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。

Application Note

RENESAS

HIシリーズOS

アプリケーションノート

ルネサスマイクロコンピュータ開発環境システム

ルネサス エレクトロニクス
www.renesas.com

Rev.3.00　2004.12

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。

【商標などについて】

1. μITRON は、Micro ITRON の略称です。ITRON は Industrial TRON の略称です。TRON は The Realtime Operating System Nucleus の略称です。
2. TRON、ITRON および μITRON は、特定の商品ないしは商品群を指す名称ではありません。
3. μITRON4.0 仕様は、トロン協会 ITRON 部会が中心となって策定されたオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は、ITRON プロジェクトホームページ(http://www.assoc.tron.org/itron/home-j.html)から入手が可能です。
4. Microsoft® Windows® 98 operating system、Microsoft® Windows® Millennium Edition(Windows® Me) operating system、Microsoft® WindowsNT® 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。
5. SuperHTM は、株式会社ルネサス テクノロジの商標です。
6. 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。
7. 本資料に関し詳細についてのお問合せ、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。

はじめに

HI シリーズ OS は、（株）ルネサス テクノロジ製のμITRON 仕様に準拠した機器組み込み用リアルタイム・マルチタスク OS（Operating System）です。

本アプリケーションノート記載内容は、下記に示すエンジニアを対象にしたものです。

対　象	詳　細
ITRON 仕様について知識を有するエンジニア	ITRON 仕様で用いられる用語などについて理解しているレベル
HI シリーズ OS の概要について知識を有するエンジニア	HI シリーズ OS が提供する機能について理解しているレベル
HI シリーズ OS を使用してシステム開発を始めるエンジニア	アセンブリ言語、C 言語記述のプログラミング経験があり、記述プログラムを理解できるレベル

本アプリケーションノートは､HI シリーズ OS を使用したアプリケーションの開発を補足すると同時に、HI シリーズ OS を使用頂いているユーザから、数多く寄せられた質問内容と回答を記載したものです。

〈アプリケーションノートの構成〉

本アプリケーションノートは、以下の内容で構成されています。

章	章内容
第 1 章 HI シリーズ OS の機能	第 1 章では、HI シリーズ OS の機能、オブジェクトなどに関する解説、および FAQ を記載しています。
第 2 章 アプリケーション作成手法	HI シリーズ OS を用いたシステムにおけるアプリケーションプログラムの作成に関する解説、および FAQ を記載しています。
第 3 章 構築	HI シリーズ OS の構築に関する解説、および FAQ を記載しています。
第 4 章 デバイス依存	HI シリーズ OS 使用時、デバイスに依存する内容を FAQ 形式で記載しています。
第 5 章 デバッグ手法	HI シリーズ OS を用いたシステムにおけるアプリケーションプログラムのデバッグに関する解説、および FAQ を記載しています。

なお、HI シリーズ OS の詳細については、ご使用の HI シリーズ OS にて提供しているユーザーズマニュアルもあわせてご参照頂くと、より一層ご理解頂き易くなります。

〈関連マニュアル〉

本アプリケーションノートに関連する以下の製品マニュアルも、あわせて参照してください。

項番	関連マニュアル
1	使用する HI シリーズ OS　ユーザーズマニュアル
2	使用するマイコンのハードウェアマニュアル、プログラミングマニュアル
3	使用するクロスコンパイラ　ユーザーズマニュアル
4	High-performance Embedded Workshop（HEW）ユーザーズマニュアル
5	HI7000/4 構築ガイド
6	HI7700/4 構築ガイド
7	HI7750/4 構築ガイド

〈本アプリケーションノートで使用する用語、記号などの意味〉

用語、記号	用語、記号の意味
H'と D'	“H”'は 16 進数、“D”'は 10 進数を意味するプリフィックスです。プリフィックスのない場合は 10 進数です。
コピーバック	SH-4 シリーズマイコンにおけるキャッシュ方式の呼称です。SH-3、SH3-DSP シリーズマイコンでは「ライトバック」と呼称しますが、本アプリケーションノートでは「コピーバック」と総称します。

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

〈本アプリケーションノートにおける製品名称等の略記〉

製品名称	製品名称の内容
HI7000/4	(株)ルネサステクノロジ製 SH-1、SH-2、SH2-DSP シリーズマイコン用μITRON4.0 仕様準拠 OS の略記です
HI7700/4	(株)ルネサステクノロジ製 SH-3、SH3-DSP、SH4AL-DSP シリーズマイコン用μITRON4.0 仕様準拠 OS の略記です
HI7750/4	(株)ルネサステクノロジ製 SH-4、SH-4A シリーズマイコン用μITRON4.0 仕様準拠 OS の略記です
HI7000/4 シリーズ	HI7000/4、HI7700/4、HI7750/4 を統括した略記です
HI2000/3	(株)ルネサステクノロジ製 H8S ファミリマイコン用μITRON3.0 仕様準拠 OS の略記です
HI1000/4	(株)ルネサステクノロジ製 H8SX ファミリマイコン用μITRON4.0 仕様準拠 OS の略記です
HEW	(株)ルネサステクノロジ製統合開発環境「High-performance Embedded Workshop」の略記です

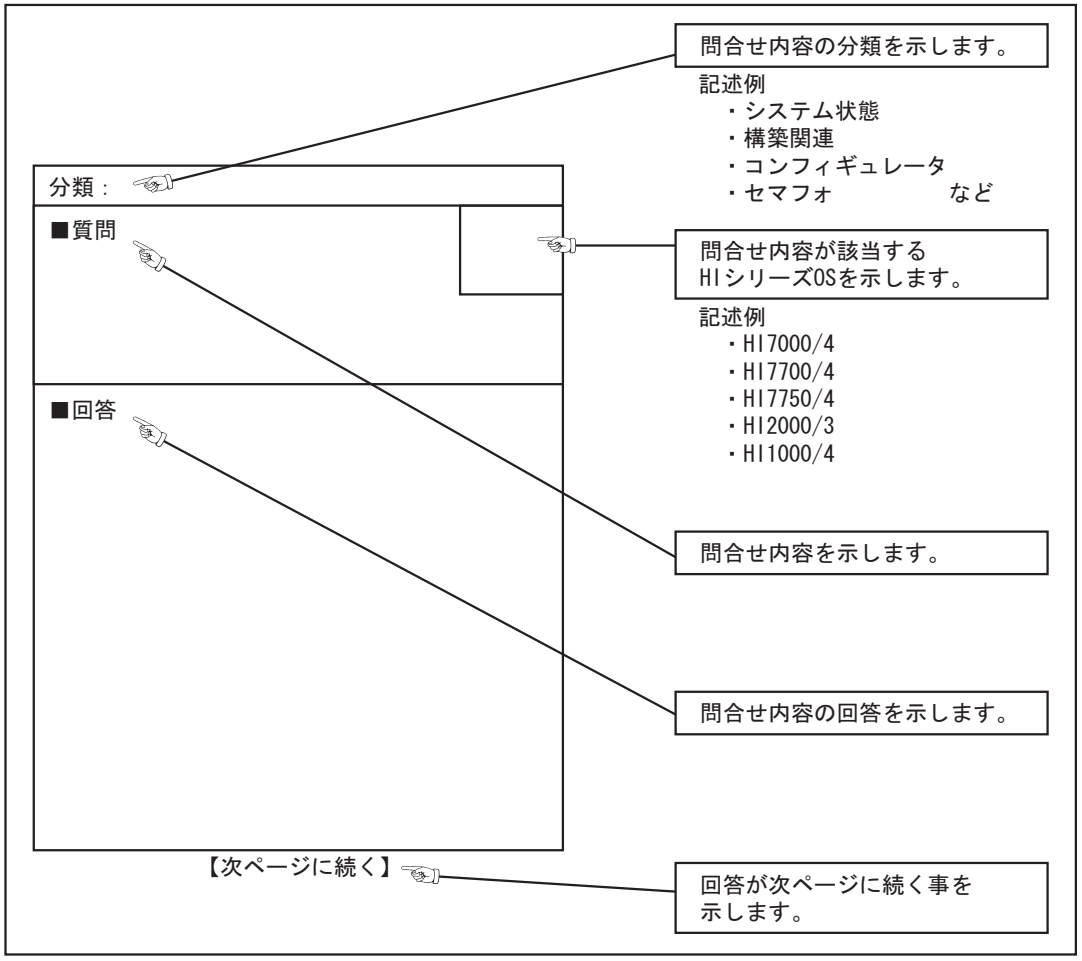
〈本アプリケーションノートで使用するμITRON 仕様について〉

本アプリケーションノートでは、μITRON4.0 仕様をベースに記載しています。μITRON3.0 仕様準拠 OS をご使用の場合は、以下に示す内容にご注意ください。

呼　称	意　味
サービスコール	μITRON4.0 仕様における呼称。μITRON3.0 仕様ではシステムコールと呼称しますが、本アプリケーションノートでは「サービスコール」と総称します。
タスクコンテキスト	μITRON4.0 仕様におけるシステム状態の呼称。μITRON3.0 仕様ではタスク部など、準拠するμITRON 仕様によって呼称が変わりますが、本アプリケーションノートでは「タスクコンテキスト」と総称します。
非タスクコンテキスト	μITRON4.0 仕様におけるシステム状態の呼称。μITRON3.0 仕様では非タスク部、タスク独立部など、準拠するμITRON 仕様によって呼称が変わりますが、本アプリケーションノートでは「非タスクコンテキスト」と総称します。
割込みマスクビット	SuperH™ ファミリマイコンのステータスレジスタ（SR）、および H8S ファミリマイコン、および H8SX ファミリマイコンのコンディションコードレジスタ（CCR）、ならびにエクステンドレジスタ（EXR）における割込みマスクビットを総称します。
オブジェクト	サービスコールの操作対象となるものを「オブジェクト」と総称します。 オブジェクトには、タスク、セマフォ、イベントフラグ、メールボックス、メッセージバッファ、固定長メモリプール、可変長メモリプール、周期ハンドラ、アラームハンドラ、オーバーランハンドラがあります。
メールボックス	μITRON4.0 仕様におけるオブジェクトの呼称。μITRON3.0 仕様では「メイルボックス」と呼称が違いますが、本アプリケーションノートでは「メールボックス」と総称します。
周期ハンドラ	μITRON4.0 仕様におけるオブジェクトの呼称。μITRON3.0 仕様では「周期起動ハンドラ」と呼称が違いますが、本アプリケーションノートでは「周期ハンドラ」と総称します。
初期化ルーチン	μITRON4.0 仕様における呼称。μITRON3.0 仕様では「システム初期化ハンドラ」と呼称が違いますが、本アプリケーションノートでは「初期化ルーチン」と総称します。

〈本アプリケーションノートにおける FAQ 記載構成について〉

本アプリケーションノートに記載している FAQ の構成を以下に示します。



目次

3. 構　築

4. デバイス依存

5. デバッグ手法

図目次

1. HI シリーズ OS の機能

2.　アプリケーション作成手法

3.　構　築

4. デバイス依存

5. デバッグ手法

表目次

1. HI シリーズ OS の機能

2. アプリケーション作成手法

3. 構　築

4.　デバイス依存

5.　デバッグ手法

1. HI シリーズ OS の機能

1.1 システム状態

HI シリーズ OS におけるシステム状態は、以下に示す 2 通りに大別されます。

表 1-1 システム状態

システム状態の名称	システムの状態
タスクコンテキスト （タスク部含む）	タスクを実行している状態、またはコンテキスト
非タスクコンテキスト （非タスク部、タスク独立部含む）	タスク以外の割込みハンドラ、割込みサービスルーチン、タイムイベントハンドラなどが実行されている状態、またはコンテキスト

サービスコール発行時、タスクコンテキスト用、非タスクコンテキスト用にそれぞれ準備されている場合がありますので、システム状態を識別し適切なサービスコールを呼び出す必要があります。

表 1-2 システム状態によるサービスコールの違い

項番	コンテキスト	サービスコール	概 要
1	タスクコンテキスト	xxx_yyy *1	待ちに入ることができる
2	非タスクコンテキスト	ixxx_yyy *1	待ちに入れない

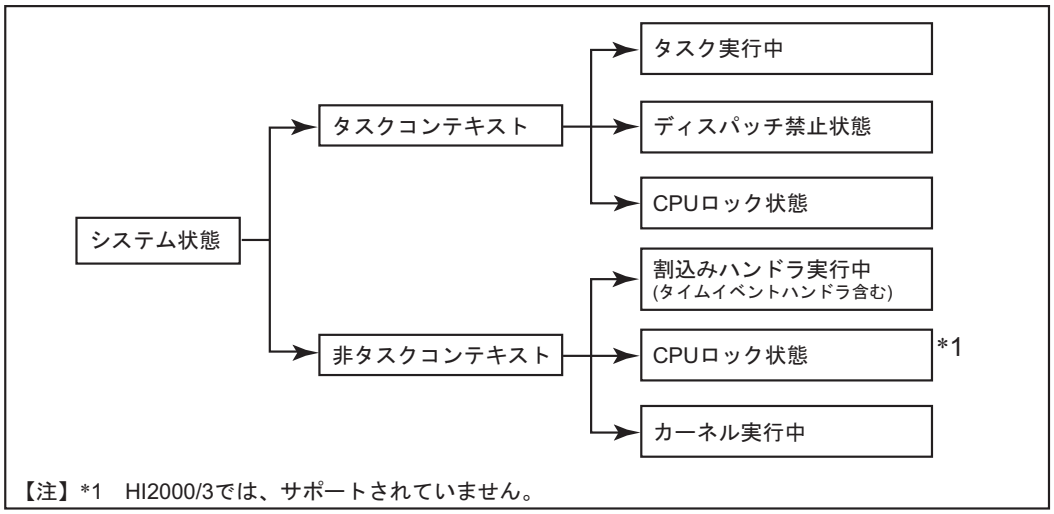
【注】 *1 sns_yyy など、タスクコンテキスト用と非タスクコンテキスト用が同一名のサービスコールもあります。サービスコールの詳細については、ご使用 OS のユーザーズマニュアルを参照してください。

上記のシステム状態は、割込みマスクビット値（IMASK 値）によって識別することもできます。システム状態は上記のほかにも、以下に示す状態があります。

表 1-3 ディスパッチ禁止状態と CPU ロック状態

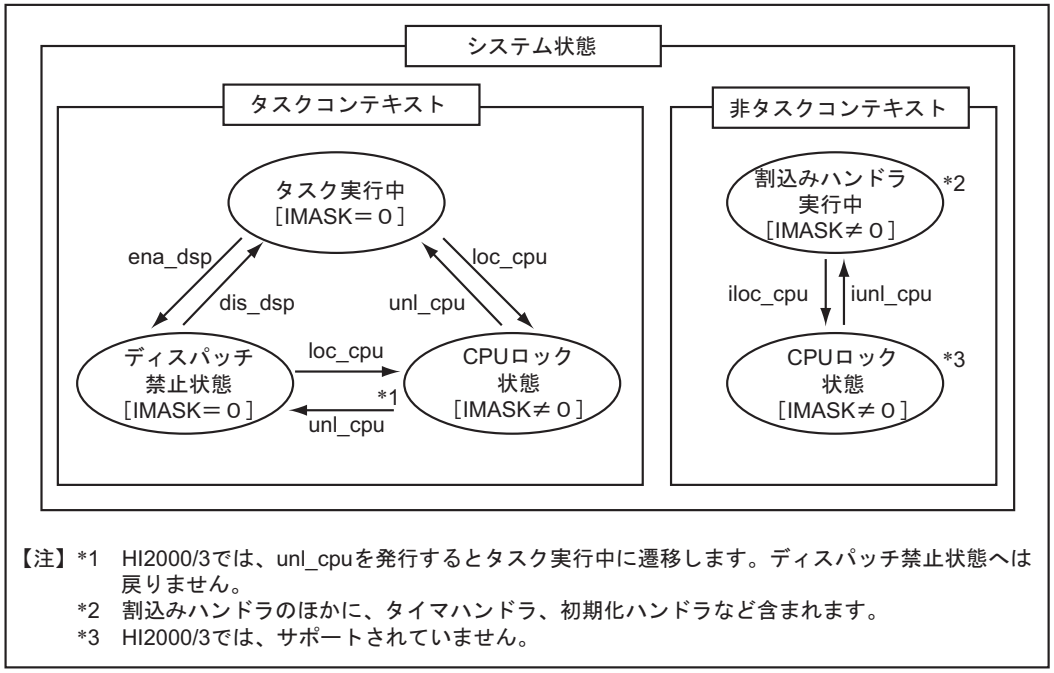
システム状態の名称		システムの状態
ディスパッチ禁止 ／許可状態	ディスパッチ禁止状態	タスクコンテキスト状態で、割込みは受け付けが、ディスパッチ処理は行われない（タスク切り替えが発生しない）状態
	ディスパッチ許可状態	ディスパッチ禁止状態が解除された状態
CPU ロック ／CPU ロック解除状態	CPU ロック状態	割込みの受け付けも、ディスパッチ処理も行われない状態
	CPU 解除状態	CPU ロック状態が解除された状態

上記のシステム状態は、割込みマスクビット値（IMASK 値）によって識別するはできません。システム状態によっては、割込みマスクビット値（IMASK 値）が 0 以外でも、タスクコンテキストとして処理する場合があります。次に示す、HI シリーズ OS におけるシステム状態をご参照ください。



【注】*1　HI2000/3では、サポートされていません。

図 1-1　HI シリーズ OS におけるシステム状態



【注】*1　HI2000/3では、unl_cpuを発行するとタスク実行中に遷移します。ディスパッチ禁止状態へは戻りません。
*2　割込みハンドラのほかに、タイマハンドラ、初期化ハンドラなど含まれます。
*3　HI2000/3では、サポートされていません。

図 1-2　システム状態と割込みマスク値

尚、アプリケーションプログラムとシステム状態の関係については、**「2.6　アプリケーションプログラムの種類」**をご参照ください

次に、タスク、ディスパッチャ（カーネル実行中）、割込みハンドラなどの処理優先順位の関係を示します。

表 1-4　各処理単位の優先順位

処理の優先順位	処理単位
高	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、etc.
↑ ↓	ディスパッチャ（HI シリーズ OS の処理の一部）
低	タスク

- 割込みハンドラの優先順位は、ディスパッチャの優先順位より高い。
- 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）の優先順位は、時間管理機能処理を実行するタイマ割込みハンドラの優先順位以下で、ディスパッチャの優先順位より高い。
- CPU 例外ハンドラの優先順位は、CPU 例外が発生した処理の優先順位と、ディスパッチャの優先順位の何れよりも高い。
- タスクの優先順位は、ディスパッチャの優先順位よりも低い。

1.1.1 システム状態に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたシステム状態に関する質問についての回答を記述します。

《 FAQ 目次 》

(1) タスクコンテキスト／非タスクコンテキストの共通サブルーチン

分類：システム状態

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

共通サブルーチンコール元のコンテキストを識別する方法について、教えてください。

HI7000/4
HI7700/4
HI7750/4

HI1000/4

■回答

サブルーチンが呼び出されたシステム状態を識別するには、sns_ctx サービスコール（コンテキストの参照）を使用します。リターンパラメータ「BOOL state」に、TRUE（＝0）が返された場合は非タスクコンテキスト、FALSE（≠0）が返された場合はタスクコンテキストとなります。以下にコンテキストを識別するプログラムのコーディング例を示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void Common_Sub_Routine(VP_INT exinf)
{

BOLL state;

      《 中略 》

    state = sns_ctx();
    if(state == TURE){     /* 非タスクコンテキストからの呼出し */
       《 処理中略 》
    }
    else{                  /* タスクコンテキストからの呼出し */
       《 処理中略 》
    }

}
```

図 1-3　コンテキスト識別コーディング例(HI7000/4 シリーズ、HI1000/4)

【次ページに続く】

【前ページからの続き】

HI2000/3

■回答

サブルーチンが呼び出されたシステム状態を識別するには、ref_ims システムコール（割込みマスクレベルの参照）を使用します。リターンパラメータ「UINT imask」に、0 が返された場合はタスク部、0 以外が返された場合は非タスク部となります。以下にコンテキストを識別するプログラムのコーディング例を示します。

```
#include "hi2000.h"

void Common_Sub_Routine(INT stacd)
{

ER   ercd;
UINT imask;

      《 中略 》

    ercd = ref_ims(&imask);
    if(imask != 0){
        /* 非タスクコンテキスト、またはCPUロック状態から */
        /* 呼び出された場合の処理                       */
    }
    else{
        /* タスクコンテキストから呼び出された場合の処理 */
    }

}
```

図 1-4　コンテキスト識別コーディング例(HI2000/3)

タスク部実行中の CPU ロック状態のから ref_ims システムコールを発行した場合、リターンパラメータ「UINT imask」に 0 以外が返され、非タスクコンテキストと判断されます。

したがって、HI2000/3 では ref_ims システムコールを使用しても、非タスクコンテキストと CPU ロック状態を識別する手段がないため、共通サブルーチンに引数を持たせるなどアプリケーション側で管理する必要があります。

(2) CPU の占有

分類：システム状態

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

特定のタスク実行中、他のアプリケーションの実行を一切禁止したい（カーネル動作も）のですが、良い方法がありますか。

■回答

loc_cpu サービスコールを使用することにより対応できます。
loc_cpu 後は、カーネル割込みマスクレベルで割込み応答やタスク切り替えを禁止します。ただし、カーネル割込みマスクレベル以上の割込みレベルに設定されている割込みは受け付けますので設定に注意してください。CPU を占有した処理が終了したら、必ず unl_cpu サービスコールで CPU ロック状態を解除してください。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma noregsave(task)

void task(VP_INT exinf)
{

BOLL state;

      《 中略 》

    loc_cpu();              /* CPUロック状態の開始 */
      /* CPUロック状態で行う処理の開始 */

        《 処理中略 》

      /* CPUロック状態で行う処理の終了 */
    unl_cpu();              /* CPUロック状態の解除 */

      《 中略 》

}
```

図 1-5　loc_cpu 使用コーディング例

1.2 オブジェクト

1.2.1 オブジェクトとは

タスクなどサービスコールの操作対象となるものを「オブジェクト」と総称します。

各オブジェクトは、同時に複数個存在できるため、識別のために ID 番号が与えられ、この番号により管理します。

1.2.2 ID 割り付け

オブジェクトの ID 番号は、オブジェクト生成時に割り付けます。

オブジェクトへの ID 番号割り付け方法を、以下に示します。

表 1-5 オブジェクトへの ID 番号割り付け方法

HI シリーズ OS	ID 番号割り付け方法
HI7000/4 シリーズ	・コンフィギュレータによる割り付け ・サービスコール発行による割り付け
HI2000/3	・セットアップテーブルによる割り付け
HI1000/4	・コンフィギュレータによる割り付け

HI2000/3、および HI1000/4 は、動的生成（サービスコールによる生成）機能がないため、セットアップテーブル、およびコンフィギュレータで予め割り付けておく必要があります。

1.2.3 オブジェクトに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたオブジェクトに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) 登録タスクとタスク ID の関連

分類：オブジェクト

■質問

HI2000/3

　タスク ID はタスクの登録順に 1 から割り当てるのですか。それとも他の方法で任意の ID（値）を割り付けることができるのですか。

■回答

登録タスクの ID 割り付けは、登録順に 1 から順に割り付けます。
HI2000/3 では動的タスク生成機能がないため、あらかじめ定義しておく必要があります。
以下にサンプル例を示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    TASK define section                                             %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ------------------------------------------------------------
;               TASK_TOP_LABEL                  ;: COMMENT
;-------------------------------------------------------------------------
        .import _TASKA                          ;: TASK.C
        .import _TASKB                          ;: TASK.C
;
```

使用するタスク先頭アドレスを、
外部参照シンボルとして宣言
（必要に応じて追加、変更してください）

図 1-6　セットアップテーブル(H8S/2655 用 2655asup.src)(1/2)

【次ページに続く】

【前ページからの続き】

■回答

```
;------ Usage ---------------------------------------------------------------
;               .res.b  SIZE +TSKSTKSIZ ;[RANGE];: COMMENT
;TSK?_SP_LABEL: .equ    $                       ;: COMMENT
;--------------------------------------------------------------------------
TSKSTKSIZ:      .equ    50+(10*2)+(6*1)+6+8;[50...];: Task minimum stack size
          .section      h2sstack,stack,align=2
                .res.b  (36) +TSKSTKSIZ ;[50...];: tskid1 stack area
TSK1_SP:        .equ    $
                .res.b  8
                .res.b  (36) +TSKSTKSIZ ;[50...];: tskid2 stack area
TSK2_SP:        .equ    $
                .res.b  8
                .res.b  (32) +TSKSTKSIZ ;[50...
TSK3_SP:        .equ    $
                .res.b  8
                .res.b  (32) +TSKSTKSIZ ;[50...
TSK4_SP:        .equ    $
                .res.b  8
;
        .section        h2ssetup,code,align=2
_HI_H8S:        .res.b  10                      ;: System Area
;------ Usage ---------------------------------------------------------------
;LABEL          .data.b IMOD, ITSKPRI
;               .data.l ITSKADR, ITSKSP
;--------------------------------------
NOEXS:          .assign 0
RDY:            .assign 1
DMT:            .assign (-1)
TDTLEN:         .assign 10;<- Not Change
        .section        h2ssetup,code,al
_HI_TDT:        .equ    $-TDTLEN
TDT_TOP:        .equ    $
tdt_id1:        .data.b DMT, 1
                .data.l _TASKA, TSK1_SP
tdt_id2:        .data.b DMT, 2
                .data.l _TASKB, TSK2_SP
tdt_id3:        .data.b NOEXS, 3
                .data.l 0, TSK3_SP
tdt_id4:        .data.b NOEXS, 4
                .data.l 0, TSK4_SP
tdt_id5:        .data.b NOEXS, 5
                .data.l 0, TSK4_SP
TDT_BTM:
TSKCNT:         .equ    (TDT_BTM-TDT_TOP
                                ;:[0...2
;
```

〔タスクスタック領域の定義〕
1行目：使用スタックサイズの定義
2行目：スタックラベルの定義
（タスクスタックボトム）
3行目：共有スタック管理領域の定義
（共有スタック機能を
使用しない場合は不要）

〔タスクの定義〕
【書式】
LABEL： .data.b IMOD, ITSKPRI
.data.l ITSKADR, ITSKSP

①LABEL
自由に設定可(定義がなくても問題なし)
②IMOD(タスク初期状態)
タスクの登録と初期起動時の状態を定義
(1) NOEXS(=0)：未登録
(2) RDY (=1)：起動時、実行可能状態
(3) DMT (=-1)：起動時、休止状態
③ITSKPRI(タスク初期優先度)
タスクの初期優先度を定義
④ITSKADR(タスク初期起動アドレス)
タスクの先頭アドレスを定義
※外部参照シンボルで定義した
先頭アドレスを定義
⑤ITSKSP(タスク初期スタックポインタ)
タスク起動時のスタックポインタ(ボトム
アドレス)を定義
※タスクスタック領域定義部で定義した
スタックラベルを定義

図 1-6　セットアップテーブル(H8S/2655 用 2655asup.src)(2/2)

(2) コンフィギュレータの静的定義情報

分類：オブジェクト

HI7000/4
HI7700/4
HI7750/4

■質問

コンフィギュレータでタスクやイベントフラグなどを生成（定義）する項目がありますが、この項目は、静的に生成する場合に定義（記述）が必要で、コード上で動的に生成する場合（cre_tsk など）は定義（記述）してはいけないのでしょうか。

■回答

各オブジェクトの生成ビューの定義は必須ではありません。コード（プログラム）上で動的に生成する場合は定義する必要はありません。各オブジェクトの生成ビューに定義して生成した場合は、コード（プログラム）上で生成する必要がなく、システム起動時、直ちにオブジェクトを使用することができます。

コンフィギュレータにおけるオブジェクト（タスクやイベントフラグなど）の定義項目で、最大オブジェクト ID（各オブジェクトの最大使用数）の定義は必須です。この定義がないと、コード（プログラム）上で動的に生成できない場合があります。

1.3 サービスコールのパラメータチェック

HI シリーズ OS では、サービスコール発行時のコンテキスト状態やパラメータなどのチェックを分けています。チェックするパラメータは、以下のように大別されます。

- 動的パラメータ：システム動作中に動的に変化するパラメータ
  - タスクやセマフォなどのオブジェクトの有無
  - サービスコール発行時のコンテキスト状態
  - 対象タスクの状態　など
- 静的パラメータ：
  - 指定された ID に対する最大値　など

以下に、パラメータチェック機能の有無による相違内容を示します。

表 1-6　パラメータチェック機能有無の違い

パラメータチェック	チェック内容	メリット	デメリット
組み込まない場合 ※パラメータチェック機能無し	動的パラメータ	– 処理が早い – プログラムサイズが小さい	– サービスコールのパラメータ誤りでシステムが暴走した場合等の解析が困難
組み込んだ場合 ※パラメータチェック機能有り	静的パラメータ 動的パラメータ	– デバッグに有効	– 処理が遅い – プログラムサイズが大きい

HI シリーズ OS におけるエラーチェックの設定方法（組み込み方法）については、当該 OS の項を参照してください。

1.3.1 HI7000/4 シリーズにおける組み込み方法

HI7000/4 シリーズにおける「パラメータチェック機能」の組み込み設定は、コンフィギュレータのカーネル拡張機能ビューにて行います。

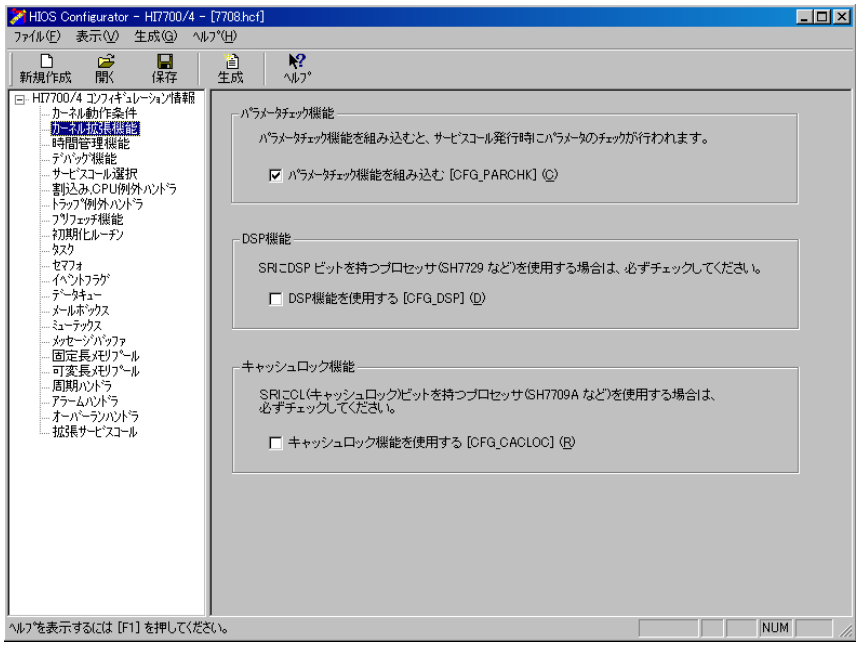


図 1-7 カーネル拡張機能ビュー

カーネル拡張機能ビューの「パラメータチェック機能」にて、「パラメータチェック機能を組み込む[CFG_PARCHK](C)」のチェックボックスを選択する事で組み込むことが出来ます（コンフィギュレータのデフォルト設定です）。

1.3.2 HI2000/3、HI1000/4 における組み込み方法

HI2000/3、および HI1000/4 における「パラメータチェック機能」の組み込み設定は、構築時に使用する（組み込む）ライブラリファイルによって行います。

構築時、当該 HEW プロジェクトファイルにて、「パラメータチェック機能あり」ライブラリファイルを定義する事で「パラメータチェック機能」が組み込まれます（標準提供の HEW プロジェクトファイルではデフォルト設定です）。

以下にライブラリファイルの変更手順を、「H8S,H8/300 Series C/C++ Compiler Package V6.0.00」、HI2000/3 V1.10r1 の H8S/2655 アドバンストモードで使用した場合を例に示します。

起動した HEW ワークスペースにて、ヘッダメニューのオプション[O]→H8S,H8/300 Standard Toolchain…を選択します。

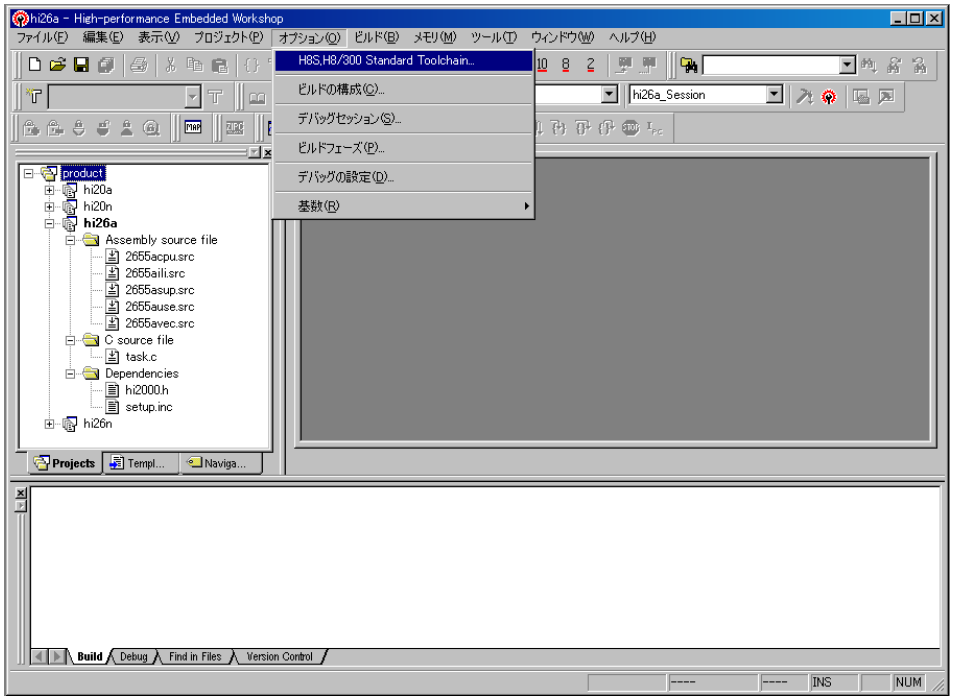


図 1-8 ライブラリファイルの定義(1)

「H8S,H8/300 Standard Toolchain」ダイアログの最適化リンカタグを選択すると、現在設定情報が表示されます。表示内容を以下に示します。

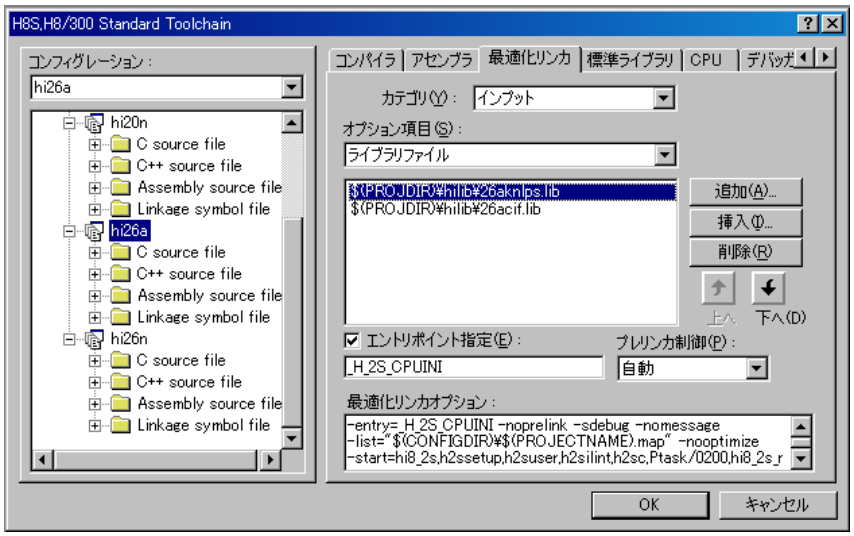


図 1-9　ライブラリファイルの定義(2)

デフォルトで設定されているカーネルライブラリファイルは、「パラメータチェックあり」です。デフォルト設定のライブラリファイルから、パラメータチェック機能なしへの変更を例に示します。現在設定されているライブラリファイルを選択し、［挿入(I)...］ボタンを押下します。

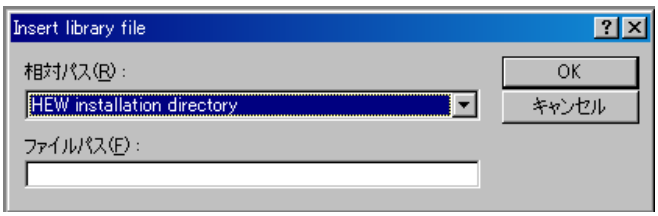


図 1-10　ライブラリファイルの定義(3)

［相対パス(R) : ］と［ファイルパス(F) : ］を入力し、［OK］ボタンを押下します。

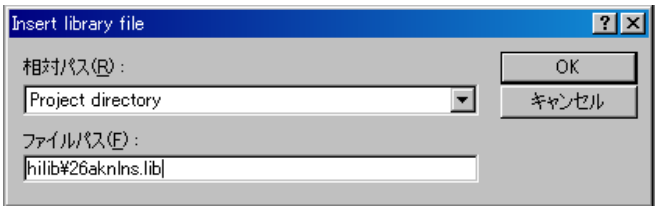


図 1-11　ライブラリファイルの定義(4)

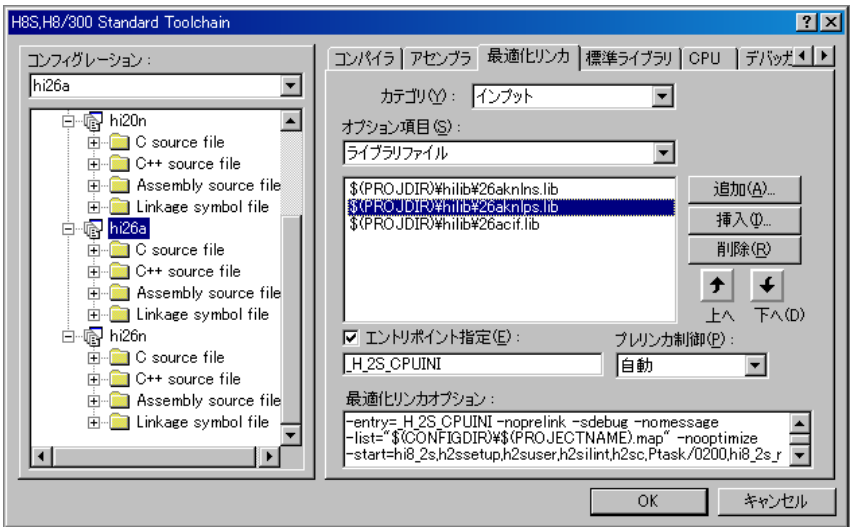


図 1-12　ライブラリファイルの定義(5)

「H8S,H8/300 Standard Toolchain」ダイアログにて、既存ライブラリファイルを選択し、［削除(R)］ボタンを押下します。

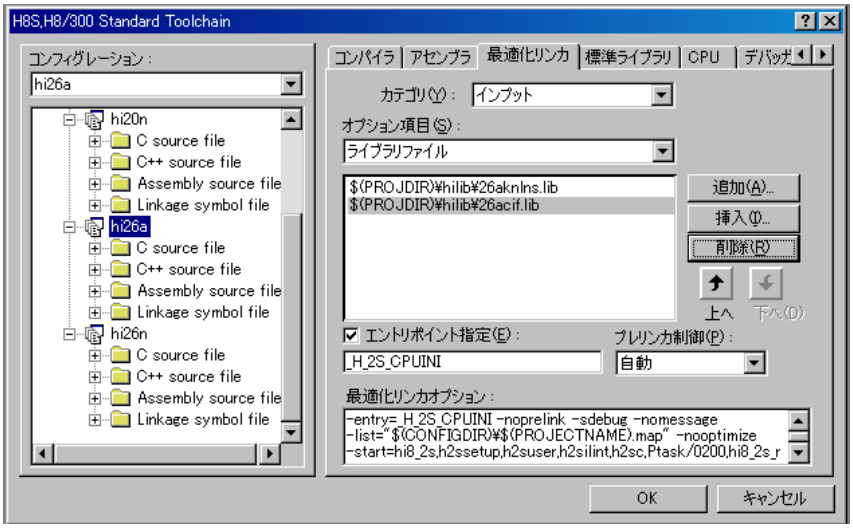


図 1-13　ライブラリファイルの定義(6)

「H8S,H8/300 Standard Toolchain」ダイアログ内の［OK］ボタンを押下すると設定内容が HEW ワークスペースに反映されます。

このようにして、カーネルライブラリファイルの変更を行います。

1.3.3 サービスコールのパラメータチェックに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたサービスコールのパラメータチェックに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) パラメータチェックの有無

分類：サービスコールのパラメータチェック

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

OS が提供している機能ライブラリが、「パラメータチェックなし」と「パラメータチェックあり」に分かれていますが、どのように使い分ければよいのですか。

■回答

OS が提供するライブラリで「パラメータチェックあり」のライブラリは、デバッグ用に準備したものです。作成したアプリケーションプログラムのコーディング不正（パラメータなどの正当性）を摘出するものです。

「パラメータチェックあり」のライブラリを使用する事で、通常のサービスコール処理時間に比べ、パラメータのチェック処理分、オーバヘッドが大きくなり、さらに処理コードが増加します。

デバッグ完了後は、「パラメータチェックなし」のライブラリを使用して生成したロードモジュールを組み込むことをお勧めします。

【注】 動的パラメータのチェックは、パラメータチェック機能の組み込みに関わらず、常に行われます。

1.4 タスク

1.4.1 タスクと関数の関係

タスクと関数の違いを以下に示します。

表 1-7 タスクと関数の違い

項目	タスク	関数
プログラム記述	記述的な相違は特になし ※タスクは一つの関数（または複数の関数の集合体）で構成される ケースもあります。	
起動	優先度と起動順序から OS が判断	main 関数から起動
管理	OS	関数
インタフェース	OS の機能（サービスコールなど）	引数
依存度、結合度	素結合で依存関係はない	密結合で依存度は大きい

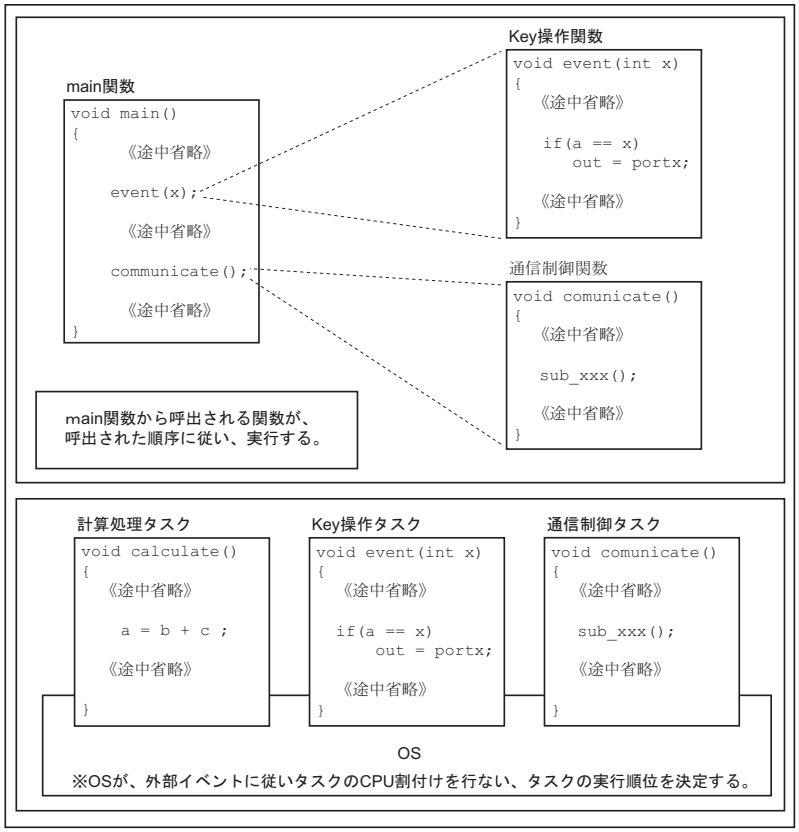


図 1-14 タスクと関数の違い

1.4.2 タスクの起動

以下にタスクが起動するまでを説明します。

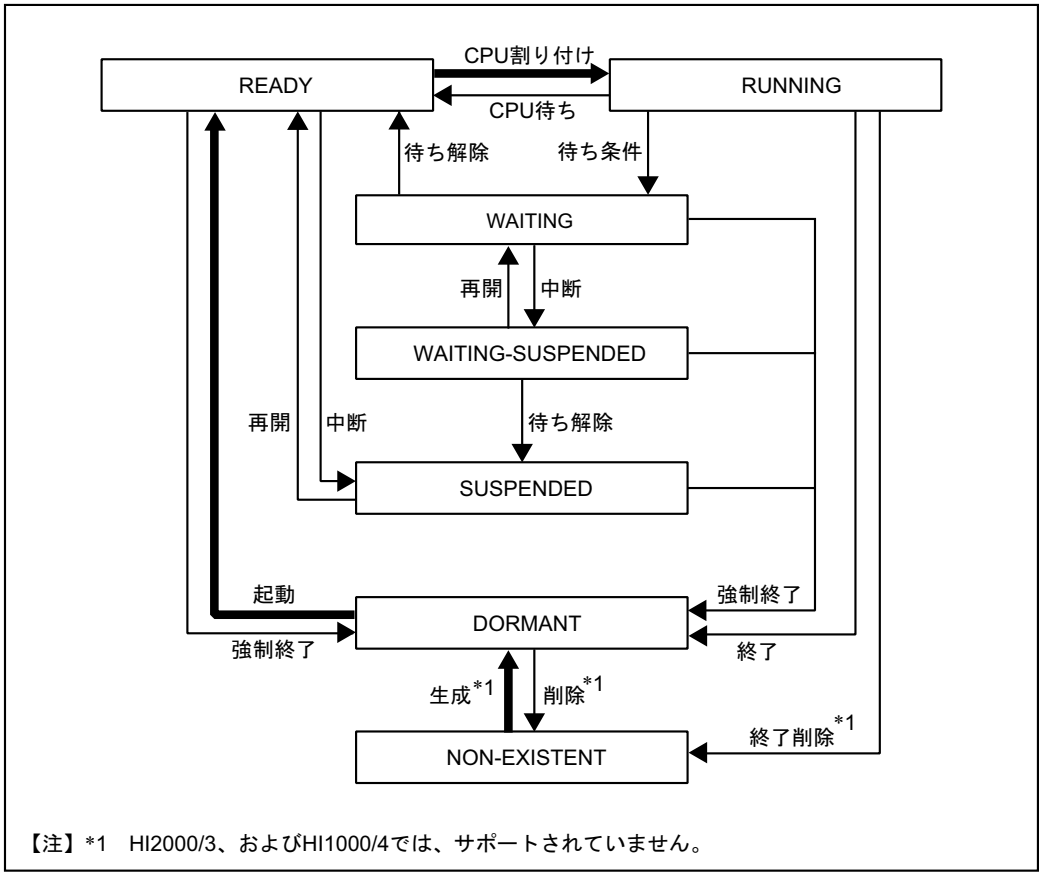


図 1-15 タスクの状態遷移

項番	タスクの状態	説明
1	NON-EXISTEND （未登録状態）	カーネルに登録されていない状態
	↓ cre_tsk()*1、acre_tsk()*1 etc.	
2	DORMANT （休止状態）	登録されたが起動指示を受けていない状態
	↓ sta_tsk()、act_tsk() etc.	
3	READY （実行可能状態）	実行準備が整い、CPU 割り付けを待っている状態
4	RUNNING （実行状態）	CPU が割り付けられ、実行している状態

【注】*1 HI2000/3、および HI1000/4 では、サポートされていません。

1.4.3 タスクのスタック

タスクが使用するスタックについて、以下に示します。

表 1-8 タスクスタックの種類と HI シリーズ OS の対応

スタックの種類	HI7000/4 シリーズ	HI2000/3	HI1000/4
ダイナミックスタック	○	×	×
スタティックスタック	○	○	○
（共有スタック機能）	（○）	（○）	（○）

○：使用可、×：使用不可

各スタック領域の確保、および共有スタック機能に関する詳細は、ご使用の HI シリーズ OS ユーザーズマニュアルをご参照ください。

(1) タスクスタックの種類

以下に、タスクスタックの種類を示します。

表 1-9 タスクスタックの種類

スタックの種類	内 容
ダイナミックスタック	必要なスタックサイズを OS が管理する領域から確保し、タスク起動時に割り当てる。
スタティックスタック	タスクごとに使用するスタック領域を確保し、タスク起動時に割り当てる。

(2) 共有スタック機能

スタティックスタックを使用するタスクでは、複数のタスクで 1 つのスタック領域を共有すること（共有スタック機能）ができます。共有スタック機能を使用することで、タスクスタックサイズを抑止することができます。

ダイナミックスタック、スタティックスタック、共有スタックにおけるメモリの使用について、以下に示します。

表 1-10 スタックの種類とメモリの使用

スタックの種類	メモリ容量
ダイナミックスタック	– 全タスクスタックの合計サイズ(Σ)は確保する必要なし
スタティックスタック （共有スタック機能）	– 全タスクスタックの合計サイズ(Σ)を確保する必要ある – 共有スタック機能を使用すると、複数のタスク間で 1 つのタスクスタック領域を共有できるためメモリ容量を抑止可能

共有スタック機能を使用するタスクの状態遷移を以下に示します。

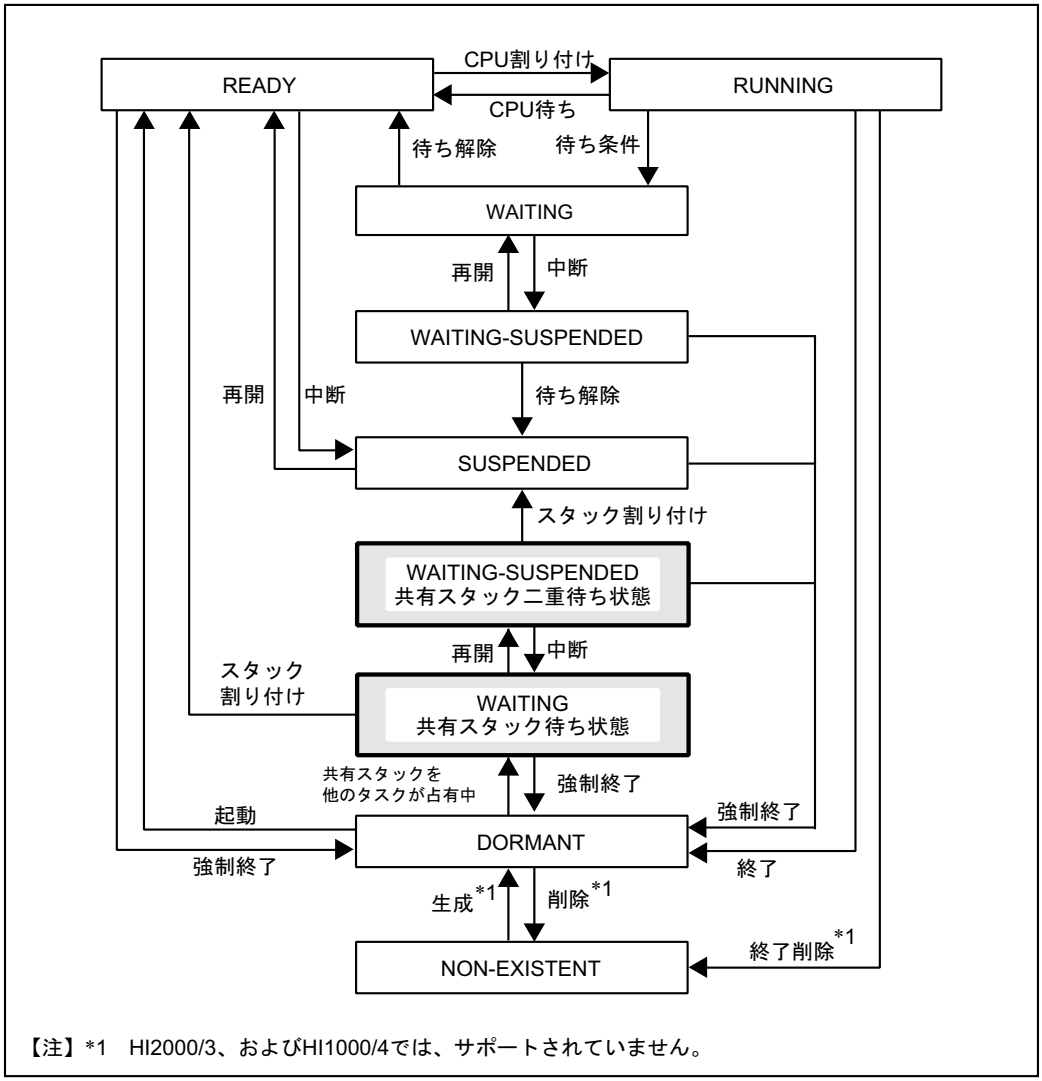


図 1-16　共有スタック機能を使用するタスクの状態遷移

【注】ダイナミックスタックを使用しているタスクでは、共有スタック機能は使用できません。

1.4.4 タスクの CPU 割り付け

タスクへの CPU 割り付けは、タスクに定義された優先度を基準にして行われます。タスクの優先度は、値が小さいほど優先度が高く、大きいほど優先度は低くなります。

また、タスク間の優先順位は、タスクの優先度に基づいて決定されます。この様子を以下のように仮定して図解説します。

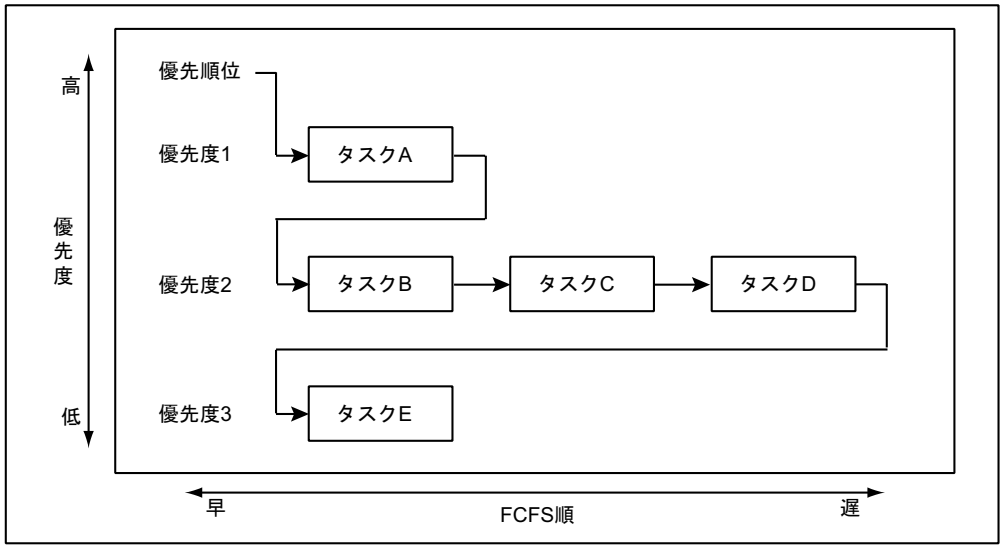


図 1-17 優先順位(1)

タスク A が実行権放棄（タスク終了（&削除）サービスコールを呼び出した、またはサービスコールによる事象待ち状態に移行）したときの優先順位を以下に示します。

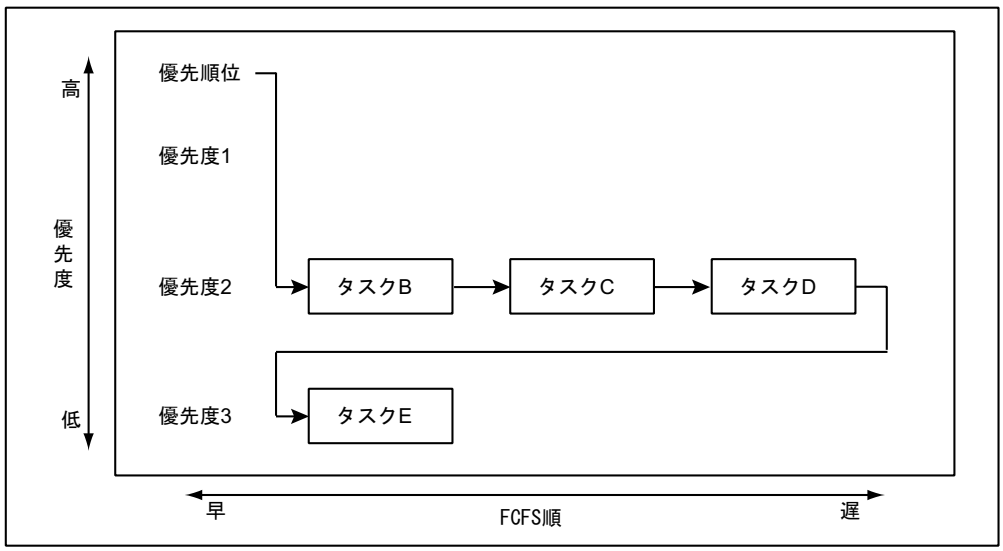


図 1-18 優先順位(2)

タスク B が実行権放棄（サービスコールによる事象待ち状態に移行）すると、タスク C が実行状態になります。その後、タスク B の待ち状態が解除された時の優先順位を以下に示します。

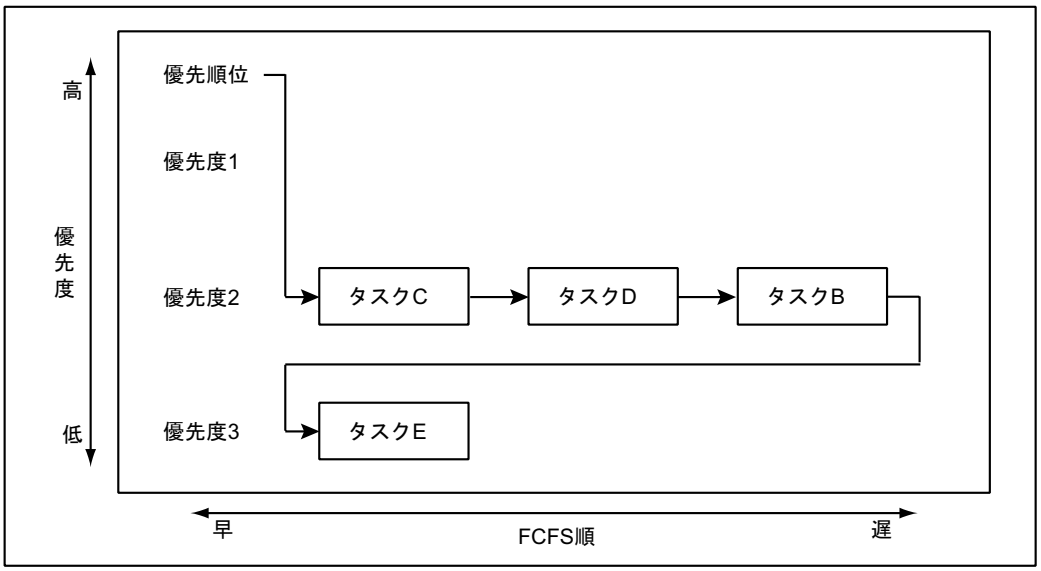


図 1-19　優先順位(3)

【注】　同一優先度を持つタスクは、FCFS 方式(First Come First Service：最初に来たものから処理する)の原則により、同一優先度の最後にスケジューリング(優先度キューへ挿入)されます。

低い優先順位のタスクが実行されている間に、高い優先順位のタスクが実行可能状態になると、実行中だった低い優先度のタスクは処理を中断され（実行状態から実行可能状態に移る）、高い優先順位のタスクを実行します。

μITRON 仕様では、「高い優先順位のタスクによって低い優先順位のタスクの実行が中断される」ことを「プリエンプト（Preempt）」とよびます。

次に優先度の違う他タスクへのサービスコール発行による優先度の変化、自タスクへのサービスコール発行による優先度の変化について示します。

(1) 他タスクへのサービスコール発行の場合

最初の状態を以下のように仮定します。

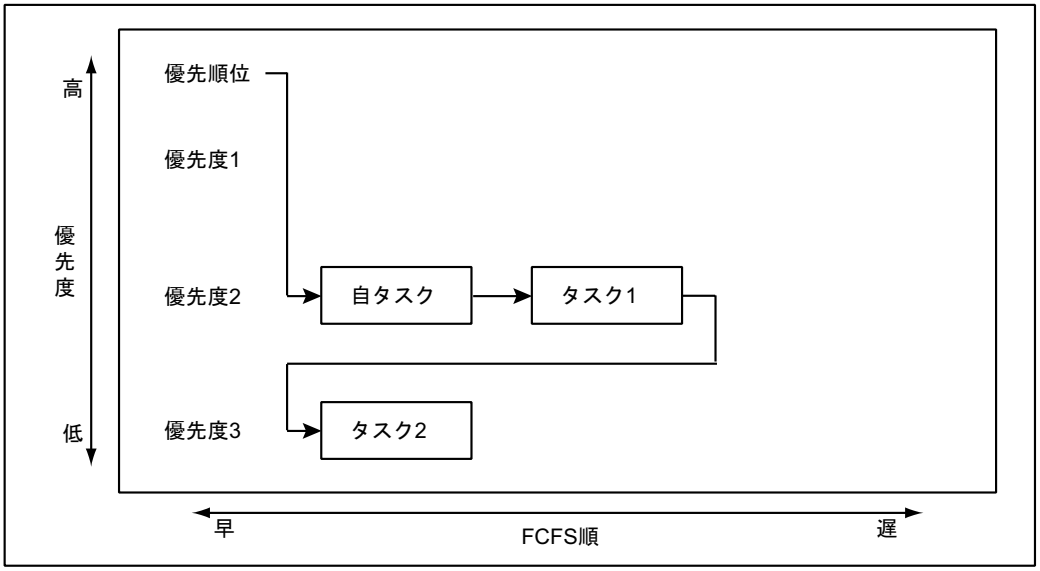


図 1-20 他タスクへサービスコール発行前の優先順位

サービスコールを発行するタスク（以下、自タスクと称す）と同等の優先度を持つ「タスク A」と、自タスクより低い優先度を持つ「タスク B」に起動要求サービルコール（sta_tsk、または act_tsk など）を発行し、タスク A とタスク B が実行可能状態になった状態を以下に示します。

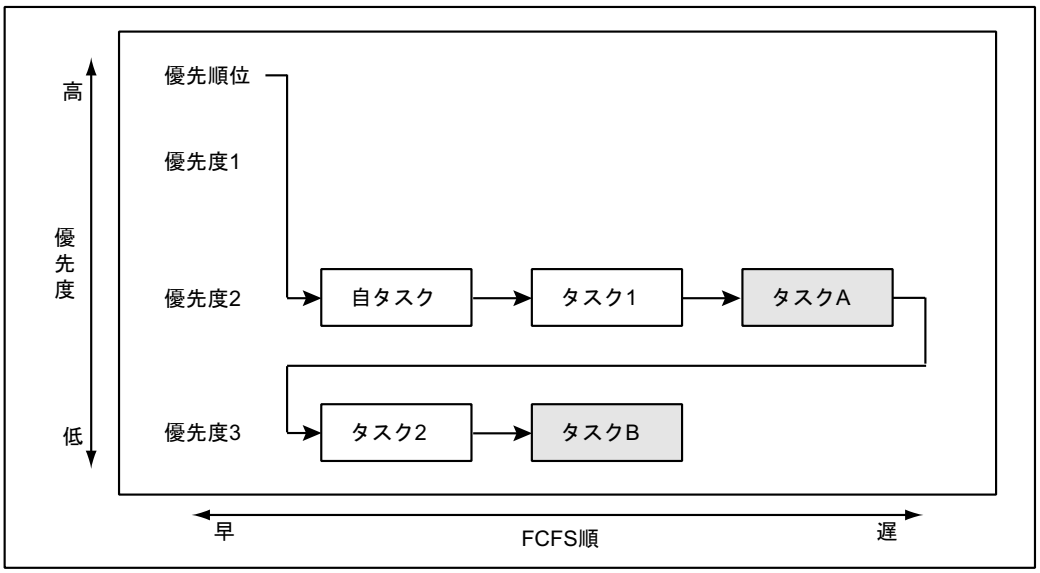


図 1-21 他タスクへサービスコール発行後の優先順位(1)

次に、自タスクより高い優先度を持つ「タスク C」に対して起動要求サービスコールを発行し、タスク C が実行可能状態になった状態を以下に示します。

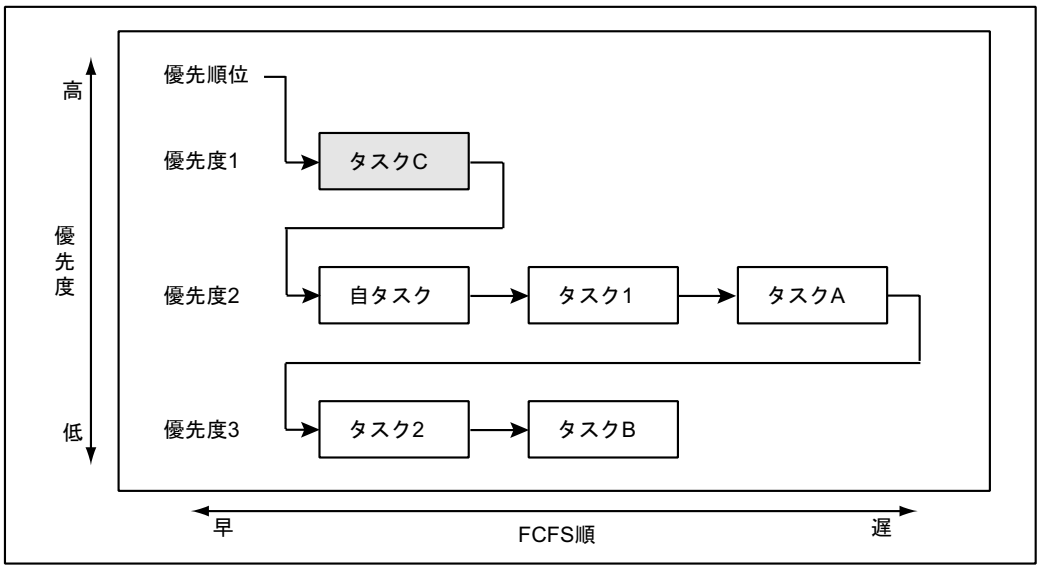


図 1-22　他タスクへサービスコール発行後の優先順位(2)

このように、サービスコールを発行したタスクの優先順位が、自タスクより高い優先度を持つ「タスク C」によって変わるため、自タスクはサービスコール発行時点で「プリエンプト」されてしまいます。

(2) 自タスクへのサービスコール発行の場合

自タスクへのサービスコール発行による優先順位の遷移について解説します。最初の状態を以下のように仮定します。

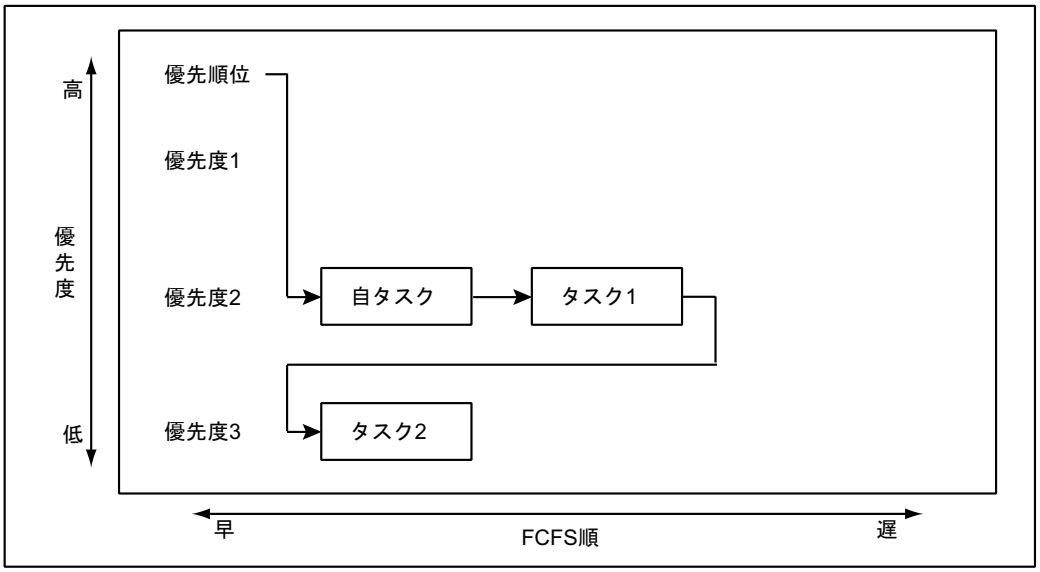


図 1-23 自タスクへサービスコール発行前の優先順位

自タスク優先度を上げた（優先度 2→優先度 1 に変更）場合の優先順位の遷移を以下に示します。

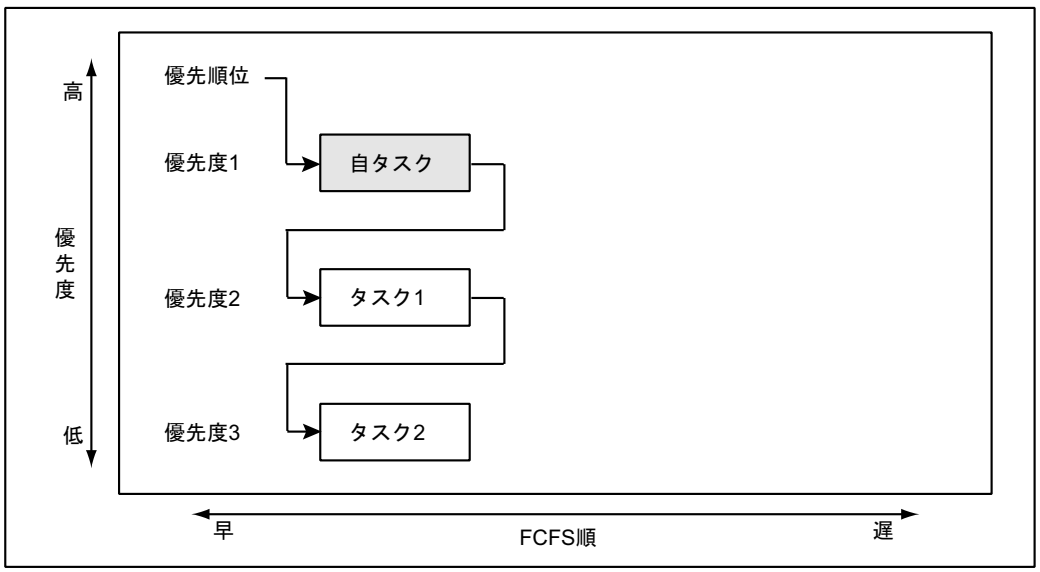


図 1-24 自タスクへサービスコール発行後の優先順位(1)

次に、自タスクの優先度を元に戻した（優先度 1→優先度 2 に変更）場合の状態を以下に示します。

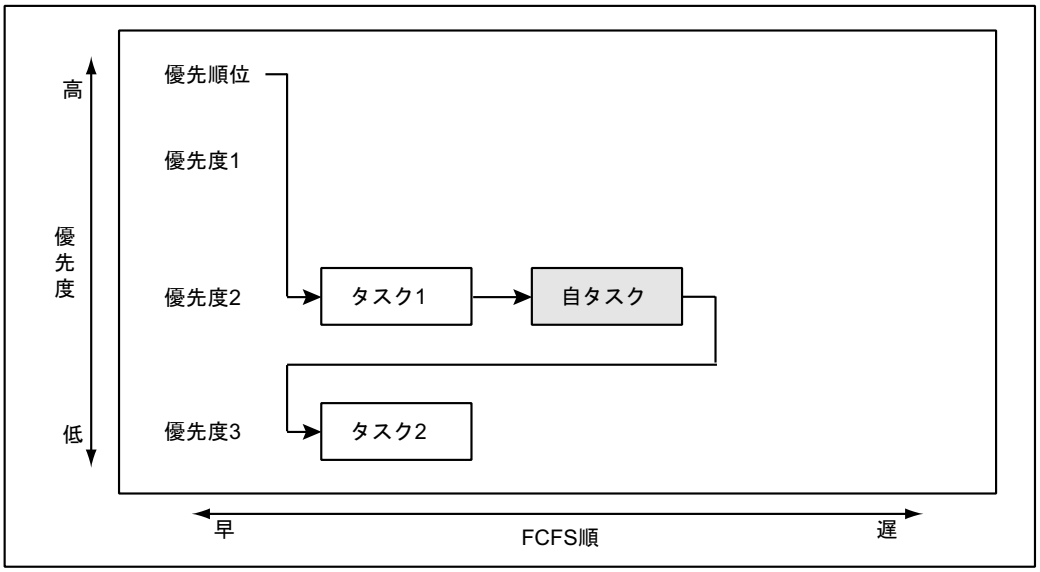


図 1-25　自タスクへサービスコール発行後の優先順位(2)

優先度変更後、自タスクと同一優先度におけるタスクの有無で処理が違います。
以下に、変更後の同一優先度におけるタスクの有無による処理の違いを示します。

表 1-11　変更後の同一優先度におけるタスクの有無による処理の違い

タスクの有無	自タスクへの処理	自タスクの実行
有り ※存在する場合	スケジューリング規則（FCFS 方式）により、同一優先度の最後にキューイング	プリエンプト
無し ※存在しない場合	同一優先度の先頭にキューイング	処理を継続

1.4.5 ポーリング

オブジェクトにおける事象発生待ちサービスコールには、通常の待ち、タイムアウトあり、ポーリングの 3 種類があります。イベントフラグを例にそれぞれの処理の違いを示します。

通常の事象発生待ちサービスコール（wai_flg サービスコールを例に）の処理概要を示します。

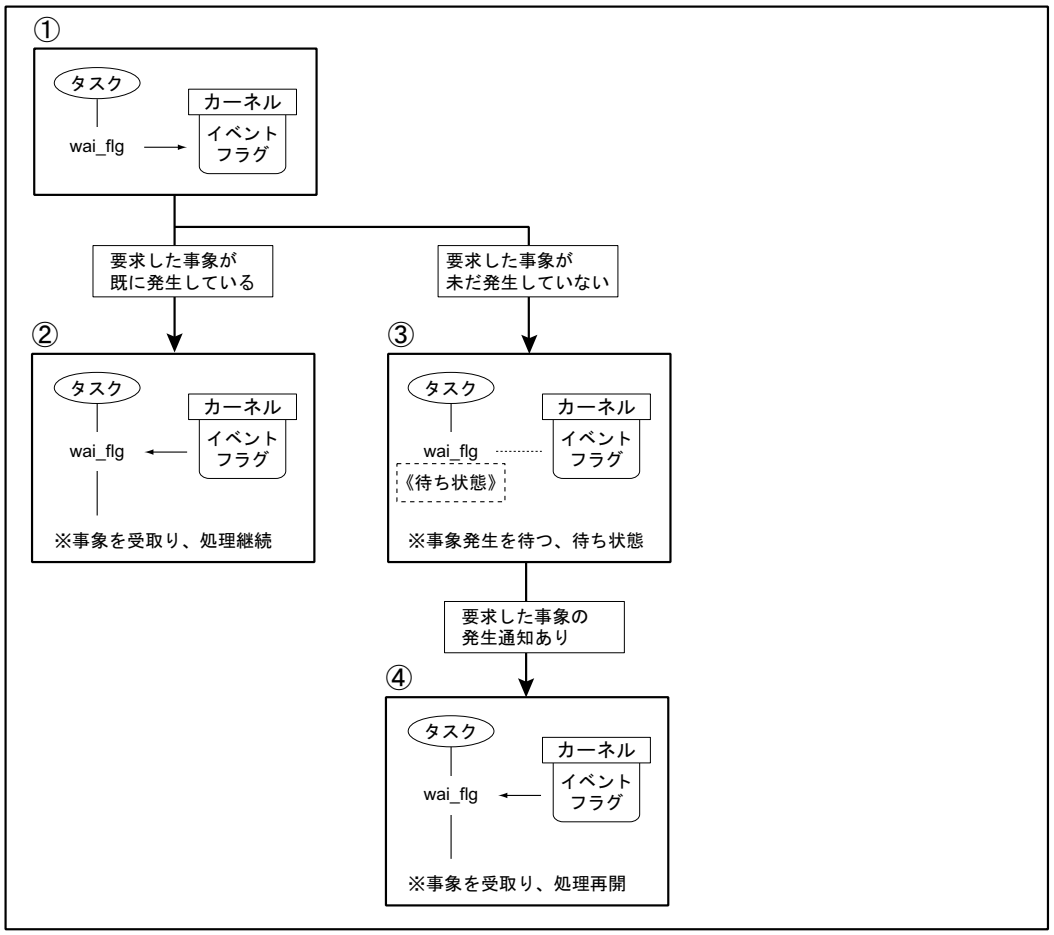


図 1-26　通常の事象発生待ちサービスコールの処理概要

① タスクがイベントフラグに wai_flg サービスコールを発行
② 指定した事象が既に発生している場合、リターンコードが正常終了（E_OK）となり、タスクの処理を継続
③ 指定した事象が発生していない場合、タスクの処理を中断し事象発生が通知されるまで待ち状態（WAITING 状態）に遷移
④ タスク（または割込みハンドラ）からの事象通知（set_flg）により、指定された事象が通知されると、リターンコードが正常終了（E_OK）となり、タスクの処理を再開

タイムアウトあり事象発生待ちサービスコール（twai_flg サービスコールを例に）の処理概要を示します。

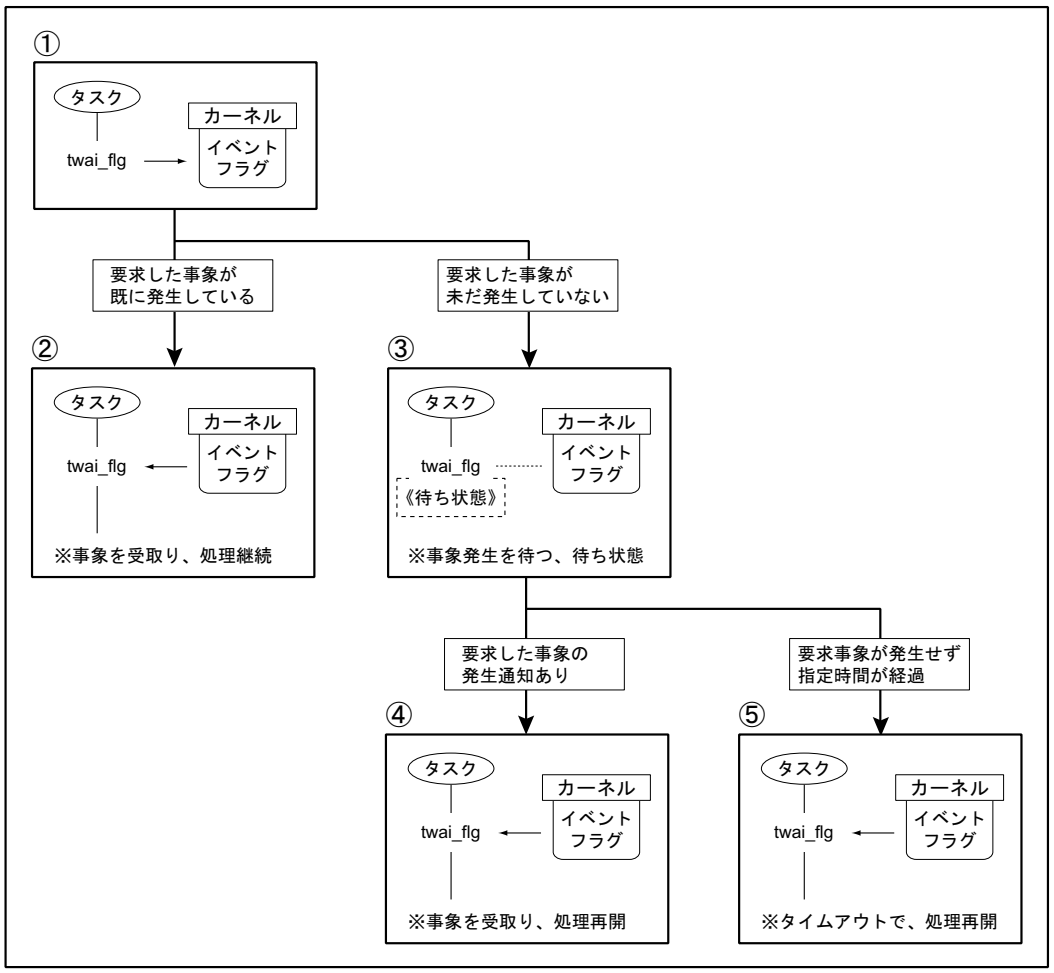


図 1-27　タイムアウトあり事象発生待ちサービスコールの処理概要

① タスクがイベントフラグに twai_flg サービスコールを発行
② 指定した事象が既に発生している場合、リターンコードが正常終了（E_OK）となり、タスクの処理を継続
③ 指定した事象が発生していない場合、タスクの処理を中断し、事象発生が通知されるまで、指定した時間だけ、待ち状態（WAITING 状態）に遷移
④ タスク（または割込みハンドラ）からの事象通知（set_flg）により、指定された事象が通知されると、リターンコードが正常終了（E_OK）となり、タスクの処理を再開
⑤ 指定した時間内に、事象の通知がなかった場合、リターンコードがタイムアウト（E_TMOUT）となり、タスクの処理を再開

ポーリング事象発生待ちサービスコール（pol_flg サービスコールを例に）の処理概要を示します。

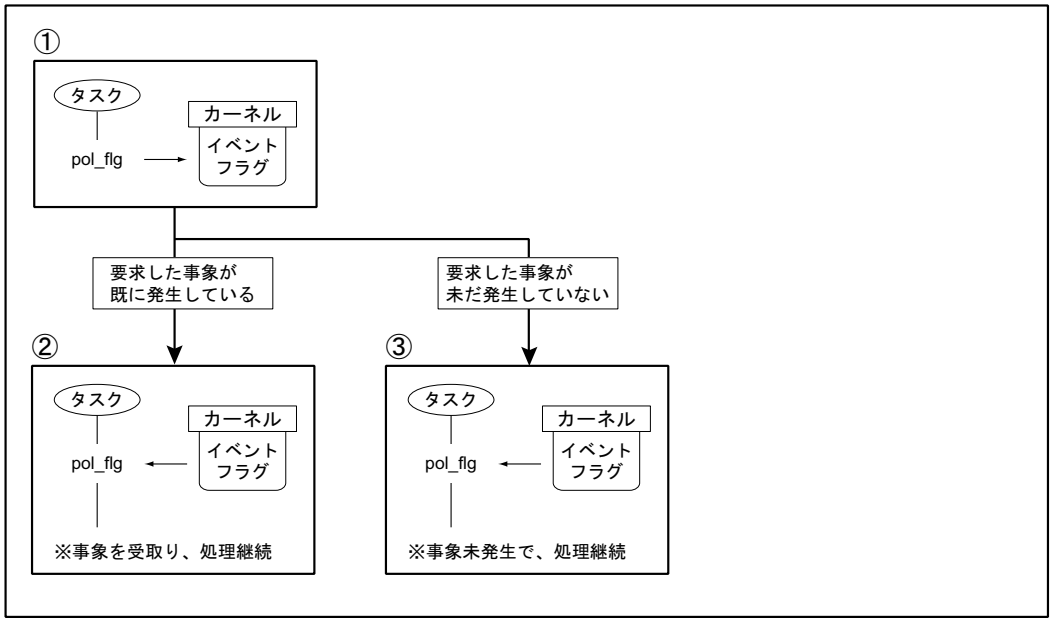


図 1-28　ポーリングによる事象発生待ちサービスコールの処理概要

① タスクがイベントフラグに pol_flg サービスコールを発行
② 指定した事象が既に発生している場合、リターンコードが正常終了（E_OK）となり、タスクの処理を継続
③ 指定した事象が発生していない場合、リターンコードがポーリング失敗（E_TMOUT）となり、タスクの処理を継続

以下に通常の事象発生待ちとタイムアウト指定あり、ポーリングの違いを示します。

表 1-12　通常の事象発生待ちとタイムアウト指定あり、ポーリングの違い

待ちサービスコール	待ち状態への遷移	待ち時間の指定
通常	あり	なし
タイムアウトあり	あり	あり
ポーリング	なし	なし

1.4.6 タスクに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたタスクに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) 初期化処理とタスクの起動

分類：タスク、タスクの起動

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

main()関数とタスクの関係について具体的に教えてください。

■回答

システム起動開始関数：main()関数の概念は、μITRON にはありません。
μITRON を使用したシステムでは、システムに登録されたタスクの優先度と、起動要求順に従い、カーネルが起動タスクを識別します。

システムの初期化処理やタスクの起動を、main()関数から行うことは可能です。
この場合、main()関数を「初期起動タスク」、または「システム初期化ルーチン」として登録し、各タスクの起動は、main()関数の処理の中で、サービスコールを発行して行います。

本アプリケーションノートの**「1.4.1 タスクと関数の関係」**もご参照ください。

(2) コンフィギュレーションファイルへのタスクの登録、起動

分類：タスク、タスクの起動

HI7000/4
HI7700/4
HI7750/4

HI1000/4

■質問

コンフィギュレータのタスク一覧で Main_Task()を生成＆起動モードで登録したのですが、実行させるには何かほかに定義が必要ですか。

■回答

コンフィギュレータにおける、タスクの登録、起動に関する定義内容は、ほかにはありません。初期起動タスクは、コンフィギュレータのタスクビューのタスク一覧で、生成（または変更）を選択後に表示されたウィンドウにて、属性に「生成後、起動（TA_ACT）」を指定して下さい。カーネル初期化処理にて該当タスクを実行可能状態にします。

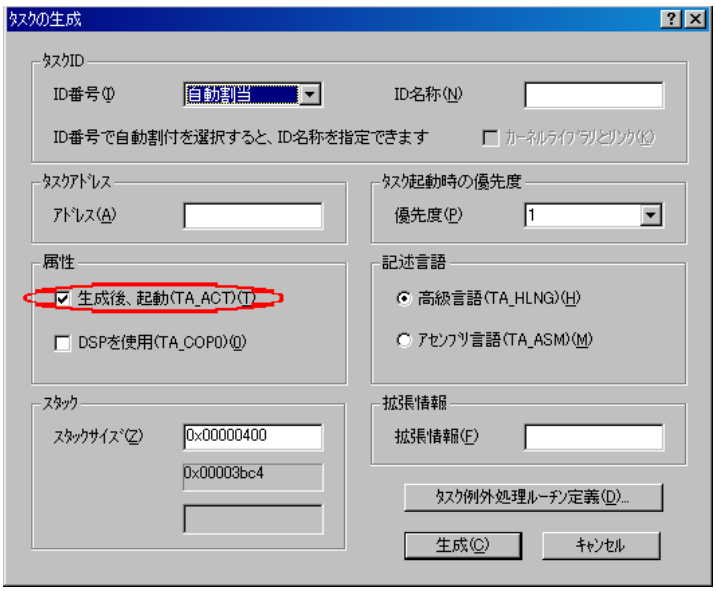


図 1-29 タスクの生成ウィンドウ

(3) タスクの起動

分類：タスク、タスクの起動

■質問

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

全タスクを「休止状態」にした場合、起動する方法はありますか。

■回答

HI7000/4
HI7700/4
HI7750/4

HI1000/4

すべてのタスクを休止状態で登録した場合、それらのタスクを起動する方法を以下に示します。

1. 初期化ルーチンを登録し、サービスコールで起動
2. 割込みハンドラ（または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）を登録し、サービスコールで起動

タスクを生成する場合は、休止状態ではなく「生成後、起動（TA_ACT）」で生成することを、一般的に推奨しています。

■回答

HI2000/3

すべてのタスクを休止状態で登録した場合、それらのタスクを起動する方法を以下に示します。

1. システム初期化ハンドラを登録し、サービスコールで起動
2. 割込みハンドラ（または周期起動ハンドラ）を登録し、サービスコールで起動

セットアップテーブルのタスク初期状態の定義を、「起動時、休止状態（DMT）」ではなく、「起動時、実行可能状態（RDY）」にすることを、一般的に推奨しています。

(4) 初期起動タスクのスタック

分類：タスクのスタック

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

初期化処理終了後に起動するタスクが使用するスタックは、どの領域を使用するのですか。

■回答

起動タスクが使用するスタックは、生成時に割り付けられたタスクスタックを使用します。
タスク生成時の情報をもとに、スタック割り付けをカーネルが行います。

また、タスクスタック領域は、以下に示すセクション領域を使用します。

- HI7000/4 シリーズ
  (1) スタティックスタック：B_histstk
  (2) ダイナミックスタック：B_hidystk
- HI2000/3
  (1) スタティックスタック：h2sstack（ダイナミックスタックは未サポート）
- HI1000/4
  (1) スタティックスタック：B_histack（ダイナミックスタックは未サポート）

スタティックスタックは、上記セクション領域から生成時に定義された領域をタスク起動時に割り付けます。
ダイナミックスタックは、上記セクション領域から、カーネルが指定されたサイズを確保し、タスク起動時に割り付けます。
スタックのセクション領域は、ユーザ任意で配置が可能です。スタックのセクション領域の配置については、以下に示す内容を参照してください。

HI シリーズ OS	参照個所
HI7000/4 シリーズ	各「構築ガイド書」の「リンケージアドレスの変更」
HI2000/3	本アプリケーションノートの「3.4.4」項
HI1000/4	本アプリケーションノートの「3.4.5」項

(5) DSP コプロセッサのタスク管理

分類：タスク

HI7000/4
HI7700/4

■質問

HI シリーズ OS で DSP ユニットを使用する際に留意する事はありますか。

■回答

タスクで DSP 機能を使用する場合、タスク生成時のパラメータ：タスク属性に「TA_COP0」を指定します。「TA_COP0」属性が指定されたタスクでは、DSP 用レジスタも汎用レジスタ同様にレジスタ情報を退避／復帰します。

コンフィギュレータで登録したタスクに「TA_COP0」属性を指定する場合は、タスク生成ウィンドウの属性枠内、「DSP を使用する(TA_COP0)(0)」のチェックボックスを定義してください。

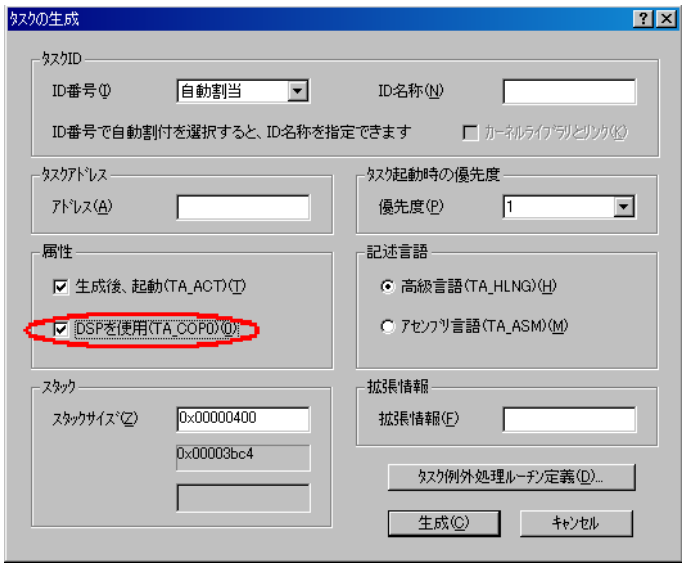


図 1-30　コンフィギュレータにおける DSP 機能選択

【次ページに続く】

【前ページからの続き】

■回答

システム動作中にサービスコールでタスクを生成する場合は、cre_tsk サービスコールのパラメータ：タスク属性に「TA_COP0」を指定します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma noregsave(MainTask)

void MainTask(VP_INT stacd)
{
  ER     ercd;
  T_CTSK pk_ctsk;

       《 処理中略 》

   pk_ctsk.tskatr =(TA_HLNG|TA_COP0)     /* ﾀｽｸ属性=高級言語記述、DSPｺﾌﾟﾛｾｯｻ使用 */
   pk_ctsk.exinf  =0;                    /* 拡張情報=0                          */
   pk_ctsk.task   =(FP)task_A;           /* ﾀｽｸ起動ｱﾄﾞﾚｽ                        */
   pk_ctsk.itskpri=1;                    /* 初期ﾀｽｸ優先度                       */
   pk_ctsk.stksz  =264;                  /* ﾀｽｸｽﾀｯｸｻｲｽﾞ                        */
   pk_ctsk.stk    =(VP)sp_taskA;         /* ﾀｽｸｽﾀｯｸ領域先頭ｱﾄﾞﾚｽ                */

   ercd=cre_tsk(TASK_A,&pk_ctsk);        /* ﾀｽｸAを生成                          */
   ercd=sta_tsk(TASK_A,(VP_INT)0x00000001); /* ﾀｽｸAを起動(起動ｺｰﾄﾞは0x1)        */

       《 処理中略 》

}
```

図 1-31　サービスコールによるタスク生成時の DSP 機能選択(コーディング例)

タスク生成時のパラメータ:タスク属性などに関する詳細については、**「HI7000/4 シリーズ　ユーザーズマニュアル」**を参照してください。

タスク以外のプログラム（割込みハンドラ、タイムイベントハンドラなど）で DSP 機能を使用する場合は、各プログラムで DSP レジスタを退避、復帰する処理が必要です。詳細については、**「HI7000/4 シリーズ　ユーザーズマニュアル」**を参照してください。

(6) FPU コプロセッサのタスク管理

分類：タスク

HI7750/4

■質問

HI シリーズ OS で FPU 機能を使用する際に留意する事はありますか。

■回答

タスクで FPU 機能を使用する場合、タスク生成時のパラメータ：タスク属性に「TA_COP1」または「TA_COP2」を指定します。以下に「TA_COP1」、「TA_COP2」の意味を示します。

表 1-13 「TA_COP1」、「TA_COP2」の意味

タスク属性	意　味
TA_COP1	FPU レジスタバンク 0 を使用する
TA_COP2	FPU レジスタバンク 1 を使用する

「TA_COP1」、「TA_COP2」属性が指定されたタスクでは、FPU 用レジスタも汎用レジスタ同様にレジスタ情報を退避／復帰します。「TA_COP1」、「TA_COP2」属性の指定について、以下に示します。

表 1-14 「TA_COP1」、「TA_COP2」の指定方法

ケース	属性指定	備　考
マトリックス演算などを行う場合	[TA_COP1｜TA_COP2]	両方の FPU レジスタバンクを使用する場合
浮動小数点演算を行う場合	[TA_COP1] *1	通常の浮動小数点演算は片方の FPU レジスタバンクのみを使用します。
浮動小数点演算を行わない場合	指定なし	

【注】 *1 TA_COP2 を指定すると、タスク、タスク例外処理ルーチンの先頭で FPSCR.FR=1 に設定する必要があるため推奨しません。

各属性の指定方法を以下に示します。

【次ページに続く】

【前ページからの続き】

■回答

コンフィギュレータで登録したタスクが FPU 機能をバンク 0 で使用する場合、以下のようにタスク生成ウィンドウの属性枠内、「FPU(バンク 0)を使用(TA_COP1)(1)」のチェックボックスを定義してください。

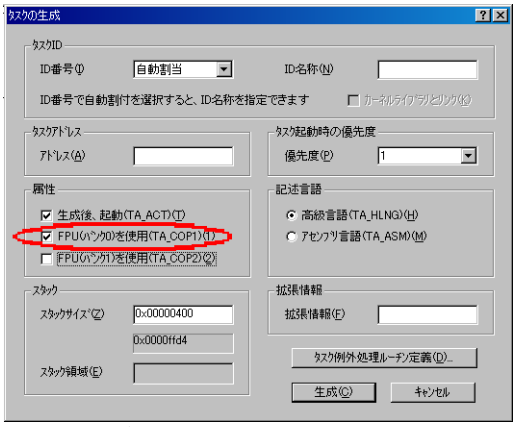


図 1-32　コンフィギュレータにおける FPU 機能選択(TA_COP1)

コンフィギュレータで登録したタスクが FPU 機能をバンク 1 で使用する場合、以下のようにタスク生成ウィンドウの属性枠内、「FPU(バンク 1)を使用(TA_COP2)(2)」のチェックボックスを定義してください。

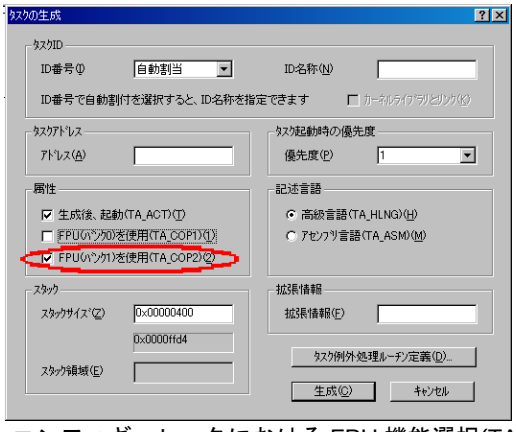


図 1-33　コンフィギュレータにおける FPU 機能選択(TA_COP2)

【次ページに続く】

【前ページからの続き】

■回答

コンフィギュレータで登録したタスクが FPU 機能をバンク 0 とバンク 1 両方で使用する場合、以下のようにタスク生成ウィンドウの属性枠内、「FPU(バンク 0)を使用(TA_COP1)(1)」と「FPU(バンク 1)を使用(TA_COP2)(2)」のチェックボックス両方を定義してください。

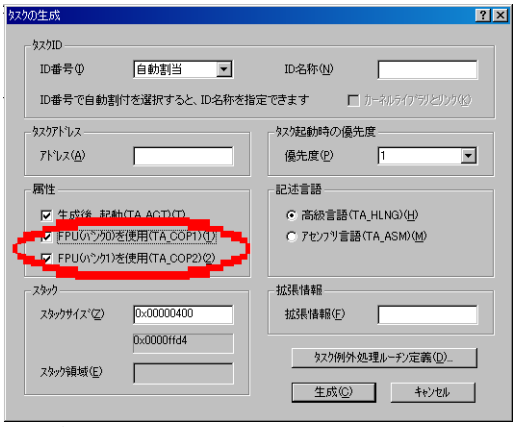


図 1-34　コンフィギュレータにおける FPU 機能選択(TA_COP1、TA_COP2)

【次ページに続く】

【前ページからの続き】

■回答

システム動作中にサービスコールでタスクを生成する場合は、cre_tsk サービスコールのパラメータ：タスク属性に「TA_COP1」、「TA_COP2」を指定します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma noregsave(MainTask)

void MainTask(VP_INT stacd)
{
  ER     ercd;
  T_CTSK pk_ctsk;

      《 処理中略 》

    pk_ctsk.tskatr =(TA_HLNG|TA_COP1)          /* ﾀｽｸ属性=高級言語記述、FPUｺﾌﾟﾛｾｯｻをﾊﾞﾝｸ0で使用        */ ←①
//  pk_ctsk.tskatr =(TA_HLNG|TA_COP2)          /* ﾀｽｸ属性=高級言語記述、FPUｺﾌﾟﾛｾｯｻをﾊﾞﾝｸ1で使用        */ ←②
//  pk_ctsk.tskatr =(TA_HLNG|TA_COP1|TA_COP2) /* ﾀｽｸ属性=高級言語記述、FPUｺﾌﾟﾛｾｯｻをﾊﾞﾝｸ0とﾊﾞﾝｸ1で使用 */ ←③
    pk_ctsk.exinf  =0;                         /* 拡張情報=0                                      */
    pk_ctsk.task   =(FP)task_A;                /* ﾀｽｸ起動ｱﾄﾞﾚｽ                                    */
    pk_ctsk.itskpri=1;                         /* 初期ﾀｽｸ優先度                                   */
    pk_ctsk.stksz  =264;                       /* ﾀｽｸｽﾀｯｸｻｲｽﾞ                                     */
    pk_ctsk.stk    =(VP)sp_taskA;              /* ﾀｽｸｽﾀｯｸ領域先頭ｱﾄﾞﾚｽ                            */

    ercd=cre_tsk(TASK_A,&pk_ctsk);             /* ﾀｽｸAを生成                                      */
    ercd=sta_tsk(TASK_A,(VP_INT)0x00000001);   /* ﾀｽｸAを起動(起動ｺｰﾄﾞは0x1)                        */

      《 処理中略 》

}
```

図 1-35　サービスコールによるタスク生成時の FPU 機能選択(コーディング例)

① FPU 機能をバンク 0 で使用する場合のタスク属性の指定
② FPU 機能をバンク 1 で使用する場合のタスク属性の指定
③ FPU 機能をバンク 0 とバンク 1 で使用する場合のタスク属性の指定

タスク生成時のパラメータ:タスク属性などに関する詳細については、**「HI7000/4 シリーズ　ユーザーズマニュアル」**を参照してください。

タスク以外のプログラム（割込みハンドラ、タイムイベントハンドラなど）で FPU 機能を使用する場合は、各プログラムで FPU レジスタを退避、復帰する処理が必要です。詳細については、**「HI7000/4 シリーズ　ユーザーズマニュアル」**を参照してください。

1.5 割込み

1.5.1 割込み発生から割込みハンドラが起動するまで

タスク実行中に割込みが発生した場合の処理概要を以下に示します。

(1) H8S ファミリマイコン、H8SX ファミリマイコンの場合

割込み発生から割込みハンドラが起動するまでの処理概要を以下に示します。

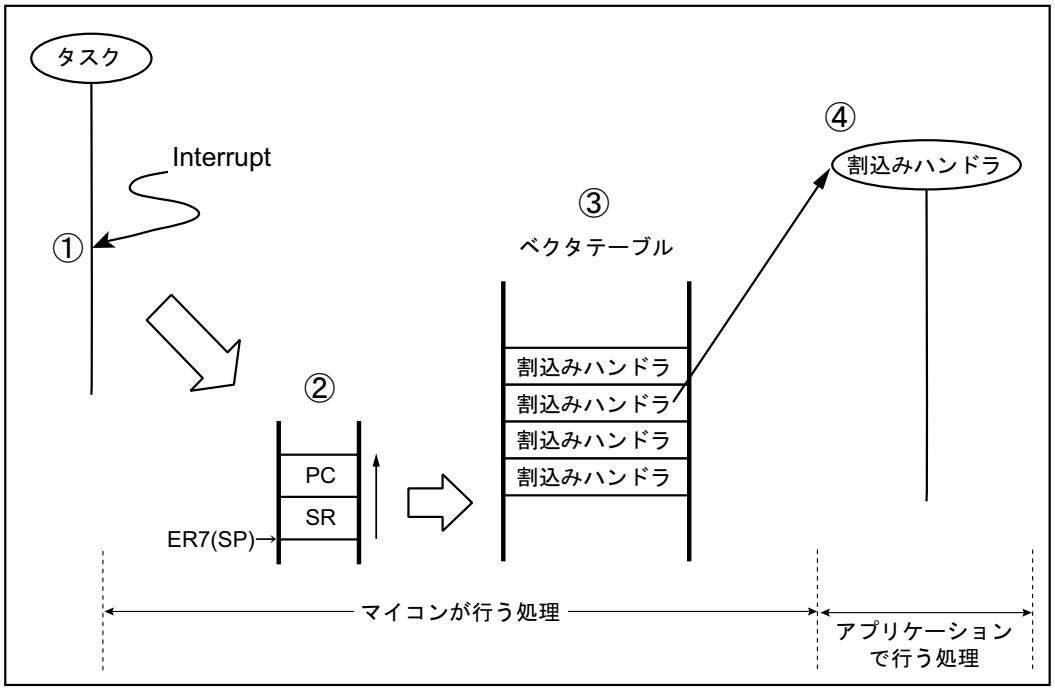


図 1-36 割込み発生から割込みハンドラ起動までの処理概要(1)

① タスク（または割込みハンドラ）実行中に発生した割込みをマイコンが認識
② マイコンは、現在のスタックに、SR レジスタ情報と PC レジスタ情報を退避
③ マイコンは、発生した割込み要因から、ベクタテーブルに登録された割込みハンドラアドレスを取得
④ ベクタテーブルに登録された割込みハンドラが起動

(2) SH-1 シリーズマイコン、SH-2&SH2-DSP シリーズマイコンの場合

割込み発生から割込みハンドラが起動するまでの処理概要を以下に示します。

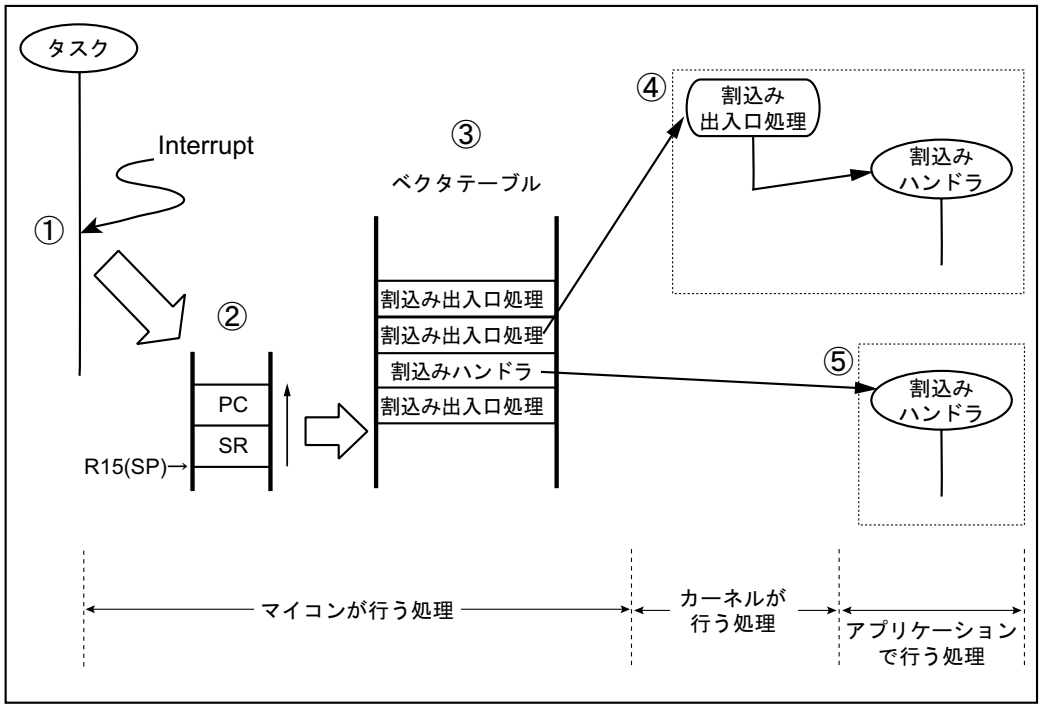


図 1-37 割込み発生から割込みハンドラ起動までの処理概要(2)

① タスク（または割込みハンドラ）実行中に発生した割込みをマイコンが認識
② マイコンは、現在のスタックに SR レジスタ情報と PC レジスタ情報を退避
③ マイコンは、発生した割込み要因から、ベクタテーブルに登録されたアドレスを取得
④ ベクタテーブルに登録されたアドレスが「割込み出入口処理」の場合、カーネルが提供する割込み出入口処理実行後、割込みハンドラが起動
※割込みサービスルーチン（カーネル）を介して起動される割込みハンドラが、通常の割込みハンドラです。
⑤ ベクタテーブルに登録されたアドレスが「割込みハンドラ」の場合、カーネルの介入なしに直接割込みハンドラが起動
※割込みサービスルーチン（カーネル）を介さずに起動される（直接起動される）割込みハンドラを「ダイレクト割込みハンドラ」とよびます。

【注】ダイレクト割込みハンドラは、HI7000/4 のみサポートされている機能です。

割込み出入口処理を「割込みサービスルーチン」と呼びます。

(3) SH-3&SH3-DSP シリーズマイコン、SH-4 シリーズマイコンの場合

割込み発生から割込みハンドラが起動するまでの処理概要を以下に示します。

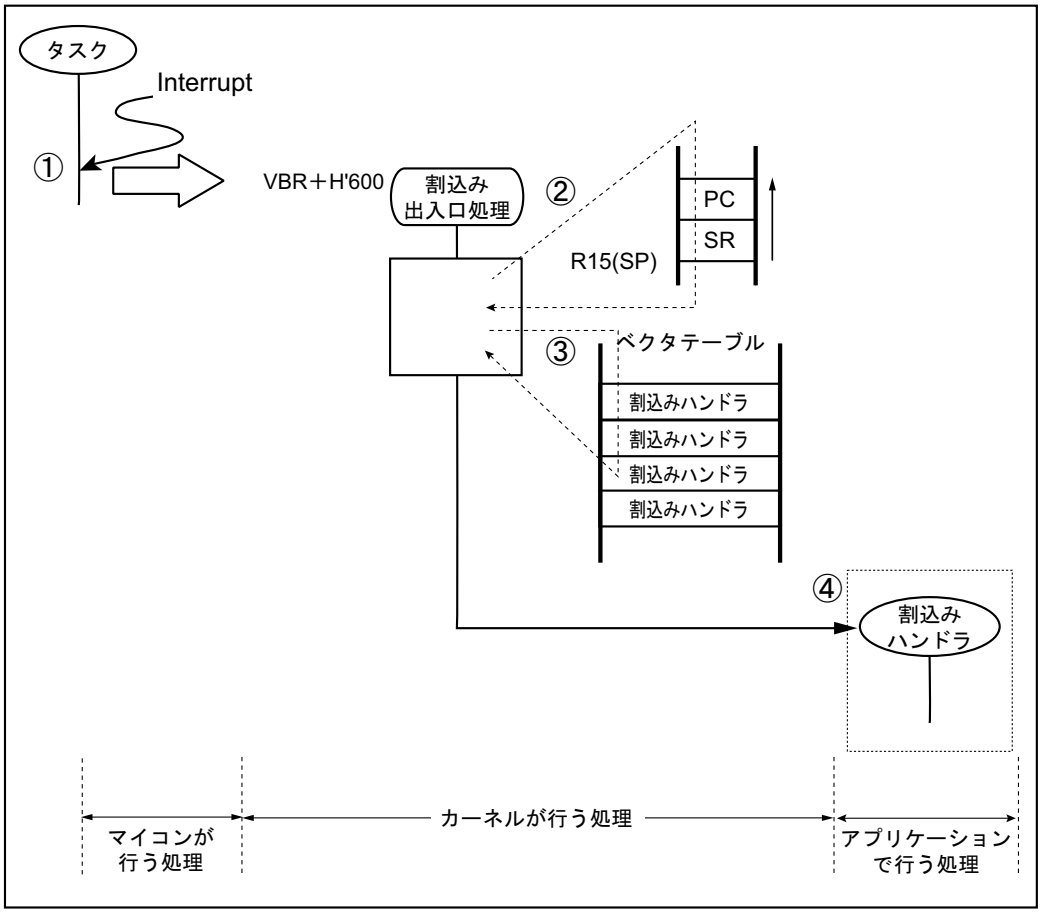


図 1-38 割込み発生から割込みハンドラが起動するまでの処理概要(3)

① タスク（または割込みハンドラ）実行中に発生した割込みを認識すると、マイコンは特定アドレス（VBR レジスタ値＋H'600）に PC を変更。
※HI シリーズ OS では、この特定アドレス（VBR レジスタ値＋H'600）に予め「割込み出入口処理（割込みサービスルーチン）」が登録してあります。

② 現在のスタックに SR レジスタ情報と PC レジスタ情報を退避

③ 割込み要因を分析し、ベクタテーブルから該当要因に登録された割込みハンドラアドレスを取得

④ 割込みハンドラが起動

1.5.2 カーネル割込みマスクレベル

カーネルでは、内部情報に矛盾が生じるのを防ぐため、割込みをマスクして実行する部分（クリティカルセクション）があります。

- クリティカルセクション実行中に発生した割込みは、クリティカルセクションが終わるまで受け付けが遅延されます。
- クリティカルセクションは、カーネル割込みマスクレベルで処理されます。

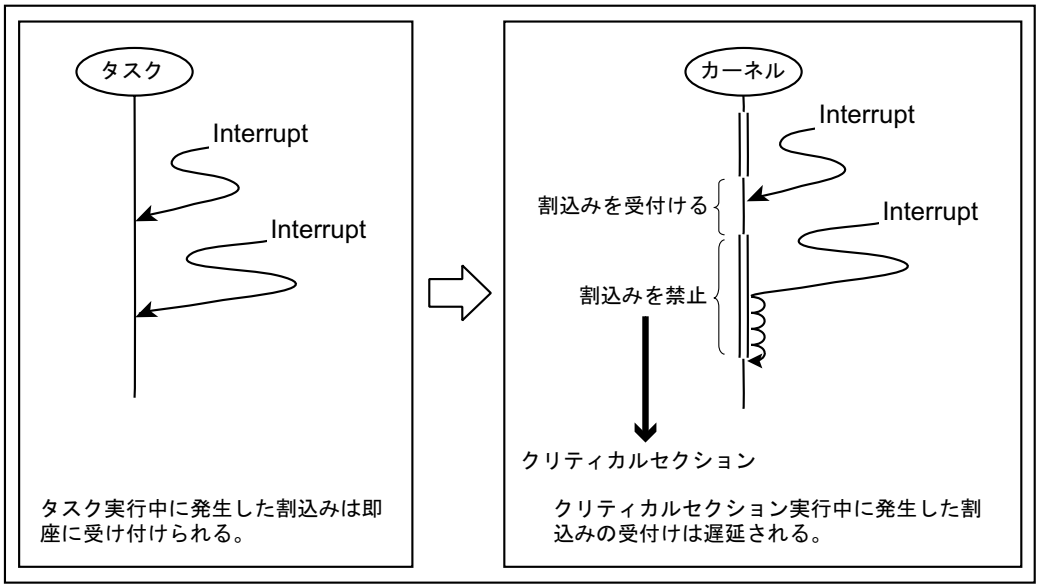


図 1-39 カーネルによる割込みマスクの概要

【注】 カーネル割込みマスクレベルより高い割込みについては、クリティカルセクション実行中でも即座に受け付けられます。

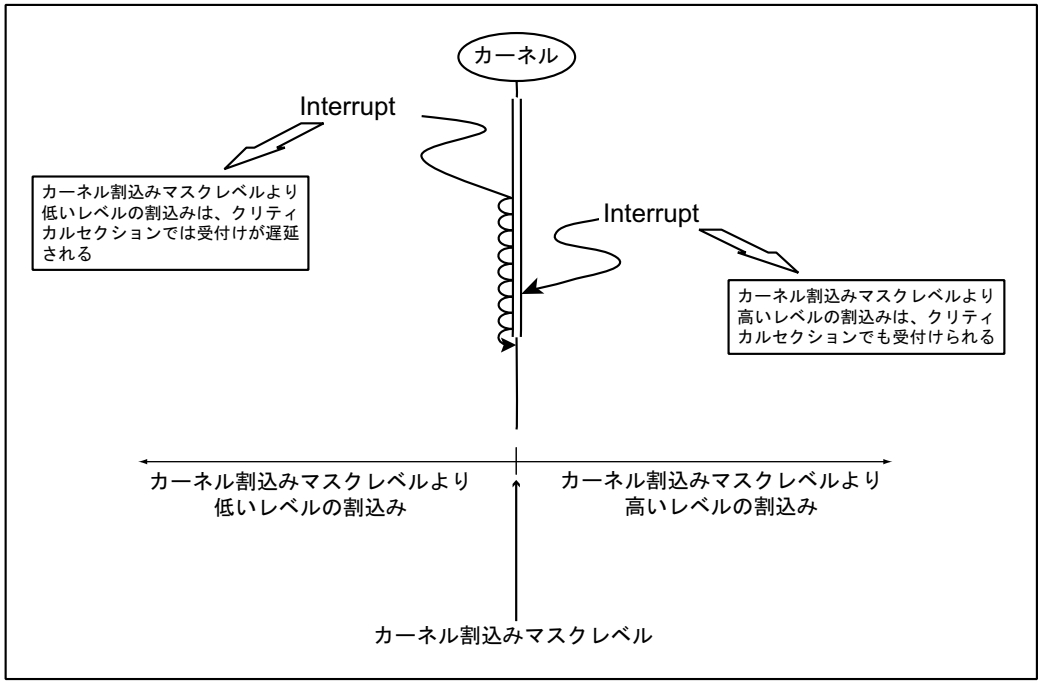


図 1-40　カーネル割込みマスクレベルと割込みレベル

《カーネル割込みマスクレベルよりも高いレベルの割込みハンドラ使用の注意事項》

- カーネル割込みマスクレベルよりも高いレベルの割込みハンドラからは、サービスコールは発行できません。発行した場合、システムの正常な動作は保証されません。
- カーネル割込みマスクレベルよりも高いレベルの割込みハンドラからの復帰は RTE 命令を実行してください。

1.5.3 H8S ファミリマイコン、および H8SX ファミリマイコン使用時の注意

H8S ファミリマイコン、および H8SX ファミリマイコン使用時は、割込み制御モードと割込みマスクレベル値との組合せによって受け付ける割込みが異なりますので注意してください。HI シリーズ OS では、H8S ファミリマイコンの持つ 4 種類、H8SX ファミリマイコンの持つ 2 種類の割込み制御モードを使用することができます。各割込み制御モードの割込みマスクレベル値の組合せと受け付ける割込みの関係を以下に示します（表中の塗りつぶし部分は Don't care を表します）。

表 1-15 割込み制御モード 0 の割込みマスクレベル

割込みマスクレベル値	CCR 値		EXR 値			受け付ける割込み
(imask)	I	UI	I2	I1	I0	
1	1					NMI のみ
0	0					すべて

表 1-16 割込み制御モード 1 の割込みマスクレベル

割込みマスクレベル値	CCR 値		EXR 値			受け付ける割込み
(imask)	I	UI	I2	I1	I0	
3	1	1				NMI のみ
2	1	0				コントロールレベル 1
1	0	1				すべて
0	0	0				すべて

表 1-17 割込み制御モード 2 の割込みマスクレベル

割込みマスクレベル値	CCR 値		EXR 値			受け付ける割込み
(imask)	I	UI	I2	I1	I0	
7			1	1	1	NMI のみ
6			1	1	0	プライオリティレベル 7
5			1	0	1	プライオリティレベル 6～7
4			1	0	0	プライオリティレベル 5～7
3			0	1	1	プライオリティレベル 4～7
2			0	1	0	プライオリティレベル 3～7
1			0	0	1	プライオリティレベル 2～7
0			0	0	0	すべて

表 1-18　割込み制御モード 3 の割込みマスクレベル

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

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

1.5.4 割込みハンドラ作成時の注意事項

以下に示す注意事項について留意してください。

表 1-19 割込みハンドラ作成時の注意事項

注意事項	詳 細
割込みハンドラの実行（処理）時間	実行時間が長すぎるとシステム全体のスループットが低下
	システムの応答性に大きな影響
割込みハンドラからのサービスコール発行 *1	カーネル割込みマスクレベルより高いレベルの割込みハンドラはサービスコール発行不可
	NMI（Non Maskable Interrupt）の割込みハンドラもサービスコール発行不可
割込みハンドラからの復帰処理 *2	カーネル割込みマスクレベル以下の割込みハンドラからの復帰は、ret_int サービスコール*3 、カーネル割込みマスクレベルより高いレベルの割込みハンドラからの復帰は、RTE 命令を使用

【注】 *1 ext_tsk（exd_tsk）サービスコールを発行した場合は、システム異常終了処理ルーチンに移行します。

*2 ret_int サービスコール以外を使用した場合、システムの正常な動作は保証されません。

*3 HI7000/4 シリーズでは ret_int サービスコールがありませんので不要です。

1.5.5 割込みに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられた割込みに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) 割込みマスクの変更

分類：割込み

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

割込みマスクレベルを変更する際、set_imask()を使用して良いのですか。

■回答

set_imask()では、OS 内部情報の処理は行いません。
したがって、set_imask()コール後、OS のサービスコールを発行すると、正常な動作を保証できません。

割込みマスク情報をもとに、システム状態を識別しています。識別している状態は、タスクコンテキスト状態と非タスクコンテキスト状態だけではなく、タスクコンテキスト状態におけるディスパッチ禁止状態や CPU ロック状態、さらに非タスクコンテキスト状態における CPU ロック状態まで管理しています。したがって、割込みマスクだけではなく、OS が管理する内部情報の処理もサービスコールで行っています。

このような理由により、割込みマスクレベルの変更についても、OS 提供のサービスコールをご使用してください。

(2) 多重割込み

分類：割込み

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

システム動作中に発生する割込み数によって何か影響されることがありますか。

■回答

「使用する割込みの本数もシステムの性能に影響します」が、厳密には割込みの本数よりも、割込みレベル（多重割込み）が影響します。

たとえば、意図する割込み関数（以後、割込みハンドラと称します）の割込みレベルより、高い割込みレベルの割込みが発生した場合、高い割込みレベルの割込みハンドラの処理を優先するため、意図する割込みハンドラの処理が中断（高い割込みレベルの割込みハンドラの処理が終了するまで）されます。割込み要因の発生順序よりも割込みレベルによって処理順序が変わるため、システムに対して影響があります。
システムで使用する割込みハンドラの割込みレベルをすべて同一にしてしまうと、緊急処理用割込みハンドラの起動が遅延（割込みの受け付けが遅延）されるケースもあります。

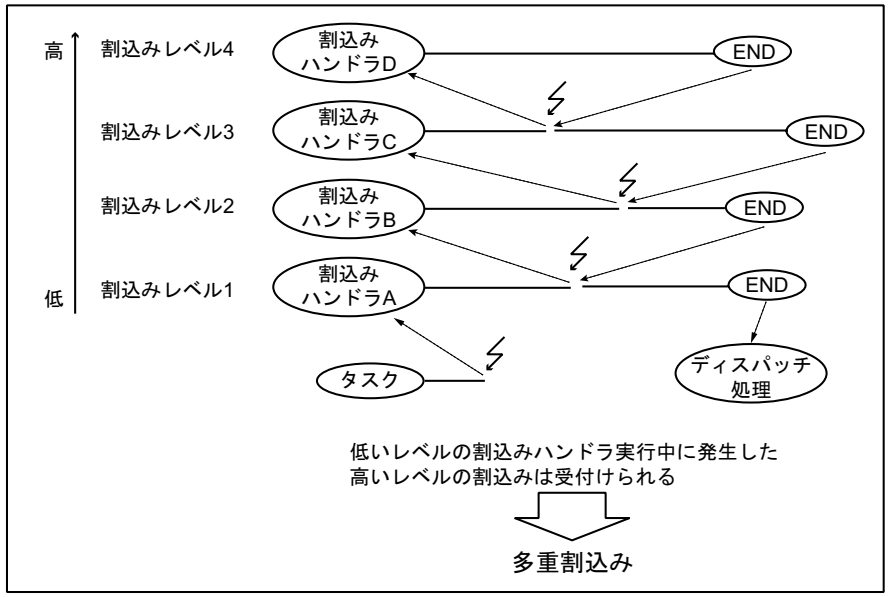
したがって、システムで使用する割込みハンドラに関する割込みレベルや処理優先度などについては、システム設計時に十分注意する必要があります。

【次ページに続く】

【前ページからの続き】

■回答

《システムで使用する割込みハンドラの割込みレベルが違う場合》



《システムで使用する割込みハンドラの割込みレベルが同一の場合》

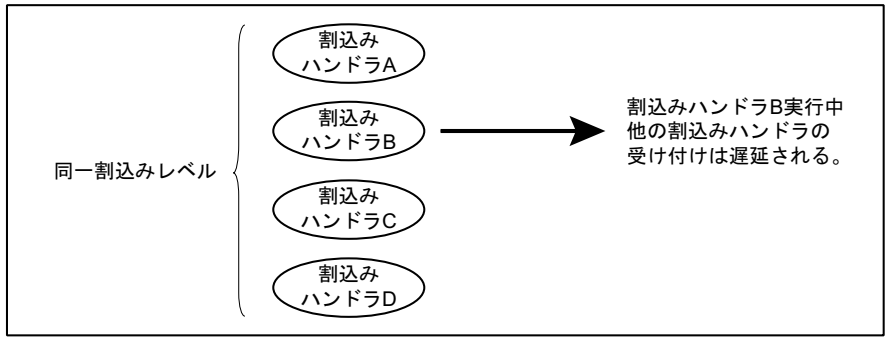


図 1-41　多重割込み

(3) 割込みハンドラ起動までの処理

分類：割込み

■質問

HI7700/4
HI7750/4

割込み発生から割込みハンドラ処理起動まで異常に時間がかかります。
割込み発生から割込みハンドラ処理実行までの処理について教えてください。

■回答

割込み発生から割込みハンドラ起動までの処理を以下に示します。

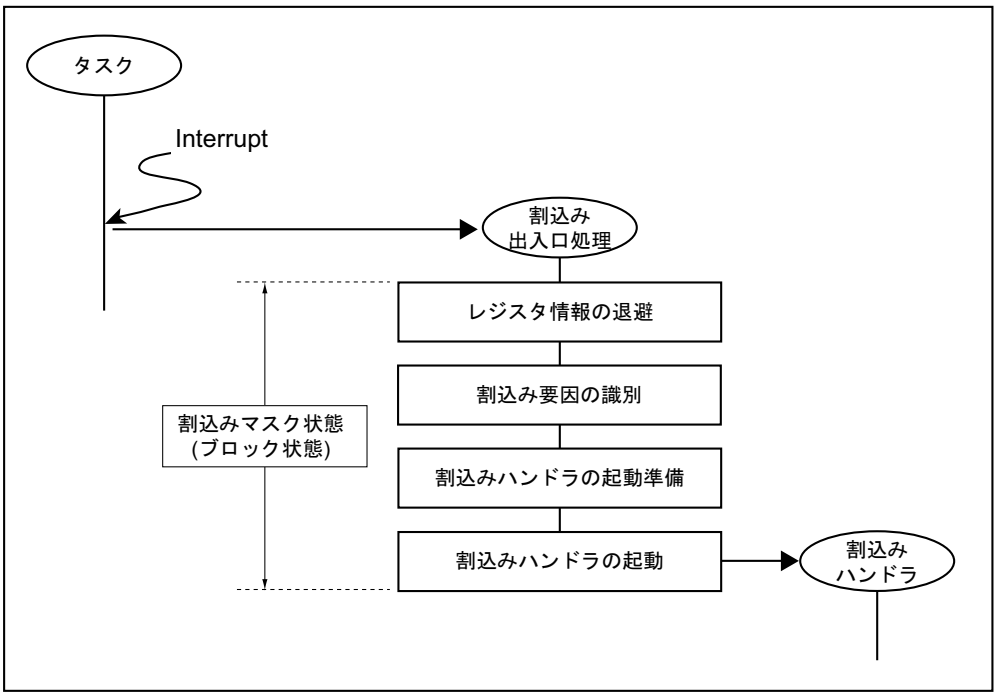


図 1-42 割込み発生から割込みハンドラ起動までの処理概要

【次ページに続く】

【前ページからの続き】

■回答

当該割込みハンドラの起動までに、時間がかかる要因を以下に示します。

- 当該割込みハンドラ起動時に、高いレベルの割込みが発生している場合
- 当該割込みハンドラ起動直前に、高いレベルの割込みが発生した場合
- 当該割込みハンドラ起動時、現在実行中の処理が、当該割込みレベルより高いレベルで割込みマスクされている場合

(4) 割込みハンドラ処理の終了

分類：割込み

■質問

HI2000/3
HI1000/4

カーネル割込みマスクレベル以下の割込みハンドラ内でサービスコールを全く発行しない場合でも、必ず「ret_int」で終了する必要がありますか。

■回答

割込みハンドラの終了は、ret_int サービスコールで行います。

ret_int サービスコールを使用する理由は、

(1) 割込みネストの識別
(2) タスクスイッチの識別

などを行い、適正な復帰処理を行うためです。

RTE 命令を使用した場合、割込み発生元に復帰しますので、割込みハンドラからのサービスコール発行によるタスクスイッチや、タイマハンドラによりタイムアウトしたタスクを識別できないため、システム状態に矛盾が発生します。
このような矛盾を排除するため、ret_int サービスコールが準備されています。

システムでタイムアウト機能を使用している場合、タイマドライバの割込みレベルより低いレベルの割込みハンドラは、サービスコール発行の有無に関わらず、すべて ret_int サービスコールで割込みハンドラの終了を行います。タイマハンドラによるタスクのタイムアウトを識別し、システムの矛盾を排除するためです。

システムでタイムアウト機能を使用しない場合、割込みハンドラからのサービスコール発行により、終了方法が異なります。
タイムアウト機能を使用しない場合で、割込みネストの有無にかかわらず、システムで発生するすべての割込みハンドラからタスクスイッチを伴うサービスコールを発行しない場合は、RTE 命令で割込みハンドラの終了を行っても構いません。

【次ページに続く】

【前ページからの続き】

■回答

タイムアウト機能を使用しない場合で、タスクスイッチを伴うサービスコールを発行する割込みハンドラが存在するが、割込みがネストしない場合は、

(1) タスクスイッチを伴うサービスコールを発行する割込みハンドラはret_int
(2) タスクスイッチを伴うサービスコールを発行しない割込みハンドラはRTE命令

で割込みハンドラの終了処理を行って下さい。
割込みがネストする場合は、

(1) タスクスイッチを伴うサービスコールを発行する割込みハンドラの割込みレベル以下の割込みハンドラは、タスクスイッチを伴うサービスコールの発行有無にかかわらず、すべてret_int（タスクスイッチの有無を識別する必要があるため）
(2) タスクスイッチを伴うサービスコールを発行する割込みハンドラの割込みレベルより高いレベルの割込みハンドラで、タスクスイッチを伴うサービスコールを発行しない割込みハンドラはRTE命令

で割込みハンドラの終了処理を行ってください。
割込みハンドラのコーディング例を以下に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

extern VP int_stk001;                                   ←①

static const VP p_stk=(VP)&int_stk001;                  ←②

#pragma interrupt(Inhhdr(sp=p_stk,sy=$ret_int))         ←③

void Inhhdr(void){                                      ←④

    /* 割込みハンドラの処理 */

}
```

図 1-43　割込みハンドラコーディング例

① 確保した割込みスタックを指定
② スタックポインタの初期値を const 型として定義
③ #pragma interrupt により、割込みハンドラを割込み関数として宣言。
- スタック切り替え指定（sp=p_stk）
- 割込み関数終了指定（sy=$ret_int）

④ 割込みハンドラは、void 型の関数として記述

(5) OS 管理外の割込みハンドラ

分類：割込み

HI7000/4
HI7700/4
HI7750/4

■質問

特定の割込みハンドラを最優先で処理するため、OS の介入なしで割り込めるようにしたいのですが、どのようにすれば良いですか。

■回答

カーネル割込みマスクレベルより高いレベルの割込みハンドラは、カーネル管理外として扱うため、特定割込みハンドラを OS の介入なしで最優先的処理したい場合に適しています。ただし、サービスコールなどを一切発行することができません。

HI7000/4 では、カーネルの介入なしに割込みハンドラが起動できる“ダイレクト割込みハンドラ”機能がサポートされています。ダイレクト割込みハンドラもカーネル管理外として扱うため、サービスコールなどは一切発行することができませんが、特定割込みハンドラを OS の介入なしで最優先的処理したい場合に適しています。

(6) ダイレクト割込みハンドラ使用時の制限事項

分類：割込み

HI7000/4

■質問

ダイレクト割込みハンドラを使用した場合の制限事項はありますか。

■回答

ダイレクト割込みハンドラ使用時は、以下に示す制限事項があります。

- ハンドラからサービスコールを発行できません。
- コンフィギュレータで定義しなければなりません。
  （動的に定義することはできません。）
- 割込みハンドラ用スタックへの切り替え処理が必要です。
- 割込みからの復帰処理は「TRAPA　#25」を使用します。

当機能の詳細については、ご使用の OS にて提供しているユーザーズマニュアルをあわせてご参照ください。

(7) サンプル定義ファイルの情報

分類：割込み

HI2000/3
HI1000/4

■質問

以下の内容でシステムを構築しています

[割込みモード：2]
ＮＭＩ：未使用
割込みレベル 7：未使用
割込みレベル 6：カーネル、周期ハンドラ（TPU0）
割込みレベル 5：タイマ割込み（TPU1、2、3、4、5）
割込みレベル 4：外部割込み（IRQ0、10、15）
割込みレベル 3：外部割込み（IRQ1、11）
割込みレベル 2：外部割込み（IRQ4、5）
割込みレベル 1：DMAC（DMTEND0A）
割込みレベル 0：未使用

このとき、最大ネストがある場合、

Q1.カーネル割込みマスクレベル以下の割込みのネスト数
Q2.カーネル割込みマスクレベルより高い割込み（NMI を含む）のネスト数
Q3.カーネル割込みマスクレベル以下で、かつタイマ割込みレベルより高い割込みのネスト数
Q4.カーネル割込みマスクレベル以下で、かつ TPU0 割込みレベルより高い割込みのネスト数
Q5.カーネル割込みマスクレベル以下で、かつ IRQ0 割込みレベルより高い割込みのネスト数

について、教えてください。

■回答

A1
「カーネル割込みマスクレベル以下の割込みのネスト数」は、カーネル割込みマスクレベルがレベル 6 なので、レベル 6 以下の割込みがネストするケースは、「タスク実行中にレベル 1 の割込みが発生」、「レベル 1 の割込みハンドラ処理中にレベル 2 の割込みハンドラが発生」とレベル 6 までネストするので、割込みネスト数は「6」になります。

A2
「カーネル割込みマスクレベルより高い割込み（NMI を含む）のネスト数」は、カーネル割込みマスクレベルより高いレベルの割込みの定義がないので「0」になります。

【次ページに続く】

【前ページからの続き】

■回答

A3

「カーネル割込みマスクレベル以下で、かつタイマ割込みレベルより高い割込みのネスト数」は、カーネル割込みマスクレベルのレベル 6 以下で、タイマ割込みレベルのレベル 5 より高いレベルに設定されている割込みハンドラは、「周期起動ハンドラ（TPU0）」が該当するので「1」になります。

A4

「カーネル割込みマスクレベル以下で､かつ TPU0 割込みレベルより高い割込みのネスト数は」は、カーネル割込みマスクレベルのレベル 6 以下で、TPU0 割込みレベルのレベル 6 より高いレベルに設定されている割込みハンドラは、登録されていないので「0」になります。

A5

「カーネル割込みマスクレベル以下で、かつ IRQ0 割込みレベルより高い割込みのネスト数」は、カーネル割込みマスクレベルのレベル 6 以下で、IRQ0 割込みレベルのレベル 4 より高いレベルに設定されている割込みハンドラは、「タイマ割込み（TPU1、2、3、4、5）」と「周期起動ハンドラ（TPU0）」が該当するので「2」になります。

(8) 割込みハンドラからのタスク切り替え

分類：割込み

■質問

割込みハンドラから irot_rdq(0)を実行しても、直ちにタスク切り替えが起きません。何か理由がありますか。

HI2000/3
HI1000/4

■回答

割込みハンドラの処理終了後、ディスパッチャが起動すると、タスク切り替えが行われます。
しかし、次に示す要因により、ディスパッチャが起動されないケースがあります。
サービスコールを発行している割込みハンドラの記述、割込み発生時のシステム状態について調べてください。

1. 割込みハンドラの記述に関する要因

HI2000/3 における割込みハンドラの記述は、クロスコンパイラのアセンブル制御命令を使用して以下のように記述します。

```
#pragma interrupt (parameter1 (sp=parameter2, sy=parameter3) )

  1.parameter1：割込みハンドラ先頭アドレス
  2.parameter2：割込みハンドラ用スタック領域のボトムアドレス
  3.parameter3：割込みハンドラ終了処理
```

図 1-44 #pragma interrupt 記述例

「parameter3」の指定は、割込みハンドラの割込みレベルとカーネル割込みマスクレベルの組み合わせにより、以下のように設定します。

(1) カーネル割込みマスクレベルより高いレベルの割込みハンドラでは，サービスコールは発行できませんのでRTE命令で終了します。したがって、「parameter3」の指定は行わない（記述しない）でください。

【次ページに続く】

【前ページからの続き】

■回答

(2) カーネル割込みマスクレベル以下の割込みハンドラでは、割込みハンドラ処理終了時、ret_intサービスコールを発行する必要があります。したがって、「parameter3」に「sy=$ret_int」と指定します。

カーネル割込みマスクレベル以下（カーネル割込みマスクレベル以下）の割込みハンドラの終了処理指定が記述されていない場合、割込み処理終了後、スケジューリングが行われません。

2. 割込みハンドラ処理終了時のシステム状態に関する要因

割込み要因が発生したときのシステム状態によって、スケジューリングが行われない場合があります。以下に、その要因を示します。

(1) 割込み要因が発生したときのシステム状態がディスパッチ禁止状態の場合、タスク切り替えを禁止している状態なので、割込みハンドラ処理終了後は、割込み発生時に実行中だったタスクの処理を継続します。

(2) 割込み要因が発生したときのシステム状態がタスク部実行中状態でも、実行中のタスクがchg_imsサービスコールを発行して、割込みマスクレベルが0以外の場合、割込みハンドラ処理終了後は、割込み発生時に実行中だったタスクの処理を継続します。

1.6 イベントフラグ

1.6.1 イベントフラグにおけるクリア指定

HI シリーズ OS では、イベントフラグのクリア指定（TA_CLR 属性）の方法が以下のように異なります。

表 1-20 イベントフラグにおけるクリア指定方法の違い

HI シリーズ OS	イベントフラグのクリア指定方法
HI2000/3	サービスコール発行時のパラメータ（第 4 引数）として、タスクごとに指定
HI7000/4 シリーズ、HI1000/4	イベントフラグ生成時、イベントフラグごとに指定

クリア指定がない場合のイベントフラグの処理概要を以下に示します。

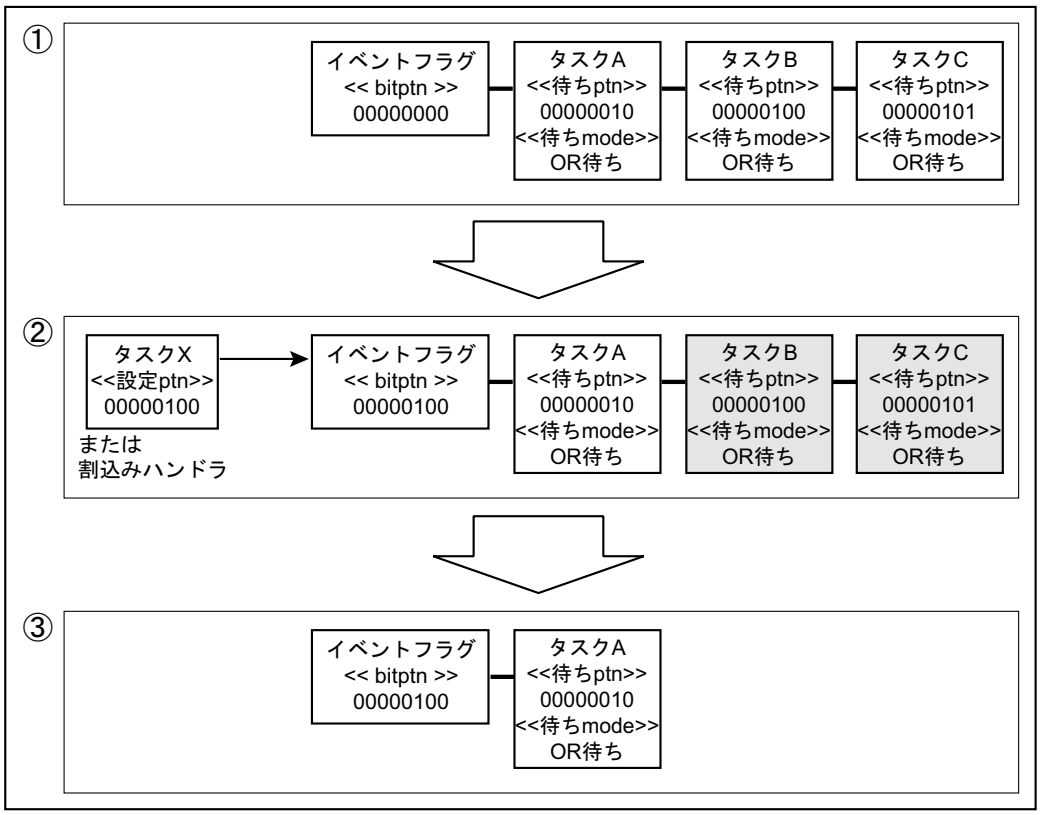


図 1-45 クリア指定なしのイベントフラグ処理概要

① イベントフラグに、タスク A、タスク B、タスク C が存在
② イベントフラグに対して、タスク X（または割込みハンドラ）が事象通知（セット）
③ イベントフラグは、条件が成立したタスク（タスク B とタスク C）の待ち状態を解除

(1) HI2000/3 における TA_CLR 属性の指定方法

HI2000/3 におけるクリア指定がある場合のイベントフラグの処理概要を以下に示します。

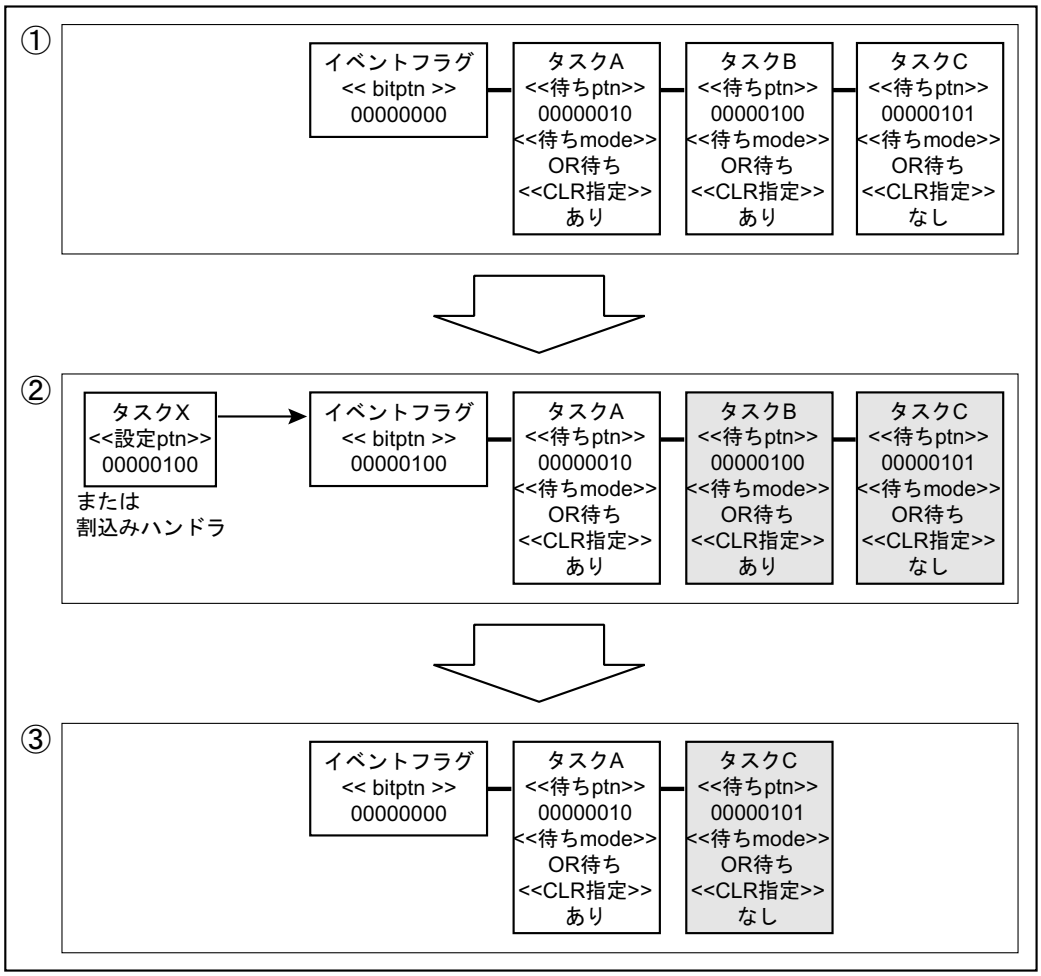


図 1-46 クリア指定ありのイベントフラグ処理概要(HI2000/3)

① イベントフラグに、タスク A、タスク B、タスク C が存在
② イベントフラグに対して、タスク X（または割込みハンドラ）が事象通知（セット）
③ イベントフラグは、条件が成立したタスク（タスク B）の待ち状態と同時にビットパターンをクリア

TA_CLR 属性の指定がある場合、待ち状態のタスク 1 つを待ち解除した時点で、イベントフラグのビットパターンがクリアされるため、それ以降待ち解除処理は行いません。待ち解除時のビットパターンには、クリアされる前のビットパターン情報が渡されます。

(2) HI7000/4 シリーズ、および HI1000/4 における TA_CLR 属性の指定方法

HI7000/4 シリーズ、および HI1000/4 におけるクリア指定がある場合のイベントフラグの処理概要を以下に示します。

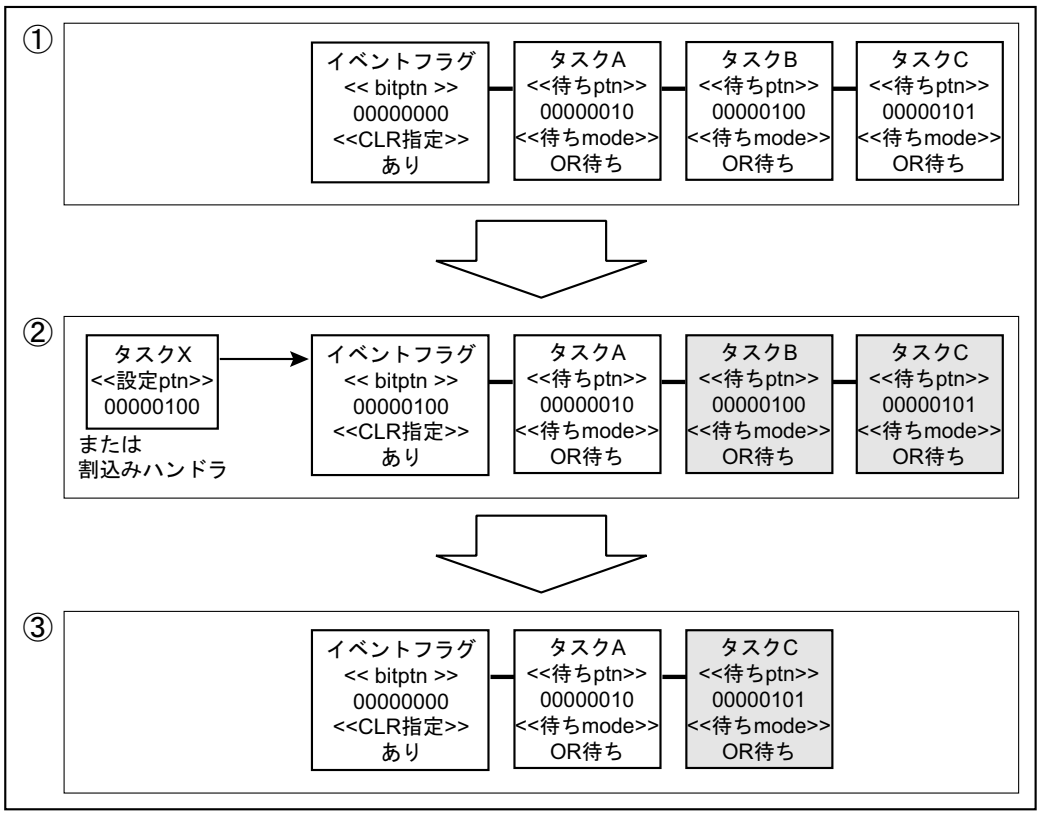


図 1-47 クリア指定がありのイベントフラグ処理概要(HI7000/4 シリーズ、HI1000/4)

① イベントフラグに、タスクＡ、タスクＢ、タスクＣが存在
② イベントフラグに対して、タスクＸ（または割込みハンドラ）が事象通知（セット）
③ イベントフラグは、条件が成立したタスク（タスク B）の待ち状態と同時にビットパターンをクリア

HI2000/3 との違いは、イベントフラグに CLR 指定を定義する事です。

1.6.2 イベントフラグに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたイベントフラグに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) イベントフラグ情報のクリア

分類：イベントフラグ

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

同じフラグパターンで待つ複数のタスクをレディにし、フラグをクリアする方法がありますか。

■回答

同じフラグパターンで待つ複数タスクの待ち状態を解除した後、フラグパターンをクリアする方法はあります。以下に、タスクで行う場合と、割込みハンドラで行う場合に分けて示します。

1. フラグセット処理がタスクの場合

イベントフラグ待ち状態の各タスクの優先度より、セットするタスクの優先度を高くすることで、サービスコールの終了後、セットしたタスクに戻ってきます。したがって、セット処理後、クリア処理を行うことで実現できます。コーディング例を以下に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma noregsave(EventFlag_Set_Task)

void EventFlag_Set_Task(VP_INT exinf)
{

  ER ercd;

      《 途中処理省略 》
    ercd = set_flg((ID)flgid, (FLGPTN)setptn);       /* イベントフラグのセット */
    if(ercd != E_OK){
        /* エラー時の処理 */
    }else{
      ercd = clr_flg((ID)flgid, (FLGPTN)clrptn);  /* ビットパターンをクリアする */
      if(ercd != E_OK){
            /* エラー時の処理 */
       }
    }
       《 途中処理省略 》
}
```

図 1-48　セット処理がタスク時のコーディング例

【次ページに続く】

【前ページからの続き】

■回答

2. フラグセット処理が割込みハンドラの場合

割込みハンドラの処理は、タスク、ディスパッチャより優先して行われるため、イベントフラグのセット処理とクリア処理を連続して行うことで実現できます。コーディング例を以下に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void EventFlag_Set_Interrupt(void)
{

ER ercd;

       《 途中処理省略 》
    ercd = iset_flg((ID)flgid, (FLGPTN)setptn);      /* イベントフラグのセット */
    if(ercd != E_OK){
        /* エラー時の処理 */
    }else{
        ercd = iclr_flg((ID)flgid, (FLGPTN)clrptn); /* ビットパターンのクリア */
        if(ercd != E_OK){
          /* エラー時の処理 */
        }
    }
       《 途中処理省略 》
}
```

図 1-49　セット処理が割込みハンドラ時のコーディング例

1.7 セマフォ

1.7.1 セマフォによるタスクのデッドロック

排他制御が必要な資源（資源としては、実際のハードウェア資源だけでなく、共有メモリや非リエントラント関数などソフトウェア資源も対象となります）の管理に使用します。

以下にセマフォの使用例を示します。

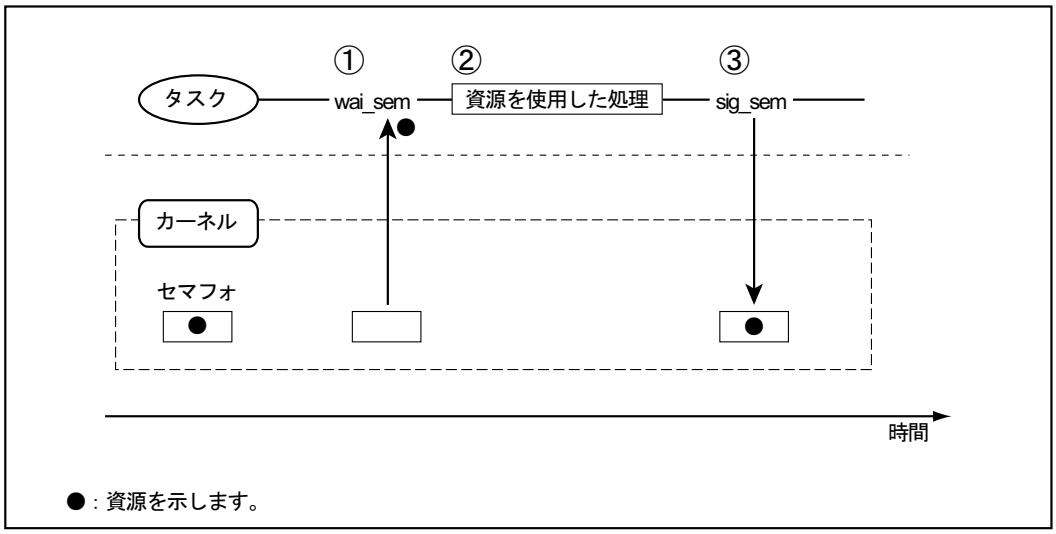


図 1-50　セマフォ使用例

① タスクがセマフォを獲得
② 獲得した資源を使用した処理を実行
③ 処理が終了したら、獲得した資源を解放

排他制御が必要な資源を使用する場合、セマフォを獲得してから資源を使用した処理を行います。処理終了後はセマフォを解放します。

カーネルがタスク終了時、獲得しているセマフォを、自動的解放する機能はありませんので、タスク終了時、獲得しているセマフォは、タスクで解放するする必要があります。

以下にタスクが動けない状態（デッドロック）になる例を示します。

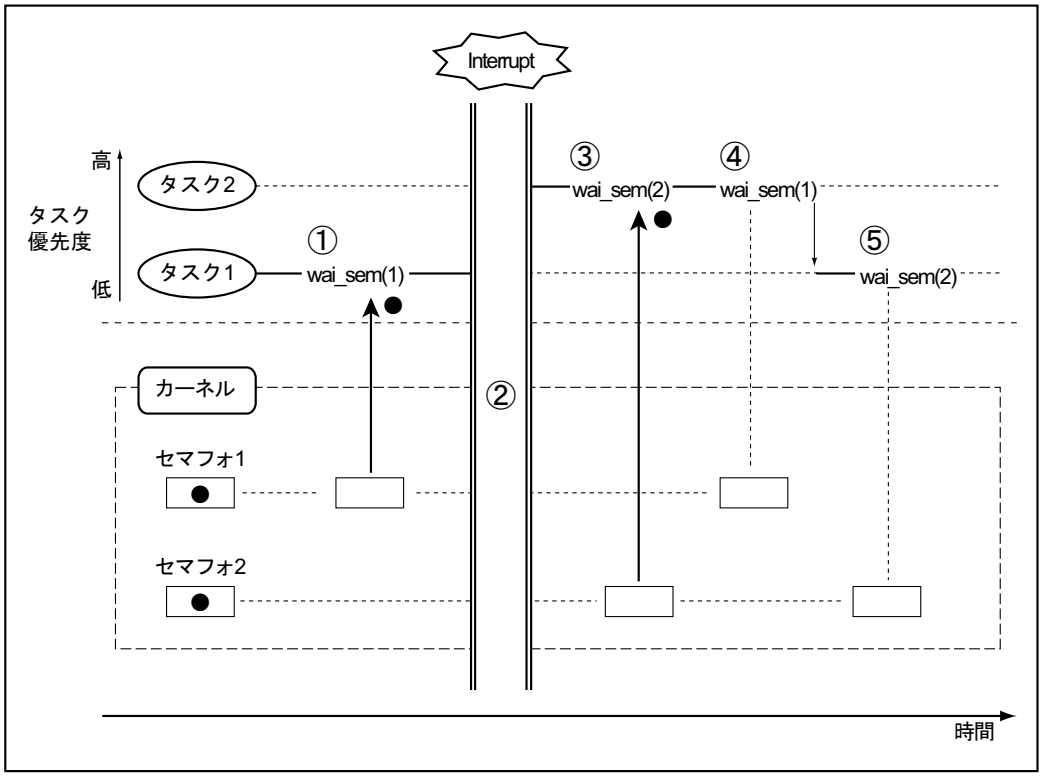


図 1-51　タスクが動けない状態(デッドロック)の例

① タスク 1 がセマフォ 1 を獲得
② 割込みが発生し、割込みハンドラ処理により、タスク切替えが発生（タスク 1→タスク 2）
③ タスク 2 がセマフォ 2 を獲得
④ タスク 2 が、既にタスク 1 が獲得済みのセマフォ 1 を獲得しようとしますが、獲得できる資源（セマフォ）がないため、資源解放を待つ WAITING 状態に遷移したため、タスク切替えが発生（タスク 2→タスク 1）
⑤ タスク 1 が、既にタスク 2 が獲得済みのセマフォ 2 を獲得しようとしますが、獲得できる資源（セマフォ）がないため、資源解放を待つ WAITING 状態に遷移

以上により、タスク 1，2 は、お互いに相手が獲得したセマフォの解放を待つことになり、永久に待ち状態から解除されないことになります。
このような状態を、タスクが動かない状態（デットロック）といいます

このような状態は、OS では回避できません。
アプリケーション側（ユーザシステム側）で、システム設計時に十分検討してください。

1.8 ミューテックス

1.8.1 優先度逆転現象

優先度逆転現象について以下に示します。

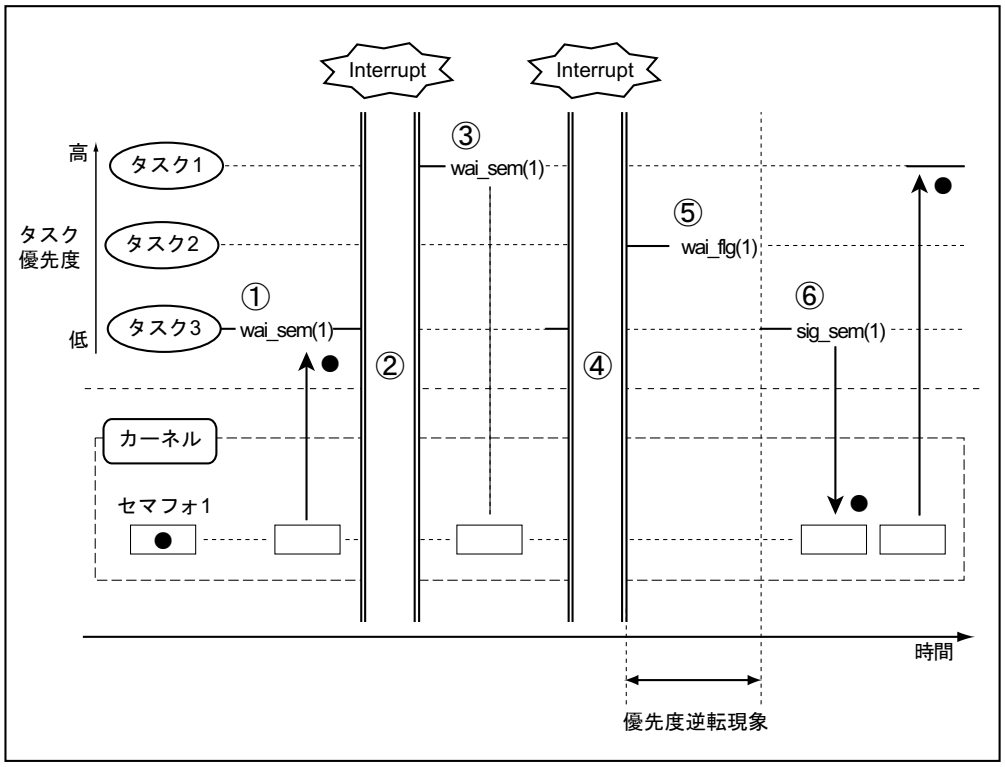


図 1-52 優先度逆転現象の概要

① タスク 3 がセマフォ 1 を獲得し、処理を継続
② 割込みが発生し、割込みハンドラ処理により、タスク切替えが発生（タスク 3→タスク 1）
③ タスク 1 が、既にタスク 3 が獲得済みのセマフォ 1 を獲得しようとしますが、獲得できる資源（セマフォ）がないため、資源解放を待つ WAITING 状態に遷移するため、タスク切替えが発生（タスク 1→タスク 3）
④ 割込みが発生し、割込みハンドラ処理により、タスク切替えが発生（タスク 3→タスク 2）
⑤ タスク 2 が、事象待ち要求発行により、タスク切替えが発生（タスク 2→タスク 3）
⑥ タスク 3 が、資源を使用した処理が終了したため、セマフォ 1 を解放
同時に資源解放を待つ WAITING 状態のタスク 1 がセマフォ 1 を獲得し処理を再開

優先度が高いタスク 1 を、タスク 2 より優先して実行したいが、タスク 1 が使用したい資源（セマフォ）を優先度の低いタスク 3 によって確保されているために、タスク 2 より優先して実行出来なくなっています。このような、優先度の高いタスクの処理が、優先度の低いタスクの処理によって待たされる現象を、優先度逆転現象と言います。

1.8.2 ミューテックスの処理概要

ミューテックスの処理概要を以下に示します。

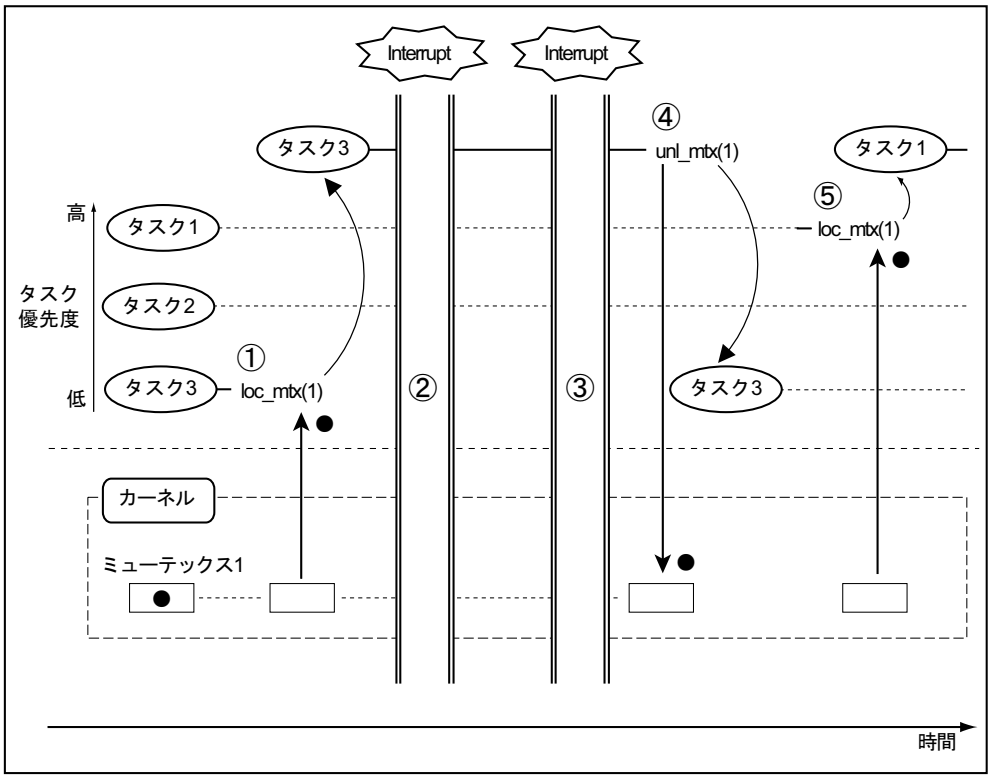


図 1-53 ミューテックスの処理概要

① タスク 3 がミューテックス 1 を獲得し、処理を継続
（この時、タスク 3 の優先度が 3→1(上限優先度)へ上がる）
② 割込みが発生し、割込みハンドラ処理により、タスク 1 が起床するが、タスク 3 の優先度が高いため、タスク切替えは発生しない（タスク 3 が優先度 1 のまま処理を継続）
③ 割込みが発生し、割込みハンドラ処理により、タスク 2 が起床するが、タスク 3 の優先度が高いため、タスク切替えは発生しない（タスク 3 が優先度 1 のまま処理を継続）
④ タスク 3 が、資源を使用した処理が終了したため、ミューテックス 1 を解放
（この時、タスク 3 の優先度が 1→3 に戻り、タスク 1 にタスク切替えが発生）
⑤ タスク 1 がミューテックス 1 を獲得し、処理を継続
（この時、タスク 1 の優先度が 1→1(ceilpri)へ上がる）

ミューテックスを獲得（ミューテックスをロックすると言う）したタスクは、自動的にミューテックスに設定された上限優先度まで引き上げられて実行されるため、タスク 1 やタスク 2 が起動されても待ち状態にならず実行を続けます。

タスク 3 がミューテックスを解放（ミューテックスのロック解除と言う）した時、元の優先度に戻るため、タスク 1、タスク 2 の順で実行されます。

1.9 メールボックス

1.9.1 メールボックスの処理概要

メールボックス使用時のメリット、デメリットを以下に示します。

表 1-21 メールボックス使用時のメリット、デメリット

メリット	・メッセージ格納アドレスの受け渡しのため、オーバヘッドが少ない ・メッセージはリンクリストで管理されるため、メッセージ送信数量に制限なし ・大容量メッセージの送信が可能
デメリット	・共有メモリ（あるいは共通のアドレス空間）を持つ必要がある

以下に処理概要を示します。

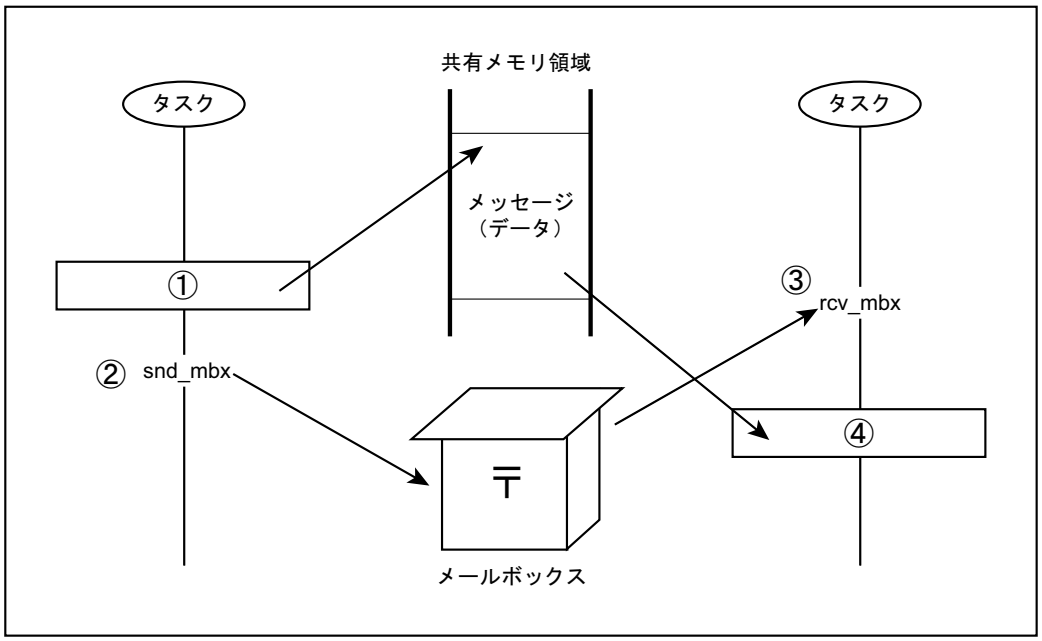


図 1-54 メールボックスの処理概要

① メッセージを格納するメモリを確保して、そのエリアに送信すべき情報を書き込む
② snd_mbx サービスコールで、メールボックスに送信
③ rcv_mbx サービスコールで、メールボックスから受信
④ 受信したエリアに格納されている情報を読み出す

1.9.2 メールボックスのメッセージ送信処理概要

メールボックスにおけるメッセージ送信処理の概要を以下に示します。

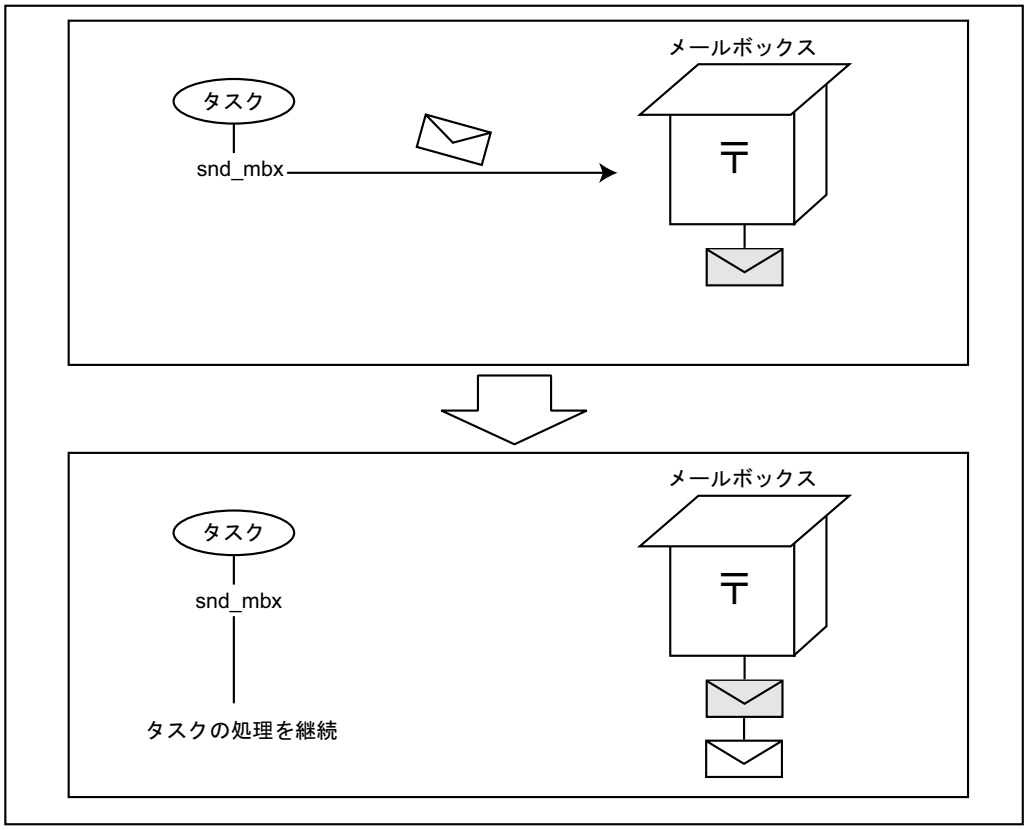


図 1-55 メールボックスのメッセージ送信処理概要

リンクリストを管理するため、メッセージの先頭には、リンクリストの管理に使用するカーネル管理領域を確保する必要があります。この領域をメッセージヘッダと呼びます。

メッセージ管理方式として、FIFO 順（First In First Out：受付順）とメッセージ優先度順の二つの方式があります。したがって、送信するメッセージのメッセージヘッダは、メールボックスにおけるメッセージ管理方式で違います。

以下にメッセージ管理方式の違いによるメッセージヘッダを示します。

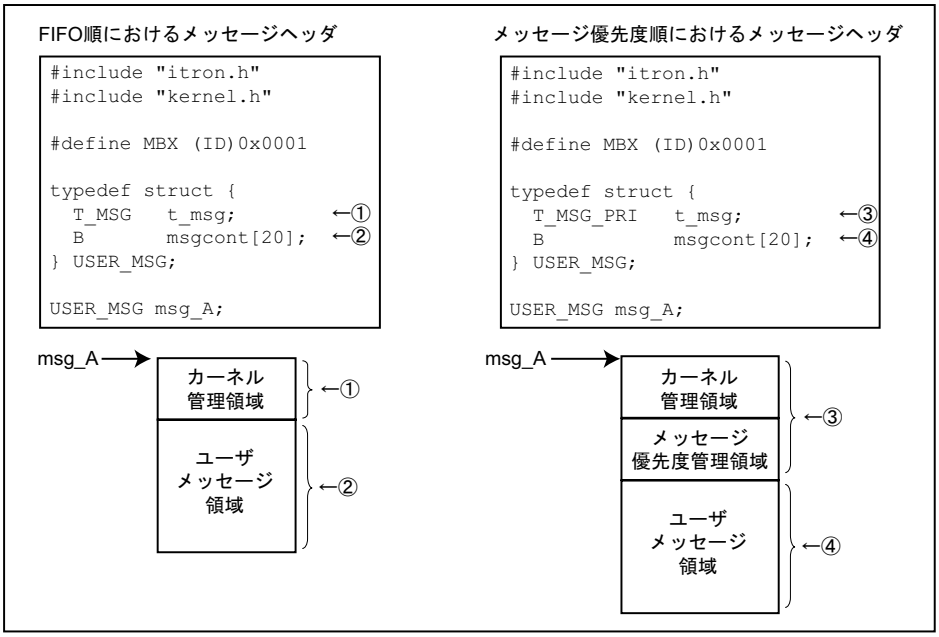


図 1-56　メッセージのメッセージヘッダ構造

HI シリーズ OS は、メッセージヘッダを識別することはできません。以下の留意事項に充分ご注意ください。

表 1-22　メールボックスの属性とメッセージヘッダの組合せ

項番	メッセージ管理方式	メッセージヘッダ	
		FIFO 順	メッセージ優先度順
1	FIFO 順	正常に処理される	処理に影響なし。但し、使用するメモリ領域に無駄が発生
2	メッセージ優先度順	ユーザメッセージ領域先頭 4 バイトを優先度として処理*1	正常に処理される

【注】 *1　ユーザメッセージ領域先頭 4 バイトの情報により、送信できない(エラーが発生する)場合があります

あわせて、次の内容にも注意してください。

《メッセージデータ送信時の注意事項》

(1) メッセージデータ送信後は、カーネル管理領域を書き換えないでください。
(2) メッセージデータの最初の送信時、カーネル管理領域を 0 にして送信してください。コーディング例を次に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma noregsave(Task)

typedef struct user_msg{                                    ←①
    T_MSG t_msg;
    B     data[10];
} USER_MSG;

// typedef struct user_primsg{                              ←②
//     T_MSG_PRI t_pri_msg;
//     B         data[10];
// } USER_PRIMSG;

void Task(VP_INT exinf)
{

ER ercd;
USER_MSG *message;

       《途中省略》

    ercd = get_mpf((ID)mpfid, (VP)message);                 ←③
    if(ercd != E_OK){
        /* エラー処理 */
    }

    /* ユーザメッセージの格納処理 */

    message->t_msg.msghead = 0;                             ←④
//  message->t_pri_msg.msghead = 0;                         ←⑤

    ercd = snd_mbx((ID)mbxid, (T_MSG *)message);            ←⑥
    if(ercd != E_OK){
        /* エラー処理 */
    }

       《途中省略》

    ext_tsk();
}
```

図 1-57　メッセージ送信時のコーディング例

① ユーザメッセージの宣言(メッセージヘッダ：FIFO 順用)
② ユーザメッセージの宣言(メッセージヘッダ：メッセージ優先度順用)
③ メッセージ用メモリ領域の獲得
④ メッセージ内カーネル管理領域の 0 クリア(FIFO 順用使用時)
⑤ メッセージ内カーネル管理領域の 0 クリア(メッセージ優先度順用使用時)
⑥ メッセージの送信

1.9.3 メールボックスのメッセージ受信処理概要

メールボックスにおけるデータ受信処理の概要を以下に示します。

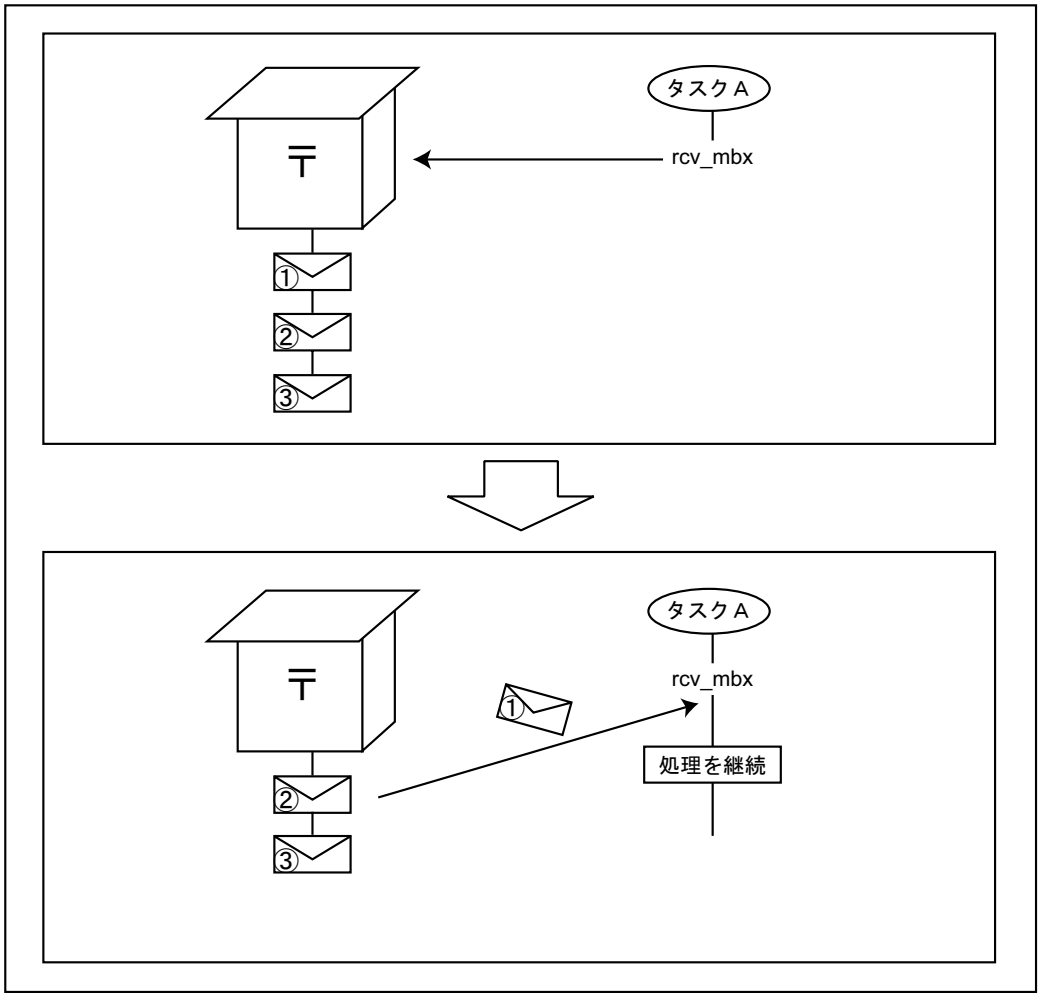


図 1-58 メールボックスにメッセージがある場合の受信処理概要

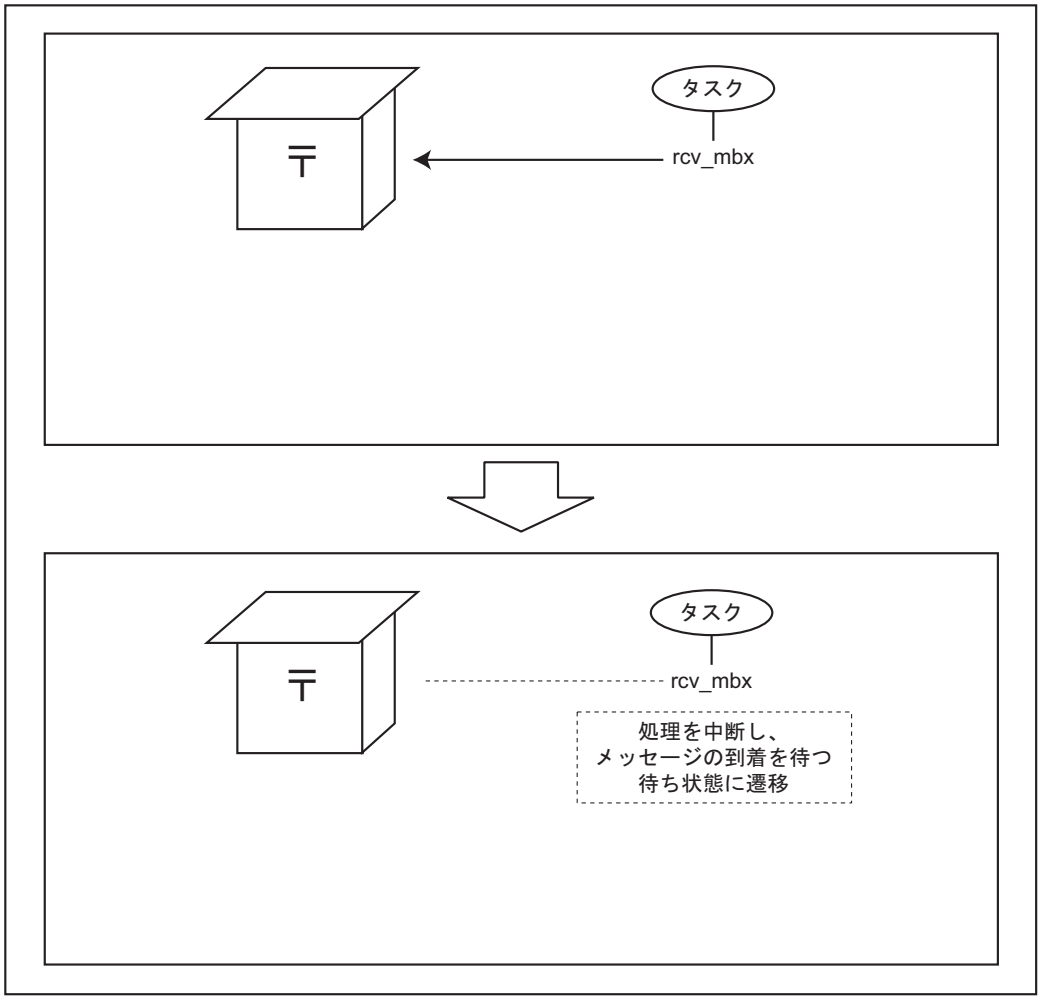


図 1-59　メールボックスにメッセージがない場合の受信処理概要

1.9.4 メールボックスに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたメールボックスに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) メールボックスへの連続送信

分類：メールボックス

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

メールボックスに対し、連続して同一メッセージを送信できますか。

■回答

同一メッセージを連続して送信する事はできません。送信したメッセージが受信されていない場合、メールボックスのメッセージ管理（リンクリスト）情報が破壊されます。

送信したメッセージを相手側が受信した事を確認してから、再度送信する事は可能です。以下に確認方法の例を示します。

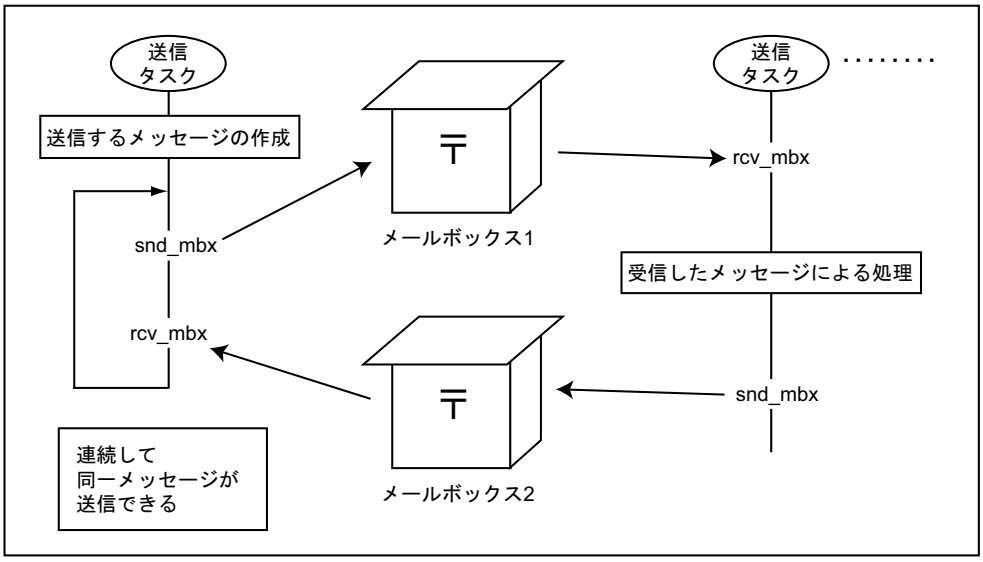


図 1-60　メッセージ受信確認方法の例

1.10 メッセージバッファ

1.10.1 メッセージバッファの処理概要

メッセージバッファ使用時のメリット、デメリットを以下に示します。

表 1-23 メッセージバッファ使用時のメリット、デメリット

メリット	・共有メモリ（あるいは共通のアドレス空間）不要
デメリット	・メッセージがコピーされるため、オーバヘッドが大きい

以下に処理概要を示します。

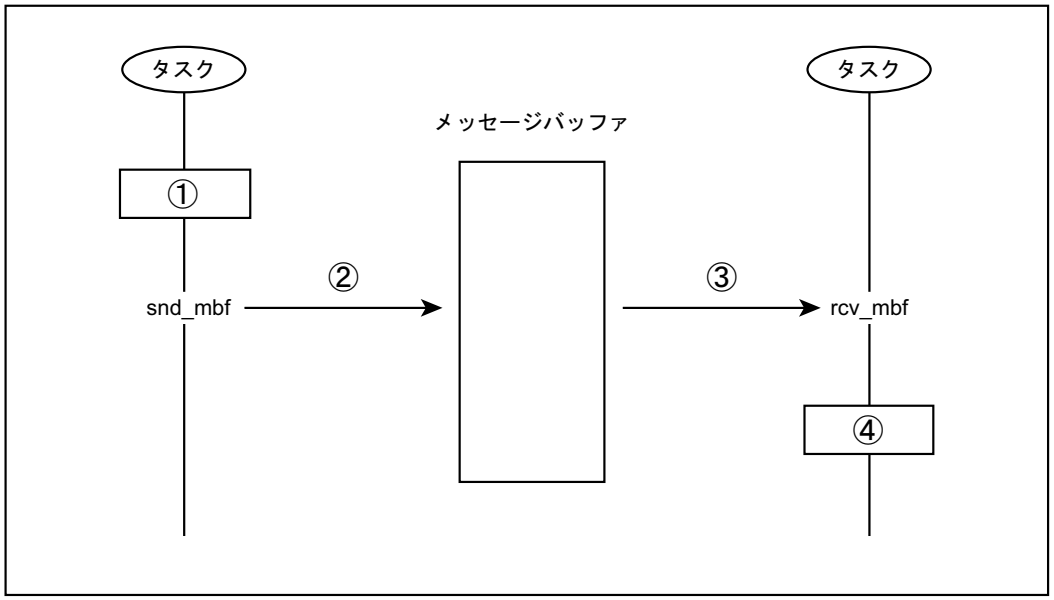


図 1-61 メッセージバッファの処理概要

① メッセージを格納するメモリを確保して、そのエリアに送信すべき情報を書き込む
② snd_mbf サービスコールで、メッセージバッファに送信
③ rcv_mbf サービスコールで、メッセージバッファから受信
④ 受信した情報を読み出す

HI シリーズ OS では、バッファサイズ＝0 でメッセージバッファを生成する事ができます。以下に概要を示します。

- バッファサイズ＝0 で生成されたメッセージバッファでは、メッセージを蓄えておく事ができないため、送信側と受信側が、完全に同期した動作になります。
- 送信タスクから受信タスクへのメッセージコピーが 1 回で済むため、メッセージバッファを介したコピー動作を伴わないというメリットもあります。

1.10.2 メッセージバッファのメッセージ送信処理概要

メッセージバッファは、メッセージバッファの空き領域の状態によって、以下に示すような処理を行います。

表 1-24 メッセージバッファ空き領域と送信処理

メッセージバッファの空き領域あり	メッセージバッファの空き領域なし
メッセージバッファに、送信メッセージを格納して、送信タスクの処理を継続	送信メッセージを格納できる領域が、メッセージバッファに確保できるまで、送信タスクを待ち状態《メッセージ送信待ち状態》に遷移

メッセージバッファに空き領域がある場合の送信処理概要を以下に示します。

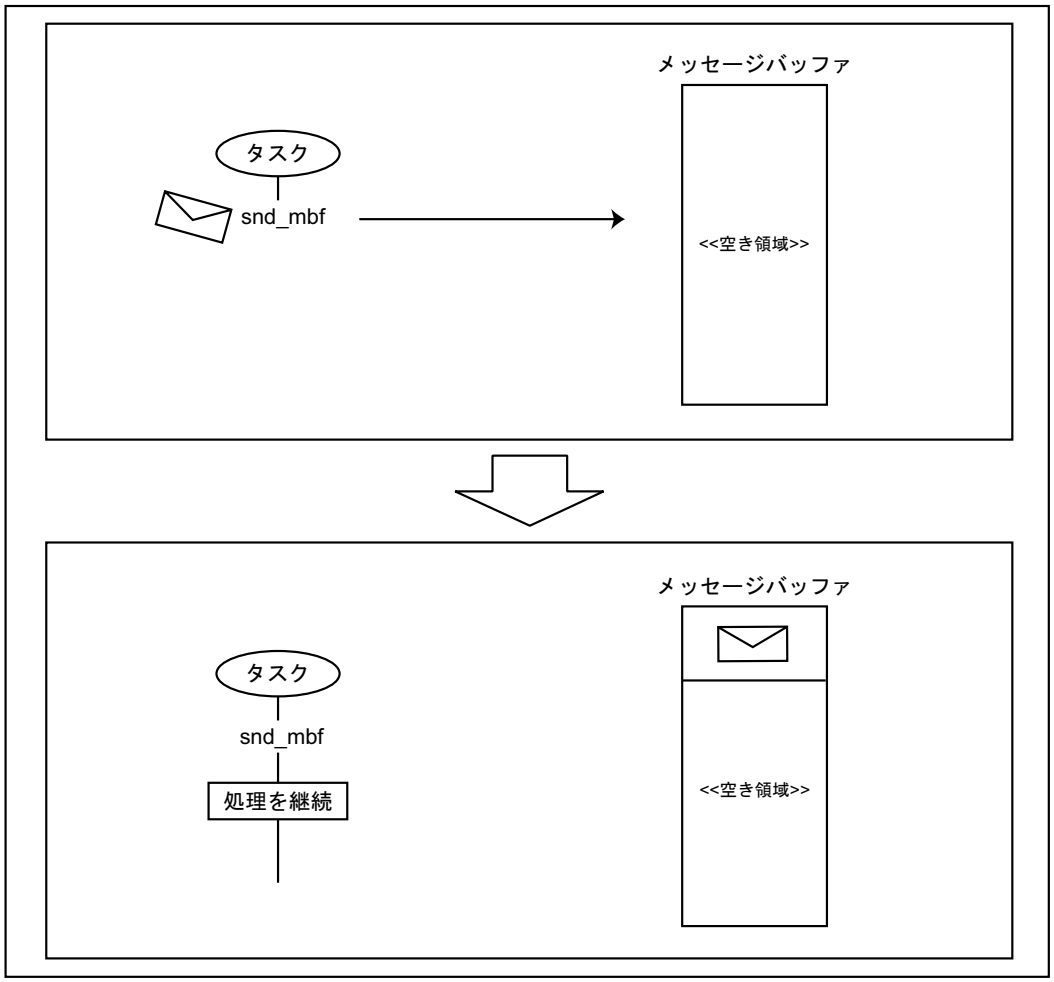


図 1-62 メッセージバッファに空き領域がある場合の送信処理概要

メッセージバッファに空き領域がない場合の送信処理概要を以下に示します。

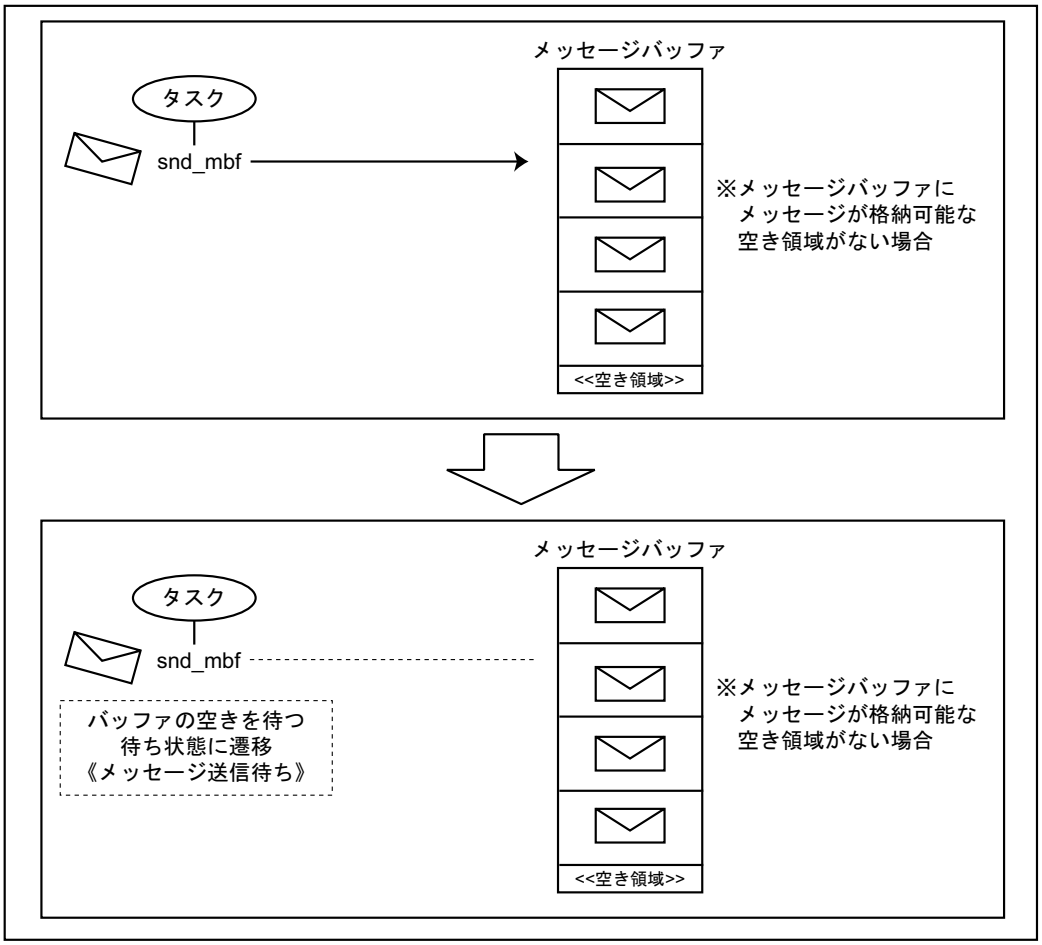


図 1-63　メッセージバッファに空き領域がない場合の送信処理概要

1.10.3 メッセージバッファのメッセージ受信処理概要

メッセージバッファにおけるデータ受信処理の概要を以下に示します。

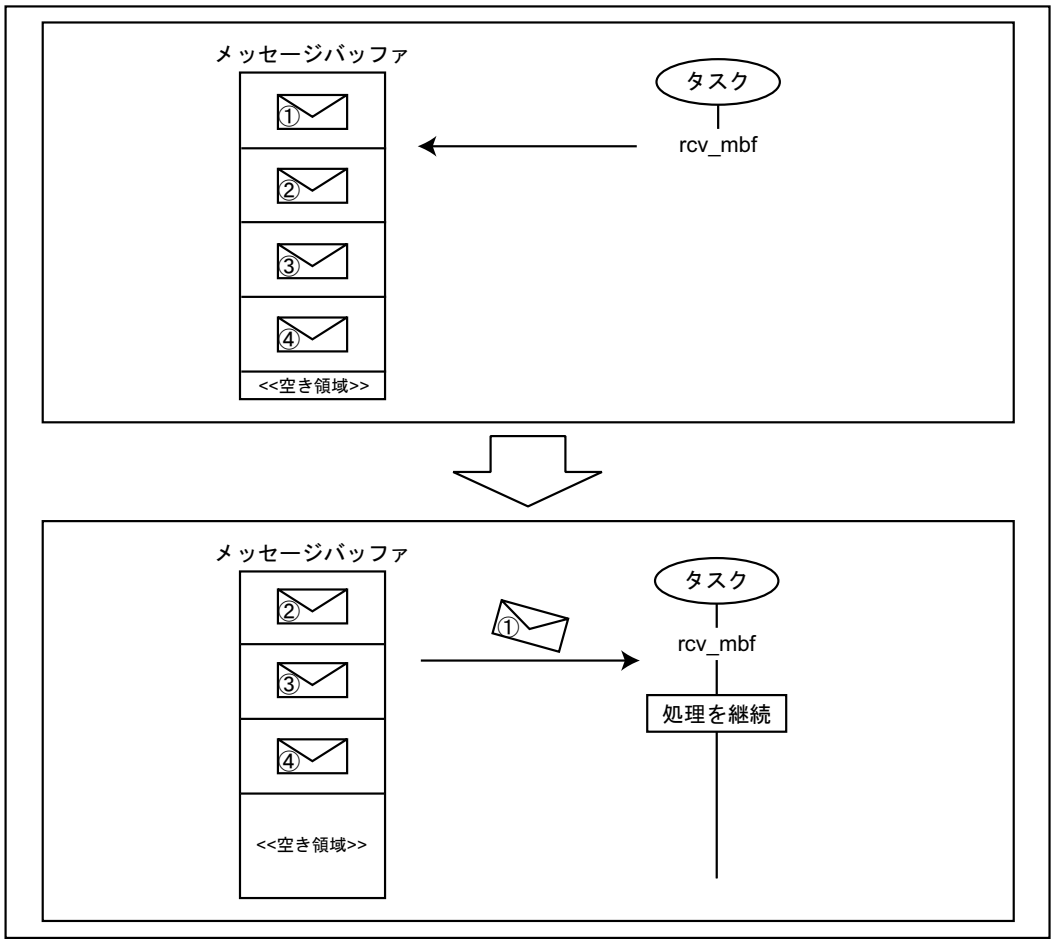


図 1-64 メッセージバッファにメッセージがある場合の受信処理概要

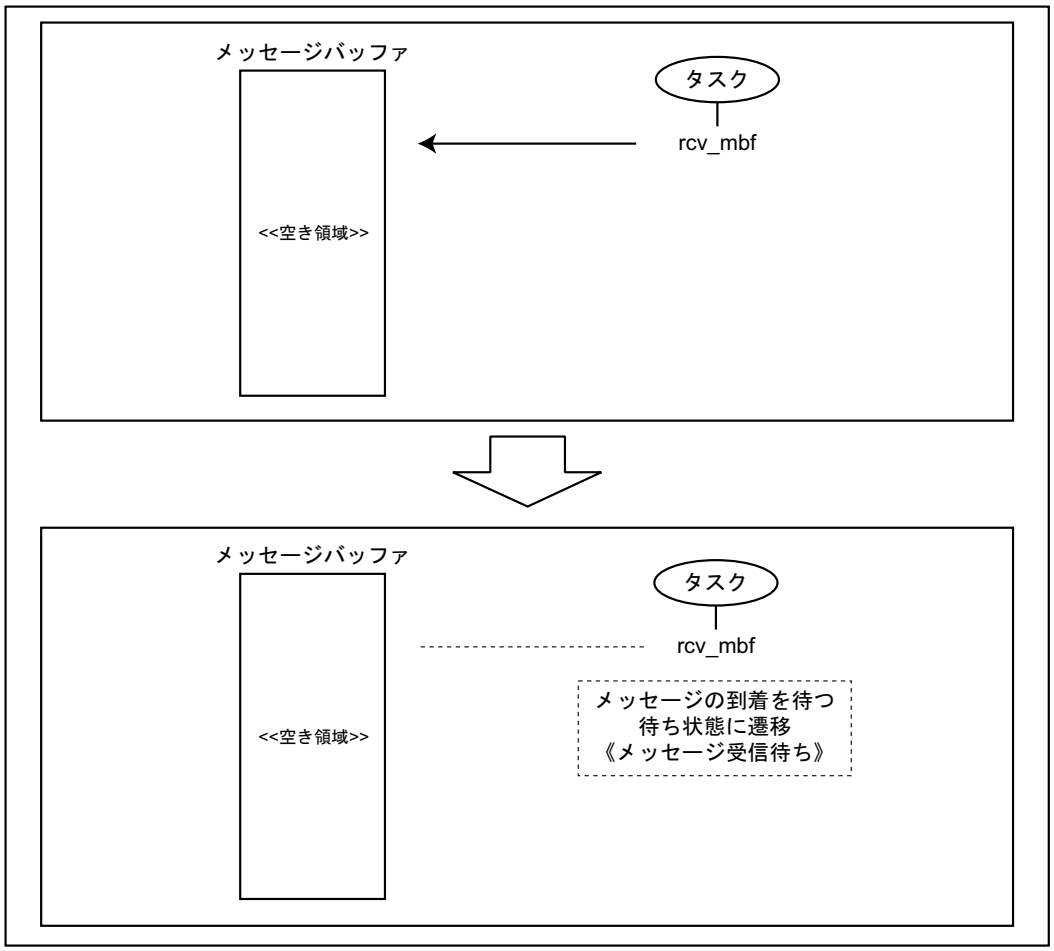


図 1-65　メッセージバッファにメッセージがない場合の受信処理概要

1.11 データキュー

1.11.1 データキューの処理概要

データキュー使用時のメリット、デメリットを以下に示します。

表 1-25 データキュー使用時のメリット、デメリット

メリット	・共有メモリ（あるいは共通のアドレス空間）不要 ・メッセージがコピーされるが、メッセージサイズが 4 バイト固定なのでオーバヘッドは小さい
デメリット	・データサイズ固定のため、大容量メッセージの送信が不可

以下に処理概要を示します。

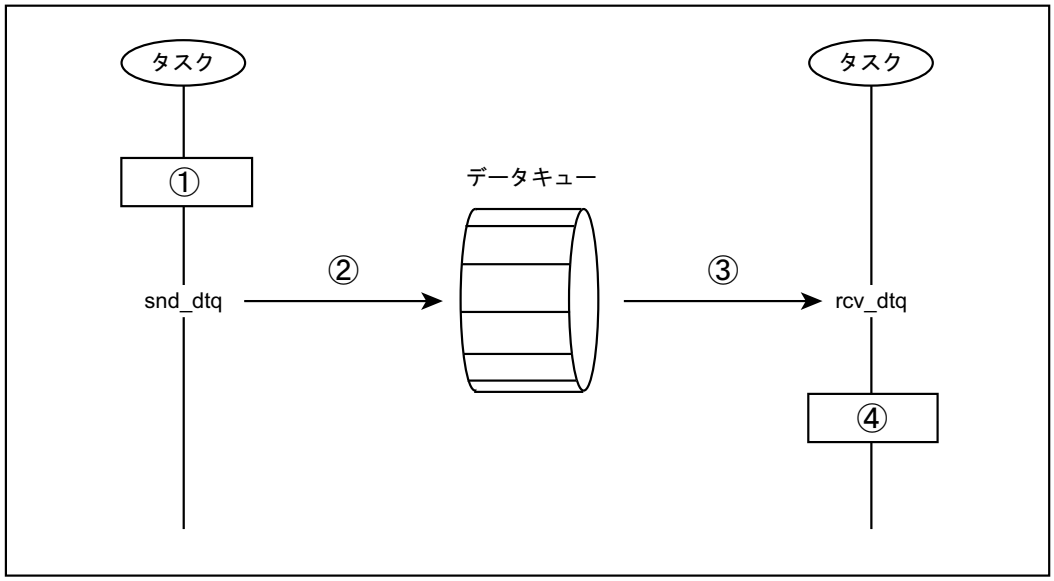


図 1-66 データキューの処理概要

① メッセージを格納するメモリを確保して、そのエリアに送信すべき情報を書き込む
② snd_dtq サービスコールで、データキューにメッセージ送信
③ rcv_dtq サービスコールで、データキューからメッセージ受信
④ 受信した情報を読み出す

1.11.2 データキューのメッセージ送信処理概要

データキューは、データキューの空き状態によって、以下に示すような処理を行います。

表 1-26 データキュー空き領域と送信処理

データキューに空き領域あり	データキューに空き領域なし
データキューに、送信メッセージを格納して、送信タスクの処理を継続	送信メッセージを格納できる領域が、データキューにできるまで、送信タスクを待ち状態《メッセージ送信待ち状態》に遷移

データキューに空き領域がある場合の送信処理概要を以下に示します。

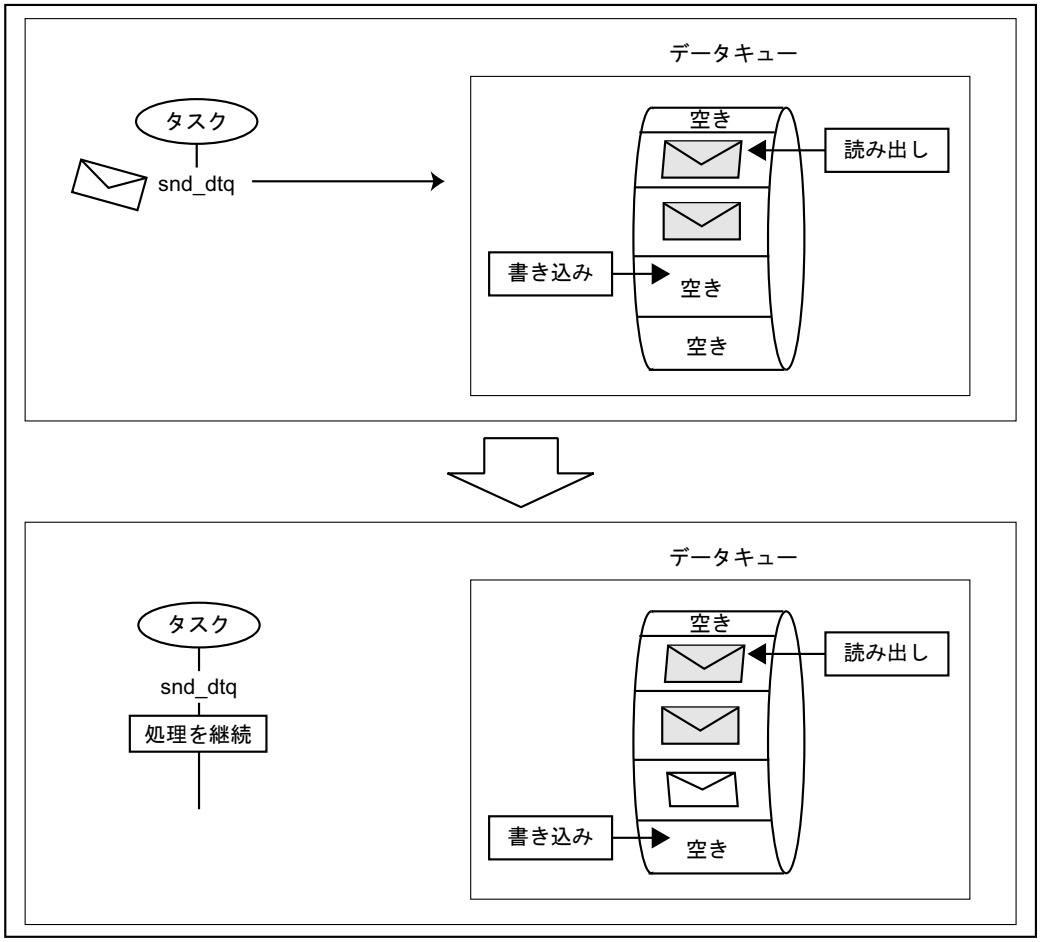


図 1-67 データキューに空き領域がある場合の送信処理概要

データキューに空き領域がない場合の送信処理概要を以下に示します。

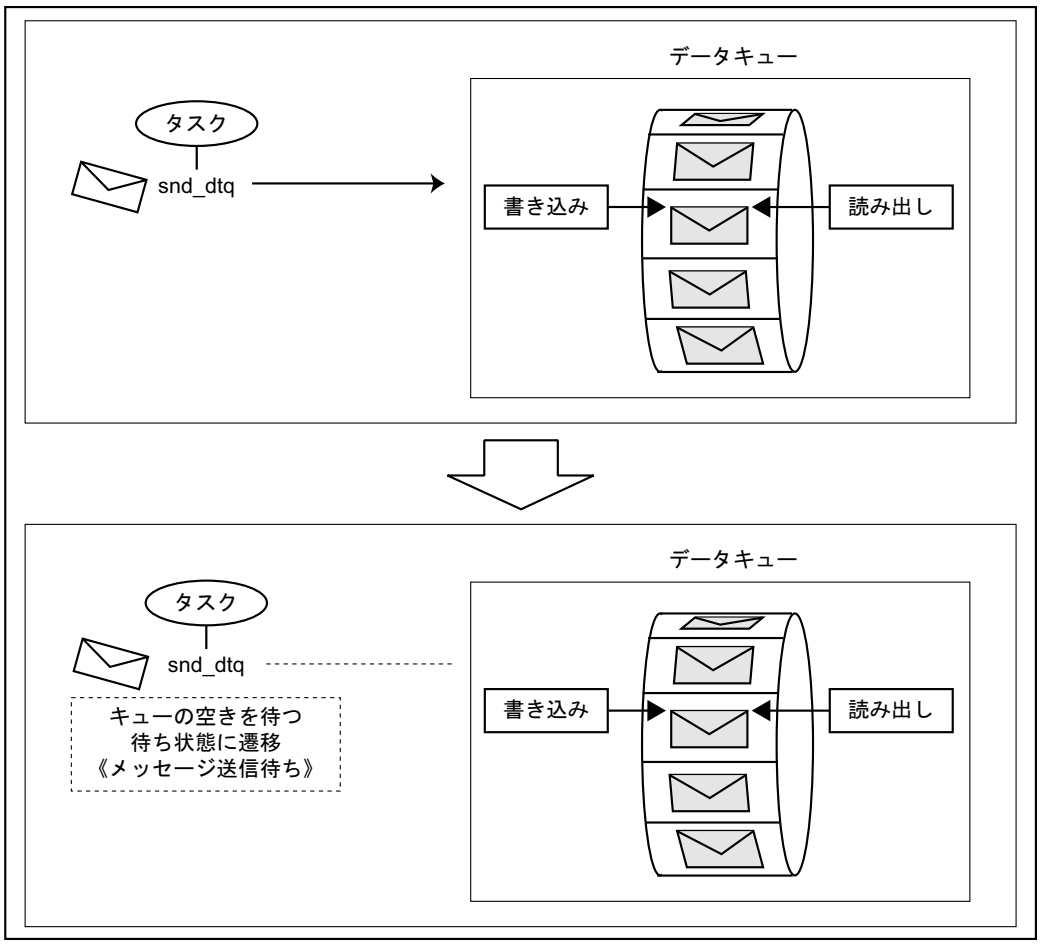


図 1-68　データキューに空き領域がない場合の送信処理概要

データキューのみが持っている機能の 1 つに、「強制送信機能」があります。

強制送信機能は、送信したメッセージを格納できる空き領域がデータキュー領域に確保できない（空き領域がない）場合、データキュー領域の格納されている一番古いメッセージを上書きして送信処理を行う機能です。上書きされたメッセージは、最新のメッセージとして管理されるため、読み出される順番は一番後になります。

以下に、強制送信処理の概要を示します。

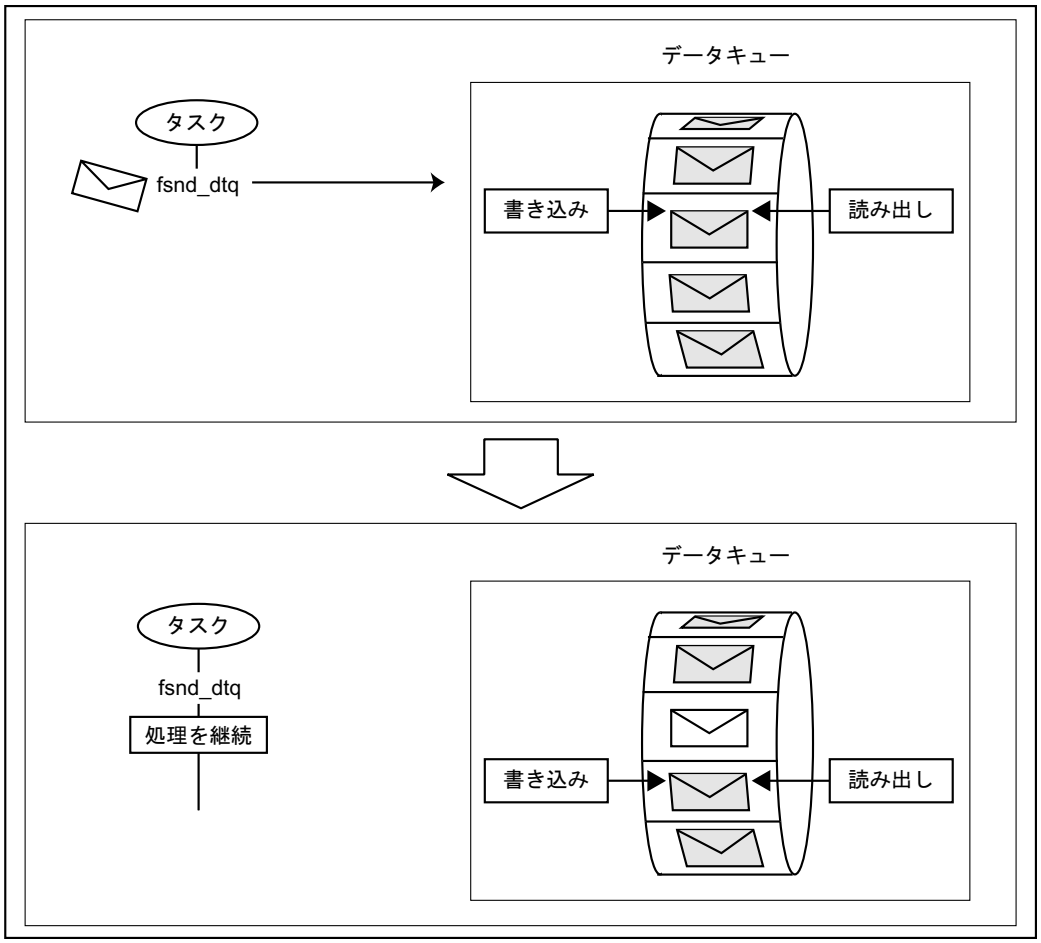


図 1-69　データキューの強制送信処理概要

1.11.3 データキューのメッセージ受信処理概要

データキューにおけるデータ受信処理の概要を以下に示します。

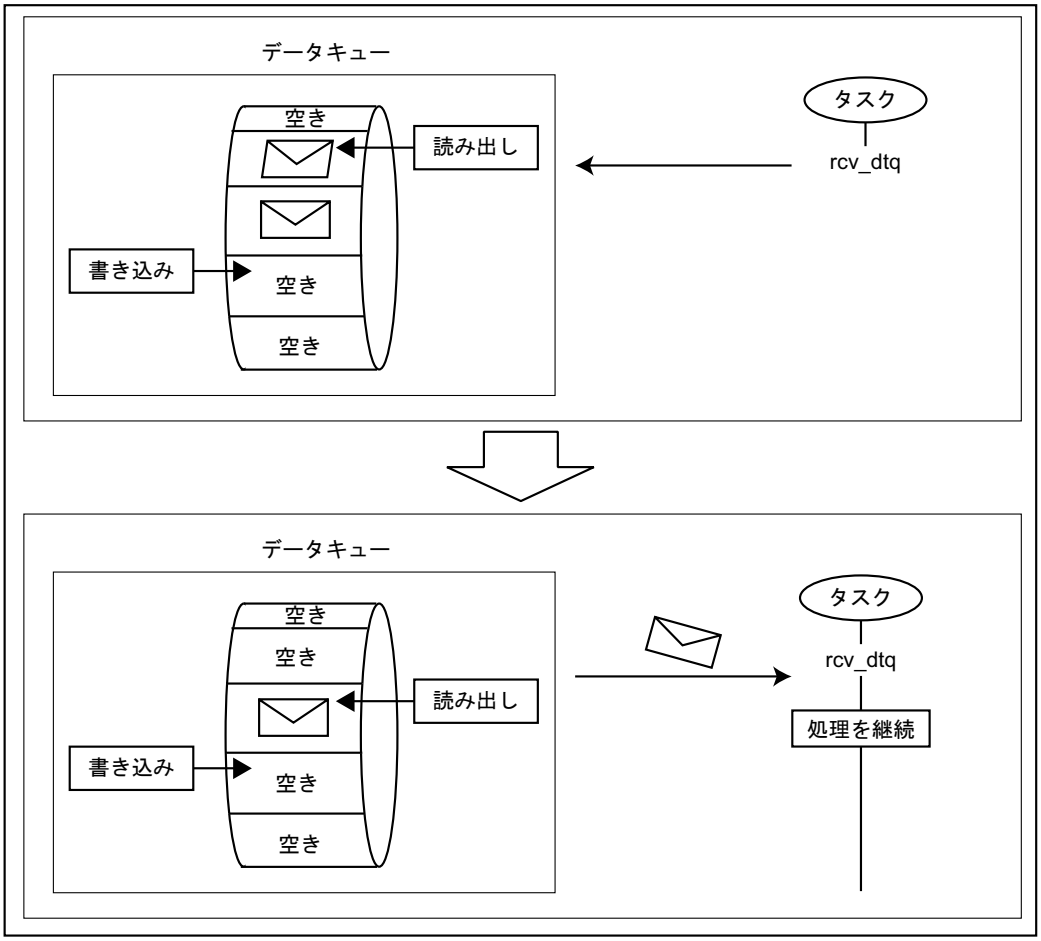


図 1-70 データキューにメッセージがある場合の受信処理概要

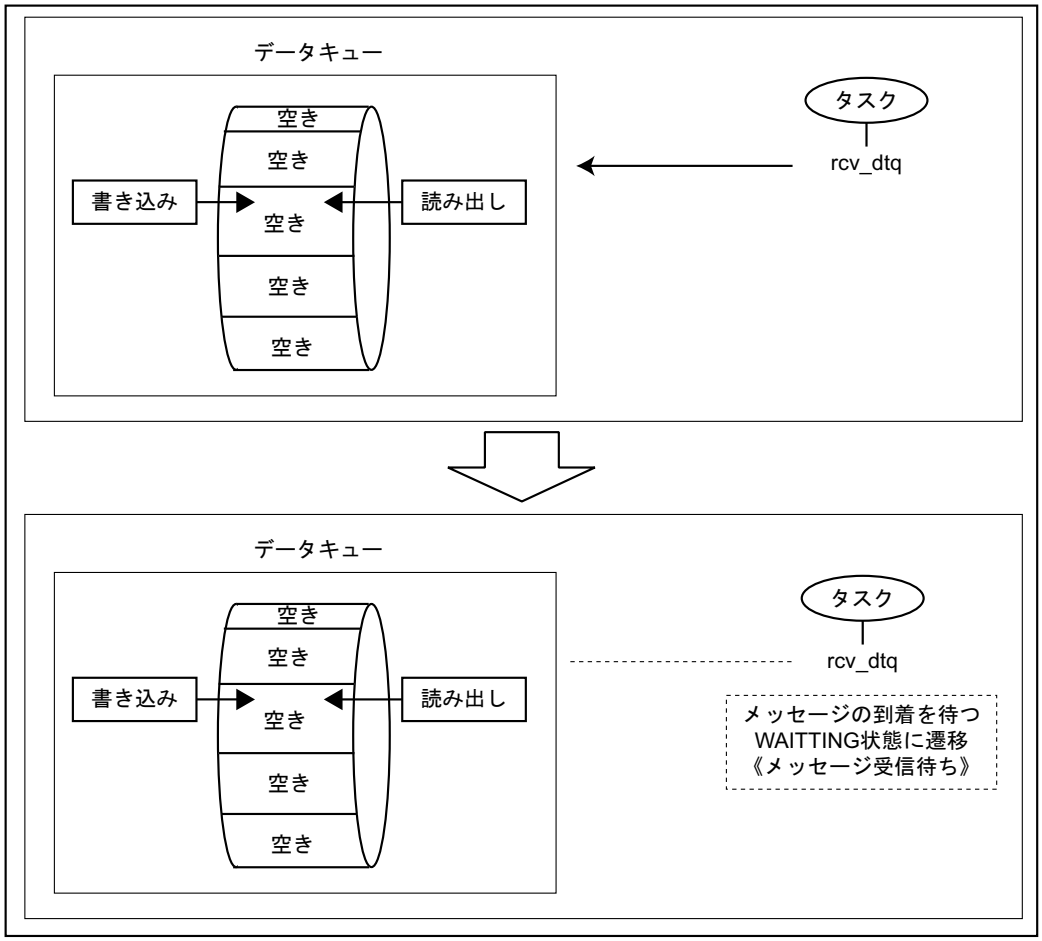


図 1-71　データキューにメッセージがない場合の受信処理概要

1.12 メモリプール

1.12.1 フラグメンテーション

フラグメンテーションとは、メモリ使用領域の断片化のことです。フラグメンテーションの概要を以下に示します。

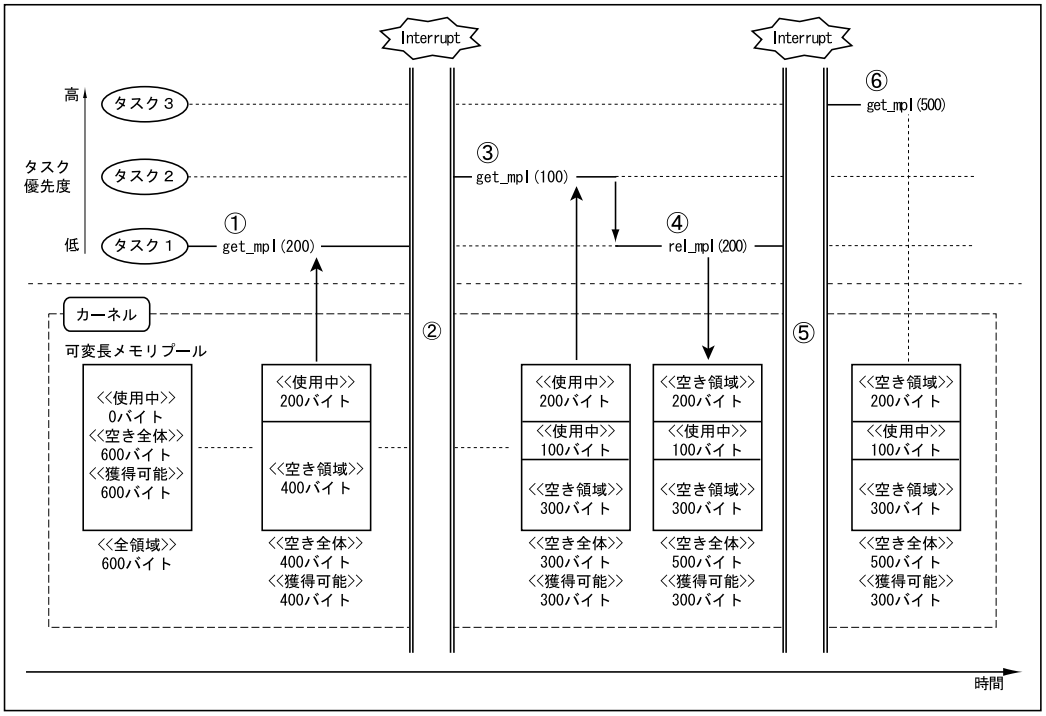


図 1-72　フラグメンテーションの概要

① タスク 1 が、200 バイトの領域を要求し獲得
② 割込みが発生し、割込みハンドラ処理により、タスク切替えが発生（タスク 1→タスク 2）
③ タスク 2 が、100 バイトの領域を要求し獲得
（獲得したメモリを使用した処理にてタスクスイッチが発生：タスク 2→タスク 1）
④ タスク 1 が、獲得していた 200 バイトの領域を返却
⑤ 割込みが発生し、割込みハンドラ処理により、タスク切替えが発生（タスク 1→タスク 3）
⑥ タスク 3 が 500 バイトの領域を要求するが、全体空き領域は 500 バイトで要求を満たしているが、連続最大空き領域は 300 バイトのため、空き領域を待つ WAITING 状態に遷移

このような状態を、フラグメンテーションと言います。

フラグメンテーションを排除する「ガーベージコレクション機能」はサポートしていません。
メモリプール領域のフラグメンテーションについては、アプリケーション側（ユーザシステム側）で管理する必要があります。

1.12.2 メモリプールに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたメモリプールに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) malloc()関数との使い分け

分類：メモリプール

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

μITRON を使用したシステムで、malloc()関数は使用できますか。

■回答

μITRON を使用したシステムで、malloc()関数は使用できません。

malloc()関数にて割り付けられた領域は、OS で識別することができません。

malloc()関数で獲得した領域と、メモリプール機能で獲得した領域が重複した場合、データ破壊が発生することが考えられます。

システムにてメモリ管理が必要な場合、OS のメモリプール機能を使用してください。

1.13 時間管理機能

1.13.1 時間管理機能の概念

時間管理機能で使用するパラメータ：tmout 値について以下に示します。

表 1-27 パラメータ(tmout 値)の意味

HI シリーズ OS	tmout 値の意味
HI7000/4 シリーズ、HI1000/4	tmout 値(ms)
HI2000/3	tmout 値×ハードウェアタイマの周期時間

tslp_tsk(3)を、ハードウェアタイマ周期を 1ms（CFG_TICNUME＝1、CFG_TICDENO＝1）とした場合を例に処理概要を以下に示します。

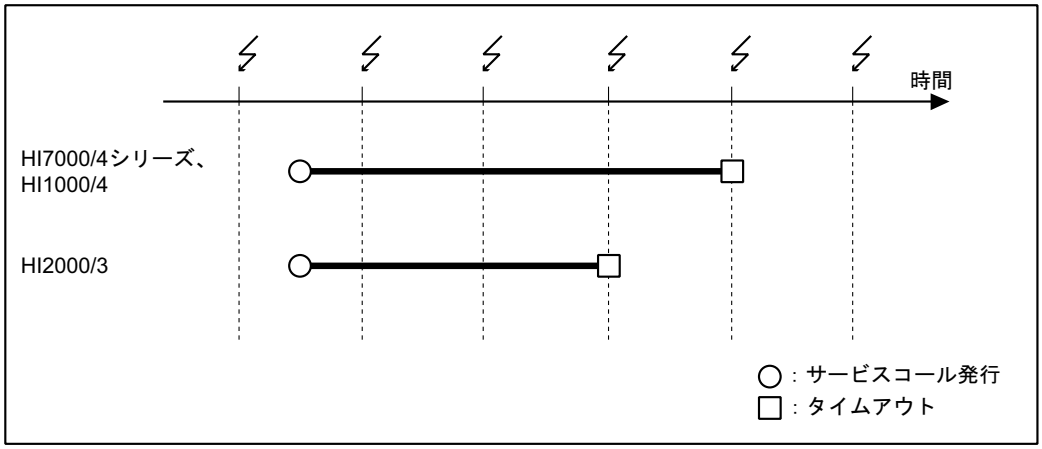


図 1-73 tslp_tsk(3)における処理概要

上記図における tmout 値との誤差について以下に示します。

表 1-28 tslp_tsk(3)発行時の誤差

HI シリーズ OS	tmout 値	誤 差
HI7000/4 シリーズ、HI1000/4	tmout 値＝3 ※待ち時間は 3ms	サービスコールを発行し、タイマ管理下に登録されてから、次のタイムティック供給がされるまで(X)
HI2000/3	tmout 値＝3 ※ハードウェアタイマ周期 3 回目	ハードウェアタイマ周期が入ってから、サービスコールを発行し、タイマ管理下に登録されるまで(Y)

上記表中の誤差(X)、(Y)について、次に示します。

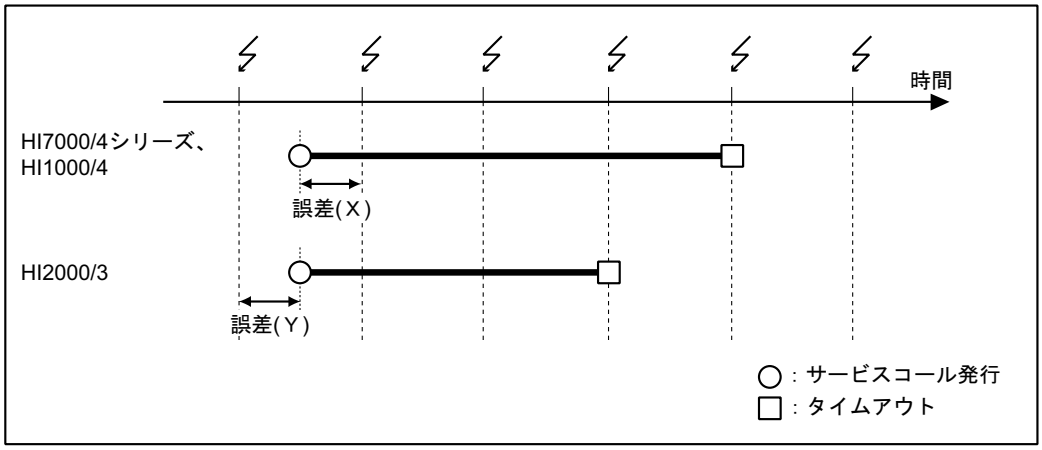


図 1-74　tslp_tsk(3)における誤差

誤差は、ハードウェアタイマ周期時間に影響します。以下に、ハードウェアタイマ周期時間と誤差について示します。

表 1-29　ハードウェアタイマ周期時間と誤差

ハードウェアタイマ 周期時間	メリット	デメリット
短くした場合	時間管理における誤差が小さくなる	ハードウェアタイマ周期処理が増加する分、タスクへの処理時間が減少する
長くした場合	ハードウェアタイマ周期処理が減少する分、タスクへの処理時間が増加する	時間管理における誤差が大きくなる

1.13.2 ハードウェアタイマ周期時間の精度変更

ハードウェアタイマ周期時間の精度の変更方法を以下に示します。

- HI7000/4 シリーズ、HI1000/4：コンフィギュレータ
- HI2000/3：タイマドライバ用ヘッダファイル

各変更手順について、以下に説明します。

(1) HI7000/4 シリーズ、HI1000/4

ハードウェアタイマ周期時間（タイムティックを供給する周期時間。以下、タイムティック供給周期時間と称す）は、標準で 1ms に設定されており、コンフィギュレータで変更することができます。

以下にコンフィギュレータの時間管理機能に関する設定画面を示します。

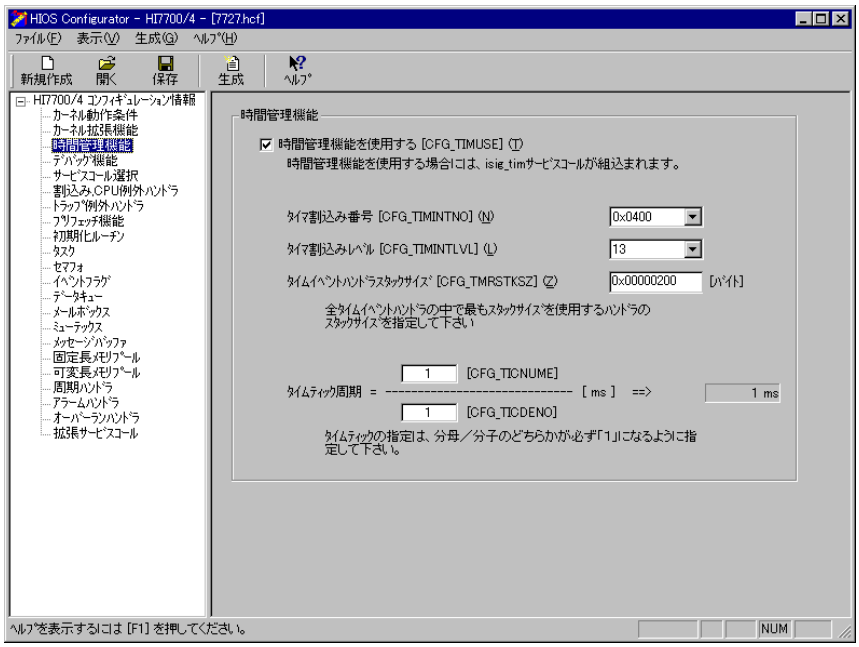


図 1-75 コンフィギュレータ：時間管理機能設定画面

コンフィギュレータの時間管理機能設定画面に示すとおり、タイムティック供給周期時間間隔は、CFG_TICNUME（タイムティック周期の分子）と CFG_TICDENO（タイムティック周期の分母）により、以下のような式で表されます。

$$\text{タイムティック供給周期時間間隔} = \left\{ 1 \times \left[\frac{\text{TIC\_NUME（タイムティック周期の分子}}{\text{TIC\_DENO（タイムティック周期の分母}} \right] \right\}$$

図 1-76 タイムティック供給周期時間間隔の計算式

これは、1ms 周期で供給されるタイムティック供給周期時間間隔を調整（供給時間間隔を長くしたり、短くしたり）するためのものです。標準提供では、タイムティック供給周期時間間隔：1ms を、CFG_TICNUME＝1、CFG_TICDENO＝1 の 1/1 で供給する（標準提供のタイムティック供給周期時間間隔でシステム全体の時間管理を行う）ように定義しています。

CFG_TICNUME、CFG_TICDENO に設定可能な値は以下のとおりです。

◆CFG_TICNUME（タイムティック周期の分子）：1～65,535

◆CFG_TICDENO（タイムティック周期の分母）：1～100

したがって、調整できる 1ms 周期で供給されるタイムティック供給周期時間間隔は、最小 0.01ms（10μs：CFG_TICNUME＝1、CFG_TICDENO＝100 の 1/100）から、最大 65,535ms（65s：CFG_TICNUME＝65,535、CFG_TICDENO＝1 の 65,535/1）までとなります。

(2) HI2000/3

ハードウェアタイマ周期時間は、標準サンプルで 1ms に設定されており、タイマドライバ用ヘッダファイルへの定義で変更することができます。

以下に標準サンプルのタイマドライバ用ヘッダファイルを示します。

```
;************************************************************************
;*specifications ;                                                      *
;*name      = _HIPRG_TIMINI : H8S/2655 TPU0 initialize handler  ;       *
;*function  =                                                   ;       *
;*notes     =                                                   ;       *
;*date      = 99/02/22                                          ;       *
;*author    = Hitachi, Ltd.                                     ;       *
;*attribute = public                                            ;       *
;*class     = system                                            ;       *
;*linkage   =                                                   ;       *
;*input     = ccr(B):interrupt disable                          ;       *
;*          = exr(B):interrupt disable                          ;       *
;*output    = all register unchanged                            ;       *
;*end of specifications ;                                               *
;************************************************************************

     《途中省略》

;###### setting data ####################;:
;
TGRA_DATA: .assign h'30d3;d'12500-1;:(10000us (p/16))-1:10ms=10000us,p=20MHz
```

ハードウェアタイマ周期時間

図 1-77　標準サンプルのタイマドライバ用ヘッダファイル(2655ause.src)

ハードウェアタイマ周期時間の算出例を次に示します。

【参考】ハードウェアタイマ周期時間の算出

以下に、H8S/2655（動作周波数：20MHz）を HI2000/3 で使用することを想定し、10ms のハードウェアタイマ周期時間算出例を示します。

ハードウェアタイマ周期時間(T)は、カウンタクロック周期時間(t)とカウンタ値(n)で決定され、以下のようになります。

- T ＝ ｛ t × (n ＋ 1) ｝

t はタイマコントロールレジスタ(TCR)でカウンタクロックの選択（φ／1、φ／4、φ／16、φ／64）を行うことで決定します。

φ（CPU クロック）が 20MHz の場合、t は以下の数値になります。

- カウンタクロック＝φ／1　：　t＝50ns
- カウンタクロック＝φ／4　：　t＝200ns
- カウンタクロック＝φ／16　：　t＝800ns
- カウンタクロック＝φ／64　：　t＝3.2μs

n はアウトプットコンペアマッチA(TGRA)に 0x0000～0xFFFF の値を設定する事で決定します。

したがって、φ（CPU クロック）が 20MHz の場合、T は以下の範囲になります。

- カウンタクロック＝φ／1　：　T＝50ns ～ 3.27ms
- カウンタクロック＝φ／4　：　T＝200ns ～ 13.1ms
- カウンタクロック＝φ／16　：　T＝800ns ～ 52.4ms
- カウンタクロック＝φ／64　：　T＝3.2μs ～ 209.7ms

《10ms 周期の導き方》

アウトプットコンペアマッチA(TGRA) ＝ タイマ周期時間(s) × n － 1

上式より、タイマ周期時間(s)を 10ms とするので、タイマ周期時間(s) ＝ 10×10^{-3}

カウンタクロックの選択をφ／16 を選択すると、φ（CPU クロック）が 20MHz の場合、

n ＝ $20 \times 10^{6} \div 16$

したがって、

アウトプットコンペアマッチA(TGRA) ＝ タイマ周期時間(s) × n － 1

＝ $(10 \times 10^{-3}) \times (20 \times 10^{6} \div 16) - 1$

＝ 12,499 (0x30D3)

φ（CPU クロック）が 20MHz を使用してタイマ周期時間(s)を 10ms とする場合、アウトプットコンペアマッチA(TGRA)に設定する値は、12,499(0x30D3)となる。

1.13.3 周期ハンドラ

(1) HI7000/4 シリーズ、HI1000/4

周期ハンドラの起動概要について、起動位相：2ms、起動周期：3ms を、ハードウェアタイマ周期を 1ms（CFG_TICNUME＝1、CFG_TICDENO＝1）とした場合を例に以下に示します。

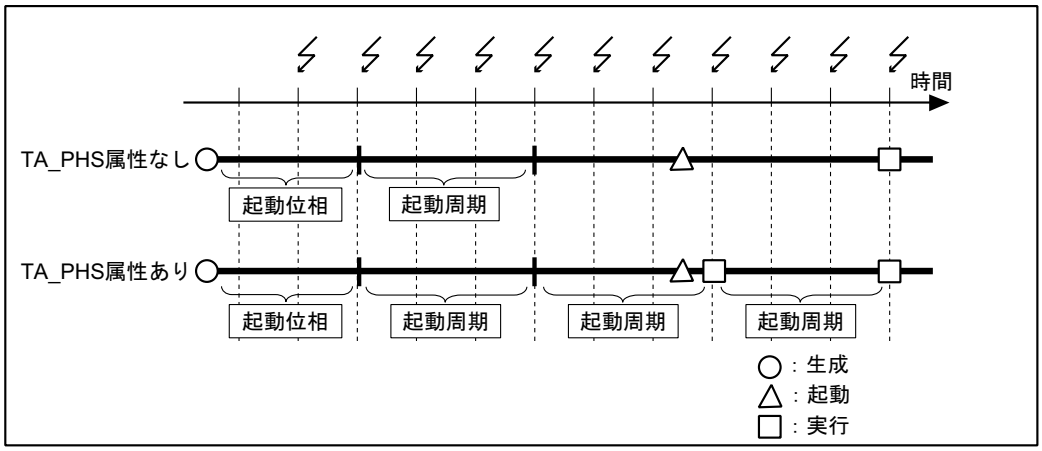


図 1-78 周期ハンドラの起動概要(HI7000/4 シリーズ、HI1000/4)

(2) HI2000/3

周期起動ハンドラの起動概要について、周期起動時間間隔：3ms を、ハードウェアタイマ周期を 1ms とした場合を例に以下に示します。

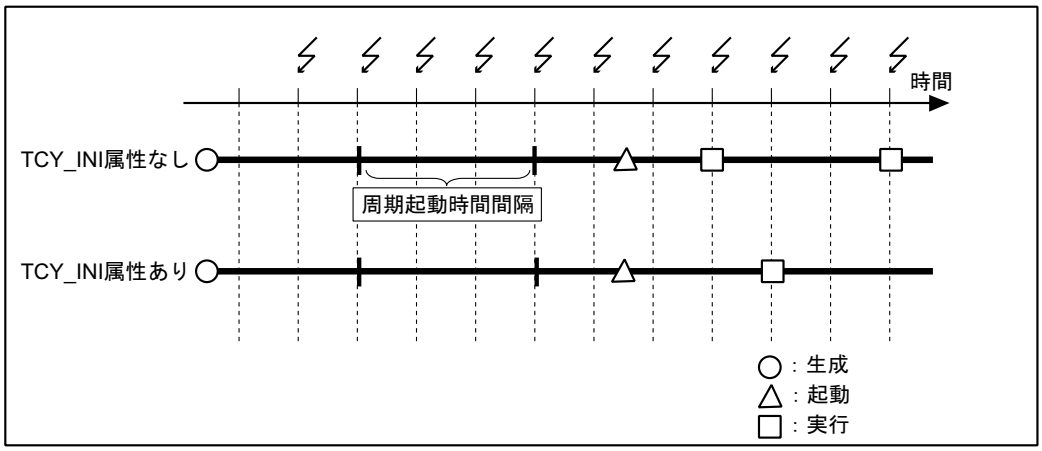


図 1-79 周期起動ハンドラの起動概要(HI2000/3)

1.13.4 タイマ管理機能の処理概要

タイマ管理機能の処理概要について説明します。

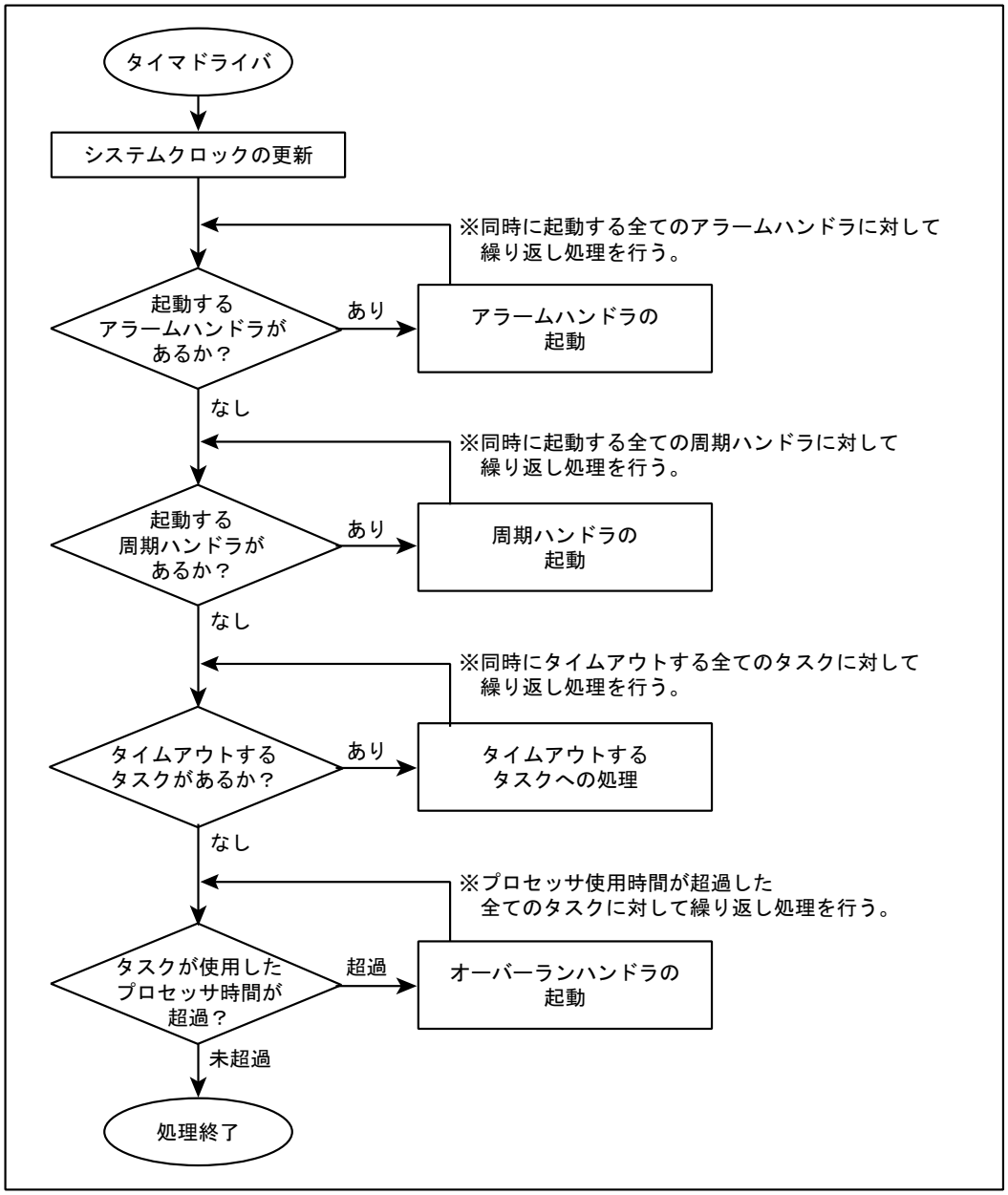


図 1-80 タイマドライバの処理概要(HI7000/4 シリーズ)

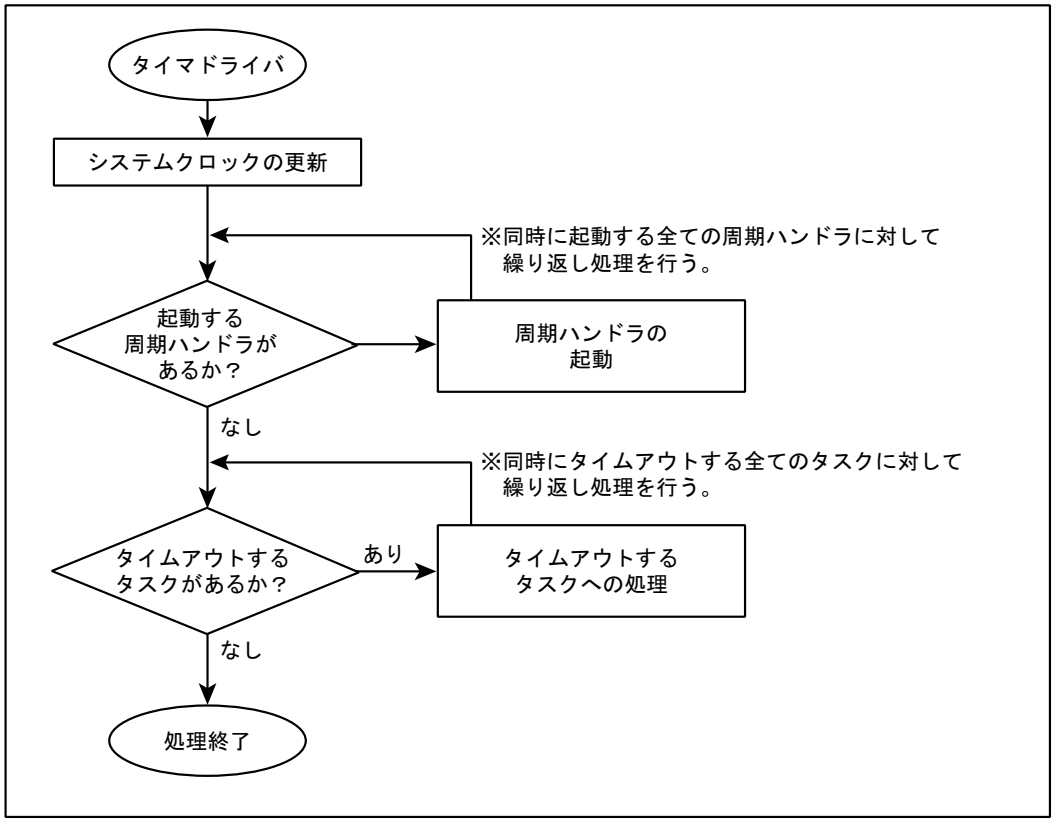


図 1-81　タイマドライバの処理概要(HI2000/3、HI1000/4)

タイマドライバでは、下記のケースで処理時間に影響があります。

- 同時に起動するアラームハンドラ数（HI7000/4 シリーズのみ）
- 同時に起動する周期ハンドラ数
- 同時にタイムアウトするタスク数
- 同時にオーバーランハンドラを起動するタスク数（HI7000/4 シリーズのみ）

同時にタイムアウトするタスク数や、または同時に起動する全てのハンドラ（周期ハンドラ、アラームハンドラ）数が多くなると、それぞれに対するサービス処理の繰り返しが多くなるため、タイマドライバの処理時間が長くなります。また、タイマドライバの処理時間が長くなると、

- 他の割込みに対するレスポンスの低下
- システム時刻の遅れ

などの弊害をもたらします。

2. アプリケーション作成手法

2.1 リセットからタスク起動までの処理概要

HI シリーズ OS における CPU リセット（Power ON リセット含む）からタスクが実行するまでの概要を以下に示します。

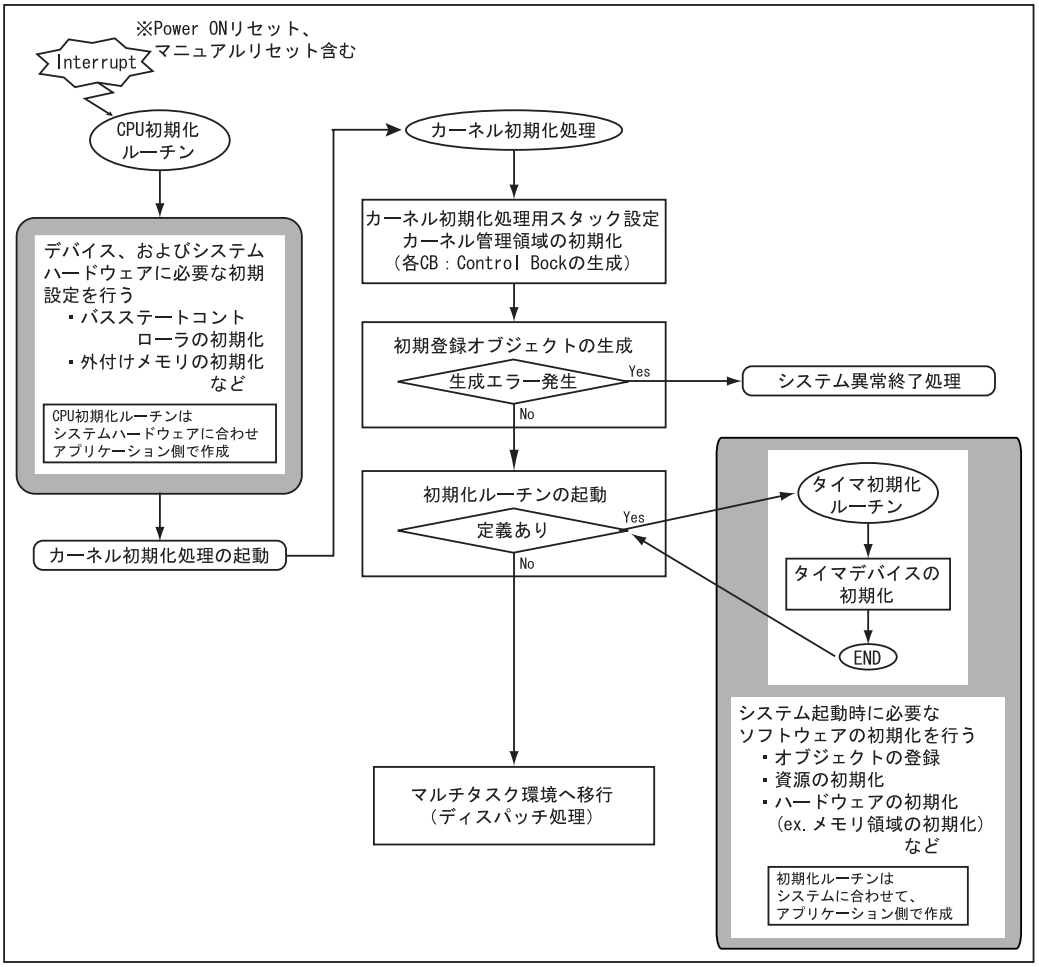


図 2-1　リセットからタスクが起動するまで

CPU リセット信号が入ると、リセットベクタに登録されている「CPU 初期化ルーチン」が起動します。

2.2 CPU 初期化ルーチンの概要

CPU 初期化ルーチンでは、カーネルを含めたソフトウェア全体が動作するのに必要な処理を行います。具体的には、

- 外付けメモリ（SDRAM、SRAM など）を有効にするための BSC（バスステートコントローラ）の設定
- CPU 初期化ルーチン用スタックポインタの設定
- セクションの初期化

などです。「CPU 初期化ルーチン」では、使用するマイコン、およびハードウェアで必要な初期化を行います。したがって、ご使用になるハードウェアやマイコンに応じて、アプリケーション側で作成する必要があります。

「CPU 初期化ルーチン」をすべて C 言語プログラムで記述する事はできません。一部、アセンブラ言語プログラムで作成する必要があります。

C 言語プログラムではスタック（メモリ）をアクセスします。スタック領域の準備が整っていないうちにアクセスすると、CPU 例外が発生する場合があります（CPU 例外が発生すると、システム異常終了となります）。スタックのアクセス準備が完了するまで、「CPU 初期化ルーチン」はアセンブリ言語で記述する必要があります。

HI シリーズ OS では、「CPU 初期化ルーチン」のサンプル提供ファイルを提供しています。サンプル提供の「CPU 初期化ルーチン」を参考に、ご使用になるハードウェアやマイコンにあわせて作成してください。以下に、サンプル提供の CPU 初期化ルーチンについて示します。

表 2-1　CPU 初期化ルーチンの処理概要

項番	HI シリーズ OS	CPU 初期化ルーチン	
		アセンブリ記述	C言語記述
1	HI7000/4 シリーズ	・外付けメモリ(SDRAM、SRAM など)を有効にするための BSC 設定 ・スタックポインタの設定	・セクションの初期化 ・キャッシュの有効化
2	HI2000/3	・スタックポインタの設定 ・割込み制御モードの設定 ・周辺モジュールの設定	*1
3	HI1000/4	・スタックポインタの設定 ・外付けメモリ(SDRAM、SRAM など)を有効にするための BSC 設定 ・割込み制御モードの設定 ・周辺モジュールの設定	*1

【注】 *1　HI2000/3、および HI1000/4 では、「CPU 初期化ルーチン」の C 言語記述サンプル提供ファイルはありません。本アプリケーションノートの「2.6.3 CPU 初期化ルーチン作成例」を参考に作成してください。

以下に各 HI シリーズ OS で提供している「CPU 初期化ルーチン」について説明します。

```
;*************************************************************************;
;*              HI7000/4 CPU initialize routine                            ;*;
;*              Copyright (c)  Hitachi, Ltd. 2000.                         ;*;
;*              Licensed Material of Hitachi, Ltd.                         ;*;
;*              HI7000/4(HS0700ITI41SR) V1.0                               ;*;
;*************************************************************************;
;*************************************************************************;
;*  FILE      = 7604_cpuasm.src ;                                          ;*;
;*  CPU type  = SH7604                                                     ;*;
;*************************************************************************;
        .program        _hi_cpuasm
        .heading        "hi_cpuasm : CPU
        .export         _hi_cpuasm
        .import         _hi_cpuini
        .section        P_hicpuasm,code
;
;*************************************************************************;
;* BSC address                                                            ;*;
;*************************************************************************;
BSC_BASE        .assign  h'fffffe0          ; BSC   base address
BCR1            .assign  h'fffffe0-BSC_BASE ; BCR1  address offset
BCR2            .assign  h'fffffe4-BSC_BASE ; BCR2  address offset
WCR             .assign  h'fffffe8-BSC_BASE ; WCR   address offset
MCR             .assign  h'fffffec-BSC_BASE ; MCR   address offset
RTCSR           .assign  h'ffffff0-BSC_BASE ; RTCSR address offset
RTCNT           .assign  h'ffffff4-BSC_BASE ; RTCNT address offset
RTCOR           .assign  h'ffffff8-BSC_BASE ; RTCOR address offset
;
MD_REG_BASE     .assign  h'ffff8000         ; mode register base address of SDRAM
;
CMF_BIT         .assign  h'0080             ; CMF bit in RTCSR
;
;*************************************************************************;
;* BSC initial data                                                       ;*;
;*  After reset, you must initialize BSC for memory(stack) access at first.;*;
;*  Please modify these definition in order to your hardware.             ;*;
;*************************************************************************;
BCR1_DATA       .assign  h'a55a0000 + h'03f0 ; BCR1  initial data
BCR2_DATA       .assign  h'a55a0000 + h'00fc ; BCR2  initial data
WCR_DATA        .assign  h'a55a0000 + h'aaff ; WCR   initial data
MCR_DATA        .assign  h'a55a0000 + h'0000 ; MCR   initial data
RTCSR_DATA      .assign  h'a55a0000 + h'0000 ; RTCSR initial data
RTCNT_DATA      .assign  h'a55a0000 + h'0000 ; RTCNT initial data
RTCOR_DATA      .assign  h'a55a0000 + h'0000 ; RTCOR initial data
;
STP_REFRESH     .assign  h'a55a0000         ; RTCSR initial data(stop count-up)
;
MODE_DATA       .assign  h'0000             ; data of SDRAM mode register
MODE_ADDRESS    .assign  MD_REG_BASE+MODE_DATA ; address to set MODE_DATA
IDLE_TIME       .assign  566                ; loop counter for idle-time
REFRESH_CNT     .assign  h'8                ; counter for dummy refresh
;
```

初期化処理用データ定義
※必要に応じて設定値の変更、データの追加などを行ってください。

図 2-2　HI7000/4 CPU 初期化ルーチン：アセンブラ記述(SH7604 用)(1/2)

```
;***************************************************************************;
;*  NAME       = _hi_cpuasm                                                  .*.
;*  FUNCTION   = CPU initialize routine ;
;*****************************************
_hi_cpuasm:
;***** Initialize BSC
;       mov.l   #BSC_BASE,r0            ; set BCR base address to gbr
;       ldc     r0,gbr
;
;       mov.l   #BCR1_DATA,r0           ; initialize BCR1
;       mov.l   r0,@(BCR1,gbr)
;
;       mov.l   #BCR2_DATA,r0           ; initialize BCR2
;       mov.l   r0,@(BCR2,gbr)
;
;       mov.l   #WCR_DATA,r0            ; initialize WCR
;       mov.l   r0,@(WCR,gbr)
;
;       mov.l   #MCR_DATA,r0            ; initialize MCR
;       mov.l   r0,@(MCR,gbr)
;
;       mov.l   @(RTCSR,gbr),r0         ; dummy read for CMF off
;       mov.l   #STP_REFRESH,r0         ; stop refresh
;       mov.l   r0,@(RTCSR,gbr)
;
;       mov.l   #RTCNT_DATA,r0          ; initialize RTCNT
;       mov.l   r0,@(RTCNT,gbr)
;
;       mov.l   #RTCOR_DATA,r0          ; initialize RTCOR
;       mov.l   r0,@(RTCOR,gbr)
;
;*** Initialize SDRAM
;
;       mov.l   #IDLE_TIME,r0           ; i
;hi_cpuasm010:
;       add     #-1,r0
;       cmp/eq  #0,r0
;       bf      hi_cpuasm010
;
;       mov.w   #MODE_DATA,r0           ; set mode register
;       mov.l   #MODE_ADDRESS,r1
;       mov.w   r0,@r1
;
;       mov.l   #RTCSR_DATA,r0          ; initialize RTCSR
;       mov.l   r0,@(RTCSR,gbr)
;
;       mov     #0,r1                   ; loop for dummy refresh
;       mov.w   #REFRESH_CNT,r2
;hi_cpuasm020:
;       mov.l   @(RTCSR,gbr),r0
;       tst     #CMF_BIT,r0             ; check CMF bit
;       bt      hi_cpuasm020
;
;       add     #1,r1                   ; loop counter up
;       cmp/eq  r1,r2                   ; if end dummy refresh
;       bt      hi_cpuasm030            ; then goto hi_cpuasm030
;       mov.l   #RTCSR_DATA,r0          ; clear CMF bit
;       bra     hi_cpuasm020
;       mov.l   r0,@(RTCSR,gbr)
;
;hi_cpuasm030:
;
        mov.l   #_hi_cpuini,r0          ; c
        jmp     @r0                     ; jump to hi_cpuini()
        nop                             ; never return to this point
;
                .pool
        .end
```

バスステートコントローラの初期化
※必要に応じてコメント";"を外して
ご使用ください。

外付けメモリ（SDRAM）の初期化
※必要に応じてコメント";"を外して
ご使用ください。

アセンブラ記述のCPU初期化処理が
終了したら、C言語記述の初期化処理へ
分岐します。

図 2-2　HI7000/4 CPU 初期化ルーチン：アセンブラ記述(SH7604 用)(2/2)

```
/************************************************************************/
/*              HI7000/4 CPU initialize routine                         */
/*              Copyright (c)  Hitachi, Ltd. 2000.                      */
/*              Licensed Material of Hitachi, Ltd.                      */
/*              HI7000/4(HS0700ITI41SR) V1.0                            */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7604_cpuini.c ;                                         */
/*  CPU type  = SH7604                                                  */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

/* extern  void    _INITSCT(void); */   /* section-initialize routine      */

#pragma section _hicpuini
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment **

/*** Initialize Software Environment ***/

/*  _INITSCT();   */              /* Call section-initialize routine */

    vsta_knl();                   /* Start kernel                    */
}
```

セクション展開処理の呼び出し
※必要に応じてコメント"/* */”を外して
ご使用ください。

カーネル初期化処理の呼び出し
※CPU初期化処理終了後は必ず
カーネル初期化処理を呼び出してください。

図 2-3　HI7000/4 CPU 初期化ルーチン：C言語記述(SH7604 用)

```
;***********************************************************************;
;*              HI7700/4 CPU initialize routine                        ;*;
;*              Copyright (c) 2000(2003) Renesas Technology Corp.      ;*;
;*              and Renesas Solutions Corp. All Rights Reserved.       ;*;
;*              HI7700/4(HS0770ITI41SR) V1.0                           ;*;
;***********************************************************************;
;***********************************************************************;
;*  FILE      = 7708_cpuasm.src ;                                      ;*;
;*  CPU type  = SH7708                                                 ;*;
;***********************************************************************;
        .program        _hi_cpuasm
        .export         _hi_cpuasm
        .import         _hi_cpuini
        .import         __kernel_pon_sp
        .import         __kernel_man_sp
        .section        P_hicpuasm,code,align=4
;
;***********************************************************************;
;*              EXPEVT address, data                                   ;*;
;***********************************************************************;
CCN_BASE    .assign  h'fffffd0
EXPEVT      .assign  h'ffffffd4-CCN_BAS
;
PON_CODE    .assign  h'000                  ; power-on reset exception code
;
;***********************************************************************;
;*              BSC address                                            ;*;
;***********************************************************************;
BSC_BASE    .assign  h'ffffff60             ; BSC         base address
BCR1        .assign  h'ffffff60-BSC_BASE    ; BCR1        address offset
BCR2        .assign  h'ffffff62-BSC_BASE    ; BCR2        address offset
WCR1        .assign  h'ffffff64-BSC_BASE    ; WCR1        address offset
WCR2        .assign  h'ffffff66-BSC_BASE    ; WCR2        address offset
MCR         .assign  h'ffffff68-BSC_BASE    ; MCR         address offset
DCR         .assign  h'ffffff6a-BSC_BASE    ; DCR         address offset
PCR         .assign  h'ffffff6c-BSC_BASE    ; PCR         address offset
RTCSR       .assign  h'ffffff6e-BSC_BASE    ; RTCSR       address offset
RTCNT       .assign  h'ffffff70-BSC_BASE    ; RTCNT       address offset
RTCOR       .assign  h'ffffff72-BSC_BASE    ; RTCOR       address offset
RFCR        .assign  h'ffffff74-BSC_BASE    ; RFCR        address offset
SDMR_CS2    .assign  h'ffffd000             ; SDMR(CS2)   base address
SDMR_CS3    .assign  h'ffffe000             ; SDMR(CS3)   base address
CMF_BIT     .assign  h'0080                 ; CMF bit in RTCSR
;
;***********************************************************************;
;*              BSC initial data                                       ;*;
;***********************************************************************;
;*  After reset, you must initialize BSC for memory(stack) access at first.;*;
;*  Please modify these definition in order to your hardware.          ;*;
;***********************************************************************;
BCR1_DATA   .assign  h'0000                 ; BCR1        initial data
BCR2_DATA   .assign  h'3ffc                 ; BCR2        initial data
WCR1_DATA   .assign  h'3fff                 ; WCR1        initial data
WCR2_DATA   .assign  h'ffff                 ; WCR2        initial data
MCR_DATA    .assign  h'0000                 ; MCR         initial data
DCR_DATA    .assign  h'0000                 ; DCR         initial data
PCR_DATA    .assign  h'0000                 ; PCR         initial data
RTCSR_DATA  .assign  h'a500 + h'00          ; RTCSR       initial data
RTCNT_DATA  .assign  h'a500 + h'00          ; RTCNT       initial data
RTCOR_DATA  .assign  h'a500 + h'00          ; RTCOR       initial data
RFCR_DATA   .assign  h'a400 + h'000         ; RFCR        initial data
STP_REFRESH .assign  h'a500                 ; RTCSR initial data(stop count-up)
SDMR2_DATA  .assign  h'0230                 ; SDMR_CS2    initial data
SDMR3_DATA  .assign  h'0230                 ; SDMR_CS3    initial data
IDLE_TIME   .assign  h'566                  ; loop counter for idle-time
REFRESH_CNT .assign  h'8                    ; counter for dummy refresh
;
```

初期化処理用データ定義
※必要に応じて設定値の変更、データの追加などを行ってください。

図 2-4　HI7700/4 CPU 初期化ルーチン：アセンブラ記述(SH7708 用)(1/3)

```
;*************************************************************************;
;*  NAME       = _hi_cpuasm                                                ;*;
;*  FUNCTION   = CPU initialize routine                                    ;*;
;*************************************************************************;
_hi_cpuasm:
;***** Initialize BSC
;       mov.l   #BSC_BASE,r0            ; set BCR base address to gbr
;       ldc     r0,gbr
;
;       mov.w   #BCR1_DATA,r0           ; Initialize BCR1
;       mov.w   r0,@(BCR1,gbr)
;
;       mov.w   #BCR2_DATA,r0           ; Initialize BCR2
;       mov.w   r0,@(BCR2,gbr)
;
;       mov.w   #WCR1_DATA,r0           ; Initialize WCR1
;       mov.w   r0,@(WCR1,gbr)
;
;       mov.w   #WCR2_DATA,r0           ; Initialize WCR2
;       mov.w   r0,@(WCR2,gbr)
;
;       mov.w   #MCR_DATA,r0            ; Initialize MCR
;       mov.w   r0,@(MCR,gbr)
;
;       mov.w   #DCR_DATA,r0            ; Initialize DCR
;       mov.w   r0,@(DCR,gbr)
;
;       mov.w   #PCR_DATA,r0            ; Initialize PCR
;       mov.w   r0,@(PCR,gbr)
;
;       mov.w   #STP_REFRESH,r0         ; stop refresh
;       mov.w   r0,@(RTCSR,gbr)
;
;       mov.w   #RTCNT_DATA,r0          ; Initialize RTCNT
;       mov.w   r0,@(RTCNT,gbr)
;
;       mov.w   #RTCOR_DATA,r0          ; Initialize RTCOR
;       mov.w   r0,@(RTCOR,gbr)
;
;       mov.w   #RFCR_DATA,r0           ; Initialize RFCR
;       mov.w   r0,@(RFCR,gbr)
;
```

バスステートコントローラの初期化
※必要に応じてコメント";"を外して
ご使用ください。

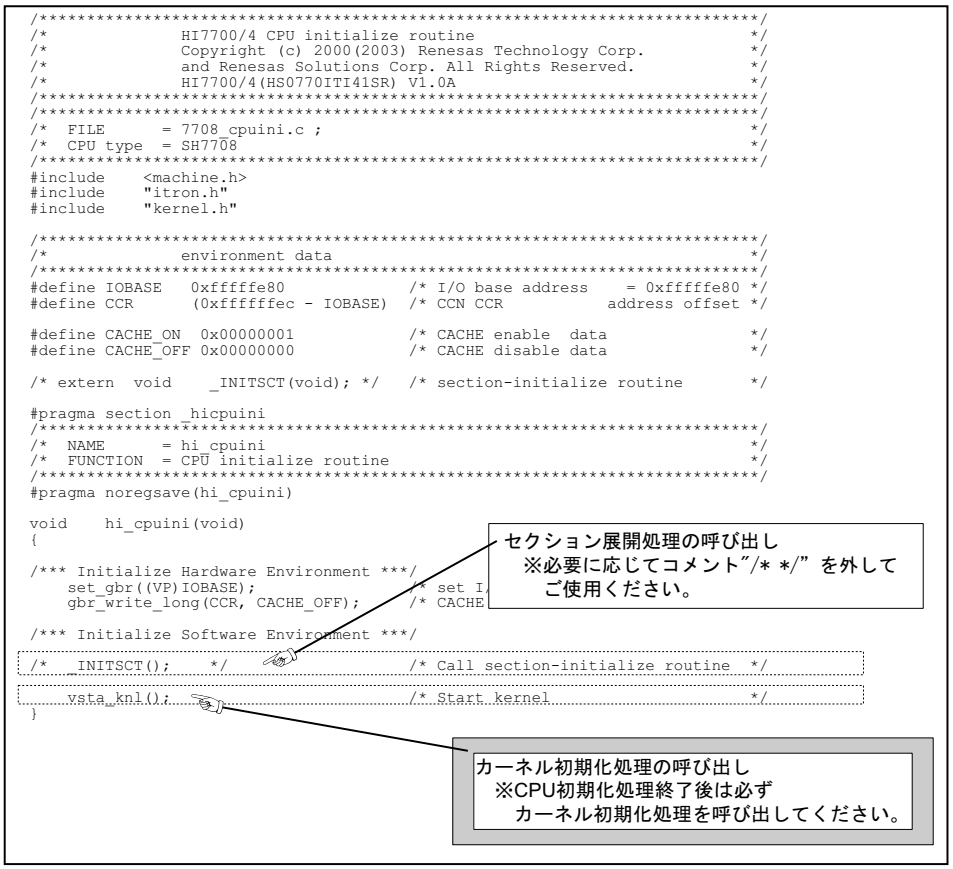
図 2-4　HI7700/4 CPU 初期化ルーチン：アセンブラ記述(SH7708 用)(2/3)

外付けメモリ（SDRAM）の初期化
※必要に応じてコメント";"を外して
ご使用ください。

```
;*** Initialize SDRAM
;
;       mov.l   #IDLE_TIME,r0           ; loop for idle-time
;hicpuasm010:
;       add     #-1,r0
;       cmp/eq  #0,r0
;       bf      hicpuasm010
;
;       mov.l   #SDMR_CS2,r0            ; Initialize SDMR(CS2)
;       mov.l   #SDMR2_DATA*4,r2
;       mov.b   r1,@(r0,r2)             ; write dummy data(r1)
;
;       mov.l   #SDMR_CS3,r0            ; Initialize SDMR(CS3)
;       mov.l   #SDMR3_DATA*4,r2
;       mov.b   r1,@(r0,r2)             ; write dummy data(r1)
;
;       mov.w   #RTCSR_DATA,r0          ; Initialize RTCSR
;       mov.w   r0,@(RTCSR,gbr)
;
;       mov.w   #REFRESH_CNT,r2
;hi_cpuasm020:
;       mov.w   @(RFCR,gbr),r0          ; read RFCR
;       cmp/ge  r2,r0                   ; if end dummy refresh
;       bf      hi_cpuasm020            ; else goto hi_cpuasm020
;
;hi_cpuasm030:
;
;***** Initialize sp and jump to hi_cpuini() written by C-language
        mov.l   #CCN_BASE,r2            ; get CCN base address
        mov.l   #PON_CODE,r3            ; get exception code to power-on
        mov.l   @(EXPEVT,r2),r0         ; get exception code
        cmp/eq  r3,r0                   ; if exception != power-on
        bf      hi_cpuasm050            ; then hi_cpuasm050
;
        mov.l   #__kernel_pon_sp,r2          ; get stack address
;
hi_cpuasm040:
        mov     r2,r15                  ; set SP
;
        mov.l   #_hi_cpuini,r0          ; get hi_cpuini address
        jmp     @r0                     ; jump to hi_cpuini()
        nop                             ; never return to this point
;
hi_cpuasm050:
        mov.l   #__kernel_man_sp,r2          ; get stack address
        bra     hi_cpuasm040
        nop
;
                .pool
;
        .end
```

アセンブラ記述のCPU初期処理が
終了したら、C言語記述の初期化処理へ
分岐します。

図 2-4　HI7700/4 CPU 初期化ルーチン：アセンブラ記述(SH7708 用)(3/3)




```
/************************************************************************/
/*              HI7700/4 CPU initialize routine                         */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.       */
/*              and Renesas Solutions Corp. All Rights Reserved.        */
/*              HI7700/4(HS0770ITI41SR) V1.0A                           */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7708_cpuini.c ;                                         */
/*  CPU type  = SH7708                                                  */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

/************************************************************************/
/*              environment data                                        */
/************************************************************************/
#define IOBASE   0xfffffe80             /* I/O base address    = 0xfffffe80 */
#define CCR      (0xffffffec - IOBASE)  /* CCN CCR           address offset */

#define CACHE_ON  0x00000001            /* CACHE enable  data              */
#define CACHE_OFF 0x00000000            /* CACHE disable data              */

/* extern  void    _INITSCT(void); */   /* section-initialize routine      */

#pragma section _hicpuini
/************************************************************************/
/*  NAME      = hi_cpuini                                               */
/*  FUNCTION  = CPU initialize routine                                  */
/************************************************************************/
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment ***/
    set_gbr((VP)IOBASE);                /* set I
    gbr_write_long(CCR, CACHE_OFF);     /* CACHE

/*** Initialize Software Environment ***/

/*  _INITSCT();     */                  /* Call section-initialize routine  */

    vsta_knl();                         /* Start kernel                     */
}
```

図 2-5　HI7700/4 CPU 初期化ルーチン：C言語記述(SH7708 用)

```
;*********************************************************************;
;*              HI7750/4 CPU initialize routine                      ;*;
;*              Copyright (c) 2000(2003) Renesas Technology Corp.    ;*;
;*              and Renesas Solutions Corp. All Rights Reserved.     ;*;
;*              HI7750/4(HS0775ITI41SR) V1.0                         ;*;
;*********************************************************************;
;*********************************************************************;
;*  FILE      = 7750_cpuasm.src ;                                    ;*;
;*  CPU type  = SH7750                                               ;*;
;*********************************************************************;
        .program        _hi_cpuasm
        .heading        "hi_cpuasm : CPU initialize routine"
        .export         _hi_cpuasm
        .import         _hi_cpuini
        .import         __kernel_pon_sp
        .import         __kernel_man_sp
        .section        P_hicpuasm,code,align=4
;
;*********************************************************************;
;* EXPEVT address, data                                              *;
;*********************************************************************;
CCN_BASE     .assign  h'ff000020
EXPEVT       .assign  h'ff000024-CCN_BAS
;
PON_CODE     .assign  h'000                 ; power-on reset exception code
;
;*********************************************************************;
;* BSC address                                                       ;*;
;*********************************************************************;
BSC_BASE     .assign  h'ff800000            ; BSC         base address
BCR1         .assign  h'ff800000-BSC_BASE   ; BCR1        address offset
BCR2         .assign  h'ff800004-BSC_BASE   ; BCR2        address offset
WCR1         .assign  h'ff800008-BSC_BASE   ; WCR1        address offset
WCR2         .assign  h'ff80000c-BSC_BASE   ; WCR2        address offset
WCR3         .assign  h'ff800010-BSC_BASE   ; WCR3        address offset
MCR          .assign  h'ff800014-BSC_BASE   ; MCR         address offset
PCR          .assign  h'ff800018-BSC_BASE   ; PCR         address offset
RTCSR        .assign  h'ff80001c-BSC_BASE   ; RTCSR       address offset
RTCNT        .assign  h'ff800020-BSC_BASE   ; RTCNT       address offset
RTCOR        .assign  h'ff800024-BSC_BASE   ; RTCOR       address offset
RFCR         .assign  h'ff800028-BSC_BASE   ; RFCR        address offset
SDMR2        .assign  h'ff900000            ; SDMR2 address
SDMR3        .assign  h'ff940000            ; SDMR3 address
CMF_BIT      .assign  h'0080                ; CMF bit in RTCSR
;
;*********************************************************************;
;* BSC initial data                                                  ;*;
;*  After reset, you must initialize BSC for memory(stack) access at first.;*;
;*  Please modify these definition in order to your hardware.        ;*;
;*********************************************************************;
BCR1_DATA    .assign  h'00000000            ; BCR1        initial data
BCR2_DATA    .assign  h'3ffc                ; BCR2        initial data
WCR1_DATA    .assign  h'77777777            ; WCR1        initial data
WCR2_DATA    .assign  h'fffeefff            ; WCR2        initial data
WCR3_DATA    .assign  h'07777777            ; WCR3        initial data
MCR_DATA     .assign  h'00000000            ; MCR         initial data
PCR_DATA     .assign  h'0000                ; PCR         initial data
RTCSR_DATA   .assign  h'a500 + h'00         ; RTCSR       initial data
RTCNT_DATA   .assign  h'a500 + h'00         ; RTCNT       initial data
RTCOR_DATA   .assign  h'a500 + h'00         ; RTCOR       initial data
RFCR_DATA    .assign  h'a400 + h'000        ; RFCR        initial data
STP_REFRESH  .assign  h'a500                ; RTCSR initial data(stop count-up)
SDMR2_DATA   .assign  h'0230                ; SDMR2       initial data
SDMR3_DATA   .assign  h'0230                ; SDMR3       initial data
IDLE_TIME    .assign  h'1000                ; loop counter for idle-time
REFRESH_CNT  .assign  h'8                   ; counter for dummy refresh
;
```

初期化処理用データ定義
※必要に応じて設定値の変更、データの追加などを行ってください。

図 2-6　HI7750/4 CPU 初期化ルーチン：アセンブラ記述(SH7750 用)(1/3)

```
;*********************************************************************;
;*NAME       = _hi_cpuasm                                             ;*;
;*FUNCTION   = CPU initialize routine ;                               ;*;
;*********************************************************************;
_hi_cpuasm:
;***** Initialize BSC
;       mov.l   #BSC_BASE,r0            ; set BSC base address to gbr
;       ldc     r0,gbr
;
;       mov.l   #BCR1_DATA,r0           ; Initialize BCR1
;       mov.l   r0,@(BCR1,gbr)
;
;       mov.w   #BCR2_DATA,r0           ; Initialize BCR2
;       mov.w   r0,@(BCR2,gbr)
;
;       mov.l   #WCR1_DATA,r0           ; Initialize WCR1
;       mov.l   r0,@(WCR1,gbr)
;
;       mov.l   #WCR2_DATA,r0           ; Initialize WCR2
;       mov.l   r0,@(WCR2,gbr)
;
;       mov.l   #WCR3_DATA,r0           ; Initialize WCR3
;       mov.l   r0,@(WCR3,gbr)
;
;       mov.l   #MCR_DATA,r0            ; Initialize MCR
;       mov.l   r0,@(MCR,gbr)
;
;       mov.w   #PCR_DATA,r0            ; Initialize PCR
;       mov.w   r0,@(PCR,gbr)
;
;       mov.w   #STP_REFRESH,r0         ; stop refresh
;       mov.w   r0,@(RTCSR,gbr)
;
;       mov.w   #RTCNT_DATA,r0          ; Initialize RTCNT
;       mov.w   r0,@(RTCNT,gbr)
;
;       mov.w   #RTCOR_DATA,r0          ; Initialize RTCOR
;       mov.w   r0,@(RTCOR,gbr)
;
;       mov.w   #RFCR_DATA,r0           ; Initialize RFCR
;       mov.w   r0,@(RFCR,gbr)
;
```

バスステートコントローラの初期化
※必要に応じてコメント";"を外して
ご使用ください。

図 2-6　HI7750/4 CPU 初期化ルーチン：アセンブラ記述(SH7750 用)(2/3)

外付けメモリ（SDRAM）の初期化
※必要に応じてコメント";" を外して
ご使用ください。

```
;*** Initialize SDRAM
;
;       mov.l   #IDLE_TIME,r0           ; loop for idle-time
;hicpuasm010:
;       add     #-1,r0
;       cmp/eq  #0,r0
;       bf      hicpuasm010
;
;       mov.l   #SDMR2,r0               ; Initialize SDMR(CS2)
;       mov.l   #SDMR2_DATA*4,r2
;       mov.b   r1,@(r0,r2)             ; write dummy data(r1)
;
;       mov.l   #SDMR3,r0               ; Initialize SDMR(CS3)
;       mov.l   #SDMR3_DATA*4,r2
;       mov.b   r1,@(r0,r2)             ; write dummy data(r1)
;
;       mov.w   #RTCSR_DATA,r0          ; Initialize RTCSR
;       mov.w   r0,@(RTCSR,gbr)
;
;       mov.w   #REFRESH_CNT,r2
;hi_cpuasm020:
;       mov.w   @(RFCR,gbr),r0          ; read RFCR
;       cmp/ge  r2,r0                   ; if end dummy refresh
;       bf      hi_cpuasm020            ; else goto hi_cpuasm020
;
;hi_cpuasm030:
;
;***** Initialize sp and jump to hi_cpuini() written by C-language
        mov.l   #CCN_BASE,r2            ; get CCN base address
        mov.l   #PON_CODE,r3            ; get exception code to power-on
        mov.l   @(EXPEVT,r2),r0         ; get exception code
        cmp/eq  r3,r0                   ; if exception != power-on
        bf      hi_cpuasm050            ; then hi_cpuasm050
;
        mov.l   #__kernel_pon_sp,r2     ; get stack address
;
hi_cpuasm040:
        mov     r2,r15                  ; set SP
;
        mov.l   #_hi_cpuini,r0          ; get hi_cpuini address
        jmp     @r0                     ; jump to hi_cpuini()
        nop                             ; never return to this point
;
hi_cpuasm050:
        mov.l   #__kernel_man_sp,r2     ; get stack address
        bra     hi_cpuasm040
        nop
;
                .pool
;
        .end
```

アセンブラ記述のCPU初期処理が
終了したら、C言語記述の初期化処理へ
分岐します。

図 2-6　HI7750/4 CPU 初期化ルーチン：アセンブラ記述(SH7750 用)(3/3)

```
/************************************************************************/
/*              HI7750/4 CPU initialize routine                         */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.       */
/*              and Renesas Solutions Corp. All Rights Reserved.        */
/*              HI7750/4(HS0775ITI41SR) V1.0A                           */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7750_cpuini.c ;                                         */
/*  CPU type  = SH7750                                                  */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

#define CCR_DATA 0x0000090d             /* CACHE enable data            */

/* extern  void    _INITSCT(void); */   /* section-initialize routine   */

#pragma section _hicpuini
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment ***/
/*  vini_cac((UW)CCR_DATA); */          /* CACH

/*** Initialize Software Environment ***/

/*  _INITSCT();     */                  /* Call section-initialize routine  */

    vsta_knl();                         /* Start kernel                 */
}
```

セクション展開処理の呼び出し
※必要に応じてコメント"/* */" を外して
ご使用ください。

カーネル初期化処理の呼び出し
※CPU初期化処理終了後は必ず
カーネル初期化処理を呼び出してください。

図 2-7　HI7750/4 CPU 初期化ルーチン：C言語記述(SH7750 用)

```
;***********************************************************************
;***                                                                 ***
;***    HI2000/3 Version (uITRON V3.0)                               ***
;***    HI2000/3 user/system application file                        ***
;***                                                                 ***
;***    Copyright (c)  Hitachi, Ltd. 1998.                           ***
;***    Licensed Material of Hitachi, Ltd.                           ***
;***                                                                 ***
;***********************************************************************
        .program        _2655acpu
        .heading        "### 2655acpu.src : H8S/2655 initialize module ###"
;
        .section        h2susr_ram,data,align=2
        .res.b          18
CPUINI_SP:      .equ    $
;
        .section        h2suser,code,align=2
;
        .export         _H_2S_CPUINI
        .import         _H_2S_INIT
;
        .aifdef DX
        .import         _HI_DEAMON_INI
        .aendi
;
;***********************************************************************
;*specifications ;                                                     *
;*name      = _H_2S_CPUINI : H8S/2655 initialize module         ;     *
;*function  =                                                   ;     *
;*notes     =                                                   ;     *
;*date      = 99/02/22                                          ;     *
;*author    = Hitachi, Ltd.                                     ;     *
;*attribute = public                                            ;     *
;*class     = system                                            ;     *
;*linkage   =                                                   ;     *
;*input     = none                                              ;     *
;*output    = none                                              ;     *
;*end of specifications ;                                       ;     *
;***********************************************************************
;
        .radix  d
;
;###### interrupt register address #####;:
SYSCR:          .assign h'00ffff39      ;:system control register
MSTPCRH:        .assign h'00ffff3c      ;:module stop control register H
MSTPCRL:        .assign h'00ffff3d      ;:module stop control register L
;
;###### system control register ########;:SYSCR
RAME:           .assign b'00000001      ;:RAM enable
NMIEG:          .assign b'00001000      ;:NMI edge select
INTM0:          .assign b'00010000      ;:interrupt mode 0
INTM1:          .assign b'00100000      ;:interrupt mode 1
MACS:           .assign b'10000000      ;:MAC register saturation
;
;### module stop control register H ####;:MSTPCRH
A_D:            .assign b'11111101      ;:A/D module select
D_A:            .assign b'11111011      ;:D/A module select
PPG:            .assign b'11110111      ;:PPG module select
TMR:            .assign b'11101111      ;:TMR module select
TPU:            .assign b'11011111      ;:TPU module select
DTC:            .assign b'10111111      ;:DTC module select
DMAC:           .assign b'01111111      ;:DMAC module select
;
;### module stop control register L ####;:MSTPCRL
SCI0:           .assign b'11011111      ;:SCI0 module select
SCI1:           .assign b'10111111      ;:SCI1 module select
SCI2:           .assign b'01111111      ;:SCI2 module select
;
```

初期化処理用データ定義
※必要に応じて設定値の変更、データの追加などを行ってください。

図 2-8　HI2000/3 CPU 初期化ルーチン(H8S/2655 用)(1/2)

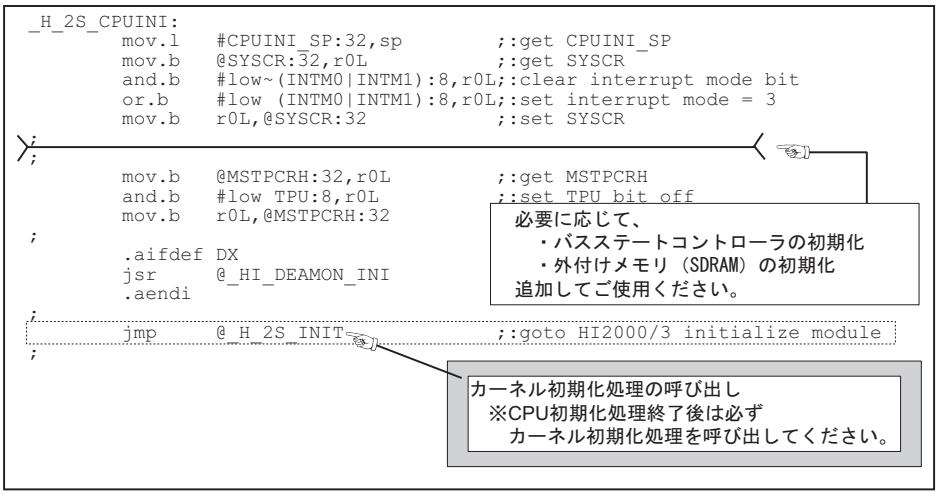


図 2-8　HI2000/3 CPU 初期化ルーチン(H8S/2655 用)(2/2)

```
;***********************************************************************
;*                                                                     *
;*  HI1000/4 Version (uITRON V4.0)                                     *
;*  HI1000/4 user/system application file                              *
;*                                                                     *
;*  Copyright (C) 1998,2003  Renesas Technology Corp. All right reserved *
;*                                                                     *
;***********************************************************************
        .program        _1650cpu
        .heading        "### 1650cpu.src : H8SX/1650 initialize module ###"
;
        .section        P_hicpuini,code,align=2
;
        .export         _KERNEL_H_CPUINI
        .import         _KERNEL_HI_OS_SP
        .import         _vsta_knl
;
;***********************************************************************
;*specifications ;                                                     *
;*name      = 1650cpu.src        : H8SX/1650 initialize module  ;      *
;*function  = CPU Initialize routine                            ;      *
;*notes     =                                                   ;      *
;*input     = none                                              ;      *
;*output    = none                                              ;      *
;*end of specifications ;                                       ;      *
;***********************************
;
        .radix  d
;
;############ register address ##########;:
INTCR:          .assign h'00FFFF32      ;;interrupt control register
MSTPCRA:        .assign h'00FFFDC8      ;:module stop control register A
ABWCR:          .assign h'00FFFD84      ;:bus width control register
ASTCR:          .assign h'00FFFD86      ;:access state control register
WTCRA:          .assign h'00FFFD88      ;:wait control register A
WTCRB:          .assign h'00FFFD8A      ;:wait control register B
;
;#### interrupt control register #######;:INTCR
INTM0:          .assign b'00010000      ;:interrupt mode bit0
INTM1:          .assign b'00100000      ;:interrupt mode bit1
;
;### module stop control register A ####;:MSTPCRA
MSTPA0:         .assign h'FFFE          ;:TPU ch 5 - 0
VBR_ADR:        .assign 0               ;:VBR address
;
_KERNEL_H_CPUINI:
        mov.l   #_KERNEL_HI_OS_SP:32,sp
        mov.l   #VBR_ADR,er0
        ldc.l   er0,vbr
        mov.l   #h'ffffff00,er0
        ldc.l   er0,sbr                 ;:initial SBR
;
;       mov.w   #h'00ff,@ABWCR:32       ;:set ABWCR
;       mov.w   #h'0000,@ASTCR:32       ;:set ASTCR
;       mov.w   #h'0000,@WTCRA:32       ;:set WTCRA
;       mov.w   #h'0000,@WTCRB:32       ;:set WTCRB
;
        mov.b   #INTM1,r0L
        mov.b   r0L,@INTCR:32
;
        mov.w   @MSTPCRA:32,r0
        and.w   #MSTPA0:16,r0
        mov.w   r0,@MSTPCRA:32
;
        jmp     @_vsta_knl              ;:goto vsta_knl
;
```

初期化処理用データ定義
※必要に応じて設定値の変更、データの追加などを行ってください。

必要に応じて、
・バスステートコントローラの初期化
・外付けメモリ（SDRAM）の初期化
追加してご使用ください。

カーネル初期化処理の呼び出し
※CPU初期化処理終了後は必ずカーネル初期化処理を呼び出してください。

図 2-9　HI1000/4 CPU 初期化ルーチン(H8SX/1650 用)

2.2.1 CPU 初期化ルーチンに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられた CPU 初期化ルーチンに関する質問についての回答を記述します。

《 FAQ 目次 》

(1) プログラムの転送

分類：CPU 初期化ルーチン

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

すべてのセクションを「ROM 化支援機能（Optlinker の ROM to RAM mappedsections）」を用いて ROM→RAM へ転送する場合、どのようにすればよいのか、具体的な方法を教えてください。

■回答

P_xxx セクション（コード部）を ROM から RAM へ転送して動作させる場合は、CPU 初期化ルーチンで「セクションの初期化」を行います。具体的には、P_xxx セクションの内容を R セクションにコピーします。

B_xxx セクションは RAM に配置されるセクションです。したがって、ROM 上に配置して RAM に転送する必要はありません。

ROM 化支援機能を使っていれば CPU 初期化ルーチンからの vsta_knl 呼び出されたカーネル初期化処理から、RAM に転送された内容で動作が始まります。

「プログラムの転送」に関する詳細は、コンパイラ アプリケーションノートの以下の内容を参照してください。

- SuperH™ RISC engine Series C/C++ Compiler Package：
  Q&Aの「プログラムのRAMへの転送」
- H8S,H8/300 Series C/C++ Compiler Package：
  Q&Aの「RAM上でプログラムを実行したい」

以下に、SH7770 を例に、プログラム転送処理を示します。

【次ページに続く】

【前ページからの続き】

■回答

```
/************************************************************************/
/*              HI7750/4 CPU initialize routine                         */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.       */
/*              and Renesas Solutions Corp. All Rights Reserved.        */
/*              HI7750/4(HS0775ITI41SR) V1.1.00                         */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7770_cpuini.c ;                                         */
/*  CPU type  = SH7770                                                  */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

/* extern  void    _INITSCT(void); */   /* section-initialize routine       */

#pragma section _hicpuini
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{
/*    ER  ercd; */

/*** Initialize Hardware Environment ***/
/*  ercd = vini_cac((ATR)(TCAC_IC_ENABLE|TCAC_OC_ENABLE|TCAC_P1_CB)); */

/*** Initialize Software Environment ***/
/*  _INITSCT();    */ /* Call section-initialize routine  */

    vsta_knl();                 /* Start kernel                     */
}
```

コメント"/* */"を外し、
「セクションの初期化」処理を呼び出します。

図 2-10　CPU 初期化ルーチンへの定義

```
/************************************************************************/
/*              HI7750/4 section initialize routine                     */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.       */
/*              and Renesas Solutions Corp. All Rights Reserved.        */
/*              HI7750/4(HS0775ITI41SR) V1.1.00                         */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7770_initsct.c  ;                                       */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"

extern int *B_BGN, *B_END, *D_BGN, *D_END, *D_ROM;
extern void _INITSCT(void);

#pragma section _hicpuini
/************************************************************************/
/*  NAME      = _INITSCT ;                                              */
/*  FUNCTION  = Section Initialize routine ;                            */
/************************************************************************/
void _INITSCT(void)
{
    register int *p, *q;
    for(p=B_BGN; p<B_END; p++)          /* 0 clear B-section            */
        *p = 0;
    for(p=D_BGN,q=D_ROM; p<D_END; p++, q++) /* Copy D-section -> R-section  */
        *p = *q;
}
```

図 2-11　INITSCT()処理の内容

(2) 初期スタックポインタの定義

分類：CPU 初期化ルーチン

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

構築で使用するプロジェクトファイルに定義されているスタックポインタは、カーネルが起動するまでのスタックポインタですか。

■回答

構築で使用するプロジェクトファイルに定義されているスタックポインタは、カーネル初期化処理を呼び出すまで、すなわち CPU 初期化ルーチンで使用するスタックポインタです。

CPU 初期化ルーチン起動時、指定したスタック領域がアクセス可能になっている必要があります。CPU 初期化ルーチンでスタックポインタを設定する前に、スタック領域の有効化（具体的には、外付けメモリ(SDRAM、SRAM など)を有効にするための BSC(バスステートコントローラ)の設定）などを行う必要があります。

CPU 初期化ルーチン終了後に起動するカーネル初期化処理では、コンフィギュレータで確保したカーネルスタックに切り替えます。

(3) 初期化処理後のハングアップ

分類：CPU 初期化ルーチン

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

CPU 初期化処理でハングアップするようなことはありますか。

■回答

CPU 初期化ルーチンの処理完了後、カーネル初期化処理がコールされますが、カーネル初期化処理完了後は、CPU 初期化ルーチンには戻って来ません。

カーネル初期化処理完了後は、初期起動タスクに制御が渡されます。

したがって、初期起動タスクに制御が渡って来ないケースでハングアップしている場合、以下のようなことが考えられますので、調査してください。

- カーネル初期化で、使用するスタック領域が不足しているため他の領域を破壊。
- カーネル初期化で、使用する RAM 領域がアクセスできない。
- ターゲットボードから不当割込み、未定義例外が発生している。
- カーネル初期化で、初期登録情報に誤りがあり、エラーが発生した。

CPU 初期化ルーチン起動からの処理概要については、本アプリケーションノート**「2.1 リセットからタスク起動までの処理概要」**をご参照ください。

2.3 カーネル初期化処理の概要

カーネル初期化処理で行う内容を以下に示します。

- カーネルスタックポインタへの切り替え
- カーネル管理領域（各種管理テーブル）の生成、初期化
- 初期登録オブジェクトの生成、初期化
- 初期化ルーチンの呼び出し　など

カーネル初期化処理は、カーネル動作に必要な情報の生成や初期化を行います。

2.3.1 初期化ルーチン

初期化ルーチンは、C言語関数として記述できます。
以下に初期化ルーチンのコーディング例を示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void inirtn(VP_INT exinf)
{

  /* 初期化ルーチンの処理 */

}
```

標準ヘッダファイルのインクルード

図 2-12　初期化ルーチンのコーディング例

初期化ルーチンは、アプリケーションプログラムに応じて作成する必要があります。
サンプル提供の初期化ルーチン（タイマ初期化ルーチン）を参考に、アプリケーションプログラムにあわせて作成してください。

2.3.2 マルチタスク環境への移行

「カーネル初期化処理」が完了すると、ディスパッチャが起動します。ディスパッチャでは、以下のようにタスクをスケジューリングします。

- 実行可能状態のタスクが存在する場合は、実行可能状態のタスクの中から、最も高い優先順位（最も高い優先度で、かつ、同一優先度のタスクの中で一番早く起動要求を受けたタスク）のタスクに CPU を割付けます。

- 実行可能状態のタスクが存在しない場合は、「システムアイドリング処理」に制御を移します。「システムアイドリング処理」では、実行可能状態に遷移するタスクが存在（起動）するまで、システムをアイドリング状態（中断状態）にします。

2.3.3 カーネル初期化処理に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたカーネル初期化処理に関する質問についての回答を記述します。

《 FAQ 目次 》

(1) カーネル作業領域の初期化

分類：カーネル初期化処理

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

CPU 初期化ルーチンで、カーネル作業領域を初期化（0 クリア）する必要はありますか。

■回答

CPU 初期化ルーチンで、カーネル作業領域を初期化する必要はありません。

カーネル作業領域（B_hiwrk セクション領域）は、カーネル初期化処理で、カーネル動作に必要な情報の生成や初期化を行います。

2.4 システムアイドリング処理の概要

実行すべきタスク（実行可能状態のタスク）が存在しない場合、カーネルはシステムアイドリング状態（具体的には、割込みマスクを解除し、無限ループ）となります。

2.4.1 SLEEP 命令を使用したシステムアイドリング処理

(1) HI7000/4 シリーズ

システムアイドリング処理でマイコンの持つ低消費電力機能を使用する場合は、最低優先度のタスクを作成し、そのタスクの中で各種設定を行い、SLEEP 命令を実行してください。

以下に、コーディング例を示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#pragma inline_asm(sleep)
static void sleep(void)
{
    sleep
}

#pragma noregsave(IdleTask)

void IdleTask(VP_INT exinf)
{

    while(1){
        /* 必要に応じてSBYCRレジスタの設定処理を行います */
        sleep();
    }

}
```

標準ヘッダファイルのインクルード

SLEEP命令実行のため、
#pragma inline_asmを使用します。

タスク起動時のレジスタ保証は不要なので、
#pragma noregsaveを指定します。

図 2-13 SLEEP 命令を使用したシステムアイドリング処理(HI7000/4 シリーズ用)

(2) HI2000/3

以下に、サンプル提供のシステムアイドリング処理を示します。

```
;************************************************************************
;*specifications ;                                                      *
;*name      = _H_SYSTEM_IDLE : HI2000/3 SYSTEM IDLING DEFINE    ;       *
;*function  =                                                   ;       *
;*notes     =                                                   ;       *
;*date      = 99/02/22                                          ;       *
;*author    = Hitachi, Ltd.                                     ;       *
;*attribute = public                                            ;       *
;*class     = system                                            ;       *
;*linkage   =                                                   ;       *
;*input     =                                                   ;       *
;*output    =                                                   ;       *
;*end of specifications ;                                               *
;************************************************************************
_H_SYSTEM_IDLE:
;       bra     $                       ;:forever loop
;
        sleep                           ;:sleep define
        bra     _H_SYSTEM_IDLE:8        ;:branch _H_SYSTEM_IDLE
;
```

図 2-14 SLEEP 命令を使用したシステムアイドリング処理(HI2000/3 用)

(3) HI1000/4

以下に、サンプル提供のシステムアイドリング処理を示します。

```
;***********************************************************************
;*                                                                     *
;*  HI1000/4 Version (uITRON V4.0)                                     *
;*  HI1000/4 kernel idle routine                                       *
;*                                                                     *
;*  Copyright (C) 1998,2003  Renesas Technology Corp. All right reserved *
;*                                                                     *
;***********************************************************************
        .program        _1650idle
        .heading        "### 1650idle.src : kernel idle routine ###"
        .section        P_hiidle,code,align=2
;
        .export         _KERNEL_H_SYSTEM_IDLE
;
;***********************************************************************
;*specifications ;                                                     *
;*name      = _KERNEL_H_SYSTEM_IDLE : HI1000/4 kernel idle routine     *
;*function  =                                                 ;        *
;*notes     =                                                 ;        *
;*input     =                                                 ;        *
;*output    =                                                 ;        *
;*end of specifications ;                                              *
;***********************************************************************
_KERNEL_H_SYSTEM_IDLE:
;       bra     $                       ;:forever loop
;
        sleep                           ;:sleep define
        bra     _KERNEL_H_SYSTEM_IDLE:8 ;:branch _KERNEL_H_SYSTEM_IDLE
;
;************************************;
        .end; of 1650idle.src
```

図 2-15　SLEEP 命令を使用したシステムアイドリング処理(HI1000/4 用)

2.4.2 システムアイドリング処理に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたシステムアイドリング処理に関する質問についての回答を記述します。

《 FAQ 目次 》

(1) アイドリング状態からの復帰

分類：システムアイドリング処理

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

slp_tsk を実行後、システムアイドリング状態のままです。原因を教えてください。

■回答

slp_tsk を実行後、カーネルがシステムアイドリング状態のままになる要因として以下に示す内容が考えられます。

(1) slp_tsk実行タスクが起床されない。
- slp_tsk 実行タスクを起床するタスクがない。
- slp_tsk 実行タスクを起床するタスクが起動しない。
- slp_tsk 実行タスクを起床する割込みハンドラが起動しない。

(2) slp_tsk実行タスク以外に、実行するタスクがない。

カーネルがシステムアイドリング状態に移行する要因は、実行可能状態のタスクがなくなったからです。

上記に示す通り、slp_tsk 実行タスクを起床するタスク（または割込みハンドラ）を作成し、システムアイドリング状態から復帰するようにしてください。

(2) アイドルリング状態における SLEEP 命令の実行

分類：システムアイドリング処理

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

カーネルがシステムアイドリング状態に遷移すると SLEEP 命令を利用してスリープしますが、このときの OS の処理について具体的に教えてください。

■回答

カーネルは、単に SLEEP 命令を実行しているのみです。

SLEEP 命令実行時、アプリケーション側で必要に応じて SBYCR レジスタの設定を行ってください。カーネルでは SBYCR レジスタの設定は行っておりません。

2.5 システム異常終了処理の概要

システムに異常が発生した場合、システム異常終了処理が起動されます。システム異常（システムダウン）となる要因を以下に示します。

- アプリケーションプログラムから強制的に呼び出した場合
- 初期登録オブジェクトの情報に誤り、矛盾が発生した場合
- カーネル内部で異常を検出した場合
- 未定義割込み、未定義例外を検出した場合

システム異常終了処理は、アプリケーションプログラムとしてユーザが作成します。サンプル提供ファイルを参考に、アプリケーションプログラムにあわせて作成してください。

システム異常終了処理には各種エラー情報が渡されます。デバッグ時に、エミュレータや ICE によるブレークポイントを設定することで、システム異常発生時の各種エラー情報がパラメータとして渡されますので、システム異常の解析に便利です。

システム異常終了処理に渡されるパラメータなどに関する詳細については、ご使用 OS のユーザーズマニュアル、または本アプリケーションノートの**「５章　デバッグ手法」**をご参照ください。

2.5.1 システム異常終了処理のサンプル例

(1) HI7000/4

以下に、サンプル提供のシステム異常終了処理を示します。

```
/*****************************************************************************/
/*              HI7000/4 System down routine                                 */
/*              Copyright (c)  Hitachi, Ltd. 2000.                           */
/*              Licensed Material of Hitachi, Ltd.                           */
/*              HI7000/4(HS0700ITI41SR) V1.0                                 */
/*****************************************************************************/
/*****************************************************************************/
/*  FILE      = 7604_sysdwn.c ;                                              */
/*****************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"
#include    "kernel_id.h"

#pragma section _hisysdwn
/*#pragma interrupt(_kernel_sysdwn)*/
/*****************************************************************************/
/*  NAME      = _kernel_sysdwn ;                                             */
/*  FUNCTION  = System down routine ;                                        */
/*****************************************************************************/
void    _kernel_sysdwn(type, ercd, inf1, inf2)
W type; /*system down type */
                /* type >=  1 : system down of user program                 */
                /* type ==  0 : initial information error                   */
                /* type == -1 : context error of ext_tsk                    */
                /* type == -2 : context error of exd_tsk                    */
                /* type == -16: undefined interrupt/exception               */
ER ercd; /* error code  */
                /* type >=  0 : error code of user program                  */
                /* type ==  0 : error code of initial information           */
                /* type == -1 : error code of ext_tsk                       */
                /* type == -2 : error code of exd_tsk                       */
                /* type == -16: interrupt vector number                     */
VW inf1; /* information-1  */
                /* type >=  0 : information of user program                 */
                /* type ==  0 : indicator of initial information error      */
                /* type == -1 : address of ext_tsk call                     */
                /* type == -2 : address of exd_tsk call                     */
                /* type == -16: address of interrupt occurrence             */
VW inf2; /* information-2  */
                /* type >=  0 : information of user program                 */
                /* type ==  0 : number of error initial information         */
                /* type == -16: SR of interrupt occurrence                  */
{
    set_imask(SR_IMS15); /* mask all interrupt                              */
    while(TRUE);    /* endless loop                                         */
}
```

図 2-16 システム異常終了処理(HI7000/4)

(2) HI7700/4、HI7750/4

以下に、サンプル提供のシステム異常終了処理を示します。

```
/************************************************************************/
/*              HI7700/4 System down routine                            */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.       */
/*              and Renesas Solutions Corp. All Rights Reserved.        */
/*              HI7700/4(HS0770ITI41SR) V1.0                            */
/************************************************************************/
/************************************************************************/
/*  FILE      = 7708_sysdwn.c ;                                         */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"
#include    "kernel_id.h"

/************************************************************************/
/*              environment data                                        */
/************************************************************************/
#define MD_BIT 0x40000000  /* SR.MD bit                                 */

#pragma section _hisysdwn
/*#pragma interrupt(_kernel_sysdwn)*/
/************************************************************************/
/*  NAME      = _kernel_sysdwn ;                                        */
/*  FUNCTION  = System down routine ;                                   */
/************************************************************************/
void    _kernel_sysdwn(type, ercd, inf1, inf2)
W type; /*system down type */
                /* type >=  1 : system down of user program            */
                /* type ==  0 : initial information error              */
                /* type == -1 : context error of ext_tsk               */
                /* type == -2 : context error of exd_tsk               */
                /* type == -16: undefined interrupt/exception          */
ER ercd; /* error code  */
                /* type >=  0 : error code of user program            */
                /* type ==  0 : error code of initial information      */
                /* type == -1 : error code of ext_tsk                  */
                /* type == -2 : error code of exd_tsk                  */
                /* type == -16: interrupt vector number                */
VW inf1; /* information-1  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : indicator of initial information error */
                /* type == -1 : address of ext_tsk call                */
                /* type == -2 : address of exd_tsk call                */
                /* type == -16: address of interrupt occurrence        */
VW inf2; /* information-2  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : number of error initial information    */
                /* type == -16: SR of interrupt occurrence             */
{
    set_cr(MD_BIT | (SR_IMS15 << 4));  /* mask all interrupt           */
    while(TRUE);    /* endless loop                                    */
}
```

図 2-17　システム異常終了処理(HI7700/4、HI7750/4)

(3) HI2000/3

以下に、サンプル提供のシステム異常終了処理を示します。

```
;************************************************************************
;*specifications ;                                                      *
;*name      = _HIPRG_ABNOML : abnormal quit handler             ;       *
;*function  =                                                   ;       *
;*notes     =                                                   ;       *
;*date      = 99/02/22                                          ;       *
;*author    = Hitachi, Ltd.                                     ;       *
;*attribute = public                                            ;       *
;*class     = system                                            ;       *
;*linkage   =                                                   ;       *
;*input     =                                                   ;       *
;*output    =                                                   ;       *
;*end of specifications ;                                               *
;************************************************************************
_HIPRG_ABNOML:
        orc     #HIDEF_IMASK_CCR:8,ccr  ;:interrupt mask for CCR register
        orc     #HIDEF_IMASK_EXR:8,exr  ;:interrupt mask for EXR register
        bra     $                       ;:forever loop
;
```

図 2-18　システム異常終了処理(HI2000/3)

(4) HI1000/4

以下に、サンプル提供のシステム異常終了処理を示します。

```
;************************************************************************;
;*NAME = vsys_dwn                                                       ;*;
;*FILE = vsys_dwn.src                                                   ;*;
;*FUNC = System down routine                                            ;*;
;*NOTE =                                                                ;*;
;*INPU = none       :                                                   ;*;
;*OUTP = none       :                                                   ;*;
;************************************************************************;
;
        .section        P_hisysdwn,code,align=2
;
        .export         _vsys_dwn
        .export         _ivsys_dwn
_vsys_dwn:
_ivsys_dwn:
        bra   _vsys_dwn:8
        rts
;
        .end; of vsys_dwn.src
```

図 2-19　システム異常終了処理(HI1000/4)

2.5.2 システム異常終了処理に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたシステム異常終了処理に関する質問についての回答を記述します。

《 FAQ 目次 》

(1) システムダウン要因

分類：システム異常終了処理

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

初期化処理終了後、システムダウンします。システムダウンの要因を調べる方法を教えてください。

■回答

システムダウンになる条件を以下に示します。

- アプリケーションプログラムから強制的に呼び出した場合
- 初期登録オブジェクトの情報に誤り、矛盾が発生した場合
- カーネル内部で異常を検出した場合
- 未定義割込み、未定義例外を検出した場合

システム異常終了処理の先頭に、ブレークポイントを設定することで、システム異常時のパラメータなどからシステムダウンになった要因を知ることができます。

システム異常終了処理に渡されるパラメータなどに関する詳細については、ご使用 OS のユーザーズマニュアル、または本アプリケーションノートの**「５章　デバッグ手法」**をご参照ください。

2.6 アプリケーションプログラムの種類

HI シリーズ OS を使用したシステムを開発する際、以下に示すアプリケーションプログラムを準備する必要があります。

表 2-2 アプリケーションプログラムの種類と準備

◎：必須、△：必要に応じて作成

項番	種　類	必要性	備　考
1	タスク	◎	
2	割込みハンドラ	◎	
3	CPU 初期化ルーチン	◎	
4	システム異常終了処理ルーチン	◎	
5	システムアイドリング処理ルーチン	◎	
6	初期化ルーチン	△	
7	タイマ割込みルーチン (タイマ初期化ルーチン含む)	*1	
8	タスク例外処理ルーチン	△	*2
9	拡張サービスコールルーチン	△	*2
10	CPU 例外ハンドラ	△	*3
11	周期ハンドラ	△	
12	アラームハンドラ	△	*2
13	オーバランハンドラ	△	*2

【注】 *1 時間管理機能を使用しない場合は不要です。
*2 HI7000/4 シリーズでサポートしている機能です。HI2000/3、および HI1000/4 ではサポートされていません。
*3 HI7000/4 シリーズおよび HI1000/4 でサポートしている機能です。HI2000/3 ではサポートされていません。

上記アプリケーションプログラムとシステム状態、発行可能なサービスコールの種類を以下に示します。

表 2-3 アプリケーションプログラムとシステム状態

項番	種　類	システム状態	サービスコールの種類
1	タスク	タスクコンテキスト	タスクコンテキスト用
2	割込みハンドラ	非タスクコンテキスト	非タスクコンテキスト用
3	初期化ルーチン	非タスクコンテキスト	非タスクコンテキスト用
4	タスク例外処理ルーチン	タスクコンテキスト	タスクコンテキスト用
5	拡張サービスコールルーチン	発行元コンテキスト*1	発行元コンテキスト *1
6	CPU 例外ハンドラ	*2	*3
7	周期ハンドラ	非タスクコンテキスト	非タスクコンテキスト用
8	アラームハンドラ	非タスクコンテキスト	非タスクコンテキスト用
9	オーバーランハンドラ	非タスクコンテキスト	非タスクコンテキスト用

【注】 *1 呼出しもとのコンテキストを引き継ぎます。
*2 HI7000/4 シリーズでは「発行元コンテキスト」、HI1000/4 では「非タスクコンテキスト」となります。HI2000/3 ではサポートされていません。
*3 発行可能なサービスコールは、ユーザーズマニュアルを参照ください。

2.6.1 タスク作成例

タスクは、C 言語関数として記述します。タスクを終了する場合の注意事項を以下に示します。

表 2-4 タスク終了時のサービスコールと注意事項

HI シリーズ OS	サービスコール	注意事項
HI7000/4 シリーズ	ext_tsk()サービスコール、 または exd_tsk()サービスコール	サービスコール省略可（省略した場合は、ext_tsk()サービスコールと同じ）
HI2000/3	ext_tsk()システムコール	サービスコール省略不可（必ず ext_tsk()システムコールで終了してください）。 タスクから呼び出し元にリターンした場合、システムの正常な動作は保証されません。
HI1000/4	ext_tsk()サービスコール	サービスコール省略可（省略した場合は、ext_tsk()サービスコールと同じ）

タスク起動時の各コンテキストレジスタ値については、ご使用の HI シリーズ OS ユーザーズマニュアルをご参照ください。

タスクの記述例を以下に示します。

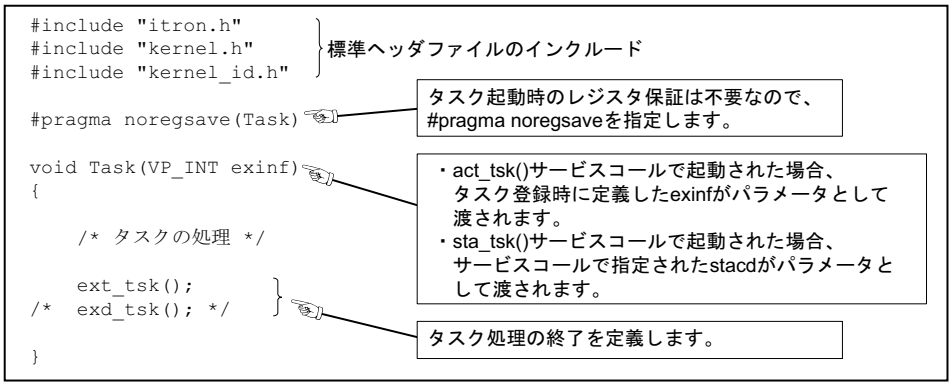


図 2-20 タスクのコーディング例

【注】 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

2.6.2 割込みハンドラ作成例

割込みハンドラの記述例を、HI シリーズ OS 毎に示します。

(1) HI7000/4 シリーズ使用時の割込みハンドラコーディング例

割込みハンドラのコーディング例を以下に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void Inh(void)
{

    /* 割込みハンドラの処理*/

}
```

標準ヘッダファイルのインクルード

割込みハンドラは、
void型の関数として記述します。

図 2-21 割込みハンドラコーディング例(HI7000/4 シリーズ)

【注】(1) インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。
(2) コプロセッサを使用する場合は、その全レジスタを保証しなければなりません。

IRL 割込みを使用すると、1 つのベクタテーブルに、異なるレベルの 2 つの割込み要因を割当てられます。IRL 割込みを使用する場合は、割込みハンドラを以下のように記述します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

#define I_HILEVEL 15

void vec071_handler14(void)
{
    /* IRL14割込み処理 */
}

void vec071_handler15(void)
{
    /* IRL15割込み処理 */
}

void vec071(void)
{

    if((get_imask()) == I_HILEVEL)
        vec071_handler15();
    else
        vec071_handler14();

}
```

標準ヘッダファイルのインクルード

高い方のレベルを定義します。

各レベルの割込みハンドラを記述します。

割込み要因へ登録する割込みハンドラを、
void型の関数として記述します。

図 2-22 IRL 割込み使用時の割込みハンドラコーディング例(HI7000/4 シリーズ)

HI7000/4 のダイレクト割込みハンドラ使用時の注意事項について以下に示します。

- 割込みが発生するとカーネルの介入なしに起動されます。
- サービスコールを発行できません。

ダイレクト割込みハンドラのコーディング例を以下に示します。

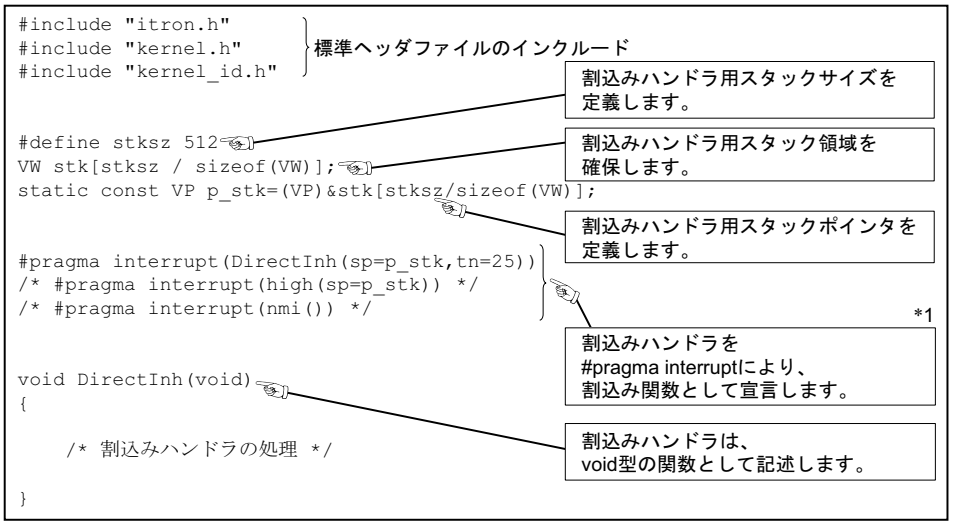


図 2-23　ダイレクト割込みハンドラコーディング例(HI7000/4)

【注】*1　#pragma interrupt 記述は、割込み仕様として以下のように指定してください。

- スタック切り替え指定（sp=）
  ※NMI割込みハンドラでは、スタック切り替えは行わないで下さい。

- トラップリターン指定（tn=25）
  カーネル割込みマスクレベルより低いレベルの割込みハンドラは、“tn=25”を指定してください。カーネル割込みマスクレベルよりも高いレベル(NMIを含む)の割込みハンドラは、RTE命令で終了するので、トラップリターン指定は行わないでください。

ダイレクト割込みハンドラは、HI7700/4、および HI7750/4 ではサポートされていません。

(2) HI2000/3、HI1000/4 使用時の割込みハンドラコーディング例

割込みハンドラでは、割込み発生時レジスタを保証しなければなりません。割込みハンドラの作成は以下の手順で行います。

表 2-5 割込みハンドラ作成手順

項番	処 理	内 容
1	割込みハンドラで使用する レジスタの退避	・スタックポインタの保存 ※割込みハンドラ専用スタック領域にスタックポインタを変更(割込みハンドラでスタックを使用しない場合は不要) ・レジスタの内容保存
2	割込み処理	割込みハンドラで行う処理
3	割込みハンドラで使用した レジスタの復帰	・レジスタの内容復帰 ※スタックポインタの変更(割込みハンドラでスタックを使用しない場合は不要)
4	割込みハンドラ終了処理	ret_int ルーチンの呼び出し(カーネル割込みマスクレベル以下の場合)、または RTE 命令実行(カーネル割込みマスクレベルより高い場合)

割込みハンドラは、C コンパイラの割込み関数作成機能（#pragma interrupt）を用いて記述します。割込みハンドラのコーディング例を以下に示します。

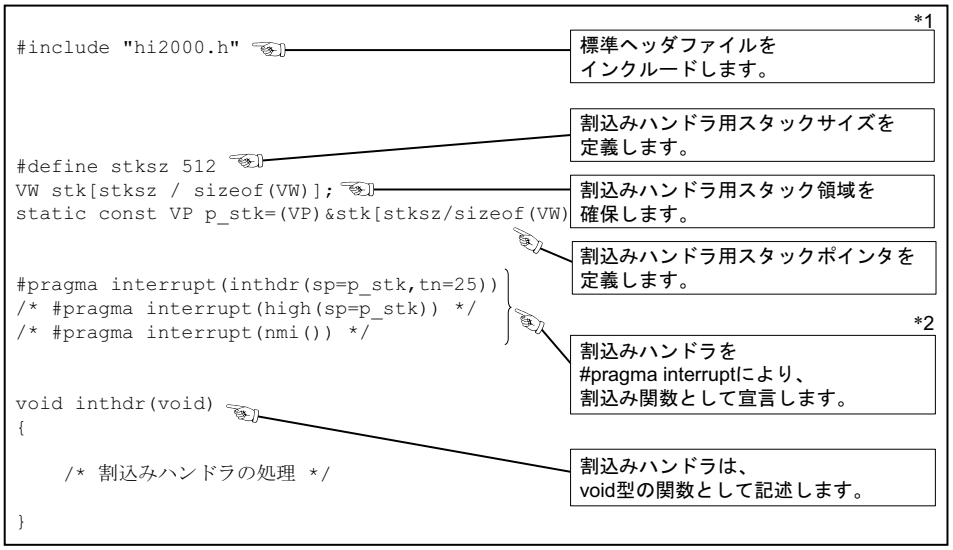


図 2-24 割込みハンドラのコーディング例(HI2000/3、HI1000/4)

【注】*1 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

*2 割込みの仕様として「スタック切り替え指定」と「割込み関数終了指定」を行います。詳細については、ご使用 OS のユーザーズマニュアルを参照してください。

2.6.3 CPU 初期化ルーチン作成例

HI2000/3、および HI1000/4 では、サンプル提供ファイルはアセンブリ言語記述です。C 言語記述の CPU 初期化ルーチンを使用する場合、アセンブリ言語記述の CPU 初期化ルーチンに C 言語記述の CPU 初期化ルーチン呼び出しを追加します。

アセンブリ言語記述の CPU 初期化ルーチンの変更例、および C 言語記述の CPU 初期化ルーチンの記述例を HI2000/3 と HI1000/4 を分けて示します。

(1) HI2000/3 の記述例

```
_H_2S_CPUINI:
        mov.l   #CPUINI_SP:32,sp        ;:get CPUINI_SP
        mov.b   @SYSCR:32,r0L           ;:get SYSCR
        and.b   #low~(INTM0|INTM1):8,r0L;:clear interrupt mode bit
        or.b    #low (INTM0|INTM1):8,r0L;:set interrupt mode = 3
        mov.b   r0L,@SYSCR:32           ;:set SYSCR
;
        mov.b   @MSTPCRH:32,r0L         ;:get MSTPCRH
        and.b   #low TPU:8,r0L          ;:set TPU bit off
        mov.b   r0L,@MSTPCRH:32         ;:set MSTPCRH
;
        .aifdef DX
        jsr     @_HI_DEAMON_INI         ;:call to init deamon code
        .aendi
;
        bsr     @_h_cpuini_c            ;:call to C-language initialize routine
;
        jmp     @_H_2S_INIT             ;:goto HI2000/3 initialize module
;
```

C言語記述のCPU初期化ルーチンの呼出し
※アセンブラ記述のCPU初期化ルーチン終了後、
C言語記述のCPU初期化ルーチンを
呼び出してください。

【注】本例では、C言語記述のCPU初期化ルーチンを「h_cpuini_c」と仮定しています。

図 2-25 アセンブリ言語記述の CPU 初期化ルーチン変更例(HI2000/3)

```
void h_cpuini_c(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Software Environment ***/

//    _INITSCT();                   /* Call section-initialize routine  */

}
```

図 2-26 C 言語記述の CPU 初期化ルーチン記述例(HI2000/3)

(2) HI1000/4 の記述例

```
_KERNEL_H_CPUINI:
        mov.l   #_KERNEL_HI_OS_SP:32,sp ;:SP <- OS stack
        mov.l   #VBR_ADR,er0            ;:
        ldc.l   er0,vbr                 ;:set VBR address
        mov.l   #h'ffffff00,er0         ;:initial SBR
        ldc.l   er0,sbr                 ;:initial SBR
;
;       mov.w   #h'00ff,@ABWCR:32       ;:set ABWCR
;       mov.w   #h'0000,@ASTCR:32       ;:set ASTCR
;       mov.w   #h'0000,@WTCRA:32       ;:set WTCRA
;       mov.w   #h'0000,@WTCRB:32       ;:set WTCRB
;
        mov.b   #INTM1,r0L              ;:set interrupt mode 2
        mov.b   r0L,@INTCR:32           ;:set INTCR
;
        mov.w   @MSTPCRA:32,r0          ;:get MSTPCRA
        and.w   #MSTPA0:16,r0           ;:set TPU bit off
        mov.w   r0,@MSTPCRA:32          ;:set MSTPCRA
;
        jmp     @_h_cpuini              ;:goto  _h_cpuini
;
```

C言語記述のCPU初期化ルーチンの呼出し
※アセンブラ記述のCPU初期化ルーチン終了後、
C言語記述のCPU初期化ルーチンへ分岐してください。

【注】本例では、C言語記述のCPU初期化ルーチンを「h_cpuini」と仮定しています。

図 2-27 アセンブリ言語記述の CPU 初期化ルーチン変更例(HI1000/4)

```
void h_cpuini(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Software Environment ***/

//    _INITSCT();                   /* Call section-initialize routine  */

    vsta_knl();                     /* Start kernel                     */
}
```

カーネル初期化処理の呼び出し
※CPU初期化処理終了後は必ず
カーネル初期化処理を呼び出してください。

図 2-28 C言語記述の CPU 初期化ルーチン記述例(HI1000/4)

合わせて「2.2 CPU 初期化ルーチンの概要」もご参照ください。

2.6.4 システム異常終了処理作成例

HI2000/3、および HI1000/4 では、サンプル提供ファイルはアセンブリ言語記述になっています。C言語記述でシステム異常終了処理を作成する場合、以下に示すコーディング例を参考に作成してください。

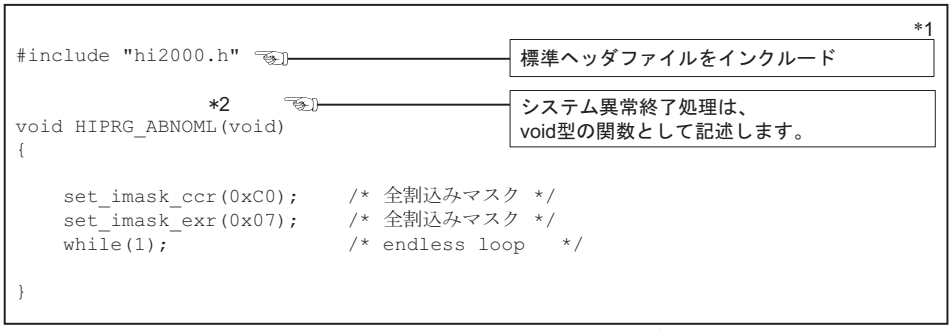


図 2-29 システム異常終了処理のコーディング例(HI2000/3)

【注】*1 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

*2 関数名はカーネルからも参照しているため、HI2000/3 では「HIPRG_ABNOML」、HI1000/4 では「vsys_dwn」としてください。

2.6.5 システムアイドリングルーチン作成例

HI2000/3、および HI1000/4 では、サンプル提供ファイルはアセンブリ言語記述になっています。C言語記述でシステムアイドリングルーチンを作成する場合、以下に示すコーディング例を参考に作成してください。

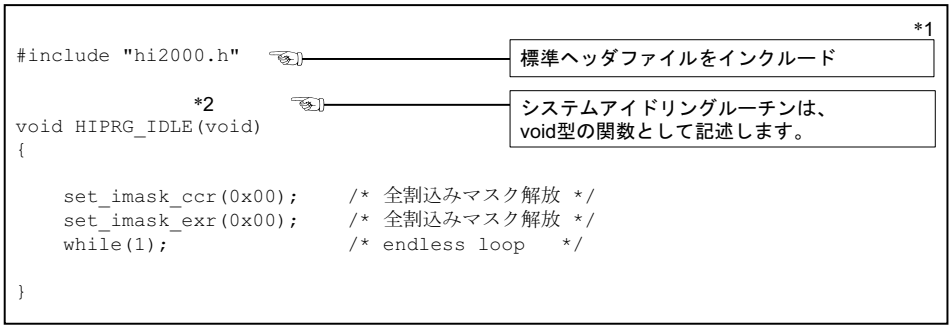


図 2-30 システムアイドリングルーチンのコーディング例(HI2000/3)

【注】*1 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

*2 関数名はカーネルからも参照しているため、HI2000/3 では「HIPRG_IDLE」、HI1000/4 では「KERNEL_H_SYSTEM_IDLE」としてください。

2.6.6 初期化ルーチン作成例

HI2000/3、および HI1000/4 では、サンプル提供ファイルはアセンブリ言語記述になっています。C 言語記述で初期化ルーチンを作成する場合、以下に示すコーディング例を参考に作成してください。

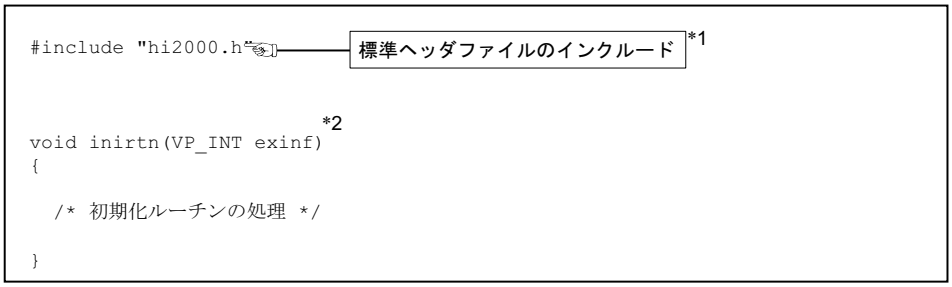


図 2-31 初期化ルーチンのコーディング例

【注】*1 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

*2 HI2000/3 では初期化ルーチンへの拡張情報（exinf）は渡されませんので、これを受け取る記述はしないでください（引数はありません）。

2.6.7 タイマ割込みルーチン作成例

タイマ割込みルーチンのコーディング例を以下に示します。

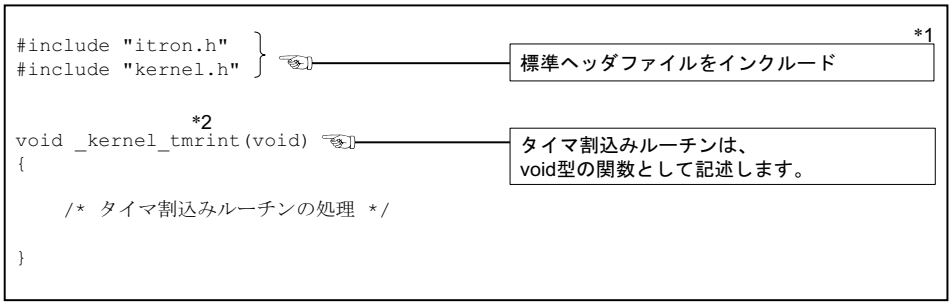


図 2-32 タイマ割込みルーチンのコーディング例

【注】*1 インクルードする標準ヘッダファイルについては、ご使用 OS のユーザーズマニュアルをご参照ください。

*2 関数名はカーネルからも参照しているため、以下のようにしてください。

HI シリーズ OS	関数名
HI7000/4 シリーズ	_kernel_tmrint 固定
HI2000/3	ユーザ任意
HI1000/4	_KERNEL_H_TIM 固定

2.6.8 タスク例外処理ルーチン作成例

タスク例外処理ルーチンは、HI7000/4 シリーズのみがサポートしている機能です。以下にコーディング例を示します。

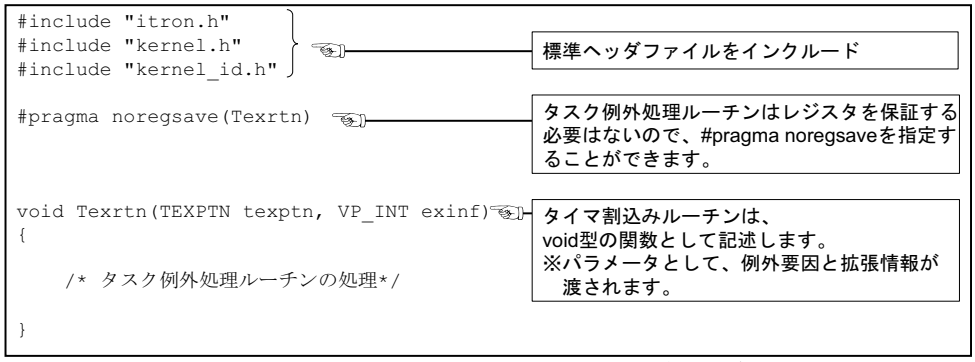


図 2-33 タスク例外処理ルーチンのコーディング例

2.6.9 拡張サービスコールルーチン作成例

拡張サービスコールルーチンは、HI7000/4 シリーズのみがサポートしている機能です。以下にコーディング例を示します。

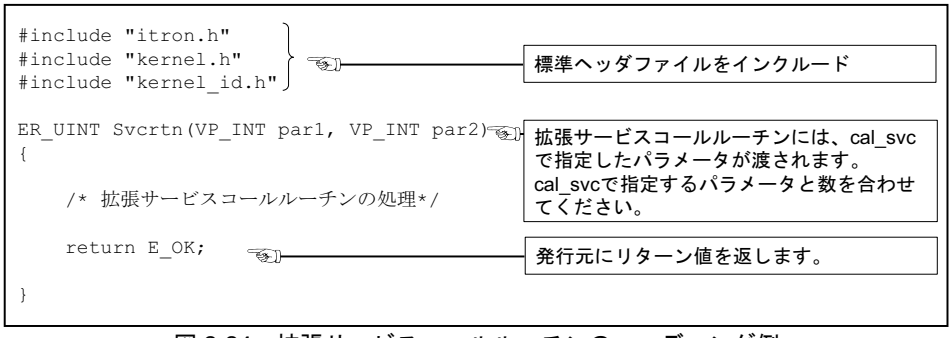


図 2-34 拡張サービスコールルーチンのコーディング例

2.6.10 CPU 例外ハンドラ作成例

拡張サービスコールルーチンは、HI7000/4 シリーズ、および HI1000/4 がサポートしている機能です。以下にコーディング例を示します。

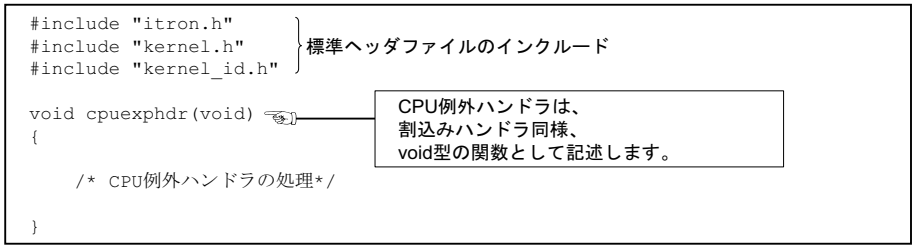


図 2-35 CPU 例外ハンドラのコーディング例

2.6.11 タイムイベントハンドラ作成例

(1) 周期ハンドラ作成例

(a) HI7000/4 シリーズ、HI1000/4 使用時の周期ハンドラコーディング例

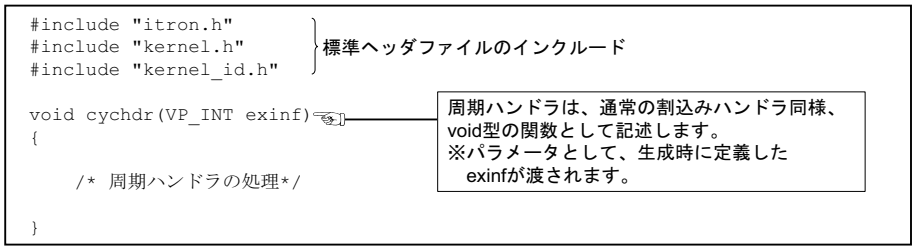


図 2-36 周期ハンドラのコーディング例(HI7000/4 シリーズ、HI1000/4)

(b) HI2000/3 使用時の周期起動ハンドラコーディング例

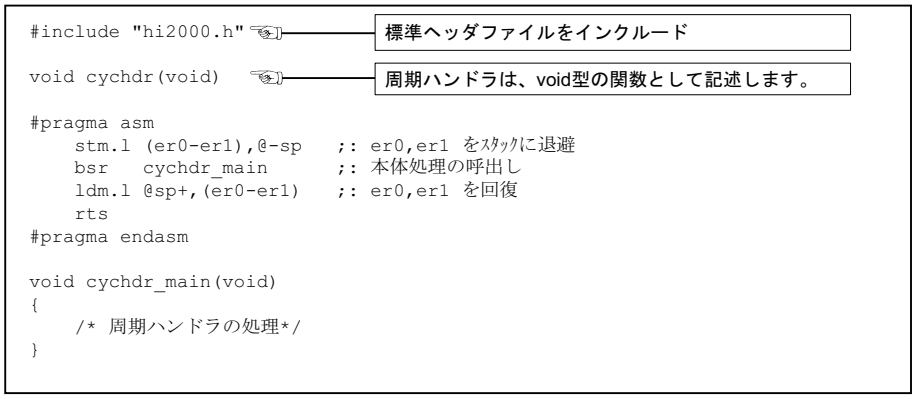


図 2-37 周期起動ハンドラのコーディング例(HI2000/3)

(2) アラームハンドラ（HI7000/4 シリーズサポート機能）作成例

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void almhdr(VP_INT exinf)
{

    /* アラームハンドラの処理*/

}
```

標準ヘッダファイルのインクルード

アラームハンドラは、通常の割込みハンドラ同様、
void型の関数として記述します。
※パラメータとして、生成時に定義した
exinfが渡されます。

図 2-38 アラームハンドラのコーディング例(HI7000/4 シリーズのみ)

(3) オーバーランハンドラ（HI7000/4 シリーズサポート機能）作成例

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

void ovrhdr(ID tskid, VP_INT exinf)
{

    /* オーバーランハンドラの処理*/

}
```

標準ヘッダファイルのインクルード

オーバーランハンドラは、通常の割込みハンドラ同様、
void型の関数として記述します。
※パラメータとして、起動の原因となったtskidと、
生成時に定義したexinfが渡されます。

図 2-39 オーバーランハンドラのコーディング例(HI7000/4 シリーズのみ)

2.7 アプリケーションプログラムの開発手順

HI シリーズ OS を使用したシステムを開発する際の手順を以下に示します。

(1) 新規にシステムを開発
(2) 既存システムの資産を流用して開発

上記(1)の場合は、**「2.5 アプリケーションプログラムの種類」**に示すプログラムを作成し、最終的なシステムへと開発を進めていきます。

この場合、アプリケーションプログラムはすべてはじめから作成するため、HI シリーズ OS を組み込む最適なプログラムを開発することができます。

(1) 機能分割は「トップダウン」で分割

機能分割できなくなるまで実施します。これにより、「並列かつ同時処理」が可能となる機能が定義されます。分割された機能を、タスク、または、割込みハンドラとして定義します。

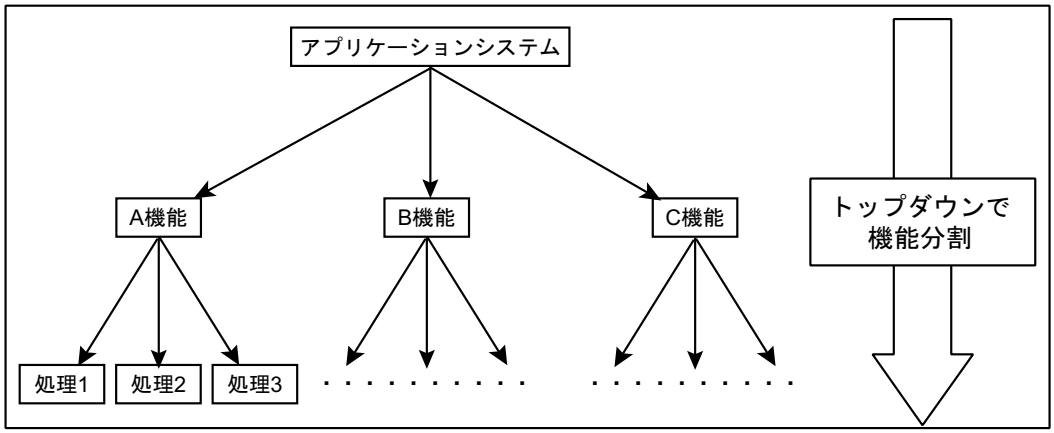


図 2-40 トップダウンによる機能分割

(2) 細かく分割した後、同じ処理を行うタスク（機能）同士をまとめる

同じ処理を行っているタスク同士をまとめる作業を「タスク融合」といいます（割込みハンドラの場合、割込み要因ごとにハンドラを定義するため、融合は発生しない）。これにより、機能的依存度が排除されたタスク（機能）が定義できます。

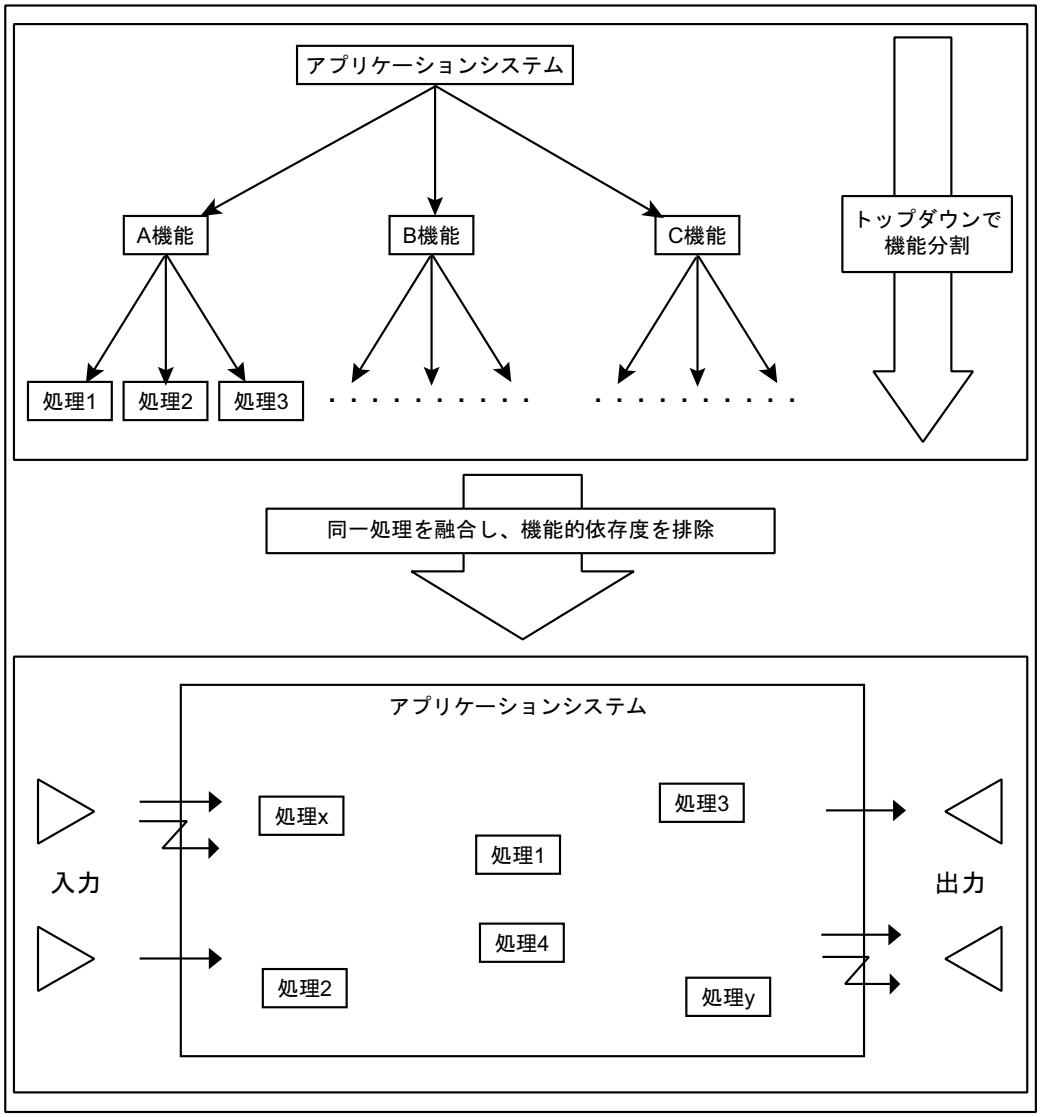


図 2-41 同一機能同士の融合による機能的依存度の排除

これらの作業が完了した後、タスク間、またはタスクと割込みハンドラ間のインタフェース（同期、通信の機能）に、HI シリーズ OS のオブジェクトを割付けていく。

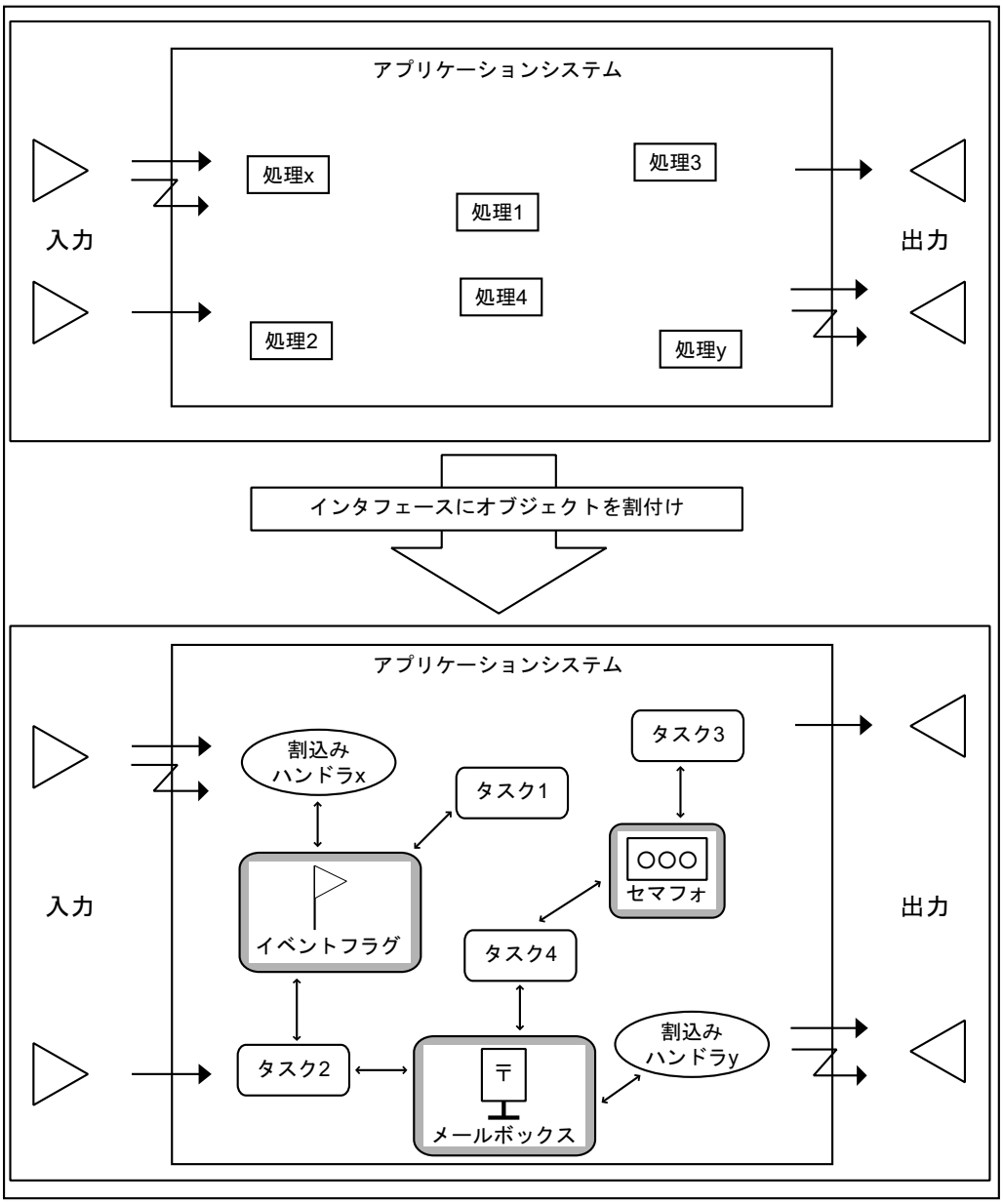


図 2-42　ITRON オブジェクトによるインタフェースの割り付け例

これらの作業を行うことで、既存製品に搭載している RTOS を組み込んでいないアプリケーションプログラムへ HI シリーズ OS を組み込むことができます。

3. 構　築

3.1 構築手順の概要

HI シリーズ OS を使用したシステムの構築手順を以下に示します。

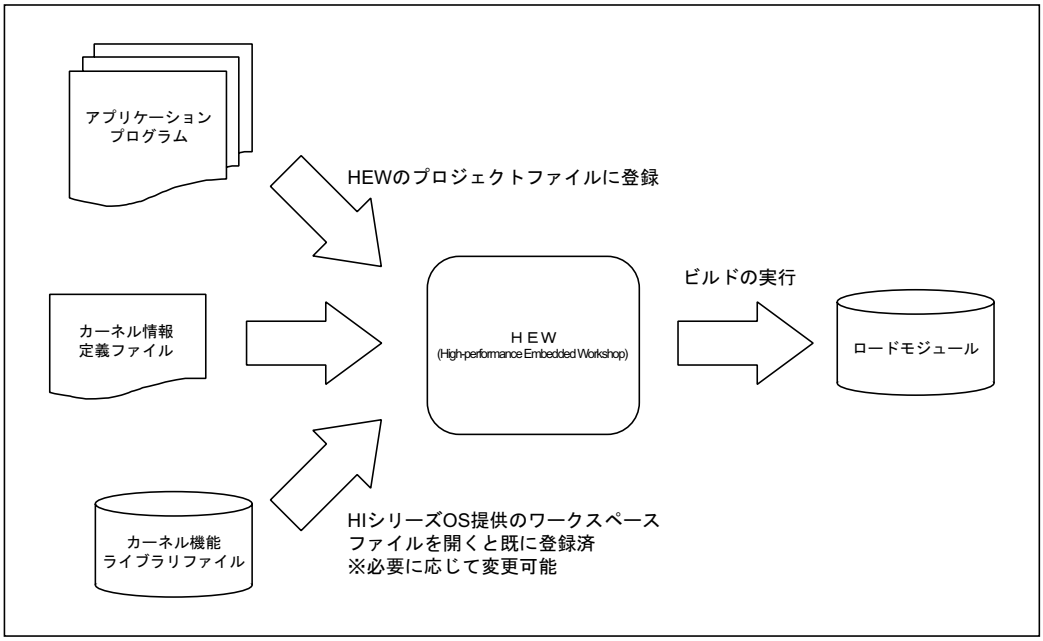


図 3-1　構築手順の概要

システムの構築は、ユーザが作成したアプリケーションプログラムと、カーネル情報定義ファイル（セットアップテーブル、またはコンフィギュレーションファイル）、HI シリーズ OS が提供しているカーネル機能ライブラリファイルを、HEW を使用してロードモジュールを生成します。

ユーザが作成するアプリケーションプログラムについては、**「2.6　アプリケーションプログラムの種類」**を参照してください。

カーネル情報定義ファイル（セットアップテーブル、またはコンフィギュレーションファイル）については、**「3.2　カーネル環境の定義」**を参照してください。

HI シリーズ OS が提供しているカーネル機能ライブラリファイルについては、ご使用の HI シリーズ OS ユーザーズマニュアルをご参照ください。

HEW については、ご使用の Compiler Package オンラインヘルプ、またはユーザーズマニュアルをご参照ください。

システムを構築する方法には、以下に示す二つの方式があります。

表 3-1　システム構築方式

構築方式	概　要	サポート OS
一括リンク方式 *1	カーネルとコンフィギュレーションファイル、アプリケーションプログラムを 1 つのロードモジュール（これを「一括ロードモジュール」と呼びます）にする方式	HI7000/4 シリーズ、HI2000/3、HI1000/4
分割リンク方式 *2	カーネルのコード部分とデータ部分を別々のロードモジュールにする方式	HI7000/4 シリーズ
	カーネルのコード部分を「カーネルロードモジュール」と呼び、カーネルロードモジュールのリンク単位を「カーネル側」と呼ぶ	
	カーネルのデータ部分を「カーネル環境ロードモジュール」と呼び、カーネルロードモジュールのリンク単位を「カーネル環境側」と呼ぶ	

【注】 *1　アプリケーションは、一括ロードモジュールに含めることも、別ロードモジュール（これを「アプリケーションロードモジュール」と呼びます）にすることも可能。

*2　アプリケーションは、カーネルロードモジュール、カーネル環境ロードモジュールに含めることも、別のアプリケーションロードモジュールにすることも可能。

以下に、一括リンク方式と、分割リンク方式の概要を各々示します。

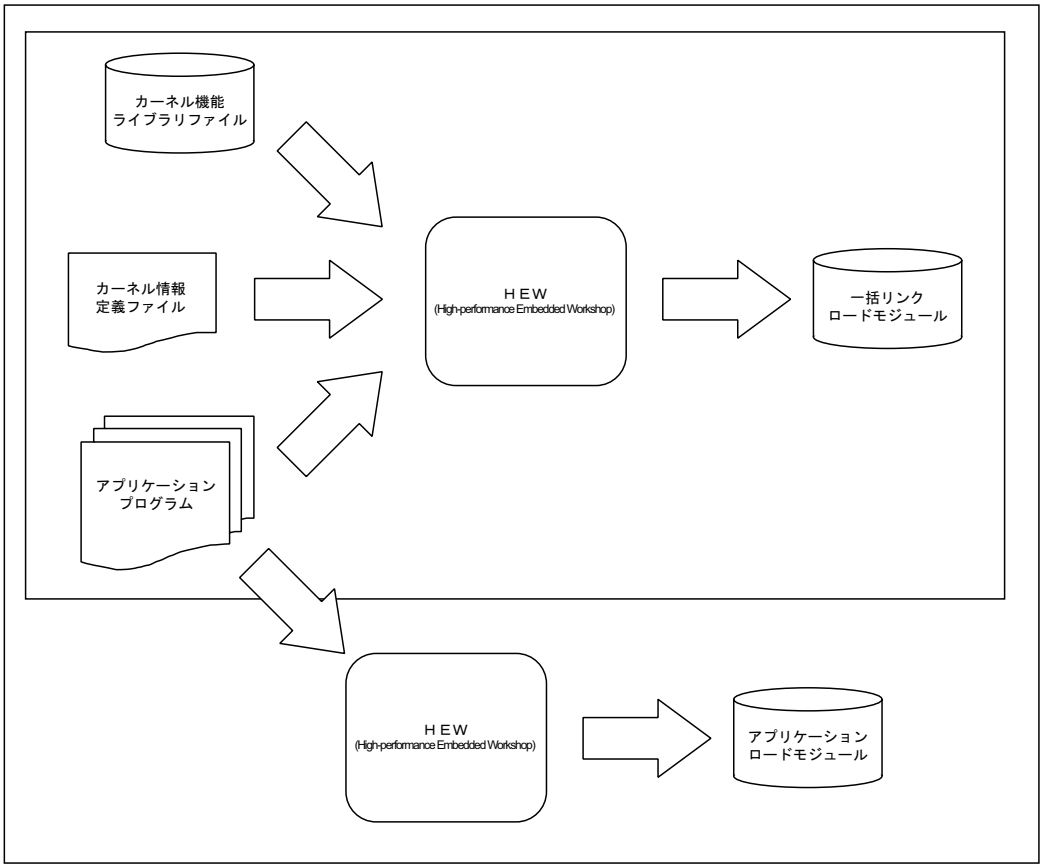
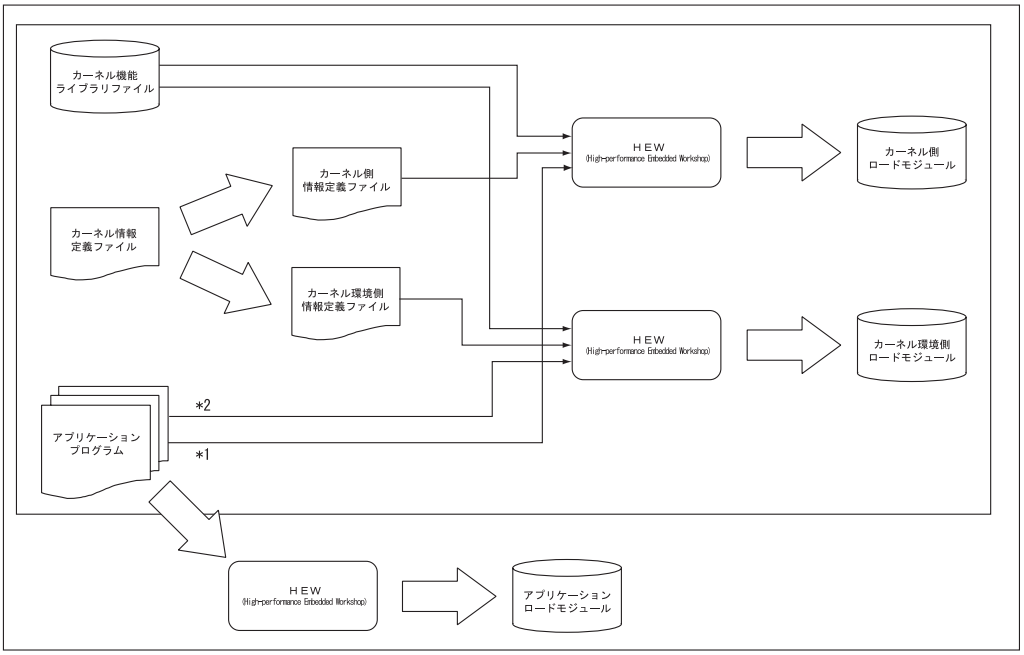


図 3-2　一括リンク方式の概要



【注】 ＊1 コンフィギュレータでカーネル側に指定したアプリケーションプログラムは必須です。
＊2 コンフィギュレータでカーネル環境側に指定したアプリケーションプログラムは必須です。

図 3-3　分割リンク方式の概要

一括リンク方式に対する分割リンク方式の長所、短所を以下に示します。

《長所》

- カーネル単独でロードモジュールを生成することができるので、アプリケーションファイル、カーネル環境側ファイルの変更のたびに、ロードモジュールを再生成する必要がありません。
- カーネルロードモジュールを ROM 化した後にもカーネルロードモジュールを更新せずに、タスクの最大数(CFG_MAXTSKID)などといったコンフィギュレータパラメータを変更してカーネル環境ロードモジュールを再生成することができます。

《短所》

- カーネルはカーネル環境ファイル情報を参照して動作するため、カーネル環境ファイル情報の配置アドレスをあらかじめ決定しておき、リンク時にこのアドレスを定義する必要があります。
- このアドレスは、カーネルロードモジュールを再リンクしない限り変更する事は出来ません。

3.2　カーネル環境の定義

カーネル環境の定義は、セットアップテーブルとコンフィギュレータによる 2 種類があります。

- HI7000/4 シリーズ、HI1000/4　：　コンフィギュレータでカーネル環境を定義
- HI2000/3　：　セットアップテーブルでカーネル環境を定義

それぞれの定義方法を以下に示します。

3.2.1　コンフィギュレータによる定義（HI7000/4 シリーズ、HI1000/4）

コンフィギュレータの出力ファイル（カーネル環境定義ファイル。以下、コンフィギュレーションファイルと称す）を以下に示します。

表 3-2　コンフィギュレータの出力ファイル(HI7000/4 シリーズ)

項番	ファイル名	内容	備考
1	kernel_def_main.h	組み込むサービスコールなどのカーネルの機能定義など	
2	kernel_def_inidata.def	カーネルロードモジュール側のオブジェクト初期登録情報など	
3	kernel_def_vct.inc	ベクタ情報(アセンブラ記述)	HI7000/4 のみ
4	kernel_cfg_main.h	最大タスク ID などのカーネル環境情報定義など	
5	kernel_cfg_inidata.def	カーネル環境ロードモジュール側のオブジェクト初期登録情報など	
6	kernel_id.h	kernel_cfg_inidata.def に対する ID 番号自動割付け結果	
7	kernel_macro.h	カーネル構成マクロを定義したヘッダファイル	

表 3-3　コンフィギュレータの出力ファイル(HI1000/4)

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

上記ファイルの詳細については、HI シリーズ OS の「HI7000/4 シリーズユーザーズマニュアル」、または「HI1000/4 ユーザーズマニュアル」をご参照ください。

(1)　コンフィギュレータの操作概要

コンフィギュレータの構成や操作について、HI7000/4 を例に説明します。

(a)　コンフィギュレータ画面

コンフィギュレータ起動画面を以下に示します。

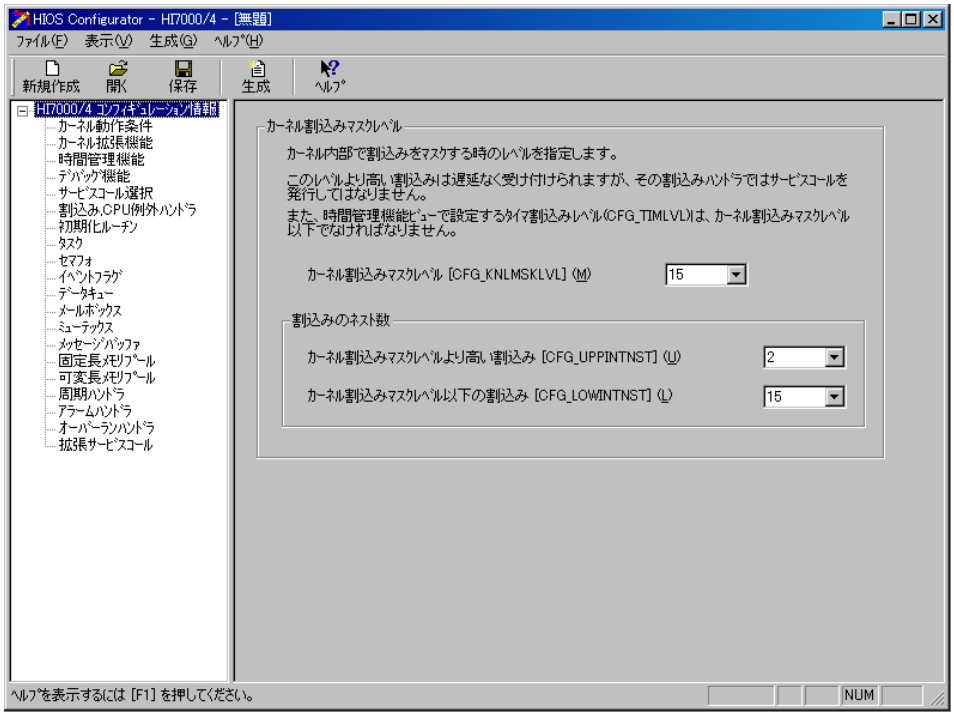


図 3-4　コンフィギュレータ起動画面

コンフィギュレータの構成を以下に示します。

- 画面左側：構築情報入力パート一覧ウィンドウ
- 画面右側：構築情報入力ウィンドウ

コンフィギュレータ起動画面は、各 HI シリーズ OS で相違があります。詳細については、コンフィギュレータのオンラインヘルプをご参照ください。

(b)　コンフィギュレータ情報の保存

コンフィギュレータでの定義作業がすべて完了したら、登録した内容を保存し、コンフィギュレーションファイルの生成を行います。コンフィギュレータのヘッダメニュー「保存」と「生成」の内容を次に示します。

表 3-4　コンフィギュレータにおける「保存」と「生成」の内容

ボタン	内　　容
保存 	サンプルフォルダ内に、拡張子：hcf のファイルを作成し、コンフィギュレータで定義した内容を保存
生成 	コンフィギュレータで定義した内容をもとに、コンフィギュレーションファイルを生成します。

コンフィギュレータでの定義内容変更後は、

- 「保存」で定義情報を更新
- 「生成」で更新内容にもとづくコンフィギュレーションファイルの生成

を必ず行ってください。

(c)　コンフィギュレータの定義操作

構築情報入力パート一覧ウィンドウの「タスク」を例に、定義操作について説明します。

図 3-5　タスク入力パート

タスク情報入力パートは各種情報の入力およびタスクの生成／削除を行うウィンドウです。タスク情報入力パートにおける構築情報入力ウィンドウで表示される内容について以下に示します。

表 3-5　タスク情報入力パートにおける構築情報入力ウィンドウの表示内容

項番	構築情報入力ウィンドウ	内　　容
1	タスク情報 *1	現在定義されている以下の内容を表示 – 最大タスク ID – 最大タスク優先度 – スタティックスタック使用の最大タスク ID – ダイナミックスタック領域サイズ
2	スタティックスタック一覧 *2	現在定義されているスタティックスタックに関する以下の内容を表示 – スタック領域名称 – スタック領域サイズ – 領域を使用するタスク ID
3	タスク一覧 *2	現在定義されているタスクに関する以下の内容を表示 – 「カーネルライブラリとのリンク指定」の有無 – タスク ID／タスク名称 – 生成後の状態 – タスク起動アドレス – タスク初期優先度 – スタックサイズ／領域 – 記述言語 – コプロセッサ属性 – 拡張情報 – タスク例外処理ルーチン定義情報 ・タスク例外ルーチン起動アドレス ・タスク例外ルーチンのコプロセッサ属性 ・タスク例外ルーチン記述言語

【注】 *1　「タスク情報」の変更は、［変更］ボタンをクリックし、［タスク情報の変更］ダイアログボックスをオープンして行います。

*2　「スタティックスタック一覧」、「タスク一覧」の変更は、ポップアップメニュー（右ボタンクリックで表示）をオープンして行います。

タスク情報入力パートにおける「タスク情報」の変更について、次に示します。

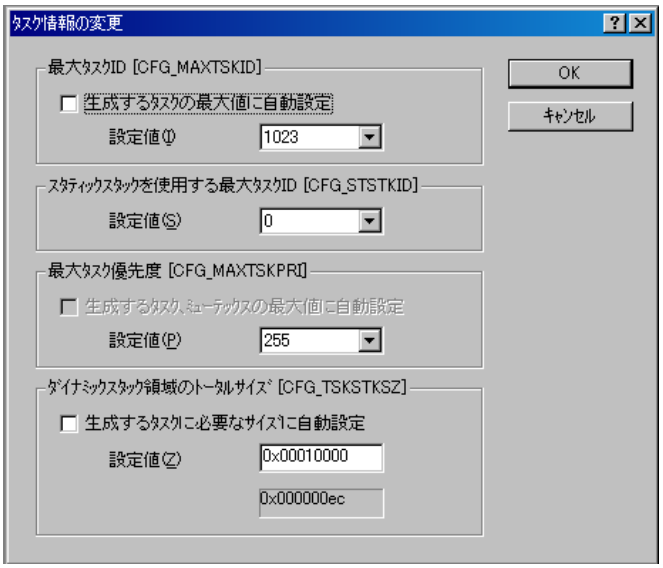


図 3-6　タスク情報の変更画面

表 3-6　タスク情報の変更内容

項番	項　目	表示内容
1	最大タスク ID	システムに登録されるタスク数の最大値 《設定方法》 – 「生成するタスクの最大値に自動設定」を選択 ［設定値(I)］ボックスの指定内容は無視され、コンフィギュレータ上で生成されるタスクに応じ最小値を自動計算 – ［設定値(I)］はプルダウンメニューから選択
2	スタティックスタックを使用する最大タスク ID	スタティックスタックを使用するタスクの最大タスク ID 《設定方法》 – ［設定値(S)］はプルダウンメニューから選択 ※0 以外を指定して[OK]ボタンをクリックすると、［スタティックスタック情報の定義］ダイアログボックスが開きます
3	最大タスク優先度	システムに登録されるタスクに定義された優先度の最大値 《設定方法》 – ［設定値(P)］はプルダウンメニューから選択
4	ダイナミックスタック領域のトータルサイズ	ダイナミックスタック領域のトータルサイズ 《設定方法》 – 「生成するタスクに必要なサイズに自動設定」を選択 ［設定値(Z)］ボックスの指定内容は無視され、コンフィギュレータ上で生成されるタスクに応じ最小値を自動計算 – ［設定値(Z)］はダイナミックスタック領域のトータルサイズを入力 ※［設定値(Z)］ボックス下に表示されているサイズは、現在登録されているタスクで使用するサイズから算出した値

［スタック情報の定義］ダイアログボックスについて、次に示します。

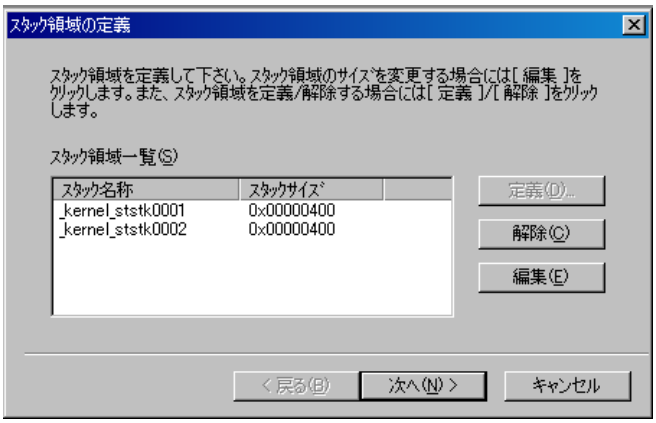


図 3-7　スタティックスタック情報の定義画面

［スタティックスタックを使用する最大タスク ID］の設定時、0 以外を指定して[OK]ボタンをクリックすると、［スタティックスタック情報の定義］ダイアログボックスが開きます

［スタック名称］に表示されたスタックをクリックして［編集(E)］ボタンをクリックすると、スタックサイズを変更する事ができます。以下に変更画面を示します。

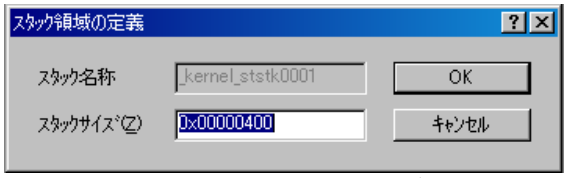


図 3-8　スタティックスタックサイズの変更画面

［スタックサイズ(Z)］には、スタティックスタック領域に必要なサイズを入力後、［OK］ボタンをクリックして変更します。

各スタティックスタックサイズの設定が完了したら、［次に(N) >］ボタンをクリックして、各スタティックスタックを使用するタスク ID の定義を行います。タスク ID を定義する［スタティックスタック情報の定義］ダイアログボックスを次に示します。

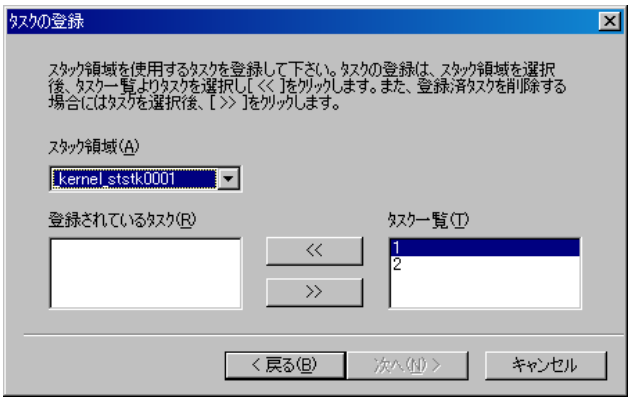


図 3-9　スタティックスタックを使用するタスク ID の定義画面

設定手順

(1) ［スタック領域(A)］のプルダウンメニューから、タスクIDを定義するスタティックスタックを選択

(2) ［スタック領域(A)］に選択したスタティックスタックを使用するタスクIDを［タスク一覧(T)］から選択し、［<<］ボタンをクリックして登録
※［登録されているタスク(R)］に表示されているタスクIDを選択し、［>>］ボタンをクリックすると登録解除

(3) 「共有スタック機能」を使用する場合は、［スタック領域(A)］に選択したスタティックスタックを使用するタスクIDを複数登録することで定義

全スタティックスタックへ使用するタスク ID の登録が完了したら、［次に(N) >］ボタンをクリックすると、次に示す画面が表示され、スタティックスタックに関する設定作業を終了します。

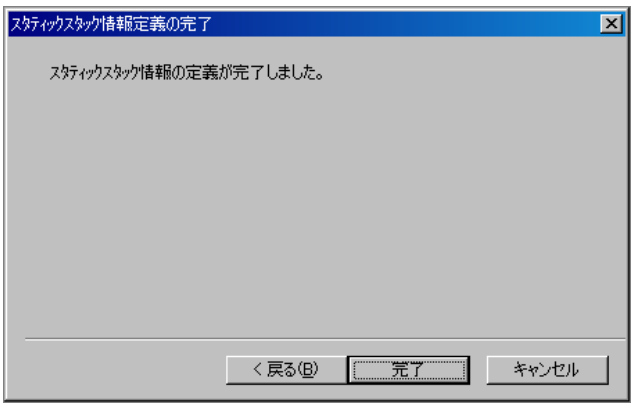


図 3-10　スタティックスタック定義作業の終了画面

［完了］ボタンをクリックすると、タスク情報入力パートのスタティックスタック一覧に定義した内容が反映されます。

タスク情報入力パートにおける「タスク一覧」の変更について、次に示します。
「タスク一覧」内で右ボタンクリックすると表示されるポップアップメニューから、項目を選択して行います。以下にポップアップメニューの表示を示します。

図 3-11　ポップアップメニュー

表 3-7　ポップアップメニューの内容

項番	メニュー項目	内　容
1	生成	［タスクの生成］ダイアログボックスを開き、タスクの生成内容を定義します
2	削除	選択された位置のタスク生成内容を削除します
3	変更	［タスクの生成情報を変更］ダイアログボックスを開き、選択されたタスクの生成内容を変更します
4	上へ	選択されたタスクの位置を一つ上へ移動します *1
5	下へ	選択されたタスクの位置を一つ下へ移動します *1

【注】 *1　表示されている順番で生成、起動の処理が行われるため、システム起動時の生成順や起動順の変更に使用します。

ポップアップメニューの［生成］選択時、［タスクの生成］ダイアログボックスが表示されます。次に［タスクの生成］ダイアログボックスへの設定について示します。

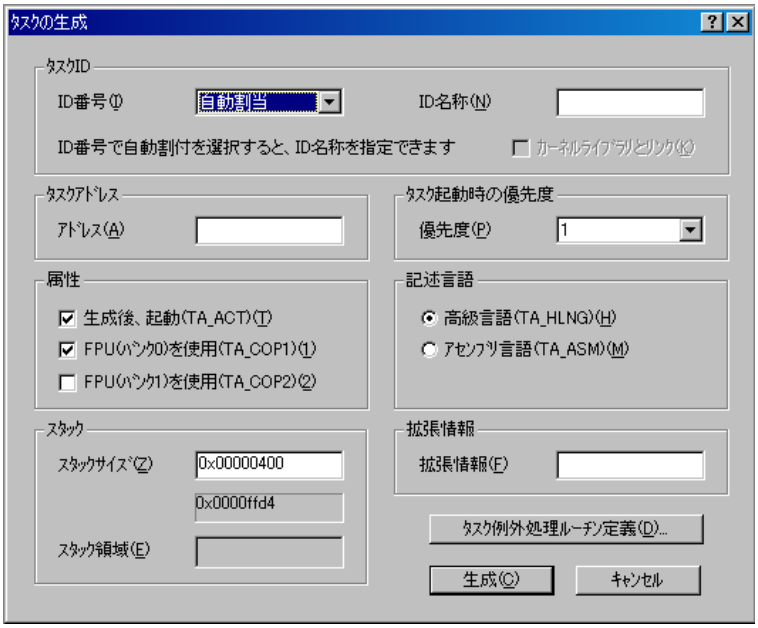


図 3-12　［タスクの生成］ダイアログボックス

表 3-8　［タスクの生成］ダイアログボックスの内容(1/2)

項番	項目内容	設定内容
1	ID 番号(I)	生成するタスクの ID 番号を指定 《設定方法》 – 「自動割当」指定時は、構築ファイル生成時にコンフィギュレータが自動的に未使用 ID から割り当てます – プルダウンメニューから選択
2	ID 名称(N)	「自動割当」指定時、生成タスクの ID 名称を入力
3	カーネルライブラリとリンク(K) *1	生成するタスクを、カーネルライブラリとリンクする場合にオン（チェック）
4	アドレス(A)	生成するタスクの起動アドレスを、シンボルまたは数値で入力
5	優先度(P)	生成タスク起動時の優先度を指定
6	属性 *2	生成時のタスク状態を指定 《設定方法》 – 「実行可能状態」で生成する場合、［生成後、起動(TA_ACT)］チェックボックスをオン

【注】 *1　［ID 番号(I)］ボックスの指定で、「自動割当」以外を指定した場合は定義できません。
*2　コプロセッサ属性を定義する項目もあります。詳細はコンフィギュレータのオンラインヘルプを参照してください。

表 3-8　［タスクの生成］ダイアログボックスの内容(2/2)

項番	項目内容	設定内容
7	記述言語	生成するタスクの記述言語を指定 – 高級言語で記述されている場合は［高級言語(TA_HLNG)］を選択 – 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
8	スタックサイズ(Z)	生成するタスクが使用するスタックサイズを入力 ※ ［スタックサイズ］ボックス下の表示サイズは、ダイナミックスタック領域の残容量から指定可能なサイズを算出した値です
9	スタック領域(E) *3	生成するタスクの使用するスタック領域が表示
10	拡張情報(F)	拡張情報をシンボルまたは数値で入力

【注】 *3　［ID 番号(I)］ボックスで、スタティックスタック使用タスク ID を指定した場合のみ表示されます。

［タスクの生成］ダイアログボックスにおける全ての設定が完了したら、［生成］ボタンをクリックして定義します。
登録するタスクの定義が完了したら、［キャンセル］ボタンをクリックして定義作業を終了します。

生成するタスクにタスク例外処理ルーチンを定義する場合は、［タスク例外処理ルーチン定義(D)］ボタンをクリックすると、［タスク例外処理ルーチンの定義］ダイアログボックスが表示されます。
次に［タスク例外処理ルーチンの定義］ダイアログボックスへの設定について示します。

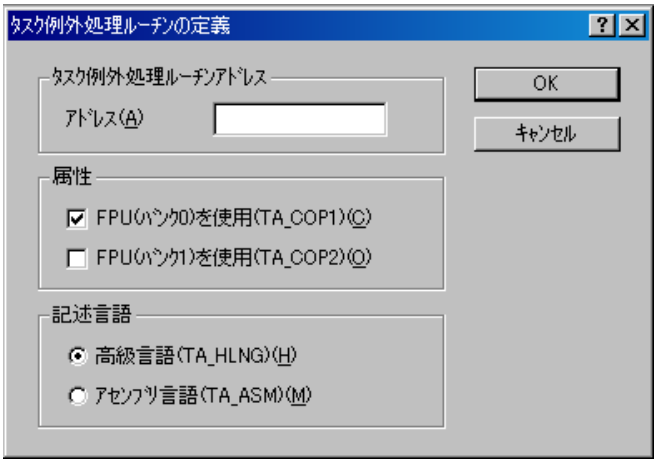


図 3-13　［タスク例外処理ルーチンの定義］ダイアログボックス

表 3-9　［タスク例外処理ルーチンの定義］ダイアログボックスの内容

項番	項目内容	設定内容
1	アドレス(A)	定義するタスク例外処理ルーチンのアドレスをシンボルまたは数値で入力
2	属性 *1	使用するコプロセッサ属性を選択
3	記述言語	生成するタスクの記述言語を指定 – 高級言語で記述されている場合は［高級言語(TA_HLNG)］を選択 – 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

【注】 *1　コプロセッサ属性定義に関する項目の詳細は、コンフィギュレータのオンラインヘルプを参照してください。

［タスク例外処理ルーチンの定義］ダイアログボックスにおける全ての設定が完了したら、［OK］ボタンをクリックして定義します。

このようにして、コンフィギュレータに必要な情報を定義します。

次に、コンフィギュレータの各構築情報入力パートについて説明します。

(2)　コンフィギュレータの各構築情報入力パート

起動画面を以下に示します。

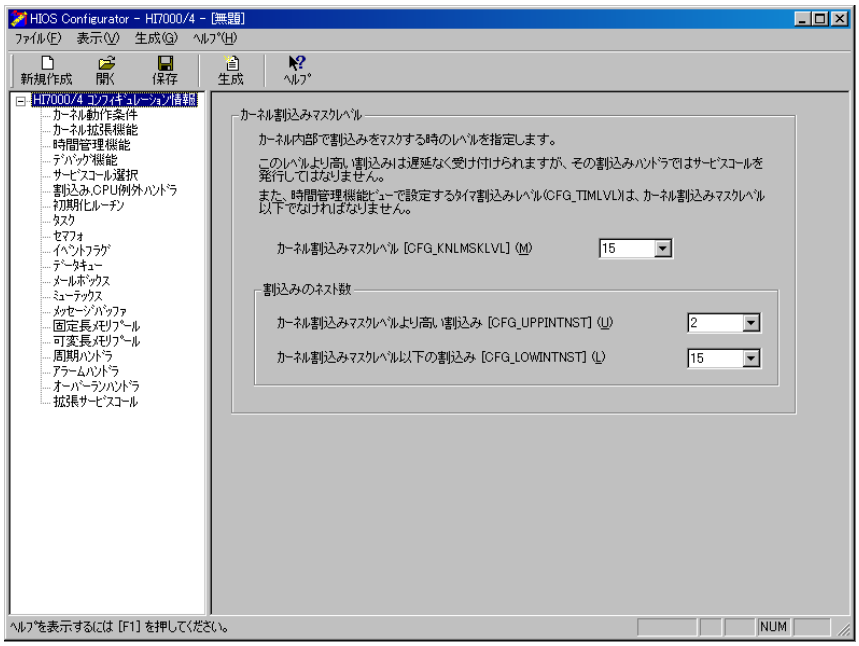


図 3-14　コンフィギュレータ起動画面(HI7000/4)

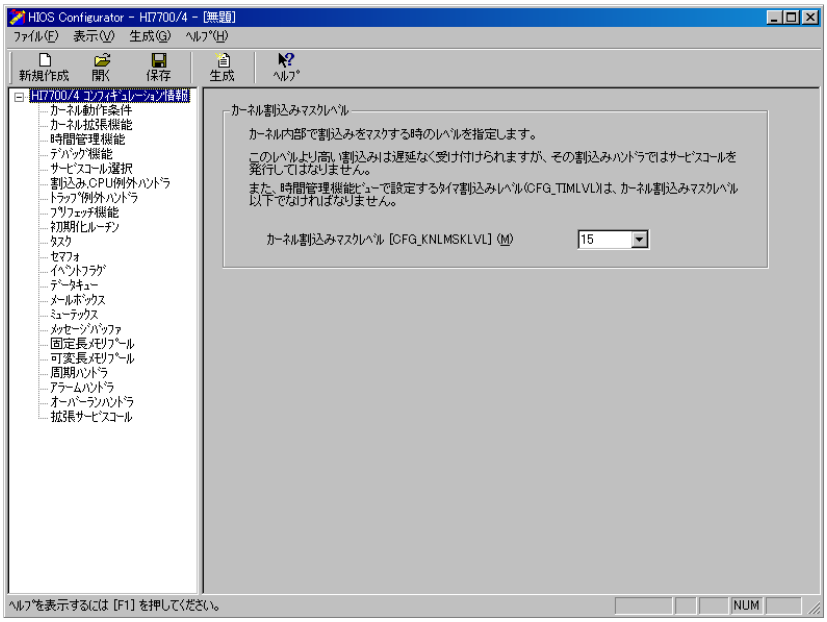


図 3-15　コンフィギュレータ起動画面(HI7700/4、HI7750/4)

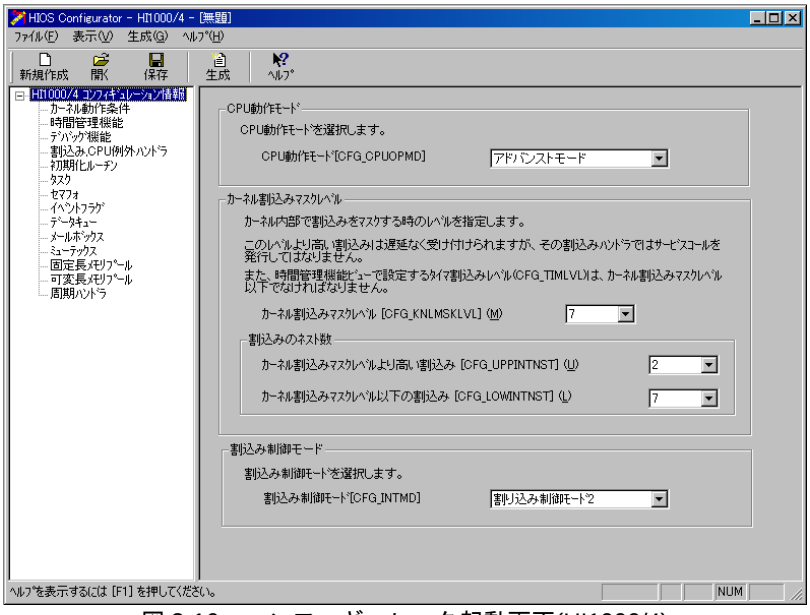


図 3-16　コンフィギュレータ起動画面(HI1000/4)

コンフィギュレータは、上記の通り、「構築情報入力パート一覧ウィンドウ（左側）」と「構築情報入力ウィンドウ（右側）」で構成されます。

(a)　カーネル動作条件入力パート

起動画面は、「カーネル動作条件」入力パートと兼用になっています。
カーネル動作条件入力パートにおける設定項目を以下に示します。

表 3-10　カーネル動作条件入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	カーネル割込みマスクレベル	カーネル内部で割込みをマスクする時の割込みマスクレベルを定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
2	割込みネスト数	「カーネル割込みマスクレベルより高いレベルの割込みハンドラのネスト数」と「カーネル割込みマスクレベル以下の割込みのネスト数」を定義	HI7000/4、HI1000/4
3	CPU 動作モード	CPU 動作モードを選択	HI1000/4
4	割込み制御モード	割込み制御モードの選択	HI1000/4

項目毎に準備された▼ボタンを押下し、表示されたプルダウンメニューから選択する事で、設定します。

(b)　カーネル拡張機能入力パート

カーネル拡張機能入力パート画面を以下に示します。

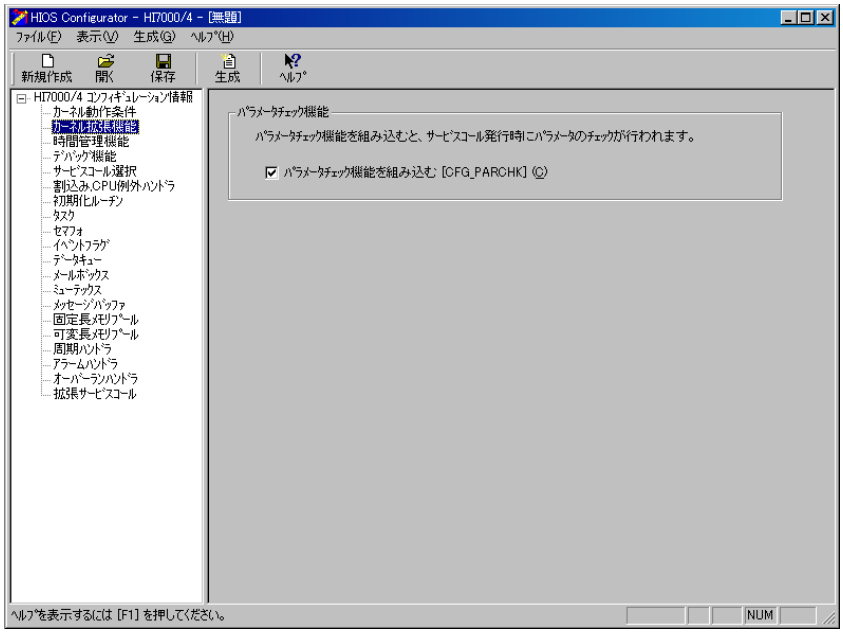


図 3-17　カーネル拡張機能入力パート(HI7000/4)

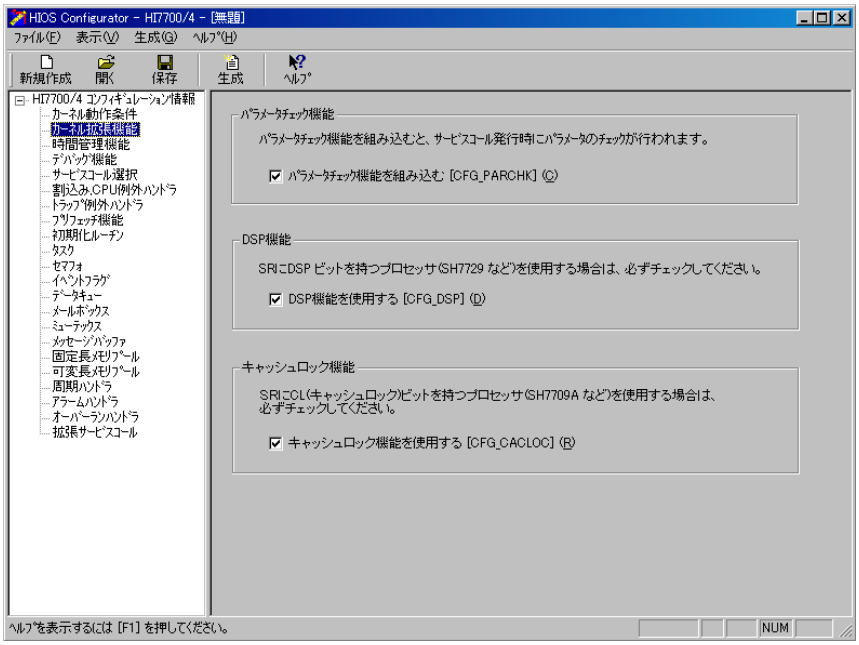


図 3-18　カーネル拡張機能入力パート(HI7700/4、HI7750/4)

HI1000/4 のコンフィギュレータでは、「カーネル拡張機能入力パート」はありません。

カーネル拡張機能入力パートにおける設定項目を以下に示します。

表 3-11　カーネル拡張機能入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	パラメータチェック機能	パラメータチェック機能を組み込む場合に選択	HI7000/4、HI7700/4、HI7750/4
2	DSP 機能 *1	DSP 機能を使用する場合に選択	HI7700/4
3	キャッシュロック機能 *1	キャッシュロック機能を使用する場合に選択	HI7700/4

【注】 *1　「DSP 機能」、「キャッシュロック機能」を持つプロセッサを使用する場合は、必ず設定してください。

項目毎のチェックボックスを選択する事で設定します。

(c)　時間管理機能入力パート

カーネル拡張機能入力パート画面を以下に示します。

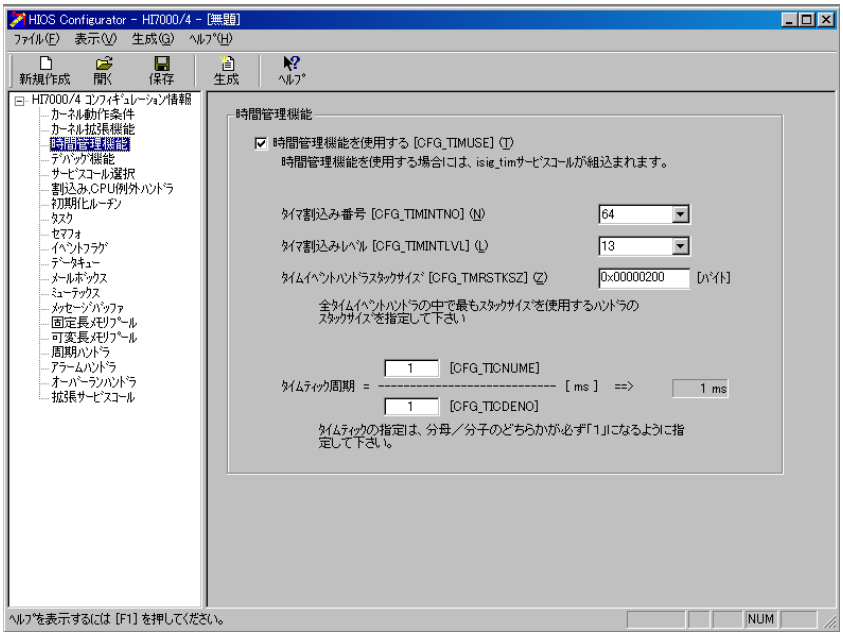


図 3-19　時間管理機能入力パート(HI7000/4、HI7700/4、HI7750/4)

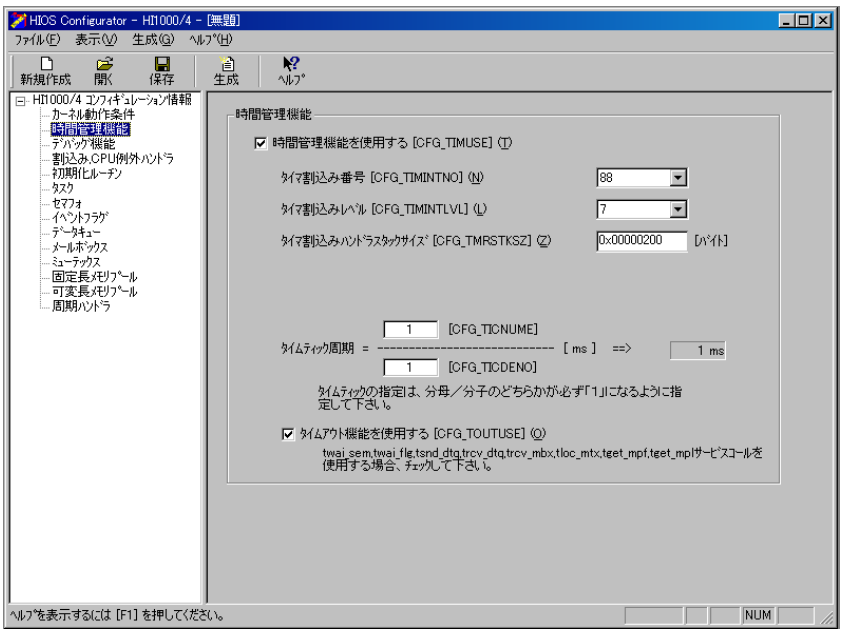


図 3-20　時間管理機能入力パート(HI1000/4)

時間管理機能入力パートにおける設定項目を以下に示します。

表 3-12　時間管理機能入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	時間管理機能	時間管理機能を組み込む場合に選択	HI7000/4、HI7700/4、HI7750/4、HI1000/4
2	タイマ割込み番号	タイマ割込みのベクタ番号（または INTEVT コード）を定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
3	タイマ割込みレベル	タイマ割込みの割込みレベルを定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
4	タイムイベントハンドラスタックサイズ	タイムイベントハンドラが使用するスタックのサイズを定義	HI7000/4、HI7700/4、HI7750/4
	タイマ割込みハンドラスタックサイズ	タイマ割込みハンドラが使用するスタックのサイズを定義	HI1000/4
5	タイムティック周期	タイムティック供給周期の精度を変更する場合に定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
6	タイムアウト機能	タイムアウト機能付サービスコールを使用する場合に選択	HI1000/4

【注】タイムティック周期の指定では、分母／分子のどちらかは必ず「1」を指定してください。

項目毎に準備された▼ボタンを押下し、表示されたプルダウンメニューから選択、または項目に直接入力する事で設定します。

(d)　デバッグ機能入力パート

デバッグ機能入力パート画面を以下に示します。

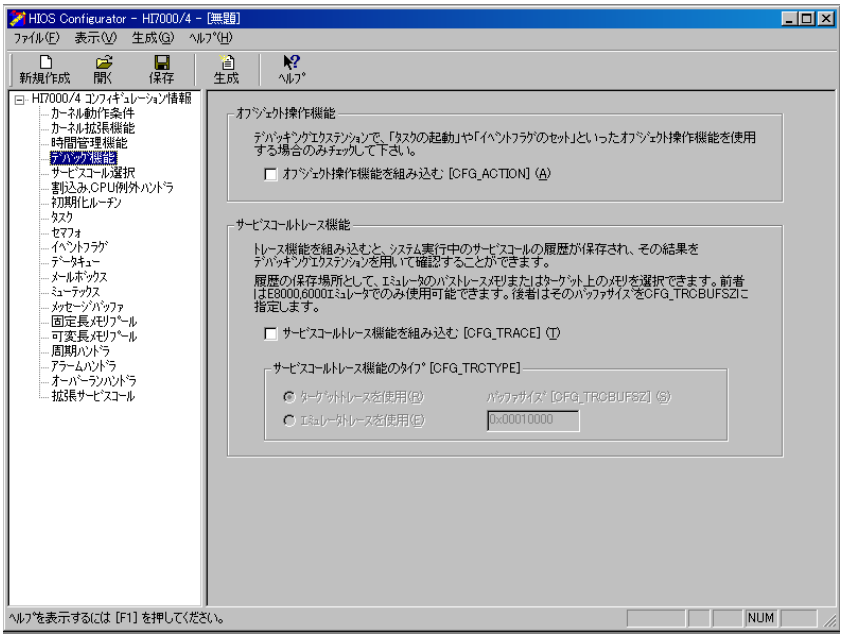


図 3-21　デバッグ機能入力パート(HI7000/4、HI7700/4、HI7750/4)

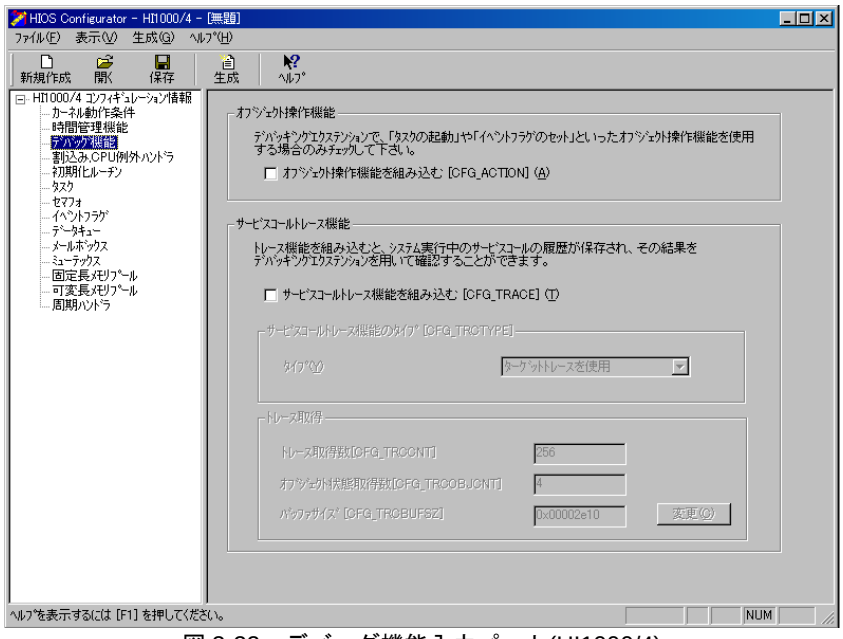


図 3-22　デバッグ機能入力パート(HI1000/4)

デバッグ機能入力パートにおける設定項目を以下に示します。

表 3-13　デバッグ機能入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	オブジェクト操作機能	デバッギングエクステンションにて「タスクの起動」や「イベントフラグのセット」などのオブジェクト操作を使用する場合に選択	HI7000/4、HI7700/4、HI7750/4、HI1000/4
2	サービスコールトレース機能	サービスコールトレース機能を組み込む場合に選択	HI7000/4、HI7700/4、HI7750/4、HI1000/4

項目毎に準備された▼ボタンを押下し、表示されたプルダウンメニューから、またはチェックボックス、ラジオボタンの選択、必要項目に直接入力する事で設定します。

(e)　サービスコール選択入力パート

サービスコール選択入力パート画面を以下に示します。

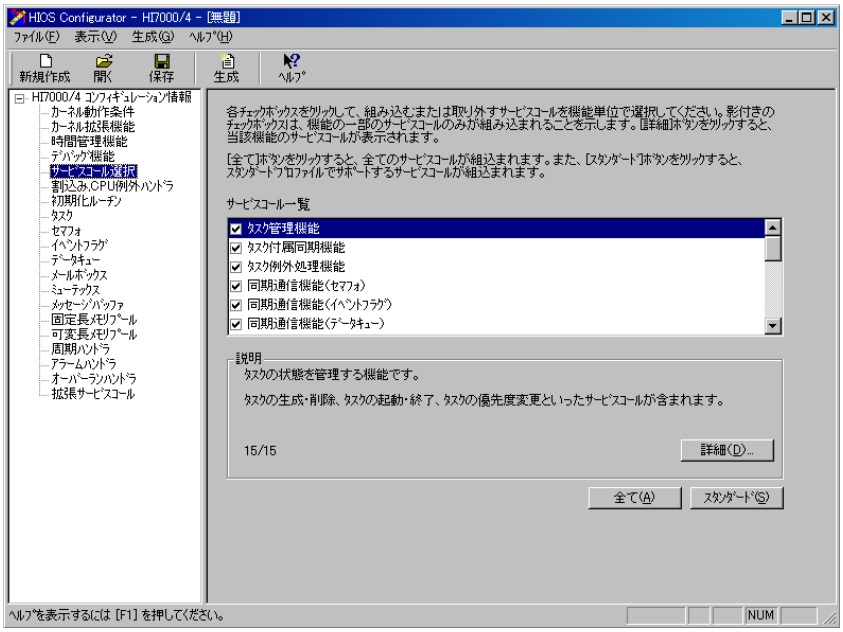


図 3-23　サービスコール選択入力パート(HI7000/4、HI7700/4、HI7750/4)

HI1000/4 のコンフィギュレータでは、「サービスコール選択入力パート」はありません。

サービスコール選択入力パートでは、「サービスコール一覧」ダイアログで、組み込む、または取り外すサービスコールを機能単位で選択します。

サービスコール単位で選択する場合は、「説明」枠内の詳細ボタンをクリックして行います。

「全て」ボタンをクリックすると、全てのサービスコールが組み込まれます。また、「スタンダード」ボタンをクリックすると、スタンダードプロファイルでサポートするサービスコールだけが組み込まれます。

(f)　割込み、CPU 例外ハンドラ入力パート

割込み、CPU 例外ハンドラ入力パート画面を以下に示します。

図 3-24　割込み、CPU 例外ハンドラ入力パート(HI7000/4)

図 3-25　割込み、CPU 例外ハンドラ入力パート(HI7700/4、HI7750/4)

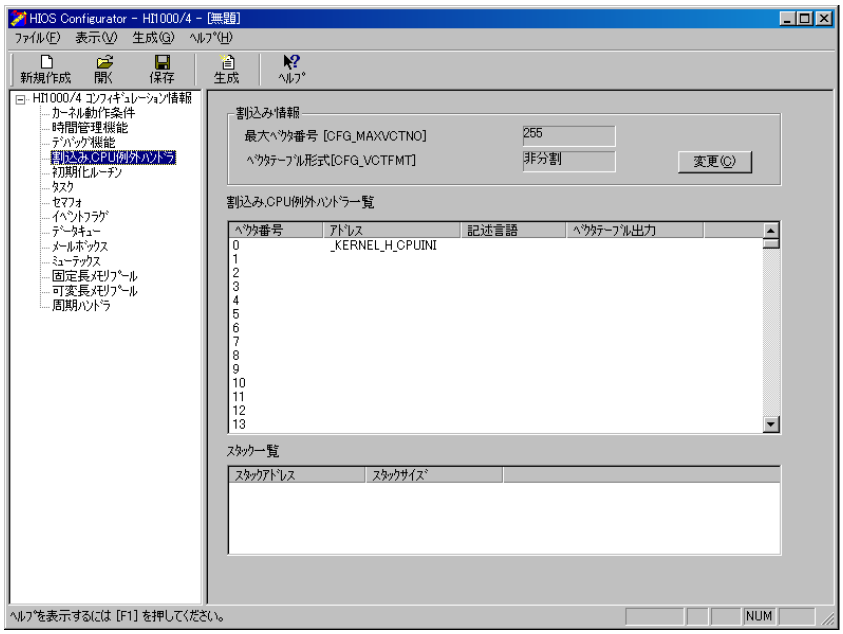


図 3-26　割込み、CPU 例外ハンドラ入力パート(HI1000/4)

割込み、CPU 例外ハンドラ入力パートにおける設定項目を以下に示します。

表 3-14　割込み、CPU 例外ハンドラ入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	割込み情報	割込みハンドラなどに関する情報を定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
	【割込み情報】 最大ベクタ番号、割込みハンドラスタックのトータルサイズ、ダイレクト割込みハンドラ使用の有無、割込みハンドラ動的生成組み込みの有無		HI7000/4
	【割込み情報】 最大例外コード、割込みハンドラスタックのトータルサイズ		HI7700/4、HI7750/4
	【割込み情報】 最大ベクタ番号、ベクタテーブル形式		HI1000/4
2	割込み、CPU 例外ハンドラ、トラップ例外ハンドラ一覧	各ベクタ要因で起動するハンドラを定義	HI7000/4
	割込み、CPU 例外ハンドラ一覧	各例外要因で起動するハンドラを定義	HI7700/4、HI7750/4
	割込み、CPU 例外ハンドラ一覧	各ベクタ要因で起動するハンドラを定義	HI1000/4
3	スタック一覧	割込みハンドラで使用するスタックの定義情報	HI1000/4

割込み、CPU 例外ハンドラなどの登録手順を以下に示します。

《ハンドラ登録手順》
1. ハンドラを登録するベクタ番号（または例外コード）を選択。
2. 右ボタンクリックで表示されるサブメニューから「定義」を選択。
3. 表示された定義ウィンドウに必要事項を設定し、OK ボタン押下で登録完了。

(g)　トラップ例外ハンドラ入力パート

トラップ例外ハンドラ入力パート画面を以下に示します。

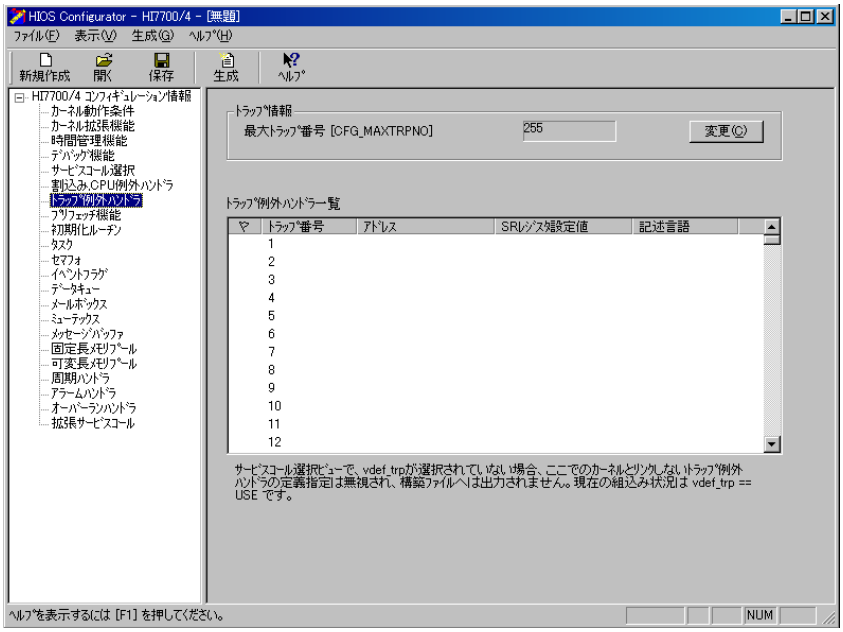


図 3-27　トラップ例外ハンドラ入力パート(HI7700/4、HI7750/4)

HI7000/4、および HI1000/4 のコンフィギュレータでは、「トラップ例外ハンドラ入力パート」はありません。

トラップ例外ハンドラ入力パートにおける設定項目を以下に示します。

表 3-15　トラップ例外ハンドラ入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	トラップ情報	最大トラップ番号を定義	HI7700/4、HI7750/4
2	トラップ例外ハンドラー覧	トラップ例外要因で起動するハンドラを定義	HI7700/4、HI7750/4

トラップ例外ハンドラの登録手順を以下に示します。

《トラップ例外ハンドラ登録手順》

1. ハンドラを登録するトラップ番号を選択。
2. 右ボタンクリックで表示されるサブメニューから「定義」を選択。
3. 表示された定義ウィンドウに必要事項を設定し、OK ボタン押下で登録完了。

(h)　プリフェッチ機能入力パート

プリフェッチ機能入力パート画面を以下に示します。

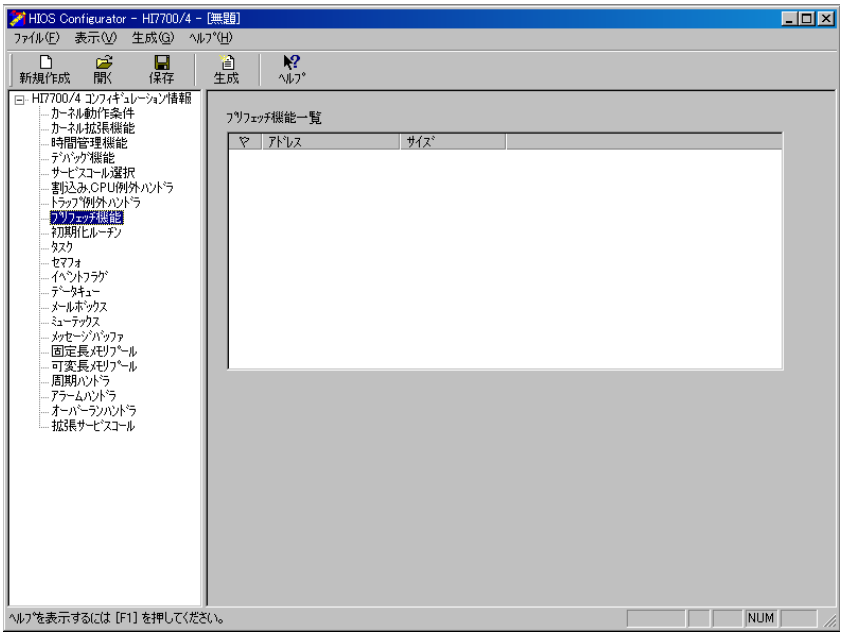


図 3-28　プリフェッチ機能入力パート(HI7700/4、HI7750/4)

HI7000/4、および HI1000/4 のコンフィギュレータでは、「プリフェッチ機能入力パート」はありません。

プリフェッチ機能入力パートにおける設定項目を以下に示します。

表 3-16　プリフェッチ機能入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	プリフェッチ機能一覧	カーネルがアイドリング時にプリフェッチする領域の先頭アドレスを定義	HI7700/4、HI7750/4

プリフェッチ機能の設定手順を以下に示します。

《プリフェッチ機能設定手順》

1. プリフェッチ機能一覧内で、右ボタンをクリックし、表示されるサブメニューから「登録」を選択。
2. 表示された登録ウィンドウに必要事項を設定し、登録ボタン押下で完了。
3. 連続登録可能なため、OK ボタン押下後、次のプリフェッチ登録が行えます。
   登録がすべて完了したら、キャンセルボタンで登録作業を終了します。

(i)　初期化ルーチン入力パート

初期化ルーチン入力パート画面を以下に示します。

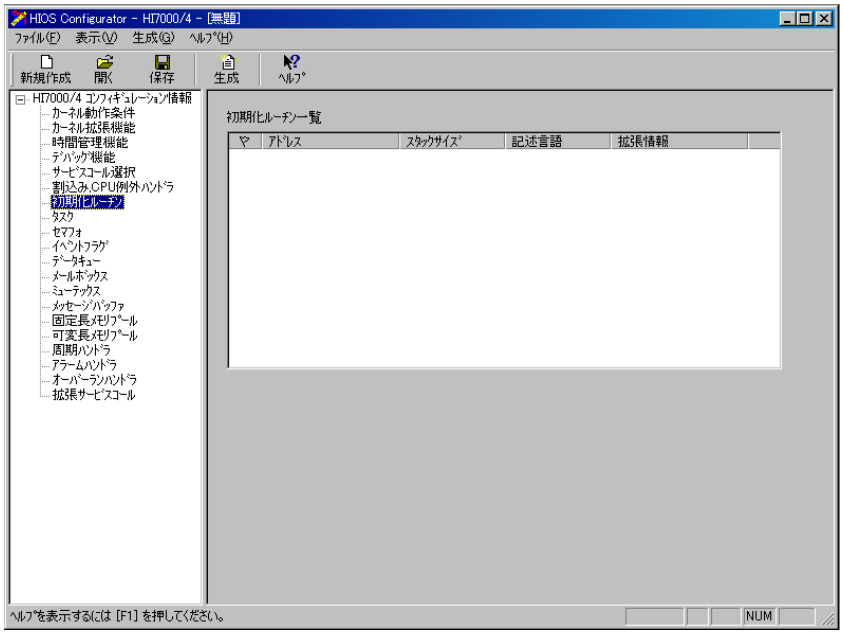


図 3-29　初期化ルーチン入力パート

初期化ルーチン入力パートにおける設定項目を以下に示します。

表 3-17　初期化ルーチン入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	初期化ルーチン一覧	カーネル初期化処理から呼び出す初期化ルーチンを定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4

初期化ルーチンの登録手順を以下に示します。

《初期化ルーチン登録手順》

1. 初期化ルーチン一覧内で、右ボタンをクリックし、表示されるサブメニューから「登録」を選択。
2. 表示された登録ウィンドウに必要事項を設定し、登録ボタン押下で完了。
3. 連続登録可能なため、OK ボタン押下後、次の初期化ルーチン登録が行えます。
   登録がすべて完了したら、キャンセルボタンで登録作業を終了します。

(j)　タスク入力パート

タスク入力パート画面を以下に示します。

図 3-30　タスク入力パート(HI7000/4、HI7700/4、HI7750/4)

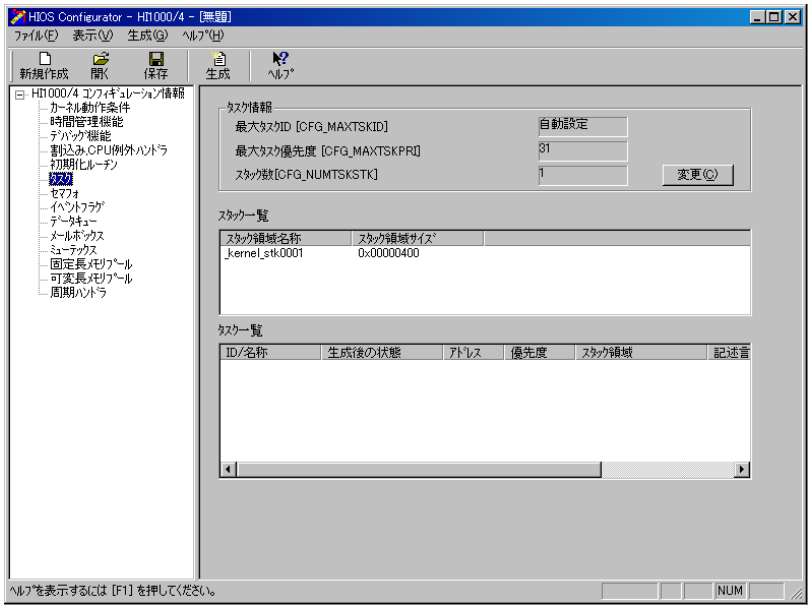


図 3-31　タスク入力パート(HI1000/4)

タスク入力パートにおける設定項目を以下に示します。

表 3-18　タスク入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	最大タスク ID	カーネルに登録するタスクの最大 ID を定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
2	スタティックスタックを使用する最大タスク ID	スタティックスタックを使用するタスクの最大 ID を定義	HI7000/4、HI7700/4、HI7750/4
3	最大タスク優先度	カーネルに登録するタスクの最大優先度を定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
4	ダイナミックスタック領域サイズ	ダイナミックスタック使用サイズの合計を定義	HI7000/4、HI7700/4、HI7750/4
5	スタック数	スタックの数	HI1000/4
6	スタティックスタック一覧	登録されているスタティックスタック情報	HI7000/4、HI7700/4、HI7750/4
7	タスク一覧	登録されているタスク情報	HI7000/4、HI7700/4、HI7750/4、HI1000/4
8	スタック一覧	登録されているタスクスタック情報	HI1000/4

タスクの登録手順を以下に示します。

《初期化ルーチン登録手順》
1. タスク一覧内で、右ボタンをクリックし、表示されるサブメニューから「生成」を選択。
2. 表示された生成ウィンドウに必要事項を設定し、生成ボタン押下で完了。
3. 連続登録可能なため、生成ボタン押下後、次のタスク登録が行えます。
   登録がすべて完了したら、キャンセルボタンで登録作業を終了します。

(k)　タスク以外のオブジェクト入力パート

「セマフォ入力パート」、「イベントフラグ入力パート」など、タスク以外の各オブジェクト入力パートの構成や設定項目が同じため、タスク以外の各オブジェクト入力パートについて「セマフォ入力パート」を例に説明します。セマフォ入力パート画面を以下に示します。

図 3-32　セマフォ入力パート

セマフォ入力パートにおける設定項目を以下に示します。

表 3-19　セマフォ入力パート設定項目

項番	設定項目	設定内容	対象 OS
1	最大セマフォ ID	カーネルに登録するセマフォの最大 ID を定義	HI7000/4、HI7700/4、HI7750/4、HI1000/4
2	セマフォ一覧	登録されているセマフォ情報	HI7000/4、HI7700/4、HI7750/4、HI1000/4

タスク以外のオブジェクト入力パートでは、上記設定項目を各オブジェクト名称（イベントフラグやメールボックスなど）に置き換えてご参照ください。

セマフォを例に、オブジェクト登録手順を以下に示します。

《セマフォ登録手順》

1. セマフォ一覧内で、右ボタンをクリックし、表示されるサブメニューから「生成」を選択。
2. 表示された生成ウィンドウに必要事項を設定し、生成ボタン押下で完了。
3. 連続登録可能なため、生成ボタン押下後、次のセマフォ登録が行えます。
   登録がすべて完了したら、キャンセルボタンで登録作業を終了します。

3.2.2　コンフィギュレータに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたコンフィギュレータに関する質問についての回答を記述します。

《 FAQ 目次 》

(1)　コンフィギュレータによる多重割込みの設定

分類：構築関連、カーネル環境定義、コンフィギュレータ

HI7000/4
HI7700/4
HI7750/4

HI1000/4

■質問

　多重割込みを許可するように設定したのですが、コンフィギュレータでどのように設定するのですか。
　また、登録する割込みハンドラには、多重割込みを行うために必要な記述はありますか。

■回答

　コンフィギュレータの割込み、CPU 例外ハンドラビューに、使用する割込みの各例外コードの内容を設定します。具体的には、各例外コードについて、アドレスと SR レジスタ設定値を設定します。この SR 設定値が、割込みハンドラ起動時の SR 値になりますので、割込みレベルにあわせて SR を設定してください。この SR 値設定だけで、多重割込みが実現できます。

3.2.3　セットアップテーブルによる定義（HI2000/3）

HI2000/3 におけるカーネル環境の定義は、セットアップテーブルで行います。
セットアップテーブルは、以下に示す各定義部で構成されます。

表 3-20　セットアップテーブルの構成

定義部名称	定義内容
定数定義部	カーネルの機能（同期通信機能、時間管理機能等）を使用するために必要な情報を定義
タスク登録部	タスクを使用するために必要な情報を定義
固定長メモリプール登録部	固定長メモリプールを使用するために必要な情報を定義
可変長メモリプール登録部	可変長メモリプールを使用するために必要な情報を定義
周期起動ハンドラ登録部	周期起動ハンドラを使用するために必要な情報を定義
システムコールトレース機能登録部	システムコールトレース機能を使用するために必要な情報を定義
拡張情報登録部	タスク、イベントフラグ、セマフォ、メイルボックス、固定長メモリプール、可変長メモリプール、周期起動ハンドラの拡張情報を定義

上記設定項目は、登録／未登録、使用／未使用に関わらず、すべての項目を定義します。定義されていない場合、システム結合時に未定義エラーとなります。

(1) 定数定義部

カーネルの各種機能（同期通信機能、時間管理機能など）を使用するために必要な情報を登録します。セットアップテーブルの定数定義部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VALUE define section                                             %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;LABEL                  VALUE   ;:[ RANGE ]     ;: COMMENT
;--------------------------------------------------------------------------
CPUINTM:      .assign 3    ;:[0.......3]   ;: CPU interrupt mode         ← ①
IMASK:        .assign 6    ;:[0.......8]   ;: Max interrupt level        ← ②
MAXPRI:       .assign 31   ;:[0......31]   ;: Max low priority           ← ③
FLGCNT:       .assign 4    ;:[0.....255]   ;: Eventflag definition count ← ④
SEMCNT:       .assign 4    ;:[0.....255]   ;: Semaphore definition count ← ⑤
MBXCNT:       .assign 4    ;:[0.....255]   ;: Mailbox definition count   ← ⑥
;
OSSTKSIZ:     .equ  18+(10*2)+(6*1)+8 ;:[18...]   ;: OS stack size       ← ⑦
TIMSTKSIZ:    .equ  40+(10*1)+(6*1)+8 ;:[0,40...] ;: Timer stack size    ← ⑧
TRCSTKSIZ:    .equ  26+(6*1)+8        ;:[0,26...] ;: Trace stack size    ← ⑨
;
TTMOUT:       .assign USE  ;:[USE / NOTUSE];: Time-out Function define   ← ⑩
;
```

図 3-33　セットアップテーブルの定数定義部

① CPUINTM【割込み制御モード】
　使用する割込み制御モードを指定

② IMASK【カーネル割込みマスクレベル】
　カーネル内部で割込みをマスクする時のレベルを指定

③ MAXPRI【最大タスク優先度数】
　もっとも低いタスク優先度を指定

④ FLGCNT【イベントフラグ登録数】
　カーネルに登録するイベントフラグの最大 ID を指定

⑤ SEMCNT【セマフォ登録数】
　カーネルに登録するセマフォの最大 ID を指定

⑥ MBXCNT【メイルボックス登録数】
　カーネルに登録するメイルボックスの最大 ID を指定

⑦ OSSTKSIZ【カーネル用スタックサイズ】
　カーネル（OS）が使用するスタックサイズを指定

⑧ TIMSTKSIZ【タイマ割込みハンドラ用スタックサイズ】
　タイマ割込みハンドラが使用するスタックサイズを指定

⑨ TRCSTKSIZ【システムコールトレース処理用スタックサイズ】
　システムコールトレース機能使用時、処理で使用するスタックサイズを指定

⑩ TTMOUT【タイムアウト機能使用の有無】
　タイムアウト付きシステムコールの使用有無を指定

【注】定数定義部にて使用しているシンボルに対する変更、削除は行わないでください。

OSSTKSIZ、TIMSTKSIZ、TRCSTKSIZ の算出方法については、HI2000/3 ユーザーズマニュアルをご参照ください。

(2) タスク登録部

タスクを登録するための各種定義を行います。セットアップテーブルのタスク登録部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    TASK define section                                            %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;               TASK_TOP_LABEL                  ;: COMMENT
;--------------------------------------------------------------------------
        .import _TASKA                          ;: TASK.C
        .import _TASKB                          ;: TASK.C                       ①
;
;------ Usage -------------------------------------------------------------
;               .res.b  SIZE +TSKSTKSIZ ;[RANGE];: COMMENT
;TSK?_SP_LABEL: .equ    $                       ;: COMMENT
;--------------------------------------------------------------------------
TSKSTKSIZ:      .equ    50+(10*2)+(6*1)+6+8;[50...];: Task minimum stack size
        .section        h2sstack,stack,align=2
                .res.b  (36) +TSKSTKSIZ ;[50...];: tskid1 stack area
TSK1_SP:        .equ    $                                                         ③
                .res.b  8
                .res.b  (36) +TSKSTKSIZ ;[50...];: tskid2 stack area
TSK2_SP:        .equ    $
                .res.b  8
                .res.b  (32) +TSKSTKSIZ ;[50...];: tskid3 stack area            ②
TSK3_SP:        .equ    $
                .res.b  8
                .res.b  (32) +TSKSTKSIZ ;[50...];: tskid4 stack area
TSK4_SP:        .equ    $
                .res.b  8
;
        .section        h2ssetup,code,align=2
_HI_H8S:        .res.b  10                      ;: System Area
;------ Usage -------------------------------------------------------------
;LABEL          .data.b IMOD, ITSKPRI           ;: COMMENT
;               .data.l ITSKADR, ITSKSP         ;: COMMENT
;--------------------------------------------------------------------------
NOEXS:          .assign 0                       ;: initial mode = NO EXIST
RDY:            .assign 1                       ;: initial mode = READY
DMT:            .assign (-1)                    ;: initial mode = DORMANT
TDTLEN:         .assign 10;<- Not Change !      ;: TDT Length
        .section        h2ssetup,code,align=2
_HI_TDT:        .equ    $-TDTLEN                ;: Task define table
TDT_TOP:        .equ    $                       ;:
tdt_id1:        .data.b DMT, 1                  ;: init. mode, init. priority
                .data.l _TASKA, TSK1_SP         ;: top address, stack pointer    ⑤
tdt_id2:        .data.b DMT, 2                  ;: init. mode, init. priority
                .data.l _TASKB, TSK2_SP         ;: top address, stack pointer
tdt_id3:        .data.b NOEXS, 3                ;: init. mode, init. priority    ④
                .data.l 0, TSK3_SP              ;: top address, stack pointer
tdt_id4:        .data.b NOEXS, 4                ;: init. mode, init. priority
                .data.l 0, TSK4_SP              ;: top address, stack pointer
tdt_id5:        .data.b NOEXS, 5                ;: init. mode, init. priority
                .data.l 0, TSK4_SP              ;: top address, stack pointer
TDT_BTM:
TSKCNT:         .equ    (TDT_BTM-TDT_TOP)/TDTLEN
                                ;:[0...255]     ;: Task definition count
;
```

図 3-34　セットアップテーブルのタスク登録部

RENESAS

① 使用するタスク先頭アドレスを、外部参照シンボルとして宣言
② タスクスタック定義部
　各タスクにて使用するスタック領域の確保
③ タスクスタック領域の定義
　各タスクのスタック領域を定義
④ タスク定義部
　カーネルに登録するタスクを定義
⑤ タスクの定義
　カーネルに登録する各タスクの情報を定義

【注】　タスク登録部にて使用している、「TDTLEN」、「_HI_TDT」、「TDT_TOP」、「TDT_BTM」、「TSKCNT」のシンボルに対する変更、削除は行わないでください。
「TSKCNT」定義の行に対する変更、削除は行わないでください。

タスクスタック領域の定義における詳細は以下の通りです。

1 行目：使用スタックサイズの定義
2 行目：スタックラベルの定義（タスクスタックボトム）
3 行目：共有スタック管理領域の定義（共有スタック機能を使用しない場合は不要）

タスクの定義における詳細は以下の通りです。

【書式】　LABEL :　.data.b　IMOD, ITSKPRI
　　.data.l　ITSKADR, ITSKSP

1.LABEL：自由に設定可（定義がなくても問題なし）
2.IMOD（タスク初期状態）：タスクの登録と初期起動時の状態を定義
　(1) NOEXS(=0)　：未登録
　(2) RDY　(=1)　：起動時、実行可能状態
　(3) DMT　(=-1)：起動時、休止状態
3.ITSKPRI（タスク初期優先度）：タスクの初期優先度を定義
4.ITSKADR（タスク初期起動アドレス）：タスクの先頭アドレスを定義
　（外部参照シンボルで定義した先頭アドレスを定義）
5.ITSKSP（タスク初期スタックポインタ）：タスク起動時のスタックポインタを定義
　（タスクスタック領域定義部で定義したスタックラベルを定義）

タスク登録を追加する場合は、「TDT_BTM」の前に情報定義を挿入して行います。

(3)　固定長メモリプール登録部

固定長メモリプールを登録するための各種定義を行います。セットアップテーブルの固定長メモリプール登録部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    FIXED-SIZE MEMORYPOOL define section                                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;MB?_CNT_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;MB?_LEN_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;----------------------------------------------------------------------------
MB1_CNT:        .assign 14      ;:[0...65535]   ;: memory block count
MB1_LEN:        .assign 12      ;:[2...65530]   ;: memory block length
;
MB2_CNT:        .assign 14      ;:[0...65535]   ;: memory block count
MB2_LEN:        .assign 12      ;:[2...65530]   ;: memory block length
;
MB3_CNT:        .assign 14      ;:[0...65535]   ;: memory block count
MB3_LEN:        .assign 12      ;:[2...65530]   ;: memory block length
;
MB4_CNT:        .assign 14      ;:[0...65535]   ;: memory block count
MB4_LEN:        .assign 12      ;:[2...65530]   ;: memory block length
;
;------ Usage ---------------------------------------------------------------
;MPF?_TOP_LABEL:.res.b  MEMORYPOOL_SIZE         ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2smpf,data,align=2
MPF1_TOP:       .res.b  MB1_CNT * (MB1_LEN + 4) ;: mpfid1 memorypool area
MPF2_TOP:       .res.b  MB2_CNT * (MB2_LEN + 4) ;: mpfid2 memorypool area
MPF3_TOP:       .res.b  MB3_CNT * (MB3_LEN + 4) ;: mpfid3 memorypool area
MPF4_TOP:       .res.b  MB4_CNT * (MB4_LEN + 4) ;: mpfid4 memorypool area
;
;------ Usage ---------------------------------------------------------------
;LABEL          .data.w BLFCNT, BLFLEN          ;: COMMENT
;               .data.l MPF_TOP_ADDRESS         ;: COMMENT
;----------------------------------------------------------------------------
MPFDTLEN:       .assign 8;<- Not Change !       ;: MPFDT Length
        .section        h2ssetup,code,align=2
_HI_MPFDT:      .equ    $-MPFDTLEN              ;: Fixed-size MemoryPool define table
MPFDT_TOP:      .equ    $                       ;:
mpfdt_id1:      .data.w MB1_CNT, MB1_LEN        ;: blf count, blf length
                .data.l MPF1_TOP                ;: mpf top address
mpfdt_id2:      .data.w MB2_CNT, MB2_LEN        ;: blf count, blf length
                .data.l MPF2_TOP                ;: mpf top address
mpfdt_id3:      .data.w MB3_CNT, MB3_LEN        ;: blf count, blf length
                .data.l MPF3_TOP                ;: mpf top address
mpfdt_id4:      .data.w MB4_CNT, MB4_LEN        ;: blf count, blf length
                .data.l MPF4_TOP                ;: mpf top address
MPFDT_BTM:
MPFCNT:         .equ    (MPFDT_BTM-MPFDT_TOP)/MPFDTLEN
                        ;:[0...255]     ;: Fixed-size Memorypool definition count
;
```

②　①　③　⑤　④

図 3-35　セットアップテーブルの固定長メモリプール登録部

① メモリブロックサイズ、メモリブロック数定義部
カーネルに登録する固定長メモリプールで使用するメモリブロックのサイズとブロック数を定義（使用しているシンボルはこの後の領域確保、定義テーブル情報に使用）
② メモリブロックサイズとブロック数の定義
各固定長メモリプールで使用するメモリブロックのサイズとブロック数を定義
③ 固定長メモリプール領域の確保
メモリブロックサイズとブロック数から各固定長メモリプール領域を確保
④ 固定長メモリプール定義部
カーネルに登録する固定長メモリプールを定義
⑤ 固定長メモリプールの定義
カーネルに登録する各固定長メモリプールの情報を定義

【注】 固定長メモリプール登録部にて使用している、「MB?_CNT」、「MB?_SIZ」、「MPF?_TOP」、「MPFDTLEN」、「MPFDT_TOP」、「MPFDT_BTM」のシンボルに対する変更、削除は行わないでください。
「MPFCNT」定義の行に対する変更、削除は行わないでください。

固定長メモリプールの定義における詳細は以下の通りです。

```
【書式】      LABEL :   .data.w    BLFCNT, BLFLEN
                        .data.l    MPF_TOP_ADDRESS

1.LABEL：自由に設定可（定義がなくても問題なし）
2.BLFCNT（ブロック数）：固定長メモリプールのメモリブロック数を定義
3.BLFLEN（ブロックサイズ）：固定長メモリプールのメモリブロックサイズを定義
4.MPF_TOP_ADDRESS（固定長メモリプールアドレス）：
                                        固定長メモリプールの先頭アドレスを定義
```

固定長メモリプール登録を追加する場合は、「MPFDT_BTM」の前に情報定義を挿入して行います。

(4)　可変長メモリプール登録部

　可変長メモリプールを登録するための各種定義を行います。セットアップテーブルの可変長メモリプール登録部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VARIABLE-SIZE MEMORYPOOL define section                              %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;MPL?_SIZ_LABEL:.assign VALUE    ;:[  RANGE  ]   ;: COMMENT
;--------------------------------------------------------------------------
MPL1_SIZ:       .assign 380     ;:[18.....]     ;: memorypool size
MPL2_SIZ:       .assign 380     ;:[18.....]     ;: memorypool size
MPL3_SIZ:       .assign 380     ;:[18.....]     ;: memorypool size
MPL4_SIZ:       .assign 380     ;:[18.....]     ;: memorypool size
;
;------ Usage ---------------------------------------------------------------
;MPL?_TOP_LABEL:.res.b VARIABLE_MEMORYPOOL_SIZE ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2smpl,data,align=2
MPL1_TOP:       .res.b  MPL1_SIZ                ;: mplid1 memorypool area
MPL2_TOP:       .res.b  MPL2_SIZ                ;: mplid2 memorypool area
MPL3_TOP:       .res.b  MPL3_SIZ                ;: mplid3 memorypool area
MPL4_TOP:       .res.b  MPL4_SIZ                ;: mplid4 memorypool area
;
;------ Usage ---------------------------------------------------------------
;LABEL          .data.l BLKSIZ                  ;: COMMENT
;               .data.l VARIABLE_MEMORYPOOL_TOP ;: COMMENT
;--------------------------------------------------------------------------
MPLDTLEN:       .assign 8;<- Not Change !       ;: MPLDT Length
        .section        h2ssetup,code,align=2
_HI_MPLDT:      .equ    $-MPLDTLEN              ;: Variable-size MemoryPool define table
MPLDT_TOP:      .equ    $                       ;:
mpldt_id1:      .data.l MPL1_SIZ                ;: mpl size
                .data.l MPL1_TOP                ;: mpl top address
mpldt_id2:      .data.l MPL2_SIZ                ;: mpl size
                .data.l MPL2_TOP                ;: mpl top address
mpldt_id3:      .data.l MPL3_SIZ                ;: mpl size
                .data.l MPL3_TOP                ;: mpl top address
mpldt_id4:      .data.l MPL4_SIZ                ;: mpl size
                .data.l MPL4_TOP                ;: mpl top address
MPLDT_BTM:
MPLCNT:         .equ    (MPLDT_BTM-MPLDT_TOP)/MPLDTLEN
                    ;:[0...255]     ;: Variable-size Memorypool definition count
;
```

①　②　③　④

図 3-36　セットアップテーブルの可変長メモリプール登録部

① メモリプールサイズ定義部
カーネルに登録する可変長メモリプールで使用するメモリプールサイズを定義(使用しているシンボルはこの後の領域確保、定義テーブル情報に使用)
② 可変長メモリプール領域の確保
メモリサイズをもとに各可変長メモリプール領域を確保
③ 可変長メモリプール定義部
カーネルに登録する可変長メモリプールを定義
④ 可変長メモリプールの定義
カーネルに登録する各可変長メモリプールの情報を定義

【注】 可変長メモリプール登録部にて使用している、「MPL?_SIZ」、「MPL?_TOP」、「MPLDTLEN」、「MPLDT_TOP」、「MPLDT_BTM」のシンボルに対する変更、削除は行わないでください。「MPLCNT」定義の行に対する変更、削除は行わないでください。

可変長メモリプールの定義における詳細は以下の通りです。

```
【書式】      LABEL :  .data.l   MPL?_SIZ
                       .data.l   MPL?_TOP

1.LABEL：自由に設定可（定義がなくても問題なし）
2.BLKSIZ（ブロックサイズ）：可変長メモリプールのサイズを定義
3.VARIABLE_MEMORYPOOL_TOP（可変長メモリプールアドレス）：
                                        可変長メモリプールの先頭アドレスを定義
```

可変長メモリプール登録を追加する場合は、「MPLDT_BTM」の前に情報定義を挿入して行います。

(5)　周期起動ハンドラ登録部

周期起動ハンドラを登録するための各種定義を行います。セットアップテーブルの周期起動ハンドラ登録部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    cyclic handler define section                                        %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;       .import CYCHDR_TOP_LABEL              ;: COMMENT                    ①
;------------------------------------------------------------------------
;
;------ Usage -------------------------------------------------------------
;LABEL:         .data.w CYC_ACTIVATE          ;: COMMENT
;               .data.l CYC_TIME, CYCHDR_TOP  ;: COMMENT
;------------------------------------------------------------------------
CYHOFF          .assign 0                     ;:initial cycact data = OFF
CYHON           .assign 1                     ;:initial cycact data = ON
CYHDTLEN        .assign 10;<-Dont't Change!   ;:CYHDT length
;
_HI_CYHDT:      .equ    $-CYHDTLEN            ;: cyclic handler define table
CYHDT_TOP:      .equ    $                     ;:
cyhdt_no1:      .data.w CYHOFF                ;: init. cycact data          ③
                .data.l 0, NADR               ;: cyctim, top address
cyhdt_no2:      .data.w CYHOFF                ;: init. cycact data          ②
                .data.l 0, NADR               ;: cyctim, top address
cyhdt_no3:      .data.w CYHOFF                ;: init. cycact data
                .data.l 0, NADR               ;: cyctim, top address
cyhdt_no4:      .data.w CYHOFF                ;: init. cycact data
                .data.l 0, NADR               ;: cyctim, top address
                .aifdef DX
cyhdt_no5:      .data.w CYHON                 ;: init. cycact data          ④
                .data.l 5, HI_DEAMON_MAIN     ;: cyctim, top address
                .aendi
CYHDT_BTM:
CYHCNT:         .equ    (CYHDT_BTM-CYHDT_TOP)/CYHDTLEN
                                ;:[0...255]   ;: cyclic handler definition count
;
```

図 3-37　セットアップテーブルの周期起動ハンドラ登録部

① 使用する周期起動ハンドラ先頭アドレスを、外部参照シンボルとして宣言
② 周期起動ハンドラ定義部
　カーネルに登録する周期起動ハンドラを定義
③ 周期起動ハンドラの定義
　カーネルに登録する各周期起動ハンドラの情報を定義
④ デバッギングエクステンションを使用する場合、デバッグデーモンハンドラが周期起動ハンドラとして登録されます

【注】　周期起動ハンドラ登録部にて使用している、「_HI_CYHDT」、「CYHDTLEN」、「CYHDT_TOP」、「CYHDT_BTM」のシンボルに対する変更、削除は行わないでください。
「CYHCNT」定義の行に対する変更、削除は行わないでください。

周期起動ハンドラの定義における詳細は以下の通りです。

【書式】　　LABEL :　.data.w　CYC_ACTIVATE
　　.data.l　　CYC_TIME, CYCHDR_TOP

1.LABEL：自由に設定可（定義がなくても問題なし）
2. CYC_ACTIVATE（周期起動ハンドラ活性状態）：周期起動ハンドラの活性状態を定義
　(1) CYCOFF(=0)：起動されない（非活性）
　(2) CYCON (=1)：起動される（活性）
3. CYC_TIME（周期起動時間間隔）：周期起動ハンドラを起動する周期時間を定義
4. CYCHDR_TOP（周期起動ハンドラアドレス）：周期起動ハンドラの先頭アドレスを定義

周期起動ハンドラ登録を追加する場合は、「CYHDT_BTM」の前に情報定義を挿入して行います。

(6)　システムコールトレース機能登録部

システムコールトレース機能を登録するための各種定義を行います。セットアップテーブルのシステムコール機能登録部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    SVC trace define section                                      %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------
;TRC_CNT:.assign   TRACE COUNT                   ;: COMMENT
;TRC_BUF:.assign   TRACE BUFFER ADDRESS          ;: COMMENT
;------------------------------------------------------------------------
      .section        h2strc,data,align=2
TRC_CNT:       .assign 4                    ;: trace count           ← ①
TRC_BUF:       .res.b  16+(TRC_CNT*28)      ;: trace buffer address  ← ②
;
;------ Usage -----------------------------------------------------------
;INITRC        .data.l TRACE BUFFER ADDRESS ;: COMMENT
;              .data.w TRACE COUNT          ;: COMMENT
;------------------------------------------------------------------------
      .section        h2ssetup,code,align=2
INITRC:        .equ    $                    ;:                       ┐
               .data.l TRC_BUF              ;: trace buffer address  ├ ← ③
               .data.w TRC_CNT              ;: trace count           ┘
;
```

図 3-38　セットアップテーブルのシステムコールトレース機能登録部

① TRC_CNT【最大トレース情報取得数】
　システムコールトレース機能で取得するトレース情報の最大値を指定

② TRC_BUF【トレース情報取得領域の確保】
　システムコールトレース機能で取得するトレース情報を格納する領域の確保

③ システムコールトレース機能の定義
　システムコールトレース機能の情報を定義

【注】　システムコールトレース機能登録部にて使用しているシンボルに対する変更、削除は行わないでください。

システムコールトレース機能の定義における詳細は以下の通りです。

【書式】　INITRC :　.data.l　TRACE BUFFER ADDRESS
　　　　　　　　　　.data.l　TRACE_COUNT

1.INITRC：システムコールトレース機能情報定義シンボル
2.TRACE BUFFER ADDRESS（システムコールトレース機能用トレースバッファアドレス）：
　システムコールトレース機能で使用するトレース取得領域の先頭アドレスを定義
3. TRACE_COUNT（システムコールトレース機能用トレース取得数）：
　システムコールトレース機能で取得するトレース情報数を定義

(7)　拡張情報登録部

カーネルの各種機能（同期通信機能、時間管理機能など）を使用するために必要な情報を登録します。セットアップテーブルの定数定義部を以下に示します。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Task Extended Information define section                     %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------
;LABEL          .data.l TSK?_EXINF             ;: COMMENT
;-----------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_TSKEXINF:   .equ    $-EXLEN                ;: TSK exinf define area
TSKE_TOP:       .equ    $                      ;:
tsk1_exinf:     .data.l 00000000               ;: tskid=1 exinf  ┐
tsk2_exinf:     .data.l 00000000               ;: tskid=2 exinf  │
tsk3_exinf:     .data.l 00000000               ;: tskid=3 exinf  ├①
tsk4_exinf:     .data.l 00000000               ;: tskid=4 exinf  │
tsk5_exinf:     .data.l 00000000               ;: tskid=5 exinf  ┘
TSKE_BTM:
TSKECNT:        .equ    (TSKE_BTM-TSKE_TOP)/EXLEN
                                ;:[0...255]    ;: tsk exinf count
;
```

図 3-39　セットアップテーブルのタスク拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Event Flag Extended Information define section               %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------
;LABEL          .data.l FLG?_EXINF             ;: COMMENT
;-----------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_FLGEXINF:   .equ    $-EXLEN                ;: FLG exinf define area
FLGE_TOP:       .equ    $                      ;:
flg1_exinf:     .data.l 00000000               ;: flgid=1 exinf  ┐
flg2_exinf:     .data.l 00000000               ;: flgid=2 exinf  ├①
flg3_exinf:     .data.l 00000000               ;: flgid=3 exinf  │
flg4_exinf:     .data.l 00000000               ;: flgid=4 exinf  ┘
FLGE_BTM:
FLGECNT:        .equ    (FLGE_BTM-FLGE_TOP)/EXLEN
                                ;:[0...255]    ;: flg exinf count
;
```

図 3-40　セットアップテーブルのイベントフラグ拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Semaphore Extended Information define section                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ------------------------------------------------------------
;LABEL          .data.l SEM?_EXINF              ;: COMMENT
;------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_SEMEXINF:   .equ    $-EXLEN                 ;: SEM exinf define area
SEME_TOP:       .equ    $                       ;:
sem1_exinf:     .data.l 00000000                ;: semid=1 exinf  ┐
sem2_exinf:     .data.l 00000000                ;: semid=2 exinf  │
sem3_exinf:     .data.l 00000000                ;: semid=3 exinf  ├ ①
sem4_exinf:     .data.l 00000000                ;: semid=4 exinf  ┘
SEME_BTM:
SEMECNT:        .equ    (SEME_BTM-SEME_TOP)/EXLEN
                                ;:[0...255]     ;: sem exinf count
;
```

図 3-41　セットアップテーブルのセマフォ拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Mailbox Extended Information define section                    %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ------------------------------------------------------------
;LABEL          .data.l MBX?_EXINF              ;: COMMENT
;------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MBXEXINF:   .equ    $-EXLEN                 ;: MBX exinf define area
MBXE_TOP:       .equ    $                       ;:
mbx1_exinf:     .data.l 00000000                ;: mbxid=1 exinf  ┐
mbx2_exinf:     .data.l 00000000                ;: mbxid=2 exinf  │
mbx3_exinf:     .data.l 00000000                ;: mbxid=3 exinf  ├ ①
mbx4_exinf:     .data.l 00000000                ;: mbxid=4 exinf  ┘
MBXE_BTM:
MBXECNT:        .equ    (MBXE_BTM-MBXE_TOP)/EXLEN
                                ;:[0...255]     ;: mbx exinf count
;
```

図 3-42　セットアップテーブルのメイルボックス拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Fixed-size MemoryPool Extended Information define section    %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------------
;LABEL          .data.l MPF?_EXINF              ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPFEXINF:   .equ    $-EXLEN                 ;: MPF exinf define area
MPFE_TOP:       .equ    $                       ;:
mpf1_exinf:     .data.l 00000000                ;: mpfid=1 exinf      ┐
mpf2_exinf:     .data.l 00000000                ;: mpfid=2 exinf      ├ ①
mpf3_exinf:     .data.l 00000000                ;: mpfid=3 exinf      ┘
mpf4_exinf:     .data.l 00000000                ;: mpfid=4 exinf
MPFE_BTM:
MPFECNT:        .equ    (MPFE_BTM-MPFE_TOP)/EXLEN
                                ;:[0...255]     ;: mpf exinf count
;
```

図 3-43　セットアップテーブルの固定長メモリプール拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Variable-size MemoryPool Extended Information define section %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------------
;LABEL          .data.l MPL?_EXINF              ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPLEXINF:   .equ    $-EXLEN                 ;: MPL exinf define area
MPLE_TOP:       .equ    $                       ;:
mpl1_exinf:     .data.l 00000000                ;: mplid=1 exinf      ┐
mpl2_exinf:     .data.l 00000000                ;: mplid=2 exinf      ├ ①
mpl3_exinf:     .data.l 00000000                ;: mplid=3 exinf      ┘
mpl4_exinf:     .data.l 00000000                ;: mplid=4 exinf
MPLE_BTM:
MPLECNT:        .equ    (MPLE_BTM-MPLE_TOP)/EXLEN
                                ;:[0...255]     ;: mpl exinf count
;
```

図 3-44　セットアップテーブルの可変長メモリプール拡張情報登録部

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Cyclic Handler Extended Information define section                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL        .data.l CYH?_EXINF             ;: COMMENT
;--------------------------------------------------------------------------
        .section      h2ssetup,code,align=2
_HI_CYCEXINF: .equ    $-EXLEN                ;: CYH exinf define area
CYHE_TOP:     .equ    $                      ;:
cyh1_exinf:   .data.l 00000000               ;: cyhno=1 exinf        ┐
cyh2_exinf:   .data.l 00000000               ;: cyhno=2 exinf        ├ ①
cyh3_exinf:   .data.l 00000000               ;: cyhno=3 exinf        ┘
cyh4_exinf:   .data.l 00000000               ;: cyhno=4 exinf
              .aifdef DX
cyh5_exinf:   .data.l 00000000               ;: cyhno=5 exinf ☜ ―― ②
              .aendi
CYHE_BTM:
CYHECNT:      .equ    (CYHE_BTM-CYHE_TOP)/EXLEN
                                ;:[0...255]  ;: cyh exinf count
;
```

図 3-45　セットアップテーブルの周期起動ハンドラ拡張情報登録部

① 拡張情報の定義
　各オブジェクトに登録する拡張情報を定義
② デバッギングエクステンションを使用する場合、デバッグデーモンハンドラ用の周期起動ハンドラに登録する拡張情報が定義されます

【注】　拡張情報登録部にて使用しているシンボルに対する変更、削除はしないでください。

拡張情報登録を追加する場合は、各「???E_BTM」の前に定義を挿入して行います。

3.2.4　セットアップテーブルに関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたセットアップテーブルに関する質問についての回答を記述します。

《 FAQ 目次 》

(1)　セットアップテーブルの最適化

分類：構築関連、カーネル環境定義、セットアップテーブル

HI2000/3

■質問

インストール時にできるファイルをそのまま使用して構築したところ、エラーが発生し構築ができません。何が原因か教えてください。

■回答

セットアップテーブルが最適化対象になっているため、エラーが発生します。

セットアップテーブルは最適化対象としないでください。

セットアップテーブルは、定義された内容をもとに、カーネルが必要な情報（データテーブル）の生成、および定義値をもとにカーネルが使用するメモリ領域の確保（カーネルが使用するスタックサイズの算出など）を行います。コード（プログラム）は一切記述されていませんので、最適化対象としても、コードサイズやスピード（性能）などに全く影響することはありません。

最適化対象としてアセンブルした場合、最適化用の処理でエラーとなってしまいます。

3.3　スタック使用量の算出

タスクや割込みハンドラのスタック使用量は、以下の手順で算出します。

1. タスクや割込みハンドラを構成する関数のスタック使用量を求める
2. プログラムのネストを加味したスタック使用量を求める

3.3.1　スタックフレームサイズによるスタック使用量の算出

C 言語関数の場合は、関数起動時にスタックフレームをスタック上に確保します。

スタックフレームは、関数の局所変数や関数呼び出し時の引数領域として利用されます。

このスタックフレームサイズは、C コンパイラが出力するコンパイルリスト上の「frame size」から取得できます。

C コンパイラでは、HI シリーズ OS 用サービスコール使用時のスタックサイズは把握できないため、コンパイルリスト上の「frame size」にサービスコール使用時のスタックサイズを加算する必要があります。

3.3.2　CallWalker によるスタック使用量の算出

C コンパイラ付属ツール［CallWalker］を使用してスタックサイズを算出する事ができます。
以下に［CallWalker］を使用したタスクスタックサイズの算出例を示します。
以下の算出例では、HI7750/4、SuperH™ RISC engine Series C/C++ Compiler Package Ver8.0.01、HEW ワークスペースのサブプロジェクトとして SH7770 用一括リンクプロジェクト（7770_mix）を使用して説明します。

(1)　HEW の起動

HEW を起動し、HI7750/4 インストールフォルダの［¥kernel¥for_shc8¥hios¥hios.hws］をオープンし、7770_mix をカレントプロジェクトとして選択します。

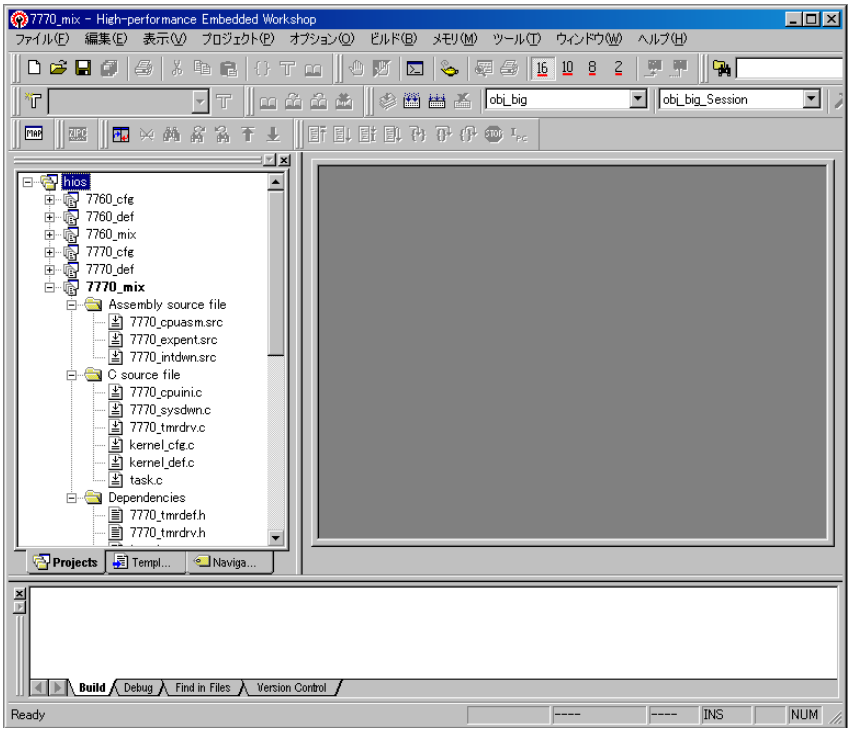


図 3-46　HEW 起動画面

カレントプロジェクト設定後の画面で、ヘッダメニュー［オプション(O)］→［SuperH RISC engine Standard Toolchain…］を選択し、HEW のオプション設定メニューを選択します。

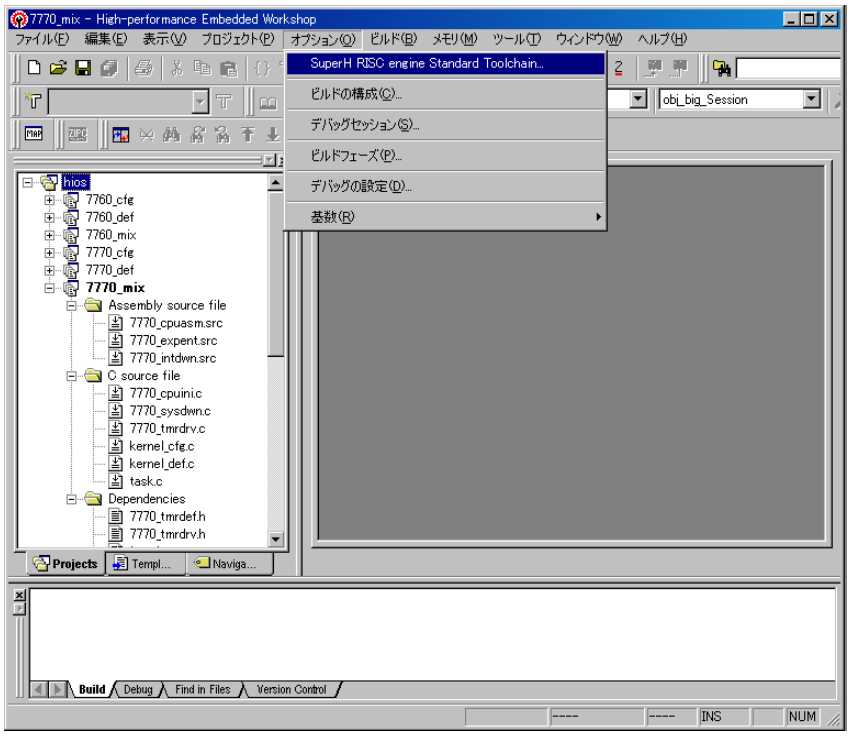


図 3-47　メニュー選択画面

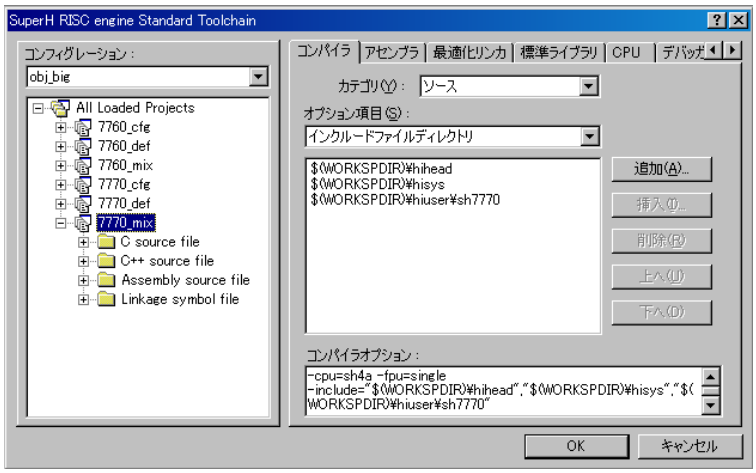


図 3-48　HEW オプション選択画面

「最適化リンカ」タブにてカテゴリに「その他」を選択し、「スタック情報ファイル(sni)出力」をチェックします。

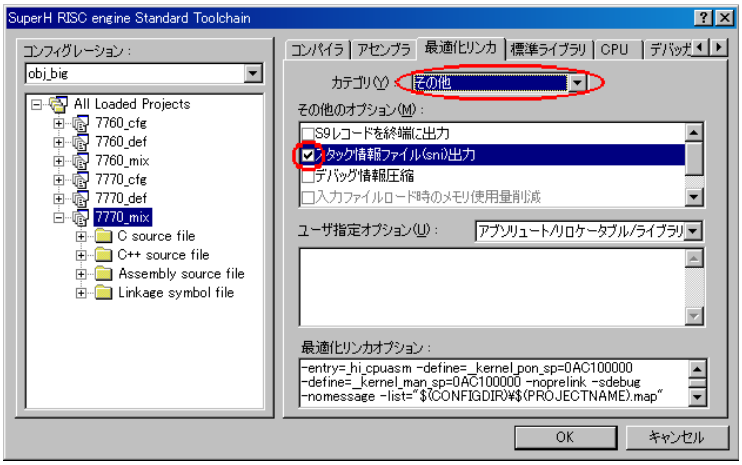


図 3-49　HEW オプション設定内容

「OK」をクリックして設定を終了し、ビルドを実行します。

(2)　CallWalker の起動

スタートメニューから［プログラム(P)］→［Renesas High-performance Embedded Workshop］→［CallWalker］を選択し起動します。

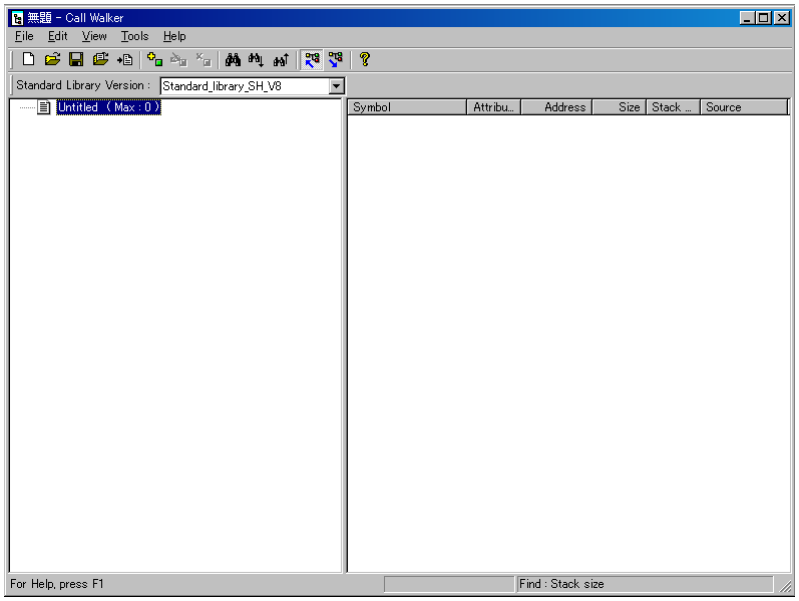


図 3-50　CallWalker 起動画面

起動画面のヘッダメニュー［File］→［Import Stack File］を選択し、生成したスタック情報ファイルを開きます。

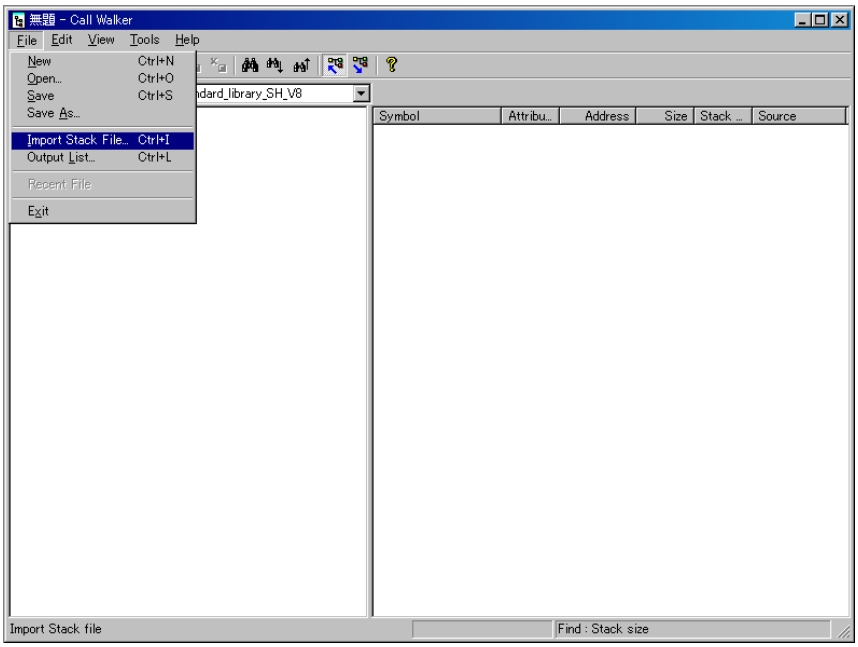


図 3-51　ファイル読み込み画面

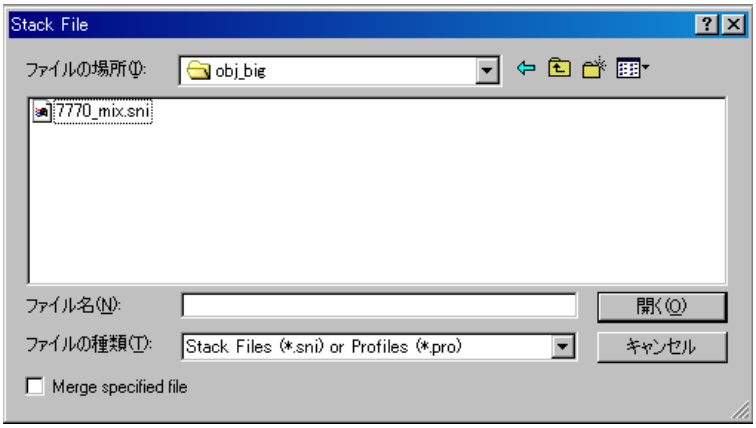


図 3-52　読み込みファイル選択画面

(3)　タスクスタックサイズの算出例

本例ではシステムは、次のようなアプリケーションプログラムで構成されています。

表 3-21　例題システムの構成

項番	関数名	アプリケーション種別	備考
1	_hi_cpuini	CPU 初期化ルーチン	
2	_kernel_reset	（vsta_knl の呼び出しを示します）	使用サイズは 0 で計算
3	_inithdr1	初期化ルーチン	
4	_MainTask	タスク	
5	_texrtn1	_MainTask 用タスク例外処理ルーチン	
6	_sub1	_MainTask からコールされる関数	
7	_Task7	タスク	
8	_svchdr1	拡張サービスコールハンドラ	_MainTask からコール
9	_inthdr_level1	割込みハンドラ（割込みレベル 1）	
10	_inthdr_level5	割込みハンドラ（割込みレベル 5）	
11	_kernel_tmrini	タイマドライバ（タイマ初期化ルーチン）	初期化ルーチン
12	_kernel_tmrint	タイマドライバ（タイマ割込みルーチン）	割込みハンドラ
13	_cychdr1	周期ハンドラ	
14	_kernel_sysdwn	システムダウン	

上記アプリケーションはスタティックスタック使用無し、コプロセッサ使用無しとし、CFG_TRACE のチェックも無しとして説明します。

上記アプリケーションにおける CallWalker のスタックサイズ表示例を以下に示します。

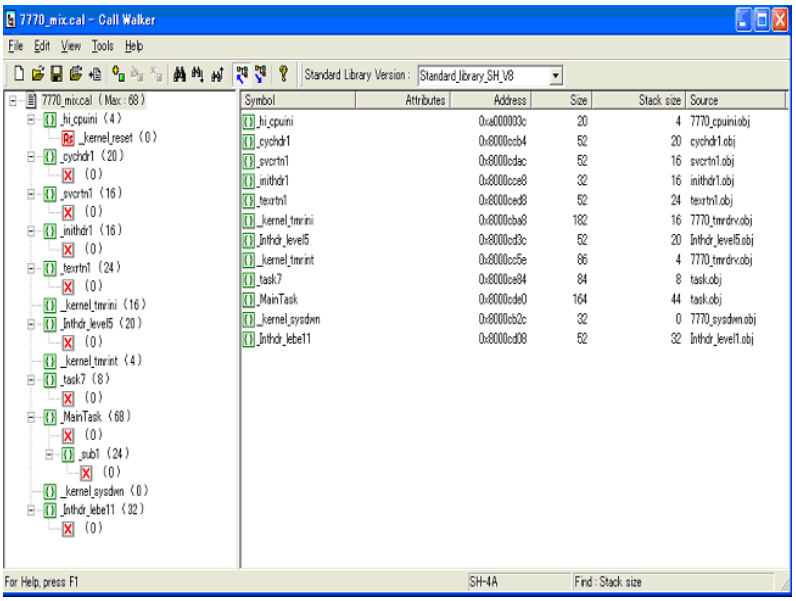


図 3-53　CallWalker によるスタックサイズ表示例

表示された情報からスタックサイズを計算します。CallWalker が表示するスタックサイズは、タスクや割込みハンドラが独自で使用するスタックサイズです。この表示されたサイズに対してカーネルの「必須サイズ」を加算して、その合計が求めるスタックサイズとなります。以下に、CallWalker によるスタックサイズ表示例をもとに、各スタックサイズを求めます。

タスク「_MainTask」を例にスタックサイズを計算します。
_MainTask は、以下の関数、およびサービスコールを呼び出しています。またタスク例外処理ルーチンを定義しています

- _sub1
- 拡張サービスコールルーチン（_svcrtn1）
- タスク例外処理ルーチン（_texrtn1）

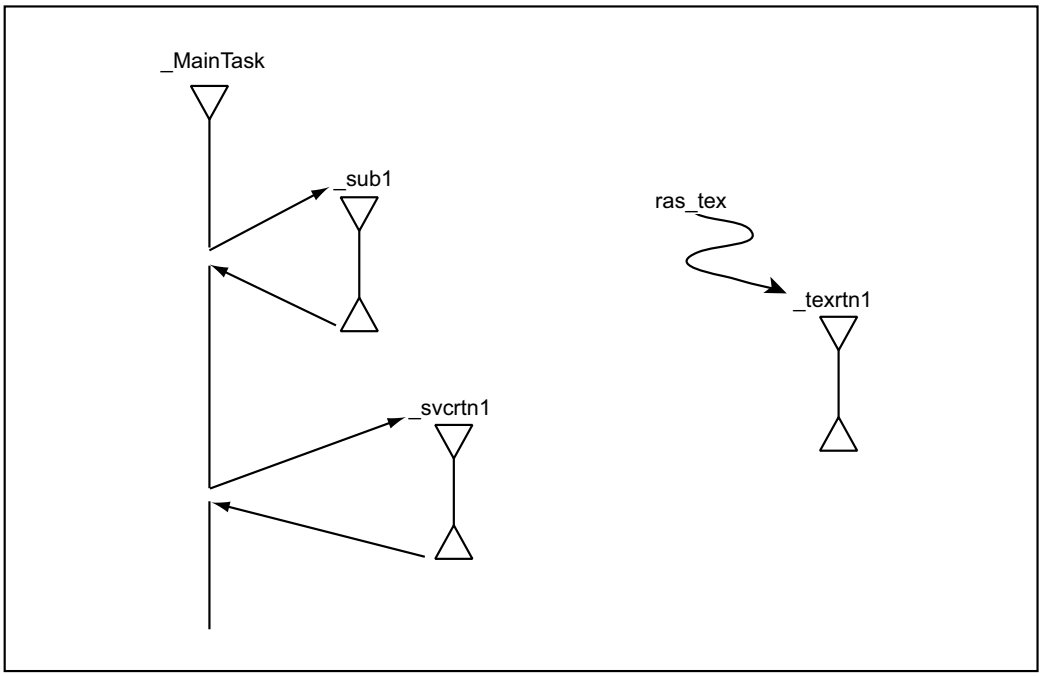


図 3-54　サンプルタスクの処理概要

_MainTask は、CallWakler の呼び出し情報ビューから_sub1 関数をコールしているのが分かります。しかし、その他の呼び出し（サービスコールなど）情報は分かりません。この 2 つの情報は CallWalker では表示できませんので、手動で計算を行ってください。

_MainTask と_sub1 のスタックサイズは、呼び出し情報ビューとシンボル詳細ビューからそれぞれのサイズが分かります。これらの値に、その他の拡張サービスコール、タスク例外処理ルーチンは手動で加算します。

タスク「_MainTask」単体のスタックサイズは次のようになります。

表 3-22　タスク「_MainTask」単体のスタックサイズ

項番	関数名	使用サイズ
1	_MainTask	44 バイト
2	_sub1	24 バイト
3	_svcrtn1	16 バイト
4	_texrtn1	24 バイト＋152 バイト *1
合　計		260 バイト

【注】 *1　呼び出しルーチン、ハンドラの加算(必須)サイズです。詳細については、「HI7000/4 シリーズ　ユーザーズマニュアル」を参照してください。

ここで求めた値は、タスク「_MainTask」のタスク単体の使用サイズです。

この値を HI7000/4 シリーズユーザーズマニュアル「表 C.5　個々のタスクのスタック使用量」の項番 1 に代入します。タスク「_MainTask」のスタックサイズは次のようになります。

表 3-23　タスクスタックサイズの算出

項番	項目		使用サイズ
1	求めたサイズ		260 バイト
2	必須分		196 バイト
3	タスク	TA_COP0 属性	ー
4		TA_COP1 属性	ー
5		TA_COP2 属性	ー
6		スタティックスタックを使用	ー
7	CFG_TRACE をチェック		ー
8	多重割込み加算値		ー
合　計			456 バイト

(4)　割込みハンドラスタックサイズの算出例

この例では、割込みハンドラを 2 つ使用しています。

- inthdr_level1
- inthdr_level5

また、タイマを使用しています。これらの割込みレベルはすべて異なるので、それぞれのスタックサイズを求める必要があります。つまり、これらの割込みは割込みのネストを考慮する必要があります。

それぞれのスタック使用サイズを HI7000/4 シリーズユーザーズマニュアル「表 C.6　個々の割込みハンドラのスタック使用量」の項番 1 に代入します。

表 3-24　割込みハンドラスタックサイズの算出

項番	項目	使用サイズ		
		_Inthdr_level1	_Inthdr_level5	_kernel_tmrint
1	求めたサイズ	32 バイト	20 バイト	4 バイト
2	割込みハンドラから サービスコールを呼び出す	192 バイト	192 バイト	192 バイト
3	CFG_TRACE をチェック	－	－	－
4	多重割込み加算値	－	－	－
合　計		224 バイト	212 バイト	196 バイト

これらの値から指定する割込みハンドラスタックサイズを求めます。
HI7000/4 シリーズのユーザーズマニュアルから、

CFG_IRQSTKSZ = ∑（各割込みレベルで最も多く使用するハンドラの使用量）+ 28
+ （C.4 節, C.5 節にしたがって求めた NMI 割込みハンドラの使用量 + 48）× NMI のネスト数

に代入すると、

CFG_IRQSTKSZ = 224 + 212 + 196 + 28 + 0（NMI 無しとします）
= 660 バイト

となります。

(5)　タイムイベントハンドラスタックサイズ

この例では、周期ハンドラ（_cychdr1）を 1 つ使用しています。
この値を、HI7000/4 シリーズのユーザーズマニュアル「表 C.7　タイムイベントハンドラのスタック使用量」の項番 1 に代入します。

表 3-25　タイムイベントハンドラスタックサイズの算出

項番	項目	使用サイズ
1	求めたサイズ	20 バイト
2	必須分	192 バイト
3	タイムイベントハンドラからサービスコールを呼び出す	144 バイト
4	CFG_TRACE をチェック	－
5	多重割込み加算値	－
6	NMI 使用時の加算値	－
合　計		356 バイト

この例では、タイムイベントハンドラとして周期ハンドラを 1 つのみ使用していますが、タイムイベントハンドラを複数使用する場合は、その中で最もスタックを使用するハンドラのサイズを使用して計算します。

(6)　初期化ルーチンスタックサイズ

この例では、初期化ルーチン（inithdr1）を使用しています。

しかし、タイマドライバを使用しているので、実際はタイマドライバのタイマ初期化ハンドラ：_kernel_tmrini と 2 つ使用していることになります。

したがってこの 2 つの内で使用サイズの多い方の値を使用して計算します。

表 3-26　初期化ルーチンスタックサイズの算出

項番	項目	使用サイズ
1	求めたサイズ	16 バイト
2	必須分	192 バイト
3	CFG_TRACE をチェック	－
4	NMI 使用時の加算値	－
合　計		208 バイト

(7)　CallWalker 使用上の注意事項

CallWalker を使用する場合の注意事項を以下に示します。

- CallWalker の Tools メニューー＞「RealTime OS Option…」機能は、現在未サポートです。
- アセンブリ言語記述の関数は CallWalker では計算されませんので、手動で計算してください。
- 以下の関数も CallWalker では計算されませんので、上記同様に手動で計算してください。
  - 再帰呼び出し関数
  - 循環関数
  - 参照元シンボル不明関数
  - アドレス参照未解決関数

タスク開始関数、タスク例外処理ルーチン開始関数など、アプリケーションプログラムの開始関数がアセンブリ言語記述の関数は、CallWalker の呼び出し情報ビューに表示されない場合がありますので注意してください。

3.4　システム構築手順

HI シリーズ OS を使用したシステムの構築は、HEW（High-performance Embedded Workshop）を使用して行います。

以下にシステム構築概要を示します。

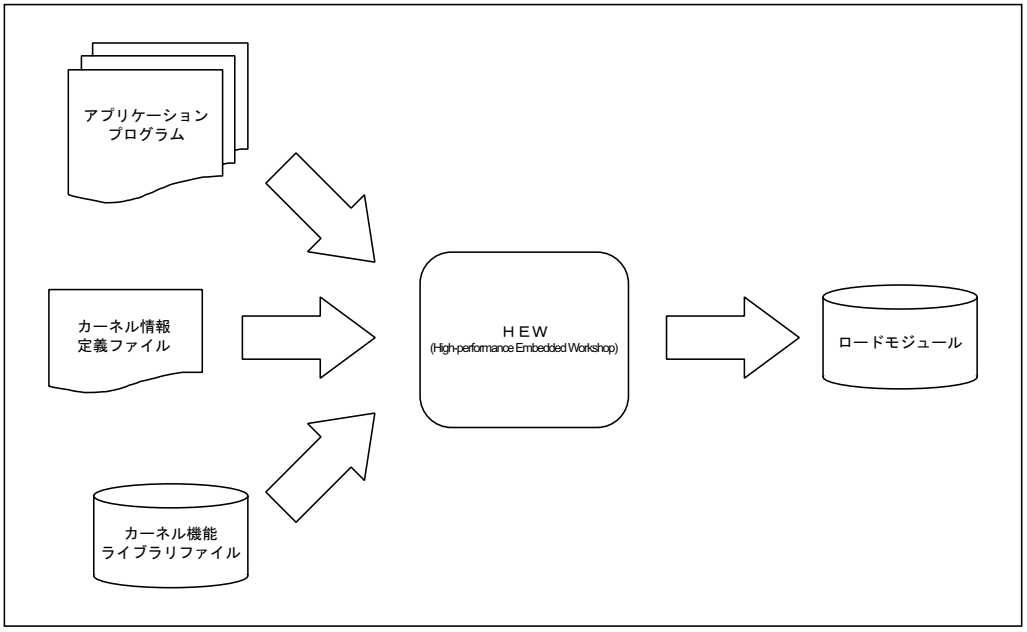


図 3-55　システム構築手順

各 HI シリーズ OS では、標準提供しているサンプルプログラム用の HEW 構築ファイル（HEW ワークスペース）を提供しております。

標準提供の HEW 構築ファイルを使用した構築手順を以下に示します。

3.4.1 HI7000/4

HI7000/4 の構築手順については、SuperH™ RISC engine Series C/C++ Compiler Package Ver6.0AR2、SH7612 用 HEW 構築ファイル（HEW ワークスペースと称す）、一括リンク方式を使用した構築ガイド書を提供しておりますので、(株)ルネサステクノロジのホームページからダウンロードできます。

3.4.2 HI7700/4

HI7700/4 の構築手順については、SuperH™ RISC engine Series C/C++ Compiler Package Ver6.0AR2、SH7729 用 HEW 構築ファイル（HEW ワークスペースと称す）、一括リンク方式を使用した構築ガイド書を提供しておりますので、(株)ルネサステクノロジのホームページからダウンロードできます。

3.4.3 HI7750/4

HI7750/4 の構築手順については、SuperH™ RISC engine Series C/C++ Compiler Package Ver6.0AR2、SH7750 用 HEW 構築ファイル（HEW ワークスペースと称す）、一括リンク方式を使用した構築ガイド書を提供しておりますので、(株)ルネサステクノロジのホームページからダウンロードできます。

3.4.4　HI2000/3

HEW による構築方法を以下に示します。
当例では、「H8S,H8/300 Series C/C++ Compiler Package Ver4.0Ar2」を使用して説明します。

HI2000/3 インストールフォルダ［product］内のサンプルワークスペースファイル「product.hws」をダブルクリックすると、HI2000/3 を構築するための HEW が起動されます。以下に HEW 起動画面を示します。

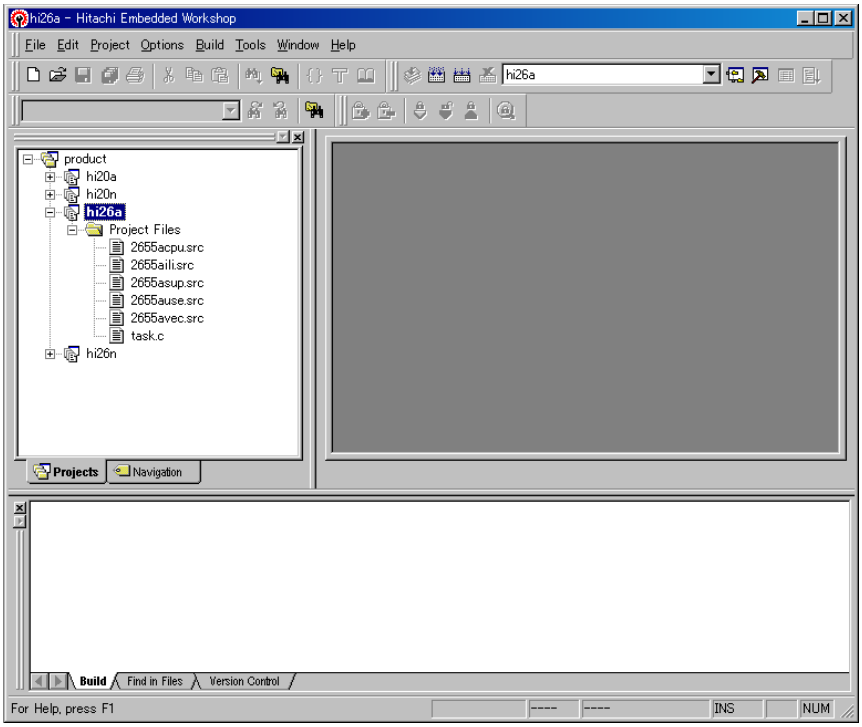


図 3-56　HEW 起動画面

ワークスペース「product.hws」には、各種デバイスに対応したサンプルのプロジェクトがあらかじめ登録されています。
次に示すように、CPU と動作モードに対応した 4 種類のサンプルプロジェクトがあります。
ユーザの使用する環境（CPU、動作モード）に合ったプロジェクトを選択し、以降の説明を参考に設定を変更してください。

表 3-27　標準サンプルプロジェクト

項番	プロジェクト名	コンフィギュレーション *1	内　容
1	hi26a	hi26a	H8S/2600CPU、アドバンストモード用 ロードモジュール生成用プロジェクト （H8S/2655 用に登録済）
2	hi26n	hi26n	H8S/2600CPU、ノーマルトモード用 ロードモジュール生成用プロジェクト （H8S/2655 用に登録済）
3	hi20a	hi20a	H8S/2000CPU、アドバンストモード用 ロードモジュール生成用プロジェクト （H8S/2655 用に登録済）
4	hi20n	hi20n	H8S/2000CPU、ノーマルトモード用 ロードモジュール生成用プロジェクト （H8S/2655 用に登録済）

【注】 *1　コンフィギュレーション内にロードモジュールが生成されるように設定してあります。

サンプルプロジェクトを選択するには、HEW のワークスペースウィンドウでプロジェクトを選び、ポップアップメニューで［Set as Current Project］を選択してください。

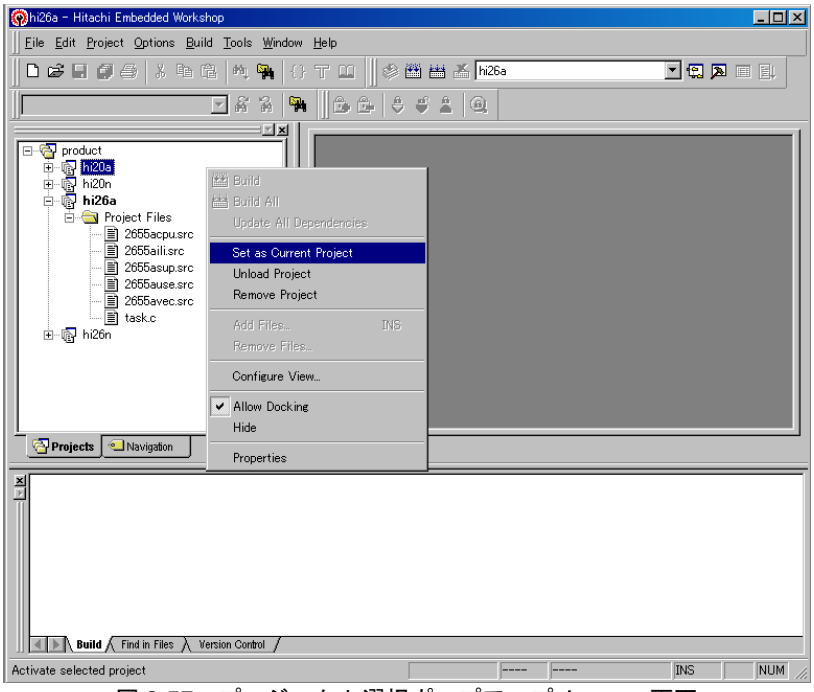


図 3-57　プロジェクト選択ポップアップメニュー画面

なお、使用しない環境用のプロジェクトは削除しても構いません。

H8S/2655、H8S/2245 以外のデバイスを使用する場合、プロジェクトの選択後、既に登録されているシステム構築用ファイルを、使用する CPU 用ファイルに変更してください。
また、**「2 章　アプリケーション作成手法」**で作成した各アプリケーションプログラムを、プロジェクトファイルに定義（追加）します。以下にファイル追加手順を示します。

カレントプロジェクト設定後の画面で、ヘッダメニュー［Project］→［Add Files…］を選択し、プロジェクトファイルに、作成したアプリケーションプログラムファイルを追加します。

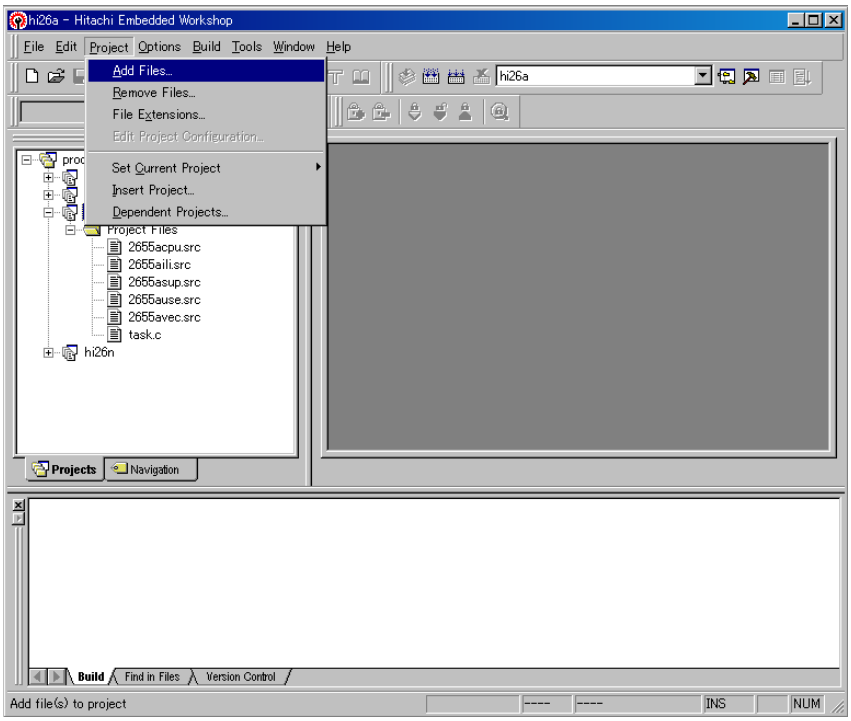


図 3-58　ファイル追加メニュー画面

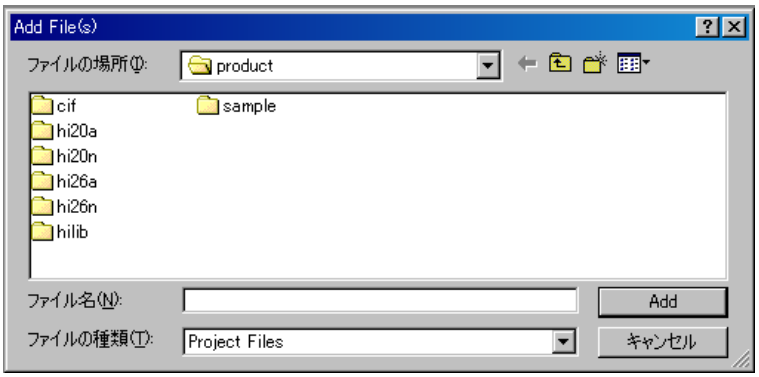


図 3-59　追加ファイル選択画面

追加ファイル選択画面で、追加するファイルを格納したフォルダに移動し、Shift キーを押しながらファイルを選択する事で、一度に複数のファイルを指定することも出来ます。

追加ファイルのセクション情報を定義します。
ヘッダメニュー［Options］→［Optlinker…］を選択し、「Optlinker Options(hi26a)」ダイアログの［Section］タグを選択し、セクション情報の追加設定を行います。

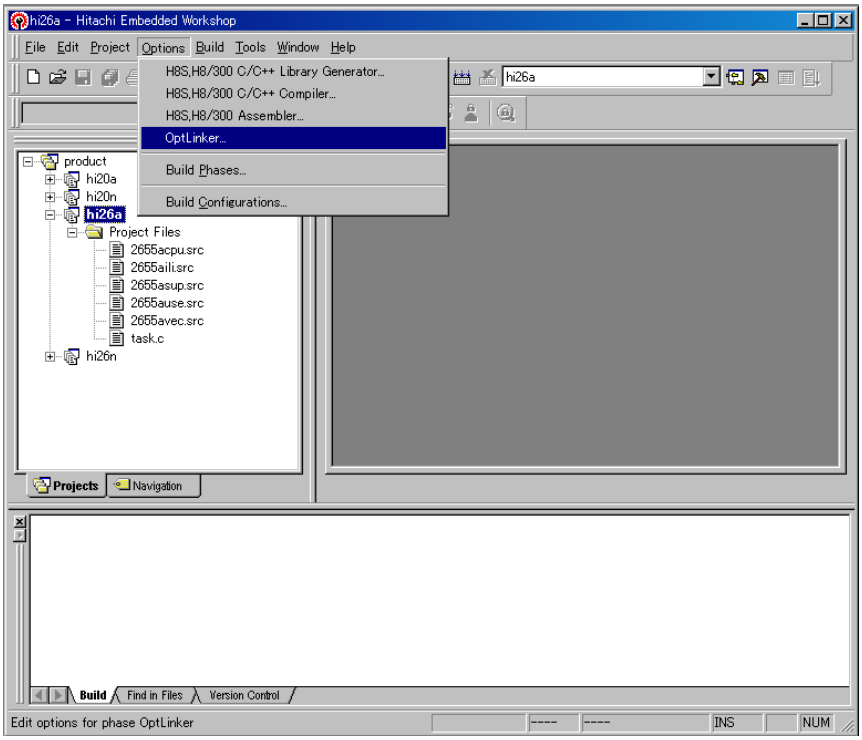


図 3-60　Optlinker 選択メニュー画面

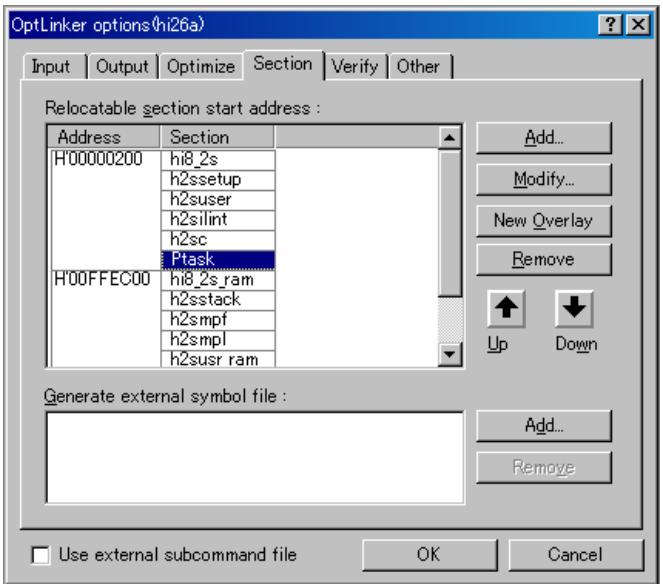


図 3-61　セクション情報追加画面

セクションの追加方法について説明します。追加したアプリケーションファイルのプログラムセクション「P_section」の追加を例に示します。

［Ptask］を選択し、「Add」ボタンを押します。

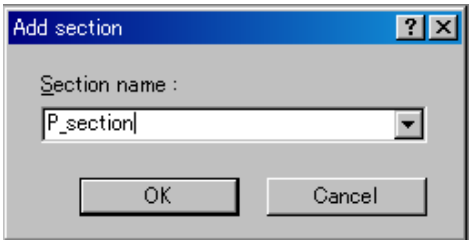


図 3-62　追加セクション情報入力画面

「Add section」ダイアログの［Section name：］に「P_section」を入力し、「OK」ボタンを押下すると、［Ptask］セクションの下に追加した［P_section］セクションが表示されます。

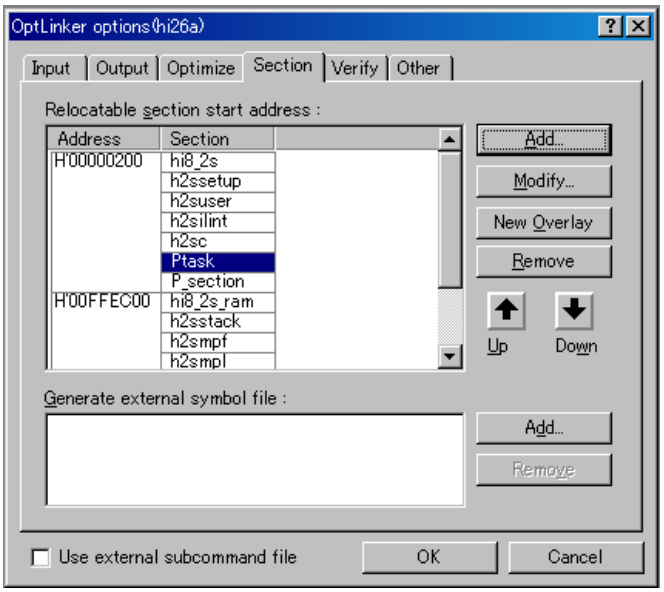


図 3-63　追加セクション情報確認画面

更新したセクション情報を反映するため、「OK」ボタンを押下します。

次に、ヘッダメニュー［Build］→［Build］を選択し、システムの構築（ビルド）を行います。

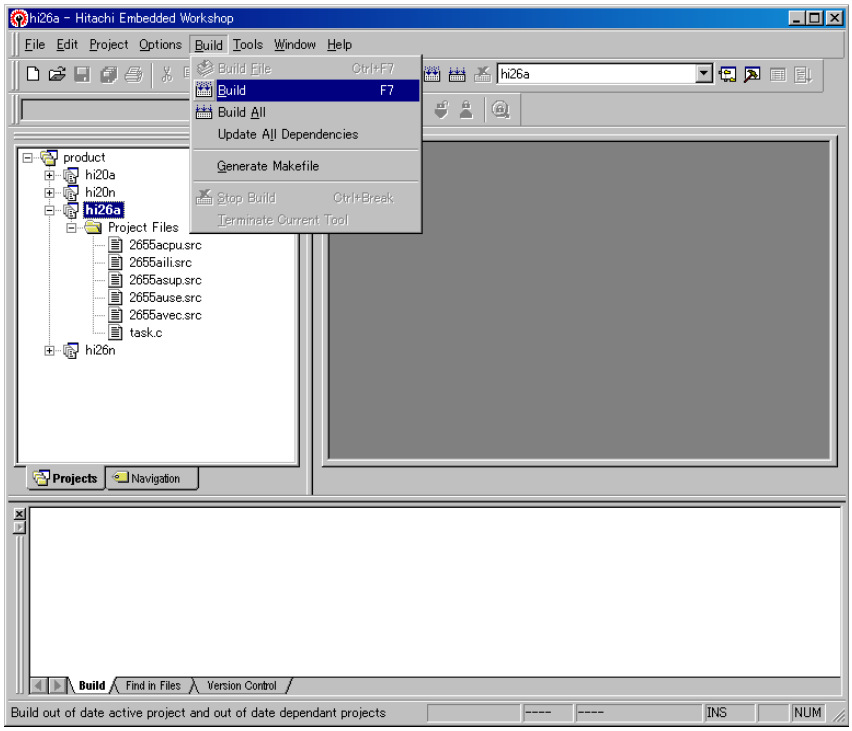


図 3-64　Build 実行画面

以上の操作で実行形式ファイルが生成されます。

なお、コンパイル、アセンブル、リンクの結果が最下位部のウィンドウに表示されますので、エラーが発生した場合は、該当するソースプログラムを修正した後、再度、ビルドを行ってください。

実行形式ファイル（拡張子.abs）は、当該プロジェクトで指定されているコンフィギュレーションで指定したフォルダ（［project］フォルダ下の当該プロジェクト名フォルダ）に生成されます。

標準構成によるビルドでは、パラメータチェック機能あり、共有スタック機能ありのカーネルライブラリを使用しています。

アプリケーションプログラムのデバッグが完了し、実際の製品に組み込むレベルに仕上がった場合、システムコールの先頭で行うパラメータチェック機能は無駄なルーチンになるため、HI2000/3 では、このパラメータチェック機能を取り外す事ができるようになっています。

パラメータチェック機能の取り外し方法については、「1.3.2　HI2000/3 における組み込み方法」を参照してください。

3.4.5　HI1000/4

HEW による構築方法を以下に示します。
当例では、「H8S,H8/300 Series C/C++ Compiler Package Ver6.0.00」を使用して説明します。

HI1000/4 インストールフォルダ［product］内のサンプルワークスペースファイル「product.hws」をダブルクリックすると、HI1000/4 を構築するための HEW が起動されます。以下に HEW 起動画面を示します。

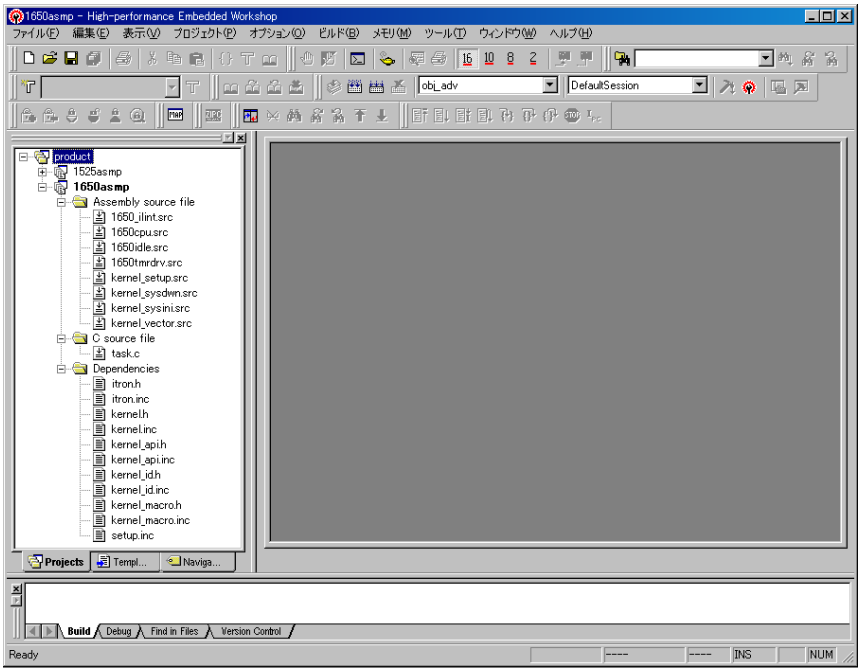


図 3-65　HEW 起動画面

ワークスペース「product.hws」には、各種デバイスに対応したサンプルのプロジェクトがあらかじめ登録されています。以下に示すように、CPU と動作モードに対応した 2 種類のサンプルプロジェクトがあります。ユーザの使用する環境（CPU、動作モード）に合ったプロジェクトを選択し、以降の説明を参考に設定を変更してください。

- 1650asmp：H8SX/1650、アドバンストモード用ロードモジュール生成プロジェクト
- 1525asmp：H8SX/1525、アドバンストモード用ロードモジュール生成プロジェクト

サンプルプロジェクトを選択するには、HEW のワークスペースウィンドウでプロジェクトを選び、ポップアップメニューで［アクティブプロジェクトに設定］を選択してください。

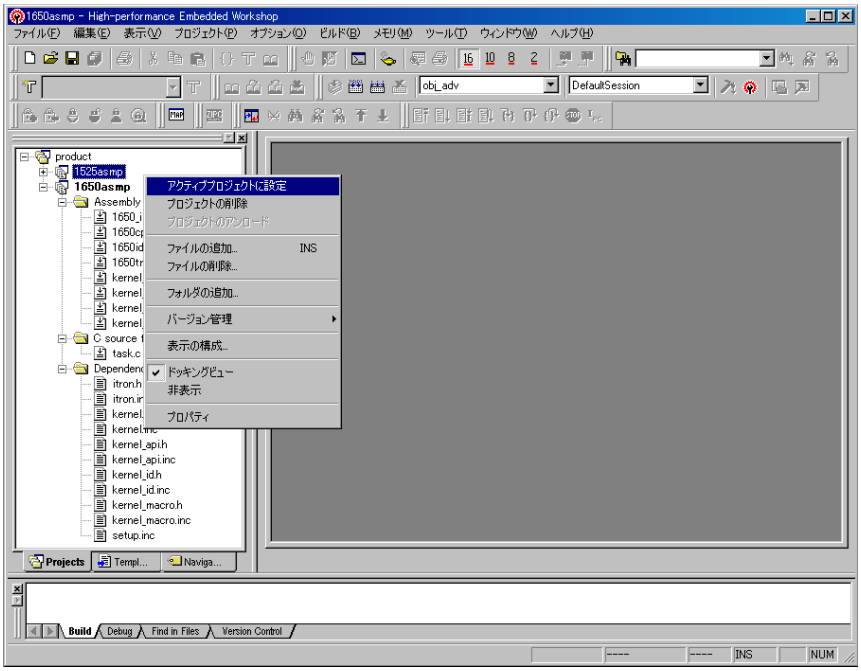


図 3-66　プロジェクト選択ポップアップメニュー画面

なお、使用しない環境用のプロジェクトは削除しても構いません。

「2 章　アプリケーション作成手法」で作成した各アプリケーションプログラムを、プロジェクトファイルに定義（追加）します。以下にファイル追加手順を示します。

カレントプロジェクト設定後の画面で、ヘッダメニュー［プロジェクト(P)］→［ファイルの追加(A)］を選択し、プロジェクトファイルに、作成したアプリケーションプログラムファイルを追加します。

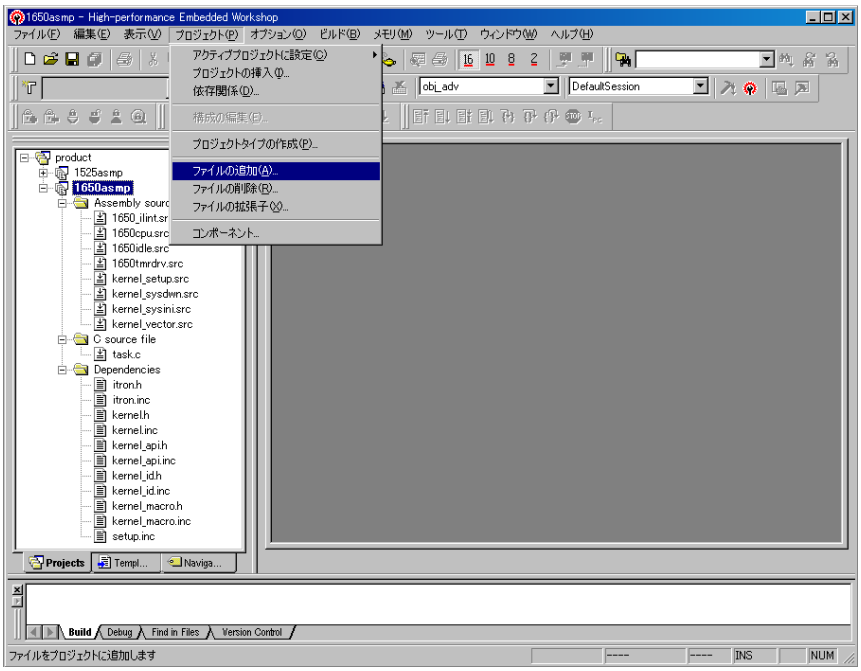


図 3-67　ファイル追加メニュー画面

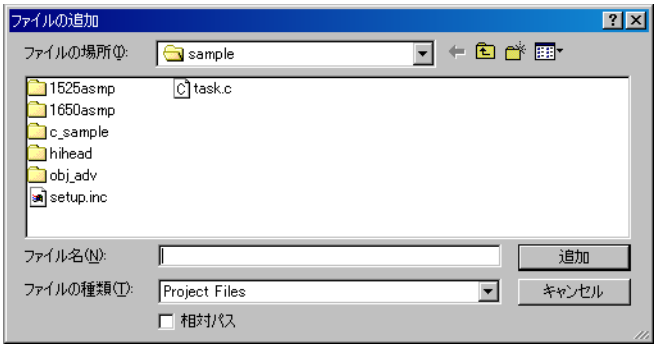


図 3-68　追加ファイル選択画面

追加ファイル選択画面で、追加するファイルを格納したフォルダに移動し、Shift キーを押しながらファイルを選択する事で、一度に複数のファイルを指定することも出来ます。

追加ファイルのセクション情報を定義します。

ヘッダメニュー［オプション(O)］→［H8S,H8/300 Standard Toolchain…］を選択し、「H8S,H8/300 Standard Toolchain」ダイアログの［最適化リンカ］タグを選択し、セクション情報の追加設定を行います。

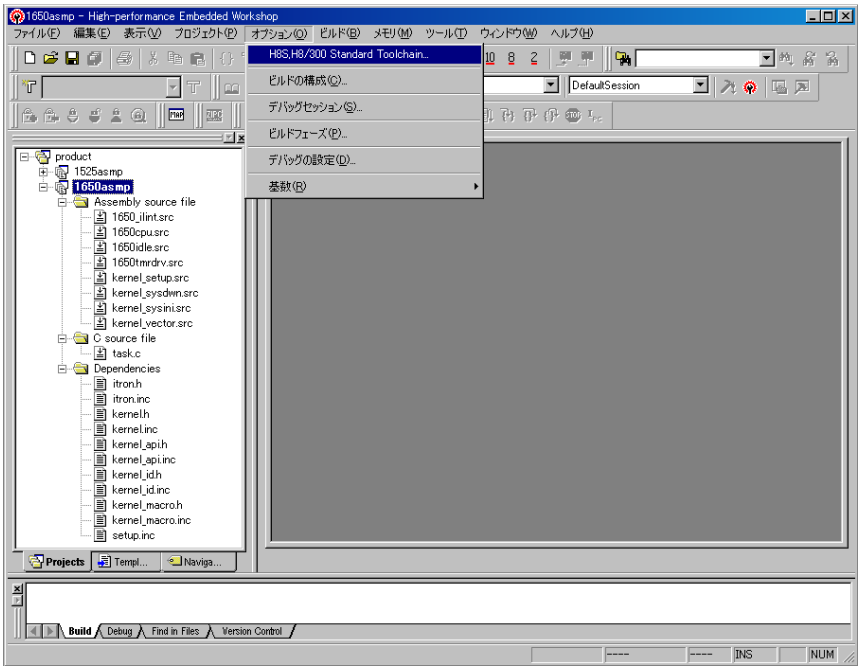


図 3-69　H8S,H8/300 Standard Toolchain 選択メニュー画面

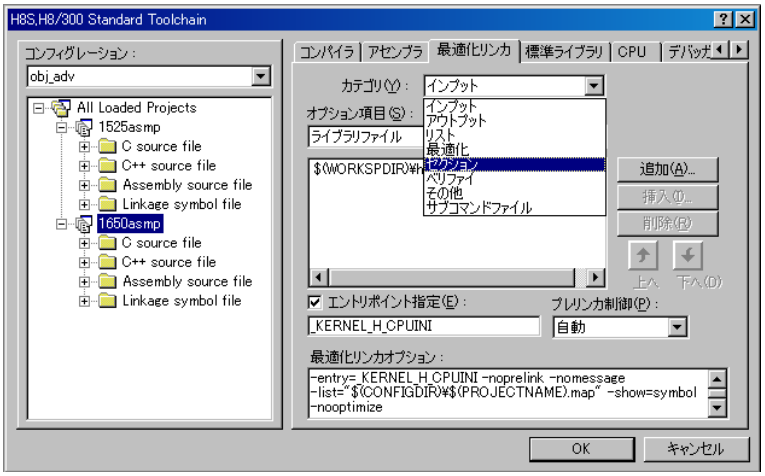


図 3-70　セクション設定メニュー画面

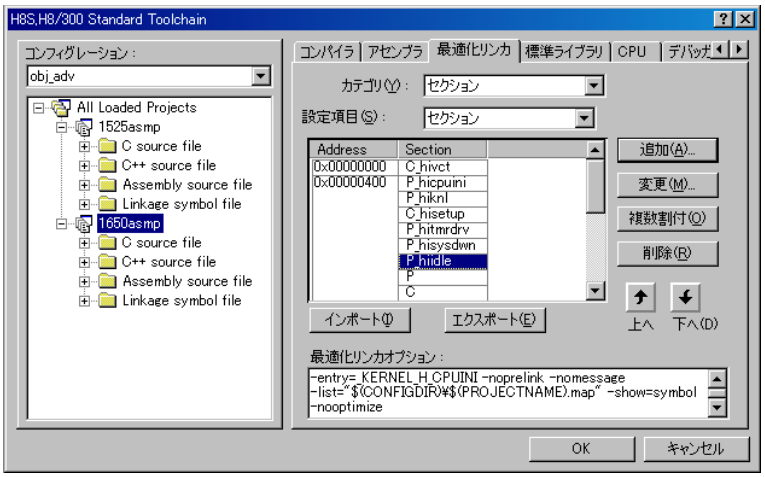


図 3-71　セクション情報追加画面

セクションの追加方法について説明します。追加したアプリケーションファイルのプログラムセクション「P_section」の追加を例に示します。

［P_hiidle］を選択し、「追加(A)」ボタンを押します。

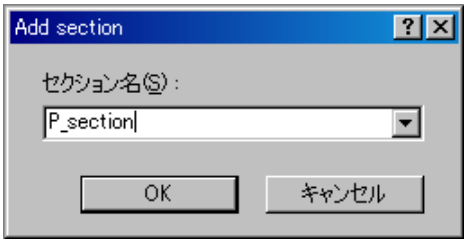


図 3-72　追加セクション情報入力画面

「Add section」ダイアログの［セクション名(S)：］に「P_section」を入力し、「OK」ボタンを押下すると、［P_hiidle］セクションの下方に追加した［P_section］セクションが表示されます。

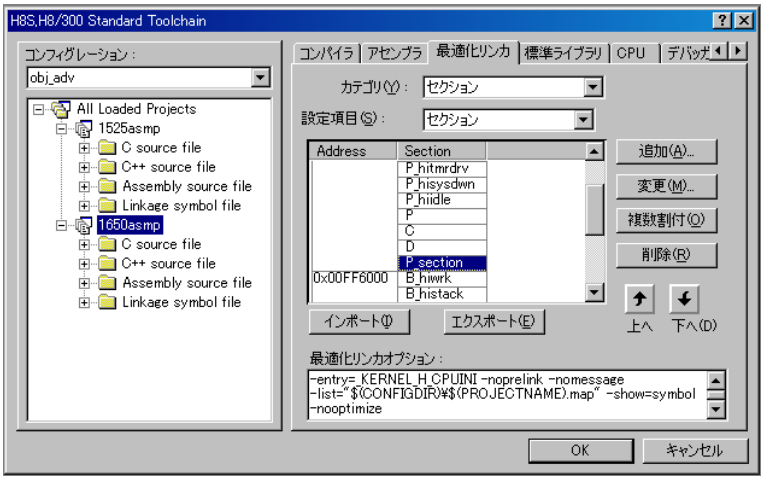


図 3-73　追加セクション情報確認画面

更新したセクション情報を反映するため、「OK」ボタンを押下します。

次に、ヘッダメニュー［ビルド(B)］→［ビルド(B)］を選択し、システムの構築（ビルド）を行います。

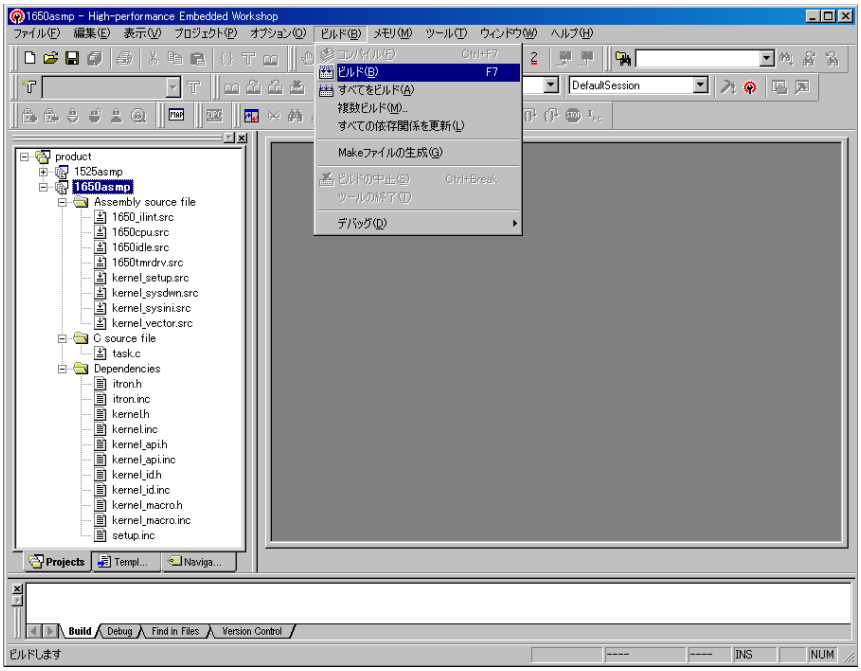


図 3-74　ビルド実行画面

以上の操作で実行形式ファイルが生成されます。
なお、コンパイル、アセンブル、リンクの結果が最下位部のウィンドウに表示されますので、エラーが発生した場合は、該当するソースプログラムを修正した後、再度、ビルドを行ってください。
実行形式ファイル（拡張子.abs）は、当該プロジェクトで指定されているフォルダ（［project］フォルダ下の「obj_adv」フォルダ）に生成されます。

標準構成によるビルドでは、パラメータチェック機能あり、共有スタック機能ありのカーネルライブラリを使用しています。
アプリケーションプログラムのデバッグが完了し、実際の製品に組み込むレベルに仕上がった場合、システムコールの先頭で行うパラメータチェック機能は無駄なルーチンになるため、HI1000/4 では、このパラメータチェック機能を取り外す事ができるようになっています。
パラメータチェック機能の取り外し方法については、**「1.3.3　HI1000/4 における組み込み方法」**を参照してください。

3.4.6　システム構築に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたシステム構築に関する質問についての回答を記述します。

《 FAQ 目次 》

(1)　サービスコールで使用するスタックサイズ

分類：構築関連

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

HI7000/4 のマニュアルに記述しているスタックサイズ算出時の必須分に、サービスコールで使用するスタックが含まれていますか。

■回答

マニュアルに記載されているスタックサイズ算出時の必須分に、サービスコールで使用するスタックサイズは含まれています。

ただし、サービスコールには、タスクスイッチが必要なものと必要ないものがあります。
タスクスイッチが必要なサービスコールを使用する場合、そのときに使用するスタックサイズは、HI7000/4 のマニュアルに記述している“スタックサイズ算出時の必須分”の中に含まれています。
もう 1 つのタスクスイッチが必要ないサービスコールでは、タスクスタックを切り替えることなく高速に処理するため、タスクスタックのまま処理します。タスク切り替えがないためにできる処理です。この場合に使用されるスタックサイズも、“スタックサイズ算出時の必須分”の中に含まれています。

(2)　OS スタックサイズの算出

分類：構築関連

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

OS などのスタックサイズの計算式について、割込みネスト数にはランタイムライブラリからさらにランタイムライブラリがコールされる場合もネスト数として数えるのですか。

■回答

割込みネスト数は、関数の呼び出しネストのことではありません。
当該割込みで、さらに割込みのネストが生じる場合、そのネスト数がいくつなのかを算出してください。

(3)　分割リンクの定義

分類：構築関連

HI7000/4
HI7700/4
HI7750/4

■質問

分割リンクを使用してシステム構築を検討しています。
アプリケーション側のプログラムは特に意識しない限りロードモジュールの更新によってアドレスなどが変化しますが、問題なく動作しますか。
また、タスク生成について考慮する点などありますか。

■回答

ロードモジュールの更新によってアドレスなど変化しますが、アプリケーションは問題なく動作します。
タスク生成について考慮する内容を以下に示します。

アプリケーション側のプログラムを ROM に入れない場合は、サービスコールを使って動的にオブジェクトを生成する必要があります。したがって、コンフィギュレータにて動的生成で使用するサービスコールを組み込むように選択してください。
コンフィギュレータにて動的生成で使用するサービスコールを組み込むようにしない場合、分割リンクでは以下の点について留意してください。

(1)　カーネル側にリンク
(2)　コンフィギュレータで「カーネルライブラリとリンク」をチェックした状態でハンドラなどを定義することはできません。

分割リンクにおけるタスク生成について考慮すべき点は、以下のとおりです。

- スタティックスタックを使用するタスクをコンフィギュレータで生成する場合には、常に「カーネルライブラリとリンク」をチェックした状態にします。
- コンフィギュレータでタスク ID を「自動割り当て」する場合は、常に「カーネルライブラリとリンク」をチェックしない状態にします。

分割リンク時では、それぞれのロードモジュール生成時に、他方のシンボルを 1 つずつ参照する必要があります。

【次ページに続く】

【前ページからの続き】

■回答

カーネル環境ロードモジュール生成時には、__kernel_cnfgtbl（サービスコールインタフェース情報 : C_hibase セクションの先頭）のアドレスを定義する必要があります。この定義アドレスと、ここでの C_hibase セクションの配置アドレスは同じでなければなりません。

カーネルロードモジュール生成時には、__kernel_sysmt（カーネル環境情報 : C_hisysmt セクションの先頭）のアドレスを定義する必要があります。この定義アドレスと、ここでの C_hisysmt セクションの配置アドレスは同じでなければなりません。

したがって、C_hibase セクションの先頭アドレスと C_hisysmt セクションの先頭アドレスを同じアドレスにするわけではありません。表にすると、以下のようになります。

カーネル側	カーネル環境側
C_hibase セクション 【実体】 サービスコールインタフェース情報を配置	シンボル「_kernel_cnfgtbl」を C_hibase セクション先頭アドレスに強制定義 【参照】 シンボル「_kernel_cnfgtbl」のアドレスをもとにサービスコールを呼び出す
シンボル「_kernel_sysmt」を C_hisysmt セクション先頭アドレスに強制定義 【参照】 シンボル「_kernel_sysmt」のアドレスからカーネル情報を参照	C_hisysmt セクション 【実体】 カーネル環境情報を配置

(4)　割込みネスト数の算出

分類：構築関連

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

以下、割込みネスト数の場合はどうなるのでしょうか（割込みレベルの設定値が連続していない）。

《割込み要因レベル》
- 割込み_IRQ0：割込みレベル15
- 割込み_IRQ1：割込みレベル14
- 割込み_IRQ2：割込みレベル 12
- 割込み_IRQ3：割込みレベル 10
- DMAC DEIO：割込みレベル 10
- CMT：割込みレベル 08

- カーネル割込みマスクレベル12

このときの以下の値は何ですか。単純にネスト数をカウントすればいいのですか。それとも、割込みレベルの最上位とマスクレベルの差、最下位とマスクレベルの差から出すのですか。

1. カーネル割込みマスクレベルより高い割込み
2. カーネル割込みマスクレベル以下の割込み

■回答

単純にネスト数をカウントしてかまいません。
割込みレベルの設定値が連続しているかどうかは関係ありません。
上記の例では、

- カーネル割込みマスクレベルより高い割込み：2
- カーネル割込みマスクレベル以下の割込み：3

となります。

(5)　セクション情報

分類：構築関連

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

関数をつくって構築した場合、P、C、D、B のセクションは現れません。
OS 側であらかじめ定義されている各セクションとは別に、P、C、D、B、これらセクションを用意する必要はありませんか。

■回答

ユーザ作成のプログラムに関するセクションの配置は、ユーザ任意で配置できます。

ユーザプログラムは、OS セクション名に組み込む必要はありません。また、OS 用セクション名に組み込むような機能もありません。

ユーザプログラムは、独自のセクション名で配置することができます。

4. デバイス依存

4.1 デバイス依存に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたデバイス依存に関する質問についての回答を記述します。

《 FAQ目次 》

4.1.1 キャッシュ有効化の設定

分類：デバイス依存

HI7700/4
HI7750/4

■質問

キャッシュを使用する場合、どのように設定すればよいのですか。

■回答

キャッシュの有効化（初期化）は、CPU 初期化処理で行います。
キャッシュの初期化用サービスコール（vini_cac サービスコール）が準備されていますので、CPU 初期化処理に追加してください。

HI7700/4 の SH7708 用 CPU 初期化ルーチンを例に以下にコーディング例を示します。

```
/************************************************************************/
/*  NAME      = hi_cpuini                                               */
/*  FUNCTION  = CPU initialize routine                                  */
/************************************************************************/
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment ***/
    set_gbr((VP)IOBASE);        /* set I/O base address to GBR      */
    vini_cac(9, 128, 4);        /* CACHE disable                    */ ①

/*** Initialize Software Environment ***/

/*  _INITSCT();    */          /* Call section-initialize routine  */

    vsta_knl();                 /* Start kernel                     */
}
```

図 4-1　キャッシュ使用時の CPU 初期化ルーチン(SH7708)

図中①にあるvini_cacサービスコールのCPU別指定例を次に示します。

【次ページに続く】

【前ページからの続き】

■回答

表 4-1　CPU 別 vini_cac 指定例

SH7708 シリーズ	vini_cac(9, 128, 4);	内蔵 RAM モード未使用、P0、U0、P3 領域はコピーバックモード、エントリ数 128、ウェイ数 4
	vini_cac(0x2E, 128, 2);	内蔵 RAM モード使用、P0、U0、P3 領域はライトスルーモード、エントリ数 128、ウェイ数 2
SH7709	vini_cac(0xF, 128, 4);	内蔵 RAM モード未使用、P0、U0、P3 領域はライトスルーモード、エントリ数 128、ウェイ数 4
SH7706, SH7709S, SH7727, SH7641, SH7660	vini_cac(0xF, 256, 4);	P0、U0、P3 領域はライトスルーモード、P1 領域はコピーバックモード、エントリ数 256、ウェイ数 4
SH7290, SH7294, SH7300, SH7705, SH7710	vini_cac(0xF, 512, 4);	P0、U0、P3 領域はライトスルーモード、P1 領域はコピーバックモード、エントリ数 512、ウェイ数 4
	vini_cac(0xF, 256, 4);	P0、U0、P3 領域はライトスルーモード、P1 領域はコピーバックモード、エントリ数 256、ウェイ数 4

キャッシュ使用時の注意事項については、「HI7000/4 シリーズ　ユーザーズマニュアル」をご参照ください。

4.1.2 キャッシュの使い方

分類：デバイス依存

■質問

HI7700/4
HI7750/4

キャッシュ使用時における注意点を教えてください。

■回答

HI7700/4
HI7750/4

1. キャッシュする／しない領域の区別

キャッシュする／しないを区別したい場合は、リンク時に以下のように配置してください。

- キャッシュしたいプログラム・データ・・・・・P0 または P1 または P3 領域
- キャッシュしたくないプログラム・データ・・・P2 領域

なお、P2 領域はキャッシュ ON 時もキャッシュされません。

動的にキャッシュを ON/OFF する場合のコーディング例として、項番 2 に HI7700/4 の場合を、項番 3 に HI7750/4 の場合を示します。

【次ページに続く】

【前ページからの続き】

■回答

HI7700/4

2. HI7700/4 の場合

(1) ON→OFFするとき

```
/* 途中で割込まれないように、SR.BL=1にするとより安全です。              */

  old_sr = get_cr();
  set_cr(old_sr|0x10000000);    /* BL=1にします。                      */
  vini_cac(0, entnum, waynum);  /* キャッシュOFF、CFビットは0          */
  vfls_cac(0, 0x1bfffff);       /* 必要な領域を実メモリにコピーバック */
  /* この時点でキャッシュ全エントリは破棄されても大丈夫です。          */

  vini_cac(8, entnum, waynum);  /* キャッシュOFF、CFビットは1          */
  /* これによりキャッシュの全エントリは無効化されます。                */

  set_cr(old_sr);
```

図 4-2 キャッシュ ON→OFF 時のコーディング例(HI7700/4)

(2) OFF→ONするとき

```
vini_cac(9, entnum, waynum);  /* CE=1、CF=1     */
/* CF=0でもよいが、CF=1にしたほうがより安全です */
```

図 4-3 キャッシュ OFF→ON 時のコーディング例(HI7700/4)

◇vfls_cac の注意事項

vfls_cac で指定するアドレスは、物理アドレスの H'0～H'1BFFFFFF の範囲を指定してください（つまり、アドレスの上位 3 ビットを 0 にしてください）。

詳細は、**「HI7000/4 シリーズ ユーザーズマニュアル」**を参照してください。

◇vini_cac の注意事項

vini_cac のパラメータ entnum、 waynum は、以下のように指定してください。

（1）SH7709S、 SH7729など16kバイトキャッシュの場合

– entnum=256、 waynum=4

（2）SH7705、 SH7290などで、32kバイトモードを使用する場合

– CCR3 レジスタを 32k バイトモードに設定した後に、entnum=512、 waynum=4 として vini_cac サービスコールを発行してください。

【次ページに続く】

【前ページからの続き】

■回答

HI7750/4

3. HI7750/4 の場合

(1) ON→OFFするとき

```
/* 途中で割込まれないように、SR.BL=1にするとより安全です。              */

  old_sr = get_cr();
  set_cr(old_sr|0x10000000);        /* BL=1にします。                  */
  vini_cac(0x00000000);             /* ICE=OFF、OCE=OFF                */
  vfls_cac(0x80000000, 0x9bffffff); /* 必要な領域を実メモリにコピーバック */
  /* この時点でキャッシュ全エントリは破棄されても大丈夫です。          */

  vini_cac(0x00000808);             /* ICE=OFF、OCE=OFF、ICI=1、OCI=1 */
  /* これによりキャッシュの全エントリは無効化されます。                */

  set_cr(old_sr);
```

図 4-4 キャッシュ ON→OFF 時のコーディング例(HI7750/4)

(2) OFF→ONするとき

```
vini_cac(0x0000090d);       /* ICI=1、ICE=1、OCI=1、CB=1、OCE=1 */
/* ICI=0、OCI=0でもよいが、ICI=1、OCI=1にしたほうがより安全です */
```

図 4-5 キャッシュ OFF→ON 時のコーディング例(HI7750/4)

◇vfls_cac の注意事項

vfls_cac で指定するアドレスは、論理アドレスの H'80000000～H'9BFFFFFF の範囲を指定してください。

詳細は、「HI7000/4 シリーズ ユーザーズマニュアル」を参照してください。

4.1.3 コピーバックモード使用時の制約（１）

分類：デバイス依存

HI7700/4
HI7750/4

■質問

キャッシュをコピーバックモードに設定して使用する場合、何か注意点がありますか。HI7000/4 使用時、キャッシュ設定に関する制限があれば教えてください。

■回答

コヒーレンシに注意して使用してください。

たとえば、プログラムでライトしたデータを DMA 転送する場合には、以下のいずれかのようにしてください。

1) プログラムでライトするデータのアドレスをキャッシュスルー領域にする（キャッシュを通さずにライトする）。
2) キャッシュ内容をメモリにコピーバックさせる関数を作成し、データライト後にその関数を呼び出し、その後DMA転送を行う。

また、DMA 転送されたデータをプログラムで読み出すときには、以下のいずれかのようにしてください。

1) キャッシュスルー領域のアドレスを通してリードする（キャッシュを通さずにリードする）。
2) キャッシュ内容を無効化する関数を作成し、その関数を呼び出してからDMA転送されたデータをキャッシュ領域のアドレスから読み出す。

次に、コピーバックの概要を示します。

【次ページに続く】

【前ページからの続き】

■回答

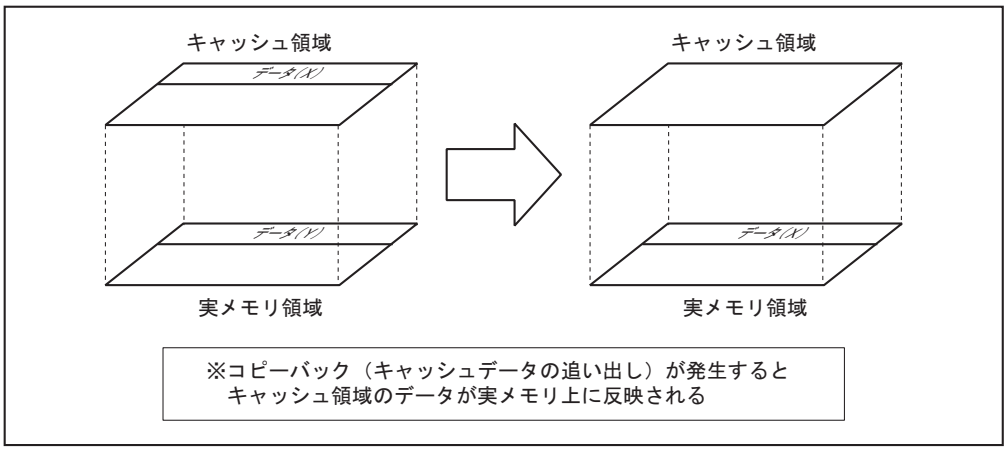


図 4-6　コピーバックモードの概要

4.1.4 コピーバックモード使用時の制約（２）

分類：デバイス依存

HI7750/4

■質問

獲得した可変長メモリブロックの領域に対してキャッシュの無効化（vinv_cac サービスコール）を行い、その後 DMA を用いてデータ転送を行ったところ、可変長メモリブロック先頭のデータが不正になりました。原因として何が考えられますか。

■回答

メモリブロックがキャッシングされることにより発生します。キャッシュラインサイズが 32 バイトである SH-4 で発生し、キャッシュラインサイズが 16 バイトである SH-3 では発生しません。

可変長メモリブロックを獲得すると、16 バイトの管理領域もメモリプールから獲得されます。その時の構成は、次のようになります。

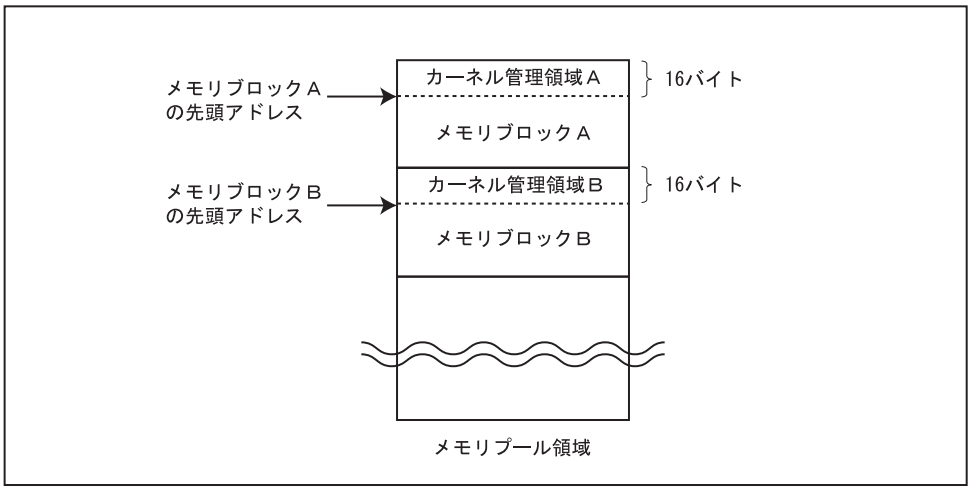


図 4-7　可変長メモリブロックの構成

【次ページに続く】

【前ページからの続き】

■回答

キャッシュラインサイズが32バイトで、獲得したメモリブロックAのアドレスが32n+16（nは整数）である場合、カーネルが管理領域Aをアクセスすると、メモリブロックAの先頭16バイトまでキャッシングされます。以下にメモリブロックがキャッシングされる例を示します。

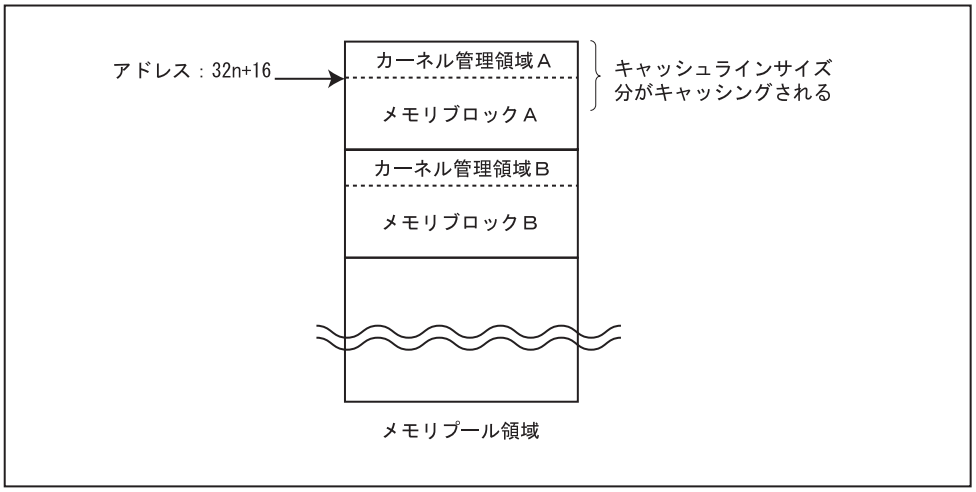


図 4-8　可変長メモリブロックのキャッシング例

(1) DMA転送前にカーネルが管理領域Aをアクセスすると、キャッシュされるデータはDMA転送前のデータとなります。
(2) DMA転送後、キャッシュの追い出しが起きるとメモリブロックAの先頭16バイトがキャッシュのデータにより上書き、破壊されてしまいます。

これを防止するには、獲得するメモリブロックの先頭アドレスを常に32nにする必要があります。これは、以下のようにして実現します。

- 獲得する可変長メモリブロックサイズ＝実獲得サイズ＋28とします。
- 使用するメモリブロック領域は、カーネルから返された先頭アドレスに対して32nに切り上げたアドレス（下方アドレスへの補正）を先頭アドレスとします。

4.1.5 キャッシュサポート機能

分類：デバイス依存

HI7700/4
HI7750/4

■質問

キャッシュサポートサービスコールでは CCR レジスタを操作してますが、これらのサービスコールを使用する場合、配置するメモリ空間について留意点がありますか。

■回答

キャッシュサポートサービスコールでは、CCR レジスタやアドレス割り付けされたキャッシュアレイをアクセスします。このとき、カーネルは内部で PC（プログラムカウンタ）を P2 領域（＝キャッシングされない）に補正して実行します。

4.1.6 X/Y メモリの使用方法

分類：デバイス依存

HI7700/4

■質問

SH7729R の X/Y メモリを使用する上で注意する点を教えてください。

■回答

プログラムからのアクセスアドレス（リンク時に指定するセクションのアドレス）は、

(1) P2/Uxyである場合
- X-RAM：H'A5007000ーH'A5008FFF
- Y-RAM：H'A5017000ーH'A5018FFF

としてください。

- X-RAM：H'05007000ーH'05008FFF
- Y-RAM：H'05017000ーH'05018FFF

にした場合、キャッシュ Enable 時に「2 サイクルアクセスを保証しなければならない。」という X/Yーram 使用時の制限事項があります。

4.1.7 MMU ユニットのサポート

分類：デバイス依存

HI7700/4
HI7750/4

■質問

MMU ユニットを使用するにあたり、使用上の制限がありますか。

■回答

基本的に MMU を Enable にすることを想定しれていませんが、以下の制約を満たせば MMU を使用することができます。

(1) カーネルセクションを、アドレス変換されない領域（P1、P2）に配置
カーネルの処理では、SR.BL=1の状態でアクセスされる部分があります。SR.BL=1のときにTLBミスが発生すると、CPUはリセットベクタに遷移してしまいます。そのような領域はアドレス変換されない領域（P1、P2）に配置する必要があります。具体的に、該当するセクションは以下のとおりです。

P_hiknl、P_hireset、 C_hivct、C_hitrp、C_hibase、C_hisysmt、C_hicfg、 B_hitrcbuf、B_hitrceml、B_hiwrk、B_hidystk、B_histstk、 B_hiirqstk、P_hisysdwn、P_hiintdwn

(2) サービスコールのポインタ渡しのパラメータアドレス（pk_xxxなど）
カーネルは、サービスコールで指定されたパラメータアドレスをSR.BLがサービスコール呼び出し時と同じ状態（0）でアクセスします。ここでTLBミスが発生する可能性がある場合は、TLBミスハンドラではサービスコールを発行してはなりません。ここでTLBミスが発生する可能性がないようにした場合は、TLBミスハンドラでサービスコールを発行できます。ただし、SR.BL=1の状態でサービスコールを発行できないのは、HI7700/4の仕様です。

【次ページに続く】

【前ページからの続き】

■回答

(3) 特権／ユーザモード
HI7700/4では、タスクやハンドラなどすべてのプログラムは特権モードの状態で実行する仕様です。アプリケーション側でユーザモードに遷移してはなりません。

(4) TLBミスハンドラnnnn_expent.src内のシンボル"__kernel_tlb_ent"の箇所にプログラムを書き込んでください。

4.1.8 タイマドライバ

分類：デバイス依存

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

ボードに乗っている水晶が 33.333MHz の場合で、ハードウェアタイマ周期時間 1ms を作るためには、33.333 で計算すればよいのですか。

■回答

33.333 ではなく、33.333×10^6 で計算してください。
タスク切り替え処理には影響はありませんが、OS の時間管理機能に影響します。

【参考】タイマドライバ周期時間の算出
以下に、SH7604 を HI7000/4 で使用する事を想定し、1ms のタイマ周期時間算出例を示します。

ハードウェアタイマ周期時間(T)は、カウンタクロック周期時間(t)とカウンタ値(n)で決定され、以下のようになります。

- $T = \{ t \times (n + 1) \}$

t はタイマコントロールレジスタ(TCR)でカウンタクロックの選択（φ／8、φ／32、φ／128）を行うことで決定します。
φ（CPU クロック）が 28.6364MHz の場合、t は以下の数値になります。

- カウンタクロック＝φ／8　　：　t＝279ns
- カウンタクロック＝φ／32　：　t＝1.11μs
- カウンタクロック＝φ／128：　t＝4.46μs

【次ページに続く】

【前ページからの続き】

■回答

n はアウトプットコンペアマッチA(OCRA)に 0x0000～0xFFFF の値を設定する事で決定します。

したがって、φ（CPU クロック）が 28.6364MHz の場合、T は以下の範囲になります。

- カウンタクロック＝φ／8　　：　T＝279ns ～ 18.2ms
- カウンタクロック＝φ／32　：　T＝1.11μs ～ 72.7ms
- カウンタクロック＝φ／128：　T＝4.46μs ～ 292ms

《1ms 周期の導き方》

アウトプットコンペアマッチA(OCRA) ＝ タイマ周期時間(s) × n － 1

上式より、タイマ周期時間(s)を 1ms とするので、タイマ周期時間(s) ＝ 1 × 10^{-3}

カウンタクロックの選択をφ／8 を選択すると、φ（CPU クロック）が 28.6364MHz の場合、

n ＝ 28.6364 × 10^{6} ÷ 8

したがって、

アウトプットコンペアマッチA(OCRA) ＝ タイマ周期時間(s) × n － 1

＝ (1 × 10^{-3}) × (28.6364 × 10^{6} ÷ 8) － 1

＝ 3578.55 (0x0DFA)

φ（CPU クロック）が 28.6364MHz を使用してタイマ周期時間(s)を 1ms とする場合、アウトプットコンペアマッチA(OCRA)に設定する値は、3578.55 (0x0DFA)となります。

4.1.9 OS が使用するタイマの調整

分類：デバイス依存

HI7000/4
HI7700/4
HI7750/4

■質問

タイマの調整方法について教えてください。

■回答

SH7751 を HI7750/4 で使用する場合を例に以下に説明します。

SH7751 フォルダ内の 7751_tmrdef.h ファイルを参照ください。
19 行目の「Peripheral clock」の値を実環境にて使用している値に変更し、再構築後、再度確認してください。
OS におけるタイマの調整は、当該ファイル以外にありません。

```
/***********************************************************************/
/*              HI7750/4 header file for timer driver                  */
/*              Copyright (c) 2000(2003) Renesas Technology Corp.      */
/*              and Renesas Solutions Corp. All Rights Reserved.       */
/*              HI7750/4(HS0775ITI41SR) V1.0                           */
/***********************************************************************/
/***********************************************************************/
/*  FILE      = 7751_tmrdef.h ;                                        */
/*  CPU type  = SH7751                                                 */
/*  Module    = TMU                                                    */
/*              INTC                                                   */
/***********************************************************************/
/***********************************************************************/
/* TMU,IPR setting data                                               */ ①
/*  Condition:                                                        */
/*    (1) Peripheral clock      : 42MHz                               */
/*    (2) Timer interrupt level : 13                                  */
/***********************************************************************/
#define PCLOCK    41666667          /* Peripheral clock (Hz)          */
```

図 4-9　7751_tmrdef.h ファイル

図中①の部分を、ご使用のマイコン動作周波数にあわせて変更してください。

4.1.10 CPU 初期化ルーチンの C 言語記述

分類：デバイス依存

■質問

CPU 初期化ルーチンをＣ言語で記述する（Ｃ言語で記述したい）場合、どのようにすればよいですか。

HI2000/3
HI1000/4

■回答

CPU 初期化ルーチンをＣ言語で記述することはできます。
しかし、Ｃ言語で記述されたプログラムは、スタック（メモリ）をアクセスします。スタック領域のアクセス準備が整っていないうちにアクセスすると、CPU 例外が発生する場合があります（CPU 例外が発生するとシステム異常終了となります）。スタックのアクセス準備が完了するまで「CPU 初期化ルーチン」はアセンブリ言語で記述する必要があります。

スタック領域のアクセス準備が整った後、Ｃ言語で記述した CPU 初期化ルーチンを実行することができます。
Ｃ言語で記述した CPU 初期化ルーチンを実行するために必要な、標準提供の CPU 初期化ルーチン（アセンブリ言語記述）の変更などについては、本アプリケーションノートの**「2 章　アプリケーションプログラムの作成手法」**をご参照ください。

4.1.11 割込み出入り口処理ルーチンの配置

分類：デバイス依存

HI7000/4
HI7700/4
HI7750/4

■質問

割込み出入り口の処理ルーチン（P_hiexpent セクション）の配置アドレスは、どこにすればよいですか（カーネルが VBR レジスタを初期化するときの初期値はどこになりますか）。

■回答

割込み出入り口の処理ルーチン（P_hiexpent セクション）は、ユーザ任意で配置できます。

カーネルにおける VBR レジスタの初期化は、P_expent 配置アドレスから H'100 番地引いたアドレスを設定します（カーネル初期化処理の中で自動的に算出しています）。

サンプルの内容は、デバイスのマニュアル『例外処理』を参照してください。その中に、例外処理ベクタアドレスに関する説明があります。

表 4-2 割込み/例外出入口処理

シンボル名	配置アドレス	処理内容
__kernel_exp_ent	P_hiexpent セクション	VBR+H'100 は、 一般例外のベクタアドレス
__kernel_tlb_ent	P_hiexpent セクション＋H'300	VBR+H'400 は、 TLB ミス例外のベクタアドレス
__kernel_int_ent	P_hiexpent セクション＋H'500	VBR+H'600 は、 割込みのベクタアドレス

一般例外が発生した場合、VBR+H'100 から処理を始めるため、各該当アドレスに必要な処理を配置しなくてはなりません。

したがって、上記のような設定になります。

4.1.12 外部メモリの初期化

分類：デバイス依存

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

タスクのスタック領域を外部空間に設定するとタスクが起床されません。原因について教えてください。

■回答

外部 RAM 空間を使用しているため、I/O ポートの設定（初期化）が必要です。

OS 起動前に I/O ポートの初期化処理を行ってください。
リセット後、カーネル初期化処理でタスクスタック領域などの初期化を行うため、外部アドレスをアクセスします。

H8S マイコンを使用した場合を例に以下に示します。
たとえば、モード 6 使用時では、ポート A、B、C はリセット直後、入力ポートです。ポート A、B は、PFCR1（端子機能コントロールレジスタ 1）、ポート C は、DDR（データディレクションレジスタ）を 1 に設定し、アドレス出力にする必要があります。

4.1.13 低消費電力モードへの移行

分類：デバイス依存

HI7000/4
HI7700/4
HI7750/4

■質問

システムタイマを動作させたままで、ソフトウェアスタンバイ状態に遷移させても問題はありませんか。また、ソフトウェアスタンバイモードに遷移させる場合、何か注意点がありますか。

■回答

ソフトウェアスタンバイ状態に移行した場合、OS のシステムタイマに使用しているタイマデバイスも停止します。そのため、以下のような誤差が生じます。

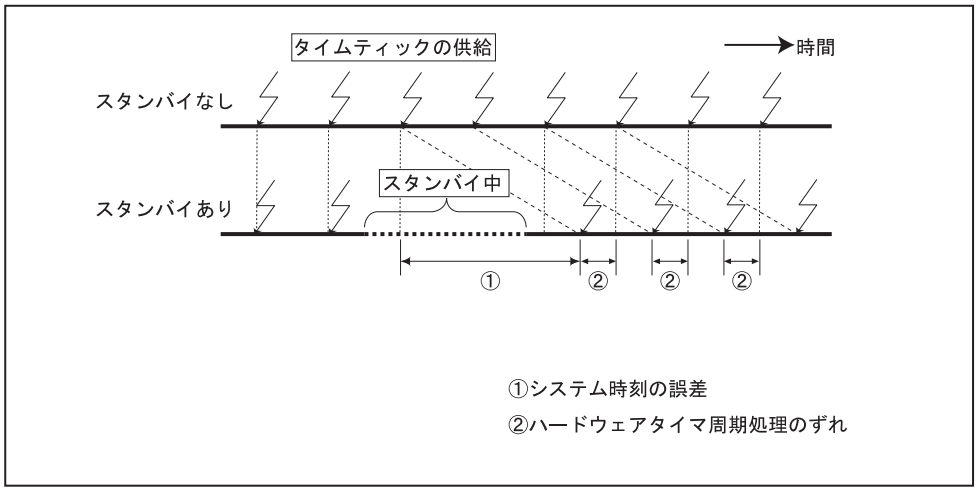


図 4-10 スタンバイによるシステム時刻の誤差

【次ページに続く】

【前ページからの続き】

■回答

また、ソフトウェアスタンバイにて、OS のシステムタイマに使用しているタイマデバイスのレジスタが初期化される場合、次の注意が必要です。

1. ソフトウェアスタンバイ中はシステム時刻を停止、復帰時にシステム時刻を継続させる

例えば、ソフトウェアスタンバイ移行時に、前回のタイマ割り込みから 0.6msec 経過しており、ソフトウェアスタンバイ解除から 0.4msec 後（タイムティックが 1msec の場合）にタイマ割込みを発生させるようにする場合、以下の処理が必要です。

- ソフトウェアスタンバイ移行時に、タイマデバイスレジスタのタイマカウンタ値を保存
- ソフトウェアスタンバイ解除時に、保存した値を復帰

2. ソフトウェアスタンバイ中はシステム時刻を停止、復帰時にタイマカウンタ値を初期化する

例えば、ソフトウェアスタンバイ移行時に、前回のタイマ割り込みから 0.6msec 経過していた場合でも、ソフトウェアスタンバイ解除から 1msec 後（タイムティックが 1msec の場合）にタイマ割込みを発生させるようにする場合、以下の処理が必要です。

- ソフトウェアスタンバイ解除時に、タイマデバイスレジスタを再度初期設定（タイマ初期化ルーチン_kernel_tmrini()をコール）する。

5. デバッグ手法

5.1 デバッグの概要

HI シリーズ OS を使用したシステムでは、コンフィギュレータにて設定した初期登録オブジェクトの誤りや、未定義割込み／未定義例外の発生など、カーネルが異常を検出すると、システムダウンルーチンが起動されます。また、システムダウンルーチンはアプリケーションプログラムから任意に起動することも可能です。

本章では、システムダウンルーチンを活用したデバッグ手法、およびシステムダウンルーチンが起動された時のエラー要因の解析方法について説明します。

システム異常が発生した場合、以下の順序で問題を解決します。

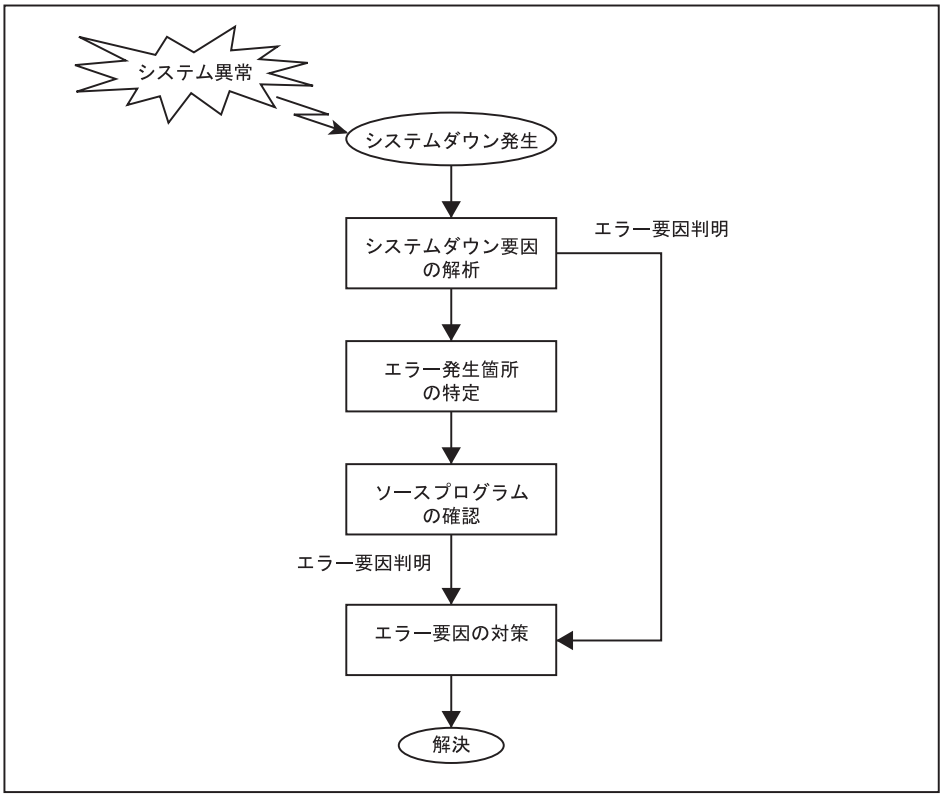


図 5-1　システム異常発生時のデバッグの流れ

【注】「システムダウンルーチン」は、HI7000/4 シリーズ、および HI1000/4 における呼称です。HI2000/3 では「システム異常終了ルーチン」と呼称が違いますが、本章では「システムダウンルーチン」と総称します。

5.2 HI7000/4 シリーズ

5.2.1 デバッグの準備

(1) パラメータチェック機能の有効化

デバッグ時はサービスコールの“パラメータチェック機能あり”で使用することを推奨します。パラメータチェック機能については、本アプリケーションノートの**「1.3 サービスコールのパラメータチェック」**をご参照ください。

(2) デバッグ用コードの追加

サービスコールにてパラメータエラーなど、処理を継続する上で致命的なエラーコードが返された場合、システムダウンルーチンを呼び出すよう、アプリケーションプログラムにシステムダウンルーチンを呼び出すコードを追加します。デバッグ用のコードは最終的には不要となる為、マクロやコンパイラのプリプロセッサを利用し、必要なときだけコードが生成されるようにすると便利です。

システムダウンルーチンの呼出し形式と、コーディング例を以下に示します。

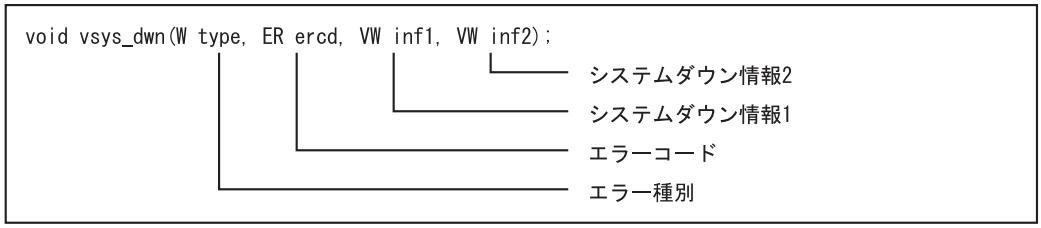


図 5-2 システムダウンルーチンの呼出し形式(HI7000/4 シリーズ)

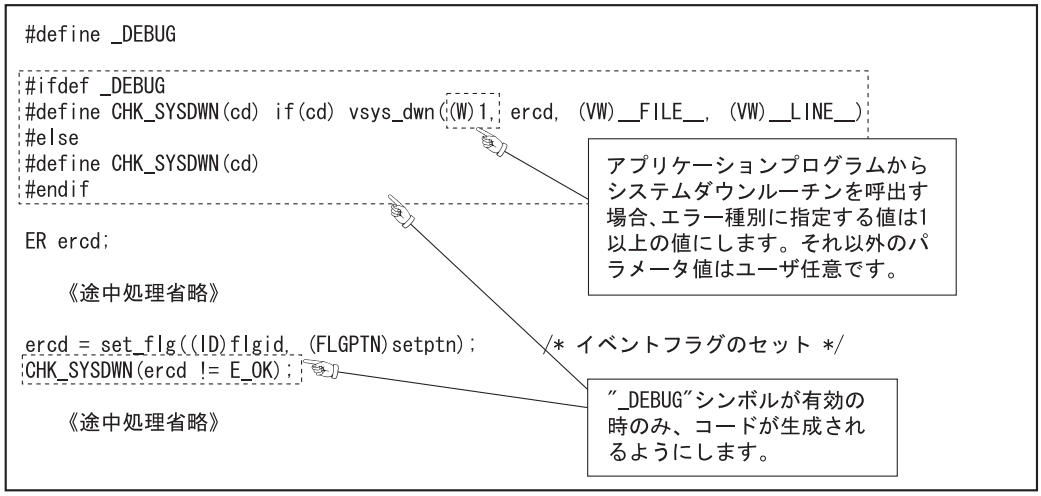


図 5-3 デバッグ用コードのコーディング例(HI7000/4 シリーズ)

(3) ブレークポイントの設定

以下に示す箇所にエミュレータや ICE を使用してブレークポイントを設定し、アプリケーションプログラムを実行します。

```
/************************************************************************/
/*  NAME      = _kernel_sysdwn ;                                        */
/*  FUNCTION  = System down routine ;                                   */
/************************************************************************/
void    _kernel_sysdwn(type, ercd, inf1, inf2)
W type; /*system down type */
                /* type >=  1 : system down of user program            */
                /* type ==  0 : initial information error              */
                /* type == -1 : context error of ext_tsk               */
                /* type == -2 : context error of exd_tsk               */
                /* type == -16: undefined interrupt/exception          */
ER ercd; /* error code  */
                /* type >=  0 : error code of user program             */
                /* type ==  0 : error code of initial information      */
                /* type == -1 : error code of ext_tsk                  */
                /* type == -2 : error code of exd_tsk                  */
                /* type == -16: interrupt vector number                */
VW inf1; /* information-1  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : indicator of initial information error */
                /* type == -1 : address of ext_tsk call                */
                /* type == -2 : address of exd_tsk call                */
                /* type == -16: address of interrupt occurrence        */
VW inf2; /* information-2  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : number of error initial information    */
                /* type == -16: SR of interrupt occurrence             */
{
    set_imask(SR_IMS15); /* mask all interrupt                         */
    while(TRUE);    /* endless loop                                    */
}
```

この行にブレークポイントを設定します。

図 5-4　ブレークポイントの設定例(HI7000/4)

```
/************************************************************************/
/*  NAME      = _kernel_sysdwn ;                                        */
/*  FUNCTION  = System down routine ;                                   */
/************************************************************************/
void    _kernel_sysdwn(type, ercd, inf1, inf2)
W type; /*system down type */
                /* type >=  1 : system down of user program            */
                /* type ==  0 : initial information error              */
                /* type == -1 : context error of ext_tsk               */
                /* type == -2 : context error of exd_tsk               */
                /* type == -16: undefined interrupt/exception          */
ER ercd; /* error code  */
                /* type >=  0 : error code of user program            */
                /* type ==  0 : error code of initial information      */
                /* type == -1 : error code of ext_tsk                  */
                /* type == -2 : error code of exd_tsk                  */
                /* type == -16: interrupt vector number                */
VW inf1; /* information-1  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : indicator of initial information error */
                /* type == -1 : address of ext_tsk call                */
                /* type == -2 : address of exd_tsk call                */
                /* type == -16: address of interrupt occurrence        */
VW inf2; /* information-2  */
                /* type >=  0 : information of user program            */
                /* type ==  0 : number of error initial information    */
                /* type == -16: SR of interrupt occurrence             */
{
    set_cr(MD_BIT | (SR_IMS15 << 4));  /* mask all interrupt           */
    while(TRUE);    /* endless loop                                    */
}
```

この行にブレークポイントを設定します。

図 5-5　ブレークポイントの設定例(HI7700/4、HI7750/4)

5.2.2 システムダウンの発生

システム異常が発生すると、「5.2.1 (3) ブレークポイントの設定」にて設定したブレークポイントにてプログラムが停止します。HI7000/4 シリーズにおいては、システム異常発生時のエラー情報がレジスタにて渡されます。

エラー情報のパラメータ形式を以下に示します。

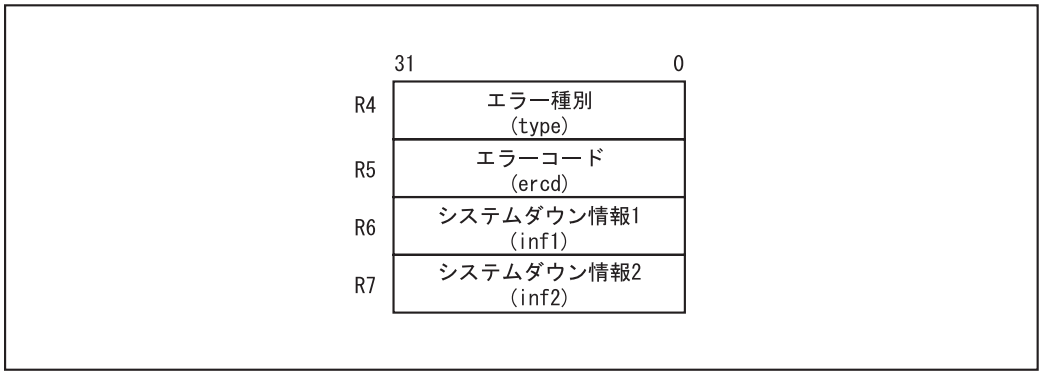


図 5-6 システムダウン情報のパラメータ形式(HI7000/4 シリーズ)

5.2.3 システムダウン要因の種類

HI7000/4 シリーズにおけるシステムダウン要因の種類を以下に示します。

表 5-1 システムダウンの種類(HI7000/4 シリーズ)

項番	エラー種別(R4)	内 容
1	0	初期登録情報エラー
2	H'FFFFFFFF(-1)	コンテキストエラー(ext_tsk サービスコール)
3	H'FFFFFFFE(-2)	コンテキストエラー(exd_tsk サービスコール)
4	H'FFFFFFF0(-16)	未定義割込み／例外の発生
5	1 以上（ユーザ任意）*1	vsys_dwn、ivsys_dwn サービスコールの呼出し

【注】 *1 アプリケーションが設定する値により異なります。詳細に関しては「(6) アプリケーションプログラムからの呼出し」を参照してください。

以降、エラー種別毎に内容を説明します。

(1) 初期登録情報エラー

コンフィギュレータにて設定した初期登録オブジェクトの情報に誤りがあります。エラー情報として次の値が渡されます。

表 5-2 エラー情報一覧(初期登録情報エラー)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R4	H'0
エラーコード(ercd)	R5	発生したエラーコード
システムダウン情報 1 (inf1)	R6	0（カーネル側）または 1（カーネル環境側）
システムダウン情報 2 (inf2)	R7	エラーとなった初期登録オブジェクトの番号

エラーコード（ercd）に発生したエラーコード（サービスコールエラーコード）が渡されます。

システムダウン情報 1 （inf1）には、カーネル側の登録処理で発生したなら 0、カーネル環境側の登録処理で発生したなら 1 が渡されます。カーネル側とカーネル環境側の違いを以下に示します。

表 5-3 カーネル側とカーネル環境側の区別

カーネル側	カーネルロードモジュールに含まれるオブジェクトで、コンフィギュレータの各オブジェクト生成ダイアログボックスにて、「カーネルライブラリとリンク」チェックボックスをオンにしたオブジェクト
カーネル環境側	カーネル環境ロードモジュールに含まれるオブジェクトで、コンフィギュレータの各オブジェクト生成ダイアログボックスにて、「カーネルライブラリとリンク」チェックボックスをオフにしたオブジェクト

システムダウン情報 2 （inf2）には、何番目の登録処理でエラーとなったかを示す番号が渡されます。ただし、カーネル側が先に処理され、次にカーネル環境側が処理されます。

以下にシステムダウン情報 1 および 2 に渡される番号の例を示します。

カーネル側初期登録内容	カーネル環境側初期登録内容
・タスクA ・周期ハンドラA ・拡張サービスコールA	・タスクB ・タスクC ・セマフォA ・イベントフラグA

①タスクAの初期登録でエラーとなった場合 ..... inf1＝0、inf2＝1
②周期ハンドラAの初期登録でエラーとなった場合 ..... inf1＝0、inf2＝2
③拡張サービスコールAの初期登録でエラーとなった場合 ... inf1＝0、inf2＝3
④タスクBの初期登録でエラーとなった場合 ..... inf1＝1、inf2＝1
⑤タスクCの初期登録でエラーとなった場合 ..... inf1＝1、inf2＝2
⑥セマフォAの初期登録でエラーとなった場合 ..... inf1＝1、inf2＝3
⑦イベントフラグAの初期登録でエラーとなった場合 ..... inf1＝1、inf2＝4

図 5-7 システムダウン情報 1 および 2 の設定例

エラーが発生した番号をもとに、コンフィギュレータにて登録内容を確認してください。

オブジェクト毎の処理順序に関しては、**「HI7000/4 シリーズ ユーザーズマニュアル」**をご参照ください。

(2) コンテキストエラー（ext_tsk サービスコール）

非タスクコンテキストから ext_tsk サービスコールが発行されました。エラー情報として次の値が渡されます。

表 5-4 エラー情報一覧(コンテキストエラー)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R4	H'FFFFFFFF(-1)
エラーコード(ercd)	R5	H'FFFFFFE7(-25)
システムダウン情報 1 (inf1)	R6	ext_tsk を呼び出したアドレス
システムダウン情報 2 (inf2)	R7	不定

システムダウン情報 1 に渡されるアドレスに該当する部分のアプリケーションプログラムを確認し、タスクコンテキストから ext_tsk サービスコールを発行するように修正してください。

アドレスから該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

(3) コンテキストエラー（exd_tsk サービスコール）

非タスクコンテキストから exd_tsk サービスコールが発行されました。エラー情報として次の値が渡されます。

表 5-5 エラー情報一覧(コンテキストエラー)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R4	H'FFFFFFFE(-2)
エラーコード(ercd)	R5	H'FFFFFFE7(-25)
システムダウン情報 1 (inf1)	R6	exd_tsk を呼び出したアドレス
システムダウン情報 2 (inf2)	R7	不定

システムダウン情報 1 に渡されるアドレスに該当する部分のアプリケーションプログラムを確認し、タスクコンテキストから exd_tsk サービスコールを発行するように修正してください。

アドレスから該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

(4) 未定義割込み／例外の発生

未定義割込み、または未定義一般例外が発生しました。エラー情報として次の値が渡されます。

表 5-6 エラー情報一覧(未定義割込み／例外の発生)

項 目	格納対象のレジスタ	内 容	
		HI7000/4	HI7700/4, HI7750/4
エラー種別(type)	R4	H'FFFFFFF0(-16)	
エラーコード(ercd)	R5	ベクタ番号	例外コード
システムダウン情報 1 (inf1)	R6	割込みまたは例外発生時の PC レジスタ情報*1*2*3	
システムダウン情報 2 (inf2)	R7	割込みまたは例外発生時の SR レジスタ情報*3	

【注】 *1 スロット不当命令例外の場合、PC レジスタ情報の値は、遅延スロットに配置された未定義コード、または遅延分岐命令のアドレス（HI7000/4 でのみ次命令の先頭アドレス）となります。

*2 トラップ命令例外の場合、TRAPA 命令の次の命令の先頭アドレスとなります。

*3 HI7000/4 の CPU アドレスエラー／DMAC アドレスエラーで、スタックポインタ（SP）が 4 の倍数以外の場合、PC レジスタ情報、SR レジスタ情報は不定となります。

エラーコード（ercd）には、HI7000/4 の場合は発生したベクタ番号が、HI7700/4、HI7750/4 の場合は発生した例外コードが渡されます。エラーコード（ercd）から発生した事象を特定してください。ベクタ番号、および例外コードの詳細に関しては、使用している CPU のハードウェアマニュアルをご参照ください。

(a) 発生した事象が未定義割込みの場合

必要とする割込みであれば、割込みハンドラを作成、登録してください。意図しない割込みであれば原因を調査し、割込みが発生しないようにしてください。

意図しない割込みが発生する主な要因として、以下のものが考えられます。

- 割込み要求元となる外部デバイス、または CPU 内蔵周辺モジュールのレジスタ設定ミス
- 割込みコントローラの IRQ／IRL モードの設定ミス
- 割込みコントローラの割込み優先レベル設定ミスによる、誤ったレベル割込みの認識
- ノイズによる誤った割込み要求信号の検出
- ハードウェア回路の不備など

(b) 発生した事象が未定義一般例外の場合

必要とする例外であれば、CPU 例外またはトラップ例外ハンドラを作成、登録してください。意図しない例外であれば、システムダウン情報 1 （inf1）にて渡される PC から異常が発生した箇所を特定し、原因を調査する必要があります。

システムダウン情報 2（inf2）にて渡される SR からは、例外発生時の CPU の動作モードや、割込みマスクレベル等の情報を得ることができます。

システムダウン情報 1 （inf1）にて渡される PC から該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

未定義例外要因の調査に関しては、本アプリケーションノートの**「5.6 CPU 例外発生の具体例と解決策」**をご参照ください。

(5) vsys_dwn、ivsys_dwn サービスコールの呼出し

このエラーは、アプリケーションプログラムから vsys_dwn、ivsys_dwn サービスコールを呼び出した場合に発生します。渡されるエラー情報は、vsys_dwn、ivsys_dwn サービスコールを呼び出した際のパラメータ値となります。

「5.2.1 (1) **デバッグ用コードの追加**」に示すデバッグ用コードにおいては、エラー情報として次の値が渡されます。

表 5-7 エラー情報一覧(vsys_dwn、ivsys_dwn サービスコールの呼出し)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R4	1
エラーコード(ercd)	R5	発生したサービスコールのエラーコード
システムダウン情報 1 (inf1)	R6	発生したソースプログラムファイルパスへのアドレス
システムダウン情報 2 (inf2)	R7	発生したソースプログラムの行番号

エラー情報をもとに原因を調査し、該当するアプリケーションプログラムを修正してください。

サービスコールのエラーコードに関しては、「HI7000/4 シリーズ ユーザーズマニュアル」をご参照ください。

5.3 HI2000/3

5.3.1 デバッグの準備

(1) パラメータチェック機能の有効化

デバッグ時はサービスコールのパラメータチェック機能ありで使用することを推奨します。パラメータチェック機能については、本アプリケーションノートの**「1.3 サービスコールのパラメータチェック」**をご参照ください。

(2) デバッグ用コードの追加

サービスコールにてパラメータエラーなど、処理を継続する上で致命的なエラーコードが返された場合、システムダウンルーチンを呼び出すよう、アプリケーションプログラムにシステムダウンルーチンを呼び出すコードを追加します。デバッグ用のコードは最終的には不要となる為、マクロやコンパイラのプリプロセッサを利用し、必要なときだけコードが生成されるようにすると便利です。

システムダウンルーチンの呼出し形式と、コーディング例を以下に示します。

```
void HIPRG_ABNOML(void);
```

図 5-8 システムダウンルーチンの呼出し形式例(HI2000/3)

```
extern void HIPRG_ABNOML(void);

#define _DEBUG

#ifdef _DEBUG
#define CHK_SYSDWN(cd) if(cd) HIPRG_ABNOML()
#else
#define CHK_SYSDWN(cd)
#endif

ER ercd;

    《途中処理省略》

ercd = set_flg((ID)flgid, (UINT)setptn);        /* イベントフラグのセット */
CHK_SYSDWN(ercd != E_OK);

    《途中処理省略》
```

"_DEBUG"シンボルが有効の時のみ、コードが生成されるようにします。

図 5-9 デバッグ用コードのコーディング例(HI2000/3)

(3) ブレークポイントの設定

以下に示す箇所にエミュレータや ICE を使用してブレークポイントを設定し、アプリケーションプログラムを実行します。

```
;************************************************************************
;*specifications ;                                                      *
;*name      = _HIPRG_ABNOML : abnormal quit handler             ;       *
;*function  =                                                   ;       *
;*notes     =                                                   ;       *
;*date      = 99/02/22                                          ;       *
;*author    = Hitachi, Ltd.                                     ;       *
;*attribute = public                                            ;       *
;*class     = system                                            ;       *
;*linkage   =                                                   ;       *
;*input     =                                                   ;       *
;*output    =                                                   ;       *
;*end of specifications ;                                               *
;************************************************************************
_HIPRG_ABNOML:
        orc     #HIDEF_IMASK_CCR:8,ccr  ;:interrupt mask for CCR register
        orc     #HIDEF_IMASK_EXR:8,exr  ;:interrupt mask for EXR register
        bra     $                       ;:forever loop
;
```

この行にブレークポイントを設定します。

図 5-10　ブレークポイントの設定例(HI2000/3)

5.3.2 システムダウンの発生

システム異常が発生すると「5.3.1 (3) ブレークポイントの設定」にて設定したブレークポイントにてプログラムが停止します。HI2000/3 においては、システム異常発生時のエラー情報がスタックにて渡されます。

エラー情報のパラメータ形式を以下に示します。

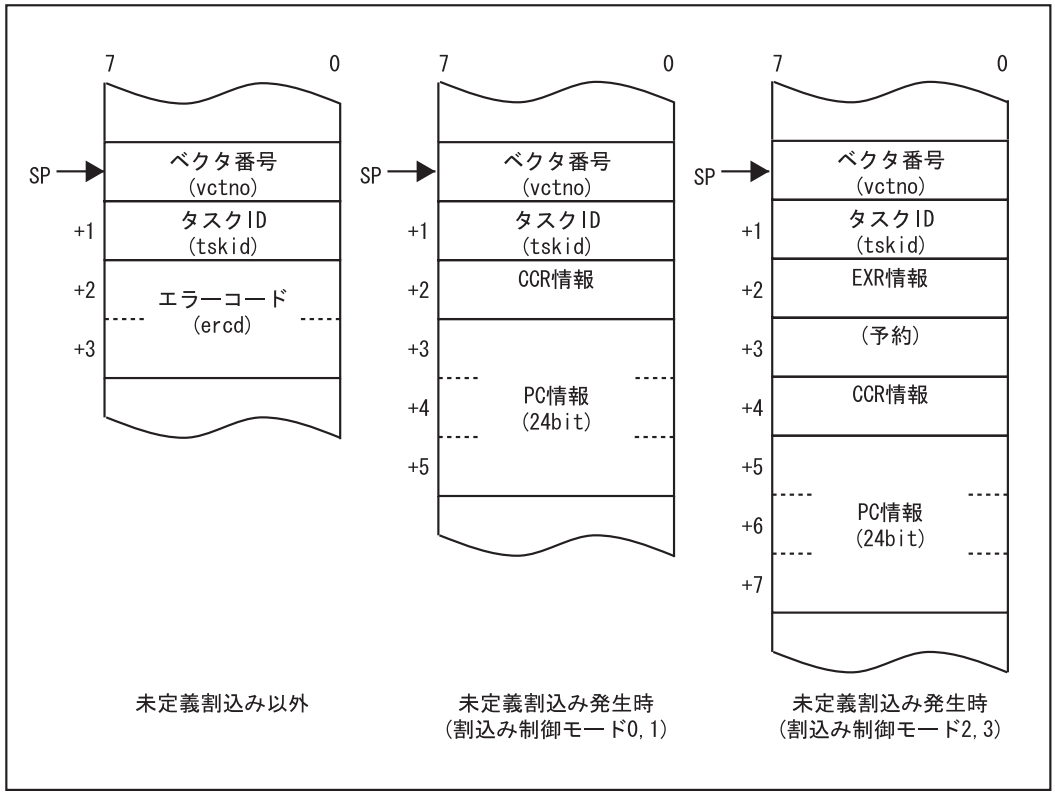


図 5-11 システムダウン情報のパラメータ形式(HI2000/3)

5.3.3 システムダウン要因の種類

HI2000/3 におけるシステムダウン要因の種類を以下に示します。

表 5-8 システムダウンの種類(HI2000/3)

項番	エラー種別		内 容
	ベクタ番号(SP+0)	エラーコード(SP+2、SP+3)	
1	0	H'0～H'0FFF	セットアップ情報エラー
2		H'F9ED	タイマ未サポートエラー
3		H'FFEB	コンテキストエラー(ext_tsk サービスコール)
4		H'FFBB	コンテキストエラー(ret_int サービスコール)
5	0 以上	－	未定義割込みの発生
6	－*1	－*1	アプリケーションプログラムからの呼出し

【注】 *1 アプリケーションが設定する値により異なります。詳細は「(6) アプリケーションプログラムからの呼出し」を参照してください。

以降、エラー種別毎に内容を説明します。

(1) セットアップ情報エラー

セットアップテーブルに誤りがあります。エラー情報として次の値が渡されます。

表 5-9 エラー情報一覧(セットアップ情報エラー)

項 目	格納対象のスタック	内 容
ベクタ番号(vecno)	SP+0	0
タスク ID(tskid)	SP+1	0
エラーコード(ercd)	SP+2 SP+3	セットアップ情報エラーコード（H'0～H'0FFF）

エラーコード（ercd）には、セットアップテーブルの不正内容に対応するコード（H'0000～H'0FFF）が渡されます。エラーコードに該当するセットアップテーブルの設定値を確認してください。エラーコードの詳細に関しては、「HI2000/3 ユーザーズマニュアル」をご参照ください。

(2) タイマ未サポートエラー

セットアップテーブルにてタイムアウト機能を未使用に設定した場合で、タイムアウト機能を使用しようとしました。エラー情報として次の値が渡されます。

表 5-10 エラー情報一覧(タイマ未サポート)

項 目	格納対象のスタック	内 容
ベクタ番号(vecno)	SP+0	0
タスク ID(tskid)	SP+1	0
エラーコード(ercd)	SP+2 SP+3	H'F9ED

セットアップテーブルにてタイムアウト機能を「使用する」に変更するか、サービスコールの呼出しにてタイムアウト指定を使用しないようにアプリケーションプログラムを修正してください。

(3) コンテキストエラー（ext_tsk サービスコール）

非タスク部から ext_tsk サービスコールが発行されました。エラー情報として次の値が渡されます。

表 5-11 エラー情報一覧(コンテキストエラー)

項 目	格納対象のスタック	内 容
ベクタ番号(vecno)	SP+0	0
タスク ID(tskid)	SP+1	0
エラーコード(ercd)	SP+2 SP+3	H'FFEB

ext_tsk を使用している箇所のアプリケーションプログラムを確認し、タスク部以外から ext_tsk サービスコールを発行しないように修正してください。

(4) コンテキストエラー（ret_int サービスコール）

タスク実行状態、または CPU ロック状態から ret_int サービスコールが発行されました。エラー情報として次の値が渡されます。

表 5-12 エラー情報一覧(コンテキストエラー)

項 目	格納対象のスタック	内 容
ベクタ番号(vecno)	SP+0	0
タスク ID(tskid)	SP+1	0
エラーコード(ercd)	SP+2 SP+3	H'FFBB

ret_int を使用している箇所のアプリケーションプログラムを確認し、割込みハンドラ以外から ret_int サービスコールを発行しないように修正してください。

(5) 未定義割込みの発生

未定義割込みが発生しました。エラー情報として次の値が渡されます。

表 5-13 エラー情報一覧(未定義割込みの発生)

項 目	格納対象のスタック		内 容
	割込みモード 0,1	割込みモード 2,3	
ベクタ番号(vecno)	SP+0	SP+0	ベクタ番号
タスク ID(tskid)	SP+1	SP+1	タスク ID または 0
EXR	ー	SP+2	割込み発生時の EXR レジスタ情報
CCR	SP+2	SP+4	割込み発生時の CCR レジスタ情報
PC	(SP+3)*1 SP+4 SP+5	(SP+5)*1 SP+6 SP+7	割込み発生時の PC レジスタ情報

【注】 *1 アドバンストモードでのみ有効で、ノーマルモードでは無効（PC の下位 16 ビットのみ有効）となります。

ベクタ番号（vecno）には発生した割込みのベクタ番号が渡されます。ベクタ番号から発生した割込みを特定してください。ベクタ番号の詳細に関しては、使用している CPU のハードウェアマニュアルをご参照ください。

必要とする割込みであれば、割込みハンドラを作成、登録してください。意図しない割込みであれば原因を調査し、割込みが発生しないようにしてください。

意図しない割込みが発生する主な要因として、以下のものが考えられます。

- 割込み要求元となる外部デバイス、または CPU 内蔵周辺モジュールのレジスタ設定ミス
- 割込みコントローラの IRQ／IRL モードの設定ミス
- 割込みコントローラの割込み優先レベル設定ミスによる、誤ったレベル割込みの認識
- ノイズによる誤った割込み要求信号の検出
- ハードウェア回路の不備など

EXR、CCR レジスタ情報からは、割込み発生時の割込みマスクレベル等の情報を得ることができます。

タスク部にて未定義割込みが発生した場合、タスク ID（tskid）からは、そのとき動作していたタスクの ID を知ることができます。

PC から該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

(6) アプリケーションプログラムからの呼出し

C 言語で記述したアプリケーションプログラムからサンプル提供のシステムダウンルーチン（_HIPRG_ABNOML）を呼出す場合、スタックには戻り番地のアドレスが格納されるため、スタックを使用したエラー情報の受け渡しを行うことはできません。

アプリケーションプログラムからサンプル提供のシステムダウンルーチンを呼出す場合、エラー要因はユーザが判断してください。

他のシステムダウン発生時と同様な形式で、スタックによるエラー情報の受け渡しを行いたい場合は、以下のようにシステムダウンルーチンを変更し、アプリケーションプログラムから呼出すシンボル名を「_HIPRG_ABNOML_CSUB」（名称はユーザ任意）にする必要があります。

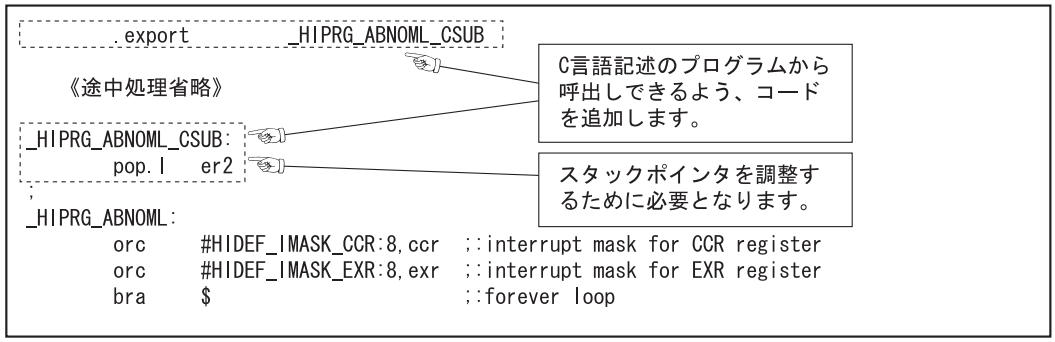


図 5-12 システムダウンルーチンの変更例(HI2000/3)

上記のようにシステムダウンルーチンを変更した場合のデバッグ用コードのコーディング例を次に示します。

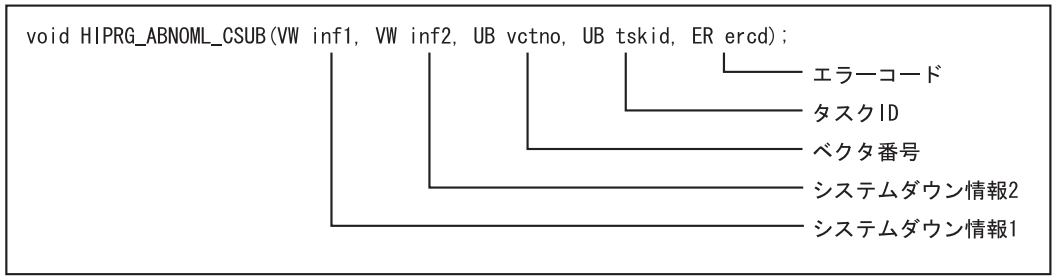


図 5-13 システムダウンルーチンの呼出し形式例(HI2000/3)

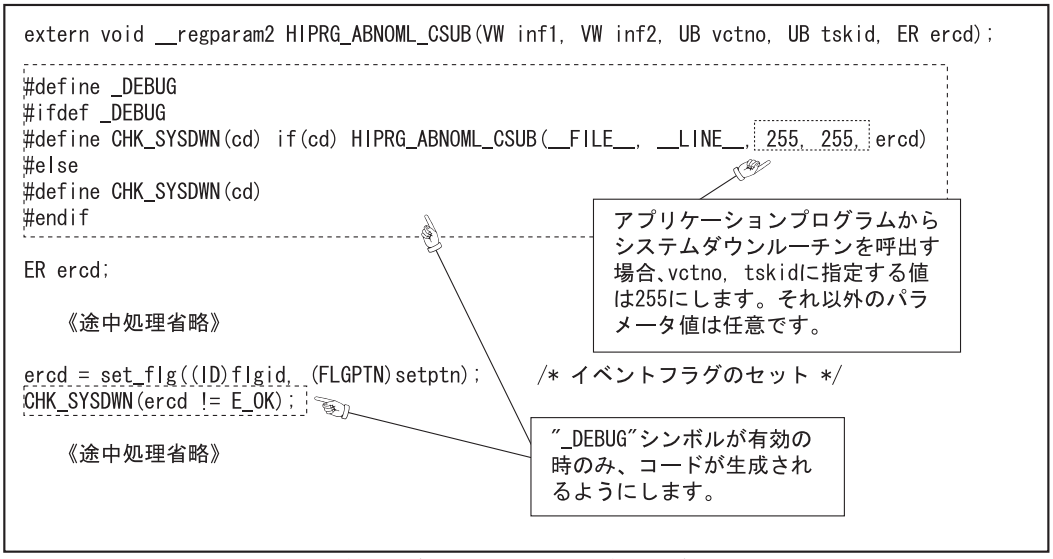


図 5-14　デバッグ用コードのコーディング例(HI2000/3)

上記のデバッグ用コードを追加し、アプリケーションプログラムから呼び出した場合、エラー情報として次の値が渡されます。

表 5-14　エラー情報一覧(アプリケーションプログラムからの呼出し)

項　目	格納対象の スタック／レジスタ	内　容
ベクタ番号(vecno)	SP+0	H'FF(255)
タスク ID(tskid)	SP+1	H'FF(255)
エラーコード(ercd)	SP+2 SP+3	発生したサービスコールのエラーコード
システムダウン情報 1	ER0	発生したソースプログラムファイルパスへのアドレス
システムダウン情報 2	ER1	発生したソースプログラムの行番号

アプリケーションプログラムからの呼出しによるシステムダウンが発生した場合、エラー情報をもとに原因を調査し、該当するアプリケーションプログラムを修正してください。

サービスコールのエラーコードに関しては、**「HI2000/3 ユーザーズマニュアル」**をご参照ください。

5.4 HI1000/4

5.4.1 デバッグの準備

(1) パラメータチェック機能の有効化

デバッグ時はサービスコールのパラメータチェック機能ありで使用することを推奨します。パラメータチェック機能については、本アプリケーションノートの**「1.3 サービスコールのパラメータチェック」**をご参照ください。

(2) デバッグ用コードの追加

サービスコールにて、パラメータエラーなど、処理を継続する上で致命的なエラーコードが返された場合、システムダウンルーチンを呼び出すよう、アプリケーションプログラムにシステムダウンルーチンを呼び出すコードを追加します。デバッグ用のコードは最終的には不要となる為、マクロやコンパイラのプリプロセッサを利用し、必要なときだけコードが生成されるようにすると便利です。

システムダウンルーチンの呼出し形式と、コーディング例を以下に示します。

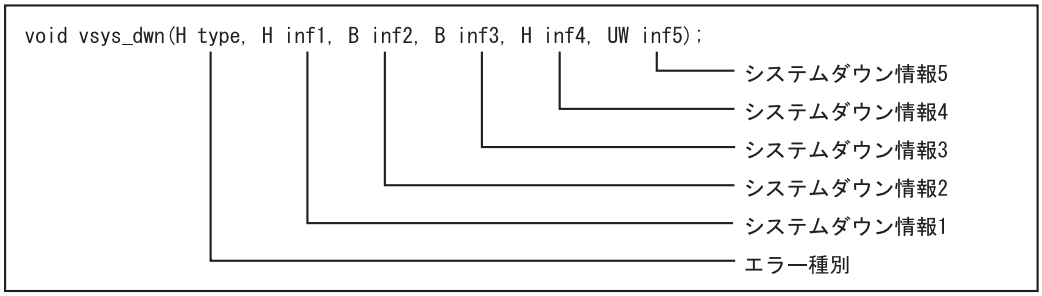


図 5-15 システムダウンルーチンの呼出し形式例(HI1000/4)

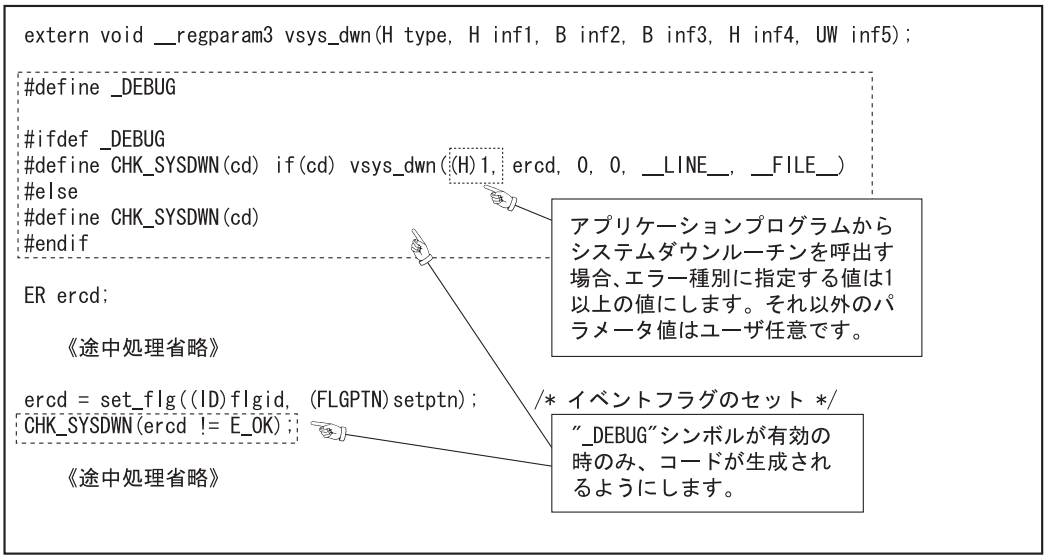


図 5-16　デバッグ用コードのコーディング例(HI1000/4)

(3)　ブレークポイントの設定

以下に示す箇所にエミュレータや ICE を使用してブレークポイントを設定し、アプリケーションプログラムを実行します。

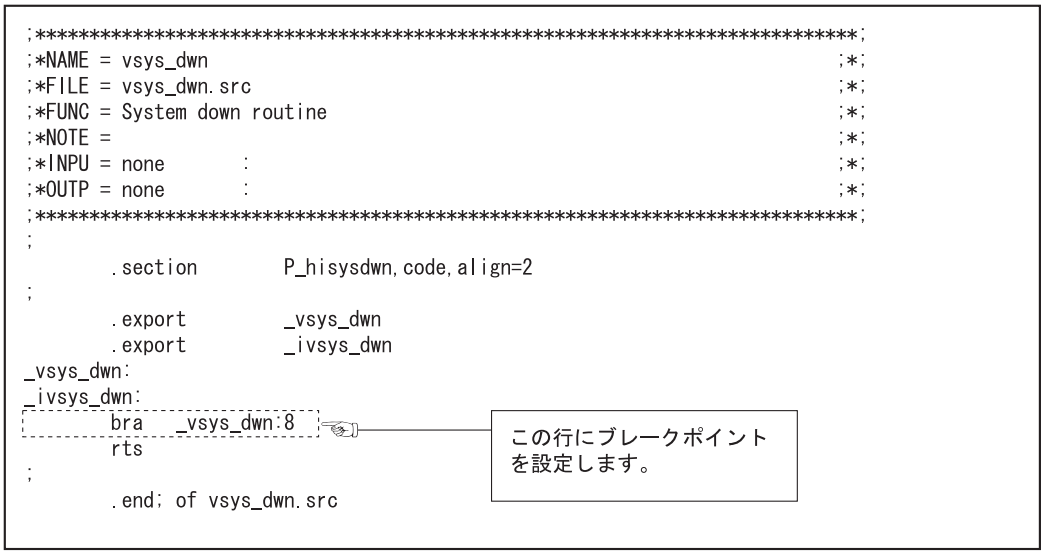


図 5-17　ブレークポイントの設定例(HI1000/4)

5.4.2 システムダウンの発生

システム異常が発生すると「5.4.1 (3) ブレークポイントの設定」にて設定したブレークポイントにてプログラムが停止します。HI1000/4 においては、システム異常発生時のエラー情報がレジスタにて渡されます。

エラー情報のパラメータ形式を以下に示します。

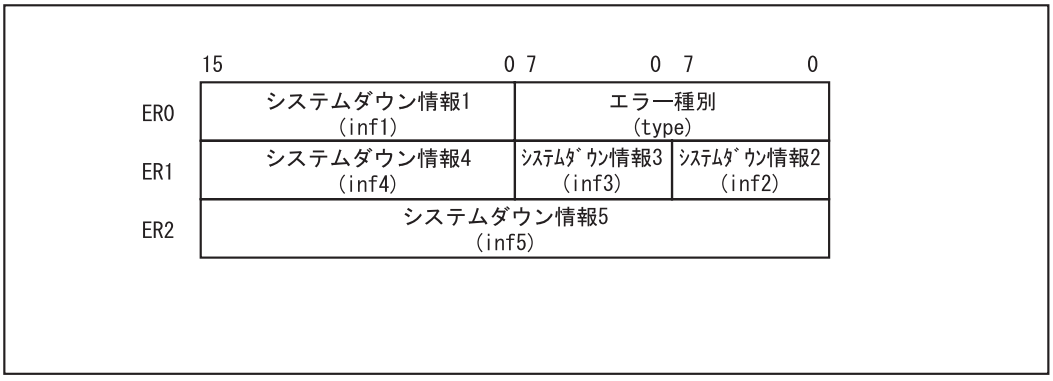


図 5-18 システムダウン情報のパラメータ形式(HI1000/4)

5.4.3 システムダウン要因の種類

HI1000/4 におけるシステムダウン要因の種類を以下に示します。

表 5-15 システムダウンの種類(HI1000/4)

項番	エラー種別(R0)	内 容
1	H'FFFB(-5)	初期登録情報エラー
2	H'FFFD(-3)	コンテキストエラー1
3	H'FFFE(-2)	コンテキストエラー2
4	H'FFFF(-1)	未定義割込みの発生
5	1 以上（ユーザ任意）	vsys_dwn、ivsys_dwn サービスコールの呼出し

以降、エラー種別毎に内容を説明します。

(1) 初期登録情報エラー

コンフィギュレータにて設定した情報に誤りがあります。エラー情報として次の値が渡されます。

表 5-16 エラー情報一覧(初期登録情報エラー)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R0	H'FFFB
システムダウン情報 1 (inf1)	E0	エラー番号（H'0000～H'0FFF）
システムダウン情報 2 (inf2)	R1L	0
システムダウン情報 3 (inf3)	R1H	0
システムダウン情報 4 (inf4)	E1	0
システムダウン情報 5 (inf5)	ER2	0

システムダウン情報 1 （inf1）には、セットアップ情報の不正内容に対応するエラー番号（H'0000～H'0FFF）が渡されます。コンフィギュレータにて、エラー番号に該当するセットアップ情報の設定値を確認してください。エラー番号の詳細に関しては、**「HI1000/4 ユーザーズマニュアル」**をご参照ください。

(2) コンテキストエラー1

カーネルがサービスコール（ext_tsk）の呼出しでコンテキストエラーを検出しました。エラー情報として次の値が渡されます。

表 5-17 エラー情報一覧(コンテキストエラー1)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R0	H'FFFD
システムダウン情報 1 (inf1)	E0	H'FFE7
システムダウン情報 2 (inf2)	R1L	発生時の CCR レジスタ情報
システムダウン情報 3 (inf3)	R1H	発生時の EXR レジスタ情報
システムダウン情報 4 (inf4)	E1	0
システムダウン情報 5 (inf5)	ER2	発生時の PC レジスタ情報

発生したアドレスに該当する部分のプログラムコードを確認し、タスク部から ext_tsk サービスコールを発行するように修正してください。

システムダウン情報 5 （inf5）にて渡される PC から該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

(3) コンテキストエラー2

カーネルが ret_int ルーチンの呼出しでコンテキストエラーを検出しました。エラー情報として次の値が渡されます。

表 5-18 エラー情報一覧(コンテキストエラー2)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R0	H'FFFE
システムダウン情報 1 (inf1)	E0	0
システムダウン情報 2 (inf2)	R1L	タスク ID
システムダウン情報 3 (inf3)	R1H	0
システムダウン情報 4 (inf4)	E1	0
システムダウン情報 5 (inf5)	ER2	0

ret_int ルーチンを使用している箇所のアプリケーションプログラムを確認し、割込みハンドラまたは例外ハンドラ以外から ret_int ルーチンを発行しないように修正してください。

(4) 未定義割込みの発生

未定義割込みが発生しました。エラー情報として次の値が渡されます。

表 5-19 エラー情報一覧(未定義割込みの発生)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R0	H'FFFF
システムダウン情報 1 (inf1)	E0	割込みベクタ番号
システムダウン情報 2 (inf2)	R1L	割込み発生時の CCR レジスタ情報
システムダウン情報 3 (inf3)	R1H	割込み発生時の EXR レジスタ情報
システムダウン情報 4 (inf4)	E1	タスク ID または 0
システムダウン情報 5 (inf5)	ER2	割込み発生時の PC レジスタ情報

システムダウン情報 1 （inf1）には発生した割込みベクタ番号が渡されます。ベクタ番号から発生した割込みを特定してください。ベクタ番号の詳細に関しては、使用している CPU のハードウェアマニュアルをご参照ください。

(a) 発生した事象が未定義割込みの場合

必要とする割込みであれば、割込みハンドラを作成、登録してください。意図しない割込みであれば原因を調査し、割込みが発生しないようにしてください。

意図しない割込みが発生する主な要因として、以下のものが考えられます。

- 割込み要求元となる外部デバイス、または CPU 内蔵周辺モジュールのレジスタ設定ミス
- 割込みコントローラの IRQ／IRL モードの設定ミス
- 割込みコントローラの割込み優先レベル設定ミスによる、誤ったレベル割込みの認識
- ノイズによる誤った割込み要求信号の検出
- ハードウェア回路の不備など

(b) 発生した事象が未定義例外の場合

必要とする例外であれば、CPU 例外またはトラップ例外ハンドラを作成、登録してください。意図しない例外であれば、システムダウン情報５（inf5）にて渡される PC から異常が発生した箇所を特定し、原因を調査する必要があります。

システムダウン情報２（inf2）、システムダウン情報３（inf3）からは、割込みマスクレベル等の情報を得ることができます。

タスクコンテキストにて未定義例外が発生した場合、システムダウン情報４（inf4）からは、そのとき動作していたタスクの ID を知ることができます。

システムダウン情報５（inf5）にて渡される PC から該当するプログラムモジュールを割り出す場合は、本アプリケーションノートの**「5.5 システムダウン発生箇所の特定」**をご参照ください。

未定義例外要因の調査に関しては、本アプリケーションノートの**「5.6 CPU 例外発生の具体例と解決策」**をご参照ください。

(5) vsys_dwn、ivsys_dwn サービスコールの呼出し

このエラーは、アプリケーションプログラムから vsys_dwn、ivsys_dwn サービスコールを呼び出した場合に発生します。渡されるエラー情報は、サービスコールを呼び出した際のパラメータ値となります。

「5.4.1 (2) デバッグ用コードの追加」に示すデバッグ用コードにおいては、エラー情報として次の値が渡されます。

表 5-20 エラー情報一覧(vsys_dwn、ivsys_dwn サービスコールの呼出し)

項 目	格納対象のレジスタ	内 容
エラー種別(type)	R0	H'1
システムダウン情報 1 (inf1)	E0	発生したサービスコールのエラーコード
システムダウン情報 2 (inf2)	R1L	0
システムダウン情報 3 (inf3)	R1H	0
システムダウン情報 4 (inf4)	E1	発生したソースプログラムの行番号
システムダウン情報 5 (inf5)	ER2	発生したソースプログラムファイルパスへのアドレス

エラー情報をもとに原因を調査し、該当するアプリケーションプログラムを修正してください。

サービスコールのエラーコードに関しては、**「HI1000/4 ユーザーズマニュアル」**をご参照ください。

5.5 システムダウン発生箇所の特定

システムダウン情報として、PC レジスタ情報が渡されます。この PC レジスタ情報によりプログラムの位置を特定するには、エミュレータや ICE のソースレベルデバッガ機能を使用するか、またはリンカから出力されるマップファイルによっておおまかな位置を特定することができます。

5.5.1 Mapviwe によるプログラム位置の割り出し

C コンパイラ付属ツール「Mapviwe」を使用して PC からプログラムモジュールの位置を特定する方法を示します。以下の例では、HI7700/4、SuperH™ RISC engine Series C/C++ Compiler Package Ver8.0.01、HEW ワークスペースのサブプロジェクトとして SH7641 用一括リンクプロジェクト（7641_mix）を使用して説明します。

Mapview を使用するには、リンカにてシンボル情報を含むマップファイルを出力する必要があります。HEW の最適化リンカオプション設定画面にて、シンボル情報を含むマップファイルを出力するように設定してください。

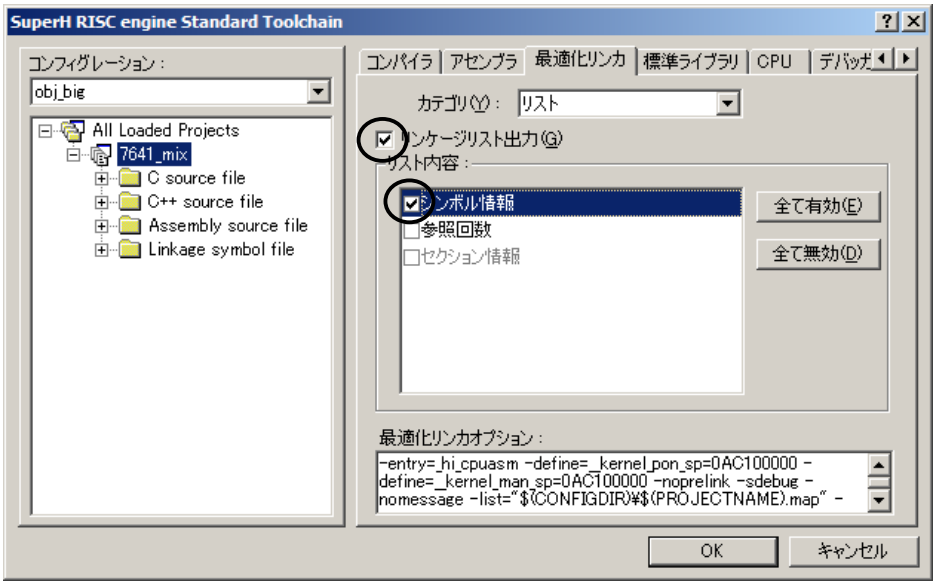


図 5-19 最適化リンカ リスト出力設定画面

(1) Mapviwe の起動

スタートメニューから［プログラム(P)］→［Renesas High-performance Embedded Workshop］→［Mapview］を選択し起動します。

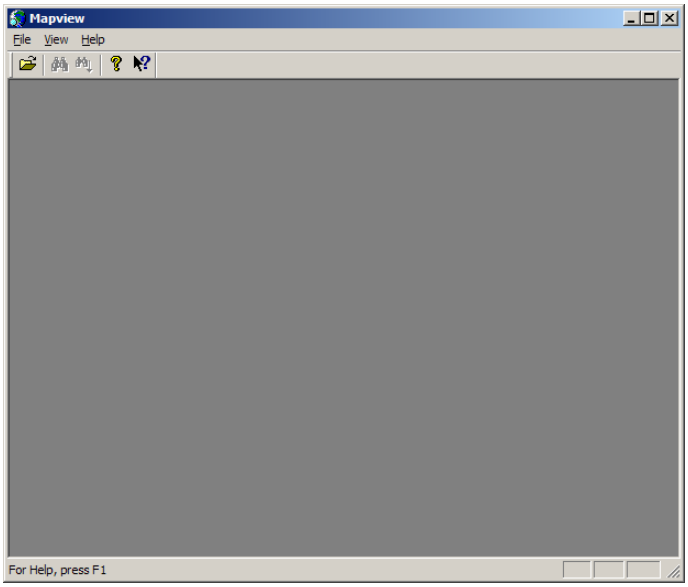


図 5-20 Mapview 起動画面

起動画面のヘッダメニュー［File］→［Open…］を選択し、リンカが出力したマップファイルを開きます。

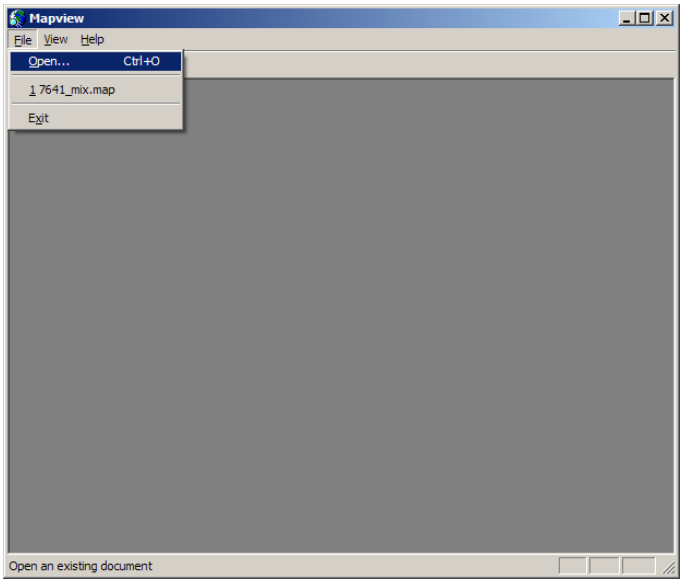


図 5-21 ファイル読み込み画面

(2) アドレスの検索

セクション名をクリックすることにより、シンボル情報ビューにそのセクションのシンボル名一覧が表示されます。表示されるアドレスとサイズにより、PC がどのシンボルに含まれるかを検索することで、システムダウンが発生したプログラムの位置を特定することができます。

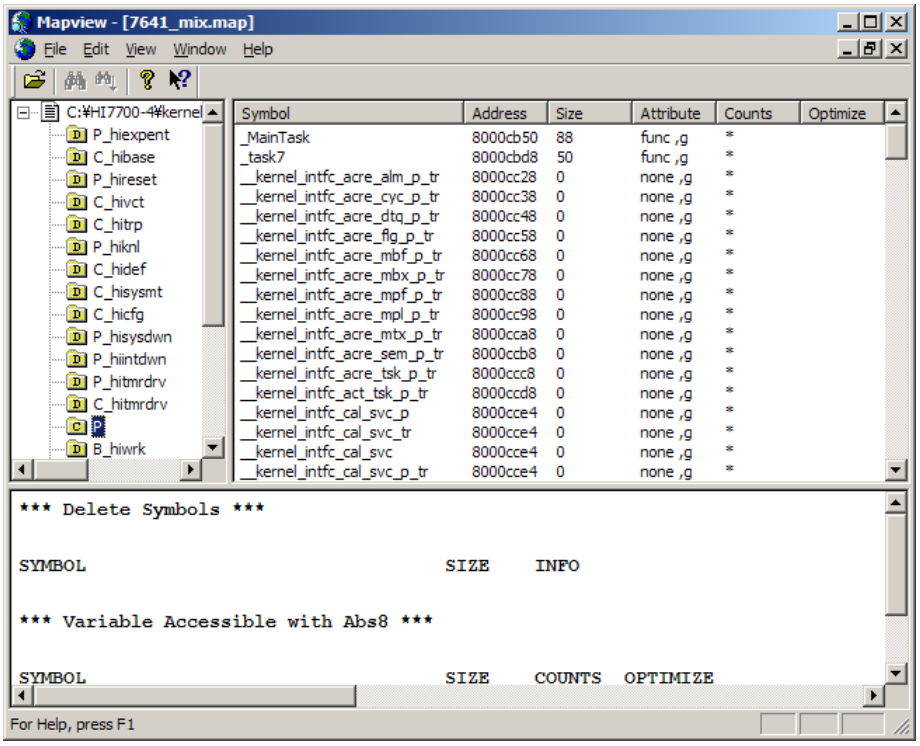


図 5-22 シンボル一覧表示画面

5.6 CPU 例外発生の具体例と解決策

CPU 例外によってシステムダウンが発生する事例をもとに、その発生の過程と、解決策を説明します。CPU 例外が発生する原因として考えられる主な要因を以下に示します。ただし、メモリマネジメントユニット（MMU）、ユーザブレークコントローラ（UBC）、トラップ命令は使用していないものとし、仮定します。

表 5-21　主なエラー要因の一覧

例外要因	例外発生位置	考えられる要因	
一般不当命令 スロット不当命令	ユーザプログラム内	直接的要因	・コンパイラの CPU オプション指定ミス（5.6.2） ・プログラム領域の破壊（5.6.3） ・ハードウェア異常（5.6.1）
		間接的要因	・スタックオーバフロー（5.6.2）
	カーネル内	直接的要因	・プログラム領域の破壊（5.6.3） ・ハードウェア異常（5.6.1）
		間接的要因	・カーネル管理領域の破壊（5.6.3） ・スタックオーバフロー（5.6.2）
	その他、 プログラムエリア外	間接的要因	・ポインタ変数による関数呼出しミス（5.6.3） ・スタックオーバフロー（5.6.2）
CPU アドレスエラー DMAC/DTC アドレスエラー	ユーザプログラム内	直接的要因	・データ境界のアクセス違反（5.6.3） ・物理アドレス空間のアクセス違反（5.6.3） ・DMAC/DTC レジスタの設定ミス ・ハードウェア異常（5.6.1）
		間接的要因	・リンカのセクション情報設定ミス（5.6.2） ・スタックオーバフロー（5.6.2）
	カーネル内	直接的要因	・ハードウェア異常（5.6.1）
		間接的要因	・カーネル管理領域の破壊（5.6.3） ・リンカのセクション情報設定ミス（5.6.2） ・スタックオーバフロー（5.6.2）
	その他、 プログラムエリア外	間接的要因	・ポインタ変数による関数呼出しミス（5.6.3） ・スタックオーバフロー（5.6.2）

直接的要因：その要因が直接、システムダウンの要因となるもの
間接的要因：その要因によってプログラムの誤動作を起こし、結果的にシステムダウンに繋がるもの
【注】カッコ内の数字は、本アプリケーションノートの参照先章番号を表します。

5.6.1 ハードウェア異常

(1) メモリ初期化ミス

外付けメモリ（SDRAM、SRAM など）を使用する場合、BSC（バスステートコントローラ）などの設定に誤りがないか、使用するメモリ全領域に対してアクセス（リード／ライト）が正しく行えるかを確認してください。エミュレーションメモリなどを使用する場合は、エミュレータの設定が正しいかを確認してください。

ハードウェアの初期化は、カーネル初期化処理を呼出す前に行う必要があります。詳細に関しては本アプリケーションノートの「2.2 CPU 初期化ルーチンの概要」をご参照ください。

5.6.2 構築によるミス

(1) コンパイラの CPU オプション設定ミス

コンパイラの CPU オプションの設定（CPU 種別、エンディアン指定）が、ターゲットハードウェアと相違がないかを確認してください。特に、SH-2、SH-3、SH-3DSP、SH-4 シリーズマイコンにおいては、ターゲットハードウェアが、ビッグエンディアンに設定されているのか、リトルエンディアンに設定されているのかを確認してください。コンパイラの CPU オプション設定画面の例を以下に示します。

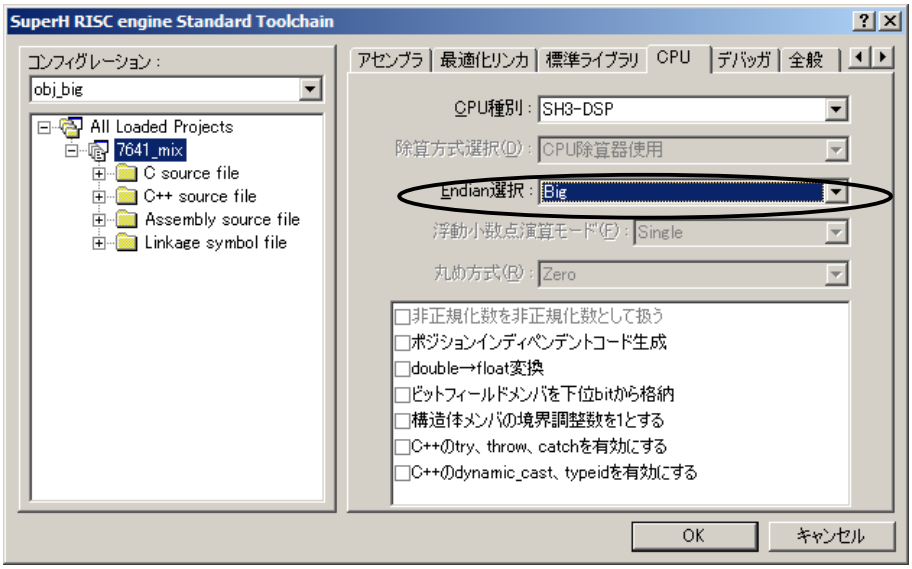


図 5-23 CPU 設定画面

(2) リンカのセクション情報設定ミス

HI シリーズ OS、およびアプリケーションプログラムが使用する作業領域（B_hixxxx、B、R セクションなど）が、利用可能な RAM 領域内に設定されているか、または RAM 容量を越えていないかを確認してください。

セクションが利用可能な RAM 領域を越えていないかを確認するには、リンカが出力するマップファイルにて確認する必要があります。マップファイルの出力方法に関しては、使用するコンパイラの

ユーザーズマニュアルを参照してください。

SuperH™ RISC engine Series C/C++ Compiler Package Ver8.0.01 が出力するマップファイルの例を以下に示します。

```
Optimizing Linkage Editor (Ver. 8.0.02.000)       03-Sep-2004 10:35:31
《途中省略》
*** Mapping List ***
SECTION                          START      END        SIZE   ALIGN
P_hiexpent
                                 80000100   800007df    6e0   4
C_hibase
                                 80001000   80001363    364   4
P_hireset
                                 80001364   80001530    1cd   4
C_hivct
                                 80001534   80001933    400   4
C_hitrp
                                 80001934   80002133    800   4
P_hiknl
                                 80002134   8000c7a7   a674   4
C_hidef
                                 8000c7a8   8000c7ef     48   4
C_hisysmt
                                 8000c7f0   8000c9c3    1d4   4
C_hicfg
                                 8000c9c4   8000ca2f     6c   4
P_hisysdwn
                                 8000ca30   8000ca4f     20   4
P_hiintdwn
                                 8000ca50   8000cab3     64   4
P_hitmrdrv
                                 8000cab4   8000cb4b     98   4
C_hitmrdrv
                                 8000cb4c   8000cb4d      2   4
P
                                 8000cb50   8000d6cf    b80   4
B_hiwrk
                                 8c000000   8c009ddb   9ddc   4
B_himpl
                                 8c009ddc   8c021ddb  18000   4
B_hidystk
                                 8c021ddc   8c025ddb   4000   4
B_histstk
                                 8c025ddc   8c026ddb   1000   4
B_hiirqstk
                                 8c026ddc   8c027fdb   1200   4
B_hitrcbuf
                                 8c027fdc   8c037fdb  10000   4
P_hicpuasm
                                 a0000000   a000002f     30   4
P_hicpuini
                                 a0000030   a0000057     28   4
```

OS、およびアプリケーションの作業領域が、RAM領域を超えていないかを確認します。

図 5-24　マップファイルのマッピングリスト部

(3) スタックオーバフロー

タスク、割込みハンドラ、初期化ルーチン、タイムイベントハンドラ、それぞれに設定されたスタックサイズが不足していないかを確認してください。

スタックサイズの算出方法に関しては、本アプリケーションノートの**「3.3 スタック使用量の算出」**をご参照ください。

スタックは決められた領域の最上位アドレスから順次使用されます。スタックが不足すると、そのスタック領域の下位アドレス（0 番地方向）の内容が破壊されてしまいます。スタックオーバフローによって例外が発生する例を以下に示します。

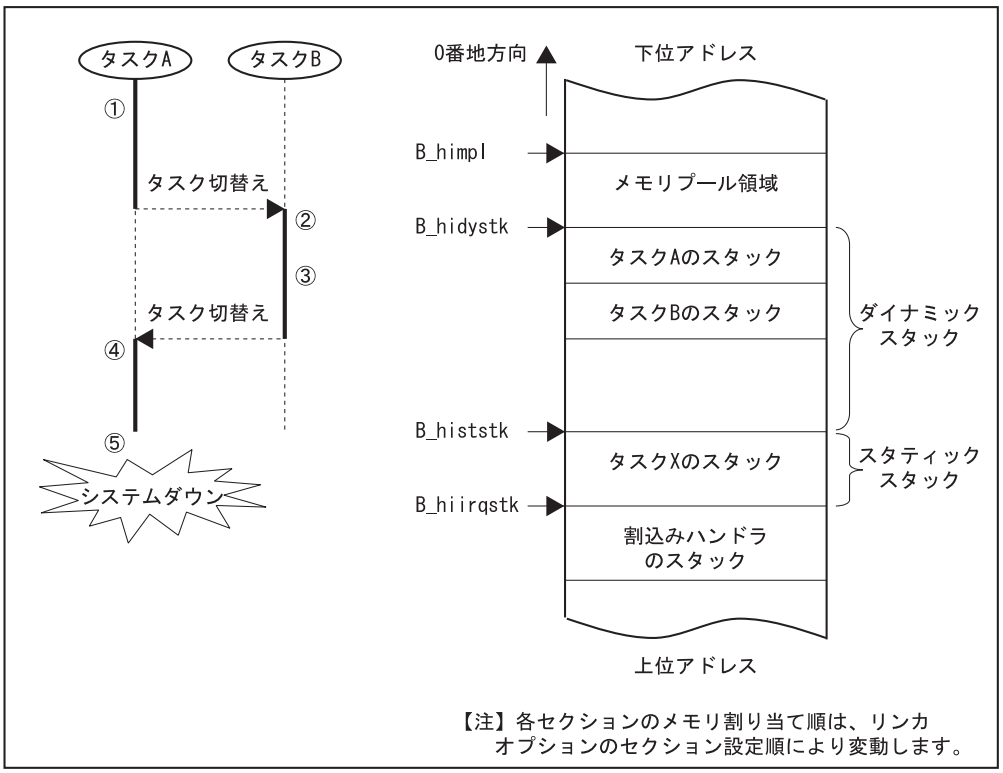


図 5-25 タスクの動作とスタック割り当て例

① タスクＡが実行されることで、タスクＡのスタックが使用されます。
② タスクの切り替えによりタスクＢが実行されると、タスクＢのスタックが使用されます。
③ タスクＢのスタックが不足した場合、その下位エリアにあるタスクＡのスタック領域が破壊されてしまいます。
④ タスクの切り替えによりタスクＡの実行が再開され、タスクＡが再びスタックの内容を使用すると、すでに内容が書き換えられているため、プログラムが誤動作を起こします。
⑤ プログラムの誤動作により、CPU 例外やハングアップを引き起こします。

スタックの下位アドレスに割り当てられているセクションによって、他のプログラム領域やメモリプール領域など破壊される領域が異なり、それによって引き起こされる現象も変わります。

5.6.3 プログラムの記述ミス

(1) カーネル管理領域の破壊

プログラムの記述ミスにより、カーネルが使用する管理領域が破壊さていないかを確認してください。特に以下の機能を使用している場合は、カーネル管理領域を破壊しないよう注意が必要です。

- メールボックス
- 可変長メモリプール

メールボックスを使用した場合の、悪いコーディング例を以下に示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

typedef struct user_msg{                    ┐
    T_MSG t_msg;                            ├ ①ユーザメッセージの型宣言
    B     data[10];                         ┘
} USER_MSG;

void Task_sub_sndmsg(VP_INT exinf)
{
    ER ercd;
    USER_MSG message;  ☜──────────────── ②ユーザメッセージの宣言

        《途中省略》

    /* ユーザメッセージの格納処理 */

    message->t_msg.msghead = 0;  ☜──── ③メッセージ内カーネル管理領域の0クリア

    ercd = snd_mbx((ID)mbxid, (T_MSG *)&message);  ☜── ④メッセージの送信
    if(ercd != E_OK){
        /* エラー処理 */
    }

    return;
}
```

図 5-26 メッセージ送信時の悪いコーディング例

メッセージ送信元のタスク優先度が、メッセージを受信するタスクの優先度より高い場合、Task_sub_sndmsg 関数からリターンした時点で、ローカル変数である message の領域が無効となります。その後、カーネルが message 内の管理領域をアクセスすることで、プログラムが誤動作を起こし、結果的にシステムダウンに繋がります。

メールボックスのメッセージデータ領域は、メモリプールから取得するなど、そのメッセージが受信されるまで内容が保持されるよう、コーディングしてください。

(2) プログラム領域の破壊

プログラム領域（ユーザプログラム、および OS を含む）が RAM 上にある場合、ユーザプログラムの記述ミスや、ハードウェア異常など、プログラム領域が書き換え、破壊されることによりシステムダウンを引き起こす場合があります。

例として、プログラム領域の破壊要因を特定する手順を次に示します。

① RAM 上のプログラム領域とロードした実行ファイルのベリファイチェックを行い、プログラム領域が書き換えられたかを確認します。
② 書き換えられた箇所に対してライトアクセスがあった場合にブレークするよう、ハードウェアブレークを設定します。
③ プログラムをロードし、再度実行します。
④ プログラムがハードウェアブレークによって停止した場合は、プログラムによって書き換えられたことが分かります。停止した位置のプログラムコードを確認してください。
⑤ ハードウェアブレークによって停止せずに、同じ領域が破壊されていた場合は、ハードウェアに異常がある可能性があります。

(3) データ境界のアクセス違反

SH-2、SH-3、SH-3DSP、SH-4 シリーズマイコンにおいて、ポインタ変数を介してメモリ操作を行っている場合、プログラムで次の記述を行っていないかを確認してください。

- ワード境界以外（2n+1 番地）のアドレスからワードデータをリード／ライトしようとした。
- ロングワード境界以外（4n+1、4n+2、4n+3 番地）のアドレスからロングワードデータをリード／ライトしようとした。

上記を行っているプログラムコードが実行された時点で、システムダウン（CPU アドレスエラー）が発生する場合があります。以下に悪いコーディング例を示します。

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"

UB buf[16];                    ←― ①グローバル変数として16バイトの領域（buf）を確保

void Task_sub1(void)
{
    UW *ptr;
    int i;

    ptr = (UW *)&buf;          ←― ②ポインタ変数（ptr）にbufの先頭アドレスを設定

    for(i=0; i < 4; i++) {
        *ptr++ = 0;            ←― ③bufの内容をゼロクリア
    }

        《途中省略》
}
```

図 5-27　システムダウンとなる悪いコーディング例

buf エリアがリンカによって 4n のアドレスにマッピングされると問題なく動作しますが、奇数また

は 2n のアドレスにマッピングされた場合、③の箇所にてシステムダウンが発生します。
この場合の対策方法としては、図中①を“UW buf[4]”と記述することで、buf 領域が必ずロングワード境界に配置されるようになり、システムダウンを回避することができます。

(4) 物理アドレス空間のアクセス違反

ポインタ変数を介してメモリ操作を行っている場合、プログラムで次の記述を行っていないかを確認してください。

- グローバル変数、またはローカル変数を未初期化で使用しようとしたことにより、予期しないエリアをアクセスした。

上記を行っているプログラムコードが実行された時点で、システムダウン（CPU アドレスエラー）が発生します。
グローバル変数、スタティック変数の未初期化データ領域（B セクション）のゼロクリアが必要な場合は、CPU 初期化ルーチンでセクションの初期化を行う必要があります。CPU 初期化ルーチンについては、本アプリケーションノートの「2.2 CPU 初期化ルーチンの概要」をご参照ください。
ローカル変数を未初期化で使用していないかを確認するには、コンパイラのインフォメーションメッセージを利用することで確認できます。ただし、コーディングの仕方によっては、インフォメーションメッセージでは確認できない場合がありますので、別途、ユーザが確認する必要があります。

HEW を使用した場合のインフォメーションメッセージの出力設定画面を以下に示します。

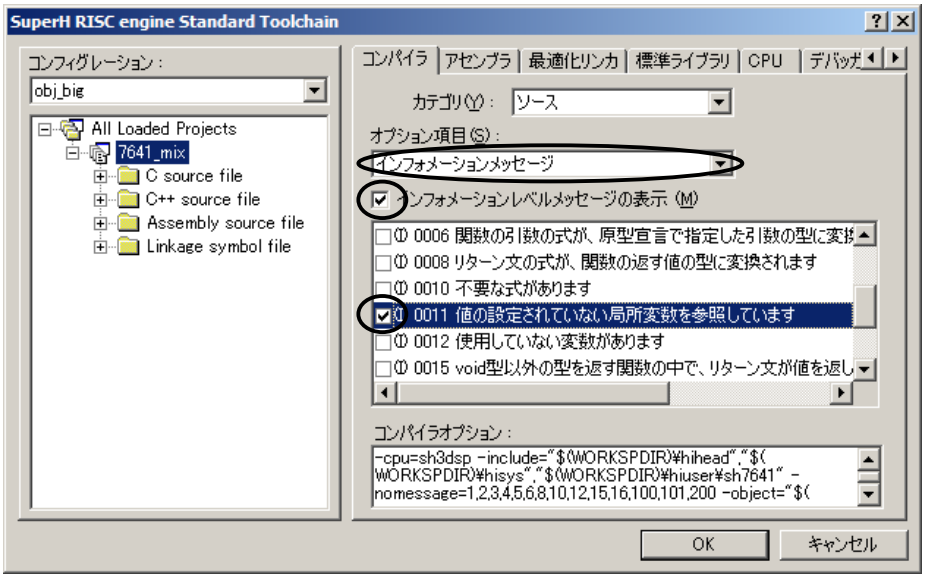


図 5-28 コンパイラ インフォメーションメッセージの設定画面

(5) ポインタ変数による関数呼出しミス

ポインタ変数を介した関数呼出しにて、ポインタ変数の値が不正となった場合、呼出し先にてプログラム実行番地が不正となり、システムダウン、またはシステムのリセットといった状態が発生します。ポインタ変数を使用した関数の呼出しを行っている場合は、そのソースコードが正しいことを十分に確認してください。

システムダウンの発生元がプログラムエリア外で、原因となっている呼出し元の判別がつかない場合は、ICE やエミュレータなどのトレース機能を使用し、プログラムの流れを確認してください。

不正なポインタ変数による関数呼出しの例を、以下に示します。

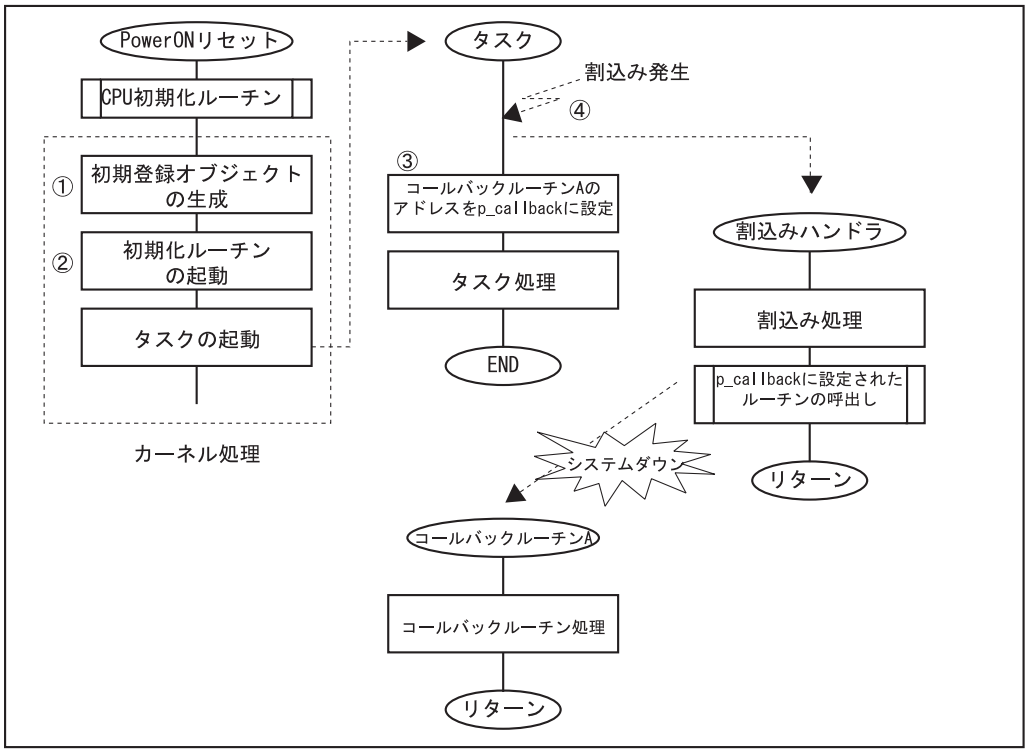


図 5-29 不正なポインタ変数による関数呼出し例

① 初期登録にて割込みハンドラを登録する。
② 初期化ルーチンにてハードウェアの割込みを有効にする。
③ 割込みが入らなければ、タスク部にてコールバックルーチンのアドレスがポインタ変数（p_callback）に設定される。
④ コールバックルーチンのアドレスを設定する前に割込みが入った場合、不正なアドレスを呼び出すこととなり、実行アドレスが不正、システムダウンが発生する。

上記のようなケースでは、コールバックルーチンのアドレスが決定されていない場合は、コールバックルーチンを呼び出さないか、または、ポインタ変数（p_callback）設定前に割込みが入らないよう対策を行う必要があります。

5.7 デバッグ手法に関する FAQ

HI シリーズ OS をご使用頂いているお客様より、数多く寄せられたデバッグ手法に関する質問についての回答を記述します。

《 FAQ目次 》

5.7.1 プログラムの ROM 化

分類：デバッグ手法

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

ICE 上では問題なく動作するプログラムが、ROM 化すると正常に動作しません。原因を教えてください。

■回答

プログラム実行開始時に、セクションの初期化を行う必要があります。

C 言語で記述したプログラムの初期化データ領域（D セクション）は、プログラム実行開始時に ROM から RAM へコピーしなければなりません。したがって、初期化データ領域は ROM 上と RAM 上に二重に割り当てる必要があります。これを行うには、リンケージエディタの ROM 化支援機能を使用します。ROM 化支援機能に関しては、使用するクロスコンパイラのユーザーズマニュアルを参照してください。

セクションの初期化は、CPU 初期化ルーチンにて行います。
次に、各 HI シリーズ OS で提供している CPU 初期化ルーチンを例に、セクションの初期化方法を示します。

【次ページに続く】

【前ページからの続き】

■回答

```
/************************************************************************/
/*  FILE      = 7604_cpuini.c ;                                         */
/*  CPU type  = SH7604                                                  */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

/* extern  void    _INITSCT(void); */   /* section-initialize routine       */

#pragma section _hicpuini
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Software Environment ***/

/*  _INITSCT();    */                   /* Call section-initialize routine  */

    vsta_knl();                         /* Start kernel                     */
}
```

コメント"/* */"を外し、
「セクションの初期化」処理
を呼出します。

図 5-30　CPU 初期化ルーチン記述例(HI7000/4 シリーズ)

```
/************************************************************************/
/*  FILE      = 7604_initsct.c  ;                                       */
/************************************************************************/
#include    <machine.h>
#include    "itron.h"

extern int *B_BGN, *B_END, *D_BGN, *D_END, *D_ROM;
extern void _INITSCT(void);

#pragma section _hicpuini
/************************************************************************/
/*  NAME      = _INITSCT ;                                              */
/*  FUNCTION  = Section Initialize routine ;                            */
/************************************************************************/
void _INITSCT(void)
{
    register int *p, *q;
    for(p=B_BGN; p<B_END; p++)          /* 0 clear B-section                */
        *p = 0;
    for(p=D_BGN,q=D_ROM; p<D_END; p++, q++) /* Copy D-section -> R-section  */
        *p = *q;
}
```

図 5-31　セクション初期化処理の記述例(HI7000/4 シリーズ)

【次ページに続く】

【前ページからの続き】

■回答

```
_H_2S_CPUINI:
        mov.l   #CPUINI_SP:32,sp        ;:get CPUINI_SP
        mov.b   @SYSCR:32,r0L           ;:get SYSCR
        and.b   #low~(INTMO|INTM1):8,r0L;:clear interrupt mode bit
        or.b    #low (INTMO|INTM1):8,r0L;:set interrupt mode = 3
        mov.b   r0L,@SYSCR:32           ;:set SYSCR
;
        mov.b   @MSTPCRH:32,r0L         ;:get MSTPCRH
        and.b   #low TPU:8,r0L          ;:set TPU bit off
        mov.b   r0L,@MSTPCRH:32         ;:set MSTPCRH
;
        .aifdef DX
        jsr     @_HI_DEAMON_INI         ;:call to init deamon code
        .aendi
;
        jsr     @_h_cpuini_c            ;:call to C-language initialize routine
;
        jmp     @_H_2S_INIT             ;:goto HI2000/3 initialize module
;
```

C言語記述のCPU初期化ルーチンの呼出しを追加します。

【注】本例では、C言語記述のCPU初期化ルーチンを「h_cpuini_c」と仮定しています。

図 5-32　CPU 初期化ルーチン記述例(HI2000/3)

```
void h_cpuini_c(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Hardware Environment ***/

    _INITSCT();                 /* Call section-initialize routine  */

}
```

必要に応じて、
・バスステートコントローラの初期化
・外付けメモリ(SDRAM)の初期化
を追加してご使用ください。

標準ライブラリのセクション初期化処理をコールし、未初期化データをゼロクリア、初期化データをROMからRAMにコピーします。

図 5-33　セクション初期化処理の呼出し例(HI2000/3)

【次ページに続く】

【前ページからの続き】

■回答

```
_KERNEL_H_CPUINI:
        mov.l   #_KERNEL_HI_OS_SP:32,sp ;:SP <- OS stack
        mov.l   #VBR_ADR,er0            ;:
        ldc.l   er0,vbr                 ;:set VBR address
        mov.l   #h'ffffff00,er0         ;:initial SBR
        ldc.l   er0,sbr                 ;:initial SBR
;
;       mov.w   #h'00ff,@ABWCR:32       ;:set ABWCR
;       mov.w   #h'0000,@ASTCR:32       ;:set ASTCR
;       mov.w   #h'0000,@WTCRA:32       ;:set WTCRA
;       mov.w   #h'0000,@WTCRB:32       ;:set WTCRB
;
        mov.b   #INTM1,r0L              ;:set interrupt mode 2
        mov.b   r0L,@INTCR:32           ;:set INTCR
;
        mov.w   @MSTPCRA:32,r0          ;:get MSTPCRA
        and.w   #MSTPA0:16,r0           ;:set TPU bit off
        mov.w   r0,@MSTPCRA:32          ;:set MSTPCRA
;
    jmp     @_h_cpuini_c
;
```

C言語記述のCPU初期化ルーチンの呼出しを追加します。

【注】本例では、C言語記述のCPU初期化ルーチンを「h_cpuini_c」と仮定しています。

図 5-34　CPU 初期化ルーチン記述例(HI1000/4)

```
void h_cpuini_c(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Hardware Environment ***/

    _INITSCT;                       /* Call section-initialize routine  */

    vsta_knl();                     /* Start kernel */
}
```

必要に応じて、
・バスステートコントローラの初期化
・外付けメモリ(SDRAM)の初期化
を追加してご使用ください。

標準ライブラリのセクション初期化処理をコールし、未初期化データをゼロクリア、初期化データをROMからRAMにコピーします。

図 5-35　セクション初期化処理の呼出し例(HI1000/4)

5.7.2 メモリプール使用時のシステムダウン

分類：デバッグ手法

HI7000/4
HI7700/4
HI7750/4
HI2000/3
HI1000/4

■質問

可変長メモリプールにて、メモリブロックの獲得／返却を行うとシステムダウンとなります。原因を教えてください。

■回答

可変長メモリプールから獲得したメモリブロックに対して、使用可能な領域を超えてメモリを使用していることが考えられます。

可変長メモリプールにおいては、メモリブロックを獲得するとカーネルが使用する 16 バイトのカーネル管理領域がメモリプールから獲得されます。その時の構成は、次のようになります。

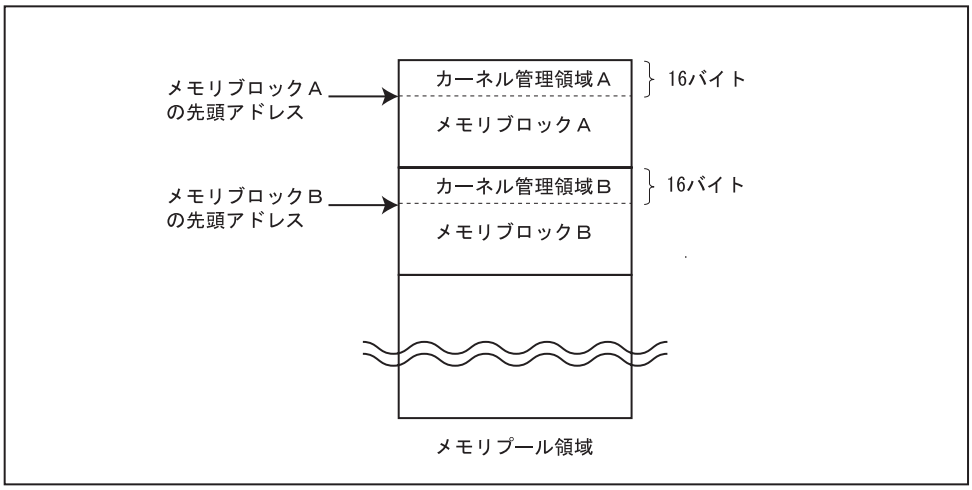


図 5-36 可変長メモリブロックの構成

【次ページに続く】

■回答

カーネル管理領域がユーザプログラムによって誤って書き換えられた場合、カーネルがメモリブロックの解放のためカーネル管理領域をアクセスすることによってシステムダウンやハングアップに陥ります。

獲得したメモリブロックの領域を超えてメモリを使用しているかを確認する手段として、ICEやエミュレータを使用した以下の方法が上げられます。

① メモリブロックを獲得する際、メモリブロックサイズに要求サイズ＋4 を指定します。
② 獲得したメモリブロックの終端アドレス（先頭アドレス＋要求サイズ＋1）にハードウェアブレークを設定し、このアドレスに対してリード／ライトが発生したらブレークするようにします。
③ プログラムを実行します。

設定したハードウェアブレークによってプログラムが停止した場合、使用可能なメモリブロックの領域を超えてユーザプログラムが使用しようとしたことが分かります。

ルネサスマイクロコンピュータ開発環境システム
アプリケーションノート
HIシリーズOS

発行年月日　2003 年 12 月 5 日　Rev.1.00
　　2004 年 12 月 13 日　Rev.3.00
発　　行　株式会社ルネサス テクノロジ 営業企画統括部
　　〒100-0004 東京都千代田区大手町 2-6-2
編　　集　株式会社ルネサス小平セミコン 技術ドキュメント部

株式会社ルネサス テクノロジ　営業企画統括部　〒100-0004 東京都千代田区大手町2-6-2 日本ビル

営業お問合せ窓口
株式会社ルネサス販売

http://www.renesas.com

本社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
京浜支社	〒212-0058	川崎市幸区鹿島田890-12 (新川崎三井ビル)	(044) 549-1662
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
札幌支店	〒060-0002	札幌市中央区北二条西4-1 (札幌三井ビル5F)	(011) 210-8717
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 (損保ジャパンいわき第二ビル3F)	(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
中部営業本部	〒460-0008	名古屋市中区栄3-13-20 (栄センタービル4F)	(052) 261-3000
浜松支店	〒430-7710	浜松市板屋町111-2（浜松アクトタワー10F）	(053) 451-2131
西部営業本部	〒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
鹿児島支店	〒890-0053	鹿児島市中央町12-2 (明治安田生命鹿児島中央町ビル)	(099) 284-1748

■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：カスタマサポートセンタ　E-Mail: csc@renesas.com

Colophon 3.0

HI シリーズ OS
アプリケーションノート

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

RJJ05B0317-0300