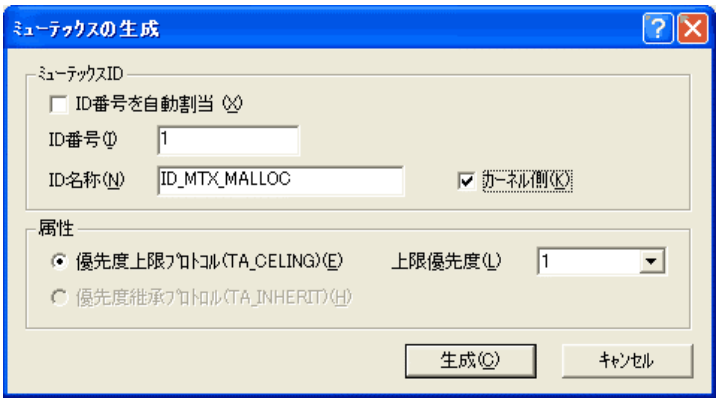


図10.39 [ミューテックスの生成]ダイアログボックス

[ミューテックスの生成]ダイアログボックスは、[ミューテックス]ページのポップアップメニューから[生成]を選択した時に、[ミューテックスの生成情報を変更]ダイアログボックスは、[ミューテックス]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、ミューテックスは cre_mtx, acre_mtx で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

ミューテックス ID を数値で入力してください。指定できる値は、1～CFG_MAXMTXID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するミューテックスをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [優先度上限プロトコル]

属性として、優先度上限プロトコルのみを選択できます。

(6) [上限優先度]

上限優先度を選択します。選択できるのは、1～CFG_MAXTSKPRI です。

(7) [生成]ボタン([ミューテックスの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のミューテックスの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(8) [OK]ボタン([ミューテックスの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(9) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.36 [メッセージバッファ]ページ

本ページでは、メッセージバッファに関する項目を設定します。

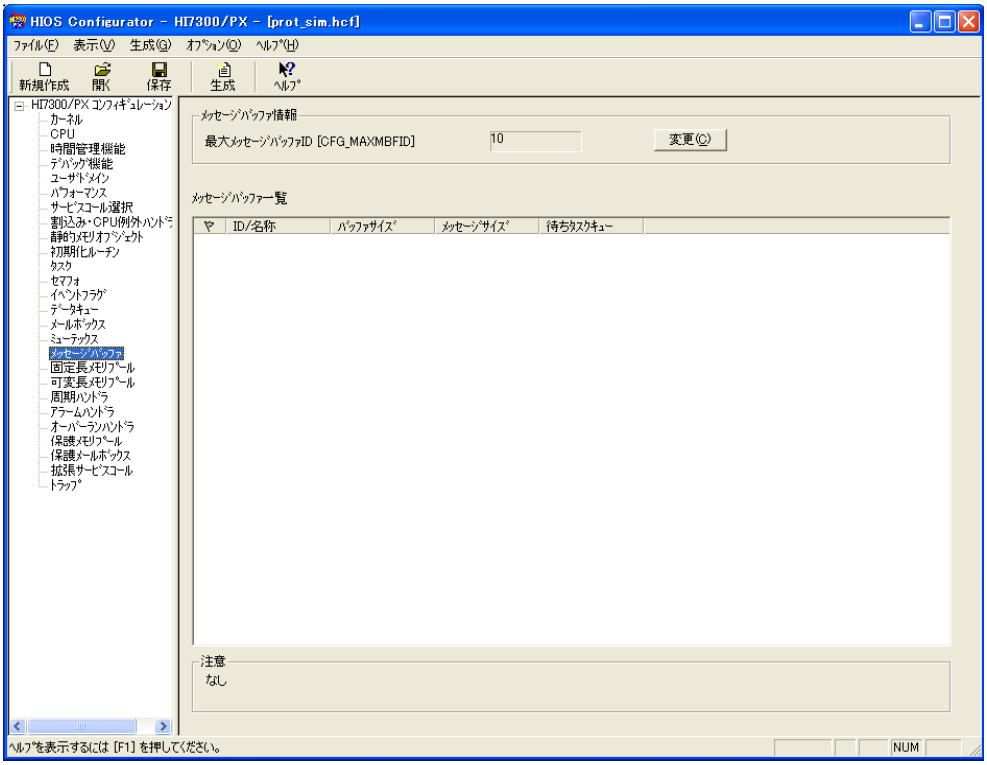


図10.40 [メッセージバッファ]ページ

表 10.28に、[メッセージバッファ]ページの項目を示します。

表10.28 [メッセージバッファ]ページの項目

No	項目	CFG 名	リンク区分
1	最大メッセージバッファ ID	CFG_MAXMBFID	カーネル環境側
2	メッセージバッファの生成	−	カーネル側/カーネル環境側

(1) [メッセージバッファ情報]

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[メッセージバッファ情報の変更]ダイアログボックスが開きます。

(a) [最大メッセージバッファ ID[CFG_MAXMBFID]]

使用可能なメッセージバッファ ID の範囲は、1～CFG_MAXMBFID となります。

(2) [メッセージバッファ一覧]

ここには、生成済みのメッセージバッファが表示されます。旗のアイコンは、そのメッセージバッファが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : メッセージバッファを生成するために、[メッセージバッファの生成]ダイアログボックスを開きます。
- [削除] : 選択されたメッセージバッファを削除します。
- [変更] : 選択されたメッセージバッファの設定を変更するために、[メッセージバッファの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_mbf が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.29に示すメッセージが表示されます。

表10.29 [メッセージバッファ]ページの[注意]

条件	表示メッセージ
cre_mbf が未選択	cre_mbf が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.37 [メッセージバッファ情報の変更]ダイアログボックス

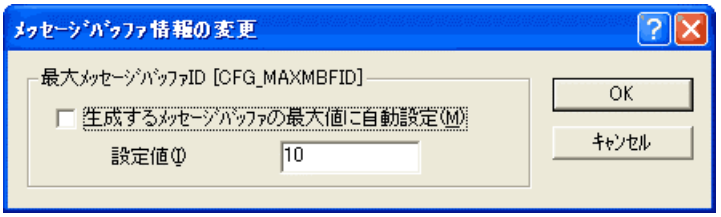


図10.41 [メッセージバッファ情報の変更]ダイアログボックス

本ダイアログボックスは、[メッセージバッファ]ページの[変更]ボタンを押したときに開きます。

(1) [最大メッセージバッファ ID[CFG_MAXMBFID]]

1～CFG_MAXMBFID の範囲のメッセージバッファ ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、メッセージバッファを使用できなくなりますが、メッセージバッファを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するメッセージバッファの最大値に自動設定]をチェックすると、[メッセージバッファ]ページで生成するメッセージバッファに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.38 [メッセージバッファの生成]ダイアログボックス、[メッセージバッファの生成情報を変更]ダイアログボックス

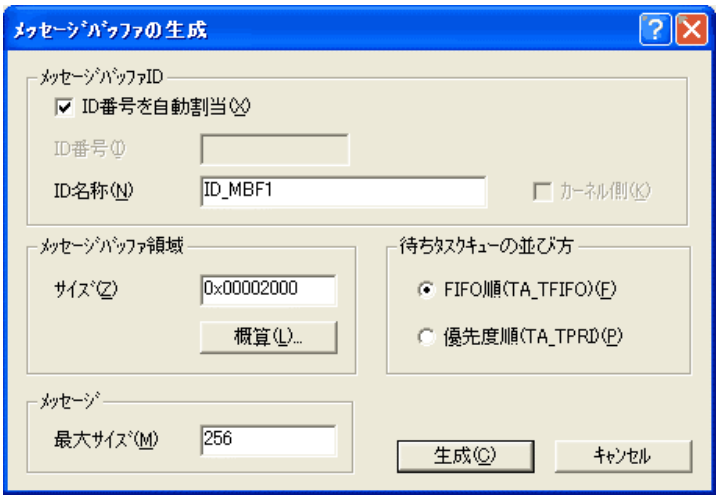


図10.42 [メッセージバッファの生成]ダイアログボックス

[メッセージバッファの生成]ダイアログボックスは、[メッセージバッファ]ページのポップアップメニューから[生成]を選択した時に、[メッセージバッファの生成情報を変更]ダイアログボックスは、[メッセージバッファ]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、メッセージバッファは cre_mbf, acre_mbf で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

メッセージバッファ ID を数値で入力してください。指定できる値は、1～CFG_MAXMBFID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するメッセージバッファをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [メッセージバッファ領域のサイズ]

メッセージバッファのサイズを指定します。0～0x20000000 を指定でき、4 の倍数に切上げられます。

[概算]ボタンを押すと、[サイズ]の概算値を算出するための[メッセージバッファ領域サイズの概算]ダイアログボックスが開きます。

(6) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(7) [最大サイズ]

メッセージバッファに送信するメッセージの最大サイズを指定します。0～0x20000000 を指定でき、4 の倍数に切上げられます。

(8) [生成]ボタン([メッセージバッファの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次のメッセージバッファの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(9) [OK]ボタン([メッセージバッファの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(10)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.39 [メッセージバッファ領域サイズの概算]ダイアログボックス

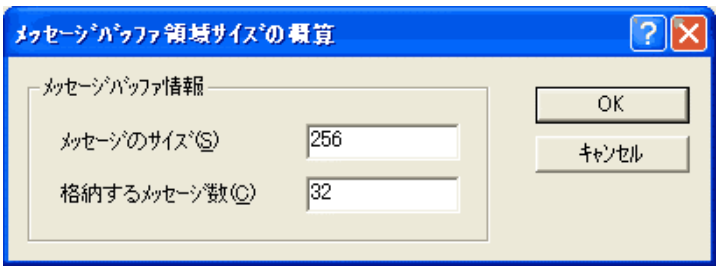


図10.43 [メッセージバッファ領域サイズの概算]ダイアログボックス

本ダイアログボックスは、[メッセージバッファの生成]または[メッセージバッファの生成情報を変更]ダイアログボックスの[概算]ボタンを押したときに開きます。

本ダイアログボックスは、[メッセージのサイズ]に指定したサイズのメッセージを[格納するメッセージ数]に指定した数だけ格納することができるメッセージバッファのサイズを算出します。具体的には、TSZ_MBFMB, TSZ_MBF マクロと同じ計算を行います。

[OK]ボタンを押すと、[メッセージバッファの生成]または[メッセージバッファの生成情報を変更]ダイアログボックスに戻り、このダイアログボックスの[サイズ]が本ダイアログボックスの計算結果に更新されます。

[キャンセル]ボタンを押すと、[メッセージバッファの生成]または[メッセージバッファの生成情報を変更]ダイアログボックスに戻ります。この時、このダイアログボックスの[サイズ]は更新されません。

10.7.40 [固定長メモリプール]ページ

本ページでは、固定長メモリプールに関する項目を設定します。

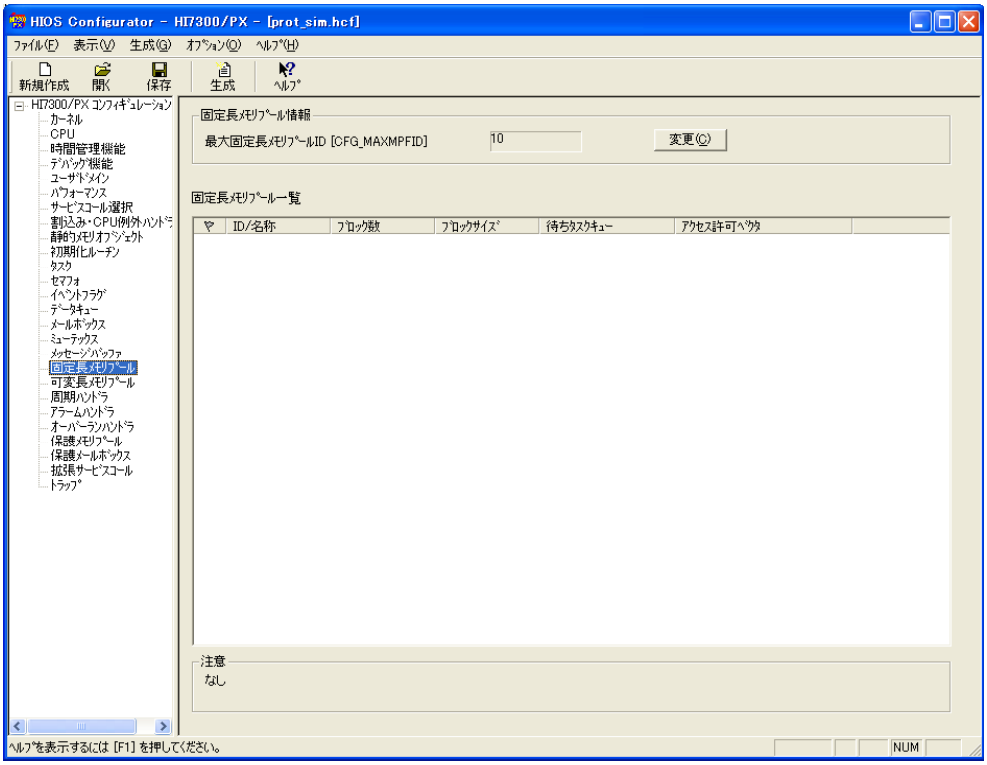


図10.44 [固定長メモリプール]ページ

表 10.30に、[固定長メモリプール]ページの項目を示します。

表10.30 [固定長メモリプール]ページの項目

No	項目	CFG 名	リンク区分
1	**最大固定長メモリプール ID**	CFG_MAXMPFID	**カーネル環境側**
2	**固定長メモリプールの生成**	−	**カーネル側/カーネル環境側**

(1) [固定長メモリプール情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[固定長メモリプール情報の変更]ダイアログボックスが開きます。

(a) [最大固定長メモリプール ID[CFG_MAXMPFID]]

使用可能な固定長メモリプール ID の範囲は、1～CFG_MAXMPFID となります。

(2) [固定長メモリプール一覧]グループ

ここには、生成済みの固定長メモリプールが表示されます。旗のアイコンは、その固定長メモリプールが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : 固定長メモリプールを生成するために、[固定長メモリプールの生成]ダイアログボックスを開きます。
- [削除] : 選択された固定長メモリプールを削除します。
- [変更] : 選択された固定長メモリプールの設定を変更するために、[固定長メモリプールの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_mpf が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.31に示すメッセージが表示されます。

表10.31 [固定長メモリプール]ページの[注意]

条件	表示メッセージ
cre_mpf が未選択	cre_mpf が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.41 [固定長メモリプール情報の変更]ダイアログボックス

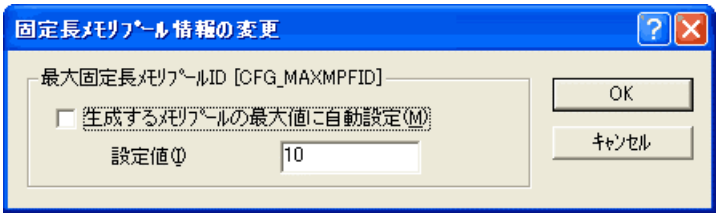


図10.45 [固定長メモリプール情報の変更]ダイアログボックス

本ダイアログボックスは、[固定長メモリプール]ページの[変更]ボタンを押したときに開きます。

(1) 最大固定長メモリプール ID[CFG_MAXMPFID]

1～CFG_MAXMPFID の範囲の固定長メモリプール ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、固定長メモリプールを使用できなくなりますが、固定長メモリプールを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するメモリプールの最大値に自動設定]をチェックすると、[固定長メモリプール]ページで生成する固定長メモリプールに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.42 [固定長メモリプールの生成]ダイアログボックス、[固定長メモリプールの生成情報を変更]ダイアログボックス

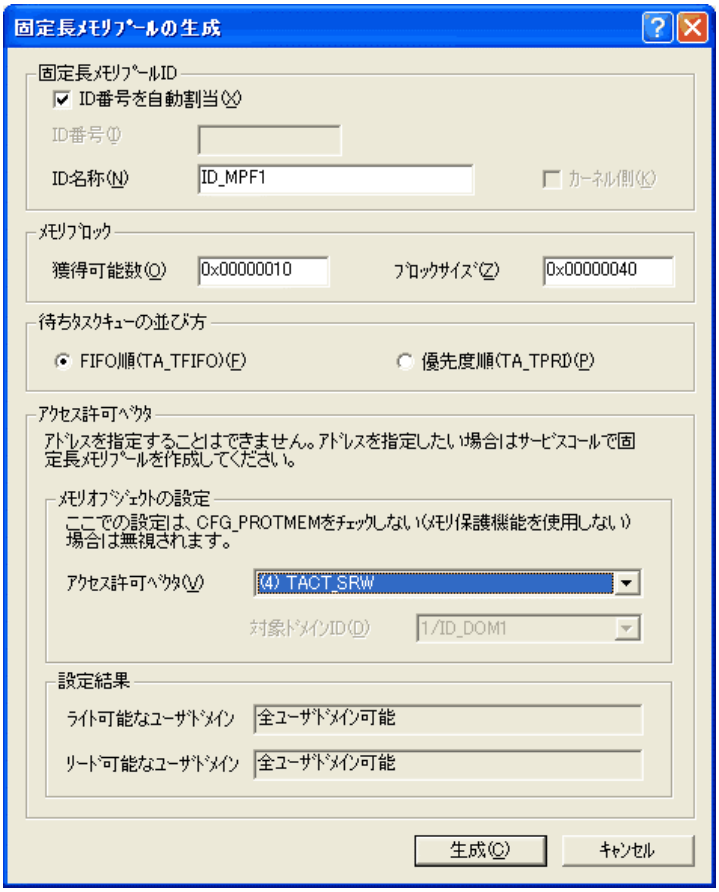


図10.46 [固定長メモリプールの生成]ダイアログボックス

[固定長メモリプールの生成]ダイアログボックスは、[固定長メモリプール]ページのポップアップメニューから[生成]を選択した時に、[固定長メモリプールの生成情報を変更]ダイアログボックスは、[固定長メモリプール]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、固定長メモリプールは cre_mpf, acre_mpf で動的に生成することもできます。これらのサービスコールで固定長メモリプールを生成する場合は、アプリケーション側で確保した領域をメモリプールとして使用する指定ができますが、コンフィギュレータではこの指定はできず、常にシステムプールから固定長メモリプールが割り付けられます。

(1) [固定長メモリプール ID]グループ

(a) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これを

チェックすると[カーネル側]は選択できなくなります。

(b) [ID 番号]

固定長メモリプール ID を数値で入力してください。指定できる値は、1～CFG_MAXMPFID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(c) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(d) [カーネル側]

生成する固定長メモリプールをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(2) [メモリブロック]グループ

[ブロックサイズ]に指定したサイズのメモリブロックを[獲得可能数]だけ獲得可能な固定長メモリプールを生成します。
[獲得可能数]には、1～0x08000000 を指定できます。
[ブロックサイズ]には、4～0x20000000 を指定でき、4 の倍数に切上げられます。

(3) [待ちタスクキューの並び方]グループ

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(4) [アクセス許可ベクタ]グループ

このグループの項目は、全て[カーネル]ページの CFG_PROTMEM をチェックした場合のみ、意味を持ちます。[アクセス許可ベクタ]は、以下の中から選択してください。

(1) TACT_KERNEL
(2) TACT_PRW(domid)
(3) TACT_PRO(domid)
(4) TACT_SRW
(5) TACT_SRO
(6) TACT_SRPW(domid)

(2),(3),(6)を選択した場合のみ[対象ドメイン ID]が有効となります。この場合、[対象ドメイン ID]は許可対象とするユーザドメイン ID を選択してください。

[設定結果]には、[アクセス許可ベクタ]、[対象ドメイン ID]の設定によって、どのユーザドメインからライトまたはリード可能かが表示されます。
表 10.32に、[設定結果]の表示内容を示します。
なお、メモリ属性は常に以下となります。

- TA_RW (リード・ライト可能)
- TA_CACHE (キャッシャブル)
- TA_WBACK (コピーバックモード)

表10.32　[設定結果]の表示内容

設定		[設定結果]の表示	
アクセス許可ベクタ	対象ドメイン	ライト可能なユーザドメイン	リード可能なユーザドメイン
TACT_KERNEL	無効	全ユーザドメイン不可	全ユーザドメイン不可
TACT_PRW(domid)	有効	対象ドメインのみ可能	対象ドメインのみ可能
TACT_PRO(domid)	有効	全ユーザドメイン不可	対象ドメインのみ可能
TACT_SRW	無効	全ユーザドメイン可能	全ユーザドメイン可能
TACT_SRO	無効	全ユーザドメイン不可	全ユーザドメイン可能
TACT_SRPW(domid)	有効	対象ドメインのみ可能	全ユーザドメイン可能

(5) [生成]ボタン([固定長メモリプールの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の固定長メモリプールの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(6) [OK]ボタン([固定長メモリプールの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(7) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.43 [可変長メモリプール]ページ

本ページでは、可変長メモリプールに関する項目を設定します。

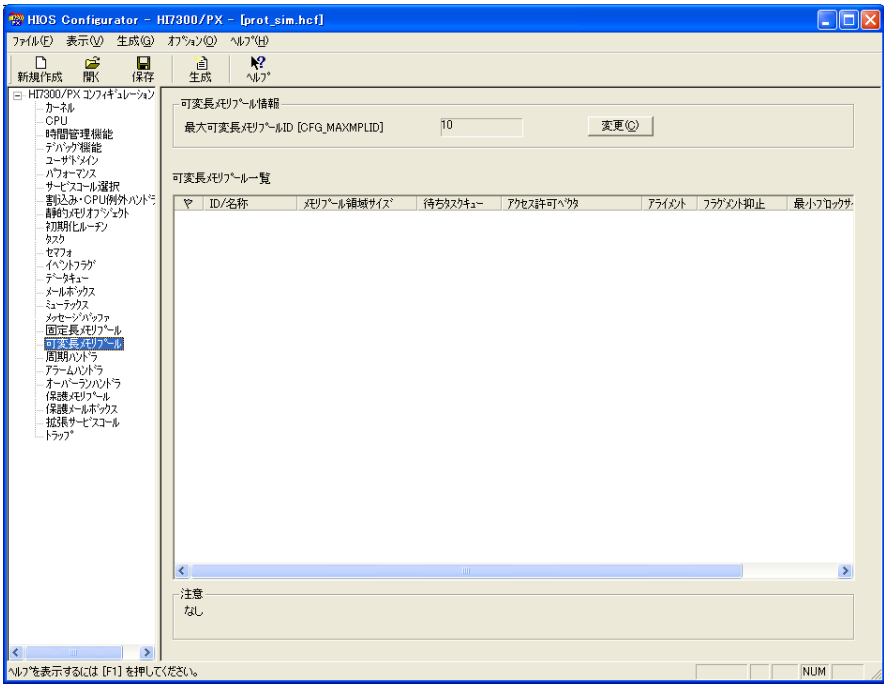


図10.47 [可変長メモリプール]ページ

表 10.33に、[可変長メモリプール]ページの項目を示します。

表10.33 [可変長メモリプール]ページの項目

No	項目	CFG 名	リンク区分
1	最大可変長メモリプール ID	CFG_MAXMPLID	カーネル環境側
2	可変長メモリプールの生成	−	カーネル側/カーネル環境側

(1) [可変長メモリプール情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[可変長メモリプール情報の変更]ダイアログボックスが開きます。

(a) [最大可変長メモリプール ID[CFG_MAXMPLID]]

使用可能な可変長メモリプール ID の範囲は、1～CFG_MAXMPLID となります。

(2) [可変長メモリプール一覧]グループ

ここには、生成済みの可変長メモリプールが表示されます。旗のアイコンは、その可変長メモリプー

ルが[カーネル側]として生成されていることを示します。
以下のポップアップメニューがあります。

- [生成] : 可変長メモリプールを生成するために、[可変長メモリプールの生成]ダイアログボックスを開きます。
- [削除] : 選択された可変長メモリプールを削除します。
- [変更] : 選択された可変長メモリプールの設定を変更するために、[可変長メモリプールの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_mpl が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。
この場合、[注意]に表 10.34に示すメッセージが表示されます。

表10.34 [可変長メモリプール]ページの[注意]

条件	表示メッセージ
cre_mpl が未選択	cre_mpl が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.44 [可変長メモリプール情報の変更]ダイアログボックス

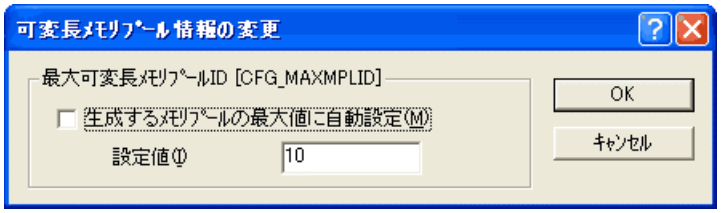


図10.48 [可変長メモリプール情報の変更]ダイアログボックス

本ダイアログボックスは、[可変長メモリプール]ページの[変更]ボタンを押したときに開きます。

(1) [最大可変長メモリプール ID[CFG_MAXMPLID]]

1～CFG_MAXMPLID の範囲の可変長メモリプール ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、可変長メモリプールを使用できなくなりますが、可変長メモリプールを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するメモリプールの最大値に自動設定]をチェックすると、[可変長メモリプール]ページで生成する可変長メモリプールに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.45 [可変長メモリプールの生成]ダイアログボックス、[可変長メモリプールの生成情報を変更]ダイアログボックス

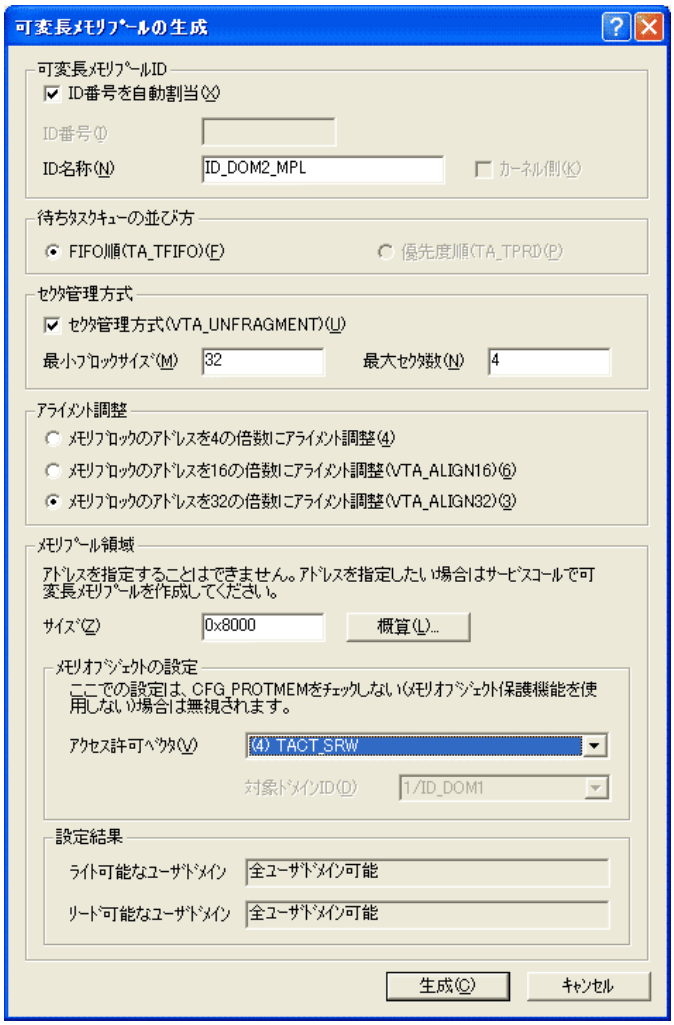


図10.49 [可変長メモリプールの生成]ダイアログボックス

[可変長メモリプールの生成]ダイアログボックスは、[可変長メモリプール]ページのポップアップメニューから[生成]を選択した時に、[可変長メモリプールの生成情報を変更]ダイアログボックスは、[可変長メモリプール]ページのポップアップメニューから[変更]を選択した時に開きます。この2つのダイアログボックスの構成は同じです。

なお、可変長メモリプールは cre_mpl, acre_mpl で動的に生成することもできます。これらのサービスコールで可変長メモリプールを生成する場合は、アプリケーション側で確保した領域をメモリプールとして使用する指定ができますが、コンフィギュレータではこの指定はできず、常にシステムプールから可変長メモリプールが割り付けられます。

(1) [可変長メモリプール ID]グループ

(a) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(b) [ID 番号]

可変長メモリプール ID を数値で入力してください。指定できる値は、1～CFG_MAXMPLID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(c) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(d) [カーネル側]

生成する可変長メモリプールをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(2) [待ちタスクキューの並び方]グループ

待ちタスクキューの並び方は、FIFO 順のみ選択できます。

(3) [セクタ管理方式]グループ

[セクタ管理方式(VTA_UNFRAGMENT)]を選択すると、可変長メモリプールをセクタ方式で管理します。この場合のみ[最小ブロックサイズ]と[最大セクタ数]が有効になります。

セクタ管理方式は、微小なメモリブロックを大量に獲得するメモリプールに適した属性で、微小なブロックをできるだけ連続して使用するようにすることで、大きなサイズの連続空き領域が維持されやすくします。セクタ管理方式の詳細、および[最小ブロックサイズ]と[最大セクタ数]の意味は、以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

[最小ブロックサイズ]には、0 以外の整数を指定でき、[アライメント調整]グループで選択されたサイズの倍数に切上げられます。

[最大セクタ数]は 0 以外の整数を指定できます。0 を指定した場合は 1 に補正します。

[最大セクタ数]が[サイズ]/([最小ブロックサイズ]×32)よりも大きい場合は、実際の最大セクタ数はカーネルによって[サイズ]/([最小ブロックサイズ]×32)に補正されます。

なお、[セクタ管理方式(VTA_UNFRAGMENT)]を選択した場合は、メモリプールから獲得するメモリブロックのサイズは、[最小ブロックサイズ]の倍数に切上げられます。

(4) [アライメント調整]グループ

メモリプールから獲得するメモリブロックのアドレスのアライメント調整に関する指定を行います。

以下の３つのいずれかから選択します。

(a) [メモリブロックのアドレスを4の倍数にアライメント]
(b) [メモリブロックのアドレスを16の倍数にアライメント(VTA_ALIGN16)]
(c) [メモリブロックのアドレスを32の倍数にアライメント(VTA_ALIGN32)]

(5) [メモリプール領域]グループ

[サイズ]には、メモリプールのサイズを指定してください。4～0x20000000 を指定でき、4 の倍数に切上げられます。

[概算]ボタンを押すと、[サイズ]の概算値を算出するための[可変長メモリプール領域サイズの概算]ダイアログボックスが開きます。

[アクセス許可ベクタ]は、[カーネル]ページの CFG_PROTMEM をチェックした場合のみ、意味を持ちます。以下の中から選択してください。

(1) TACT_KERNEL
(2) TACT_PRW(domid)
(3) TACT_PRO(domid)
(4) TACT_SRW
(5) TACT_SRO
(6) TACT_SRPW(domid)

(2),(3),(6)を選択した場合のみ[対象ドメイン ID]が有効となります。この場合、[対象ドメイン ID]は許可対象とするユーザドメイン ID を選択してください。

[設定結果]には、[アクセス許可ベクタ]、[対象ドメイン ID]の設定によって、どのユーザドメインからライトまたはリード可能かが表示されます。

表 10.35に、[設定結果]の表示内容を示します。

なお、メモリ属性は常に以下となります。

- TA_RW (リード・ライト可能)
- TA_CACHE (キャッシャブル)
- TA_WBACK (コピーバックモード)

表10.35 [設定結果]の表示内容

設定		[設定結果]の表示	
アクセス許可ベクタ	対象ドメイン	ライト可能なユーザドメイン	リード可能なユーザドメイン
TACT_KERNEL	無効	全ユーザドメイン不可	全ユーザドメイン不可
TACT_PRW(domid)	有効	対象ドメインのみ可能	対象ドメインのみ可能
TACT_PRO(domid)	有効	全ユーザドメイン不可	対象ドメインのみ可能
TACT_SRW	無効	全ユーザドメイン可能	全ユーザドメイン可能
TACT_SRO	無効	全ユーザドメイン不可	全ユーザドメイン可能
TACT_SRPW(domid)	有効	対象ドメインのみ可能	全ユーザドメイン可能

(6) [生成]ボタン([可変長メモリプールの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の可変長メモリプールの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(7) [OK]ボタン([可変長メモリプールの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(8) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.46 [可変長メモリプール領域サイズの概算]ダイアログボックス

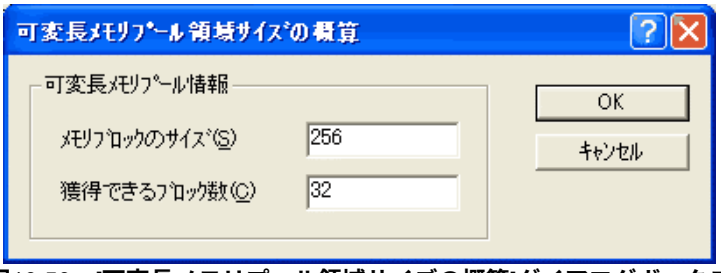


図10.50 [可変長メモリプール領域サイズの概算]ダイアログボックス

本ダイアログボックスは、[可変長メモリプールの生成]または[可変長メモリプールの生成情報を変更]ダイアログボックスの[概算]ボタンを押したときに開きます。

本ダイアログボックスは、[メモリブロックのサイズ]に指定したサイズのメモリブロックを[獲得できるブロック数]に指定した数だけ獲得できる可変長メモリプールのサイズを算出します。具体的には、TSZ_MPL マクロと同じ計算を行います。

[OK]ボタンを押すと、[可変長メモリプールの生成]または[可変長メモリプールの生成情報を変更]ダイアログボックスに戻ります。この時、[可変長メモリプールのサイズ]は算出結果に更新されます。

[キャンセル]ボタンを押すと、[可変長メモリプールの生成]または[可変長メモリプールの生成情報を変更]ダイアログボックスに戻ります。この時、[可変長メモリプールのサイズ]は更新されません。

10.7.47 [周期ハンドラ]ページ

本ページでは、周期ハンドラに関する項目を設定します。

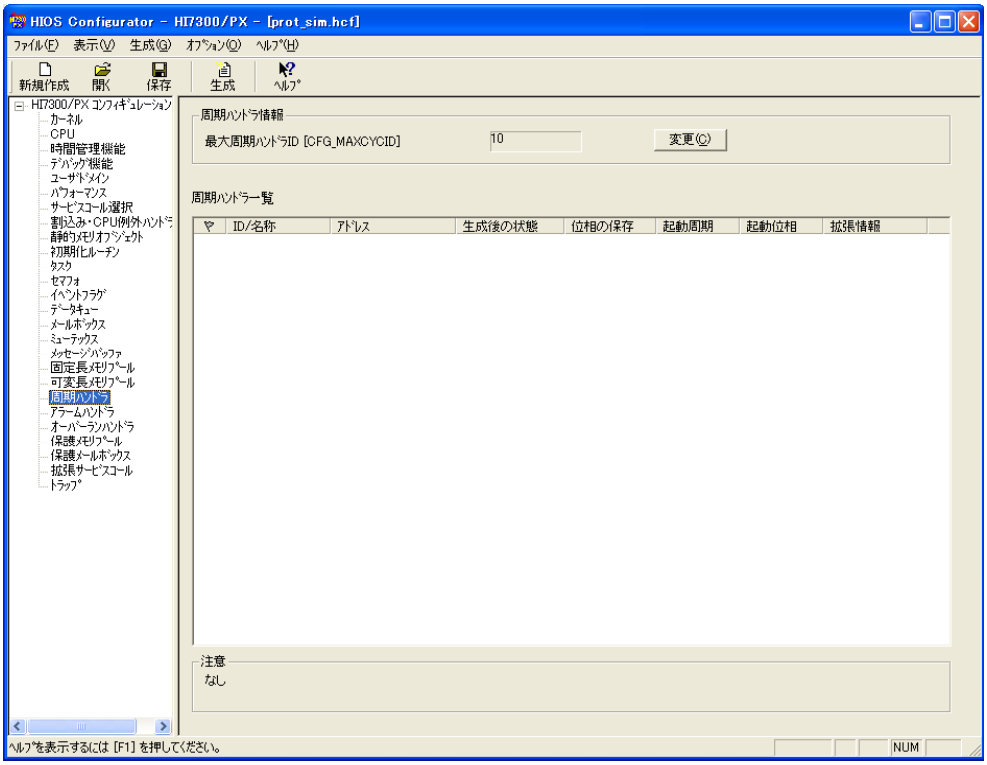


図10.51 [周期ハンドラ]ページ

表 10.36に、[周期ハンドラ]ページの項目を示します。

表10.36 [周期ハンドラ]ページの項目

No	項目	CFG 名	リンク区分
1	**最大周期ハンドラ ID**	CFG_MAXCYCID	**カーネル環境側**
2	**周期ハンドラの生成**	−	**カーネル側/カーネル環境側**

(1) [周期ハンドラ情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[周期ハンドラ情報の変更]ダイアログボックスが開きます。

(a) [最大周期ハンドラ ID[CFG_MAXCYCID]]

使用可能な周期ハンドラ ID の範囲は、1～CFG_MAXCYCID となります。

(2) [周期ハンドラー覧]グループ

ここには、生成済みの周期ハンドラが表示されます。旗のアイコンは、その周期ハンドラが[カーネル側]として生成されていることを示します。

周期ハンドラは、カーネル起動時に、「カーネル側」(旗のアイコンがあるもの)の周期ハンドラが本リストの上から順に生成され、次に「カーネル環境側」(旗のアイコンが無いもの)の周期ハンドラが本リストの上から順に生成されます。生成時に[生成後、動作状態へ(TA_STA)]または[起動位相を保存する(TA_PHS)]が指定された場合は、この順序に従って周期ハンドラが生成されることに留意してください。

以下のポップアップメニューがあります。

- [生成] : 周期ハンドラを生成するために、[周期ハンドラの生成]ダイアログボックスを開きます。
- [削除] : 選択された周期ハンドラを削除します。
- [変更] : 選択された周期ハンドラの設定を変更するために、[周期ハンドラの生成情報を変更]ダイアログボックスを開きます。
- [上へ] : 選択された周期ハンドラをひとつ上の周期ハンドラと順序を入れ替えます。
- [下へ] : 選択された周期ハンドラをひとつ下の周期ハンドラと順序を入れ替えます。

(3) [注意]

[サービスコール選択]ページの cre_cyc が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.37に示すメッセージが表示されます。

表10.37　[周期ハンドラ]ページの[注意]

条件	表示メッセージ
cre_cyc が未選択	cre_cyc が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.48 [周期ハンドラ情報の変更]ダイアログボックス

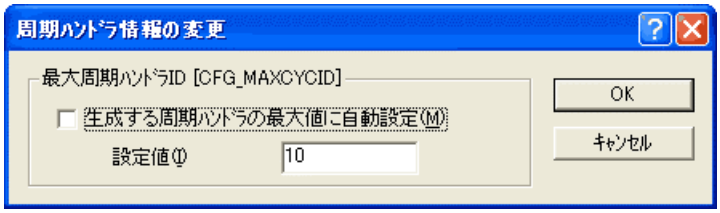


図10.52 [周期ハンドラ情報の変更]ダイアログボックス

本ダイアログボックスは、[周期ハンドラ]ページの[変更]ボタンを押したときに開きます。

(1) [最大周期ハンドラ ID[CFG_MAXCYCID]]

1～CFG_MAXCYCID の範囲の周期ハンドラ ID を使用できます。指定できるのは、0～254 の整数です。0 を指定すると、周期ハンドラを使用できなくなりますが、周期ハンドラを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成する周期ハンドラの最大値に自動設定]をチェックすると、[周期ハンドラ]ページで生成する周期ハンドラに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.49 [周期ハンドラの生成]ダイアログボックス、[周期ハンドラの生成情報を変更]ダイアログボックス

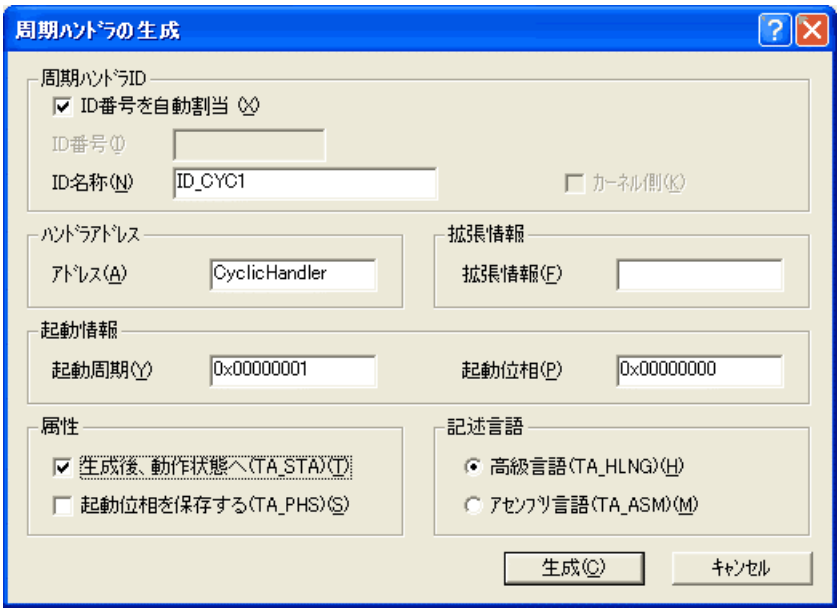


図10.53 [周期ハンドラの生成]ダイアログボックス

[周期ハンドラの生成]ダイアログボックスは、[周期ハンドラ]ページのポップアップメニューから[生成]を選択した時に、[周期ハンドラの生成情報を変更]ダイアログボックスは、[周期ハンドラ]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、周期ハンドラは cre_cyc, acre_cyc で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

周期ハンドラ ID を数値で入力してください。指定できる値は、1～CFG_MAXCYCID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成する周期ハンドラをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [アドレス]

周期ハンドラのアドレスを、C 言語シンボルまたは数値で指定してください。

(6) [拡張情報]

拡張情報は、周期ハンドラにパラメータとして渡されます。C 言語シンボルまたは数値で指定してください。

(7) [起動周期]、[起動位相]

周期ハンドラの起動周期と起動位相を指定してください。指定できるのは、ともに 1～0x7fffffff です。ただし、起動周期≧起動位相でなければなりません。

また、CFG_TICDENO>1 の場合は、0x7fffffff/CFG_TICDENO を超えてはなりません。コンフィギュレータはこの異常を検出しないので、注意してください。

(8) [生成後、動作状態へ(TA_STA)]

これをチェックすると、カーネル起動時に周期ハンドラが動作状態となります。

(9) [起動位相を保存する(TA_PHS)]

これをチェックすると、周期ハンドラが動作していないときにも起動位相が保存されます。

(10)[記述言語]

周期ハンドラが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(11)[生成]ボタン([周期ハンドラの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の周期ハンドラの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(12)[OK]ボタン([周期ハンドラの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(13)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.50 [アラームハンドラ]ページ

本ページでは、アラームハンドラに関する項目を設定します。

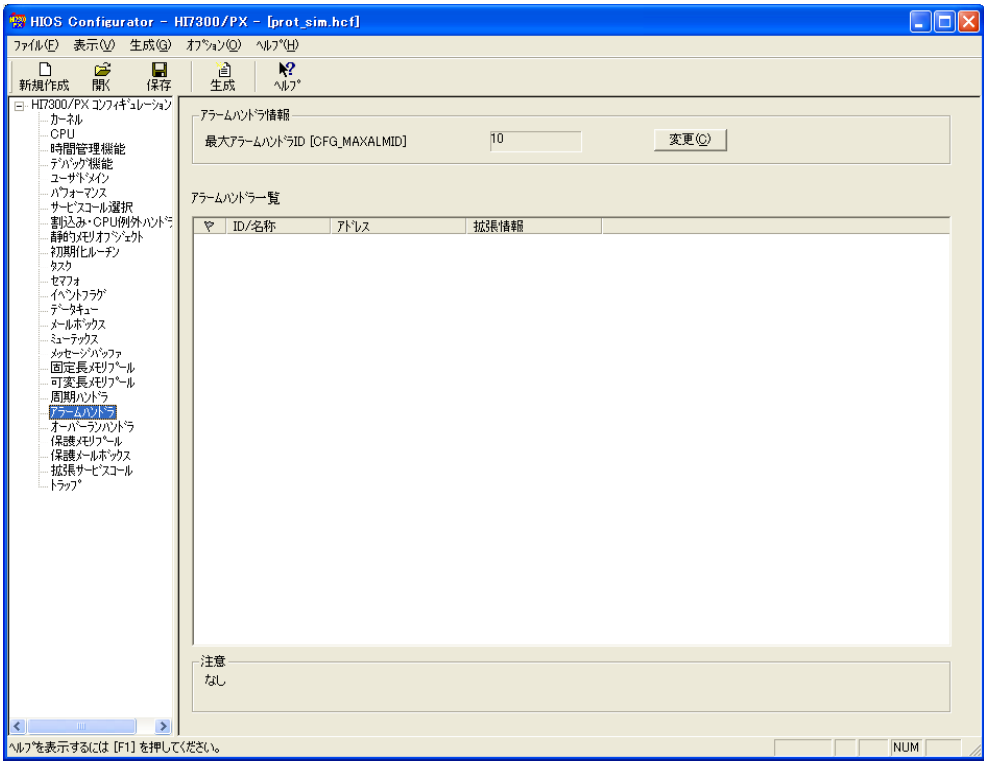


図10.54 [アラームハンドラ]ページ

表 10.38に、[アラームハンドラ]ページの項目を示します。

表10.38 [アラームハンドラ]ページの項目

No	項目	CFG 名	リンク区分
1	最大アラームハンドラ ID	CFG_MAXALMID	カーネル環境側
2	アラームハンドラの生成	−	カーネル側/カーネル環境側

(1) [アラームハンドラ情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[アラームハンドラ情報の変更]ダイアログボックスが開きます。

(a) [最大アラームハンドラ ID[CFG_MAXALMID]]

使用可能なアラームハンドラ ID の範囲は、1～CFG_MAXALMID となります。

(2) [アラームハンドラー覧]グループ

ここには、生成済みのアラームハンドラが表示されます。旗のアイコンは、そのアラームハンドラが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : アラームハンドラを生成するために、[アラームハンドラの生成]ダイアログボックスを開きます。
- [削除] : 選択されたアラームハンドラを削除します。
- [変更] : 選択されたアラームハンドラの設定を変更するために、[アラームハンドラの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[サービスコール選択]ページの cre_alm が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.39に示すメッセージが表示されます。

表10.39 [アラームハンドラ]ページの[注意]

条件	表示メッセージ
cre_alm が未選択	cre_alm が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.51 [アラームハンドラ情報の変更]ダイアログボックス

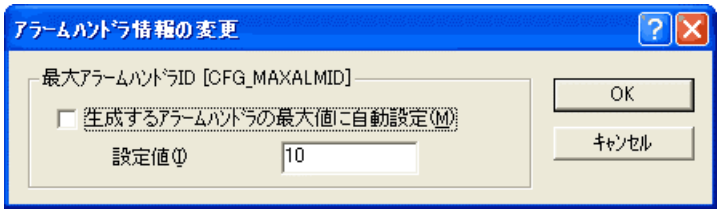


図10.55 [アラームハンドラ情報の変更]ダイアログボックス

本ダイアログボックスは、[アラームハンドラ]ページの[変更]ボタンを押したときに開きます。

(1) 最大アラームハンドラ ID[CFG_MAXALMID]

1～CFG_MAXALMID の範囲のアラームハンドラ ID を使用できます。指定できるのは、0～255 の整数です。0 を指定すると、アラームハンドラを使用できなくなりますが、アラームハンドラを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成するアラームハンドラの最大値に自動設定]をチェックすると、[アラームハンドラ]ページで生成するアラームハンドラに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.52 [アラームハンドラの生成]ダイアログボックス、[アラームハンドラの生成情報を変更]ダイアログボックス

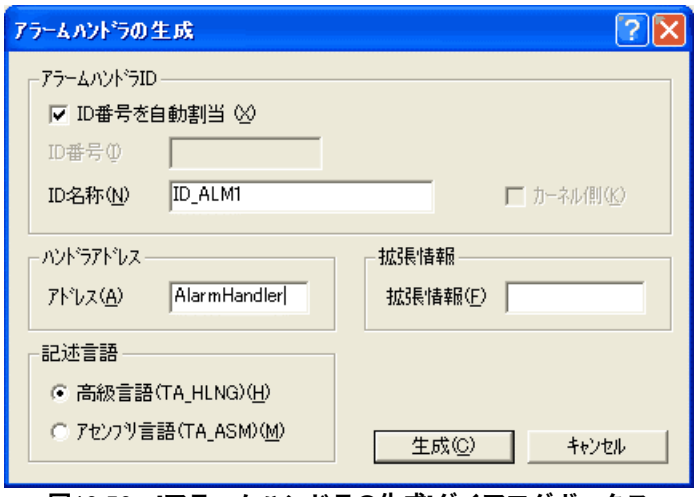


図10.56 [アラームハンドラの生成]ダイアログボックス

[アラームハンドラの生成]ダイアログボックスは、[アラームハンドラ]ページのポップアップメニューから[生成]を選択した時に、[アラームハンドラの生成情報を変更]ダイアログボックスは、[アラームハンドラ]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、アラームハンドラは cre_alm, acre_alm で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

アラームハンドラ ID を数値で入力してください。指定できる値は、1～CFG_MAXALMID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成するアラームハンドラをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [アドレス]

アラームハンドラのアドレスを、C 言語シンボルまたは数値で指定してください

(6) [拡張情報]

拡張情報は、アラームハンドラにパラメータとして渡されます。C 言語シンボルまたは数値で指定してください。

(7) [記述言語]

アラームハンドラが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(8) [生成]ボタン([アラームハンドラの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の周期ハンドラの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(9) [OK]ボタン([アラームハンドラの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(10)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.53 [オーバーランハンドラ]ページ

本ページでは、オーバーランハンドラを定義します。なお、オーバーランハンドラは def_ovr で動的に定義することもできます。

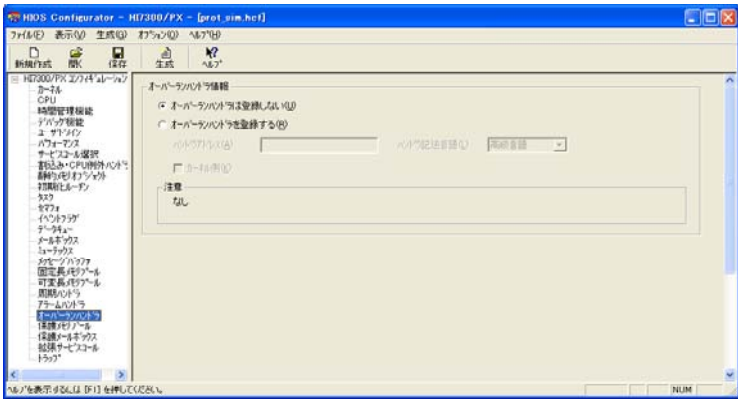


図10.57 [オーバーランハンドラ]ページ

(1) [オーバーランハンドラは登録しない]、[オーバーランハンドラを登録する]

オーバーランハンドラを登録する場合は、[オーバーランハンドラを登録する]を選択してください。この場合、以降の項目が有効となります。

(2) [ハンドラアドレス]

オーバーランハンドラのアドレスを、C 言語シンボルまたは数値で指定してください。

(3) [ハンドラ記述言語]

オーバーランハンドラが高級言語で記述されている場合には[高級言語]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語]を選択してください。

(4) [カーネル側]

登録するオーバーランハンドラをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常に変更不可となります。

(5) [注意]

[サービスコール選択]ページの def_ovr が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。
この場合、[注意]に表 10.40に示すメッセージが表示されます。

表10.40 [オーバーランハンドラ]ページの[注意]

条件	表示メッセージ
def_ovr が未選択	def_ovr が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.54 [保護メモリプール]ページ

本ページでは、保護メモリプールに関する項目を設定します。

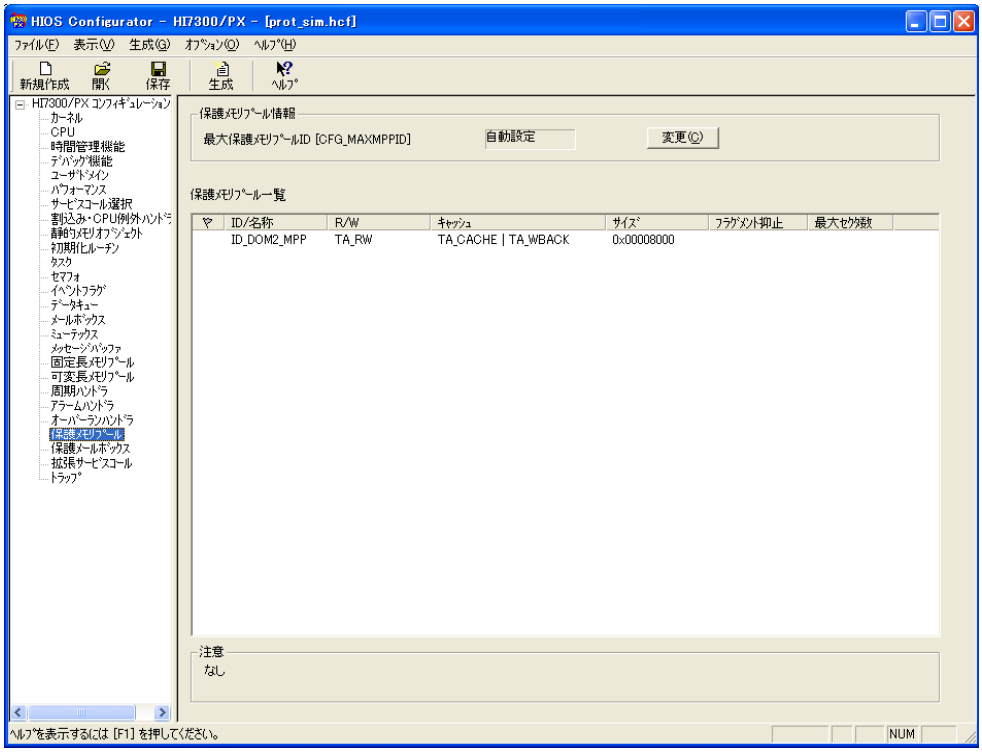


図10.58 [保護メモリプール]ページ

表 10.41に、[保護メモリプール]ページの項目を示します。

表10.41 [保護メモリプール]ページの項目

No	項目	CFG 名	リンク区分
1	最大保護メモリプール ID	CFG_MAXMPPID	カーネル環境側
2	保護メモリプールの生成	−	カーネル側/カーネル環境側

(1) [保護メモリプール情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[保護メモリプール情報の変更]ダイアログボックスが開きます。

(a) [最大保護メモリプール ID[CFG_MAXMPPID]]

使用可能な保護メモリプール ID の範囲は、1～CFG_MAXMPPID となります。

(2) [保護メモリプール一覧]グループ

ここには、生成済みの保護メモリプールが表示されます。旗のアイコンは、その保護メモリプールが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : 保護メモリプールを生成するために、[保護メモリプールの生成]ダイアログボックスを開きます。
- [削除] : 選択された保護メモリプールを削除します。
- [変更] : 選択された保護メモリプールの設定を変更するために、[保護メモリプールの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[カーネル]ページの CFG_PROTMEM が選択されていない場合、または[サービスコール選択]ページの icre_mpp が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.42に示すメッセージが表示されます。

表10.42 [保護メモリプール]ページの[注意]

条件	表示メッセージ
CFG_PROTMEM **が未選択、または** icre_mpp **が未選択**	CFG_PROTMEM **が選択されていないか、**icre_mpp **が組み込まれない設定になっているので、本ページの全設定項目は無効です**

10.7.55 [保護メモリプール情報の変更]ダイアログボックス

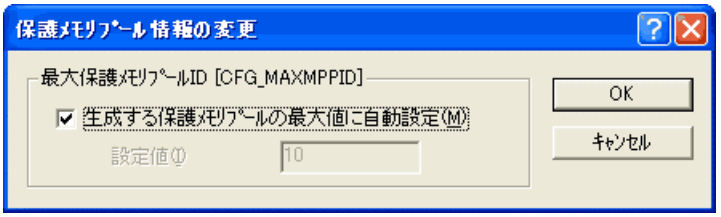


図10.59 [保護メモリプール情報の変更]ダイアログボックス

本ダイアログボックスは、[保護メモリプール]ページの[変更]ボタンを押したときに開きます。

(1) [最大保護メモリプール ID[CFG_MAXMPPID]]

1～CFG_MAXMPPID の範囲の保護メモリプール ID を使用できます。指定できるのは、0～31 の整数です。0 を指定すると、保護メモリプールを使用できなくなりますが、保護メモリプールを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成する保護メモリプールの最大値に自動設定]をチェックすると、[保護メモリプール]ページで生成する保護メモリプールに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.56 [保護メモリプールの生成]ダイアログボックス、[保護メモリプールの生成情報を変更]ダイアログボックス

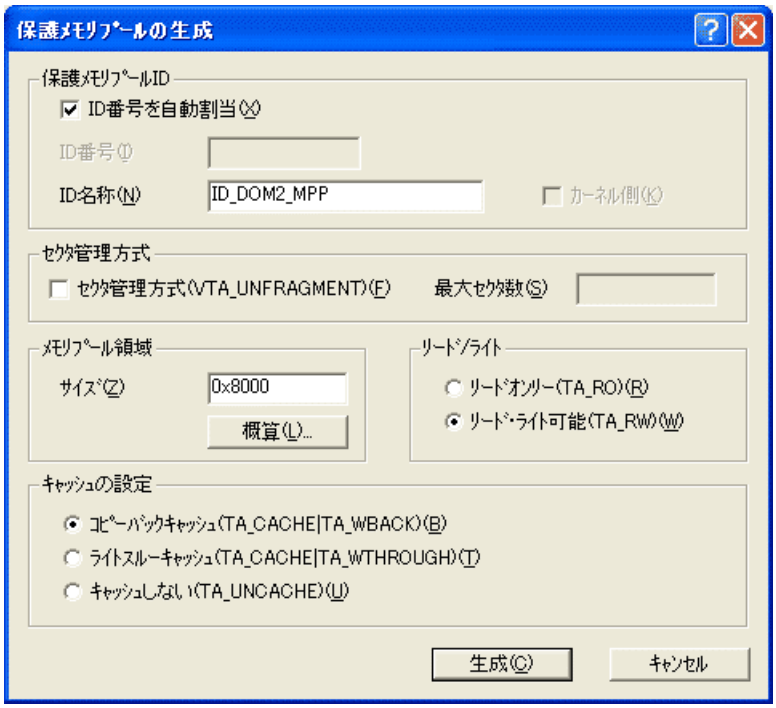


図10.60 [保護メモリプールの生成]ダイアログボックス

[保護メモリプールの生成]ダイアログボックスは、[保護メモリプール]ページのポップアップメニューから[生成]を選択した時に、[保護メモリプールの生成情報を変更]ダイアログボックスは、[保護メモリプール]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

(1) [保護メモリプール ID]グループ

(a) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(b) [ID 番号]

保護メモリプール ID を数値で入力してください。指定できる値は、1～CFG_MAXMPPID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(c) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(d) [カーネル側]

生成する保護メモリプールをカーネル側とする場合に、チェックしてください。
カーネルロックモードでは、常にチェック不可となります。

(2) [セクタ管理方式]グループ

[セクタ管理方式(VTA_UNFRAGMENT)]を選択すると、保護メモリプールをセクタ方式で管理します。この場合のみ[最大セクタ数]が有効になります。

セクタ管理方式は、微小なメモリブロックを大量に獲得するメモリプールに適した属性で、微小なブロックをできるだけ連続して使用するようにすることで、大きなサイズの連続空き領域が維持されやすくします。セクタ管理方式の詳細、および[最大セクタ数]の意味は、以下を参照してください。

■関連ページ　「4.31 メモリの断片化とその対策」

[最大セクタ数]は 0 以外の整数を指定できます。0 を指定した場合は 1 に補正します。
[最大セクタ数]が[サイズ]/(4096×32)よりも大きい場合は、実際の最大セクタ数はカーネルによって[サイズ]/(4096×32)に補正されます。

(3) [メモリプール領域]グループ

[サイズ]には、メモリプールのサイズを指定してください。1～0x20000000 を指定でき、CFG_PAGESZ(4096)の倍数に切上げられます。

保護メモリプール領域は、以下のセクション名として生成されます。

```
BUCM_himpp_<ID>
```

<ID>は、ID 名称を指定した場合は ID 名称、そうでない場合は ID 番号の 10 進数表記となります。このセクションは、リンク時に MMU 対象領域で、CFG_PAGESZ(4096)の境界アドレスに配置しなければなりません。

[概算]ボタンを押すと、[サイズ]の概算値を算出するための[保護メモリプール領域サイズの概算]ダイアログボックスが開きます。

(4) [リード/ライト]グループ

リードオンリーと扱うか、リードライト可能と扱うかを選択してください。

(5) [キャッシュの設定] グループ

キャッシュ Enable 時の振る舞いを、以下の中から選択してください。

- コピーバックキャッシュ(TA_CACHE|TA_WBACK)
- ライトスルーキャッシュ(TA_CACHE|TA_WTHROUGH)
- キャッシュしない(TA_UNCACHE)

(6) [生成]ボタン([保護メモリプールの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の保護メモリプールの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(7) [OK]ボタン([保護メモリプールの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(8) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.57 [保護メモリプール領域サイズの概算]ダイアログボックス

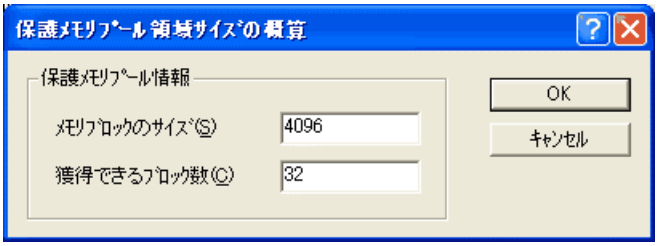


図10.61 [保護メモリプール領域サイズの概算]ダイアログボックス

本ダイアログボックスは、[保護メモリプールの生成]または[保護メモリプールの生成情報を変更]ダイアログボックスの[概算]ボタンを押したときに開きます。

本ダイアログボックスは、[メモリブロックのサイズ]に指定したサイズのメモリブロックを[獲得できるブロック数]に指定した数だけ獲得できる保護メモリプールのサイズを算出します。具体的には、TSZ_MPP マクロと同じ計算を行います。

[OK]ボタンを押すと、[保護メモリプールの生成]または[保護メモリプールの生成情報を変更]ダイアログボックスに戻ります。この時、保護メモリプールの[サイズ]は算出結果に更新されます。

[キャンセル]ボタンを押すと、[保護メモリプールの生成]または[保護メモリプールの生成情報を変更]ダイアログボックスに戻ります。この時、保護メモリプールの[サイズ]は更新されません。

10.7.58 [保護メールボックス]ページ

本ページでは、保護メールボックスに関する項目を設定します。

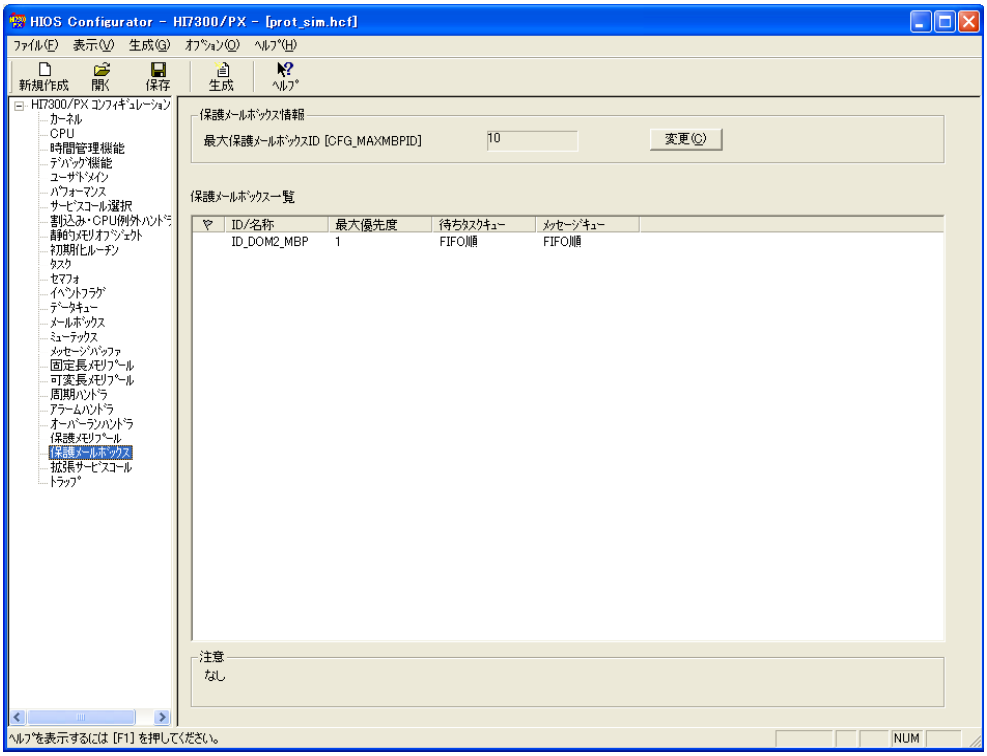


図10.62 [保護メールボックス]ページ

表 10.43に、[保護メールボックス]ページの項目を示します。

表10.43 [保護メールボックス]ページの項目

No	項目	CFG 名	リンク区分
1	最大保護メールボックス ID	CFG_MAXMBPID	カーネル環境側
2	保護メールボックスの生成	−	カーネル側/カーネル環境側

(1) [保護メールボックス情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[保護メールボックス情報の変更]ダイアログボックスが開きます。

(a) [最大保護メールボックス ID[CFG_MAXMBPID]]

使用可能な保護メールボックス ID の範囲は、1～CFG_MAXMBPID となります。

(2) [保護メールボックス一覧]グループ

ここには、生成済みの保護メールボックスが表示されます。旗のアイコンは、その保護メールボックスが[カーネル側]として生成されていることを示します。

以下のポップアップメニューがあります。

- [生成] : 保護メールボックスを生成するために、[保護メールボックスの生成]ダイアログボックスを開きます。
- [削除] : 選択された保護メールボックスを削除します。
- [変更] : 選択された保護メールボックスの設定を変更するために、[保護メールボックスの生成情報を変更]ダイアログボックスを開きます。

(3) [注意]

[カーネル]ページの CFG_PROTMEM が選択されていない場合、または[サービスコール選択]ページの cre_mbp が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.44に示すメッセージが表示されます。

表10.44 [保護メールボックス]ページの[注意]

条件	表示メッセージ
CFG_PROTMEM **が未選択、または** cre_mbp **が未選択**	CFG_PROTMEM **が選択されていないか、**cre_mbp **が組み込まれない設定になっているので、本ページの全設定項目は無効です**

10.7.59 [保護メールボックス情報の変更]ダイアログボックス

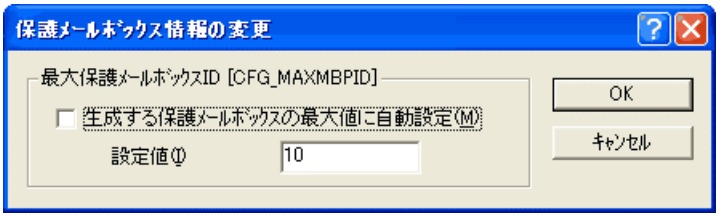


図10.63 [保護メールボックス情報の変更]ダイアログボックス

本ダイアログボックスは、[保護メールボックス]ページの[変更]ボタンを押したときに開きます。

(1) [最大保護メールボックス ID[CFG_MAXMBPID]]

1～CFG_MAXMBPID の範囲の保護メールボックス ID を使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、保護メールボックスを使用できなくなりますが、保護メールボックスを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

[生成する保護メールボックスの最大値に自動設定]をチェックすると、[保護メールボックス]ページで生成する保護メールボックスに応じて、コンフィギュレータが自動計算します。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.60 [保護メールボックスの生成]ダイアログボックス、[保護メールボックスの生成情報を変更]ダイアログボックス

図10.64 [保護メールボックスの生成]ダイアログボックス

[保護メールボックスの生成]ダイアログボックスは、[保護メールボックス]ページのポップアップメニューから[生成]を選択した時に、[保護メールボックスの生成情報を変更]ダイアログボックスは、[保護メールボックス]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、周期ハンドラは cre_mbp, acre_mbp で動的に生成することもできます。

(1) [ID 番号を自動割当]

これをチェックすると、コンフィギュレータが ID 番号を自動的に割り当てます。ただし、これをチェックすると[カーネル側]は選択できなくなります。

(2) [ID 番号]

保護メールボックス ID を ID を数値で入力してください。指定できる値は、1～CFG_MAXMBPID の範囲です。ただし、[ID 番号を自動割当]をチェックしている場合は、ID 番号は指定できません。

(3) [ID 名称]

ID 名称を指定してください。[ID 番号を自動割当]とした場合は、名称の指定は必須です。その他の場合は、空欄でもかまいません。

(4) [カーネル側]

生成する保護メールボックスをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(5) [待ちタスクキューの並び方]

待ちタスクキューの並び方として、FIFO 順または優先度順を選択してください。

(6) [メッセージキューの並び方]

メッセージキューの並び方として、FIFO 順または優先度順を選択してください。

(7) [最大優先度]

[メッセージキューの並び方]として優先度順を選択した場合は、メッセージの最大優先度を選択してください。選択できるのは、1～CFG_MAXMSGPRI です。

[メッセージキューの並び方]として FIFO 順を選択した場合は、本項目は意味を持ちません。

(8) [生成]ボタン([保護メールボックスの生成]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の保護メールボックスの生成を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(9) [OK]ボタン([保護メールボックスの生成情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(10)[キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.61 [拡張サービスコール]ページ

本ページでは、拡張サービスコールに関する項目を設定します。

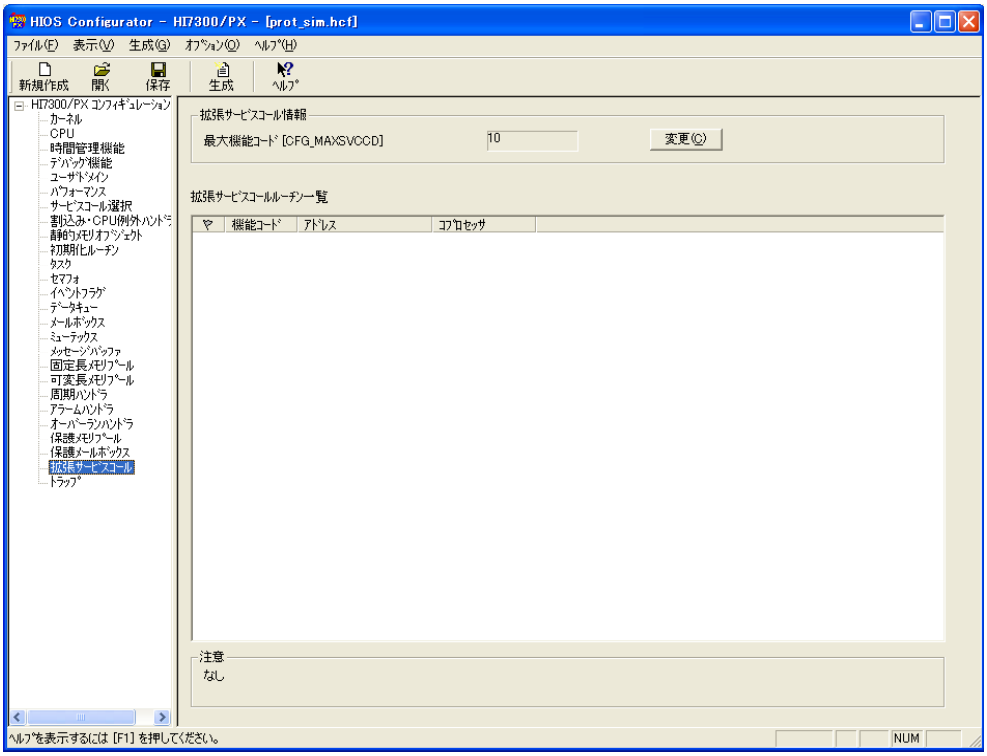


図10.65 [拡張サービスコール]ページ

表 10.45に、[拡張サービスコール]ページの項目を示します。

表10.45 [拡張サービスコール]ページの項目

No	項目	CFG 名	リンク区分
1	最大機能コード	CFG_MAXSVCCD	カーネル環境側
2	拡張サービスコールルーチンの定義	−	カーネル側/カーネル環境側

(1) [拡張サービスコール情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[拡張サービスコール情報の変更]ダイアログボックスが開きます。

(a) 最大機能コード[CFG_MAXSVCCD]

使用可能な機能コードの範囲は、1～CFG_MAXSVCCD となります。

(2) [拡張サービスコールルーチン一覧]グループ

ここには、定義済みの拡張サービスコールルーチンが表示されます。旗のアイコンは、そのルーチンが[カーネル側]として定義されていることを示します。

ポップアップメニューとして、以下の項目があります。カーネルロックモードでは、[カーネル側]のルーチンについてはこれらのポップアップメニューは選択できなくなります。

- [定義]：拡張サービスコールルーチンを定義するために、[拡張サービスコールルーチンの定義]ダイアログボックスを開きます。
- [削除]：選択された拡張サービスコールルーチンの定義を解除します。
- [変更]：選択された拡張サービスコールの設定を変更するために、[拡張サービスコールルーチンの定義情報を変更]ダイアログボックスを開きます

(3) [注意]

[サービスコール選択]ページの def_svc が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.46に示すメッセージが表示されます。

表10.46　[拡張サービスコール]ページの[注意]

条件	表示メッセージ
def_svc が未選択	def_svc が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.62 [拡張サービスコール情報の変更]ダイアログボックス

図10.66 [拡張サービスコール情報の変更]ダイアログボックス

本ダイアログボックスは、[拡張サービスコール]ページの[変更]ボタンを押したときに開きます。

(1) [最大機能コード[CFG_MAXSVCCD]]

1～CFG_MAXSVCCD の範囲の機能コードを使用できます。指定できるのは、0～32767 の整数です。0 を指定すると、拡張サービスコールを使用できなくなりますが、拡張サービスコールを管理するための変数領域を確保しなくなるため、RAM 使用量を削減できます。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.63 [拡張サービスコールルーチンの定義]ダイアログボックス、[拡張サービスコールルーチンの定義情報を変更]ダイアログボックス

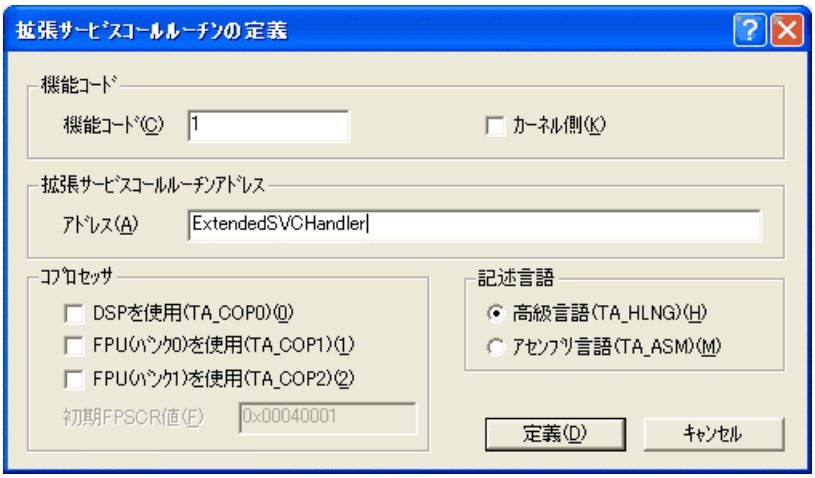


図10.67 [拡張サービスコールルーチンの定義]ダイアログボックス

[拡張サービスコールルーチンの定義]ダイアログボックスは、[拡張サービスコール]ページのポップアップメニューから[定義]を選択した時に、[拡張サービスコールルーチンの定義情報を変更]ダイアログボックスは、[拡張サービスコール]ページのポップアップメニューから[変更]を選択した時に開きます。この 2 つのダイアログボックスの構成は同じです。

なお、拡張サービスコールは def_svc で動的に定義することもできます。

(1) [機能コード]

機能コードを数値で入力してください。指定できる値は、1～CFG_MAXSVCCD の範囲です。

(2) [カーネル側]

定義する拡張サービスコールルーチンをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(3) [拡張サービスコールルーチンアドレス]

拡張サービスコールルーチンのアドレスを、C 言語シンボルまたは数値で指定してください。

(4) [DSP を使用(TA_COP0)]、[FPU(バンク 0)を使用(TA_COP1)]、[FPU(バンク 1)を使用(TA_COP2)]

TA_COP0 は、[CPU]ページで CFG_DSP をチェックした場合のみ有効です。DSP を使用した演算を行う場合にチェックしてください。

TA_COP1, TA_COP2 は、[CPU]ページで CFG_FPU をチェックした場合のみ有効です。通常の FPU 演算を行う場合は TA_COP1 のみをチェックしてください。マトリックス演算など、FPU の両バンクを使用する場合は、両方をチェックしてください。TA_COP1 がチェック無しで TA_COP2 がチェック有り、という設定はできません。

また、TA_COP0 と TA_COP1,TA_COP2 を同時にチェックすることはできません。

(5) [初期 FPSCR 値]

初期 FPSCR 値は、TA_COP1, TA_COP2 のいずれかをチェックをした場合のみ意味を持ちます。0～0xffffffff の整数を指定できます。以下の節を参考に指定してください。

■関連ページ　「15. FPU に関する注意事項」

(6) [記述言語]

拡張サービスコールルーチンが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(7) [定義]ボタン([拡張サービスコールの定義]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とします。そして、次の拡張サービスコールの定義を連続して行えるように、本ダイアログボックスの表示を初期状態に戻します。本ダイアログボックスは閉じません。

(8) [OK]ボタン([拡張サービスコールの定義情報を変更]ダイアログボックスのみ)

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(9) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.64 [トラップ]ページ

本ページでは、トラップに関する項目を設定します。

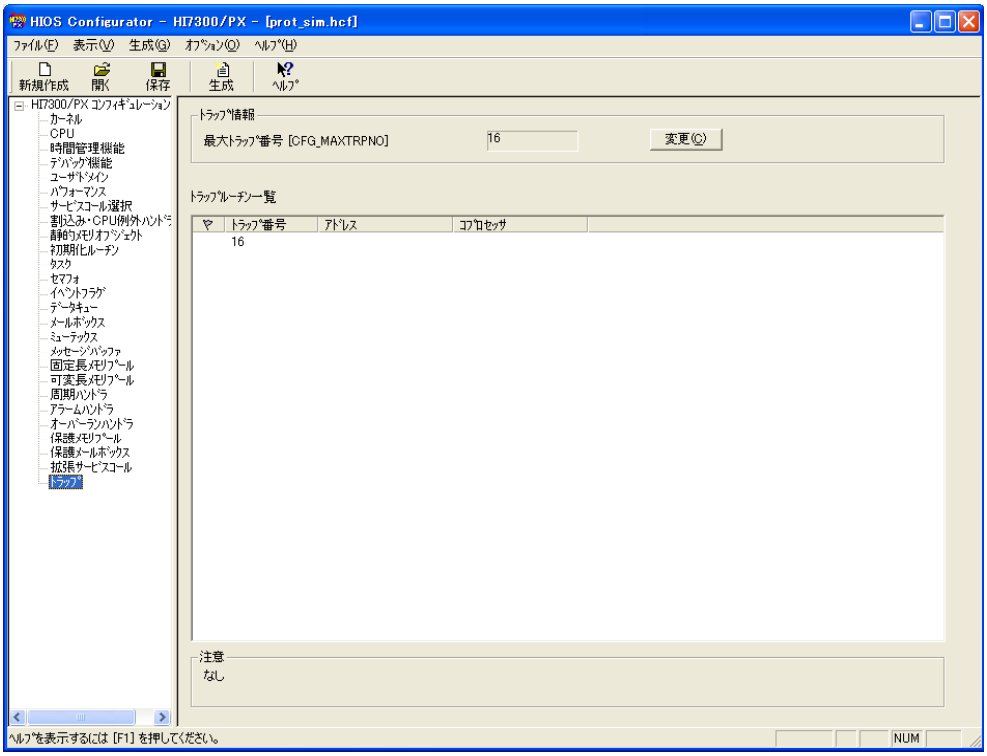


図10.68 [トラップ]ページ

表 10.47に、[トラップ]ページの項目を示します。

表10.47 [トラップ]ページの項目

No	項目	CFG 名	リンク区分
1	**最大トラップ番号**	CFG_MAXTRPNO	**カーネル環境側**
2	**トラップルーチンの定義**	−	**カーネル側/カーネル環境側**

(1) [トラップ情報]グループ

ここには、以下の情報が表示されます。[変更]ボタンを押すと、これらを変更するための[トラップ情報の変更]ダイアログボックスが開きます。

(a) 最大トラップ番号[CFG_MAXTRPNO]

使用可能なトラップ番号の範囲は、16～CFG_MAXTRPNO となります。

(2) [トラップルーチン一覧]グループ

ここには、定義済みのトラップルーチンが表示されます。旗のアイコンは、そのルーチンが[カーネル側]として定義されていることを示します。

ポップアップメニューとして、以下の項目があります。カーネルロックモードでは、[カーネル側]のルーチンについてはこれらのポップアップメニューは選択できなくなります。

- [定義]：トラップルーチンを定義するために、[トラップルーチンの定義]ダイアログボックスを開きます。
- [解除]：選択されたトラップルーチンの定義を解除します。

(3) [注意]

[サービスコール選択]ページの vdef_trp が選択されていない場合は、本ページでの設定項目は全て無効となります。また、本ページの全ての項目は変更できなくなります。

この場合、[注意]に表 10.48に示すメッセージが表示されます。

表10.48 [トラップ]ページの[注意]

条件	表示メッセージ
vdef_trp が未選択	vdef_trp が組み込まれない設定になっているので、本ページの全設定項目は無効です

10.7.65 [トラップ情報の変更]ダイアログボックス

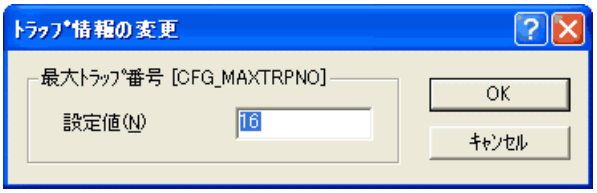


図10.69 [トラップ情報の変更]ダイアログボックス

本ダイアログボックスは、[トラップ]ページの[変更]ボタンを押したときに開きます。

(1) [最大トラップ番号[CFG_MAXTRPNO]]

16～CFG_MAXTRPNO の範囲のトラップ番号を使用できます。指定できるのは、16～255 の整数です。

(2) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(3) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.7.66 [トラップルーチンの定義]ダイアログボックス

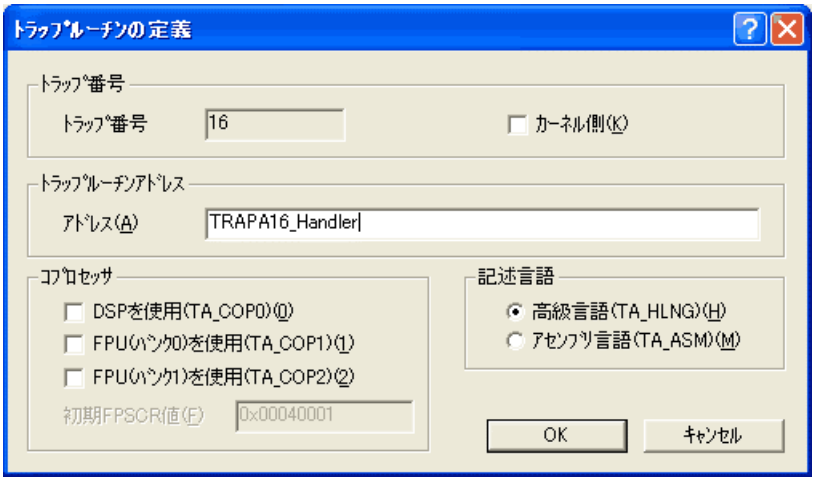


図10.70 [トラップルーチンの定義]ダイアログボックス

[トラップルーチンの定義]ダイアログボックスは、[トラップ]ページのポップアップメニューから[定義]を選択した時に開きます。

なお、トラップは vdef_trp で動的に定義することもできます。

(1) [トラップ番号]

[トラップ]ページで選択したトラップ番号が表示されます。

(2) [カーネル側]

定義するトラップルーチンをカーネル側とする場合に、チェックしてください。

カーネルロックモードでは、常にチェック不可となります。

(3) [トラップルーチンアドレス]

トラップルーチンのアドレスを、C 言語シンボルまたは数値で指定してください。

(4) [DSP を使用(TA_COP0)]、[FPU(バンク 0)を使用(TA_COP1)]、[FPU(バンク 1)を使用(TA_COP2)]

TA_COP0 は、[CPU]ページで CFG_DSP をチェックした場合のみ有効です。DSP を使用した演算を行う場合にチェックしてください。

TA_COP1, TA_COP2 は、[CPU]ページで CFG_FPU をチェックした場合のみ有効です。通常の FPU 演算を行う場合は TA_COP1 のみをチェックしてください。マトリックス演算など、FPU の両バンクを使用する場合は、両方をチェックしてください。TA_COP1 がチェック無しで TA_COP2 がチェック有り、という設定はできません。

また、TA_COP0 と TA_COP1,TA_COP2 を同時にチェックすることはできません。

(5) [初期 FPSCR 値]

初期 FPSCR 値は、TA_COP1, TA_COP2 のいずれかをチェックをした場合のみ意味を持ちます。0～0xffffffff の整数を指定できます。以下の節を参考に指定してください。

■関連ページ　「15. FPU に関する注意事項」

(6) [記述言語]

トラップルーチンが高級言語で記述されている場合には[高級言語(TA_HLNG)]を選択し、アセンブリ言語で記述されている場合には[アセンブリ言語(TA_ASM)]を選択してください。

(7) [OK]ボタン

本ダイアログボックスの設定内容を有効とし、本ダイアログボックスを閉じます。

(8) [キャンセル]ボタン

本ダイアログボックスの設定内容を破棄し、本ダイアログボックスを閉じます。

10.8 エディットボックスの仕様

■入力可能文字数

エディットボックスに入力できる文字数は 255 バイトまでです。これを超える入力は行えないようになっています。

■入力可能な文字

(1) 数値と文字列の認識

エディットボックスに入力するのは ASCII コードのみとしてください。その他の文字コードは入力しないでください。

以下の場合は数値として扱い、それ以外の場合は文字列と扱います。

(1) '0'～'9'のみで構成される文字列：10進数と扱います。

(2) "0x"または"0X"で始まり、以降'の1～8文字が'0'～'9', 'a'～'f', 'A'～'F'のみで構成される文字列：16進数と扱います。

数値として扱う場合で、32bit で表現できない値の場合は、以下のエラーを表示します。

「0xffffffff を超える数値は入力できません。」

(2)のケースで、"0x"または"0X"以降が 9 文字以上ある場合も、このエラーを表示します。

以下に例を示します。

- "123"：10 進数の 123 として扱います。
- "0x1000"：16 進数の 0x1000(4096)として扱います。
- "0x012345678"：エラー「0xffffffff を超える数値は入力できません。」を表示します。
- "0x0123Z"：文字列として扱います。

(2) 文字列

コンフィギュレータで、文字列として意味があるのは以下です。

- ID 名称(C 言語マクロ名)
- C 言語シンボル
- セクション名

これらの入力に関して、C 言語文法として適切かどうかは検査しません。適切でない場合、コンパイル時に文法エラーが検出されます。

■空欄

表 10.49に示すエディットボックスでは、何も入力しない場合にはデフォルトの設定として扱います。

その他のエディットボックスは、空欄の場合には以下のいずれかのエラーメッセージが表示されます。

「値を入力してください。」

「文字列を入力してください。」

「値または文字列を入力してください。」

表10.49　空欄が許可されるエディットボックス

ページ/ダイアログボックス	エディットボックス	空欄が許可される条件	デフォルトの扱い
[タスクの生成]・[タスクの生成情報を変更]ダイアログボックス [初期化ルーチンの登録]・[初期化ルーチンの登録情報を変更]ダイアログボックス [周期ハンドラの生成]・[周期ハンドラの生成情報を変更]ダイアログボックス [アラームハンドラの生成]・[アラームハンドラの生成情報を変更]ダイアログボックス	拡張情報	常に	0 として扱います。
[タスク例外処理ルーチンの定義]ダイアログボックス	アドレス	常に	定義解除と扱います。
各種オブジェクトの[生成]および[生成情報を変更]ダイアログ	ID 名称	[ID 番号を自動割当]をチェックしない場合	名称なしと扱います。

■ID 名称、C 言語シンボル、セクション名の重複

下記のケースでは、以下のエラーメッセージが表示されます。

「指定されたシンボル(名称)は、既に使用されています。他のシンボル(名称)を指定してください。」

- ID 名称の入力：指定した ID 名称が、既に登録済みの ID 名称・C 言語シンボル・セクション名と重複する場合
- C 言語シンボルの入力：指定したシンボルが、既に登録済みの ID 名称・セクション名と重複する場合
- セクション名の入力：指定したセクション名が、既に登録済みの ID 名称・C 言語シンボル・セクション名と重複する場合

10.9 チューニング

10.9.1 RAM 使用量の削減

表 10.50に、コンフィギュレータの設定項目の中で RAM 使用量に関係する項目を示します。

表10.50 RAM 使用量の削減

ページ	項目	該当セクション名	使用量を削減するには
[カーネル]ページ	CFG_NTSKSTKSZ	BSCP_hintskstk	小さくする
	CFG_RESPOOLSZ	BSCP_hirespl	小さくする
	CFG_SYSPOOLSZ	BSCP_hisyspl	小さくする
	CFG_PROTMEM	BSCP_hidef, BSCP_hicfg, BSCP_hiwrk	チェックしない
[デバッグ機能]ページ	CFG_ACTION	BSCP_hiwrk	チェックしない
	CFG_TRACE	BSCP_hiwrk	チェックしない
	CFG_TRCOBJCNT	BSCP_hiwrk	小さくする
	CFG_TRCBUFSZ	BSCP_hitrcbuf	小さくする
[時間管理]ページ	CFG_OPTTMR	BSCP_hiwrk	チェックしない
[パフォーマンス]ページ	CFG_PERFORM	BSCP_hiwrk	チェックしない
[サービスコール選択]ページ	vset_tfl, vclr_tfl, vwai_tfl, vtwai_tfl, vpwai_tfl	BSCP_hiwrk	全て選択しない
	def_tex	BSCP_hiwrk	選択しない
	cre_sem	BSCP_hiwrk	選択しない
	cre_flg	BSCP_hiwrk	選択しない
	cre_dtq	BSCP_hiwrk	選択しない
	cre_mbx	BSCP_hiwrk	選択しない
	cre_mtx	BSCP_hiwrk	選択しない
	cre_mbf	BSCP_hiwrk	選択しない
	cre_mpf	BSCP_hiwrk	選択しない
	cre_mpl	BSCP_hiwrk	選択しない
	cre_cyc	BSCP_hiwrk	選択しない
	cre_alm	BSCP_hiwrk	選択しない
	def_ovr	BSCP_hiwrk	選択しない
	icre_mpp	BSCP_hiwrk	選択しない
	cre_mbp	BSCP_hiwrk	選択しない
	def_svc	BSCP_hiwrk	選択しない
	vdef_trp	BSCP_hiwrk	選択しない
[割込み,CPU 例外]ページ	CFG_MAXINTNO	BSCP_hiwrk	小さくする
[タスク]ページ	CFG_MAXTSKID	BSCP_hiwrk	小さくする
	CFG_MAXTSKPRI	BSCP_hiwrk	小さくする
[セマフォ]ページ	CFG_MAXSEMID	BSCP_hiwrk	小さくする
[イベントフラグ]ページ	CFG_MAXFLGID	BSCP_hiwrk	小さくする
[データキュー]ページ	CFG_MAXDTQID	BSCP_hiwrk	小さくする
[メールボックス]ページ	CFG_MAXMBXID	BSCP_hiwrk	小さくする
[ミューテックス]ページ	CFG_MAXMTXID	BSCP_hiwrk	小さくする
[メッセージバッファ]ページ	CFG_MAXMBFID	BSCP_hiwrk	小さくする

[固定長メモリプール]ページ	CFG_MAXMPFID	BSCP_hiwrk	小さくする
[可変長メモリプール]ページ	CFG_MAXMPLID	BSCP_hiwrk	小さくする
[周期ハンドラ]ページ	CFG_MAXCYCID	BSCP_hiwrk	小さくする
[アラームハンドラ]ページ	CFG_MAXALMID	BSCP_hiwrk	小さくする
[保護メモリプール]ページ	CFG_MAXMPPID	BSCP_hiwrk	小さくする
[保護メールボックス]ページ	CFG_MAXMBPID	BSCP_hiwrk	小さくする
[拡張サービスコール]ページ	CFG_MAXSVCCD	BSCP_hiwrk	小さくする
[トラップ]ページ	CFG_MAXTRPNO	BSCP_hiwrk	小さくする

10.9.2 ROM 消費量の削減

最も効果があるのは、[サービスコール選択]ページで選択するサービスコールを少なくすることです。特に、def_???, cre_???を未選択にすると、その機能モジュールが組み込まれなくなるので効果が大きいです。

その他に、コンフィギュレータの設定項目の中で ROM 使用量に関係する項目を、表 10.51に示します。

表10.51　ROM 使用量の削減

ページ	項目	該当セクション名	使用量を削減するには
[カーネル]ページ	CFG_PARCHK	PSCP_hiknl	チェックしない
	CFG_PROTMEM	PSCP_hiknl, CSCP_hidef, CSCP_hicfg	チェックしない
	CFG_MEMCHK	PSCP_hiknl	チェックしない
	CFG_MAXLOCPAGE	PSCP_hiknl	0 にする
[CPU]ページ	CFG_DSPSTBY	PSCP_hiknl, PSCP_hidef	チェックしない
[時間管理機能]ページ	CFG_OPTTMR	PSCP_hiknl	チェックしない
[デバッグ機能]ページ	CFG_ACTION	PSCP_hiknl	チェックしない
	CFG_TRACE	PSCP_hiknl	チェックしない
[パフォーマンス]ページ	CFG_PERFORM	PSCP_hiknl	チェックしない
全般	各種オブジェクトの初期登録	PSCP_hidef, CSCP_hidef, PSCP_hicfg, CSCP_hicfg	少なくする

10.9.3 性能向上

表 10.52に、コンフィギュレータの設定項目の中で性能に影響を与える項目を示します。

表10.52 性能向上

ページ	項目	説明
[カーネル]ページ	CFG_PARCHK	チェックしないほうが、サービスコール処理時間が短くなります。
	CFG_KNLLVL	小さいほど、カーネルのクリティカルセクション実行中でもより多くの割込みレベルが受け付けられるようになります。
	CFG_PROTMEM	チェックしない場合、TLB ミスオーバヘッドは発生しなくなります。
	CFG_MEMCHK	チェックしないほうが、アドレスパラメータを持つサービスコール処理時間が短くなります。
[時間管理機能]ページ	CFG_TICNUME, CFG_TICDENO	タイムティックが大きいほど、タイマ割込みによる負荷が小さくなりますが、カーネルの時間管理の精度は粗くなります。
	CFG_OPTTMR	チェックした方が、時間管理処理を要求するサービスコール処理時間が長くなる可能性があります。一方、タイマ割込みの頻度は少なくなります。
[デバッグ機能]ページ	CFG_ACTION	チェックした場合、100msec 周期の周期ハンドラが実行されるため、それだけ負荷が増えます。
	CFG_TRACE	チェックした場合、全般に悪化します。
	CFG_TRCOBJCNT	大きいほど、全般に悪化します。
[パフォーマンス]ページ	CFG_PERFORM	チェックした場合、全般に悪化します。
[サービスコール選択]ページ	def_tex	チェックした場合、タスクスイッチ時間が悪化します。

11. ビルド

本章では、ターゲットに組み込む絶対アドレス形式のロードモジュールの生成方法について、提供するサンプルを例に解説します。

11.1 ロードモジュールの種類

(1) ロードモジュールの種類

本製品を組み込んだシステムは、以下の 3 種類のロードモジュールから構成されます。

(a) カーネルロードモジュール(knl_side)
カーネルロードモジュールには、以下のようなものが含まれます。
- カーネルのコード
- キャッシュサポート関数
- カーネルの静的確保の変数領域
- アプリケーションのコード・データ(必要なら)

(b) カーネル環境ロードモジュール(env_side)
カーネル環境ロードモジュールには、以下のようなものが含まれます。
- カーネルの静的確保の変数領域
- システムプール
- リソースプール
- 非タスクコンテキスト用スタック領域
- アプリケーションのコード・データ(必要なら)

(c) アプリケーションロードモジュール
文字通り、アプリケーションのみで構成されるロードモジュールです。アプリケーションロードモジュールは複数生成することもできます。逆に、全てのアプリケーションをカーネルロードモジュールまたはカーネル環境ロードモジュールに含めれば、アプリケーションロードモジュールは作成しなくても良いことになります。

カーネルを起動するには、knl_side と env_side の両方が必須です。

各ロードモジュールには、以下の依存関係があります。

カーネルロードモジュール
→カーネル環境ロードモジュール
→アプリケーションロードモジュール

即ち、カーネルロードモジュールを更新した場合は、カーネル環境ロードモジュールとアプリケーションロードモジュールの更新が必要です。同様に、カーネル環境ロードモジュールを更新した場合は、アプリケーションロードモジュールの更新が必要です。

この構成には、以下の利点があります。

- カーネルロードモジュールのみを ROM 化した後で、カーネル環境ロードモジュールとアプリケーションロードモジュールを変更できます。
- デバッグ対象をカーネル環境ロードモジュールまたはアプリケーションロードモジュールに含めることで、デバッグ時のビルド・ダウンロード時間の短縮が期待できます。

(2) コンフィギュレータの[カーネル側]チェックボックス

コンフィギュレータでタスクのアドレスや静的メモリオブジェクトなどの生成・定義時に、[カーネル側]チェックボックスをチェックして C 言語シンボルやセクション名を指定した場合は、それらの C 言語シンボルやセクションは、カーネルロードモジュールのリンク時に解決されなければなりません。同様に、[カーネル側]を指定しない場合は、それらの C 言語シンボルやセクションは、カーネル環境ロードモジュールのリンク時に解決されなければなりません。

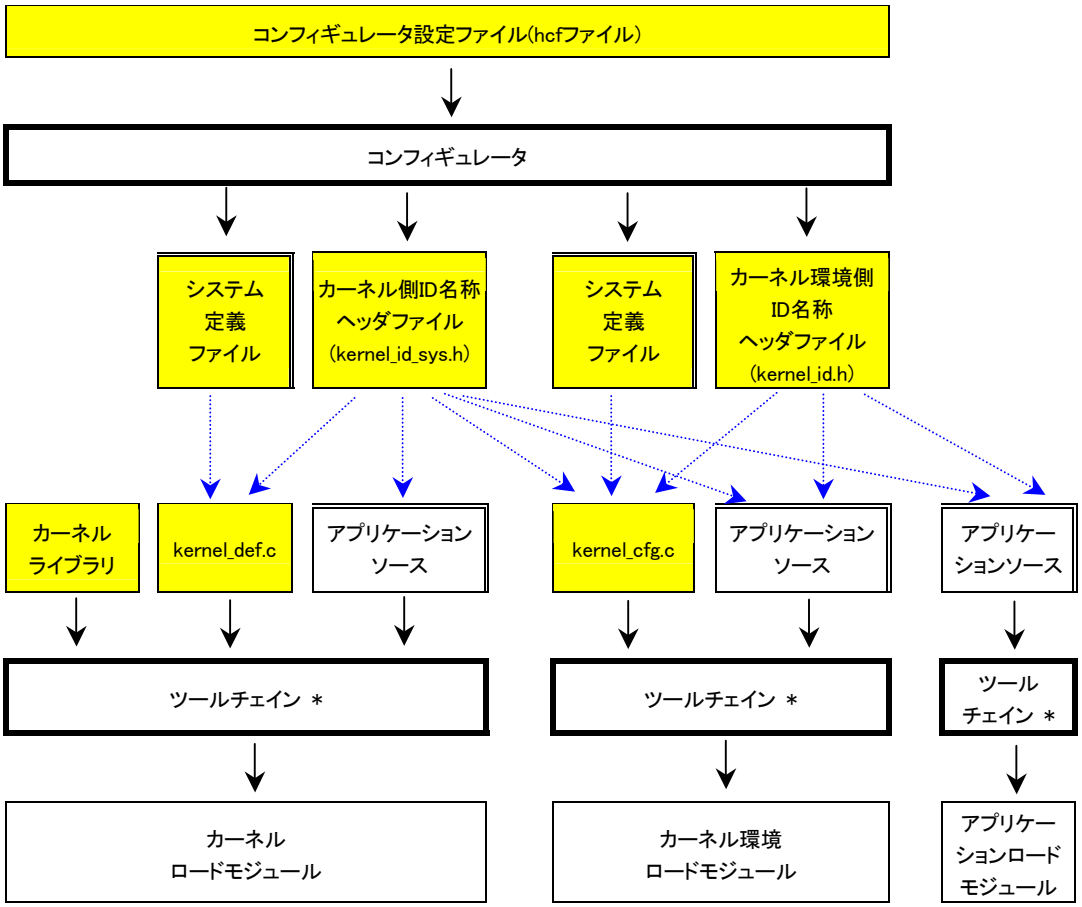
ただし、C 言語シンボルの場合は厳密にはシンボル実体をリンクしなくてもシンボル値の強制定義やシンボルファイルの入力など、リンク時にそのシンボルが解決できれば問題ありません。

(3) ID 名称ヘッダファイルに関する注意

カーネルロードモジュールをその他のロードモジュールに依存しないようにするために、カーネルロードモジュールに含めるアプリケーションでは、カーネル環境側に位置する kernel_id.h をインクルードしてはなりません。

(4) まとめ

図 11.1に、ここまでの内容をまとめましたので、参考にしてください。



：ファイルの入出力

：ファイルのインクルード

(黄色網掛)：本製品で提供するもの(サンプルとして提供するものを含む)

＊：コンパイラ、アセンブラ、最適化リンケージエディタなど

図11.1　ロードモジュールの生成

11.2 ディレクトリ構成

カーネルのインストールディレクトリの構成を、図 11.2に示します。

図11.2 ディレクトリ構成

(1) include¥

itron.h などのヘッダファイルが格納されています。

すべてのアプリケーションのコンパイル・アセンブル時に、インクルードパスとして指定するディレクトリです。

このディレクトリ内のファイルは変更しないでください。

(2) knl¥

カーネルのソースコードが格納されています。ソースコード付きの製品でのみ提供されます。

(3) lib¥

この下の elf¥にカーネルライブラリとキャッシュサポート関数のリロケータブルオブジェクトが格納されています。

(4) system¥

コンフィギュレータ出力ファイルをコンパイルするときに使用するシステム定義ファイルが格納されています。

このディレクトリ内のファイルは変更しないでください。

(5) samples¥

このディレクトリ以下に、サンプルファイル(サンプルプログラムや HEW ワークスペースなど)が格納されています。

11.3 サンプルシステムの概要

11.3.1 概要

samples¥以下に格納されているサンプルシステムには、以下のアプリケーションがあります。いずれも、シミュレータ使用時は標準ライブラリ関数を使用して I/O シミュレーションウィンドウにメッセージを表示します。

- ユーザドメイン 1(dom1)
  ドメインID=1のサンプルです。
  固定長メモリプールとメールボックスを使って、ドメイン内でデータ通信を行う例です。
- ユーザドメイン 2(dom2)、ユーザドメイン 3(dom3)
  それぞれ、ドメインID=2とドメインID=3のサンプルです。
  ドメイン間でのデータ通信を行う例で、メモリオブジェクト保護機能を使用する場合は保護メモリプールと保護メールボックス、メモリオブジェクト保護機能を使用しない場合は可変長メモリプールとメールボックスを使用します。
  ドメイン2が受信側、ドメイン3が送信側です。
- ユーザドメイン 4(dom4)
  ドメインID=4のサンプルです。
  意図的に不正アクセスを行います。
- ユーザドメイン 5(dom5)
  ドメインID=5のサンプルです。
  各種カーネルサービスコールを使った例で、一方のタスクが待ち状態になり、もう一方のタスクが待ち状態を解除します。
- アイドルタスク(idle)
  システムで最低優先度のタスクです。単に無限ループしています。
  アイドルタスクは、カーネルドメインに所属します。

また、シミュレータ専用のアプリケーションとして、以下があります。

- モニタ(monitor)
  シミュレータのI/Oシミュレーションウィンドウを使って、ユーザの入力に対して結果を表示します。コマンドとして、各種カーネルオブジェクト状態の参照などを要求できます。
  モニタタスクはカーネルドメインに所属します。

また、システムアプリケーションとして、以下があります。

- メモリアクセス違反ハンドラ
- システムダウンルーチン
- CPU 例外ハンドラ
- 割込み・例外フックルーチン

本サンプルシステムでは、標準ライブラリ関数を組み込んでいます。

samples¥以下のディレクトリ構成を図 11.3に示します。

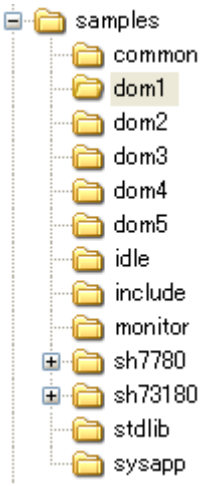


図11.3　samples¥以下のディレクトリ構成

(1) dom1¥

ドメイン 1 のサンプルソースが格納されています。

(2) dom2¥

ドメイン 2 のサンプルソースが格納されています。

(3) dom3¥

ドメイン 3 のサンプルソースが格納されています。

(4) dom4¥

ドメイン 4 のサンプルソースが格納されています。

(5) dom5¥

ドメイン 5 のサンプルソースが格納されています。

(6) idle¥

アイドルタスクが格納されています。

(7) include¥

サンプルシステム内での共通ヘッダが格納されています。

(8) stdlib¥

標準ライブラリを組み込むためのソースが格納されています。

(9) sysapp¥

各種システムアプリケーションが格納されています。

(10)*shnnnn*¥

SHnnnn マイコンに依存したサンプルソース、コンフィギュレータファイル、HEW ワークスペース・プロジェクトが格納されています。

11.3.2 使用するカーネルオブジェクト一覧

本サンプルシステムで使用するカーネルオブジェクトの一覧を以下に示します。

表11.1 タスク一覧

分類	ID	関数名	生成・起動方法	優先度	備考
dom1	ID_DOM1_MAIN *2	DOM1_Main()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	5	
	動的割当 *1	DOM1_Input()	DOM1_Main が生成・起動	6	
	動的割当 *1	DOM1_Output()	DOM1_Main が生成・起動	7	
dom2	ID_DOM2_MAIN *2	DOM2_Main()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	10	
dom3	ID_DOM3_Main *2	DOM3_Main()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	8	
dom4	ID_DOM4_Main *2	DOM4_Main()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	15	
dom5	ID_DOM5_Main *2	DOM5_Main()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成 *3	9	
	動的割当 *1	DOM1_Sub()	DOM5_Main が生成・起動	10	
アイドルタスク	ID_IDLETASK(20)	IdleTask()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	20	
モニタ	ID_MONITOR(19)	MonitorTask()	ｺﾝﾌｨｷﾞｭﾚｰﾀで生成・起動	1	シミュレータ使用時のみ

【注】 *1 acre_tsk サービスコールによる自動割当
*2 コンフィギュレータによる自動割当
*3 出荷時の状態では、生成されるだけで起動はされません。

表11.2 セマフォ一覧

分類	ID	生成方法
dom5	動的割当 *1	DOM5_Main が生成

【注】 *1 acre_sem サービスコールによる自動割当

表11.3 データキュー一覧

分類	ID	生成方法
dom5	動的割当 *1	DOM5_Main が生成

【注】 *1 acre_dtq サービスコールによる自動割当

表11.4 メールボックス一覧

分類	ID	生成方法	備考
dom1	動的割当 *1	DOM1_Main が生成	
dom2	ID_DOM2_MBX *2	コンフィギュレータで生成	メモリオブジェクト保護機能未使用時のみ

【注】 *1 acre_mbx サービスコールによる自動割当
*2 コンフィギュレータによる自動割当

表11.5 ミューテックス一覧

分類	ID	生成方法	備考
stdlib	ID_MTX_MALLOC(1)	コンフィギュレータで生成	
stdlib	ID_MTX_STRTOK(2)	コンフィギュレータで生成	
stdlib	ID_MTX_FILETBL(3)	コンフィギュレータで生成	シミュレータ使用時のみ
stdlib	ID_MTX_STDIO(4)	コンフィギュレータで生成	シミュレータ使用時のみ

表11.6　固定長メモリプール一覧

分類	ID	生成方法
dom1	動的割当 *1	DOM1_Main が生成

【注】 *1　acre_mpf サービスコールによる自動割当

表11.7　可変長メモリプール一覧

分類	ID	生成方法	備考
dom1	ID_DOM2_MPL *1	コンフィギュレータで生成	メモリオブジェクト保護機能未使用時のみ

【注】 *1　コンフィギュレータによる自動割当

表11.8　保護メモリプール一覧

分類	ID	生成方法	備考
dom2	ID_DOM2_MPP *1	コンフィギュレータで生成	メモリオブジェクト保護機能使用時のみ

【注】 *1　コンフィギュレータによる自動割当

表11.9　保護メールボックス一覧

分類	ID	生成方法	備考
dom2	ID_DOM2_MBP *1	コンフィギュレータで生成	メモリオブジェクト保護機能使用時のみ

【注】 *1　コンフィギュレータによる自動割当

表11.10　割込み・CPU 例外ハンドラ一覧

割込み・例外コード	関数名	定義方法	備考
H'0E0	samples¥sysapp¥exchdr.c の exception_handler()	コンフィギュレータで定義	・命令アドレスエラー ・データアドレスエラー(リード)
H'100		コンフィギュレータで定義	データアドレスエラー(ライト)
H'180		コンフィギュレータで定義	一般不当命令例外
H'1A0		コンフィギュレータで定義	スロット不当命令例外
H'E00	samples¥monitor¥monitor.c の MonitorWakeup()	MonitorTask が生成	モニタを起動用割込み(シミュレータ専用)
CFG_TIMINTNO	samples¥***shnnnn***¥kernel¥knl_side¥tmrdrv.c の_kernel_tmrint()	コンフィギュレータで定義	標準タイマドライバ

静的メモリオブジェクトについては、以下を参照してください。

■関連ページ　「11.15.4 メモリマップと静的メモリオブジェクト」

11.3.3 タスク例外処理機能

本サンプルシステムでは、カーネルのタスク例外処理機能を使用しています。

各タスクにタスク例外処理ルーチンを定義し、アクセス違反などの CPU 例外が発生した場合にそのタスクに対してタスク例外を要求します。タスク例外要求パターンは、不正アクセスされたアドレス(TEA レジスタの値)です。ただし、アドレス 0 の場合はタスク例外要求パターンとして不適切なので、この場合のみタスク例外処理パターンを H'ffffffff としています。

各タスクのタスク例外処理ルーチンでは、それを受けて CPU 例外を発生させた処理を中断します。

タスク例外の要求は、以下の中で行っています。

- メモリアクセス違反ハンドラ(メモリオブジェクト保護機能使用時のみ)(sysapp¥mavhdr.c)
- CPU 例外ハンドラ(sysapp¥exchdr.c)

サンプルシステムで実際にアクセス違反を発生させるのは、ドメイン 4 のみです。

詳細は、各ソースコードを参照してください。

11.4 各サンプルアプリケーション

各サンプルアプリケーションでは、シミュレータ使用時は printf()を用いて I/O シミュレーションウィンドウにメッセージを出力します。その他の標準ライブラリ関数は使用していません。

11.4.1 ユーザドメイン 1(dom1)

ドメイン ID=1 のサンプルで、固定長メモリプールとメールボックスを使って、ドメイン内でデータ通信を行う例です。

メインタスクが入力タスクと出力タスクを生成します。

入力タスクは固定長メモリプールからメモリブロックを取得し、そのメモリブロックにメッセージを作成してメールボックスに送信します。

出力タスクは、メールボックスからデータを受信し、そのメッセージ領域を固定長メモリプールに返却します。

以下のカーネルオブジェクトを使用します。

- メインタスク(dom1_main.c の DOM1_Main())
  ドメイン1で最初に実行されます。コンフィギュレータによって生成・起動されます。
- 入力タスク(dom1_input.c の DOM1_Input())
  メインタスクが生成・起動します。
- 出力タスク(dom1_output.c の DOM1_Output())
  メインタスクが生成・起動します。
- 固定長メモリプールとメールボックス
  メインタスクが生成します。

11.4.2 ユーザドメイン 2(dom2)

ドメイン ID=2 のサンプルで、ドメイン間でのデータ通信を行う例です。

メモリオブジェクト保護機能を使用する場合は保護メモリプールと保護メールボックス、メモリオブジェクト保護機能を使用しない場合は可変長メモリプールとメールボックスを使用します。

メインタスクは、保護メールボックスまたはメールボックスからメッセージを受信し、そのメッセージ領域を保護メモリプールまたは可変長メモリプールに返却します。

以下のカーネルオブジェクトを使用します。

- メインタスク(dom2.c の DOM2_Main())
  コンフィギュレータによって生成・起動されます。
- 保護メモリプールと保護メールボックス(メモリオブジェクト保護機能使用時)
  コンフィギュレータによって生成されます。
- 可変長メモリプールとメールボックス(メモリオブジェクト保護機能未使用時)
  コンフィギュレータによって生成されます。

11.4.3 ユーザドメイン 3(dom3)

ドメイン ID=3 のサンプルで、ユーザドメイン 2 に対してデータを送信します。

メモリオブジェクト保護機能を使用する場合は、ドメイン 2 の保護メモリプールからメモリブロックを取得し、そのメモリブロックにメッセージを作成して、ドメイン 2 の保護メールボックスに送信します。

メモリオブジェクト保護機能を使用しない場合は、ドメイン 2 の可変長メモリプールからメモリブロックを取得し、そのメモリブロックにメッセージを作成して、ドメイン 2 のメールボックスに送信

します。
以下のカーネルオブジェクトを使用します。

- メインタスク(dom3.c の DOM3_Main())
  コンフィギュレータによって生成・起動されます。

11.4.4 ユーザドメイン 4(dom4)

ドメイン ID=4 のサンプルです。
このサンプルでは、意図的に不正アドレスをアクセスします。メモリオブジェクト保護機能使用時は、この不正アクセスが検出され、メモリアクセス違反ハンドラによってタスク例外が要求されます。その後、タスク例外処理ルーチンが起動され、アクセス違反を引き起こした命令は再実行されません。
タスク例外処理ルーチンでは、自タスクを再起動し、再びアクセス違反を起こすようになっています。
以下のカーネルオブジェクトを使用します。

- メインタスク(dom4.c の DOM4_Main())
  コンフィギュレータによって生成・起動されます。

11.4.5 ユーザドメイン 5(dom5)

ドメイン ID=5 のサンプルです。
ドメイン 5 は、各種カーネル機能を使って、WAITING 状態とその解除を行う例になっています。
メインタスクがサブタスクを生成し、その後メインタスクが WAITING 状態に移行する以下のサービスコールを発行します。

- slp_tsk
- wai_sem
- wai_flg
- rcv_dtq

サブタスクは、メインタスクの WAITING 状態を解除する以下のサービスコールを発行します。

- wup_tsk
- sig_sem
- set_flg
- psnd_dtq

以下のカーネルオブジェクトを使用します。

- メインタスク(dom5.c の DOM5_Main())
  コンフィギュレータによって生成されますが、出荷時の状態では起動されません。起動するには、モニタで起動(ACTコマンドまたはSTAコマンド)、あるいは別途アプリケーションあるいはコンフィギュレータで起動する設定を行ってください。
- サブタスク(dom5.c の DOM5_Sub())
  DOM5_Mainによって生成・起動されます。
- セマフォ、イベントフラグ、データキュー
  DOM5_Mainによって生成されます。

11.5 システムアプリケーション

11.5.1 システムダウンルーチン(sysapp¥sysdwn.c)

システムダウンルーチン(_kernel_sysdwn())は、常にカーネルに組み込む必要があります。
サンプルのシステムダウンルーチンでは、単に無限ループしています。

■関連ページ　システムダウンルーチンの作成方法→「8.10 システムダウンルーチン」

11.5.2 メモリアクセス違反ハンドラ(sysapp¥mavhdr.c)

メモリアクセス違反ハンドラ(_kernel_mavhdr())は、メモリオブジェクト保護機能を使用する場合は、必ずカーネルに組み込む必要があります。
サンプルのメモリアクセス違反ハンドラでは、タスクコンテキストで発生したアクセス違反の場合はそのタスクに対してタスク例外を要求(iras_tex サービスコール)し、非タスクコンテキストの場合はシステムダウンするようになっています。
なお、条件コンパイルにより、メモリオブジェクト保護機能を使用しない場合は空のオブジェクトコードが生成されるようになっています。

■関連ページ　メモリアクセス違反ハンドラの作成方法→「8.9 メモリアクセス違反ハンドラ」

11.5.3 CPU 例外ハンドラ(sysapp¥exchdr.c)

本サンプルシステムでは、以下の例外コードに対して、exchdr.c の exception_handler()を CPU 例外ハンドラとして定義しています。

- 例外コード H'0E0：命令アドレスエラー、データアドレスエラー(リード)
- 例外コード H'100：データアドレスエラー(ライト)
- 例外コード H'180：一般不当命令例外
- 例外コード H'1A0：スロット不当命令例外

exception_handler()では、タスクコンテキストで発生した例外の場合はそのタスクに対してタスク例外を要求(iras_tex サービスコール)し、非タスクコンテキストの場合はシステムダウンするようになっています。

■関連ページ　CPU 例外ハンドラの作成方法→「8.8 CPU 例外ハンドラ」

11.5.4 割込み・例外フックルーチン(sysapp¥inthook.src)

割込み・例外フックルーチンを使用するかどうかは、コンフィギュレータの CFG_INTHOOK で指定します。本サンプルシステムではフックを使用するように設定しています。
本サンプルのフックルーチンでは、何もせずにリターンするようになっています。

■関連ページ　割込み・例外フックルーチンの作成方法→「8.5 割込み・例外フックルーチン」

11.6 CPU 依存部

CPU 依存部のソースは、samples¥*shnnnn*¥kernel¥knl_side¥に格納されています。以下のものがあります。

11.6.1 標準タイマドライバ(tmrdrv.c)

対象マイコンの内蔵タイマモジュール(通常はマイコン内蔵の TMU)用の標準タイマドライバです。

■関連ページ　標準タイマドライバの作成方法→「9. 標準タイマドライバ」

11.6.2 CPU リセット処理

CPU リセット処理として、以下のファイルがあります。

- reset.src
- resetprg.c
- init_mmu.c

reset.src は、CPU リセット直後に実行するプログラムで、アセンブリ言語記述です。スタックポインタの初期化やバスステートコントローラの初期化などを行った後、resetprg.c の PowerON_Reset_PC()を呼び出します。

PowerON_Reset_PC()では、MMU の初期化(init_mmu.c の InitializeMMU())やキャッシュの初期化を行った後、vsta_knl サービスコールによってカーネルを起動します。vsta_knl コール後は、リターンすることはありません。

通常、これらのファイルは、使用する環境に応じて修正が必要です。

11.7 標準ライブラリ関数と実行時ルーチン

11.7.1 概要

標準ライブラリ関数および実行時ルーチンは、「標準ライブラリ構築ツール」によってひとつのライブラリファイルに生成されます。つまり、一般にはリンク単位毎にライブラリファイルのリンク、および初期化が必要になります。

しかし本カーネルでは、複数リンク単位でシステムを構成できるように、標準ライブラリ関数についてはカーネルロードモジュール(knl_side)にのみ標準ライブラリ構築ツールによって生成されたライブラリをリンクすることとし、その他のリンク単位ではライブラリリンクを不要としています。具体的には、あらかじめシステムで使用するライブラリ関数を選択し、そのライブラリ関数のみをカーネルロードモジュール(knl_side)に組み込みます。その他のリンク単位では、カーネルロードモジュール生成時に出力したシンボルファイルを使用することで、ライブラリ関数へのアドレス参照を解決します。

一方、実行時ルーチンに関しては、コンパイル時点で必要な実行時ルーチンが決まるため、同様の手法をとることはできません。このため、実行時ルーチンのみのライブラリを生成(runtime.lib)し、各リンク単位でリンクします。

11.7.2 組み込む標準ライブラリ関数の選択

stdlib¥select.c で、システムに組み込むライブラリ関数を選択します。

この選択により、標準ライブラリ構築ツール生成ライブラリから、選択された関数だけが抽出されてカーネルロードモジュールに組み込まれます。

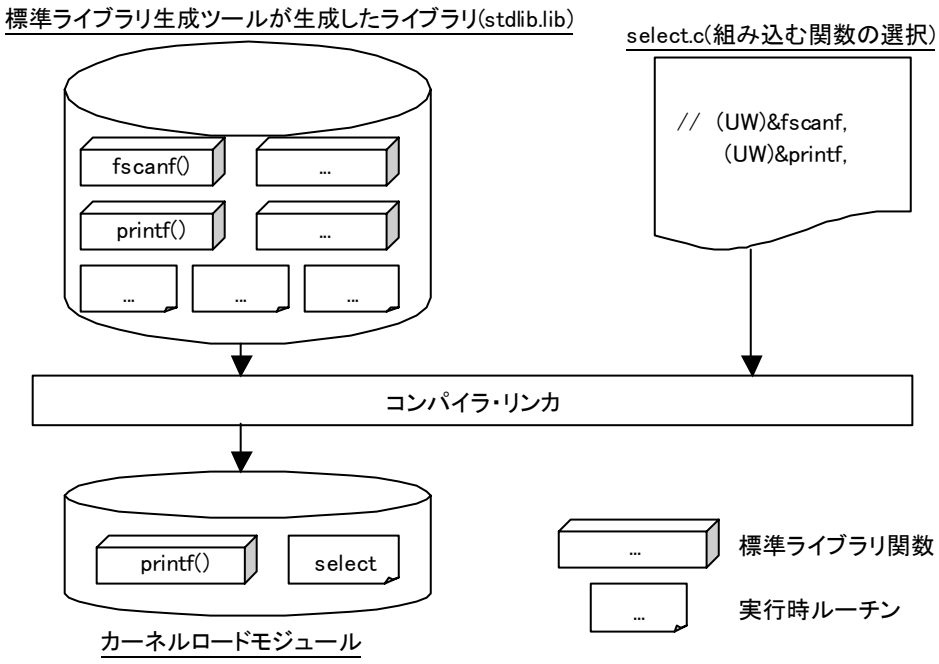


図11.4 標準ライブラリ関数の組み込みイメージ

11.7.3 stdio.h の扱い

本サンプルでは、シミュレータ使用時は"stdin", "stdio", "stderr"のみをサポートしています。これらは、シミュレータの I/O シミュレーションウィンドウとの入出力になります。

"stdin"からの入力時は、低水準インタフェースルーチンの charget()(samples¥stdlib¥lowsrc.c)の中でポーリングします。このため、入力があるまで全プログラムの実行が停止します。

シミュレータ未使用時は、stdio.h は未サポートです。

11.7.4 使用するカーネルオブジェクト

以下の ID 番号のミューテックスを使用します。これらは、いずれもコンフィギュレータで生成しています。

(1) ID_MTX_MALLOC(1)

標準ライブラリ関数をリエントラントライブラリとして構築した場合に必要となります。

malloc 系の排他制御に使用されます。

この ID 番号は、標準ライブラリ構築ツールの仕様によって定められています。

(2) ID_MTX_STRTOK(2)

標準ライブラリ関数をリエントラントライブラリとして構築した場合に必要となります。

strtok()の排他制御に使用されます。

この ID 番号は、標準ライブラリ構築のツール仕様によって定められています。

(3) ID_MTX_FILETBL(3)

標準ライブラリ関数をリエントラントライブラリとして構築した場合に必要となります。

__iob(ファイルテーブル)の排他制御に使用されます。

この ID 番号は、標準ライブラリ構築のツール仕様によって定められています。

前述のように、シミュレータ未使用時は stdio.h は未使用のため、このミューテックスは使用しません。

(4) ID_MTX_STDIO(4)

本サンプルシステムとして、I/O シミュレーションウィンドウの入出力が乱れないように、1 行単位でこのミューテックスをロック/解放するようにしています。

前述のように、シミュレータ未使用時は stdio.h は未使用のため、このミューテックスは使用しません。

11.7.5 標準ライブラリ関数を使用するために必要な関数

標準ライブラリ関数を使用するには、以下の関数の実装が必要です。

- ライブラリ関数の初期化
- 低水準インタフェースルーチン

これらは、stdlib¥lowsrc.c にて実装しています。

初期化関数は_INITLIB()です。

_INITLIB()は、コンフィギュレータで、初期化ルーチンとして定義しています。

11.7.6 標準ライブラリ関数の環境設定のカスタマイズ

samples¥include¥lowsrc.h の以下の箇所を、必要に応じて修正してください。

```
/*****************************************************************************
 * User setting
 *****************************************************************************/
#ifdef SIM
#define SIM_IO     4  /* Simulated I/O system call address */  ←(1)
#endif

#define IOSTREAM    3UL     /* Number of I/O Stream*/
#define HEAPSIZE    8192UL  /* for malloc() */  ←(2)
#define HEAPSIZE__X 1024UL  /* for mallox__X() */  ←(3)
#define HEAPSIZE__Y 1024UL  /* for mallox__Y() */  ←(4)

#ifdef _REENTRANT
#define MAXTSKID 20 /* Define maximum task ID which uses standard library */  ←(5)
#define TMOUT    300000L    /* Timeout when cannot lock mutex.  */  ←(6)
#endif
```

(1) I/Oシミュレーションで使用するシミュレータに対するシステムコールアドレスです。
(2) malloc()で使用するヒープサイズです。
(3) malloc__X()で使用するヒープサイズです。ただし、出荷時の構成ではmalloc__X()は未サポートです。
(4) malloc__Y()で使用するヒープサイズです。ただし、出荷時の構成ではmalloc__Y()は未サポートです。
(5) リエントラントライブラリとして標準ライブラリを構築した場合は、コンフィギュレータで指定するCFG_MAXTSKIDと同じ値を指定してください。
(6) リエントラントライブラリとして標準ライブラリを構築した場合に、指定してください。この値は、低水準インタフェースルーチンのwait_sem()でミューテックスをロックする際のタイムアウト値です。TMO_FEVR(-1)を指定すると永久待ちに、TMO_POL(0)を指定するとポーリングとなります。

11.7.7 標準ライブラリ関数に関する注意事項

リエントラントライブラリを使用する場合は、printf()や malloc()など、低水準ルーチン wait_sem()を使用するライブラリ関数は、非タスクコンテキストから呼び出してはなりません。呼び出した場合、ライブラリ関数がエラーリターンします。

11.7.8 セクション初期化関数(_INITSCT())

標準ライブラリ関数ではありませんが、セクション初期化関数が必要です。セクション初期化関数とは、B 属性のセクションの 0 クリア、およびリンカの ROM 化支援機能で指定されたセクションデータを、ROM(D 属性セクション)から RAM(R 属性セクション)にコピーする関数です。

標準ライブラリ構築ツールは、標準的なセクション初期化関数_INITSCT()をライブラリファイルに生成します。しかし、この_INITSCT()は、B 属性・D 属性・R 属性セクションのアドレス・サイズ情報を、それぞれ固定のセクション名に生成されたテーブル(C$BSEC セクション, "C$DSEC セクション")から取得する仕様となっています。このため、初期化対象のセクションが複数あり、それぞれのセクションに対するアクセス許可が異なる場合は、_INITSCT()をカーネルドメインでしか実行できない、という制約が付くことになります。

本サンプルシステムでは、これを避けるために、独自に_INITSCT()を実装しています。この _INTSCT()では、パラメータとしてセクション初期化情報を得るようになっています。

_INITSCT()は、stdlib¥initsct.c にあります。インタフェース仕様等の詳細は、ソースコードを参照してください。

11.7.9 実行時ルーチン

実行時ルーチンでは、初期化処理としてセクション初期化が必要です。

(1) knl_side 以外のリンク単位

knl_side 以外のリンク単位で実行時ルーチンが必要な場合は、runtime.hwp によって生成される runtime.lib と、common¥init_runtime.c をリンクします。init_runtime.c の init_runtime()が、runtime.lib のセクション初期化関数です。

init_runtime()は、そのリンク単位で最初に実行されるようにしてください。

本サンプルシステムでは、以下のようにしています。

- env_side：init_runtime()を初期化ルーチンとしてコンフィギュレータに登録しています
- app_dom5: ドメイン 5 のエントリ関数(DOM5_Main())の先頭で init_runtim()を呼び出しています。

(2) knl_side のリンク単位

knl_side では、stdlib.lib から実行時ルーチンがリンクされます。_INITLIB()の中で実行時ルーチンのセクション初期化も実施されるため、init_rintime.c は不要です。

11.8 モニタ

11.8.1 概要

モニタは、シミュレータの I/O シミュレーションウィンドウを介して、ユーザ入力コマンドを処理するプログラムです。シミュレータでのみ機能します。実機では動作しません。

以下のカーネルオブジェクトを使用します。

- モニタタスク(monitor.c の MonitorTask 関数)
  コンフィギュレータによって生成・起動されます。
- モニタの割込みハンドラ(monitor.c の MonitorWakeup 関数)
  モニタタスクが定義します。

モニタでは、標準ライブラリ関数を使用しています。

11.8.2 モニタの動作

システムを起動すると、モニタタスクが起動され、I/O シミュレーションウィンドウに起動メッセージが表示されます。その後、モニタタスクは WAITING 状態に移行します。この状態では、コマンドは入力できません。

モニタをコマンド入力可能な状態にするには、H'FE0 の割込みコードの割込みを発生させます。出荷時の HEW デバッグセッションでは、[トリガボタン]にこの割込みが設定されているので、このボタンを押すことでコマンド入力待ちになります。

コマンド入力待ちになると、"MON>"というプロンプトが表示されます。

コマンド入力待ち・コマンド処理中は、モニタは ID_MTX_SIMIO のミューテックスをロックします。つまり、この間は他のタスクは I/O シミュレーションウィンドウへの入出力はできなくなります。

Q コマンドを入力すると、モニタは ID_MTX_SIMIO を解放し、割り込み発生待ち状態に戻ります。

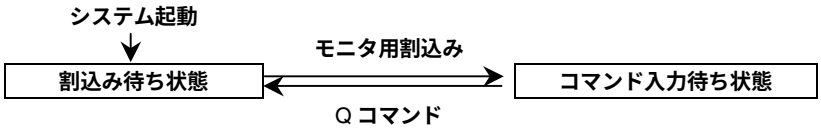


図11.5 モニタの状態遷移

11.8.3 モニタ用割込みの変更

割込み番号を変更するには、monitor.h の以下の部分を変更してください。

```
/************************************************************************
 * User setting
 ************************************************************************/
/*** Please define interrupt number to wake-up monitor ***/
#define INTNO_MONITOR 0xfe0UL    ←使用する割込み番号に変更
```

トリガボタンは、[トリガ]ウィンドウのポップアップメニューから[設定]を選択して開く[トリガ設定]ダイアログボックスで設定します。以下の条件で設定してください。

- 割込み条件：上記 INTNO_MONITOR と同じ値
- 割込み優先順位：1～CFG_KNLLVL の間の値

図11.6　トリガ設定ダイアログボックス

11.8.4 モニタのコマンド

コマンドシンタックスは以下の通りです。

```
MON> <コマンド名>[△<パラメータ 1>△<パラメータ 2>...]
```

各コマンドのシンタックス詳細は、"？△<コマンド名>"で確認してください。

コマンド	機能
TSK	**タスク状態の表示**
ACT	**タスクの起動**(act_tsk **サービスコール使用**)
STA	**タスクの起動**(sta_tsk **サービスコール使用**)
TER	**タスクの強制終了**(ter_tsk **サービスコール使用**)
SEM	**セマフォ状態の表示**
FLG	**イベントフラグ状態の表示**
DTQ	**データキュー状態の表示**
MBX	**メールボックス状態の表示**
MTX	**ミューテックス状態の表示**
MBF	**メッセージバッファ状態の表示**
MPF	**固定長メモリプール状態の表示**
MPL	**可変長メモリプール状態の表示**
TIM	**現在のシステムクロックの表示**
CYC	**周期ハンドラ状態の表示**
ALM	**アラームハンドラ状態の表示**
OVR	**オーバーランハンドラ状態の表示**
MPP	**保護メモリプール状態の表示**
MBP	**保護メールボックス状態の表示**
RPL	**リソースプール状態の表示**
SYP	**システムプール状態の表示**
MEM	**メモリオブジェクト状態の表示**
PRB	**メモリ領域のアクセスチェック**
?	**ヘルプ**
Q	**割込み待ち状態に復帰する**

■注意

(1) パラメータに数値を入力する場合は、"0x"を付加した場合は16進数、付加しない場合は10進数と扱います。

(2) バックスペースの入力は認識しません。

11.9 HEW ワークスペースとプロジェクト

11.9.1 概要

samples¥***shnnnn***¥以下には、SHnnnn マイコンに依存したソースのほかに、ロードモジュールを生成するための HEW ワークスペースとプロジェクトが格納されています。

本サンプルシステムでは、2 つのワークスペースを使って 3 つの絶対アドレス形式ロードモジュールを生成します。

ワークスペースのひとつは、カーネル部分のロードモジュールを生成する samples¥***shnnn***¥kernel¥kernel.hws、もう一方はドメイン 5 のみから構成されるロードモジュールを生成する samples¥***shnnnn***¥app_dom5¥app_dom5.hws です。

図 11.7に、各ワークスペースとプロジェクトの関係を示します。

kenrel.hws には、以下のプロジェクトがあります。

- knl_side.hwp
  カーネル側のロードモジュールを生成します。このロードモジュールには、図11.7に示すような構成要素を含めます。
- knl_side_sym.hwp
  カーネル側ロードモジュールのシンボルファイルのオブジェクトを生成します。このオブジェクトを他のリンク時に入力することで、カーネル側のシンボルを参照することができます。
- env_side.hwp
  カーネル環境側のロードモジュールを生成します。このロードモジュールには、図11.7に示すような構成要素を含めます。
- runtime.hwp
  実行時ルーチンのみからなるライブラリを生成します。このライブラリは、knl_side以外のリンク時にリンクします。このライブラリを使用する場合は、common¥init_runtime.cをリンクし、最初にこの中のinit_runtime()が実行されるようにしなければなりません。

app_dom5.hws には、プロジェクトとして app_dom5.hwp のみがあります。このプロジェクトでは、ドメイン 5 のみから構成されるロードモジュールを生成します。

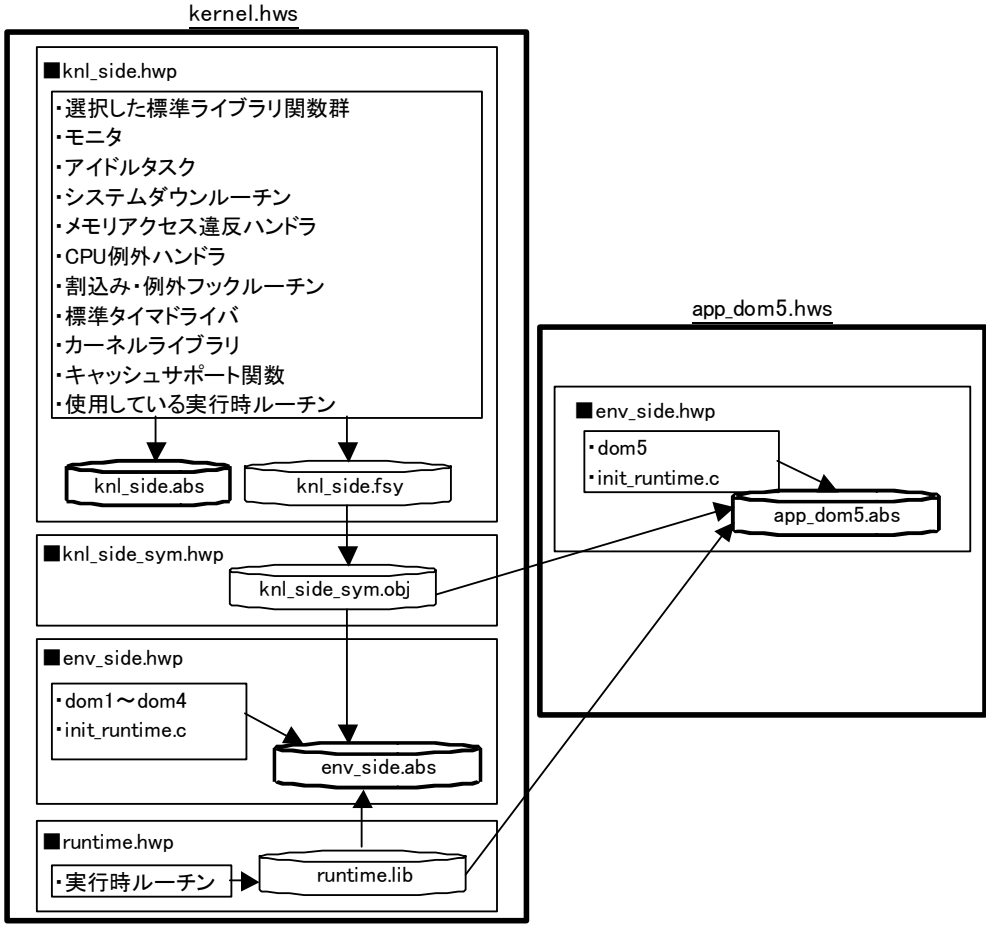


図11.7　HEW ワークスペース、プロジェクトの構成

11.9.2　ワークスペースのディレクトリ構成

shnnnn ディレクトリとして、V.1.01 では表 11.11に示すものを提供しています。使用するマイコンがこの中に無い場合は、一番近いマイコン用のディレクトリを選択して、必要な変更を加えてください。

マイコン依存のソースは、samples¥***shnnnn***¥kernel¥knl_side¥に格納されています。

表11.11　*shnnnn*

shnnnn	CPU コア	対象マイコン	ワークスペース作成時に使用した HEW(コンパイラパッケージ)バージョン
sh73180	SH4AL-DSP	SH73180	4.00.02(9.00 Release 03)
sh7343	SH4AL-DSP（拡張機能あり）	SH7343	4.00.02 (9.00 Release 03)
sh7780	SH-4A	SH7780	4.00.02 (9.00 Release 03)
sh7785	SH-4A（拡張機能あり）	SH7785	4.00.02 (9.00 Release 03)

図 11.8に、samples¥*shnnnn*¥以下のディレクトリ構成を示します。

図11.8　samples¥*shnnnn*¥以下のディレクトリ構成

kernel¥knl_side¥には、当該マイコン用の以下のサンプルソースが格納されています。

- reset.src, resetprg.c, init_mmu.c：CPU リセット処理
- tmrdrv.c：標準タイマドライバ

11.9.3 HEW のビルドコンフィギュレーションとコンフィギュレータファイルのディレクトリ

各プロジェクトは、全て同じ名称の HEW コンフィギュレーションを設定してあります。

コンフィギュレータ設定もこのコンフィギュレーションに連動するため、コンフィギュレータファイル(拡張子 hcf)およびその出力ファイルも、コンフィギュレーション毎に ***shnnnn***¥config_out¥の下の HEW コンフィギュレーションと同じ名称のディレクトリに格納するようにしています。各プロジェクトでは、HEW のプレースホルダ"$(CONFIGNAME)"(コンフィギュレーション名)を用いて、各コンフィギュレーション毎のコンフィギュレータ出力ディレクトリを指定するようになっています。

各コンフィギュレーションの相違は、表 11.12に示す通りです。

表11.12　HEW コンフィギュレーション

コンフィギュレーション	メモリオブジェクト保護機能	シミュレータの I/O シミュレーションウィンドウ、モニタの使用	メモリマップ *1
noprot	未使用	なし	ボード上の RAM にダウンロードすることを想定した配置
noprot_sim *2	未使用	あり	ROM 化を想定した配置
port	使用	なし	ボード上の RAM にダウンロードすることを想定した配置
prot_sim *2	使用	あり	ROM 化を想定した配置

【注】 *1　詳細は、「11.15 メモリ配置」を参照してください。

*2　V.1.00 Release 00 では、***sh7780***(SH7780 用)では、このコンフィギュレーションは提供していません。

シミュレータ用のコンフィギュレーション"prot_sim", "noprot_sim"では、コンパイルオプションに"-def=SIM"を指定するようになっています。samples¥stdlib¥lowsrc.c では、この条件に応じて表 11.13に示すように条件コンパイルしています。

表11.13　"-def=SIM"オプションによる相違

"-def=SIM"指定	stdio.h サポート
あり	"stdin", "stdout", "stder"のみサポート。これらは、シミュレータの I/O シミュレーションウィンドウに対する入出力となります。
なし	未サポート

また、共通ヘッダの samples¥include¥sim_printf.h では、各ドメインで使用する sim_printf()を、"-def=SIM"の指定がある場合は printf()に、無い場合は空文に定義しています。

11.9.4 HEW ワークスペースの移動

カーネルインストールディレクトリ以下の構造を保ったまま移動するようにしてください。

サンプルワークスペースでは、インクルードパスやライブラリファイルなどを、カーネルインストールディレクトリ下の範囲内で、各種ファイル・ディレクトリのパスを相対パスで設定(HEW のプレースホルダを使用)しています。

11.9.5 ビルド時のオプション設定について

(1) 全プロジェクトで統一が必要なオプション

コンパイラのマニュアルを参照し、必要なオプションの設定を統一してください。

「SuperH™ RISC engine C/C++コンパイラ、アセンブラ、最適化リンケージエディタ (コンパイラパッケージ V.9.00) ユーザーズマニュアル」(RJJ10B0156-0100H)では、「9.4.3 プログラム開発上の注意事項」に関連する記載があります。

(2) インクルードパス

以下をインクルードパスに設定します。出荷時は、プレースホルダ「$(WORKSPDIR)」(ワークスペースディレクトリ)からの相対パスで、あらかじめこれらが設定されています。

- include¥ : itron.h など、カーネルの標準ヘッダが格納されています。
- samples¥include : サンプルシステムで使用する共通ヘッダが格納されています。
- config_out¥$(CONFIGNAME):該当する HEW コンフィギュレーション用のコンフィギュレータ出力ファイルが格納されています。

(3) リエントラントライブラリ

標準ライブラリをリエントラントライブラリとして構築した場合は、ライブラリ関数を使用するソースでは、コンパイルオプション"-def=_REENTRANT"を指定する必要があります。本サンプルシステムでは、全ファイルにこれを指定しています。

(4) シミュレータ用コンフィギュレーション("prot_sim", "noprot_sim")

コンパイルオプション"-def=_SIM"を指定しています。詳細は、「11.9.3 HEW のビルドコンフィギュレーションとコンフィギュレータファイルのディレクトリ」を参照してください。

(5) エンディアン

出荷時の構成では、big/little いずれかの設定になっています。必要に応じて変更してください。

knl_side のリンク時に指定するカーネルライブラリとキャッシュサポートオブジェクトは、エンディアンによってファイル名が異なるので、エンディアン変更時はこの設定も変更する必要があります。

(6) その他のオプション

その他のオプション設定については、基本的にはサンプルプロジェクトの設定を参照してください。

11.10 kernel.hws の knl_side.hwp プロジェクト

11.10.1 概要

本プロジェクトでは、以下のファイルを生成します。

(1) knl_side.abs

kenrel¥knl_side¥$(CONFIGNAME)¥に生成します。

knl_side.abs には、本製品提供のカーネルライブラリとキャッシュサポートオブジェクト、標準ライブラリ関数などが含まれます。

(2) knl_side.fsy

kenrel¥knl_side¥$(CONFIGNAME)¥に生成します。

knl_side.fsy は、knl_side.abs の中で他のリンク単位へ公開するシンボルアドレスが定義されたアセンブリ言語ソースファイルです。このファイルは、knl_side_sym.hwp プロジェクトでアセンブルされ、そのオブジェクトが他のリンク単位でリンクされます。

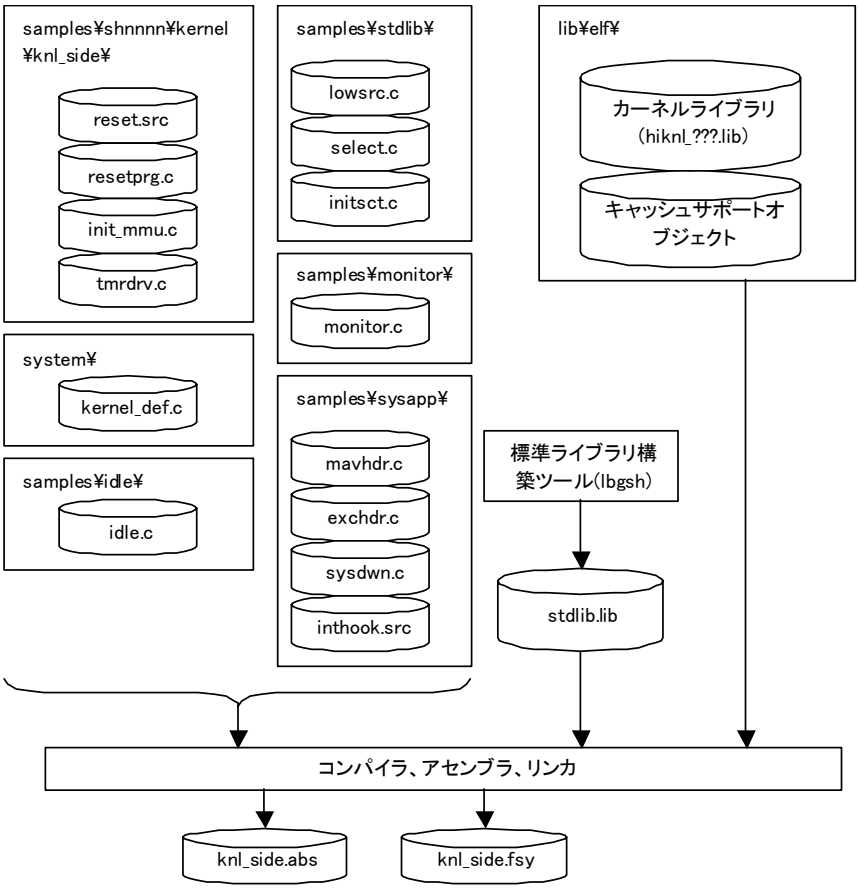


図11.9 knl_side.hwp の概要

11.10.2 プロジェクトに登録するソース

プロジェクトに登録するソースには、以下があります。

(1) system¥kernel_def.c[必須]
コンフギュレータ出力ファイルのうち、カーネル側の情報を取り込むファイルです。

(2) samples¥sysapp¥sysdwn.c[必須]
システムダウンルーチンです。必須です。

(3) samples¥sysapp¥mavhdr.c
メモリアクセス違反ハンドラです。コンフィギュレータでメモリオブジェクト保護機能を使用する設定になっている場合は必須です。なお、使用しない設定になっている場合は、コンパイルしても空のオブジェクトとなります。

(4) samples¥sysapp¥inthook.src
割込み・例外フックルーチンです。コンフィギュレータで割込み・CPU例外フックを使用する設定になっている場合は必須です。

(5) samples¥sysapp¥exchdr.c
CPU例外ハンドラです。本サンプルでは、コンフィギュレータで例外コードH'0E0, 100, 180,1A0に対して、このCPU例外ハンドラを定義しています。

(6) samples¥***shnnnn***¥kernel¥knl_side¥reset.src, resetprg.c, init_mmu.c
CPUリセット処理のサンプルです。

(7) samples¥***shnnnn***¥kernel¥knl_side¥tmrdrv.c
標準タイマドライバのサンプルです。コンフィギュレータで標準ドライバを使用する設定になっている場合は必須です。なお、最適化タイマドライバを使用する設定になっている場合は、コンパイルしても空のオブジェクトとなります。

(8) samples¥stdlib¥lowsrc.c, select.c, initsct.c
標準ライブラリを使用する場合は登録してください。

(9) samples¥monitor¥monitor.c
シミュレータ使用時でモニタを使用する場合は登録してください。

(10) samples¥idle¥idle.c
アイドルタスクです。

その他、コンフィギュレータで[カーネル側]を指定したオブジェクトのシンボルおよびセクションも、本プロジェクトでリンクする必要があります。

11.10.3 標準ライブラリ構築ツールの設定

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[標準ライブラリ]タブを選択してください。

[カテゴリ]から[標準ライブラリ]を選択すると、図 11.10に示す画面になります。ここでは、標準ライブラリ構築ツールが生成するライブラリファイルに含めるライブラリ関数を選択します。samples¥stdlib¥select.c で選択した関数が包含されるように指定してください。包含していないと、リンク時に select.c が参照するライブラリ関数が見つからずにリンクエラーになります。

図11.10　標準ライブラリ構築ツールの設定([標準ライブラリ]カテゴリ)

次に、[カテゴリ]で[オブジェクト]を選択すると、図 11.11に示す画面になります。通常は[リエントラントライブラリを生成]をチェックしてください。

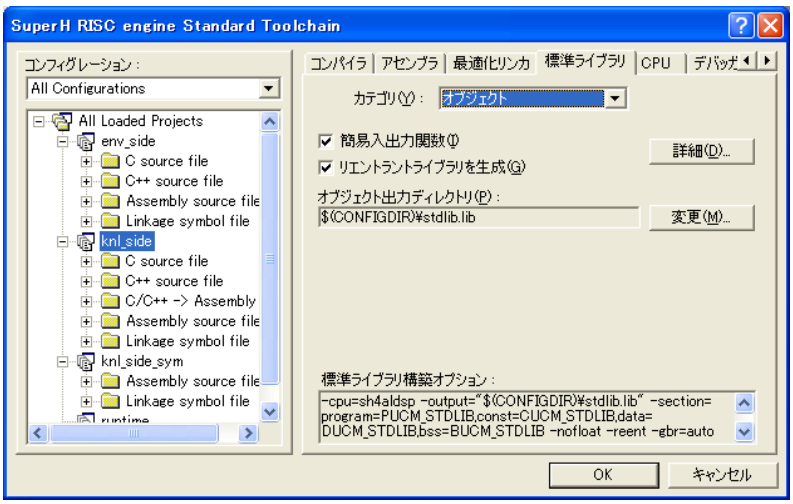


図11.11　標準ライブラリ構築ツールの設定([オブジェクト]カテゴリ)

さらに[詳細]ボタンを押すと、図 11.12に示すダイアログボックスが表示されます。

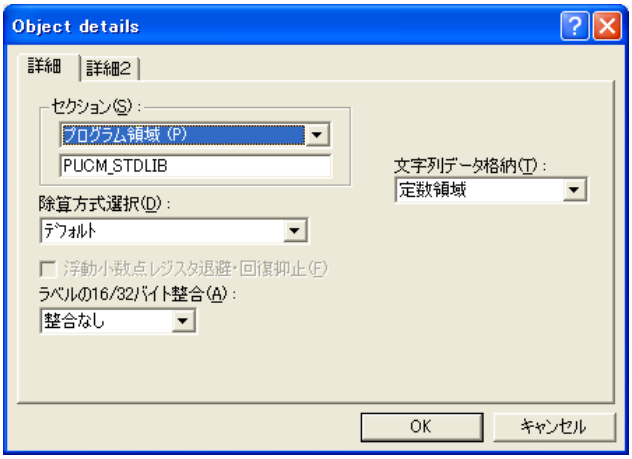


図11.12　標準ライブラリ構築ツールの設定([Object details]ダイアログボックス)

この中の[セクション]グループで、標準ライブラリ関数のセクション名を設定します。以下のように設定します。samples¥stdlib¥以下のソースでも、これと同じセクション名を#pragma section 文を用いて指定しています。ここでのセクション名を変更する場合は、前述のソースの記述も変更してください。

- [プログラム領域(P)]：PUCM_STDLIB
- [定数領域(C)]：CUCM_STDLIB
- [初期化データ領域(D)]：DUCM_STDLIB
- [未初期化データ領域(B)]：BUCM_STDLIB

11.10.4 リンカの設定

(1) カーネルライブラリの指定

必ず、以下のいずれかのカーネルライブラリとリンクします。カーネルライブラリは、lib¥elf¥に格納されています。

- $(WORKSPDIR)¥..¥..¥..¥lib¥elf¥hiknl_big.lib：ビッグエンディアン用
- $(WORKSPDIR)¥..¥..¥..¥lib¥elf¥hiknl_little.lib：リトルエンディアン用

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[入力]、[オプション項目]として[ライブラリファイル]を選択し、図 11.13のようにカーネルライブラリを指定してください。

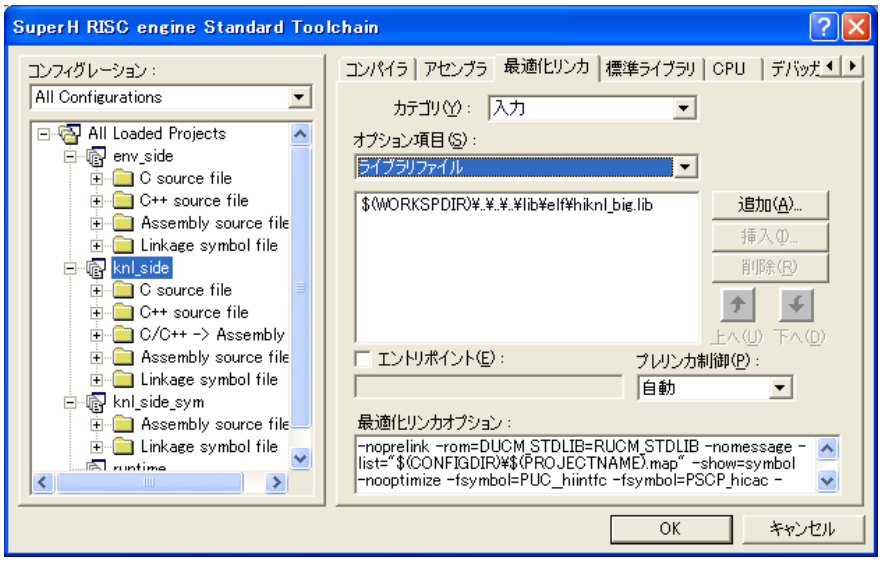


図11.13　リンカの設定(カーネルライブラリの指定)

(2) キャッシュサポートオブジェクトの指定

キャッシュサポート関数をシステムに組み込む場合は、リンカにキャッシュサポートオブジェクトを入力します。キャッシュサポートオブジェクトは、lib¥elf¥に格納されています。

- $(WORKSPDIR)¥..¥..¥..¥lib¥elf¥cache_sh4a_big.rel：SH-4A/SH4AL-DSP 用、ビッグエンディアン用
- $(WORKSPDIR)¥..¥..¥..¥lib¥elf¥cache_sh4a_little.rel：SH-4A/SH4AL-DSP 用、リトルエンディアン用

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[入力]、[オプション項目]として[リロケータブルファイル/オブジェクトファイルライブラリファイル]を選択し、図 11.14のようにキャッシュサポートオブジェクトを指定してください。

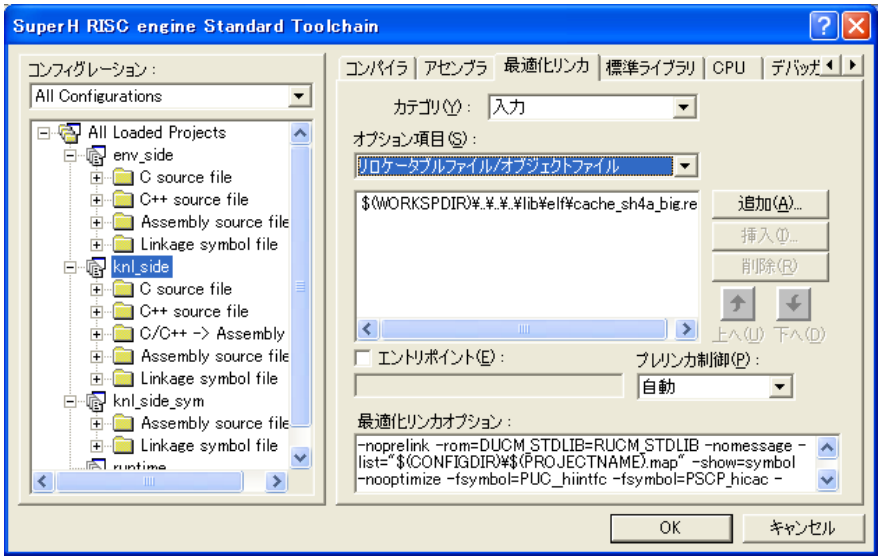


図11.14　リンカの設定(キャッシュサポートオブジェクトの指定)

(3) 初期化データセクションに関する設定

リンクするオブジェクトの中に初期化データセクション(D セクション)がある場合は、リンカの設定で[ROM から RAM にマップするセクション]の指定(R セクションの生成)が必要です。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[出力]を選択し、[オプション項目]から[ROM から RAM にマップするセクション]を選択してください。

出荷時の構成では、以下の初期化データセクションがあります。

- DSCP_MON(モニタ)("prot_sim", "noprot_sim"コンフィギュレーションのみ)
- DUCM_STDLIB(標準ライブラリ)

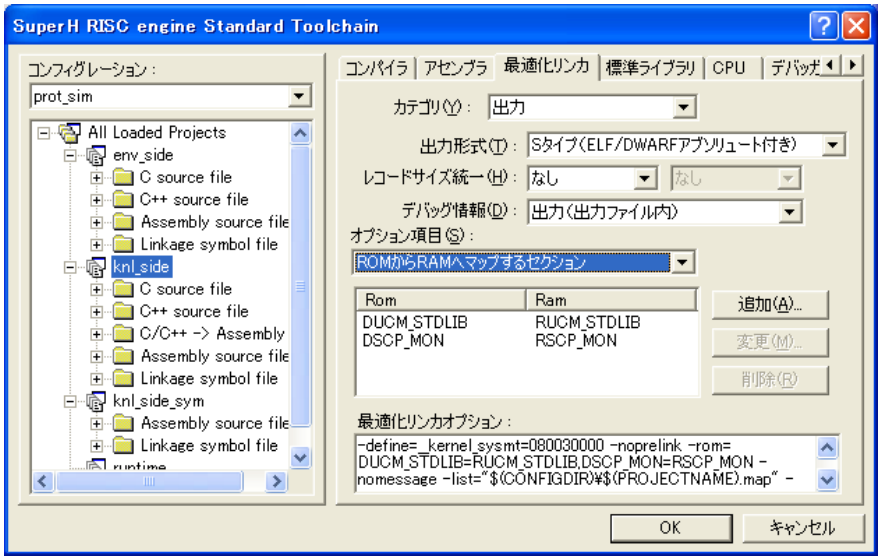


図11.15　リンカの設定(初期化データセクション)

(4) シンボルアドレスファイル(knl_side.fsy)の出力

knl_side 以外に公開したいシンボルを持つセクションを指定します。そのようなセクションとして、以下があります。

- PUC__hiintfc：カーネルサービスコール(指定必須)
- PSCP_hicac：キャッシュサポート関数
- PUCM_STDLIB：標準ライブラリ関数、セクション初期化関数(_INITSCT())

リンカは、ここで指定したセクション内の外部定義シンボルのアドレスが記述されたアセンブリ言語ソースファイル knl_side.fsy を knl_side.abs と同じディレクトリ(通常は$(CONFIGDIR))に生成します。このファイルは、knl_side_sym.hwp で利用されます。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[セクション]を選択し、[設定項目]から[シンボルアドレスファイル]を選択してください。

図11.16　リンカの設定(シンボルアドレスファイルの出力)

(5) __kernel_sysmt のアドレス定義

__kernel_sysmt とは、カーネル環境側(env_side)で生成されるカーネルの情報テーブルで、PSCP_hisysmt セクションの先頭アドレスです。PSCP_hisysmt セクションの配置アドレスは、システム設計時にあらかじめ決定しておき、そのアドレスを指定します。

■関連ページ　「11.15.4(2)(a) 論理アドレス H'80030000 のセクションブロック(CSCP_hisysmt など)」

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。
左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。
[カテゴリ]として[入力]を選択し、[オプション項目]から[シンボル定義]を選択してください。

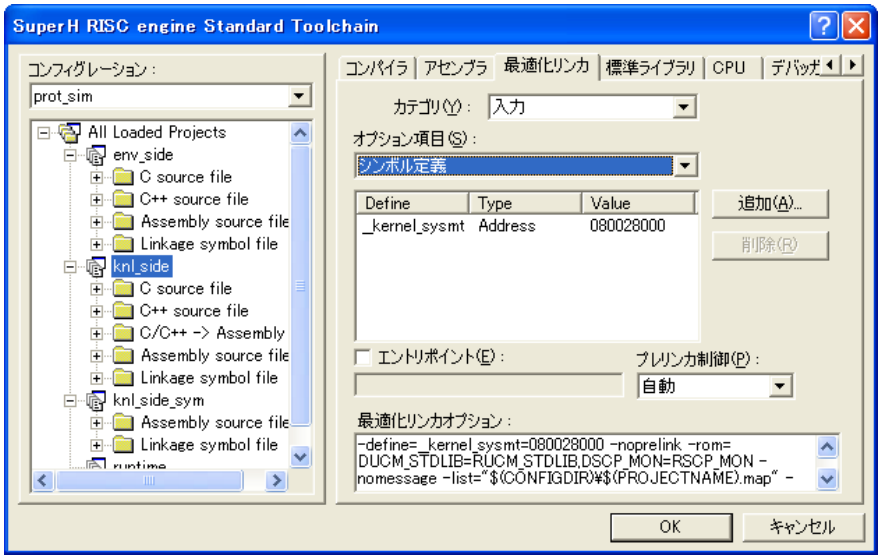


図11.17　リンカの設定(__kernel_sysmt アドレスの定義)

(6) セクションの配置

セクションの配置については、「11.15 メモリ配置」を参照してください。

11.10.5　ビルドの実行

ビルドを実行すると、$(WORKSPDIR)¥knl_side¥$(CONFIGNAME)¥に、knl_side.abs と knl_side.fsy が生成されます。

11.11 kernel.hws の knl_side_sym.hwp プロジェクト

本プロジェクトでは、knl_side.hwp で生成されたシンボルファイルのオブジェクトを、kernel_out¥以下のコンフィギュレーション名と同じ名称のディレクトリに knl_side_sym.obj という名称で生成します。このオブジェクトは、knl_side 以外のリンク時に knl_side 内のシンボル参照を解決するために使用します。

本プロジェクトの処理内容は、以下の通りです。

(1) 前回のビルド時の(3)で生成されたknl_side_sym.objを削除します。これは、プロジェクトディレクトリにあるDelFile.batによって実施します。

(2) knl_side.hwpによって$(WORKSPDIR)¥knl_side¥$(CONFIGNAME)¥に生成されたknl_side.fsyを、プロジェクトディレクトリにknl_side_sym.fsyというファイル名でコピーします。これは、プロジェクトディレクトリにあるCopyFile.batによって実施します。

(3) コピーされたknl_side_sym.fsyをアセンブルし、そのオブジェクトファイルknl_side_sym.objを$(WORKSPDIR)¥kernel_out¥$(CONFIGNAME)¥に生成します。

11.12 kernel.hws の runtime.hwp プロジェクト

11.12.1 概要

本プロジェクトでは、標準ライブラリ構築ツールを使用して、実行時ルーチンのみが格納されたライブラリファイル runtime.lib を kernel_out¥以下のコンフィギュレーション名と同じ名称のディレクトリに生成します。ソースファイルはありません。

runtime.lib は、knl_side 以外のリンク時に実行時ルーチンをリンクするために使用します。

11.12.2 標準ライブラリ構築ツールの設定

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"runtiime"プロジェクトを選択し、[標準ライブラリ]タブを選択してください。

[カテゴリ]から[標準ライブラリ]を選択し、図 11.18に示すように実行時ルーチンのみを選択してください。

図11.18　標準ライブラリ構築ツールの設定([標準ライブラリ]カテゴリ)

次に、[カテゴリ]で[オブジェクト]を選択し、[詳細]ボタンを押すと、図 11.11に示すダイアログボックスが表示されます。

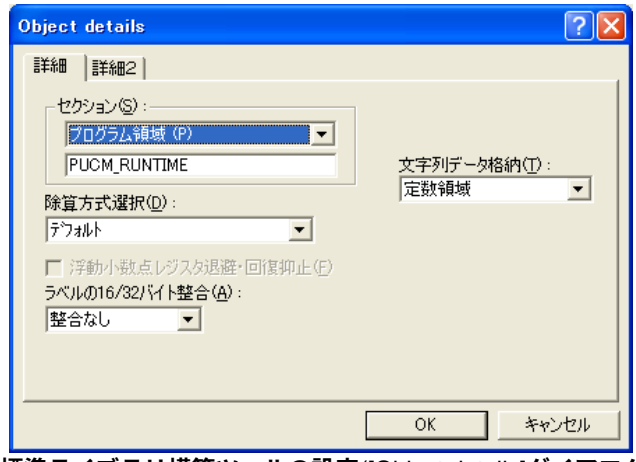


図11.19　標準ライブラリ構築ツールの設定([Object details]ダイアログボックス)

この中の[セクション]グループで、実行時ルーチンのセクション名を設定します。以下のように設定します。

- [プログラム領域(P)]：PUCM_RUNTIME
- [定数領域(C)]：CUCM_RUNTIME
- [初期化データ領域(D)]：DUCM_RUNTIME
- [未初期化データ領域(B)]：BUCM_RUNTIME

次に、[カテゴリ]で[オブジェクト]を選択し、[変更]ボタンを押し、図 11.20に示すように出力ファイルを$(WORKSPDIR)¥kernel_out¥$(CONFIGNAME)¥以下に runtime.lib という名称で生成するように設定してください。

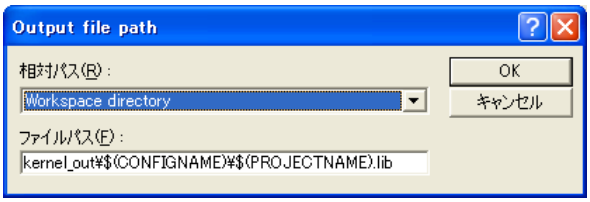


図11.20　標準ライブラリ構築ツールの設定([Output file path]ダイアログボックス)

11.12.3　ビルドの実行

ビルドを実行すると、$(WORKSPDIR)¥kernel_out¥$(CONFIGNAME)¥runtime.lib が生成されます。

11.12.4　セクション初期化に関する注意

「11.7.9 実行時ルーチン」を参照してください。

11.13 kernel.hws の env_side.hwp プロジェクト

11.13.1 概要

本プロジェクトでは、kenrel¥env_side¥$(CONFIGNAME)¥に env_side.abs を生成します。

env_side.abs には、ドメイン 1～4 が含まれます。また、knl_side にあるシンボルの参照を解決するための knl_side_sym.obj、および実行時ルーチンをリンクするための runtime.lib、実行時ルーチンを初期化するための init_runtime.c をリンクします。

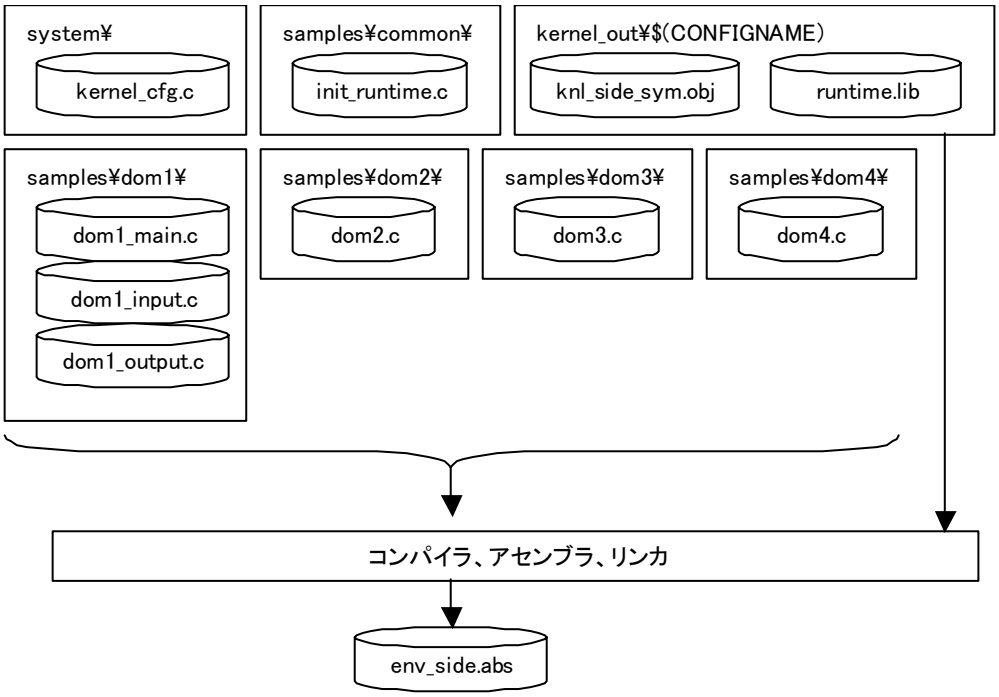


図11.21　env_side.hwp のビルドフェーズ

11.13.2 プロジェクトに登録するソース

プロジェクトに登録するソースには、以下があります。

(1) system¥kernel_cfg.c[必須]
コンフギュレータ出力ファイルのうち、カーネル環境側の情報を取り込むファイルです。

(2) samples¥common¥init_runtime.c
実行時ルーチン(runtime.lib)の初期化関数です。

(3) samples¥dom1¥dom1_main.c, dom1_input.c, dom1_output.c
ドメイン1のサンプルです。

(4) samples¥dom2¥dom2.c
ドメイン2のサンプルです。

(5) samples¥dom3¥dom3.c
ドメイン3のサンプルです。

(6) samples¥dom4¥dom4.c
ドメイン4のサンプルです。

その他、コンフィギュレータで[カーネル側]を指定しなかったオブジェクトのシンボルおよびセクションも、本プロジェクトでリンクする必要があります。

11.13.3 標準ライブラリ構築ツールの設定

kernel.hws の runtime.hwp で生成した runtime.lib を指定します。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"env_side"プロジェクトを選択し、[標準ライブラリ]タブを選択してください。

[カテゴリ]から[モード]を選択し、図 11.22に示すように kenrel_out¥$(CONFIGNAME)¥runtime.lib を指定してください。

図11.22 標準ライブラリ構築ツールの設定(runtime.lib の指定)

11.13.4 リンカの設定

(1) knl_side.sym.obj

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[入力]、[オプション項目]として[リロケータブルファイル/オブジェクトファイル]を選択し、図 11.23のように$(WORKSPDIR)¥kernel_out¥$(CONFIGNAME)¥knl_side_sym.obj を指定してください。

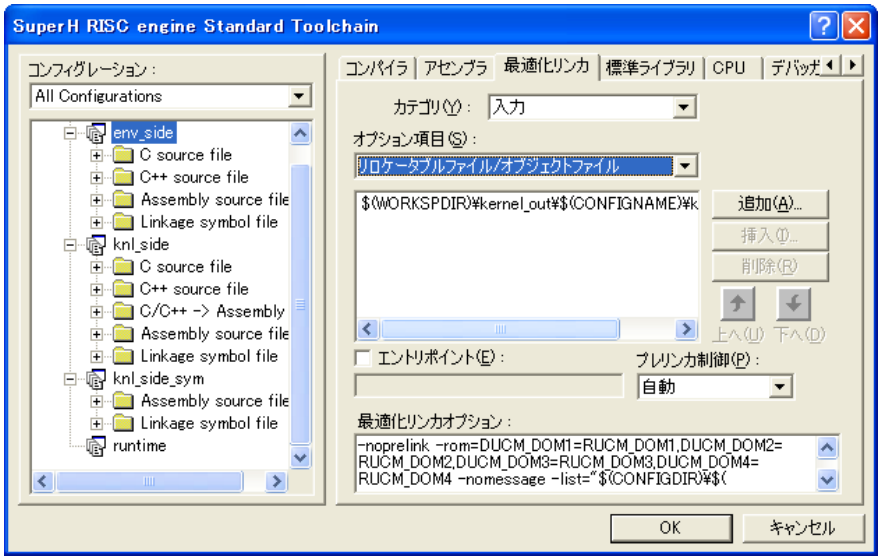


図11.23 リンカの設定(knl_side_sym.obj の指定)

(2) 初期化データセクションに関する設定

リンクするオブジェクトの中に初期化データセクション(D セクション)がある場合は、リンカの設定で[ROM から RAM にマップするセクション]の指定(R セクションの生成)が必要です。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"env_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[出力]を選択し、[オプション項目]から[ROM から RAM にマップするセクション]を選択してください。

出荷時の構成では、以下の初期化データセクションがあります。

- DUCM_DOM1(ドメイン 1)
- DUCM_DOM2(ドメイン 2)
- DUCM_DOM3(ドメイン 3)
- DUCM_DOM4(ドメイン 4)
- DUCM_RUNTIME(runtime.lib)

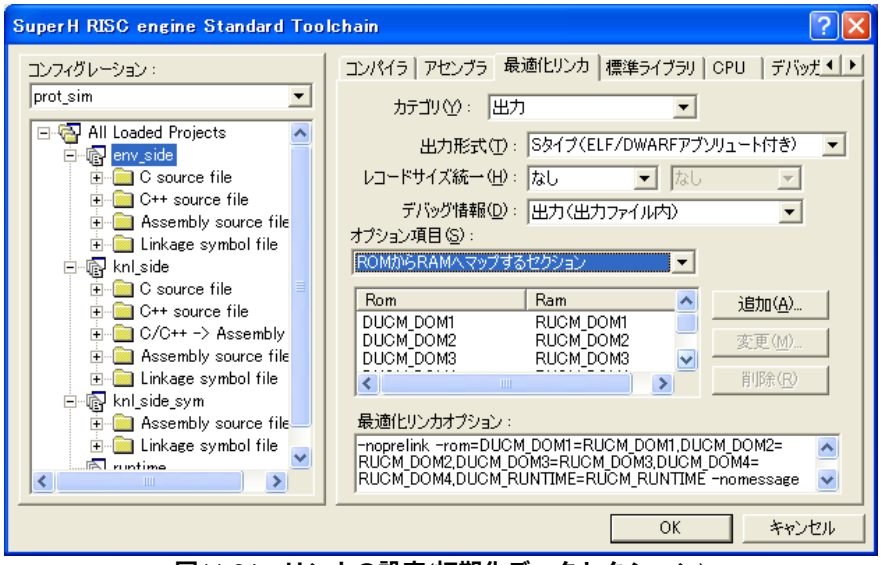


図11.24　リンカの設定(初期化データセクション)

(3) セクションの配置

セクションの配置については、「11.15 メモリ配置」を参照してください。

11.13.5 ビルドの実行

ビルドを実行すると、$(WORKSPDIR)¥env_side¥$(CONFIGNAME)¥env_side.abs が生成されます。

11.14 app_dom5.hws の app_dom5.hwp プロジェクト

11.14.1 概要

本ワークスペースは、アプリケーションのみで構成されるロードモジュールを生成する例です。

本プロジェクトでは、app_dom5.abs を生成します。

app_dom5.abs には、ドメイン 5 が含まれます。また、knl_side にあるシンボルの参照を解決するための knl_side_sym.obj、および実行時ルーチンをリンクするための runtime.lib、実行時ルーチンを初期化するための init_runtime.c をリンクします。

11.14.2 プロジェクトに登録するソース

プロジェクトに登録するソースには、以下があります。

(1) samples¥common¥init_runtime.c
実行時ルーチン(runtime.lib)の初期化関数です。

(2) samples¥dom5¥dom5.c
ドメイン5のサンプルです。

11.14.3 標準ライブラリ構築ツールの設定

kernel.hws の runtime.hwp で生成した runtime.lib を指定します。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"env_side"プロジェクトを選択し、[標準ライブラリ]タブを選択してください。

[カテゴリ]から[モード]を選択し、図 11.25に示すように $(WORKSPDIR)¥..¥kernel¥kenrel_out¥$(CONFIGNAME)¥runtime.lib を指定してください。

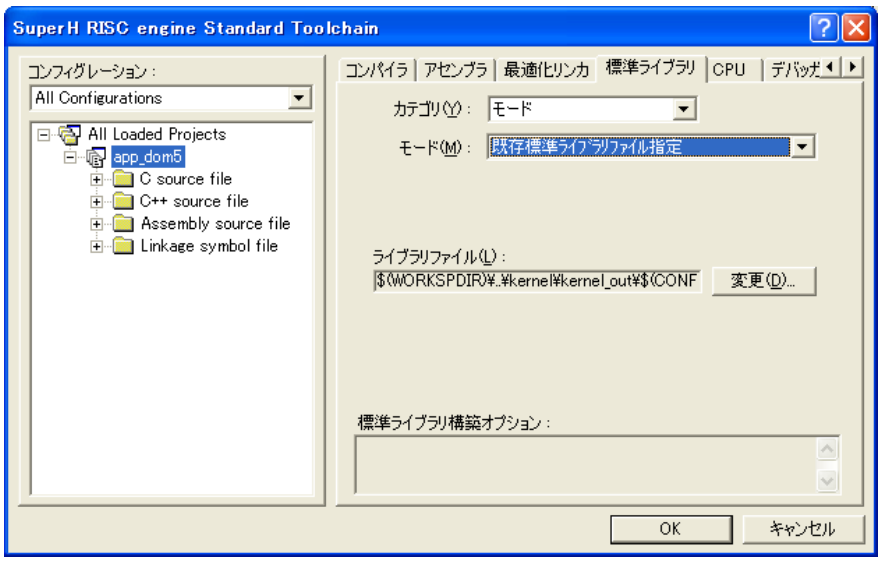


図11.25 標準ライブラリ構築ツールの設定(runtime.lib の指定)

11.14.4 リンカの設定

(1) knl_side.sym.obj の指定

knl_side_sym で生成されたシンボルファイル knl_side_sym.obj を指定します。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

左側の画面で"knl_side"プロジェクトを選択し、[最適化リンカ]タブを選択してください。

[カテゴリ]として[入力]、[オプション項目]として[リロケータブルファイル/オブジェクトファイル]を選択し、図 11.23のように$(WORKSPDIR)¥..¥kernel¥kenrel_out¥$(CONFIGNAME)¥knl_side_sym.obj を指定してください。

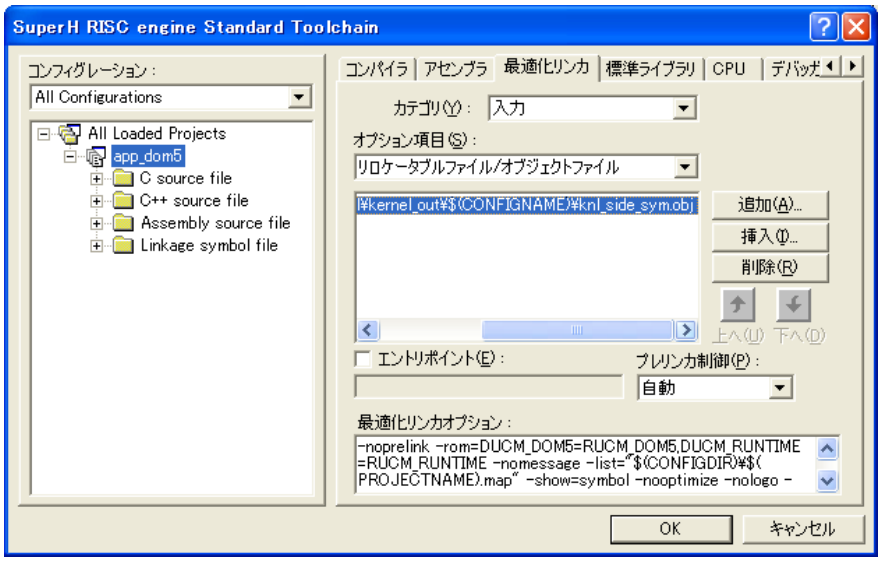


図11.26 リンカの設定(knl_side_sym.obj の指定)

(2) 初期化データセクションに関する設定

リンクするオブジェクトの中に初期化データセクション(D セクション)がある場合は、リンカの設定で[ROM から RAM にマップするセクション]の指定(R セクションの生成)が必要です。

HEW のメニューから[オプション->SuperH RISC engine Standard Toolchain...]を選択し、[SuperH RISC engine Standard Toolchain]ダイアログボックスを開きます。

[最適化リンカ]タブを選択してください。

[カテゴリ]として[出力]を選択し、[オプション項目]から[ROM から RAM にマップするセクション]を選択してください。

出荷時の構成では、以下の初期化データセクションがあります。

- DUCM_DOM5(ドメイン 1)
- DUCM_RUNTIME(runtime.lib)

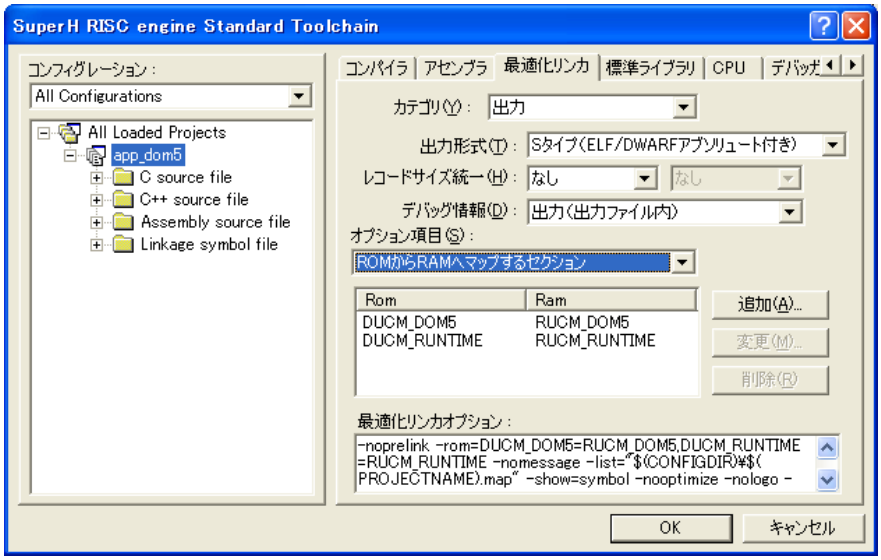


図11.27　リンカの設定(初期化データセクション)

(3) セクションの配置

セクションの配置については、「11.15 メモリ配置」を参照してください。

11.14.5 ビルドの実行

ビルドを実行すると、$(WORKSPDIR)¥app_dom5$(CONFIGNAME)¥app_dom5.abs が生成されます。

11.15 メモリ配置

11.15.1 概要

本サンプルシステムでは、複数のロードモジュールによってひとつのシステムを構成しています。したがって、どのロードモジュールのどのセクションがどのアドレスを使用するかを検討・決定し、それに従って、それぞれのロードモジュールのセクション配置アドレスをリンカに設定します。

セクション配置を指定せずに仮リンクすれば、そのマップファイルから各セクションのサイズを得ることができます。

リンカでのセクション配置は論理アドレスで指定しますが、もちろんどの物理メモリに割り付けるかを意識する必要があります。通常は、P,C,D セクションと、B,R セクションを分けて配置します。P,C,D セクションは ROM 化可能です。

■関連ページ　「5.論理アドレス空間の扱い」

11.15.2 セクション一覧

セクション名は、以下のルールで付与しています。

ＰＳＣＰ＿ｈｉｋｎｌ

(1) 1文字目
- P：プログラムセクション
- C：定数セクション
- B：未初期化データセクション
- D：初期化データセクション(ROM部)
- R：初期化データセクション(RAM部、リンカの[ROMからRAMにマップするセクション]で生成されるセクション)

(2) 2文字目
- S：ユーザモードからアクセスすることは無い
- U：ユーザモードからアクセスする場合がある

(3) 3文字目
- C：キャッシャブルアクセス可能
- D：キャッシャブルアクセス不可

(4) 4文字目(メモリオブジェクト保護機能使用時のみ意味があります)

(a) メモリオブジェクト保護機能使用時
- P：MMU非対象領域で、かつユーザモードからアクセスできない領域に配置しなければなりません。
- M：MMU対象領域に配置しなければなりません。
- _：ユーザモードからアクセス可能な領域に配置しなければなりません。MMU対象領域かどうかは問いません。

(b) メモリオブジェクト保護機能未使用時
- P：ユーザモードからアクセスできない領域に配置しなければなりません
- M：そのセクションをどこからアクセスするかによって、ユーザモードからアクセスできない領域に配置するかどうかを決定してください。
- _：ユーザモードからアクセス可能な領域に配置しなければなりません。

-

(1) カーネルライブラリ(knl_side.hwp)

表11.14　hiknl_big.lib, hiknl_little.lib のセクション

セクション名	説明
PSCP_hireset	vsta_knl
PSDP_hiknl	MMUCR レジスタなどを操作するプログラム
PSCP_hiexp	主に、割込み・TLB ミス発生時に実行されるプログラム
PSCP_hicom	サービスコールで共通的に実行されるプログラム
PUC__hiintfc	サービスコールのエントリー関数 全ユーザドメインからリードアクセス可能でなければなりません。
PSCP_hiknl	その他

(2) キャッシュサポートオブジェクト(knl_side.hwp)

表11.15　cache_sh4a_big.rel, cache_sh4a_little.rel

セクション名	説明
PSDP_hicac	CCR レジスタなどを操作するプログラム
PSCP_hicac	その他のプログラム
BSCP_hicac	管理テーブル

(3) system¥kernel_def.c(knl_side.hwp)

表11.16　kernel_def.c のセクション

セクション名	説明
PSCP_hidef	オブジェクト初期登録を処理する関数など
CSCP_hidef	−
BSCP_hidef	−
BSCP_himpp_<ID>	保護メモリプール([ｶｰﾈﾙ側]のチェックがあるもの) <ID>は、ID 名称がある場合は ID 名称、ない場合は ID 番号の 10 進表現

(4) system¥kernel_cfg.c(env_side.hwp)

表11.17　kernel_cfg.c のセクション

セクション名	説明
PSCP_hicfg	オブジェクト初期登録を処理する関数など
CSCP_hidef	−
CSCP_hisysmt	カーネル環境側構築情報テーブル
BSCP_hintskstk	非タスクコンテキスト用スタック領域
BUCM_hisyspl	システムプール
BSCP_hirespl	リソースプール
BSCP_hiwrk	カーネル作業領域
BSCP_hitrcbuf	ターゲットトレース用バッファ
BSCP_hitooltrc	ツールトレース用領域
BSCP_hictxid	常に 4 バイト
BSCP_hicfg	−
BUCM_himpp_<ID>	保護メモリプール([カーネル側]のチェックがないもの) <ID>は、ID 名称がある場合は ID 名称、ない場合は ID 番号の 10 進表現

(5) サンプルプログラム

表11.18　システムアプリケーションのセクション

リンク単位	セクション名	ファイル
knl_side	PSCP_hiknl	samples¥sysapp¥mavhdr.c samples¥sysapp¥sysdwn.c samples¥sysapp¥exchdr.c samples¥sysapp¥inthook.src
	BSCP_hiknl	samples¥sysapp¥sysdwn.c

表11.19　ターゲット依存部のセクション

リンク単位	セクション名	ファイル
knl_side	PSDP_RESET　*	samples¥***shnnnn***¥kernel¥knl_side¥reset.src
	PSDP_RESET2	samples¥***shnnnn***¥kernel¥knl_side¥resetprg.c samples¥***shnnnn***¥kernel¥knl_side¥init_mmu.c
	PSCP_hiknl	samples¥***shnnnn***¥kernel¥knl_side¥tmrdrv.c
	CSCP_hiknl	samples¥***shnnnn***¥kernel¥knl_side¥tmrdrv.c

【注】 ROM 化する場合は、CPU のリセットアドレスである H'A0000000 に配置してください。

表11.20　標準ライブラリのセクション

リンク単位	セクション名	ファイル
knl_side	PUCM_STDLIB	stdlib.lib(標準ライブラリ構築ツール生成)
	CUCM_STDLIB	samples¥stdlib¥lowsrc.c
	BUCM_STDLIB	samples¥stdlib¥initsct.c
	DUCM_STDLIB	
	RUCM_STDLIB	samples¥stdlib¥lowsrc.c

表11.21　モニタのセクション

リンク単位	セクション名	ファイル
knl_side	PSCP_MON	samples¥monitor¥monitor.c
	CSCP_MON	
	BSCP_MON	
	DSCP_MON	
	RSCP_MON	

表11.22　アイドルタスクのセクション

リンク単位	セクション名	ファイル
knl_side	PSCP_IDLE	samples¥idle¥idle.c

表11.23　ドメイン 1 のセクション

リンク単位	セクション名	ファイル
env_side	PUCM_DOM1	samples¥dom1¥dom1_main.c
	CUCM_DOM1	samples¥dom1¥dom1_input.c
	BUCM_DOM1	samples¥dom1¥dom1_output.c
	DUCM_DOM1	
	RUCM_DOM1	

表11.24　ドメイン 2 のセクション

リンク単位	セクション名	ファイル
env_side	PUCM_DOM2	samples¥dom2¥dom2.c
	CUCM_DOM2	
	BUCM_DOM2	
	DUCM_DOM2	
	RUCM_DOM2	

表11.25　ドメイン 3 のセクション

リンク単位	セクション名	ファイル
env_side	PUCM_DOM3	samples¥dom3¥dom3.c
	CUCM_DOM3	
	BUCM_DOM3	
	DUCM_DOM3	
	RUCM_DOM3	

表11.26　ドメイン 4 のセクション

リンク単位	セクション名	ファイル
env_side	PUCM_DOM4	samples¥dom4¥dom4.c
	CUCM_DOM4	
	BUCM_DOM4	
	DUCM_DOM4	
	RUCM_DOM4	

表11.27　ドメイン 5 のセクション

リンク単位	セクション名	ファイル
app_dom5	PUCM_DOM5	samples¥dom5¥dom5.c
	CUCM_DOM5	
	BUCM_DOM5	
	DUCM_DOM5	
	RUCM_DOM5	

表11.28　runtime.lib のセクション

リンク単位	セクション名	ファイル
knl_side 以外	PUCM_RUNTIIME	runtime.lib(標準ライブラリ構築ツール生成)
	CUCM_RUNTIIME	samples¥common¥init_runtime.c
	BUCM_RUNTIIME	
	DUCM_RUNTIIME	
	RUCM_RUNTIIME	

表11.29　knl_side_sym.obj のセクション

リンク単位	セクション名	ファイル
knl_side 以外	P　*	samples¥***shnnnn***¥kernel_out¥$(CONFIGNAME)¥knl_side_sym.obj

【注】　このセクションのサイズは 0 バイトです。プログラムでこのセクションへアクセスすることは無いため、リンク時には任意のアドレスに配置してください。

11.15.3 注意事項

(1) メモリオブジェクト保護機能使用時

静的メモリオブジェクト、システムプール(BUCM_hisyspl セクション)、保護メモリプールのセクション配置に関して、以下の注意があります。

- 先頭アドレスは、静的メモリオブジェクトの場合は指定したページ境界、システムプールと保護メモリプールはデフォルトのページサイズ(4kB)境界でなければなりません。
- その最終アドレスから次のページサイズ境界までの範囲には、何もデータを配置してはなりません。
- MMU 対象領域に配置しなければなりません。

(2) メモリオブジェクト保護機能未使用時

システムプール(BUCM_hisyspl セクション)は、32 バイト境界に配置しなければなりません。そうでない場合は、可変長メモリプールをシステムプールから割り付ける場合に、VTA_ALIGN16, VTA_ALIGN32 属性が指定されていても、メモリブロックのアドレスが期待したアライメントにならなくなります。また、システムプールはユーザモードからアクセス可能な領域に配置しなければなりません。

11.15.4 メモリマップと静的メモリオブジェクト

各サンプルでは、シミュレータ使用のコンフィギュレーション("prot_sim", "noprot_sim")と、シミュレータ無しのコンフィギュレーション("prot", "noprot")で、メモリマップが異なります。

表11.30 各コンフィギュレーションのメモリマップ

コンフィギュレーション	セクションを配置する物理メモリ *	
	P,C,D セクション	B,R セクション
"prot_sim", "noprot_sim"	0 番地から	H'0C000000 から (外部 RAM を想定したアドレス)
"prot", "noprot"	外部 RAM を想定したアドレス	外部 RAM を想定したアドレス

【注】 具体的なアドレスは、サンプルプロジェクトの設定を参照してください。

"prot_sim", "noprot_sim"では、シミュレータで[リセット後 Go]でシステムを起動できるようになっています。

一方、"prot", "noprot"では、ターゲットボードに搭載されている外部 RAM にダウンロードして実行することを想定したメモリマップになっています。

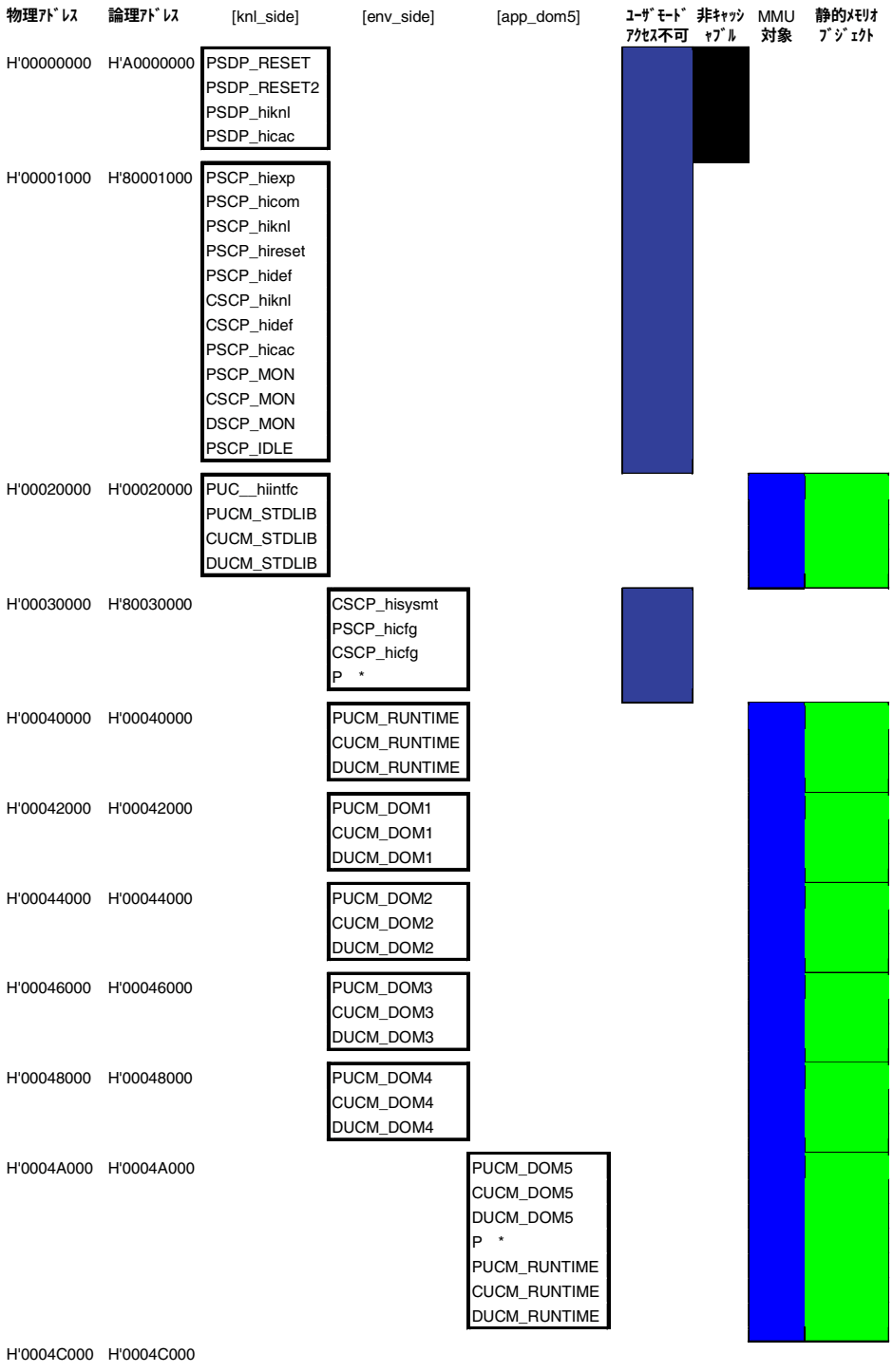
なお、いずれのコンフィギュレーションでも、内蔵メモリは使用していません。

サンプルシステムで使用するメモリサイズは、概ね以下の通りです。

- P,C,D セクション：約 310kB
- B,R セクション：約 440kB

以降では、SH73180 用の"prot_sim", "noprot_sim"でのメモリマップを例に説明します。

図 11.28に P, C, D セクションの配置、図 11.29に、B, D セクションの配置を示します。



【注】シンボルファイル knl_side_sym.obj **は、サイズ** 0 **の** P **セクションを持ちます。**

図11.28　P, C, D **セクションの配置**

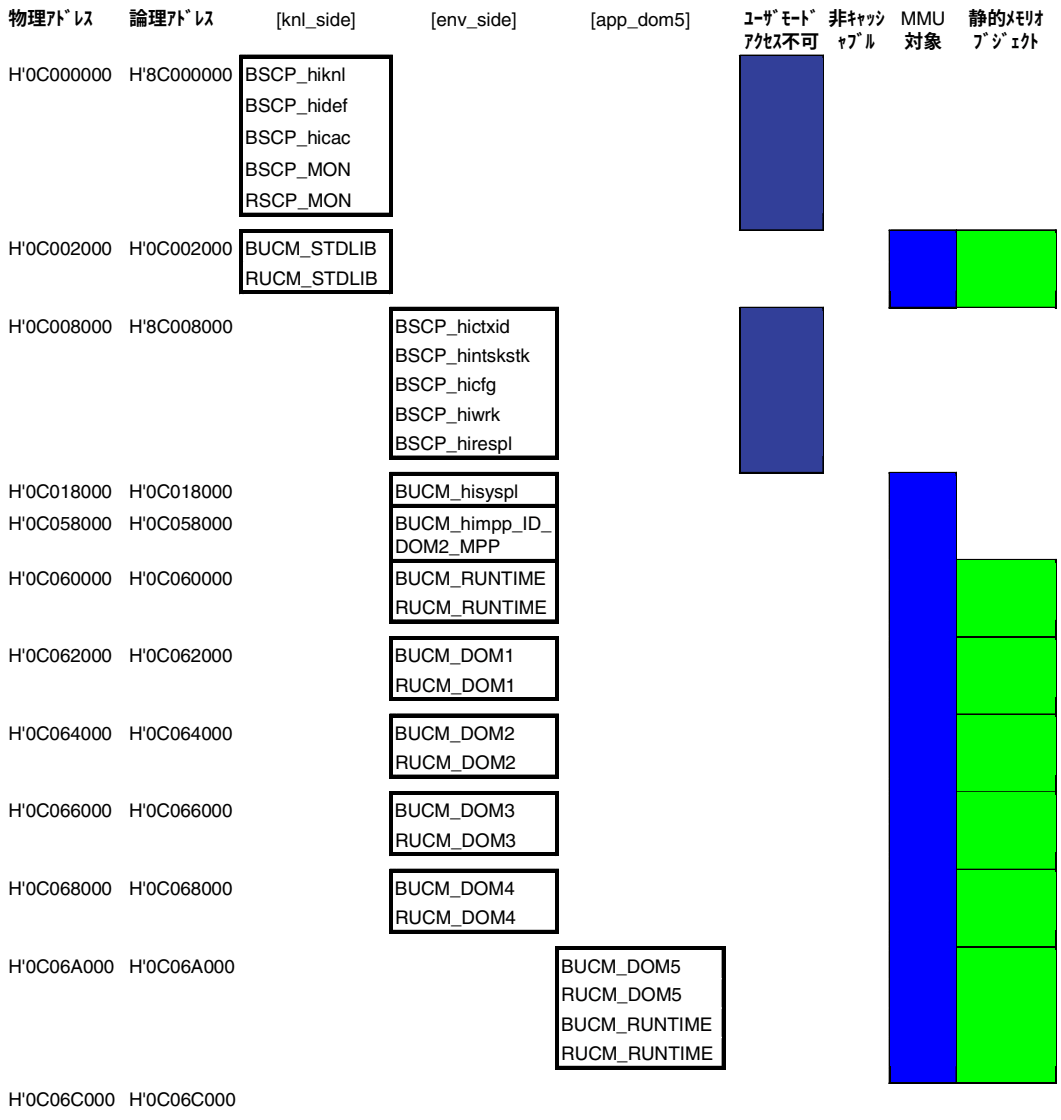


図11.29　B, R **セクションの配置**

以下、リンク単位毎にその詳細について解説します。

(1) knl_side

(a) 論理アドレス H'A0000000 のセクションブロック(PSDP_RESET など)

PSDP_RESET(reset.src)は、CPU リセット直後に実行するプログラムセクションなので、CPU のリセットアドレスである H'A0000000 に配置します。その他のセクションの順序は不問です。

(b) 論理アドレス H'80001000 のセクションブロック(PSCP_hiexp など)

これらは、特権モードからのみリードアクセスされるセクションです。セクションの順序は不問です。

(c) 論理アドレス H'00020000 のセクションブロック(PUC__hiintfc など)

これらは、全ユーザドメインからリードアクセスされる可能性のあるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUC__hiintfc セクション～DUCM_STDLIB セクション
- ページサイズ：64kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_SRO(全ドメインがリード可能)

(d) 論理アドレス H'8C000000 のセクションブロック(BSCP_hiknl など)

これらは、特権モードからのみリード・ライトアクセスされるセクションです。セクションの順序は不問です。

(e) 論理アドレス H'0C002000(BUCM_STDLIB など)

これらは、全ユーザドメインからリード・ライトアクセスされる可能性のあるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_STDLIB セクション～RUCM_STDLIB セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_SRW(全ドメインがリード・ライト可能)

(2) env_side

(a) 論理アドレス H'80030000 のセクションブロック(CSCP_hisysmt など)

これらは、特権モードからのみリードアクセスされるセクションです。

CSCP_hisysmt は、カーネル環境側情報テーブル(__kernel_sysmt)のセクションです。このセクションの先頭アドレスは、システム設計時点であらかじめ決定しておき、そのアドレスを knl_side のリンク時に指定する必要があります。開発途中に各セクションサイズが変わってもアドレスがずれないようにするために、CSCP_hisysmt はセクションブロックの先頭に配置することを推奨します。

その他のセクションの順序は不問です。

(b) 論理アドレス H'00040000 のセクションブロック(PUCM_RUNTIME など)

これらは、env_side に含まれる全ドメインからリードアクセスされる可能性があります。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータ

に登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUCM_RUNTIME セクション～DUCM_RUNTIME セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_SRO(全ユーザドメインがリード可能)

(c) 論理アドレス H'00042000 のセクションブロック(PUCM_DOM1 など)

これらは、ドメイン 1 からのみリードアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUCM_DOM1 セクション～DUCM_DOM1 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRO(ID_DOM1)(ドメイン 1 のみがリード可能)

(d) 論理アドレス H'00044000 のセクションブロック(PUCM_DOM2 など)

これらは、ドメイン 2 からのみリードアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUCM_DOM2 セクション～DUCM_DOM2 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRO(ID_DOM2)(ドメイン 2 のみがリード可能)

(e) 論理アドレス H'00046000 のセクションブロック(PUCM_DOM3 など)

これらは、ドメイン 3 からのみリードアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUCM_DOM3 セクション～DUCM_DOM3 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRO(ID_DOM3)(ドメイン 3 のみがリード可能)

(f) 論理アドレス H'00048000 のセクションブロック(PUCM_DOM4 など)

これらは、ドメイン 4 からのみリードアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：PUCM_DOM4 セクション～DUCM_DOM4 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRO(ID_DOM4)(ドメイン 4 のみがリード可能)

(g) 論理アドレス H'8C008000 のセクションブロック(BSCP_hictxid など)

これらは、特権モードからのみリード・ライトアクセスされるセクションです。セクションの順序

は不問です。

(h) 論理アドレス H'0C0180000 のセクションブロック(BUCM_hisyspl)

システムプールのセクションです。

(i) 論理アドレス H'0C0580000 のセクションブロック(BUCM_himpp_ID_DOM2MPP)

コンフィギュレータで登録した保護メモリプールのセクションです。
本サンプルでは、保護メモリプールのサイズを H'8000 としています。
"noprot_sim", "noprot"コンフィギュレーションでは、このセクションは存在しません。

(j) 論理アドレス H'0C060000 のセクションブロック(BUCM_RUNTIME など)

これらは、env_side に含まれる全ドメインからリード・ライトアクセスされる可能性があります。
メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_RUNTIME セクション～RUCM_RUNTIME セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_SRW(全ドメインがリード・ライト可能)

(k) 論理アドレス H'0C062000 のセクションブロック(BUCM_DOM1 など)

これらは、ドメイン 1 からのみリード・ライトアクセスされるセクションです。
メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_DOM1 セクション～RUCM_DOM1 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRW(ID_DOM1)(ドメイン 1 のみがリード・ライト可能)

(l) 論理アドレス H'0C064000 のセクションブロック(BUCM_DOM2 など)

これらは、ドメイン 2 からのみリード・ライトアクセスされるセクションです。
メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_DOM2 セクション～RUCM_DOM2 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRW(ID_DOM2)(ドメイン 2 のみがリード・ライト可能)

(m) 論理アドレス H'0C066000 のセクションブロック(BUCM_DOM3 など)

これらは、ドメイン 3 からのみリード・ライトアクセスされるセクションです。
メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_DOM3 セクション～RUCM_DOM3 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRW(ID_DOM3)(ドメイン 3 のみがリード・ライト可能)

(n) 論理アドレス H'0C068000 のセクションブロック(BUCM_DOM4 など)

これらは、ドメイン 4 からのみリード・ライトアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションの順序とコンフィギュレータ設定が矛盾しないように注意してください。

- アドレス範囲：BUCM_DOM4 セクション～RUCM_DOM4 セクション
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRW(ID_DOM4)(ドメイン 4 のみがリード・ライト可能)

(3) app_dom5

app_dom5 は、ドメイン 5 からのみで構成されるロードモジュールです。したがって、含まれるセクションは実行時ルーチンも含めて全てドメイン 5 からのみアクセスされます。

(a) 論理アドレス H'0004A000 のセクションブロック(PUCM_DOM5 など)

これらは、ドメイン 5 からのみリードアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションのアドレスおよび割り付け結果の最終アドレスが、コンフィギュレータ設定と矛盾しないように注意してください。

- アドレス範囲：H'0004A000 から H'2000(8192)バイト
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRO(ID_DOM5)(ドメイン 5 のみがリード可能)

(b) 論理アドレス H'0C06A000 のセクションブロック(BUCM_DOM5 など)

これらは、ドメイン 5 からのみリード・ライトアクセスされるセクションです。

メモリオブジェクト保護機能使用時は、以下の静的メモリオブジェクトとしてコンフィギュレータに登録しています。リンカで指定するセクションのアドレスおよび割り付け結果の最終アドレスが、コンフィギュレータ設定と矛盾しないように注意してください。

- アドレス範囲：H'0C06A000 から H'2000(8192)バイト
- ページサイズ：4kB
- キャッシュの設定：コピーバック
- アクセス許可ベクタ：TACT_PRW(ID_DOM5)(ドメイン 5 のみがリード・ライト可能)

11.16 シミュレータでの実行

11.16.1 デバッグセッション

シミュレータで実行するためのデバッグセッションが提供されています。デバッグセッションは、以下のプロジェクトに設定されています。

- kernel.hws の knl_side.hwp
- kernel.hws の envl_side.hwp
- app_dom5.hws の app_dom5.hwp

デバッグセッション名は ***shnnnn*** によって異なりますが、例えば"sim_sh4aldsp-cyc_env_side"といった名称です。

knl_side.hwp と env_side.hwp のセッションでは､knl_side.abs と env_side のみをダウンロードします。ドメイン 5 はダウンロードされません。

一方、app_dom5.hwp のセッションでは、knl_side.abs と env_side.abs に加え、app_dom5.abs もダウンロードします。このため、app_dom5.hwp では、モニタを使ってﾄﾞメイン 5 のメインタスクを起動することができます。

V.1.01 Release 00 では、*sh7780*(SH7780 用)のデバッグセッションのみ提供しています。

これらのデバッグセッションの設定は、３つともほぼ同一です。特記すべき設定項目は、以下の通りです。

(1) マップ設定済み
(2) I/Oシミュレーション：使用する(I/Oシステムコールアドレス＝４)
(3) タイマシミュレーション：使用する
(4) 割込み・例外発生時：実行を継続する(「停止する」にした場合は、TLBミス例外発生時にシミュレーションが停止します)

これらの設定は、ワークスペースディレクトリにある HEW コマンドファイル(拡張子 hdc)によって実施しています。各セッションには、シミュレータ接続時およびダウンロード時にこのコマンドファイルが自動的に実行されるように設定してあります。

なお、このコマンドファイルでは、上記の設定に加えてMMU を Disable にする処理を加えてあります。これは、MMU が Enable の場合、ダウンロードで TLB ミスが発生すると、ダウンロードできなくなるためです。

11.16.2 実行

実行するには､それぞれのデバッグセッションで[デバッグ->ダウンロード->All Download Modules]を選択し、その後リセットしてから実行させます。

11.16.3 モニタの起動

"prot_sim", "noprot_sim"コンフィギュレーションでは、モニタを使用することができます。

モニタを起動するには､[Monitor]と表示されたトリガボタンを押してください(図 11.30)。これにより、[I/O シミュレーション]ウィンドウにモニタのプロンプト"MON>"が表示されます。

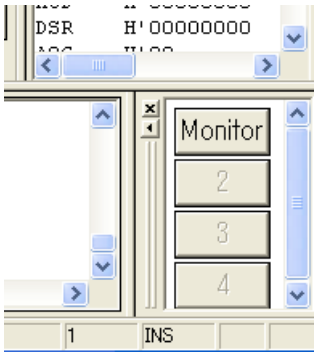


図11.30　モニタを起動するトリガボタン

11.16.4　ドメイン 4 の不正アクセスの検出

ドメイン 4 は、意図的に不正アクセスを行うサンプルです。

メモリオブジェクト保護機能使用時("prot_sim"コンフィギュレーション)は、この不正アクセスが MMU または CPU によってそれが検出され、メモリアクセス違反ハンドラ(sysapp¥mavhdr.c)がドメイン 4 のメインタスクにタスク例外を要求します。その結果、アクセス違反の次の行は実行されず、先にタスク例外処理ルーチンが実行されます。タスク例外処理ルーチンでは、アクセス違反が発生した旨のメッセージを I/O シミュレーションウィンドウに出力し、再度アクセス違反を発生させるためにタスクを再起動します。

"prot_sim"コンフィギュレーションでこれを実行すると、この様子が分かります。

一方、メモリオブジェクト保護機能を使用しない"noprot_sim"コンフィギュレーションでは、ドメイン 4 が不正アクセスを行う点は全く同じですが、メモリオブジェクト保護機能がないために、CPU が検出可能なアクセス違反(ユーザモードからの H'80000000 以降のアドレスへのアクセス)以外は検出されません。

詳細は、ドメイン 4 のソースを参照の上、適時ブレークポイントなどを設定して確認してみてください。

11.16.5　ドメイン 5 の実行

ドメイン 5 のタスクはコンフィギュレータによって生成されますが、起動はされません。

起動するには、モニタの act コマンドを使用します。以下のように入力します。

```
MON> act 5(RET)
```

ドメイン 5 のタスク ID は、コンフィギュレータで ID 名称を"ID_DOM5_MAIN"で自動割当にしています。実際の ID 番号は、コンフィギュレータが samples¥***shnnnn***¥config_out¥$(CONFIGNAME)¥にある kernel_id.h に"ID_DOM5_MAIN"として出力します。出荷時の状態では、このタスク ID は 5 です。

12. スタック使用サイズの算出

12.1 スタックの種類

本カーネルでは、以下の種類のスタックがあります。

(1) タスクのスタック

各タスクは、異なるスタックを持ちます。

ユーザドメインに所属するタスクは、２つのスタックを持ちます。ひとつはタスク自身が使用するスタックであり、これとは別にシステムスタックを持ちます。システムスタックはタスクから呼び出された拡張サービスコールルーチンとトラップルーチンが使用します。また、カーネルがタスクのコンテキストを保存するためにも使用します。

一方カーネルドメインに所属するタスクは、スタックを一つだけ持ちます。このスタックは、タスク自身が使用するのはもちろん、タスクから呼び出された拡張サービスコールルーチンとトラップルーチンが使用します。また、カーネルがタスクのコンテキストを保存するためにも使用します。

(2) 非タスクコンテキストスタック

文字通り、非タスクコンテキストが実行するときに使用するスタックです。カーネルはタスクコンテキストから非タスクコンテキストに遷移するときに、スタックポインタを非タスクコンテキストスタックに切り替えます。

非タスクコンテキストスタックは、システムにひとつだけ存在します。

12.2 スタック使用サイズの算出方法の概要

タスクやハンドラ毎に、その開始関数からのサブルーチン(関数)ネストをたどった使用サイズを求めます。このサイズを「独自使用サイズ」と呼びます。

実際に必要なサイズは、「独自使用サイズ」にカーネルが使用するサイズを加算した値となります。

12.3 各タスクのスタック使用サイズ

各タスクのスタックサイズは、タスク生成時に指定します。以下を参考に、指定する値を算出してください。

12.3.1 ユーザドメインに所属するタスク

ユーザドメインに所属するタスクは、タスクが使用するスタックに加え、システムスタックを持ちます。

(1) タスクが使用するスタック

使用サイズ＝(タスクの独自使用サイズ)＋(タスク例外処理ルーチンの独自使用サイズ)

タスク例外処理ルーチンがネストして起動される場合は、ネストも加味して合計サイズを算出してください。

(2) システムスタック

使用サイズ＝(タスクのコンテキスト保存サイズ)
＋(呼び出される拡張サービスコールルーチン・トラップルーチンの独自使用サイズ)
＋(拡張サービスコールルーチン・トラップルーチン・タスク例外処理ルーチンの
コンテキスト保存サイズ)

コンテキスト保存サイズとは、プログラムの実行のために必要なレジスタを保存するためのサイズです。タスクのコンテキスト保存サイズは必須であり、拡張サービスコールルーチン・トラップルーチン・タスク例外処理ルーチンは、起動される毎にコンテキスト保存サイズが必要になります。
各コンテキスト保存サイズは、表 12.1に示す通りです。

表12.1 コンテキスト保存サイズ

分類	サイズ	必要となる条件
タスク	180	必須
	56	TA_COP0 属性の場合
	72	TA_COP1 属性の場合
	64	TA_COP2 属性の場合
拡張サービスコールルーチン、トラップルーチン、タスク例外処理ルーチン	112	必須
	56	TA_COP0 属性の場合
	72	TA_COP1 属性の場合
	64	TA_COP2 属性の場合

拡張サービスコールルーチン、トラップルーチン、タスク例外処理ルーチンがネストする場合は、ネストを考慮して加算してください。

(3) 算出例

図 12.1に示す例で、算出例を示します。なお、図の[ ]内の数値は、それぞれの独自使用サイズです。また、すべて TA_COP0, TA_COP1, TA_COP2 属性の指定が無い前提とします。

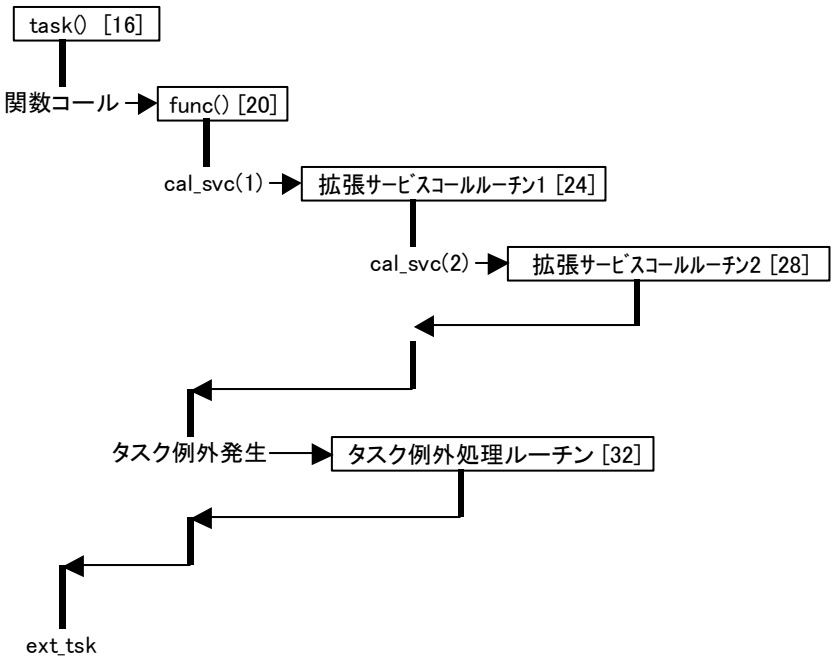


図12.1　タスクのスタック使用サイズの算出例

(a) タスクが使用するスタック

使用サイズ＝(タスクの独自使用サイズ)＋(タスク例外処理ルーチンの独自使用サイズ)

＝(16＋20)＋32

＝68

(b) システムスタック

- 拡張サービスコールルーチン 2 のネストケース

使用サイズ＝(タスクのコンテキスト保存サイズ)

＋(呼び出される拡張サービスコールルーチン・トラップルーチンの独自使用サイズ)

＋(拡張サービスコールルーチン・トラップルーチン・タスク例外処理ルーチンの

コンテキスト保存サイズ)

＝180

＋(24 + 28)

＋(112 + 112)

＝456

- タスク例外処理ルーチンのネストケース
  使用サイズ＝(タスクのコンテキスト保存サイズ)
  ＋(呼び出される拡張サービスコールルーチン・トラップルーチンの独自使用サイズ)
  ＋(拡張サービスコールルーチン・トラップルーチン・タスク例外処理ルーチンの
  コンテキスト保存サイズ)
  ＝180
  ＋0
  +112
  ＝292

よって、システムスタックの使用サイズは 456 となります。

12.3.2 カーネルドメインに所属するタスクのスタック

カーネルドメインに所属するタスクは、スタックを 1 本だけ持ちます。

スタック使用サイズは、「12.3.1(1) タスクが使用するスタック」と「12.3.1(2) システムスタック」のそれぞれの計算結果の和になります。

12.4 非タスクコンテキストスタックサイズの算出

非タスクコンテキストスタックはシステムでひとつのみ存在し、そのサイズは CFG_NTSKSTKSZ で指定します。

CFG_NTSKSTKSZ は、以下のサイズ 1、サイズ 2 の大きいほうのサイズ以上が必要です。

●サイズ 1

＝256

＋(各初期化ルーチンでの最大使用サイズ) .....(1)

＋(NMI 割込みハンドラ使用サイズ)×(NMI ネスト数) .....(2)

●サイズ 2

＝256

＋(タスクコンテキスト実行中に発生する CPU 例外ハンドラの最大使用サイズ) .....(3)

＋Σ(各割込みレベルの割込みハンドラの最大使用サイズ) .....(4)

＋(NMI 割込みハンドラ使用サイズ)×(NMI ネスト数) .....(2)

(1) 各初期化ルーチンについて「12.4.1 各初期化ルーチン・標準タイマドライバのタイマ初期化ルーチンの使用サイズ」に従って使用サイズを算出し、全初期化ルーチンの中で最大のサイズで計算してください。

(2) 「NMI割込みハンドラ使用サイズ」は、「12.4.3 NMI割込みハンドラの使用サイズ」を参照してください「NMIネスト数」は、割込みコントローラでSR.BL=1の時にNMIを受理する設定にしている場合は、考えられるNMIのネスト数としてください。そうでない場合は1としてください。

(3) 全タスクコンテキスト実行中に発生しえるCPU例外について「12.4.4 各CPU例外ハンドラの使用サイズ」に従って使用サイズを求め、それらのハンドラの中で最大のサイズで計算してください。

(4) 「12.4.2 各割込みハンドラ・タイムイベントハンドラ・標準タイマドライバのタイマ割込みルーチンの使用サイズ」に従って各ハンドラの使用サイズを求め、各割込みレベル毎にハンドラの中で最大のサイズを、全割込みレベルについて加算してください。

■関連ページ　「8.4.4 NMI に関する注意事項」

12.4.1 各初期化ルーチン・標準タイマドライバのタイマ初期化ルーチンの使用サイズ

使用サイズ＝(初期化ルーチンの独自使用サイズ)

＋216.....(サービスコールを呼び出す場合のみ必要)

＋(呼び出される拡張サービスコールルーチン・トラップルーチンの独自使用サイズ)

＋(拡張サービスコールルーチン・トラップルーチンのコンテキスト保存サイズ)

＋(CPU 例外ハンドラの使用サイズ)

＋(CPU 例外の発生毎に140)

コンテキスト保存サイズは、前述の表 12.1に示す通りです。

拡張サービスコールルーチン、トラップルーチンする場合は、ネストを考慮して加算してください。

「CPU 例外ハンドラの使用サイズ」は、算出対象の初期化ルーチンを実行中に CPU 例外が発生する場合に必要です。算出方法は、「12.4.4 各 CPU 例外ハンドラの使用サイズ」を参照してください。

なお、標準タイマドライバの初期化ルーチン(_kernel_tmrini())も本計算式で算出してください。

12.4.2 各割込みハンドラ・タイムイベントハンドラ・標準タイマドライバのタイマ割込みルーチンの使用サイズ

使用サイズ＝(割込みハンドラの独自使用サイズ)

＋76

＋216.....(サービスコールを呼び出す場合のみ必要)

＋(呼び出される拡張サービスコールルーチン・トラップルーチンの独自使用サイズ)

＋(拡張サービスコールルーチン・トラップルーチンのコンテキスト保存サイズ)

＋(CPU 例外ハンドラの使用サイズ)

＋(CPU 例外の発生毎に140)

コンテキスト保存サイズは、前述の表 12.1に示す通りです。

「CPU 例外ハンドラの使用サイズ」は、算出対象のハンドラやルーチンを実行中に CPU 例外が発生する場合に必要です。算出方法は、「12.4.4 各 CPU 例外ハンドラの使用サイズ」を参照してください。

拡張サービスコールルーチン、トラップルーチンがネストする場合は、ネストを考慮して加算してください。

なお、タイムイベントハンドラと標準タイマドライバのタイマ割込みルーチン(_kernel_tmrint())は、割込みレベルが CFG_KNLLVL の割込みハンドラとして、本式で算出してください。ただし、160を加算して計算してください。

12.4.3 NMI 割込みハンドラの使用サイズ

使用サイズ＝(NMI 割込みハンドラの独自使用サイズ)＋
＋(CPU 例外ハンドラの使用サイズ)
＋(CPU 例外の発生毎に140)

「CPU 例外ハンドラの使用サイズ」は、NMI 割込みハンドラを実行中に CPU 例外が発生する場合に必要です。算出方法は、「12.4.4 各 CPU 例外ハンドラの使用サイズ」を参照してください。CPU 例外ハンドラがネストする場合は、ネストを考慮して加算してください。

12.4.4 各 CPU 例外ハンドラの使用サイズ

使用サイズ＝(CPU 例外ハンドラの独自使用サイズ)
＋216.....(サービスコールを呼び出す場合のみ必要)
＋(さらに発生する CPU 例外ハンドラの使用サイズ)
＋(さらに発生する CPU 例外の発生毎に140)

「さらに発生する CPU 例外」とは、算出対象の CPU 例外ハンドラの実行中に CPU 例外が発生する場合です。CPU 例外ハンドラがネストする場合は、ネストを考慮して加算してください。

13. リソースプールサイズの見積り

13.1 概要

リソースプールのサイズは、CFG_RESPOOLSZ で指定します。

リソースプールは、主にカーネル内部の管理テーブルを動的に割り付けるために使用されます。

メモリが必要になった時に取得し、不要になれば解放することでメモリを再利用可能としています。これは、静的にメモリを確保する場合に比べて、消費するメモリ量を抑制できる効果があります。

一方で、リソースプールの消費状況はシステムの状況に依存するために、必要な最小サイズを正確に求めることは一般には困難です。

システム動作中にリソースプールが不足すると、各種サービスコールがエラーE_NOMEM になるなど、システムにとって致命的な状況になります。以下の手順で必要なサイズの目安を見積もり、十分なサイズを CFG_RESPOOLSZ に指定してください。

(1) リソースプールが消費されるタイミングと、その要求サイズを調べる。→「13.2 いつどれだけ要求されるか」

(2) 以下で説明しているアルゴリズムを考慮して、必要なサイズを見積もる。具体的には、「13.3 計算」を参考にして見積もります。

■関連ページ　「4.31 メモリの断片化とその対策」

なお、本章では以下の記号を使います。

ROUND_UP(a, b)：a を b の倍数に切上げた値

13.2 いつどれだけ要求されるか

13.2.1 カーネル起動時(vsta_knl)

(1) 静的メモリオブジェクトの管理

全ての静的メモリオブジェクトについて、VTSZ_MEMMB(ページサイズ, 静的メモリオブジェクトサイズ)で算出されるサイズの総和を消費します。このマクロの内容は、以下の通りです。

```
VTSZ_MEMMB(ページサイズ, 静的メモリオブジェクトサイズ)
  =(ROUND_UP(静的メモリオブジェクトサイズ, ページサイズ)÷CFG_PAGESZ(4096)×12)+4
```

この領域は、決して解放されません。

13.2.2 各種オブジェクトの生成時

以下の各種オブジェクトを生成するときには、リソースプールを要求します。このとき消費された領域は、そのオブジェクトを削除するときに解放されます。

(1) タスク

(a) リソースプールから割り付けるシステムスタック

タスクの生成時にシステムスタック領域をカーネルが割り付ける指定を行った場合、リソースプールを(システムスタックサイズ+4)バイト要求します。なお、コンフィギュレータでタスクを生成する場合は、カーネルがスタック領域を割り付ける指定しかできません。

(b) システムプールから割り付けるスタック領域の管理

ユーザドメインに所属するタスクの生成時にスタック領域をカーネルが割り付ける指定を行った場合、スタック領域はシステムプールから割り付けられますが、この管理のためにリソースプールを最大で VTSZ_SPLALCMB だけ要求します。なお、コンフィギュレータでタスクを生成する場合は、カーネルがスタック領域を割り付ける指定しかできません。

このマクロの内容は、以下の通りです。

- CFG_PROTMEM 選択有りの場合
  `VTSZ_SPLALCMB＝60`
- CFG_PROTMEM 選択なしの場合
  `VTSZ_SPLALCMB＝36`

(2) データキュー

データキュー生成時、データ数が 0 以外の場合は、リソースプールを TSZ_DTQMB(データ数)だけ要求します。

このマクロの内容は、以下の通りです。

```
TSZ_DTQMB(データ数)＝((データ数)×4) + 4
```

消費された領域は、データキュー削除時に解放されます。

なお、本マクロと同じ定義で TSZ_DTQ(データ数)というマクロもあります。TSZ_DTQMB はμITRON4.0 仕様の保護機能拡張仕様で規定されているマクロで、TSZ_DTQ はオリジナルのμITRON4.0 仕様で規定されているマクロです。

(3) メールボックス

「TA_MPRI 属性かつ最大メッセージ優先度>1」の条件を満たす場合、リソースプールを TSZ_MBXMB(メッセージ数, 最大メッセージ優先度)だけ要求します。

このマクロの内容は、以下の通りです。

```
TSZ_MBXMB(メッセージ数, 最大メッセージ優先度)＝((最大メッセージ優先度)×8) + 4
```

消費された領域は、メールボックス削除時に解放されます。

(4) メッセージバッファ

メッセージバッファ生成時、バッファサイズが 0 以外の場合は、リソースプールをそのバッファサイズ+4 だけ要求します。

消費された領域は、メッセージバッファ削除時に解放されます。

(5) 固定長メモリプール

(a) 固定長メモリブロックの管理

固定長メモリブロックを管理するためにリソースプールを TSZ_MPFMB(ブロック数, ブロックサイズ)だけ要求します。このマクロの内容は、以下の通りです。

```
TSZ_MPFMB(ブロック数, ブロックサイズ)＝((ブロック数)×4) + 4
```

(b) 固定長メモリプールをシステムプールから割り付ける場合

固定長メモリプールの生成時にプール領域をカーネルが割り付ける指定を行った場合、プール領域はシステムプールから割り付けられますが、この管理のためにリソースプールを最大で VTSZ_SPLALCMB だけ消費します。なお、コンフィギュレータで固定長メモリプールを生成する場合は、カーネルがプール領域を割り付ける指定しかできません。このマクロの内容は、以下の通りです。

- CFG_PROTMEM 選択有りの場合
  `VTSZ_SPLALCMB＝60`
- CFG_PROTMEM 選択なしの場合
  `VTSZ_SPLALCMB＝36`

(6) 可変長メモリプール

(a) 可変長メモリプールをシステムプールから割り付ける場合

可変長メモリプールの生成時にプール領域をカーネルが割り付ける指定を行った場合、プール領域はシステムプールから割り付けられますが、この管理のためにリソースプールを最大で VTSZ_SPLALCMB だけ要求します。なお、コンフィギュレータで可変長メモリプールを生成する場合は、カーネルがプール領域を割り付ける指定しかできません。このマクロの内容は、以下の通りです。

- CFG_PROTMEM 選択有りの場合
  `VTSZ_SPLALCMB＝60`
- CFG_PROTMEM 選択なしの場合
  `VTSZ_SPLALCMB＝36`

(b) セクタ管理

VTA_UNFRAGMENT 属性が指定された場合は、VTSZ_SCTMB(最大セクタ数)だけ要求します。このマクロの内容は、以下の通りです。

```
VTSZ_SCTMB(最大セクタ数)＝(20×(最大セクタ数)＋72) + 4
```

(7) 保護メールボックス

保護メールボックス生成時に、「TA_MPRI 属性かつ最大メッセージ優先度>1」の条件を満たす場合、リソースプールを TSZ_MBPMB(メッセージ数, 最大メッセージ優先度)だけ要求します。

このマクロの内容は、以下の通りです。

```
TSZ_MBPMB(メッセージ数, 最大メッセージ優先度)＝((最大メッセージ優先度)×8) + 4
```

消費された領域は、メールボックス削除時に解放されます。

13.2.3 その他のタイミングで消費・解放されるサイズ

(1) メールボックス：snd_mbx, isnd_mbx

受信待ちタスクが存在せず、メッセージがメールボックスにキューイングされる場合には、VTSZ_MSGMB だけ要求します。このマクロの内容は、以下の通りです。

```
VTSZ_MSGMB＝20
```

この領域は、そのメッセージが受信されるとき、および対象のメールボックスが削除されるときに解放されます。

(2) 可変長メモリプール：get_mpl, pget_mpl, ipget_mpl, tget_mpl

メモリブロックを獲得する際、最大で VTSZ_BLKMB だけ要求します。このマクロの内容は、以下の通りです。

```
VTSZ_BLKMB＝36
```

この領域は、そのブロックが解放されるとき、および対象の可変長メモリプールが削除されるときに解放されます。

(3) 保護メモリプール：pget_mpp

保護メモリブロックを獲得する際、最大で VTSZ_MPPBLKMB だけ要求します。このマクロの内容は、以下の通りです。

```
VTSZ_MPPBLKMB＝60
```

この領域は、そのブロックが解放されるとき、に解放されます。

(4) 保護メールボックス：snd_mbp

受信待ちタスクが存在せず、メッセージが保護メールボックスにキューイングされる場合には、VTSZ_MSGMB だけ消費します。このマクロの内容は、以下の通りです。

```
VTSZ_MSGMB＝20
```

この領域は、そのメッセージが受信されるとき、および対象の保護メールボックスが削除されるときに解放されます。

13.3 計算

リソースプールは、「最小ブロックサイズ」が20バイトとしてセクタ方式で管理されます。

(1) 160バイト以下の要求(セクタとして確保されるもの)

以下の式で求めます。

SZ_RESSCT＝Σ((ROUND_UP(Num[n]/Cnt[n], 1) × 696)

・n：1,2,4,8
・Num[n]：(20×n)バイト以下の同時要求数（下表参照）
・Cnt[n]：1 セクタ内のブロック数（下表参照）

n	Num[n]	Cnt[n]
1	20 バイト以下の同時要求数	32
2	21～40 バイトの同時要求数	16
4	41～80 バイトの同時要求数	8
8	81～160 バイトの同時要求数	4

(2) 160バイトを超える要求

同時要求されるものについて、以下の式で求めます。

SZ_RESLARGE＝Σ(要求サイズ＋32)

(3) 全必要サイズ

リソースプールの管理のために、VTSZ_RPLMB バイトを使用します。このマクロの内容は、以下の通りです。

```
VTSZ_RPLMB = 32
```

この領域は、決して解放されません。
全必要サイズは、次式で算出されます。

全必要サイズ＝VTSZ_RPLMB + SZ_RESSCT ＋ SZ_RESLARGE

14. システムプールサイズの見積り

14.1 概要

システムプールのサイズは、CFG_SYSPOOLSZ で指定します。

システムプールサイズは、以下の手順で見積もります。

(1) リソースプールが消費されるタイミングと、その要求サイズを調べる。→「14.2 いつどれだけ要求されるか」

(2) 以下で説明しているアルゴリズムを考慮して、必要なサイズを見積もる。

■関連ページ 「4.31 メモリの断片化とその対策」

14.2 いつどれだけ要求されるか

システムプールは、以下の場合に使用されます。逆に言うと、以下の領域をアプリケーション側で確保すれば、システムプールサイズを 0 にすることもできます。

(1) タスク生成時

ユーザドメインに所属するタスクの生成時にスタック領域をカーネルが割り付ける指定を行った場合、指定されたスタックサイズのスタック領域がシステムプールから割り付けられます。なお、コンフィギュレータでタスクを生成する場合は、カーネルがスタック領域を割り付ける指定しかできません。

この領域は、そのタスクを削除するときに解放されます。

(2) 固定長メモリプール生成時

固定長メモリプールの生成時にプール領域をカーネルが割り付ける指定を行った場合、TSZ_MPF(ブロック数, ブロックサイズ)バイトの固定長メモリプール領域がシステムプールから割り付けられます。

この領域は、その固定長メモリプールを削除するときに解放されます。

なお、TSZ_MPF マクロの内容は、以下の通りです。

```
TSZ_MPF(ブロック数, ブロックサイズ)＝(ブロックサイズ)×(ブロック数)
```

(3) 可変長メモリプール領域

可変長メモリプールの生成時にプール領域をカーネルが割り付ける指定を行った場合、指定したメモリプールサイズのメモリプール領域がシステムプールから割り付けられます。

この領域は、その可変長メモリプールを削除するときに解放されます。

15. FPU に関する注意事項

15.1 「FPU を使用する」の意味

カーネルから見て、「FPU を使用する」とはアプリケーションで FPU のレジスタにアクセスすることを意味します。

FPU のレジスタは、コンパイラオプションで"cpu=sh4a"を指定し、かつ以下のいずれかを満たす場合にアクセスされます。

(1) コンパイラのfpuオプションを指定していない。HEWでは、[オプション->SuperH RISC engine Standard Toolchain...]を選択して開く[SuperH RISC engine Standard Toolchain]ダイアログボックスの[CPU]タブにある[浮動小数点演算モード]が[Mix]になっている。

(2) 浮動小数点型のデータタイプを使用している。

特に(1)のケースでは、浮動小数点データを扱わなくても FPU を使用する扱いになることに注意してください。言い換えると、(1)の指定は強く推奨しません。

15.2 各アプリケーションでの FPU の使用

15.2.1 タスク、タスク例外処理ルーチン、拡張サービスコールルーチン、トラップルーチン

(1) TA_COP1, TA_COP2 属性

これらでは、属性に TA_COP1 または TA_COP1|TA_COP2 を指定した場合のみ、FPU を使用できます。TA_COP1, TA_COP2 属性は、表 15.1に示すように指定してください。

表15.1 TA_COP1, TA_COP2 属性の指定

条件	属性
マトリックス演算などを行う場合(両方の FPU レジスタバンクを使用する場合)	TA_COP1|TA_COP2
通常の浮動小数点演算を行う場合(FPU レジスタバンク 0 のみを使用)	TA_COP1
浮動小数点演算は行わない場合	(不要)

(2) 起動時の FPSCR レジスタ

起動時の FPSCR レジスタの値は、それぞれの生成・定義時に指定します。この指定値と、コンパイラオプションの設定が矛盾しないように注意してください。この関係を表 15.2に示します。

表15.2　生成・定義時に指定する FPSCR 初期値とコンパイラオプションの関係

コンパイラオプション			生成・定義時に指定すべき FPSCR 初期値(inifpscr)
fpu	denormalize	round	
指定無しまたは single *	on *	zero *	H'00040001　(FR=0, PR=0. DN=1, RM=1)
		nearest	H'00040000　(FR=0, PR=0. DN=1, RM=0)
	off	zero	H'00000001　(FR=0, PR=0. DN=0, RM=1)
		nearest	H'00000000　(FR=0, PR=0. DN=0, RM=0)
double	on	zero	H'000C0001　(FR=0, PR=1. DN=1, RM=1)
		nearest	H'000C0000　(FR=0, PR=1. DN=1, RM=0)
	off	zero	H'00080001　(FR=0, PR=1. DN=0, RM=1)
		nearest	H'00080000　(FR=0, PR=1. DN=0, RM=0)

【注】　コンパイラのデフォルト設定

15.2.2　その他のアプリケーション

その他のアプリケーションでは、FPU は使用できません。
FPU を使用するには、アプリケーション側で FPU のレジスタを保証してください。

16. システムダウン等の対処

16.1 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 16.1に示す情報が渡されます。

表16.1 システムダウンルーチンに渡される情報

項番	システムダウン要因	システムダウンルーチンに渡されるパラメータ				対策
		種別 ER type (R4)	情報 1 VW inf1 (R5)	情報 2 VW inf2 (R6)	情報 3 VW inf3 (R7)	
1	vsys_dwn, ivsys_dwn	1～ H'7fffffff	vsys_dwn, ivsys_dwn のパラメータ			
2	vsta_knl システムプールを生成できない	0	1	不定	不定	次節参照
3	vsta_knl		2	E_PAR	不定	次節参照
4	静的メモリオブジェクトを生成できない			E_NOMEM	不定	
5	ext_tsk 非タスクコンテキストからの呼び出し	-1	E_CTX	ext_tsk を呼び出したアドレス	不定	
6	exd_tsk 非タスクコンテキストからの呼び出し	-2	E_CTX	exd_tsk を呼び出したアドレス	不定	
7	未定義 CPU 例外が発生	-H'10	EXPEVT コード	VT_EXC *pk_exc	不定	
8	未定義割込みが発生	-H'11	INTEVT コード	不定	不定	
9	vsta_knl CFG_ACTION に関する初期化に失敗	-H'20	不定	不定	不定	
10	メモリアクセス違反が発生 *(MMU によって検出)	-H'80	VT_MAV *pk_mav	VT_EXC *pk_exc	不定	pk_mav, pk_exc を元に、メモリアクセス違反の要因を取り除いてください。

【注】 サンプルのメモリアクセス違反ハンドラ(samples¥sysapp¥mavhdr.c)によって実現されています。

16.2 カーネル起動時(vsta_knl)のエラー

16.2.1 システムダウンになるケース

(1) システムプールを生成できない

システムプールのセクションは、メモリオブジェクト保護機能を組み込んだ場合は先頭アドレスがCFG_PAGESZ(4kB)境界の MMU 対象領域、そうでない場合は先頭アドレスが 4 の倍数でユーザモードからアクセス可能な領域に配置しなければなりません。これに反する場合は、システムダウンとなります。

(2) 静的メモリオブジェクトを生成できない

静的メモリオブジェクトに関して、以下のエラーが検出された場合はシステムダウンとなります。

- 静的メモリオブジェクトの先頭アドレスが、静的メモリオブジェクトのページサイズの境界アドレスで無い。(表 16.1の項番 3)
- リソースプールが不足している。(表 16.1の項番 4)

16.2.2 コンフィギュレータで指定したオブジェクトを生成できない場合

コンフィギュレータで生成・定義した各種オブジェクトは、コンフィギュレータが出力する「初期登録ルーチン」からサービスコールを呼び出すことによって生成・定義されます。

初期登録ルーチンには、以下の 2 種類があります。

(1) kernel_def_inireg.h
[カーネル側]をチェックして生成・定義されたオブジェクト

(2) kernel_cfg_inireg.h
[カーネル側]をチェックせずに生成・定義されたオブジェクト

これらのファイルでは、以下のようなステートメントでオブジェクトの生成を行います。

```
if(_CRE_MPL(ID_DOM2_MPL) != E_OK)
   while(1);
```

_CRE_MPL( )は、可変長メモリプールを生成するサービスコールを呼び出すマクロです。
表 16.2に、使用するサービスコールとマクロを示します。

表16.2 初期登録ルーチンが使用するサービスコール

項番	オブジェクト種類	使用するサービスコール	マクロ
1	割込みハンドラ	idef_inh	_DEF_INH
2	CPU 例外ハンドラ	idef_inh	_DEF_INH
3	オーバーランハンドラ	def_ovr	_DEF_OVR
4	周期ハンドラ	icre_cyc	_CRE_CYC
5	アラームハンドラ	icre_alm	_CRE_ALM
6	拡張サービスコールルーチン	idef_svc	_DEF_SVC
7	トラップルーチン	ivdef_trp	_DEF_TRP
8	セマフォ	icre_sem	_CRE_SEM
9	イベントフラグ	icre_flg	_CRE_FLG
10	データキュー	icre_dtq	_CRE_DTQ
11	メールボックス	icre_mbx	_CRE_MBX
12	ミューテックス	cre_mtx	_CRE_MTX
13	メッセージバッファ	icre_mbf	_CRE_MBF
14	固定長メモリプール	メモリオブジェクト保護機能使用時：icra_mpf メモリオブジェクト保護機能未使用時：icre_mpf	_CRE_MPF
15	可変長メモリプール	メモリオブジェクト保護機能使用時：ivcra_mpl メモリオブジェクト保護機能未使用時：icre_mpl	_CRE_MPL
16	保護メモリプール	icre_mpp	_CRE_MPP
17	保護メールボックス	icre_mbp	_CRE_MBP
18	タスク	icre_tsk	_CRE_TSK
19	タスク例外処理ルーチン	idef_tex	_DEF_TEX

これらのサービスコールがエラーとなった場合は、そのサービスコールの呼び出し部の直後で無限ループするようになっています。無限ループしている箇所から、対応するコンフィギュレータの生成オブジェクトが分かりますので、そのオブジェクトに関するコンフィギュレータの設定を確認してく

ださい。

■関連ページ　「6.22.12 カーネルの起動(vsta_knl, ivsta_knl)」

17. 各種一覧

17.1 サービスコール一覧

(1) タスク管理機能

1	ER cre_tsk(ID, T_CTSK *);	タスクの生成
	ER icre_tsk(ID, T_CTSK *);	同上(非タスクコンテキスト用)
2	ER_ID acre_tsk(T_CTSK *);	タスクの生成(ID 番号自動割当て)
	ER_ID iacre_tsk(T_CTSK *);	同上(非タスクコンテキスト用)
3	ER del_tsk(ID);	タスクの削除
4	ER act_tsk(ID);	タスクの起動
	ER iact_tsk(ID);	同上(非タスクコンテキスト用)
5	ER_UINT can_act(ID);	タスク起動要求のキャンセル
	ER_UINT ican_act(ID);	同上(非タスクコンテキスト用)
6	ER sta_tsk(ID, VP_INT);	タスクの起動(起動コード指定)
	ER ista_tsk(ID, VP_INT);	同上(非タスクコンテキスト用)
7	void ext_tsk(void);	自タスクの終了
8	void exd_tsk(void);	自タスクの終了と削除
9	ER ter_tsk(ID);	タスクの強制終了
10	ER chg_pri(ID, PRI);	タスク優先度の変更
	ER ichg_pri(ID, PRI);	同上(非タスクコンテキスト用)
11	ER get_pri(ID, PRI *);	タスク優先度の参照
	ER iget_pri(ID, PRI *);	同上(非タスクコンテキスト用)
12	ER ref_tsk(ID, T_RTSK *);	タスクの状態参照
	ER iref_tsk(ID, T_RTSK *);	同上(非タスクコンテキスト用)
13	ER ref_tst(ID, T_RTST *);	タスクの状態参照(簡易版)
	ER iref_tst(ID, T_RTST *);	同上(非タスクコンテキスト用)
14	ER vchg_tmd(MODE);	タスク実行モードの変更

(2) タスク付属同期機能

15	ER slp_tsk(void);	起床待ち
16	ER tslp_tsk(TMO);	同上(タイムアウト指定)
17	ER wup_tsk(ID);	タスクの起床
	ER iwup_tsk(ID);	同上(非タスクコンテキスト用)
18	ER_UINT can_wup(ID);	タスク起床要求のキャンセル
	ER_UINT ican_wup(ID);	同上(非タスクコンテキスト用)
19	ER rel_wai(ID);	待ち状態の強制解除
	ER irel_wai(ID);	同上(非タスクコンテキスト用)
20	ER sus_tsk(ID);	強制待ち状態への移行
	ER isus_tsk(ID);	同上(非タスクコンテキスト用)
21	ER rsm_tsk(ID);	強制待ち状態からの再開
	ER irsm_tsk(ID);	同上(非タスクコンテキスト用)
22	ER frsm_tsk(ID);	強制待ち状態からの強制再開

	ER	ifrsm_tsk(ID);	同上(非タスクコンテキスト用)
23	ER	dly_tsk(RELTIM);	自タスクの遅延
24	ER	vset_tfl(ID, FLGPTN);	タスク付属イベントフラグのセット
	ER	ivset_tfl(ID, FLGPTN);	同上(非タスクコンテキスト用)
25	ER	vclr_tfl(ID, FLGPTN);	タスク付属イベントフラグのクリア
	ER	ivclr_tfl(ID, FLGPTN);	同上(非タスクコンテキスト用)
26	ER	vwai_tfl(FLGPTN, FLGPTN *);	タスク付属イベントフラグ待ち
27	ER	vpol_tfl(FLGPTN, FLGPTN *);	同上(ポーリング)
28	ER	vtwai_tfl(FLGPTN, FLGPTN *, TMO);	同上(タイムアウト指定)

(3) タスク例外処理機能

29	ER	def_tex(ID, T_DTEX *);	タスク例外処理ルーチンの定義
	ER	idef_tex(ID, T_DTEX *);	同上(非タスクコンテキスト用)
30	ER	ras_tex(ID, TEXPTN);	タスク例外処理の要求
	ER	iras_tex(ID, TEXPTN);	同上(非タスクコンテキスト用)
31	ER	dis_tex(void);	タスク例外処理の禁止
32	ER	ena_tex(void);	タスク例外処理の許可
33	BOOL	sns_tex(void);	タスク例外処理禁止状態の参照
34	ER	ref_tex(ID, T_RTEX *);	タスク例外処理の状態参照
	ER	iref_tex(ID, T_RTEX *);	同上(非タスクコンテキスト用)

(4) 同期・通信機能(セマフォ)

35	ER	cre_sem(ID, T_CSEM *);	セマフォの生成
	ER	icre_sem(ID, T_CSEM *);	同上(非タスクコンテキスト用)
36	ER_ID	acre_sem(T_CSEM *);	セマフォの生成(ID 番号自動割当て)
	ER_ID	iacre_sem(T_CSEM *);	同上(非タスクコンテキスト用)
37	ER	del_sem(ID);	セマフォの削除
38	ER	sig_sem(ID);	セマフォ資源の返却
	ER	isig_sem(ID);	同上(非タスクコンテキスト用)
39	ER	wai_sem(ID);	セマフォ資源の獲得
40	ER	pol_sem(ID);	同上(ポーリング)
	ER	ipol_sem(ID);	同上(ポーリング、非タスクコンテキスト用)
41	ER	twai_sem(ID, TMO);	同上(タイムアウト指定)
42	ER	ref_sem(ID, T_RSEM *);	セマフォの状態参照
	ER	iref_sem(ID, T_RSEM *);	同上(非タスクコンテキスト用)

(5) 同期・通信機能(イベントフラグ)

43	ER	cre_flg(ID, T_CFLG *);	イベントフラグの生成
	ER	icre_flg(ID, T_CFLG *);	同上(非タスクコンテキスト用)
44	ER_ID	acre_flg(T_CFLG *);	イベントフラグの生成(ID 番号自動割当て)
	ER_ID	iacre_flg(T_CFLG *);	同上(非タスクコンテキスト用)
45	ER	del_flg(ID);	イベントフラグの削除
46	ER	set_flg(ID, FLGPTN);	イベントフラグのセット
	ER	iset_flg(ID, FLGPTN);	同上(非タスクコンテキスト用)
47	ER	clr_flg(ID, FLGPTN);	イベントフラグのクリア
	ER	iclr_flg(ID, FLGPTN);	同上(非タスクコンテキスト用)
48	ER	wai_flg(ID, FLGPTN, MODE, FLGPTN *);	イベントフラグ待ち
49	ER	pol_flg(ID, FLGPTN, MODE, FLGPTN *);	同上(ポーリング)

	ER	ipol_flg(ID, FLGPTN, MODE, FLGPTN *);	同上(ポーリング、非タスクコンテキスト用)
50	ER	twai_flg(ID, FLGPTN, MODE, FLGPTN *, TMO);	同上(タイムアウト指定)
51	ER	ref_flg(ID, T_RFLG *);	イベントフラグの状態参照
	ER	iref_flg(ID, T_RFLG *);	同上(非タスクコンテキスト用)

(6) 同期・通信機能(データキュー)

52	ER	cre_dtq(ID, T_CDTQ *);	データキューの生成
	ER	icre_dtq(ID, T_CDTQ *);	同上(非タスクコンテキスト用)
53	ER_ID	acre_dtq(T_CDTQ *);	データキューの生成(ID 番号自動割当て)
	ER_ID	iacre_dtq(T_CDTQ *);	同上(非タスクコンテキスト用)
54	ER	del_dtq(ID);	データキューの削除
55	ER	snd_dtq(ID, VP_INT);	データキューへの送信
56	ER	psnd_dtq(ID, VP_INT);	同上(ポーリング)
	ER	ipsnd_dtq(ID, VP_INT);	同上(ポーリング、非タスクコンテキスト用)
57	ER	tsnd_dtq(ID, VP_INT, TMO);	同上(タイムアウト指定)
58	ER	fsnd_dtq(ID, VP_INT);	データキューへの強制送信
	ER	ifsnd_dtq(ID, VP_INT);	同上(非タスクコンテキスト用)
59	ER	rcv_dtq(ID, VP_INT *);	データキューからの受信
60	ER	prcv_dtq(ID, VP_INT *);	同上(ポーリング)
61	ER	trcv_dtq(ID, VP_INT *, TMO);	同上(タイムアウト指定)
62	ER	ref_dtq(ID, T_RDTQ *);	データキューの状態参照
	ER	iref_dtq(ID, T_RDTQ *);	同上(非タスクコンテキスト用)

(7) 同期・通信機能(メールボックス)

63	ER	cre_mbx(ID , T_CMBX *);	メールボックスの生成
	ER	icre_mbx(ID , T_CMBX *);	同上(非タスクコンテキスト用)
64	ER_ID	acre_mbx(T_CMBX *);	メールボックスの生成(ID 番号自動割当て)
	ER_ID	iacre_mbx(T_CMBX *);	同上(非タスクコンテキスト用)
65	ER	del_mbx(ID);	メールボックスの削除
66	ER	snd_mbx(ID, T_MSG *);	メールボックスへの送信
	ER	isnd_mbx(ID, T_MSG *);	同上(非タスクコンテキスト用)
67	ER	rcv_mbx(ID, T_MSG **);	メールボックスからの受信
68	ER	prcv_mbx(ID, T_MSG **);	同上(ポーリング)
	ER	iprcv_mbx(ID, T_MSG **);	同上(ポーリング、非タスクコンテキスト用)
69	ER	trcv_mbx(ID, T_MSG **, TMO);	同上(タイムアウト指定)
70	ER	ref_mbx(ID, T_RMBX *);	メールボックスの状態参照
	ER	iref_mbx(ID, T_RMBX *);	同上(非タスクコンテキスト用)

(8) 拡張同期・通信機能(ミューテックス)

71	ER	cre_mtx(ID, T_CMTX *);	ミューテックスの生成
72	ER_ID	acre_mtx(T_CMTX *);	同上(ID 番号自動割当て)
73	ER	del_mtx(ID);	ミューテックスの削除
74	ER	loc_mtx(ID);	ミューテックスのロック
75	ER	ploc_mtx(ID);	同上(ポーリング)
76	ER	tloc_mtx(ID, TMO);	同上(タイムアウト指定)
77	ER	unl_mtx(ID);	ミューテックスのロック解除
78	ER	ref_mtx(ID, T_RMTX *);	ミューテックスの状態参照

(9) 拡張同期・通信機能(メッセージバッファ)

79	ER cre_mbf(ID, T_CMBF *);	メッセージバッファの生成
	ER icre_mbf(ID, T_CMBF *);	同上(非タスクコンテキスト用)
80	ER_ID acre_mbf(T_CMBF *);	メッセージバッファの生成(ID 番号自動割当て)
	ER_ID iacre_mbf(T_CMBF *);	同上(非タスクコンテキスト用)
81	ER del_mbf(ID);	メッセージバッファの削除
82	ER snd_mbf(ID, VP, UINT);	メッセージバッファへの送信
83	ER psnd_mbf(ID, VP, UINT);	同上(ポーリング)
	ER ipsnd_mbf(ID, VP, UINT);	同上(ポーリング、非タスクコンテキスト用)
84	ER tsnd_mbf(ID, VP, UINT, TMO);	同上(タイムアウト指定)
85	ER_UINT rcv_mbf(ID, VP);	メッセージバッファからの受信
86	ER_UINT prcv_mbf(ID, VP);	同上(ポーリング)
87	ER_UINT trcv_mbf(ID, VP, TMO);	同上(タイムアウト指定)
88	ER ref_mbf(ID, T_RMBF *);	メッセージバッファの状態参照
	ER iref_mbf(ID, T_RMBF *);	同上(非タスクコンテキスト用)

(10)固定長メモリプール管理機能

89	ER cre_mpf(ID, T_CMPF *);	固定長メモリプールの生成
	ER icre_mpf(ID, T_CMPF *);	同上(非タスクコンテキスト用)
90	ER_ID acre_mpf(T_CMPF *);	固定長メモリプールの生成(ID 番号自動割当て)
	ER_ID iacre_mpf(T_CMPF *);	同上(非タスクコンテキスト用)
91	ER del_mpf(ID);	固定長メモリプールの削除
92	ER get_mpf(ID, VP *);	固定長メモリブロックの獲得
93	ER pget_mpf(ID, VP *);	同上(ポーリング)
	ER ipget_mpf(ID, VP *);	同上(ポーリング、非タスクコンテキスト用)
94	ER tget_mpf(ID, VP *, TMO);	同上(タイムアウト指定)
95	ER rel_mpf(ID, VP);	固定長メモリブロックの返却
	ER irel_mpf(ID, VP);	同上(非タスクコンテキスト用)
96	ER ref_mpf(ID, T_RMPF *);	固定長メモリプールの状態参照
	ER iref_mpf(ID, T_RMPF *);	同上(非タスクコンテキスト用)

(11)可変長メモリプール管理機能

97	ER cre_mpl(ID, T_CMPL *);	可変長メモリプールの生成
	ER icre_mpl(ID, T_CMPL *);	同上(非タスクコンテキスト用)
98	ER_ID acre_mpl(T_CMPL *);	可変長メモリプールの生成(ID 番号自動割当て)
	ER_ID iacre_mpl(T_CMPL *);	同上(非タスクコンテキスト用)
99	ER del_mpl(ID);	可変長メモリプールの削除
100	ER get_mpl(ID, UINT, VP *);	可変長メモリブロックの獲得
101	ER pget_mpl(ID, UINT, VP *);	同上(ポーリング)
	ER ipget_mpl(ID, UINT, VP *);	同上(ポーリング、非タスクコンテキスト用)
102	ER tget_mpl(ID, UINT, VP *, TMO);	同上(タイムアウト指定)
103	ER rel_mpl(ID, VP);	可変長メモリブロックの返却
	ER irel_mpl(ID, VP);	同上(非タスクコンテキスト用)
104	ER ref_mpl(ID, T_RMPL *);	可変長メモリプールの状態参照
	ER iref_mpl(ID, T_RMPL *);	同上(非タスクコンテキスト用)

(12)時間管理機能(システム時刻管理)

105	ER set_tim(SYSTIM *);	システム時刻の設定

	ER	iset_tim(SYSTIM *);	同上(非タスクコンテキスト用)
106	ER	get_tim(SYSTIM *);	システム時刻の参照
	ER	iget_tim(SYSTIM *);	同上(非タスクコンテキスト用)

(13)時間管理機能(周期ハンドラ)

107	ER	cre_cyc(ID, T_CCYC *);	周期ハンドラの生成
	ER	icre_cyc(ID, T_CCYC *);	同上(非タスクコンテキスト用)
108	ER_ID	acre_cyc(T_CCYC *);	周期ハンドラの生成(ID 番号自動割当て)
	ER_ID	iacre_cyc(T_CCYC *);	同上(非タスクコンテキスト用)
109	ER	del_cyc(ID);	周期ハンドラの削除
110	ER	sta_cyc(ID);	周期ハンドラの動作開始
	ER	ista_cyc(ID);	同上(非タスクコンテキスト用)
111	ER	stp_cyc(ID);	周期ハンドラの動作停止
	ER	istp_cyc(ID);	同上(非タスクコンテキスト用)
112	ER	ref_cyc(ID, T_RCYC *);	周期ハンドラの状態参照
	ER	iref_cyc(ID, T_RCYC *);	同上(非タスクコンテキスト用)

(14)時間管理機能(アラームハンドラ)

113	ER	cre_alm(ID, T_CALM *);	アラームハンドラの生成
	ER	icre_alm(ID, T_CALM *);	同上(非タスクコンテキスト用)
114	ER_ID	acre_alm(T_CALM *);	アラームハンドラの生成(ID 番号自動割当て)
	ER_ID	iacre_alm(T_CALM *);	同上(非タスクコンテキスト用)
115	ER	del_alm(ID);	アラームハンドラの削除
116	ER	sta_alm(ID, RELTIM);	アラームハンドラの動作開始
	ER	ista_alm(ID, RELTIM);	同上(非タスクコンテキスト用)
117	ER	stp_alm(ID);	アラームハンドラの動作停止
	ER	istp_alm(ID);	同上(非タスクコンテキスト用)
118	ER	ref_alm(ID, T_RALM *);	アラームハンドラの状態参照
	ER	iref_alm(ID, T_RALM *);	同上(非タスクコンテキスト用)

(15)時間管理機能(オーバーランハンドラ)

119	ER	def_ovr(T_DOVR *);	オーバーランハンドラの定義
120	ER	sta_ovr(ID, OVRTIM);	オーバーランハンドラの動作開始
	ER	ista_ovr(ID, OVRTIM);	同上(非タスクコンテキスト用)
121	ER	stp_ovr(ID);	オーバーランハンドラの動作停止
	ER	istp_ovr(ID);	同上(非タスクコンテキスト用)
122	ER	ref_ovr(ID, T_ROVR *);	オーバーランハンドラの状態参照
	ER	iref_ovr(ID, T_ROVR *);	同上(非タスクコンテキスト用)

(16)システム状態管理機能

123	ER	rot_rdq(PRI);	タスクの優先順位の回転
	ER	irot_rdq(PRI);	同上(非タスクコンテキスト用)
124	ER	get_tid(ID *);	実行状態のタスク ID の参照
	ER	iget_tid(ID *);	同上(非タスクコンテキスト用)
125	ER	get_did(ID *);	実行状態のタスクの所属ドメインの参照
	ER	iget_did(ID *);	同上(非タスクコンテキスト用)
126	ER	loc_cpu(void);	CPU ロック状態への移行
	ER	iloc_cpu(void);	同上(非タスクコンテキスト用)
127	ER	unl_cpu(void);	CPU ロック状態の解除

	ER iunl_cpu(void);	同上(非タスクコンテキスト用)
128	ER dis_dsp(void);	ディスパッチの禁止
129	ER ena_dsp(void);	ディスパッチの許可
130	BOOL sns_ctx(void);	コンテキストの参照
131	BOOL sns_loc(void);	CPU ロック状態の参照
132	BOOL sns_dsp(void);	ディスパッチ禁止状態の参照
133	BOOL sns_dpn(void);	ディスパッチ保留状態の参照
134	void vsta_knl(void);	カーネルの起動
	void ivsta_knl(void);	同上(非タスクコンテキスト用)
135	void vsys_dwn(ER, VW, VW, VW);	システムダウン
	void ivsys_dwn(ER, VW, VW, VW);	同上(非タスクコンテキスト用)
136	ER vget_trc(VW, VW, VW, VW);	トレースの取得
	ER ivget_trc(VW, VW, VW, VW);	同上(非タスクコンテキスト用)
137	ER_UINT vchg_cop(ATR);	DSP(TA_COP0)属性の変更

(17)割込み管理機能

138	ER def_inh(INHNO, T_DINH *);	割込みハンドラの定義
	ER idef_inh(INHNO, T_DINH *);	同上(非タスクコンテキスト用)
139	ER chg_ims(IMASK);	割込みマスクの変更
	ER ichg_ims(IMASK);	同上(非タスクコンテキスト用)
140	ER get_ims(IMASK *);	割込みマスクの参照
	ER iget_ims(IMASK *);	同上(非タスクコンテキスト用)

(18)拡張サービスコール・トラップ管理機能

141	ER def_svc(FN, T_DSVC *);	拡張サービスコールの定義
	ER idef_svc(FN, T_DSVC *);	同上(非タスクコンテキスト用)
142	ER_UINT cal_svc(FN, VP_INT, VP_INT, VP_INT, VP_INT);	拡張サービスコールの呼び出し
	ER_UINT ical_svc(FN, VP_INT, VP_INT, VP_INT, VP_INT);	同上(非タスクコンテキスト用)
143	ER vdef_trp(UINT, VT_DTRP *);	トラップルーチンの定義
	ER ivdef_trp(UINT, VT_DTRP *);	同上(非タスクコンテキスト用)

(19)システム構成管理機能

144	ER def_exc(EXCNO, T_DEXC *);	CPU 例外ハンドラの定義
	ER idef_exc(EXCNO, T_DEXC *);	同上(非タスクコンテキスト用)
145	ER ref_cfg(T_RCFG *);	コンフィギュレーション情報の参照
	ER iref_cfg(T_RCFG *);	同上(非タスクコンテキスト用)
146	ER ref_ver(T_RVER *);	バージョン情報の参照
	ER iref_ver(T_RVER *);	同上(非タスクコンテキスト用)

(20)メモリオブジェクト管理機能

147	ER sac_mem(VP, ACVCT *);	メモリオブジェクトのアクセス許可ベクタの変更
148	ER prb_mem(VP, SIZE, ID, MODE);	メモリ領域に対するアクセス許可のチェック
149	ER ref_mem(VP, T_RMEM *);	メモリオブジェクトの状態参照
150	ER vloc_tlb(VP);	TLB エントリのロック
151	ER vunl_tlb(VP);	TLB エントリのロック解除

(21)保護メモリプール管理機能

152	ER_UINT pget_mpp(ID, UINT, VP *);	保護メモリブロックの獲得(ポーリング)
153	ER rel_mpp(ID, VP);	保護メモリブロックの返却

154	ER ref_mpp(ID, T_RMPP *);	保護メモリプールの状態参照

(22)保護メールボックス管理機能

155	ER cre_mbp(ID, T_CMBP *);	保護メールボックスの生成
	ER icre_mbp(ID, T_CMBP *);	同上(非タスクコンテキスト用)
156	ER_ID acre_mbp(T_CMBP *);	保護メールボックスの生成(ID 番号自動割当て)
	ER_ID iacre_mbp(T_CMBP *);	同上(非タスクコンテキスト用)
157	ER del_mbp(ID);	保護メールボックスの削除
158	ER snd_mbp(ID, VP, PRI);	保護メールボックスへの送信
159	ER_UINT rcv_mbp(ID, VP *);	保護メールボックスからの受信
160	ER_UINT prcv_mbp(ID, VP *);	同上(ポーリング)
161	ER_UINT trcv_mbp(ID, VP *, TMO);	同上(タイムアウト指定)
162	ER ref_mbp(ID, T_RMBP *);	保護メールボックスの状態参照
	ER iref_mbp(ID, T_RMBP *);	同上(非タスクコンテキスト用)

(23)システムメモリ管理機能

163	ER vref_syp(VT_RSYP *);	システムプールの状態参照
164	ER vref_rsp(VT_RRSP *);	リソースプールの状態参照

(24)パフォーマンス管理機能

165	ER_UINT vchg_ppc(ID, MODE);	パフォーマンス測定の開始・停止・初期化
	ER_UINT ivchg_ppc(ID, MODE);	同上(非タスクコンテキスト用)
166	ER_UINT vref_ppc(ID, VT_RPPC *);	パフォーマンス測定結果の参照
	ER_UINT ivref_ppc(ID, VT_RPPC *);	同上(非タスクコンテキスト用)

17.2 サービスコールエラーコード一覧

表17.1 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		エラー内容
E_OK	H'00000000	(D'0)	正常終了
E_NOSPT	H'fffffff7	(-D'9)	未サポート機能
E_RSFN	H'fffffff6	(-D'10)	予約機能コード
E_RSATR	H'fffffff5	(-D'11)	予約属性(属性が不正)
E_PAR	H'ffffffef	(-D'17)	パラメータエラー
E_ID	H'ffffffee	(-D'18)	不正 ID 番号
E_CTX	H'ffffffe7	(-D'25)	コンテキストエラー
E_MACV	H'ffffffe6	(-D'26)	メモリアクセス違反
E_ILUSE	H'ffffffe4	(-D'28)	サービスコール不正使用
E_NOMEM	H'ffffffdf	(-D'33)	メモリ不足
E_NOID	H'ffffffde	(-D'34)	ID 番号不足
E_OBJ	H'ffffffd7	(-D'41)	オブジェクトの状態不正
E_NOEXS	H'ffffffd6	(-D'42)	オブジェクトが存在しない
E_QOVR	H'ffffffd5	(-D'43)	キューイングまたはネストのオーバフロー
E_RLWAI	H'ffffffcf	(-D'49)	待ち状態強制解除、または待ち禁止状態の時に待ち状態に移行しようとした
E_TMOUT	H'ffffffce	(-D'50)	ポーリング失敗またはタイムアウト
E_DLT	H'ffffffcd	(-D'51)	待ちオブジェクトが削除された

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
HI7300/PX V1.01

発行年月日　2006 年 7 月 25 日　Rev.3.00
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

HI7300/PX　V.1.01
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J1482-0300