User's Manual

RI78V4 V2.00.00

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

RL78 ファミリ

お知らせ；
本資料の以下のページに追加、修正がございます。
・Page15　.k_const の意味に注意事項を追記
・Page117　-マスカブル割り込みの受け付けが許可された状態を修正

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2015.3

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，RL78 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　RI78V4 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　このマニュアルを読むにあたっては，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が必要となります。

RL78 ファミリ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 のべき数を示す接頭語（アドレス空間，メモリ容量）：
K（キロ）　$2^{10} = 1024$
M（メガ）　$2^{20} = 1024^2$

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
RI78V4	コーディング編	このマニュアル	R20UT3375E
	デバッグ編	R20UT3374J	R20UT3374E
	解析編	R20UT3373J	R20UT3373E

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

RI78V4 は，効率の良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。

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 CPU 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 CPU 自体は 1 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。

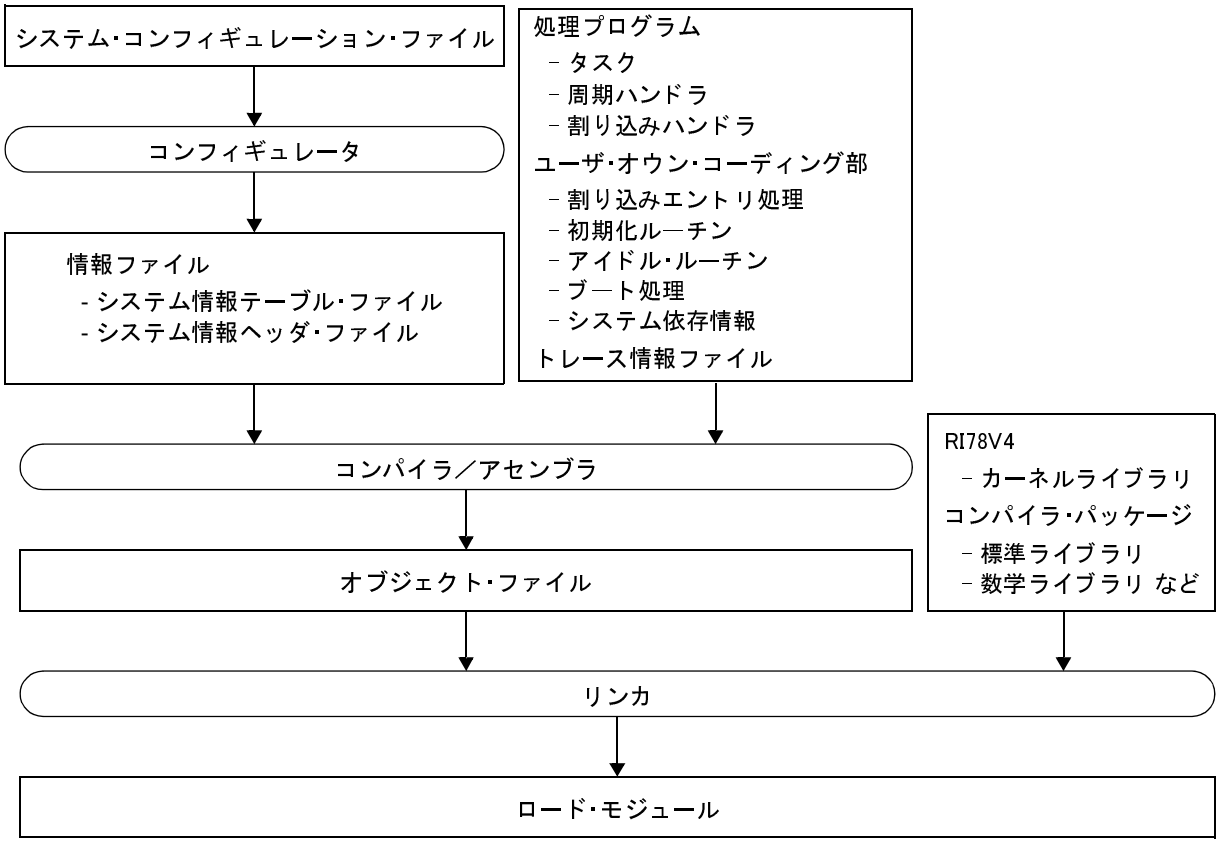
第２章　システム構築

本章では，RI78V4 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，RI78V4 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。
以下に，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



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，RI78V4 では，処理プログラムを実現すべき処理の種類／用途にあわせて以下に示した 3 種類に分類しています。

- タ ス ク
  他の処理プログラム（周期ハンドラ，割り込みハンドラなど）とは異なり，RI78V4 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限り実行されることのない処理プログラムです。

  備考　　タスクについての詳細は，「3.2　タ ス ク」を参照してください。

- 周期ハンドラ
  一定の時間（起動周期）を単位として周期的に起動される周期処理専用ルーチンです。
  なお，RI78V4 では，周期ハンドラを“非タスク（タスクとは独立したもの）”として位置づけています。このため，起動周期に達した際には，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。

  備考　　周期ハンドラについての詳細は，「7.5　周期ハンドラ」を参照してください。

- 割り込みハンドラ
  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，RI78V4 では，割り込みハンドラを“非タスク（タスクとは独立したもの）”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 1　割り込みハンドラについての詳細は，「9.3　割り込みハンドラ」を参照してください。

  備考 2　“タイマ・ハンドラの呼び出し”を行う割り込みハンドラについても，ユーザが記述する必要があります。

2.3　システム・コンフィギュレーション・ファイルの記述

RI78V4 に提供するデータを保持した情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル，割り込み情報定義ファイル）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　　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 13 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。

2.4 ユーザ・オウン・コーディング部の記述

RI78V4 を様々な実行環境に対応するために切り出されたユーザ・オウン・コーディング部を記述します。

なお，RI78V4 では，ユーザ・オウン・コーディング部を実現すべき処理の種類／用途にあわせて以下に示した 4 種類に分類しています。

- 割り込みエントリ処理
  割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに該当処理（割り込みハンドラ，ブート処理など）への分岐命令を割り付けるために割り込み管理機能から切り出されたエントリ処理専用ルーチンです。

  備考 1 割り込みエントリ処理についての詳細は，「9.2 割り込みエントリ処理」を参照してください。

  備考 2 割り込みハンドラを C 言語で記述（システム・コンフィギュレーション・ファイルの割り込みハンドラ定義（DEF_INH）にて TA_HLNG 属性を指定）する場合，C コンパイラが“割り込み要求名に対応した割り込みエントリ処理”を自動的に出力するため，ユーザが割り込みエントリ処理を記述する必要はありません。

- ブート処理
  RI78V4 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を初期化するためにシステム構成管理機能から切り出された初期化処理専用ルーチンであり，リセット割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに割り付けられた割り込みエントリ処理から呼び出されます。

  備考 ブート処理についての詳細は，「10.2 ブート処理」を参照してください。

- 初期化ルーチン
  ユーザの実行環境に依存したハードウエア（周辺コントローラなど）を初期化するためにシステム構成管理機能から切り出された初期化処理専用ルーチンであり，カーネル初期化部から呼び出されます。

  備考 初期化ルーチンについての詳細は，「10.3 初期化ルーチン」を参照してください。

- アイドル・ルーチン
  CPU が提供しているスタンバイ機能を有効活用（低消費電力システムの実現）するためにスケジューリング機能から切り出されたアイドル処理専用ルーチンであり，RI78V4 のスケジューリング対象となるタスク（RUNNING 状態，または READY 状態のタスク）がシステム内に 1 つも存在しなくなった際にスケジューラから呼び出されます。

  備考 アイドル・ルーチンについての詳細は，「11.7 アイドル・ルーチン」を参照してください。

2.5 セクションの開始アドレスの指定

リンカが行うアドレス割り付けをユーザが固定化するため，セクションの開始アドレスを指定します。本指定はリンカのオプションで指定します。

なお，RI78V4 では，機能単位にモジュール化された管理オブジェクトの割り付け先（セクション名）を規定しています。以下に，RI78V4 が規定しているセクション名の一覧を示します。

表２－１ RI78V4 のセクション

セクション名	配置領域	再配置属性	意味
.kernel_system	フラッシュメモリ領域	TEXTF	RI78V4 の核となる処理部，および，RI78V4 が提供するサービス・コールの本体処理部が割り付けられる領域。 先頭が偶数番地で，0x000c0 ～ 0xeffff の領域に配置可能。
.kernel_system_timer_n	フラッシュメモリ領域	TEXT	システム・タイマ割り込み処理部，および，far 分岐情報が割り付けられる領域。 先頭が偶数番地で，0x000c0 ～ 0x0ffff の領域に配置可能。
.kernel_info	フラッシュメモリ領域	CONSTF	RI78V4 のバージョンなどといった情報が割り付けられる領域。 フラッシュメモリ領域内で，先頭が偶数番地，かつ 64K-1 境界にまたがらない領域に配置可能。
.kernel_const .kernel_const_f	フラッシュメモリ領域	CONSTF	動的に変化することのない OS 資源に関する初期情報が，システム情報テーブルや割り込み情報定義ファイルに割り付けられる領域。 フラッシュメモリ領域内で，先頭が偶数番地，かつ 64K-1 境界にまたがらない領域に配置可能。 **ただし、本セクションは、64K境界＋4番地以降に配置してください。**
.kernel_stack	内部 RAM 領域	BSS	システム・スタック，および，タスク・スタックが割り付けられる領域。 先頭が偶数番地で，内蔵 RAM 内 0xf0000 ～ 0xfffff の領域，かつ，64KB-1 境界にまたがらない領域に配置可能。
.kernel_data	内部 RAM 領域	BSS	RI78V4 が提供する機能を実現するうえで必要となる情報，および，動的に変化する OS 資源に関する情報が管理オブジェクトとして割り付けられる領域。 先頭が偶数番地で，内蔵 RAM 内 0xf0000 ～ 0xfffff の領域，かつ，64KB-1 境界にまたがらない領域に配置可能。
.kernel_data_init	内部 RAM 領域	BSS	RI78V4 の初期化情報が割り付けられる領域。 先頭が偶数番地で，内蔵 RAM 内 0xf0000 ～ 0xfffff の領域，かつ，64KB-1 境界にまたがらない領域に配置可能。
.kernel_work0 .kernel_work1 .kernel_work2 .kernel_work3	内部 RAM 領域	BSS	データキュー，および固定長メモリ・プールが割り付けられる領域。 先頭が偶数番地で，内蔵 RAM 内 0xf0000 ～ 0xfffff の領域，かつ，64KB-1 境界にまたがらない領域に配置可能。
.kernel_data_trace_n	内部 RAM 領域	BSS	トレース・データが割り付けられる領域。 先頭が偶数番地で，内蔵 RAM 内 0xf0000 ～ 0xfffff の領域，かつ，64KB-1 境界にまたがらない領域に配置可能。

セクション名	配置領域	再配置属性	意味
.kernel_const_trace_f	フラッシュメモリ領域	CONSTF	トレース・データの取得に必要な情報が割り付けられる領域。 フラッシュメモリ領域内で，先頭が偶数番地，かつ 64K-1 境界にまたがらない領域に配置可能。
.kernel_system_trace_f	フラッシュメモリ領域	TEXTF	トレース・データ取得処理部が割り付けられる領域。 先頭が偶数番地で，0x000c0 ～ 0xeffff の領域に配置可能。
.kernel_sbss	内部 RAM 領域	SBSS	RI78V4 が使用する SADDR 領域。 saddr 領域内で先頭が偶数番地の領域に配置。

備考 1 .kernel_work0，.kernel_work1，.kernel_work2，.kernel_work3 については，データ・キュー情報，または固定長メモリ・プール情報で該当セクション名の指定が行われた場合に限り必要となります。

備考 2 RI78V4 では，saddr 領域（0xffe20 ～ 0xfff1f）から 8 バイト分の領域を占有します。したがって，ユーザ側で利用可能な saddr 領域は，最大で 247 バイトとなります。

備考 3 セクション開始アドレスで指定するセクション名は，RI78V4 を使用した CS+ 環境では自動的に設定されます。開始アドレスを変更したい場合は，リンカの設定で変更してください。リンカの設定についての詳細は，「CS+ 統合開発環境 ユーザーズマニュアル RL78 ビルド編」を参照してください。

備考 4 セクションの開始アドレスを指定についての詳細は，「CS+ 統合開発環境 ユーザーズマニュアル RL78 コーディング編」を参照してください。

2.5.1 .kernel_system セクション

.kernel_system セクションのサイズは，処理プログラム内で使用するサービス・コールに依存し，約 1K バイト～ 9K バイトとなります。

2.5.2 .kernel_system_timer_n セクション

以下に，.kernel_system_timer_n セクションのサイズ（単位：バイト）を見積もる際に必要となる計算式を示します。

system_timer_n = 16 ＋ (*inthnum_FAR* × 8)

tnthnum_FAR :割り込みハンドラ情報のうち TA_FAR 属性指定の割り込みハンドラの総数

2.5.3 .kernel_system_trace_f セクション

以下に，.kernel_system_trace_f セクションのサイズ（単位：バイト）は示します。

【 トレース・モードが「トレースなし」の場合 】

system_trace_f = 0

【 トレース・モードが「ハードウェア・トレース・チャート・モード」の場合 】

system_trace_f = 184

【 トレース・モードが「ソフトウェア・トレース・チャート・モード」の場合 】

system_trace_f = 706

【 トレース・モードが「長時間統計トレース・モード」の場合 】

system_trace_f = 590

2.5.4 .kernel_info セクション

.kernel_info セクションのサイズは，16 バイトとなります。

2.5.5 .kernel_const セクション

以下に，.kernel_const セクションのサイズ（単位：バイト）を見積もる際に必要となる計算式を示します。

const = (*tsknum* × 10) + *semnum* + *flgnum* + (*dtqnum* × 5) + (*mpfnum* × 8) + (*cycnum* × 12) + (*kindnum* × 4) + 16

tsknum： タスク情報の総数
semnum： セマフォ情報の総数
flgnum： イベントフラグ情報の総数
dtqnum： データ・キュー情報の総数
mpfnum： 固定長メモリ・プール情報の総数
kindnum： OS 資源に関する 5 種類の情報（セマフォ情報，イベントフラグ情報，データ・キュー情報，メールボックス情報，固定長メモリ・プール情報，周期ハンドラ情報）のうち，システム・コンフィギュレーション・ファイルに定義されている種類の総数

2.5.6 .kernel_const_f セクション

以下に，.kernel_const_f セクションのサイズ（単位：バイト）を示します。

【 トレース・モードが「トレースなし」の場合 】

const_f = 0

【 トレース・モードが「ハードウェア・トレース・チャート・モード」の場合 】

const_f = 0

【 トレース・モードが「ソフトウェア・トレース・チャート・モード」の場合 】

const_f = 0

【 トレース・モードが「長時間統計トレース・モード」の場合 】

const_f = 63

2.5.7 .kernel_stack セクション

以下に，.kernel_stack セクションのサイズ（単位：バイト）を見積もる際に必要となる計算式を示します。

$$stack = \sum_{k=1}^{tsknum} (stksz_k + 20) + (sys\_stksz + 2)$$

tsknum： タスク情報の総数
$stksz_k$： タスク情報で指定した“スタック・サイズ”
sys_stksz： スタック情報で指定した“スタック・サイズ”。多重割り込みが入る場合は 1 回につき 18 バイト加算。

2.5.8 .kernel_data セクション

以下に，.kernel_data セクションのサイズ（単位：バイト）を見積もる際に必要となる計算式を示します。
ただし，該当計算式は，システム・コンフィギュレーション・ファイルに“セマフォ情報が定義されているか否か”により異なります。

【 セマフォ情報が定義されている場合 】

data = align2 (*maxtpri* ＋ 1) ＋ align2 { (*tsknum* × 24) ＋ (*semnum* × 2) ＋ 1 } ＋ align2 (*flgnum* × 3) ＋ align2 { (*dtqnum* × 4) ＋ 1 } ＋ (*mbxnum* × 8) ＋ align2 (*primbx*) ＋ (*mpfnum* × 4) ＋ (*cycnum* × 8) ＋ 20

【 セマフォ情報が定義されていない場合 】

data = align2 (*maxtpri* ＋ 1) ＋ (*tsknum* × 24) ＋ align2 (*flgnum* × 3) ＋ align2 { (*dtqnum* × 4) ＋ 1 } ＋ (*mbxnum* × 8) ＋ align2 (*primbx*) ＋ (*mpfnum* × 4) ＋ (*cycnum* × 8) ＋ 20

maxtpri： 優先度情報で指定した“優先度範囲”
tsknum： タスク情報の総数
semnum： セマフォ情報の総数
flgnum： イベントフラグ情報の総数
dtqnum： データ・キュー情報の総数
mbxnum： メールボックス情報の総数
primbx： メールボックス情報で属性（メッセージのキューイング方式）に“優先度順”を指定した総数
mpfnum： 固定長メモリ・プール情報の総数
cycnum： 周期ハンドラ情報の総数

2.5.9 .kernel_data_init セクション

.kernel_data_init セクションのサイズは，2 バイトとなります。

2.5.10 .kernel_work0，.kernel_work1，.kernel_work2，.kernel_work3 セクション

以下に，.kernel_work0，.kernel_work1，.kernel_work2，.kernel_work3 セクションのサイズ（単位：バイト）を見積もる際に必要となる計算式を示します。

$$\text{work}X = \sum_{k=1}^{mpfnum} (blkcnt_k \times blksz_k) + \sum_{k=1}^{dtqnum} (dtqcnt_k \times 4)$$

mpfnum： 固定長メモリ・プール情報のセクション単位の総数
$blkcnt_k$： 固定長メモリ・プール情報で指定した“メモリ・ブロック数”
$blksz_k$： 固定長メモリ・プール情報で指定した“ブロック・サイズ”

dtqnum： データ・キュー情報のセクション単位の総数
$dtqcnt_k$： データ・キュー情報で指定した“データ数”

2.5.11 .kernel_data_init セクション

.kernel_data_init セクションのサイズは，2 バイトとなります。

2.5.12 .kernel_data_trace_n セクション

以下に，.kernel_data_trace_n セクションのサイズ（単位：バイト）を示します。

【 トレース・モードが「トレースなし」の場合 】

data_trace_n = 0

【 トレース・モードが「ハードウェア・トレース・チャート・モード」の場合 】

data_trace_n = 2

【 トレース・モードが「ソフトウェア・トレース・チャート・モード」の場合 】

data_trace_n = 8 + *bufsize*

bufsize：　トレース・バッファ・サイズ

【 トレース・モードが「長時間統計トレース・モード」の場合 】

data_trace_n = { (*tsknum* ＋ 1) × 20) } ＋ { (*inhnum* ＋ 1) × 8) } ＋ 34

tsknum：　タスク情報の総数
inhnum：　割り込みハンドラ情報の総数

2.5.13　.kernel_const_trace_f セクション

以下に，.kernel_const_trace_f セクションのサイズ（単位：バイト）を示します。

【 トレース・モードが「トレースなし」の場合 】

const_trace_n = 6

【 トレース・モードが「ハードウェア・トレース・チャート・モード」の場合 】

const_trace_n = 64

【 トレース・モードが「ソフトウェア・トレース・チャート・モード」の場合 】

const_trace_n = 70

【 トレース・モードが「長時間統計トレース・モード」の場合 】

const_trace_n = 70

2.6 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「2.5 セクションの開始アドレスの指定」で作成されたファイル群，トレース情報ファイル，および，RI78V4 や 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，CS+ 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」，および「CS+ 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） 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，「CS+ 統合開発環境 ユーザーズマニュアル RL78 ビルド編」を参照してください。

3） 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，「CS+ 統合開発環境 ユーザーズマニュアル RL78 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。

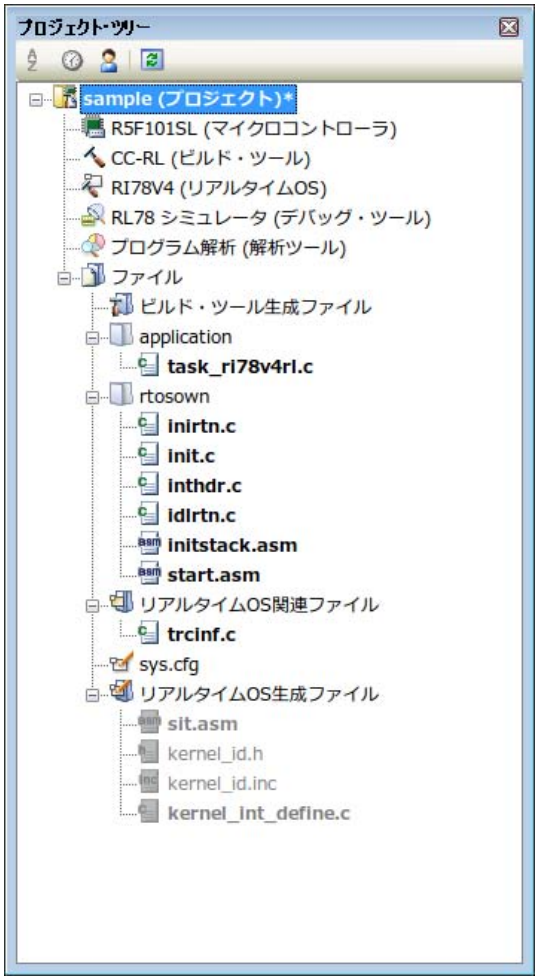
- 「2.2 処理プログラムの記述」で作成された C 言語／アセンブリ言語ソース・ファイル
  - タ ス ク，周期ハンドラ，割り込みハンドラ
- 「2.3 システム・コンフィギュレーション・ファイルの記述」で作成されたシステム・コンフィギュレーション・ファイル
  - システム・コンフィギュレーション・ファイル

  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 C 言語／アセンブリ言語ソース・ファイル
  - 割り込みエントリ処理，ブート処理，初期化ルーチン，アイドル・ルーチン
- RI78V4 が提供しているファイル
  - トレース情報ファイル
- RI78V4 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
- C コンパイラ／アセンブラ・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。

- システム情報テーブル・ファイル
- システム情報ヘッダ・ファイル（C 言語用）
- システム情報ヘッダ・ファイル（アセンブリ言語用）
- 割り込み情報定義ファイル

図 2 － 2　プロジェクト・ツリー パネル（sys.cfg 追加後）



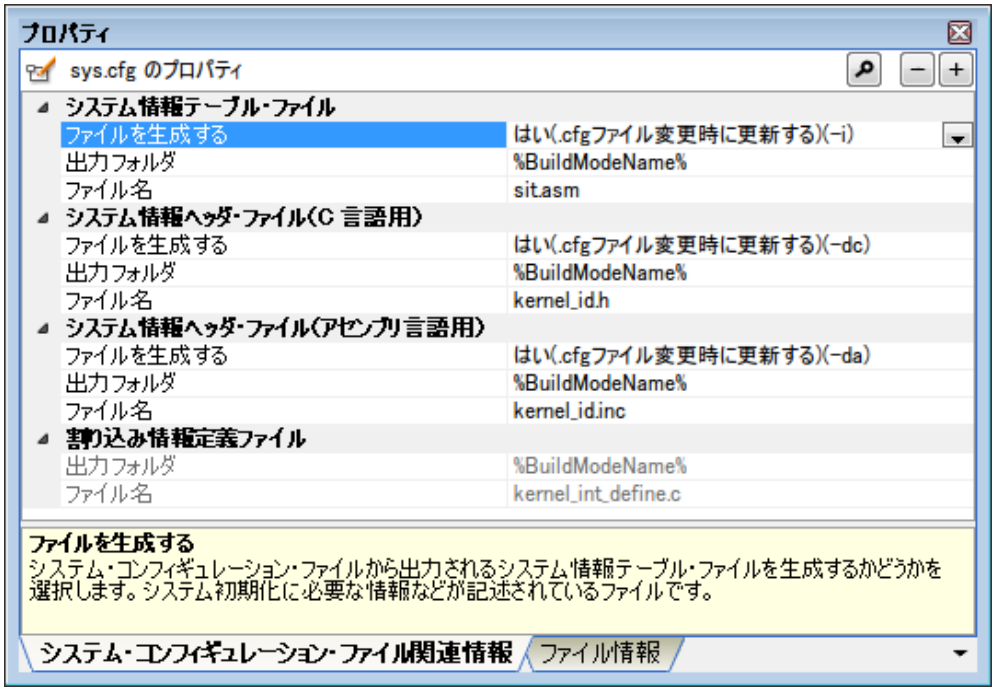
備考 2　システム・コンフィギュレーション・ファイルを差し替える場合は，追加しているシステム・コンフィギュレーション・ファイルを一旦プロジェクトから外したのち，再度ファイルを追加してください。

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。

4） 情報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，情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル）を出力することを設定します。

図２－３ 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



5） ロード・モジュール・ファイルの出力指定

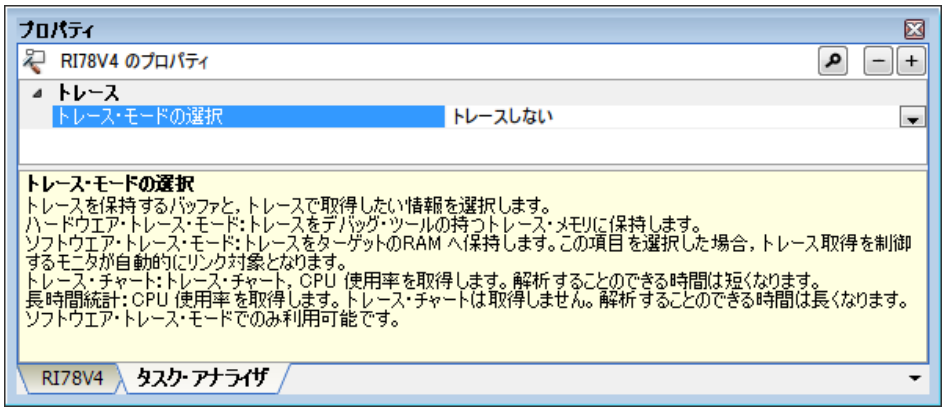
ビルドの生成物として，ロード・モジュール・ファイルを出力することを設定します。

備考 ロード・モジュールの出力指定についての詳細は，「CS+ 統合開発環境 ユーザーズマニュアル RL78 ビルド編」を参照してください。

6） トレース機能の設定

プロパティ パネルの［タスク・アナライザ］タブで，RI78V4 が提供しているユーティリティ・ツール“タスク・アナライザ・ツール”を利用して処理プログラムの実行履歴（トレース・データ）を解析する際に必要となる情報を設定します。

図２－４ ［タスク・アナライザ］タブ



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。

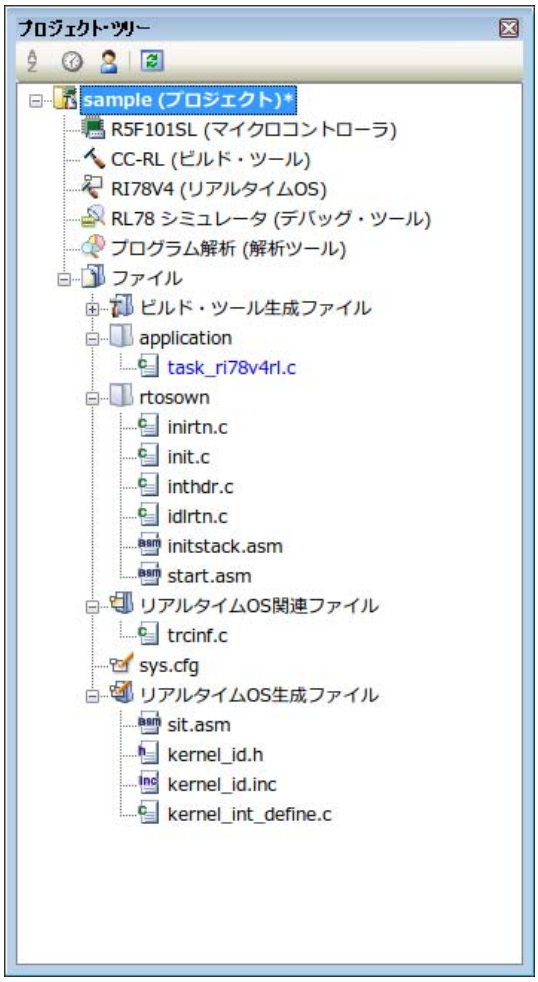
備考 ビルド・オプションの設定についての詳細は，「CS+ 統合開発環境 ユーザーズマニュアル RL78 ビルド編」を参照してください。

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，「CS+ 統合開発環境 ユーザーズマニュアル RL78 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，「CS+ 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。

2.7　システムへの組み込み

「2.6　ロード・モジュールの生成」の 4) でヘキサ・ファイルの出力を設定すると，ヘキサ・ファイルも生成します。
その後，フラッシュメモリへの書き込みなどによりシステムへの組み込みを行います。

第３章　タスク管理機能

本章では，RI78V4 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

RI78V4 におけるタスク管理機能では，タスクの状態を操作する機能のほかに，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

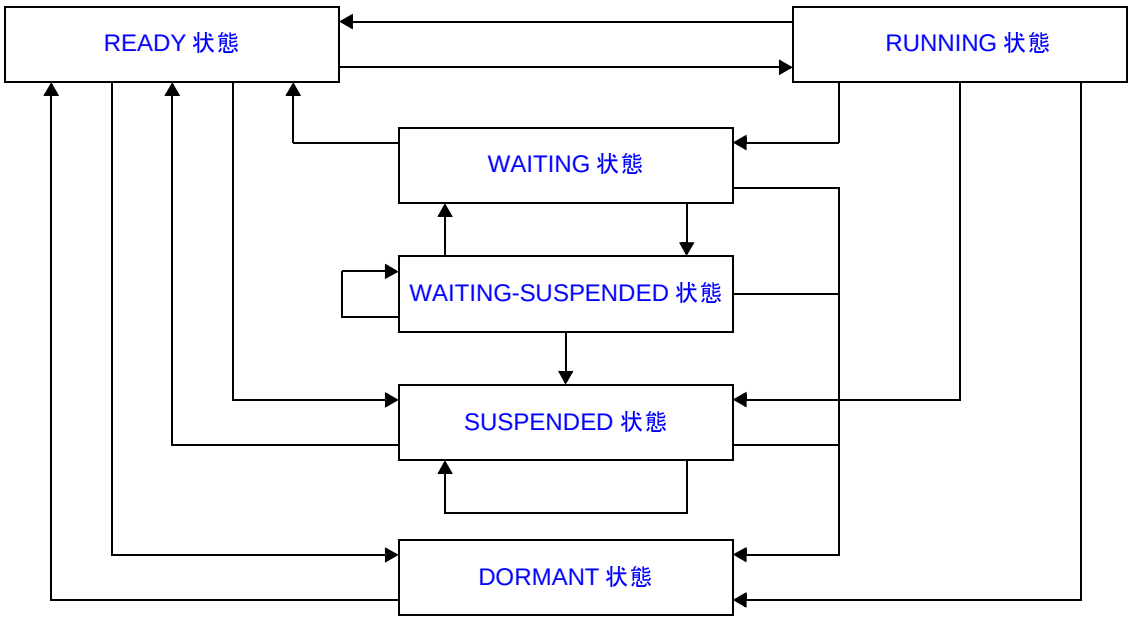
タスクは，他の処理プログラム（周期ハンドラ，割り込みハンドラなど）とは異なり，RI78V4 が提供するサービス・コールを使用して明示的に操作しない限り実行されることのない処理プログラムであり，スケジューラから呼び出されます。

備考　タスクが処理を実行するうえで必要となる実行環境情報は，“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，RI78V4 により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 OS 資源の獲得状況，および，事象発生の有無などにより，様々な状態へと遷移していきます。そこで，RI78V4 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，RI78V4 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



- 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，RI78V4 の管理下にありながらも，RI78V4 のスケジューリング対象からは除外されています。

- 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，または同一優先度の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。

- 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。

- 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・コンテキストなど）は，中断直前の値が復元されます。
  また，RI78V4 では，要求条件の種類により，WAITING 状態を以下に示す 6 種類に細分化し，管理しています。

表 3 － 1　WAITING 状態の種類

状態種別	意味
起床待ち状態	slp_tsk，または tslp_tsk を発行した際，自タスクの起床要求カウンタ（起床要求の発行回数を保持）が 0x0 の場合に遷移する状態です。
時間経過待ち状態	dly_tsk を発行した際に遷移する状態です。
資源待ち状態	wai_sem，または twai_sem 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場合に遷移する状態です。
イベントフラグ待ち状態	wai_flg，または twai_flg 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遷移する状態です。
データ送信待ち状態	snd_dtq，または tsnd_dtq を発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むことができなかった場合に遷移する状態です。
データ受信待ち状態	rcv_dtq，または trcv_dtq を発行した際，対象データ・キューからデータを受信することができなかった場合に遷移する状態です。
メッセージ待ち状態	rcv_mbx，または trcv_mbx 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場合に遷移する状態です。
メモリ・ブロック待ち状態	get_mpf，または tget_mpf を発行した際，対象固定長メモリ・プールからメモリ・ブロックを獲得することができなかった場合に遷移する状態です。

- 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・コンテキストなど）は，中断直前の値が復元されます。

- 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”が付与されています。
これにより，RI78V4 では，実行可能な状態（RUNNING 状態，および，READY 状態）へと遷移している全タスクの中から“最も高い優先度（最高優先度）を持つタスク”を選び出し，CPU の利用権を与えます。
なお，RI78V4 では，“優先度”を以下に示した 2 種類に分類し，管理しています。

- 初期優先度
  タスクの生成時に設定される優先度です。

- 現在優先度
  タスクが DORMANT 状態から READY 状態へと遷移したのち，再び DORMANT 状態へと遷移するまでの優先度の総称です。
  したがって，タスクが DORMANT 状態から READY 状態へと遷移した際の現在優先度は“初期優先度と同値”となり，chg_pri，ichg_pri の発行により優先度の変更が行われた際の現在優先度は“変更後優先度と同値”となります。

備考 1　RI78V4 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は，優先度情報で指定された優先度範囲となります。

3.2.3　タスクの生成

RI78V4 では，タスクの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，タスクを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  タスクの静的な生成は，システム・コンフィギュレーション・ファイルにタスク情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとにタスクの生成処理を実行し，管理対象とします。

3.2.4　タスクの削除

RI78V4 では，カーネル初期化部において静的に生成されたタスクを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

3.2.5　タスクの基本型

タスクを記述する場合，VP_INT 型の引き数を 1 つ持った void 型の関数（関数名：任意）として記述します。
なお，引き数 *exinf* には“タスク情報で指定した拡張情報，または sta_tsk，ista_tsk 発行時に指定した起動コード”が設定されます。
以下に，タスクの基本型を示します。

【C 言語で記述する場合】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        .............                   /* タスクの本体処理 */
        .............

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

備考　#pragma rtos_task 指令は，kernel_id.h 内に定義されています（CF78V4 が自動的に出力します）。そのため，kernel_id.h は必ずインクルードしてください。

【アセンブリ言語で記述する場合】

```
$INCLUDE        (kernel.inc)            ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)         ; システム情報ヘッダ・ファイルの定義

        .PUBLIC  _func_task
        .section .text, TEXT
_func_task:
        PUSH    BC                      ; 引き数 exinf の上位 2 バイトをスタックに保存
        PUSH    AX                      ; 引き数 exinf の下位 2 バイトをスタックに保存

        .............                   ; タスクの本体処理
        .............

        BR      !!_ext_tsk              ; 自タスクの終了
```

3.2.6　タスク内での処理

RI78V4 では，タスクを切り替える際に“独自のディスパッチ処理（タスクのスケジューリング処理）”を実行しています。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。

- 記述方法
  タスクは，「3.2.5　タスクの基本型」で示された関数形式で C 言語，またはアセンブリ言語を用いて記述します。

- スタックの切り替え
  RI78V4 では，タスクを切り替える際に“切り替え先のタスク用スタック（タスク・スタック）への切り替え処理”を実行しています。
  したがって，ユーザは，タスク内でスタックの切り替えに関する処理を記述する必要はありません。

- 割り込み状態
  RI78V4 では，タスクを READY 状態から RUNNING 状態へと遷移した際，“タスク情報において指定した初期割り込み状態”としています。
  したがって，タスク内で割り込み状態を変更（禁止／許可）する場合は，__DI，__EI 関数の呼び出しが必要となります。

- サービス・コールの発行
  タスク内で発行可能なサービス・コールは，“タスクから発行可能なサービス・コール”に限られます。

  備考　各サービス・コールの発行有効範囲についての詳細は，表 12 － 8 ～表 12 － 17 を参照してください。

3.3 タスクの起動

RI78V4 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。

3.3.1 起動要求をキューイングする

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 状態へと遷移させます。
  これにより，対象タスクは，初期優先度に対応したレディ・キューの最後尾にキューイングされ，RI78V4 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 状態以外の状態へと遷移していた場合には，状態操作処理は実行されず，起動要求カウンタの加算処理（起動要求カウンタに 0x1 を加算）が実行されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 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　RI78V4 が管理する起動要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，起動要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

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，起動コードとして“拡張情報 exinf”が引き渡されます。

3.3.2 起動要求をキューイングしない

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 状態へと遷移させます。
  これにより，対象タスクは，初期優先度に対応したレディ・キューの最後尾にキューイングされ，RI78V4 のスケジューリング対象となります。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 変数の宣言，初期化 */
        VP_INT  stacd = 1048575;        /* 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）*/

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

備考 1　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，状態操作処理は実行されず，戻り値として“E_OBJ”が返されます。

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，起動コードとして“*stacd*”が引き渡されます。

3.4　起動要求の解除

起動要求の解除は，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_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 0x0 を設定）します。
  なお，本サービス・コールが正常終了した際には，戻り値として“解除した起動要求数”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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

        ercd = can_act ( tskid );       /* 起動要求の解除 */

        if ( ercd >= 0x0 ) {
                .............           /* 正常終了処理 */
                .............
        }

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

3.5 タスクの終了

RI78V4 では，タスクの終了において，“自タスクの終了”，“他タスクの強制終了”の 2 種類のインタフェースを用意しています。

3.5.1 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 状態へと遷移させます。
  これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起動要求がキューイングされていた（起動要求カウンタが 0x0 以外であった）場合には，RUNNING 状態から DORMANT 状態への状態操作処理，および，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）を行ったのち，DORMANT 状態から READY 状態への状態操作処理もあわせて実行されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ............
        ............

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

備考 1　本サービス・コールでは，自タスクが sig_sem，get_mpf などの発行により獲得した OS 資源の返却は行いません。したがって，獲得中の OS 資源については，本サービス・コールを発行する以前に返却する必要があります。

備考 2　本サービス・コールでは，RUNNING 状態から DORMANT 状態への状態操作処理を実行する際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスクの生成時に設定される値で初期化しています。

備考 3　タスク内で return 命令を記述した場合，本サービス・コールと同様の動作が実行されます。

備考 4　RI78V4 では，“自タスクの終了”として return 命令を記述した方がコード効率が良くなります。

3.5.2 他タスクの強制終了

他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，RI78V4 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，対象タスクに起動要求がキューイングされていた（起動要求カウンタが 0x0 以外であった）場合には，DORMANT 状態への状態操作処理，および，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）を行ったのち，DORMANT 状態から READY 状態への状態操作処理もあわせて実行されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

備考 1　本サービス・コールでは，対象タスクが sig_sem，get_mpf などの発行により獲得した OS 資源の返却は行いません。したがって，獲得中の OS 資源については，本サービス・コールを発行する以前に返却する必要があります。

備考 2　本サービス・コールでは，DORMANT 状態への状態操作処理を実行する際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスクの生成時に設定される値で初期化しています。

3.6　優先度の変更

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
  パラメータ *tskid* で指定されたタスクの優先度（現在優先度）をパラメータ *tskpri* で指定された値に変更します。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 変数の宣言，初期化 */
        PRI     tskpri = 15;            /* 変数の宣言，初期化 */

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

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

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

備考　本サービス・コールを発行した際，対象タスクが RUNNING 状態，または READY 状態であった場合には，優先度の変更処理を実行したのち，パラメータ *tskpri* で指定された優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

3.7 タスクの状態参照

タスクの状態参照は，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
  パラメータ *tskid* で指定されたタスクのタスク状態情報（現在状態など）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 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 */
        STAT    tskwait;                /* 変数の宣言 */
        ID      wobjid;                 /* 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 );    /* タスクの状態参照 */

        tskstat = pk_rtsk.tskstat;      /* 現在状態の獲得 */
        tskpri = pk_rtsk.tskpri;        /* 現在優先度の獲得 */
        tskwait = pk_rtsk.tskwait;      /* 待ち要因の獲得 */
        wobjid = pk_rtsk.wobjid;        /* 管理オブジェクトの ID 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　タスク状態情報 T_RTSK についての詳細は，「12.5.1　タスク状態情報」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，RI78V4 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

RI78V4 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

起床待ち状態への移行（永久待ち，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。

- 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。
  これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起床要求がキューイングされていた（起床要求カウンタが 0x0 以外であった）場合には，状態操作処理は実行されず，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）が実行されます。
  なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
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，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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 */
```

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

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

- tslp_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。
  これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起床要求がキューイングされていた（起床要求カウンタが 0x0 以外であった）場合には，状態操作処理は実行されず，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）が実行されます。
  なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
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，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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の WAITING 状態（起床待ち状態）を解除します。
  これにより，対象タスクは，起床待ち状態からREADY 状態へ，またはWAITING-SUSPENDED 状態からSUSPENDED 状態へと遷移します。
  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であった場合には，状態操作処理は実行されず，起床要求カウンタの加算処理（起床要求カウンタに 0x1 を加算）が実行されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

備考 1　本サービス・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　RI78V4 が管理している起床要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，起床要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

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 0x0 を設定）します。
  なお，本サービス・コールが正常終了した際には，戻り値として“解除した起床要求数”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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

        if ( ercd >= 0x0 ) {
                .............           /* 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”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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 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・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　本サービス・コールでは，強制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”が返されます。

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のサスペンド要求カウンタに 0x1 を加算したのち，対象タスクを 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，状態操作処理は実行されず，サスペンド要求カウンタの加算処理のみが実行されます。
  なお，SUSPENDED 状態の解除は，以下の場合に行われ，SUSPENDED 状態から READY 状態へと遷移します。

SUSPENDED 状態の解除操作	エラー・コード
rsm_tsk の発行により，解除要求が発行された	E_OK
irsm_tsk の発行により，解除要求が発行された	E_OK
frsm_tsk の発行により，SUSPENDED 状態を強制的に解除された	E_OK
ifrsm_tsk の発行により，SUSPENDED 状態を強制的に解除された	E_OK

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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

備考 1　本サービス・コールを発行した際，対象タスクが自タスクの場合は，レディ・キューから外れ，RI78V4 のスケジューリング対象から除外されます。

備考 2　RI78V4 が管理するサスペンド要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，サスペンド要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

4.7 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のサスペンド要求カウンタから0x1を減算したのち，対象タスクのSUSPENDED状態を解除します。
  これにより，対象タスクは，SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移します。
  ただし，本サービス・コールを発行した際，サスペンド要求がキューイングされていた（減算結果が 0x0 以外）場合には，状態操作処理は実行されず，サスペンド要求カウンタの減算処理のみが実行されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 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・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　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”が返されます。

- frsm_tsk，ifrsm_tsk

  パラメータ *tskid* で指定されたタスクのサスペンド要求カウンタに 0x0 を設定し，対象タスクの SUSPENDED 状態を強制的に解除します。
  これにより，対象タスクは，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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      tskid = ID_tskA;        /* 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・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　本サービス・コールでは，強制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”が返されます。

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 RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
  これにより，自タスクは，レディ・キューから外れ，RI78V4 のスケジューリング対象から除外されます。
  なお，時間経過待ち状態の解除は，以下の場合に行われ，時間経過待ち状態から READY 状態へと遷移します。

時間経過待ち状態の解除操作	エラー・コード
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，時間経過待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，時間経過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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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 */
                .............
        }

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

第５章　同期通信機能

本章では，RI78V4 が提供している同期通信機能について解説しています。

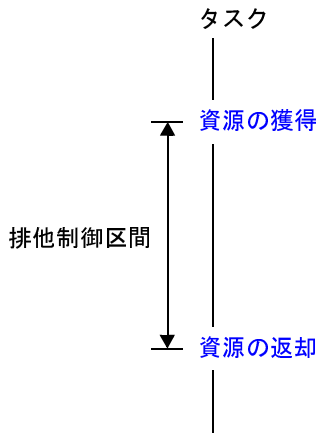
5.1　概　　要

RI78V4 における同期通信機能では，タスク間の排他制御，待ち合わせ，通信を実現する手段としてセマフォ，イベントフラグ，メールボックスを提供しています。

5.2　セマフォ

RI78V4 では，並行に動作するタスクが限られた数の資源（ハードウエア・デバイス，ライブラリ関数など）を同時に利用するといった資源使用の競合を防ぐ手段（排他制御機能）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

RI78V4 では，セマフォの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，セマフォを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  セマフォの静的な生成は，システム・コンフィギュレーション・ファイルにセマフォ情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとにセマフォの生成処理を実行し，管理対象とします。

5.2.2　セマフォの削除

RI78V4 では，カーネル初期化部において静的に生成されたセマフォを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

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 0x1 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，カウンタ操作処理は実行されず，該当タスク（待ちキューの先頭タスク）に資源が渡されます。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      semid = ID_semA;        /* 変数の宣言，初期化 */

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

        sig_sem ( semid );              /* 資源の返却 */

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

備考 1　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　RI78V4 が管理しているセマフォ・カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，資源数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値とし“て E_QOVR”が返されます。

5.2.4 資源の獲得

資源の獲得（永久待ち，ポーリング，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。

- wai_sem
  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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，自タスクを対象セマフォの待ちキューに資源の獲得要求順（FIFO 順）でキューイングします。
  これにより，自タスクは，レディ・キューから外れ，RUNNING状態からWAITING状態（資源待ち状態）へと遷移します。
  なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から READY 状態へと遷移します。

資源待ち状態の解除操作	エラー・コード
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，資源待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

- pol_sem
  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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，戻り値として“E_TMOUT”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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

        ercd = pol_sem ( semid );       /* 資源の獲得（ポーリング）*/

        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 */
                .............
        }

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

- twai_sem
  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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，自タスクを対象セマフォの待ちキューに資源の獲得要求順（FIFO 順）でキューイングします。
  これにより，自タスクは，レディ・キューから外れ，RUNNING状態からWAITING状態（資源待ち状態）へと遷移します。
  なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から READY 状態へと遷移します。

資源待ち状態の解除操作	エラー・コード
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，資源待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源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>           /* システム情報ヘッダ・ファイルの定義 */

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

        .............
        .............
                                        /* 資源の獲得（タイムアウト付き）*/
        ercd = twai_sem ( semid, 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　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。

5.2.5 セマフォの状態参照

セマフォの状態参照は，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

  パラメータ *semid* で指定されたセマフォのセマフォ状態情報（待ちタスクの有無など）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。

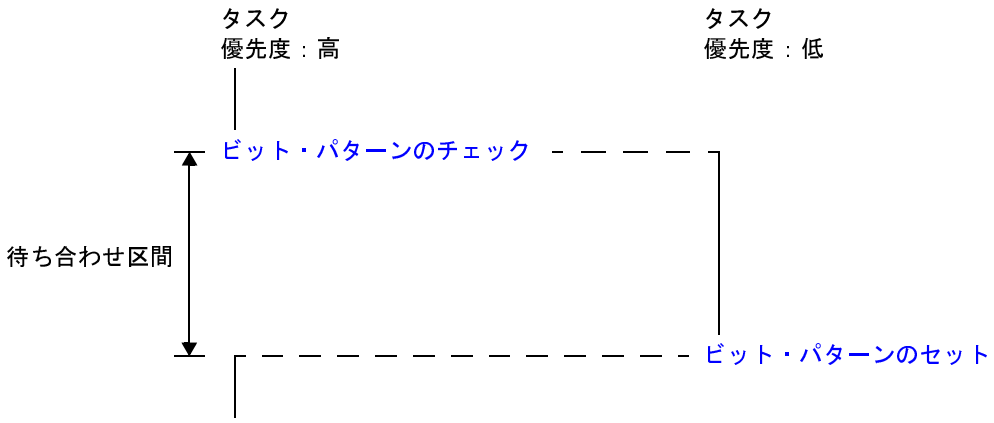
```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      semid = ID_semA;        /* 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 );    /* セマフォの状態参照 */

        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　セマフォ状態情報 T_RSEM についての詳細は，「12.5.2　セマフォ状態情報」を参照してください。

5.3　イベントフラグ

RI78V4 では，ある処理プログラムの実行結果が出るまでの間，“処理の実行を待つ”といったタスク間の待ち合わせ機能として“16 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

RI78V4 では，イベントフラグの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，イベントフラグを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  イベントフラグの静的な生成は，システム・コンフィギュレーション・ファイルにイベントフラグ情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとにイベントフラグの生成処理を実行し，管理対象とします。

  備考　RI78V4 では，イベントフラグの生成処理において，“0x0”を初期ビット・パターンとして設定します。

5.3.2　イベントフラグの削除

RI78V4 では，カーネル初期化部において静的に生成されたイベントフラグを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

5.3.3 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 を対象イベントフラグのビット・パターンとして設定します。
  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待ちキューにキューイングされているタスクの要求条件を満足した場合には，ビット・パターンの設定処理とともに，該当タスクを待ちキューから外します。
  これにより，該当タスクは，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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      flgid = ID_flgA;        /* 変数の宣言，初期化 */
        FLGPTN  setptn = 0B1010;        /* 変数の宣言，初期化 */

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

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

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

備考 1　本サービス・コールを発行した際，待ちキューのタスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　本サービス・コールを発行した際，対象イベントフラグに設定されているビット・パターンが B'1100，パラメータ *setptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1110 となります。

5.3.4 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

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 を対象イベントフラグのビット・パターンとして設定します。
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>           /* システム情報ヘッダ・ファイルの定義 */

func_task ( VP_INT exinf )
{
        ID      flgid = ID_flgA;        /* 変数の宣言，初期化 */
        FLGPTN  clrptn = 0B1010;        /* 変数の宣言，初期化 */

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

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

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

備考 1　本サービス・コールでは，クリア要求のキューイングが行われません。このため，ビット・パターンがクリア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・コールを発行した際，対象イベントフラグに設定されているビット・パターンが B'1100，パラメータ *clrptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1000 となります。

備考 3　本サービス・コールでは，イベントフラグ待ち状態のタスクが“待ち解除”されることはありません。

5.3.5 ビット・パターンのチェック

ビット・パターンのチェック（永久待ち，ポーリング，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。

- wai_flg
  パラメータ *waiptn* で指定されたビット・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ます。
  これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移します。
  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，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
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，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ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
  パラメータ *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
  パラメータ *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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = ID_flgA;        /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 0B1110;        /* 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（永久待ち）*/
        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 RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

- pol_flg

パラメータ *waiptn* で指定されたビット・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”が返されます。
以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  パラメータ *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
  パラメータ *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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = ID_flgA;        /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 0B1110;        /* 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（ポーリング）*/
        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　RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

- twai_flg
  パラメータ *waiptn* で指定されたビット・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ます。
  これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移します。
  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，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
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，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ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
  パラメータ *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
  パラメータ *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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = ID_flgA;        /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 0B1110;        /* 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（タイムアウト付き）*/
        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　RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

備考 3　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。

5.3.6 イベントフラグの状態参照

イベントフラグの状態参照は，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
  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状態情報（待ちタスクの有無など）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      flgid = ID_flgA;        /* 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 );    /* イベントフラグの状態参照 */

        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 */

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

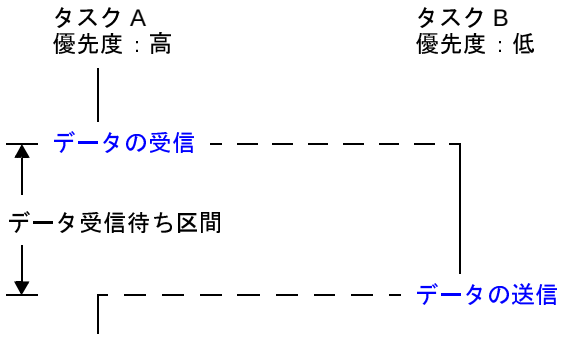
備考　イベントフラグ状態情報 T_RFLG についての詳細は，「12.5.3　イベントフラグ状態情報」を参照してください。

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，RI78V4 では，規定されたデータ・サイズの通信機能として“データの書き込み／読み出しが可能なデータ・キュー領域を有するデータ・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

RI78V4 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“CRE_DTQ”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“CRE_DTQ”の詳細は，「13.4.4　データ・キュー情報」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
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，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，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域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・キューのデータ・キュー領域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・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。
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>           /* システム情報ヘッダ・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）に行われます。

- psnd_dtq，ipsnd_dtq

  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，戻り値として E_TMOUT を返します。
  また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。
  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>           /* システム情報ヘッダ・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（ポーリング）*/

        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

パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，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，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域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・キューのデータ・キュー領域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・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信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

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。
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>           /* システム情報ヘッダ・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（タイムアウト付き）*/
        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）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，ipsnd_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・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，書き込まれてから最も時間が経過しているデータの領域に該当データを上書きします。
  また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。
  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>           /* システム情報ヘッダ・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 */

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

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
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，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，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信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>           /* システム情報ヘッダ・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 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 rel_wai，または irel_wai の発行によりデータ受信待ち状態を解除された場合，パラメータ *p_data* で指定された領域の内容は不定となります。

- prcv_dtq

パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，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>           /* システム情報ヘッダ・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（ポーリング）*/
        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 */
                .............
        }

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

備考　本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合，パラメータ *p_data* で指定された領域の内容は不定となります。

- trcv_dtq
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，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 状態からタイムアウト付きの 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，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信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>           /* システム情報ヘッダ・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（タイムアウト付き）*/
        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 rel_wai，または irel_wai の発行，または待ち時間の経過によりデータ受信待ち状態を解除された場合，パラメータ *p_data* で指定された領域の内容は不定となります。

備考 3 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
  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など）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。

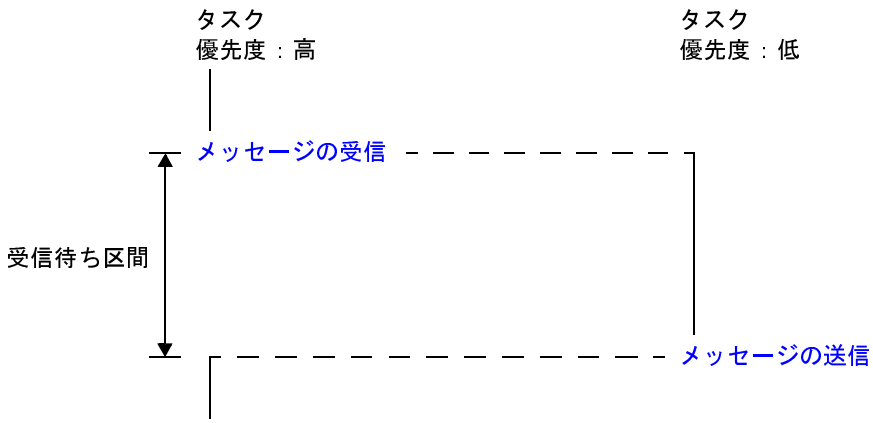
```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

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，「12.5.4　データ・キュー詳細情報」を参照してください。

5.5　メールボックス

RI78V4 では，ある処理プログラムの実行結果を他の処理プログラムに引き渡すといったタスク間の通信機能として“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　メールボックスの生成

RI78V4 では，メールボックスの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，メールボックスを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  メールボックスの静的な生成は，システム・コンフィギュレーション・ファイルにメールボックス情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとにメールボックスの生成処理を実行し，管理対象とします。

5.5.2　メールボックスの削除

RI78V4 では，カーネル初期化部において静的に生成されたメールボックスを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

5.5.3 メッセージ

RI78V4 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，RI78V4 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。

- メモリ領域の確保
  RI78V4 では，get_mpf，pget_mpf などといったサービス・コール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。

  備考 RI78V4 では，メッセージの先頭領域をメールボックスのメッセージ用待ちキューにキューイングする際のリンク領域として使用します。このため，メッセージ用のメモリ領域を RI78V4 が管理しているメモリ領域以外から確保する場合は，4 バイト・アラインされたアドレスから確保する必要があります。

- メッセージの基本型
  RI78V4 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
    RI78V4 では，メッセージの先頭 4 バイト（システム予約領域 msgque）以降の内容，長さについては特に規定していません。
    したがって，先頭 4 バイト以降の内容，長さについては，TA_MFIFO 属性の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 C 言語で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 T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   __near   *msgque;        /* システム予約領域 */
} T_MSG;
```

  - TA_MPRI 属性のメールボックスを利用する場合
    RI78V4 では，メッセージの先頭 5 バイト（システム予約領域 msgque，優先度 msgpri）以降の内容，長さについては特に規定していません。
    したがって，先頭 5 バイト以降の内容，長さについては，TA_MPRI 属性の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 C 言語で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 T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   __near   *msgque;        /* システム予約領域 */
        PRI     msgpri;                         /* 優先度 */
} T_MSG_PRI;
```

    備考 1 RI78V4 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，“1 ～ 31”に限られます。
    備考 3 メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5 メッセージ」を参照してください。

5.5.4 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

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（待ちキューにメッセージをキューイング）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，メッセージのキューイング処理は実行されず，該当タスク（待ちキューの先頭タスク）にメッセージが渡されます。
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>           /* システム情報ヘッダ・ファイルの定義 */

func_task ( VP_INT exinf )
{
        ID      mpfid = ID_mpfA;        /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */
        char    *p;                     /* 変数の宣言 */
        ID      mbxid = ID_mbxA;        /* 変数の宣言，初期化 */
        T_MSG_PRI       *pk_msg;        /* データ構造体の宣言 */

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

        get_mpf ( mpfid, &p_blk );      /* メモリ領域（メッセージ用）の確保 /
                                        /* 変数の初期化 */
        p = (char *)p_blk + sizeof (T_MSG_PRI);

        while ( expr ) {
                *p++ = .............    /* メッセージ本体（内容）の作成 */
        }

        (T_MSG_PRI *)p_blk->msgpri = 8; /* データ構造体の初期化 */
                                        /* メッセージの送信 */
        snd_mbx ( mbxid, (T_MSG_PRI *)p_blk );

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

備考 1　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　メッセージ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属性（キューイング方式）mbxatr で定義された順（FIFO 順，優先度順）となります。

備考 3　RI78V4 のメールボックスは，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ありません。したがって，本サービス・コールの発行後であってもメッセージの内容を書き換えることができます。

備考 4　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」を参照してください。

5.5.5 メッセージの受信

メッセージの受信（永久待ち，ポーリング，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。

- rcv_mbx

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，メッセージの受信処理は実行されず，自タスクを対象メールボックスの待ちキューにメッセージの受信要求順（FIFO 順）でキューイングします。
これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メッセージ待ち状態）へと遷移します。
なお，メッセージ待ち状態の解除は，以下の場合に行われ，メッセージ待ち状態から READY 状態へと遷移します。

メッセージ待ち状態の解除操作	エラー・コード
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
rel_wai の発行により，メッセージ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ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>           /* システム情報ヘッダ・ファイルの定義 */

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

        ............
        ............
                                        /* メッセージの受信（永久待ち）*/
        ercd = rcv_mbx ( mbxid, &ppk_msg );

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

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

備考　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」を参照してください。

- prcv_mbx

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，メッセージの受信処理は実行されず，戻り値として“E_TMOUT”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>           /* システム情報ヘッダ・ファイルの定義 */

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

        .............
        .............
                                        /* メッセージの受信（ポーリング）*/
        ercd = prcv_mbx ( mbxid, &ppk_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，「12.5.5　メッセージ」を参照してください。

- trcv_mbx

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，メッセージの受信処理は実行されず，自タスクを対象メールボックスの待ちキューにメッセージの受信要求順（FIFO 順）でキューイングします。
これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メッセージ待ち状態）へと遷移します。
なお，メッセージ待ち状態の解除は，以下の場合に行われ，メッセージ待ち状態から READY 状態へと遷移します。

メッセージ待ち状態の解除操作	エラー・コード
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
rel_wai の発行により，メッセージ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = ID_mbxA;        /* 変数の宣言，初期化 */
        T_MSG   *ppk_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（タイムアウト付き）*/
        ercd = trcv_mbx ( mbxid, &ppk_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　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 2　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」を参照してください。

5.5.6 メールボックスの状態参照

メールボックスの状態参照は，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
  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状態情報（待ちタスクの有無など）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      mbxid = ID_mbxA;        /* 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 );    /* メールボックスの状態参照 */

        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　メールボックス状態情報 T_RMBX についての詳細は，「12.5.6　メールボックス状態情報」を参照してください。

第 6 章　メモリ・プール管理機能

本章では，RI78V4 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。

6.1　概　　要

RI78V4 におけるメモリ・プール管理機能では，カーネル初期化部において静的に確保されたメモリ領域を管理対象としています。
なお，RI78V4 では，機能単位にモジュール化された管理オブジェクトの割り付け先（セクション名）を規定しています。
以下に，RI78V4 が規定しているセクション名の一覧を示します。

- .kernel_system セクション
  RI78V4 の核となる処理部，および，RI78V4 が提供するサービス・コールの本体処理部が割り付けられる領域です。

- .kernel_system_timer_n セクション
  システム・タイマ割り込み処理部，および，far 分岐情報が割り付けられる領域です。

- .kernel_info セクション
  RI78V4 のバージョンなどといった情報が割り付けられる領域です。

- .kernel_const ／ .kernel_const_f セクション
  動的に変化することのない OS 資源に関する初期情報が，システム情報テーブルや割り込み情報定義ファイルとして割り付けられる領域です。

- .kernel_stack セクション
  システム・スタック，および，タスク・スタックが割り付けられる領域です。

- .kernel_data セクション
  RI78V4 が提供する機能を実現するうえで必要となる情報，および，動的に変化する OS 資源に関する情報が管理オブジェクトとして割り付けられる領域です。

- .kernel_data_init セクション
  RI78V4 の初期化情報が割り付けられる領域です。

- .kernel_work0，.kernel_work1，.kernel_work2，.kernel_work3 セクション
  データキューのデータ，固定長メモリ・プールが割り付けられる領域です。

- .kernel_data_trace_n セクション
  トレース・データが割り付けられる領域です。

- .kernel_const_trace_f セクション
  トレース・データの取得に必要な情報が割り付けられる領域です。

- .kernel_system_trace_f セクション
  トレース・データ取得処理部が割り付けられる領域です。

- .kernel_sbss セクション
  RI78V4 が使用する SADDR 領域です。

6.2　固定長メモリ・プール

RI78V4 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可能なメモリ領域として“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，固定サイズのメモリ・ブロックを単位として行います。

6.2.1　固定長メモリ・プールの生成

RI78V4 では，固定長メモリ・プールの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，固定長メモリ・プールを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  固定長メモリ・プールの静的な生成は，システム・コンフィギュレーション・ファイルに固定長メモリ・プール情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとに固定長メモリ・プールの生成処理を実行し，管理対象とします。

6.2.2　固定長メモリ・プールの削除

RI78V4 では，カーネル初期化部において静的に生成された固定長メモリ・プールを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

6.2.3　メモリ・ブロックの獲得

メモリ・ブロックの獲得（永久待ち，ポーリング，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。

- get_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールからメモリ・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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，自タスクを対象固定長メモリ・プールの待ちキューにメモリ・ブロックの獲得要求順（FIFO 順）でキューイングします。これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メモリ・ブロック待ち状態）へと遷移します。
  なお，メモリ・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から READY 状態へと遷移します。

メモリ・ブロック待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールにメモリ・ブロックが返却された	E_OK
rel_wai の発行により，メモリ・ブロック待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メモリ・ブロック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>           /* システム情報ヘッダ・ファイルの定義 */

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

        .............
        .............
                                        /* メモリ・ブロックの獲得（永久待ち）*/
        ercd = get_mpf ( mpfid, &p_blk );

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

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

- pget_mpf

パラメータ *mpfid* で指定された固定長メモリ・プールからメモリ・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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，戻り値として“E_TMOUT”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>           /* システム情報ヘッダ・ファイルの定義 */

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

        .............
        .............
                                        /* メモリ・ブロックの獲得（ポーリング）*/
        ercd = pget_mpf ( mpfid, &p_blk );

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

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

- tget_mpf

パラメータ *mpfid* で指定された固定長メモリ・プールからメモリ・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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，自タスクを対象固定長メモリ・プールの待ちキューにメモリ・ブロックの獲得要求順（FIFO 順）でキューイングします。
これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メモリ・ブロック待ち状態）へと遷移します。
なお，メモリ・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から READY 状態へと遷移します。

メモリ・ブロック待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールにメモリ・ブロックが返却された	E_OK
rel_wai の発行により，メモリ・ブロック待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メモリ・ブロック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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = ID_mpfA;        /* 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 */

        ............
        ............
                                        /* メモリ・ブロックの獲得（タイムアウト付き）*/
        ercd = tget_mpf ( mpfid, &p_blk, tmout );

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

6.2.4 メモリ・ブロックの返却

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

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，メモリ・ブロックの返却処理は実行されず，該当タスク（待ちキューの先頭タスク）にメモリ・ブロックが渡されます。
これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メモリ・ブロック待ち状態）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>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

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

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

備考 1　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　RI78V4　では，メモリ・ブロックを返却する際，メモリ・ブロックのクリア処理を行っていません。したがって，返却されたメモリ・ブロックの内容は不定となります。

6.2.5　固定長メモリ・プールの状態参照

固定長メモリ・プールの状態参照は，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
  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・プール状態情報（待ちタスクの有無など）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      mpfid = ID_mpfA;        /* 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・プールの状態参照 */

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

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

備考　固定長メモリ・プール状態情報 T_RMPF についての詳細は，「12.5.7　固定長メモリ・プール状態情報」を参照してください。

第 7 章　時間管理機能

本章では，RI78V4 が提供している時間管理機能について解説しています。

7.1　概　　要

RI78V4 における時間管理機能では，時間に依存した処理を実現する手段として，一定の周期で発生するタイマ割り込みを利用した遅延起床，タイムアウト，周期ハンドラを提供しています。

備考　RI78V4 では，タイマ割り込みを発生させるハードウエア（クロック・コントローラなど）の初期化処理を行いません。したがって，該当初期化処理については，ブート処理，または初期化ルーチンにおいて，ユーザが記述する必要があります。

7.2　タイマ・ハンドラ

タイマ・ハンドラは，タスクの遅延起床，WAITING 状態のタイムアウト，周期ハンドラの起動を実現するうえで必要となる処理を切り出した時間管理処理専用ルーチンであり，タイマ割り込みが発生した際に起動する割り込みハンドラから呼び出されます。

備考　タイマ・ハンドラは，RI78V4 が提供する機能の一部です。したがって，ユーザは，タイマ・ハンドラの処理内容を記述する必要はありません。

7.2.1　タイマ・ハンドラの登録

タイマ・ハンドラの登録は，CF78V4 がシステム・コンフィギュレーション・ファイルに定義された基本クロック用タイマ割り込み要因を元に割り込み情報定義ファイルに出力するため，ユーザは，タイマ・ハンドラの登録処理を記述する必要はありません。

7.3　遅延起床

遅延起床とは，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。

表７－１　遅延起床

サービス・コール名	機能概要
dly_tsk	時間経過待ち状態への移行

7.4　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。

表７－２　タイムアウト

サービス・コール名	機能概要
tslp_tsk	起床待ち状態への移行
twai_sem	資源の獲得
twai_flg	ビット・パターンのチェック
tsnd_dtq	データの送信
trcv_dtq	データの受信
trcv_mbx	メッセージの受信
tget_mpf	メモリ・ブロックの獲得

7.5 周期ハンドラ

周期ハンドラは，一定の時間（起動周期）を単位として周期的に起動される周期処理専用ルーチンであり，タイマ・ハンドラから呼び出されます。
なお，RI78V4 では，周期ハンドラを“非タスク（タスクとは独立したもの）”として位置づけています。このため，起動周期に達した際には，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。

7.5.1 周期ハンドラの生成

RI78V4 では，周期ハンドラの生成方法を“カーネル初期化部において静的に生成する”に限定しています。
したがって，RI78V4 では，周期ハンドラを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  周期ハンドラの静的な生成は，システム・コンフィギュレーション・ファイルに周期ハンドラ情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとに周期ハンドラの生成処理を実行し，管理対象とします。

7.5.2 周期ハンドラの削除

RI78V4 では，カーネル初期化部において静的に生成された周期ハンドラを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

7.5.3 周期ハンドラの基本型

周期ハンドラを記述する場合，引き数を持たない void 型の関数（関数名：任意）として記述します。
以下に，周期ハンドラの基本型を示します。

【C 言語で記述する場合】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
func_cychdr ( void )
{
        .............                   /* 周期ハンドラの本体処理 */
        .............

        return;                         /* 周期ハンドラの終了 */
}
```

【アセンブリ言語で記述する場合】

```
$INCLUDE        (kernel.inc)            ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)         ; システム情報ヘッダ・ファイルの定義

        .PUBLIC  _func_cychdr
        .SECTION .text, TEXT
_func_cychdr:
        .............                   ; 周期ハンドラの本体処理
        .............

        RET                             ; 周期ハンドラの終了
```

7.5.4 周期ハンドラ内での処理

RI78V4 では，周期ハンドラを“非タスク”として位置づけています。
また，RI78V4 では，周期ハンドラに制御を移す際に“独自の前処理”を，周期ハンドラから制御を戻す際にも“独自の後処理”を実行しています。
このため，周期ハンドラを記述する際には，以下に示す注意点があります。

- 記述方法
  周期ハンドラは，「7.5.3 周期ハンドラの基本型」で示された関数形式で C 言語，またはアセンブリ言語を用いて記述します。

- スタックの切り替え
  RI78V4 では，周期ハンドラに制御を移す際に“システム・スタックへの切り替え処理”を，周期ハンドラから制御を戻す際に“切り替え先の処理プログラム用スタック（システム・スタック，またはタスク・スタック）への切り替え処理”を実行しています。
  したがって，ユーザは，周期ハンドラ内でスタックの切り替えに関する処理を記述する必要はありません。

- 割り込み状態
  RI78V4 では，周期ハンドラに制御を移す際に“マスカブル割り込みの受け付けが禁止された状態”としています。
  したがって，周期ハンドラ内で割り込み状態を変更（許可）する場合は，__EI 関数の呼び出しが必要となります。

- サービス・コールの発行
  RI78V4 では，周期ハンドラを“非タスク”として位置づけています。
  このため，周期ハンドラ内で発行可能なサービス・コールは，“非タスクから発行可能なサービス・コール”に限られます。

  備考 1 サービス・コールの発行有効範囲についての詳細は，表 12 － 8 ～表 12 － 17 を参照してください。

  備考 2 RI78V4 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に“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（ichg_pri，isig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，周期ハンドラからの復帰命令（return 命令の発行など）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。

7.5.5　周期ハンドラの動作開始状態への移行

周期ハンドラの動作開始状態（STA 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。

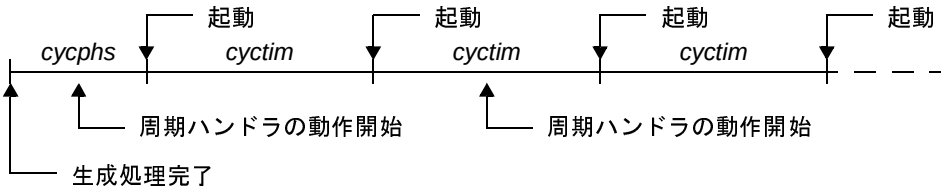
- sta_cyc
  パラメータ *cycid* で指定された周期ハンドラを動作停止状態（STP 状態）から動作開始状態（STA 状態）へと遷移させます。
  これにより，対象周期ハンドラは，RI78V4 の起動対象となります。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      cycid = ID_cycA;        /* 変数の宣言，初期化 */

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

        sta_cyc ( cycid );              /* 動作開始状態（STA 状態）への移行 */

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

備考　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 属性を指定しているか否かにより異なります。

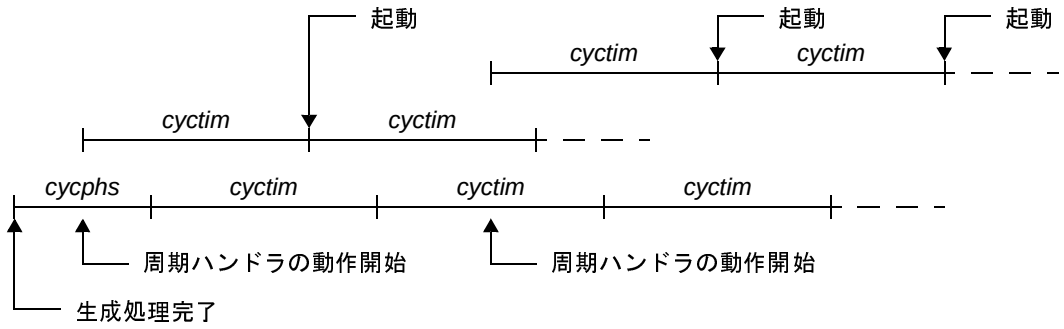
【周期ハンドラの起動イメージ（TA_PHS 属性の指定あり）】

コンフィギュレーション時に定義した起動位相（初期起動位相 *cycphs*，起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
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。



【周期ハンドラの起動イメージ（TA_PHS 属性の指定なし）】

本サービス・コールの発行を基準点とした起動位相（起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
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。


起動
起動
起動
cyctim
cyctim
cyctim
cyctim
cycphs
cyctim
cyctim
cyctim
周期ハンドラの動作開始
周期ハンドラの動作開始
生成処理完了

7.5.6 周期ハンドラの動作停止状態への移行

周期ハンドラの動作停止状態（STP 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。

- stp_cyc
  パラメータ *cycid* で指定された周期ハンドラを動作開始状態（STA 状態）から動作停止状態（STP 状態）へと遷移させます。
  これにより，対象周期ハンドラは，本サービス・コールの発行から sta_cyc が発行されるまでの間，RI78V4 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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      cycid = ID_cycA;        /* 変数の宣言，初期化 */

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

        stp_cyc ( cycid );              /* 動作停止状態（STP 状態）への移行 */

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

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，対象周期ハンドラが動作停止状態（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。

7.5.7 周期ハンドラの状態参照

周期ハンドラの状態参照は，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
  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状態情報（現在状態など）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ID      cycid = ID_cycA;        /* 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 );    /* 周期ハンドラの状態参照 */

        cycstat = pk_rcyc.cycstat;      /* 現在状態の獲得 */
        lefttim = pk_rcyc.lefttim;      /* 残り時間の獲得 */

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

備考　周期ハンドラ状態情報 T_RCYC についての詳細は，「12.5.8　周期ハンドラ状態情報」を参照してください。

第 8 章　システム状態管理機能

本章では，RI78V4 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。

8.1　概　　要

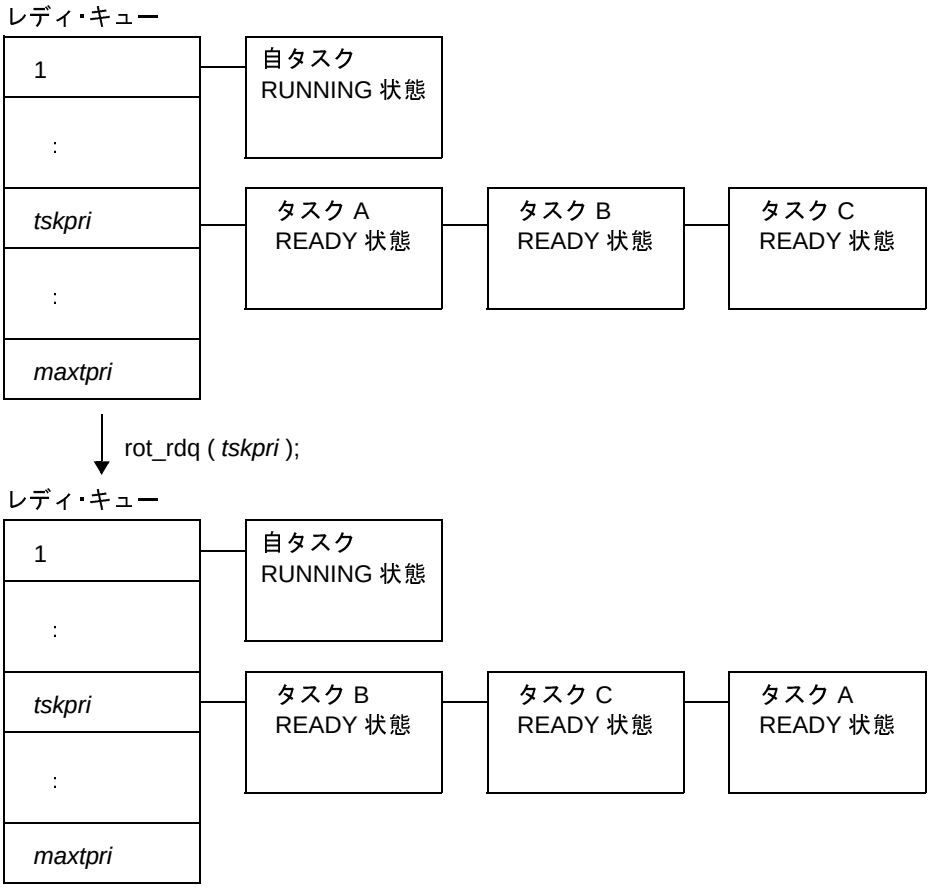
RI78V4 におけるシステム状態管理機能では，CPU ロック状態への移行，ディスパッチ禁止状態への移行などといったシステムの状態を操作する機能のほかに，コンテキスト種別の参照，CPU ロック状態の参照などといったシステムの状態を参照する機能も提供しています。

8.2　レディ・キューの回転

レディ・キューの回転は，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。
  以下に，本サービス・コールを利用した際の状態変化を示します。

図 8 － 1　レディ・キューの回転



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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_cychdr ( void )
{
        PRI     tskpri = 8;             /* 変数の宣言，初期化 */

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

        irot_rdq ( tskpri );            /* レディ・キューの回転 */

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

        return;                         /* 周期ハンドラの終了 */
}
```

備考 1　本サービス・コールでは，回転要求のキューイングが行われません。このため，該当優先度に対応したレディ・キューにタスクが 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　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（READY 状態，または RUNNING 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“タスクの検出処理”を実行し，キューイングされているタスクを検出した際には，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，RI78V4 のスケジューリング方式を実現しています。

8.3　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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_cychdr ( void )
{
        ID      p_tskid;                /* 変数の宣言 */

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

        iget_tid ( &p_tskid );          /*RUNNING 状態のタスクの参照 */

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

        return;                         /* 周期ハンドラの終了 */
}
```

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（IDLE 状態）には，パラメータ *p_tskid* で指定された領域に TSK_NONE を格納します。

8.4　CPU ロック状態への移行

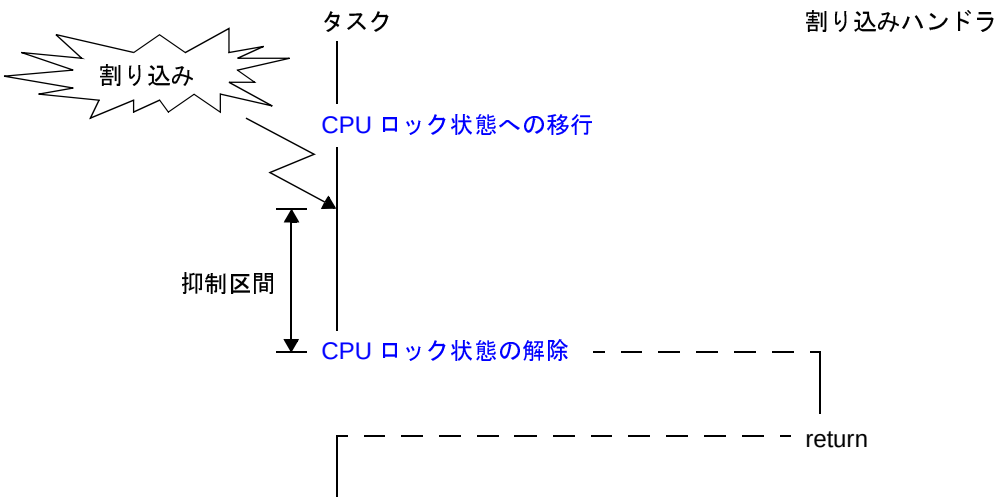
CPU ロック状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- loc_cpu，iloc_cpu
  システム状態種別を CPU ロック状態へと変更します。
  これにより，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，“マスカブル割り込みの受け付け処理”が禁止されるとともに，サービス・コールの発行が制限されます。
  なお，RI78V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を unl_cpu，または iunl_cpu が発行されるまで遅延しています。
  また，RI78V4 では，CPU ロック状態で発行可能なサービス・コールを以下に示したものに制限しています。

サービス・コール名	機能概要
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_ctx	コンテキスト種別の参照
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_dpn	ディスパッチ保留状態の参照

以下に，本サービス・コールを利用した際の処理の流れを示します。

図 8 － 2　CPU ロック状態への移行



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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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　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，システム状態種別が CPU ロック状態であった場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　RI78V4 では，割り込みマスク・フラグ・レジスタ MK*xx*，および，プログラム・ステータス・ワード PSW のインサービス・プライオリティ・フラグ ISP*x* を操作して，“マスカブル割り込みの受け付け禁止”を実現しています。このため，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，処理プログラムから該当レジスタを操作することは禁止されています。

8.5 CPU ロック状態の解除

CPU ロック状態の解除は，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。

- unl_cpu，iunl_cpu
  システム状態種別を非 CPU ロック状態へと変更します。
  これにより，loc_cpu，または iloc_cpu の発行により禁止されていた“マスカブル割り込みの受け付け処理”が許可されるとともに，および，サービス・コールの発行制限が解除されます。
  なお，RI78V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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　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，システム状態種別が非 CPU ロック状態であった場合には，何も処理は行わず，エラーとしても扱いません。

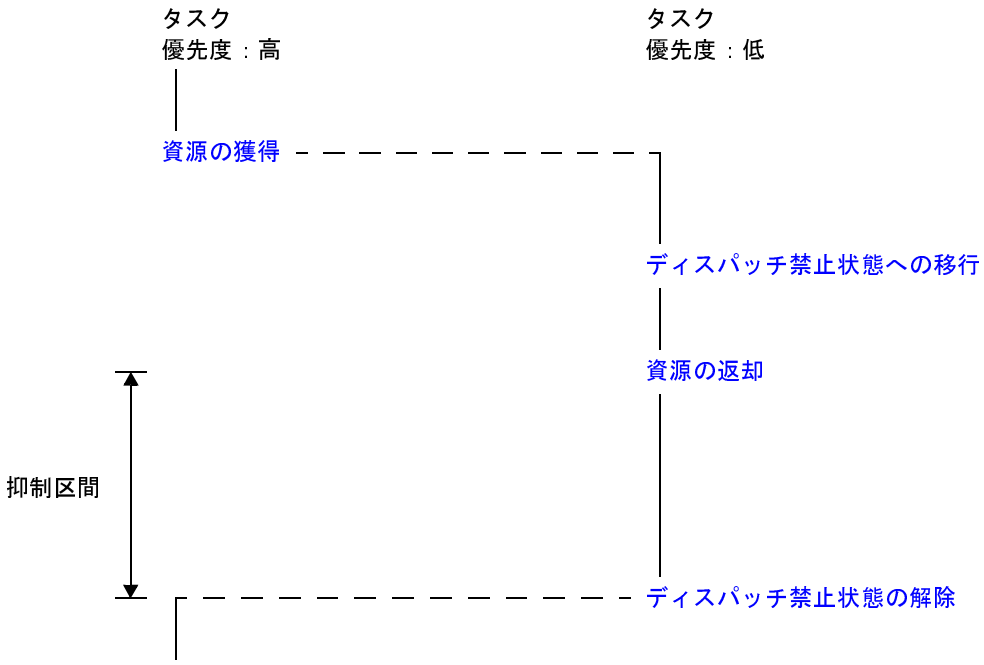
備考 2　RI78V4 では，割り込みマスク・フラグ・レジスタ MK*xx*，および，プログラム・ステータス・ワード PSW のインサービス・プライオリティ・フラグ ISP*x* を操作して，“マスカブル割り込みの受け付け許可”を実現しています。このため，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間，処理プログラムから該当レジスタを操作することは禁止されています。

8.6　ディスパッチ禁止状態への移行

ディスパッチ禁止状態への移行は，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。

- dis_dsp
  システム状態種別をディスパッチ禁止状態へと変更します。
  これにより，本サービス・コールの発行から ena_dsp が発行されるまでの間，ディスパッチ処理（タスクのスケジューリング処理）が禁止されます。
  なお，RI78V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，ena_dsp 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。
  以下に，本サービス・コールを利用した際の処理の流れを示します。

図 8 － 3　ディスパッチ禁止状態への移行



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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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　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，システム状態種別がディスパッチ禁止状態であった場合には，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。

8.7　ディスパッチ禁止状態の解除

ディスパッチ禁止状態の解除は，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。

- ena_dsp
  システム状態種別をディスパッチ許可状態へと変更します。
  これにより，dis_dsp の発行により禁止されていたディスパッチ処理（タスクのスケジューリング処理）が許可されます。
  なお，RI78V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，本サービス・コール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。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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　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，システム状態種別がディスパッチ許可状態であった場合には，何も処理は行わず，エラーとしても扱いません。

8.8　コンテキスト種別の参照

コンテキスト種別の参照は，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。

- sns_ctx
  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）を獲得します。
  なお，本サービス・コールが正常終了した際には，戻り値として“獲得したコンテキスト種別（TRUE：非タスク・コンテキスト，FALSE：タスク・コンテキスト）”が返されます。

  非タスク・コンテキスト：　周期ハンドラ，割り込みハンドラ
  タスク・コンテキスト：　　タスク

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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_sub ( 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 */
                ............
        } else if ( ercd == FALSE ) {
                ............            /* タスク・コンテキスト */
                ............
        }

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

8.9　CPU ロック状態の参照

CPU ロック状態の参照は，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。

- sns_loc
  本サービス・コールを発行した際のシステム状態種別（CPU ロック状態，非 CPU ロック状態）を獲得します。
  なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：CPU ロック状態，FALSE：非 CPU ロック状態）”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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_sub ( 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 */
                ............
        }

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

備考　CPU ロック状態へは loc_cpu，または iloc_cpu を発行することにより，非 CPU ロック状態へは unl_cpu，または iunl_cpu を発行することにより遷移します。

8.10　ディスパッチ禁止状態の参照

ディスパッチ禁止状態の参照は，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。

- sns_dsp
  本サービス・コールを発行した際のシステム状態種別（ディスパッチ禁止状態，ディスパッチ許可状態）を獲得します。
  なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：ディスパッチ禁止状態，FALSE：ディスパッチ許可状態）”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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_sub ( 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 */
                .............
        }

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

備考　ディスパッチ禁止状態へは dis_dsp を発行することにより，ディスパッチ許可状態へは ena_dsp を発行することにより遷移します。

8.11　ディスパッチ保留状態の参照

ディスパッチ保留状態の参照は，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。

- sns_dpn
  本サービス・コールを発行した際のシステム状態種別（ディスパッチ保留状態であるか否か）を獲得します。
  なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：ディスパッチ保留状態，FALSE：非ディスパッチ保留状態）”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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_sub ( 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 */
                .............
        }

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

備考　ディスパッチ保留状態とは，dis_dsp，loc_cpu，iloc_cpu　といったサービス・コールを発行して明示的にディスパッチ処理（タスクのスケジューリング処理）の実行が禁止された状態，および，非タスクが処理を実行中の状態を指します。

第 9 章　割り込み管理機能

本章では，RI78V4 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。

9.1　概　　要

RI78V4 における割り込み管理機能では，マスカブル割り込みが発生した際に起動する割り込みハンドラに関連した機能を提供しています。

なお，RI78V4 では，RI78V4 が管理する割り込み処理を“割り込みハンドラ”と呼び，RI78V4 の管理外で動作する割り込み処理とは区別しています。

以下に，割り込みハンドラと RI78V4 の管理外で動作する割り込み処理の差異を示します。

表 9 － 1　割り込みハンドラと割り込み処理の差異

	割り込みハンドラ	RI78V4 管理外の割り込み処理
サービス・コールの発行	可	不可
割り込みの種別	マスカブル割り込み	マスカブル割り込み ソフトウエア割り込み リセット割り込み
割り込みの優先順位	レベル 2，3	レベル 0，1（※）
システム・コンフィギュレーション・ファイルでの定義	DEF_INH で定義する	DEF_INH で定義しない

※　多重割り込みを禁止するアプリケーションの場合は，レベル 2，3 に割り付けることも可能です。

備考 1　割り込み優先度は，対象 CPU の優先順位指定フラグ・レジスタで設定します。
なお，割り込み優先度は，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　RI78V4 では，割り込みを発生させるハードウエア（割り込みコントローラなど）の初期化処理を行いません。したがって，該当初期化処理については，ブート処理，または初期化ルーチンにおいて，ユーザが記述する必要があります。

9.2　割り込みエントリ処理

割り込みエントリ処理は，割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに該当処理（割り込みハンドラ，ブート処理など）への分岐命令を割り付けるためにユーザ・オウン・コーディング部として切り出されたエントリ処理専用ルーチンです。

なお，割り込みハンドラを C 言語で記述（システム・コンフィギュレーション・ファイルの割り込みハンドラ定義（DEF_INH）にて TA_HLNG 属性を指定）する場合，C コンパイラが“割り込み要求名に対応した割り込みエントリ処理”を自動的に出力するため，ユーザが割り込みエントリ処理を記述する必要はありません。

割り込みハンドラをアセンブリ言語で記述（システム・コンフィギュレーション・ファイルの割り込みハンドラ定義（DEF_INH）にて TA_ASM 属性を指定）する場合，割り込みエントリ処理はユーザが記述する必要があります。なお，ブート処理への分岐に関してはアセンブリ言語で記述する必要があります。

9.2.1 割り込みエントリ処理の基本型

割り込みエントリ処理の記述方法は，該当処理（割り込みハンドラ，ブート処理など）が near 領域に割り付けられているのか，または far 領域に割り付けられているのかにより異なります。
以下に，割り込みエントリ処理の記述例を示します。

【 near 領域に該当処理（割り込みハンドラ，ブート処理など）を割り付ける場合 】

```
      .PUBLIC   _func_inthdr
_func_inthdr    .VECTOR  0x002C     ; 割り込みハンドラに制御を移す

      .SECTION  .text, TEXT         ; ベクタ・テーブル・アドレスの設定
_func_inthdr:

          .............             ; 割り込みハンドラの本体処理
```

【 far 領域に該当処理（割り込みハンドラ，ブート処理など）を割り付ける場合 】

```
      .EXTERN _intent_RESET         ; シンボルの外部参照宣言
      .EXTERN _intent_INTTM00       ; シンボルの外部参照宣言

      .SECTION   .vecttable, TEXT   ; ベクタ・テーブルのセクション設定
_intent_RESET   .VECTOR  0x0000     ; ベクタ・テーブル・アドレスの設定
_intent_INTTM00 .VECTOR  0x002C     ; ベクタ・テーブル・アドレスの設定

      .SECTION  .textf, TEXTF       ; ベクタ・テーブルの設定
_intent_RESET:
          BR      !!_boot           ; ブート処理に制御を移す
_intent_INTTM00:
          BR      !!_func_inthdr    ; 割り込みハンドラに制御を移す
```

9.2.2 割り込みエントリ処理内での処理

割り込みエントリ処理は，割り込みが発生した際に RI78V4 を介在させることなく呼び出されるエントリ処理専用ルーチンです。このため，割り込みエントリ処理を記述する際には，以下に示す注意点があります。

- 記述方法
  割り込みエントリ処理は，アセンブリ言語で記述します。
  なお，割り込みエントリ処理を記述する場合は，アセンブラの関数呼び出し規約に従った形式で作成してください。

- スタックの切り替え
  割り込みエントリ処理を実行するうえで切り替えを必要とするスタックは存在しません。したがって，割り込みエントリ処理内でスタックの切り替えに関する記述を行う必要はありません。

- サービス・コールの発行
  RI78V4 では，割り込みエントリ処理内でサービス・コールを発行することを禁止しています。

以下に，割り込みエントリ処理として実行すべき処理の一覧を示します。

- ベクタ・テーブル・アドレスの設定
- 該当処理（割り込みハンドラ，ブート処理など）に制御を移す

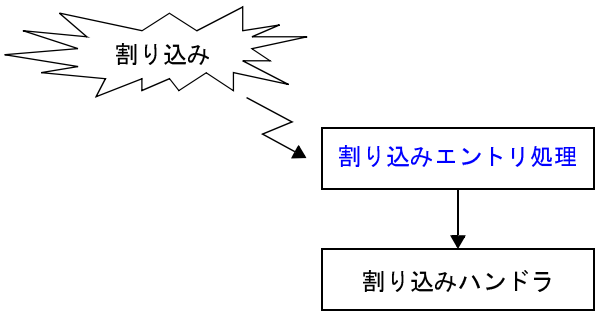
9.3　割り込みハンドラ

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あり，割り込みエントリ処理から呼び出されます。

なお，RI78V4 では，割り込みハンドラを“非タスク（タスクとは独立したもの）”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。

以下に，割り込みの発生から割り込みハンドラ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 9 － 1　処理の流れ（割り込みハンドラ）



9.3.1　割り込みハンドラの登録

割り込みハンドラの登録は，割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに割り込みエントリ処理（割り込みハンドラへの分岐命令）を記述することにより実現されます。

ただし，割り込みエントリ処理の記述方法は，割り込みハンドラが near 領域に割り付けられているのか，または far 領域に割り付けられているのかにより異なります。

なお，割り込みハンドラを C 言語で記述（システム・コンフィギュレーション・ファイルの割り込みハンドラ定義（DEF_INH）にて TA_HLNG 属性を指定）する場合，C コンパイラが“割り込み要求名に対応した割り込みエントリ処理”を自動的に出力するため，ユーザが該当割り込みエントリ処理を記述する必要はありません。

割り込みハンドラをアセンブリ言語で記述（システム・コンフィギュレーション・ファイルの割り込みハンドラ定義（DEF_INH）にて TA_ASM 属性を指定）する場合，割り込みエントリ処理はユーザが記述する必要があります。

9.3.2 割り込みハンドラの基本型

割り込みハンドラを C 言語で記述する場合，引き数を持たない void 型の関数（関数名：任意）として記述します。
また，割り込みハンドラを near 領域に配置するか，far 領域に配置するかにより，記述内容が若干異なります。
割り込みハンドラを C 言語で記述する場合の基本型を示します。

【 C 言語記述の割り込みハンドラを near 領域に配置する場合 】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
__near func_inthdr ( void )
{
        ............                    /* 割り込みハンドラの本体処理 */
        ............

        return;                         /* 割り込みハンドラの終了 */
}
```

備考 1 システム・コンフィギュレーション・ファイルの割り込みハンドラの定義（DEF_INH）において TA_HLNG 属性，および TA_NEAR 属性を指定します。

備考 2 #pragma rtos_interrupt 指令は，kernel_id.h 内に定義されています（CF78V4 が自動的に出力します）。そのため，kernel_id.h は必ずインクルードしてください。

【 C 言語記述の割り込みハンドラを far 領域に配置する場合 】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
__far func_inthdr ( void )
{
        ............                    /* 割り込みハンドラの本体処理 */
        ............

        return;                         /* 割り込みハンドラの終了 */
}
```

備考 1 システム・コンフィギュレーション・ファイルの割り込みハンドラの定義（DEF_INH）において TA_HLNG 属性，および TA_FAR 属性を指定します。

備考 2 #pragma rtos_interrupt 指令は，kernel_id.h 内に定義されています（CF78V4 が自動的に出力します）。そのため，kernel_id.h は必ずインクルードしてください。

割り込みハンドラをアセンブリ言語で記述する場合，引き数を持たない void 型の関数（関数名：任意）として記述します。なお，割り込みハンドラを near 領域に配置する場合と far 領域に配置する場合で，記述内容が若干異なります。

- 割り込みハンドラを near 領域に配置する場合
  割り込みハンドラの開始部分で AX レジスタの退避，および AX レジスタに割り込み要因のベクタ・テーブル・アドレスを設定したのち，システム・スタックへの切り替え処理（関数名：_kernel_int_entry）の呼び出しを，終了部分で終了処理（関数名：_kernel_int_exit）の呼び出しを行います。なお，割り込みハンドラの配置セクションとして，near 配置属性のセクションを指定します。

【 アセンブリ言語記述の割り込みハンドラを near 領域に配置する場合 】

```
$INCLUDE        (kernel.inc)      ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)   ; システム情報ヘッダ・ファイルの定義
     .PUBLIC    _func_inthdr

_func_inthdr    .VECTOR  0x002C   ; 割り込みハンドラに制御を移す

     .SECTION   .text, TEXT
_func_inthdr:                     ; 割り込みハンドラ

     PUSH    AX                   ;AX レジスタの退避
     MOV     AX, #0x002C          ; 割り込み要因のベクタ・テーブル・アドレスを AX レジスタに設定

     CALL    !!__kernel_int_entry ; システム・スタックの切り替え，レジスタの退避

        .............             ; 割り込みハンドラの本体処理

     BR      !!__kernel_int_exit  ; 割り込みハンドラの終了，レジスタの復帰

;割り込みハンドラ記述例終了
```

備考　システム・コンフィギュレーション・ファイルの割り込みハンドラの定義（DEF_INH）において TA_ASM 属性，および TA_NEAR 属性を指定します。

- 割り込みハンドラを far 領域に配置する場合
  割り込みハンドラの開始部分でシステム・スタックへの切り替え処理（関数名：_kernel_int_entry）の呼び出しを，終了部分で終了処理（関数名：_kernel_int_exit）の呼び出しを行います。なお，割り込みハンドラの配置セクションとして，far 配置属性のセクションを指定します。

【 アセンブリ言語記述の割り込みハンドラを far 領域に配置する場合 】

```
$INCLUDE        (kernel.inc)      ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)   ; システム情報ヘッダ・ファイルの定義
     .PUBLIC    _func_inthdr

_func_inthdr    .VECTOR  0x002C

     .SECTION   .textf, TEXTF     ; 割り込みハンドラ
_func_inthdr:
     CALL    !!__kernel_int_entry ; システム・スタックの切り替え，レジスタの退避

        .............             ; 割り込みハンドラの本体処理

     BR      !!__kernel_int_exit  ; 割り込みハンドラの終了，レジスタの復帰

;割り込みハンドラ記述例終了
```

備考 1　システム・コンフィギュレーション・ファイルの割り込みハンドラの定義（DEF_INH）において TA_ASM 属性，および TA_FAR 属性を指定します。

備考 2　割り込みハンドラを far 領域に配置した場合，far 分岐情報が必要となります。つまり，ベクタ・テーブル・アドレスから far 分岐情報へ分岐し，そこから割り込みハンドラへ分岐します。この far 分岐情報は CF78V4 がシステム情報テーブル・ファイルに自動的に出力します。

備考 3　割り込みハンドラの開始部分で AX レジスタの退避，および，AX レジスタに割り込み要因のベクタ・テーブル・アドレスを設定する処理は，CF78V4 がシステム情報テーブル・ファイルに自動的に出力します（上記の far 分岐情報に含まれます）。

9.3.3　割り込みハンドラ内での処理

RI78V4 では，割り込みハンドラを“非タスク”として位置づけています。
また，RI78V4 では，割り込みハンドラに制御を移す際に“独自の前処理”を，割り込みハンドラから制御を戻す際にも“独自の後処理”を実行しています。
このため，割り込みハンドラを記述する際には，以下に示す注意点があります。

- 記述方法
  割り込みハンドラは，「9.3.2　割り込みハンドラの基本型」で示された関数形式で C 言語，またはアセンブリ言語を用いて記述します。

- スタックの切り替え
  C 言語で記述した割り込みハンドラについては，C コンパイラが“システム・スタックへの切り替え処理（関数名：_kernel_int_entry）の呼び出し”を自動的に出力するため，ユーザが該当切り替え処理を記述する必要がありません。
  なお，アセンブリ言語で記述された割り込みハンドラについては，割り込みハンドラの開始部分で AX レジスタの退避，および，AX レジスタに割り込み要因のベクタ・テーブル・アドレスの設定を行ったのち，“システム・スタックへの切り替え処理（関数名：_kernel_int_entry）の呼び出し”を，終了部分で“終了処理（関数名：_kernel_int_exit）の呼び出し”を明示的に行う必要があります。

- レジスタの退避／復帰
  C 言語で記述した割り込みハンドラについては，C コンパイラが“_kernel_int_entry の呼び出し”および“_kernel_int_exit の呼び出し”を自動的に出力するため，ユーザが該当退避／復帰処理を記述する必要がありません。
  なお，アセンブリ言語で記述された割り込みハンドラについては，“レジスタの退避処理（関数名：_kernel_int_entry）の呼び出し”を，終了部分で“レジスタの復帰処理（関数名：_kernel_int_exit）の呼び出し”を明示的に行うことにより，汎用レジスタ（AX, BC, DE, HL），および，ES, CS といったレジスタの退避／復帰が該当関数内で実行されます。

  備考　PSW，および，PC については，CPU によって自動的に退避／復帰されます。

- 割り込み状態
  RI78V4 では，割り込みハンドラに制御を移す際に以下の状態となります。
  したがって，割り込みハンドラに制御が移った後，該当レベルよりも高い優先順位の割り込みが発生した場合，多重割り込みが受け付けられます。

  - マスカブル割り込みの受け付けが許可された状態
    IE = 1

  - 以下の優先順位の割り込みが禁止された状態
    レベル 2 の割り込みハンドラ処理中の場合：ISP1 = 0，ISP0 = 1
    レベル 3 の割り込みハンドラ処理中の場合：ISP1 = 1，ISP0 = 0
    備考　レベル 0，1 は割り込みハンドラとして定義することができません。

  備考　割り込みハンドラ内でマスカブル割り込みの受け付けを禁止（IE = 0）した場合も，割り込みハンドラからの復帰後は，マスカブル割り込みの受け付けが許可状態（IE = 1）となります。

- サービス・コールの発行
  RI78V4 では，割り込みハンドラを“非タスク”として位置づけています。
  このため，割り込みハンドラ内で発行可能なサービス・コールは，“非タスクから発行可能なサービス・コール”に限られます。

  備考 1　サービス・コールの発行有効範囲についての詳細は，表 12 － 8 ～表 12 － 17 を参照してください。

  備考 2　RI78V4 では，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，割り込みハンドラ内の処理が完了するまでの間に“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（ichg_pri, isig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，割り込みハンドラからの復帰命令（return 命令の発行など）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。

9.4　割り込み許可・禁止の制御

9.4.1　RI78V4 管理下の割り込みレベル

マイクロコントローラで管理する割り込みレベルは“レベル 0～レベル 3”の 4 レベルです。そのうち，RI78V4 では，割り込みからサービス・コールを発行できる割り込みレベルを“レベル 2 とレベル 3”に固定し，“RI78V4 管理下の割り込みレベル”としています。

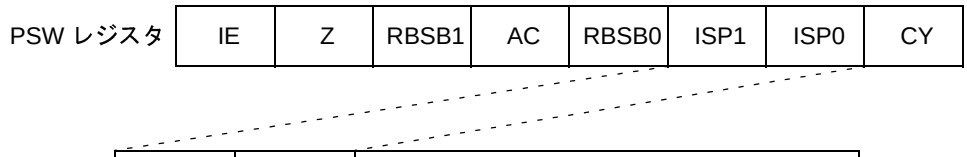
- レベル 2 とレベル 3 の割り込みは RI78V4 管理下の割り込みレベルです。
  レベル 2 とレベル 3 からは，サービス・コールを発行することができます。RI78V4 管理下の割り込みである“割り込みハンドラ”（タイマ割り込みも割り込みハンドラに含みます）は，レベル 2 とレベル 3 に設定する必要があります。

- レベル 0 とレベル 1 の割り込みは RI78V4 管理外の割り込みレベルです。
  レベル 0 とレベル 1 からは，サービス・コールを発行することができません。レベル 0 とレベル 1 からサービス・コールを発行した場合は動作を保証することができません。RI78V4 管理外の割り込みである“割り込み処理”は，レベル 0 とレベル 1 に設定する必要があります。ただし，後述の多重割り込みを禁止するユーザ・アプリケーションの場合のみ，例外的にレベル 2 とレベル 3 に設定することが可能です。

9.4.2　RI78V4 内での割り込み許可・禁止の制御

RI78V4 では，割り込みの許可・禁止は PSW レジスタの“ISP1”，“ISP0”ビットを使用して行われます。RI78V4 内で割り込みを禁止する場合は“ISP1=0”，“ISP0=1”が設定されます。また，RI78V4 内で割り込みを許可する場合は“ISP1=1”，“ISP0=1”が設定されます。

図 9－2　PSW レジスタの ISP1，ISP0 ビット



ISP1	ISP0	現在処理中の割り込み優先順位
0	0	レベル 0 の割り込み許可 （レベル 1，または 0 の割り込み処理中）
0	1	レベル 0，レベル 1 の割り込み許可 （レベル 2 の割り込み処理中）
1	0	レベル 0，レベル 1，レベル 2 の割り込み許可 （レベル 3 の割り込み処理中）
1	1	すべての割り込み許可 （割り込み受け付け待ち）

RI78V4 内では PSW レジスタの“IE”ビットの値はサービス・コールや RI78V4 用関数の発行元の値が引き継がれ，EI 命令，DI 命令による“IE”の操作は行われません。ただし，例外的に RI78V4 内で EI 命令，DI 命令が使用される箇所があります。

- 割り込み禁止指定のタスク起動直前には DI 命令が使用され“IE=0”となります。
- 割り込み許可指定のタスク起動直前には EI 命令が使用され“IE=1”となります。
- アイドル・ルーチンの開始直前には EI 命令が使用され“IE=1”となります。
- 割り込みハンドラの開始処理である __kernel_int_entry 関数内の最初は“IE=1”となります。

9.4.3　ユーザ処理上での割り込み許可・禁止の制御

ユーザ・アプリケーション上での割り込み操作には，__EI 関数（または EI 命令），__DI 関数（または DI 命令）を使用します。タスクなどのユーザ処理上では，DI 関数を使用するとすべてのマスカブル割り込みの受け付けが禁止され，EI 関数を使用すると“ISP1”，“ISP0”の状態に従ったマスカブル割り込みの受け付けが許可される動作となります。

ユーザ処理開始時の割り込み許可・禁止状態は RI78V4 によって設定されます。以下に，その状態の一覧を示します。

表 9 － 2　処理開始時の割り込み許可・禁止状態

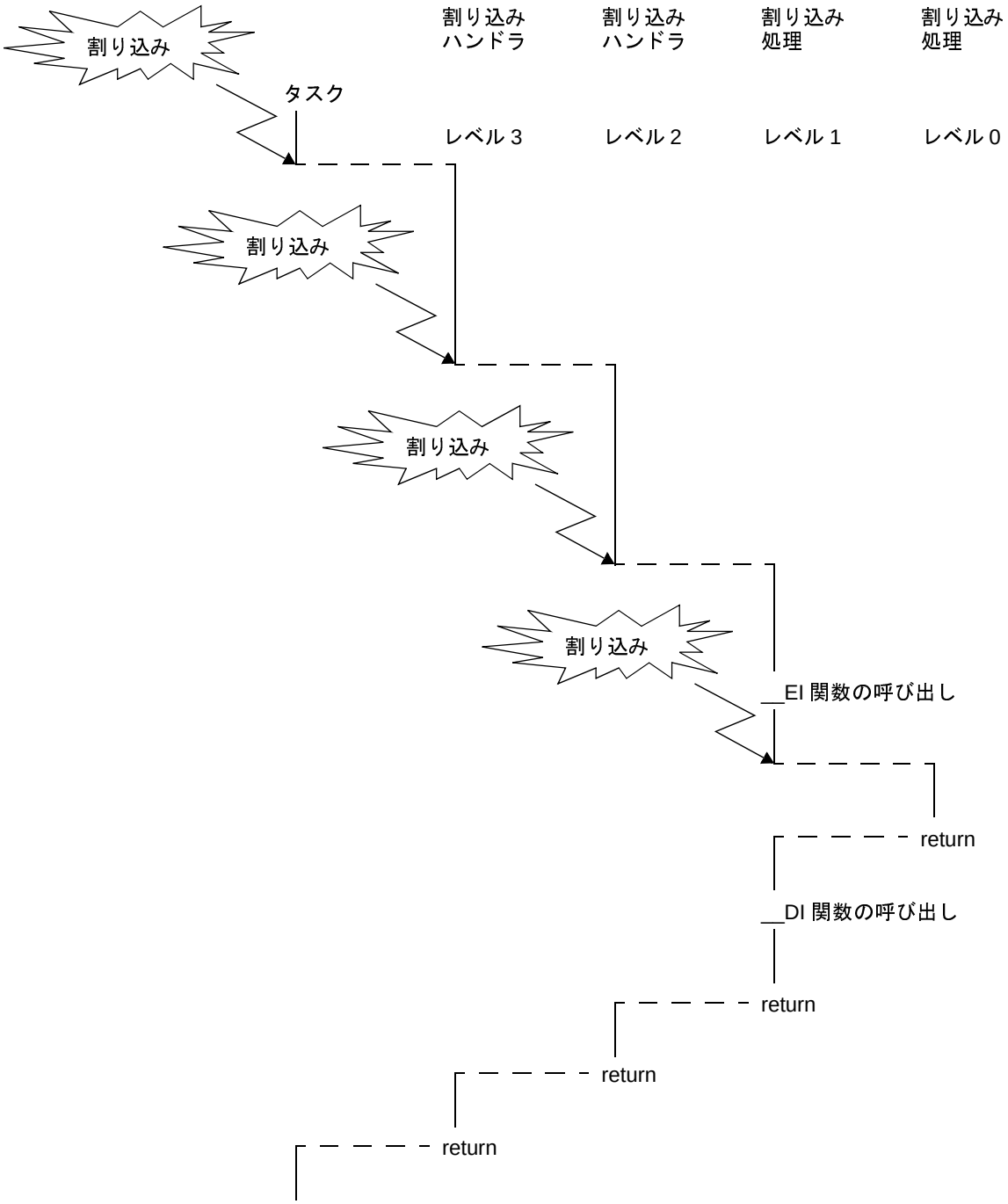
開始する処理		IE	ISP1	ISP0	開始時の割り込み許可・禁止状態
初期化ルーチン		0	1	1	割り込み禁止（処理上で許可をした場合は動作保証外）
アイドル・ルーチン		1	1	1	割り込み許可，全割り込みレベル受け付け
タスク	割り込み許可指定時	1	1	1	割り込み許可，全割り込みレベル受け付け
	割り込み禁止指定時	0	1	1	割り込み禁止（許可された場合は全割り込みレベル受け付け）
周期ハンドラ	レベル2割り込み発生時	1	0	1	割り込み許可，割り込みレベル 0，1 受け付け
	レベル3割り込み発生時	1	1	0	割り込み許可，割り込みレベル 0，1，2 受け付け
割り込みハンドラ	レベル2割り込み発生時	1	0	1	割り込み許可，割り込みレベル 0，1 受け付け
	レベル3割り込み発生時	1	1	0	割り込み許可，割り込みレベル 0，1，2 受け付け
割り込み処理	レベル0割り込み発生時	0	0	0	割り込み禁止（許可された場合は割り込みレベル 0 受け付け）
	レベル1割り込み発生時	0	0	0	割り込み禁止（許可された場合は割り込みレベル 0 受け付け）
	レベル2割り込み発生時	0	0	1	割り込み禁止（許可された場合は割り込みレベル 0，1 受け付け）
	レベル3割り込み発生時	0	1	0	割り込み禁止（許可された場合は割り込みレベル 0，1，2 受け付け）

なお，タスクはそれぞれ固有に“IE”の状態を保持します。中断されたタスクが再開される際には，中断前の“IE”の状態に戻ります。

9.5 多重割り込み

RI78V4 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。
以下に，多重割り込みが発生した際の処理の流れを示します。

図 9 － 3 多重割り込み



割り込みハンドラに制御が移る際は，マスカブル割り込みの受け付けが許可された状態“IE = 1”となります。このため，割り込みハンドラ上では基本的に多重割り込みが受け付けられます。タイマ割り込みやそこから呼び出される周期ハンドラも同様に，多重割り込みが受け付けられます。

割り込み処理に制御が移る際は，マスカブル割り込みの受け付けが禁止された状態“IE = 0”となります（RI78V4 が介在しないため，マイクロコントローラの動作に従った動作となります）。このため，割り込み処理上では基本的に多重割り込みが受け付けられません。多重割り込みの受け付けを許可したい場合は，割り込み処理上で __EI 関数を呼び出す必要があります。なお，割り込み処理上から割り込みハンドラを多重に受け付けることは禁止しており，割り込みハンドラを多重に受け付けた場合の動作は保証外となります。

多重割り込みを許可するユーザ・アプリケーションの場合は，以下のように割り込みハンドラ／割り込み処理の割り込みレベルを設定する必要があります。

表９－３　設定可能な割り込みレベル（多重割り込みを許可するユーザ・アプリケーションの場合）

	割り込みハンドラ	割り込み処理
割り込みレベル 0	不可	可
割り込みレベル 1	不可	可
割り込みレベル 2	可	不可
割り込みレベル 3	可	不可

多重割り込みを禁止するユーザ・アプリケーションの場合は，以下のいずれかのパターンに割り込みハンドラ／割り込み処理の割り込みレベルを設定する必要があります。

パターン１：全割り込みハンドラと全割り込み処理を割り込みレベル 2 に設定する。
パターン２：全割り込みハンドラと全割り込み処理を割り込みレベル 3 に設定する。
パターン３：全割り込みハンドラを割り込みレベル 2 に設定し，全割り込み処理を割り込みレベル 2，3 のいずれかに設定する。割り込みレベル 3 の割り込み処理中は割り込み禁止（IE = 0）。

表９－４　設定可能な割り込みレベル（多重割り込みを禁止するユーザ・アプリケーションの場合）

	パターン１		パターン２		パターン３	
	割り込みハンドラ	割り込み処理	割り込みハンドラ	割り込み処理	割り込みハンドラ	割り込み処理
割り込みレベル 0	不可	不可	不可	不可	不可	不可
割り込みレベル 1	不可	不可	不可	不可	不可	不可
割り込みレベル 2	可	可	不可	不可	可	可
割り込みレベル 3	不可	不可	可	可	不可	可（※）

※この割り込み処理中は割り込み禁止（IE = 0）

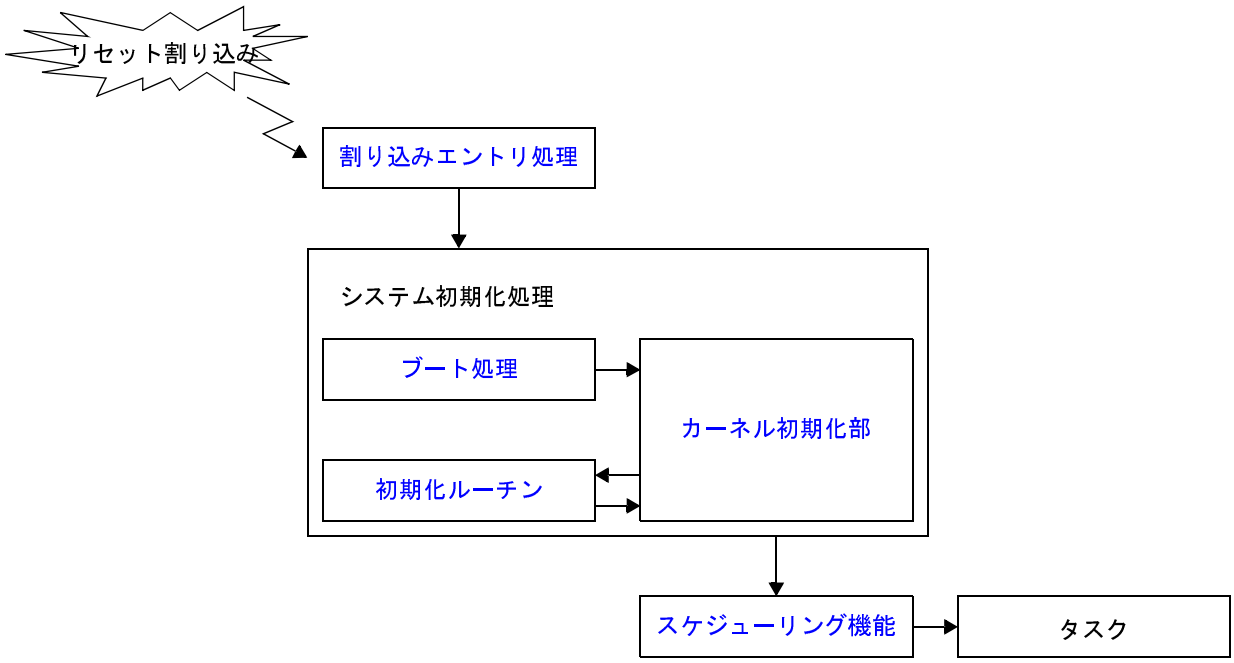
第 10 章　システム構成管理機能

本章では，RI78V4 が提供しているシステム構成管理機能に規程ついて解説しています。

10.1　概　　要

RI78V4 におけるシステム構成管理機能では，リセット割り込みの発生からタスクに制御を移すまでに必要となるシステム初期化処理，および，バージョン情報の参照処理を提供しています。
以下に，リセット割り込みの発生から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 10 － 1　処理の流れ（システム初期化処理）



10.2　ブート処理

ブート処理は，RI78V4 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あり，リセット割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに割り付けられた割り込みエントリ処理から呼び出されます。

10.2.1　ブート処理の登録

ブート処理の登録は，リセット割り込みが発生した際に CPU が強制的に制御を移すベクタ・テーブル・アドレスに割り込みエントリ処理（ブート処理への分岐命令）を記述することにより実現されます。
ただし，割り込みエントリ処理の記述方法は，ブート処理が near 領域に割り付けられているのか，または far 領域に割り付けられているのかにより異なります。
以下に，割り込みエントリ処理の記述例を示します。

【 near 領域にブート処理を割り付ける場合 】

```
      .PUBLIC    _boot         ; ベクタ・テーブルのセクション設定
_boot    .VECTOR  0x0000      ; ブート処理に制御を移す
```

【 far 領域にブート処理を割り付ける場合 】

```
      .EXTERN    _intent_RESET      ; シンボルの外部参照宣言

      .SECTION   .vecttable, TEXT   ; ベクタ・テーブルのセクション設定
_intent_RESET    .VECTOR  0x0000    ; ベクタ・テーブル・アドレスの設定

      .SECTION  .textf, TEXTF       ; ベクタ・テーブルの設定
_intent_RESET:
         BR      !!_boot            ; ブート処理に制御を移す
```

10.2.2　ブート処理の基本型

ブート処理を記述する場合，引き数，および，戻り値を持たない関数（関数名：任意）として記述します。
以下に，ブート処理の基本型を示します。

```
        .PUBLIC  _boot
        .EXTERN  __kernel_start, _hdwinit, __init_ri_stackarea, _reset

        .SECTION .stack_bss, BSS        ; ブート内でスタックを使用する場合は領域確保
_stackend:
        .DS      0x100
_stacktop:

_boot   .VECTOR  0x0000

        .SECTION .text, TEXT
_boot:
        SEL      RB0                    ; レジスタ・バンクの設定

        MOVW     SP, #LOWW(_stacktop)   ; スタック・ポインタ SP の設定

        CALL     !!_reset

; カーネル初期化情報のクリア
        MOVW     HL, #LOWW(STARTOF(.kernel_data_init))
        MOVW     AX, #LOWW(STARTOF(.kernel_data_init) + SIZEOF(.kernel_data_init))
        BR       $L2_KERNEL_DATA
L1_KERNEL_DATA:
        MOV      [HL+0], #0
        INCW     HL
L2_KERNEL_DATA:
        CMPW     AX, HL
        BNZ      $L1_KERNEL_DATA


        CALL     !!__init_ri_stackarea  ;RAM 領域のクリア（別関数で実施）

        BR       !!__kernel_start       ; カーネル初期化部に制御を移す

        CLRW     AX
_exit:
        BR       $exit
```

10.2.3　ブート処理内での処理

ブート処理は，割り込みエントリ処理から RI78V4 を介在させることなく呼び出される初期化処理専用ルーチンです。このため，ブート処理を記述する際には，以下に示す注意点があります。

- 記述方法
  ブート処理は，アセンブリ言語で記述します。

- スタックの切り替え
  ブート処理に制御が移った時点では，“スタック・ポインタ SP の設定”が実行されていません。
  したがって，ブート処理専用スタックを使用する場合は，ブート処理の開始部分で“スタック・ポインタ SP の設定”を記述する必要があります。

- 割り込み状態
  ブート処理に制御が移った時点では，“カーネル初期化部”が実行されていません。
  したがって，該当処理が完了する以前に割り込みが発生した際には，システムが暴走する可能性があります。そこで，ブート処理では，プログラム・ステータス・ワード PSW の割り込み許可フラグ IE を操作して，マスカブル割り込みの受け付けを明示的に禁止してください。

- レジスタ・バンクの設定
  RI78V4 では，ブート処理内で _kernel_start を呼び出す以前に設定されたレジスタ・バンクから他のレジスタ・バンクへと切り替えることを禁止（RI78V4 管理外の割り込み処理を除く）しています。

- サービス・コールの発行
  RI78V4 では，ブート処理内でサービス・コールを発行することを禁止しています。

以下に，ブート処理として実行すべき処理の一覧を示します。

- スタック・ポインタ SP の設定
- 割り込み許可フラグ IE の設定
- 内部ユニット，周辺コントローラの初期化
- RAM 領域の初期化（初期値なしメモリ領域の初期化，初期化データのコピー）
- カーネル初期化部（関数名：_kernel_start）に制御を移す

備考　“スタック・ポインタ SP の設定”については，ブート処理内でブート処理専用スタックを使用する場合に限り必要となります。

備考

10.2.4　システム依存情報

システム依存情報は，RI78V4 が処理を実行するうえで必要となる各種情報をユーザ・オウン・コーディング部として切り出したヘッダ・ファイル（ファイル名：usrown.h）です。

- システム依存情報の基本型
  システム依存情報を記述する場合，規定されたファイル名（usrown.h），規定されたマクロ名（KERNEL_USR_TMCNTREG，KERNEL_USR_TMCMPREG）を用いて記述します。
  以下に，システム依存情報を C 言語で記述する場合の基本型を示します。

```
#include    <kernel_id.h>                    /* システム情報ヘッダ・ファイルの定義 */

#define     KERNEL_USR_TMCNTREG 0x0180       /*I/O アドレス */
#define     KERNEL_USR_TMCMPREG 0xff18       /*I/O アドレス */
```

以下に，システム依存情報として定義すべき情報の一覧を示します。

- システム情報ヘッダ・ファイルの定義
  システム・コンフィギュレーション・ファイルに対してコンフィギュレータを実行することにより出力されるシステム情報ヘッダ・ファイルのインクルード

  備考　本情報の記述は，プロパティ パネル → ［タスク・アナライザ］タブ → ［トレース］カテゴリ → ［トレース・モードの選択］で“ソフトウエア・トレース・モードで，長時間統計を取得”を選択した場合に限り必要となります。

- 基本クロック用タイマ情報
  基本クロック用タイマのカウンタ・レジスタの I/O アドレス，および基本クロック用タイマのコンペア・レジスタの I/O アドレスをマクロ定義

  備考　本情報の記述は，プロパティ パネル → ［タスク・アナライザ］タブ → ［トレース］カテゴリ → ［トレース・モードの選択］で“ソフトウエア・トレース・モードで，トレース・チャートを取得”，または“ソフトウエア・トレース・モードで，長時間統計を取得”を選択した場合に限り必要となります。

10.3　初期化ルーチン

初期化ルーチンは，ユーザの実行環境に依存したハードウエア（周辺コントローラなど）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あり，カーネル初期化部から呼び出されます。

10.3.1　初期化ルーチンの登録

RI78V4 では，初期化ルーチンの登録方法を“カーネル初期化部において静的に登録する”に限定しています。
したがって，RI78V4 では，初期化ルーチンを処理プログラムからサービス・コールを発行するなどして動的に登録することはできません。

- 静的な登録
  初期化ルーチンの静的な登録は，規定された関数名 init_handler で初期化ルーチンを記述することにより実現されます。
  RI78V4 では，カーネル初期化部において，該当シンボル情報をもとに初期化ルーチンの登録処理を実行し，管理対象とします。

10.3.2　初期化ルーチンの登録解除

RI78V4 では，カーネル初期化部において静的に登録された初期化ルーチンを処理プログラムからサービス・コールを発行するなどして動的に登録解除することはできません。

10.3.3　初期化ルーチンの基本型

初期化ルーチンを記述する場合，引き数を持たない void 型の関数（関数名：init_handler）として記述します。
以下に，初期化ルーチンの基本型を示します。

【C 言語で記述する場合 】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
init_handler ( void )
{
        .............                   /* 初期化ルーチンの本体処理 */
        .............

        return;                         /* 初期化ルーチンの終了 */
}
```

【アセンブリ言語で記述する場合】

```
$INCLUDE   (kernel.inc)             ; 標準ヘッダ・ファイルの定義
$INCLUDE   (kernel_id.inc)          ; システム情報ヘッダ・ファイルの定義

        .PUBLIC _init_handler

        .SECTION .textf, TEXTF
_init_handler:
        .............               ; 初期化ルーチンの本体処理
        .............

        RET                         ; 初期化ルーチンの終了
```

10.3.4　初期化ルーチン内での処理

RI78V4 では，初期化ルーチンに制御を移す際に“独自の前処理”を，初期化ルーチンから制御を戻す際にも“独自の後処理”を実行しています。
このため，初期化ルーチンを記述する際には，以下に示す注意点があります。

- 記述方法
  初期化ルーチンは，「10.3.3　初期化ルーチンの基本型」で示された関数形式で C 言語，またはアセンブリ言語を用いて記述します。

- スタックの切り替え
  RI78V4 では，初期化ルーチンに制御を移す際に“システム・スタックへの切り替え処理”を，初期化ルーチンから制御を戻す際に“切り替え先のカーネル初期化部用スタックへの切り替え処理”を実行しています。
  したがって，ユーザは，初期化ルーチン内でスタックの切り替えに関する処理を記述する必要はありません。

- 割り込み状態
  RI78V4 では，初期化ルーチンに制御を移す際に“マスカブル割り込みの受け付けが禁止された状態”としています。
  なお，初期化ルーチンに制御が移った時点では，カーネル初期化部の処理が完了していません。
  したがって，初期化ルーチン内で明示的にマスカブル割り込みの受付けを許可した際には，システムが暴走する可能性があります。そこで，RI78V4 では，初期化ルーチン内でマスカブル割り込みの受け付けを許可することを禁止しています。

- サービス・コールの発行
  RI78V4 では，初期化ルーチン処理内でサービス・コールを発行することを禁止しています。

以下に，初期化ルーチンとして実行すべき処理の一覧を示します。

- 内部ユニット，周辺コントローラの初期化
- RAM 領域の初期化（初期値なしメモリ領域の初期化，初期化データのコピー）
- カーネル初期化部に制御を戻す

10.4　カーネル初期化部

カーネル初期化部は，RI78V4 が処理を実行するうえで必要となる最低限のソフトウエアを初期化するために用意された初期化処理専用ルーチンであり，ブート処理から呼び出されます。
なお，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。

- メモリ領域の確保
- 管理オブジェクトの生成／登録
- 初期化ルーチンの呼び出し
- スケジューラに制御を移す

備考　カーネル初期化部は，RI78V4 が提供する機能の一部です。したがって，ユーザは，カーネル初期化部の処理内容を記述する必要はありません。

10.5　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
  RI78V4 のバージョン情報（メーカ・コードなど）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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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 */
        UH      spver;                  /* 変数の宣言 */
        UH      prver;                  /* 変数の宣言 */
        UH      prno[4];                /* 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;            /* 識別番号の獲得 */
        spver = pk_rver.spver;          /* バージョン番号の獲得 */
        prver = pk_rver.prver;          /* バージョン番号の獲得 */
        prno[0] = pk_rver.prno[0];      /* 版数の獲得 */
        prno[1] = pk_rver.prno[1];      /* メモリ・モデルの獲得 */

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

備考　　バージョン情報 T_RVER についての詳細は，「12.5.9　バージョン情報」を参照してください。

第 11 章　スケジューリング機能

本章では，RI78V4 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。

11.1　概　　要

RI78V4 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遷移状態を観察することにより，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。

11.2　駆動方式

RI78V4 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

  RI78V4 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，ディスパッチ処理（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
  - 時間管理機能を実現する際に利用しているクロック割り込みの発生

11.3　スケジューリング方式

RI78V4 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，RI78V4 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

  RI78V4 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）へと遷移している全タスクの中から“最も高い優先度を持つタスク”を選び出し，CPU の利用権を与えます。

  備考　　RI78V4 における優先度は，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。

- FCFS 方式

  RI78V4 では，同一の優先度を“複数のタスク”に対して定義することが可能です。このため，優先度方式におけるタスクを選び出す基準である“最も高い優先度を持つタスク”が複数存在する場合があります。
  そこで，RI78V4 では，このような場合には，FCFS 方式（First Come First Service 方式）によるディスパッチ処理（タスクのスケジューリング処理）を実行し，実行可能な状態（READY 状態）へと遷移してから“最も時間が経過しているタスク”を選び出し，CPU の利用権を与えます。

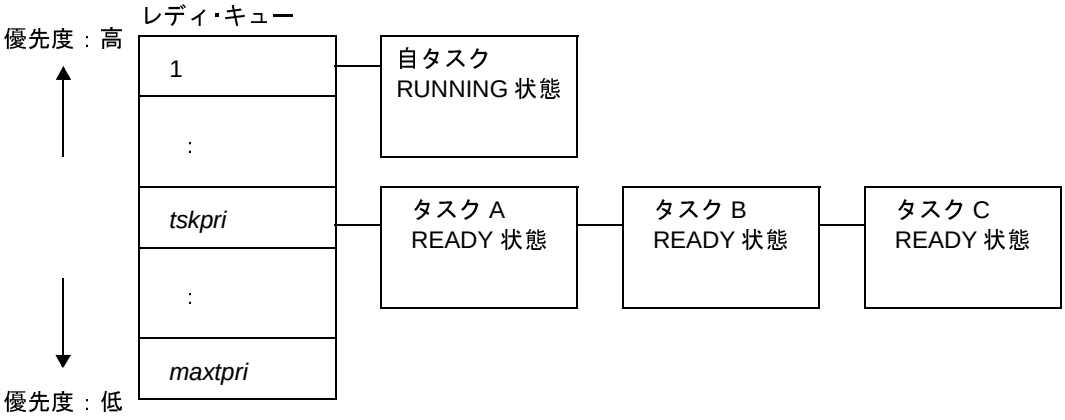
11.4　レディ・キュー

RI78V4 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”を利用しています。

なお，RI78V4 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“タスクの検出処理”を実行し，キューイングされているタスクを検出した際には，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，タスク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 11 － 1　スケジューリング方式（優先度方式，FCFS 方式）の実現



11.4.1　レディ・キューの生成

RI78V4 では，レディ・キューの生成方法を“カーネル初期化部において静的に生成する”に限定しています。

したがって，RI78V4 では，レディ・キューを処理プログラムからサービス・コールを発行するなどして動的に生成することはできません。

- 静的な生成
  レディ・キューの静的な生成は，システム・コンフィギュレーション・ファイルに優先度情報を定義することにより実現されます。
  RI78V4 では，カーネル初期化部において，情報ファイルに格納されているデータをもとにレディ・キューの生成処理を実行し，管理対象とします。

11.4.2　レディ・キューの削除

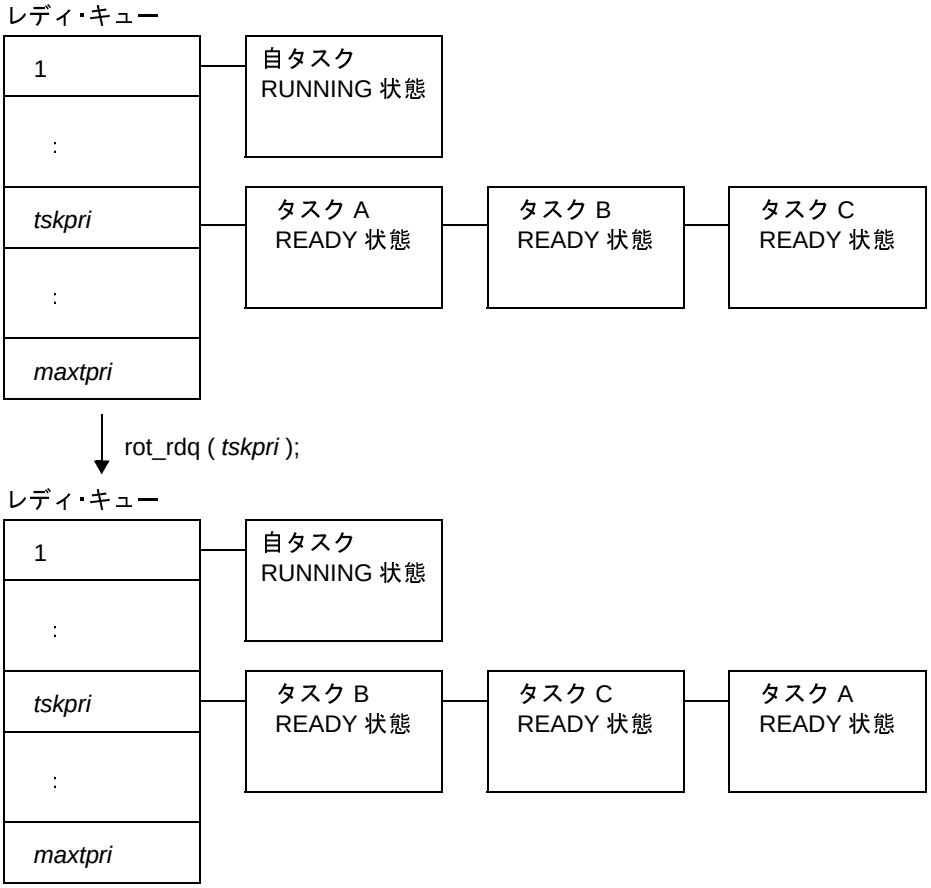
RI78V4 では，カーネル初期化部において静的に生成されたレディ・キューを処理プログラムからサービス・コールを発行するなどして動的に削除することはできません。

11.4.3　レディ・キューの回転

RI78V4 では，処理プログラムからタスクのキューイング順序を変更し，“タスクの実行順序” を明示的に入れ替える機能を提供しています。

以下に，タスクのキューイング順序を変更した際の状態変化を示します。

図 11 － 2　レディ・キューの回転



なお，レディ・キューの回転は，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。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_cychdr ( void )
{
        PRI     tskpri = 8;             /* 変数の宣言，初期化 */

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

        irot_rdq ( tskpri );            /* レディ・キューの回転 */

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

        return;                         /* 周期ハンドラの終了 */
}
```

備考１　本サービス・コールでは，回転要求のキューイングが行われません。このため，該当優先度に対応したレディ・キューにタスクが 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。

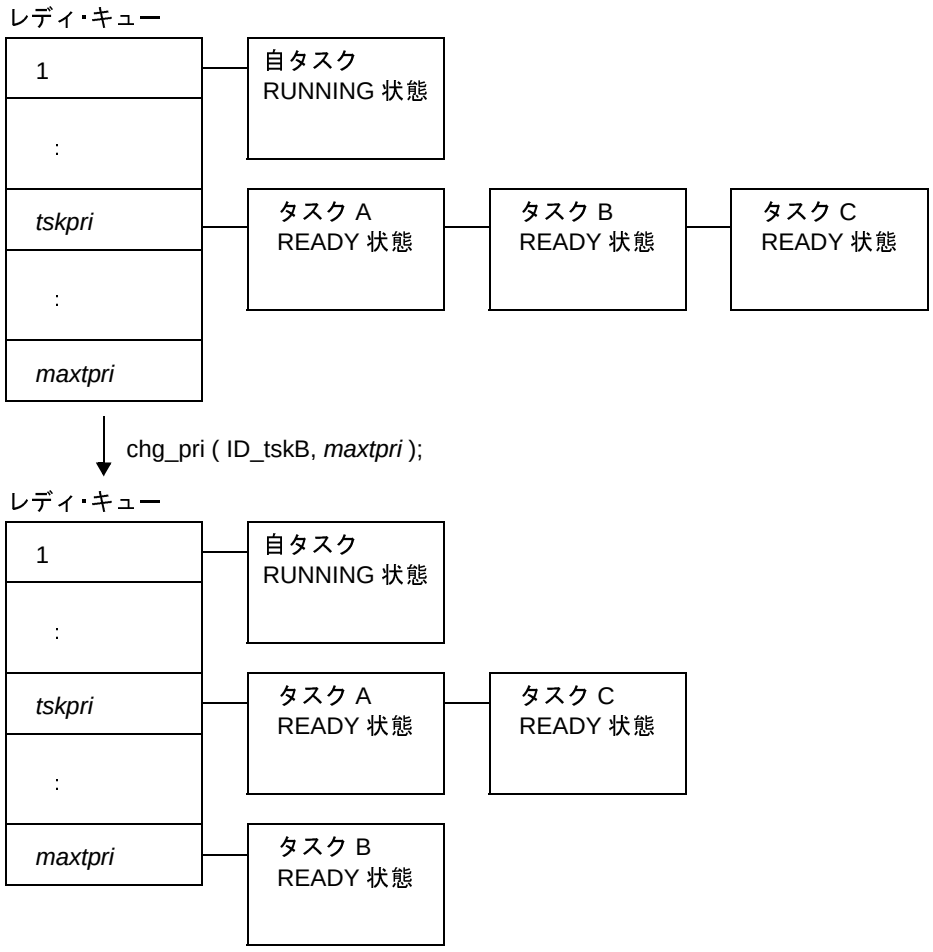
備考２　本サービス・コールを周期ハンドラなどから一定周期で発行することにより，ラウンドロビン・スケジューリングを実現することができます。

11.4.4　優先度の変更

RI78V4 では，処理プログラムからタスクの優先度を変更し，“タスクの実行順序”を明示的に入れ替える機能を提供しています。

以下に，タスクの優先度を変更した際の状態変化を示します。

図 11 － 3　優先度の変更



なお，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
  パラメータ *tskid* で指定されたタスクの優先度（現在優先度）をパラメータ *tskpri* で指定された値に変更します。
  以下に，本サービス・コールの記述例を示します。

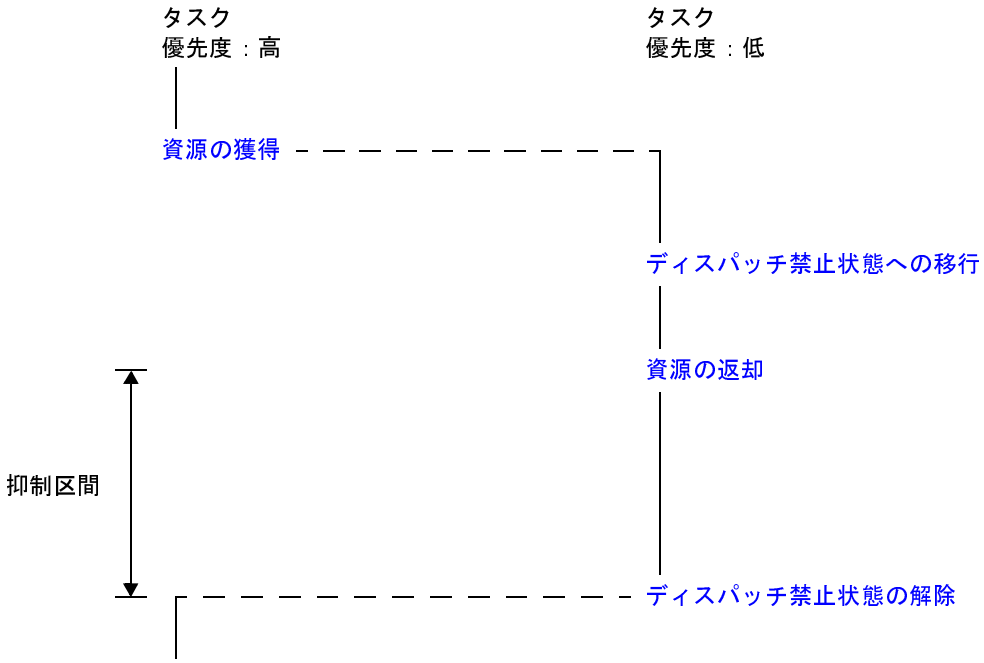
```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

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

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

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

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

備考　本サービス・コールを発行した際，対象タスクが RUNNING 状態，または READY 状態であった場合には，優先度の変更処理を実行したのち，パラメータ *tskpri* で指定された優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

11.5　スケジューリングの抑制

RI78V4 では，処理プログラムからシステム状態を操作し，ディスパッチ処理（タスクのスケジューリング処理）を明示的に禁止することにより，スケジューラの起動を抑制する機能を提供しています。
以下に，スケジューリングの抑制機能を利用した際の処理の流れを示します。

図 11 － 4　スケジューリングの抑制機能



11.5.1　ディスパッチ禁止状態への移行

ディスパッチ禁止状態への移行は，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。

- dis_dsp
  システム状態種別をディスパッチ禁止状態へと変更します。
  これにより，本サービス・コールの発行から ena_dsp が発行されるまでの間，ディスパッチ処理（タスクのスケジューリング処理）が禁止されます。
  なお，RI78V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，ena_dsp 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。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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　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，システム状態種別がディスパッチ禁止状態であった場合には，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。

11.5.2　ディスパッチ禁止状態の解除

ディスパッチ禁止状態の解除は，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。

- ena_dsp
  システム状態種別をディスパッチ許可状態へと変更します。
  これにより，dis_dsp の発行により禁止されていたディスパッチ処理（タスクのスケジューリング処理）が許可されます。
  なお，RI78V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，本サービス・コール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。
  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>           /* システム情報ヘッダ・ファイルの定義 */

void
func_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 */

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

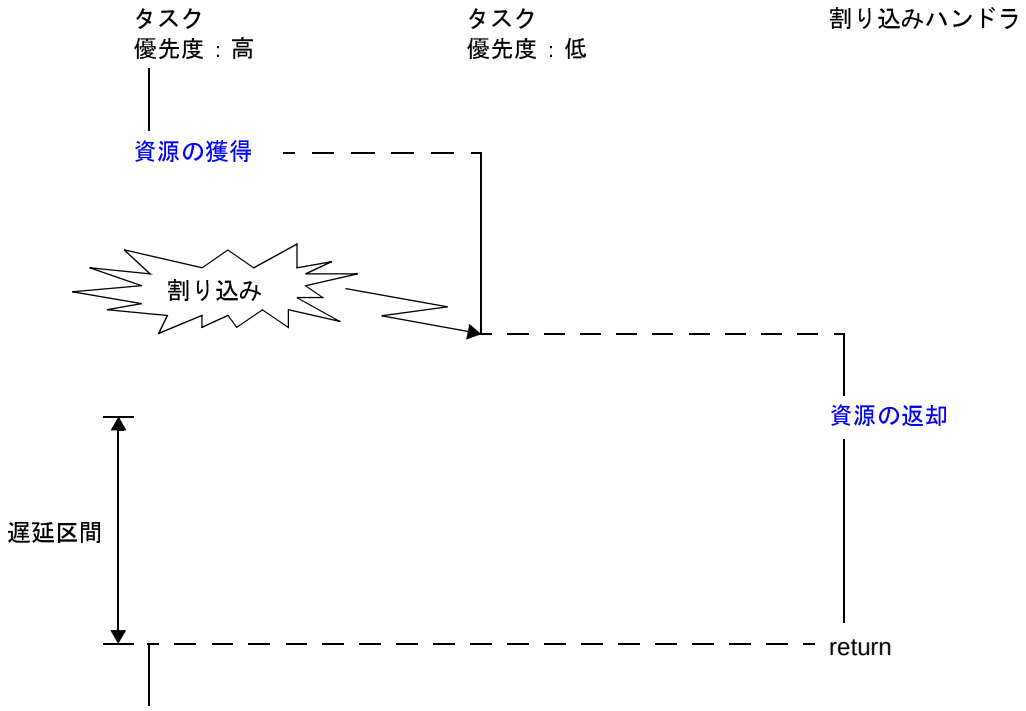
備考　本サービス・コールでは，許可要求のキューイングが行われません。このため，システム状態種別がディスパッチ許可状態であった場合には，何も処理は行わず，エラーとしても扱いません。

11.6　スケジューリングの遅延

RI78V4 では，非タスク（周期ハンドラ，割り込みハンドラなど）内の処理を高速に終了させる目的から，非タスク内の処理が完了するまでの間に“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（ichg_pri，isig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，非タスクからの復帰命令（return 命令の発行など）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。

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処理の流れを示します。

図 11 － 5　スケジューリングの遅延



11.7　アイドル・ルーチン

アイドル・ルーチンは，CPU が提供しているスタンバイ機能を有効活用（低消費電力システムの実現）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アイドル処理専用ルーチンであり，RI78V4 のスケジューリング対象となるタスク（RUNNING 状態，または READY 状態のタスク）がシステム内に 1 つも存在しなくなった際にスケジューラから呼び出されます。

11.7.1　アイドル・ルーチンの登録

RI78V4 では，アイドル・ルーチンの登録方法を“カーネル初期化部において静的に登録する”に限定しています。
したがって，RI78V4 では，アイドル・ルーチンを処理プログラムからサービス・コールを発行するなどして動的に登録することはできません。

- 静的な登録
  アイドル・ルーチンの静的な登録は，規定された関数名 idle_handler でアイドル・ルーチンを記述することにより実現されます。
  RI78V4 では，カーネル初期化部において，該当シンボル情報をもとにアイドル・ルーチンの登録処理を実行し，管理対象とします。

11.7.2　アイドル・ルーチンの登録解除

RI78V4 では，カーネル初期化部において静的に登録されたアイドル・ルーチンを処理プログラムからサービス・コールを発行するなどして動的に登録解除することはできません。

11.7.3　アイドル・ルーチンの基本型

アイドル・ルーチンを記述する場合，引き数を持たない void 型の関数（関数名：idle_handler）として記述します。
以下に，アイドル・ルーチンの基本型を示します。

【C 言語で記述する場合】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
idle_handler ( void )
{
        ............                    /* アイドル・ルーチンの本体処理 */
        ............

        return;                         /* アイドル・ルーチンの終了 */
}
```

【アセンブリ言語で記述する場合】

```
$INCLUDE        (kernel.inc)            ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)         ; システム情報ヘッダ・ファイルの定義

        .PUBLIC  _idle_handler

        .SECTION .textf, TEXTF
_idle_handler:
        ............                    ; アイドル・ルーチンの本体処理
        ............

        RET                             ; アイドル・ルーチンの終了
```

11.7.4　アイドル・ルーチン内での処理

RI78V4 では，アイドル・ルーチンを“非タスク（タスクとは独立したもの）”として位置づけています。
また，RI78V4 では，アイドル・ルーチンに制御を移す際に“独自の前処理”を，アイドル・ルーチンから制御を戻す際にも“独自の後処理”を実行しています。
このため，アイドル・ルーチンを記述する際には，以下に示す注意点があります。

- 記述方法
  アイドル・ルーチンは，「11.7.3　アイドル・ルーチンの基本型」で示された関数形式で C 言語，またはアセンブリ言語を用いて記述します。

- スタックの切り替え
  RI78V4 では，アイドル・ルーチンに制御を移す際に“システム・スタックへの切り替え処理”を，アイドル・ルーチンから制御を戻す際に“切り替え先の処理プログラム用スタック（システム・スタック，またはタスク・スタック）への切り替え処理”を実行しています。
  したがって，ユーザは，アイドル・ルーチン内でスタックの切り替えに関する処理を記述する必要はありません。

- 割り込み状態
  RI78V4 では，アイドル・ルーチンに制御を移す際に“マスカブル割り込みの受け付けが許可された状態”としています。
  したがって，ユーザは，アイドル・ルーチン内でマスカブル割り込みの受け付けに関する処理を記述する必要はありません。

- サービス・コールの発行
  RI78V4 では，アイドル・ルーチン内でサービス・コールを発行することを禁止しています。

以下に，アイドル・ルーチンとして実行すべき処理の一覧を示します。

- CPU が提供しているスタンバイ機能の有効活用

第 12 章　サービス・コール

本章では，RI78V4 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。

12.1　概　　要

RI78V4 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 RI78V4 が管理している資源（タスク，セマフォなど）を間接的に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，RI78V4 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	sta_tsk	ista_tsk	ext_tsk	ter_tsk
chg_pri	ichg_pri	ref_tsk				

- 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）

sig_sem	isig_sem	wai_sem	pol_sem	twai_sem	ref_sem

- 同期通信機能（イベントフラグ）

set_flg	iset_flg	clr_flg	wai_flg	pol_flg	twai_flg	ref_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	trcv_dtq	ref_dtq				

- 同期通信機能（メールボックス）

snd_mbx	rcv_mbx	prcv_mbx	trcv_mbx	ref_mbx

- メモリ・プール管理機能

get_mpf	pget_mpf	tget_mpf	rel_mpf	ref_mpf

- 時間管理機能

sta_cyc	stp_cyc	ref_cyc

- 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	ena_dsp	dis_dsp	sns_ctx	sns_loc	sns_dsp	sns_dpn

- システム構成管理機能

ref_ver

12.2　サービス・コールの呼び出し

サービス・コールを C 言語，またはアセンブリ言語で記述された処理プログラムから呼び出す方法を以下に示します。

12.2.1　C 言語形式の呼び出し

通常の C 言語関数と同様の方法で呼び出しを行うことにより，サービス・コールのパラメータは RI78V4 に引き数として渡され，該当処理が実行されます。

【C 言語におけるサービス・コールの呼び出しイメージ】

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */
#include        <kernel_id.h>           /* システム情報ヘッダ・ファイルの定義 */

void
func_task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      tskid = ID_tskA;        /* 変数の宣言，初期化 */

        ercd = act_tsk ( tskid );       /* サービス・コールの呼び出し */

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

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

備考　RI78V4 が提供するサービス・コールを処理プログラムから呼び出す場合，以下に示したヘッダ・ファイルの定義（インクルード処理）が必要となります。

kernel.h：　標準ヘッダ・ファイル（C 言語用）
kernel_id.h：　システム情報ヘッダ・ファイル（C 言語用）

12.2.2　アセンブリ言語形式の呼び出し

アセンブラの関数呼び出し規約にしたがってパラメータの設定を行ったのち，CALL命令で呼び出しを行うことにより，サービス・コールのパラメータはRI78V4に引き数として渡され，該当処理が実行されます。

【アセンブリ言語におけるサービス・コールの呼び出しイメージ】

```
$INCLUDE        (kernel.inc)            ; 標準ヘッダ・ファイルの定義
$INCLUDE        (kernel_id.inc)         ; システム情報ヘッダ・ファイルの定義

        .SECTION .bss, BSS
_ercd:
        DS      (2)                     ; 戻り値の保存用領域を確保

       .PUBLIC  _func_task
       .SECTION  .textf, TEXTF
_func_task:
        MOV     A, #ID_tskA             ; パラメータの設定
        CALL    !!_act_tsk              ; サービス・コールの呼び出し
        MOVW    !LOWW(_ercd), AX        ; 戻り値の設定

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

        CALL    !!_ext_tsk              ; 自タスクの終了
        BR      !!__kernel_int_exit     ; カーネル終了処理へ分岐
```

備考　RI78V4が提供するサービス・コールを処理プログラムから呼び出す場合，以下に示したヘッダ・ファイルの定義（インクルード処理）が必要となります。

kernel.inc：　標準ヘッダ・ファイル（アセンブリ言語用）
kernel_id.inc：システム情報ヘッダ・ファイル（アセンブリ言語用）

12.3　サービス・コールのスタック使用量

RI78V4 では，サービス・コールの前処理／後処理で PC，PSW，HL といったレジスタ群の値を“該当サービス・コールを発行した処理プログラムのスタック（タスク・スタック，またはシステム・スタック）”に退避／復帰する処理を行っています。

また，サービス・コールの引き数を格納するための領域として“該当サービス・コールを発行した処理プログラムのスタック”が，サービス・コールの内部処理を実行する際に必要となるスタック領域として“システム・スタック”が使用されます。

したがって，タスク・スタック，システム・スタックの各領域を確保する際には，“サービス・コールの発行に伴うスタックの消費”を考慮する必要があります。

以下に，サービス・コールの発行に伴い必要となる各スタックのサイズを示します。

表 12 － 1　サービス・コールのスタック使用量（単位：バイト）

サービス・コール名	発行元スタック 引き数用	発行元スタック 内部処理用	システム・スタック 内部処理用
タスク管理機能			
act_tsk，iact_tsk	0	10	6
can_act	0	10	6
sta_tsk，ista_tsk	0	10	6
ext_tsk	0	10	12
ter_tsk	0	10	12
chg_pri，ichg_pri	0	10	16
ref_tsk	0	10	6
タスク付属同期機能			
slp_tsk	0	10	12
tslp_tsk	0	10	14
wup_tsk，iwup_tsk	0	10	6
can_wup，ican_wup	0	10	6
rel_wai，irel_wai	0	10	8
sus_tsk，isus_tsk	0	10	12
rsm_tsk，irsm_tsk	0	10	6
frsm_tsk，ifrsm_tsk	0	10	6
dly_tsk	0	10	14
同期通信機能（セマフォ）			
sig_sem，isig_sem	0	10	6
wai_sem	0	10	12
pol_sem	0	10	12
twai_sem	0	10	12
ref_sem	0	10	6
同期通信機能（イベントフラグ）			
set_flg，iset_flg	0	10	6
clr_flg	0	10	6
wai_flg	8	10	16

サービス・コール名	発行元スタック 引き数用	発行元スタック 内部処理用	システム・スタック 内部処理用
pol_flg	0	10	16
twai_flg	4	10	16
ref_flg	0	10	6
同期通信機能（データ・キュー）			
snd_dtq	0	10	16
psnd_dtq，ipsnd_dtq	0	10	16
tsnd_dtq	4	10	16
fsnd_dtq，ifsnd_dtq	0	10	6
rcv_dtq	0	10	14
prcv_dtq	0	10	14
trcv_dtq	4	10	14
ref_dtq	0	10	6
同期通信機能（メールボックス）			
snd_mbx	0	10	8
rcv_mbx	0	10	12
prcv_mbx	0	10	12
trcv_mbx	4	10	12
ref_mbx	0	10	6
メモリ・プール管理機能			
get_mpf	0	10	14
pget_mpf	0	10	14
tget_mpf	4	10	14
rel_mpf	0	10	6
ref_mpf	0	10	6
時間管理機能			
sta_cyc	0	10	12
stp_cyc	0	10	6
ref_cyc	0	10	6
システム状態管理機能			
rot_rdq，irot_rdq	0	10	8
get_tid，iget_tid	0	10	6
loc_cpu，iloc_cpu	0	10	6
unl_cpu，iunl_cpu	0	10	8
ena_dsp	0	10	6
dis_dsp	0	10	6
sns_ctx	0	10	6
sns_loc	0	10	6
sns_dsp	0	10	6

サービス・コール名	発行元スタック 引き数用	発行元スタック 内部処理用	システム・スタック 内部処理用
sns_dpn	0	10	6
システム構成管理機能			
ref_ver	0	10	6

12.4　データ・マクロ

RI78V4 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・マクロ（データ・タイプ，現在状態など）について以下に示します。

12.4.1　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・タイプを示します。
なお，データ・タイプのマクロ定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h から呼び出されるヘッダ・ファイル <ri_root>¥os¥types.h で行われています。

表 12 － 2　データ・タイプ

マクロ	型	意味
UH	unsigned short int	符号なし 16 ビット整数
VP	void __near	データ・タイプが一定しない値（ポインタ）
UINT	unsigned int	符号なし 16 ビット整数
VP_INT	signed long int	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
ID 注	unsigned char	ID
BOOL	signed int	真偽値
STAT	unsigned short int	現在状態，または待ち要因
ER	signed short int	戻り値
ER_UINT	unsigned short int	符号なし 16 ビット整数
PRI	signed char	優先度
FLGPTN	unsigned short int	ビット・パターン
MODE	unsigned char	要求条件
TMO	signed long int	待ち時間（単位：ティック）
RELTIM	unsigned long int	相対時間（単位：ティック）

注　　RI78V4 の ID 型の定義は，μITRON4.0 仕様の定義と異なります。

12.4.2　現在状態

以下に，サービス・コール（ref_tsk，ref_cyc）の発行により獲得される現在状態を示します。
なお，現在状態のマクロ定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h で行われています。

表 12 － 3　現在状態

マクロ	値	意味
TTS_RUN	0x01	RUNNING 状態
TTS_RDY	0x02	READY 状態
TTS_WAI	0x04	WAITING 状態
TTS_SUS	0x08	SUSPENDED 状態
TTS_WAS	0x0c	WAITING-SUSPENDED 状態
TTS_DMT	0x10	DORMANT 状態
TCYC_STP	0x00	動作停止状態（STP 状態）
TCYC_STA	0x01	動作開始状態（STA 状態）

12.4.3　WAITING 種別

以下に，サービス・コール（ref_tsk）の発行により獲得される WAITING 種別を示します。
なお，WAITING 種別のマクロ定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h で行われています。

表 12 － 4　WAITING 種別

マクロ	値	意味
TTW_SLP	0x0001	slp_tsk，または tslp_tsk による起床待ち状態
TTW_DLY	0x0002	dly_tsk による時間経過待ち状態
TTW_SEM	0x0004	wai_sem，または twai_sem による資源待ち状態
TTW_FLG	0x0008	wai_flg，または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ	0x0010	snd_dtq，または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ	0x0020	rcv_dtq，または trcv_dtq によるデータ受信待ち状態
TTW_MBX	0x0040	rcv_mbx，または trcv_mbx によるメッセージ待ち状態
TTW_MPF	0x2000	get_mpf，または tget_mpf によるメモリ・ブロック待ち状態

12.4.4　戻 り 値

以下に，サービス・コールからの戻り値を示します。
なお，戻り値のマクロ定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h で行われています。

表 12 － 5　戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ILUSE	-28	サービス・コールの使用方法が不正である
E_OBJ	-41	処理プログラムの現在状態が不正である
E_QOVR	-43	カウンタがオーバーフローした
E_RLWAI	-49	rel_wai，または irel_wai の発行により，WAITING 状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した
FALSE	0	偽
TRUE	1	真

12.4.5　条件コンパイル・マクロ

RI78V4 のヘッダ・ファイルは，以下のマクロにより条件コンパイルされます。

表 12 － 6　条件コンパイル・マクロ

分類	マクロ	内容
C コンパイラ・パッケージ	__REL__	CC-RL を使用

12.4.6　その他のマクロ

以下に，サービス・コールを発行する際に使用するその他のマクロを示します。
なお，その他のマクロのマクロ定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h で行われています。

表 12 － 7　その他のマクロ

マクロ	値	意味
TSK_SELF	0	自タスク
TPRI_INI	0	初期優先度
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング
TWF_ANDW	0x00	AND 待ち
TWF_ORW	0x01	OR 待ち
TPRI_SELF	0	自タスクの現在優先度
TSK_NONE	0	タスクはキューイングされていない
NULL	0	メッセージはキューイングされていない

12.5　データ構造体

RI78V4 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構造体（タスク状態情報，セマフォ状態情報など）について以下に示します。

12.5.1　タスク状態情報

以下に，ref_tsk を発行する際に使用するタスク状態情報 T_RTSK を示します。
なお，タスク状態情報 T_RTSK の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
typedef struct  t_rtsk {
        STAT    tskstat;        /* 現在状態 */
        PRI     tskpri;         /* 現在優先度 */
        PRI     tskbpri;        /* システム予約領域 */
        STAT    tskwait;        /* 待ち要因 */
        ID      wobjid;         /* 管理オブジェクトの ID*/
        TMO     lefttmo;        /* システム予約領域 */
        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;
```

【 packet.inc 】

```
rtsk_tskstat    .EQU    0x00    ; 現在状態
rtsk_tskpri     .EQU    0x02    ; 現在優先度
rtsk_tskbpri    .EQU    0x03    ; システム予約領域
rtsk_tskwait    .EQU    0x04    ; 待ち要因
rtsk_wobjid     .EQU    0x06    ; 管理オブジェクトの ID
rtsk_lefttmo    .EQU    0x08    ; システム予約領域
rtsk_actcnt     .EQU    0x0c    ; 起動要求数
rtsk_wupcnt     .EQU    0x0e    ; 起床要求数
rtsk_suscnt     .EQU    0x10    ; サスペンド要求数
```

以下に，タスク状態情報 T_RTSK の詳細を示します。

- tskstat，rtsk_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，rtsk_tskpri
  タスクの現在優先度が格納されます。

- tskbpri，rtsk_tskbpri
  システム予約領域です。

- tskwait，rtsk_tskwait
  WAITING 種別（WAITING 状態の種類）が格納されます。

  TTW_NONE： WAITING 状態に遷移していない
  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 によるイベントフラグ待ち状態
  TTW_RDTQ： rcv_dtq，または trcv_dtq によるイベントフラグ待ち状態
  TTW_MBX： rcv_mbx，または trcv_mbx によるメッセージ待ち状態
  TTW_MPF： get_mpf，または tget_mpf によるメモリ・ブロック待ち状態

- wobjid，rtsk_wobjid
  タスクが WAITING 状態へと遷移するきっかけとなった管理オブジェクト（セマフォ，イベントフラグなど）の ID が格納されます。

- lefttmo，rtsk_lefttmo
  システム予約領域です。

- actcnt，rtsk_actcnt
  タスクの起動要求数が格納されます。

- wupcnt，rtsk_wupcnt
  タスクの起床要求数が格納されます。

- suscnt，rtsk_suscnt
  タスクのサスペンド要求数が格納されます。

12.5.2　セマフォ状態情報

以下に，ref_sem を発行する際に使用するセマフォ状態情報 T_RSEM を示します。
なお，セマフォ状態情報 T_RSEM の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
typedef struct  t_rsem {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    semcnt;         /* 現在資源数 */
} T_RSEM;
```

【 packet.inc 】

```
rsem_wtskid     .EQU    0x00    ; 待ちタスクの有無
rsem_semcnt     .EQU    0x02    ; 現在資源数
```

以下に，セマフォ状態情報 T_RSEM の詳細を示します。

- wtskid，rsem_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，rsem_semcnt
  セマフォの現在資源数が格納されます。

12.5.3　イベントフラグ状態情報

以下に，ref_flg を発行する際に使用するイベントフラグ状態情報 T_RFLG を示します。
なお，イベントフラグ状態情報 T_RFLG の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
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;
```

【 packet.inc 】

```
rflg_wtskid     .EQU     0x00     ; 待ちタスクの有無
rflg_flgptn     .EQU     0x02     ; 現在ビット・パターン
```

以下に，イベントフラグ状態情報 T_RFLG の詳細を示します。

- wtskid，rflg_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，rflg_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。

12.5.4 データ・キュー詳細情報

以下に，ref_dtq を発行する際に使用するデータ・キュー詳細情報 T_RDTQ を示します。
なお，データ・キュー詳細情報 T_RDTQ の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
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;
```

【 packet.inc 】

```
rdtq_stskid     .EQU     0x00     ; データ送信待ちタスクの有無
rdtq_rtskid     .EQU     0x01     ; データ受信待ちタスクの有無
rdtq_sdtqcnt    .EQU     0x02     ; 未受信データの総数
```

以下に，データ・キュー詳細情報 T_RDTQ の詳細を示します。

- 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。

12.5.5　メッセージ

以下に，snd_mbx，rcv_mbx，prcv_mbx，および，trcv_mbx を発行する際に使用するメッセージ T_MSG，T_MSG_PRI を示します。

なお，メッセージ T_MSG，T_MSG_PRI の定義は，標準ヘッダ・ファイル <ri_root>¥include¥kernel.h から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥types.h で行われています。

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   __near   *msgque;        /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   __near   *msgque;        /* システム予約領域 */
        PRI     msgpri;                         /* 優先度 */
} T_MSG_PRI;
```

以下に，メッセージ T_MSG，T_MSG_PRI の詳細を示します。

- msgque
  システム予約領域です。

- msgpri
  メッセージの優先度が格納されます。

  備考 1　RI78V4 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，“1 ～ 31”に限られます。

12.5.6　メールボックス状態情報

以下に，ref_mbx を発行する際に使用するメールボックス状態情報 T_RMBX を示します。
なお，メールボックス状態情報 T_RMBX の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
typedef struct  t_rmbx {
        ID      wtskid;                 /* 待ちタスクの有無 */
        T_MSG   __near   *pk_msg;       /* 待ちメッセージの有無 */
} T_RMBX;
```

【 packet.inc 】

```
rmbx_wtskid     .EQU     0x00            ; 待ちタスクの有無
rmbx_pk_msg     .EQU     0x02            ; 待ちメッセージの有無
```

以下に，メールボックス状態情報 T_RMBX の詳細を示します。

- wtskid，rmbx_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，rmbx_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

12.5.7　固定長メモリ・プール状態情報

以下に，ref_mpf を発行する際に使用する固定長メモリ・プール状態情報 T_RMPF を示します。
なお，固定長メモリ・プール状態情報 T_RMPF の定義は，標準ヘッダ・ファイル <ri_root>¥inc¥ude¥{kernel.h, kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥inc¥ude¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
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;
```

【 packet.inc 】

```
rmpf_wtskid     .EQU     0x00     ; 待ちタスクの有無
rmpf_fblkcnt    .EQU     0x02     ; 空きメモリ・ブロックの総数
```

以下に，固定長メモリ・プール状態情報 T_RMPF の詳細を示します。

- wtskid，rmpf_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，rmpf_fblkcnt
  固定長メモリ・プールから獲得可能な空きメモリ・ブロックの総数が格納されます。

12.5.8　周期ハンドラ状態情報

以下に，ref_cyc を発行する際に使用する周期ハンドラ状態情報 T_RCYC を示します。
なお，周期ハンドラ状態情報 T_RCYC の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
typedef struct  t_rcyc {
        STAT    cycstat;        /* 現在状態 */
        RELTIM  lefttim;        /* 残り時間 */
} T_RCYC;
```

【 packet.inc 】

```
rcyc_cycstat    .EQU    0x00    ; 現在状態
rcyc_lefttim    .EQU    0x02    ; 残り時間
```

以下に，周期ハンドラ状態情報 T_RCYC の詳細を示します。

- cycstat，rcyc_cycstat
  周期ハンドラの現在状態が格納されます。

  TCYC_STP：　動作停止状態（STP 状態）
  TCYC_STA：　動作開始状態（STA 状態）

- lefttim，rcyc_lefttim
  周期ハンドラが次に起動するまでの残り時間（単位：ティック）が格納されます。
  なお，対象周期ハンドラが動作停止状態（STP 状態）の場合には，本メンバの内容は不定値となります。

12.5.9　バージョン情報

以下に，ref_ver を発行する際に使用するバージョン情報 T_RVER を示します。
なお，バージョン情報 T_RVER の定義は，標準ヘッダ・ファイル <ri_root>¥include¥{kernel.h，kernel.inc} から呼び出されるヘッダ・ファイル <ri_root>¥include¥os¥{packet.h，packet.inc} で行われています。

【 packet.h 】

```
typedef struct  t_rver {
        UH      maker;          /* メーカ・コード */
        UH      prid;           /* 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];        /* 管理情報 */
} T_RVER;
```

【 packet.inc 】

```
verinf_maker    .EQU    0x00    ; メーカ・コード
verinf_prid     .EQU    0x02    ; 識別番号
verinf_spver    .EQU    0x04    ;ITRON 仕様のバージョン番号
verinf_prver    .EQU    0x06    ; カーネルのバージョン番号
verinf_prno     .EQU    0x08    ; 管理情報
```

以下に，バージョン情報 T_RVER の詳細を示します。

- maker，verinf_maker
  カーネルのメーカ・コードが格納されます。
  0x011b：　ルネサスエレクトロニクス製

- prid，verinf_prid
  カーネルの識別番号が格納されます。
  0x0006：　識別番号

- spver，verinf_spver
  カーネルが準拠している ITRON 仕様のバージョン番号が格納されます。
  0x5403：　μITRON4.0 仕様 Ver.4.03.03

- prver，verinf_prver
  カーネルのバージョン番号が格納されます。
  0x02*xx*：　Ver.2.*xx*

- prno[0]，verinf_prno
  カーネルの版数が格納されています。
  0x0：　V バージョン

- prno[1]，verinf_prno + 0x2
  カーネルのメモリ・モデルが格納されています。
  0x2：　ミディアム・モデル

- prno[2]，verinf_prno + 0x4
  システム予約領域です。

- prno[3]，verinf_prno + 0x6
  システム予約領域です。

12.6　タスク管理機能

以下に，RI78V4 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 12 － 8　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，非タスク
iact_tsk	タスクの起動（起動要求をキューイングする）	タスク，非タスク
can_act	起動要求の解除	タスク，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，非タスク
ista_tsk	タスクの起動（起動要求をキューイングしない）	タスク，非タスク
ext_tsk	自タスクの終了	タスク
ter_tsk	他タスクの強制終了	タスク
chg_pri	優先度の変更	タスク，非タスク
ichg_pri	優先度の変更	タスク，非タスク
ref_tsk	タスクの状態参照	タスク，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 );
```

アセンブリ言語形式

```
MOV      A, #tskid
CALL     !!_act_tsk

MOV      A, #tskid
CALL     !!_iact_tsk
```

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 状態へと遷移させます。

これにより，対象タスクは，初期優先度に対応したレディ・キューの最後尾にキューイングされ，RI78V4 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 状態以外の状態へと遷移していた場合には，状態操作処理は実行されず，起動要求カウンタの加算処理（起動要求カウンタに 0x1 を加算）が実行されます。

備考 1　RI78V4 が管理する起動要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，起動要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

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，起動コードとして“拡張情報 exinf”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_QOVR	-43	起動要求数が 127 を越えた

can_act

概要

起動要求の解除

C 言語形式

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

アセンブリ言語形式

```
MOV      A, #tskid
CALL     !!_can_act
```

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 0x0 を設定）します。
なお，本サービス・コールが正常終了した際には，戻り値として“解除した起動要求数”が返されます。

戻り値

マクロ	数値	意味
その他	－	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 );
```

アセンブリ言語形式

```
MOVW    BC, #stacd_lo
MOVW    DE, #stacd_hi
MOV     A, #tskid
CALL    !!_sta_tsk

MOVW    BC, #stacd_lo
MOVW    DE, #stacd_hi
MOV     A, #tskid
CALL    !!_ista_tsk
```

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 状態へと遷移させます。
これにより，対象タスクは，初期優先度に対応したレディ・キューの最後尾にキューイングされ，RI78V4 のスケジューリング対象となります。

備考 1　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，状態操作処理は実行されず，戻り値として“E_OBJ”が返されます。

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，起動コードとして“*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_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 );
```

アセンブリ言語形式

```
BR      !!_ext_tsk
```

パラメータ

なし

機能

自タスクを RUNNING 状態から DORMANT 状態へと遷移させます。
これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起動要求がキューイングされていた（起動要求カウンタが 0x0 以外であった）場合には，RUNNING 状態から DORMANT 状態への状態操作処理，および，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）を行ったのち，DORMANT 状態から READY 状態への状態操作処理もあわせて実行されます。

備考 1　本サービス・コールでは，自タスクが sig_sem，get_mpf などの発行により獲得した OS 資源の返却は行いません。したがって，獲得中の OS 資源については，本サービス・コールを発行する以前に返却する必要があります。

備考 2　本サービス・コールでは，RUNNING 状態から DORMANT 状態への状態操作処理を実行する際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスクの生成時に設定される値で初期化しています。

備考 3　タスク内で return 命令を記述した場合，本サービス・コールと同様の動作が実行されます。

備考 4　RI78V4 では，“自タスクの終了”として return 命令を記述した方がコード効率が良くなります。

戻り値

なし

ter_tsk

概要

他タスクの強制終了

C 言語形式

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

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_ter_tsk
```

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，RI78V4 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，対象タスクに起動要求がキューイングされていた（起動要求カウンタが 0x0 以外であった）場合には，DORMANT 状態への状態操作処理，および，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）を行ったのち，DORMANT 状態から READY 状態への状態操作処理もあわせて実行されます。

備考 1　本サービス・コールでは，対象タスクが sig_sem，get_mpf などの発行により獲得した OS 資源の返却は行いません。したがって，獲得中の OS 資源については，本サービス・コールを発行する以前に返却する必要があります。

備考 2　本サービス・コールでは，DORMANT 状態への状態操作処理を実行する際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスクの生成時に設定される値で初期化しています。

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_OBJ	-41	対象タスクが DORMANT 状態である

chg_pri
ichg_pri

概要

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 );
```

アセンブリ言語形式

```
MOVW    AX, #(tskid | tskpri)
CALL    !!_chg_pri

MOVW    AX, #(tskid | tskpri)
CALL    !!_ichg_pri
```

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*;	タスクの優先度 TPRI_INI：　タスクの初期優先度 数値：　　タスクの優先度

機能

tskid で指定されたタスクの優先度（現在優先度）を *tskpri* で指定された値に変更します。

備考　本サービス・コールを発行した際，対象タスクが RUNNING 状態，または READY 状態であった場合には，優先度の変更処理を実行したのち，*tskpri* で指定された優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_OBJ	-41	対象タスクが DORMANT 状態である

ref_tsk

概要

タスクの状態参照

C 言語形式

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

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rtsk)
MOV     A,  #tskid
CALL    !!_ref_tsk
```

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;	タスク状態情報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タスク状態情報（現在状態など）を *pk_rtsk* で指定された領域に格納します。

備考　　タスク状態情報 T_RTSK についての詳細は，「12.5.1　タスク状態情報」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

12.7　タスク付属同期機能

以下に，RI78V4 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 12 － 9　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，非タスク
iwup_tsk	タスクの起床	タスク，非タスク
can_wup	起床要求の解除	タスク，非タスク
ican_wup	起床要求の解除	タスク，非タスク
rel_wai	WAITING 状態の強制解除	タスク，非タスク
irel_wai	WAITING 状態の強制解除	タスク，非タスク
sus_tsk	SUSPENDED 状態への移行	タスク，非タスク
isus_tsk	SUSPENDED 状態への移行	タスク，非タスク
rsm_tsk	SUSPENDED 状態の解除	タスク，非タスク
irsm_tsk	SUSPENDED 状態の解除	タスク，非タスク
frsm_tsk	SUSPENDED 状態の強制解除	タスク，非タスク
ifrsm_tsk	SUSPENDED 状態の強制解除	タスク，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 );
```

アセンブリ言語形式

```
CALL    !!_slp_tsk
```

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。
これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起床要求がキューイングされていた（起床要求カウンタが 0x0 以外であった）場合には，状態操作処理は実行されず，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）が実行されます。
なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
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，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床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_RLWAI	-49	rel_wai，または irel_wai の発行により，起床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 );
```

アセンブリ言語形式

```
MOVW    BC, #tmout_hi
MOVW    AX, #tmout_lo
CALL    !!_tslp_tsk
```

パラメータ

I/O	パラメータ	説明
I	TMO tmout;	待ち時間（単位：ティック） 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 WAITING 状態（起床待ち状態）へと遷移させます。
これにより，自タスクは，レディ・キューから外れ，RI78V4 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，自タスクに起床要求がキューイングされていた（起床要求カウンタが 0x0 以外であった）場合には，状態操作処理は実行されず，起床要求カウンタの減算処理（起床要求カウンタから 0x1 を減算）が実行されます。
なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
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，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床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_RLWAI	-49	rel_wai，または irel_wai の発行により，起床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

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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_wup_tsk

MOV     A, #tskid
CALL    !!_iwup_tsk
```

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の WAITING 状態（起床待ち状態）を解除します。

これにより，対象タスクは，起床待ち状態から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

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であった場合には，状態操作処理は実行されず，起床要求カウンタの加算処理（起床要求カウンタに 0x1 を加算）が実行されます。

備考 1 本サービス・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2 RI78V4 が管理している起床要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，起床要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_OBJ	-41	対象タスクが DORMANT 状態である
E_QOVR	-43	起床要求数が 127 を越えた

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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_can_wup

MOV     A, #tskid
CALL    !!_ican_wup
```

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 0x0 を設定）します。
なお，本サービス・コールが正常終了した際には，戻り値として“解除した起床要求数”が返されます。

戻り値

マクロ	数値	意味
E_OBJ	-41	対象タスクが DORMANT 状態である
その他	－	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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_rel_wai

MOV     A, #tskid
CALL    !!_irel_wai
```

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”が返されます。

備考 1 本サービス・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2 本サービス・コールでは，強制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”が返されます。

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_OBJ	-41	対象タスクが WAITING 状態，または WAITING-SUSPENDED 状態でない

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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_sus_tsk

MOV     A, #tskid
CALL    !!_isus_tsk
```

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のサスペンド要求カウンタに 0x1 を加算したのち，対象タスクを 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，状態操作処理は実行されず，サスペンド要求カウンタの加算処理のみが実行されます。

なお，SUSPENDED 状態の解除は，以下の場合に行われ，SUSPENDED 状態から READY 状態へと遷移します。

SUSPENDED 状態の解除操作	エラー・コード
rsm_tsk の発行により，解除要求が発行された	E_OK
irsm_tsk の発行により，解除要求が発行された	E_OK
frsm_tsk の発行により，SUSPENDED 状態を強制的に解除された	E_OK
ifrsm_tsk の発行により，SUSPENDED 状態を強制的に解除された	E_OK

備考 1　本サービス・コールを発行した際，対象タスクが自タスクの場合は，レディ・キューから外れ，RI78V4 のスケジューリング対象から除外されます。

備考 2　RI78V4 が管理するサスペンド要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，サスペンド要求数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_OBJ	-41	対象タスクが DORMANT 状態である
E_QOVR	-43	サスペンド要求数が 127 を越えた

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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_rsm_tsk

MOV     A, #tskid
CALL    !!_irsm_tsk
```

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サスペンド要求カウンタから 0x1 を減算したのち，対象タスクの SUSPENDED 状態を解除します。

これにより，対象タスクは，SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移します。

ただし，本サービス・コールを発行した際，サスペンド要求がキューイングされていた（減算結果が 0x0 以外）場合には，状態操作処理は実行されず，サスペンド要求カウンタの減算処理のみが実行されます。

備考 1 本サービス・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2 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”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_OBJ	-41	対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態でない

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 );
```

アセンブリ言語形式

```
MOV     A, #tskid
CALL    !!_frsm_tsk

MOV     A, #tskid
CALL    !!_ifrsm_tsk
```

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サスペンド要求カウンタに 0x0 を設定し，対象タスクの SUSPENDED 状態を強制的に解除します。

これにより，対象タスクは，SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移します。

備考 1 本サービス・コールを発行した際，対象タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2 本サービス・コールでは，強制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”が返されます。

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_OBJ	-41	対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態でない

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 );
```

アセンブリ言語形式

```
MOVW    BC, #dlytim_hi
MOVW    AX, #dlytim_lo
CALL    !!_dly_tsk
```

パラメータ

I/O	パラメータ	説明
I	RELTIM　*dlytim*;	遅延時間（単位：ティック）

機能

自タスクを RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
これにより，自タスクは，レディ・キューから外れ，RI78V4 のスケジューリング対象から除外されます。
なお，時間経過待ち状態の解除は，以下の場合に行われ，時間経過待ち状態から READY 状態へと遷移します。

時間経過待ち状態の解除操作	エラー・コード
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，時間経過待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，時間経過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_RLWAI	-49	rel_wai，または irel_wai の発行により，起床待ち状態を強制的に解除された

12.8　同期通信機能（セマフォ）

以下に，RI78V4 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 12 － 10　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
sig_sem	資源の返却	タスク，非タスク
isig_sem	資源の返却	タスク，非タスク
wai_sem	資源の獲得（永久待ち）	タスク
pol_sem	資源の獲得（ポーリング）	タスク，非タスク
twai_sem	資源の獲得（タイムアウト付き）	タスク
ref_sem	セマフォの状態参照	タスク，非タスク

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 );
```

アセンブリ言語形式

```
MOV     A, #semid
CALL    !!_sig_sem

MOV     A, #semid
CALL    !!_isig_sem
```

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 0x1 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，カウンタ操作処理は実行されず，該当タスク（待ちキューの先頭タスク）に資源が渡されます。

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　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　RI78V4 が管理しているセマフォ・カウンタは，7 ビット幅で構成されています。このため，本サービス・コールの発行により，資源数が最大カウント値 127 を越える場合には，カウンタ操作処理は実行されず，戻り値として“E_QOVR”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_QOVR	-43	資源数が 127 を越えた

wai_sem

概要

資源の獲得（永久待ち）

C 言語形式

```
ER      wai_sem ( ID semid );
```

アセンブリ言語形式

```
MOV     A, #semid
CALL    !!_wai_sem
```

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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，自タスクを対象セマフォの待ちキューに資源の獲得要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（資源待ち状態）へと遷移します。

なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から READY 状態へと遷移します。

資源待ち状態の解除操作	エラー・コード
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，資源待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源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_RLWAI	-49	rel_wai，または irel_wai の発行により，資源待ち状態を強制的に解除された

pol_sem

概要

資源の獲得（ポーリング）

C 言語形式

```
ER      pol_sem ( ID semid );
```

アセンブリ言語形式

```
MOV     A, #semid
CALL    !!_pol_sem
```

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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，戻り値として“E_TMOUT”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_TMOUT	-50	対象セマフォの資源数が 0x0 である

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 );
```

アセンブリ言語形式

```
MOVW    BC, #tmout_lo
MOVW    DE, #tmout_hi
MOV     A,  #semid
CALL    !!_twai_sem
```

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：ティック） 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 0x1 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（セマフォ・カウンタが 0x0 であった）場合には，カウンタ操作処理は実行されず，自タスクを対象セマフォの待ちキューに資源の獲得要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（資源待ち状態）へと遷移します。

なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から READY 状態へと遷移します。

資源待ち状態の解除操作	エラー・コード
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，資源待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源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“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_RLWAI	-49	rel_wai，または irel_wai の発行により，資源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

ref_sem

概要

セマフォの状態参照

C 言語形式

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

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rsem)
MOV     A, #semid
CALL    !!_ref_sem
```

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;	セマフォ状態情報を格納する領域へのポインタ

機能

semid で指定されたセマフォのセマフォ状態情報（待ちタスクの有無など）を *pk_rsem* で指定された領域に格納します。

備考　　セマフォ状態情報 T_RSEM についての詳細は，「12.5.2　セマフォ状態情報」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

12.9　同期通信機能（イベントフラグ）

以下に，RI78V4 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 12 － 11　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，非タスク
iset_flg	ビット・パターンのセット	タスク，非タスク
clr_flg	ビット・パターンのクリア	タスク，非タスク
wai_flg	ビット・パターンのチェック（永久待ち）	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，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	イベントフラグの状態参照	タスク，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 );
```

アセンブリ言語形式

```
MOVW    BC, #setptn
MOV     A,  #flgid
CALL    !!_set_flg

MOVW    BC, #setptn
MOV     A,  #flgid
CALL    !!_iset_flg
```

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（16 ビット幅）

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 を対象イベントフラグのビット・パターンとして設定します。

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待ちキューにキューイングされているタスクの要求条件を満足した場合には，ビット・パターンの設定処理とともに，該当タスクを待ちキューから外します。

これにより，該当タスクは，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 状態へと遷移します。

備考 1 本サービス・コールを発行した際，待ちキューのタスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2 本サービス・コールを発行した際，対象イベントフラグに設定されているビット・パターンが B'1100，*setptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1110 となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

clr_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 );
```

アセンブリ言語形式

```
MOVW    BC, #clrptn
MOV     A,  #flgid
CALL    !!_clr_flg
```

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（16 ビット幅）

機能

flgid で指定されたイベントフラグに設定されているビット・パターンと *clrptn* で指定されたビット・パターンの論理積 AND を対象イベントフラグのビット・パターンとして設定します。

備考 1　本サービス・コールでは，クリア要求のキューイングが行われません。このため，ビット・パターンがクリア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・コールを発行した際，対象イベントフラグに設定されているビット・パターンが B'1100，*clrptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1000 となります。

備考 3　本サービス・コールでは，イベントフラグ待ち状態のタスクが“待ち解除”されることはありません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

wai_flg

概要

ビット・パターンのチェック（永久待ち）

C 言語形式

```
ER      wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

アセンブリ言語形式

```
MOVW    DE, #LOWW(_p_flgptn)
MOVW    BC, #waiptn
MOVW    AX, #(flgid | wfmode)
CALL    !!_wai_flg
```

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（16 ビット幅）
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ます。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移します。

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，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
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

イベントフラグ待ち状態の解除操作	エラー・コード
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，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ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 RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ILUSE	-28	待ちタスクがキューイングされているイベントフラグに対して本サービス・コールを発行した
E_RLWAI	-49	rel_wai，または irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された

pol_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 );
```

アセンブリ言語形式

```
MOVW    DE, #LOWW(_p_flgptn)
MOVW    BC, #waiptn
MOVW    AX, #(flgid | wfmode)
CALL    !!_pol_flg
```

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（16 ビット幅）
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”が返されます。

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　RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ILUSE	-28	タスクがキューイングされているイベントフラグに対して本サービス・コールを発行した
E_TMOUT	-50	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い

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
);
```

アセンブリ言語形式

```
MOVW    AX, #tmout_hi
PUSH    AX
MOVW    AX, #tmout_lo
PUSH    AX
MOVW    DE, #LOWW(_p_flgptn)
MOVW    BC, #waiptn
MOVW    AX, #(flgid | wfmode)
CALL    !!_twai_flg
addw    sp, #0x0004
```

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（16 ビット幅）
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：ティック） 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ます。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移します。

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，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
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，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ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 RI78V4 では，イベントフラグの待ちキューにキューイング可能なタスク数は 1 個としています。このため，タスクがキューイングされているイベントフラグに対して本サービス・コール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_ILUSE”が返されます。

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，RI78V4 はビット・パターンのクリア処理（0x0 の設定）を実行します。

備考 3 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_ILUSE	-28	タスクがキューイングされているイベントフラグに対して本サービス・コールを発行した
E_RLWAI	-49	rel_wai，または irel_wai の発行により，イベントフラグ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

ref_flg

概要

イベントフラグの状態参照

C 言語形式

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

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rflg)
MOV     A, #flgid
CALL    !!_ref_flg
```

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*;	イベントフラグ状態情報を格納する領域へのポインタ

機能

flgid で指定されたイベントフラグのイベントフラグ状態情報（待ちタスクの有無など）を *pk_rflg* で指定された領域に格納します。

備考　イベントフラグ状態情報 T_RFLG についての詳細は，「12.5.3　イベントフラグ状態情報」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

12.10 同期通信機能（データ・キュー）

以下に，RI78V4 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 12 － 12　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，非タスク， 初期化ルーチン
ipsnd_dtq	データの送信（ポーリング）	タスク，非タスク， 初期化ルーチン
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，非タスク， 初期化ルーチン
ifsnd_dtq	データの強制送信	タスク，非タスク， 初期化ルーチン
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，非タスク， 初期化ルーチン
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，非タスク， 初期化ルーチン

snd_dtq

概要

データの送信

C 言語形式

```
ER      snd_dtq ( ID dtqid, VP_INT data );
```

アセンブリ言語形式

```
MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOV     A,  #dtqid
CALL    !!_snd_dtq
```

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・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，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，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域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・キューのデータ・キュー領域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・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。

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）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ送信待ち状態を強制的に解除された

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 );
```

アセンブリ言語形式

```
MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOV     A,  #dtqid
CALL    !!_psnd_dtq

MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOV     A,  #dtqid
CALL    !!_ipsnd_dtq
```

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・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，戻り値として E_TMOUT を返します。

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。

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_TMOUT	-50	対象データ・キューのデータ・キュー領域に空き領域が存在しない

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 );
```

アセンブリ言語形式

```
MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOVW    AX, #tmout_hi
PUSH    AX
MOVW    AX, #tmout_lo
PUSH    AX
MOV     A,  #dtqid
CALL    !!_tsnd_dtq
```

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：ティック） 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・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，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，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域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・キューのデータ・キュー領域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・キューのデータ・キュー領域に空き領域が確保された	E_OK

データ送信待ち状態の解除操作	エラー・コード
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信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

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。

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）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，ipsnd_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_RLWAI	-49	rel_wai，または irel_wai の発行により，データ送信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

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 );
```

アセンブリ言語形式

```
MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOV     A,  #dtqid
CALL    !!_fsnd_dtq

MOVW    DE, !LOWW(_data+0x00002)
MOVW    BC, !LOWW(_data)
MOV     A,  #dtqid
CALL    !!_ifsnd_dtq
```

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・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，書き込まれてから最も時間が経過しているデータの領域に該当データを上書きします。

また，本サービス・コールを発行した際，対象データ・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た場合には，データの書き込みは行わず，該当タスクにデータを渡します。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。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

rcv_dtq

概要

データの受信

C 言語形式

```
ER      rcv_dtq ( ID dtqid, VP_INT *p_data );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_data)
MOV     A,  #dtqid
CALL    !!_rcv_dtq
```

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･キューのデータ･キュー領域からデータを読み込み, *p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，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，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI

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　rel_wai，または irel_wai の発行によりデータ受信待ち状態を解除された場合，*p_data* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ受信待ち状態を強制的に解除された

prcv_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 );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_data)
MOV     A,  #dtqid
CALL    !!_prcv_dtq
```

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･キューのデータ･キュー領域からデータを読み込み，*p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，戻り値として E_TMOUT を返します。

備考　本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合，*p_data* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_TMOUT	-50	対象データ・キューのデータ・キュー領域にデータが書き込まれていない

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 );
```

アセンブリ言語形式

```
MOVW    AX, #tmout_hi
PUSH    AX
MOVW    AX, #tmout_lo
PUSH    AX
MOVW    BC, #LOWW(_data)
MOV     A,  #dtqid
CALL    !!_trcv_dtq
```

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：ティック） 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･キューのデータ･キュー領域からデータを読み込み，*p_data* 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，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，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 状態からタイムアウト付きの 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，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK

データ受信待ち状態の解除操作	エラー・コード
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信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　rel_wai，または irel_wai の発行，または待ち時間の経過によりデータ受信待ち状態を解除された場合，*p_data* で指定された領域の内容は不定となります。

備考 3　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_RLWAI	-49	rel_wai，または irel_wai の発行により，データ受信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

ref_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 );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rdtq)
MOV     A, #dtqid
CALL    !!_ref_dtq
```

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。

備考　データ・キュー詳細情報 T_RDTQ についての詳細は，「12.5.4　データ・キュー詳細情報」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

12.11 同期通信機能（メールボックス）

以下に，RI78V4 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 12 － 13　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，非タスク
rcv_mbx	メッセージの受信（永久待ち）	タスク
prcv_mbx	メッセージの受信（ポーリング）	タスク，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	メールボックスの状態参照	タスク，非タスク

snd_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 );
```

アセンブリ言語形式

```
SUBW    SP, #0x06
MOVW    [SP+0x02], AX
MOVW    AX, BC
MOVW    [SP+0x04], AX
MOVW    AX, SP
MOVW    BC, AX
MOV     A, #mbxid
CALL    !!_snd_mbx
```

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

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（待ちキューにメッセージをキューイング）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，メッセージのキューイング処理は実行されず，該当タスク（待ちキューの先頭タスク）にメッセージが渡されます。

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　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　メッセージ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属性（キューイング方式）mbxatr で定義された順（FIFO 順，優先度順）となります。

備考 3　RI78V4　のメールボックスは，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ありません。したがって，本サービス・コールの発行後であってもメッセージの内容を書き換えることができます。

備考 4　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」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

rcv_mbx

概要

メッセージの受信（永久待ち）

C 言語形式

```
ER      rcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_ppk_msg)
MOV     A, #mbxid
CALL    !!_rcv_mbx
```

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

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，メッセージの受信処理は実行されず，自タスクを対象メールボックスの待ちキューにメッセージの受信要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メッセージ待ち状態）へと遷移します。

なお，メッセージ待ち状態の解除は，以下の場合に行われ，メッセージ待ち状態から READY 状態へと遷移します。

メッセージ待ち状態の解除操作	エラー・コード
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
rel_wai の発行により，メッセージ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ待ち状態を強制的に解除された	E_RLWAI

備考　　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」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_RLWAI	-49	rel_wai，または irel_wai の発行により，メッセージ待ち状態を強制的に解除された

prcv_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 );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_ppk_msg)
MOV     A, #mbxid
CALL    !!_prcv_mbx
```

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

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，メッセージの受信処理は実行されず，戻り値として“E_TMOUT”が返されます。

備考　　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」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_TMOUT	-50	対象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ない

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 );
```

アセンブリ言語形式

```
MOVW    AX, #tmout_hi
PUSH    AX
MOVW    AX, #tmout_lo
PUSH    AX
MOVW    BC, #LOWW(_ppk_msg)
MOV     A, #mbxid
CALL    !!_trcv_mbx
ADDW    SP, #0x0004
```

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：ティック） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

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，メッセージの受信処理は実行されず，自タスクを対象メールボックスの待ちキューにメッセージの受信要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メッセージ待ち状態）へと遷移します。

なお，メッセージ待ち状態の解除は，以下の場合に行われ，メッセージ待ち状態から READY 状態へと遷移します。

メッセージ待ち状態の解除操作	エラー・コード
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

メッセージ待ち状態の解除操作	エラー・コード
rel_wai の発行により，メッセージ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ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　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 2　メッセージ T_MSG，T_MSG_PRI についての詳細は，「12.5.5　メッセージ」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_RLWAI	-49	rel_wai，または irel_wai の発行により，メッセージ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

ref_mbx

概要

メールボックスの状態参照

C 言語形式

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

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rmbx)
MOV     A,  #mbxid
CALL    !!_ref_mbx
```

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*;	メールボックス状態情報を格納する領域へのポインタ

機能

mbxid で指定されたメールボックスのメールボックス状態情報（待ちタスクの有無など）を *pk_rmbx* で指定された領域に格納します。

備考　メールボックス状態情報 T_RMBX についての詳細は，「12.5.6　メールボックス状態情報」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

12.12 メモリ・プール管理機能

以下に，RI78V4 がメモリ・プール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 12 － 14　メモリ・プール管理機能

サービス・コール名	機能概要	発行有効範囲
get_mpf	メモリ・ブロックの獲得（永久待ち）	タスク
pget_mpf	メモリ・ブロックの獲得（ポーリング）	タスク，非タスク
tget_mpf	メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpf	メモリ・ブロックの返却	タスク，非タスク
ref_mpf	固定長メモリ・プールの状態参照	タスク，非タスク

get_mpf

概要

メモリ・ブロックの獲得（永久待ち）

C 言語形式

```
ER      get_mpf ( ID mpfid, VP *p_blk );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_p_blk)
MOV     A,  #mpfid
CALL    !!_get_mpf
```

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*;	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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，自タスクを対象固定長メモリ・プールの待ちキューにメモリ・ブロックの獲得要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メモリ・ブロック待ち状態）へと遷移します。

なお，メモリ・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から READY 状態へと遷移します。

メモリ・ブロック待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールにメモリ・ブロックが返却された	E_OK
rel_wai の発行により，メモリ・ブロック待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メモリ・ブロック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_RLWAI	-49	rel_wai，または irel_wai の発行により，メモリ・ブロック待ち状態を強制的に解除された

pget_mpf

概要

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 );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_p_blk)
MOV     A, #mpfid
CALL    !!_pget_mpf
```

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*;	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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，戻り値として“E_TMOUT”が返されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_TMOUT	-50	対象固定長メモリ・プールに空きメモリ・ブロックが存在しない

tget_mpf

概要

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 );
```

アセンブリ言語形式

```
MOVW    AX, #tmout_hi
PUSH    AX
MOVW    AX, #tmout_lo
PUSH    AX
MOVW    BC, #LOWW(_p_blk)
MOV     A, #mpfid
CALL    !!_tget_mpf
ADDW    SP, #0x0004
```

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*;	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：ティック） 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・プールからメモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，メモリ・ブロックの獲得処理は実行されず，自タスクを対象固定長メモリ・プールの待ちキューにメモリ・ブロックの獲得要求順（FIFO 順）でキューイングします。

これにより，自タスクは，レディ・キューから外れ，RUNNING 状態から WAITING 状態（メモリ・ブロック待ち状態）へと遷移します。

なお，メモリ・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から READY 状態へと遷移します。

メモリ・ブロック待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールにメモリ・ブロックが返却された	E_OK
rel_wai の発行により，メモリ・ブロック待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メモリ・ブロック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“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_RLWAI	-49	rel_wai，または irel_wai の発行により，メモリ・ブロック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

rel_mpf

概要

メモリ・ブロックの返却

C 言語形式

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

アセンブリ言語形式

```
MOVW    BC, #LOWW(_blk)
MOV     A, #mpfid
CALL    !!_rel_mpf
```

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*;	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，メモリ・ブロックの返却処理は実行されず，該当タスク（待ちキューの先頭タスク）にメモリ・ブロックが渡されます。

これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メモリ・ブロック待ち状態）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　本サービス・コールを発行した際，待ちキューの先頭タスクが READY 状態へと遷移する場合は，該当タスクの優先度に対応したレディ・キューの最後尾にキューイングし直す処理もあわせて実行されます。

備考 2　RI78V4 では，メモリ・ブロックを返却する際，メモリ・ブロックのクリア処理を行っていません。したがって，返却されたメモリ・ブロックの内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

ref_mpf

概要

固定長メモリ・プールの状態参照

C 言語形式

```
ER      ref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rmpf)
MOV     A,  #mpfid
CALL    !!_ref_mpf
```

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・プール状態情報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・プールの固定長メモリ・プール状態情報（待ちタスクの有無など）を *pk_rmpf* で指定された領域に格納します。

備考　固定長メモリ・プール状態情報 T_RMPF についての詳細は，「12.5.7　固定長メモリ・プール状態情報」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

12.13 時間管理機能

以下に，RI78V4 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 12 － 15　時間管理機能

サービス・コール名	機能概要	発行有効範囲
sta_cyc	動作開始状態（STA 状態）への移行	タスク，非タスク
stp_cyc	動作停止状態（STP 状態）への移行	タスク，非タスク
ref_cyc	周期ハンドラの状態参照	タスク，非タスク

sta_cyc

概要

動作開始状態（STA 状態）への移行

C 言語形式

```
ER      sta_cyc ( ID cycid );
```

アセンブリ言語形式

```
MOV     A, #cycid
CALL    !!_sta_cyc
```

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

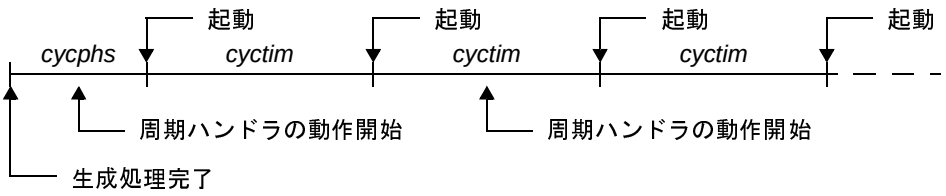
cycid で指定された周期ハンドラを動作停止状態（STP 状態）から動作開始状態（STA 状態）へと遷移させます。
これにより，対象周期ハンドラは，RI78V4 の起動対象となります。

備考　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 属性を指定しているか否かにより異なります。

【 TA_PHS 属性の指定ありの場合 】

コンフィギュレーション時に定義した起動位相（初期起動位相 *cycphs*，起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
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。

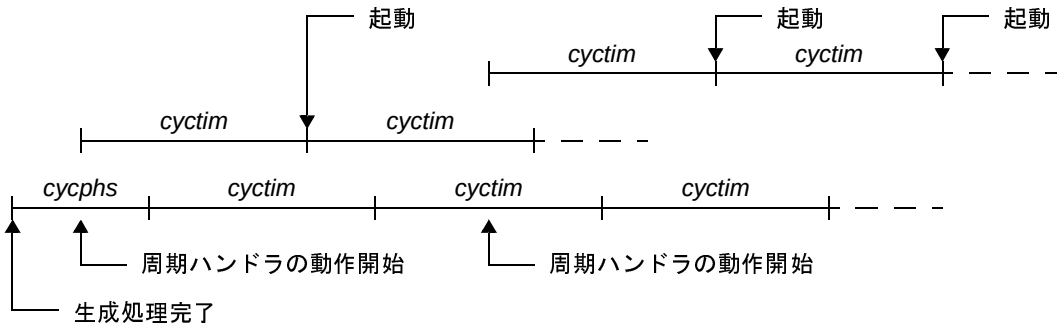
この場合の周期ハンドラの起動イメージは以下のようになります



【 TA_PHS 属性の指定なしの場合 】

本サービス・コールの発行を基準点とした起動位相（起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
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。

この場合の周期ハンドラの起動イメージは以下のようになります



戻り値

マクロ	数値	意味
E_OK	0	正常終了

stp_cyc

概要

動作停止状態（STP 状態）への移行

C 言語形式

```
ER      stp_cyc ( ID cycid );
```

アセンブリ言語形式

```
MOV     A, #cycid
CALL    !!_stp_cyc
```

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を動作開始状態（STA 状態）から動作停止状態（STP 状態）へと遷移させます。

これにより，対象周期ハンドラは，本サービス・コールの発行から sta_cyc が発行されるまでの間，RI78V4 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，対象周期ハンドラが動作停止状態（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

ref_cyc

概要

周期ハンドラの状態参照

C 言語形式

```
ER      ref_cyc ( ID cycid, T_RCYC *pk_rcyc );
```

アセンブリ言語形式

```
MOVW    BC, #LOWW(_pk_rcyc)
MOV     A,  #cycid
CALL    !!_ref_cyc
```

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;	周期ハンドラ状態情報を格納する領域へのポインタ

機能

cycid で指定された周期ハンドラの周期ハンドラ状態情報（現在状態など）を *pk_rcyc* で指定された領域に格納します。

備考　　周期ハンドラ状態情報 T_RCYC についての詳細は，「12.5.8　周期ハンドラ状態情報」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

12.14 システム状態管理機能

以下に，RI78V4 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 12 － 16　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	レディ・キューの回転	タスク，非タスク
irot_rdq	レディ・キューの回転	タスク，非タスク
get_tid	RUNNING 状態のタスクの参照	タスク，非タスク
iget_tid	RUNNING 状態のタスクの参照	タスク，非タスク
loc_cpu	CPU ロック状態への移行	タスク，非タスク
iloc_cpu	CPU ロック状態への移行	タスク，非タスク
unl_cpu	CPU ロック状態の解除	タスク，非タスク
iunl_cpu	CPU ロック状態の解除	タスク，非タスク
dis_dsp	ディスパッチ禁止状態への移行	タスク
ena_dsp	ディスパッチ禁止状態の解除	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

rot_rdq
irot_rdq

概要

レディ・キューの回転

C 言語形式

```
ER      rot_rdq ( PRI tskpri );

ER      irot_rdq ( PRI tskpri );
```

アセンブリ言語形式

```
MOV     A, #tskpri
CALL    !!_rot_rdq

MOV     A, #tskpri
CALL    !!_irot_rdq
```

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： 自タスクの現在優先度 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，回転要求のキューイングが行われません。このため，該当優先度に対応したレディ・キューにタスクが 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 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（READY 状態，または RUNNING 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“タスクの検出処理”を実行し，キューイングされているタスクを検出した際には，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，RI78V4 のスケジューリング方式を実現しています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

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 )
```

アセンブリ言語形式

```
SUBW    SP, #0x06
MOVW    [SP+0x02], AX
MOVW    AX, BC
MOVW    [SP+0x04], AX
MOVW    AX, SP
CALL    !!_get_tid

SUBW    SP, #0x06
MOVW    [SP+0x02], AX
MOVW    AX, BC
MOVW    [SP+0x04], AX
MOVW    AX, SP
CALL    !!_iget_tid
```

パラメータ

I/O	パラメータ	説明
O	ID *p_tskid;	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。

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（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

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 );
```

アセンブリ言語形式

```
CALL    !!_loc_cpu

CALL    !!_iloc_cpu
```

パラメータ

なし

機能

システム状態種別を CPU ロック状態へと変更します。

これにより，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，“マスカブル割り込みの受け付け処理”が禁止されるとともに，サービス・コールの発行が制限されます。

なお，RI78V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を unl_cpu，または iunl_cpu が発行されるまで遅延しています。

また，RI78V4 では，CPU ロック状態で発行可能なサービス・コールを以下に示したものに制限しています。

サービス・コール名	機能概要
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_ctx	コンテキスト種別の参照
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_dpn	ディスパッチ保留状態の参照

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，システム状態種別が CPU ロック状態であった場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　RI78V4 では，割り込みマスク・フラグ・レジスタ MK*xx*，および，プログラム・ステータス・ワード PSW のインサービス・プライオリティ・フラグ ISP*x* を操作して，“RI78V4 の管理下にあるマスカブル割り込みの受け付け禁止”を実現しています。このため，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，処理プログラムから該当レジスタを操作することは禁止されています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

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 );
```

アセンブリ言語形式

```
CALL    !!_unl_cpu

CALL    !!_iunl_cpu
```

パラメータ

なし

機能

システム状態種別を非 CPU ロック状態へと変更します。

これにより，loc_cpu，または iloc_cpu の発行により禁止されていた“マスカブル割り込みの受け付け処理”が許可されるとともに，および，サービス・コールの発行制限が解除されます。

なお，RI78V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。

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，システム状態種別が非 CPU ロック状態であった場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　RI78V4 では，割り込みマスク・フラグ・レジスタ MK*xx*，および，プログラム・ステータス・ワード PSW のインサービス・プライオリティ・フラグ ISP*x* を操作して，“マスカブル割り込みの受け付け許可”を実現しています。このため，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間，処理プログラムから該当レジスタを操作することは禁止されています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

dis_dsp

概要

ディスパッチ禁止状態への移行

C 言語形式

```
ER      dis_dsp ( void );
```

アセンブリ言語形式

```
CALL    !!_dis_dsp
```

パラメータ

なし

機能

システム状態種別をディスパッチ禁止状態へと変更します。
これにより，本サービス・コールの発行から ena_dsp が発行されるまでの間，ディスパッチ処理（タスクのスケジューリング処理）が禁止されます。
なお，RI78V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，ena_dsp 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。

備考 1　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，システム状態種別がディスパッチ禁止状態であった場合には，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。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

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 );
```

アセンブリ言語形式

```
CALL    !!_ena_dsp
```

パラメータ

なし

機能

システム状態種別をディスパッチ許可状態へと変更します。
これにより，dis_dsp の発行により禁止されていた“ディスパッチ処理（タスクのスケジューリング処理）”が許可されます。
なお，RI78V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“ディスパッチ処理を伴うサービス・コール（chg_pri，sig_sem など）”が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際のディスパッチ処理は，本サービス・コール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。

備考　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，システム状態種別がディスパッチ許可状態であった場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

sns_ctx

概要

コンテキスト種別の参照

C 言語形式

```
BOOL    sns_ctx ( void );
```

アセンブリ言語形式

```
CALL    !!_sns_ctx
```

パラメータ

なし

機能

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）を獲得します。
なお，本サービス・コールが正常終了した際には，戻り値として“獲得したコンテキスト種別（TRUE：非タスク・コンテキスト，FALSE：タスク・コンテキスト）”が返されます。

非タスク・コンテキスト：　周期ハンドラ，割り込みハンドラ
タスク・コンテキスト：　タスク

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）

sns_loc

概要

CPU ロック状態の参照

C 言語形式

```
BOOL    sns_loc ( void );
```

アセンブリ言語形式

```
CALL    !!_sns_loc
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（CPU ロック状態，非 CPU ロック状態）を獲得します。
なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：CPU ロック状態，FALSE：非 CPU ロック状態）”が返されます。

備考　CPU ロック状態へは loc_cpu，または iloc_cpu を発行することにより，非 CPU ロック状態へは unl_cpu，または iunl_cpu を発行することにより遷移します。

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）

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 );
```

アセンブリ言語形式

```
CALL    !!_sns_dsp
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（ディスパッチ禁止状態，ディスパッチ許可状態）を獲得します。
なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：ディスパッチ禁止状態，FALSE：ディスパッチ許可状態）”が返されます。

備考　ディスパッチ禁止状態へは dis_dsp を発行することにより，ディスパッチ許可状態へは ena_dsp を発行することにより遷移します。

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）

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 );
```

アセンブリ言語形式

```
CALL    !!_sns_dpn
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（ディスパッチ保留状態であるか否か）を獲得します。
なお，本サービス・コールが正常終了した際には，戻り値として“獲得したシステム状態種別（TRUE：ディスパッチ保留状態，FALSE：非ディスパッチ保留状態）”が返されます。

備考　ディスパッチ保留状態とは，dis_dsp，loc_cpu，iloc_cpu といったサービス・コールを発行して明示的にディスパッチ処理（タスクのスケジューリング処理）の実行が禁止された状態，および，非タスクが処理を実行中の状態を指します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ保留状態）
FALSE	0	正常終了（非ディスパッチ保留状態）

12.15 システム構成管理機能

以下に，RI78V4 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 12 － 17　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，非タスク

ref_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 );
```

アセンブリ言語形式

```
MOVW    AX, #LOWW(_pk_rver)
CALL    !!_ref_ver
```

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

機能

RI78V4 のバージョン情報（メーカ・コードなど）を *pk_rver* で指定された領域に格納します。

備考　　バージョン情報 T_RVER についての詳細は，「12.5.9　バージョン情報」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

第 13 章　システム・コンフィギュレーション・ファイル

本章では，RI78V4 に提供するデータを保持した情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル，割り込み情報定義ファイル）を出力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。

13.1　表記方法

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。

- 文字コード
  システム・コンフィギュレーション・ファイルは，ASCII コードで記述します。
  なお，CF78V4 では，英小文字“a ～ z”と英大文字“A ～ Z”の区別が行われます。

  備考　　日本語については，SJIS コードのものをコメント内でのみ記述することができます。

- コメント
  システム・コンフィギュレーション・ファイルでは，/* から */ で囲まれた部分，および，//（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
  システム・コンフィギュレーション・ファイルでは，数字“0 ～ 9”で始まる単語が数値として扱われます。
  なお，CF78V4 では，数値を以下のように区別しています。

  8 進数：　0 で始まる単語
  10 進数：　0 以外の数字で始まる単語
  16 進数：　0x，または 0X で始まる単語

  備考　　単語の構成要素は，数字“0 ～ 9”に限定されます。

- オブジェクト名
  システム・コンフィギュレーション・ファイルでは，英字“a ～ z，A ～ Z”，またはアンダースコア“_”で始まる 24 文字以内の単語がオブジェクト名として扱われます。

  備考　　単語の構成要素は，英数字“a ～ z，A ～ Z，0 ～ 9”，および，アンダースコア“_”に限定されます。

- シンボル名
  システム・コンフィギュレーション・ファイルでは，英字“a ～ z，A ～ Z”，またはアンダースコア“_”で始まる 30 文字以内の単語がシンボル名として扱われます。

  備考 1　単語の構成要素は，英数字“a ～ z，A ～ Z，0 ～ 9”，および，アンダースコア“_”に限定されます。
  備考 2　CF78V4 では，“オブジェクト名”と“シンボル名”をシステム・コンフィギュレーション・ファイルの文脈から区別しています。

- キーワード
  CF78V4 では，以下に示した単語を“キーワード”として予約しています。
  このため，これらの文字列を指定された用途以外に使用することは禁止されています。

  CLK_INTNO，CRE_CYC，CRE_DTQ，CRE_FLG，CRE_MBX，CRE_MPF，CRE_SEM，CRE_TSK，DEF_INH，.kernel_work0，.kernel_work1，.kernel_work2，.kernel_work3，MAX_PRI，null，NULL，SYS_STK，TA_ACT，TA_ASM，TA_CLR，TA_DISINT，TA_ENAINT，TA_FAR，TA_HLNG，TA_MFIFO，TA_MPRI，TA_NEAR，TA_PHS，TA_RSTR，TA_STA，TA_TFIFO，TA_TPRI，TA_WMUL，TA_WSGL

備考　　CF78V4 では，C 言語プリプロセッサの呼び出しを行いません。したがって，システム・コンフィギュレーション・ファイル内で前処理指令（#include，#define，#if など）を記述することは禁止されています。

13.2　コンフィギュレーション情報

システム・コンフィギュレーション・ファイルに記述するコンフィギュレーション情報は，以下に示した 2 種類に大別されます。

- システム情報
  RI78V4 が動作するうえで必要となる基本的なデータから構成されています。
  - スタック情報
  - 優先度情報
  - 基本クロック用タイマ割り込み要因

- 静的 API 情報
  RI78V4 が提供する機能を実現するうえで必要となる管理オブジェクトに対するデータから構成されています。
  - タスク情報
  - セマフォ情報
  - イベントフラグ情報
  - データ・キュー情報
  - メールボックス情報
  - 固定長メモリ・プール情報
  - 周期ハンドラ情報
  - 割り込みハンドラ情報

13.2.1　記述上の注意点

以下に，システム・コンフィギュレーション・ファイルの記述イメージを示します。

図 13 － 1　システム・コンフィギュレーション・ファイルの記述イメージ

```
-- システム情報（スタック情報，優先度情報）の記述
............
............

-- 静的 API 情報（タスク情報，セマフォ情報など）の記述
............
............
```

備考　システム・コンフィギュレーション・ファイルの最大行数は 40000 行，1 行当たりの最大文字数は 1000 文字です。

13.3　システム情報

システム・コンフィギュレーション・ファイルに記述するシステム情報の記述形式を以下に示します。
ただし，表記中のゴシック書体は予約語であることを，イタリック書体はユーザが該当する数値を記述する部分であることを表しています。
また，“[　]”で囲まれた項目は，省略可能な項目であることを表しています。

13.3.1　スタック情報

スタック情報では，

1）システムのスタック・サイズ sys_stksz

といった項目を定義します。
なお，スタック情報として定義可能な数は，1 個に限られます。
以下に，スタック情報の記述形式を示します。

```
SYS_STK ( sys_stksz );
```

以下に，スタック情報で記述する項目について示します。

1）システムのスタック・サイズ *sys_stksz*

システムのスタック・サイズ（単位：バイト）を指定します。
なお，*sys_stksz* として指定可能な値は，“0 ～ 65534 の 2 バイト境界値”に限られます。

備考 1　システム・スタックの割り付け先は，“.kernel_stack セクション”となります。

備考 2　システムのスタック・サイズの見積もりについては，「13.5.1　システムのスタック・サイズ」を参照してください。

13.3.2　優先度情報

優先度情報では，

1）優先度範囲 maxtpri

といった項目を定義します。
なお，優先度情報として定義可能な数は，0 ～ 1 個に限られます。
以下に，優先度情報の記述形式を示します。

```
[MAX_PRI ( maxtpri );]
```

以下に，優先度情報で記述する項目について示します。

1）優先度範囲 *maxtpri*

タスクの優先度範囲（初期優先度 itskpri の最大値，または chg_pri を発行する際に指定する優先度の最大値）を指定します。
なお，*maxtpri* として指定可能な値は，“1 ～ 15”に限られます。

備考　　本情報の定義を省略した場合，タスクの優先度範囲は，“15”となります。

13.3.3　基本クロック用タイマ割り込み要因

基本クロック用タイマ割り込み要因では，

1）基本クロック用タイマ割り込み要因 tim_intno

といった項目を定義します。
なお，基本クロック用タイマ割り込み要因として定義可能な数は 1 個に限られます。
以下に，基本クロック用タイマ割り込み要因の記述形式を示します。

```
CLK_INTNO ( tim_intno );
```

以下に，基本クロック用タイマ割り込み要因で記述する項目について示します。

1）基本クロック用タイマ割り込み要因 *tim_intno*

RI78V4 が提供する時間管理機能を実現する上で必要となる基本クロック用タイマ割り込みの割り込み要因名，またはベクタ・テーブル・アドレスを指定します。
tim_intno として指定可能な値は，デバイス・ファイルで規定されている割り込み要因名，または 0x0 ～ 0x7c に限られます。
なお，*tim_intno* に割り込み要因名を指定した場合，CF78V4 の起動オプションとして -cpu △ <*name*> の指定が必須となります。

13.4　静的 API 情報

システム・コンフィギュレーション・ファイルに記述する静的 API 情報の記述形式を以下に示します。
ただし，表記中のゴシック書体は予約語であることを，イタリック書体はユーザが該当する数値，シンボル名，キーワードを記述する部分であることを表しています。
また，“[　]”で囲まれた項目は，省略可能な項目であることを表しています。

13.4.1　タスク情報

タスク情報では，

1）タスク名 tskid
2）属性（記述言語，初期起動状態，初期割り込み状態）tskatr
3）拡張情報 exinf
4）起動アドレス task
5）初期優先度 itskpri
6）スタック・サイズ stksz
7）システム予約領域 stk

といった項目を個々のタスクに対して定義します。
なお，タスク情報として定義可能な数は，1 ～ 127 個に限られます。
以下に，タスク情報の記述形式を示します。

```
CRE_TSK ( tskid, { tskatr, exinf, task, itskpri, stksz, stk } );
```

以下に，タスク情報で記述する項目について示します。

1）タスク名 *tskid*

タスクの名前を指定します。
なお，*tskid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，タスク名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，タスク名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define tskid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
tskid   .EQU    ID
```

2）属性（記述言語，初期起動状態，初期割り込み状態）*tskatr*

タスクの属性（記述言語，初期起動状態，初期割り込み状態）を指定します。
なお, *tskatr* として指定可能な値は, “TA_HLNG, TA_ASM のいずれか, TA_ACT, および, TA_ENAINT, TA_DISINT のいずれか”に限られます。

【 タスクの記述言語 】
TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

【 タスクの初期起動状態 】
TA_ACT：　READY 状態

【 タスクの初期割り込み状態 】
TA_ENAINT：　マスカブル割り込みの受け付けを許可
TA_DISINT：　マスカブル割り込みの受け付けを禁止

備考 1　TA_ACT の指定を省略した場合，タスクの初期起動状態は，“DORMANT 状態”となります。

備考 2　TA_ENAINT，および，TA_DISINT の指定を省略した場合，初期割り込み状態は“受け付け許可状態”となります。

3） 拡張情報 *exinf*

タスクの拡張情報を指定します。
なお，*exinf* として指定可能な値は，“0 ～ 1048575，またはC 言語で記述する際のシンボル名”に限られます。

備考　*exinf* は，タスクを act_tsk，または iact_tsk で起動した際，起動コードとして対象タスクに引き渡されます。したがって，対象タスクは，*exinf* を関数パラメータと同様に取り扱うことで操作することができます。

4） 起動アドレス *task*

タスクの起動アドレスを指定します。
なお，*task* として指定可能な値は，“C 言語で記述する際のシンボル名”に限られます。

備考 1　以下のようにタスクを C 言語で記述した場合，本項目で指定する値は，“func_task”となります。

```
#include        <kernel.h>
#include        <kernel_id.h>

void
func_task ( VP_INT exinf )
{
        .............
        .............

        ext_tsk ( );
}
```

備考 2　以下のようにタスクをアセンブリ言語で記述した場合，本項目で指定する値は，“func_task”となります。

```
$INCLUDE        (kernel.inc)
$INCLUDE        (kernel_id.inc)

        .PUBLIC  _func_task
        .SECTION .text, TEXT
_func_task:
        PUSH    BC
        PUSH    AX

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

        BR      !!_ext_tsk
```

5） 初期優先度 *itskpri*

タスクの初期優先度を指定します。
なお，*itskpri* として指定可能な値は，“1 ～優先度範囲 maxtpri”に限られます。

6） スタック・サイズ *stksz*

タスクのスタック・サイズ（単位：バイト）を指定します。
なお，*stksz* として指定可能な値は，“0 ～ 65534 の 2 バイト境界値”に限られます。

備考 1　タスク・スタックの割り付け先は，“.kernel_stack セクション”となります。

備考 2　タスクのスタック・サイズの見積もりについては，「13.5.2　タスクのスタック・サイズ」を参照してください。

7） システム予約領域 *stk*

システム予約領域です。
なお，*stk* として指定可能な値は，“NULL”に限られます。

13.4.2　セマフォ情報

セマフォ情報では，

1）セマフォ名 semid
2）属性（キューイング方式）sematr
3）初期資源数 isemcnt
4）システム予約領域 maxsem

といった項目を個々のセマフォに対して定義します。
なお，セマフォ情報として定義可能な数は，0 ～ 127 個に限られます。
以下に，セマフォ情報の記述形式を示します。

```
CRE_SEM ( semid, { sematr, isemcnt, maxsem } );
```

以下に，セマフォ情報で記述する項目について示します。

1）セマフォ名 *semid*

セマフォの名前を指定します。
なお，*semid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，セマフォ名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，セマフォ名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define semid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
semid   .EQU    ID
```

2）属性（キューイング方式）*sematr*

セマフォの属性（キューイング方式）を指定します。
なお，*sematr* として指定可能な値は，“TA_TFIFO”に限られます。

【 タスクのキューイング方式 】
TA_TFIFO：　タスクから wai_sem，または twai_sem を発行した際，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，該当タスクをセマフォの待ちキューに“資源の獲得要求順”でキューイング

3）初期資源数 *isemcnt*

セマフォの初期資源数を指定します。
なお，*isemcnt* として指定可能な値は，“0 ～ 127”に限られます。

4）システム予約領域 *maxsem*

システム予約領域です。
なお，*maxsem* として指定可能な値は，“127”に限られます。

13.4.3　イベントフラグ情報

イベントフラグ情報では，

1）イベントフラグ名 flgid
2）属性（キューイング方式，キューイング数，クリア）flgatr
3）システム予約領域 iflgptn

といった項目を個々のイベントフラグに対して定義します。
なお，イベントフラグ情報として定義可能な数は，0 ～ 127 個に限られます。
以下に，イベントフラグ情報の記述形式を示します。

```
CRE_FLG ( flgid, { flgatr, iflgptn } );
```

以下に，イベントフラグ情報で記述する項目について示します。

1）イベントフラグ名 *flgid*

イベントフラグの名前を指定します。
なお，*flgid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，イベントフラグ名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，イベントフラグ名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define flgid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
flgid   .EQU     ID
```

2）属性（キューイング方式，キューイング数，クリア）*flgatr*

イベントフラグの属性（キューイング方式，キューイング数，クリア）を指定します。
なお，*flgatr* として指定可能な値は，“TA_TFIFO，TA_WSGL，および，TA_CLR”に限られます。

【 タスクのキューイング方式 】
TA_TFIFO：　タスクから wai_flg，または twai_flg を発行した際，要求条件を満足していなかった場合には，該当タスクをイベントフラグの待ちキューにキューイング

【 タスクのキューイング数 】
TA_WSGL：　イベントフラグの待ちキューにキューイング可能なタスク数は 1 個

【 ビット・ターンのクリア 】
TA_CLR：　要求条件を満足した際，ビット・パターンをクリア

備考　TA_CLR の指定を省略した場合，“要求条件を満足した際，ビット・パターンをクリアしない”となります。

3）システム予約領域 *iflgptn*

システム予約領域です。
なお，*iflgptn* として指定可能な値は，“0”に限られます。

13.4.4　データ・キュー情報

データ・キュー情報では，

1）データ・キュー名 dtqid
2）属性（キューイング方式）dtqatr
3）データ数 dtqcnt，メモリ領域名 sec_nam
4）システム予約領域 dtq

といった項目を個々のデータ・キューに対して定義します。
なお，データ・キュー情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，データ・キュー情報の記述形式を示します。

```
CRE_DTQ ( dtqid, { dtqatr, dtqcnt[:sec_nam ], dtq } );
```

以下に，データ・キュー情報で記述する項目について示します。

1）データ・キュー名 *dtqid*

データ・キューの名前を指定します。
なお，*dtqid* として指定可能な値は“オブジェクト名”に限られます。

備考　CF78V4 では，データ・キュー名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，データ・キュー名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define dtqid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
dtqid   .EQU     ID
```

2）属性（キューイング方式）*dtqatr*

データ・キューの属性（キューイング方式）を指定します。
なお，*dtqatr* として指定可能な値は“TA_TFIFO”に限られます。

TA_TFIFO：　データの送信要求を行った順

3）データ数 *dtqcnt*，メモリ領域名 *sec_nam*

データ・キューのデータ・キュー領域にキューイング可能なデータの最大数，および，データ・キュー領域用に確保するメモリ領域の名前を指定します。
なお，*dtqcnt* として指定可能な値は“0x0 ～ 0xff”に，*seg_nam* として指定可能な値は，“kernel_work0, kernel_work1, kernel_work2，kernel_work3 のいずれか”に限られます。

【 データの割り付け先 】
kernel_work0：　データを .kernel_work0 セクションに割り付ける
kernel_work1：　データを .kernel_work1 セクションに割り付ける
kernel_work2：　データを .kernel_work2 セクションに割り付ける
kernel_work3：　データを .kernel_work3 セクションに割り付ける

4）システム予約領域 *dtq*

システム予約領域です。
なお，*dtq* として指定可能な値は“NULL”に限られます。

13.4.5　メールボックス情報

メールボックス情報では，

1）メールボックス名 mbxid
2）属性（キューイング方式）mbxatr
3）システム予約領域 maxmpri
4）システム予約領域 mprihd

といった項目を個々のメールボックスに対して定義します。
なお，メールボックス情報として定義可能な数は，0 ～ 127 個に限られます。
以下に，メールボックス情報の記述形式を示します。

```
CRE_MBX ( mbxid, { mbxatr, maxmpri, mprihd } );
```

以下に，メールボックス情報で記述する項目について示します。

1）メールボックス名 *mbxid*

メールボックスの名前を指定します。
なお，*mbxid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，メールボックス名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，メールボックス名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define mbxid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
mbxid   .EQU    ID
```

2）属性（キューイング方式）*mbxatr*

メールボックスの属性（キューイング方式）を指定します。
なお，*mbxatr* として指定可能な値は，“TA_TFIFO，および，TA_MFIFO，TA_MPRI のいずれか”に限られます。

【 タスクのキューイング方式 】
TA_TFIFO：　タスクから rcv_mbx，または trcv_mbx を発行した際，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，該当タスクをメールボックスの待ちキューに“メッセージの受信要求順”でキューイング

【 メッセージのキューイング方式 】
TA_MFIFO：　処理プログラムから snd_mbx を発行した際，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なかった場合には，該当メッセージをメールボックスの待ちキューに“メッセージの送信要求順”でキューイング
TA_MPRI：　処理プログラムから snd_mbx を発行した際，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なかった場合には，該当メッセージをメールボックスの待ちキューに“メッセージの優先度順”でキューイング

3）システム予約領域 *maxmpri*

システム予約領域です。
なお，*maxmpri* として指定可能な値は，“0”に限られます。

4） システム予約領域 *mprihd*

システム予約領域です。
なお，*mprihd* として指定可能な値は，“NULL”に限られます。

13.4.6　固定長メモリ・プール情報

固定長メモリ・プール情報では，

1）固定長メモリ・プール名 mpfid
2）属性（キューイング方式）mpfatr
3）メモリ・ブロック数 blkcnt
4）ブロック・サイズ blksz
5）セクション名 sec_nam
6）システム予約領域 mpf

といった項目を個々の固定長メモリ・プールに対して定義します。
なお，固定長メモリ・プール情報として定義可能な数は，0 ～ 127 個に限られます。
以下に，固定長メモリ・プール情報の記述形式を示します。

```
CRE_MPF ( mpfid, { mpfatr, blkcnt, blksz[:seg_nam], mpf } );
```

以下に，固定長メモリ・プール情報で記述する項目について示します。

1）固定長メモリ・プール名 *mpfid*

固定長メモリ・プールの名前を指定します。
なお，*mpfid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，固定長メモリ・プール名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，固定長メモリ・プール名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define mpfid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
mpfid   .EQU    ID
```

2）属性（キューイング方式）*mpfatr*

固定長メモリ・プールの属性（キューイング方式）を指定します。
なお，*mpfatr* として指定可能な値は，“TA_TFIFO”に限られます。

【 タスクのキューイング方式 】
TA_TFIFO：　タスクから get_mpf，または tget_mpf を発行した際，メモリ・ブロックを獲得することができなかった（空きメモリ・ブロックが存在しなかった）場合には，該当タスクを固定長メモリ・プールの待ちキューに“メモリ・ブロックの獲得要求順”でキューイング

3）メモリ・ブロック数 *blkcnt*

メモリ・ブロックの総数を指定します。
なお，*blkcnt* として指定可能な値は，“1 ～ 16383”に限られます。

4）ブロック・サイズ *blksz*

1 ブロック当たりのサイズ（単位：バイト）を指定します。
なお，*blksz* として指定可能な値は，“4 ～ 65534 の 2 バイト境界値”に限られます。

5） セクション名 *sec_nam*

固定長メモリ・プールの割り付け先を指定します。
なお，*sec_nam* として指定可能な値は，“kernel_work0，kernel_work1，kernel_work2，kernel_work3 のいずれか”に限られます。

【 固定長メモリ・プールの割り付け先 】
kernel_work0： 固定長メモリ・プールを .kernel_work0 セクションに割り付ける
kernel_work1： 固定長メモリ・プールを .kernel_work1 セクションに割り付ける
kernel_work2： 固定長メモリ・プールを .kernel_work2 セクションに割り付ける
kernel_work3： 固定長メモリ・プールを .kernel_work3 セクションに割り付ける

備考　*sec_nam* の指定を省略した場合，固定長メモリ・プールの割り付け先は，“.kernel_work0 セクション”となります。

6） システム予約領域 *mpf*

システム予約領域です。
なお，*mpf* として指定可能な値は，“NULL”に限られます。

13.4.7　周期ハンドラ情報

周期ハンドラ情報では，

1）周期ハンドラ名 cycid
2）属性（記述言語，初期活性状態）cycatr
3）システム予約領域 exinf
4）起動アドレス cychdr
5）起動周期 cyctim
6）初期起動位相 cycphs

といった項目を個々の周期ハンドラに対して定義します。
なお，周期ハンドラ情報として定義可能な数は，0 ～ 127 個に限られます。
以下に，周期ハンドラ情報の記述形式を示します。

```
CRE_CYC ( cycid, { cycatr, exinf, cychdr, cyctim, cycphs } );
```

以下に，周期ハンドラ情報で記述する項目について示します。

1）周期ハンドラ名 *cycid*

周期ハンドラの名前を指定します。
なお，*cycid* として指定可能な値は，“オブジェクト名”に限られます。

備考　CF78V4 では，周期ハンドラ名と ID の対応を以下に示した形式でシステム情報ヘッダ・ファイルに出力します。このため，処理プログラム内で該当システム情報ヘッダ・ファイルをインクルードすることにより，周期ハンドラ名を ID の代わりに利用することが可能となります。

【 システム情報ヘッダ・ファイル（C 言語用）への出力形式 】

```
#define cycid   ID
```

【 システム情報ヘッダ・ファイル（アセンブリ言語用）への出力形式 】

```
cycid   .EQU     ID
```

2）属性（記述言語，初期活性状態）*cycatr*

周期ハンドラの属性（記述言語，初期活性状態，起動位相保存の有無）を指定します。
なお，*cycatr* として指定可能な値は，“TA_HLNG，TA_ASM のいずれか，および，TA_STA，TA_PHS”に限られます。

【 周期ハンドラの記述言語 】
TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

【 周期ハンドラの初期活性状態 】
TA_STA：　動作開始状態（STA 状態）

【 周期ハンドラの起動位相保存の有無 】
TA_PHS：　起動位相を保存

備考　TA_STA の指定を省略した場合，周期ハンドラの初期活性状態は，“動作停止状態（STP 状態）”となります。

3）システム予約領域 *exinf*

システム予約領域です。
なお，*exinf* として指定可能な値は，“0”に限られます。

4） 起動アドレス *cychdr*

周期ハンドラの起動アドレスを指定します。
なお，*cychdr* として指定可能な値は，“C 言語で記述する際のシンボル名”に限られます。

備考 1　以下のように周期ハンドラを C 言語で記述した場合，本項目で指定する値は，“func_cychdr”となります。

```
#include        <kernel.h>
#include        <kernel_id.h>

void
func_cychdr ( void )
{
        ............
        ............

        return;
}
```

備考 2　以下のように周期ハンドラをアセンブリ言語で記述した場合，本項目で指定する値は，“func_cychdr”となります。

```
$INCLUDE        (kernel.inc)
$INCLUDE        (kernel_id.inc)

        .PUBLIC  _func_cychdr
        .SECTION .text, TEXT
_func_cychdr:
        ............
        ............

        RET
```

5） 起動周期 *cyctim*

周期ハンドラの起動周期（単位：ティック）を指定します。
なお，*cyctim* として指定可能な値は，“1 ～ 4294967295”に限られます。

6） 初期起動位相 *cycphs*

周期ハンドラの初期起動位相（単位：ティック）を指定します。
なお，*cycphs* として指定可能な値は，“0x1 ～ 0x7fffffff”に限られます。

備考 1　RI78V4 における初期起動位相は，周期ハンドラの生成処理が完了してから 1 回目の起動要求が発行されるまでの相対時間間隔を意味しています。

備考 2　*cycphs* に基本情報で定義した基本クロック周期の整数倍値以外の値を指定した場合，CF78V4 は整数倍値が指定されていたものとして処理を行います。

13.4.8　割り込みハンドラ情報

割り込みハンドラ情報では，

1）割り込み要因 inhno

2）属性（記述言語，メモリ配置）inhatr

3）起動アドレス inthdr

といった項目を個々の割り込みハンドラに対して定義します。
なお，割り込みハンドラ情報として定義可能な数は，1 つの割り込み要因に対して 1 個に限られます。
以下に，割り込みハンドラ情報の記述形式を示します。

```
DEF_INH ( inhno, { inhatr, inthdr } );
```

以下に，割り込みハンドラ情報で記述する項目について示します。

1）割り込み要因 *inhno*

割り込みハンドラを登録するマスカブル割り込みの割り込み要因名，またはベクタ・テーブル・アドレスを指定します。
inhno として指定可能な値は，デバイス・ファイルで規定されている割り込み要因名，または 0x0 ～ 0x7c に限られます。
なお，*inhno* に割り込み要因名を指定した場合，CF78V4 の起動オプションとして -cpu △ <*name*> の指定が必須となります。

2）属性（記述言語，メモリ配置）*inhatr*

割り込みハンドラの属性（記述言語，メモリ配置）を指定します。
なお，*inhatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか，および TA_NEAR，TA_FAR のいずれか”に限られます。

【割り込みハンドラの記述言語】

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

【割り込みハンドラのメモリ配置】

TA_NEAR：　NEAR 配置
TA_FAR：　FAR 配置

3）*起動アドレス inthdr*

割り込みハンドラの起動アドレスを指定します。
なお，*inthdr* として指定可能な値は，“C 言語で記述する際のシンボル名”に限られます。

備考 1　以下のように割り込みハンドラを C 言語で記述した場合，本項目で指定する値は，“func_inthdr”となります。

```
#include        <kernel.h>
#include        <kernel_id.h>

void
func_inthdr ( void )
{
        ............
        ............

        return;
}
```

備考 2　以下のように割り込みハンドラをアセンブリ言語で記述した場合，本項目で指定する値は，“func_inthdr”となります。

```
$INCLUDE        (kernel.inc)
$INCLUDE        (kernel_id.inc)

        .PUBLIC  _func_inthdr
        .SECTION .text, TEXT
_func_inthdr:
        ............
        ............

        RET
```

13.5　スタック・サイズの見積もり

13.5.1　システムのスタック・サイズ

システムのスタック・サイズの計算式を以下に示します。

【式 1：システム・スタック・サイズ】
sys_stk =　MAX(sys_stkA, sys_stkB, sys_stkC) ＋ 2 （バイト）

【式 2：システム・スタック・サイズが使用されるパターン A】
sys_stkA = tsksvc ＋ int0 ＋ int1 ＋ int2 ＋ int3

【式 3：システム・スタック・サイズが使用されるパターン B】
sys_stkB = アイドル・ルーチン上のユーザ使用サイズ

【式 4：システム・スタック・サイズが使用されるパターン C】
sys_stkC = 初期化ルーチン上のユーザ使用サイズ

【式 5：タスクで実行したサービス・コール中，最大のシステム・スタック使用サイズ】
タスクで実行したサービス・コール中，最大のシステム・スタック使用サイズ

【式 6：int0，int1 のサイズ】
Intx = レベル x の割り込みの中で，最もスタックが使用される割り込みのサイズ
= 割り込み上のユーザ使用サイズ

【式 7：int2，int3 のサイズ】
intx = レベル x の割り込みの中で，最もスタックが使用される割り込みのサイズ
= 割り込み上のユーザ使用サイズ＋ allsvc ＋ 18

【式 8：割り込み上で使用するサービス・コールの使用サイズ合計】
allsvc = 発行元スタック引数用＋発行元スタック内部処理用＋システム・スタック内部処理用

システム・スタック・サイズは，システム・コンフィギュレーション・ファイルへ指定します。ただし，実際はコンフィギュレータに指定する値＋ 2 バイトのサイズが確保されます。よって，実際にシステム・コンフィギュレーション・ファイルに指定する値は式 1 で求めた sys_stk から 2 バイトを減算したものとなります。

なお，システム・スタック・サイズは，スタックのオーバフローの危険を減らすため，この見積もり結果よりも多めに取ることを推奨します。

以下に，例を示します。

【条件】

- タスク task1 は pol_flg サービス・コールを実行。
- タスク task2 は snd_mbx サービス・コールを実行。
- 割り込み int0 はレベル 0 の OS 管理外の割り込み処理。割り込み上でスタック使用なし。
- 割り込み int2 はレベル 2 の OS 割り込みハンドラ。snd_mbx サービス・コールを実行，割り込み上で 12 バイトのスタック使用。
- 割り込み int3A はレベル 3 の OS 割り込みハンドラ。pol_flg サービス・コールを実行，割り込み上で 16 バイトのスタック使用。
- 割り込み int3B はレベル 3 の OS 割り込みハンドラ。Timer_Handler を実行，割り込み上でスタック使用なし。
- アイドル idl はスタック使用なし。
- 初期化ルーチン ini は初期化ルーチン上で 24 バイトのスタック使用。

【計算式】

tsksvc　= MAX(pol_flg のシステム・スタック使用サイズ，snd_mbx のシステム・スタック使用サイズ)
　　= MAX(16,8) = 16 バイト

int0 = 0 ＋ 0 = 0 バイト

int1 = 定義なし = 0 バイト

int2 = 12 ＋ (0 ＋ 6 ＋ 4) ＋ 18 = 40 バイト

int3 = MAX(int3A, int3B) = MAX(56,32) = 56 バイト

int3A = 16 ＋ (0 ＋ 6 ＋ 16) ＋ 18 = 56 バイト

int3B = 0 ＋ (0 ＋ 0 ＋ 14) ＋ 18 = 32 バイト

sys_stkA　= tsksvc ＋ int0 ＋ int1 ＋ int2 ＋ int3
　　= 16 ＋ 0 ＋ 0 ＋ 40 ＋ 56
　　= 112 バイト ※ sys_stkA,B,C 内で最大なので，このサイズを確保

sys_stkB　= アイドル・ルーチンのユーザ使用スタック・サイズ = 0 バイト

sys_stkC　= 初期化ルーチンのユーザ使用スタック・サイズ = 20 バイト

sys_stk　= MAX(sys_stkA, sys_stkB, sys_stkC) ＋ 2
　　= MAX(112, 0, 20)
　　= 112 ＋ 2 = 114 バイト

以上から，システム・スタック・サイズは sys_stkA の 112 バイトです。
システム・コンフィギュレーション・ファイルへ指定するサイズは，この 112 バイトとなります。

備考　以下に，例で使用するサービス・コール／関数のスタック使用サイズを示します。

	発行元スタック 引数用	発行元スタック 内部処理用	システム・スタック 内部処理用
pol_flg サービス・コール	0	6	16
twai_flg サービス・コール	4	6	16
snd_mbx サービス・コール	0	6	4
Timer_Handler 関数	0	－	14

13.5.2　タスクのスタック・サイズ

タスクのスタック・サイズの計算式を以下に示します。

【式 1：タスク上で割り込みが発生しない場合】
　タスク・スタック・サイズ＝　ユーザ使用サイズ＋サービス・コールの引数用サイズ＋ 6 （バイト）

【式 2：タスク上で割り込みが発生する場合】
　タスク・スタック・サイズ＝　ユーザ使用サイズ＋サービス・コールの引数用サイズ＋ 6 ＋ 20 （バイト）

タスク・スタック・サイズはシステム・コンフィギュレーション・ファイルへ指定します。ただし，実際は“コンフィギュレータに指定した値＋ 6 バイト”のサイズが確保されます。よって，実際にシステム・コンフィギュレーション・ファイルに指定する値は，式 1 または式 2 で求めた値から 6 バイトを減算したものとなります。

この 6 バイトには，サービス・コール発行時に使用されるスタック・サイズが含まれています。ただし，サービス・コール発行時に使用されるスタック・サイズの中でも，引数用スタック・サイズは 6 バイトとは別にユーザ使用サイズとして確保する必要があります。なお，各サービス・コールで使用される引数用スタック・サイズは異なります。その一覧表は表 12 － 1 にまとめてあります。

タスク・スタック・サイズは“対象タスク上で最もスタックが使用される場合のスタック・サイズ”ですので，サービス・コールで，引数用スタックが 4 バイトのものと，8 バイトのものがあれば，よりスタックが使用されるパターンの 8 バイト分を確保します。

以上は，割り込みを受け付けない（すべて割り込み禁止の）タスクに関するものです。割り込みを受け付けるタスクに関しては，さらに 20 バイトの領域を確保する必要があります。

なお，この 20 バイトの中には，割り込み開始時に呼び出しが必要な関数 _kernel_int_entry 呼び出し時のスタック・サイズも含まれています。_kernel_int_entry は 20 バイトのデータをスタックへ退避しただけで復帰は行いません。復帰は割り込み終了時に呼び出しが必要な関数 _kernel_int_exit 呼び出し時に行われます。

例 1　タスク task1 で twai_flg サービス・コールと snd_mbx サービス・コールを使用し，ほかにスタックを使用する関数や処理がなく，タスク上で割り込みは受け付けられない場合

task1 上で割り込みは受け付けられないので，使用する計算式は式 1 です。
スタックを使用する関数や処理はないことから，ユーザ使用サイズは 0 バイトです。
全サービス・コールの引数用サイズを調査すると，以下となります。

サービス・コールの引数用サイズ（twai_flg）= 4 バイト
サービス・コールの引数用サイズ（snd_mbx）= 0 バイト

最もスタックが使用されるのは，twai_flg 呼び出し時のスタック・サイズなので，この値を式 1 へ指定します。

タスク・スタック・サイズ　= ユーザ使用サイズ＋サービス・コールの引数用サイズ（twai_flg）＋ 6
= 0 ＋ 4 ＋ 6
= 10 バイト

システム･コンフィギュレーション･ファイルへ指定するサイズは，上記から 6 を減算した 4 バイトとなります。

例 2　タスク task1 で関数 A（スタックを 12 バイト使う）上から twai_flg サービス・コールを呼び出し，関数 B（スタックを 20 バイト使う）上から snd_mbx サービス・コールを呼び出し，タスク上で割り込みが受け付けられる場合

task1 上で割り込みは受け付けられるので，使用する計算式は式 2 です。最もスタックが使用されるパターンを見つけるため，パターンを列挙します。

パターン A　= ユーザ使用サイズ（関数 A の場合）＋サービス・コールの引数用サイズ（twai_flg）＋ 6 ＋ 20
= 12 ＋ 4 ＋ 6 ＋ 20
= 42 バイト

パターン B　= ユーザ使用サイズ（関数 B の場合）＋サービス・コールの引数用サイズ（snd_mbx）＋ 6 ＋ 20
= 20 ＋ 0 ＋ 6 ＋ 20
= 26 バイト

以上のパターン A とパターン B を比較し，最もスタックが使用されるものはパターン A の 42 バイトです。
システム･コンフィギュレーション･ファイルへ指定するサイズは，上記から 6 を減算した 36 バイトとなります。

13.6　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。

図 13 － 2　システム・コンフィギュレーション・ファイルの記述例

```
-- システム情報の記述
SYS_STK ( 256 );
MAX_PRI ( 15 );
CLK_INTNO( INTTM00 );

-- 静的 API 情報の記述
CRE_TSK ( ID_tsk, { TA_HLNG | TA_ACT | TA_DISINT, 0xa, func_task, 1 256, NULL } );
CRE_TSK ( ID_tskA, { TA_HLNG | TA_ACT, 0x14, func_taskA, 2, 256, NULL } );
CRE_TSK ( ID_tskB, { TA_ASM | TA_ENAINT, 0x1e, func_taskB, 3, 512, NULL } );

CRE_SEM ( ID_semA, { TA_TFIFO, 0, 127 } );
CRE_SEM ( ID_semB, { TA_TFIFO, 127, 127 } );

CRE_FLG ( ID_flgA, { TA_TFIFO | TA_WSGL | TA_CLR, 0 } );
CRE_FLG ( ID_flgB, { TA_TFIFO | TA_WSGL, 0 } );

CRE_DTQ ( ID_DTQ1, { TA_TFIFO,20:kernel_work1, NULL } );

CRE_MBX ( ID_mbxA, { TA_TFIFO | TA_MFIFO, 0, NULL } );
CRE_MBX ( ID_mbxB, { TA_TFIFO | TA_MPRI, 0, MULL } );

CRE_MPF ( ID_mpfA, { TA_TFIFO, 10, 8:kernel_work1, NULL } );
CRE_MPF ( ID_mpfB, { TA_TFIFO, 8, 16, NULL } );

CRE_CYC ( ID_cycA, { TA_HLNG | TA_STA | TA_PHS, 0, func_cychdrA, 1, 0x50 } );
CRE_CYC ( ID_cycB, { TA_ASM, 0, func_cychdrB, 2, 0x100 } );

DEF_INH ( INTP0, { TA_HLNG | TA_FAR, inthdr0 } );
DEF_INH ( INTP1, { TA_HLNG | TA_NEAR, inthdr1 } );
```

第 14 章　コンフィギュレータ CF78V4

本章では, RI78V4 がシステム構築時に有益なユーティリティ･ツールとして提供しているコンフィギュレータ CF78V4 について解説しています。

14.1　概　　要

RI78V4 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，RI78V4 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内容となっているため，記述に際しては，時間と労力が必要となります。

そこで，RI78V4 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を生成するユーティリティ・ツール（コンフィギュレータ CF78V4）を提供しています。

CF78V4 は，入力ファイルとしてシステム・コンフィギュレーション・ファイルを読み込み，出力ファイルとして情報ファイルを出力します。

以下に，CF78V4 が出力する情報ファイルについて示します。

- システム情報テーブル・ファイル
  RI78V4 が動作するうえで必要となる OS 資源に関するデータを保持した情報ファイルです。

- システム情報ヘッダ・ファイル
  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。
  なお，CF78V4 では，C 言語用，アセンブリ言語用の 2 種類のシステム情報ヘッダ・ファイルを出力させることができます。

- 割り込み情報定義ファイル
  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割り込みハンドラに関する情報を保持したファイルです。

14.2　起動方法

14.2.1　コマンド・ラインからの起動

以下に，CF78V4 をコマンド・ラインから起動する際の方法を示します。
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，“[　]”で囲まれた起動オプションは，省略可能な起動オプションであることを表しています。

```
C>  cf78v4.exe △ [ @command file ] △ [ -cpu △ <name> ] △ [ -devpath=path ] △ [ -i △ <SIT file> | -ni ] △ [ -dc △
      <C header file> | -ndc ] △ [ -da △ <ASM header file>  | -nda ] △ [ -V ] △ [ -help ] △ <CF file>  ［Enter］
```

以下に，各起動オプションの詳細を示します。

- @command file
  CF78V4 への入力ファイル名“コマンド・ファイル”を指定します。

  省略時　コマンド・ライン上で指定された起動オプションが有効となります。

  備考 1　入力ファイル名 command file として指定可能な文字列は，255 文字以内（パス名を含む）に限られます。
  備考 2　コマンド・ファイルについての詳細は，「14.2.3　コマンド・ファイル」を参照してください。

- -cpu △ <*name*>
  ターゲット・デバイスの品種指定名を指定します。

  省略時　基本クロック用タイマ割り込み要因 tim_intno や割り込みハンドラの割り込み要因 inhno に“デバイス・ファイルで規定されている割り込み要因名”を用いた定義が行えなくなります。

- -devpath=*path*
  -cpu △ <*name*> で指定されたターゲット・デバイスに対応したデバイス・ファイルを *path* フォルダから検索します。

  省略時　カレント・フォルダに対して検索処理を行います。

- -i △ <SIT file>
  CF78V4 からの出力ファイル名“システム情報テーブル・ファイル”を指定します。

  省略時　-i △ sit.asm が指定されていたものとして処理を行います。

  備考　　出力ファイル名 <SIT file> として指定可能な文字列は，255 文字以内（パス名を含む）に限られます。

- -ni
  システム情報テーブル・ファイルの出力を抑制します。

  省略時　-i △ sit.asm が指定されていたものとして処理を行います。

- -dc △ <C header file>
  CF78V4 からの出力ファイル名“システム情報ヘッダ・ファイル（C 言語用）”を指定します。

  省略時　-dc △ kernel_id.h が指定されていたものとして処理を行います。

  備考　　出力ファイル名 <C header file> として指定可能な文字列は，255 文字以内（パス名を含む）に限られます。

- -ndc
  システム情報ヘッダ・ファイル（C 言語用）の出力を抑制します。

  省略時　-dc △ kernel_id.h が指定されていたものとして処理を行います。

- -da △ <ASM header file>
  CF78V4 からの出力ファイル名“システム情報ヘッダ・ファイル（アセンブリ言語用）”を指定します。

  省略時　-da △ kernel_id.inc が指定されていたものとして処理を行います。

  備考　　出力ファイル名 <ASM header file> として指定可能な文字数は, 255 文字以内（パス名を含む）に限られます。

- -nda
  システム情報ヘッダ・ファイル（アセンブリ言語用）の出力を抑制します。

  省略時　-da △ kernel_id.inc が指定されていたものとして処理を行います。

- -V
  CF78V4 のバージョン情報を標準出力に出力します。

  備考　　CF78V4 では，本起動オプションが指定された場合，“他の起動オプションは，無効な起動オプション”として扱い，情報ファイルの出力を抑制します。

- -help
  CF78V4 の起動オプションに関する情報（種類，用途など）を標準出力に出力します。

  備考　　CF78V4 では，本起動オプションが指定された場合，“他の起動オプションは，無効な起動オプション”として扱い，情報ファイルの出力を抑制します。

- <CF file>
  CF78V4 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”を指定します。

  備考 1　入力ファイル名 <cf file> として指定可能な文字列は，255 文字以内（パス名を含む）に限られます。
  備考 2　本入力ファイル名は，-V，または -help を指定した場合に限り省略することができます。

14.2.2　CS+ 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，CS+ のビルド時に起動されます。

14.2.3 コマンド・ファイル

CF78V4 では，コマンド・ライン上で指定可能な起動オプションの文字数制限を解消する目的からコマンド・ファイル対応を行っています。
以下に，コマンド・ファイルの記述形式を示します。

1） コメント行
行頭に # が記述された行については，コメント行として扱われます。

2） 起動オプションの区切り
起動オプションと起動オプションは，スペース・コード，タブ・コード，または改行コードで区切ります。

備考 “-i △ <SIT file>”，“-dc △ <C header file>”，“-da △ <ASM header file>”といった *-xxx* 部とパラメータ部から構成される起動オプションでは，*-xxx* 部とパラメータ部をスペース・コード，タブ，コード，または改行コードで区切ります。
なお，パラメータ部で“空白文字を含むフォルダ名”を指定する際には，図 14 － 1 で示したようにパラメータ部を“”で括ります。

3） 文字数制限
コマンド・ファイルの最大行数は 50 行，1 行当たりの最大文字数は 4096 文字です。

以下に示した記述例では，“システム・コンフィギュレーション・ファイル CF_file.cfg をカレント・フォルダから読み込んだのち，システム情報テーブル・ファイル sit_file.asm をフォルダ C:¥Program Files¥tmp に出力，システム情報ヘッダ・ファイル（C 言語用）C_header.h をフォルダ C:¥tmp に出力，システム情報ヘッダ・ファイル（アセンブリ言語用）ASM_header.inc をフォルダ C:¥tmp に出力”といった場合の起動オプションが記述されています。

図 14 － 1 コマンド・ファイルの記述例

```
# Command File
-i "C:¥Program Files¥tmp¥sit_file.asm"
-dc C:¥tmp¥C_header.h
-da
"C:¥tmp¥ASM_header.inc"
CF_file.cfg
```

14.2.4　コマンド入力例

以下に，CF78V4 のコマンド入力例を示します。
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。

1） コマンド・ファイル cmd_file をカレントフォルダから読み込んだのち，cmd_file 内に定義されている起動オプションを実行します。

```
C> cf78v4.exe △ @cmd_file ［Enter］
```

2） システム・コンフィギュレーション・ファイル CF_file.cfg をカレント・フォルダから読み込んだのち，システム情報テーブル・ファイル sit_file.asm，システム情報ヘッダ・ファイル（C 言語用）C_header.h，システム情報ヘッダ・ファイル（アセンブリ言語用）ASM_header.inc をカレント・フォルダに出力します
（指定デバイス型名 R5F10A6A，デバイスファイルパスは
”C:¥Program Files¥Renesas Electronics¥CS+¥CC¥Device¥RL78¥Devicefile”）。

```
C> cf78v4.exe △ -cpu △ R5F10A6A △
     -devpath=”C:¥Program Files¥Renesas Electronics¥CS+¥CC¥Device¥RL78¥Devicefile” △
     -i △ sit_file.asm △ -dc △ C_header.h △ -da △ ASM_header.inc △ CF_file.cfg ［Enter］
```

3） システム・コンフィギュレーション・ファイル CF_file.cfg をカレント・フォルダから読み込んだのち，システム情報テーブル・ファイル sit.asm，システム情報ヘッダ・ファイル（C 言語用）kernel_id.h，システム情報ヘッダ・ファイル（アセンブリ言語用）kernel_id.inc をカレント・フォルダに出力します。

```
C> cf78v4.exe △ CF_file.cfg ［Enter］
```

4） システム・コンフィギュレーション・ファイル CF_file.cfg をフォルダ C:¥tmp から読み込んだのち，システム情報テーブル・ファイル sit_file.asm，システム情報ヘッダ・ファイル（C 言語用）C_header.h をフォルダ C:¥tmp に出力します。

```
C> cf78v4.exe △ -i △ C:¥tmp¥sit_file.asm △ -dc △ C:¥tmp¥C_header.h △ -nda △ C:¥tmp¥CF_file.cfg［Enter］
```

5） システム・コンフィギュレーション・ファイル CF_file.cfg をフォルダ C:¥tmp から読み込んだのち，システム情報テーブル・ファイル sit_file.asm をフォルダ C:¥Program Files¥tmp に出力します。

```
C> cf78v4.exe △ -i △ ”C:¥Program Files¥tmp¥sit_file.asm" △ -ndc △ -nda △ C:¥tmp¥CF_file.cfg ［Enter］
```

6） CF78V4 のバージョン情報を標準出力に出力します。

```
C> cf78v4.exe △ -V ［Enter］
```

7） CF78V4 の起動オプションに関する情報（種類，用途など）を標準出力に出力します。

```
C> cf78v4.exe △ -help ［Enter］
```

付録 A　ウインドウ・リファレンス

本付録では，CF78V4 の起動オプションを統合開発環境プラットホーム CS+ 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／パネル一覧

ウインドウ／パネル名	機能概要
メイン・ウインドウ	CS+ 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

プロジェクト・ツリー パネル

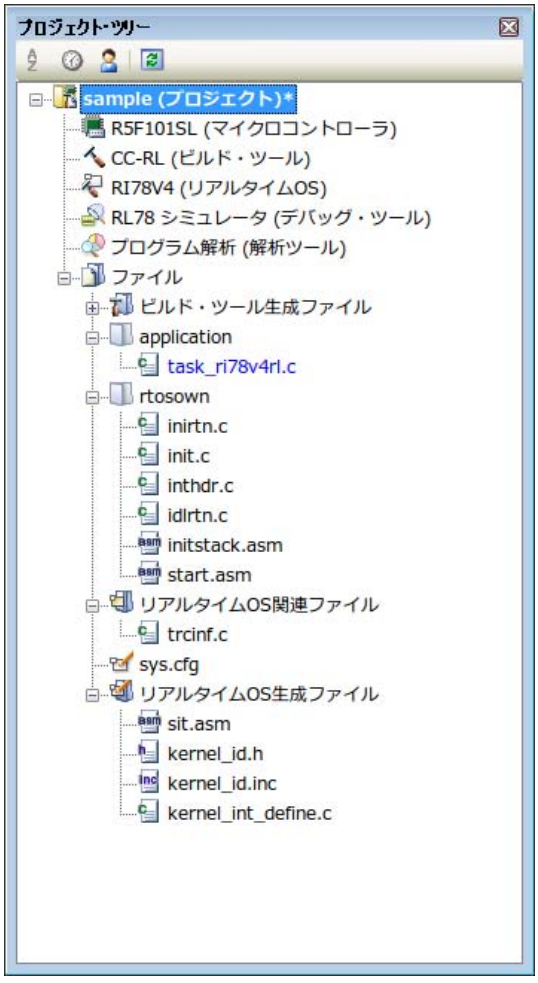
概要

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
RI78V4(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。 - システム情報テーブル・ファイル（.asm） - システム情報ヘッダ・ファイル（C 言語用）（.h） - システム情報ヘッダ・ファイル（アセンブリ言語用）（.inc） - 割り込み情報定義ファイル 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。
リアルタイムOS関連ファイル（“リアルタイムOS関連ファイル・ノード”と呼びます。）	以下の情報ファイルが直下に表示されます。 - トレース情報ファイル（*trcinf.c*） 本ノード，および本ノードに表示されているファイルを削除することはできません。

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。

メイン・ウインドウ

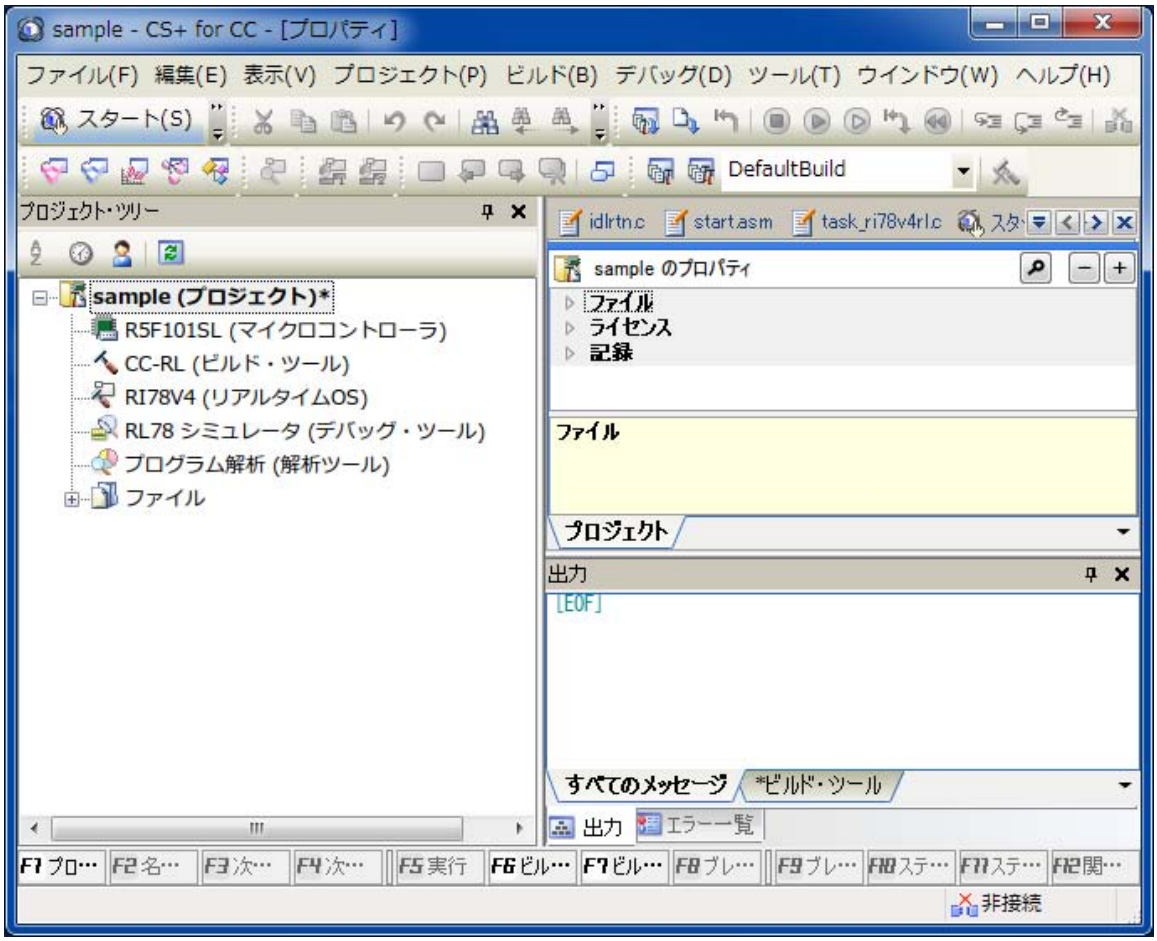
概要

CS+ 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 CS+］→［CS+］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	タスク・アナライザ ウインドウをオープンします。 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。

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 リソース情報ツールバー

	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。

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，「CS+ 統合開発環境 ユーザーズマニュアル RL78 ビルド編」を参照してください。

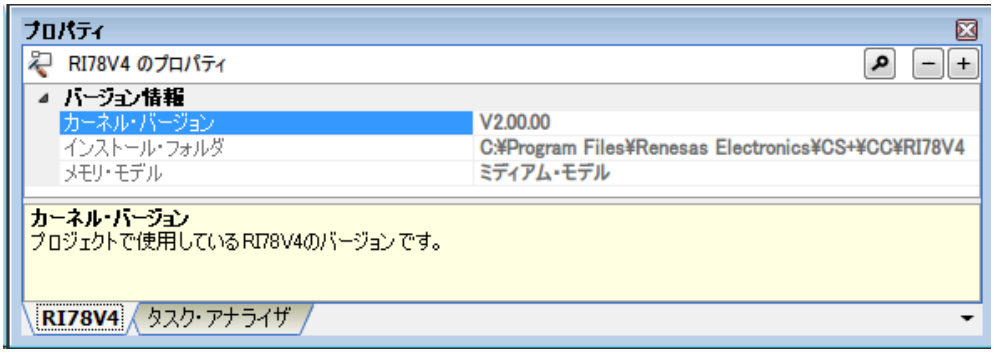
［RI78V4］タブ

概要

本タブでは，使用する RI78V4 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］

RI78V4 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 RI78V4 のバージョンを表示します。 なお，バージョンはプロジェクト作成時に確定するため，変更することはできません。	
	デフォルト	*使用する RI78V4 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 RI78V4 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 RI78V4 がインストールされているフォルダ*
	変更方法	変更不可
メモリ・モデル	プロジェクトで設定しているメモリ・モデルを表示します。 ビルド・ツールの［メモリ・モデルの種類］プロパティで選択しているメモリ・モデル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メモリ・モデル*
	変更方法	変更不可

プロパティ パネル

概要

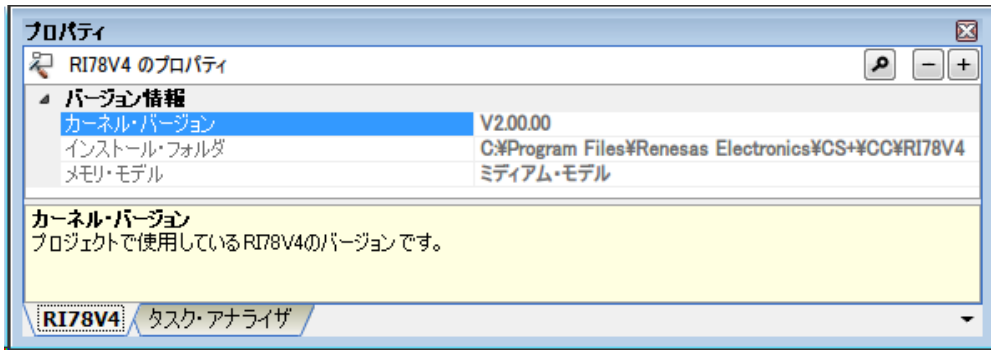
プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等について，カテゴリ別に詳細情報の表示，および設定の変更を行います。

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
  - ［RI78V4］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
- プロジェクト・ツリー パネルで割り込み情報定義ファイルを選択している場合
  - ［ファイル情報］タブ
- プロジェクト・ツリー パネルでトレース情報ファイル *trcinf.c* 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，「CS+ 統合開発環境 ユーザーズマニュアル RL78 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

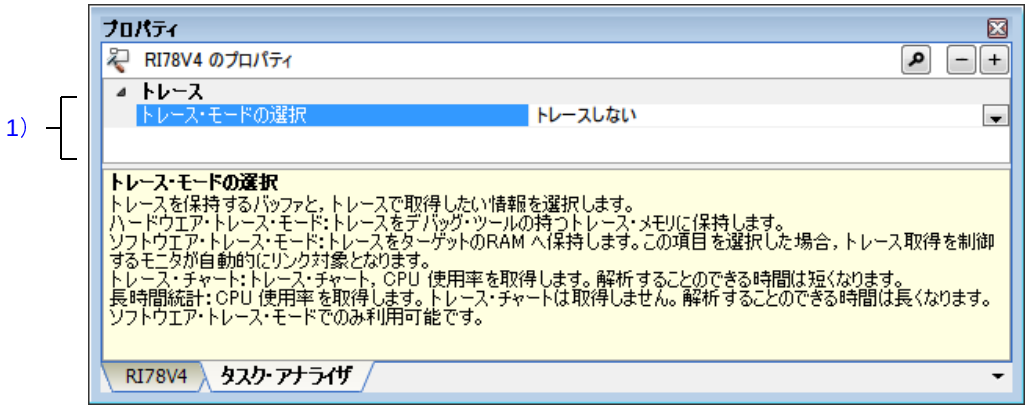
元に戻す	直前に行ったプロパティの値の編集作業を取り消します。

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。

［タスク・アナライザ］タブ

本タブでは，RI78V4 が提供しているユーティリティ・ツール“タスク・アナライザ・ツール”を利用して処理プログラムの実行履歴（トレース・データ）を解析する際に必要となる各種情報の表示，および設定の変更を行います。

図 A－1 ［タスク・アナライザ］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- 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。

［各エリアの説明］

1）［トレース］カテゴリ

ユーティリティ・ツール“タスク・アナライザ・ツール”を利用して処理プログラムの実行履歴（トレース・データ）を解析する際に必要となる各種情報の表示，および設定の変更を行います。

トレース・モードの選択	トレース・データとして取得する情報の種類，およびトレース・データの格納先を選択します。	
	デフォルト	トレースしない
	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	タスク・アナライザ・ツールを利用しないことになります。
		ハードウエア・トレース・モードで，トレース・チャートを取得	トレース・データとして，トレース・チャート（処理プログラムの実行遷移状況，リアルタイムOS資源の利用状況など），およびCPU使用率を取得します。 なお，トレース・バッファは，デバッグ・ツールが用意しているトレース・メモリから確保されます。
		ソフトウエア・トレース・モードで，トレース・チャートを取得	トレース・データとして，トレース・チャート（処理プログラムの実行遷移状況，リアルタイムOS資源の利用状況など），およびCPU使用率を取得します。 なお，トレース・バッファは，［バッファを選択する］で選択された領域から確保されます。
		ソフトウエア・トレース・モードで，長時間統計を取得	トレース・データとして，CPU使用率を取得します。 なお，トレース・バッファは，規定セクション.kernel_data_trace_nから確保されます。
バッファを使い切った後の動作	トレース・バッファを使い切った際の動作を選択します。 なお，本項目は，［トレース・モードの選択］で“ソフトウエア・トレース・モードで，トレース・チャートを取得”を選択した場合に限り表示されます。		
	デフォルト	バッファを上書きし実行し続ける	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	バッファを上書きし実行し続ける	書き込まれてから最も時間が経過しているトレース・データを上書きします。
		トレースを停止する	トレース・バッファへの書き込みを中止します。
バッファ・サイズ	トレース・バッファのサイズ（単位：バイト）を指定します。 なお，本項目は，［トレース・モードの選択］で“ソフトウエア・トレース・モードで，トレース・チャートを取得”を選択した場合に限り表示されます。		
	デフォルト	0x100	
	変更方法	テキスト・ボックスによる直接入力	
	指定可能値	0xa ～ 0xfffe	
バッファを選択する	トレース・データの格納先を選択します。 なお，本項目は，［トレース・モードの選択］で“ソフトウエア・トレース・モードで，トレース・チャートを取得”を選択した場合に限り表示されます。		
	デフォルト	カーネルのバッファ	
	変更方法	ドロップダウン・リストによる選択	
	指摘可能値	カーネルのバッファ	トレース・バッファは，規定セクション.kernel_data_trace_n から確保されます。
		その他のバッファ	トレース・バッファは，［バッファ・アドレス］で指定されたアドレスから確保されます。

バッファ・アドレス	トレース・バッファ用に確保する領域の先頭アドレスを指定します。 なお，本項目は，［バッファを選択する］で“その他のバッファ”を選択した場合に限り表示されます。		
	デフォルト	0xf0000	
	変更方法	テキスト・ボックスによる直接入力	
	指定可能値	0xf0000 ～ 0xffff4	
基本クロック用タイマ割り込みをトレースする	基本クロック用タイマ割り込みをトレースするかどうかを指定します。 なお，本項目は，［トレース・モードの選択］で”ハードウエア・トレース・モードで，トレース・チャートを取得”または“ソフトウエア・トレース・モードで，トレース・チャートを取得”を選択した場合に限り表示されます。		
	デフォルト	いいえ	
	変更方法	ドロップダウン・リストによる選択	
	指摘可能値	はい	基本クロック用タイマ割り込みをトレースします。
		いいえ	基本クロック用タイマ割り込みをトレースしません。

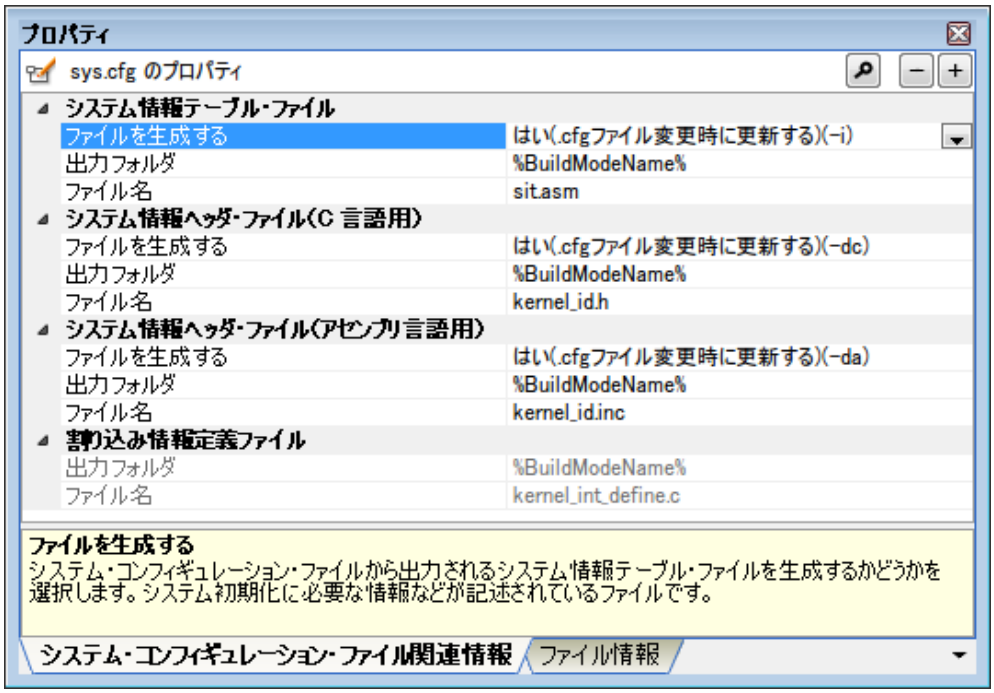
［システム・コンフィギュレーション・ファイル関連情報］タブ

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。

- システム情報テーブル・ファイル
- システム情報ヘッダ・ファイル（C 言語用）
- システム情報ヘッダ・ファイル（アセンブリ言語用）
- 割り込み情報定義ファイル

表示イメージ



機能

1）［システム情報テーブル・ファイル］

システム情報テーブル・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。

<table>
<tr><td rowspan="6">ファイルを生成する</td><td colspan="3">システム情報テーブル・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システム情報テーブル・ファイルを更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-i)</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="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-i)</td><td>システム情報テーブル・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。
システム情報テーブル・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，システム情報テーブル・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-ni)</td><td>システム・コンフィギュレーション・ファイルを変更しても，システム情報テーブル・ファイルを更新しません。
システム情報テーブル・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-ni)</td><td>システム情報テーブル・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。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="3">システム情報テーブル・ファイルを出力するフォルダを指定します。
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）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName% : ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ni)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">%BuildModeName%</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，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td colspan="2">247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="3">システム情報テーブル・ファイル名を指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.asm”を指定してください。拡張子が異なる場合や省略した場合は，“.asm”が自動的に付加されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ni)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">sit.asm</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</td></tr>
<tr><td>指定可能値</td><td colspan="2">259 文字までの文字列</td></tr>
</table>

2）［システム情報ヘッダ・ファイル（C 言語用）］

システム情報ヘッダ・ファイル（C 言語用）に関する詳細情報の表示，および設定の変更を行います。

<table>
<tr><td rowspan="6">ファイルを生成する</td><td colspan="3">システム情報ヘッダ・ファイル（C 言語用）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システム情報ヘッダ・ファイル（C 言語用）を更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-dc)</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="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-dc)</td><td>システム情報ヘッダ・ファイルを生成し，プロジェクト・ツリーに表示します。
システム情報ヘッダ・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，システム情報ヘッダ・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-ndc)</td><td>システム・コンフィギュレーション・ファイルを変更しても，システム情報ヘッダ・ファイルを更新しません。
システム情報ヘッダ・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-ndc)</td><td>システム情報ヘッダ・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。
システム情報ヘッダ・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。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="2">システム情報ヘッダ・ファイル（C 言語用）を出力するフォルダを指定します。
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）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName%：ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ndc)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td>%BuildModeName%</td></tr>
<tr><td>変更方法</td><td>テキスト・ボックスによる直接入力，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td>247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="2">システム情報ヘッダ・ファイル名（C 言語用）を指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.h”を指定してください。拡張子が異なる場合や省略した場合は，“.h”が自動的に付加されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ndc)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td>kernel_id.h</td></tr>
<tr><td>変更方法</td><td>テキスト・ボックスによる直接入力</td></tr>
<tr><td>指定可能値</td><td>259 文字までの文字列</td></tr>
</table>

3）［システム情報ヘッダ・ファイル（アセンブリ言語用）］

システム情報ヘッダ・ファイル（アセンブリ言語用）に関する詳細情報の表示，および設定の変更を行います。

<table>
<tr><td rowspan="6">ファイルを生成する</td><td colspan="3">システム情報ヘッダ・ファイル（アセンブリ言語用）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システム情報ヘッダ・ファイル（アセンブリ言語用）を更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-da)</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="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-da)</td><td>システム情報ヘッダ・ファイルを生成し，プロジェクト・ツリーに表示します。
システム情報ヘッダ・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，システム情報ヘッダ・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-nda)</td><td>システム・コンフィギュレーション・ファイルを変更しても，システム情報ヘッダ・ファイルを更新しません。
システム情報ヘッダ・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-nda)</td><td>システム情報ヘッダ・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。
システム情報ヘッダ・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。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="3">システム情報ヘッダ・ファイル（アセンブリ言語用）を出力するフォルダを指定します。
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）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName% : ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-nda)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">%BuildModeName%</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，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td colspan="2">247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="3">システム情報ヘッダ・ファイル名（アセンブリ言語用）を指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.inc”を指定してください。拡張子が異なる場合や省略した場合は，“.inc”が自動的に付加されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-nda)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">kernel_id.inc</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</td></tr>
<tr><td>指定可能値</td><td colspan="2">259 文字までの文字列</td></tr>
</table>

4) ［割り込み情報定義ファイル］

割り込み情報定義ファイルに関する詳細情報の表示を行います。

出力フォルダ	割り込み情報定義ファイルを出力するフォルダを表示します。 なお，出力フォルダは %BuildModeName% 固定のため，変更することはできません。	
	デフォルト	%BuildModeName%
	変更方法	変更不可
ファイル名	割り込み情報定義ファイル名を表示します。 なお，ファイル名は .kernel_int_define.c 固定のため，変更することはできません。	
	デフォルト	.kernel_int_define.c
	変更方法	変更不可

付録 B 注意事項

B.1 コンパイル・オプションの制限

RI78V4 を組み込んだシステムでは，以下のコンパイル・オプションを指定してください。

オプション	意味
-base_number=prefix	アセンブリ言語の 16 進数の基数表現形式として Prefix 表現形式（0xn..n）を指定。

B.2 レジスタ・バンクの扱い

RI78V4 を組み込んだシステムは，基本的にレジスタ・バンク 0 で動作させてください。
レジスタ・バンクの変更が必要な場合，以下の仕様に基づいて変更してください。ルーチンによってレジスタ・バンクの変更を許可しているルーチン，禁止しているルーチンがあります。

【レジスタ・バンク変更を許可しているルーチン】

- タスク
  タスクでは，初期レジスタ・バンク番号は 0 固定です。
  RI78V4 では，タスク切り替え時に，レジスタ・バンク番号と 1 バンク分（タスク切り替え時のバンク）の汎用レジスタのみ退避／復帰されます。
  残り 3 バンク分の汎用レジスタは退避／復帰されないため，タスク処理上で 2 つ以上のレジスタ・バンクを使用する場合には，レジスタ・バンク変更時に変更前のレジスタ・バンクの汎用レジスタを退避する必要があります。この退避を行わない場合，タスク切り替え先でレジスタ・バンクが破壊される可能性があります。

- OS 管理外の割り込み処理
  OS 管理外の割り込み処理でレジスタ・バンク変更を行う場合，割り込み終了時には割り込み発生元のレジスタ・バンク番号へ戻してください。

【レジスタ・バンク変更を禁止しているルーチン】

- 割り込みハンドラ
  割り込みハンドラでは，レジスタ・バンク番号は割り込み発生元の値を継承します。

- 周期ハンドラ
  周期ハンドラでは，レジスタ・バンク番号はタイマ・ハンドラ用割り込み発生元の値を継承します。

- アイドル・ルーチン
  アイドル・ルーチンでは，初期レジスタ・バンク番号は 0 固定です。

- 初期化ルーチン
  初期化ルーチンでは，初期レジスタ・バンク番号は 0 固定です。OS 初期化前（_kernel_start 関数呼び出し前）に設定されたレジスタ・バンクに無関係にレジスタ・バンク 0 に上書きされます。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2015. 3.25	―	初版発行

RI78V4 V2.00.00　ユーザーズマニュアル
コーディング編

発行年月日　2015年3月25日　Rev.1.00

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 エレクトロニクス株式会社 〒100-0004 千代田区大手町2-6-2（日本ビル）

■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/

RI78V4 V2.00.00

ルネサス エレクトロニクス株式会社

R20UT3375JJ0100