User's Manual

RI600V4

リアルタイム・オペレーティング・システム
ユーザーズマニュアル　コーディング編

対象デバイス

RXファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.04　2013.09

ご注意書き

1. 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・システムの設計において、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て、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   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、産業用ロボット等

   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、各種安全装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図し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損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。
9. 本資料に記載されている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・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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禁じます。

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。

(2012. 4)

このマニュアルの使い方

対 象 者　このマニュアルは，RX ファミリの各製品の応用システムを設計，開発するユーザを対象としています。

目　　的　このマニュアルは，次の構成に示すルネサス エレクトロニクス製リアルタイム OS RI600V4 の機能をユーザに理解していただくことを目的としています。

構　　成　このマニュアルは，大きく分けて次の内容で構成しています。

読 み 方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータ，C 言語，アセンブラの一般知識を必要とします。

RX MCU のハードウエア機能を知りたいとき
→各製品のユーザーズマニュアルを参照してください。

凡　　例

データ表記の重み:	左が上位桁，右が下位桁
注	: 本文中につけた注の説明
注意	: 気をつけて読んでいただきたい内容
備考	: 本文中の補足説明
数の表記	: 10 進数　...　XXXX
	16 進数　...　0xXXXX
2 のべき乗を示す接頭辞（アドレス空間，メモリ容量）:	
	K（キロ）　$2^{10} = 1024$
	M（メガ）　$2^{20} = 1024^2$
up4(*data*)	: *data* を 4 の倍数に切り上げた値
down(*data*)	: *data* の整数部

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

資 料 名		資料番号	
		和文	英文
RI シリーズ	起動編	R20UT0751J	R20UT0751E
	メッセージ編	R20UT0756J	R20UT0756E
RI600V4	コーディング編	このマニュアル	R20UT0711E
	デバッグ編	R20UT0775J	R20UT0775E
	解析編	R20UT2185J	R20UT2185E

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料を使用してください。

目　　次

第 1 章 概 説

1.1 概 要

RI600V4 は，効率のよいリアルタイム処理環境，および，マルチタスク処理環境を提供するとともに，対象 CPU の制御機器分野における応用範囲を拡大することを目的として開発された“ リアルタイム・マルチタスク OS” です。

また，実行環境に組み込んで使用することを前提として開発されているため，ROM 化を意識し，コンパクトな設計が行われています。

RI600V4 は，“ リアルタイム・マルチタスク OS” として普及している μITRON4.0 仕様に準拠しています。

1.1.1 リアルタイム OS

制御機器分野におけるシステムでは，内外の事象変化に対するリアルタイム性が要求されます。しかし，従来のシステムでは，このような要求をユーザが用意した単純な割り込み処理で対処してきたため，制御機器が高性能化，多様化するにつれ，単純な割り込み処理だけでの対処が困難になってきています。

つまり，処理プログラム量の増大，システムの複雑化により，内外の事象変化に対する処理を“ どのような順序で実行させるのか ” を管理することが煩雑になってきたといえます。

そこで，このような問題を解決するために考えられたのが“ リアルタイム OS” です。

リアルタイム OS は，内外の事象変化に対するリアルタイム性を保証するとともに，最適な処理プログラムを最適な順序で実行させることを主な目的（仕事）としています。

1.1.2 マルチタスク OS

OS の世界では, OS の管理下で実行する処理プログラムを“ タスク ”, 1 つのプロセッサ上で複数のタスクを同時実行させることを“ マルチタスキング ” と呼んでいます。

しかし，厳密にはプロセッサ自体は一度に 1 つのタスク（命令）しか実行することができないため，タスクの実行を何らかの基準（きっかけ）を利用して非常に短い間隔で切り替えることにより，疑似的に複数のタスクが同時実行しているかのように見せています。

このように，システム内で規定されている何らかの基準を利用してタスクを切り替え，タスクの並列処理を可能としたのが“ マルチタスク OS” です。

マルチタスク OS は，複数のタスクを並列実行させることにより，システム全体の処理能力を向上させることを主な目的（仕事）としています。

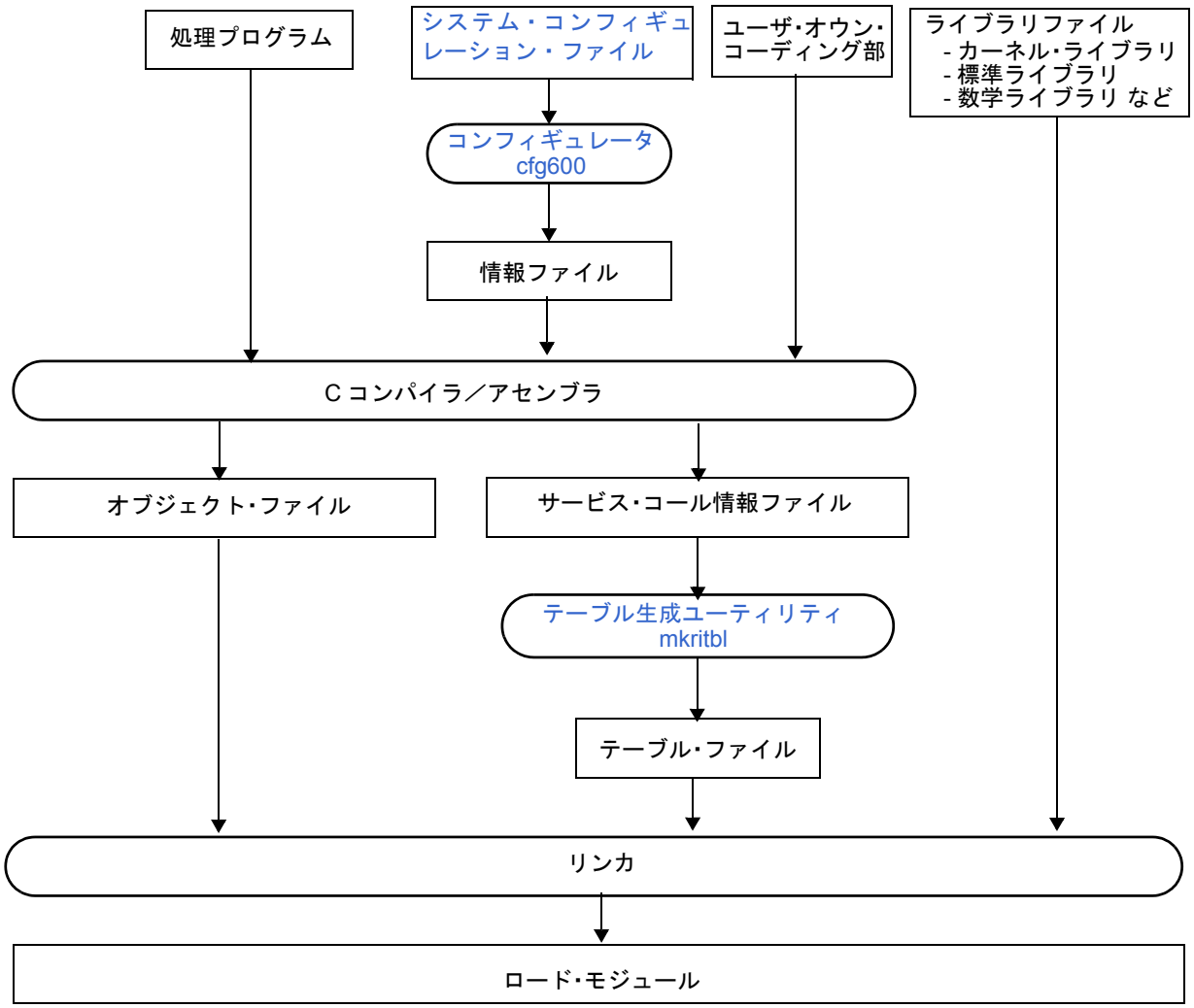
第 2 章　システム構築

本章では, RI600V4 が提供している機能を利用したシステム（ロード・モジュール）の構築手順について解説しています。

2.1　概　　要

システム構築とは，RI600V4 の提供媒体からユーザの開発環境（ホスト・マシン）上にインストールされたファイル群（カーネル・ライブラリなど）を用いてロード・モジュールを生成することです。
図２－１に，システム構築の手順を示します。

図２－１　システム構築の手順



RI600V4 では，ロード・モジュールを生成する際に必要となるファイル群のサンプル・プログラムを提供しています。
サンプル・プログラムは，以下のフォルダに格納されています。ソース・ファイルは，appli サブ・フォルダに格納されています。

<ri_sample> = <CubeSuite+_root>¥SampleProjects¥RX¥ *デバイス名* _RI600V4

- <CubeSuite+_root>
  CubeSuite+ のインストール・フォルダを表します。
  デフォルトでは，“ C:¥Program Files¥Renesas Electronics¥CubeSuite+” となります。

- SampleProjects
  CubeSuite+ のサンプル・プロジェクト・フォルダです。

- RX
  RX MCU 用のサンプル・プロジェクト・フォルダです。

- *デバイス名* _RI600V4
  RI600V4 のサンプル・プロジェクト・フォルダです。このフォルダに，プロジェクト・ファイルがあります。
  デバイス名 : サンプルを提供しているデバイス名を表しています。

2.2 処理プログラムの記述

システムとして実現すべき処理を記述します。
なお，RI600V4 では，処理プログラムを実現すべき処理の種類，および，用途にあわせて以下に示した 4 種類に分類しています。

- タ ス ク
  他処理プログラム（割り込みハンドラ，周期ハンドラ，およびアラーム・ハンドラ）とは異なり，RI600V4 が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。

- 周期ハンドラ
  指定された周期時間毎に起動されるルーチンです。
  なお，RI600V4 では，周期ハンドラを “ タスクとは独立したもの（非タスク）” として位置づけています。このため，一定の時間が経過した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，周期ハンドラに制御が移ります。

- アラーム・ハンドラ
  指定した時間後に一度だけ起動されるルーチンです。
  なお，RI600V4 では，アラーム・ハンドラを “ タスクとは独立したもの（非タスク）” として位置づけています。このため，一定の時間が経過した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，アラーム・ハンドラに制御が移ります。

- 割り込みハンドラ
  割り込みが発生した際に起動されるルーチンです。
  なお，RI600V4 では，割り込みハンドラを “ タスクとは独立したもの（非タスク）” として位置づけています。このため，割り込みが発生した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。

備考 処理プログラムについての詳細は，「第 3 章 タスク管理機能」，「第 8 章 時間管理機能」，「第 10 章 割り込み管理機能」を参照してください。

2.3 システム・コンフィギュレーション・ファイルの記述

RI600V4 に提供するデータを保持した情報ファイルを生成する際に必要となるシステム・コンフィギュレーション・ファイルを記述します。

備考 1 システム・コンフィギュレーション・ファイルについての詳細は，「第 19 章 システム・コンフィギュレーション・ファイル」を参照してください。

備考 2 リアルタイム OS タスク・アナライザを「ソフトウェア・トレース・モードでトレース・チャートを取得」または「ソフトウェア・トレース・モードで長時間統計を取得」で使用する場合は，システム・コンフィギュレーション・ファイルに，ユーザ・オウン・コーディング部で作成した割り込みハンドラを定義する必要があります。詳細は，「第 15 章 リアルタイム OS タスク・アナライザ」を参照してください。

2.4 ユーザ・オウン・コーディング部の記述

- システム・ダウン
  - システム・ダウン・ルーチン（_RI_sys_dwn__( )）
    システム・ダウン・ルーチンは，システム・ダウンが発生したときに呼び出されます。
- リアルタイム OS タスク・アナライザ
  - ソフトウェア・トレース・モードのユーザ・オウン・コーディング部
    ソフトウェア・トレース・モードを使用する場合は，タイム・スタンプを取得するためのユーザ・オウン・コーディング部を実装する必要があります。
- システム初期化処理
  - ブート処理関数（PowerON_Reset_PC( )）
    ブート処理は，リセットベクタに登録され，RI600V4 が処理を実行するうえで必要となる最低限のハードウエアやソフトウエアを初期化するためにユーザ・オウン・コーディング部として切り出された初期化処理専用ルーチンです。また，cfg600 が出力した ROM 定義ファイルと RAM 定義ファイルを取り込む役割も担います。
  - セクション情報ファイル（ユーザ・オウン・コーディング部）
    初期化する未初期化データ・セクションおよび初期化データ・セクションを定義します。

備考 ユーザ・オウン・コーディング部についての詳細は，「第 13 章 システム・ダウン」，「第 15 章 リアルタイム OS タスク・アナライザ」，「第 16 章 システム初期化処理」を参照してください。

2.5 ロード・モジュールの生成

「2.2 処理プログラムの記述」から「2.4 ユーザ・オウン・コーディング部の記述」で作成されたファイル群，および，RI600V4，C コンパイラ・パッケージが提供しているライブラリ・ファイルに対して，CubeSuite+ 上でビルドを実行し，ロード・モジュールを生成します。

1） プロジェクトの作成／読み込み
プロジェクトの新規作成，または既存のプロジェクトの読み込みを行います。

備考 プロジェクトの新規作成，および既存のプロジェクトの読み込みについての詳細は，「RI シリーズ リアルタイム・オペレーティング・システム ユーザーズマニュアル 起動編」，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」，および本製品のリリースノートを参照してください。

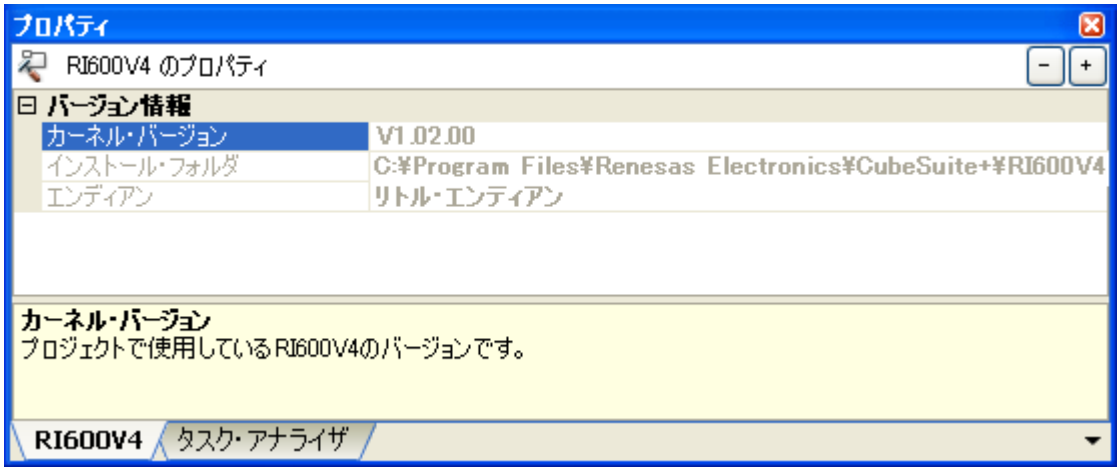
2） ビルド対象プロジェクトの設定
ビルドの設定や実行を行う場合は，アクティブ・プロジェクトを設定します。
なお，サブプロジェクトがない場合，プロジェクトは常にアクティブになります。

備考 アクティブ・プロジェクトの設定についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

3） バージョンの確認

プロジェクト・ツリーでリアルタイム OS ノードを選択し，プロパティ パネルをオープンします。
［RI600V4］タブの［カーネル・バージョン］プロパティにおいて，使用する RI600V4 のバージョンを確認します。

図２－２ プロパティ パネル：［RI600V4］タブ



4） ビルド対象ファイルの設定

プロジェクトへのビルド対象ファイルの追加／削除，依存関係の更新などを行います。

備考 プロジェクトへのビルド対象ファイルの追加／削除，依存関係の更新についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

以下に，ロード・モジュールを生成する際に必要となるファイル群の一覧を示します。

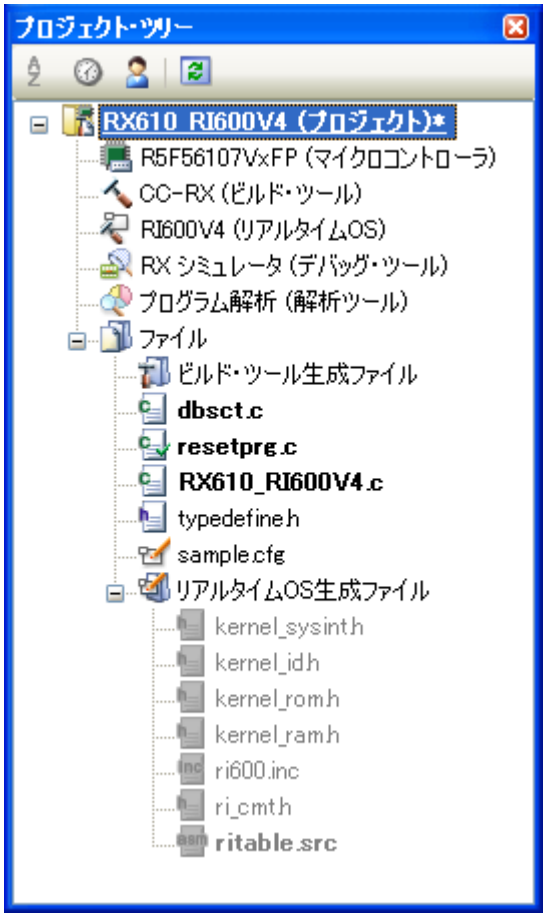
- 「2.2 処理プログラムの記述」で作成されたソース・ファイル
  - 処理プログラム（タスク，周期ハンドラ，アラーム・ハンドラ，割り込みハンドラ）
- 「2.3 システム・コンフィギュレーション・ファイルの記述」で作成されたシステム・コンフィギュレーション・ファイル
  - システム・コンフィギュレーション・ファイル

  備考 システム・コンフィギュレーション・ファイル名の拡張子は，“cfg” を指定してください。
  拡張子が異なる場合は，“cfg” が自動的に付加されます（例えば，ファイル名に “aaa.c” を指定した場合，“aaa.c.cfg” となります）。
- 「2.4 ユーザ・オウン・コーディング部の記述」で作成されたソース・ファイル
  - ユーザ・オウン・コーディング部（システム・ダウン・ルーチン，ブート処理）
- RI600V4 が提供しているライブラリ・ファイル
  - カーネル・ライブラリ（「2.6.3 カーネル・ライブラリ」参照）
- C コンパイラ・パッケージが提供しているライブラリ・ファイル
  - 標準ライブラリ，数学ライブラリなど

備考 1 プロジェクト・ツリー パネルにシステム・コンフィギュレーション・ファイルを追加すると，リアルタイム OS 生成ファイル・ノードが表示されます。
リアルタイム OS 生成ファイル・ノードには，以下の情報ファイルが表示されます。ただし，この時点では，これらのファイルは生成されません。
- システム情報ヘッダ・ファイル（kernel_id.h）
- サービス・コール定義ファイル（kernel_sysint.h）

- ROM 定義ファイル（kernel_rom.h）
- RAM 定義ファイル（kernel_ram.h）
- システム定義ファイル（ri600.inc）
- CMT タイマ定義ファイル（ri_cmt.h）
- テーブル・ファイル（ritable.src）

図２－３　プロジェクト・ツリー パネル



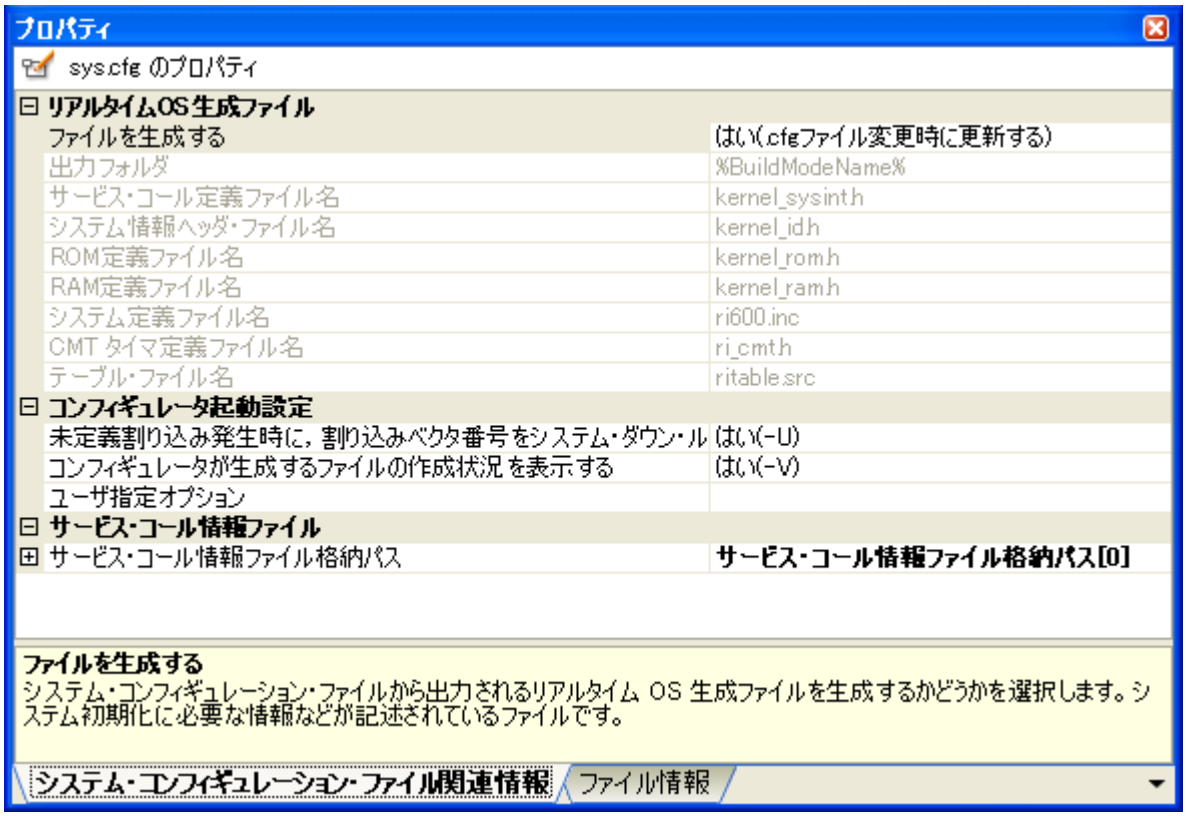
備考２　システム・コンフィギュレーション・ファイルを差し替える場合は，追加しているシステム・コンフィギュレーション・ファイルを一旦プロジェクトから外したのち，再度ファイルを追加してください。

備考３　システム・コンフィギュレーション・ファイルは，プロジェクトに複数追加することができますが，有効となるのは最初に追加したファイルです。有効なファイルをプロジェクトから外しても，追加済みのファイルは有効にならないため，再度ファイルを追加してください。

5） リアルタイム OS 生成ファイルの出力指定

プロジェクト・ツリーでシステム・コンフィギュレーション・ファイルを選択し，プロパティ パネルをオープンします。
［システム・コンフィギュレーション・ファイル関連情報］タブにおいて，リアルタイム OS 生成ファイルの出力設定などを行います。

図２－４　プロパティ パネル：［システム・コンフィギュレーション・ファイル情報］タブ



6） ロード・モジュール・ファイルの出力指定

ビルドの生成物として，ロード・モジュール・ファイルを出力することを設定します。

備考　ロード・モジュールの出力指定についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

7） ビルド・オプションの設定

コンパイラ，アセンブラ，リンカなどに対するオプションを設定します。
必ず「2.6　ビルド・オプション」を参照してください。

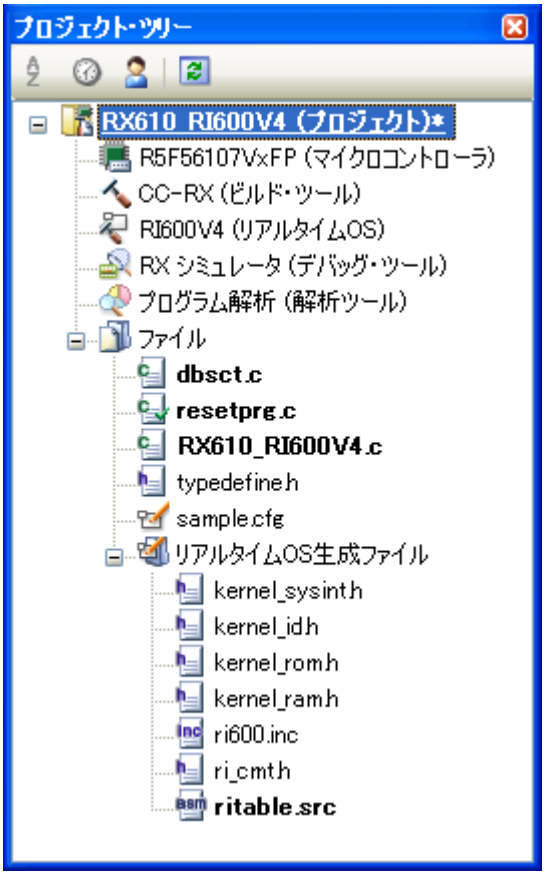
備考　ビルド・オプションの設定についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

8） ビルドの実行

ビルドを実行し，ロード・モジュール・ファイルを生成します。

備考　ビルドの実行についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

図２－５　プロジェクト・ツリー パネル（ビルド実行後）



9） プロジェクトの保存

プロジェクトの設定情報をプロジェクト・ファイルに保存します。

備考　プロジェクトの保存についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

2.6 ビルド・オプション

ここでは，特に留意すべきビルド・オプションについて解説します。

2.6.1 サービス・コール情報ファイルとコンパイラ・オプション “-ri600_preinit_mrc”

サービス・コール情報ファイル（mrc ファイル）は，kernel.h をインクルードするファイルのコンパイルによって，オブジェクト・ファイルと同じフォルダに生成されます。

mrc ファイルには，ソース中で使用しているサービス・コール名が出力されます。テーブル生成ユーティリティ mkritbl には，mrc ファイルが格納されたパスを漏れなく指定する必要があります。漏れがある場合，アプリケーションで使用しているサービス・コール・モジュールがリンクされない場合があります。この場合，そのサービス・コールを呼び出したときにシステム・ダウンとなります。

逆に，過去に生成され，現在は無効な mrc ファイルを mkritbl に入力した場合は，アプリケーションで使用していないサービス・コール・モジュールがリンクされる場合があります。この場合，RI600V4 の動作に問題は生じませんが，コード・サイズが無駄に大きくなってしまうことになります。

また, kernel.h をインクルードするファイルのコンパイル時には, オプション “-ri600_preinit_mrc” を指定してください。本オプションを指定しなくても RI600V4 の動作に問題は生じませんが，アプリケーションで使用していないサービス・コール・モジュールがリンクされる場合があります。この場合，RI600V4 の動作に問題は生じませんが，コード・サイズが無駄に大きくなってしまうことになります。

アプリケーションをライブラリ化する場合は, コンパイル時に生成された mrc ファイルも mkritbl に入力してください。これが難しい場合，使用しているサービス・コール名を羅列した mrc ファイル（下記例を参照）を作成し，mkritbl に入力してください。

なお，組み込まれていないサービス・コールを呼び出した場合は，システム・ダウンとなります。

```
sta_tsk
snd_mbx
rcv_mbx
prcv_mbx
```

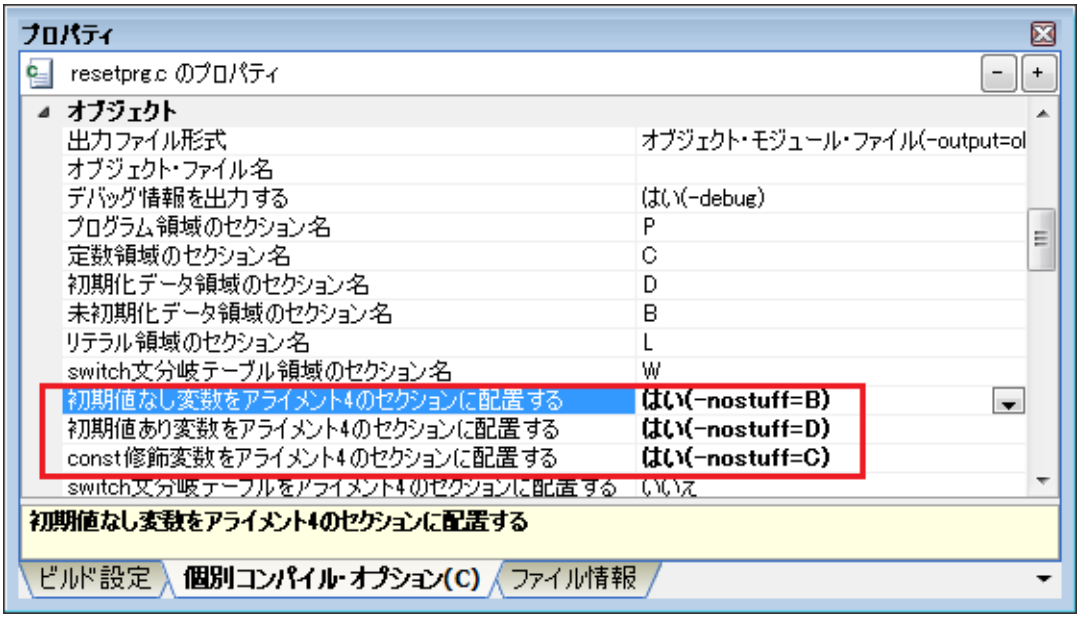
2.6.2 ブート処理ファイルのコンパイラ・オプション

「16.2.3 ブート処理ファイルのコンパイラ・オプション」に記載のように，ブート処理ファイル（サンプル・プロジェクトでは “resetprg.c”）にはオプション “-nostuff” を設定する必要があります。そうでない場合，RI600V4 は正常に動作しません。

オプション “-nostuff” をブート処理ファイルのみに設定するにはブート処理ファイルの [プロパティ] パネルの [個別コンパイル・オプション] タブで，オプション “-nostuff” を全ファイルに設定するには [CC-RX（ビルド・ツール）] の [プロパティ] パネルの [コンパイル・オプション] タブで，以下のいずれかを設定してください。

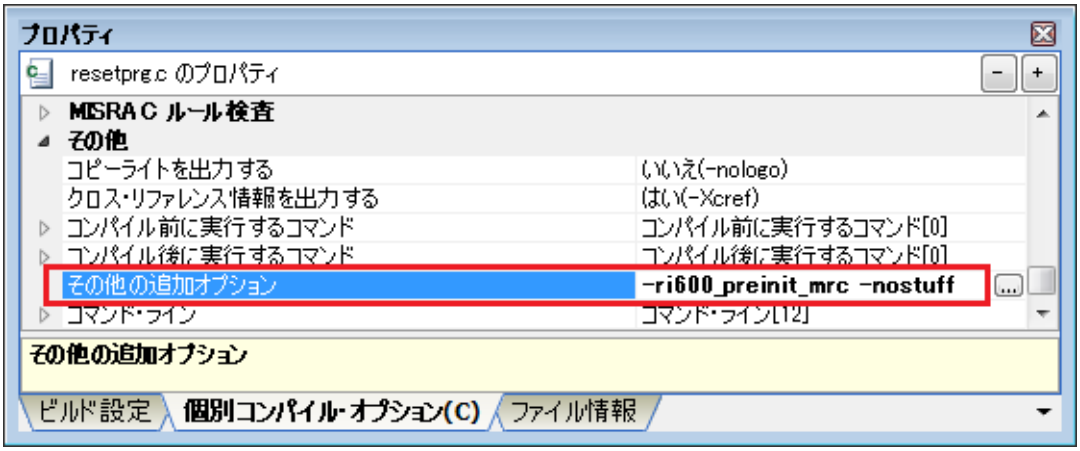
1） [オブジェクト] カテゴリで設定する
図 2 － 6 のように，[初期値なし変数をアライメント 4 のセクションに配置する]，[初期値あり変数をアライメント 4 のセクションに配置する], および [const 修飾変数をアライメント 4 のセクションに配置する] を ” はい ” に設定してください。

図２－６ [オブジェクト]カテゴリ



2） [その他]カテゴリで設定する
図２－７ のように，[その他の追加オプション]に “-nostuff” を追加してください。

図２－７ [その他]カテゴリ



2.6.3 カーネル・ライブラリ

カーネル・ライブラリは，表２－１に示すフォルダに格納されています。ただし，CubeSuite+ が自動的に適切なカーネル・ライブラリをリンクするので，ユーザはカーネル・ライブラリを意識する必要はありません。

表２－１ カーネル・ライブラリ

	フォルダ	対応コンパイラ・バージョン	対応 CPU コア	ファイル名	説明
1	<ri_root>¥library¥rxv1	V1.02.01 以降	RXv1 アーキテクチャ	ri600lit.lib	リトル・エンディアン用
				ri600big.lib	ビッグ・エンディアン用
2	<ri_root>¥library¥rxv2	V2.01.00 以降	RXv1 アーキテクチャおよび RXv2 アーキテクチャ	ri600lit.lib	リトル・エンディアン用
				ri600big.lib	ビッグ・エンディアン用

備考１ <ri_root> は，RI600V4 のインストール・フォルダを表しています。
デフォルトは，“ C:¥Program Files¥Renesas Electronics¥CubeSuite+¥RI600V4” です。

備考２ コンパイラ CC-RX V2.01.00 以降を使用時は項番 2, それ以外の場合は項番 1 のライブラリがリンクされます。

2.6.4 セクション配置

セクション配置は，リンカのオプション “-start” で指定します。CubeSuite+ では，[CC-RX（ビルド・ツール）] プロパティの [リンク・オプション] タブの [セクション] カテゴリで設定します。

1） RI600V4 のセクション
表 2 － 2 に，RI600V4 のセクションを示します。

表 2 － 2 RI600V4 セクション一覧

セクション名	属性	アライメント数	ROM/RAM	意味
PRI_KERNEL	CODE	1	ROM/RAM	RI600V4 プログラム
CRI_ROM	ROMDATA	4	ROM/RAM	RI600V4 定数
FIX_INTERRUPT_VECTOR	ROMDATA	4	ROM	固定ベクタ・テーブル／例外ベクタ・テーブルです。後述の「FIX_INTERRUPT_VECTOR セクション」を参照してください。
INTERRUPT_VECTOR	ROMDATA	4	ROM/RAM	可変ベクタ・テーブル（1KB）
BRI_RAM	DATA	4	RAM	RI600V4 の変数セクションです。データ・キュー領域も含まれます。
DRI_ROM	ROMDATA	4	ROM/RAM	RI600V4 の初期化データです。サイズは 4 バイト固定です。
RRI_RAM	DATA	4	RAM	
BRI_TRCBUF	DATA	4	RAM	本セクションは，［タスク・アナライザ］タブで [ソフトウェア・トレース・モードでトレース・チャートを取得] および [カーネルのバッファ] を選択した場合にのみ生成されます。サイズは，［タスク・アナライザ］タブで指定します。
BRI_HEAP	DATA	4	RAM	メッセージ・バッファ，可変長メモリ・プール，固定長メモリ・プール領域に付与するセクション名は，システム・コンフィギュレーション・ファイルで指定することができます。 これを省略した場合，その領域のセクション名がBURI_HEAP となります。
SI	DATA	4	RAM	システム・スタック
SURI_STACK	DATA	4	RAM	タスクのユーザ・スタックに付与するセクション名は，システム・コンフィギュレーション・ファイルで指定することができます。これを省略した場合，ユーザ・スタックのセクション名が SURI_STACK となります。

2） FIX_INTERRUPT_VECTOR セクション

コンフィギュレータ cfg600 は，システム・コンフィギュレーション・ファイルの “interrupt_fvector[]” の定義内容に従って，固定ベクタ・テーブル／例外ベクタ・テーブルを FIX_INTERRUPT_VECTOR セクションとして生成します。

- RXv1 アーキテクチャ使用時
  RXv1 アーキテクチャでは，固定ベクタ・テーブルは 0xFFFFFF80 番地に固定されています。FIX_INTERRUPT_VECTOR セクションは 0xFFFFFF80 番地に配置してください。
  FIX_INTERRUPT_VECTOR セクションを 0xFFFFFF80 番地以外に配置した場合は，システム・コンフィギュレーション・ファイルの “interrupt_fvector[]” はすべて無視され，固定ベクタ・テーブルに割り当てられている例外要因（リセットを除く）が発生した時にシステム・ダウンとなる機能は正常に動作しません。ユーザ側で固定ベクタ・テーブルを 0xFFFFFF80 番地に生成してください。

- RXv2 アーキテクチャ使用時
  RXv2 アーキテクチャでは，RXv1 アーキテクチャの「固定ベクタ・テーブル」は「例外ベクタ・テーブル」と呼称が変更され，その先頭アドレスを EXTB レジスタで設定できるようになりました。EXTB レジスタのリセット時の初期値は，RXv1 アーキテクチャの固定ベクタ・テーブルと同じ 0xFFFFFF80 です。
  FIX_INTERRUPT_VECTOR セクションは，通常はリンク時に 0xFFFFFF80 番地に配置してください。
  FIX_INTERRUPT_VECTOR セクションを 0xFFFFFF80 番地以外に配置した場合は，システム・コンフィギュレーション・ファイルの “interrupt_fvector[31]”（リセット・ベクタ）は無視されます。ユーザ側でリセット・ベクタ（0xFFFFFFFC 番地）を用意してください。また，ブート処理関数（PowerON_Reset_PC( )）で EXTB レジスタを FIX_INTERRUPT_VECTOR セクションの先頭アドレスに初期化してください。

3） 0 番地に関する注意

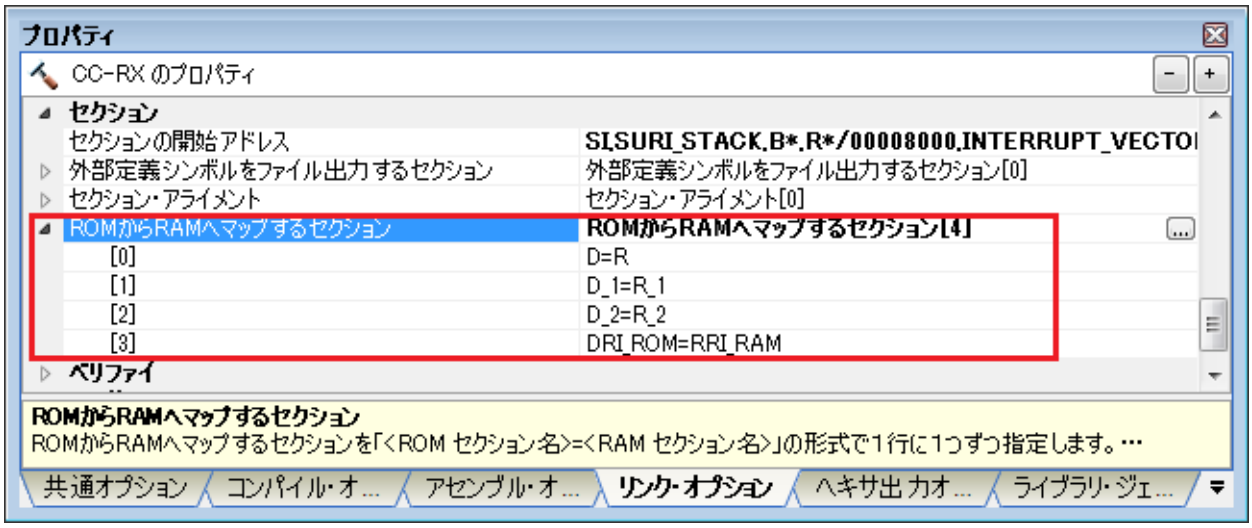
以下が 0 番地とならないようにしてください。

- 固定長メモリ・プール領域
- 可変長メモリ・プール領域
- メールボックスに送信するメッセージ

2.6.5 初期化データ・セクション

セクション情報ファイル（ユーザ・オウン・コーディング部）の DTBL に記述した初期化データセクションは，リンカのオプション “-rom” を用いて ROM に配置されたセクションを RAM 上のセクションにマップする設定を行う必要があります。図２－８のように，[CC-RX（ビルド・ツール）] の [プロパティ] パネルの [リンク・オプション] タブの [セクション] カテゴリを設定してください。

図２－８ ROM から RAM へマップするセクション



備考 RI600V4 が提供するサンプルプロジェクトでは, RI600V4 の “DRI_ROM” セクションの “RRI_RAM” セクションへのマップを設定済みです。

2.6.6 リアルタイム OS タスク・アナライザに関するオプション

［タスク・アナライザ］タブのトレース・モードの設定に応じて，表２－３に示すビルド・オプションが自動設定されます。ただし，この自動設定機能は，対応する機能のプロパティパネルとは連動していないため，自動設定された内容を対応する機能のプロパティパネルで変更してはなりません。

表２－３ リアルタイム OS タスク・アナライザ用に自動設定されるオプション

トレース・モード	アセンブラのオプション	リンカのオプション
ハードウェア・トレース・モードでトレース・チャートを取得	-define=TRCMODE=1	なし
ソフトウェア・トレース・モードでトレース・チャートを取得，カーネルのバッファを使用	-define=TRCMODE=2 -define=TRCBUFSZ=<*バッファ・サイズ*> また，「バッファを使い切った後の動作」として「トレースを停止する」を選択した場合は，さらに以下を設定します。 -define=TRCBUFMODE=1	なし
ソフトウェア・トレース・モードでトレース・チャートを取得，その他のバッファを使用	-define=TRCMODE=2 また，「バッファを使い切った後の動作」として「トレースを停止する」を選択した場合は，さらに以下を設定します。 -define=TRCBUFMODE=1	-define=__RI_TRCBUF =<*バッファ・アドレス*> -define=__RI_TRCBUFSZ =<*バッファ・サイズ*>
ソフトウェア・トレース・モードで長時間統計を取得	-define=TRCMODE=3	なし
トレースしない	なし	なし

備考１ “TRCMODE” は，以下のファイルで利用しています。

- ritable.src：コンフィギュレータ cfg600 が出力
- trcSW_cmt.src :「ソフトウェア・トレース・モードでトレース・チャートを取得」用のユーザ・オウン・コーディング部のサンプル
- trcLONG_cmt.src :「ソフトウェア・トレース・モードで長時間統計を取得」用のユーザ・オウン・コーディング部のサンプル

備考２ “TRCBUFSZ” および “TRCBUFMODE” は，“ritable.src” で利用しています。

第 3 章　タスク管理機能

本章では，RI600V4 が提供しているタスク管理機能について解説しています。

3.1　概　　要

RI600V4 におけるタスク管理機能では，タスクの生成／起動／終了などといったタスクの状態を操作する機能のほかに，優先度の参照，タスク詳細情報の参照などといったタスクの状態を参照する機能も提供しています。

3.2　タ ス ク

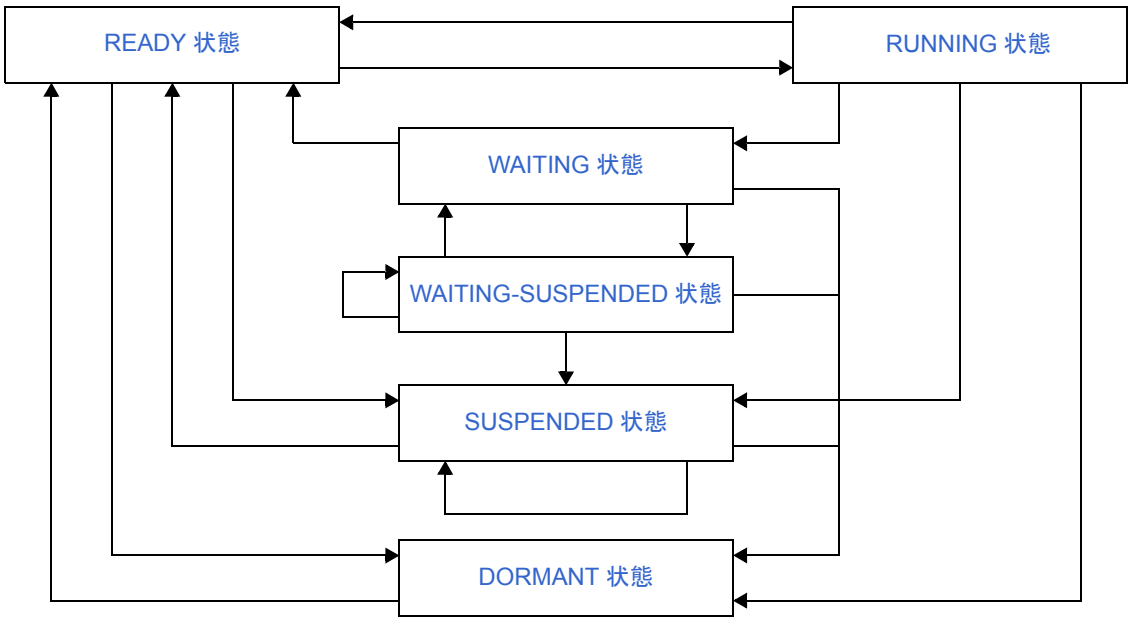
タスクは，他処理プログラム（割り込みハンドラ，周期ハンドラ，およびアラーム・ハンドラ）とは異なり，RI600V4 が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。

備考　タスクが処理を実行するうえで必要となるプログラム・カウンタ，汎用レジスタなどの実行環境情報は，“タスク・コンテキスト”と呼ばれ，タスクの実行が切り替わる際には，現在実行中のタスクのタスク・コンテキストがセーブされ，次に実行されるタスクのタスク・コンテキストがロードされます。

3.2.1　タスクの状態

タスクは，処理を実行するうえで必要となる資源の獲得状況，および，事象発生の有無などにより，さまざまな状態へと遷移していきます。そこで，RI600V4 では，各タスクが現在どのような状態にあるかを認識し，管理する必要があります。
なお，RI600V4 では，タスクが取り得る状態を以下に示した 6 種類に分類し，管理しています。

図 3 － 1　タスクの状態遷移



1） DORMANT 状態

タスクとして起動されていない状態，またはタスクとしての処理を終了した際に遷移する状態です。
なお，DORMANT 状態のタスクは，RI600V4 の管理下にありながらも，RI600V4 のスケジューリング対象からは除外されています。

2） READY 状態

処理を実行するうえで必要となる準備は整っているが，より高い優先度（同一優先度の場合もある）を持つタスクが処理を実行中のため，CPU の利用権が割り当てられるのを待っている状態です。

3） RUNNING 状態

CPU の利用権が割り当てられ，処理を実行中の状態です。
なお，RUNNING 状態のタスクは，システム全体を通して同時に複数存在することはありません。

4） WAITING 状態

処理を実行するうえで必要となる条件が整わないため，処理の実行が中断した状態です。
なお，WAITING 状態からの処理再開は，処理の実行が中断した箇所からとなります。したがって，処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ，汎用レジスタなど）は，中断直前の値が復元されます。
また，RI600V4 では，要求条件の種類により，WAITING 状態を以下に示す 12 状態に細分化し，管理しています。

表 3 － 1　WAITING 状態の種類

状態種別	状態遷移するサービス・コール
起床待ち状態	slp_tsk または tslp_tsk
時間経過待ち状態	dly_tsk
資源獲得待ち状態	wai_sem または twai_sem
イベントフラグ待ち状態	wai_flg または twai_flg
データ送信待ち状態	snd_dtq または tsnd_dtq
データ受信待ち状態	rcv_dtq または trcv_dtq
メッセージ受信待ち状態	rcv_mbx または trcv_mbx
ミューテックス待ち状態	loc_mtx または tloc_mtx
メッセージ・バッファ送信待ち状態	snd_mbf または tsnd_mbf
メッセージ・バッファ受信待ち状態	rcv_mbf または trcv_mbf
固定長メモリ・ブロック待ち状態	get_mpf または tget_mpf
可変長メモリ・ブロック待ち状態	get_mpl または tget_mpl

5） SUSPENDED 状態

強制的に処理の実行を中断させられた状態です。
なお，SUSPENDED 状態からの処理再開は，処理の実行が中断した箇所からの再開となります。したがって，処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ，汎用レジスタなど）は，中断直前の値が復元されます。

6） WAITING-SUSPENDED 状態

WAITING 状態と SUSPENDED 状態が複合した状態です。
なお，WAITING 状態が解除された際には SUSPENDED 状態へ，SUSPENDED 状態が解除された際には WAITING 状態へと遷移します。

3.2.2　タスクの優先度

タスクには，処理を実行するうえでの優先順位を決定する優先度が付けられています。そこで，RI600V4 のスケジューラでは，実行可能な状態（RUNNING 状態および READY 状態）にあるタスクの優先度を参照し，その中から最も高い優先度（最高優先度）を持つタスクを選び出し，CPU の利用権を与えています。

なお，RI600V4 では，“ タスクの優先度 ” を以下に示した 2 種類に分類し，管理しています。

- 現在優先度
  RI600V4 は，以下の処理を現在優先度にしたがって実施します。
  - タスクのスケジューリング（「14.4　タスクのスケジューリング方式」参照）
  - 優先度順の待ちキューへのキューイング

  備考　タスクが DORMANT 状態から READY 状態へと遷移した直後の現在優先度（初期優先度）は，タスク生成時に指定します。

- ベース優先度
  ミューテックスを使用しない限りは，ベース優先度と現在優先度は同じです。ミューテックスを使用する場合は，「6.2.2　現在優先度とベース優先度」を参照してください。

備考 1　RI600V4 におけるタスクの優先度は，その値が小さいほど，高い優先度であることを意味します。

備考 2　システム内で利用可能な優先度の範囲については，システム・コンフィギュレーション・ファイル作成時にシステム情報（system）のタスク優先度の最大値（priority）を定義することにより実現されます。

3.2.3　タスクの基本型

以下に，タスクを記述する場合の基本型を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ............
        ............

        ext_tsk ( );                    /* タスクの終了 */
}
```

備考 1　*exinf* には，以下の情報が渡されます。

起動方法	*exinf*
タスク情報（task[]）でタスクを定義した際に，TA_ACT 属性（initial_start）に ON を指定	タスク情報（task[]）の拡張情報（exinf）
act_tsk または iact_tsk	
sta_tsk または ista_tsk	sta_tsk または ista_tsk で指定した起動コード *stacd*

備考 2　タスク内で return 命令が発行された場合，ext_tsk と同等の処理が実行されます。

備考 3　タスクの拡張情報についての詳細は，「3.4　タスクの起動」を参照してください。

備考 4　タスク関数のプロトタイプ宣言は，cfg600 が kernel_id.h に出力します。

3.2.4 タスク内での処理

RI600V4 では，タスクを切り替える際，独自のスケジューリング処理を行っています。このため，タスクを記述する際には，以下に示す注意点があります。

- スタック
  タスクは，タスク情報（task[]）において指定されたユーザ・スタックを使用します。

- サービス・コールの発行
  タスクでは，“ 発行有効範囲 ” が “ タスク ” のサービスコールを発行可能です。

- 処理開始時の PSW

表 3 － 2 タスク処理開始時の PSW

ビット	値	備考
I	1	全割り込み受け付け可能
IPL	0	
PM	1	ユーザ・モード
U	1	ユーザ・スタック
C, Z, S, O	不定	
その他	0	

- 処理開始時の FPSW
  システム情報（system）のタスク・コンテキスト・レジスタ（context）に FPSW を含む設定をした場合の処理開始時の FPSW を表 3 － 3 に示します。それ以外の場合は不定です。

表 3 － 3 タスク処理開始時の FPSW

コンパイラ・オプション		値
-round	-denormalize	
nearest（デフォルト）	off（デフォルト）	0x00000100（DN ビットのみ 1)
	on	0
zero	off（デフォルト）	0x00000101（DN ビットと RM ビットのみ 1)
	on	1（RM ビットのみ 1）

3.2.5 タスク実行時のプロセッサ・モード

タスクは，常にユーザ・モードで実行されます。タスクををスーパバイザ・モードで実行させることはできません。
スーパバイザ・モードで実行させたい処理は，INT 命令の割り込みハンドラとして実装してください。
例えば，CPU を低消費電力モードに遷移させる WAIT 命令は特権命令なので，スーパバイザ・モードで実行する必要があります。
なお，INT #1 ～ 8 は，RI600V4 で予約されているので，使用しないでください。

3.3　タスクの生成

RI600V4 では，タスクの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

タスクの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“task[]” を使用してタスクを定義することをいいます。

静的 API“task[]” の詳細は，「19.7　タスク情報（task[]）」を参照してください。

3.4　タスクの起動

RI600V4 では，タスクの起動において，“ 起動要求をキューイングする ”，“ 起動要求をキューイングしない ” の 2 種類のインタフェースを用意しています。

3.4.1　起動要求をキューイングする起動

タスクの起動（起動要求をキューイングする）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- act_tsk，iact_tsk

  パラメータ *tskid* で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RI600V4 のスケジューリング対象となります。ただし，本サービス・コールを発行した際，対象タスクが DORMANT 状態以外の場合には，対象タスクのキューイング処理，および，状態操作処理は行わず，対象タスクに起動要求をキューイング（起動要求カウンタに 1 を加算）しています。

  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        ............
        ............

        act_tsk ( tskid );              /* タスクの起動（起動要求をキューイングする）*/

        ............
        ............
}
```

備考 1　RI600V4 が管理する起動要求カウンタは，8 ビット幅で構成されています。このため，本サービス・コールでは，起動要求数が 255 回を超える場合には，起動要求の発行起動要求の発行（起動要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

備考 2　本サービス・コールの発行により起動されたタスクには，“ タスク情報（task[]）で指定した拡張情報 ” が渡されます。

3.4.2　起動要求をキューイングしない起動

タスクの起動（起動要求をキューイングしない）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sta_tsk，ista_tsk

  パラメータ *tskid* で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RI600V4 のスケジューリング対象となります。ただし，本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスクが DORMANT 状態以外の場合には，対象タスクの状態操作処理は行わず，戻り値として E_OBJ を返します。
  なお，パラメータ *stacd* には，対象タスクに引き渡す拡張情報を指定します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        VP_INT  stacd = 123;            /* 変数の宣言，初期化 */

        ............
        ............

        sta_tsk ( tskid, stacd );       /* タスクの起動（起動要求をキューイングしない）*/

        ............
        ............
}
```

3.5　起動要求のキューイング解除

起動要求のキューイング解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- can_act，ican_act
  パラメータ *tskid* で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0 を設定）します。
  なお，正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER_UINT ercd;                   /* 変数の宣言 */
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = can_act ( tskid );       /* 起動要求のキューイング解除 */

        if ( ercd >= 0 ) {
                .............           /* 正常終了処理 */
                .............
        }

        .............
        .............
}
```

備考　本サービス・コールでは，状態操作処理は行わず，起動要求カウンタの設定処理のみを行います。したがって，READY 状態などから DORMANT 状態に遷移することはありません。

3.6 タスクの終了

3.6.1 自タスクの終了

自タスクの終了は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ext_tsk

自タスクを RUNNING 状態から DORMANT 状態へと遷移させ，レディ・キューから外します。これにより，自タスクは，RI600V4 のスケジューリング対象から除外されます。
ただし，本サービス・コールを発行した際，自タスクの起動要求がキューイングされていた（起動要求カウンタ＞0）場合には，自タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        .............
        .............

        ext_tsk ( );                    /* タスクの終了 */
}
```

備考 1　本サービス・コールでは，自タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

備考 2　タスク内で return 命令が発行された場合，本サービス・コールと同等の処理が実行されます。

3.6.2　他タスクの強制終了

タスクの強制終了は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ter_tsk
  パラメータ *tskid* で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより，対象タスクは，RI600V4 のスケジューリング対象から除外されます。
  ただし，本サービス・コールを発行した際，対象タスクの起動要求がキューイングされていた（起動要求カウンタ＞0）場合には，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        ............
        ............

        ter_tsk ( tskid );              /* タスクの強制終了 */

        ............
        ............
}
```

備考　本サービス・コールでは，対象タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

3.7 タスク優先度の変更

タスク優先度の変更は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

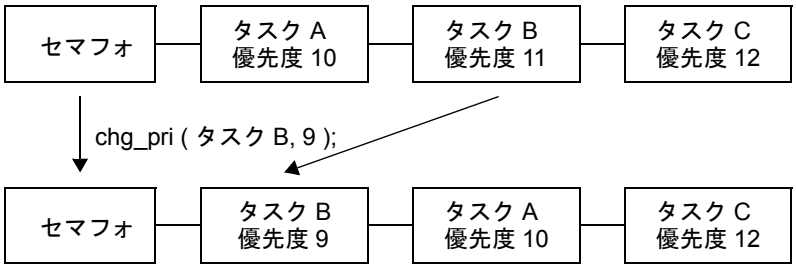
- chg_pri，ichg_pri

tskid で指定されたタスクのベース優先度を *tskpri* で指定された値に変更します。
変更されたタスクのベース優先度は，タスクが終了，または本サービス・コールを呼び出すまで有効です。次回のタスク起動時，タスクのベース優先度はタスク生成時に指定した初期タスク優先度になります。
本サービス・コールは，対象タスクの現在優先度も，*tskpri* で示された値に変更します。ただし，対象タスクがミューテックスをロックしている場合は，現在優先度は変更しません。
対象タスクがミューテックスをロックしているかロックを待っている場合で，*tskpri* がそれらのミューテックスのいずれかの上限優先度よりも高い場合には，戻り値として E_ILUSE を返します。
現在優先度が変更された場合，以下の状態変化が生じます。

1） 対象タスクが RUNNING 状態または READY 状態の場合
本サービス・コールは，対象タスクを *tskpri* で指定された優先度に応じたレディ・キューの最後尾につなぎかえます。

2） 対象タスクが TA_TPRI 属性または TA_CEILING 属性のオブジェクトの待ちキューにキューイングされている場合
本サービス・コールは，*tskpri* で指定された優先度にしたがって対象タスクを待ちキューにつなぎかえます。待ちキューに *tskpri* で指定された現在優先度のタスクが複数ある場合は，対象タスクをそれらの中の最後尾につなぎかえます。

例 セマフォの待ちキューに 3 つのタスク（タスク A：優先度 10，タスク B：優先度 11，タスク C：優先度 12）が優先度順でキューイングされているとき，タスク B の優先度を 11 から 9 に変更した場合，待ちキューの順序は，以下のように変更されます。



以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        PRI     tskpri = 9;             /* 変数の宣言，初期化 */

        .............
        .............

        chg_pri ( tskid, tskpri );      /* タスク優先度の変更 */

        .............
        .............
}
```

備考 現在優先度とベース優先度については，「6.2.2 現在優先度とベース優先度」を参照してください。

3.8　タスク優先度の参照

タスク優先度の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_pri，iget_pri
  パラメータ *tskid* で指定されたタスクの現在優先度をパラメータ *p_tskpri* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        PRI     p_tskpri;               /* 変数の宣言 */

        ............
        ............

        get_pri ( tskid, &p_tskpri );   /* タスク優先度の参照 */

        ............
        ............
}
```

備考　　現在優先度とベース優先度については，「6.2.2　現在優先度とベース優先度」を参照してください。

3.9　タスク状態の参照

3.9.1　タスク詳細情報の参照

タスク詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_tsk，iref_tsk
  パラメータ *tskid* で指定されたタスクのタスク詳細情報（現在状態，現在優先度など）をパラメータ *pk_rtsk* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        T_RTSK  pk_rtsk;                /* データ構造体の宣言 */
        STAT    tskstat;                /* 変数の宣言 */
        PRI     tskpri;                 /* 変数の宣言 */
        PRI     tskbpri;                /* 変数の宣言 */
        STAT    tskwait;                /* 変数の宣言 */
        ID      wobjid;                 /* 変数の宣言 */
        TMO     lefttmo;                /* 変数の宣言 */
        UINT    actcnt;                 /* 変数の宣言 */
        UINT    wupcnt;                 /* 変数の宣言 */
        UINT    suscnt;                 /* 変数の宣言 */

        .............
        .............

        ref_tsk ( tskid, &pk_rtsk );    /* タスク詳細情報の参照 */

        tskstat = pk_rtsk.tskstat;      /* 現在状態の獲得 */
        tskpri = pk_rtsk.tskpri;        /* 現在優先度の獲得 */
        tskbpri = pk_rtsk.tskbpri;      /* ベース優先度の獲得 */
        tskwait = pk_rtsk.tskwait;      /* 待ち要因の獲得 */
        wobjid = pk_rtsk.wobjid;        /* 待ちオブジェクトの ID の獲得 */
        lefttmo = pk_rtsk.lefttmo;      /* タイムアウトするまでの時間の獲得 */
        actcnt = pk_rtsk.actcnt;        /* 起動要求数の獲得 */
        wupcnt = pk_rtsk.wupcnt;        /* 起床要求数の獲得 */
        suscnt = pk_rtsk.suscnt;        /* サスペンド要求数の獲得 */

        .............
        .............
}
```

備考　タスク詳細情報 T_RTSK についての詳細は，「【 タスク詳細情報 T_RTSK の構造 】」を参照してください。

3.9.2　タスク基本情報の参照

タスク基本情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_tst，iref_tst
  パラメータ *tskid* で指定されたタスクのタスク基本情報（現在状態，待ち要因）をパラメータ *pk_rtst* で指定された領域に格納します。
  タスク情報のうち，現在状態，待ち要因のみを参照したい場合に使用します。
  取得する情報が少ないので ref_tsk，iref_tsk より高速に応答します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        T_RTST  pk_rtst;                /* データ構造体の宣言 */
        STAT    tskstat;                /* 変数の宣言 */
        STAT    tskwait;                /* 変数の宣言 */

        .............
        .............

        ref_tst ( tskid, &pk_rtst );    /* タスク基本情報の参照 */

        tskstat = pk_rtst.tskstat;      /* 現在状態の獲得 */
        tskwait = pk_rtst.tskwait;      /* 待ち要因の獲得 */

        .............
        .............
}
```

備考　　タスク基本情報 T_RTST についての詳細は，「【 タスク基本情報 T_RTST の構造 】」を参照してください。

第 4 章　タスク付属同期機能

本章では，RI600V4 が提供しているタスク付属同期機能について解説しています。

4.1　概　　要

RI600V4 におけるタスク付属同期機能では，タスクに従属した同期機能を提供しています。

4.2　起床待ち状態への移行

4.2.1　永久待ち

起床待ち状態への移行（永久待ち）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- slp_tsk
  自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。
  ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタ＞0）場合には，状態操作処理は行わず，起床要求カウンタから 1 を減算します。
  なお，起床待ち状態の解除は，以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により，起床要求が発行された。	E_OK
iwup_tsk の発行により，起床要求が発行された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */

        .............
        .............

        ercd = slp_tsk ( );             /* 起床待ち状態への移行（永久待ち）*/

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
```

```
                .............
        } else if ( ercd == E_RLWAI ) {
                .............               /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

4.2.2 タイムアウト付き

起床待ち状態への移行（タイムアウト付き）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- tslp_tsk
  自タスクを RUNNING 状態からタイムアウト付きの WAITING 状態（起床待ち状態）へと遷移させます。
  ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタ＞0）場合には，状態操作処理は行わず，起床要求カウンタから 1 を減算します。
  なお，起床待ち状態の解除は，以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により，起床要求が発行された。	E_OK
iwup_tsk の発行により，起床要求が発行された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;               /* 変数の宣言 */
        TMO     tmout = 3600;       /* 変数の宣言，初期化 */

        .............
        .............

        ercd = tslp_tsk ( tmout );  /* 起床待ち状態への移行（タイムアウト付き）*/

        if ( ercd == E_OK ) {
                .............       /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............       /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............       /* タイムアウト処理 */
                .............
        }

        .............
        .............
}
```

備考　待ち時間 *tmout* に TMO_FEVR が指定された際には “slp_tsk と同等の処理 ” を実行します。

4.3　タスクの起床

タスクの起床は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wup_tsk，iwup_tsk
  パラメータ *tskid* で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。
  ただし，本サービス・コールを発行した際，対象タスクが起床待ち状態以外の場合には，状態操作処理は行わず，起床要求カウンタに 1 を加算します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        ............
        ............

        wup_tsk ( tskid );              /* タスクの起床 */

        ............
        ............
}
```

備考　RI600V4 が管理する起床要求カウンタは，8 ビット幅で構成されています。このため，本サービス・コールでは，起床要求数が 255 回を超える場合には，起床要求のキューイング（起床要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

4.4　起床要求の解除

起床要求の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- can_wup，ican_wup
  パラメータ *tskid* で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0 を設定）します。
  なお，本サービス・コールは戻り値として解除した起床要求数を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER_UINT ercd;                   /* 変数の宣言 */
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        ............
        ............

        ercd = can_wup ( tskid );       /* 起床要求の解除 */

        if ( ercd >= 0 ) {
                ............            /* 正常終了処理 */
                ............
        }

        ............
        ............
}
```

4.5　WAITING 状態の強制解除

WAITING 状態の強制解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_wai，irel_wai
  パラメータ *tskid* で指定されたタスクの WAITING 状態を強制的に解除します。これにより，対象タスクは待ちキューから外れ，WAITING 状態から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，本サービス・コールの発行により WAITING 状態を解除されたタスクには，WAITING 状態へと遷移するきっかけとなったサービス・コール（slp_tsk，wai_sem など）の戻り値として E_RLWAI を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        rel_wai ( tskid );              /*WAITING 状態の強制解除 */

        .............
        .............
}
```

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが WAITING 状態または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

備考 2　本サービス・コールでは，SUSPENDED 状態の解除は行われません。

4.6　SUSPENDED 状態への移行

SUSPENDED 状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sus_tsk，isus_tsk
  パラメータ *tskid* で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ，READY 状態から SUSPENDED 状態へ，または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。
  ただし，本サービス・コールを発行した際，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態へと遷移していた場合には，戻り値として E_QOVR を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        sus_tsk ( tskid );              /*SUSPENDED 状態への移行 */

        .............
        .............
}
```

4.7 SUSPENDED 状態の解除

4.7.1 SUSPENDED 状態の解除

SUSPENDED 状態の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rsm_tsk，irsm_tsk
  パラメータ *tskid* で指定されたタスクを SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        ............
        ............

        rsm_tsk ( tskid );              /*SUSPENDED 状態の解除 */

        ............
        ............
}
```

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

備考 2　RI600V4 では，サスペンド要求のキューイング機能はサポートしていません。サスペンド要求のキューイングも含めて SUSPEND 要求を解除する frsm_tsk，ifrsm_tsk もサポートしていますが，これらの振る舞いは rsm_tsk，irsm_tsk と同じです。

4.7.2　SUSPENDED 状態の強制解除

SUSPENDED 状態の強制解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- frsm_tsk，ifrsm_tsk
  パラメータ *tskid* で指定されたタスクを SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        frsm_tsk ( tskid );             /*SUSPENDED 状態の強制解除 */

        .............
        .............
}
```

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

備考 2　RI600V4 では，サスペンド要求のネストはできません。このため，本サービス・コールは rsm_tsk，irsm_tsk とまったく同じ処理を行います。

4.8　時間経過待ち状態への移行

時間経過待ち状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dly_tsk
  自タスクをパラメータ *dlytim* で指定された遅延時間が経過するまでの間，RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
  なお，時間経過待ち状態の解除は，以下の場合に行われます。

時間経過待ち状態の解除操作	戻り値
パラメータ *dlytim* で指定された遅延時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        RELTIM  dlytim = 3600;          /* 変数の宣言，初期化 */

        .............
        .............

        ercd = dly_tsk ( dlytim );      /* 時間経過待ち状態への移行 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考　　*dlytim* に 0 を指定すると，次回の基本クロック割り込み発生までが遅延時間となります。

4.9　タイムアウト付き起床待ちと時間経過待ちの違い

タイムアウト付き起床待ちと時間経過待ちには，以下にのような違いがあります。

表４－１　タイムアウト付き起床待ちと時間経過待ちの違い

	タイムアウト付き起床待ち	時間経過待ち
状態遷移するサービス・コール	tslp_tsk	dly_tsk
時間経過時の戻り値	E_TMOUT	E_OK
wup_tsk，iwup_tsk が発行された際の動作	起床	起床要求をキューイング（時間経過待ちの解除は行われない）

第 5 章　同期通信機能

本章では，RI600V4 が提供している同期通信機能について解説しています。

5.1　概　　要

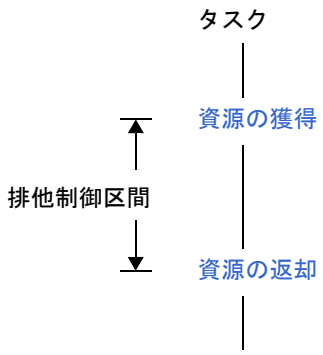
RI600V4 における同期通信機能では，タスク間の排他制御，同期，通信を実現する手段としてセマフォ，イベントフラグ，データ・キュー，メールボックスを提供しています。

5.2　セマフォ

マルチタスク処理では，並行に動作するタスクが限られた数の資源（A/D コンバータ，コプロセッサ，ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで，RI600V4 では，このような資源使用の競合を防ぐ機能として“非負数の計数型セマフォ”を提供しています。

以下に，セマフォを利用した場合の処理の流れを示します。

図５－１　処理の流れ（セマフォ）



5.2.1　セマフォの生成

RI600V4 では，セマフォの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

セマフォの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“semaphore[]”を使用してセマフォを定義することをいいます。

静的 API“semaphore[]”の詳細は，「19.8　セマフォ情報（semaphore[]）」を参照してください。

5.2.2 資源の獲得

資源の獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wai_sem（待つ）
- pol_sem，ipol_sem（ポーリング）
- twai_sem（タイムアウト付きで待つ）

- wai_sem（待つ）
  パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。
  ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（資源獲得待ち状態）へと遷移させます。
  なお，資源獲得待ち状態の解除は，以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により，対象セマフォに資源が返却された。	E_OK
isig_sem の発行により，対象セマフォに資源が返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = wai_sem ( semid );       /* 資源の獲得 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */

                sig_sem ( semid );      /* 資源の返却 *
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

- pol_sem，ipol_sem（ポーリング）
  パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。
  ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = pol_sem ( semid );       /* 資源の獲得 */

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */

                sig_sem ( semid );      /* 資源の返却 *
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

- twai_sem（タイムアウト付きで待つ）
  パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。
  ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（資源獲得待ち状態）へと遷移させます。
  なお，資源獲得待ち状態の解除は，以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により，対象セマフォに資源が返却された。	E_OK
isig_sem の発行により，対象セマフォに資源が返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............

                                        /* 資源の獲得 */
        ercd = twai_sem ( semid, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */

                sig_sem ( semid );      /* 資源の返却 *
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

        .............
        .............
}
```

備考 1　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には “wai_sem と同等の処理 ” を，TMO_POL が指定された際には “pol_sem と同等の処理 ” を実行します。

5.2.3 資源の返却

資源の返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sig_sem，isig_sem
  パラメータ *semid* で指定されたセマフォに資源を返却（セマフォ・カウンタに 1 を加算）します。
  ただし，本サービス・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（資源獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = wai_sem ( semid );       /* 資源の獲得 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                sig_sem ( semid );      /* 資源の返却 *
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考　RI600V4 では，セマフォの資源数として取り得る最大値（最大資源数）をコンフィギュレーション時に定義させています。このため，本サービス・コールでは，資源数が最大資源数を超える場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

5.2.4 セマフォ詳細情報の参照

セマフォ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_sem，iref_sem
  パラメータ *semid* で指定されたセマフォのセマフォ詳細情報（待ちタスクの有無，現在資源数）をパラメータ *pk_rsem* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      semid = 1;              /* 変数の宣言，初期化 */
        T_RSEM  pk_rsem;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        UINT    semcnt;                 /* 変数の宣言 */

        .............
        .............

        ref_sem ( semid, &pk_rsem );    /* セマフォ詳細情報の参照 */

        wtskid = pk_rsem.wtskid;        /* 待ちタスクの有無の獲得 */
        semcnt = pk_rsem.semcnt;        /* 現在資源数の獲得 */

        .............
        .............
}
```

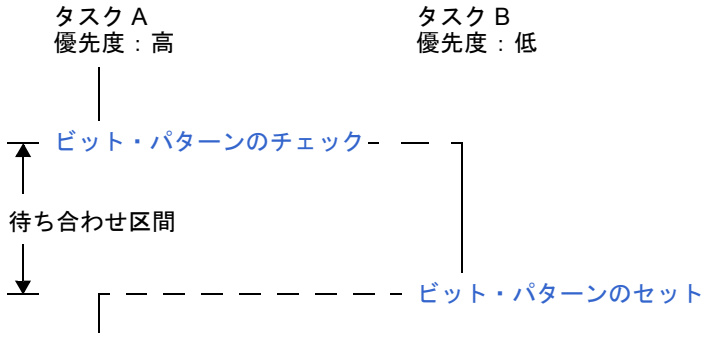
備考　セマフォ詳細情報 T_RSEM についての詳細は，「【 セマフォ詳細情報 T_RSEM の構造 】」を参照してください。

5.3　イベントフラグ

マルチタスク処理では，あるタスクの処理結果が出るまでの間，他タスクが処理の実行を待つといったタスク間の待ち合わせ機能（事象の発生有無を判断できる機能）が必要となります。そこで，RI600V4 では，このようなタスク間の待ち合わせ機能として “32 ビット幅のイベントフラグ ” を提供しています。

以下に，イベントフラグを利用した場合の処理の流れを示します。

図５－２　処理の流れ（イベントフラグ）



5.3.1　イベントフラグの生成

RI600V4 では，イベントフラグの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

イベントフラグの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“flag[]” を使用してイベントフラグを定義することをいいます。

静的 API“flag[]” の詳細は，「19.9　イベントフラグ情報（flag[]）」を参照してください。

5.3.2 ビット・パターンのセット

ビット・パターンのセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- set_flg，iset_flg

  パラメータ *flgid* で指定されたイベントフラグのビット・パターンとパラメータ *setptn* で指定されたビット・パターンの論理和 OR をとり，その結果を対象イベントフラグにセットします。
  そして，待ちキューの順に待ちキューにつながれているタスクの待ち解除条件を満たすかどうかを調べます。待ち解除条件を満たせば，該当タスクを待ちキューから外し，WAITING 状態（イベントフラグ待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。このとき，対象のイベントフラグ属性に TA_CLR 属性が指定されている場合には，イベントフラグのビット・パターンを 0 クリアし，処理を終了します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  setptn = 0x00000001UL;  /* 変数の宣言，初期化 */

        ............
        ............

        set_flg ( flgid, setptn );      /* ビット・パターンのセット */

        ............
        ............
}
```

5.3.3 ビット・パターンのクリア

ビット・パターンのクリアは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- clr_flg，iclr_flg
  パラメータ *flgid* で指定されたイベントフラグのビット・パターンとパラメータ *clrptn* で指定されたビット・パターンの論理積 AND をとり，その結果を対象イベントフラグに設定します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  clrptn = 0xFFFFFFFEUL;  /* 変数の宣言，初期化 */

        .............
        .............

        clr_flg ( flgid, clrptn );      /* ビット・パターンのクリア */

        .............
        .............
}
```

5.3.4 ビット・パターンのチェック

ビット・パターンのチェックは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wai_flg（待つ）
- pol_flg，ipol_flg（ポーリング）
- twai_flg（タイムアウト付きで待つ）

- wai_flg（待つ）

パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移させます。
なお，イベントフラグ待ち状態の解除は，以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */

        ............
        ............

                                        /* ビット・パターンのチェック */
        ercd = wai_flg ( flgid, waiptn, wfmode, &p_flgptn );

        if ( ercd == E_OK ) {
                ............            /* 正常終了処理 */
                ............
        } else if ( ercd == E_RLWAI ) {
                ............            /* 強制終了処理 */
                ............
        }

        ............
        ............
}
```

備考 1　RI600V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。
ただし，TA_CLR 属性が指定されていない場合は，優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは，μITRON4.0 仕様の範囲外です。

備考 3　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

- pol_flg，ipol_flg（ポーリング）
  パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
  なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，戻り値として E_TMOUT を返します。
  以下に，要求条件 *wfmode* の指定形式を示します。

  - *wfmode* = TWF_ANDW
    waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
  - *wfmode* = TWF_ORW
    waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */

        ............
        ............

                                        /* ビット・パターンのチェック */
        ercd = pol_flg ( flgid, waiptn, wfmode, &p_flgptn );

        if ( ercd == E_OK ) {
                ............            /* ポーリング成功処理 */
                ............
        } else if ( ercd == E_TMOUT ) {
                ............            /* ポーリング失敗処理 */
                ............
        }

        ............
        ............
}
```

備考 1　RI600V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

- twai_flg（タイムアウト付きで待つ）
  パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
  なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（イベントフラグ待ち状態）へと遷移させます。
  なお，イベントフラグ待ち状態の解除は，以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        ............
        ............

                                        /* ビット・パターンのチェック */
        ercd = twai_flg ( flgid, waiptn, wfmode, &p_flgptn, tmout );

        if ( ercd == E_OK ) {
                ............            /* 正常終了処理 */
```

```
                .............
        } else if ( ercd == E_RLWAI ) {
                .............                   /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............                   /* タイムアウト処理 */
                .............
        }

        .............
        .............
}
```

備考 1 RI600V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性： キューイング可能なタスクは，1 個
TA_WMUL 属性： キューイング可能なタスクは，複数

備考 2 自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。
ただし，TA_CLR 属性が指定されていない場合は，常に FIFO 順に行われます。この振る舞いは，μITRON4.0 仕様の範囲外です。

備考 3 対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

備考 4 待ち時間 *tmout* に TMO_FEVR が指定された際には “wai_flg と同等の処理 ” を，TMO_POL が指定された際には “pol_flg と同等の処理 ” を実行します。

5.3.5 イベントフラグ詳細情報の参照

イベントフラグ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_flg，iref_flg
  パラメータ *flgid* で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無，現在ビット・パターン）をパラメータ *pk_rflg* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        T_RFLG  pk_rflg;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        FLGPTN  flgptn;                 /* 変数の宣言 */

        .............
        .............

        ref_flg ( flgid, &pk_rflg );    /* イベントフラグ詳細情報の参照 */

        wtskid = pk_rflg.wtskid;        /* 待ちタスクの有無の獲得 */
        flgptn = pk_rflg.flgptn;        /* 現在ビット・パターンの獲得 */

        .............
        .............
}
```

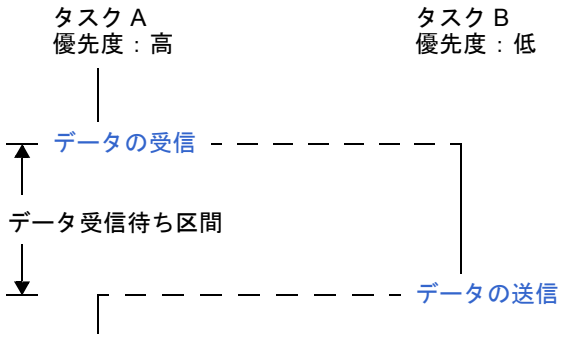
備考　イベントフラグ詳細情報 T_RFLG についての詳細は，「【 イベントフラグ詳細情報 T_RFLG の構造 】」を参照してください。

5.4 データ・キュー

マルチタスク処理では，あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（データの受け渡し機能）が必要となります。そこで，RI600V4 では，規定されたデータ・サイズの通信機能として“ データ・キュー” を提供しています。

以下に，データ・キューを利用した場合の処理の流れを示します。

図 5 － 3 処理の流れ（データ・キュー）



備考 1 回のデータ送信／受信処理で送信／受信するデータのサイズは，4 バイトです。

5.4.1 データ・キューの生成

RI600V4 では，データ・キューの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

データ・キューの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“dataqueue[]” を使用してデータ・キューを定義することをいいます。

静的 API“dataqueue[]” の詳細は，「19.10 データ・キュー情報（dataqueue[]）」を参照してください。

5.4.2 データの送信

データの送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- snd_dtq（待つ）
- psnd_dtq，ipsnd_dtq（ポーリング）
- tsnd_dtq（タイムアウト付きで待つ）

- snd_dtq（待つ）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。
  - 受信待ちキューにタスクがキューイングされている場合
    パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
    パラメータ *data* で指定されたデータをデータ・キューに格納します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ送信待ち状態）へと遷移させます。
    なお，データ送信待ち状態の解除は，以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
rcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
prcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
iprcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
trcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_dtq の発行により，対象データ・キューがリセットされた。	EV_RST

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

        .............
        .............

        ercd = snd_dtq ( dtqid, data ); /* データの送信 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

- psnd_dtq，ipsnd_dtq（ポーリング）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。
  - 受信待ちキューにタスクがキューイングされている場合
    パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
    パラメータ *data* で指定されたデータをデータ・キューに格納します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    戻り値として E_TMOUT を返します。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

        .............
        .............

        ercd = psnd_dtq ( dtqid, data ); /* データの送信 */

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

備考　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

- tsnd_dtq（タイムアウト付きで待つ）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。

  - 受信待ちキューにタスクがキューイングされている場合
    パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
    パラメータ *data* で指定されたデータをデータ・キューに格納します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ送信待ち状態）へと遷移させます。
    なお，データ送信待ち状態の解除は，以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
rcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
prcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
iprcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
trcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_dtq の発行により，対象データ・キューがリセットされた。	EV_RST
tmout で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............

                                        /* データの送信 */
        ercd = tsnd_dtq ( dtqid, data, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }
        .............
        .............
}
```

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “snd_dtq と同等の処理 ” を，TMO_POL が指定された際には “psnd_dtq と同等の処理 ” を実行します。

5.4.3 データの強制送信

データの強制送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- fsnd_dtq，ifsnd_dtq
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。
  - 受信待ちキューにタスクがキューイングされている場合
    パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされていない場合
    パラメータ *data* で指定されたデータをデータ・キューに格納します。
    データ・キューに空きがない場合は，その前に最古のデータを削除します。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

        .............
        .............

        fsnd_dtq ( dtqid, data );       /* データの強制送信 */

        .............
        .............
}
```

備考　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

5.4.4 データの受信

データの受信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rcv_dtq（待つ）
- prcv_dtq，iprcv_dtq（ポーリング）
- trcv_dtq（タイムアウト付きで待つ）

- rcv_dtq（待つ）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。
  - データ・キューにデータが格納されている場合
    データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
    送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
    なお，データ受信待ち状態の解除は，以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
snd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
psnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ipsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
tsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
fsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ifsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */

        .............
        .............

                                        /* データの受信 */
        ercd = rcv_dtq ( dtqid, &p_data );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

- prcv_dtq，iprcv_dtq（ポーリング）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。
  - データ・キューにデータが格納されている場合
    データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
    送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    戻り値として E_TMOUT を返します。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */

        .............
        .............

                                        /* データの受信 */
        ercd = prcv_dtq ( dtqid, &p_data );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

- trcv_dtq（タイムアウト付きで待つ）
  パラメータ *dtqid* で指定されたデータ・キューの状況に応じて，以下の処理を行います。

  - データ・キューにデータが格納されている場合
    データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
    送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。
  - データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
    なお，データ受信待ち状態の解除は，以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
snd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
psnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ipsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
tsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
fsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ifsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............

                                        /* データの受信 */
        ercd = trcv_dtq ( dtqid, &p_data, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }
        .............
        .............
}
```

備考 1　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_dtq と同等の処理 ” を，TMO_POL が指定された際には “prcv_dtq と同等の処理 ” を実行します。

5.4.5 データ・キュー詳細情報の参照

データ・キュー詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_dtq，iref_dtq
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無，未受信データの総数）をパラメータ *pk_rdtq* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        T_RDTQ  pk_rdtq;                /* データ構造体の宣言 */
        ID      stskid;                 /* 変数の宣言 */
        ID      rtskid;                 /* 変数の宣言 */
        UINT    sdtqcnt;                /* 変数の宣言 */

        ............
        ............

        ref_dtq ( dtqid, &pk_rdtq );    /* データ・キュー詳細情報の参照 */

        stskid = pk_rdtq.stskid;        /* データ送信待ちタスクの有無の獲得 */
        rtskid = pk_rdtq.rtskid;        /* データ受信待ちタスクの有無の獲得 */
        sdtqcnt = pk_rdtq.sdtqcnt;      /* 未受信データの総数の獲得 */

        ............
        ............
}
```

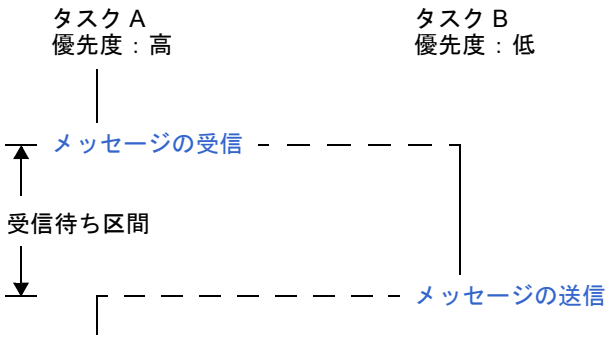
備考　データ・キュー詳細情報 T_RDTQ についての詳細は，「【 データ・キュー詳細情報 T_RDTQ の構造 】」を参照してください。

5.5　メールボックス

マルチタスク処理では，あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（メッセージの受け渡し機能）が必要となります。そこで，RI600V4 では，共有されているメモリ領域に書き込まれたメッセージの先頭アドレス受け渡し機能として“メールボックス”を提供しています。

以下に，メールボックスを利用した場合の処理の流れを示します。

図５－４　処理の流れ（メールボックス）



5.5.1　メッセージ

RI600V4 では，処理プログラム間でメールボックスを介してやり取りされる情報を“メッセージ”と呼んでいます。

なお，メッセージは，メールボックスを介することにより任意の処理プログラムに対して送信することができますが，RI600V4 における同期通信機能（メールボックス）では，メッセージの先頭アドレスを受信側処理プログラムに渡すだけであり，メッセージの内容が他領域にコピーされるわけではないので注意が必要です。

- メッセージの領域
  RI600V4 では，get_mpf，get_mpl を発行して確保したメモリ領域をメッセージ用に使用することを推奨しています。

- メッセージの基本型
  RI600V4 では，メッセージの内容，長さについては，利用するメールボックスの属性により，以下の規定を設けています。
  - TA_MFIFO 属性のメールボックスを利用する場合
    メッセージの先頭には T_MSG 構造体が必要です。この領域は，RI600V4 が使用します。ユーザのメッセージは，T_MSG 構造体以降に設定します。
    ユーザのメッセージの長さについては，メールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります。
    以下に，TA_MFIFO 属性用メッセージを記述する場合の基本型を示します。

    【 TA_MFIFO 属性用メッセージの構造 】

```
/* kernel.h で定義されている T_MSG 構造体 */
typedef struct  {
        VP    *msghead;        /*RI600V4 管理領域 */
} T_MSG;

/* ユーザが定義するメッセージ構造体 */
typedef struct user_msg {
        T_MSG    t_msg;        /* T_MSG 構造体 */
        B        data[8];      /* ユーザのメッセージ */
} USER_MSG;
```

  - TA_MPRI 属性のメールボックスを利用する場合
    メッセージの先頭には T_MSG_PRI 構造体が必要です。T_MSG_PRI.msgque の領域は，RI600V4 が使用します。T_MSG_PRI.msgpri には，メッセージの優先度を設定してください。ユーザのメッセージは，T_MSG_PRI 構造体以降に設定します。
    ユーザのメッセージの長さについては，メールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります
    以下に，TA_MPRI 属性用メッセージを記述する場合の基本型を示します。

    【 TA_MPRI 属性用メッセージの構造 】

```
/* kernel.h で定義されている T_MSG 構造体 */
typedef struct  {
        VP    *msghead;        /*RI600V4 管理領域 */
} T_MSG;

/* kernel.h で定義されている T_MSG_PRI 構造体 */{
typedef struct  {
        T_MSG    msgque;       /* メッセージヘッダ */
        PRI      msgpri;       /* メッセージ優先度 */
} T_MSG_PRI;

/* ユーザが定義するメッセージ構造体 */
typedef struct user_msg {
        T_MSG_PRI    t_msg;    /* T_MSG_PRI 構造体 */
        B        data[8];      /* ユーザのメッセージ */
} USER_MSG;
```

    備考 1　RI600V4 におけるメッセージの優先度は，その値が小さいほど，高い優先度であることを意味します。

    備考 2　メッセージの優先度として指定可能な値は，システム・コンフィギュレーション・ファイル作成時にメールボックス情報（mailbox[]）の最大メッセージ優先度（max_pri）で定義された値域に限定されます。

5.5.2 メールボックスの生成

RI600V4 では，メールボックスの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

メールボックスの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“mailbox[]” を使用してメールボックスを定義することをいいます。

静的 API“mailbox[]” の詳細は，「19.11 メールボックス情報（mailbox[]）」を参照してください。

5.5.3 メッセージの送信

メッセージの送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- snd_mbx，isnd_mbx

  パラメータ *mbxid* で指定されたメールボックスにパラメータ *pk_msg* で指定されたメッセージを送信します。

  ただし，本サービス・コールを発行した際，対象メールボックスの待ちキューにタスクがキューイングされていた場合には，メッセージの送信（メッセージのキューイング処理）は行わず，該当タスクにメッセージを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG_PRI       *pk_msg;        /* データ構造体の宣言 */

        ............
        ............

        ............                    /* メモリ領域（メッセージ用）を確保し ,/
        pk_msg = ...                    /* そのポインタを pk_msg に設定 */
        ............

        ............                    /* 本体（内容）の作成 */
        ............

        pk_msg->msgpri = 8;             /* メッセージ優先度を設定 */

                                        /* メッセージの送信 */
        snd_mbx ( mbxid, ( T_MSG * ) pk_msg );

        ............
        ............
}
```

備考 1 メッセージを対象メールボックスのメッセージ・キューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順またはメッセージ優先度順）に行われます。

備考 2 メッセージ T_MSG，T_MSG_PRI についての詳細は，「5.5.1 メッセージ」を参照してください。

5.5.4 メッセージの受信

メッセージの受信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rcv_mbx（待つ）
- prcv_mbx，iprcv_mbx（ポーリング）
- trcv_mbx タイムアウト付きで待つ）

- rcv_mbx（待つ）

パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *pk_msg;                /* データ構造体の宣言 */

        .............
        .............

                                        /* メッセージの受信 */
        ercd = rcv_mbx ( mbxid, &pk_msg );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }
        .............
}
```

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　メッセージ T_MSG，T_MSG_PRI についての詳細は，「5.5.1　メッセージ」を参照してください。

- prcv_mbx，iprcv_mbx（ポーリング）
  パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *pk_msg;                /* データ構造体の宣言 */

        .............
        .............

                                        /* メッセージの受信 */
        ercd = prcv_mbx ( mbxid, &pk_msg );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

備考　メッセージ T_MSG，T_MSG_PRI についての詳細は，「5.5.1　メッセージ」を参照してください。

- trcv_mbx タイムアウト付きで待つ）
  パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
  なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *pk_msg;                /* データ構造体の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        ............

                                        /* メッセージの受信 */
        ercd = trcv_mbx ( mbxid, &pk_msg, tmout );

        if ( ercd == E_OK ) {
                ............            /* 正常終了処理 */
        } else if ( ercd == E_RLWAI ) {
                ............            /* 強制終了処理 */
                ............
        } else if ( ercd == E_TMOUT ) {
                ............            /* タイムアウト処理 */
        }
        ............
}
```

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_mbx と同等の処理 ” を，TMO_POL が指定された際には “prcv_mbx と同等の処理 ” を実行します。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「5.5.1　メッセージ」を参照してください。

5.5.5 メールボックス詳細情報の参照

メールボックス詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mbx，iref_mbx
  パラメータ *mbxid* で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無，待ちメッセージの有無）をパラメータ *pk_rmbx* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_RMBX  pk_rmbx;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        T_MSG   *pk_msg;                /* データ構造体の宣言 */

        .............
        .............

        ref_mbx ( mbxid, &pk_rmbx );    /* メールボックス詳細情報の参照 */

        wtskid = pk_rmbx.wtskid;        /* 待ちタスクの有無の獲得 */
        pk_msg = pk_rmbx.pk_msg;        /* 待ちメッセージの有無の獲得 */

        .............
        .............
}
```

備考　メールボックス詳細情報 T_RMBX についての詳細は，「【 メールボックス詳細情報 T_RMBX の構造 】」を参照してください。

第 6 章　拡張同期通信機能

本章では，RI600V4 が提供している拡張同期通信機能について解説しています。

6.1　概　　要

RI600V4 における拡張同期通信機能では，タスク間の排他制御を実現する手段としてミューテックス，任意のサイズのメッセージをコピーして受け渡す手段としてメッセージ・バッファを提供しています。

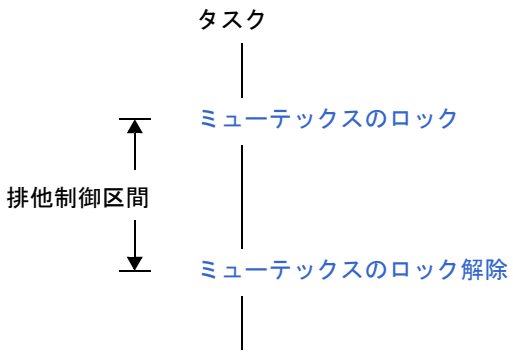
6.2　ミューテックス

マルチタスク処理では，並行に動作するタスクが限られた数の資源（A/D コンバータ，コプロセッサ，ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで，RI600V4 では，このような資源使用の競合を防ぐ機能として“ミューテックス”を提供しています。

以下に，ミューテックスを利用した場合の処理の流れを示します。

RI600V4 のミューテックスは優先度上限プロトコルをサポートしています。

図 6 － 1　処理の流れ（ミューテックス）



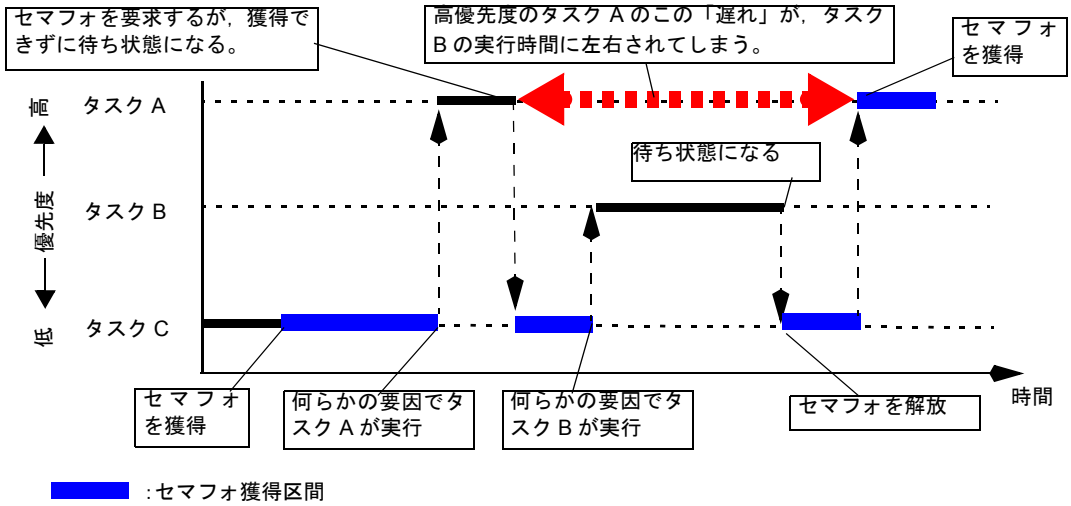
6.2.1 優先度逆転問題

セマフォを用いた排他制御では，優先度逆転という問題が発生するケースがあります。優先度逆転とは，資源を要求するタスクの実行が，資源を使用しない別のタスクによって遅延されてしまうという現象です。

この様子を図６－２に示します。この図では，タスク A とタスク C は同じ資源を使用し，タスク B はその資源を使用しない例になっています。タスク A は資源を使用するためにセマフォを獲得しようとしますが，すでにタスク C がセマフォを獲得しているので待ち状態になります。ところが，タスク C がセマフォを解放する前に，優先度がタスク C よりも高くてタスク A より低く，かつ資源とは関係のないタスク B が実行すると，タスク C によるセマフォの解放はタスク B の実行によって遅れ，その結果タスク A がセマフォを獲得するのも遅れます。タスク A の立場では，資源競合しておらず，かつ自分より優先度の低いタスク B が優先的に実行されてしまうことになります。

ミューテックスは，この問題を回避するために導入された機能です。

図６－２ 優先度逆転問題



6.2.2 現在優先度とベース優先度

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

ミューテックスをロックしていない時は，両者は常に同じです。

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

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

なお，get_pri，iget_pri を用いると，現在優先度を参照することができます。また，ref_tsk，iref_tsk を用いると現在優先度とベース優先度を参照することができます。

6.2.3　簡略化した優先度上限プロトコル

優先度上限プロトコルの本来の振る舞いは，タスクの現在優先度を，そのタスクがロックしているミューテックスの中で最高の上限優先度に制御することです。これは，ミューテックスのロック・アンロック時に，タスクの現在優先度を以下のように制御することで実現されます。

- ミューテックスのロック時に，タスクの現在優先度をそのタスクがロックしているミューテックスの中で最高の上限優先度に変更する。
- ミューテックスのアンロック時に，タスクの現在優先度をそのタスクが以降もロックを継続するミューテックスの中で最高の上限優先度に変更する。ロックしているミューテックスがなくなる場合は，現在優先度をベース優先度に戻す。

しかし，RI600V4 ではオーバーヘッドの低減を目的に「簡略化した優先度上限プロトコル」を採用しているため，上記の下線部の制御は行われません。

6.2.4　セマフォとの相違点

ミューテックスは最大資源数が 1 つのセマフォ（バイナリ・セマフォ）と似た動作をしますが，以下のような違いがあります。

- ミューテックスをロックしたタスクの現在優先度は，ミューテックスのロックを解除するまで，ミューテックスの上限優先度に引き上げられます。これによって，優先度逆転問題が回避されます。
  → セマフォでは優先度は変更されません。
- ミューテックスのロック解除（資源返却に相当）できるのはミューテックスをロックしたタスクのみです。
  → セマフォはどのタスク／ハンドラからでも資源の返却が可能です。
- ミューテックスをロックしているタスクを終了する（ext_tsk，ter_tsk）際に，自動的にロック解除処理が行われます。
  →セマフォは自動的に資源の返却を行わないので，資源を獲得したまま終了します。
- セマフォは複数の資源を管理できる（最大資源数を指定できる）がミューテックスの資源最大数に相当する値は 1 固定です。

6.2.5　ミューテックスの生成

RI600V4 では，ミューテックスの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。
ミューテックスの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“mutex[]” を使用してミューテックスを定義することをいいます。
静的 API“mutex[]” の詳細は，「19.12　ミューテックス情報（mutex[]）」を参照してください。

6.2.6　ミューテックスのロック

ミューテックスのロックは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- loc_mtx（待つ）
- ploc_mtx（ポーリング）
- tloc_mtx（タイムアウト付きで待つ）

- loc_mtx（待つ）

パラメータ *mtxid* で指定されたミューテックスをロックします。
ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。
なお，ミューテックス待ち状態の解除は，以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
unl_mtx の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

        .............

        ercd = loc_mtx ( mtxid );       /* ミューテックスのロック */

        if ( ercd == E_OK ) {
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
        }
        .............
}
```

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，優先度順に行われます。ただし，同じ優先度のタスクの中では FIFO 順に行われます。

備考 2　自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

- ploc_mtx（ポーリング）
  パラメータ *mtxid* で指定されたミューテックスをロックします。
  ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，戻り値として E_TMOUT を返します。
  ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = ploc_mtx ( mtxid );      /* ミューテックスのロック */

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

備考　自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

- tloc_mtx（タイムアウト付きで待つ）
  パラメータ *mtxid* で指定されたミューテックスをロックします。
  ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（ミューテックス待ち状態）へと遷移させます。
  なお，ミューテックス待ち状態の解除は，以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
unl_mtx の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
                                        /* ミューテックスのロック */
        ercd = tloc_mtx ( mtxid, tmout );

        if ( ercd == E_OK ) {
                .............           /* ロック状態 */

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
        }
        .............
}
```

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，優先度順に行われます。ただし，同じ優先度のタスクの中では FIFO 順に行われます。

備考 2　自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “loc_mtx と同等の処理 ” を，TMO_POL が指定された際には “ploc_mtx と同等の処理 ” を実行します。

6.2.7 ミューテックスのロック解除

ミューテックスのロック解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- unl_mtx

パラメータ *mtxid* で指定されたミューテックスのロック状態を解除します。その結果，自タスクがロックしているミューテックスがなくなった場合には，自タスクの現在優先度をベース優先度に変更します。
本サービス・コールを発行した際，対象ミューテックスの待ちキューにタスクがキューイングされていた場合には，ミューテックスのロック解除処理後，ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。このとき，該当タスクは，待ちキューから外れ，WAITING 状態（ミューテックス待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，該当タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，該当タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，該当タスクの現在優先度は変更しません。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = loc_mtx ( mtxid );       /* ミューテックスのロック */

        if ( ercd == E_OK ) {
                .............           /* ロック状態 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考 1　ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため，自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には，何も処理は行わず，戻り値として E_ILUSE を返します。

備考 2　タスクの終了時，そのタスクがロックしていたミューテックスは解除されます。

6.2.8 ミューテックス詳細情報の参照

ミューテックス詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mtx

パラメータ *mtxid* で指定されたミューテックスのミューテックス詳細情報（ロックの有無，待ちタスクの有無）をパラメータ *pk_rmtx* で指定された領域に格納します。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mtxid = 1;              /* 変数の宣言，初期化 */
        T_RMTX  pk_rmtx;                /* データ構造体の宣言 */
        ID      htskid;                 /* 変数の宣言 */
        ID      wtskid;                 /* 変数の宣言 */

        .............
        .............

        ref_mtx ( mtxid, &pk_rmtx );    /* ミューテックス詳細情報の参照 */

        htskid = pk_rmtx.htskid;        /* ロックの有無の獲得 */
        wtskid = pk_rmtx.wtskid;        /* 待ちタスクの有無の獲得 */

        .............
        .............
}
```

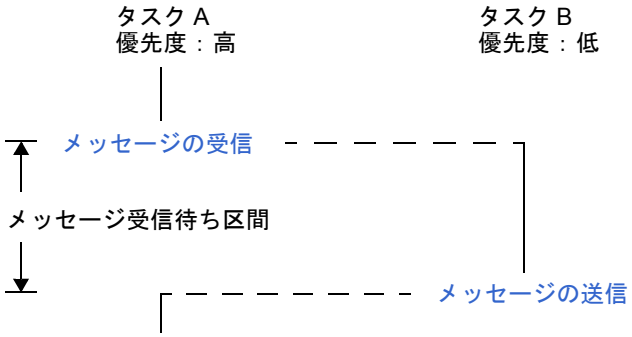
備考　ミューテックス詳細情報 T_RMTX についての詳細は，「【 ミューテックス詳細情報 T_RMTX の構造 】」を参照してください。

6.3　メッセージ・バッファ

マルチタスク処理では，あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（メッセージの受け渡し機能）が必要となります。そこで，RI600V4 では，任意のサイズのメッセージをコピーして受け渡す機能として“メッセージ・バッファ”を提供しています。

以下に，メッセージ・バッファを利用した場合の処理の流れを示します。

図６－３　処理の流れ（メッセージ・バッファ）



6.3.1　メッセージ・バッファの生成

RI600V4 では，メッセージ・バッファの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

メッセージ・バッファの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“message_buffer[]”を使用してミューテックスを定義することをいいます。

静的 API“message_buffer[]”の詳細は，「19.13　メッセージ・バッファ情報（message_buffer[]）」を参照してください。

6.3.2 メッセージの送信

メッセージの送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- snd_mbf（待つ）
- psnd_mbf，ipsnd_mbf（ポーリング）
- tsnd_mbf（タイムアウト付きで待つ）

- snd_mbf（待つ）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。
  - 受信待ちキューにタスクがキューイングされている場合
    msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
    msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

    減少サイズ＝ up4(*msgsz*) ＋ VTSZ_MBFTBL
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ送信待ち状態）へと遷移させます。
    なお，メッセージ送信待ち状態の解除は，以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
rcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
prcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
trcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tsnd_mbf のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mbf の発行により，対象メッセージ・バッファがリセットされた。	EV_RST

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[] = {1,2,3};        /* 変数の宣言，初期化 */
        UINT    msgsz = sizeof( msg );  /* 変数の宣言，初期化 */

        .............

                                        /* メッセージの送信 */
        ercd = snd_mbf ( mbfid, (VP)msg, msgsz );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
        }
        .............
}
```

備考 1　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

- psnd_mbf，ipsnd_mbf（ポーリング）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。
  - 受信待ちキューにタスクがキューイングされている場合
    msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
    msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

    減少サイズ＝ up4(*msgsz*) ＋ VTSZ_MBFTBL
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    戻り値として E_TMOUT を返します。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[] = {1,2,3};        /* 変数の宣言，初期化 */
        UINT    msgsz = sizeof( msg );  /* 変数の宣言，初期化 */

        ............

                                        /* メッセージの送信 */
        ercd = psnd_mbf ( mbfid, (VP)msg, msgsz );

        if ( ercd == E_OK ) {
                ............            /* ポーリング成功処理 */
        } else if ( ercd == E_TMOUT ) {
                ............            /* ポーリング失敗処理 */
        }
        ............
}
```

備考　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

- tsnd_mbf（タイムアウト付きで待つ）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

  - 受信待ちキューにタスクがキューイングされている場合
    msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
    msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

    減少サイズ＝ up4(*msgsz* ）＋ VTSZ_MBFTBL
  - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
    自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAIITING 状態（メッセージ送信待ち状態）へと遷移させます。
    なお，メッセージ送信待ち状態の解除は，以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
rcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
prcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
trcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tsnd_mbf のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mbf の発行により，対象メッセージ・バッファがリセットされた。	EV_RST
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[] = {1,2,3};        /* 変数の宣言，初期化 */
        UINT    msgsz = sizeof( msg );  /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............

                                        /* メッセージの送信 */
        ercd = tsnd_mbf ( mbfid, (VP)msg, msgsz, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
        }
        .............
}
```

備考 1　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “snd_mbf と同等の処理 ” を，TMO_POL が指定された際には “psnd_mbf と同等の処理 ” を実行します。

6.3.3　メッセージの受信

メッセージの受信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rcv_mbf（待つ）
- prcv_mbf（ポーリング）
- trcv_mbf（タイムアウト付きで待つ）

- rcv_mbf（待つ）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。
  - メッセージ・バッファにメッセージが格納されている場合
    メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

    増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL

    さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。
    - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

      減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL
  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。
  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
    なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
psnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
ipsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
tsnd_mbf の発行により，対象メッセージ・バッファバッファにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[16];                /* 変数の宣言（最大メッセージサイズ）*/

        .............

                                        /* メッセージの受信 */
        ercd = rcv_mbf ( mbfid, (VP)msg );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
        }
        .............
}
```

備考 1　最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域は，必ず最大メッセージ・サイズ以上としてください。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

- prcv_mbf（ポーリング）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。
  - メッセージ・バッファにメッセージが格納されている場合
    メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

    増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL

    さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。
    - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

      減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL
  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。
  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    戻り値として E_TMOUT を返します。

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[16];                /* 変数の宣言（最大メッセージサイズ）*/

        .............

                                        /* メッセージの受信 */
        ercd = prcv_mbf ( mbfid, (VP)msg );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
        }
        .............
}
```

備考　最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域は，必ず最大メッセージ・サイズ以上としてください。

- trcv_mbf（タイムアウト付きで待つ）
  パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

  - メッセージ・バッファにメッセージが格納されている場合
    メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

    増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL

    さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。

    - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

      減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL

  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
    送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
    なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。

  - メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
    自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
    なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
psnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
ipsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
tsnd_mbf の発行により，対象メッセージ・バッファバッファにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        B       msg[16];                /* 変数の宣言（最大メッセージサイズ）*/
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        ............

                                        /* メッセージの受信（タイムアウト付き）*/
        ercd = trcv_mbf ( mbfid, (VP)msg, tmout );

        if ( ercd == E_OK ) {
                ............            /* 正常終了処理 */
        } else if ( ercd == E_RLWAI ) {
                ............            /* 強制終了処理 */
        } else if ( ercd == E_TMOUT ) {
                ............            /* タイムアウト処理 */
        }
        ............
}
```

備考 1　最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域は，必ず最大メッセージ・サイズ以上としてください。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_mbf と同等の処理 ” を，TMO_POL が指定された際には “prcv_mbf と同等の処理 ” を実行します。

6.3.4 メッセージ・バッファ詳細情報の参照

メッセージ・バッファ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mbf，iref_mbf
  パラメータ *mbfid* で指定されたメッセージ・バッファの詳細情報（待ちタスクの有無，空き領域のサイズなど）をパラメータ *pk_rmbf* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mbfid = 1;              /* 変数の宣言，初期化 */
        T_RMBF  pk_rmbf;                /* データ構造体の宣言 */
        ID      stskid;                 /* 変数の宣言 */
        ID      rtskid;                 /* 変数の宣言 */
        UINT    smsgcnt;                /* 変数の宣言 */
        SIZE    fmbfsz;                 /* 変数の宣言 */

        ............
        ............

        ref_mbf ( mbfid, &pk_rmbf );    /* メッセージ・バッファ詳細情報の参照 */

        stskid = pk_rmbf.stskid;        /* メッセージ送信待ちタスクの有無の獲得 */
        rtskid = pk_rmbf.rtskid;        /* メッセージ受信待ちタスクの有無の獲得 */
        smsgcnt = pk_rmbf.smsgcnt;      /* 未受信メッセージ総数の獲得 */
        fmbfsz = pk_rmbf.fmbfsz;        /* 空き領域のサイズ  */

        ............
        ............
}
```

備考　メッセージ・バッファ詳細情報 T_RMBF についての詳細は，「【 メッセージ・バッファ詳細情報 T_RMBF の構造 】」を参照してください。

第 7 章　メモリ・プール管理機能

本章では，RI600V4 が提供しているメモリ・プール管理機能について解説しています。

7.1　概　　要

RI600V4 では, 動的なメモリ獲得機能として“固定長メモリ・プール”および“可変長メモリ・プール”を提供しています。

固定長メモリ・プールは，扱えるメモリ・サイズは固定ですが，獲得／解放のオーバーヘッドが小さいという特長があります。

一方，可変長メモリ・プールは任意のメモリサイズを扱うことができますが，獲得／解放のオーバーヘッドは固定長メモリ・プールより劣ります。また，メモリ・プール領域の断片化の問題が発生します。

7.2　固定長メモリ・プール

RI600V4 では，処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として “ 固定長メモリ・プール ” を提供しています。

なお，固定長メモリ・プールに対する動的なメモリ操作は，固定サイズの固定長メモリ・ブロックを単位として行われます。

7.2.1　固定長メモリ・プールの生成

RI600V4 では，固定長メモリ・プールの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

固定長メモリ・プールの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“memorypool[]” を使用して固定長メモリ・プールを定義することをいいます。

静的 API“memorypool[]” の詳細は，「19.14　固定長メモリ・プール情報（memorypool[]）」を参照してください。

7.2.2　固定長メモリ・ブロックの獲得

固定長メモリ・ブロックの獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

RI600V4 では，固定長メモリ・ブロックを獲得する際，メモリ・クリア処理を行っていません。したがって，獲得された固定長メモリ・ブロックの内容は不定となります。

- get_mpf（待つ）
- pget_mpf，ipget_mpf（ポーリング）
- tget_mpf（タイムアウト付きで待つ）

- get_mpf（待つ）

  パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpf の発行により，対象固定長メモリ・プールがリセットされた。	EV_RST

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

        .............
        .............

                                        /* 固定長メモリ・ブロックの獲得 */
        ercd = get_mpf ( mpfid, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　獲得したメモリ・ブロックの内容は不定です。

備考 3　メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

- pget_mpf，ipget_mpf（ポーリング）
  パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

        .............
        .............

                                        /* 固定長メモリ・ブロックの獲得 */
        ercd = pget_mpf ( mpfid, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

備考 1　獲得したメモリ・ブロックの内容は不定です。

備考 2　メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

- tget_mpf（タイムアウト付きで待つ）
  パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpf の発行により，対象固定長メモリ・プールがリセットされた。	EV_RST
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............

                                        /* 固定長メモリ・ブロックの獲得 */
        ercd = tget_mpf ( mpfid, &p_blk, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

        .............
        .............
}
```

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　獲得したメモリ・ブロックの内容は不定です。

備考 3　メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

備考 4　待ち時間 *tmout* に TMO_FEVR が指定された際には “get_mpf と同等の処理 ” を，TMO_POL が指定された際には “pget_mpf と同等の処理 ” を実行します。

7.2.3　固定長メモリ・ブロックの返却

固定長メモリ・ブロックの返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_mpf，irel_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールにパラメータ *blk* で指定された固定長メモリ・ブロックを返却します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却は行わず，該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      blk;                    /* 変数の宣言 */

        .............
        .............

        ercd = get_mpf ( mpfid, &blk ); /* 固定長メモリ・ブロックの獲得 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                rel_mpf ( mpfid, blk ); /* 固定長メモリ・ブロックの返却 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

7.2.4　固定長メモリ・プール詳細情報の参照

固定長メモリ・プール詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mpf，iref_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無，空き固定長メモリ・ブロックの総数など）をパラメータ *pk_rmpf* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        T_RMPF  pk_rmpf;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        UINT    fblkcnt;                /* 変数の宣言 */

        ............
        ............

        ref_mpf ( mpfid, &pk_rmpf );    /* 固定長メモリ・プール詳細情報の参照 */

        wtskid = pk_rmpf.wtskid;        /* 待ちタスクの有無の獲得 */
        fblkcnt = pk_rmpf.fblkcnt;      /* 空き固定長メモリ・ブロックの総数の獲得 */

        ............
        ............
}
```

備考　固定長メモリ・プール詳細情報 T_RMPF についての詳細は，「【 固定長メモリ・プール詳細情報 T_RMPF の構造 】」を参照してください。

7.3　可変長メモリ・プール

RI600V4 では，処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として“可変長メモリ・プール”を提供しています。

なお，可変長メモリ・プールに対する動的なメモリ操作は，任意サイズの可変長メモリ・ブロックを単位として行われます。

7.3.1　可変長メモリ・プールの生成

RI600V4 では，可変長メモリ・プールの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

可変長メモリ・プールの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“variable_memorypool[]”を使用して可変長メモリ・プールを定義することをいいます。

静的 API“variable_memorypool[]”の詳細は，「19.15　可変長メモリ・プール情報（variable_memorypool[]）」を参照してください。

7.3.2　可変長メモリ・ブロックのサイズ

RI600V4 の現在の実装では，実際に獲得されるメモリ・ブロックのサイズは，最大で 12 種類のバリエーションから選択されます。このバリエーションは，可変長メモリ・プール情報（variable_memorypool[]）の可変長メモリ・ブロック・サイズの上限（max_memsize）を元に，あらかじめ規定された 24 種類のサイズの中から選択されます。以下に，メモリ・ブロックサイズのバリエーションを示します。ただし，この振る舞いは将来変更される可能性があります。

表 7 － 1　メモリ・ブロック・サイズのバリエーション

項番	メモリ・ブロックのサイズ (16 進数)	例 1 : max_memsize=0x100 の場合	例 2 : max_memsize=0x20000 の場合
1	12 (0xC)	○	×
2	36 (0x24)	○	×
3	84 (0x54)	○	○
4	180 (0xB4)	○	○
5	372 (0x174)	×	○
6	756 (0x2F4)	×	○
7	1524 (0x5F4)	×	○
8	3060 (0xBF4)	×	○
9	6132 (0x17F4)	×	○
10	12276 (0x2FF4)	×	○
11	24564 (0x5FF4)	×	○
12	49140 (0xBFF4)	×	○
13	98292 (0x17FF4)	×	○
14	196596 (0x2FFF4)	×	○
15	393204 (0x5FFF4)	×	×
16	786420 (0xBFFF4)	×	×
17	1572852 (0x17FFF4)	×	×
18	3145716 (0x2FFFF4)	×	×
19	6291444 (0x5FFFF4)	×	×
20	12582900 (0xBFFFF4)	×	×
21	25165812 (0x17FFFF4)	×	×
22	50331636 (0x2FFFFF4)	×	×
23	100663284 (0x5FFFFF4)	×	×
24	201326580 (0xBFFFFF4)	×	×

7.3.3　可変長メモリ・ブロックの獲得

可変長メモリ・ブロックの獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

RI600V4 では，可変長メモリ・ブロックを獲得する際，メモリ・クリア処理を行っていません。したがって，獲得された可変長メモリ・ブロックの内容は不定となります。

- get_mpl（待つ）
- pget_mpl，ipget_mpl（ポーリング）
- tget_mpl（タイムアウト付きで待つ）

- get_mpl（待つ）

  パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tget_mpl のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpl の発行により，対象可変長メモリ・プールがリセットされた。	EV_RST

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

        .............
        .............

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = get_mpl ( mplid, blksz, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考 1　実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　獲得したメモリ・ブロックの内容は不定です。

備考 4　獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

- pget_mpl，ipget_mpl（ポーリング）
  パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

        .............
        .............

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = pget_mpl ( mplid, blksz, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

        .............
        .............
}
```

備考 1　実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

備考 2　獲得したメモリ・ブロックの内容は不定です。

備考 3　獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

- tget_mpl（タイムアウト付きで待つ）
  パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tget_mpl のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpl の発行により，対象可変長メモリ・プールがリセットされた。	EV_RST
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = tget_mpl ( mplid, blksz, &p_blk, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

        .............
        .............
}
```

備考 1　実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　獲得したメモリ・ブロックの内容は不定です。

備考 4　獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

備考 5　待ち時間 *tmout* に TMO_FEVR が指定された際には “get_mpl と同等の処理 ” を，TMO_POL が指定された際には “pget_mpl と同等の処理 ” を実行します。

7.3.4　可変長メモリ・ブロックの返却

可変長メモリ・ブロックの返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_mpl

  パラメータ *mplid* で指定された可変長メモリ・プールにパラメータ *blk* で指定された可変長メモリ・ブロックを返却します。
  可変長メモリ・ブロックを返却したあと，対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき，待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか，メモリが割り当てられなくなるまで繰り返します。これにより，メモリを獲得できたタスクは，待ちキューから外れ，WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      blk;                    /* 変数の宣言 */

        .............
        .............

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = get_mpl ( mplid, blksz, &blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                rel_mpl ( mplid, blk ); /* 可変長メモリ・ブロックの返却 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

        .............
        .............
}
```

備考　RI600V4 では，*blk* に関して簡易的なエラー検出しか行っていません。*blk* には，必ず正しい値を指定してください。*blk* がエラー検出されない不正な値の場合，以後の動作は保証されません。

7.3.5　可変長メモリ・プール詳細情報の参照

可変長メモリ・プール詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mpl，iref_mpl

  パラメータ *mplid* で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無，空き可変長メモリ・ブロックの合計サイズなど）をパラメータ *pk_rmpl* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        T_RMPL  pk_rmpl;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        SIZE    fmplsz;                 /* 変数の宣言 */
        UINT    fblksz;                 /* 変数の宣言 */

        ............
        ............

        ref_mpl ( mplid, &pk_rmpl );    /* 可変定長メモリ・プール詳細情報の参照 */

        wtskid = pk_rmpl.wtskid;        /* 待ちタスクの有無の獲得 */
        fmplsz = pk_rmpl.fmplsz;        /* 空き領域の合計サイズの獲得 */
        fblksz = pk_rmpl.fblksz;        /* 獲得可能なメモリ・ブロックの最大サイズの獲得 */

        ............
        ............
}
```

備考　可変長メモリ・プール詳細情報 T_RMPL についての詳細は，「【 可変長メモリ・プール詳細情報 T_RMPL の構造 】」を参照してください。

第 8 章　時間管理機能

本章では，RI600V4 が提供している時間管理機能について解説しています。

8.1　概　　要

RI600V4 における時間管理機能では，一定周期で発生する基本クロック用タイマ割り込みを利用してシステム時刻を操作／参照する機能のほかに，時間に依存した処理を実現する手段（タイマ・オペレーション機能：タスクの遅延，タイムアウト，周期ハンドラ，アラーム・ハンドラ，システム時刻）を提供しています。

8.2　システム時刻

システム時刻とは，RI600V4 が時間管理を行う際に使用する“時間（単位：ミリ秒）”です。

システム時刻は，カーネル初期化部（vsta_knl, ivsta_knl）において 0 に初期化されたのち，システム情報（system）の基本クロック割り込み周期の分母（tic_deno），および基本クロック割り込み周期の分子（tic_nume）で定義された基本クロック周期ごとに更新されます。

8.2.1　基本クロック用タイマ割り込み

RI600V4 では，時間管理機能を実現するために，一定周期で発生する割り込み（基本クロック用タイマ割り込み）を使用します。

基本クロック用タイマ割り込みが発生した際は，RI600V4 の時間に関連した処理（システム時刻の更新，タスクのタイムアウト／遅延，周期ハンドラの起動，アラーム・ハンドラの起動など）が実行されます。

基本クロック用タイマは，RX MCU 内蔵のコンペア・マッチ・タイマ（CMT）のチャネル 0 ～ 3 のいずれかを使用することを基本としています。どのチャネルを使用するかは，システム・コンフィギュレーション・ファイルの基本クロック割り込み情報（clock）の基本クロック用タイマ・チャネルの選択（timer）で指定します。

基本クロック用タイマ割り込みを発生するためのハードウエアの初期化は，cfg600 が出力する“ri_cmt.h”で実装されている“void _RI_init_cmt(void)”で行われます。ブート処理関数（PowerON_Reset_PC( )）は，_RI_init_cmt() を呼び出す必要があります。

8.2.2　基本クロック周期

RI600V4 のサービス・コールでは時間指定パラメータの単位は「ミリ秒」になっています。

基本クロック用タイマ割り込みの発生周期は 1 ミリ秒とするのが望ましいですが，ターゲット・システムの性質上（処理能力，必要とする時間分解能など）1 ミリ秒とすることが困難な場合があります。

この場合，システム・コンフィギュレーション・ファイルのシステム情報（system）の基本クロック割り込み周期の分母（tic_deno），および基本クロック割り込み周期の分子（tic_nume）で基本クロック用タイマ割り込みの発生周期を指定することができます。

基本クロック周期を指定すると 1 回の基本クロック用タイマ割り込みで基本クロック周期分の時間が経過したとして処理されます。

8.3　タイマ・オペレーション機能

RI600V4 におけるタイマ・オペレーション機能では，時間に依存した処理を実現する手段として “ タスクの遅延，タイムアウト，周期ハンドラ，アラーム・ハンドラ，システム時刻 ” を提供しています。

8.4　タスクの遅延

タスクの遅延は，一定の時間が経過するまでの間，自タスクを RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させ，一定の時間が経過した際には，該当タスクを WAITING 状態から READY 状態へと遷移させるものです。

なお，遅延起床は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dly_tsk

8.5　タイムアウト

タイムアウトとは，タスクから発行された要求条件が即時成立しなかった場合，一定の時間が経過するまでの間，該当タスクを RUNNING 状態から WAITING 状態（起床待ち状態，資源獲得待ち状態，イベントフラグ待ち状態など）へと遷移させ，一定の時間が経過した際には，要求条件の成立／不成立を問わず，該当タスクを WAITING 状態から READY 状態へと遷移させるものです。

なお，タイムアウトは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- tslp_tsk
- twai_sem
- twai_flg
- tsnd_dtq
- trcv_dtq
- trcv_mbx
- tloc_mtx
- tsnd_mbf
- trcv_mbf
- tget_mpf
- tget_mpl

8.6 周期ハンドラ

周期ハンドラは，一定の時間ごとに周期的に起動される周期処理専用ルーチンです。
なお，RI600V4 では，周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，一定の時間が経過した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，周期ハンドラに制御が移ります。

8.6.1 周期ハンドラの基本形

以下に，周期ハンドラを記述する場合の基本型を示します。*exinf* には，“周期ハンドラ情報（cyclic_hand[]）の拡張情報（exinf）渡されます。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void cychdr ( VP_INT exinf )
{
        ............
        ............

        return;                         /* 周期ハンドラの終了 */
}
```

備考　ハンドラ関数のプロトタイプ宣言は，cfg600 が kernel_id.h に出力します。

8.6.2 周期ハンドラ内での処理

- スタック
  周期ハンドラは，システム・スタックを使用します。

- サービス・コールの発行
  RI600V4 では，周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。
  周期ハンドラでは，“発行有効範囲”が“非タスク”のサービスコールを発行可能です。

  備考 RI600V4 では，周期ハンドラ内の処理を高速に終了させる目的から，周期ハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，周期ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“周期ハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。

- 処理開始時の PSW

表 8 － 1 周期ハンドラ処理開始時の PSW

ビット	値	備考
I	1	
IPL	基本クロック割り込み優先レベル（IPL）	処理開始時より下げてはなりません。
PM	0	スーパバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

8.6.3 周期ハンドラの生成

RI600V4 では，周期ハンドラの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

周期ハンドラの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“cyclic_hand[]” を使用して周期ハンドラを定義することをいいます。

静的 API“cyclic_hand[]” の詳細は，「19.16 周期ハンドラ情報（cyclic_hand[]）」を参照してください。

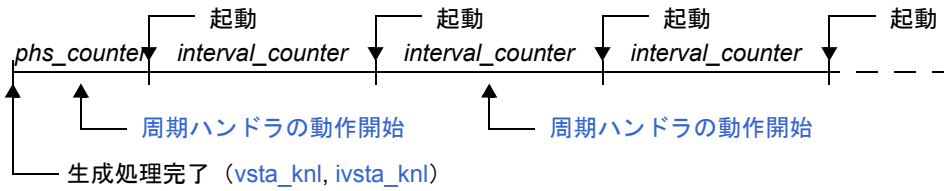
8.6.4 周期ハンドラの動作開始

周期ハンドラの動作開始は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sta_cyc，ista_cyc
  パラメータ *cycid* で指定された周期ハンドラの動作状態を停止状態（STP 状態）から動作状態（STA 状態）へと遷移させます。これにより，対象周期ハンドラは，RI600V4 の起動対象となります。
  なお，本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は，コンフィギュレーション時に対象周期ハンドラに対して TA_PHS 属性（phsatr）を指定しているか否かにより異なります。

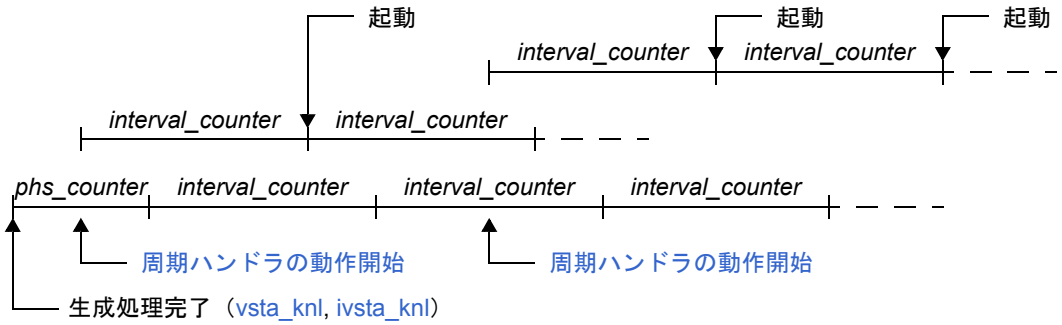
  - TA_PHS 属性を指定した場合
    起動位相（phs_counter），起動周期（interval_counter）にしたがって，起動タイミングが設定されます。
    ただし，対象周期ハンドラの動作状態が開始状態の場合には，本サービス・コールを発行しても何も処理は行わず，エラーとしても扱いません。
    図 8 － 1 に，周期ハンドラの起動タイミング・イメージを示します。

図 8 － 1 TA_PHS 属性：指定あり



  - TA_PHS 属性を指定しない場合
    本サービス・コールの発行時点を基点とし，起動周期（interval_counter）にしたがって起動タイミングが設定されます。
    なお，起動タイミング設定処理については，対象周期ハンドラの動作状態に関係なく実行されます。
    図 8 － 2 に，周期ハンドラの起動タイミング・イメージを示します。

図 8 － 2 TA_PHS 属性：指定なし



以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */

        ............
        ............

        sta_cyc ( cycid );      /* 周期ハンドラの動作開始 */

        ............
        ............
}
```

8.6.5 周期ハンドラの動作停止

周期ハンドラの動作停止は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- stp_cyc，istp_cyc
  パラメータ *cycid* で指定された周期ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_cyc または ista_cyc が発行されるまでの間，対象周期ハンドラは，RI600V4 の起動対象から除外されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */

        ............
        ............

        stp_cyc ( cycid );              /* 周期ハンドラの動作停止 */

        ............
        ............
}
```

備考　本サービス・コールでは，停止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対象周期ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

8.6.6 周期ハンドラ詳細情報の参照

周期ハンドラ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_cyc，iref_cyc
  パラメータ *cycid* で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態，残り時間）をパラメータ *pk_rcyc* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */
        T_RCYC  pk_rcyc;                /* データ構造体の宣言 */
        STAT    cycstat;                /* 変数の宣言 */
        RELTIM  lefttim;                /* 変数の宣言 */

        ............
        ............

        ref_cyc ( cycid, &pk_rcyc );    /* 周期ハンドラ詳細情報の参照 */

        cycstat = pk_rcyc.cycstat;      /* 現在状態の獲得 */
        lefttim = pk_rcyc.lefttim;      /* 残り時間の獲得 */

        ............
        ............
}
```

備考　周期ハンドラ詳細情報 T_RCYC についての詳細は，「【 周期ハンドラ詳細情報 T_RCYC の構造 】」を参照してください。

8.7 アラーム・ハンドラ

アラーム・ハンドラは，指定した時間が経過したときに起動されるルーチンです。
なお，RI600V4 では，アラーム・ハンドラを “ タスクとは独立したもの（非タスク）” として位置づけています。このため，指定した時間が経過した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，アラーム・ハンドラに制御が移ります。

8.7.1 アラーム・ハンドラの基本形

以下に，アラーム・ハンドラを記述する場合の基本型を示します。*exinf* には，アラーム・ハンドラ情報（alarm_hand[]）の拡張情報（exinf）が渡されます。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void almhdr ( VP_INT exinf )
{
        ............
        ............

        return;                         /* アラーム・ハンドラの終了 */
}
```

備考　ハンドラ関数のプロトタイプ宣言は，cfg600 が kernel_id.h に出力します。

8.7.2 アラーム・ハンドラ内での処理

- スタック
  アラーム・ハンドラは，システム・スタックを使用します。

- サービス・コールの発行
  RI600V4 では，アラーム・ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。
  アラーム・ハンドラでは，“発行有効範囲”が“非タスク”のサービスコールを発行可能です。

  備考 RI600V4 では，アラーム・ハンドラ内の処理を高速に終了させる目的から，アラーム・ハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，アラーム・ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“アラーム・ハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。

- 処理開始時の PSW

表 8 － 2 アラーム・ハンドラ処理開始時の PSW

ビット	値	備考
I	1	
IPL	基本クロック割り込み優先レベル（IPL）	処理開始時より下げてはなりません。
PM	0	スーパバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

8.7.3 アラーム・ハンドラの生成

RI600V4 では，アラーム・ハンドラの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

周期ハンドラの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“alarm_hand[]” を使用して周期ハンドラを定義することをいいます。

静的 API“alarm_hand[]” の詳細は，「19.17 アラーム・ハンドラ情報（alarm_hand[]）」を参照してください。

8.7.4 アラーム・ハンドラの動作開始

アラーム・ハンドラの動作開始は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sta_alm，ista_alm
  almid で指定されたアラーム・ハンドラの起動時刻を almtim ミリ秒後に設定し，動作状態（STA 状態）にします。これにより，対象アラーム・ハンドラは，RI600V4 の起動対象となります
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      almid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        sta_alm ( almid );      /* アラーム・ハンドラの動作開始 */

        .............
        .............
}
```

備考 1　*almtim* に 0 を指定すると，次回の基本クロック割り込み時にアラーム・ハンドラが起動されます。

備考 2　対象アラーム・ハンドラがすでに動作状態の場合でも，本サービス・コールは起動時刻を再設定します。以前の起動時刻の設定は無効となります。

8.7.5 アラーム・ハンドラの動作停止

アラーム・ハンドラの動作停止は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- stp_alm，istp_alm
  パラメータ *almid* で指定されたアラーム・ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_alm または ista_alm が発行されるまでの間，対象アラーム・ハンドラは，RI600V4 の起動対象から除外されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      almid = 1;              /* 変数の宣言，初期化 */

        ............
        ............

        stp_alm ( almid );              /* 周期ハンドラの動作停止 */

        ............
        ............
}
```

備考　本サービス・コールでは，停止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対象アラーム・ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

8.7.6 アラーム・ハンドラ詳細情報の参照

アラーム・ハンドラ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_alm，iref_alm
  パラメータ *almid* で指定されたアラーム・ハンドラの周期ハンドラ詳細情報（現在状態，残り時間）をパラメータ *pk_rcyc* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ID      almid = 1;              /* 変数の宣言，初期化 */
        T_RALM  pk_ralm;                /* データ構造体の宣言 */
        STAT    almstat;                /* 変数の宣言 */
        RELTIM  lefttim;                /* 変数の宣言 */

        .............
        .............

        ref_alm ( almid, &pk_ralm );    /* 周期ハンドラ詳細情報の参照 */

        almstat = pk_ralm.almstat;      /* 現在状態の獲得 */
        lefttim = pk_ralm.lefttim;      /* 残り時間の獲得 */

        .............
        .............
}
```

備考　アラーム・ハンドラ詳細情報 T_RALM についての詳細は，「【 アラーム・ハンドラ詳細情報 T_RALM の構造 】」を参照してください。

8.8 システム時刻

8.8.1 システム時刻の設定

システム時刻の設定は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。
なお，システム時刻を変更しても，それ以前に行われた時間管理要求（タスクのタイムアウト，dly_tsk によるタスクの遅延，周期ハンドラ，およびアラーム・ハンドラ）が発生する実時刻は変化しません。

- set_tim，iset_tim

RI600V4 のシステム時刻（単位：ミリ秒）をパラメータ *p_systim* で指定された時間に変更します。
以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        SYSTIM  p_systim;               /* データ構造体の宣言 */

        p_systim.ltime = 3600;          /* データ構造体の初期化 */
        p_systim.utime = 0;             /* データ構造体の初期化 */

        ............
        ............

        set_tim ( &p_systim );          /* システム時刻の設定 */

        ............
        ............
}
```

備考　システム時刻情報SYSTIMについての詳細は，「【 システム時刻情報 SYSTIM の構造 】」を参照してください。

8.8.2 システム時刻の参照

システム時刻の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_tim，iget_tim
  RI600V4 のシステム時刻（単位：ミリ秒）をパラメータ *p_systim* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        SYSTIM  p_systim;               /* データ構造体の宣言 */
        UW      ltime;                  /* 変数の宣言 */
        UH      utime;                  /* 変数の宣言 */

        ............
        ............

        get_tim ( &p_systim );          /* システム時刻の参照 */

        ltime = p_systim.ltime;         /* システム時刻（下位 32 ビット）の獲得 */
        utime = p_systim.utime;         /* システム時刻（上位 16 ビット）の獲得 */

        ............
        ............
}
```

備考　システム時刻情報SYSTIMについての詳細は,「【 システム時刻情報 SYSTIM の構造 】」を参照してください。

8.9　基本クロック用タイマの初期化

cfg600 は，基本クロック用タイマ初期化関数（void _RI_init_cmt(void)）が記述された ri_cmt.h を出力します。
基本クロック用タイマ初期化関数は，ブート処理関数（PowerON_Reset_PC( )）から呼び出してください。

第 9 章　システム状態管理機能

本章では，RI600V4 が提供しているシステム状態管理機能について解説しています。

9.1　概　　要

RI600V4 におけるシステム状態管理機能では，タスクの優先順位の回転，ディスパッチ禁止状態への遷移などといったシステムの状態を操作する機能のほかに，コンテキスト種別の参照，CPU ロック状態の参照などといったシステムの状態を参照する機能も提供しています。

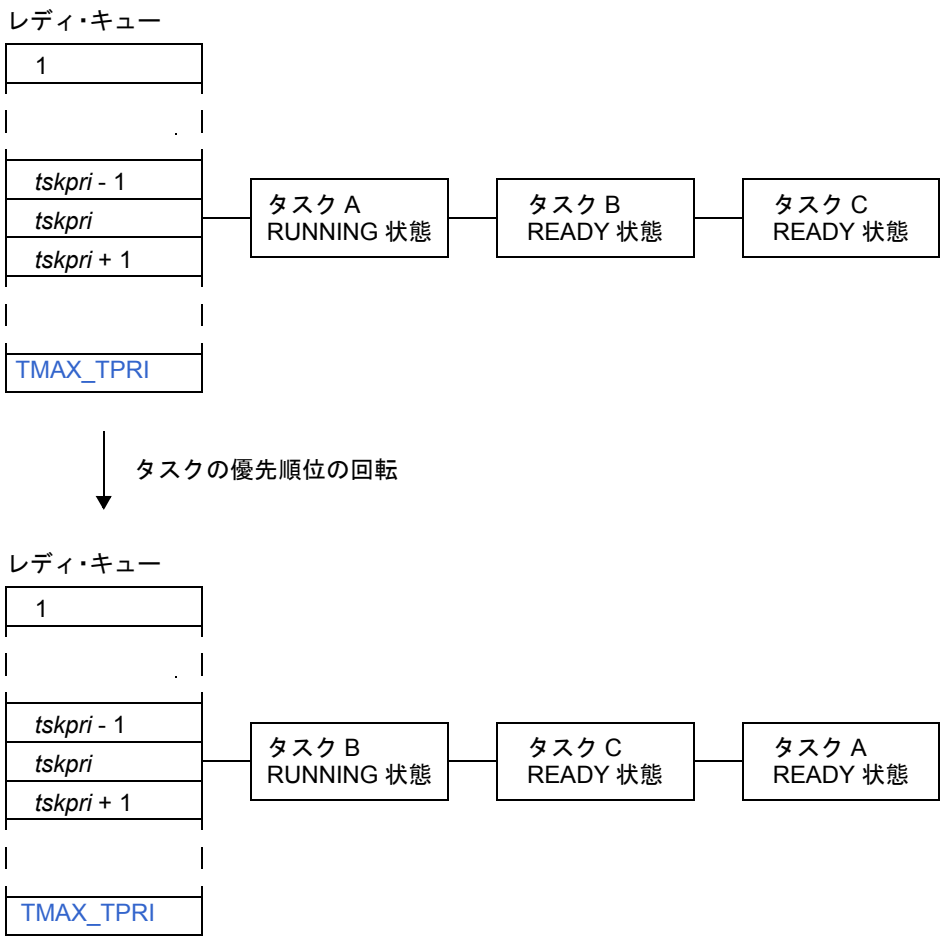
なお，システム・ダウン（vsys_dwn，ivsys_dwn）については「第 13 章　システム・ダウン」，RI600V4 の起動（vsta_knl，ivsta_knl）については「第 16 章　システム初期化処理」を参照してください。

9.2　タスクの優先順位の回転

タスクの優先順位の回転は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rot_rdq，irot_rdq
  パラメータ *tskpri* で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ，タスクの実行順序を明示的に変更します。

図 9 － 1　タスク優先順位の回転



以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void cychdr ( VP_INT exinf )            /* 周期ハンドラ */
{
        PRI     tskpri = 8;             /* 変数の宣言，初期化 */

        ............
        ............

        irot_rdq ( tskpri );            /* タスクの優先順位の回転 */

        ............
        ............

        return;                         /* 周期ハンドラの終了 */
}
```

備考 1　本サービス・コールでは，レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　本サービス・コールを周期ハンドラなどから一定周期で発行することにより，ラウンドロビン・スケジューリングを実現することができます。

備考 3　RI600V4　におけるレディ・キューは，優先度をキーとしたハッシュ・テーブルであり，実行可能な状態（RUNNING 状態または READY 状態）へと遷移したタスクが FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RI600V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。

備考 4　*tskpri* に TPRI_SELF を指定すると，自タスクのベース優先度を対象とします。
ミューテックスをロック中のタスクの現在優先度は，ベース優先度と異なる場合があります。この場合，そのタスクが本サービス・コールで *tskpri* に TPRI_SELF を指定しても，自タスクが属する現在優先度のレディ・キューを回転することはできません。

9.3　RUNNING 状態のタスクの参照

RUNNING 状態のタスクの参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_tid，iget_tid
  RUNNING 状態のタスクの ID をパラメータ *p_tskid* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void inthdr ( void )                    /* 割り込みハンドラ */
{
        ID      p_tskid;                /* 変数の宣言 */

        ............
        ............

        iget_tid ( &p_tskid );          /*RUNNING 状態のタスクの参照 */

        ............
        ............

        return;                         /* 割り込みハンドラの終了 */
}
```

備考　本サービス・コールでは，RUNNING 状態へと遷移しているタスクが存在しなかった場合には，パラメータ *p_tskid* で指定された領域に TSK_NONE を格納しています。

9.4　CPU ロック状態への移行と解除

CPU ロック状態では，タスクのスケジューリングは禁止され，カーネル管理割り込みもマスクされます。つまり，カーネル管理外割り込みハンドラを除くすべての処理プログラムに対して，排他的に処理を行うことができます。
以下のサービス・コールによって，CPU ロック状態へ移行します。

- loc_cpu，iloc_cpu
  システムを CPU ロック状態へ移行させます。
  ただし，CPU ロック状態では，発行可能なサービス・コールは以下に制限されます。

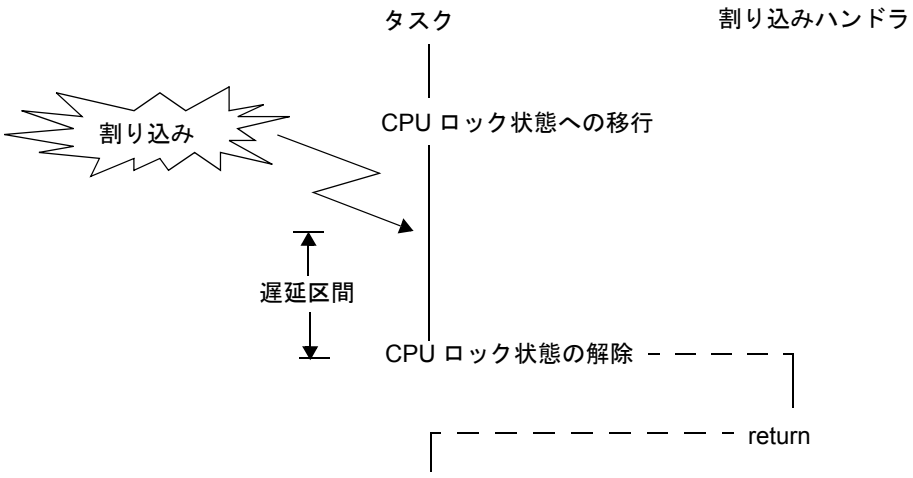
発行可能なサービス・コール	機能概要
ext_tsk	自タスクの終了（CPU ロック状態は解除されます）
loc_cpu，iloc_cpu	CPU ロック状態への移行
unl_cpu，iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_ctx	コンテキスト種別の参照
sns_dpn	ディスパッチ保留状態の参照
vsys_dwn，ivsys_dwn	システム・ダウン

CPU ロック状態は，以下のサービス・コールおよび ext_tsk によって解除されます。

- unl_cpu，iunl_cpu
  システムを CPU ロック解除状態へ移行させます。

以下に，“CPU ロック状態 ” を利用した際の処理の流れを示します。

図 9 － 2　CPU ロック状態



以下に，CPU ロック状態への移行とその解除の記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        .............
        .............

        loc_cpu ( );                    /*CPU ロック状態への移行 */

        .............                   /*CPU ロック状態 */
        .............

        unl_cpu ( );                    /*CPU ロック状態の解除 */

        .............
        .............
}
```

備考 1　CPU ロック状態の解除は，loc_cpu または iloc_cpu を発行した処理プログラムが終了する以前に行う必要があります。

備考 2　loc_cpu，iloc_cpu では，ロック要求のキューイングが行われません。このため，すでに CPU ロック状態の場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　unl_cpu，iunl_cpu では，解除要求のキューイングが行われません。このため，すでに非 CPU ロック状態の場合には，何も処理は行わず，エラーとしても扱いません。

備考 4　unl_cpu，iunl_cpu では，dis_dsp によるディスパッチ禁止状態は解除されません。

備考 5　CPU ロック状態の期間は基本クロック用タイマ割り込みはマスクされます。このため，CPU ロック状態の期間が長くなると，時間管理機能で扱う時間が遅れる場合があります。

備考 6　“ カーネル管理割り込み ” については，「10.1　割り込みの種類」を参照してください。

9.5　CPU ロック状態の参照

複数のタスクやハンドラから呼ばれる関数では，現在の CPU ロック状態を参照したい場合があります。このような場合には，sns_loc を使用します。

- sns_loc
  本サービス・コールは，CPU ロック状態か否かを調べます。本サービス・コールは，戻り値として，CPU ロック状態の場合は TRUE，CPU ロック解除状態の場合は FALSE を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
        BOOL ercd;                      /* 変数の宣言 */

        .............
        .............

        ercd = sns_loc ( );             /*CPU ロック状態の参照 */

        if ( ercd == TRUE ) {
                .............           /*CPU ロック状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* 非 CPU ロック状態 */
                .............
        }

        .............
        .............
}
```

9.6　ディスパッチ禁止状態への移行と解除

ディスパッチ禁止状態では，タスクのスケジューリングが禁止されるため，他のタスクに対して排他的に処理を行うことができます。

以下のサービス・コールによって，ディスパッチ禁止状態へ移行します。また，chg_ims によって割り込みマスク (PSW.IPL) を 0 以外に変更したときにも，ディスパッチ禁止状態へ移行します。

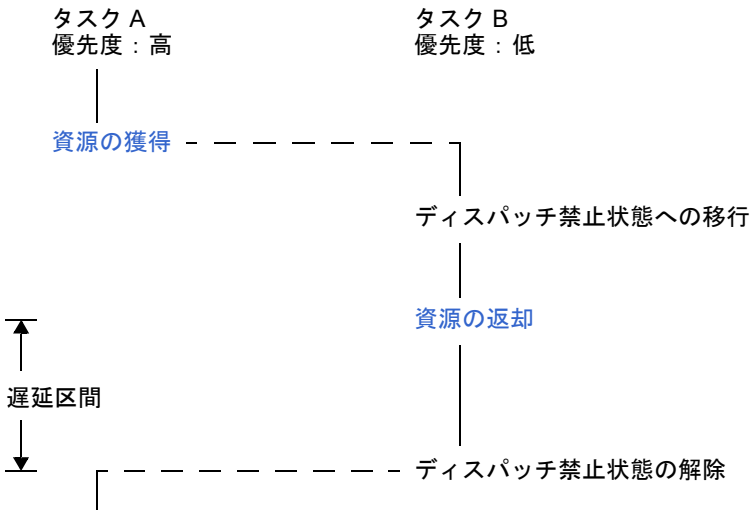
- dis_dsp
  システムをディスパッチ禁止状態へ移行させます。

ディスパッチ禁止状態は，以下のサービス・コール，ext_tsk，および chg_ims によって割り込みマスク (PSW.IPL) を 0 に変更したときに解除されます。

- ena_dsp
  システムをディスパッチ許可状態へ移行せます。

以下に，“ ディスパッチ禁止状態 ” を利用した際の処理の流れを示します。

図 9 － 3　ディスパッチ禁止状態



以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        .............
        .............

        dis_dsp ( );                    /* ディスパッチ禁止状態への移行 */

        .............                   /* ディスパッチ禁止状態 */
        .............

        ena_dsp ( );                    /* ディスパッチ禁止状態の解除 */

        .............
        .............
}
```

備考 1　ディスパッチ禁止状態の解除は，dis_dsp を発行したタスクが DORMANT 状態へ遷移する以前に行う必要があります。

備考 2　dis_dsp では，禁止要求のキューイングが行われません。このため，すでにディスパッチ禁止状態の場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　ena_dsp では，許可要求のキューイングが行われません。このため，すでにディスパッチ許可状態の場合には，何も処理は行わず，エラーとしても扱いません。

備考 4　ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem, wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

9.7　ディスパッチ禁止状態の参照

複数のタスクやハンドラから呼ばれる関数では，現在のディスパッチ禁止状態を参照したい場合があります。このような場合には，sns_dsp を使用します。

- sns_dsp
  本サービス・コールは，ディスパッチ禁止状態か否かを調べます。本サービス・コールは，戻り値として，ディスパッチ禁止状態の場合は TRUE，ディスパッチ許可状態の場合は FALSE を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
        BOOL ercd;                      /* 変数の宣言 */

        ............
        ............

        ercd = sns_dsp ( );             /* ディスパッチ禁止状態の参照 */

        if ( ercd == TRUE ) {
                ............            /* ディスパッチ禁止状態 */
                ............
        } else if ( ercd == FALSE ) {
                ............            /* ディスパッチ許可状態 */
                ............
        }

        ............
        ............
}
```

9.8　コンテキスト種別の参照

複数のタスクやハンドラから呼ばれる関数では，現在のコンテキスト種別を参照したい場合があります。このような場合には，sns_ctx を使用します。

- sns_ctx
  本サービス・コールは，本サービス・コールを発行した処理プログラムのコンテキスト種別を調べます。本サービス・コールは，戻り値として，非タスク・コンテキストの場合は TRUE，タスク・コンテキストの場合は FALSE を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
        BOOL ercd;                      /* 変数の宣言 */

        .............
        .............

        ercd = sns_ctx ( );             /* コンテキスト種別の参照 */

        if ( ercd == TRUE ) {
                .............           /* 非タスク・コンテキスト処理 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* タスク・コンテキスト処理 */
                .............
        }

        .............
        .............
}
```

9.9　ディスパッチ保留状態の参照

以下のいずれかの条件を満たすときをディスパッチ保留状態と呼びます。

- ディスパッチ禁止状態
- CPU ロック状態
- ハンドラなど，PSW.IPL>0 の状態

複数のタスクやハンドラから呼ばれる関数では，現在のディスパッチ保留を参照したい場合があります。このような場合には，sns_dpn を使用します。

- sns_dpn
  本サービス・コールは，ディスパッチ保留状態か否かを調べます。本サービス・コールは，戻り値として，ディスパッチ保留状態の場合は TRUE，ディスパッチ保留状態でない場合は FALSE を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
        BOOL ercd;                      /* 変数の宣言 */

        .............
        .............

        ercd = sns_dpn ( );             /* ディスパッチ保留状態の参照 */

        if ( ercd == TRUE ) {
                .............           /* ディスパッチ保留状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* 非ディスパッチ保留状態 */
                .............
        }

        .............
        .............
}
```

第 10 章　割り込み管理機能

本章では，RI600V4 が提供している割り込み管理機能について解説しています。

10.1　割り込みの種類

割り込みは，カーネル管理割り込みとカーネル管理外割り込みに分類されます。

- カーネル管理割り込み
  カーネル割込マスクレベルより割り込み優先レベルが低い割り込みをカーネル管理割り込みといいます。
  カーネル管理割り込みハンドラでは，サービス・コールを呼び出すことができます。
  サービス・コール処理中にカーネル管理割り込みが発生した場合，カーネル管理割り込みを受け付け可能となるまで割り込み受理が遅延されます。

- カーネル管理外割り込み
  カーネル割込マスクレベルより割り込み優先レベルが高い割り込みをカーネル管理外割り込みといいます。ノンマスカブル割り込みは，カーネル管理外割り込みの扱いとなります。
  カーネル管理外割り込みハンドラでは，サービス・コールを呼び出してはなりません。
  サービス・コール処理中にカーネル管理外割り込みが発生した場合でも，直ちに割り込み受理が受理されるため，カーネル処理に依存しない高速な割り込み応答が可能です。

備考　カーネル割り込みマスクレベルは，システム情報（system）のカーネル割り込みマスクレベル（system_IPL）で定義します。

10.2　RX MCU の高速割り込み

RX MCU は「高速割り込み」機能をサポートしています。ひとつの割り込み要因だけを高速割り込みとすることができます。高速割り込みは，割り込み優先レベル 15 として扱われます。高速割り込みを使用する場合は，割り込み優先レベル 15 の割り込み要因をひとつに限定する必要があります。

RI600V4 で高速割り込みを使用する場合は，その割り込みはカーネル管理外割り込みとして扱う必要があります。つまり，カーネル割り込みマスクレベル (system.system_IPL) は，14 以下に設定する必要があります。

システム・コンフィギュレーション・ファイルで高速割り込みを定義する際には，os_int に NO を指定し，さらに pragma_switch に F を指定する必要があります。

また，ブート処理関数（PowerON_Reset_PC( )）で RX-MCU の FINTV レジスタをそのハンドラの開始アドレスに初期化する必要があります。

10.3　CPU 例外の扱い

以下の CPU 例外はカーネル管理外割り込みの扱いとなります。

- 無条件トラップ（INT，BRK 命令）
  なお，INT #1 ～ #8 は RI600V4 で予約されています。
- 未定義命令例外
- 特権命令例外
- 浮動小数点例外

一方，アクセス例外はカーネル管理割り込みの扱いとなります。

10.4　基本クロック用タイマ割り込み

時間管理機能は，一定周期で発生する基本クロック用タイマ割り込みを利用して実現されています。基本クロック用タイマ割り込みが発生した際には，RI600V4 が提供している時間管理用割り込みハンドラが起動し，時間に関連した処理（システム時刻の更新，タスクの遅延起床／タイムアウト，周期ハンドラの起動など）が実行されます。

10.5　多重割り込み

RI600V4 では，割り込みハンドラ内で再び割り込みが発生することを “ 多重割り込み ” と呼んでいます。

多重割り込みを許可するかどうかは，可変ベクタの割り込みハンドラごとに設定できます。詳細は，「19.18　可変ベクタ情報（interrupt_vector[]）」を参照してください。

10.6　割り込みハンドラ

割り込みハンドラは，割り込みが発生した際に起動される割り込み処理専用ルーチンです。
なお，RI600V4 では，割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，割り込みが発生した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。

10.6.1　割り込みハンドラの基本型

以下に，割り込みハンドラを記述する場合の基本型を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void inthdr ( void )
{
        ............
        ............

        return;                         /* 割り込みハンドラの終了 */
}
```

備考　ハンドラ関数のプロトタイプ宣言および #pragma interrupt ディレクテブは, cfg600 が kernel_id.h に出力します。

- スタック
  割り込みハンドラは，システム・スタックを使用します。

- サービス・コールの発行
  RI600V4 では，割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。
  カーネル管理割り込みハンドラでは,“発行有効範囲”が“非タスク”のサービスコールを発行可能です。なお, カーネル管理外割り込みハンドラでは，サービス・コールを呼び出してはなりません。

  備考　RI600V4 では，カーネル管理割り込みハンドラ内の処理を高速に終了させる目的から，カーネル管理割り込みハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，カーネル管理割り込みハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行はカーネル管理割り込みハンドラが終了するまで遅延され, 一括して行うようにしています。

- 処理開始時の PSW

表 10 － 1　割り込みハンドラ処理開始時の PSW

ビット	値	備考
I	登録時に pragma_switch に “E” を指定した場合は 1，そうでない場合は 0	
IPL	割り込み：当該割り込み優先レベル CPU 例外：例外発生前と同じ	処理開始時より下げてはなりません。
PM	0	スーパバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

10.6.2　割り込みハンドラの登録

RI600V4 では，割り込みハンドラの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

割り込みハンドラの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“interrupt_vector[]”（可変ベクタ割り込みハンドラの登録），および “interrupt_fvector[]”（固定ベクタ／例外ベクタ割り込みハンドラの登録）を使用して割り込みハンドラを定義することをいいます。

静的 API“interrupt_vector[]” の詳細は，「19.18　可変ベクタ情報（interrupt_vector[]）」を，静的 API“interrupt_fvector[]” の詳細は，「19.19　固定ベクタ／例外ベクタ情報（interrupt_fvector[]）」を参照してください。

10.7　処理プログラム内におけるマスカブル割り込み受け付け状態

RX MCU のマスカブル割り込みの受け付け状態は，PSW.I，PSW.IPL の値で変わります。詳細はハードウエアのマニュアルを参照してください。

初期状態は処理プログラムごとに決まっています。その詳細は，表 10 － 2 を参照してください。

表 10 － 2　処理プログラム起動時のマスカブル割り込み受け付け状態

処理プログラム名	PSW.I	PSW.IPL
タスク	1	0
周期ハンドラ， アラーム・ハンドラ	1	基本クロック割り込み優先レベル（IPL）
割り込みハンドラ	登録時にpragma_switchに“E”を指定した場合は 1，そうでない場合は 0	割り込み：当該割り込み優先レベル CPU 例外：例外発生前と同じ

10.8　マスカブル割り込みの受け付け禁止

マスカブル割り込みの受け付け禁止には，以下の方法があります。

- loc_cpu，iloc_cpu を使用して CPU ロック状態にする
- chg_ims，ichg_ims を使用して PSW.IPL を変更する
- PSW.I，PSW.IPL を直接変更する (ハンドラ限定）

10.8.1　loc_cpu, iloc_cpu を使用して CPU ロック状態にする

CPU ロック状態では，PSW.IPL はカーネル割り込みマスクレベル（system_IPL）となります。つまり，CPU ロック状態で禁止される割り込みは，カーネル管理割り込みのみです。

また，CPU ロック状態では使用可能なサービス・コールに制限があります。詳細は，「9.4　CPU ロック状態への移行と解除」を参照してください。

10.8.2　chg_ims，ichg_ims を使用して PSW.IPL を変更する

chg_ims，ichg_ims では，PSW.IPL を任意の値に変更することができます。

タスクで chg_ims を用いて PSW.IPL を 0 以外に変更すると，同時にディスパッチ禁止状態に移行し，chg_ims を用いて PSW.IPL を 0 に戻すと，同時にディスパッチ許可状態に戻ります。

タスクが chg_ims によって PSW.IPL を 0 以外に変更している間は，ena_dsp を呼び出さないようにしてください。ena_dsp を呼び出すと，その時点でディスパッチ許可状態に遷移します。タスク・ディスパッチが発生すると，PSW はディスパッチ先のタスクの状態に更新されるので，意図せずに IPL 値が下がってしまうことがあります。

ハンドラでは，PSW.IPL を処理開始時より下げてはなりません。

10.8.3　PSW.I, PSW.IPL を直接変更する（ハンドラ限定）

ハンドラでは，PSW.I，PSW.IPL を直接変更することができます。この方法は，ichg_ims よりも高速です。

ハンドラでは，PSW.IPL を処理開始時より下げてはなりません。

なお，コンパイラでは PSW を操作する以下の組み込み関数を用意しています。組み込み関数の詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

- set_ipl() ：PSW レジスタの IPL ビットの変更
- get_ipl() ：PSW レジスタの IPL ビットの参照
- set_psw() ：PSW レジスタの設定
- get_psw() ：PSW レジスタの参照

第 11 章　システム構成管理機能

本章では，RI600V4 が提供しているシステム構成管理機能について解説しています。

11.1　概　　要

RI600V4 におけるシステム構成管理機能では，バージョン情報を参照する機能を提供しています。

11.2　バージョン情報の参照

バージョン情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_ver，iref_ver
  バージョン情報をパラメータ *pk_rver* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        T_RVER  pk_rver;                /* データ構造体の宣言 */
        UH      maker;                  /* 変数の宣言 */
        UH      prid;                   /* 変数の宣言 */

        .............
        .............

        ref_ver ( &pk_rver );           /* バージョン情報の参照 */

        maker = pk_rver.maker;          /* メーカコードの獲得 */
        prid = pk_rver.prid;            /* カーネル識別番号の獲得 */

        .............
        .............
}
```

備考　　バージョン情報 T_RVER についての詳細は，「【バージョン情報 T_RVER の構造 】」を参照してください。

第 12 章　オブジェクト・リセット機能

本章では，RI600V4 が提供しているオブジェクト・リセット機能について解説しています。

12.1　概　　要

オブジェクト・リセット機能は，データ・キュー，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，および可変長メモリ・プールを初期状態に戻す機能で，μITRON4.0 仕様外の機能です。

12.2　データ・キューのリセット

データ・キューのリセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_dtq
  パラメータ *dtqid* で指定されたデータ・キューをリセットします。
  データ・キューに蓄えられていたデータは破棄されます。また，データ送信を待っていたタスクの待ち状態は解除され，そのタスクには戻り値として EV_RST が返されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = vrst_dtq ( dtqid );      /* データ・キューのリセット */

        .............
        .............
}
```

備考　　本サービス・コールでは，データ受信を待っていたタスクの待ち状態は解除されません。

12.3　メールボックスのリセット

メールボックスのリセットは, 以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_mbx
  パラメータ *mbxid* で指定されたメールボックスをリセットします。
  メールボックスに蓄えられたメッセージは，RI600V4 の管理から外れます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = vrst_mbx( mbxid );       /* メールボックスのリセット */

        .............
        .............
}
```

備考　　本サービス・コールでは，メッセージ受信を待っていたタスクの待ち状態は解除されません。

12.4　メッセージ・バッファのリセット

メッセージ・バッファのリセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_mbf
  パラメータ *mbfid* で指定されたメッセージ・バッファをリセットします。
  メッセージ・バッファに蓄えられていたメッセージは破棄されます。また，メッセージ送信を待っていたタスクの待ち状態は解除され，そのタスクには戻り値として EV_RST が返されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbfid = 1;              /* 変数の宣言，初期化 */

        .............
        .............

        ercd = vrst_mbf( mbfid );       /* メッセージ・バッファのリセット */

        .............
        .............
}
```

備考　　本サービス・コールでは，メッセージ受信を待っていたタスクの待ち状態は解除されません。

12.5　固定長メモリ・プールのリセット

固定長メモリ・プールのリセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_mpf

  パラメータ *mpfid* で指定された固定長メモリ・プールをリセットします。
  固定長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され，戻り値として EV_RST が返されます。
  また，すでに獲得されていた固定長メモリ・ブロックはすべて，固定長メモリ・プールに返却されます。このため，本サービス・コール以降はそれらの固定長メモリ・ブロックにアクセスしてはなりません。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */

        ............
        ............

        ercd = vrst_mpf( mpfid );       /* 固定長メモリ・プールのリセット */

        ............
        ............
}
```

12.6　可変長メモリ・プールのリセット

可変長メモリ・プールのリセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_mpl

  パラメータ *mplid* で指定された可変長メモリ・プールをリセットします。
  可変長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され，戻り値として EV_RST が返されます。
  また，すでに獲得されていた可変長メモリ・ブロックはすべて，可変長メモリ・プールに返却されます。このため，本サービス・コール以降はそれらの可変長メモリ・ブロックにアクセスしてはなりません。
  以下に，本サービス・コールの記述例を示します。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

void task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */

        ............
        ............

        ercd = vrst_mpl( mplid );       /* 可変長メモリ・プールのリセット */

        ............
        ............
}
```

第 13 章　システム・ダウン

本章では，RI600V4 が提供しているシステム・ダウン機能について解説しています。

13.1　概要

RI600V4 の稼働中に回復不可能な事象が発生するとシステム・ダウンとなり，システム・ダウン・ルーチンが呼び出されます。

13.2　ユーザ・オウン・コーディング部

システム・ダウン・ルーチンは，ユーザ・オウン・コーディング部として実装する必要があります。

備考　　RI600V4　で提供するサンプルでは，ブート処理ファイル　“resetprg.c”　内にシステム・ダウン・ルーチン（_RI_sys_dwn__( )）を実装しています。

13.2.1　システム・ダウン・ルーチン（_RI_sys_dwn__( )）

以下に，システム・ダウン・ルーチンを記述する場合の基本型を示します。システム・ダウン・ルーチンからリターンしてはなりません。

```
#include        "kernel.h"              /* 標準ヘッダ・ファイルの定義 */
#include        "kernel_id.h"           /*cfg600 が出力するヘッダ・ファイルの定義 */

                                        /*  プロトタイプ宣言  */
void _RI_sys_dwn__  ( W type, VW inf1, VW inf2, VW inf3 );

void _RI_sys_dwn__  ( W type, VW inf1, VW inf2, VW inf3 )
{
        ............
        ............

        while(1);
}
```

備考　　システム・ダウン・ルーチンの関数名は “_RI_sys_dwn__” です。

- スタック
  システム・ダウン・ルーチンは，システム・スタックを使用します。

- サービス・コールの発行
  システム・ダウン・ルーチンからサービス・コールを呼び出してはなりません。

- 処理開始時の PSW

表 13 － 1　システム・ダウン・ルーチン処理開始時の PSW

ビット	値	備考
I	0	
IPL	- *type* < 0 の場合：不定 - *type* ≧ 0 の場合：システム・ダウン発生前と同じ	処理開始時より下げてはなりません。
PM	0	スーパバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

13.2.2　システム・ダウン・ルーチンのパラメータ

- *type* == -1（カーネル管理割り込みハンドラ終了時のエラー）

表 13 － 2　システム・ダウン・ルーチンのパラメータ（*type* == -1）

inf1	inf2	inf3	解説
E_CTX（-25）	2	不定	カーネル管理割り込みハンドラ終了時点で, PSW.PMが1(ユーザ・モード) である。
	3	不定	カーネル管理割り込みハンドラ終了時点で，PSW.IPL ＞カーネル割り込みマスクレベルである。
	5	不定	カーネル管理割り込みハンドラ終了時点で，CPU ロック状態である。

- *type* == -2（ext_tsk のエラー）

表 13 － 3　システム・ダウン・ルーチンのパラメータ（*type* == -2）

inf1	inf2	inf3	解説
E_CTX（-25）	1	不定	非タスク・コンテキストから ext_tsk を呼び出した。
	4	不定	ext_tsk 呼び出し時点で，PSW.IPL ＞カーネル割り込みマスクレベルである。

- *type* == -3（組み込まれていないサービス・コールの呼び出し）

表 13 － 4　システム・ダウン・ルーチンのパラメータ（*type* == -3）

inf1	inf2	inf3	解説
E_NOSPT（-9）	不定	不定	組み込まれていないサービス・コールを呼び出した。

備考　「2.6.1　サービス・コール情報ファイルとコンパイラ・オプション “-ri600_preinit_mrc”」参照してください。

- *type* == -16（未定義の可変ベクタ割り込み）

表 13 － 5　システム・ダウン・ルーチンのパラメータ（*type* == -16）

inf1	inf2	inf3
- cfg600 で “-U ” オプションを指定しない場合 不定 - cfg600 で “-U ” オプションを指定した場合 ベクタ番号	CPU の割り込み処理によってスタックに退避された PC	CPU の割り込み処理によってスタックに退避された PSW

- *type* == -17（未定義の固定ベクタ／例外ベクタ割り込み）

表 13 － 6　システム・ダウン・ルーチンのパラメータ（*type* == -17）

inf1	inf2	inf3
- cfg600 で “-U ” オプションを指定しない場合 不定 - cfg600 で “-U ” オプションを指定した場合 ベクタ番号	CPU の割り込み処理によってスタックに退避された PC	CPU の割り込み処理によってスタックに退避された PSW

- *type* > 0（アプリケーションからの vsys_dwn，ivsys_dwn の呼び出し）
0，および負の type 値は RI600V4 用に予約されています。アプリケーションから vsys_dwn，ivsys_dwn を呼び出す場合は，正の type 値を使用してください。

表 13 － 7　システム・ダウン・ルーチンのパラメータ（*type* > 0）

inf1	inf2	inf3
vsys_dwn, ivsys_dwn に指定した値		

第 14 章　スケジューリング機能

本章では，RI600V4 が提供しているスケジューリング機能について解説しています。

14.1　概　　要

RI600V4 におけるスケジューリング機能では，動的に変化していくタスクの状態を直接参照することにより，タスクの実行順序を管理／決定し，最適なタスクに CPU の利用権を与える機能を提供しています。

14.2　処理の単位と優先順位

アプリケーションは，以下の処理単位によって実行制御されます。

- タスク
- 割り込みハンドラ
- 周期ハンドラ
- アラーム・ハンドラ

各処理単位は，以下の優先順位で処理されます。

1）割り込みハンドラ，周期ハンドラ，アラーム・ハンドラ

2）スケジューラ

3）タスク

なお，スケジューラとは，実行するタスクを特定し，切り換える RI600V4 の処理のことです。
割込みハンドラ，周期ハンドラ，およびアラーム・ハンドラはスケジューラよりも優先順位が高いため，これらが実行している間は，タスクは実行されません（「14.7　非タスク内におけるタスク・スケジューリング処理」参照）。
割り込みハンドラは，割り込み優先レベルが高いほど優先順位が高くなります。
周期ハンドラおよびアラーム・ハンドラの優先順位は，基本クロック割り込みの優先レベルの割り込みハンドラと同じです。
タスク間の優先順位は，タスクに付与された現在優先度に従います。

14.3　タスクの駆動方式

RI600V4 では，スケジューラの駆動方式として何らかの事象（きっかけ）が発生した際に起動する “ 事象駆動方式 ” を採用しています。

- 事象駆動方式

  RI600V4 における事象駆動方式では，以下に示した事象が発生した場合にスケジューラを起動し，タスクのスケジューリング処理を実行します。

  - タスクの状態遷移を引き起こす可能性があるサービス・コールの発行
  - 非タスク（周期ハンドラ，割り込みハンドラなど）からの復帰命令の発行
  - 時間管理機能を行う際に使用している基本クロック用タイマ割り込みの発生

14.4　タスクのスケジューリング方式

RI600V4 では，タスクのスケジューリング方式として各タスクに定義されている優先度を利用した“優先度方式”，および，RI600V4 のスケジューリング対象となってからの経過時間を利用した“FCFS 方式”を採用しています。

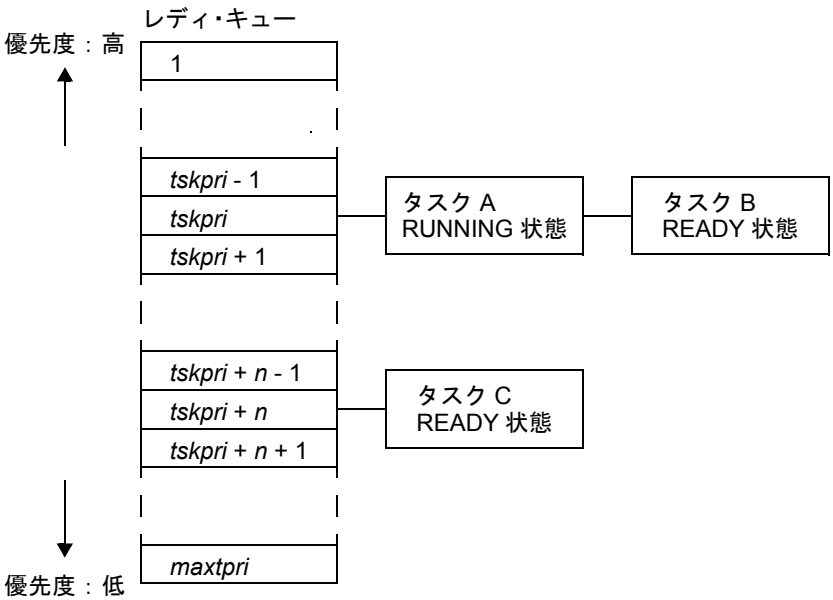
- 優先度方式
  RI600V4 における優先度方式では，実行可能な状態（RUNNING 状態または READY 状態）にある全タスクの中から“最も高い現在優先度を持つタスク”を選び出し，CPU の利用権を与えます。

- FCFS 方式
  “最も高い現在優先度を持つタスク”が複数存在する場合は，優先度方式だけではスケジューリング対象のタスクを決定することができません。この場合，RI600V4 は FCFS 方式（First Come First Served 方式）でスケジューリング対象のタスクを決定します。具体的は，それらのの中で最も早く実行可能な状態（READY 状態）へ遷移したタスクを選び出し，CPU の利用権を与えます。

14.4.1　レディ・キュー

RI600V4 では，タスクのスケジューリング方式を実現する手段として“レディ・キュー”を提供しています。
なお，RI600V4 におけるレディ・キューは，優先度をキーとしたハッシュ・テーブルであり，実行可能な状態（RUNNING 状態または READY 状態）へと遷移したタスクが FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RI600V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。
以下に，複数のタスクがレディ・キューにキューイングされている場合を示します。

図 14 － 1　スケジューリング方式（優先度方式，FCFS 方式）の実現



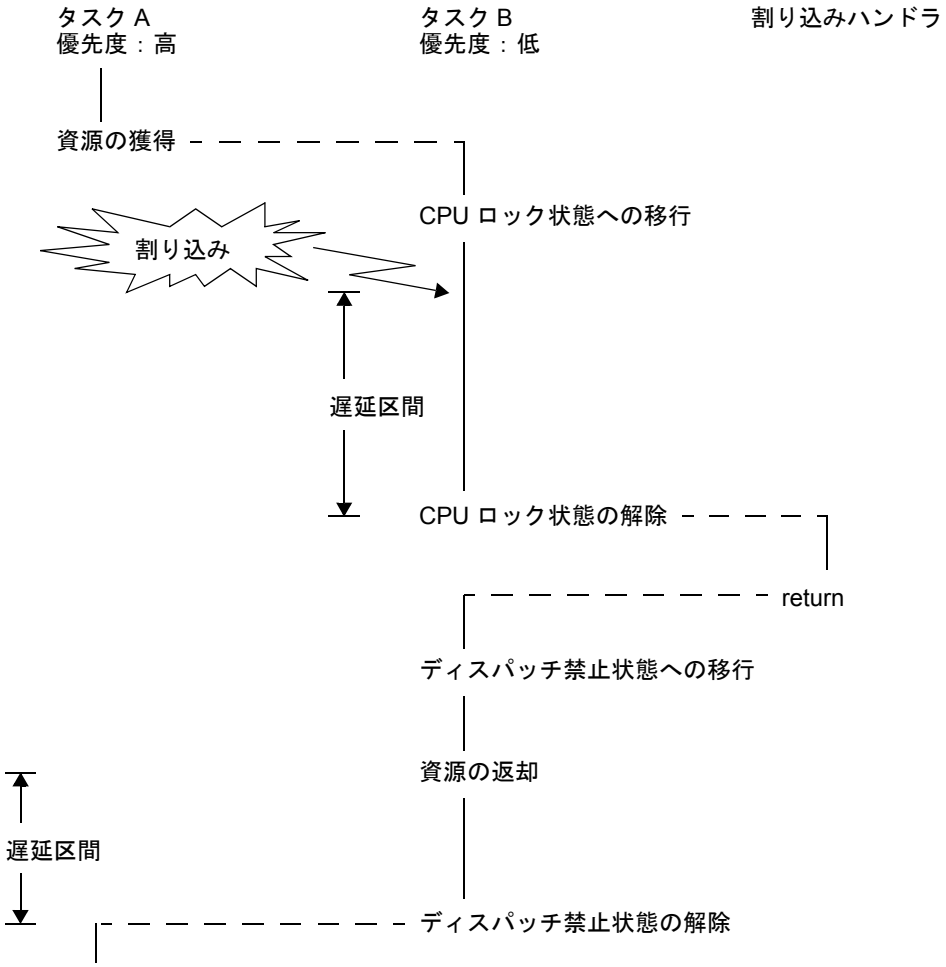
- レディ・キューの生成
  RI600V4 では，レディ・キューの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。
  レディ・キューの静的生成とは，システム・コンフィギュレーション・ファイルでシステム情報（system）のタスク優先度の最大値（priority）を定義することをいいます。

14.5　タスク・スケジューリングのロック機能

RI600V4 では，処理プログラムからスケジューラの状態を明示的に操作し，ディスパッチ処理を禁止／許可する機能（スケジューリングのロック機能）を提供しています。

以下に，スケジューリングのロック機能を利用した際の処理の流れを示します。

図 14 － 2　スケジューリングのロック機能



詳細は，「9.4　CPU ロック状態への移行と解除」，「9.6　ディスパッチ禁止状態への移行と解除」を参照してください，

14.6　アイドリング

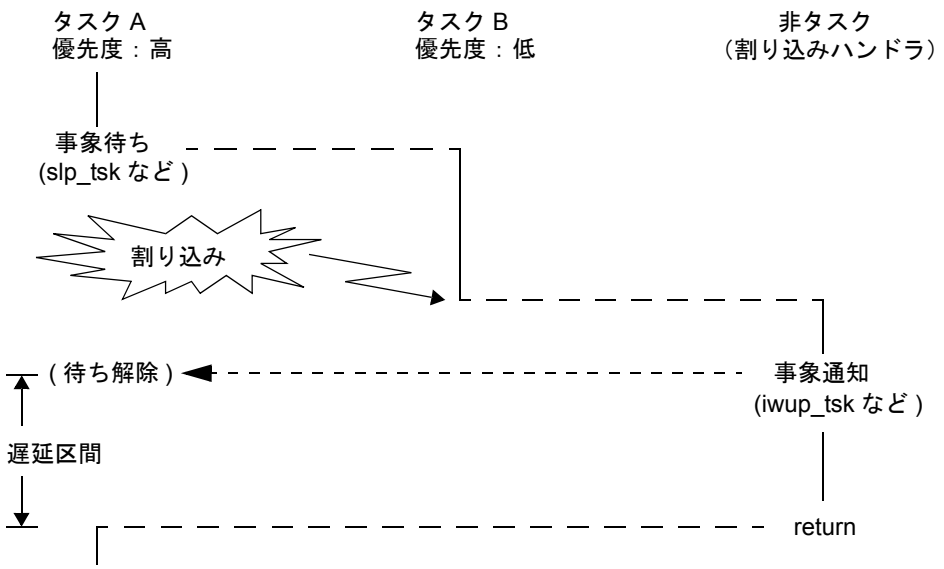
RUNNING 状態または READY 状態のタスクが存在しなくなると，RI600V4 内部で無限ループとなり，割り込みが発生するのを待ちます。

14.7　非タスク内におけるタスク・スケジューリング処理

「14.2　処理の単位と優先順位」に示したように，非タスク（割り込みハンドラ，周期ハンドラおよびアラーム・ハンドラ）はタスクよりも優先順位が高いため，非タスク処理が始まるとそれが完了するまで，タスクは実行されません。

以下に，非タスク内でディスパッチ処理を伴うサービス・コールを発行した際の例を示します。この例では，割り込みハンドラが事象通知を行った時点でタスク B よりも優先度の高いタスク A の待ち状態が解除されますが，この時点ではタスク A はまだ実行されず，非タスク (割り込みハンドラ）の処理が継続されます。非タスク処理が完了した時点でスケジューラが起動され，その結果タスク A が実行されます。

図 14 － 3　非タスク内におけるスケジューリング処理



第 15 章　リアルタイム OS タスク・アナライザ

15.1　概　　要

リアルタイム OS を組み込んだシステムを解析するとき，以下のような情報が必要になります。

- 処理プログラムの実行状況
- リアルタイム OS 資源の使用状況
- 処理プログラム単位の CPU 使用率

上記を実現するためのツールが「リアルタイム OS タスク・アナライザ」です。リアルタイム OS タスク・アナライザは，リアルタイム OS が出力した情報を解析してグラフィカルに表示します。

本章では，CubeSuite+ のリアルタイム OS タスク・アナライザを使用するための手順について説明します。リアルタイム OS タスク・アナライザの機能および操作方法は，「RI600V4 リアルタイム・オペレーティング・システム ユーザーズマニュアル 解析編」を参照してください。

15.2　トレース・モード

リアルタイム OS タスク・アナライザには，以下に示す使用形態（トレース・モード）があります。トレース・モードは，［タスク・アナライザ］タブで選択します。

- ハードウェア・トレース・モードでトレース・チャートを取得
  本モードでは，トレース情報はエミュレータやシミュレータが持つトレース・メモリに収集されます。

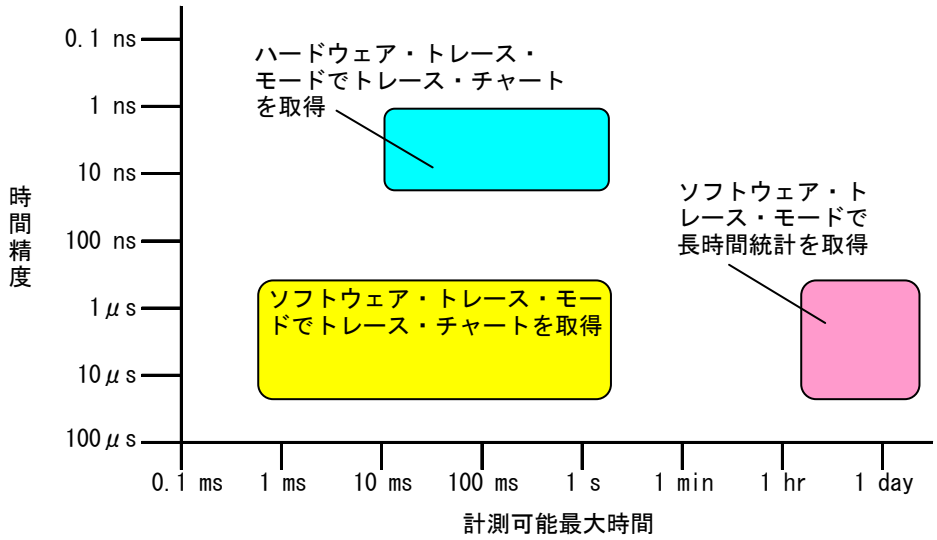
- ソフトウェア・トレース・モードでトレース・チャートを取得
  本モードでは，トレース情報はユーザ・メモリに確保したトレース・バッファに収集されます。バッファのサイズは，［タスク・アナライザ］タブで指定します。トレース・バッファ・サイズの見積もりについては，「15.4　トレース・バッファのサイズ（ソフトウェア・トレース・モードでトレース・チャートを取得）」を参照してください。
  本モードを使用するには，ユーザ・オウン・コーディング部の実装とシステム・コンフィギュレーション・ファイルの設定が必要です。「15.3.1　ソフトウェア・トレース・モードでトレース・チャートを取得」を参照してください。

- ソフトウェア・トレース・モードで長時間統計を取得
  本モードでは，トレース情報はユーザ・メモリに確保した RI600V4 の変数に収集されます。この変数のサイズは，概ね 2K バイト程度です。詳細は，「19.20.1　BRI_RAM セクション」を参照してください。
  本モードを使用するには，ユーザ・オウン・コーディング部の実装とシステム・コンフィギュレーション・ファイルの設定が必要です。「15.3.2　ソフトウェア・トレース・モードで長時間統計を取得」を参照してください。

- トレースしない
  リアルタイム OS タスク・アナライザは使用できません。

計測可能最大時間と時間精度は，トレース・モードによって異なります。図 15 － 1 にその目安を示します。

図 15 － 1　計測可能最大時間と時間精度



備考 1　「ハードウェア・トレース・モードでトレース・チャートを取得」では，計測可能最大時間は，エミュレータやシミュレータのトレース・メモリ・サイズに依存します。また，時間精度はエミュレータやシミュレータの仕様に依存します。

備考 2　「ソフトウェア・トレース・モードでトレース・チャートを取得」では計測可能最大時間はトレース・バッファのサイズに依存します。また，時間精度については，「15.3.1　ソフトウェア・トレース・モードでトレース・チャートを取得」を参照してください。

備考 3　「ソフトウェア・トレース・モードで長時間統計を取得」では，計測可能最大時間および時間精度については，「15.3.2　ソフトウェア・トレース・モードで長時間統計を取得」を参照してください。

また，リアルタイム OS タスク・アナライザを使用する場合は，使用しない場合と比較して，表 15 － 1 に示すようにターゲット・システムに対する影響があります。なお，表 15 － 1 に記載の処理時間は 100MHz 動作時の概値です。

表 15 － 1　ターゲット・システムへの影響

	ハードウェア・トレース・モードでトレース・チャートを取得	ソフトウェア・トレース・モードでトレース・チャートを取得	ソフトウェア・トレース・モードで長時間統計を取得
サービス・コール処理時間	0.5 ～ 1.5μ 秒程度悪化（状態が変化するタスク数に依存）	1.5 ～ 5μ 秒程度悪化（状態が変化するタスク数に依存）	悪化なし
タスク・ディスパッチ処理時間	0.2μ 秒程度悪化	0.7μ 秒程度悪化	0.6μ 秒程度悪化
割り込み処理時間	0.5μ 秒程度悪化	1 ～ 2μ 秒程度悪化	1 ～ 2μ 秒程度悪化
RAM 消費	悪化なし	バッファが必要	概ね 2k バイト必要
ユーザ・オウン・コーディング部の実装とシステム・コンフィギュレーション・ファイルの設定	不要	必要	必要

各モードの機能，図 15 － 1 および表 15 － 1 を参考に，使用するモードを決めてください。

トレース・モードは，［タスク・アナライザ］タブで選択します。その後，ビルドを行うことで，使用するモードに対応したリアルタイム OS モジュールが組み込まれたロード・モジュールが生成されます。

15.3　ソフトウェア・トレース・モードのユーザ・オウン・コーディング部

15.3.1　ソフトウェア・トレース・モードでトレース・チャートを取得

本モードでは，RI600V4 はタイム・スタンプをユーザ・オウン・コーディング部から取得します。タイム・スタンプを生成するために，通常はハードウェア・タイマを使用します。ハードウェア・タイマは，カウンタ長が 16 ビット以上である必要があります。なお，RX ファミリ MCU に標準的に搭載されている CMT（コンペア・マッチ・タイマ）は，この要件を満たしています。

以降で，実装が必要な関数・変数の仕様を説明します。なお，各種関数は RX ファミリ C/C++ コンパイラの ABI（Application Binary Interface）には従っていないため，アセンブリ言語で実装する必要があります。本項では，関数・変数名も，アセンブリ言語レベルで表記します。

備考　RI600V4 で提供するサンプルファイルは “trcSW_cmt.src” です。このファイルは，CMT のチャネル 1 を使用します。

1）__RIUSR_trcSW_base_time（時間精度）

__RIUSR_trcSW_read_cnt（タイム・スタンプ取得関数）が返す時間の単位を，ナノ秒単位の 32 ビット符号なし整数で，定数として定義します。
通常は，ハードウェア・タイマ・カウンタの 1 カウント時間を設定してください。
以下に，CMT を使用する場合の代表的な設定を示します。

PCLK	分周	時間精度（備考参照）
12.5 MHz	8	0.64 マイクロ秒
	32	2.56 マイクロ秒
	128	10.24 マイクロ秒
	512	40.96 マイクロ秒
25 MHz	8	0.32 マイクロ秒
	32	1.28 マイクロ秒
	128	5.12 マイクロ秒
	512	20.48 マイクロ秒

備考　時間精度 ＝ __RIUSR_trcSW_base_time

2) __RIUSR_trcSW_init_tmr（初期化関数）

説明	__RIUSR_trcSW_read_cnt（タイム・スタンプ取得関数）の仕様を実現するように，ハードウェア・タイマを初期化します。 サンプルでは，タイマ・クロックを 65536 回カウントした時に割り込みが発生するように CMT を初期化します。 本関数は vsta_knl サービスコールからコールバックされます。
パラメータ	なし
保証不要なレジスタ	R1 ～ R7, R14, R15
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	8 バイトまで

3) __RIUSR_trcSW_read_cnt（タイム・スタンプ取得関数）

説明	__RIUSR_trcSW_init_tmr（初期化関数）が呼び出された時点からの経過時間を，__RIUSR_trcSW_base_time（時間精度）の単位で，かつ 0 ～ 0x7FFFFFFF の範囲で返します。 サンプルが返す値は，下位 16 ビットは CMT カウンタ・レジスタ値，上位 16 ビットはタイマ割り込み回数です， 返す値は，前回の返値以上でなければなりません。
パラメータ	R5(出力) : 経過時間
保証不要なレジスタ	R3, R4
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	8 バイトまで

4） 割り込みハンドラ（関数名は任意）

説明	サンプルでは，タイマ・クロックを 65536 回カウントした時に本割り込みが発生します。
	本ハンドラでは，サービスコールを呼び出してはなりません。
	終了時は RTE 命令を行ってください。
	また，システム・コンフィギュレーション・ファイルで，以下のように本ハンドラを定義してください。ここでは，ベクタ番号が 29，ハンドラの関数名がアセンブリ言語レベルで __RIUSR_trcSW_interrupt の場合の例を示します。
	`interrupt_vector[29] {`
	`  entry_address = _RIUSR_trcSW_interrupt();`
	`  os_int = NO;`
	`};`

パラメータ	なし
保証不要なレジスタ	なし
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	「D.4　システム・スタック使用量の算出」で考慮してください。

15.3.2　ソフトウェア・トレース・モードで長時間統計を取得

本モードでは，RI600V4 はタイム・スタンプをユーザ・オウン・コーディング部から取得します。タイム・スタンプを生成するために，通常はハードウェア・タイマを使用します。ハードウェア・タイマは，カウンタ長が 16 ビット以上で，かつタイマ・クロックを 65536 回カウントしたときに割り込み発生可能である必要があります。なお，RX ファミリに標準的に搭載されている CMT は，この要件を満たしています。

以降で，実装が必要な関数・変数の仕様を説明します。なお，各種関数は RX ファミリ C/C++ コンパイラの ABI（Application Binary Interface）には従っていないため，アセンブリ言語で実装する必要があります。関数・変数名も，アセンブリ言語レベルで表記します。

備考　RI600V4 で提供するサンプルファイルは “trcLONG_cmt.src” です。このファイルは，マイコン内蔵の CMT のチャネル 1 を使用します。

1）__RIUSR_trcLONG_base_time（時間精度）

__RIUSR_trcLONG_read_cnt（タイム・スタンプ取得関数）が返す時間を，ナノ秒単位の 32 ビット符号なし整数で，定数として定義します。
通常は，ハードウェア・タイマ・カウンタの 1 カウント時間を設定してください。
以下に，CMT を使用する場合の代表的な設定を示します。

PCLK	分周	割り込みハンドラ実行時間の精度（備考 1 参照）	計測可能な割り込みハンドラ最大実行時間（備考 2 参照）	タスク，アイドリング実行時間の精度（備考 3 参照）	計測可能なタスク，アイドリング累計実行時間（備考 4 参照）
12.5 MHz	8	0.64 マイクロ秒	約 41 ミリ秒	5.12 マイクロ秒	約 6 時間 6 分
	32	2.56 マイクロ秒	約 167 ミリ秒	20.48 マイクロ秒	約 24 時間 26 分
	128	10.24 マイクロ秒	約 671 ミリ秒	81.92 マイクロ秒	約 97 時間 44 分
	512	40.96 マイクロ秒	約 2684 ミリ秒	327.68マイクロ秒	約 390 時間 56 分
25 MHz	8	0.32 マイクロ秒	約 20 ミリ秒	2.56 マイクロ秒	約 3 時間 3 分
	32	1.28 マイクロ秒	約 83 ミリ秒	10.24 マイクロ秒	約 12 時間 13 分
	128	5.12 マイクロ秒	約 335 ミリ秒	40.96 マイクロ秒	約 48 時間 52 分
	512	20.48 マイクロ秒	約 1342 ミリ秒	163.84マイクロ秒	約 195 時間 28 分

備考 1　割り込みハンドラ実行時間の精度 ＝ __RIUSR_trcLONG_base_time

備考 2　計測可能な割り込みハンドラ最大実行時間 ＝ __RIUSR_trcLONG_base_time × 65535

備考 3　タスク，アイドリング実行時間の精度 ＝ __RIUSR_trcLONG_base_time × 8

備考 4　計測可能なタスク累計実行時間 ＝ __RIUSR_trcLONG_base_time × 8 × 0xFFFFFFFF

2）__RIUSR_trcLONG_timer_lvl（割り込み優先レベル）

使用するハードウェア・タイマの割り込み優先レベルを，8 ビット符号なし整数で，定数として定義します。
このレベル以上の割り込み優先レベルの割り込みハンドラについては，時間は計測されません。そのハンドラの処理時間は，その割り込みが発生した時点の処理プログラム（タスク，別の割り込みハンドラ，カーネルのアイドリング）の処理時間に計上されます。
本タイマの割り込み優先レベルは，システムで最高とすることを推奨します。

3) __RIUSR_trcLONG_init_tmr（初期化関数）

説明	タイマ・クロックを 65536 回カウントした時に __RIUSR_trcLONG_timer_lvl（割り込み優先レベル）で定義した割り込み優先レベルの割り込みが発生するように，ハードウェア・タイマを初期化します。 本関数は vsta_knl サービスコールからコールバックされます。
パラメータ	なし
保証不要なレジスタ	R1 ～ R7, R14, R15
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	8 バイトまで

4) __RIUSR_trcLONG_read_cnt（タイム・スタンプ取得関数）

説明	前回の割り込みからの経過時間を，__RIUSR_trcLONG_base_time（時間精度）の単位で，かつ 0 ～ 65535 の範囲で返します。 サンプルでは，CMT カウンタ・レジスタ値を返します。
パラメータ	R1(出力) : 経過時間
保証不要なレジスタ	R4
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	8 バイトまで

5） 割り込みハンドラ（関数名は任意）

説明	RI600V4 の __RI_trcLONG_update_time 関数を呼び出してください。
	本ハンドラでは，サービスコールを呼び出してはなりません。
	終了時は RTE 命令を行ってください。
	また，システム・コンフィギュレーション・ファイルで，以下のように本ハンドラを定義してください。ここでは，ベクタ番号が 29，ハンドラの関数名がアセンブリ言語レベルで__RIUSR_trcLONG_interruptの場合の例を示します。
	`interrupt_vector[29] {`
	`entry_address = _RIUSR_trcLONG_interrupt();`
	`os_int = NO;`
	`};`
	なお，割り込み要求発生時点から，本割込みハンドラが動作するまでの間に発生したイベントのタイム・スタンプは不正となります。

パラメータ	なし
保証不要なレジスタ	なし
起動時の PSW（変更禁止）	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用可能スタックサイズ	「D.4　システム・スタック使用量の算出」で考慮してください。

6） RI600V4 の __RI_trcLONG_update_time 関数

本関数はユーザ・オウン・コーディング部ではなく，RI600V4 内に実装されています。本関数の仕様を以下に示します。

説明	RI600V4 が管理している現在時刻情報を更新します。 前述のタイマ割り込みハンドラから呼び出してください。
パラメータ	なし
保証しないレジスタ	R1,R2
呼び出し時の PSW	PM=0（スーパバイザ・モード） I=0（全割り込み禁止） U=0（システムスタック）
使用スタックサイズ	0 バイト（本関数を呼び出す BSR 命令が使用する 4 バイトは含みません）

15.4　トレース・バッファのサイズ（ソフトウェア・トレース・モードでトレース・チャートを取得）

トレース・バッファは，表 15 － 2 に示すタイミングで消費されます。

表 15 － 2　トレース・バッファの消費タイミング

タイミング	消費サイズ
サービス・コール発行直後	12 バイト
RI600V4 からアプリケーションに復帰する直前	8 バイト
タスク・ディスパッチ発生時	8 バイト
アイドリングに遷移時	8 バイト
割り込みハンドラ起動時	8 バイト
割り込みハンドラ終了時	8 バイト
周期ハンドラ起動時	8 バイト
周期ハンドラ終了時	8 バイト
アラーム・ハンドラ起動時	8 バイト
アラーム・ハンドラ終了時	8 バイト
タスク状態変化時	8 バイト

また，計測可能な時間の目安を表 15 － 3 に示します。

表 15 － 3　トレース・バッファを使い切るまでの時間の目安

イベント発生頻度	バッファ・サイズ			
	1 KB	4 KB	16 KB	64 KB
5μ 秒 / 回	約 0.6 ミリ秒	約 2.4 ミリ秒	約 9.6 ミリ秒	約 38 ミリ秒
10μ 秒 / 回	約 1.2 ミリ秒	約 4.8 ミリ秒	約 19 ミリ秒	約 77 ミリ秒
50μ 秒 / 回	約 6 ミリ秒	約 24 ミリ秒	約 96 ミリ秒	約 385 ミリ秒
100μ 秒 / 回	約 12 ミリ秒	約 48 ミリ秒	約 192 ミリ秒	約 771 ミリ秒
500μ 秒 / 回	約 60 ミリ秒	約 240 ミリ秒	約 963 ミリ秒	約 3855 ミリ秒

15.5　累計実行時間の誤差

タスクや割り込みハンドラなどの累計実行時間は，各回の実行時間の総和によって算出されます。したがって，実行回数が多くなるほど誤差が大きくなります。

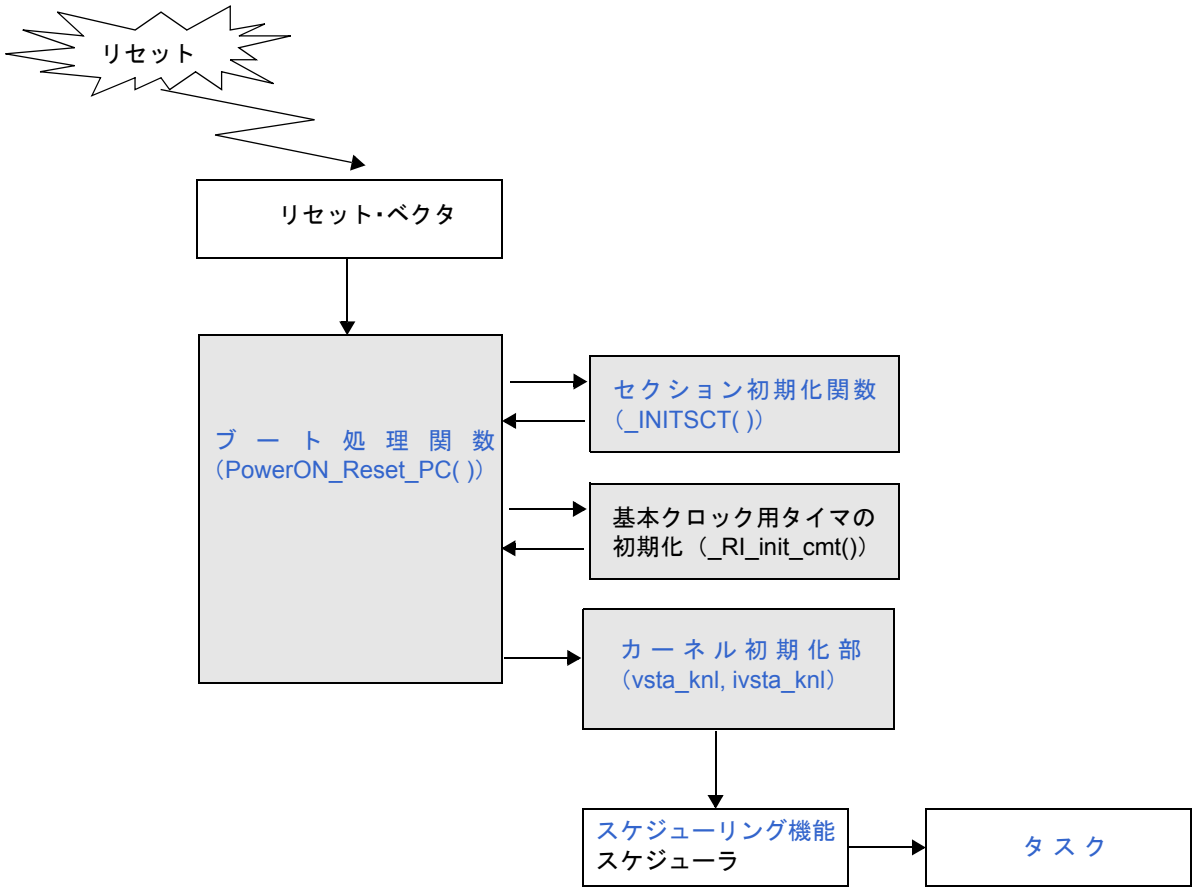
第 16 章　システム初期化処理

本章では，RI600V4 が提供しているシステム初期化処理について解説しています。

16.1　概　　要

以下に，リセットの発生から処理プログラム（タスク）に制御が移るまでに実行される処理の流れを示します。

図 16 － 1　処理の流れ（システム初期化処理）



16.2　ブート処理ファイル（ユーザ・オウン・コーディング部）

ブート処理ファイルには，以下を記述します。

1）ブート処理関数（PowerON_Reset_PC( )）
2）システム・ダウン・ルーチン（_RI_sys_dwn__( ) ）
3）kernel_ram.h および kernel_rom.h の取り込み

備考　RI600V4 で提供するサンプルのブート処理ファイルは “resetprg.c” です。このファイルには，システム・ダウン・ルーチン（_RI_sys_dwn__( ) ）も含んでいます。

16.2.1　ブート処理関数（PowerON_Reset_PC( )）

ブート処理関数はリセットベクタに登録されるプログラムで，スーパバイザ・モードで実行します。通常は , 以下のような処理を行います。

- プロセッサ，ハードウエアの初期化
  RX MCU の高速割り込みを使用する場合は，FINTV レジスタを高速割り込みハンドラの開始アドレスに初期化してください。

- C ／ C++ 言語ソフトウエアの実行環境の初期化（セクションの初期化など）

- 基本クロック用タイマの初期化
  cfg600 が出力する “ri_cmt.h” で実装されている “void _RI_init_cmt(void)” を呼び出してください。
  「8.9　基本クロック用タイマの初期化」も参照してください。

- RI600V4 を起動（vsta_knl，ivsta_knl の呼び出し）

- ブート処理の基本型
  ブート処理関数は “void PowerON_Reset_PC (void)” としてください。これ以外の関数名とする場合は，システム・コンフィギュレーション・ファイルで interrupt_fvector[31] にその関数名を定義する必要があります。
  ブート処理関数の例については，「16.2.4　ブート処理ファイルの例」を参照してください。

  備考　静的 API“interrupt_fvector[]” については，「19.19　固定ベクタ／例外ベクタ情報（interrupt_fvector[]）」を参照してください。

- ブート処理関数での処理
  以下に留意してください。

  - スタックの切り替え
    以下のように #pragma entry ディレクティブを記述してください。これにより，ブート処理関数の先頭でスタック・ポインタ (ISP) をシステム・スタックに設定するコードが生成されます。

```
#pragma entry PowerON_Reset_PC
```

  - PSW レジスタ
    カーネル初期化部（vsta_knl, ivsta_knl）を呼び出すまでは，全割り込みを禁止した状態とスーパバイザ・モードを維持してください。CPU リセット直後はこの状態（PSW.I = 0，PSW.PM=0）となっているので，通常は PSW レジスタを変更しないでください。

  - EXTB レジスタ（RXv2 アーキテクチャ）
    必要に応じ，EXTB レジスタを FIX_INTERRUPT_VECTOR セクションの先頭アドレスに初期化してください。
    2.6.4 節の「FIX_INTERRUPT_VECTOR セクション」も参照してください。

  - サービス・コールの発行
    ブート処理が開始された時点では “ カーネル初期化部（vsta_knl, ivsta_knl）の実行 ” が行われていません。したがって，ブート処理では，vsta_knl，ivsta_knl を除くサービス・コールの発行が禁止されています。

16.2.2　kernel_ram.h および kernel_rom.h の取り込み

cfg600 が出力した RAM 定義ファイル（kernel_ram.h）と ROM 定義ファイル（kernel_rom.h）を，ブート処理ファイルの最後で，この順にインクルードしてください。

16.2.3　ブート処理ファイルのコンパイラ・オプション

以下のコンパイラ・オプションの指定が必須です。

- “-lang=c” または “-lang=c99”
- “-nostuff”
- 適切な “-isa” または “-cpu”

備考　　コンパイラ・オプション “-isa” は，コンパイラ CC-RX V2.01 以降でサポートされています。

16.2.4　ブート処理ファイルの例

```
#include    <machine.h>
#include    <_h_c_lib.h>
//#include  <stddef.h>                  // Remove the comment when you use errno
//#include  <stdlib.h>                  // Remove the comment when you use rand()
#include    "typedefine.h"              // Define Types
#include    "kernel.h"                  // Provided by RI600V4
#include    "kernel_id.h"               // Generated by cfg600

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
#include    "ri_cmt.h"   // Generated by cfg600
                  // Do comment-out when clock.timer is either NOTIMER or OTHER.
#endif

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
void main(void);
#ifdef __cplusplus
}
#endif

//#ifdef __cplusplus                // Use SIM I/O
//extern "C" {
//#endif
//extern void _INIT_IOLIB(void);
//extern void _CLOSEALL(void);
//#ifdef __cplusplus
//}
//#endif

#define FPSW_init 0x00000000    // FPSW bit base pattern

//extern void srand(_UINT);     // Remove the comment when you use rand()
//extern _SBYTE *_s1ptr;        // Remove the comment when you use strtok()

//#ifdef __cplusplus            // Use Hardware Setup
//extern "C" {
//#endif
//extern void HardwareSetup(void);
//#ifdef __cplusplus
//}
//#endif

//#ifdef __cplusplus   // Remove the comment when you use global class object
//extern "C" {         // Sections C$INIT and C$END will be generated
//#endif
//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//#endif
```

```
#pragma section ResetPRG        // output PowerON_Reset to PResetPRG section

////////////////////////////////////////////////////////////////////////////////
// Boot processing
////////////////////////////////////////////////////////////////////////////////
#pragma entry PowerON_Reset_PC

void PowerON_Reset_PC(void)
{

#ifdef __ROZ                    // Initialize FPSW
#define _ROUND 0x00000001           // Let FPSW RMbits=01 (round to zero)
#else
#define _ROUND 0x00000000           // Let FPSW RMbits=00 (round to nearest)
#endif
#ifdef __DOFF
#define _DENOM 0x00000100           // Let FPSW DNbit=1 (denormal as zero)
#else
#define _DENOM 0x00000000           // Let FPSW DNbit=0 (denormal as is)
#endif

//    set_extb(__sectop("FIX_INTERRUPT_VECTOR"));// Initialize EXTB register
                                                 //  (only for RXv2 arch.)

    set_fpsw(FPSW_init | _ROUND | _DENOM);

    _INITSCT();

//  _INIT_IOLIB();                  // Use SIM I/O

//  errno=0;                        // Remove the comment when you use errno
//  srand((_UINT)1);                // Remove the comment when you use rand()
//  _s1ptr=NULL;                    // Remove the comment when you use strtok()

//  HardwareSetup();                // Use Hardware Setup
    nop();

//  set_fintv(<handler address>);   // Initialize FINTV register

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
    _RI_init_cmt();  // Initialize CMT for RI600V4
            // Do comment-out when clock.timer is either NOTIMER or OTHER.
#endif

//  _CALL_INIT();           // Remove the comment when you use global class object

    vsta_knl();                     // Start RI600V4
                                    // Never return from vsta_kn
//  _CLOSEALL();                    // Use SIM I/O

//  _CALL_END();            // Remove the comment when you use global class object

    brk();
}
```

```
////////////////////////////////////////////////////////////////////////////
// System down routine for RI600V4
////////////////////////////////////////////////////////////////////////////
#pragma section P PRI_KERNEL
#pragma section B BRI_RAM
struct SYSDWN_INF{
    W   type;
    VW  inf1;
    VW  inf2;
    VW  inf3;
};

volatile struct SYSDWN_INF _RI_sysdwn_inf;

void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 )
{
   // Now PSW.I=0 (all interrupts are masked.)
    _RI_sysdwn_inf.type = type;
    _RI_sysdwn_inf.inf1 = inf1;
    _RI_sysdwn_inf.inf2 = inf2;
    _RI_sysdwn_inf.inf3 = inf3;

   while(1)
   ;
}

////////////////////////////////////////////////////////////////////////////
// RI600V4 system data
////////////////////////////////////////////////////////////////////////////
#include "kernel_ram.h"     // generated by cfg600
#include "kernel_rom.h"     // generated by cfg600
```

16.3　カーネル初期化部（vsta_knl, ivsta_knl）

カーネル初期化部は，vsta_knl，ivsta_knl の呼び出しによって実行されます。vsta_knl，ivsta_knl は，通常はブート処理関数（PowerON_Reset_PC( )）から呼び出します。
カーネル初期化部では，以下に示した処理が実行されます。

1）ISP レジスタを，SI セクションの最終アドレス +1 に初期化
2）INTB レジスタを，cfg600 によって生成された可変ベクタ・テーブル（INTERRUPT_VECTOR セクション）先頭アドレスに初期化
3）システム時刻を 0 に初期化
4）システム・コンフィギュレーション・ファイルで定義された各種オブジェクトの生成
5）スケジューラに制御を移す

16.4　セクション初期化関数（_INITSCT( )）

ブート処理関数（PowerON_Reset_PC( )）から呼び出されるセクション初期化関数 “_INITSCT( )” は，コンパイラによって提供されます。_INITSCT( ) は，セクション情報ファイル（ユーザ・オウン・コーディング部）によって生成されたテーブルにしたがって，未初期化データ・セクションの 0 クリア，および初期化データ・セクションの初期化を行います。

初期化対象のセクションは、セクション情報ファイルのセクション初期化用テーブル (DTBL,BTBL) へ記述する必要があります。_INITSCT() が使用するセクションの先頭アドレスおよび最終アドレスを、セクションアドレス演算子を用いて設定します。セクション初期化用テーブルのセクション名は、未初期化データ領域を C$BSEC、初期化データ領域を C$DSEC で宣言します。

また，DTBL に記述した初期化データ・セクションは，リンカ・オプション “-rom” を用いて ROM から RAM にマップする必要があります。詳細は「2.6.5　初期化データ・セクション」を参照してください。

備考　_INITSCT( ) の詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

16.4.1　セクション情報ファイル（ユーザ・オウン・コーディング部）

セクション情報ファイルは，ユーザ・オウン・コーディング部としてユーザが作成する必要があります。
以下に，セクション情報ファイルの例を示します。

備考　RI600V4 が提供するサンプルのセクション情報ファイルは “dbsct.c” です。

```
#include "typedefine.h"

#pragma unpack

#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s;       /* Start address of the initialized data section in ROM */
    _UBYTE *rom_e;       /* End address of the initialized data section in ROM   */
    _UBYTE *ram_s;       /* Start address of the initialized data section in RAM */
}   _DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
    /* RI600V4 section */
    { __sectop("DRI_ROM"), __secend("DRI_ROM"), __sectop("RRI_RAM") }
};
#pragma section C C$BSEC
extern const struct {
    _UBYTE *b_s;         /* Start address of non-initialized data section */
    _UBYTE *b_e;         /* End address of non-initialized data section */
}   _BTBL[] = {
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") }
};

#pragma section

/*
** CTBL prevents excessive output of L1100 messages when linking.
** Even if CTBL is deleted, the operation of the program does not change.
*/
_UBYTE * const _CTBL[] = {
    __sectop("C_1"), __sectop("C_2"), __sectop("C"),
    __sectop("W_1"), __sectop("W_2"), __sectop("W")
};

#pragma packoption
```

16.5　固定ベクタ・テーブル／例外ベクタ・テーブルのレジスタなど

MCU によっては，固定ベクタ・テーブル（RXv1 アーキテクチャ）／例外ベクタ・テーブル（RXv2 アーキテクチャ）内の 0xFFFFFF80 ～ 0xFFFFFFBF 番地に，エンディアン選択レジスタやオンチップ・デバッガ ID コード・プロテクトなどが割り当てられています。これらを設定するには，システム・コンフィギュレーション・ファイルに“interrupt_fvector[]”を記述してください。詳細は，「19.19　固定ベクタ／例外ベクタ情報（interrupt_fvector[]）」を参照してください。

第 17 章　データ・タイプとマクロ

本章では，RI600V4 が提供するサービス・コールを発行する際に使用するデータ・タイプ，マクロについて解説しています。

備考　<ri_root> は，RI600V4 のインストール・フォルダを表しています。
デフォルトは，“ C:¥Program Files¥Renesas Electronics¥CubeSuite+¥RI600V4” です。

17.1　データ・タイプ

以下に，サービス・コールを発行する際に指定する各種パラメータのデータ・タイプ一覧を示します。
データ・タイプのマクロ定義は，<ri_root>¥inc600¥kernel.h，または kernel.h から呼び出される <ri_root>¥inc600¥itron.h で行われています。

表 17 － 1　データ・タイプ

マクロ	型	意味
B	signed char	符号付き 8 ビット整数
H	signed short	符号付き 16 ビット整数
W	signed long	符号付き 32 ビット整数
D	signed long long	符号付き 64 ビット整数
UB	unsigned char	符号なし 8 ビット整数
UH	unsigned short	符号なし 16 ビット整数
UW	unsigned long	符号なし 32 ビット整数
UD	unsigned long long	符号なし 64 ビット整数
VB	signed char	データ・タイプが一定しない値（8 ビット）
VH	signed short	データ・タイプが一定しない値（16 ビット）
VW	signed long	データ・タイプが一定しない値（32 ビット）
VD	signed long long	データ・タイプが一定しない値（64 ビット）
VP	void *	データ・タイプが一定しない値（ポインタ）
FP	void（*）	処理プログラムの起動アドレス（ポインタ）
INT	signed long	符号付き 32 ビット整数
UINT	unsigned long	符号なし 32 ビット整数
BOOL	signed long	真偽値（TRUE または FALSE）
ER	signed long	エラー・コード
ID	signed short	オブジェクト ID
ATR	unsigned short	オブジェクト属性
STAT	unsigned short	オブジェクトの状態
MODE	unsigned short	サービス・コールの動作モード
PRI	signed short	タスクまたはメッセージの優先度
SIZE	unsigned long	領域のサイズ（単位：バイト）
TMO	signed long	タイムアウト（単位：ミリ秒）

マクロ	型	意味
RELTIM	unsigned long	相対時間（単位：ミリ秒）
VP_INT	signed long	データ･タイプが一定しない値(ポインタ), または符号付き32ビット整数
ER_UINT	signed long	エラー・コード，または符号なし 32 ビット整数
FLGPTN	unsigned long	イベントフラグのビット・パターン
IMASK	unsigned short	割り込みマスクレベル

17.2　マ ク ロ

RI600V4 が提供するサービス・コールを発行する際に使用するマクロ（オブジェクトの現在状態，処理プログラムの属性など）について以下に示します。

17.2.1　定数マクロ

以下に，定数マクロ一覧を示します。
なお，定数マクロの定義は，以下のいずれかで行われています。

- <ri_root>¥inc600¥kernel.h
- kernel.h から呼び出される <ri_root>¥inc600¥itron.h
- cfg600 が出力するシステム情報ヘッダファイル kernel_id.
  本ファイルの内容は，システム・コンフィギュレーション・ファイルの定義内容によって変化します。

表 17 － 2　定数マクロ

分類	マクロ	定義内容	定義場所	説明
一般	NULL	0	itron.h	無効ポインタ
	TRUE	1	itron.h	真
	FALSE	0	itron.h	偽
	E_OK	0	itron.h	正常終了

分類	マクロ	定義内容	定義場所	説明
属性	TA_NULL	0	itron.h	オブジェクト属性を指定しない
	TA_TFIFO	0x0000	kernel.h	タスクの待ち行列は FIFO 順
	TA_TPRI	0x0001	kernel.h	タスクの待ち行列はタスクの現在優先度順。ただし，同じ現在優先度のタスクの中では FIFO 順。
	TA_MFIFO	0x0000	kernel.h	メッセージ・キューは FIFO 順
	TA_MPRI	0x0002	kernel.h	メッセージ・キューはメッセージ優先度順。ただし，同じメッセージ優先度のメッセージの中では FIFO 順。
	TA_ACT	0x0002	kernel.h	タスクを生成と同時に起動
	TA_WSGL	0x0000	kernel.h	イベント・フラグに複数タスクの待ちを許さない
	TA_WMUL	0x0002	kernel.h	イベント・フラグに複数タスクの待ちを許す
	TA_CLR	0x0004	kernel.h	待ち解除時にイベント・フラグをクリア
	TA_CEILING	0x0003	kernel.h	優先度上限プロトコル
	TA_STA	0x0002	kernel.h	周期ハンドラを動作状態で生成
	TA_PHS	0x0004	kernel.h	周期ハンドラ位相を保存
タイムアウト	TMO_POL	0	itron.h	ポーリング
	TMO_FEVR	-1	itron.h	永久待ち
動作モード	TWF_ANDW	0x0000	kernel.h	イベント・フラグの AND 待ち
	TWF_ORW	0x0001	kernel.h	イベント・フラグの OR 待ち

分類	マクロ	定義内容	定義場所	説明
状態	TTS_RUN	0x0001	kernel.h	RUNNING 状態
	TTS_RDY	0x0002	kernel.h	READY 状態
	TTS_WAI	0x0004	kernel.h	WAITING 状態
	TTS_SUS	0x0008	kernel.h	SUSPENDED 状態
	TTS_WAS	0x000C	kernel.h	WAITING-SUSPENDED 状態
	TTS_DMT	0x0010	kernel.h	DORMANT 状態
	TTW_SLP	0x0001	kernel.h	起床待ち状態
	TTW_DLY	0x0002	kernel.h	時間経過待ち状態
	TTW_SEM	0x0004	kernel.h	セマフォ資源獲得待ち状態
	TTW_FLG	0x0008	kernel.h	イベント・フラグ待ち状態
	TTW_SDTQ	0x0010	kernel.h	データ・キューへの送信待ち状態
	TTW_RDTQ	0x0020	kernel.h	データ・キューからの受信待ち状態
	TTW_MBX	0x0040	kernel.h	メールボックスからの受信待ち状態
	TTW_MTX	0x0080	kernel.h	ミューテックス待ち状態
	TTW_SMBF	0x0100	kernel.h	メッセージ・バッファへの送信待ち状態
	TTW_RMBF	0x0200	kernel.h	メッセージ・バッファからの受信待ち状態
	TTW_MPF	0x2000	kernel.h	固定長メモリ・ブロック獲得待ち待ち状態
	TTW_MPL	0x4000	kernel.h	可変長メモリブ・ロック獲得待ち待ち状態
	TCYC_STP	0x0000	kernel.h	周期ハンドラ非動作状態
	TCYC_STA	0x0001	kernel.h	周期ハンドラ動作状態
	TALM_STP	0x0000	kernel.h	アラーム・ハンドラ非動作状態
	TALM_STA	0x0001	kernel.h	アラーム・ハンドラ動作状態
その他	TSK_SELF	0	kernel.h	自タスク指定
	TSK_NONE	0	kernel.h	該当するタスクがない
	TPRI_SELF	0	kernel.h	自タスクのベース優先度の指定
	TPRI_INI	0	kernel.h	タスクの起動時優先度の指定

分類	マクロ	定義内容	定義場所	説明
カーネル構成	TMIN_TPRI	1	kernel.h	タスク優先度の最小値
	TMAX_TPRI	system.priority	kernel_id.h	タスク優先度の最大値
	TMIN_MPRI	1	kernel.h	メッセージ優先度の最小値
	TMAX_MPRI	system.message_pri	kernel_id.h	メッセージ優先度の最大値
	TKERNEL_MAKER	0x011B	kernel.h	カーネルのメーカー・コード
	TKERNEL_PRID	0x0003	kernel.h	カーネルの識別番号
	TKERNEL_SPVER	0x5403	kernel.h	ITRON 仕様のバージョン番号
	TKERNEL_PRVER	0x0130	kernel.h	カーネルのバージョン番号
	TMAX_ACTCNT	255	kernel.h	タスク起動要求キューイング数の最大値
	TMAX_WUPCNT	255	kernel.h	タスク起床要求キューイング数の最大値
	TMAX_SUSCNT	1	kernel.h	タスク強制待ち要求ネスト数の最大値
	TBIT_FLGPTN	32	kernel.h	イベントフラグのビット数
	TIC_NUME	system.tic_nume	kernel_id.h	基本クロック割り込み周期の分子
	TIC_DENO	system.tic_deno	kernel_id.h	基本クロック割り込み周期の分母
	TMAX_MAXSEM	65535	kernel.h	セマフォの最大資源数の最大値
	VTMAX_TSK	task[] 定義数	kernel_id.h	最大タスク ID
	VTMAX_SEM	semaphore[] 定義数	kernel_id.h	最大セマフォ ID
	VTMAX_FLG	flag[] 定義数	kernel_id.h	最大イベントフラグ ID
	VTMAX_DTQ	dataqueue[] 定義数	kernel_id.h	最大データ・キュー ID
	VTMAX_MBX	mailbox[] 定義数	kernel_id.h	最大メールボックス ID
	VTMAX_MTX	mutex[] 定義数	kernel_id.h	最大ミューテックス ID
	VTMAX_MBF	message_buffer[] 定義数	kernel_id.h	最大メッセージ・バッファ ID
	VTMAX_MPF	memorypool[] 定義数	kernel_id.h	最大固定長メモリ・プール ID
	VTMAX_MPL	variable_memorypool[] 定義数	kernel_id.h	最大可変長メモリ・プール ID
	VTMAX_CYH	cyclic_hand[] 定義数	kernel_id.h	最大周期ハンドラ ID
	VTMAX_ALH	alarm_hand[] 定義数	kernel_id.h	最大アラーム・ハンドラ ID
	VTSZ_MBFTBL	4	kernel.h	メッセージ・バッファのメッセージ管理テーブルのサイズ (単位 : バイト)
	VTMAX_AREASIZE	0x10000000	kernel.h	各種領域サイズの最大値 (単位 : バイト)
	VTKNL_LVL	system.system_IPL	kernel_id.h	カーネル割り込みマスクレベル
	VTIM_LVL	clock.IPL	kernel_id.h	基本クロック用タイマ割り込み優先レベル

分類	マクロ	定義内容	定義場所	説明
エラー・コード	E_NOSPT	-9	itron.h	未サポート機能
	E_PAR	-17	itron.h	パラメータ・エラー
	E_ID	-18	itron.h	不正 ID 番号
	E_CTX	-25	itron.h	コンテキスト・エラー
	E_ILUSE	-28	itron.h	サービス・コール不正使用
	E_OBJ	-41	itron.h	オブジェクト状態エラー
	E_QOVR	-43	itron.h	キューイング・オーバフロー
	E_RLWAI	-49	itron.h	待ち状態の強制解除
	E_TMOUT	-50	itron.h	ポーリング失敗またはタイムアウト
	EV_RST	-127	itron.h	オブジェクト・リセットによる待ち解除

17.2.2　関数マクロ

以下に，関数マクロ一覧を示します。
なお，関数マクロの定義は，<ri_root>¥inc600¥itron.h で行われています。

1) ER MERCD (ER *ercd*)
ercd のメイン・エラー・コードを返します。

2) ER SERCD (ER *ercd*)
ercd のサブ・エラー・コードを返します。

3) ER ERCD (ER *mercd*, ER *sercd*)
mercd のメイン・エラー・コードと *sercd* のサブ・エラー・コードからなるエラー・コードを返します。

備考　RI600V4 のサービス・コールが返すエラー・コードのサブ・エラー・コードはすべて -1，メイン・エラー・コードは表 17 － 2 に記載の値です。

第 18 章　サービス・コール

本章では，RI600V4 が提供しているサービス・コールについて解説しています。

18.1　概　　要

RI600V4 が提供しているサービス・コールは，ユーザが記述した処理プログラムから RI600V4 が管理している資源（タスク，セマフォなど）を操作するために用意されたサービス・ルーチンです。
以下に，RI600V4 が提供しているサービス・コールを管理モジュール別に示します。

- タスク管理機能

act_tsk	iact_tsk	can_act	ican_act
sta_tsk	ista_tsk	ext_tsk	ter_tsk
chg_pri	ichg_pri	get_pri	iget_pri
ref_tsk	iref_tsk	ref_tst	iref_tst

- タスク付属同期機能

slp_tsk	tslp_tsk	wup_tsk	iwup_tsk
can_wup	ican_wup	rel_wai	irel_wai
sus_tsk	isus_tsk	rsm_tsk	irsm_tsk
frsm_tsk	ifrsm_tsk	dly_tsk	

- 同期通信機能（セマフォ）

wai_sem	pol_sem	ipol_sem	twai_sem
sig_sem	isig_sem	ref_sem	iref_sem

- 同期通信機能（イベントフラグ）

set_flg	iset_flg	clr_flg	iclr_flg
wai_flg	pol_flg	ipol_flg	twai_flg
ref_flg	iref_flg		

- 同期通信機能（データ・キュー）

snd_dtq	psnd_dtq	ipsnd_dtq	tsnd_dtq
fsnd_dtq	ifsnd_dtq	rcv_dtq	prcv_dtq
iprcv_dtq	trcv_dtq	ref_dtq	iref_dtq

- 同期通信機能（メールボックス）

snd_mbx	isnd_mbx	rcv_mbx	prcv_mbx
iprcv_mbx	trcv_mbx	ref_mbx	iref_mbx

- 拡張同期通信機能（ミューテックス）

loc_mtx	ploc_mtx	tloc_mtx	unl_mtx
ref_mtx			

- 拡張同期通信機能（メッセージ・バッファ）

snd_mbf	psnd_mbf	ipsnd_mbf	tsnd_mbf
rcv_mbf	prcv_mbf	trcv_mbf	ref_mbf
iref_mbf			

- メモリ・プール管理機能（固定長メモリ・プール）

get_mpf	pget_mpf	ipget_mpf	tget_mpf
rel_mpf	irel_mpf	ref_mpf	iref_mpf

- メモリ・プール管理機能（可変長メモリ・プール）

get_mpl	pget_mpl	ipget_mpl	tget_mpl
rel_mpl	ref_mpl	iref_mpl	

- 時間管理機能

set_tim	iset_tim	get_tim	iget_tim
sta_cyc	ista_cyc	stp_cyc	istp_cyc
ref_cyc	iref_cyc	sta_alm	ista_alm
stp_alm	istp_alm	ref_alm	iref_alm

- システム状態管理機能

rot_rdq	irot_rdq	get_tid	iget_tid
loc_cpu	iloc_cpu	unl_cpu	iunl_cpu
dis_dsp	ena_dsp	sns_ctx	sns_loc
sns_dsp	sns_dpn	vsys_dwn	ivsys_dwn
vsta_knl	ivsta_knl		

- 割り込み管理機能

chg_ims	ichg_ims	get_ims	iget_ims

- システム構成管理機能

ref_ver	iref_ver

- オブジェクト・リセット機能

vrst_dtq	vrst_mbx	vrst_mbf	vrst_mpf
vrst_mpl			

18.1.1 サービス・コールの呼び出し方法

サービス・コールは，通常の C 言語関数と同様の方法で呼び出します。

備考 RI600V4 が提供するサービス・コールを処理プログラムから発行する場合，以下に示したヘッダ・ファイルの定義（インクルード処理）を行う必要があります。

- kernel.h：標準ヘッダ・ファイル
- kernel_id.h：cfg600 が出力するシステム情報ヘッダ・ファイル

18.2　サービス・コール解説

次項から RI600V4 が提供しているサービス・コールについて，以下の記述フォーマットにしたがって解説します。

1）

2）→ **概要**

3）→ **C 言語形式**

4）→ **パラメータ**

I/O	パラメータ	説明

5）→ **機能**

6）→ **戻り値**

マクロ	数値	意味

1） 名称
サービス・コールの名称を示しています。

2） 概要
サービス・コールの機能概要を示しています。

3） C 言語形式
サービス・コールを C 言語で記述された処理プログラムから発行する際の記述形式を示しています。

4） パラメータ
サービス・コールのパラメータを以下の形式で示しています。

I/O	パラメータ	説明
A	B	C

A） パラメータの種類
I： RI600V4 への入力パラメータ
O： RI600V4 からの出力パラメータ

B） パラメータのデータ・タイプ

C） パラメータの説明

5） 機能
サービス・コールの機能詳細を示しています。

6） 戻り値
サービス・コールからの戻り値を以下の形式で示しています。

マクロ	数値	意味
A	B	C

A） 戻り値のマクロ

B） 戻り値の数値

C） 戻り値の意味

18.2.1　タスク管理機能

以下に，RI600V4 がタスク管理機能として提供しているサービス・コールの一覧を示します。

表 18 － 1　タスク管理機能

サービス・コール名	機能概要	発行有効範囲
act_tsk	タスクの起動（起動要求をキューイングする）	タスク
iact_tsk	タスクの起動（起動要求をキューイングする）	非タスク
can_act	起動要求のキューイング解除	タスク
ican_act	起動要求のキューイング解除	非タスク
sta_tsk	タスクの起動（起動要求をキューイングしない）	タスク
ista_tsk	タスクの起動（起動要求をキューイングしない）	非タスク
ext_tsk	自タスクの終了	タスク
ter_tsk	タスクの強制終了	タスク
chg_pri	タスク優先度の変更	タスク
ichg_pri	タスク優先度の変更	非タスク
get_pri	タスク現在優先度の参照	タスク
iget_pri	タスク現在優先度の参照	非タスク
ref_tsk	タスク詳細情報の参照	タスク
iref_tsk	タスク詳細情報の参照	非タスク
ref_tst	タスク基本情報の参照	タスク
iref_tst	タスク基本情報の参照	非タスク

act_tsk
iact_tsk

概要

タスクの起動（起動要求をキューイングする）

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RI600V4 のスケジューリング対象となります。また，このとき以下に示す処理が行われます。

表 18 － 2 タスク起動時に行われる処理

No.	処理内容
1	ベース優先度と現在優先度を，初期優先度にする。
2	起床要求キューイング数をクリアする。
3	強制待ち要求ネスト数をクリアする。

ただし，本サービス・コールを発行した際，対象タスクが DORMANT 状態以外の場合には，対象タスクのキューイング処理，および，状態操作処理は行わず，対象タスクに起動要求をキューイング（起動要求カウンタに 1 を加算）しています。

備考 1 RI600V4 が管理する起動要求カウンタは，8 ビット幅で構成されています。このため，本サービス・コールでは，起動要求数が 255 回を超える場合には，起動要求の発行（起動要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

備考 2 本サービス・コールの発行により起動されたタスクには，拡張情報として “ タスク情報（task[]）で指定した拡張情報 ” が渡されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - iact_tsk を発行した際，*tskid* に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから iact_tsk を発行した。 - 非タスクから act_tsk を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_QOVR	-43	キューイング・オーバフロー - 起動要求数が 255 回を超えた。

can_act
ican_act

概要

起動要求のキューイング解除

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID

機能

tskid で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0 を設定）します。
なお，正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。

備考 本サービス・コールでは，状態操作処理は行わず，起動要求カウンタの設定処理のみを行います。
したがって，READY 状態などから DORMANT 状態に遷移することはありません。

戻り値

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ican_act を発行した場合，および非タスクから can_act を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

マクロ	数値	意味
―	0	正常終了 - 起動要求数が 0 である。 - 対象タスクが DORMANT 状態である。
―	正の値	正常終了（解除した起動要求数）

sta_tsk
ista_tsk

概要

タスクの起動（起動要求をキューイングしない）

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID
I	VP_INT stacd;	タスクの起動コード

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RI600V4 のスケジューリング対象となります。また，このとき表 18 － 2 に示す処理が行われます。

ただし，本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスクが DORMANT 状態以外の場合には，対象タスクの状態操作処理は行わず，戻り値として E_OBJ を返します。

対象タスクには，*stacd* が引き渡されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ≦ 0 - *tskid* ＞ VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ista_tsk を発行した。 - 非タスクから sta_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_OBJ	-41	対象タスクが DORMANT 状態でない。

ext_tsk

概要

自タスクの終了

C 言語形式

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

パラメータ

なし

機能

自タスクを RUNNING 状態から DORMANT 状態へと遷移させ，レディ・キューから外します。これにより，自タスクは，RI600V4 のスケジューリング対象から除外されます。また，このとき以下に示す処理が行われます。

表 18 － 3　タスク終了時に行われる処理

No	処理内容
1	終了するタスクがロックしていたミューテックスをロック解除する。（unl_mtx と同等の処理）

また，CPU ロック状態とディスパッチ禁止状態は解除されます。

本サービス・コールを発行した際，自タスクの起動要求がキューイングされていた（起動要求カウンタ＞ 0）場合には，自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。このとき表 18 － 2 に示す処理が行われます。

本サービス・コールは，呼び出し元には戻りません。以下の場合，システム・ダウンとなります。

- 非タスクから本サービス・コールを発行した。
- 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

備考 1　タスク開始関数で return 命令が発行された場合，本サービス・コールと同等の処理が行われます。

備考 2　本サービス・コールは，タスクが占有していたミューテックス以外の資源（セマフォやメモリ・ブロックなど）を自動的に解放する機能はありません。タスクは，必ず終了する前に資源の解放を行ってください。

戻り値

なし

ter_tsk

概要

タスクの強制終了

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID　　tskid;	タスクの ID

機能

tskid で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより，対象タスクは，RI600V4 のスケジューリング対象から除外されます。また，このとき表 18 － 3 に示す処理が行われます。

本サービス・コールを発行した際，対象タスクの起動要求がキューイングされていた（起動要求カウンタ＞ 0）場合には，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。このとき表 18 － 2 に示す処理が行われます。

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

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ≦ 0 - *tskid* ＞ VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - 対象タスクが自タスクである。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。

chg_pri
ichg_pri

概要

タスク優先度の変更

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID
I	PRI tskpri;	タスクのベース優先度 TPRI_INI： タスクの初期優先度 数値： タスクのベース優先度

機能

tskid で指定されたタスクのベース優先度を *tskpri* で指定された値に変更します。

変更されたタスクのベース優先度は，タスクが終了，または本サービス・コールを呼び出すまで有効です。次回のタスク起動時，タスクのベース優先度はタスク生成時に指定した初期タスク優先度になります。

本サービス・コールは，対象タスクの現在優先度も，*tskpri* で示された値に変更します。ただし，対象タスクがミューテックスをロックしている場合は，現在優先度は変更しません。

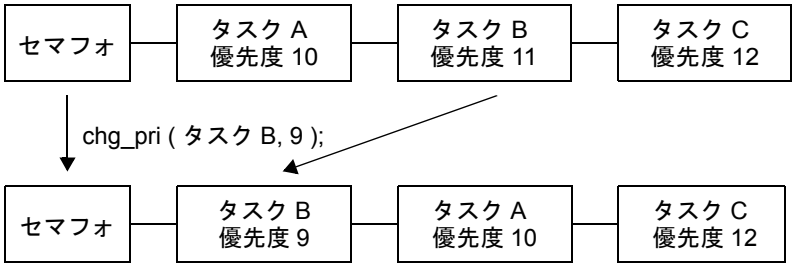
対象タスクがミューテックスをロックしているかロックを待っている場合で，*tskpri* がそれらのミューテックスのいずれかの上限優先度よりも高い場合には，戻り値として E_ILUSE を返します。

現在優先度が変更された場合，以下の状態変化が生じます。

1） 対象タスクが RUNNING 状態または READY 状態の場合
本サービス・コールは，対象タスクを *tskpri* で指定された優先度に応じたレディ・キューの最後尾につなぎかえます。

2） 対象タスクが TA_TPRI 属性または TA_CEILING 属性のオブジェクトの待ちキューにキューイングされている場合
本サービス・コールは，*tskpri* で指定された優先度にしたがって対象タスクを待ちキューにつなぎかえます。待ちキューに *tskpri* で指定された現在優先度のタスクが複数ある場合は，対象タスクをそれらの中の最後尾につなぎかえます。

例 セマフォの待ちキューに 3 つのタスク（タスク A：優先度 10，タスク B：優先度 11，タスク C：優先度 12）が優先度順でキューイングされているとき，タスク B の優先度を 11 から 9 に変更した場合，待ちキューの順序は，以下のように変更されます。



備考　現在優先度とベース優先度については，「6.2.2　現在優先度とベース優先度」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tskpri* ＜ 0 - *tskpri* ＞ TMAX_TPRI
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - ichg_pri を発行した際，*tskid* に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ichg_pri を発行した。 - 非タスクから chg_pri を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - *tskpri* ＜対象タスクがロックしているミューテックスの上限優先度 - *tskpri* ＜対象タスクがロックを待っているミューテックスの上限優先度
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。

get_pri
iget_pri

概要

タスク現在優先度の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID
O	PRI *p_tskpri;	現在優先度を格納する領域へのポインタ

機能

tskid で指定されたタスクの現在優先度を *p_tskpri* で指定された領域に格納します。

備考 現在優先度とベース優先度については，「6.2.2 現在優先度とベース優先度」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - 非タスクから本サービス・コールを発行した際，*tskid*にTSK_SELFを指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから iget_pri を発行した場合，および非タスクから get_pri を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

マクロ	数値	意味
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。

ref_tsk
iref_tsk

概要

タスク詳細情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	`ID tskid;`	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID
O	`T_RTSK *pk_rtsk;`	タスク詳細情報を格納する領域へのポインタ

【 タスク詳細情報 T_RTSK の構造 】

```
typedef struct  t_rtsk {
        STAT    tskstat;        /* 現在状態 */
        PRI     tskpri;         /* 現在優先度 */
        PRI     tskbpri;        /* ベース優先度 */
        STAT    tskwait;        /* 待ち要因 */
        ID      wobjid;         /* 待ちオブジェクト ID*/
        TMO     lefttmo;        /* タイムアウトするまでの時間 */
        UINT    actcnt;         /* 起動要求数 */
        UINT    wupcnt;         /* 起床要求数 */
        UINT    suscnt;         /* サスペンド要求数 */
} T_RTSK;
```

機能

tskid で指定されたタスクのタスク詳細情報（現在状態，現在優先度など）を *pk_rtsk* で指定された領域に格納します。

- *tskstat*
  タスクの現在状態が格納されます。

  TTS_RUN： RUNNING 状態
  TTS_RDY： READY 状態
  TTS_WAI： WAITING 状態
  TTS_SUS： SUSPENDED 状態
  TTS_WAS： WAITING-SUSPENDED 状態
  TTS_DMT： DORMANT 状態

- *tskpri*
  タスクの現在優先度が格納されます。
  tskpri は，*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *tskbpri*
  タスクのベース優先度が格納されます。
  tskbpri は，*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *tskwait*
  タスクの待ち要因（WAITING 状態の種類）が格納されます。
  tskwait は，*tskstat* が TTS_WAI または TTS_WAS の場合のみ有効です。

TTW_SLP：	slp_tsk または tslp_tsk による起床待ち状態
TTW_DLY：	dly_tsk による時間経過待ち状態
TTW_SEM：	wai_sem または twai_sem による資源獲得待ち状態
TTW_FLG：	wai_flg または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ：	snd_dtq または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ：	rcv_dtq または trcv_dtq によるデータ受信待ち状態
TTW_MBX：	rcv_mbx または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX：	loc_mtx または tloc_mtx によるミューテックス待ち状態
TTW_SMBF：	snd_mbf または tsnd_mbf によるメッセージ送信待ち状態
TTW_RMBF：	rcv_mbf または trcv_mbf によるメッセージ受信待ち状態
TTW_MPF：	get_mpf または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL：	get_mpl または tget_mpl による可変長メモリ・ブロック獲得待ち状態

- *wobjid*
  待ち対象のオブジェクト（セマフォ，イベントフラグなど）の ID が格納されます。
  wobjid は，*tskwait* が TTW_SEM，TTW_FLG，TTW_SDTQ，TTW_RDTQ，TTW_MBX，TTW_MTX，TTW_SMBF，TTW_RMBF，TTW_MPF，または TTW_MPL の場合のみ有効です。

- *lefttmo*
  タイムアウトまでの残り時間（単位：ミリ秒）が格納されます。永久待ちの場合は，TMO_FEVR が格納されます。
  lefttmo は，*tskstat* が TTS_WAI または TTS_WAS の場合で，かつ *tskwait* が TTW_DLY 以外の場合のみ有効です。

  備考　TTW_DLY の場合は，lefttmo は不定となります。

- *actcnt*
  タスクの起動要求数が格納されます。

- *wupcnt*
  タスクの起床要求数が格納されます。
  wupcnt は，*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *suscnt*
  タスクのサスペンド要求数が格納されます。
  suscnt は，*tskstat* が TTS_DMT 以外の場合のみ有効です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - 非タスクから本サービス・コールを発行した際，*tskid*にTSK_SELFを指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_tsk を発行した場合，および非タスクから ref_tsk を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

ref_tst
iref_tst

概要

タスク基本情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	`ID tskid;`	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID
O	`T_RTST *pk_rtst;`	タスク基本情報を格納する領域へのポインタ

【 タスク基本情報 T_RTST の構造 】

```
typedef struct  t_rtst {
        STAT    tskstat;        /* 現在状態 */
        STAT    tskwait;        /* 待ち要因 */
} T_RTST;
```

機能

tskid で指定されたタスクのタスク基本情報（現在状態，待ち要因）を *pk_rtst* で指定された領域に格納します。
タスク情報のうち，現在状態，待ち要因のみを参照したい場合に使用します。
取得する情報が少ないので ref_tsk，iref_tsk より高速に応答します。

- *tskstat*
  タスクの現在状態が格納されます。

  TTS_RUN： RUNNING 状態
  TTS_RDY： READY 状態
  TTS_WAI： WAITING 状態
  TTS_SUS： SUSPENDED 状態
  TTS_WAS： WAITING-SUSPENDED 状態
  TTS_DMT： DORMANT 状態

- *tskwait*
  タスクの待ち要因（WAITING 状態の種類）が格納されます。
  tskwait は，*tskstat* が TTS_WAI または TTS_WAS の場合のみ有効です。

  TTW_SLP： slp_tsk または tslp_tsk による起床待ち状態

TTW_DLY : dly_tsk による時間経過待ち状態
TTW_SEM : wai_sem または twai_sem による資源獲得待ち状態
TTW_FLG : wai_flg または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ : snd_dtq または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ : rcv_dtq または trcv_dtq によるデータ受信待ち状態
TTW_MBX : rcv_mbx または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX : loc_mtx または tloc_mtx によるミューテックス待ち状態
TTW_SMBF : snd_mbf または tsnd_mbf によるメッセージ送信待ち状態
TTW_RMBF : rcv_mbf または trcv_mbf によるメッセージ受信待ち状態
TTW_MPF : get_mpf または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL : get_mpl または tget_mpl による可変長メモリ・ブロック獲得待ち状態

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - 非タスクから本サービス・コールを発行した際, *tskid*にTSK_SELFを指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_tst を発行した場合，および非タスクから ref_tst を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.2　タスク付属同期機能

以下に，RI600V4 がタスク付属同期機能として提供しているサービス・コールの一覧を示します。

表 18 － 4　タスク付属同期機能

サービス・コール名	機能概要	発行有効範囲
slp_tsk	起床待ち状態への移行（永久待ち）	タスク
tslp_tsk	起床待ち状態への移行（タイムアウト付き）	タスク
wup_tsk	タスクの起床	タスク
iwup_tsk	タスクの起床	非タスク
can_wup	起床要求の解除	タスク
ican_wup	起床要求の解除	非タスク
rel_wai	WAITING 状態の強制解除	タスク
irel_wai	WAITING 状態の強制解除	非タスク
sus_tsk	SUSPENDED 状態への移行	タスク
isus_tsk	SUSPENDED 状態への移行	非タスク
rsm_tsk	SUSPENDED 状態の解除	タスク
irsm_tsk	SUSPENDED 状態の解除	非タスク
frsm_tsk	SUSPENDED 状態の強制解除	タスク
ifrsm_tsk	SUSPENDED 状態の強制解除	非タスク
dly_tsk	時間経過待ち状態への移行	タスク

slp_tsk

概要

起床待ち状態への移行（永久待ち）

C 言語形式

```
ER      slp_tsk ( void );
```

パラメータ

なし

機能

自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。
ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタ＞ 0）場合には，状態操作処理は行わず，起床要求カウンタから 1 を減算します。
なお，起床待ち状態の解除は，以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により，起床要求が発行された。	E_OK
iwup_tsk の発行により，起床要求が発行された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

tslp_tsk

概要

起床待ち状態への移行（タイムアウト付き）

C 言語形式

```
ER      tslp_tsk ( TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

自タスクを RUNNING 状態からタイムアウト付きの WAITING 状態（起床待ち状態）へと遷移させます。
ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタ＞ 0）場合には，状態操作処理は行わず，起床要求カウンタから 1 を減算します。
なお，起床待ち状態の解除は，以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により，起床要求が発行された。	E_OK
iwup_tsk の発行により，起床要求が発行された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考　待ち時間 *tmout* に TMO_FEVR が指定された際には “slp_tsk と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

wup_tsk
iwup_tsk

概要

タスクの起床

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID

機能

tskid で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。

ただし，本サービス・コールを発行した際，対象タスクが起床待ち状態以外の場合には，状態操作処理は行わず，起床要求カウンタに 1 を加算します。

備考 RI600V4 が管理する起床要求カウンタは，8 ビット幅で構成されています。このため，本サービス・コールでは，起床要求数が 255 回を超える場合には，起床要求のキューイング（起床要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - iwup_tsk を発行した際，*tskid* に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから iwup_tsk を発行した。 - 非タスクから wup_tsk を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。
E_QOVR	-43	キューイング・オーバフロー - 起床要求数が 255 回を超えた。

can_wup
ican_wup

概要

起床要求の解除

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID

機能

tskid で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0 を設定）し，戻り値として解除した起床要求数を返します。

戻り値

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - 非タスクから本サービス・コールを発行した際，*tskid*にTSK_SELFを指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ican_wup を発行した場合，および非タスクから can_wup を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。
―	0 以上	正常終了（解除した起床要求数）

rel_wai
irel_wai

概要

WAITING 状態の強制解除

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	`ID tskid;`	タスクの ID

機能

tskid で指定されたタスクの WAITING 状態を強制的に解除します。これにより，対象タスクは待ちキューから外れ，WAITING 状態から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお，本サービス・コールの発行により WAITING 状態を解除されたタスクには，WAITING 状態へと遷移するきっかけとなったサービス・コール（slp_tsk，wai_sem など）の戻り値として E_RLWAI を返します。

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが WAITING 状態でも WAITING-SUSPENDED 状態でもない場合には，戻り値として E_OBJ を返します。

備考 2　本サービス・コールでは，SUSPENDED 状態の解除は行われません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ≦ 0 - *tskid* ＞ VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irel_wai を発行した。 - 非タスクから rel_wai を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが，WAITING 状態でも WAITING-SUSPENDED 状態でもない。

sus_tsk
isus_tsk

概要

SUSPENDED 状態への移行

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID TSK_SELF： 自タスク 数値： タスクの ID

機能

tskid で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ，READY 状態から SUSPENDED 状態へ，または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。

ただし，本サービス・コールを発行した際，対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態へと遷移していた場合には，戻り値として E_QOVR を返します。

備考 RI600V4 では，サスペンド要求のネストはできません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ＜ 0 - *tskid* ＞ VTMAX_TSK - isus_tsk を発行した際，*tskid* に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから isus_tsk を発行した。 - 非タスクから sus_tsk を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 - ディスパッチ禁止状態で，自タスクを指定した。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。 - ディスパッチ禁止状態のときに，isus_tsk で *tskid* に実行状態のタスクを指定
E_QOVR	-43	キューイング・オーバフロー - 対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態である。

rsm_tsk
irsm_tsk

概要

SUSPENDED 状態の解除

C 言語形式

```
ER      rsm_tsk ( ID tskid );
ER      irsm_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	`ID tskid;`	タスクの ID

機能

tskid で指定されたタスクを SUSPENDED 状態から READY 状態へ, または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

備考 1 本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない場合には，戻り値として E_OBJ を返します。

備考 2 RI600V4 では，サスペンド要求のキューイング機能はサポートしていません。サスペンド要求のキューイングも含めて SUSPEND 要求を解除する frsm_tsk，ifrsm_tsk もサポートしていますが，これらの振る舞いは rsm_tsk，irsm_tsk と同じです。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ≦ 0 - *tskid* ＞ VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irsm_tsk を発行した。 - 非タスクから rsm_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない。

frsm_tsk
ifrsm_tsk

概要

SUSPENDED 状態の強制解除

C 言語形式

```
ER      frsm_tsk ( ID tskid );
ER      ifrsm_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID tskid;	タスクの ID

機能

tskid で指定されたタスクを SUSPENDED 状態から READY 状態へ, または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

備考 1 本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない場合には，戻り値として E_OBJ を返します。

備考 2 RI600V4 では，サスペンド要求のネストはできません。このため，本サービス・コールは rsm_tsk，irsm_tsk とまったく同じ処理を行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *tskid* ≦ 0 - *tskid* ＞ VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ifrsm_tsk を発行した。 - 非タスクから frsm_tsk を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない。

dly_tsk

概要

時間経過待ち状態への移行

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	RELTIM　dlytim;	遅延時間（単位：ミリ秒）

機能

自タスクを *dlytim* で指定された遅延時間が経過するまでの間，RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
なお，時間経過待ち状態の解除は，以下の場合に行われます。

時間経過待ち状態の解除操作	戻り値
dlytim で指定された遅延時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

備考　*dlytim* に 0 を指定すると，次回の基本クロック割り込み発生までが遅延時間となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *dlytim* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

18.2.3　同期通信機能（セマフォ）

以下に，RI600V4 が同期通信機能（セマフォ）として提供しているサービス・コールの一覧を示します。

表 18 － 5　同期通信機能（セマフォ）

サービス・コール名	機能概要	発行有効範囲
wai_sem	資源の獲得	タスク
pol_sem	資源の獲得（ポーリング）	タスク
ipol_sem	資源の獲得（ポーリング）	非タスク
twai_sem	資源の獲得（タイムアウト付き）	タスク
sig_sem	資源の返却	タスク
isig_sem	資源の返却	非タスク
ref_sem	セマフォ詳細情報の参照	タスク
iref_sem	セマフォ詳細情報の参照	非タスク

wai_sem

概要

資源の獲得

C 言語形式

```
ER      wai_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID semid;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお，資源獲得待ち状態の解除は，以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により，対象セマフォに資源が返却された。	E_OK
isig_sem の発行により，対象セマフォに資源が返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

備考　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *semid* ≦ 0 - *semid* ＞ VTMAX_SEM

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

pol_sem
ipol_sem

概要

資源の獲得（ポーリング）

C 言語形式

```
ER      pol_sem ( ID semid );
ER      ipol_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID semid;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *semid* ≦ 0 - *semid* ＞ VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ipol_sem を発行した場合，および非タスクから pol_sem を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_TMOUT	-50	ポーリング失敗

twai_sem

概要

資源の獲得（タイムアウト付き）

C 言語形式

```
ER      twai_sem ( ID semid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID semid;	セマフォの ID
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお，資源獲得待ち状態の解除は，以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により，対象セマフォに資源が返却された。	E_OK
isig_sem の発行により，対象セマフォに資源が返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
tmout で指定された待ち時間が経過した。	E_TMOUT

備考 1 自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2 待ち時間 *tmout* に TMO_FEVR が指定された際には “wai_sem と同等の処理 ” を，TMO_POL が指定された際には “pol_sem と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *semid* ≦ 0 - *semid* ＞ VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

sig_sem
isig_sem

概要

資源の返却

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID semid;	セマフォの ID

機能

semid で指定されたセマフォに資源を返却（セマフォ・カウンタに 1 を加算）します。

ただし，本サービス・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（資源獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　RI600V4 では，セマフォの資源数として取り得る最大値（最大資源数（max_count））をコンフィギュレーション時に定義させています。このため，本サービス・コールでは，資源数が最大資源数を超える場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *semid* ≦ 0 - *semid* ＞ VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから isig_sem を発行した。 - 非タスクから sig_sem を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_QOVR	-43	キューイング・オーバフロー - 資源数が最大資源数（max_count）を超えた。

ref_sem
iref_sem

概要

セマフォ詳細情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID semid;	セマフォの ID
O	T_RSEM *pk_rsem;	セマフォ詳細情報を格納する領域へのポインタ

【 セマフォ詳細情報 T_RSEM の構造 】

```
typedef struct  t_rsem {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    semcnt;         /* 現在資源数 */
} T_RSEM;
```

機能

semid で指定されたセマフォのセマフォ詳細情報（待ちタスクの有無，現在資源数など）を *pk_rsem* で指定された領域に格納します。

- *wtskid*
  セマフォの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

- *semcnt*
  セマフォの現在資源数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *semid* ≦ 0 - *semid* ＞ VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_sem を発行した場合，および非タスクから ref_sem を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.4　同期通信機能（イベントフラグ）

以下に，RI600V4 が同期通信機能（イベントフラグ）として提供しているサービス・コールの一覧を示します。

表 18 － 6　同期通信機能（イベントフラグ）

サービス・コール名	機能概要	発行有効範囲
set_flg	ビット・パターンのセット	タスク
iset_flg	ビット・パターンのセット	非タスク
clr_flg	ビット・パターンのクリア	タスク
iclr_flg	ビット・パターンのクリア	非タスク
wai_flg	ビット・パターンのチェック	タスク
pol_flg	ビット・パターンのチェック（ポーリング）	タスク
ipol_flg	ビット・パターンのチェック（ポーリング）	非タスク
twai_flg	ビット・パターンのチェック（タイムアウト付き）	タスク
ref_flg	イベントフラグ詳細情報の参照	タスク
iref_flg	イベントフラグ詳細情報の参照	非タスク

set_flg
iset_flg

概要

ビット・パターンのセット

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	`ID flgid;`	イベントフラグの ID
I	`FLGPTN setptn;`	セットするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *setptn* で指定されたビット・パターンの論理和 OR をとり，その結果を対象イベントフラグにセットします。

そして，待ちキューの順に待ちキューにつながれているタスクの待ち解除条件を満たすかどうかを調べます。待ち解除条件を満たせば，該当タスクを待ちキューから外し，WAITING 状態（イベントフラグ待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。このとき，対象のイベントフラグ属性に TA_CLR 属性が指定されている場合には，イベントフラグのビット・パターンを 0 クリアし，処理を終了します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから iset_flg を発行した。 - 非タスクから set_flg を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

clr_flg
iclr_flg

概要

ビット・パターンのクリア

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	`ID flgid;`	イベントフラグの ID
I	`FLGPTN clrptn;`	クリアするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *clrptn* で指定されたビット・パターンの論理積 AND をとり，その結果を対象イベントフラグに設定します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから iclr_flg を発行した場合，および非タスクから clr_flg を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

wai_flg

概要

ビット・パターンのチェック

C 言語形式

```
ER      wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	`ID flgid;`	イベントフラグの ID
I	`FLGPTN waiptn;`	要求するビット・パターン
I	`MODE wfmode;`	要求条件の指定 TWF_ANDW： AND 待ち TWF_ORW： OR 待ち
O	`FLGPTN *p_flgptn;`	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお，イベントフラグ待ち状態の解除は，以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* == TWF_ORW
  waiptn で 1 を設定しているビットのうち, いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RI600V4 では, イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。
ただし，TA_CLR 属性が指定されていない場合は，優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは，μITRON4.0 仕様の範囲外です。

備考 3　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *waiptn* == 0 - 要求条件 *wfmode* の指定が不正である。
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

pol_flg
ipol_flg

概要

ビット・パターンのチェック（ポーリング）

C 言語形式

```
ER      pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER      ipol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	`ID flgid;`	イベントフラグの ID
I	`FLGPTN waiptn;`	要求するビット・パターン
I	`MODE wfmode;`	要求条件の指定 TWF_ANDW： AND 待ち TWF_ORW： OR 待ち
O	`FLGPTN *p_flgptn;`	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，戻り値として E_TMOUT を返します。

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* == TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RI600V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *waiptn* == 0 - 要求条件 *wfmode* の指定が不正である。
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから ipol_flg を発行した場合，および非タスクから pol_flg を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_ILUSE	-28	サービス・コール不正使用 - すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL　属性）に対して本サービス・コールを発行した。
E_TMOUT	-50	ポーリング失敗

twai_flg

概要

ビット・パターンのチェック（タイムアウト付き）

C 言語形式

```
ER      twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	`ID flgid;`	イベントフラグの ID
I	`FLGPTN waiptn;`	要求するビット・パターン
I	`MODE wfmode;`	要求条件の指定 TWF_ANDW： AND 待ち TWF_ORW： OR 待ち
O	`FLGPTN *p_flgptn;`	条件成立時のビット・パターンを格納する領域へのポインタ
I	`TMO tmout;`	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお，イベントフラグ待ち状態の解除は，以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
tmout で指定された待ち時間が経過した。	E_TMOUT

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* == TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RI600V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した場合，RI600V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。
ただし，TA_CLR 属性が指定されていない場合は，優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは，μITRON4.0 仕様の範囲外です。

備考 3　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RI600V4 はビット・パターンのクリア（0 の設定）を行います。

備考 4　待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_flg と同等の処理”を，TMO_POL が指定された際には“pol_flg と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *waiptn* == 0 - 要求条件 *wfmode* の指定が不正である。 - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - すでに待ちタスクがキューイングされているイベントフラグ（TA_WSGL 属性）に対して本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

マクロ	数値	意味
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

ref_flg
iref_flg

概要

イベントフラグ詳細情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID flgid;	イベントフラグの ID
O	T_RFLG *pk_rflg;	イベントフラグ詳細情報を格納する領域へのポインタ

【 イベントフラグ詳細情報 T_RFLG の構造 】

```
typedef struct  t_rflg {
        ID      wtskid;         /* 待ちタスクの有無 */
        FLGPTN  flgptn;         /* 現在ビット・パターン */
} T_RFLG;
```

機能

flgid で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無，現在ビット・パターンなど）を *pk_rflg* で指定された領域に格納します。

- *wtskid*
  イベントフラグの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

- flgptn
  イベントフラグの現在ビット・パターンが格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *flgid* ≦ 0 - *flgid* ＞ VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_flg を発行した場合，および非タスクから ref_flg を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.5　同期通信機能（データ・キュー）

以下に，RI600V4 が同期通信機能（データ・キュー）として提供しているサービス・コールの一覧を示します。

表 18 － 7　同期通信機能（データ・キュー）

サービス・コール名	機能概要	発行有効範囲
snd_dtq	データの送信	タスク
psnd_dtq	データの送信（ポーリング）	タスク
ipsnd_dtq	データの送信（ポーリング）	非タスク
tsnd_dtq	データの送信（タイムアウト付き）	タスク
fsnd_dtq	データの強制送信	タスク
ifsnd_dtq	データの強制送信	非タスク
rcv_dtq	データの受信	タスク
prcv_dtq	データの受信（ポーリング）	タスク
iprcv_dtq	データの受信（ポーリング）	非タスク
trcv_dtq	データの受信（タイムアウト付き）	タスク
ref_dtq	データ・キュー詳細情報の参照	タスク
iref_dtq	データ・キュー詳細情報の参照	非タスク

snd_dtq

概要

データの送信

C 言語形式

```
ER      snd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
I	`VP_INT data;`	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
  パラメータ *data* で指定されたデータをデータ・キューに格納します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ送信待ち状態）へと遷移させます。
  なお，データ送信待ち状態の解除は，以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
rcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
prcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
iprcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
trcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_dtq の発行により，対象データ・キューがリセットされた。	EV_RST

備考 1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2 自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (vrst_dtq) による待ち解除

psnd_dtq
ipsnd_dtq

概要

データの送信（ポーリング）

C 言語形式

```
ER      psnd_dtq ( ID dtqid, VP_INT data );
ER      ipsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID dtqid;	データ・キューの ID
I	VP_INT data;	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
  パラメータ *data* で指定されたデータをデータ・キューに格納します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  戻り値として E_TMOUT を返します。

備考　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ipsnd_dtq を発行した。 - 非タスクから psnd_dtq を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_TMOUT	-50	ポーリング失敗

tsnd_dtq

概要

データの送信（タイムアウト付き）

C 言語形式

```
ER      tsnd_dtq ( ID dtqid, VP_INT data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
I	`VP_INT data;`	送信するデータ
I	`TMO tmout;`	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がある場合
  パラメータ *data* で指定されたデータをデータ・キューに格納します。

- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，データ・キューに空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ送信待ち状態）へと遷移させます。
  なお，データ送信待ち状態の解除は，以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
rcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
prcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
iprcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
trcv_dtq の発行により，対象データ・キューに空き領域が確保された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

データ送信待ち状態の解除操作	戻り値
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_dtq の発行により，対象データ・キューがリセットされた。	EV_RST
tmout で指定された待ち時間が経過した。	E_TMOUT

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “snd_dtq と同等の処理 ” を，TMO_POL が指定された際には “psnd_dtq と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗
EV_RST	-127	オブジェクト・リセット (vrst_dtq) による待ち解除

fsnd_dtq
ifsnd_dtq

概要

データの強制送信

C 言語形式

```
ER      fsnd_dtq ( ID dtqid, VP_INT data );
ER      ifsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID dtqid;	データ・キューの ID
I	VP_INT data;	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされていない場合
  パラメータ *data* で指定されたデータをデータ・キューに格納します。
  データ・キューに空きがない場合は，最古のデータを削除してからデータを格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ifsnd_dtq を発行した。 - 非タスクから fsnd_dtq を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_ILUSE	-28	サービス・コール不正使用 - データ・キュー領域のデータ数が 0 のデータ・キューに対して本サービス・コールを発行した。

rcv_dtq

概要

データの受信

C 言語形式

```
ER      rcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
O	`VP_INT *p_data;`	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- データ・キューにデータが格納されている場合
  データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
  送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
  なお，データ受信待ち状態の解除は，以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
snd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
psnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ipsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
tsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
fsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ifsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

備考　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

prcv_dtq
iprcv_dtq

概要

データの受信（ポーリング）

C 言語形式

```
ER      prcv_dtq ( ID dtqid, VP_INT *p_data );
ER      iprcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
O	`VP_INT *p_data;`	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- データ・キューにデータが格納されている場合
  データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
  送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。

- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。

- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから iprcv_dtq を発行した。 - 非タスクから prcv_dtq を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_TMOUT	-50	ポーリング失敗

trcv_dtq

概要

データの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_dtq ( ID dtqid, VP_INT *p_data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
O	`VP_INT *p_data;`	データを格納する領域へのポインタ
I	`TMO tmout;`	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

dtqid で指定されたデータ・キューの状況に応じて，以下の処理を行います。

- データ・キューにデータが格納されている場合
  データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
  送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（データ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
  なお，データ受信待ち状態の解除は，以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
snd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
psnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
ipsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
tsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
fsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK

データ受信待ち状態の解除操作	戻り値
ifsnd_dtq の発行により，対象データ・キューにデータが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

備考 1 自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2 待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_dtq と同等の処理 ” を，TMO_POL が指定された際には “prcv_dtq と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

ref_dtq
iref_dtq

概要

データ・キュー詳細情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID dtqid;	データ・キューの ID
O	T_RDTQ *pk_rdtq;	データ・キュー詳細情報を格納する領域へのポインタ

【 データ・キュー詳細情報 T_RDTQ の構造 】

```
typedef struct  t_rdtq {
        ID      stskid;         /* データ送信待ちタスクの有無 */
        ID      rtskid;         /* データ受信待ちタスクの有無 */
        UINT    sdtqcnt;        /* 未受信データの総数 */
} T_RDTQ;
```

機能

dtqid で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無，未受信データの総数など）を *pk_rdtq* で指定された領域に格納します。

- *stskid*
  データ・キューの送信待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 送信待ちキューにタスクはキューイングされていない
  その他： 送信待ちキューの先頭にキューイングされているタスクの ID

- *rtskid*
  データ・キューの受信待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 受信待ちキューにタスクはキューイングされていない
  その他： 受信待ちキューの先頭にキューイングされているタスクの ID

- *sdtqcnt*
  データ・キューのデータ・キュー領域にキューイングされている未受信データの総数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_dtq を発行した場合，および非タスクから ref_dtq を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.6　同期通信機能（メールボックス）

以下に，RI600V4 が同期通信機能（メールボックス）として提供しているサービス・コールの一覧を示します。

表 18 － 8　同期通信機能（メールボックス）

サービス・コール名	機能概要	発行有効範囲
snd_mbx	メッセージの送信	タスク
isnd_mbx	メッセージの送信	非タスク
rcv_mbx	メッセージの受信	タスク
prcv_mbx	メッセージの受信（ポーリング）	タスク
iprcv_mbx	メッセージの受信（ポーリング）	非タスク
trcv_mbx	メッセージの受信（タイムアウト付き）	タスク
ref_mbx	メールボックス詳細情報の参照	タスク
iref_mbx	メールボックス詳細情報の参照	非タスク

snd_mbx
isnd_mbx

概要

メッセージの送信

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID mbxid;	メールボックスの ID
I	T_MSG *pk_msg;	メッセージを格納した領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  {
        VP    *msghead;       /*RI600V4 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  {
        T_MSG   msgque;       /* メッセージヘッダ */
        PRI     msgpri;       /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスに *pk_msg* で指定されたメッセージを送信します。

ただし，本サービス・コールを発行した際，対象メールボックスの待ちキューにタスクがキューイングされていた場合には，メッセージの送信（メッセージのキューイング処理）は行わず，該当タスクにメッセージを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　メッセージを対象メールボックスのメッセージ・キューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順またはメッセージ優先度順）に行われます。

備考 2　送信したメッセージ（*pk_msg* が指す領域）は，受信されるまでの間に書き換えてはなりません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - 対象メールボックスが TA_MPRI 属性の場合： - *msgpri* ≦ 0 - *msgpri* > TMAX_MPRI
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから isnd_mbx を発行した。 - 非タスクから snd_mbx を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

rcv_mbx

概要

メッセージの受信

C 言語形式

```
ER      rcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	`ID mbxid;`	メールボックスの ID
O	`T_MSG **ppk_msg;`	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  {
        VP    *msghead;        /*RI600V4 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  {
        T_MSG   msgque;        /* メッセージヘッダ */
        PRI     msgpri;        /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

備考　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

prcv_mbx
iprcv_mbx

概要

メッセージの受信（ポーリング）

C 言語形式

```
ER      prcv_mbx ( ID mbxid, T_MSG **ppk_msg );
ER      iprcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID mbxid;	メールボックスの ID
O	T_MSG **ppk_msg;	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  {
        VP    *msghead;       /*RI600V4 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  {
        T_MSG   msgque;       /* メッセージヘッダ */
        PRI     msgpri;       /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iprcv_mbx を発行した場合，および非タスクから prcv_mbx を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_TMOUT	-50	ポーリング失敗

trcv_mbx

概要

メッセージの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mbxid;	メールボックスの ID
O	T_MSG **ppk_msg;	メッセージの先頭アドレスを格納する領域へのポインタ
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  {
        VP    *msghead;       /*RI600V4 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  {
        T_MSG   msgque;       /* メッセージヘッダ */
        PRI     msgpri;       /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK

メッセージ受信待ち状態の解除操作	戻り値
isnd_mbx の発行により，対象メールボックスにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_mbx と同等の処理 ” を，TMO_POL が指定された際には “prcv_mbx と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

ref_mbx
iref_mbx

概要

メールボックス詳細情報の参照

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID mbxid;	メールボックスの ID
O	T_RMBX *pk_rmbx;	メールボックス詳細情報を格納する領域へのポインタ

【 メールボックス詳細情報 T_RMBX の構造 】

```
typedef struct  t_rmbx {
        ID      wtskid;         /* 待ちタスクの有無 */
        T_MSG   *pk_msg;        /* 待ちメッセージの有無 */
} T_RMBX;
```

機能

mbxid で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無，待ちメッセージの有無など）を *pk_rmbx* で指定された領域に格納します。

- *wtskid*
  メールボックスの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

- *pk_msg*
  メールボックスの待ちキューにメッセージがキューイングされているか否かが格納されます。
  NULL： 待ちキューにメッセージはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているメッセージの先頭アドレス

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_mbx を発行した場合，および非タスクから ref_mbx を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.7　拡張同期通信機能（ミューテックス）

以下に，RI600V4 が拡張同期通信機能（ミューテックス）として提供しているサービス・コールの一覧を示します。

表 18 － 9　拡張同期通信機能（ミューテックス）

サービス・コール名	機能概要	発行有効範囲
loc_mtx	ミューテックスのロック	タスク
ploc_mtx	ミューテックスのロック（ポーリング）	タスク
tloc_mtx	ミューテックスのロック（タイムアウト付き）	タスク
unl_mtx	ミューテックスのロック解除	タスク
ref_mtx	ミューテックス詳細情報の参照	タスク

loc_mtx

概要

ミューテックスのロック

C 言語形式

```
ER      loc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID mtxid;	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。
ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。
なお，ミューテックス待ち状態の解除は，以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
unl_mtx の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。

備考 1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，タスクの現在優先度順に行われます。ただし，同じ優先度のタスクの中では FIFO 順に行われます。

備考 2 自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mtxid* ≦ 0 - *mtxid* ＞ VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。

ploc_mtx

概要

ミューテックスのロック（ポーリング）

C 言語形式

```
ER      ploc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	`ID mtxid;`	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。

ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，戻り値として E_TMOUT を返します。

ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。

備考　RI600V4 では，対象ミューテックスをロック状態へと遷移させたタスクがロック状態を解除することなく，本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mtxid* ≦ 0 - *mtxid* ＞ VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）

マクロ	数値	意味
E_TMOUT	-50	ポーリング失敗

tloc_mtx

概要

ミューテックスのロック（タイムアウト付き）

C 言語形式

```
ER      tloc_mtx ( ID mtxid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mtxid;	ミューテックスの ID
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

パラメータ *mtxid* で指定されたミューテックスをロックします。
ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（ミューテックス待ち状態）へと遷移させます。
なお，ミューテックス待ち状態の解除は，以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
unl_mtx の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

ミューテックスのロック時には，自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，自タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，自タスクの現在優先度は変更しません。

備考 1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，タスクの現在優先度順に行われます。ただし，同じ優先度のタスクの中では FIFO 順に行われます。

備考 2 RI600V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “loc_mtx と同等の処理 ” を，TMO_POL が指定された際には “ploc_mtx と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mtxid* ≦ 0 - *mtxid* ＞ VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗

unl_mtx

概要

ミューテックスのロック解除

C 言語形式

```
ER      unl_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	`ID mtxid;`	ミューテックスの ID

機能

mtxid で指定されたミューテックスのロック状態を解除します。その結果，自タスクがロックしているミューテックスがなくなった場合には，自タスクの現在優先度をベース優先度に変更します。

本サービス・コールを発行した際，対象ミューテックスの待ちキューにタスクがキューイングされていた場合には，ミューテックスのロック解除処理後，ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。このとき，該当タスクは，待ちキューから外れ，WAITING 状態（ミューテックス待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，該当タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし，該当タスクがすでに他のミューテックスをロックしており，かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は，該当タスクの現在優先度は変更しません。

備考 1　ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため，自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には，戻り値として E_ILUSE を返します。

備考 2　タスクの終了時，そのタスクがロックしていたミューテックスはロック解除されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mtxid* ≦ 0 - *mtxid* ＞ VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_ILUSE	-28	サービス・コール不正使用 - 自タスクは対象ミューテックスをロックしていない。

ref_mtx

概要

ミューテックス詳細情報の参照

C 言語形式

```
ER      ref_mtx ( ID mtxid, T_RMTX *pk_rmtx );
```

パラメータ

I/O	パラメータ	説明
I	ID mtxid;	ミューテックスの ID
O	T_RMTX *pk_rmtx;	ミューテックス詳細情報を格納する領域へのポインタ

【 ミューテックス詳細情報 T_RMTX の構造 】

```
typedef struct  t_rmtx {
        ID      htskid;         /* ロックの有無 */
        ID      wtskid;         /* 待ちタスクの有無 */
} T_RMTX;
```

機能

mtxid で指定されたミューテックスのミューテックス詳細情報（ロックの有無，待ちタスクの有無など）を *pk_rmtx* で指定された領域に格納します。

- *htskid*
  ミューテックスをロックしているタスクが存在しているか否かが格納されます。
  TSK_NONE： ロックしているタスクは存在しない
  その他： ロックしているタスクの ID

- *wtskid*
  ミューテックスの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mtxid* ≦ 0 - *mtxid* ＞ VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

18.2.8　拡張同期通信機能（メッセージ・バッファ）

以下に, RI600V4 が拡張同期通信機能（メッセージ･バッファ）として提供しているサービス･コールの一覧を示します。

表 18 － 10　拡張同期通信機能（メッセージ・バッファ）

サービス・コール名	機能概要	発行有効範囲
snd_mbf	メッセージの送信	タスク
psnd_mbf	メッセージの送信（ポーリング）	タスク
ipsnd_mbf	メッセージの送信（ポーリング）	非タスク
tsnd_mbf	メッセージの送信（タイムアウト付き）	タスク
rcv_mbf	メッセージの受信	タスク
prcv_mbf	メッセージの受信（ポーリング）	タスク
trcv_mbf	メッセージの受信（タイムアウト付き）	タスク
ref_mbf	メッセージ・バッファ詳細情報の参照	タスク
iref_mbf	メッセージ・バッファ詳細情報の参照	非タスク

snd_mbf

概要

メッセージの送信

C 言語形式

```
ER      snd_mbf ( ID mbfid, VP  msg, UINT msgsz );
```

パラメータ

I/O	パラメータ	説明
I	ID　　mbfid;	メッセージ・バッファの ID
I	VP　　msg;	送信メッセージへのポインタ
I	UINT　　msgsz;	送信メッセージのサイズ（単位：バイト）

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
  msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
  減少サイズ＝ up4(*msgsz*) ＋ VTSZ_MBFTBL
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ送信待ち状態）へと遷移させます。
  なお，メッセージ送信待ち状態の解除は，以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
rcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
prcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
trcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK

メッセージ送信待ち状態の解除操作	戻り値
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tsnd_mbf のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mbf の発行により，対象メッセージ・バッファがリセットされた。	EV_RST

備考 1　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *msgsz* == 0 - *msgsz* > 最大メッセージ・サイズ（max_msgsz）
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (vrst_mbf) による待ち解除

psnd_mbf
ipsnd_mbf

概要

メッセージの送信（ポーリング）

C 言語形式

```
ER      psnd_mbf ( ID mbfid, VP  msg, UINT msgsz );
ER      ipsnd_mbf ( ID mbfid, VP  msg, UINT msgsz );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID
I	VP msg;	送信メッセージへのポインタ
I	UINT msgsz;	送信メッセージのサイズ（単位：バイト）

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
  msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
  減少サイズ＝ up4(*msgsz*) ＋ VTSZ_MBFTBL
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  戻り値として E_TMOUT を返します。

備考　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - *msgsz* == 0 - *msgsz* > 最大メッセージ・サイズ（max_msgsz）
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ipsnd_mbf を発行した。 - 非タスクから psnd_mbf を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_TMOUT	-50	ポーリング失敗

tsnd_mbf

概要

メッセージの送信（タイムアウト付き）

C 言語形式

```
ER      tsnd_mbf ( ID mbfid, VP  msg, UINT msgsz, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID
I	VP msg;	送信メッセージへのポインタ
I	UINT msgsz;	送信メッセージのサイズ（単位：バイト）
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
  msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します，これにより，該当タスクは受信待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がある場合
  msg で指定されたメッセージをメッセージ・バッファに格納します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
  減少サイズ＝ up4(*msgsz*) ＋ VTSZ_MBFTBL
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず，メッセージ・バッファにメッセージを格納するための空き領域がない場合，または送信待ちキューにタスクがキューイングされている場合
  自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAIITING 状態（メッセージ送信待ち状態）へと遷移させます。
  なお，メッセージ送信待ち状態の解除は，以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
rcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
prcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK
trcv_mbf の発行により，対象メッセージ・バッファに空き領域が確保された。	E_OK

メッセージ送信待ち状態の解除操作	戻り値
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tsnd_mbf のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mbf の発行により，対象メッセージ・バッファがリセットされた。	EV_RST
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

備考 1　メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は，メッセージの送信要求を行った順に行われます。

備考 2　自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には “snd_mbf と同等の処理 ” を，TMO_POL が指定された際には “psnd_mbf と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *msgsz* == 0 - *msgsz* > 最大メッセージ・サイズ（max_msgsz） - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗
EV_RST	-127	オブジェクト・リセット (vrst_mbf) による待ち解除

rcv_mbf

概要

メッセージの受信

C 言語形式

```
ER_UINT  rcv_mbf ( ID mbfid, VP msg );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID
O	VP msg;	メッセージを格納する領域へのポインタ

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
  メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
  増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL
  さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。
  - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
    減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
  なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK

メッセージ受信待ち状態の解除操作	戻り値
psnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
ipsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
tsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI

備考 1　最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域のサイズは，最大メッセージ・サイズ以上でなければなりません。

備考 2　自タスクを対象メッセージ・バッファの受信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

戻り値

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
－	正の値	正常終了（受信したメッセージのサイズ）

prcv_mbf

概要

メッセージの受信（ポーリング）

C 言語形式

```
ER_UINT  prcv_mbf ( ID mbfid, VP msg );
```

パラメータ

I/O	パラメータ	説明
I	ID　　mbfid;	メッセージ・バッファの ID
O	VP　　msg;	メッセージを格納する領域へのポインタ

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
  メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
  増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL
  さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。
  - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
    減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  戻り値として E_TMOUT を返します。

備考　最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域のサイズは，最大メッセージ・サイズ以上でなければなりません。

戻り値

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_TMOUT	-50	ポーリング失敗
ー	正	正常終了（受信したメッセージのサイズ）

trcv_mbf

概要

メッセージの受信（タイムアウト付き）

C 言語形式

```
ER_UINT  trcv_mbf ( ID mbfid, VP msg, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID
O	VP msg;	メッセージを格納する領域へのポインタ
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mbfid で指定されたメッセージ・バッファの状況に応じて，以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
  メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。このとき，対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
  増加サイズ＝ up4(戻り値) ＋ VTSZ_MBFTBL
  さらに以下を，送信待ちキューにタスクがなくなるか，メッセージを格納できなくなるまで繰り返します。
  - 送信待ちキューにタスクがキューイングされており，キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は，そのメッセージをメッセージ・バッファに格納します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また，このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
    減少サイズ＝ up4(該当タスクが送信したメッセージ・サイズ) ＋ VTSZ_MBFTBL
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされている場合
  送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し，そのメッセージサイズを戻り値として返します。これにより，該当タスクは，送信待ちキューから外れ，WAITING 状態（メッセージ送信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，この状況は，メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず，送信待ちキューにタスクがキューイングされていない場合
  自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
  なお，メッセージ受信待ち状態の解除は，以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
psnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
ipsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
tsnd_mbf の発行により，対象メッセージ・バッファにメッセージが送信された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した。	E_TMOUT

備考 1 最大メッセージ・サイズ（max_msgsz）は，コンフィギュレーション時に定義します。*msg* の指す領域のサイズは，最大メッセージ・サイズ以上でなければなりません。

備考 2 自タスクを対象メッセージ・バッファの受信待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3 待ち時間 *tmout* に TMO_FEVR が指定された際には “rcv_mbf と同等の処理 ” を，TMO_POL が指定された際には “prcv_mbf と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗
―	正	正常終了（受信したメッセージのサイズ）

ref_mbf
iref_mbf

概要

メッセージ・バッファ詳細情報の参照

C 言語形式

```
ER      ref_mbf ( ID mbfid, T_RMBF *pk_rmbf );
ER      iref_mbf ( ID mbfid, T_RMBF *pk_rmbf );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID
O	T_RMBF *pk_rmbf;	メッセージ・バッファ詳細情報を格納する領域へのポインタ

【 メッセージ・バッファ詳細情報 T_RMBF の構造 】

```
typedef struct  t_rmbf {
        ID      stskid;         /* メッセージ送信待ちタスクの有無 */
        ID      rtskid;         /* メッセージ受信待ちタスクの有無 */
        UINT    smsgcnt;        /* 未受信メッセージの総数 */
        SIZE    fmbfsz;         /* 空き領域のサイズ */
} T_RMBF;
```

機能

mbfid で指定されたメッセージ・バッファのメッセージ・バッファ詳細情報（待ちタスクの有無，未受信データの総数など）を *pk_rmbf* で指定された領域に格納します。

- *stskid*
  メッセージ・バッファの送信待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 送信待ちキューにタスクはキューイングされていない
  その他： 送信待ちキューの先頭にキューイングされているタスクの ID

- *rtskid*
  メッセージ・バッファの受信待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 受信待ちキューにタスクはキューイングされていない
  その他： 受信待ちキューの先頭にキューイングされているタスクの ID

- *smsgcnt*
  メッセージ・バッファに格納されている未受信メッセージ総数が格納されます。

- *fmbfsz*
  メッセージ・バッファの空き領域のサイズ (単位：バイト）が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_mbf を発行した場合，および非タスクから ref_mbf を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.9　メモリ・プール管理機能（固定長メモリ・プール）

以下に，RI600V4 がメモリ・プール管理機能（固定長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 18 － 11　メモリ・プール管理機能（固定長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
get_mpf	固定長メモリ・ブロックの獲得	タスク
pget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	タスク
ipget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	非タスク
tget_mpf	固定長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpf	固定長メモリ・ブロックの返却	タスク
irel_mpf	固定長メモリ・ブロックの返却	非タスク
ref_mpf	固定長メモリ・プール詳細情報の参照	タスク
iref_mpf	固定長メモリ・プール詳細情報の参照	非タスク

get_mpf

概要

固定長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	`ID mpfid;`	固定長メモリ・プールの ID
O	`VP *p_blk;`	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpf の発行により，対象固定長メモリ・プールがリセットされた。	EV_RST

備考 1　タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2　獲得したメモリ・ブロックの内容は不定です。

備考 3　メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (vrst_mpf) による待ち解除

pget_mpf
ipget_mpf

概要

固定長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER      pget_mpf ( ID mpfid, VP *p_blk );
ER      ipget_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID mpfid;	固定長メモリ・プールの ID
O	VP *p_blk;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。

備考 1　獲得したメモリ・ブロックの内容は不定です。

備考 2　メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから ipget_mpf を発行した場合，および非タスクから get_mpf を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_TMOUT	-50	ポーリング失敗

tget_mpf

概要

固定長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpf ( ID mpfid, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mpfid;	固定長メモリ・プールの ID
O	VP *p_blk;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpf の発行により，対象固定長メモリ・プールがリセットされた。	EV_RST
tmout で指定された待ち時間が経過した。	E_TMOUT

備考 1 自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順または現在優先度順）に行われます。

備考 2 獲得したメモリ・ブロックの内容は不定です。

備考 3 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は，以下を守ってください。

- 固定長メモリ・プール情報（memorypool[]）の固定長メモリ・ブロック・サイズ（siz_block）を，目的のアライメント数の倍数とする。
- 固定長メモリ・プール情報（memorypool[]）のメモリ・プール領域に付与するセクション名（section）を個別のセクションとし，リンク時にそのセクションを目的のアライメント数のアドレスに配置する。

備考 4　待ち時間 *tmout* に TMO_FEVR が指定された際には “get_mpf と同等の処理 ” を，TMO_POL が指定された際には “pget_mpf と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗
EV_RST	-127	オブジェクト・リセット (vrst_mpf) による待ち解除

rel_mpf
irel_mpf

概要

固定長メモリ・ブロックの返却

C 言語形式

```
ER      rel_mpf ( ID mpfid, VP blk );
ER      irel_mpf ( ID mpfid, VP blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　mpfid;	固定長メモリ・プールの ID
I	VP　blk;	固定長メモリ・ブロックの先頭アドレス

機能

mpfid で指定された固定長メモリ・プールに *blk* で指定された固定長メモリ・ブロックを返却します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却は行わず，該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *blk* が不正
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irel_mpf を発行した。 - 非タスクから rel_mpf を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

ref_mpf
iref_mpf

概要

固定長メモリ・プール詳細情報の参照

C 言語形式

```
ER      ref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
ER      iref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
```

パラメータ

I/O	パラメータ	説明
I	ID mpfid;	固定長メモリ・プールの ID
O	T_RMPF *pk_rmpf;	固定長メモリ・プール詳細情報を格納する領域へのポインタ

【 固定長メモリ・プール詳細情報 T_RMPF の構造 】

```
typedef struct  t_rmpf {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    fblkcnt;        /* 空きメモリ・ブロックの総数 */
 } T_RMPF;
```

機能

mpfid で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無，空き固定長メモリ・ブロックの総数など）を *pk_rmpf* で指定された領域に格納します。

- *wtskid*
  固定長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

- *fblkcnt*
  空きメモリ・ブロックの総数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_mpf を発行した場合，および非タスクから ref_mpf を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.10 メモリ・プール管理機能（可変長メモリ・プール）

以下に，RI600V4 がメモリ・プール管理機能（可変長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 18 － 12　メモリ・プール管理機能（可変長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
get_mpl	可変長メモリ・ブロックの獲得	タスク
pget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	タスク
ipget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	非タスク
tget_mpl	可変長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpl	可変長メモリ・ブロックの返却	タスク
ref_mpl	可変長メモリ・プール情詳細報の参照	タスク
iref_mpl	可変長メモリ・プール詳細情報の参照	非タスク

get_mpl

概要

可変長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpl ( ID mplid, UINT blksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	`ID mplid;`	可変長メモリ・プールの ID
I	`UINT blksz;`	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	`VP *p_blk;`	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tget_mpl のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpl の発行により，対象可変長メモリ・プールがリセットされた。	EV_RST

備考 1　実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　獲得したメモリ・ブロックの内容は不定です。

備考 4　獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *blksz* == 0 - *blksz* が獲得可能な最大サイズを超えている。
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (vrst_mpl) による待ち解除

pget_mpl
ipget_mpl

概要

可変長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER      pget_mpl  ( ID mplid, UINT blksz, VP *p_blk );
ER      ipget_mpl ( ID mplid, UINT blksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID mplid;	可変長メモリ・プールの ID
I	UINT blksz;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP *p_blk;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。

備考 1 実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2 可変長メモリ・ブロックのサイズ」を参照してください。

備考 2 獲得したメモリ・ブロックの内容は不定です。

備考 3 獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *blksz* == 0 - *blksz* が獲得可能な最大サイズを超えている。

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから ipget_mpl を発行した場合，および非タスクから pget_mpl を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_TMOUT	-50	ポーリング失敗

tget_mpl

概要

可変長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpl ( ID mplid, UINT blksz, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID mplid;	可変長メモリ・プールの ID
I	UINT blksz;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP *p_blk;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO tmout;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された。 - rel_wai の発行により，待ち状態を強制的に解除された。 - irel_wai の発行により，待ち状態を強制的に解除された。 - ter_tsk の発行により，待ち状態を強制的に解除された。 - tget_mpl のパラメータ *tmout* で指定された待ち時間が経過した。	E_OK
rel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により，待ち状態を強制的に解除された。	E_RLWAI
vrst_mpl の発行により，対象可変長メモリ・プールがリセットされた。	EV_RST

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
tmout で指定された待ち時間が経過した。	E_TMOUT

備考 1　実際に獲得されるメモリ・ブロックのサイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，FIFO 順に行われます。

備考 3　獲得したメモリ・ブロックの内容は不定です。

備考 4　獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには，可変長メモリ・プール情報（variable_memorypool[]）のメモリ・プール領域に付与するセクション名（mpl_section）に個別のセクション名を指定し，リンク時にそのセクションを 4 バイト境界アドレスに配置してください。

備考 5　待ち時間 *tmout* に TMO_FEVR が指定された際には “get_mpl と同等の処理 ” を，TMO_POL が指定された際には “pget_mpl と同等の処理 ” を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *blksz* == 0 - *blksz* が獲得可能な最大サイズを超えている。 - *tmout* ＜ -1 - *tmout* ＞ (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により，待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した，またはポーリング失敗
EV_RST	-127	オブジェクト・リセット (vrst_mpl) による待ち解除

rel_mpl

概要

可変長メモリ・ブロックの返却

C 言語形式

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

パラメータ

I/O	パラメータ	説明
I	ID mplid;	可変長メモリ・プールの ID
I	VP blk;	可変長メモリ・ブロックの先頭アドレス

機能

mplid で指定された可変長メモリ・プールに *blk* で指定された可変長メモリ・ブロックを返却します。

可変長メモリ・ブロックを返却したあと，対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき，待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか，メモリが割り当てられなくなるまで繰り返します。これにより，メモリを獲得できたタスクは，待ちキューから外れ，WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　RI600V4 では，*blk* に関して簡易的なエラー検出しか行っていません。*blk* には，必ず正しい値を指定してください。*blk* が不正でかつエラー検出されない場合，以後の動作は保証されません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *blk* が不正である。
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

ref_mpl
iref_mpl

概要

可変長メモリ・プール詳細情報の参照

C 言語形式

```
ER      ref_mpl ( ID mplid, T_RMPL *pk_rmpl );
ER      iref_mpl ( ID mplid, T_RMPL *pk_rmpl );
```

パラメータ

I/O	パラメータ	説明
I	ID mplid;	可変長メモリ・プールの ID
O	T_RMPL *pk_rmpl;	可変長メモリ・プール詳細情報を格納する領域へのポインタ

【 可変長メモリ・プール詳細情報 T_RMPL の構造 】

```
typedef struct  t_rmpl {
        ID      wtskid;         /* 待ちタスクの有無 */
        SIZE    fmplsz;         /* 空き領域の合計サイズ */
        UINT    fblksz;         /* 獲得可能なメモリ・ブロックの最大サイズ */
} T_RMPL;
```

機能

mplid で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無，空き可変長メモリ・ブロックの合計サイズなど）を *pk_rmpl* で指定された領域に格納します。

- *wtskid*
  可変長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。
  TSK_NONE： 待ちキューにタスクはキューイングされていない
  その他： 待ちキューの先頭にキューイングされているタスクの ID

- *fmplsz*
  可変長メモリ・プールの空き領域の合計サイズ（単位：バイト）が格納されます。

- *fblksz*
  獲得可能なメモリ・ブロックの最大サイズ（単位：バイト）が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_mpl を発行した場合，および非タスクから ref_mpl を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.11　時間管理機能

以下に，RI600V4 が時間管理機能として提供しているサービス・コールの一覧を示します。

表 18 － 13　時間管理機能

サービス・コール名	機能概要	発行有効範囲
set_tim	システム時刻の設定	タスク
iset_tim	システム時刻の設定	非タスク
get_tim	システム時刻の参照	タスク
iget_tim	システム時刻の参照	非タスク
sta_cyc	周期ハンドラの動作開始	タスク
ista_cyc	周期ハンドラの動作開始	非タスク
stp_cyc	周期ハンドラの動作停止	タスク
istp_cyc	周期ハンドラの動作停止	非タスク
ref_cyc	周期ハンドラ詳細情報の参照	タスク
iref_cyc	周期ハンドラ詳細情報の参照	非タスク
sta_alm	アラーム・ハンドラの動作開始	タスク
ista_alm	アラーム・ハンドラの動作開始	非タスク
stp_alm	アラーム・ハンドラの動作停止	タスク
istp_alm	アラーム・ハンドラの動作停止	非タスク
ref_alm	アラーム・ハンドラ詳細情報の参照	タスク
iref_alm	アラーム・ハンドラ詳細情報の参照	非タスク

set_tim
iset_tim

概要

システム時刻の設定

C 言語形式

```
ER      set_tim ( SYSTIM *p_systim );
ER      iset_tim ( SYSTIM *p_systim );
```

パラメータ

I/O	パラメータ	説明
I	SYSTIM *p_systim;	システム時刻情報を格納した領域へのポインタ

【 システム時刻情報 SYSTIM の構造 】

```
typedef struct  systim {
        UH      utime;          /* システム時刻（上位 16 ビット）*/
        UW      ltime;          /* システム時刻（下位 32 ビット）*/
} SYSTIM;
```

機能

システム時刻（単位：ミリ秒）を *p_systim* で指定された時間に変更します。

備考　システム時刻を変更しても，それ以前に行われた時間管理要求 (タスクのタイムアウト，dly_tsk によるタスクの遅延，周期ハンドラ，およびアラーム・ハンドラ) が発生する実時刻は変化しません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iset_tim を発行した場合，および非タスクから set_tim を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

get_tim
iget_tim

概要

システム時刻の参照

C 言語形式

```
ER      get_tim ( SYSTIM *p_systim );
ER      iget_tim ( SYSTIM *p_systim );
```

パラメータ

I/O	パラメータ	説明
O	SYSTIM *p_systim;	システム時刻情報を格納する領域へのポインタ

【 システム時刻情報 SYSTIM の構造 】

```
typedef struct  systim {
        UH      utime;          /* システム時刻（上位 16 ビット）*/
        UW      ltime;          /* システム時刻（下位 32 ビット）*/
} SYSTIM;
```

機能

システム時刻（単位：ミリ秒）を *p_systim* で指定された領域に格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから iget_tim を発行した場合，および非タスクから get_tim を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

sta_cyc
ista_cyc

概要

周期ハンドラの動作開始

C 言語形式

```
ER      sta_cyc ( ID cycid );
ER      ista_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID cycid;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を停止状態（STP 状態）から動作状態（STA 状態）へと遷移させます。これにより，対象周期ハンドラは，RI600V4 の起動対象となります。

なお，本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は，コンフィギュレーション時に対象周期ハンドラに対して TA_PHS 属性（phsatr）を指定しているか否かにより異なります。詳細は，「8.6.4 周期ハンドラの動作開始」を参照してください。

- TA_PHS 属性を指定した場合
  起動位相（phs_counter），起動周期（interval_counter）にしたがって起動タイミングが設定されます。
  ただし，対象周期ハンドラの動作状態が開始状態の場合には，本サービス・コールを発行しても何も処理は行わず，エラーとしても扱いません。

- TA_PHS 属性を指定しない場合
  本サービス・コールの発行時点を基点に，起動周期（interval_counter）にしたがって起動タイミングが設定されます。
  なお，起動タイミング設定処理については，対象周期ハンドラの動作状態に関係なく実行されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *cycid* ≦ 0 - *cycid* ＞ VTMAX_CYH

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから ista_cyc を発行した場合，および非タスクから sta_cyc を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

stp_cyc
istp_cyc

概要

周期ハンドラの動作停止

C 言語形式

```
ER      stp_cyc ( ID cycid );
ER      istp_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID cycid;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_cyc または ista_cyc が発行されるまでの間，対象周期ハンドラは，RI600V4 の起動対象から除外されます。

備考　本サービス・コールでは，停止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対象周期ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *cycid* ≦ 0 - *cycid* ＞ VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから istp_cyc を発行した場合，および非タスクから stp_cyc を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

ref_cyc
iref_cyc

概要

周期ハンドラ詳細情報の参照

C 言語形式

```
ER      ref_cyc ( ID cycid, T_RCYC *pk_rcyc );
ER      iref_cyc ( ID cycid, T_RCYC *pk_rcyc );
```

パラメータ

I/O	パラメータ	説明
I	ID cycid;	周期ハンドラの ID
O	T_RCYC *pk_rcyc;	周期ハンドラ詳細情報を格納する領域へのポインタ

【 周期ハンドラ詳細情報 T_RCYC の構造 】

```
typedef struct  t_rcyc {
        STAT    cycstat;        /* 現在状態 */
        RELTIM  lefttim;        /* 残り時間 */
} T_RCYC;
```

機能

cycid で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態，残り時間など）を *pk_rcyc* で指定された領域に格納します。

- *cycstat*
  周期ハンドラの現在状態が格納されます。
  TCYC_STP : 停止状態（STP 状態）
  TCYC_STA : 動作状態（STA 状態）

- *lefttim*
  周期ハンドラが次に起動するまでの残り時間（単位：ミリ秒）が格納されます。対象周期ハンドラが停止状態の場合，lefttim は不定値となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *cycid* ≦ 0 - *cycid* ＞ VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_cyc を発行した場合，および非タスクから ref_cyc を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

sta_alm
ista_alm

概要

アラーム・ハンドラの動作開始

C 言語形式

```
ER      sta_alm ( ID almid, RELTIM almtim );
ER      ista_alm ( ID almid, RELTIM almtim  );
```

パラメータ

I/O	パラメータ	説明
I	ID almid;	アラーム・ハンドラの ID
I	RELTIM almtim;	起動時刻（単位：ミリ秒）

機能

almid で指定されたアラーム・ハンドラの起動時刻を，本サービス・コールが呼び出された時刻から *almtim* ミリ秒後に設定し，動作状態（STA 状態）にします。これにより，対象アラーム・ハンドラは，RI600V4 の起動対象となります。

備考 1　*almtim* に 0 を指定すると，次回の基本クロック割り込み時にアラーム・ハンドラが起動されます。

備考 2　対象アラーム・ハンドラがすでに動作状態の場合でも，本サービス・コールは起動時刻を再設定します。以前の起動時刻の設定は無効となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *almtim* > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - *almid* ≦ 0 - *almid* ＞ VTMAX_ALH

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから ista_alm を発行した場合，および非タスクから stp_alm を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

stp_alm
istp_alm

概要

アラーム・ハンドラの動作停止

C 言語形式

```
ER      stp_alm ( ID almid );
ER      istp_alm ( ID almid );
```

パラメータ

I/O	パラメータ	説明
I	`ID almid;`	アラーム・ハンドラの ID

機能

almid で指定されたアラーム・ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_alm または ista_alm が発行されるまでの間，対象アラーム・ハンドラは，RI600V4 の起動対象から除外されます。

備考　本サービス・コールでは，停止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対象アラーム・ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *almid* ≦ 0 - *almid* ＞ VTMAX_ALH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから istp_alm を発行した場合，および非タスクから stp_alm を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

ref_alm
iref_alm

概要

アラーム・ハンドラ詳細情報の参照

C 言語形式

```
ER      ref_alm ( ID almid, T_RALM *pk_ralm );
ER      iref_alm ( ID almid, T_RALM *pk_ralm );
```

パラメータ

I/O	パラメータ	説明
I	ID almid;	アラーム・ハンドラの ID
O	T_RALM *pk_ralm;	アラーム・ハンドラ詳細情報を格納する領域へのポインタ

【 アラーム・ハンドラ詳細情報 T_RALM の構造 】

```
typedef struct  t_ralm {
        STAT    almstat;        /* 現在状態 */
        RELTIM  lefttim;        /* 残り時間 */
} T_RALM;
```

機能

almid で指定されたアラーム・ハンドラのアラーム・ハンドラ詳細情報（現在状態，残り時間など）を *pk_ralm* で指定された領域に格納します。

- *almstat*
  アラーム・ハンドラの現在状態が格納されます。
  TALM_STP： 停止状態（STP 状態）
  TALM_STA： 動作状態（STA 状態）

- *lefttim*
  アラーム・ハンドラが次に起動するまでの残り時間（単位：ミリ秒）が格納されます。対象アラーム・ハンドラが停止状態の場合，lefttim は不定値となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	不正 ID 番号 - *almid* ≦ 0 - *almid* ＞ VTMAX_ALH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_alm を発行した場合，および非タスクから ref_alm を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.12 システム状態管理機能

以下に，RI600V4 がシステム状態管理機能として提供しているサービス・コールの一覧を示します。

表 18 － 14　システム状態管理機能

サービス・コール名	機能概要	発行有効範囲
rot_rdq	タスクの優先順位の回転	タスク
irot_rdq	タスクの優先順位の回転	非タスク
get_tid	RUNNING 状態のタスクの参照	タスク
iget_tid	RUNNING 状態のタスクの参照	非タスク
loc_cpu	CPU ロック状態への移行	タスク
iloc_cpu	CPU ロック状態への移行	非タスク
unl_cpu	CPU ロック状態の解除	タスク
iunl_cpu	CPU ロック状態の解除	非タスク
dis_dsp	ディスパッチの禁止	タスク
ena_dsp	ディスパッチの許可	タスク
sns_ctx	コンテキスト種別の参照	タスク，非タスク
sns_loc	CPU ロック状態の参照	タスク，非タスク
sns_dsp	ディスパッチ禁止状態の参照	タスク，非タスク
sns_dpn	ディスパッチ保留状態の参照	タスク，非タスク
vsys_dwn	システム・ダウン	タスク，非タスク
ivsys_dwn	システム・ダウン	タスク，非タスク
vsta_knl	RI600V4 の起動	タスク，非タスク
ivsta_knl	RI600V4 の起動	タスク，非タスク

rot_rdq
irot_rdq

概要

タスクの優先順位の回転

C 言語形式

```
ER      rot_rdq ( PRI tskpri );
ER      irot_rdq ( PRI tskpri );
```

パラメータ

I/O	パラメータ	説明
I	PRI tskpri;	タスクの優先度 TPRI_SELF：　自タスクのベース優先度 数値：　　タスクの優先度

機能

tskpri で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ，タスクの実行順序を明示的に変更します。

備考 1　本サービス・コールでは，レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合，何も処理は行わず，エラーとしても扱いません。

備考 2　本サービス・コールを周期ハンドラなどから一定周期で発行することにより，ラウンドロビン・スケジューリングを実現することができます。

備考 3　RI600V4　　におけるレディ・キューは，優先度をキーとしたハッシュ・テーブルであり，実行可能な状態（RUNNING 状態または READY 状態）へと遷移したタスクが FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RI600V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。

備考 4　ミューテックスをロック中のタスクの現在優先度は，ベース優先度と異なる場合があります。この場合，そのタスクが本サービス・コールで *tskpri* に TPRI_SELF を指定しても，自タスクが属する現在優先度のレディ・キューを回転することはできません。

備考 5　現在優先度とベース優先度については，「6.2.2　現在優先度とベース優先度」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - *tskpri* ＜ 0 - *tskpri* ＞ TMAX_TPRI - irot_rdq を発行した際，*tskpri* に TPRI_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irot_rdq を発行した。 - 非タスクから rot_rdq を発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

get_tid
iget_tid

概要

RUNNING 状態のタスクの参照

C 言語形式

```
ER      get_tid ( ID *p_tskid );
ER      iget_tid ( ID *p_tskid );
```

パラメータ

I/O	パラメータ	説明
O	`ID *p_tskid;`	タスク ID を格納する領域へのポインタ

機能

RUNNING 状態のタスクの ID を *p_tskid* で指定された領域に格納します。
RUNNING 状態のタスクが存在しない場合（IDLE 状態）には，*p_tskid* で指定された領域に TSK_NONE を格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iget_tid を発行した場合，および非タスクから get_tid を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

loc_cpu
iloc_cpu

概要

CPU ロック状態への移行

C 言語形式

```
ER      loc_cpu ( void );
ER      iloc_cpu ( void );
```

パラメータ

なし

機能

システムを CPU ロック状態へ移行させます。

CPU ロック状態では，タスクのスケジューリングは禁止され，カーネル管理割り込みもマスクされます。つまり，カーネル管理外割り込みハンドラを除くすべての処理プログラムに対して，排他的に処理を行うことができます。

ただし，CPU ロック状態では，発行可能なサービス・コールは以下に制限されます。

発行可能なサービス・コール	機能概要
ext_tsk	自タスクの終了（CPU ロック状態は解除されます）
loc_cpu，iloc_cpu	CPU ロック状態への移行
unl_cpu，iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_ctx	コンテキスト種別の参照
sns_dpn	ディスパッチ保留状態の参照
vsys_dwn，ivsys_dwn	システム・ダウン

CPU ロック状態は，unl_cpu，iunl_cpu，および ext_tsk によって解除されます。

備考 1　本サービス・コールの発行により変更した CPU ロック状態の解除は，本サービス・コールを発行した処理プログラムが終了する以前に行う必要があります。

備考 2　本サービス・コールでは，ロック要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　RI600V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

備考 4　“ カーネル管理割り込み ” については，「10.1　割り込みの種類」を参照してください。

備考 5　chg_ims によって割り込みマスクを 0 以外に変更している間は，loc_cpu は E_ILUSE エラーを返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iloc_cpu を発行した場合，および非タスクから loc_cpu を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。
E_ILUSE	-28	サービス・コール不正使用 - タスクで chg_ims によって割り込みマスクを 0 以外に変更している状態から本サービス・コールを発行した。

unl_cpu
iunl_cpu

概要

CPU ロック状態の解除

C 言語形式

```
ER      unl_cpu ( void );
ER      iunl_cpu ( void );
```

パラメータ

なし

機能

システムを CPU ロック解除状態へ移行させます。

備考 1 本サービス・コールでは，解除要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が非 CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2 本サービス・コールでは，dis_dsp の発行により変更されたディスパッチ禁止状態の解除処理は行われません。

備考 3 ext_tsk によっても，CPU ロック状態が解除されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから unl_cpu を発行した。 - タスクから iunl_cpu を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

dis_dsp

概要

ディスパッチの禁止

C 言語形式

```
ER      dis_dsp ( void );
```

パラメータ

なし

機能

システムをディスパッチ禁止状態へ移行させます。
ディスパッチ禁止状態では，タスクのスケジューリングが禁止されるため，他のタスクに対して排他的に処理を行うことができます。
ディスパッチ禁止状態へ遷移する操作は，以下の通りです。

- dis_dsp
- chg_ims によって割り込みマスク (PSW.IPL) を 0 以外に変更

また，ディスパッチ禁止状態を解除する操作は，以下の通りです。

- ena_dsp
- ext_tsk
- chg_ims によって割り込みマスク (PSW.IPL) を 0 に変更

備考 1　本サービス・コールの発行により変更したディスパッチ禁止状態の解除は，本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 2　本サービス・コールでは，禁止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ禁止状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　ディスパッチ禁止状態の間に “ 自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）” を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

ena_dsp

概要

ディスパッチ禁止状態の解除

C 言語形式

```
ER      ena_dsp ( void );
```

パラメータ

なし

機能

システムをディスパッチ許可状態へ移行させます。
ディスパッチ禁止状態へ遷移する操作は，以下の通りです。

- dis_dsp
- chg_ims によって割り込みマスク (PSW.IPL) を 0 以外に変更

また，ディスパッチ許可状態へ移行する操作は，以下の通りです。

- ena_dsp
- ext_tsk
- chg_ims によって割り込みマスク (PSW.IPL) を 0 に変更

備考 1　本サービス・コールでは，許可要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ許可状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_ctx

概要

コンテキスト種別の参照

C 言語形式

```
BOOL    sns_ctx ( void );
```

パラメータ

なし

機能

本サービス・コールは，本サービス・コールを発行した処理プログラムのコンテキスト種別（非タスク・コンテキスト，タスク・コンテキスト）を調べます。本サービス・コールは，戻り値として，非タスク・コンテキストの場合は TRUE，タスク・コンテキストの場合は FALSE を返します

戻り値

マクロ	数値	意味
TRUE	1	正常終了（非タスク・コンテキスト）
FALSE	0	正常終了（タスク・コンテキスト）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_loc

概要

CPU ロック状態の参照

C 言語形式

```
BOOL    sns_loc ( void );
```

パラメータ

なし

機能

本サービス・コールは，CPU ロック状態か否かを調べます。本サービス・コールは，戻り値として，CPU ロック状態の場合は TRUE，CPU ロック解除状態の場合は FALSE を返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（CPU ロック状態）
FALSE	0	正常終了（非 CPU ロック状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_dsp

概要

ディスパッチ禁止状態の参照

C 言語形式

```
BOOL        sns_dsp ( void );
```

パラメータ

なし

機能

本サービス・コールは，ディスパッチ禁止状態か否かを調べます。本サービス・コールは，戻り値として，ディスパッチ禁止状態の場合は TRUE，ディスパッチ許可状態の場合は FALSE を返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ禁止状態）
FALSE	0	正常終了（ディスパッチ許可状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_dpn

概要

ディスパッチ保留状態の参照

C 言語形式

```
BOOL    sns_dpn ( void );
```

パラメータ

なし

機能

本サービス・コールは，ディスパッチ保留状態か否かを調べます。本サービス・コールは，戻り値として，ディスパッチ保留状態の場合は TRUE，ディスパッチ保留状態でない場合は FALSE を返します。

なお，以下のいずれかの条件を満たすときをディスパッチ保留状態と呼びます。

- ディスパッチ禁止状態
- CPU ロック状態
- ハンドラなど，PSW.IPL>0 の状態

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ保留状態）
FALSE	0	正常終了（非ディスパッチ保留状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

vsys_dwn
ivsys_dwn

概要

システム・ダウン

C 言語形式

```
void    vsys_dwn ( W type, VW inf1, VW inf2, VW inf3 );
void    ivsys_dwn ( W type, VW inf1, VW inf2, VW inf3 );
```

パラメータ

I/O	パラメータ	説明
I	W type;	エラー種別
I	VW inf1;	システム・ダウン情報 1
I	VW inf2;	システム・ダウン情報 2
I	VW inf3;	システム・ダウン情報 3

機能

システム・ダウン・ルーチン（_RI_sys_dwn__( )）に制御を渡します。
type には，エラー種別として発生したエラーに対応した値 (1 ～ 0x7FFFFFFF) を設定してください。なお，0 以下の値は RI600V4 によって予約されています。
本サービス・コールからリターンすることはありません。
パラメータ仕様の詳細は，「13.2.2 システム・ダウン・ルーチンのパラメータ」を参照してください。
本サービス・コールは μITRON4.0 仕様外の機能です。

備考　RI600V4 内部で異常を検出した場合にも，システム・ダウン・ルーチンが呼び出されます。

戻り値

なし

vsta_knl
ivsta_knl

概要

RI600V4 の起動

C 言語形式

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

パラメータ

なし

機能

RI600V4 を起動します。
本サービス・コールからリターンすることはありません。
本サービス・コールを発行するときは，必ず以下を満たす状態で無ければなりません。

- 全割り込みが受け付けられないこと（例えば，PSW.I == 0）
- CPU の動作モードがスーパバイザ・モード（PSW.PM == 0）であること

本サービス・コールの処理概要を，以下に示します。

1）ISP レジスタを，SI セクションの最終アドレス +1 に初期化
2）INTB レジスタを，cfg600 によって生成された可変ベクタ・テーブル（INTERRUPT_VECTOR セクション）先頭アドレスに初期化
3）システム時刻を 0 に初期化
4）システム・コンフィギュレーション・ファイルで定義された各種オブジェクトの生成
5）スケジューラに制御を移す

本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

なし

18.2.13 割り込み管理機能

以下に，RI600V4 が割り込み管理機能として提供しているサービス・コールの一覧を示します。

表 18 － 15　割り込み管理機能

サービス・コール名	機能概要	発行有効範囲
chg_ims	割り込みマスク・パターンの変更	タスク
ichg_ims	割り込みマスク・パターンの変更	非タスク
get_ims	割り込みマスク・パターンの参照	タスク
iget_ims	割り込みマスク・パターンの参照	非タスク

chg_ims
ichg_ims

概要

割り込みマスクの変更

C 言語形式

```
ER      chg_ims ( IMASK imask );
ER      ichg_ims ( IMASK imask );
```

パラメータ

I/O	パラメータ	説明
I	IMASK imask;	割り込みマスク値

機能

PSW レジスタの IPL ビットを，*imask* で指定された値に変更します。imask には，0 ～ 15 の値を指定できます。

chg_ims サービス・コールでは，imask に 0 以外を指定するとシステムはディスパッチ禁止状態に移行 (dis_dsp と等価) し，imask に 0 を指定するとシステムはディスパッチ許可状態に移行 (ena_dsp と等価) します。

一方，ichg_ims では，ディスパッチ禁止 / 許可状態の遷移はありません。

PSW レジスタの IPL ビットをカーネル割り込みマスクレベル（system_IPL）を超える値に変更している間は，一部のサービス・コールの発行が禁止されます。

発行可能なサービス・コール	機能概要
chg_ims，ichg_ims	割り込みマスクの変更
get_ims，iget_ims	割り込みマスクの参照
vsys_dwn，ivsys_dwn	システム・ダウン
vsta_knl，ivsta_knl	RI600V4 の起動

備考 1　非タスクでは，割り込みマスクを起動時より下げてはなりません。

備考 2　chg_ims　の発行によって変更したディスパッチ禁止状態の解除は，本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 3　ディスパッチ禁止状態の間に " 自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）" を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

備考 4　RI600V4　では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

備考 5　タスクが chg_ims によって PSW.IPL を 0 以外に変更している間は，ena_dsp を呼び出さないようにしてください。ena_dsp　を呼び出すと，その時点でディスパッチ許可状態に遷移します。タスク・ディスパッチが発生すると，PSW はディスパッチ先のタスクの状態に更新されるので，意図せずに IPL 値が下がってしまうことがあります。

備考 6 「10.8　マスカブル割り込みの受け付け禁止」も参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - *imask* ＞ 15
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ichg_ims を発行した。 - 非タスクから chg_ims を発行した。

get_ims
iget_ims

概要

割り込みマスクの参照

C 言語形式

```
ER      get_ims ( IMASK *p_imask );
ER      iget_ims ( IMASK *p_imask );
```

パラメータ

I/O	パラメータ	説明
O	IMASK　　*p_imask;	割り込みマスク値を格納する領域へのポインタ

機能

PSW レジスタの IPL ビット値を，*p_imask* で指定された領域に格納します。

備考 1　本サービス・コールでは，コンテキスト・エラーは検出されません。

備考 2　コンパイラが提供する以下の組み込み関数を使用すると，本サービス・コールよりも高速に PSW レジスタの IPL ビット値を取得することができます。組み込み関数の詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

- get_ipl() ：PSW レジスタの IPL ビットの参照
- get_psw() ：PSW レジスタの参照

戻り値

マクロ	数値	意味
E_OK	0	正常終了

18.2.14 システム構成管理機能

以下に，RI600V4 がシステム構成管理機能として提供しているサービス・コールの一覧を示します。

表 18 － 16　システム構成管理機能

サービス・コール名	機能概要	発行有効範囲
ref_ver	バージョン情報の参照	タスク
iref_ver	バージョン情報の参照	非タスク

ref_ver
iref_ver

概要

バージョン情報の参照

C 言語形式

```
ER      ref_ver ( T_RVER *pk_rver );
ER      iref_ver ( T_RVER *pk_rver );
```

パラメータ

I/O	パラメータ	説明
O	T_RVER *pk_rver;	バージョン情報を格納する領域へのポインタ

【バージョン情報 T_RVER の構造 】

```
typedef struct t_rver {
    UH  maker;                  /* カーネルのメーカ・コード */
    UH  prid;                   /* カーネルの識別番号 */
    UH  spver;                  /*ITRON 仕様のバージョン番号 */
    UH  prver;                  /* カーネルのバージョン番号 */
    UH  prno[4];                /* カーネル製品の管理情報 */
} T_RVER;
```

機能

RI600V4 のバージョン情報を *pk_rver* で指定された領域に格納します。

- *maker*
  maker は，このカーネルを作ったメーカを表します。RI600V4 では，ルネサスエレクトロニクスを意味する 0x011B が返ります。
  なお，*maker* と同じ値がカーネル構成マクロ TKERNEL_MAKER に定義されています。

- *prid*
  prid は，カーネルや VLSI の種類を区別する番号を表します。RI600V4 では，0x0003 が返ります。
  なお，*prid* と同じ値がカーネル構成マクロ TKERNEL_PRID に定義されています。

- *spver*
  spver は，カーネルの準拠する仕様を表します。RI600V4 では，0x5403 が返ります。
  なお，*spver* と同じ値がカーネル構成マクロ TKERNEL_SPVER に定義されています。

- *prver*
  prver は，カーネルのバージョン番号を表します。
  例えば，カーネルのバージョンが V1.02.03 の場合は 0x0123 となります。
  なお，*prver* と同じ値がカーネル構成マクロ TKERNEL_PRVER に定義されています。

- *prno*
  prno は，製品管理情報や製品番号などを表します。RI600V4 では，すべての *prno*[] に 0x0000 が返ります

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 -「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考　タスクから iref_ver を発行した場合，および非タスクから ref_ver を発行した場合，コンテキスト・エラーは検出されず，システムの正常な動作は保証されません。

18.2.15 オブジェクト・リセット機能

以下に，RI600V4 がオブジェクト・リセット機能として提供しているサービス・コールの一覧を示します。

表 18 － 17　オブジェクト・リセット機能

サービス・コール名	機能概要	発行有効範囲
vrst_dtq	データ・キューのリセット	タスク
vrst_mbx	メールボックスのリセット	タスク
vrst_mbf	メッセージ・バッファのリセット	タスク
vrst_mpf	固定長メモリ・プールのリセット	タスク
vrst_mpl	可変長メモリ・プールのリセット	タスク

vrst_dtq

概要

データ・キューのリセット

C 言語形式

```
ER      vrst_dtq ( ID dtqid );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID

機能

dtqid で指定されたデータ・キューをリセットします。
データ・キューに蓄えられていたデータは破棄されます。また，データ送信を待っていたタスクの待ち状態は解除され，そのタスクには戻り値として EV_RST が返されます。

備考 1　本サービス・コールでは，データ受信を待っていたタスクの待ち状態は解除されません。
備考 2　本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *dtqid* ≦ 0 - *dtqid* ＞ VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

vrst_mbx

概要

メールボックスのリセット

C 言語形式

```
ER      vrst_mbx ( ID mbxid );
```

パラメータ

I/O	パラメータ	説明
I	`ID mbxid;`	メールボックスの ID

機能

mbxid で指定されたメールボックスをリセットします。
メールボックスに蓄えられていたメッセージは，RI600V4 の管理から外れます。

備考 1　本サービス・コールでは，メッセージ受信を待っていたタスクの待ち状態は解除されません。
備考 2　本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mbxid* ≦ 0 - *mbxid* ＞ VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

vrst_mbf

概要

メッセージ・バッファのリセット

C 言語形式

```
ER      vrst_mbf ( ID mbfid );
```

パラメータ

I/O	パラメータ	説明
I	ID mbfid;	メッセージ・バッファの ID

機能

mbfid で指定されたメッセージ・バッファをリセットします。
メッセージ・バッファに蓄えられていたメッセージは破棄されます。また，メッセージ送信を待っていたタスクの待ち状態は解除され，そのタスクには戻り値として EV_RST が返されます。

備考 1 本サービス・コールでは，メッセージ受信を待っていたタスクの待ち状態は解除されません。
備考 2 本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mbfid* ≦ 0 - *mbfid* ＞ VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

vrst_mpf

概要

固定長メモリ・プールのリセット

C 言語形式

```
ER      vrst_mpf ( ID mpfid );
```

パラメータ

I/O	パラメータ	説明
I	ID　　mpfid;	固定長メモリ・プールの ID

機能

mpfid で指定された固定長メモリ・プールをリセットします。
固定長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され，戻り値として EV_RST が返されます

備考 1　すでに獲得されていた固定長メモリ・ブロックはすべて，対象固定長メモリ・プールに返却されます。このため，本サービス・コール以降はそれらの固定長メモリ・ブロックにアクセスしてはなりません。

備考 2　本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mpfid* ≦ 0 - *mpfid* ＞ VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

vrst_mpl

概要

可変長メモリ・プールのリセット

C 言語形式

```
ER      vrst_mpl ( ID mplid );
```

パラメータ

I/O	パラメータ	説明
I	`ID mplid;`	可変長メモリ・プールの ID

機能

mplid で指定された可変長メモリ・プールをリセットします。
可変長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され，戻り値として EV_RST が返されます。

備考 1 すでに獲得されていた可変長メモリ・ブロックはすべて，対象可変長メモリ・プールに返却されます。このため，本サービス・コール以降はそれらの可変長メモリ・ブロックにアクセスしてはなりません。

備考 2 本サービス・コールは μITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - *mplid* ≦ 0 - *mplid* ＞ VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

第 19 章　システム・コンフィギュレーション・ファイル

本章では，RI600V4 に提供するデータを保持した情報ファイルを生成する際に必要となるシステム・コンフィギュレーション・ファイルの記述方法について解説しています。

19.1　概　　要

以下に，システム・コンフィギュレーション・ファイルの表記方法を示します。

- コメント
  “//”（連続する 2 個のスラッシュ）から行末までの部分がコメントとして扱われます。

- 数値
  数値は，以下の形式で入力できます。ただし，0xFFFFFFFF を超える数値を指定してはなりません。

  16 進数：　数値の先頭に “0x” か “0X” を付加します。または，数値の最後に 'h' か 'H' を付加します。後者の場合でかつ先頭が英文字 (A ～ F) で始まる場合は先頭に必ず '0' を付加してください。なおここで使用する数値表現で英文字 (A ～ F) は大文字・小文字を識別しません。
  10 進数：　“23” のように整数のみで表します。ただし '0' で始めることはできません。
  8 進数：　数値の先頭に '0' を付加するか数値の最後に 'O' もしくは 'o' を付加します。
  2 進数：　数値の最後に 'B' または 'b' を付加します。ただし '0' で始めることはできません。

- 演算子
  数値に対して，以下の演算子を使用できます。

表 19 － 1　演算子

演算子	優先度	演算方向
()	高	左から右
-（単項マイナス）		右から左
*，/，%		左から右
+，-（二項マイナス）	低	左から右

- シンボル
  シンボルは数字，英大文字，英小文字，'_'（アンダースコア）より構成され，数字以外の文字で始まる文字列で表されます。

- 関数名
  関数名は，数字，英大文字，英小文字，'_' (アンダースコア)，'$'(ドル記号) より構成される数字以外の文字で始まり，“()” で終わる文字列で表されます。
  アセンブリ言語で記述したモジュールを指定する場合は，その先頭ラベルを '_' で始まるように命名し，'_' を除いたものを関数名として指定してください。

- 周波数
  周波数は，数字と '.'(ピリオド) から構成され “MHz” で終わる文字列で表されます。小数点以下は 6 桁が有効です。なお周波数は 10 進数のみで記述可能です。

19.2　デフォルト・システム・コンフィギュレーション・ファイル

多くの定義項目では，ユーザが記述を省略した場合にデフォルト・システム・コンフィギュレーション・ファイルの内容が補われます。デフォルト・システム・コンフィギュレーション・ファイルは，環境変数 LIB600 で指定されるフォルダにある default.cfg です。なお，このファイルを編集してはなりません。

19.3　コンフィギュレーション情報（静的 API）

システム・コンフィギュレーション・ファイルに記述するデータ（コンフィギュレーション情報）は，以下の通りです。

- システム情報（system）
- 基本クロック割り込み情報（clock）
- タスク情報（task[]）
- セマフォ情報（semaphore[]）
- イベントフラグ情報（flag[]）
- データ・キュー情報（dataqueue[]）
- メールボックス情報（mailbox[]）
- ミューテックス情報（mutex[]）
- メッセージ・バッファ情報（message_buffer[]）
- 固定長メモリ・プール情報（memorypool[]）
- 可変長メモリ・プール情報（variable_memorypool[]）
- 周期ハンドラ情報（cyclic_hand[]）
- アラーム・ハンドラ情報（alarm_hand[]）
- 可変ベクタ情報（interrupt_vector[]）
- 固定ベクタ／例外ベクタ情報（interrupt_fvector[]）

19.4　システム情報（system）

ここでは，RI600V4 システム全般に関する情報を定義します。
なお，システム情報として定義可能な数は，1 個に限られます。また，システム情報の定義は省略できません。

形式

< > 内は，ユーザが記述する部分を示します。

```
system {
    stack_size  = <1. システム・スタック・サイズ（stack_size）>;
    priority    = <2. タスク優先度の最大値（priority）>;
    system_IPL  = <3. カーネル割り込みマスクレベル（system_IPL）>;
    message_pri = <4. メッセージ優先度の最大値（message_pri）>;
    tic_deno    = <5. 基本クロック割り込み周期の分母（tic_deno）>;
    tic_nume    = <6. 基本クロック割り込み周期の分子（tic_nume）>;
    context     = <7. タスク・コンテキスト・レジスタ（context）>;
};
```

1） システム・スタック・サイズ（*stack_size*）

- 説明
  サービス・コール処理および割り込み処理で使用するスタックサイズの合計を定義します。
- 定義形式
  数値
- 定義範囲
  8 以上の 4 の倍数
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0x800）を適用

2） タスク優先度の最大値（*priority*）

- 説明
  タスク優先度の最大値を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 32）を適用
- TMAX_TPRI
  cfg600 は, 本設定値を定義したマクロ TMAX_TPRI を, システム情報ヘッダ･ファイル kernel_id.h に出力します。

3） カーネル割り込みマスクレベル（*system_IPL*）

- 説明
  カーネルのクリティカルセクション実行時の割り込みマスクレベル（PSW レジスタの IPL 値）を定義します。これよりも高いレベルの割り込みは，カーネル管理外割り込みの扱いとなります。カーネル管理外割り込みとカーネル管理割り込みについては，「10.1　割り込みの種類」を参照してください。
- 定義形式
  数値
- 定義範囲
  1 ～ 15

- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 7）を適用
- VTKNL_LVL
  cfg600 は，本設定値を定義したマクロ VTKNL_LVL を，システム情報ヘッダ・ファイル kernel_id.h に出力します。

4） メッセージ優先度の最大値（*message_pri*）

- 説明
  メールボックス機能で使用するメッセージの優先度の最大値を定義します。なお，メールボックス機能を使用しない場合は，本項目は意味を持ちません。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 255）を適用
- TMAX_MPRI
  cfg600 は，本設定値を定義したマクロ TMAX_MPRI を，システム情報ヘッダ・ファイル kernel_id.h に出力します。

5） 基本クロック割り込み周期の分母（*tic_deno*）

- 説明
  基本クロック割り込み周期時間は，次の式で算出されます。なお，*tic_deno* と *tic_nume* の少なくともひとつは 1 でなければなりません。

  基本クロック割り込み周期時間（単位：ミリ秒）= *tic_nume* / *tic_deno*
- 定義形式
  数値
- 定義範囲
  1 ～ 100
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用
- 備考
  tic_nume, *tic_deno* の設定に関わらず，サービス・コールで扱う時間の単位は常にミリ秒になります。*tic_nume*, *tic_deno* によって，RI600V4 が管理する時間の精度を規定することになります。
- TIC_DENO
  cfg600 は，本設定値を定義したマクロ TIC_DENO を，システム情報ヘッダ・ファイル kernel_id.h に出力します。

6） 基本クロック割り込み周期の分子（*tic_nume*）

- 説明
  前項を参照してください。
- 定義形式
  数値
- 定義範囲
  1 ～ 65535
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用
- TIC_NUME
  cfg600 は，本設定値を定義したマクロ TIC_NUME を，システム情報ヘッダ・ファイル kernel_id.h に出力します。

7） タスク・コンテキスト・レジスタ（*context*）

- 説明
  タスクが使用するレジスタ・セットを定義します。ここでの設定は，全タスクに適用されます。
- 定義形式
  シンボル
- 定義範囲
  以下から選択してください。

表 19 － 2　system.context

設定値	CPU		FPU	DSP
	PSW, PC, R0 ～ R7, R14, R15	R8 ～ R13	FPSW	アキュムレータ[a]
NO	保証する	保証する	保証しない	保証しない
FPSW	保証する	保証する	保証する	保証しない
ACC	保証する	保証する	保証しない	保証する
FPSW,ACC	保証する	保証する	保証する	保証する
MIN	保証する	保証しない	保証しない	保証しない
MIN,FPSW	保証する	保証しない	保証する	保証しない
MIN,ACC	保証する	保証しない	保証しない	保証する
MIN,FPSW,ACC	保証する	保証しない	保証する	保証する

a. コンパイラオプション “-isa=rxv2” 指定時は ACC0 レジスタと ACC1 レジスタ, その他の場合は ACC0 レジスタ (RXv2 アーキテクチャの場合) または ACC レジスタ (RXv1 アーキテクチャの場合) です。

備考　　コンパイラ・オプション “-isa” は，コンパイラ CC-RX V2.01 以降でサポートされています。

- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は NO）を適用
- 備考
  必ず「19.5　system.context の注意事項」も参照してください。

19.5　system.context の注意事項

ここでは，system.context に関する注意事項を説明します。

19.5.1　FPU，DSP に関する注意事項

system.context に設定すべき値は，FPU，DSP をどのように扱うかによって異なります。
以降では，以下の各ケースについて，system.context の推奨値を示します。推奨以外の設定にした場合は，推奨設定に比べてわずかに RI600V4 の性能が悪化します。

1）FPU および DSP（アキュムレータ）を搭載した MCU を使用する場合
該当 MCU：RX600 シリーズなど

2）FPU を搭載せず，かつ DSP（アキュムレータ）を搭載する MCU を使用する場合
該当 MCU：RX200 シリーズなど

3）FPU を搭載し，かつ DSP（アキュムレータ）を搭載しない MCU を使用する場合
該当 MCU：本マニュアル作成時点では，これに該当する MCU は存在しません。

4）FPU も DSP（アキュムレータ）も搭載しない MCU を使用する場合
該当 MCU：本マニュアル作成時点では，これに該当する MCU は存在しません。

備考　コンパイラが浮動小数点演算命令を出力するのは，オプション “-fpu” 指定時のみです。
アセンブラでオプション “-chkfpu” を指定すると，浮動小数点演算命令の記述を Warning として検出します。
また，コンパイラが DSP 機能命令を出力することはありません。
アセンブラでオプション “-chkdsp” を指定すると，DSP 機能命令の記述を Warning として検出します。

1）FPU および DSP（アキュムレータ）を搭載した MCU を使用する場合

表 19 － 3　FPU および DSP（アキュムレータ）を搭載した MCU を使用する場合

タスクの命令使用状況		system.context の推奨
浮動小数点演算命令	DSP 機能命令	
あり	あり	“FPSW” と “ACC” を含む設定が必須
	なし	“FPSW” を含む設定が必須で，かつ “ACC” を含まない設定を推奨
なし	あり	“ACC” を含む設定が必須で，かつ “FPSW” を含まない設定を推奨
	なし	“FPSW”，“ ACC” を含まない設定を推奨

2） FPU を搭載せず，かつ DSP（アキュムレータ）を搭載する MCU を使用する場合

表 19 － 4　FPU を搭載せず，かつ DSP（アキュムレータ）を搭載する MCU を使用する場合

タスクの命令使用状況		system.context の推奨
浮動小数点演算命令	DSP 機能命令	
あり	あり	（FPU を搭載しない MCU のため，浮動小数点演算命令は使用できません）
	なし	
なし	あり	“FPSW” を含まず，かつ “ACC” を含む設定が必須
	なし	“FPSW” を含まない設定が必須で，かつ “ACC” を含まない設定を推奨

3） FPU を搭載し，かつ DSP（アキュムレータ）を搭載しない MCU を使用する場合

表 19 － 5　FPU を搭載し，かつ DSP（アキュムレータ）を搭載しない MCU を使用する場合

タスクの命令使用状況		system.context の推奨
浮動小数点演算命令	DSP 機能命令	
あり	あり	（DSP を搭載しない MCU のため，DSP 機能命令は使用できません）
	なし	“FPSW” を含み，かつ “ACC” を含まない設定が必須
なし	あり	（DSP を搭載しない MCU のため，DSP 機能命令は使用できません）
	なし	“ACC” を含まない設定が必須で，かつ “FPSW” を含まない設定を推奨

4） FPU も DSP（アキュムレータ）も搭載しない MCU を使用する場合

表 19 － 6　FPU も DSP（アキュムレータ）も搭載しない MCU を使用する場合

タスクの命令使用状況		system.context の推奨
浮動小数点演算命令	DSP 機能命令	
あり	あり	（FPU および DSP を搭載しない MCU のため，浮動小数点演算命令および DSP 機能命令は使用できません）
	なし	
なし	あり	
	なし	“FPSW” と “ACC” を含まない設定が必須

19.5.2　コンパイラ・オプション “-fint_register”，“-base”，および “-pid” との関係

system.context に，“MIN” を含む指定を行うことで，RI600V4 は R8 ～ R13 をタスク・コンテキストとして保存しないようになり，高速化が図れるようになっています。

system.context にこれらの設定をしても良いケースは，コンパイラ・オプション “-fint_register”，“-base”，および “-pid” で R8 ～ R13 をすべて使用する指定をした場合のみです。

その他の場合で system.context に前述の設定をした場合，RI600V4 は正常に動作しません。

【 良い例 】

例 1：-fint_register=4 -base=rom=R8 -base=ram=R9

例 2：-fint_register=3 -base=rom=R8 -base=ram=R9 -base=0x80000=R10

【 悪い例 】

例 3：“-fint_register”, “-base”, “-pid” の指定なし

例 4：-fint_register=4

例 5：-base=rom=R8 -base=ram=R9

例 6：-fint_register=3 -base=rom=R8 -base=ram=R9

19.6　基本クロック割り込み情報（clock）

ここでは，基本クロック割り込みに関する情報を定義します。cfg600 は，本定義に基づいて，基本クロック用タイマ初期化関数（void _RI_init_cmt(void)）が記述された ri_cmt.h を出力します。
なお，基本クロック割り込み情報として定義可能な数は，1 個に限られます。

形式

< > 内は，ユーザが記述する部分を示します。

```
clock {
    timer       = <1. 基本クロック用タイマ・チャネルの選択（timer）>;
    template    = <2. テンプレート・ファイル（template）>;
    timer_clock = <3. CMT 入力周波数（timer_clock）>;
    IPL         = <4. 基本クロック割り込み優先レベル（IPL）>;
};
```

1）基本クロック用タイマ・チャネルの選択（*timer*）

- 説明
  基本クロックに使用するタイマのチャネルを定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれかから選択してください。

表 19 － 7　clock.timer

設定値	説明
CMT0	可変ベクタ 28 に割り当てられた CMT チャネル 0 を使用する。
CMT1	可変ベクタ 29 に割り当てられた CMT チャネル 1 を使用する。
CMT2	可変ベクタ 30 に割り当てられた CMT チャネル 2 を使用する。
CMT3	可変ベクタ 31 に割り当てられた CMT チャネル 3 を使用する。
OTHER	上記以外のタイマを使用する。この場合，ユーザがタイマ初期化ルーチンを作成する必要があります。
NOTIMER	基本クロック割り込みを使用しない。

備考 1　CMT（Compare Match Timer）は，RX MCU に標準的に搭載されているタイマです。

備考 2　使用する RX MCU が CMT チャネル 2, 3 をサポートしていない場合は，”CMT2” および ”CMT3” を選択しないでください。また，CMT チャネル 2，3 に割り当てられた可変ベクタが表 19 － 7 と異なる RX MCU を使用する場合も，”CMT2” および ”CMT3” を選択しないでください。
例えば，RX111 は CMT チャネル 2，3 をサポートしていません。また，RX64M では，CMT チャネル 2，3 に割り当てられた可変ベクタは，30，31 ではありません。

- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は “CMT0”）を適用

2） テンプレート・ファイル（*template*）

- 説明
  CMT（コンペアマッチタイマ）の初期化関数が定義されたテンプレート・ファイルを指定します。
  timer に “NOTIMER” または “OTHER” を指定した場合，本指定は単に無視されます。
  テンプレート・ファイルは，RI600V4 によって提供されます。テンプレートファイルは，今後のバージョンで追加される場合があります。各テンプレート・ファイルがサポートする MCU は，リリースノートを参照してください。
  テンプレートによっては，CMT1, CMT2, CMT3 のいずれかがサポートされない場合があります。*timer* にそのテンプレートで未対応の CMT チャネルを指定した場合，cfg600 はエラーを報告しませんが，cfg600 が出力する ri_cmt.h をインクルードするファイルのコンパイル時にエラーになります。
- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は “rx610.tpl”）を適用

3） CMT 入力周波数（*timer_clock*）

- 説明
  CMT に供給されるクロックの周波数を定義します。PCLK（周辺モジュールクロック）の周波数を指定してください。
- 定義形式
  周波数
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 25MHz）を適用

4） 基本クロック割り込み優先レベル（*IPL*）

- 説明
  基本クロック用タイマの割り込み優先レベルを定義します。
- 定義形式
  数値
- 定義範囲
  1 ～システム情報（system）のカーネル割り込みマスクレベル（system_IPL）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 4）を適用
- VTIM_LVL
  cfg600 は，本設定値を定義したマクロ VTIM_LVL を，システム情報ヘッダ・ファイル kernel_id.h に出力します。

19.7　タスク情報（task[]）

ここでは，各タスクを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
task[ <1. ID番号> ] {
    name          = <2. ID名称（name）>;
    entry_address = <3. タスクの開始アドレス（entry_adderess）>;
    stack_size    = <4. ユーザ・スタック・サイズ（stack_size）>;
    stack_section = <5. ユーザ・スタック領域に付与するセクション名（stack_secion）>;
    priority      = <6. 起動時優先度（priority）>;
    initial_start = <7. TA_ACT属性（initial_start）>;
    exinf         = <8. 拡張情報（exinf）>;
};
```

1） ID 番号

- 説明
  タスク ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称>    <ID番号>
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） タスクの開始アドレス（*entry_adderess*）

- 説明
  タスクの実行開始関数を定義します。
- 定義形式
  関数名
- 定義範囲
  ー

- 省略時の扱い
  省略不可

4) ユーザ・スタック・サイズ（*stack_size*）

- 説明
  ユーザ・スタック・サイズを定義します。
- 定義形式
  数値
- 定義範囲
  以下に示す値以上

表 19 － 8　ユーザ・スタック・サイズの下限値

system.context 設定	コンパイラ・オプション “-isa”	下限値
NO	—	68
FPSW	—	72
ACC	“-isa=rxv2”	92
	“-isa=rxv1” またはオプション “-isa” の指定なし	76
FPSW,ACC	“-isa=rxv2”	96
	“-isa=rxv1” またはオプション “-isa” の指定なし	80
MIN	—	44
MIN,FPSW	—	48
MIN,ACC	“-isa=rxv2”	68
	“-isa=rxv1” またはオプション “-isa” の指定なし	52
MIN,FPSW,ACC	“-isa=rxv2”	72
	“-isa=rxv1” またはオプション “-isa” の指定なし	56

備考　　コンパイラ・オプション “-isa” は，コンパイラ CC-RX V2.01 以降でサポートされています。

- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 256）を適用

5) ユーザ・スタック領域に付与するセクション名（*stack_secion*）

- 説明
  ユーザ・スタック領域に付与するセクション名を定義します。
  cfg600 は，*stack_secion* で指定されたセクションに *stack_size* で指定されたサイズのユーザ・スタック領域を生成します。このセクションのセクション属性は “DATA”，アライメント数は 4 です。リンク時には，このセクションを RAM 領域に配置してください。ただし，0 番地に配置してはなりません。
- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は SURI_STACK）を適用

6) 起動時優先度（*priority*）

- 説明
  タスクの起動時の優先度を定義します。

- 定義形式
  数値
- 定義範囲
  1 ～システム情報（system）のタスク優先度の最大値（priority）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

7 ） TA_ACT 属性（*initial_start*）

- 説明
  タスクの初期状態を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。

  ON :　　TA_ACT 属性を指定する（初期状態を READY 状態とする）
  OFF :　　TA_ACT 属性を指定しない（初期状態を DORMANT 状態とする）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は OFF）を適用

8 ） 拡張情報（*exinf*）

- 説明
  タスクの拡張情報を定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 0xFFFFFFFF
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用
- 備考
  タスクが TA_ACT 属性，act_tsk または iact_tsk によって起動された場合，タスクに拡張情報が渡されます。

19.8　セマフォ情報（semaphore[]）

ここでは，各セマフォを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
semaphore[ <1. ID 番号 > ] {
    name          = <2. ID 名称（name）>;
    max_count     = <3. 最大資源数（max_count）>;
    initial_count = <4. 初期資源数（initial_count）>;
    wait_queue    = <5. 待ちキュー属性（wait_queue）>;
};
```

1） ID 番号

- 説明
  セマフォ ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID 名称 >    <ID 番号 >
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） 最大資源数（*max_count*）

- 説明
  セマフォの最大資源数を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 65535
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

4) 初期資源数（*initial_count*）

- 説明
  セマフォの初期資源数を指定します。
- 定義形式
  数値
- 定義範囲
  0 ～ *max_count*
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

5) 待ちキュー属性（*wait_queue*）

- 説明
  待ちキューの属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  TA_TFIFO： FIFO 順
  TA_TPRI： タスクの現在優先度順
  ただし，同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

19.9　イベントフラグ情報（flag[]）

ここでは，各イベントフラグを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
flag[ <1. ID番号> ] {
    name            = <2. ID名称（name）>;
    initial_pattern = <3. 初期ビット・パターン（initial_pattern）>;
    wait_multi      = <4. 複数待ちの許可属性（wait_multi）>;
    clear_attribute = <5. クリア属性（clear_attribute）>;
    wait_queue      = <6. 待ちキュー属性（wait_queue）>;
};
```

1） ID 番号

- 説明
  イベントフラグ ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称>    <ID番号>
```

- 定義形式
  シンボル
- 定義範囲
  －
- 省略時の扱い
  省略不可

3） 初期ビット・パターン（*initial_pattern*）

- 説明
  イベントフラグの初期ビット・パターンを定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 0xFFFFFFFF
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用

4） 複数待ちの許可属性（*wait_multi*）

- 説明
  複数タスク待ちの許可に関する属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  TA_WSGL： 複数タスクの待ちを許可しない
  TA_WMUL： 複数タスクの待ちを許可する
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_WSGL）を適用

5） クリア属性（*clear_attribute*）

- 説明
  イベントフラグのクリア属性（TA_CLR）を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  NO：　TA_CLR 属性を指定しない
  YES：　TA_CLR 属性を指定する
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は NO）を適用

6） 待ちキュー属性（*wait_queue*）

- 説明
  待ちキューの属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  ただし，TA_CLR 属性が指定されていない場合（*clear_attribute* = NO）は，TA_TPRI を指定した場合でも，待ちキューは FIFO 順に管理されます。この振る舞いは，μITRON4.0 仕様の範囲外です。
  TA_TFIFO： FIFO 順
  TA_TPRI：　タスクの現在優先度順
  　　ただし，同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

19.10 データ・キュー情報（dataqueue[]）

ここでは，各データ・キュー定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
dataqueue[ <1. ID番号 > ] {
    name          = <2. ID名称（name）>;
    buffer_size   = <3. データ数（buffer_size）>;
    wait_queue    = <4. 待ちキュー属性（wait_queue）>;
};
```

1） ID 番号

- 説明
  データ・キュー ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称 >    <ID番号 >
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） データ数（*buffer_size*）

- 説明
  データ・キューに格納可能なデータ数を定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 65535
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用

4） 待ちキュー属性（*wait_queue*）

- 説明
  送信待ちキューの属性を定義します。
  なお，受信待ちキューは常に FIFO 順で管理されます。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  TA_TFIFO： FIFO 順
  TA_TPRI： タスクの現在優先度順
  ただし，同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

19.11 メールボックス情報（mailbox[]）

ここでは，各メールボックスを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
mailbox[ <1. ID 番号 > ] {
    name          = <2. ID 名称（name）>;
    wait_queue    = <3. 待ちキュー属性（wait_queue）>;
    message_queue = <4. メッセージのキューイング属性（message_queue）>;
    max_pri       = <5. 最大メッセージ優先度（max_pri）>;
};
```

1） ID 番号

- 説明
  メールボックス ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID 名称 >    <ID 番号 >
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） 待ちキュー属性（*wait_queue*）

- 説明
  待ちキューの属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  TA_TFIFO： FIFO 順
  TA_TPRI： タスクの現在優先度順
  ただし，同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

4） メッセージのキューイング属性（*message_queue*）

- 説明
  メッセージのキューイング属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  TA_MFIFO：メッセージの送信要求を行った順
  TA_MPRI： メッセージの優先度順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_MFIFO）を適用

5） 最大メッセージ優先度（*max_pri*）

- 説明
  message_queue に TA_MPRI を指定した場合，1 ～ *max_pri* のメッセージ優先度を使用できます。
  なお，*message_queue* に TA_MFIFO を指定した場合，本項目は単に無視されます。
- 定義形式
  数値
- 定義範囲
  1 ～システム情報（system）のタスク優先度の最大値（priority）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

19.12 ミューテックス情報（mutex[]）

ここでは，各ミューテックスを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
mutex[ <1. ID番号> ] {
    name          = <2. ID名称（name）>;
    ceilpri       = <3. 上限優先度（ceilpri）>;
};
```

1） ID 番号

- 説明
  ミューテックス ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

  ```
  #define <ID名称>    <ID番号>
  ```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） 上限優先度（*ceilpri*）

- 説明
  RI600V4 のミューテックスは，簡略化した優先度上限プロトコルを採用しています。ここには，上限優先度を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～システム情報（system）のタスク優先度の最大値（priority）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

19.13 メッセージ・バッファ情報（message_buffer[]）

ここでは，各メッセージ・バッファを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
messsage_buffer[ <1. ID番号> ] {
    name          = <2. ID名称（name）>;
    mbf_size      = <3. バッファ・サイズ（mbf_size）>;
    mbf_section   = <4. メッセージ・バッファ領域に付与するセクション名（mbf_section）>;
    max_msgsz     = <5. 最大メッセージ・サイズ（max_msgsz）>;
};
```

1） ID 番号

- 説明
  メッセージ・バッファ ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

  ```
  #define <ID名称>    <ID番号>
  ```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） バッファ・サイズ（*mbf_size*）

- 説明
  メッセージ・バッファのサイズを，バイト単位で義します。
- 定義形式
  数値
- 定義範囲
  0，または 8 ～ 65532 の範囲の 4 の倍数
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用

4） メッセージ・バッファ領域に付与するセクション名（*mbf_section*）

- 説明
  メッセージ・バッファ領域に付与するセクション名を定義します。
  mbf_size ＞ 0 の場合，cfg600 は *mbf_section* で指定されたセクションに *mbf_size* で指定されたサイズのメッセージ・バッファ領域を生成します。このセクションのセクション属性は “DATA”，アライメント数は 4 です。リンク時には，このセクションを RAM 領域に配置してください。
  リンク時に，このセクションを 0 番地に配置してはなりません。
- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BRI_HEAP）を適用

5） 最大メッセージ・サイズ（*max_msgsz*）

- 説明
  最大メッセージ・サイズをバイト単位で指定します。
  mbf_size ＞ 0 の場合は，*max_msgsz* は（*mbf_size* - 4）以下でなければなりません。
- 定義形式
  数値
- 定義範囲
  1 ～ 65528
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 4）を適用

19.14 固定長メモリ・プール情報（memorypool[]）

ここでは，各固定長メモリ・プールを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
memorypool[ <1. ID番号> ] {
    name          = <2. ID名称（name）>;
    siz_block     = <3. 固定長メモリ・ブロック・サイズ（siz_block）>;
    num_block     = <4. 固定長メモリ・ブロック数（num_block）>;
    section       = <5. メモリ・プール領域に付与するセクション名（section）>;
    wait_queue    = <6. 待ちキュー属性（wait_queue）>;
};
```

1） ID 番号

- 説明
  固定メモリ・プール ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称>    <ID番号>
```

- 定義形式
  シンボル
- 定義範囲
  －
- 省略時の扱い
  省略不可

3） 固定長メモリ・ブロック・サイズ（*siz_block*）

- 説明
  固定長メモリ・ブロック・サイズを，バイト単位で定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 65535
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 256）を適用

4） 固定長メモリ・ブロック数（*num_block*）

- 説明
  固定長メモリ・ブロック数を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 65535
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

5） メモリ・プール領域に付与するセクション名（*section*）

- 説明
  固定長メモリ・プール領域に付与するセクション名を定義します。
  cfg600 は, *section* で指定されたセクションに *siz_block* × *num_block* で算出されたサイズの固定長メモリ・プール領域を生成します。このセクションのセクション属性は “DATA”，アライメント数は 4 です。リンク時には，このセクションを RAM 領域に配置してください。
  リンク時に，このセクションを 0 番地に配置してはなりません。
- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BRI_HEAP）を適用

6） 待ちキュー属性（*wait_queue*）

- 説明
  待ちキューの属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。

  TA_TFIFO： FIFO 順
  TA_TPRI： タスクの現在優先度順
  ただし，同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

19.15 可変長メモリ・プール情報（variable_memorypool[]）

ここでは，各可変長メモリ・プールを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
variable_memorypool[ <1. ID番号> ] {
    name          = <2. ID名称（name）>;
    heap_size     = <3. 可変長メモリ・プールのサイズ（heap_size）>;
    max_memsize   = <4. 可変長メモリ・ブロック・サイズの上限（max_memsize）>;
    mpl_section   = <5. メモリ・プール領域に付与するセクション名（mpl_section）>;
};
```

1） ID 番号

- 説明
  可変メモリ・プール ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

  ```
  #define <ID名称>    <ID番号>
  ```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） 可変長メモリ・プールのサイズ（*heap_size*）

- 説明
  可変長メモリ・プールのサイズを，バイト単位で定義します。
- 定義形式
  数値
- 定義範囲
  24 ～ 0x10000000
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1024）を適用

4） 可変長メモリ・ブロック・サイズの上限（*max_memsize*）

- 説明
  可変長メモリ・ブロックサイズの上限をバイト単位で定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 0xBFFFFF4
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 36）を適用
- 補足
  可変長メモリ・ブロック・サイズについては，「7.3.2　可変長メモリ・ブロックのサイズ」を参照してください。

5） メモリ・プール領域に付与するセクション名（*mpl_section*）

- 説明
  可変長メモリ・プール領域に付与するセクション名を定義します。
  cfg600 は，*mpl_section* で指定されたセクションに *heap_size* バイトの可変長メモリ・プール領域を生成します。このセクションのセクション属性は “DATA”，アライメント数は 4 です。リンク時には，このセクションを RAM 領域に配置してください。
  リンク時に，このセクションを 0 番地に配置してはなりません。
- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BRI_HEAP）を適用

19.16 周期ハンドラ情報（cyclic_hand[]）

ここでは，各周期ハンドラを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
cyclic_hand[ <1. ID番号> ] {
    name             = <2. ID名称（name）>;
    entry_adderess   = <3. 周期ハンドラの開始アドレス（entry_adderess）>;
    interval_counter = <4. 起動周期（interval_counter）>;
    start            = <5. 初期状態（start）>;
    phs_counter      = <6. 起動位相（phs_counter）>;
    phsatr           = <7. TA_PHS属性（phsatr）>;
    exinf            = <8. 拡張情報（exinf）>;
};
```

1） ID 番号

- 説明
  周期ハンドラ ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称>    <ID番号>
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） 周期ハンドラの開始アドレス（*entry_adderess*）

- 説明
  周期ハンドラの実行開始関数を定義します。
- 定義形式
  関数名
- 定義範囲
  ー

- 省略時の扱い
  省略不可

4 ） 起動周期（*interval_counter*）

- 説明
  起動周期をミリ秒単位で定義します。
- 定義形式
  数値
- 定義範囲
  1 ～（0x7FFFFFFF - system.tic_nume）／ system.tic_deno
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

5 ） 初期状態（*start*）

- 説明
  周期ハンドラの初期状態を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  OFF：　停止状態とする（TA_STA 属性なし）
  ON：　動作状態とする（TA_STA 属性あり）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は OFF）を適用

6 ） 起動位相（*phs_counter*）

- 説明
  起動位相をミリ秒単位で定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ *interval_counter*
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用

7 ） TA_PHS 属性（*phsatr*）

- 説明
  起動位相に関する属性を定義します。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  OFF：　起動位相を保存しない（TA_PHS 属性なし）
  ON：　起動位相を保存する（TA_PHS 属性あり）
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は OFF）を適用

8 ） 拡張情報（exinf）

- 説明
  周期ハンドラの拡張情報を定義します。
- 定義形式
  数値

- 定義範囲
  0 ～ 0xFFFFFFFF
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用
- 備考
  周期ハンドラには，拡張情報が渡されます。

19.17 アラーム・ハンドラ情報（alarm_hand[]）

ここでは，各アラーム・ハンドラを定義します。

形式

< > 内は，ユーザが記述する部分を示します。

```
alarm_hand[ <1. ID番号> ] {
    name              = <2. ID名称（name）>;
    entry_adderess    = <3. アラーム・ハンドラの開始アドレス（entry_adderess）>;
    exinf             = <4. 拡張情報（exinf）>;
};
```

1） ID 番号

- 説明
  アラーム・ハンドラ ID 番号を定義します。
- 定義形式
  数値
- 定義範囲
  1 ～ 255
- 省略時の扱い
  cfg600 が ID 番号を自動的に割り当てます。
- 備考
  ID 番号は，1 から漏れなく付与される必要があります。つまり，ID 番号を指定する場合は，その値は定義するオブジェクト数以下でなければなりません。

2） ID 名称（*name*）

- 説明
  ID 名称を定義します。指定された ID 名称は，システム情報ヘッダ・ファイル（kernel_id.h）に以下の形式で出力されます。

```
#define <ID名称>    <ID番号>
```

- 定義形式
  シンボル
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3） アラーム・ハンドラの開始アドレス（*entry_adderess*）

- 説明
  アラーム・ハンドラの実行開始関数を定義します。
- 定義形式
  関数名
- 定義範囲
  ー
- 省略時の扱い
  省略不可

4） 拡張情報（*exinf*）

- 説明
  アラーム・ハンドラの拡張情報を定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 0xFFFFFFFF
- 省略時の扱い
  デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 0）を適用
- 備考
  アラーム・ハンドラには，拡張情報が渡されます。

19.18 可変ベクタ情報（interrupt_vector[]）

ここでは，RX MCU の可変ベクタに設定する割り込みハンドラを定義します。
未定義の割り込みが発生した場合は，システム・ダウンとなります。
なお，cfg600 はここで定義した割り込みに関する割り込み制御レジスタや，割り込み要因等の初期設定のコードは生成しません。初期設定は，アプリケーションで実装いただく必要があります。

備考　ベクタ番号 1 ～ 8 は RI600V4 が使用するため，定義しないでください。また，MCU 仕様で予約となっているベクタには定義しないでください。

形式

< > 内は，ユーザが記述する部分を示します。

```
interrupt_vector[ <1. ベクタ番号 > ] {
    entry_adderess = <2. 割り込みハンドラの開始アドレス（entry_adderess）>;
    os_int         = <3. カーネル管理割り込みの指定（os_int）>;
    pragma_switch  = <4. pragma ディレクティブに渡すスイッチ（pragma_switch）>;
};
```

1）ベクタ番号

- 説明
  ベクタ番号を定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 255
- 省略時の扱い
  省略不可

2）割り込みハンドラの開始アドレス（*entry_adderess*）

- 説明
  割り込みハンドラの実行開始関数を定義します。
- 定義形式
  関数名
- 定義範囲
  ー
- 省略時の扱い
  省略不可

3）カーネル管理割り込みの指定（*os_int*）

- 説明
  この割り込みがカーネル管理割り込みかどうかを定義します。
  カーネル割り込みマスクレベル（system_IPL）以下の割り込み優先レベルの割り込みはカーネル管理割り込み，それ以外の割り込みはカーネル管理外割り込みとして定義する必要があります。
  なお，カーネル割り込みマスクレベル（system_IPL）が 15 の場合は，すべての可変ベクタ割り込みはカーネル管理割り込みとする必要があります。
- 定義形式
  シンボル
- 定義範囲
  以下のいずれか。
  YES：　カーネル管理割り込み

NO： カーネル管理外割り込み

- 省略時の扱い
  省略不可

4） pragma ディレクティブに渡すスイッチ（*pragma_switch*）

- 説明
  cfg600 は，*entry_address* で指定された関数を割り込み関数として扱う #pragma interrupt ディレクティブを，システム情報ヘッダ・ファイル kernel_id.h に出力します。この pragma ディレクティブに渡すスイッチを指定します。
- 定義形式
  シンボル
- 定義範囲
  以下を指定できます。複数指定する場合は，カンマで区切ってください。なお，“ACC” と “NOACC” を同時に指定することはできません。

  E： 多重割り込みを許可する “enable” スイッチを渡します
  F： 高速割り込みを指定する “fint” スイッチを渡します。なお，高速割り込みは必ずカーネル管理外割り込み（os_int=NO）としなければなりません。
  S： 割り込みハンドラで使用するレジスタ数を制限する “save” スイッチを渡します。
  ACC： 割り込みハンドラで ACC レジスタを保証する “acc” スイッチを渡します。
  NOACC： 割り込みハンドラで ACC レジスタを保証しない “no_acc” スイッチを渡します。
- 省略時の扱い
  何もスイッチを渡しません。

備考 1 ACC レジスタの保証については，以下を参照してください。

表 19 － 9 ACC レジスタの保証

pragma_switch の設定	コンパイラ・オプション “-save_acc”	
	なし	あり
“ACC”, “NOACC” ともに無し	“acc”，“no_acc” いずれのスイッチも渡されません。ACC は保証されません。	“acc”，“no_acc” いずれのスイッチも渡されません。ACC は保証されます。
“ACC” あり	“acc” スイッチが渡されます。ACC は保証されます。	
“NOACC” あり	“no_acc” スイッチが渡されます。ACC は保証されません。	

備考 2 基本クロック用タイマ・チャネルの選択（timer）に “CMT0”，“CMT1”，“CMT2”，または “CMT3” のいずれかを指定した場合は，暗黙的に以下の指定で interrupt_vector[] が定義がなされたものと扱います。

- ベクタ番号
  - “CMT0”：28
  - “CMT1”：29
  - “CMT2”：30
  - “CMT3”；31
- *entry_address*：RI600V4 内の基本クロック割り込み処理ルーチンの開始アドレス
- os_int：YES
- *pragma_switch*：E,ACC

19.19 固定ベクタ／例外ベクタ情報（interrupt_fvector[]）

ここでは，RXv1 アーキテクチャの固定ベクタ・テーブル（0xFFFFFF80 ～ 0xFFFFFFFF 番地）／ RXv2 アーキテクチャの例外ベクタ・テーブルを定義します。

MCU によっては，固定ベクタ・テーブル／例外ベクタ・テーブルには，割り込みハンドラのアドレスだけでなく，エンディアン選択レジスタなども含まれています。

固定ベクタ／例外ベクタの割り込み要因は，すべてカーネル管理外割り込みの扱いとなります。

RI600V4 では，ベクタ・アドレスに対して表 19 － 10 のようにベクタ番号を割り当てています。本定義を行わないベクタの扱いも表 19 － 10 に示します。

なお，固定ベクタ・テーブル／例外ベクタ・テーブルの内容は MCU 毎に異なります。詳細は，使用する MCU のハードウエア・マニュアルを参照してください。

また，cfg600 はここで定義した割り込みに関する割り込み制御レジスタや，割り込み要因等の初期設定のコードは生成しません。初期設定は，アプリケーションで実装いただく必要があります。

表 19 － 10　固定ベクタ・テーブル／例外ベクタ・テーブル

ベクタ・アドレス [a]	ベクタ番号	要因の例（MCU 毎に異なります）	定義省略時の扱い
0xFFFFFF80	0	エンディアン選択レジスタ	コンパイラのendianオプションに応じて，以下が設定されます。 - “-endian=little” の場合 0xFFFFFFFF - “-endian=big” の場合 0xFFFFFFF8
0xFFFFFF84	1	（予約領域）	0xFFFFFFFF
0xFFFFFF88	2	オプション機能選択レジスタ 1	
0xFFFFFF8C	3	オプション機能選択レジスタ 0	
0xFFFFFF90	4	（予約領域）	
0xFFFFFF94	5	（予約領域）	
0xFFFFFF98	6	（予約領域）	
0xFFFFFF9C	7	ROM コード・プロテクト（フラッシュ・メモリ）	
0xFFFFFFA0	8	オンチップ・デバッガ ID コード・プロテクト（フラッシュ・メモリ）	
0xFFFFFFA4	9		
0xFFFFFFA8	10		
0xFFFFFFAC	11		
0xFFFFFFB0	12	（予約領域）	
0xFFFFFFB4	13	（予約領域）	
0xFFFFFFB8	14	（予約領域）	
0xFFFFFFBC	15	（予約領域）	

ベクタ・アドレス [a]	ベクタ番号	要因の例（MCU 毎に異なります）	定義省略時の扱い
0xFFFFFFC0	16	（予約領域）	システム・ダウン
0xFFFFFFC4	17	（予約領域）	
0xFFFFFFC8	18	（予約領域）	
0xFFFFFFCC	19	（予約領域）	
0xFFFFFFD0	20	特権命令例外	
0xFFFFFFD4	21	アクセス例外	
0xFFFFFFD8	22	（予約領域）	
0xFFFFFFDC	23	未定義命令例外	
0xFFFFFFE0	24	（予約領域）	
0xFFFFFFE4	25	浮動小数点例外	
0xFFFFFFE8	26	（予約領域）	
0xFFFFFFEC	27	（予約領域）	
0xFFFFFFF0	28	（予約領域）	
0xFFFFFFF4	29	（予約領域）	
0xFFFFFFF8	30	ノンマスカブル割り込み	
0xFFFFFFFC	31	リセット	PowerON_Reset_PC()

a.　表 19 － 10 に記載のベクタ・アドレスは，RXv1 アーキテクチャの固定ベクタ・テーブルのアドレスです。RXv2 アーキテクチャの例外ベクタ・テーブルのアドレスは，EXTB レジスタによって決まります。EXTB レジスタの初期値は，RXv1 アーキテクチャの固定ベクタ・テーブルのアドレスと同じです。2.6.4 節の「FIX_INTERRUPT_VECTOR セクション」も参照してください。

形式

< > 内は，ユーザが記述する部分を示します。

```
interrupt_fvector[ <1. ベクタ番号 > ] {
    entry_adderess = <2. 割り込みハンドラの開始アドレス（entry_adderess）>;
    pragma_switch  = <3. pragma ディレクティブに渡すスイッチ（pragma_switch）>;
};
```

1）ベクタ番号

- 説明
  ベクタ番号を定義します。
- 定義形式
  数値
- 定義範囲
  0 ～ 31
- 省略時の扱い
  省略不可

2） 割り込みハンドラの開始アドレス（*entry_adderess*）

- 説明
  割り込みハンドラの実行開始関数，または固定ベクタ／例外ベクタへの設定値を定義します。
- 定義形式
  関数名または数値
- 定義範囲
  数値の場合は 0 ～ 0xFFFFFFFF
- 省略時の扱い
  省略不可

3） pragma ディレクティブに渡すスイッチ（*pragma_switch*）

- 説明
  cfg600 は，*entry_address* で指定された関数を，割り込み関数として kernel_id.h に #pragma interrupt ディレクティブを出力します。この pragma ディレクティブに渡すスイッチを指定します。
  なお，*entry_address* に数値を指定した場合，およびベクタ番号 31（リセット）については，#pragma interrupt ディレクティブは出力されません。
- 定義形式
  シンボル
- 定義範囲
  以下を指定できます。複数指定する場合は，カンマで区切ってください。なお，“ACC” と “NOACC” を同時に指定することはできません。

S：	割り込みハンドラで使用するレジスタ数を制限する “save” スイッチを渡します。
ACC：	割り込みハンドラで ACC レジスタを保証する “acc” スイッチを渡します。
NOACC：	割り込みハンドラで ACC レジスタを保証しない “no_acc” スイッチを渡します。

- 省略時の扱い
  何もスイッチを渡しません。
- 備考
  ACC レジスタの扱いについては，表 19 － 9 を参照してください。

19.20 RAM 使用量の算出

RI600V4 が使用／管理する RAM 領域は，その用途により，以下の 6 種類のセクションに大別されます。以降で，BRI_RAM，BRI_HEAP，SURI_STACK，および SI セクションについて説明します。

- BRI_RAM セクション：RI600V4 の管理データ，データ・キュー領域
- BRI_HEAP セクション：デフォルトのメッセージ・バッファ領域，固定長メモリ・プール領域，および可変長メモリ・プール領域
- SURI_STACK セクション：デフォルトのユーザ・スタック領域
- SI セクション：システム・スタック領域
- RRI_RAM セクション：RI600V4 の管理データです，サイズは常に 4 バイトです。
- BRI_TRCBUF セクション：本セクションは，［タスク・アナライザ］タブで [ソフトウェア・トレース・モードでトレース・チャートを取得] および [カーネルのバッファ] を選択した場合にのみ生成されます。サイズは，［タスク・アナライザ］タブで指定します。

19.20.1 BRI_RAM セクション

BRI_RAM セクションは，RI600V4 の管理データが割り付けられるセクションです。
表 19 － 11 に，BRI_RAM セクションのメモリ容量計算式（単位：バイト）を示します。なお，実際のサイズは，境界調整のために表 19 － 11 で算出される値よりも大きくなります。

表 19 － 11　BRI_RAM セクションのメモリ容量計算式

種別	メモリ容量計算式（単位：バイト）
システム管理ブロック	36 + 4 × down((*TMAX_TPRI* - 1) / 32 + 1) + *TMAX_TPRI* + *VTMAX_SEM* + 2 × *VTMAX_DTQ* + *VTMAX_FLG* + *VTMAX_MBX* + *VTMAX_MTX* + 2 × *VTMAX_MBF* + *VTMAX_MPF* + *VTMAX_MPL*
タスク管理ブロック	24 × *VTMAX_TSK*
セマフォ管理ブロック	4 × *VTMAX_SEM* + down (*VTMAX_SEM* / 8 + 1) ただし，*VTMAX_SEM* が 0 の場合は，セマフォ管理ブロックのサイズは 0 です。
イベントフラグ管理ブロック	8 × *VTMAX_FLG* + 2 × down (*VTMAX_FLG* / 8 + 1) ただし，*VTMAX_FLG* が 0 の場合は，イベントフラグ管理ブロックのサイズは 0 です。
データ・キュー管理ブロック	6 × *VTMAX_DTQ*+ down (*VTMAX_DTQ* / 8 + 1) + *DTQ_ALLSIZE* ただし，*VTMAX_DTQ* が 0 の場合は，データ・キュー管理ブロックのサイズは 0 です。
メールボックス管理ブロック	8 × *VTMAX_MBX* + 2 × down (*VTMAX_MBX* / 8 + 1) ただし，*VTMAX_MBX* が 0 の場合は，メールボックス管理ブロックのサイズは 0 です。
ミューテックス管理ブロック	*VTMAX_MTX* + down (*VTMAX_MTX* / 8 + 1) ただし，*VTMAX_MTX* が 0 の場合は，ミューテックス管理ブロックのサイズは 0 です。
メッセージ・バッファ管理ブロック	16 × *VTMAX_MBF*
固定長メモリ・プール管理ブロック	8 × *VTMAX_MPF* + down (*VTMAX_MPF* / 8 + 1) + Σ (down(memorypool[].num_block / 8 + 1)) ただし，*VTMAX_MPF* が 0 の場合は，固定長メモリ・プール管理ブロックのサイズは 0 です。
可変長メモリ・プール管理ブロック	208 × *VTMAX_MPL*
周期ハンドラ管理ブロック	8 × *VTMAX_CYH*
アラーム・ハンドラ管理ブロック	8 × *VTMAX_ALH*
［タスク・アナライザ］タブで「ハードウェア・トレース・モードでトレース・チャートを取得」を選択	4
［タスク・アナライザ］タブで「ソフトウェア・トレース・モードでトレース・チャートを取得」を選択	28
［タスク・アナライザ］タブで「ソフトウェア・トレース・モードで長時間統計を取得」を選択	1592 + 8 × (*VTMAX_TSK* + 1)

備考　メモリ容量計算式のキーワードは，以下に示した意味を持ちます。

TMAX_TPRI：システム情報（system）のタスク優先度の最大値（priority）設定値です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_TSK：タスク情報（task[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_SEM：セマフォ情報（semaphore[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_FLG：イベントフラグ情報（flag[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_DTQ：データ・キュー情報（dataqueue[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

DTQ_ALLSIZE：データ・キュー領域のサイズの合計です。具体的には，以下によって算出されます。
Σ dataqueue[].buffer_size × 4
ただし，この計算結果が 0 になる場合は，*DTQ_ALLSIZE* は 4 です。

VTMAX_MBX：メールボックス情報（mailbox[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_MTX：ミューテックス情報（mutex[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_MBF：メッセージ・バッファ情報（message_buffer[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_MPF：固定長メモリ・プール情報（memorypool[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_MPL：可変長メモリ・プール情報（variable_memorypool[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_CYH：周期ハンドラ情報（cyclic_hand[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

VTMAX_ALH：アラーム・ハンドラ情報（alarm_hand[]）の定義総数です。
cfg600 は，システム情報ヘッダ・ファイル kernel_id.h にこの名称のマクロを出力します。

19.20.2 BRI_HEAP セクション

BRI_HEAP セクションは，メッセージ・バッファ領域，固定長メモリ・プール領域，および可変長メモリ・プール領域が割り付けられるセクションです。なお，メッセージ・バッファ，固定長メモリ・プール，および可変長メモリ・プールの定義時に , ぞれぞれの領域をユーザ指定のセクションに割り当てることもできます。

BRI_HEAP セクションのサイズは，以下の合計です。

- メッセージ・バッファ領域の合計サイズ
  メッセージ・バッファ情報（message_buffer[]）で “mbf_section” を省略したメッセージ・バッファ定義について，以下の式で算出されます。

  Σ message_buffer[].mbf_size

- 固定長メモリ・プール領域の合計サイズ
  固定長メモリ・プール情報（memorypool[]）で “section” を省略した固定長メモリ・プール定義について，以下の式で算出されます。

  Σ （memorypool[].siz_block × memorypool[].num_block）

- 可変長メモリ・プール領域の合計サイズ
  可変長メモリ・プール情報（variable_memorypool[]）で “mpl_section” を省略した可変長メモリ・プール定義について，以下の式で算出されます。

  Σ variable_memorypool[].heap_size

19.20.3 SURI_STACK セクション

SURI_STACK セクションは，各タスクのユーザ・スタック領域が割り付けられるセクションです。なお，ユーザ・スタック領域は，システム・コンフィギュレーション・ファイルでの定義時にユーザ定義のセクションに割り当てることもできます。

SURI_STACK セクションのサイズは，タスク情報（task[]）で “stack_section” を省略したタスク定義について，以下の式で算出されます。

Σ task[].stack_size

備考　　スタック使用量の見積りについては，「付録 D　スタック使用量の算出」を参照してください。

19.20.4 SI セクション

SI セクションは，システム・スタック領域が割り付けられるセクションです。

SI セクションのサイズは，システム情報（system）のシステム・スタック・サイズ（stack_size）です。

備考　　スタック使用量の見積りについては，「付録 D　スタック使用量の算出」を参照してください。

19.21 記 述 例

以下に，システム・コンフィギュレーション・ファイルの記述例を示します。

```
// System Definition
system{
    stack_size  = 1024;
    priority    = 10;
    system_IPL  = 4;
    message_pri = 1;
    tic_deno    = 1;
    tic_nume    = 1;
    context     = FPSW,ACC;
};

//System Clock Definition
clock{
    template    = rx610.tpl;    // Please modify when you use other than RX610
    timer       = CMT0;         // Please modify for your H/W environment
    timer_clock = 25MHz;        // Please modify for your H/W environment
    IPL         = 3;            // Please modify for your H/W environment
};

//Task Definition
task[]{
    name            = ID_TASK1;
    entry_address   = task1();
    initial_start   = ON;
    stack_size      = 512;
    priority        = 1;
//  stack_section   = STK1;
    exinf           = 1;
};

task[]{
    name            = ID_TASK2;
    entry_address   = task2();
    initial_start   = ON;
    stack_size      = 512;
    priority        = 2;
//  stack_section   = STK2;
    exinf           = 2;
};

// Semaphore Definition
semaphore[]{
    name            = ID_SEM1;
    max_count       = 1;
    initial_count   = 1;
    wait_queue      = TA_TPRI;
};

// Cyclic Handler Definition
cyclic_hand[] {
    name            = ID_CYC1;
    entry_address   = cyh1();
    interval_counter = 100;
    start           = ON;
    phsatr          = OFF;
    phs_counter     = 100;
    exinf           = 1;
};
```

```
// Alarm Handler (dummy) Definition
alarm_hand[] {
    name            = ID_ALM1;
    entry_address   = alh1();
    exinf           = 1;
};

// Interrupt Handler for "Taking in trace chart by software trace mode"
//  Please remove the commnets when "Taking in trace chart by software trace mode"
//   is selected.

// interrupt_vector[29]{               // CMT CH1
//     os_int          = NO;
//     entry_address   = _RIUSR_trcSW_interrupt();      // in trcSW_cmt.src
// };

// Interrupt Handler for "Taking in long-statistics by software trace mode"
//  Please remove the commnets when "Taking in long-statistics by software trace
//  mode" is selected.
// interrupt_vector[29]{               // CMT CH1
//     os_int          = NO;
//     entry_address   = _RIUSR_trcLONG_interrupt();    // in trcLONG_cmt.src
// };

// Interrupt Handler (dummy) Definition
interrupt_vector[64]{
    os_int          = YES;
    entry_address   = inh64();
    pragma_switch   = E;
};
```

備考　　RI600V4 では，システム・コンフィギュレーション・ファイルのサンプル・ソース・ファイルを提供しています。

第 20 章　コンフィギュレータ cfg600

本章では，コンフィギュレータ cfg600 について解説しています。

20.1　概　　要

RI600V4 が提供している機能を利用したシステム（ロード・モジュール）を構築する場合，RI600V4 に提供するデータを保持した情報ファイルが必要となります。

基本的に情報ファイルは，規定された形式のデータ羅列であるため，各種エディタを用いて記述することは可能です。しかし，情報ファイルは，記述性／可読性の面で劣ったものとなっているため，記述に際してはかなりの時間と労力を必要とします。

そこで，RI600V4 では，記述性／可読性の面で優れたシステム・コンフィギュレーション・ファイルから情報ファイルへと変換するユーティリティ・ツール（コンフィギュレータ cfg600）を提供しています。

cfg600 は，システム・コンフィギュレーション・ファイルを入力ファイルとして読み込んだあと，情報ファイルを出力します。

以下に，cfg600 が出力する情報ファイルについて示します。

- システム情報ヘッダ・ファイル（kernel_id.h）
  システム・コンフィギュレーション・ファイルに記述されたオブジェクト名（タスク名，セマフォ名など）と ID の対応付けを保持した情報ファイルです。アプリケーションからインクルードします。

- サービス・コール定義ファイル（kernel_sysint.h）
  サービス・コールを INT 命令を用いて呼び出すための宣言が記述されたファイルです。kernel.h からインクルードされます。

- ROM 定義ファイル（kernel_rom.h），RAM 定義ファイル（kernel_ram.h）
  RI600V4 の管理データが記述されたファイルです。これらのファイルは，ブート処理のソースでのみ，インクルードする必要があります。詳細は，「16.2.1　ブート処理関数（PowerON_Reset_PC( )）」を参照してください。

- システム定義ファイル（ri600.inc）
  システム定義ファイルは，mkritbl が出力するテーブル・ファイル（ritable.src）からインクルードされます。

- ベクタ・テーブル・テンプレート・ファイル（vector.tpl）
  ベクタ・テーブル・テンプレート・ファイルは，mkritbl への入力ファイルとなります。

- CMT タイマ定義ファイル（ri_cmt.h）
  基本クロック割り込み情報（clock）の基本クロック用タイマ・チャネルの選択（timer）に CMT0，CMT，CMT2，CMT3 のいずれかを指定した場合は，テンプレート・ファイル（template）で指定されたテンプレートファイルが環境変数 LIB600 から検索され，ri_cmt.h にリネームされて出力されます。CMT タイマ定義ファイルは，ブート処理のソースでのみ，インクルードする必要があります。詳細は，「16.2.1　ブート処理関数（PowerON_Reset_PC( )）」を参照してください。

20.2　cfg600 の起動

20.2.1　コマンド・ラインからの起動

事前に，環境変数 “LIB600” を，“<ri_root>¥lib600” に設定する必要があります。
以下に，cfg600 をコマンド・ラインから起動する際の起動方法を示します。
ただし，入力例中の “C>” はコマンド・プロンプトを，“ △ ” はスペース・キーの入力を，“［Enter］” はエンター・キーの入力を表しています。
また，“[　]” で囲まれたオプションは，省略可能なオプションであることを表しています。

```
C>  cfg600.exe △ [-U] △ [-v] △ [ -V] △ file［Enter］
```

出力ファイルは，カレント・フォルダに生成されます。

以下に，各オプションの詳細を示します。

- -U
  未定義割り込みが発生した時にはシステム・ダウンとなりますが，本オプションを指定すると，発生した割り込みのベクタ番号がシステム・ダウン・ルーチンに渡されるようになる（「第 13 章　システム・ダウン」を参照）ため，デバッグに役立ちます。ただし，RI600V4 のコード・サイズが約 1.5KB 増加します。

- -v
  コマンドのオプションの説明と詳細なバージョンを表示します。

- - V
  cfg600 が生成するファイルの作成状況を表示します。

- *file*
  cfg600 への入力ファイル名（システム・コンフィギュレーション・ファイル名）を指定します。拡張子を省略した場合は，拡張子 “.cfg” を補って解釈します。

備考　<ri_root> は，RI600V4 のインストール・フォルダを表しています。
デフォルトでは，“ C:¥Program Files¥Renesas Electronics¥CubeSuite+¥RI600V4” となります。

20.2.2　CubeSuite+ からの起動

プロパティ パネルの［システム・コンフィギュレーション・ファイル関連情報］タブで設定した内容に基づき，CubeSuite+ のビルド時に起動されます。

第 21 章　テーブル生成ユーティリティ mkritbl

本章では，テーブル生成ユーティリティ mkritbl について解説しています。

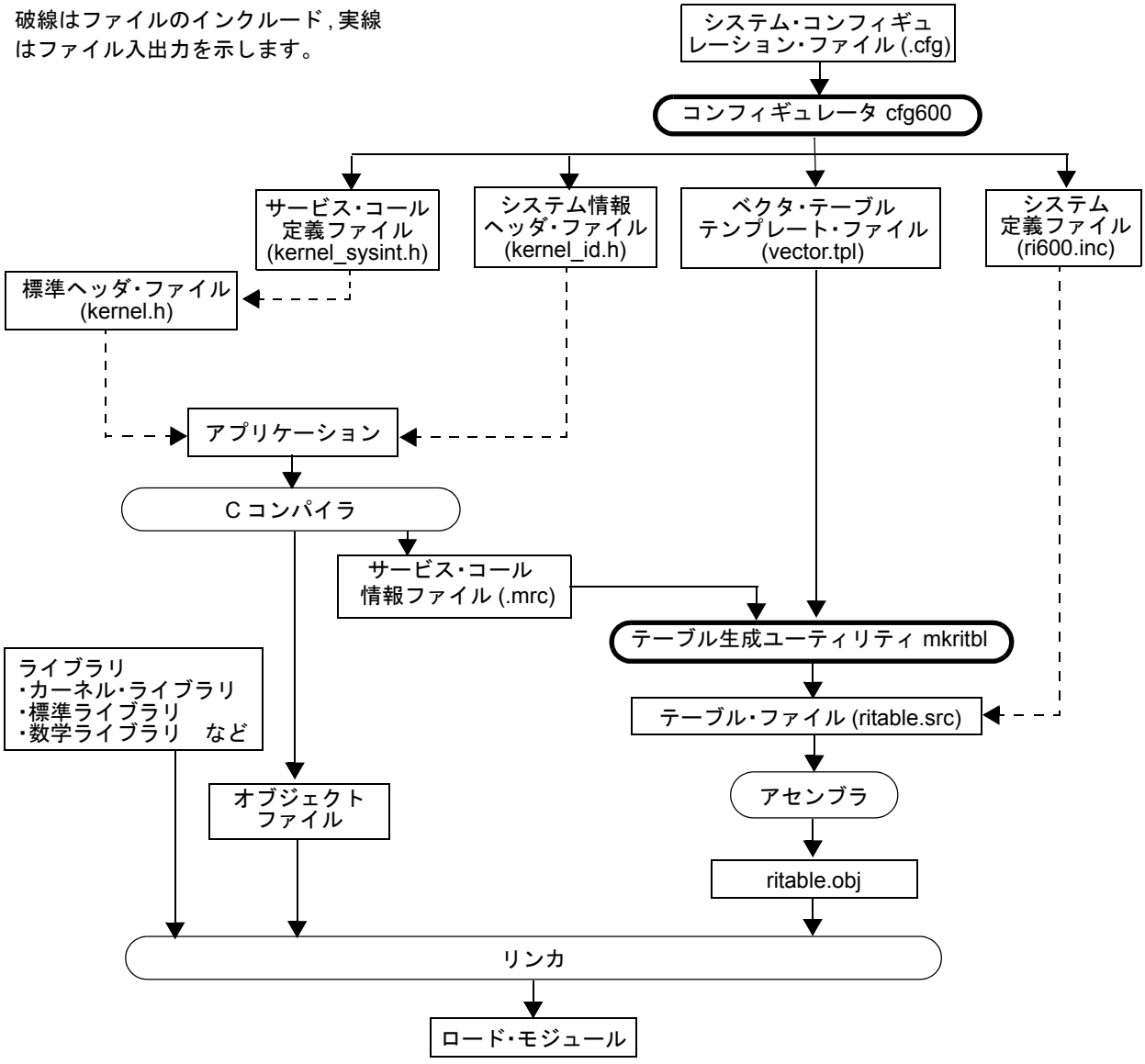
21.1　概　　要

mkritbl は，アプリケーションで使用しているサービス・コール情報を収集して，サービス・コール・テーブルと割り込みベクタ・テーブルを生成するコマンド・ライン・ツールです。

アプリケーションのコンパイルによって，アプリケーション内で使用しているサービス・コール情報がサービス・コール情報ファイル（.mrc）に出力されます。mkritbl は，これらのサービス・コール情報ファイルを入力として，システムで使用するサービス・コールだけがリンクされるようにサービス・コール・テーブルを生成します。

また，mkritbl は cfg600 が出力したベクタ・テーブル・テンプレート・ファイル（vector.tpl）とサービス・コール情報ファイルを元に，割り込みベクタ・テーブルを生成します。

図 21 － 1　mkritbl の概要



21.2　mkritbl の起動

21.2.1　コマンド・ラインからの起動

事前に，環境変数 “LIB600” を，“<ri_root>¥lib600” に設定する必要があります。
以下に，mkritbl をコマンド・ラインから起動する際の起動方法を示します。
ただし，入力例中の “C>” はコマンド・プロンプトを，“ △ ” はスペース・キーの入力を，“［Enter］” はエンター・キーの入力を表しています。
また，“[　]” で囲まれたオプションは，省略可能なオプションであることを表しています。

```
C>  mkritbl.exe △ [path]［Enter］
```

出力ファイルは，カレント・フォルダに生成されます。

以下に，各オプションの詳細を示します。

- *path*
  サービス・コール情報ファイル，またはサービス・コール情報ファイルを検索するフォルダのパスを指定します。なお，フォルダのパスを指定した場合，下位フォルダは検索されません。
  mkritbl は，*path* 指定の有無に関わらず，カレント・フォルダを検索対象とします。

備考　<ri_root> は，RI600V4 のインストール・フォルダを表しています。
デフォルトでは，“ C:¥Program Files¥Renesas Electronics¥CubeSuite+¥RI600V4” となります。

21.2.2　CubeSuite+ からの起動

プロパティ パネルの［システム・コンフィギュレーション・ファイル関連情報］タブで設定した内容に基づき，CubeSuite+ のビルド時に起動されます。

21.3　注意事項

「2.6.1　サービス・コール情報ファイルとコンパイラ・オプション “-ri600_preinit_mrc”」を参照してください。

付録 A　ウインドウ・リファレンス

本付録では，コンフィギュレータ cfg600 とテーブル生成ユーティリティ mkritbl の起動オプションを統合開発環境 CubeSuite+ から設定する際に必要となるウインドウ／パネルについて説明しています。

A.1　説　　明

以下に，ウインドウ／パネルの一覧を示します。

表 A－1　ウインドウ／パネルの一覧

ウインドウ／パネル名	機能概要
メイン・ウインドウ	CubeSuite+ を起動した際，最初にオープンするウインドウ
プロジェクト・ツリー パネル	プロジェクトの構成要素のツリー表示
プロパティ パネル	プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等について，詳細情報の表示，および設定の変更

メイン・ウインドウ

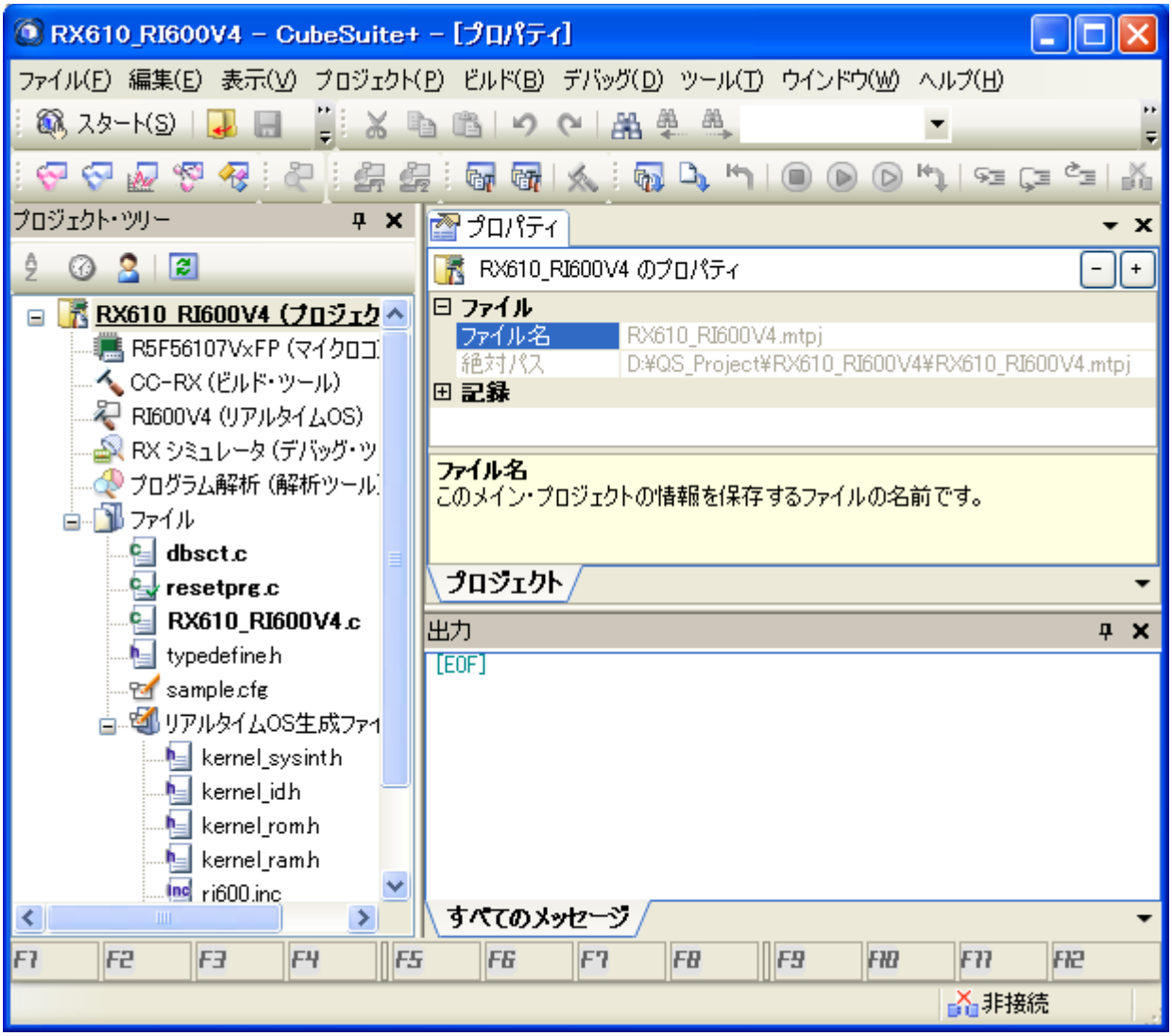
概要

CubeSuite+ を起動した際，最初にオープンするウインドウです。
ビルドを行う際は，本ウインドウからユーザ・プログラムの実行制御，および各パネルのオープンを行います。

なお，本ウインドウは，次の方法でオープンすることができます。

- Windows の［スタート］→［すべてのプログラム］→［Renesas Electronics CubeSuite+］→［CubeSuite+］を選択

表示イメージ



機能

1）メニューバー

リアルタイム OS 関連のメニューを示します。
なお，各メニューから引き出される項目は，ユーザ設定 ダイアログでカスタマイズすることができます。

- ［表示］

リアルタイム OS	リアルタイム OS の各ツールを起動するためのカスケード・メニューを表示します。
リソース情報	リアルタイム OS リソース情報 パネルをオープンします。 なお，本メニューは，デバッグ・ツールと切断時は無効となります。
タスク・アナライザ 1	リアルタイム OS タスク・アナライザ 1 パネルを開きます。 なお，本メニューは，デバッグ・ツールと切断時は無効となります。
タスク・アナライザ 2	リアルタイム OS タスク・アナライザ 2 パネルを開きます。 なお，本メニューは，デバッグ・ツールと切断時は無効となります。

2）ツールバー

リアルタイム OS 関連のボタン群を示します。
なお，ツールバー上のボタンは，ユーザ設定 ダイアログでカスタマイズすることができます。また，同ダイアログにより，新規にツールバーを作成することもできます。

- リアルタイム OS ツールバー

	リアルタイム OS リソース情報 パネルをオープンします。 なお，本ボタンは，デバッグ・ツールと切断時は無効となります。

- リアルタイム OS タスク・アナライザ ツールバー

	リアルタイム OS タスク・アナライザ 1 パネルを開きます。 なお，本ボタンは，デバッグ・ツールと切断時は無効となります。
	リアルタイム OS タスク・アナライザ 2 パネルを開きます。 なお，本ボタンは，デバッグ・ツールと切断時は無効となります。

3）パネル表示エリア

以下のパネルを表示するエリアです。

- プロジェクト・ツリー パネル
- プロパティ パネル
- 出力 パネル

表示内容の詳細については，各パネルの項を参照してください。

備考　出力 パネルについての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

プロジェクト・ツリー パネル

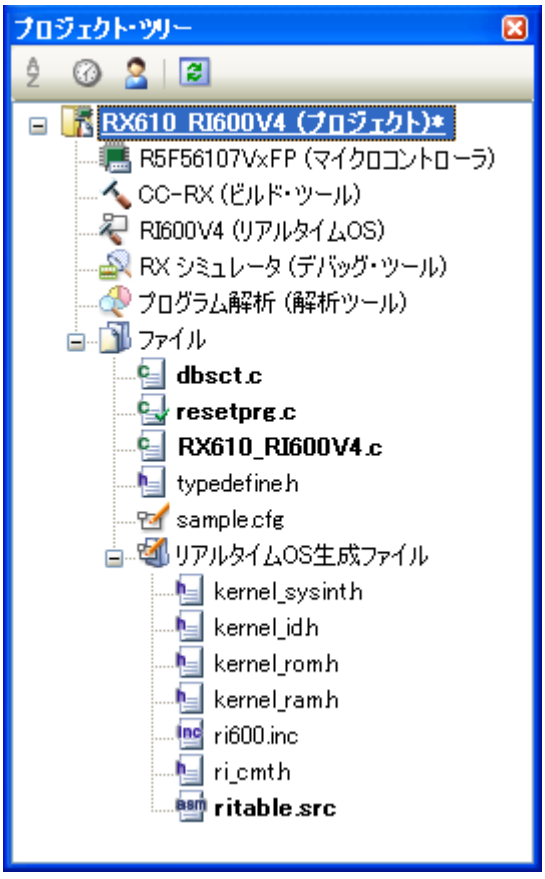
概要

プロジェクトを構成するリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等の構成要素をツリー表示します。

なお，本パネルは，次の方法でオープンすることができます。

- ［表示］メニュー→［プロジェクト・ツリー］を選択

表示イメージ



機能

1） プロジェクト・ツリー エリア

プロジェクトの構成要素を以下のノードでツリー表示します。

ノード	説明
RI600V4（リアルタイム OS） (“ リアルタイム OS ノード ” と呼びます。)	使用するリアルタイム OS です。
xxx.cfg	システム・コンフィギュレーション・ファイルです。
リアルタイム OS 生成ファイル (“リアルタイムOS生成ファイル･ノード”と呼びます。)	システム・コンフィギュレーション・ファイル追加時に作成されるノードで，以下の情報ファイルが直下に表示されます。 - システム情報ヘッダ・ファイル（kernel_id.h） - サービス・コール定義ファイル（kernel_sysint.h） - ROM 定義ファイル（kernel_rom.h） - RAM 定義ファイル（kernel_ram.h） - システム定義ファイル（ri600.inc） - CMT タイマ定義ファイル（ri_cmt.h） - テーブル・ファイル（ritable.src） 本ノード，および本ノードに表示されているファイルを直接削除することはできません。 システム・コンフィギュレーション・ファイルをプロジェクトから外した場合，本ノード，および本ノードに表示されているファイルは表示されなくなります。

コンテキスト・メニュー

1） リアルタイム OS ノード，リアルタイム OS 生成ファイル・ノードを選択している場合

プロパティ	選択しているノードのプロパティをプロパティ パネルに表示します。

2） システム・コンフィギュレーション・ファイル，情報ファイルを選択している場合

アセンブル	選択しているアセンブラ・ソース・ファイルをアセンブルします。 なお，本メニューは，システム情報テーブル・ファイル，エントリ・ファイルを選択している場合のみ表示されます。 ただし，ビルド・ツールが実行中の場合は無効となります。
開く	ファイルの拡張子に割り当てられたアプリケーションで選択しているファイルをオープンします。 なお，本メニューは，複数のファイルを選択している場合は無効となります。
内部エディタで開く ...	エディタ パネルで選択しているファイルをオープンします。 なお，本メニューは，複数のファイルを選択している場合は無効となります。
アプリケーションを指定して開く ...	プログラムから開く ダイアログをオープンし，指定したアプリケーションで選択しているファイルをオープンします。 ただし，複数のファイルを選択している場合は無効となります。
エクスプローラでフォルダを開く	選択しているファイルが存在しているフォルダをエクスプローラでオープンします。
追加	プロジェクトにファイル，カテゴリ・ノードを追加するためのカスケード・メニューを表示します。
既存のファイルを追加 ...	既存のファイルを追加 ダイアログをオープンし，選択したファイルをプロジェクトに追加します。
新しいファイルを追加 ...	ファイル追加 ダイアログをオープンし，選択した種類でファイルを作成し，プロジェクトに追加します。
新しいカテゴリを追加	選択しているファイルと同じレベルにカテゴリ・ノードを追加し，カテゴリ名が編集可能な状態になります。 なお，本メニューは，ビルド・ツールが実行中の場合，およびカテゴリのネスト数が 20 の場合は無効となります。
プロジェクトから外す	選択しているファイルをプロジェクトから外します。 ファイル自体はファイル・システム上からは削除されません。 なお，本メニューは，ビルド・ツールが実行中の場合は無効となります。
コピー	選択しているファイルをクリップ・ボードにコピーします。 ファイル名を編集中の場合は，選択している文字列をクリップ・ボードにコピーします。
貼り付け	本メニューは常に無効です。
名前の変更	選択しているファイルの名前が編集可能な状態になります。 実際のファイル名も変更されます。
プロパティ	選択しているファイルのプロパティをプロパティ パネルに表示します。

プロパティ パネル

概要

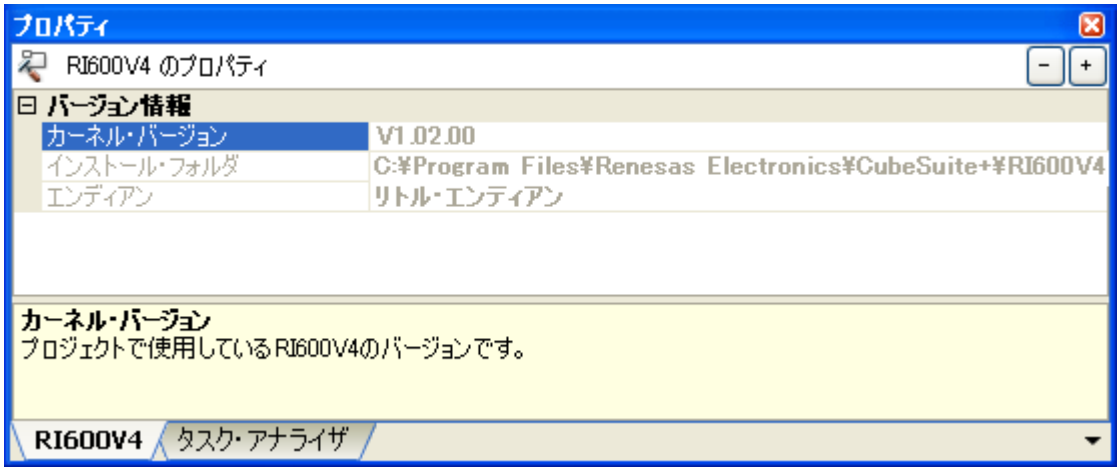
プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等について，カテゴリ別に詳細情報の表示，および設定の変更を行います。

なお，本パネルは，次の方法でオープンすることができます。

- プロジェクト・ツリー パネル上において，リアルタイム OS ノード，システム・コンフィギュレーション・ファイル等を選択したのち，［表示］メニュー→［プロパティ］を選択，またはコンテキスト・メニュー→［プロパティ］を選択

備考　すでにプロパティ パネルがオープンしている場合，プロジェクト・ツリー パネル上において，リアルタイム OS ノード，システム・コンフィギュレーション・ファイル等を選択することで，選択した項目の詳細情報を表示します。

表示イメージ



機能

1）対象名エリア

プロジェクト・ツリー パネルで選択しているノードの名称を表示します。
複数のノードを選択している場合，本エリアは空欄となります。

2）詳細情報表示／変更エリア

プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等の詳細情報を，カテゴリ別のリスト形式で表示し，設定の変更を直接行うことができるエリアです。
⊟マークは，そのカテゴリ内に含まれているすべての項目が展開表示されていることを示し，また，⊞マークは，カテゴリ内の項目が折りたたみ表示されていることを示します。展開／折りたたみ表示の切り替えは，このマークのクリック，またはカテゴリ名のダブルクリックにより行うことができます。
カテゴリ，およびそれに含まれる項目の表示内容／設定方法についての詳細は，該当するタブの項を参照してください。

3） プロパティの説明エリア

詳細情報表示／変更エリアで選択したカテゴリや項目の簡単な説明を表示します。

4） タブ選択エリア

タブを選択することにより，詳細情報を表示するカテゴリが切り替わります。
本パネルには，次のタブが存在します（各タブ上における表示内容／設定方法についての詳細は，該当するタブの項を参照してください）。

- プロジェクト・ツリー パネルでリアルタイム OS ノードを選択している場合
  - ［RI600V4］タブ
  - ［タスク・アナライザ］タブ
- プロジェクト・ツリー パネルでシステム・コンフィギュレーション・ファイルを選択している場合
  - ［システム・コンフィギュレーション・ファイル関連情報］タブ
  - ［ファイル情報］タブ
- プロジェクト・ツリー パネルでリアルタイム OS 生成ファイル・ノードを選択している場合
  - ［カテゴリ情報］タブ
- プロジェクト・ツリー パネルでシステム情報テーブル・ファイル，エントリ・ファイルを選択している場合
  - ［ビルド設定］タブ
  - ［個別アセンブル・オプション］タブ
  - ［ファイル情報］タブ
- プロジェクト・ツリー パネルでシステム情報ヘッダ・ファイルを選択している場合
  - ［ファイル情報］タブ

備考 1　［ファイル情報］タブ，［カテゴリ情報］タブ，［ビルド設定］タブ，［個別アセンブル・オプション］タブについての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

備考 2　プロジェクト・ツリー パネルで複数の構成要素を選択している場合は，その構成要素に共通するタブのみ表示されます。プロパティの値の変更は，選択している複数の構成要素に共通に反映されます。

［編集］メニュー（プロパティ パネル専用部分）

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。
切り取り	プロパティの値を編集中の場合，選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集中の場合，クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集中の場合，選択している文字列を削除します。
すべて選択	プロパティの値を編集中の場合，選択しているプロパティの値文字列をすべて選択します。

コンテキスト・メニュー

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。

切り取り	プロパティの値を編集中の場合，選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集中の場合，クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集中の場合，選択している文字列を削除します。
すべて選択	プロパティの値を編集中の場合，選択しているプロパティの値文字列をすべて選択します。
デフォルトに戻す	選択している項目の設定値をプロジェクトに設定しているデフォルト値に戻します。 ただし，［個別アセンブル・オプション］タブにおいては，全体オプションの設定値に戻します。
すべてデフォルトに戻す	現在表示しているタブの設定値をすべてプロジェクトに設定しているデフォルト値に戻します。 ただし，［個別アセンブル・オプション］タブにおいては，全体オプションの設定値に戻します。

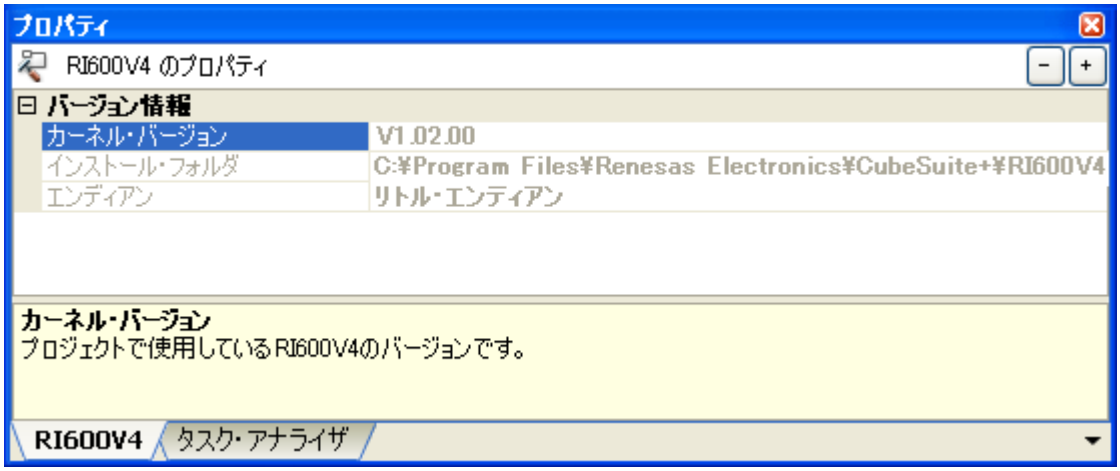
［RI600V4］タブ

概要

本タブでは，使用する RI600V4 に対して，次に示すカテゴリごとに詳細情報の表示を行います。

- バージョン情報

表示イメージ



機能

1）［バージョン情報］

RI600V4 のバージョンに関する詳細情報の表示を行います。

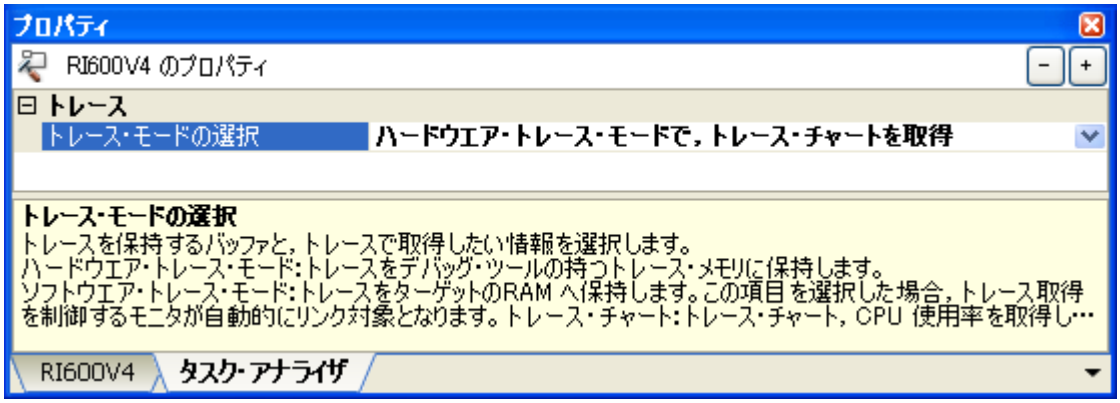
カーネル・バージョン	使用する RI600V4 のバージョンを表示します。	
	デフォルト	インストールしている RI600V4 のバージョン
	変更方法	変更不可
インストール・フォルダ	使用する RI600V4 がインストールされているフォルダを絶対パスで表示します。	
	デフォルト	使用する RI600V4 がインストールされているフォルダ
	変更方法	変更不可
エンディアン	プロジェクトで設定しているエンディアンを表示します。 ビルド・ツールの［エンディアンの選択］プロパティで選択しているレジスタ・モードと同じ値が表示されます。	
	デフォルト	ビルド・ツールでのプロパティで選択しているエンディアン
	変更方法	変更不可

［タスク・アナライザ］タブ

概要

本タブでは，リアルタイム OS タスク・アナライザの設定を行います。

表示イメージ



機能

1）［トレース］

リアルタイム OS タスク・アナライザのトレース・モードに関する設定を行います。本設定により，「2.6.6　リアルタイム OS タスク・アナライザに関するオプション」に示すビルド・オプションが自動設定されます。ただし，対応する機能のプロパティパネルとは連動していないため，自動設定された内容を対応する機能のプロパティパネルで変更してはなりません。

トレース・モードの選択	リアルタイム OS タスク・アナライザのトレース・モードを選択します。		
	デフォルト	ハードウェア・トレース・モードでトレース・チャートを取得	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	トレースしない	リアルタイム OS タスク・アナライザは使用できません。
		ハードウェア・トレース・モードでトレース・チャートを取得	トレース情報はエミュレータやシミュレータが持つトレース・メモリに収集されます。
		ソフトウェア・トレース・モードでトレース・チャートを取得	トレース情報はユーザ・メモリに確保したトレース・バッファに収集されます。 本モード使用時は，ユーザ・オウン・コーディング部の実装とシステム・コンフィギュレーション・ファイルの設定が必要です。詳細は，15.3.1 節を参照してください。
		ソフトウェア・トレース・モードで長時間統計を取得	トレース情報はユーザ・メモリに確保した RI600V4 の変数に収集されます。 本モード使用時は，ユーザ・オウン・コーディング部の実装とシステム・コンフィギュレーション・ファイルの設定が必要です。詳細は，15.3.2 節を参照してください。

<table>
<tr><td rowspan="5">バッファを使い切った後の動作</td><td colspan="3">トレース・バッファを使い切ったときの動作を選択します。
本項目は，「ソフトウェア・トレース・モードでトレース・チャートを取得」を選択した場合のみ表示されます。</td></tr>
<tr><td>デフォルト</td><td colspan="2">バッファを上書きし実行し続ける。</td></tr>
<tr><td>変更方法</td><td colspan="2">ドロップダウン・リストによる選択</td></tr>
<tr><td rowspan="2">指定可能値</td><td>バッファを上書きし実行し続ける</td><td>古い情報から順に上書きします。</td></tr>
<tr><td>トレースを停止する</td><td>トレース取得を停止します。</td></tr>
<tr><td rowspan="4">バッファ・サイズ</td><td colspan="3">トレース・バッファのサイズ（単位：バイト）を指定します。サイズの見積もりについては，「15.4 トレース・バッファのサイズ（ソフトウェア・トレース・モードでトレース・チャートを取得）」を参照してください。
本項目は，「ソフトウェア・トレース・モードでトレース・チャートを取得」を選択した場合のみ表示されます。</td></tr>
<tr><td>デフォルト</td><td colspan="2">0x100</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力。16 進数のみ入力可能。</td></tr>
<tr><td>指定可能値</td><td colspan="2">0x10 ～ 0x0FFFFFFF</td></tr>
<tr><td rowspan="5">バッファを選択する</td><td colspan="3">バッファを選択します。
本項目は，「ソフトウェア・トレース・モードでトレース・チャートを取得」を選択した場合のみ表示されます。</td></tr>
<tr><td>デフォルト</td><td colspan="2">カーネルのバッファ</td></tr>
<tr><td>変更方法</td><td colspan="2">ドロップダウン・リストによる選択</td></tr>
<tr><td rowspan="2">指定可能値</td><td>カーネルのバッファ</td><td>ビルド時に，指定されたサイズのトレース・バッファがBRI_TRCBUF セクションに生成されます。</td></tr>
<tr><td>その他のバッファ</td><td>次項でバッファのアドレスを指定します。</td></tr>
<tr><td rowspan="4">バッファ・アドレス</td><td colspan="3">「その他のバッファ」の先頭アドレスを即値で指定します。
このアドレスからバッファ・サイズ（バイト）の領域がトレース・バッファとして使用されます。トレースバッファ領域が，他のプログラムやデータ領域と重複しないように注意してください。
本項目は，「その他のバッファ」を選択した場合のみ表示されます。</td></tr>
<tr><td>デフォルト</td><td colspan="2">0x0</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力。16 進数のみ入力可能。</td></tr>
<tr><td>指定可能値</td><td colspan="2">0x0 ～ 0xFFFFFFFF で，かつ「バッファ・アドレス」＋「バッファ・サイズ」が 0xFFFFFFFF を超えないこと。</td></tr>
</table>

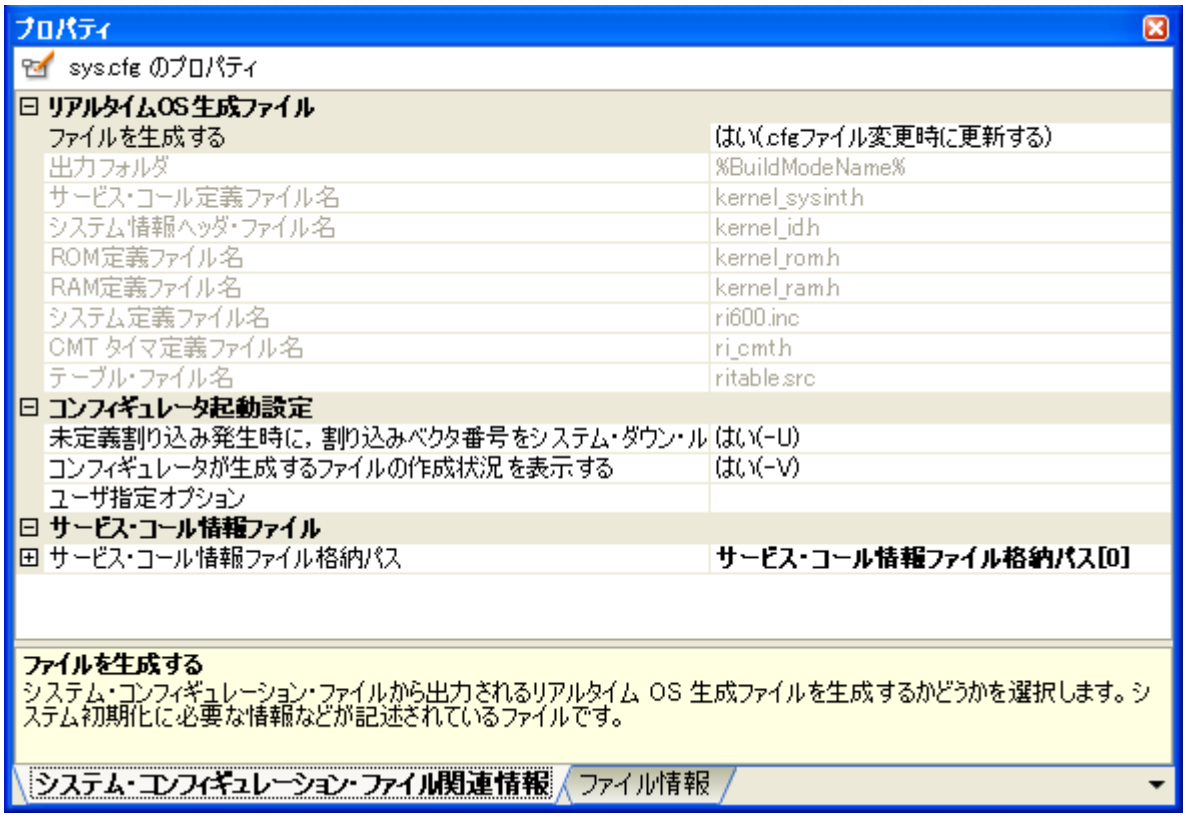
［システム・コンフィギュレーション・ファイル関連情報］タブ

概要

本タブでは，使用するシステム・コンフィギュレーション・ファイルに対して，次に示すカテゴリごとに詳細情報の表示，および設定の変更を行います。

- リアルタイム OS 生成ファイル
- コンフィギュレータ起動設定
- サービス・コール情報ファイル

表示イメージ



機能

1）［リアルタイム OS 生成ファイル］

RI600V4 生成ファイルに関する詳細情報の表示，および設定の変更を行います。

ファイルを生成する	リアルタイム OS 生成ファイルを生成するかどうか，およびシステム・コンフィギュレーション・ファイルを変更した場合にリアルタイム OS 生成ファイルを更新するかどうかを選択します。		
	デフォルト	はい（.cfg ファイル変更時に更新する）	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	はい（.cfg ファイル変更時に更新する）	リアルタイムOS生成ファイルを新規生成し，プロジェクト・ツリーに表示します。 リアルタイムOS生成ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は，リアルタイム OS 生成ファイルを更新します。
		いいえ（プロジェクトに登録しない）	リアルタイムOS生成ファイルの生成を行わず，プロジェクト・ツリーにも表示しません。 システム情報テーブル・ファイルが存在するときに本項目を選択しても，ファイル自体の削除は行いません。
出力フォルダ	リアルタイム OS 生成ファイルを出力するフォルダを表示します。		
	デフォルト	%BuildModeName%	
	変更方法	変更不可	
サービス・コール定義ファイル名	cfg600 が生成するサービス・コール定義ファイル名を表示します。		
	デフォルト	kernel_sysint.h	
	変更方法	変更不可	
システム情報ヘッダ・ファイル名	cfg600 が生成するシステム情報ヘッダ・ファイル名を表示します。		
	デフォルト	kernel_id.h	
	変更方法	変更不可	
ROM 定義ファイル	cfg600 が生成する ROM 定義ファイル名を表示します。		
	デフォルト	kernel_rom.h	
	変更方法	変更不可	
RAM 定義ファイル	cfg600 が生成する RAM 定義ファイル名を表示します。		
	デフォルト	kernel_ram.h	
	変更方法	変更不可	
システム定義ファイル	cfg600 が生成するシステム定義ファイル名を表示します。		
	デフォルト	ri600.inc	
	変更方法	変更不可	
CMT タイマ定義ファイル名	cfg600 が生成する CMT タイマ定義ファイル名を表示します。		
	デフォルト	ri_cmt.h	
	変更方法	変更不可	
テーブル・ファイル名	mkritbl が生成するテーブル・ファイル名を表示します。		
	デフォルト	ritable.src	
	変更方法	変更不可	

2）［コンフィギュレータ起動設定］

コンフィギュレータ cfg600 の起動オプションを設定します。

未定義割り込み発生時に，割り込みベクタ番号をシステム・ダウン・ルーチンに渡す	未定義割り込みが発生した時にはシステム・ダウンとなりますが，“ はい ” を選択すると，発生した割り込みのベクタ番号がシステム・ダウン・ルーチンに渡されるようになるため，デバッグに役立ちます。ただし，カーネルのコードサイズが約 1.5KB 増加します。 システム・ダウンについては，「第 13 章　システム・ダウン」を参照してください。		
	デフォルト	はい（-U）	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	はい（-U）	未定義割り込み発生時に，割り込みベクタ番号をシステム・ダウン・ルーチンに渡します。
		いいえ	未定義割り込み発生時に，割り込みベクタ番号をシステム・ダウン・ルーチンに渡しません。
コンフィギュレータが生成するファイルの作成状況を表示する	cfg600 が生成するファイルの作成状況を表示するかどうかを設定します。		
	デフォルト	はい（-V）	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	はい（-V）	コンフィギュレータが生成するファイルの作成状況を表示します。
		いいえ	コンフィギュレータが生成するファイルの作成状況を表示しません。
ユーザ指定オプション	cfg600 に渡すユーザ任意のオプションを設定します。		
	デフォルト	ー	
	変更方法	テキスト・ボックスによる直接入力	
	指定可能値	259 文字までの文字列	

3）［サービス・コール情報ファイル］

テーブル生成ユーティリティ mkritbl に入力するサービス・コール情報ファイルの検索パスを設定します。アプリケーションのコンパイルによって生成されるサービス・コール情報ファイルが格納されたフォルダを，漏れなく指定してください。

サービス・コール情報ファイル格納パス	アプリケーションのコンパイルによって生成されるサービス・コール情報ファイル（.mrc）が格納されたフォルダのパス，およびファイル名を設定します。 フォルダ・パスを指定した場合，そのサブ・フォルダは検索対象にはなりません。 相対パスで指定した場合は，プロジェクト・フォルダを基点とします。 絶対パスで指定した場合は，プロジェクト・フォルダを基点とした相対パスに変換されます（ドライブが異なる場合を除く）。 複数のパスを指定する場合は，改行で区切ります。 なお，本指定に関わらず，プロジェクト・フォルダのパスは mkritbl に渡されます。 以下のプレース・ホルダを使用できます。 %BuildModeName%：ビルド・モード名に置換します。	
	変更方法	［...］ボタンをクリックし，パス編集ダイアログによる編集
	指定可能値	259 文字までの文字列 なお，拡張子が指定されていない場合，および “.mrc” 以外の拡張子を指定した場合は，フォルダとして扱います。

備考 1　サービス・コール情報ファイルについては，「2.6.1　サービス・コール情報ファイルとコンパイラ・オプション “-ri600_preinit_mrc”」を参照してください。

備考 2　コンパイラの外部変数アクセス最適化オプションを使用する場合，CubeSuite+ は 1 回目のコンパイルによって生成されるオブジェクト・ファイルやサービス・コール情報ファイルを格納するフォルダを自動的に生成し，そのフォルダを暗黙的に検索パスに設定します。

備考 3　サービス・コール情報ファイルは，コンパイル時にオブジェクト・ファイルと同じフォルダに出力されます。オブジェクト・ファイルの出力フォルダを変更する操作を行った場合は，本項目を適切に変更してください。

付録 B　浮動小数点演算機能

コンパイラが浮動小数点演算命令を出力するのは，オプション“-fpu”指定時のみです。アセンブラでオプション“-chkfpu”を指定すると，浮動小数点演算命令の記述を Warning として検出します。

B.1　タスクで浮動小数点演算命令を使用する場合

システム情報（system）のタスク・コンテキスト・レジスタ（context）に，“FPSW” を含む設定を行ってください。これにより，FPSW レジスタはタスク毎に独立に管理されるようになります。

タスク起動時の FPSW レジスタは，使用するコンパイラ・オプションに応じた値に初期化されます。詳細は，「3.2.4 タスク内での処理」を参照してください。

B.2　ハンドラで浮動小数点演算命令を使用する場合

ハンドラは，明示的に FPSW レジスタを保証する必要があります。

また，ハンドラ起動時の FPSW レジスタは不定です。以下のようにして，FPSW レジスタの保証と初期化を行ってください。

```
#include <machine.h>     // コンパイラ提供の組み込み関数 get_fpsw(), set_fpsw() を使用する
                         // ために，machine.h をインクルード
#include "kernel.h"
#include "kernel_id.h"
void handler (void)
{
    unsigned long old_fpsw;     // FPSW レジスタを退避するための変数を宣言
    old_fpsw = get_fpsw () ;    // FPSW レジスタを退避
    set_fpsw (0x00000100) ;     // 必要なら FPSW を初期化
    /* 浮動小数点演算処理 */
    set_fpsw (old_fpsw) ;       // FPSW レジスタを復帰
}
```

付録 C　DSP 機能

DSP 機能をサポートした MCU を使用する場合は，ACC レジスタ（アキュムレータ）の扱いについて注意が必要です。具体的には，ACC レジスタを更新する DSP 機能命令（下記）を使用する場合は，後述について留意してください。

- RXv1 ／ RXv2 アーキテクチャ共通命令
  MACHI, MACLO, MULHI, MULLO, RACW, MVTACHI, MVTACLO

- RXv2 アーキテクチャ命令
  EMACA, EMSBA, EMULA, MACLH, MSBHI, MSBLH, MSBLO, MULLH, MVTACGU, RACL,RDACL, RDACW

なお，コンパイラがこれらの命令を生成することはありません。また，アセンブラでオプション “-chkdsp” を指定すると，DSP 機能命令の記述を Warning として検出します。

C.1　タスクで DSP 機能命令を使用する場合

システム情報（system）のタスク・コンテキスト・レジスタ（context）に，“ACC” を含む設定を行ってください。これにより，ACC レジスタはタスク毎に独立に管理されるようになります。

C.2　割り込みハンドラでの ACC レジスタの保証

アプリケーション内に前述の DSP 機能命令を使用するタスクおよび割り込みハンドラがひとつでもある場合は，すべての割り込みハンドラが ACC レジスタを保証する必要があります。その方法として，以下の 2 つがあります。また，表 19 － 9 も参照してください。

1） コンパイラ・オプション “-save_acc” を使用する。

2） すべての割り込みハンドラの定義（可変ベクタ情報（interrupt_vector[]），および固定ベクタ／例外ベクタ情報（interrupt_fvector[]））際に，pragma_switch に “ACC” を指定する。

付録 D　スタック使用量の算出

スタックがオーバーフロすると，システムの動作は不定となるので，本章を参考にスタックがオーバフローしないようにしてください。

D.1　スタックの種類

スタックには，ユーザ・スタックとシステム・スタックの 2 種類があります。スタック使用量の算出方法は，ユーザ・スタックとシステム・スタックで異なります。

- ユーザ・スタック
  タスクのスタックを，ユーザ・スタックと呼びます。タスク情報（task[]）でタスクを生成するときに，サイズとユーザ・スタック領域を割り当てるセクションを指定します。

- システム・スタック
  各種ハンドラとカーネルが共通に使用するスタックで，システムにひとつだけ存在します。システム・スタック・サイズは，システム情報（system）のシステム・スタック・サイズ（stack_size）に指定します。システム・スタックのセクション名は SI です。

D.2　Call Walker

CubeSuite+ には，スタック算出ユーティリティである Call Walker が付属しています。Call Walker を使用すると，各関数ツリーで消費されるスタックサイズを確認することができます。

D.3　ユーザ・スタック使用量の算出

各タスクのユーザ・スタックの使用量は，以下の式で算出されます。

ユーザ・スタックの使用量 =*treesz* + *ctxtsz*

- *treesz*
  タスク開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。

- *ctxtsz*
  タスクのコンテキスト・レジスタのサイズです。このサイズは，システム情報（system）のタスク・コンテキスト・レジスタ（context）の設定によって異なります。表 D－1 を参照してください。

表 D－1　タスク・コンテキスト・レジスタ・サイズ

system.context 設定	コンパイラ・オプション “-isa”	タスク・コンテキスト・レジスタ・サイズ（バイト）
NO	—	68
FPSW	—	72
ACC	“-isa=rxv2”	92
	“-isa=rxv1” またはオプション “-isa” の指定なし	76
FPSW,ACC	“-isa=rxv2”	96
	“-isa=rxv1” またはオプション “-isa” の指定なし	80
MIN	—	44
MIN,FPSW	—	48
MIN,ACC	“-isa=rxv2”	68
	“-isa=rxv1” またはオプション “-isa” の指定なし	52
MIN,FPSW,ACC	“-isa=rxv2”	72
	“-isa=rxv1” またはオプション “-isa” の指定なし	56

備考　コンパイラ・オプション “-isa” は，コンパイラ CC-RX V2.01 以降でサポートされています。

D.4　システム・スタック使用量の算出

システム・スタックを最も多く消費するのは，サービス・コール処理中 に割り込みが発生，さらに多重割り込みが発生した場合です。すなわち，システム・スタックの必要量 （最大サイズ）は以下の計算式で算出することができます。

システム・スタックの使用量 = *svcsz*

$$+ \sum_{k=1}^{15} inthdrsz_k$$

+ *sysdwnsz*

- *svcsz*
  すべての処理プログラムで使用しているサービス・コールの中での最大の使用量です。*svcsz* の値は RI600V4 のバージョンによって異なります。製品添付のリリースノートを参照してください。

- *inthdrsz*
  各割り込みハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。
  k は，割り込み優先レベルです。同じ優先レベルの割り込みが複数ある場合は，それらのハンドラの中で最大のサイズを $inthdrsz_k$ としてください。
  なお，基本クロック用割り込みハンドラ（割り込み優先レベルは基本クロック割り込み情報（clock）の基本クロック割り込み優先レベル（IPL）で指定します）の使用量は，以下の 3 つのサイズの最大値となります。*clocksz1*，*clocksz2*，*clocksz3* については，リリースノートを参照してください。
  なお，基本クロック用タイマを使用しない場合（clock.timer=NOTIMER）は，基本クロック用割り込みハンドラが使用するサイズを加算する必要はありません。
  - *clocksz1* + *cycsz*
  - *clocksz2* + *almsz*
  - *clocksz3*

    - *cycsz*
      周期ハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。周期ハンドラが複数ある場合は，それらのハンドラの中で最大のサイズを *cycsz* としてください。

    - *almsz*
      アラーム・ハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。アラーム・ハンドラが複数ある場合は，それらのハンドラの中で最大のサイズを *almsz* としてください。

- *sysdwnsz*
  システム・ダウン・ルーチン開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）+ 40 としてください。システム･ダウン･ルーチンに遷移するケースがない場合は，*sysdwnsz*を0として計算してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.10.01	―	初版発行
1.01	2012.04.01	―	「優先度」，「現在優先度」，「ベース優先度」の使い分けを改善
		15	2.4 節に「セクション情報ファイル」を追記
		20	2.6.2 節の表現を改善
		29	3.2.2 節の表現を改善
		31	表 3 － 2 の PM，U ビットをそれぞれ 1 に訂正
		74，74，77，262，264，266	「データ・キューにデータが格納されている場合」に，「送信待ちキューにタスクがキューイングされている場合は，送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち，WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。」を追記
		96，99，294，298	メッセージ送信待ち状態の解除操作の表に，「送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された」場合を追記
		114，314	備考を削除
		118，121，318，323	可変長メモリ・ブロック獲得待ち状態の解除操作の表に，「送信待ちキュー先頭のタスクが，以下のいずれかによって待ち状態を強制的に解除された」場合を追記
		―	第 8 章　の構成を改善
		129	8.6.4 節の表現を改善
		140	「第 8.9 章　基本クロック用タイマの初期化」を追加
		146	9.5 節の説明を改善
		149	9.7 節の説明を改善
		150	9.8 節の説明を改善
		151	9.9 節の説明を改善
		164	表 13 － 1 の IPL の説明を詳細化
		164	「type == -1」の説明を「割り込みハンドラ終了時のエラー」から「カーネル管理割り込みハンドラ終了時のエラー」に訂正
		169	14.7 節の説明を改善
		―	第 16 章　の構成を改善
		186	「第 16.4 章　セクション初期化関数（_INITSCT( )）」を追加
		187	「第 16.5 章　固定ベクタ・テーブル／例外ベクタ・テーブルのレジスタなど」を追加
		188	表 17 － 1　の INT，UINT，VP_INT，ER_UINT，および FLGPTN のデータ型を，「singed int」または「unsigend int」から，「signed long」または「unsigned long」に訂正
		190	表 17 － 2　の TA_TPRI，および TA_MPRI の説明を改善
		279	T_RMBX 構造体のメンバに，存在しないメンバ mbxatr が記載されていた誤記を訂正

Rev.	発行日	改訂内容	
		ページ	ポイント
		283，284，287	E_ILUSE エラーの「上限優先度違反」の条件式を訂正
		291	E_CTX エラーの条件を訂正
		302	E_CTX エラーの条件から「ディスパッチ禁止状態から本サービス・コールを発行した。」を削除
		323	「tget_mpl」の備考 2 を訂正
		329，330	SYSTIM 構造体のメンバ順序の誤記を訂正
		336	sta_alm, ista_alm の「戻り値」に E_PAR を追記
		345	備考 5 追記
		350	「ディスパッチ禁止状態に遷移する操作」の誤記訂正 【誤】chg_ims によって割り込みマスク (PSW.IPL) を 0 に変更 【正】chg_ims によって割り込みマスク (PSW.IPL) を 0 以外に変更
		362	ref_ver，iref_ver で返る *prid* の値を，0x0004 から 0x0003 に訂正
		383	“*max_count*” の定義範囲を「1 ～ 65535」に訂正
		393	「5) 最大メッセージ・サイズ（max_msgsz)」の説明から「指定した値は 4 の倍数に切り上げて扱います。」を削除
		396	19.15 節の「可変長メモリ・プールのサイズ（heap_size)」の説明から，「指定した値は 4 の倍数に切り上げられます。」を削除
		404	備考 2 を追加
		409	19.20.1 節に以下を追記 「なお，実際のサイズは，境界調整のために表 19 － 11 で算出される値よりも大きくなります。」
		409	表 19 － 11 において，以下の係数を訂正 - データ・キュー管理ブロック：計算式先頭の「8」を「6」に訂正 - 可変長メモリ・プール管理ブロック：計算式先頭の「36」を「208」に訂正
		410	*DTQ_ALLSIZE* の説明に以下を追記 「ただし，この計算結果が 0 になる場合は，*DTQ_ALLSIZE* は 4 です。」
1.02	2012.09.01 (RI600V4 V1.02.00)	15	リアルタイム OS タスク・アナライザ対応のユーザ・オウン・コーディング部について追記
		23	表 2 － 2 に，DRI_ROM，RRI_RAM，BRI_TRCBUF セクションを追記
		26	「2.6.6　リアルタイム OS タスク・アナライザに関するオプション」を追加
		170	「第 15 章　リアルタイム OS タスク・アナライザ」を追加
		192	RI600V4 の V1.02.00 へのリビジョン・アップに伴い，TKERNEL_PRVER の定義値を 0x0120 に変更
		408	RRI_RAM セクション，および BRI_TRCBUF セクションを追加
		409	- 「システム管理ブロック」のメモリ容量算出式の最初の数値「28」を「36」に変更 - リアルタイム OS タスク・アナライザ対応の項を追加
		420	リアルタイム OS タスク・アナライザに関するメニューとツールバーの説明を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
		428	「[タスク・アナライザ] タブ」を追加
1.03	2013.05.15 (RI600V4 V1.02.02)	14	「2.3　システム・コンフィギュレーション・ファイルの記述」に備考 2 を追記
		20	「2.6.2　ブート処理ファイルのコンパイラ・オプション」の説明を詳細化
		25	「2.6.5　初期化データ・セクション」を追加
		31, 434	タスク処理開始時の FPSW の仕様を変更
		170, 428	ソフトウェア・トレース・モード使用時の必要事項として，システム・コンフィギュレーション・ファイルの設定を追記
		186	「16.4　セクション初期化関数（_INITSCT( )）」の説明を改善
1.04	2013.09.20 (RI600V4 V1.03.00	22	RXv2 アーキテクチャのサポートに伴い，カーネル・ライブラリの構成を変更
		24,180, 187,405 など	RXv2 アーキテクチャのサポートに伴い，FIX_INTERRUPT_VERCTOR セクションおよび EXTB レジスタに関する説明を追加・変更。 また，全般に「固定ベクタ」を「固定ベクタ／例外ベクタ」に置換。
		181	RXv2 アーキテクチャのサポートに伴い，“-isa”, “-cpu” オプションについて追記
		185，356	RI600V4 起動に関する説明を改善
		192	RI600V4 V1.03.00 へのリビジョン・アップに伴い，TKERNEL_PRVER の定義値を 0x0130 に変更
		374	RXv2 アーキテクチャのサポートに伴い，表 19 － 2 を変更
		378	表 19 － 7 の説明を改善
		381	RXv2 アーキテクチャのサポートに伴い，表 19 － 8 を変更
		435	ACC を更新する命令に，RXv2 アーキテクチャ専用命令を追記
		435	すべての割り込みハンドラが明示的に ACC を保証する方法の記載を削除
		437	RXv2 アーキテクチャのサポートに伴い，表 D － 1 を変更

RI600V4 リアルタイム・オペレーティング・システム
ユーザーズマニュアル　コーディング編

発行年月日　2011年10月1日　Rev.1.00
　　2013年9月20日　Rev.1.04

発行　　ルネサス エレクトロニクス株式会社
　　〒211-8668 神奈川県川崎市中原区下沼部 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）　(03)5201-5307

■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：http://japan.renesas.com/contact/

Colophon 1.1

RI600V4

RENESAS

ルネサス エレクトロニクス株式会社

R20UT0711JJ0104