お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

ユーザーズ・マニュアル

RX850

リアルタイム・オペレーティング・システム

基礎編

対象デバイス
V850ファミリ™
対象リアルタイムOS
RX850 Ver.3.13以上

資料番号 U13430JJ2V1UM00（第2版）
発行年月 October 2001 N CP(K)

〔メ　モ〕

目次要約

V800シリーズ，V850ファミリ，V851, V852, V853, V854, V850/SA1, V850/SB1, V850/SB2, V850/SV1, V850E/MS1, V850E/MA1, V850E/IA1は日本電気株式会社の商標です。

UNIXはX/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。

Windows, WindowsNTは米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。

Green Hills Software, MULTIは米国Green Hills Software, Inc.の商標です。

PC/ATは米国IBM Corp.の商標です。

Solaris, SPARCstation, SunOSは米国SPARC International, Inc.の商標です。

HP9000シリーズ700は米国Hewlett Packard Corp.の商標です。

TRONはThe Realtime Operating system Nucleusの略称です。

ITRONはIndustrial TRONの略称です。

本製品は外国為替および外国貿易管理法の規定により規制貨物等（または役務）に該当しますので，
日本国外に輸出する場合には，同法に基づき日本国政府の輸出許可が必要です。

- **本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。**
- **文書による当社の承諾なしに本資料の転載複製を禁じます。**
- **本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。**
- **本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。**

M7A　98.8

本版で改訂された主な箇所

箇　所	内　容
p.20	1. 7　**実行環境**の記述を変更
p.20	1. 8　**開発環境**の記述を変更
p.23	**図**1ー1　CA850**を使用した場合のシステム構築手順**を修正
p.24	**図**1ー2　CCV850**を使用した場合のシステム構築手順**を修正
p.200	A. 8　**制限事項と注意事項**を追加
p.213	**付録**C　**改版履歴**を追加

本文欄外の★印は，本版で改訂された主な箇所を示しています。

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対 象 者　このマニュアルはV850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RX850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。

- 概説
- ニュークリアス
- タスク管理機能
- 同期通信機能
- 割り込み管理機能
- メモリ・プール管理機能
- 時間管理機能
- スケジューラ
- システム初期化処理
- システム・コール

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

V850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　ハードウエア編を参照してください。

V850ファミリの命令機能を知りたいとき
　→各製品のユーザーズ・マニュアル　アーキテクチャ編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	：本文の補足説明
数の表記	：2進数…XXXXまたはXXXXB
	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　**このマニュアルを使用する場合は，次の資料もあわせてご覧ください。**

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。

開発ツールに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号
IE-703002-MC（V851™, V852™, V853™, V854™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SV1™用インサーキット・エミュレータ）		U11595J
IE-703003-MC-EM1（V853用周辺I/Oボード）		U11596J
IE-703008-MC-EM1（V854用周辺I/Oボード）		U12420J
IE-703017-MC-EM1（V850/SA1用周辺I/Oボード）		U12898J
IE-703037-MC-EM1（V850/SB1, V850/SB2用周辺I/Oボード）		U14151J
IE-703040-MC-EM1（V850/SV1用周辺I/Oボード）		U14337J
IE-703102-MC（V850E/MS1™用インサーキット・エミュレータ）		U13875J
IE-703102-MC-EM1, IE-703102-MC-EM1-A（V850E/MS1用周辺I/Oボード）		U13876J
IE-V850E-MC（V850E/IA1™用インサーキット・エミュレータ）， IE-V850E-MC-A（V850E1（NB85Eコア），V850E/MA1™用インサーキット・エミュレータ）		U14487J
IE-V850E-MC-EM1-A（V850E1（NB85Eコア）用周辺I/Oボード）		作成予定
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2（V850E1（NB85Eコア）用周辺I/Oボード）		U14482J
IE-703107-MC-EM1（V850E/MA1用周辺I/Oボード）		U14481J
IE-703116-MC-EM1（V850E/IA1用周辺I/Oボード）		作成予定
V800シリーズ™開発ツール（32ビット対応）アプリケーション・ノート チュートリアル・ガイド　Windowsベース		U14218J
CA850（Cコンパイラ・パッケージ）	操作編	U14568J
	C言語編	U14566J
	プロジェクト・マネージャ編	U14569J
	アセンブリ言語編	U14567J
ID850（Ver.2.20）（統合ディバッガ）	操作編　Windowsベース	U14580J
SM850（Ver.2.20）（システム・シミュレータ）	操作編　Windowsベース	U14782J
RX850（リアルタイムOS）	基礎編	このマニュアル
	インストレーション編	U13410J
	テクニカル編	U13431J
RX850 Pro（リアルタイムOS）	基礎編	U13773J
	インストレーション編	U13774J
	テクニカル編	U13772J
RD850（タスク・ディバッガ）		U13737J
RD850 Pro（タスク・ディバッガ）		U13916J
AZ850（システム・パフォーマンス・アナライザ）		U14410J
PG-FP3（フラッシュ・メモリ・プログラマ）		U13502J

目　　次

図の目次（1/2）

図の目次（2/2）

表の目次

〔メ　モ〕

第1章　概　　説

1.1　概　　要

RX850は，効率のよいリアルタイム，マルチタスク処理環境を提供するとともに，V850ファミリ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，組み込み型制御用リアルタイム・マルチタスクOSです。

また，ターゲット・システム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，高速かつコンパクトなOSとなっています。

1.2　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即応性が要求されます。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，多機能化するにつれて，単純な割り込み処理だけでの対処が難しくなってきています。

つまり，システムの複雑化，処理プログラム量の増大により，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かを管理することが困難になってきたということです。

そこで，このような問題に対処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して即応し，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おもな目的としています。

1.3　マルチタスクOS

OSの管理下で実行する処理プログラムの最小単位を，「**タスク**」と呼びます。また，1つのプロセッサ上で複数のタスクを時間的に同時実行させることを，「**マルチタスキング**」と呼びます。

実際には，プロセッサ自体は，1度に1つの処理プログラム（命令）しか実行することができません。しかし，複数のタスクの実行を何らかの基準（きっかけ）を利用して切り替えることにより，あたかも複数のタスクが同時に実行しているかのようになります。

このように，システム内で定めた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OS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おもな目的としています。

1.4　特　　徴

RX850の特徴を次に示します。

(1) μITRON3.0仕様に準拠

RX850は，組み込み型制御用OSのアーキテクチャとして代表的なμITRON3.0仕様に準拠した設計が行われており，レベルSまでの機能を実装しています。

なお，μITRON3.0仕様とは，組み込み型制御用リアルタイム・システムのオペレーティング・システム仕様です。

(2) 高い汎用性

μ ITRON3.0仕様で規定されているシステム・コールを提供し，アプリケーション・システムの汎用性を高めています。

なお，RX850では，アプリケーション・システムが使用する機能（システム・コール）だけを選択することができるため，コンパクトでありながら，ユーザのニーズに最適なリアルタイム・マルチタスクOSを構築することができます。

(3) リアルタイム，マルチタスク処理の実現

RX850は，完全なリアルタイム，マルチタスク処理を実現するために，豊富な機能を提供しています。

- タスク管理機能
- タスク付属同期機能
- 同期通信機能
- 割り込み管理機能
- メモリ・プール管理機能
- 時間管理機能
- システム管理機能
- スケジューリング機能

(4) スケジューリングのロック機能

RX850は，ディスパッチ処理（タスクのスケジューリング処理）をユーザの処理プログラムから禁止／再開する機能を提供しています。

(5) ROM化の実現

RX850は，ターゲット・システムに組み込んで使用することを想定したリアルタイム・マルチタスクOSであるため，ROM化を意識し、コンパクトな設計が行われています。

(6) オリジナル命令の活用

RX850は，V850ファミリの高速な命令実行速度と，オリジナル命令の活用により，高速処理を実現しています。

(7) ユーティリティ・ツールの提供

RX850は，システムを構築するうえで有益なユーティリティ・ツールを提供しています。

- CF850（コンフィギュレータ）

1.5 構　成

RX850は，ニュークリアス，システム初期化処理，コンフィギュレータといった，3つのサブシステムから構成されています。

これらサブシステムの概要を次に示します。

(1) ニュークリアス

ニュークリアスとは，リアルタイム，マルチタスク制御を行うRX850の核となる部分であり，次に示す機能を提供しています。

- 管理オブジェクトの生成／初期化処理
- 処理プログラム（タスク，ハンドラ）から発行されたシステム・コールに対応した処理
- ターゲット・システムの内外で発生した事象に対応し，次に実行すべき処理プログラム（タスク，ハンドラ）の選択処理

なお，管理オブジェクトの生成／初期化処理，およびシステム・コールに対応した処理は各種管理モジュールで，処理プログラムの選択処理はスケジューラで行われます。

(2) システム初期化処理

システム初期化処理は，RX850が動作するうえで必要となるハードウエアの初期化処理，およびソフトウエアの初期化処理から構成されます。

したがって，RX850では，システムが起動したとき，まず最初に行われる処理がシステム初期化処理となります。

なお，システム初期化処理のうち，実行環境のハードウエア構成に依存する部分（リセット・ルーチン），およびユーザのソフトウエア環境を快適なものとする部分（初期化ハンドラ）については，サンプル・ソース・ファイルを提供しています。

これにより，さまざまなターゲット・システムへの移植性を向上させるとともに，カスタイマイズ化を容易なものとしています。

(3) コンフィギュレータCF850

RX850を使ったシステムを構築する場合，RX850に提供する各種データを保持した情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）が必要となります。

基本的に，これら情報ファイルは，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が，記述性／可読性の面で劣ったものとなります。

そこで，RX850では，対話形式で各種データを入力させ，その結果を情報ファイルとして出力するユーティリティ・ツール（コンフィギュレータCF850）を提供しています。

ユーザは，コンフィギュレータを利用することにより，情報ファイルの新規作成／変更などといった作業が容易なものとなります。

1.6 適用分野

RX850の適用分野を次に示します。

- サーボ・モータを使用したシステムの制御

例 PPC
プリンタ
NC工作機

・高速応答を必要とするシステムの制御

例　エンジン制御

携帯電話

PHS

ディジタル・スチル・カメラ

★ 1.7 実行環境

RX850は，組み込み型制御用のOSとして開発されているため，次に示すハードウエアを備えたターゲット・システム上で動作します。

(1) 動作対象CPU

- V851
- V852
- V853
- V854
- V850/SA1
- V850/SBx
- V850/SV1
- V850E/MS1
- V850E/MA1
- NB85Eコア
- V850E/IA1

(2) 周辺コントローラ

RX850では，さまざまな実行環境に対応するために，ニュークリアス内のハードウエア依存部を切り出し，サンプル・ソース・ファイルを提供しています。

このため，サンプル・ソース・ファイルを各ターゲット・システム用に書き換えることで，特定の周辺コントローラを要求しません。

★ 1.8 開発環境

システムを開発するうえで必要となるハードウエア環境とソフトウエア環境を次に示します。

1.8.1 ハードウエア環境

(1) ホスト・マシン

- PC-9800シリーズ
- PC/AT™互換機
- SPARCstation™
- HP9000シリーズ700™

(2) インサーキット・エミュレータ

- IE-703002-MC（V851, V852, V853, V854, V850/SA1, V850/SBx, V850/SV1）
- IE-703102-MC（V850E/MS1）
- IE-V850E-MC-A（V850E/MA1, NB85Eコア）
- IE-V850E-MC（V850E/IA1）

(3) インサーキット・エミュレータ用I/O**ボード**

- IE-703003-MC-EM1（V853）
- IE-703008-MC-EM1（V854）
- IE-703017-MC-EM1（V850/SA1）
- IE-703037-MC-EM1（V850/SBx）
- IE-703040-MC-EM1（V850/SV1）
- IE-703102-MC-EM1（V850E/MS1 5 V）
- IE-703102-MC-EM1-A（V850E/MS1 3.3 V）
- IE-703107-MC-EM1（V850E/MA1）
- IE-703116-MC-EM1（V850E/IA1）
- IE-V850E-MC-EM1-A（NB85Eコア 5 V）
- IE-V850E-MC-EM1-B（NB85Eコア 3.3 V）

注意　これらのI/O**ボードは，インサーキット・エミュレータと組み合わせて使用します。**

(4) PC**インタフェース・ボード**

- IE-70000-98-IF-C（PC-9800シリーズ Cバス用）
- IE-70000-PC-IF-C（PC/AT互換機 ISAバス用）
- IE-70000-CD-IF-A（PCMCIAソケット用）
- IE-70000-PCI-IF（PCIバス用）

1.8.2 ソフトウエア環境

(1) OS**（かっこ内はホスト・マシン）**

- Windows95/Windows98/WindowsNT™4.0（PC-9800シリーズ，PC/AT互換機）
- Solaris™2.x（SPARCstation）
- SunOS™4.1.x（SPARCstation）

(2) クロス・ツール

- CA850（NEC製）
- CCV850（GHS社製）

(3) ディバッガ

- ID850（NEC製）
- SM850（NEC製）
- MULTI™（GHS社製）
- PARTNER（KMC社製）

(4) タスク・ディバッガ

・RD850（NEC製）

(5) システム・パフォーマンス・アナライザ

・AZ850（NEC製）

1.9 システム構築手順

システム構築とは，RX850の提供媒体からユーザの開発環境（ホスト・マシン）上に転送したファイル群を用いて，ロード・モジュールを作成したのち，ターゲット・システムへ組み込むことです。

システムを構築する際の手順を次に示します。

ただし，詳細な説明は，RX850 **ユーザーズ・マニュアル　インストレーション編**（U13410J）を参照してください。

(1) 情報ファイルの作成

・システム情報テーブル

・システム情報ヘッダ・ファイル

(2) システム初期化処理の作成

・リセット・ルーチン

・初期化ハンドラ

(3) アイドル・ハンドラの作成

(4) 処理プログラムの作成

・タスク

・直接起動割り込みハンドラ

・間接起動割り込みハンドラ

・周期起動ハンドラ

(5) 初期化データの退避領域の作成

(6) リンク・ディレクティブ・ファイルの作成

(7) ロード・モジュールの生成

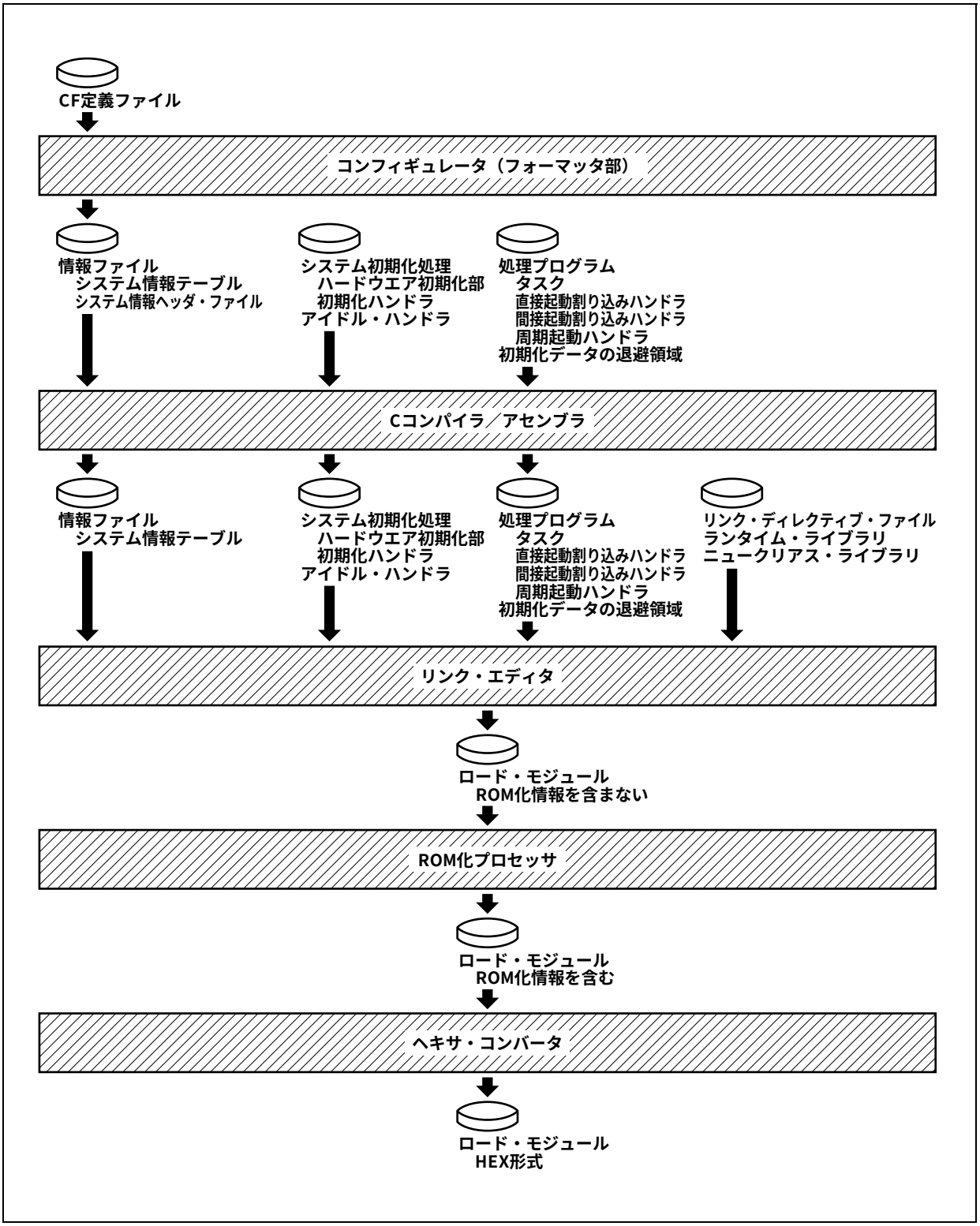
(8) システムへの組み込み

注意　CCV850を使用した場合，“初期化データの退避領域”を作成する必要はありません。

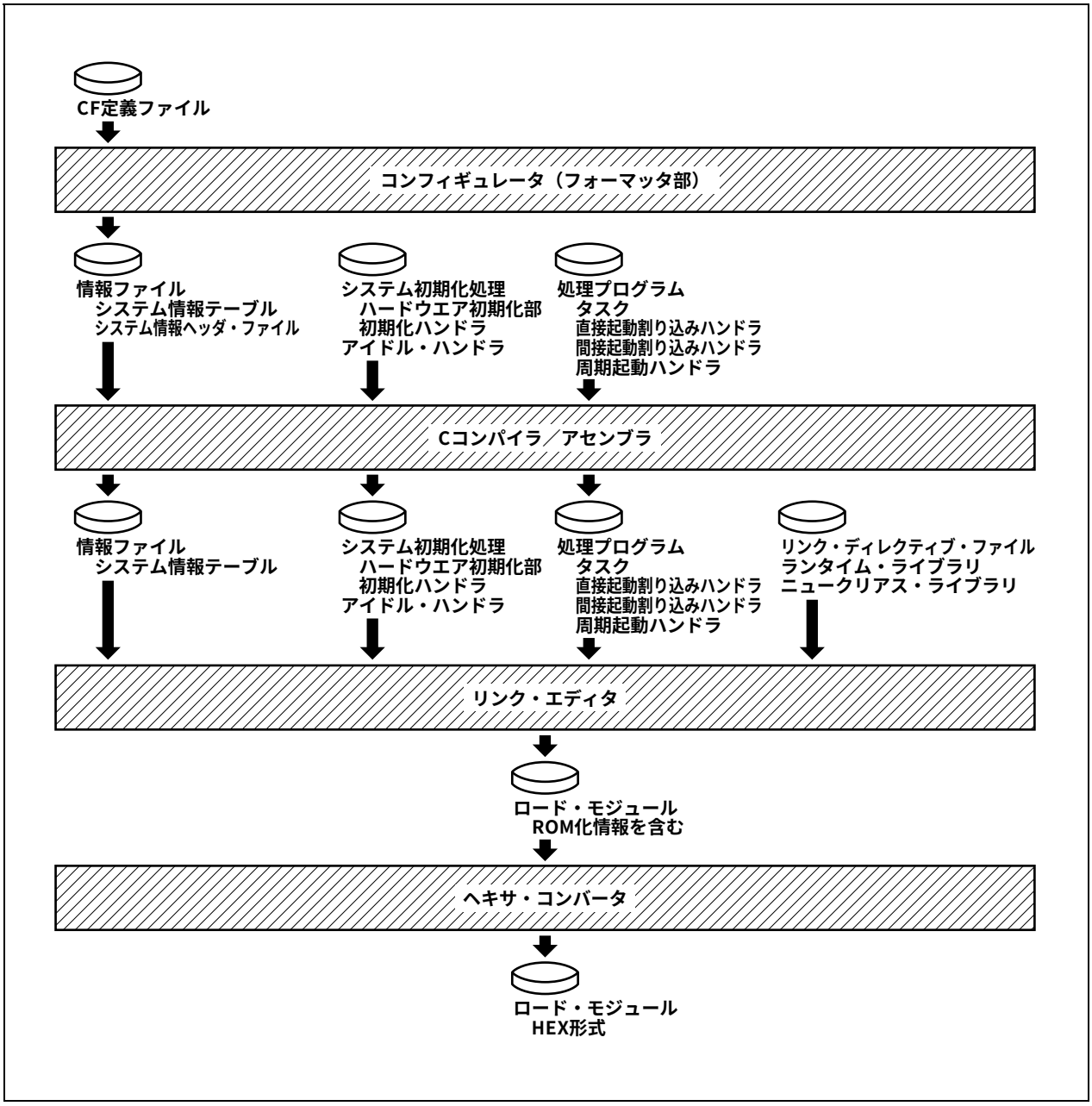
図1－1，図1－2に，システムを構築するときの手順を示します。

ただし，図1－1はCA850を使用した場合のシステム構築手順であり，図1－2はCCV850を使用した場合のシステム構築手順です。

★ **図1－1　CA850を使用した場合のシステム構築手順**



★ **図1－2　CCV850を使用した場合のシステム構築手順**



第2章　ニュークリアス

この章では，RX850の核であるニュークリアスについて示しています。

2.1 概　　要

ニュークリアスとは，リアルタイム，マルチタスク制御を行うRX850の核となる部分であり，次に示す機能を提供しています。

- 管理オブジェクトの初期化処理
- 処理プログラム（タスク，ハンドラ）から発行されたシステム・コールに対応した処理
- ターゲット・システムの内外で発生した事象に対応し，次に実行すべき処理プログラム（タスク，ハンドラ）の選択処理

なお，管理オブジェクトの初期化処理，およびシステム・コールに対応した処理は各種管理モジュールで，処理プログラムの選択処理はスケジューラで行われます。

2.2 機　　能

ニュークリアスは，各種管理モジュールとスケジューラから構成されています。
各種管理モジュールおよびスケジューラの機能概要を次に示します。
なお，これらの機能に関する詳細は，**第3章　タスク管理機能**から**第8章　スケジューラ**を参照してください。

(1) タスク管理機能

RX850の処理単位である，「タスクの起動，終了」などのタスクの状態操作，およびタスクの状態管理を行っています。

(2) 同期通信機能

「排他制御，待ち合わせ，通信」といったタスク間の同期通信機能として，次に示す3種類の機能を提供しています。

- 排他制御機能　：セマフォ
- 待ち合わせ機能：イベント・フラグ，1ビット・イベント・フラグ
- 通信機能　　：メールボックス

(3) 割り込み管理機能

「割り込みハンドラの起動，割り込みハンドラからの復帰，マスカブル割り込みの受け付け禁止／再開，割り込み制御レジスタの変更／獲得」などの割り込みに関連した処理を行っています。

(4) メモリ・プール管理機能

コンフィギュレーション時に指定されたメモリ領域を，次に示す2つの領域に分けて管理しています。

(a) RX850用の領域

・各種管理オブジェクト
・メモリ・プール

(b) 処理プログラム（タスク，ハンドラ）用の領域

・処理プログラムのテキスト領域
・処理プログラムのデータ領域
・処理プログラムのスタック領域

なお，RX850では，ダイナミックなメモリ・プール管理も行っており，作業用のメモリ領域が必要となったときに獲得し，不要となったときには返却できるといった機能が用意されています。

ユーザは，このようなダイナミックなメモリ操作を行うことにより，限りあるメモリ領域を効率よく使用することが可能となります。

(5) 時間管理機能

ハードウエアにより一定周期で発生するクロック割り込みを利用した，タイマ・オペレーション機能（タスクの時間経過待ち，周期起動ハンドラの起動）を提供しています。

(6) スケジューラ

タスクの実行順序を管理／決定し，タスクにプロセッサの使用権を与えています。

なお，RX850では，タスクの実行順序を決定する方法として，優先度方式およびFCFS方式を採用しています。このため，スケジューラは，駆動された時点で各タスクに付けられている優先度を参照し，実行可能な状態（run状態およびready状態）にあるタスクの中から最適なものを選び出し，プロセッサの使用権を与えています。

注意　RX850におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。

第3章　タスク管理機能

この章では，RX850が行うタスク管理機能について示しています。

3.1　概　　要

タスクは実行実体であり，サイズなどが一様でなく，直接管理することが困難です。そこで，RX850では，タスクと1対1に対応した管理オブジェクトを用いることにより，タスクが取り得る状態の管理，タスク自体の管理を行っています。

3.2　タスク実行権

各タスクには，それぞれ「スタック」が必要となります。タスク用のスタック（タスク・スタック）には，タスクが切り替わるときのレジスタ情報や，タスク内で使用されている自動変数の値などの情報を格納するために使われます。タスク・スタックは各タスクごとに確保しなくてはならないため，スタックによるRAMの消費量はかなり多くなります。

しかし，システム中には，同時に実行可能状態になることのないタスク同士が存在することもあります。このようなタスク同士のスタックを共有することができれば，RAM消費の低減をはかることができます。

そこでRX850ではこのようなタスク同士をグループ化し，「同一グループ内で起動可能なタスクは1タスクに限る」という機能を提供しています。ユーザは，このようなタスクのグループ化機能を利用することにより，RX850が各タスクに割り当てているスタック領域を共有化させることができます。RX850ではこのようなグループを「タスク実行権グループ」と呼び，タスクを起動するときに与える処理の実行権利を「タスク実行権」と呼びます。つまり，同一グループ内で，タスク実行権を持っているタスクが実行可能となります。同一グループ内の他のタスクを実行したいときには，現在実行権を持っているタスクがその実行権を放棄する必要があります。実行権の放棄は`ext_tsk`システム・コール（自タスク終了），`ter_tsk`（他タスク強制終了）システム・コールによって行われます。

注意1. **タスクを起動したとき，タスク実行権が獲得できなかった場合は，タスク実行権待ち状態へ移行します。**

2. **タスクが処理を実行するうえで必要となるプログラム・カウンタや作業用レジスタなどの実行環境情報は「タスク・コンテキスト」と呼ばれます。タスクの実行が切り替わるときには，現在実行中のタスク・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。なお，タスク・コンテキストはタスク・スタック内に確保されます。**

3.3　タスクの状態

タスクは，処理を実行するうえで必要となる資源の獲得状況，および事象発生の有無などにより，さまざまな状態へと遷移します。

そこで，RX850では，タスクが遷移する状態を，次に示す6種類に分けて管理しています。

(1) 休止（dormant）状態

タスクとして起動されていない状態，または処理を終了したときの状態です。

なお，dormant状態のタスクは，RX850のスケジューリング対象から除外されています。

また，wait状態との相違点は，次のようなものがあります。

- すべての資源を解放している
- プログラム・カウンタや作業用レジスタなどの実行環境情報（タスク・コンテキスト）が処理再開時に初期化される
- 状態操作を伴うシステム・コール（ter_tsk, chg_pri, sus_tsk, wup_tskなど）がエラーになる

(2) 実行可能（ready）状態

タスク実行権を獲得し，処理を実行する準備は整っているが，より高い優先度（同じ優先度の場合もある）を持つタスクが処理を実行中のため，プロセッサが割り当てられるのを待っている状態です。

なお，ready状態のタスクは，RX850のスケジューリング対象となります。

(3) 実行（run）状態

プロセッサが割り当てられ，現在，処理を実行中の状態です。

なお，run状態のタスクは，システム全体で1タスクしか存在しません。

(4) 待ち（wait）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，実行が中断した状態です。

なお，wait状態からの処理再開は，実行が中断した箇所からとなり，プログラム・カウンタや作業用レジスタなどの実行環境情報（タスク・コンテキスト）は中断直前のものが復元されます。

また，RX850では，wait状態を条件の種別により，次に示す7種類に分けて管理しています。

(a) タスク実行権待ち状態

sta_tskシステム・コールを発行したとき，対象タスク実行権グループからタスク実行権の獲得ができなかった場合に遷移する状態です。

(b) 起床待ち状態

slp_tsk, tslp_tskシステム・コールを発行したとき，自タスクのカウンタ（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。

(c) 資源待ち状態

wai_sem, twai_semシステム・コールを発行したとき，対象セマフォから資源の獲得ができなかった場合に遷移する状態です。

(d) イベント・フラグ待ち状態

wai_flg, twai_flgシステム・コールを発行したとき，対象イベント・フラグが要求した条件を満たしていなかった場合に遷移する状態です。

(e) 1ビット・イベント・フラグ待ち状態

vwai_flg1, vtwai_flg1システム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合に遷移する状態です。

(f) メッセージ待ち状態

rcv_msg, trcv_msgシステム・コールを発行したとき，対象メールボックスからメッセージの受信ができなかった場合に遷移する状態です。

(g) メモリ・ブロック待ち状態

get_blf, tget_blf, get_blk, tget_blkシステム・コールを発行したとき，対象メモリ・プールからメモリ・ブロックの獲得ができなかった場合に遷移する状態です。

(h) 時間経過待ち状態

dly_tskシステム・コールを発行したとき，遷移する状態です。

(5) 強制待ち（suspend）状態

他タスクの発行したシステム・コールにより実行が中断した状態です。

なお，suspend状態からの処理再開は，実行が中断した箇所からとなり，プログラム・カウンタや作業用レジスタなどの実行環境情報（タスク・コンテキスト）は中断直前のものが復元されます。

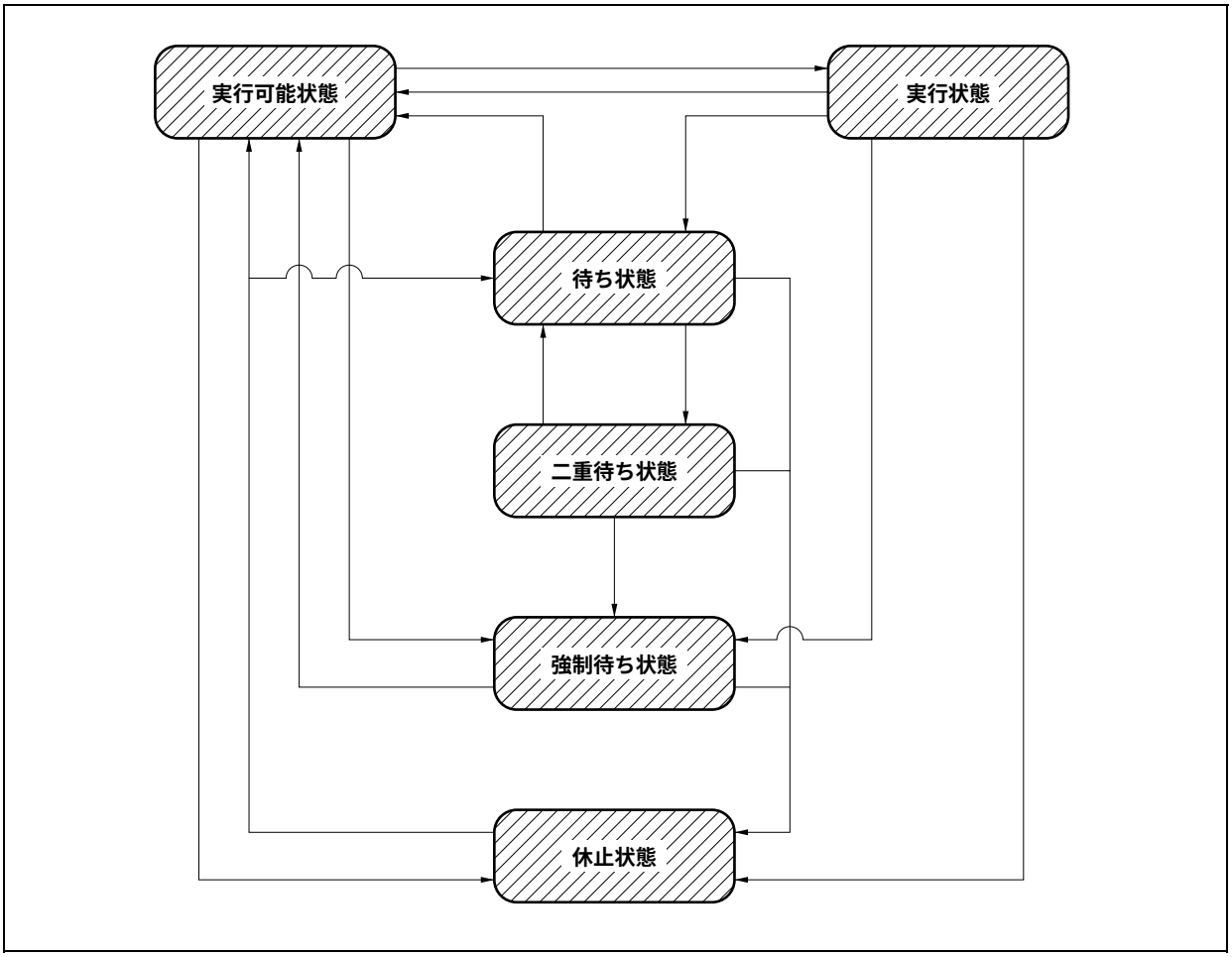
(6) 二重待ち（wait_suspend）状態

wait状態とsuspend状態が複合した状態です。

したがって，wait状態が解除されるとsuspend状態へ，suspend状態が解除されるとwait状態へと遷移します。

図3－1に，タスクが取り得る状態の関係を示します。

図3－1　タスクの状態遷移



3. 4　タスクの生成

RX850では，システムで使用するタスクをコンフィギュレーション時に指定します。つまり，RX850におけるタスクの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）だけで，システム・コールによるダイナミックな生成はできません。

なお，RX850におけるタスクの生成は，タスクを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域をタスクとして認識し，管理対象とします。

コンフィギュレーション時に指定するタスク情報を次に示します。

- タスク名
- タスクの起動アドレス
- タスク用スタックのサイズ
- タスク用スタックを割り付けるシステム・メモリの種別（.pool0, pool1セクション）
- タスクの初期優先度
- タスクの初期状態
- タスクの起動コード
- タスクを起動したときのマスカブル割り込みの受け付け許可／禁止

注意　タスク関連のシステム・コールでは，どのタスクに対して操作を行うのか（対象タスク）を指定するとき，ID番号を使用します。
このID番号は，システム情報テーブルに記述されたタスク情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各タスクごとに順次割り付けたものです。

3.5　タスクの起動

RX850におけるタスクの起動は，dormant状態のタスクにタスク実行権を与えたのち，ready状態へと遷移させることです。

なお，タスクの起動は，sta_tskシステム・コールを発行することにより行われます。

- sta_tskシステム・コール
  パラメータで指定されたタスクにタスク実行権を与えたのち，dormant状態からready状態へと遷移させます。
  ただし，このシステム・コールを発行したとき，該当するタスク実行権グループからタスク実行権を獲得することができなかった場合には，対象タスクを対象タスク実行権グループの待ちキューの最後尾にキューイングしたのち，dormant状態からwait状態（タスク実行権待ち状態）へと遷移させます。

3.6　タスクの終了

RX850におけるタスクの終了は，タスクをdormant状態へと遷移させ，RX850のスケジューリング対象から除外することです。

なお，RX850では，タスクの終了形態として，次に示す2種類を用意しています。

- 正常終了：すべての処理が順調に完了し，スケジューリング対象である必要がなくなったとき，自ら終了する形態です。
- 強制終了：処理途中で何らかの異常が発生し，緊急に処理を終了しなければならないとき，他タスクから終了させられる形態です。

なお，タスクの終了は，ext_tskまたはter_tskシステム・コールを発行することにより行われます。

- ext_tskシステム・コール
  自タスクをrun状態からdormant状態へと遷移させます。
- ter_tskシステム・コール
  パラメータ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。

3.7　タスク内での処理

RX850では，タスクを切り替えるとき，独自のスケジューリング処理を行っています。
そこで，タスクの処理を記述するときには，次に示す注意事項があります。

(1) レジスタの退避／復帰

RX850では，タスクを切り替えるとき，Cコンパイラ（CA850，または，CCV850）の関数呼び出し規約に従った作業用レジスタの退避処理／復帰処理を行っています。したがって，タスクの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要がありません。

ただし，タスクをアセンブリ言語で記述しレジスタ変数用レジスタを使用する場合は，タスクの開始部分でレジスタ変数用レジスタの退避処理を，終了部分でレジスタ変数用レジスタの復帰処理を記述する必要があります。

注意　RX850では，タスクを切り替えるとき，gp, tp, epレジスタの切り替えを行いません。

(2) スタックの切り替え

RX850では，タスクを切り替えるとき，各タスク専用のタスク用スタックへの切り替え処理を行っています。したがって，タスクの開始部分および終了部分でスタックの切り替え処理を記述する必要がありません。

(3) システム・コールの発行制限

タスク内で発行可能なシステム・コールの一覧を，次に示します。

・タスク管理機能システム・コール

```
sta_tsk     ext_tsk     ter_tsk     dis_dsp     ena_dsp
chg_pri     rot_rdq     rel_wai     get_tid     ref_tsk
```

・タスク付属同期機能システム・コール

```
sus_tsk     rsm_tsk     frsm_tsk    slp_tsk     tslp_tsk
wup_tsk     can_wup
```

・同期通信機能システム・コール

```
sig_sem     wai_sem     preq_sem    twai_sem    ref_sem
set_flg     clr_flg     wai_flg     pol_flg     twai_flg
ref_flg     vset_flg1   vclr_flg1   vwai_flg1   vpol_flg1
vtwai_flg1  vref_flg1   snd_msg     rcv_msg     prcv_msg
trcv_msg    ref_mbx
```

・割り込み管理機能システム・コール

```
loc_cpu     unl_cpu     dis_int     ena_int     chg_icr
ref_icr
```

・メモリ・プール管理機能システム・コール

```
get_blf     pget_blf    tget_blf    rel_blf     ref_mpf
get_blk     pget_blk    tget_blk    rel_blk     ref_mpl
```

・時間管理機能システム・コール

```
dly_tsk     act_cyc     ref_cyc
```

・システム管理機能システム・コール

get_ver　　ref_sys

3.8　タスクID番号の獲得

現在実行中のタスクのID番号を取得するには，get_tidシステム・コールを使用します。

タスクのID番号は，タスクを生成するときコンフィギュレーション・ファイルにてシンボルで指定するので，通常はそのシンボルをタスクのID番号として使用します。しかし，複数のタスクが同じプログラム・コードを共有している場合などで，そのコードがどのタスクにより実行されているかを知りたいときにこのシステム・コールを使用します。

3.9　タスク情報の獲得

タスク情報の獲得は，ref_tskシステム・コールを発行することにより行われます。

(1) ref_tskシステム・コール

パラメータで指定されたタスクのタスク情報（拡張情報，現在の優先度など）を獲得します。

タスク情報の詳細を次に示します。

・拡張情報

・現在の優先度

・タスクの状態

TTS_RUN（H' 01）：run状態

TTS_RDY（H' 02）：ready状態

TTS_WAI（H' 04）：wait状態

TTS_SUS（H' 08）：suspend状態

TTS_WAS（H' 0c）：wait_suspend状態

TTS_DMT（H' 10）：dormant状態

TTS_WTX（H' 20）：タスク実行権待ち状態

TTS_WTS（H' 28）：タスク実行権待ち＋suspend状態

・wait状態の種類

TTW_SLP（H' 0001）　：起床待ち状態

TTW_DLY（H' 0002）　：時間経過待ち状態

TTW_FLG（H' 0010）　：イベント・フラグ待ち状態

TTW_SEM（H' 0020）　：資源待ち状態

TTW_MBX（H' 0040）　：メッセージ待ち状態

TTW_MPL（H' 1000）　：可変長メモリ・ブロック待ち状態

TTW_MPF（H' 2000）　：固定長メモリ・ブロック待ち状態

TTW_1FLG（H' 4000）：1ビット・イベント・フラグ待ち状態

・対象オブジェクトのID番号

〔メ　モ〕

第4章　同期通信機能

この章では，RX850が行う同期通信機能について示しています。

4.1 概　　要

複数のタスクが並行に実行されている環境（マルチタスク処理）では，あるタスクの処理結果により，次に実行されるタスクが選択されるとか，タスクの処理内容に違いが出るなど，タスク間で処理の実行条件を制限しあったり，処理内容が相互に関係しているという場合があります。

このため，あるタスクが他タスクの処理結果が出るまで実行を中断したり，処理を継続するうえで必要な条件が整うまで待つといった，タスク間の連絡機能が必要となります。

RX850では，このような機能を「**同期機能**」と呼びます。なお，同期機能には，「排他制御機能」と「待ち合わせ機能」があり，RX850では，排他制御機能としてセマフォを，待ち合わせ機能としてイベント・フラグおよび1ビット・イベント・フラグを提供しています。

また，マルチタスク処理では，他タスクから処理結果を通知してもらうといった，タスク間の通信機能も必要となります。

RX850では，このような機能を「**通信機能**」と呼びます。なお，RX850では，通信機能としてメールボックスを提供しています。

4.2 セマフォ

マルチタスク処理では，並行に動作する複数のタスクが限られた数の資源（A/Dコンバータ，コプロセッサ，ファイル，プログラムなど）を同時に使用するというような資源の競合を防ぐ機能が必要となります。RX850では，このような機能を実現するために，非負数の計数型セマフォを提供しています。

セマフォは，資源の数を管理するカウンタであり，RX850のセマフォは，7ビット幅のカウンタでタスク間の排他制御を行います。

なお，セマフォに対するダイナミックな操作は，次に示すセマフォ関連のシステム・コールを用いて行います。

- `sig_sem`　：資源を返却する
- `wai_sem`　：資源を獲得する
- `preq_sem`　：資源を獲得する（ポーリング）
- `twai_sem`　：資源を獲得する（タイムアウトあり）
- `ref_sem`　：セマフォ情報を獲得する

注意1. RX850では，タスクの実行に必要となる各種要素を「資源」と呼びます。つまり，資源とは，A/Dコンバータ，コプロセッサなどのハードウエアや，ファイル，プログラムなどのソフトウエアのすべてを指します。

2. セマフォ関連のシステム・コールでは，どのセマフォに対して操作を行うのか（対象セマフォ）を指定するときにID番号を使用します。
   このID番号は，システム情報テーブルに記述されたセマフォ情報をもとに，コンフィギュレータCF850が`0x1`から始まる整数値を各セマフォごとに順次割り付けたものです。

4.2.1　セマフォの生成

RX850では，システムで使用するセマフォをコンフィギュレーション時に指定します。つまり，RX850におけるセマフォの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）だけで，システム・コールによるダイナミックな生成はできません。

なお，RX850におけるセマフォの生成は，セマフォを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域をセマフォとして認識し，管理対象とします。

コンフィギュレーション時に指定するセマフォ情報を次に示します。

- セマフォ名
- セマフォの初期資源数

4.2.2　資源の返却

資源の返却は，sig_semシステム・コールを発行することにより行われます。

(1) sig_semシステム・コール

パラメータ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，このシステム・コールを発行したとき，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却処理（セマフォ・カウンタの加算処理）は行わず，該当するタスク（待ちキューの先頭タスク）に資源を渡します。

これにより，該当するタスクは，待ちキューから外れ，wait状態（資源待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

4.2.3　資源の獲得

資源の獲得は，wai_sem, preq_sem, またはtwai_semシステム・コールを発行することにより行われます。

(1) wai_semシステム・コール

パラメータ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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は次の場合に行われ，資源待ち状態からready状態へと遷移します。

- sig_semシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に資源待ち状態を解除した

注意　タスクを対象セマフォの待ちキューにキューイングするときのキューイング方法は，資源の獲得要求を行った順（FIFO順）に行われます。

(2) preq_semシステム・コール

パラメータ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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

(3) twai_semシステム・コール

パラメータ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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は次の場合に行われ，資源待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- sig_semシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に資源待ち状態を解除した

注意　タスクを対象セマフォの待ちキューにキューイングするときのキューイング方法は，資源の獲得要求を行った順（FIFO順）に行われます。

4.2.4　セマフォ情報の獲得

セマフォ情報の獲得は，ref_semシステム・コールを発行することにより行われます。

(1) ref_semシステム・コール

パラメータで指定されたセマフォのセマフォ情報（拡張情報，待ちタスクの有無など）を獲得します。

セマフォ情報の詳細を次に示します。

- 拡張情報
- 待ちタスクの有無

  FALSE（0）：待ちタスクなし

  数値　　：待ちキューの先頭タスクのID番号
- 現在の資源数

4.2.5　セマフォによる排他制御

セマフォを使用してタスク間の排他制御を行った場合の動作例を次に示します。

（前提条件）

- タスクの優先度

  タスクA＞タスクB
- タスクの状態

  タスクA：run状態

  タスクB：ready状態
- セマフォの属性

  初期資源数：0x1

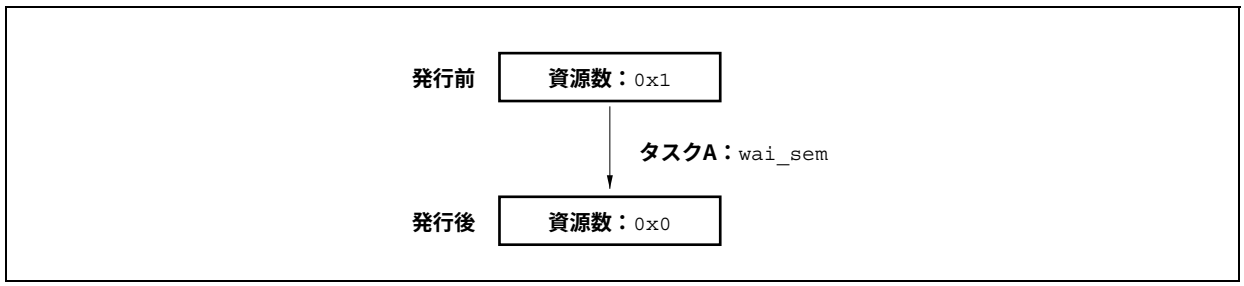
(1) タスクAがwai_semシステム・コールを発行します。

現在，RX850が管理している対象セマフォの資源数は「0x1」です。このため，RX850は，対象セマフォから0x1を減算します。

このとき，タスクAはwait状態（資源待ち状態）へと遷移することはなく，run状態のままとなります。

図4－1に，対象セマフォのカウンタの状態を示します。

図4－1　セマフォ・カウンタの状態

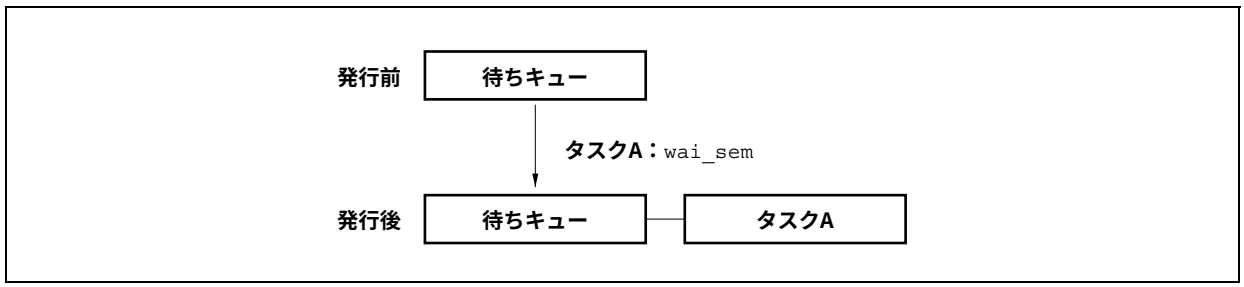


(2) タスクAがwai_semシステム・コールを発行します。

現在，RX850が管理している対象セマフォの資源数は「0x0」です。このため，RX850は，タスクAをrun状態から資源待ち状態へと遷移させたのち，対象セマフォの待ちキューの最後尾にキューイングします。

図4－2に，対象セマフォの待ちキューの状態を示します。

図4－2　待ちキューの状態



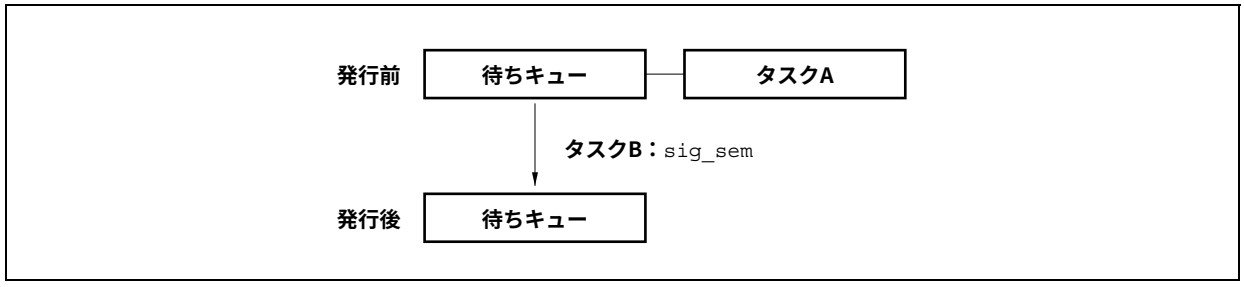
(3) タスクAの資源待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

(4) タスクBがsig_semシステム・コールを発行します。

これにより，対象セマフォの待ちキューにキューイングされているタスクAが資源待ち状態からready状態へと遷移します。

図4－3に，対象セマフォの待ちキューの状態を示します。

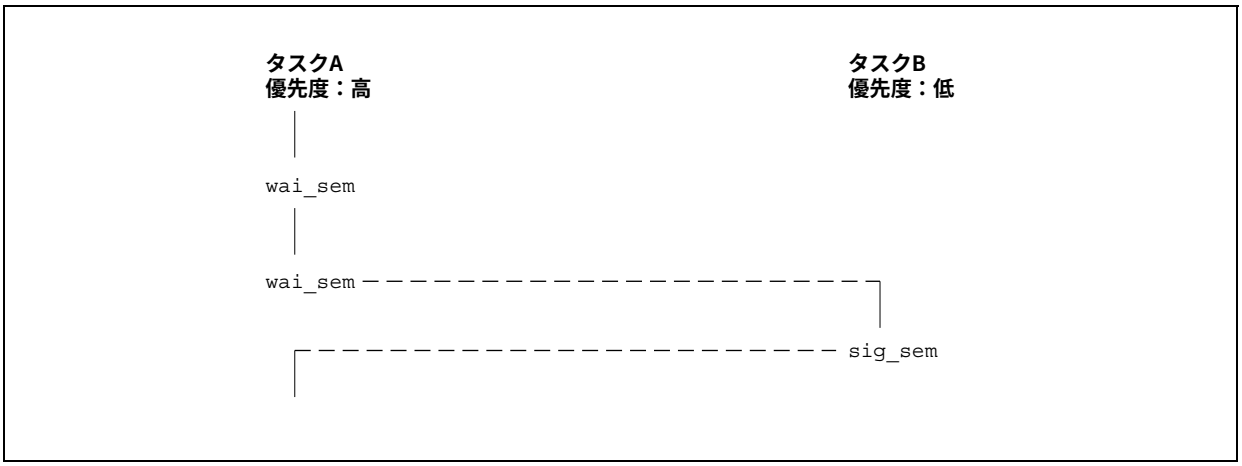
図4－3　待ちキューの状態



(5) 優先度の高いタスクAがready状態からrun状態へと遷移します。
　なお，タスクBはrun状態からready状態へと遷移します。

図4－4に，(1) ～ (5) の排他制御の流れを示します。

図4－4　セマフォによる排他制御



4.3　イベント・フラグ

マルチタスク処理では，あるタスクの処理結果が出るまで，他タスクが処理の実行を待つといった，タスク間の待ち合わせ機能が必要となります。このような場合，「処理結果が出た」という事象（イベント）が発生したかどうかを，他タスクで判断できるような機能があればよく，RX850では，このような機能を実現するために，イベント・フラグを提供しています。

イベント・フラグは，事象の有無を表す1ビットのフラグから構成されるデータの集合体であり，RX850のイベント・フラグは，32ビットを情報の単位として扱っており，各ビットに意味を持たせたり，数ビットの組み合わせで意味を持たせたりすることができます。

なお，イベント・フラグに対するダイナミックな操作は，次に示すイベント・フラグ関連のシステム・コールを用いて行います。

- set_flg ：ビット・パターンを設定する
- clr_flg ：ビット・パターンをクリアする
- wai_flg ：ビット・パターンをチェックする
- pol_flg ：ビット・パターンをチェックする（ポーリング）
- twai_flg：ビット・パターンをチェックする（タイムアウトあり）
- ref_flg ：イベント・フラグ情報を獲得する

注意　イベント・フラグ関連のシステム・コールでは，どのイベント・フラグに対して操作を行うのか（対象イベント・フラグ）を指定する際，ID番号を使用します。
このID番号は，システム情報テーブルに記述されたイベント・フラグ情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各イベント・フラグごとに順次割り付けたものです。

4.3.1　イベント・フラグの生成

RX850では,システムで使用するイベント・フラグをコンフィギュレーション時に指定します。つまり,RX850におけるイベント・フラグの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）だけで，システム・コールによるダイナミックな生成はできません。

なお，RX850におけるイベント・フラグの生成は，イベント・フラグを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域をイベント・フラグとして認識し，管理対象とします。

コンフィギュレーション時に指定するイベント・フラグ情報を次に示します。

・イベント・フラグ名

注意1. RX850では，システム初期化処理時にイベント・フラグを生成するとき，初期ビット・パターンとして，「0x0」を設定しています。

2. RX850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が1タスクに限られています。

4.3.2　ビット・パターンの設定

ビット・パターンの設定は，set_flgシステム・コールを発行することにより行われます。

(1) set_flgシステム・コール

パラメータで指定されたイベント・フラグにビット・パターンを設定します。

ただし，このシステム・コールを発行したとき，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待ち条件を満足した場合には，該当するタスクを待ちキューから外します。

これにより，該当するタスクは，wait状態（イベント・フラグ待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

4.3.3　ビット・パターンのクリア

ビット・パターンのクリアは，clr_flgシステム・コールを発行することにより行われます。

(1) clr_flgシステム・コール

パラメータで指定されたイベント・フラグのビット・パターンをクリアします。

ただし，このシステム・コールを発行したとき，すでに対象イベント・フラグのビット・パターンが0クリアされていても，エラーとして扱われないので注意が必要です。

4.3.4　ビット・パターンのチェック

ビット・パターンのチェックは，wai_flg, pol_flg, またはtwai_flgシステム・コールを発行することにより行われます。

(1) wai_flgシステム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンが設定されているか否かをチェックします。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが要求する

待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューにキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は次の場合に行われ，イベント・フラグ待ち状態からready状態へと遷移します。

- set_flgシステム・コールが発行され，要求する待ち条件を設定した
- rel_waiシステム・コールが発行され，強制的にイベント・フラグ待ち状態を解除した

(2) pol_flgシステム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンが設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが要求する待ち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。

(3) twai_flg

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンが設定されているか否かをチェックします。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューにキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は次の場合に行われ，イベント・フラグ待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- set_flgシステム・コールが発行され，要求する待ち条件を設定した
- rel_waiシステム・コールが発行され，強制的にイベント・フラグ待ち状態を解除した

備考　RX850では，イベント・フラグの待ち条件／条件成立時の指定として，次の条件を設定することができます。

(1) 待ち条件の指定

- AND待ち

  要求ビット・パターンで「1」が設定されている全ビットが，対象イベント・フラグに設定されるまで待ちます。
- OR待ち

  要求ビット・パターンで「1」が設定されているいずれかのビットが，対象イベント・フラグに設定されるまで待ちます。

(2) 条件成立時の指定

- ビット・パターンをクリアする

  待ち条件を満足したとき，対象イベント・フラグのビット・パターンのクリアを行います。

4.3.5　イベント・フラグ情報の獲得

イベント・フラグ情報の獲得は，ref_flgシステム・コールを発行することにより行われます。

(1) ref_flgシステム・コール

パラメータで指定されたイベント・フラグのイベント・フラグ情報（拡張情報，待ちタスクの有無など）を獲得します。

イベント・フラグ情報の詳細を次に示します。

- 拡張情報
- 待ちタスクの有無

  FALSE（0）：待ちタスクなし

  数値　　：待ちキューの先頭タスクのID番号
- 現在のビット・パターン

4.3.6　イベント・フラグによる待ち合わせ

イベント・フラグを使用してタスク間の待ち合わせを行った場合の動作例を次に示します。

（前提条件）

- タスクの優先度

  タスクA＞タスクB
- タスクの状態

  タスクA：run状態

  タスクB：ready状態
- イベント・フラグの属性

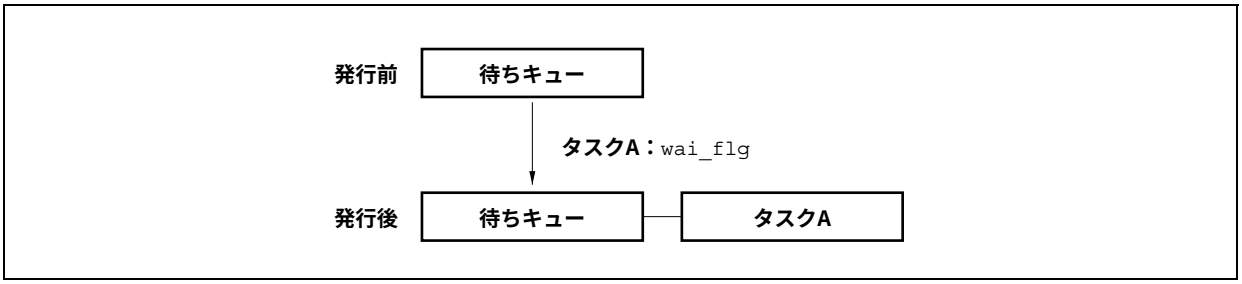
  初期ビット・パターン：0x0

(1) タスクAがwai_flgシステム・コールを発行します。なお，要求ビット・パターンは「0x1」であり，待ち条件／条件成立時の指定は「TWF_ANDW｜TWF_CLR」です。

現在，RX850が管理している対象イベント・フラグのビット・パターンは「0x0」です。このため，RX850は，タスクAをrun状態からwait状態（イベント・フラグ待ち状態）へと遷移させたのち，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。

図4－5に，対象イベント・フラグの待ちキューの状態を示します。

図4－5　待ちキューの状態



(2) タスクAのイベント・フラグ待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移し

ます。

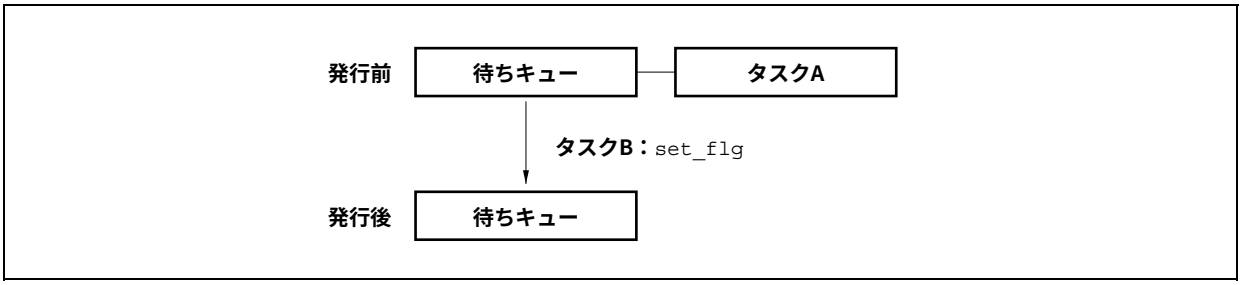
(3) タスクBがset_flgシステム・コールを発行します。なお，設定ビット・パターンは「0x1」です。

これにより，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Aの待ち条件を満足したため，タスクAがイベント・フラグ待ち状態からready状態へと遷移します。

また，タスクAは，wai_flgシステム・コールを発行したとき，「TWF_CLR」を指定していたため，対象イベント・フラグのビット・パターンはクリアされます。

図4－6に，対象イベント・フラグの待ちキューの状態を示します。

図4－6　待ちキューの状態

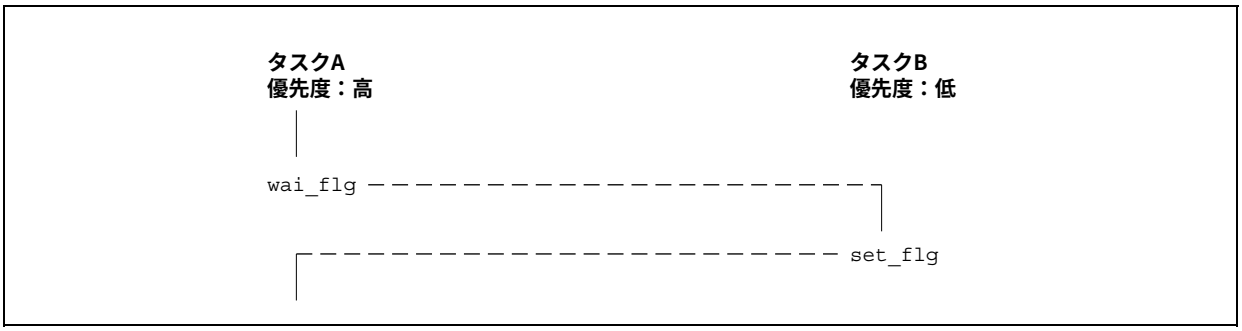


(4) 優先度の高いタスクAがready状態からrun状態へと遷移します。

なお，タスクBはrun状態からready状態へと遷移します。

図4－7に，(1) ～ (4) の待ち合わせの流れを示します。

図4－7　イベント・フラグによる待ち合わせ



4.4　1ビット・イベント・フラグ

マルチタスク処理では，あるタスクの処理結果が出るまで，他タスクが処理の実行を待つといった，タスク間の待ち合わせ機能が必要となります。このような場合，「処理結果が出た」という事象（イベント）が発生したかどうかを，他タスクで判断できるような機能があればよく，RX850では，このような機能を実現するために，1ビット・イベント・フラグを提供しています。

1ビット・イベント・フラグは，事象の有無を表す1ビットから構成されており，RX850の1ビット・イベント・フラグは，1ビットを情報の単位として扱っています。

4. 3　**イベント・フラグ**で記述したイベント・フラグでは，1つのイベント・フラグの待ちキューに1つのタス

クしかキューイングできませんが，この1ビット・イベント・フラグでは複数のタスクのキューイングが可能です。複数のタスクの待ち状態を一度に解除したいときなどに，この1ビット・イベント・フラグを使用します。

なお，1ビット・イベント・フラグに対するダイナミックな操作は，次に示す1ビット・イベント・フラグ関連のシステム・コールを用いて行います。

vset_flg1　：ビットを設定する
vclr_flg1　：ビットをクリアする
vwai_flg1　：ビットをチェックする
vpol_flg1　：ビットをチェックする（ポーリング）
vtwai_flg1：ビットをチェックする（タイムアウトあり）
vref_flg1　：1ビット・イベント・フラグ情報を獲得する

注意　1ビット・イベント・フラグ関連のシステム・コールでは，どの1ビット・イベント・フラグに対して操作を行うのか（対象1ビット・イベント・フラグ）を指定するときに，ID番号を使用します。このID番号は，システム情報テーブルに記述された1ビット・イベント・フラグ情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各1ビット・イベント・フラグごとに順次割り付けたものです。

4.4.1　1ビット・イベント・フラグの生成

RX850では，システムで使用する1ビット・イベント・フラグをコンフィギュレーション時に指定します。つまり，RX850における1ビット・イベント・フラグの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）のみで，システム・コールによるダイナミックな生成はできません。

なお，RX850における1ビット・イベント・フラグの生成は，1ビット・イベント・フラグを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域を1ビット・イベント・フラグとして認識し，管理対象とします。

コンフィギュレーション時に指定する1ビット・イベント・フラグ情報を次に示します。

・1ビット・イベント・フラグ名

注意　RX850では，システム初期化処理時に1ビット・イベント・フラグを生成するとき，初期ビット値として，“0”を設定しています。

4.4.2　ビットの設定

ビットの設定は，vset_flg1システム・コールを発行することにより行われます。

(1) vset_flg1システム・コール

パラメータで指定された1ビット・イベント・フラグに“1”を設定します。

ただし，このシステム・コールを発行したとき，対象1ビット・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た場合には，待ちキューの先頭タスクからビット・クリア指定を行っているタスクまでを待ちキューから外します。

これにより，待ちキューから外れたタスクは，wait状態（1ビット・イベント・フラグ待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

4.4.3　ビットのクリア

ビットのクリアは，vclr_flg1システム・コールを発行することにより行われます。

(1) vclr_flg1システム・コール

パラメータで指定された1ビット・イベント・フラグに“0”を設定します。

ただし，このシステム・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，すでにこのシステム・コールが発行され，対象1ビット・イベント・フラグがクリアされていた場合には，何も処理は行わず，エラーとしても扱われないので注意が必要です。

4.4.4　ビットのチェック

ビットのチェックは，vwai_flg1, vpol_flg1, またはvtwai_flg1システム・コールを発行することにより行われます。

(1) vwai_flg1システム・コール

パラメータで指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，自タスクを対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（1ビット・イベント・フラグ待ち状態）へと遷移させます。

なお，1ビット・イベント・フラグ待ち状態の解除は次の場合に行われ，1ビット・イベント・フラグ待ち状態からready状態へと遷移します。

- vset_flg1システム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に1ビット・イベント・フラグ待ち状態を解除した

注意　タスクを対象1ビット・イベント・フラグの待ちキューにキューイングするときのキューイング方法は，ビットのチェックを行った順（FIFO順）に行われます。

(2) vpol_flg1システム・コール

パラメータで指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，戻り値としてE_TMOUTが返されます。

(3) vtwai_flg1

パラメータで指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，自タスクを対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（1ビット・イベント・フラグ待ち状態）へと遷移させます。

なお，1ビット・イベント・フラグ待ち状態の解除は次の場合に行われ，1ビット・イベント・フラグ待ち状態からready状態へと遷移します。

・パラメータで指定された待ち時間が経過した
・vset_flg1システム・コールが発行された
・rel_waiシステム・コールが発行され，強制的に1ビット・イベント・フラグ待ち状態を解除した

注意　タスクを対象1ビット・イベント・フラグの待ちキューにキューイングするときのキューイング方法は，ビットのチェックを行った順（FIFO順）に行われます。

備考　RX850では，1ビット・イベント・フラグの条件成立時の指定として，次の条件を設定することができます。

（1）条件成立時の指定

・ビットをクリアしない
待ち条件を満足したとき，対象1ビット・イベント・フラグのビットのクリアを行いません。
・ビットをクリアする
待ち条件を満足したとき，対象1ビット・イベント・フラグのビットのクリアを行います。

4.4.5　1ビット・イベント・フラグ情報の獲得

1ビット・イベント・フラグ情報の獲得は，vref_flg1システム・コールを発行することにより行われます。

・vref_flg1システム・コール
パラメータで指定された1ビット・イベント・フラグの1ビット・イベント・フラグ情報（拡張情報，待ちタスクの有無など）を獲得します。
1ビット・イベント・フラグ情報の詳細を次に示します。

・拡張情報
・待ちタスクの有無
FALSE（0）：待ちタスクなし
数値　　：待ちキューの先頭タスクのID番号
・現在のビット値

4.4.6　1ビット・イベント・フラグによる待ち合わせ

1ビット・イベント・フラグを使用してタスク間の待ち合わせを行った場合の動作例を，以下に示します。

（前提条件）

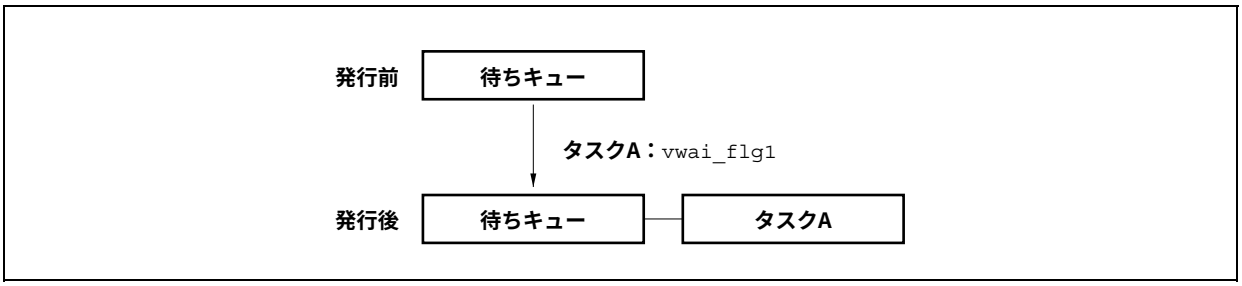
・タスクの優先度
タスクA＞タスクB
・タスクの状態
タスクA：run状態
タスクB：ready状態
・1ビット・イベント・フラグの属性
初期ビット値：0

(1) タスクAがvwai_flg1システム・コールを発行します。なお，条件成立時の指定は「TWF_CLR」です。

現在，RX850が管理している対象1ビット・イベント・フラグのビット・パターンは「0」です。このため，RX850は，タスクAをrun状態からwait状態（イベント・フラグ待ち状態）へと遷移させたのち，対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングします。

図4－8に，対象1ビット・イベント・フラグの待ちキューの状態を示します。

図4－8　待ちキューの状態



(2) タスクAのイベント・フラグ待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

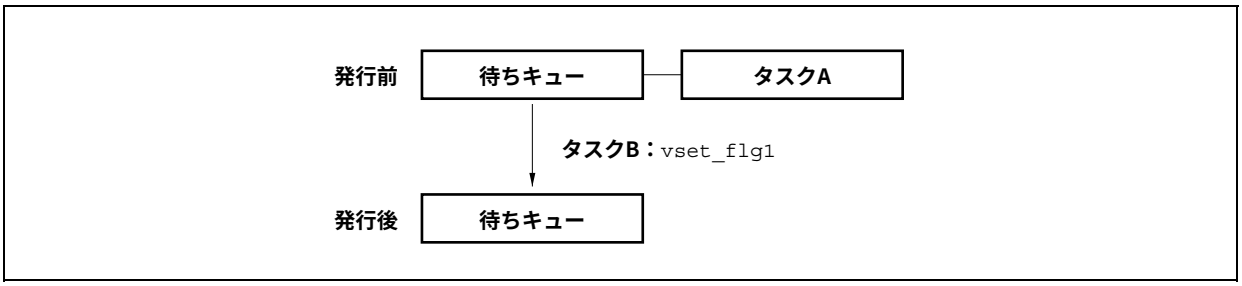
(3) タスクBがvset_flg1システム・コールを発行します。

これにより，対象1ビット・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Aは，イベント・フラグ待ち状態からready状態へと遷移します。

また，タスクAは，vwai_flg1システム・コールを発行したとき，「TWF_CLR」を指定していたため，対象1ビット・イベント・フラグのビットはクリアされます。

図4－9に，対象1ビット・イベント・フラグの待ちキューの状態を示します。

図4－9　待ちキューの状態

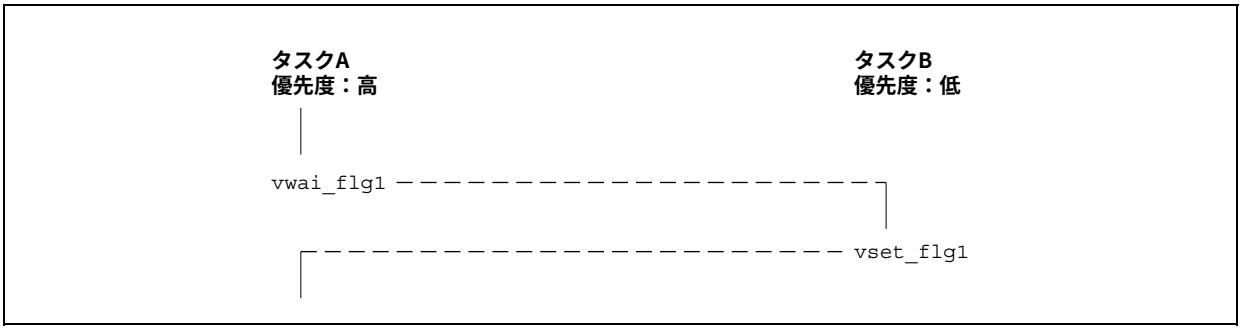


(4) 優先度の高いタスクAがready状態からrun状態へと遷移します。

なお，タスクBはrun状態からready状態へと遷移します。

図4－10に，(1) ～ (4) の待ち合わせの流れを示します。

図4－10　1ビット・イベント・フラグによる待ち合わせ



4.5　メールボックス

マルチタスク処理では，他タスクの処理結果を通知してもらうといった，タスク間の通信機能が必要となります。RX850では，このような機能を実現するために，メールボックスを提供しています。

RX850におけるメールボックスは，タスク専用の待ちキュー（タスク用待ちキュー）と，メッセージ専用の待ちキュー（メッセージ用待ちキュー）を持ち，タスク間のメッセージ通信機能として使用するほかに，タスク間の待ち合わせ機能として使用することもできます。

なお，メールボックスに対するダイナミックな操作は，次に示すメールボックス関連のシステム・コールを用いて行います。

snd_msg　：メッセージを送信する
rcv_msg　：メッセージを受信する
prcv_msg：メッセージを受信する（ポーリング）
trcv_msg：メッセージを受信する（タイムアウトあり）
ref_mbx　：メールボックス情報を獲得する

注意　メールボックス関連のシステム・コールでは，どのメールボックスに対して操作を行うのか（対象メールボックス）を指定するとき，ID番号を使用します。
このID番号は，システム情報テーブルに記述されたメールボックス情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各メールボックスごとに順次割り付けたものです。

4.5.1　メールボックスの生成

RX850では，システムで使用するメールボックスをコンフィギュレーション時に指定します。つまり，RX850におけるメールボックスの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）だけで，システム・コールによるダイナミックな生成はできません。

なお，RX850におけるメールボックスの生成は，メールボックスを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域をメールボックスとして認識し，管理対象とします。

コンフィギュレーション時に指定するメールボックス情報を次に示します。

・メールボックス名
・メッセージのキューイング方式

4.5.2　メッセージの送信

メッセージの送信は，snd_msgシステム・コールを発行することにより行われます。

(1) snd_msgシステム・コール

パラメータで指定されたメールボックスにメッセージを送信します。

ただし，このシステム・コールを発行したとき，対象メールボックスのタスク用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操作は行わず，該当するタスク（タスク用待ちキューの先頭タスク）にメッセージを渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（メッセージ待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意　メッセージを対象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ときのキューイング方法は，対象メールボックス生成時（コンフィギュレーション時）に指定された順（FIFO順，優先度順）に行われます。

4.5.3　メッセージの受信

メッセージの受信は，rcv_msg，prcv_msg，またはtrcv_msgシステム・コールを発行することにより行われます。

(1) rcv_msgシステム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたのち，run状態からwait状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態の解除は次の場合に行われ，メッセージ待ち状態からready状態へと遷移します。

- snd_msgシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的にメッセージ待ち状態を解除した

注意　タスクを対象メールボックスのタスク用待ちキューにキューイングするときのキューイング方法は，メッセージの受信要求を行った順（FIFO順）に行われます。タスクの優先度順にキューイングすることはできません。

(2) prcv_msgシステム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

(3) trcv_msgシステム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたのち，run状態からwait状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態の解除は次の場合に行われ，メッセージ待ち状態からready状態へと遷移します。

・パラメータで指定された待ち時間が経過した
・snd_msgシステム・コールが発行された
・rel_waiシステム・コールが発行され，強制的にメッセージ待ち状態を解除した

注意　タスクを対象メールボックスのタスク用待ちキューにキューイングするときのキューイング方法は，メッセージの受信要求を行った順（FIFO順）に行われます。

4.5.4　メッセージ

RX850では，メールボックスを介して，タスク間でやり取りされる情報を「**メッセージ**」と呼びます。

なお，メッセージはメールボックスを介して，任意のタスクに送信することができますが，RX850におけるタスク間通信では，メッセージの先頭アドレスを受信側タスクに渡すだけであり，メッセージの内容がほかの領域にコピーされるわけではないので注意が必要です。

(1) メッセージ領域の確保

メッセージ用の領域は，get_blf，pget_blf，tget_blf，get_blk，pget_blkまたは，tget_blkシステム・コールを発行し，RX850が管理しているメモリ・プールから確保することを推奨しています。

また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このため，メモリ・プール以外の領域からメッセージ用の領域を確保する場合は，4バイト・アラインされた領域を使用しなければなりません。

注意　メッセージが対象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されているときにrel_blfまたはrel_blkシステム・コールが発行された場合，動作の保証はありません。

(2) メッセージ内容の作成

RX850では，メールボックスに対して送信するメッセージの長さ，内容についての規定は特にありません。ただし，メッセージの先頭4バイト，およびメッセージに優先度をつける場合は，さらに1バイトが予約領域となります。

メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。そして5バイト目は，メッセージの優先度を格納する領域です。つまりメッセージ本体はそれ以降の領域となります。メッセージの領域を確保するときは，これらの分も考慮に入れる必要があります。

注意1. **メッセージに優先度をつけない場合は，5バイト目からメッセージ本体として使用できます。**
2. **メッセージに優先度をつける場合，6バイト目からメッセージ本体となりますが，コンパイラの領域アラインの関係上，実質8バイト目からがメッセージ本体となります。**
3. **メッセージの優先度は0～31の範囲で指定できます。**

4. メッセージを対象メールボックスに送信する場合は，snd_msgシステム・コールを発行する前に，必ずメッセージの先頭4バイトに“0x0”を設定してください。
5. C言語によるプログラミングでメールボックスを使用するときは，RX850で用意してある「メッセージの構造体　T_MSG」を用いると便利です。詳しくは，10. 5. 3　同期通信機能システム・コールの項に記述したsnd_msgシステム・コールの説明をご参照ください。

4. 5. 5　メールボックス情報の獲得

メールボックス情報の獲得は，ref_mbxシステム・コールを発行することにより行われます。

(1) ref_mbxシステム・コール

パラメータで指定されたメールボックスのメールボックス情報（拡張情報，待ちタスクの有無など）を獲得します。

メールボックス情報の詳細を次に示します。

- 拡張情報
- 待ちタスクの有無

  FALSE（0）　：待ちタスクなし

  数値　　：待ちキューの先頭タスクのID番号
- 待ちメッセージの有無

  NADR（－1）：待ちメッセージなし

  数値　　：待ちキューの先頭メッセージのアドレス

4. 5. 6　メールボックスによるタスク間通信

メールボックスを使用してタスク間通信を行った場合の動作例を次に示します。

（前提条件）

- タスクの優先度

  タスクA＞タスクB
- タスクの状態

  タスクA：run状態

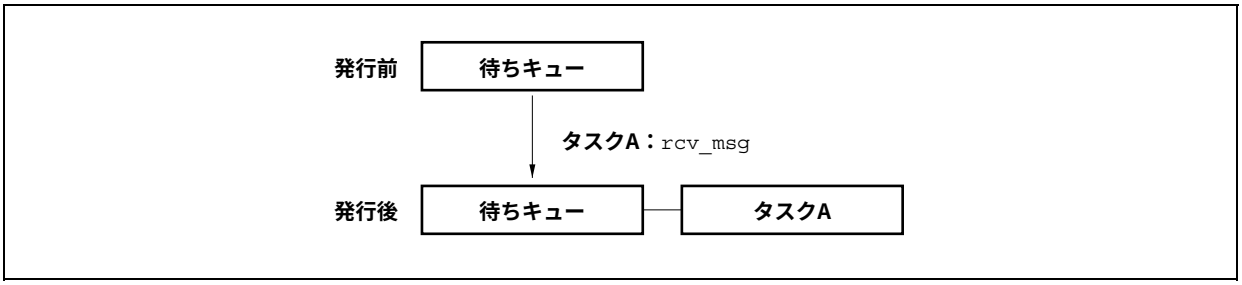
  タスクB：ready状態

(1) タスクAがrcv_msgシステム・コールを発行します。

現在，RX850が管理している対象メールボックスのメッセージ用待ちキューにはメッセージがキューイングされていません。このため，RX850は，タスクAをrun状態からwait状態（メッセージ待ち状態）へと遷移させたのち，対象メールボックスのタスク用待ちキューの最後尾にキューイングします。

図4－11に，対象メールボックスのタスク用待ちキューの状態を示します。

図4－11　タスク用待ちキューの状態



(2) タスクAのメッセージ待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

(3) タスクBがget_blfシステム・コールを発行します。

これにより，メモリ・プールからメッセージ用の領域（メモリ・ブロック）が確保されます。

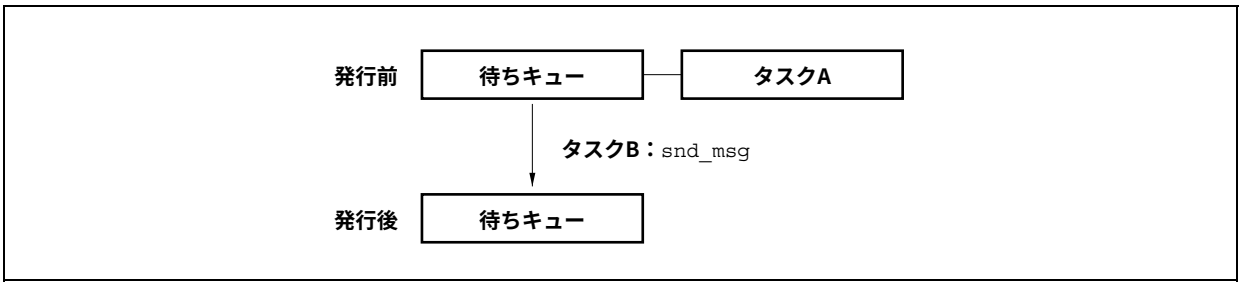
(4) タスクBがメモリ・ブロックにメッセージを書き込みます。

(5) タスクBがsnd_msgシステム・コールを発行します。

これにより，対象メールボックスのタスク用待ちキューにキューイングされているタスクAがメッセージ待ち状態からready状態へと遷移します。

図4－12に，対象メールボックスのタスク用待ちキューの状態を示します。

図4－12　タスク用待ちキューの状態



(6) 優先度の高いタスクAがready状態からrun状態へと遷移します。

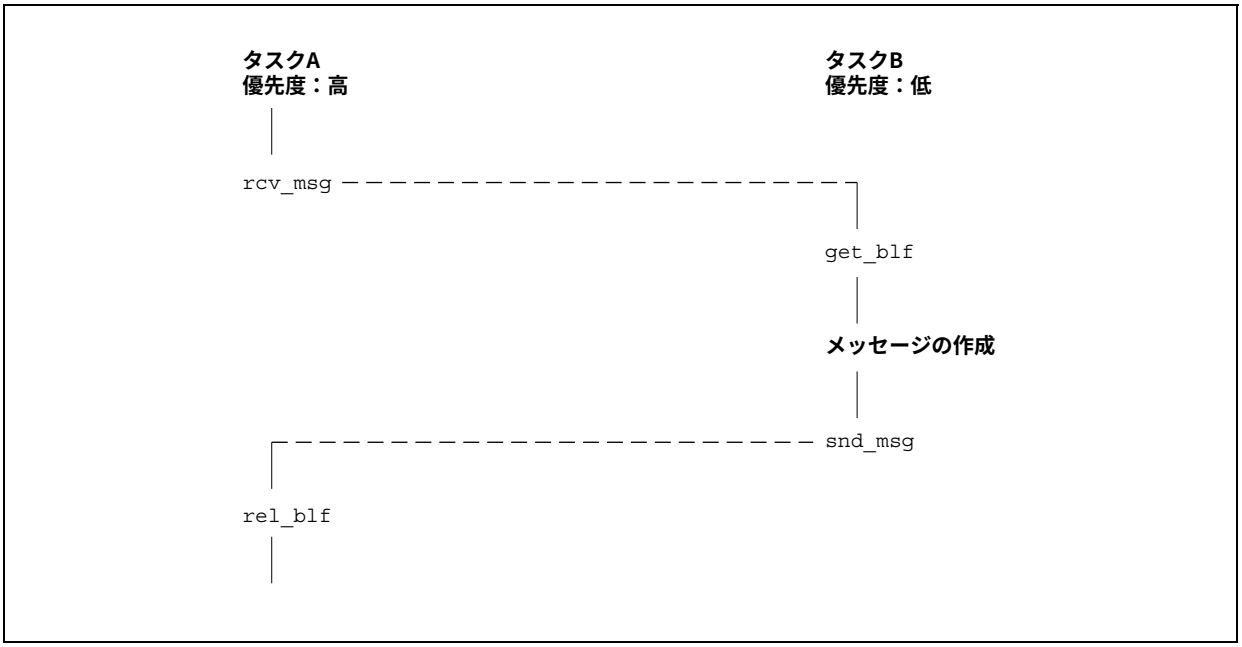
なお，タスクBはrun状態からready状態へと遷移します。

(7) タスクAがrel_blfシステム・コールを発行します。

これにより，メッセージ用の領域として確保されていたメモリ・ブロックがメモり・プールへ解放されます。

図4－13に，(1) ～ (7) のタスク間通信の流れを示します。

図4－13　メールボックスによるタスク間通信



〔メ　モ〕

第5章　割り込み管理機能

この章では，RX850が行う割り込み管理機能について示しています。

5.1 概　　要

RX850における割り込み管理では，次に示す処理が行われます。

- 割り込みハンドラの起動
- 割り込みハンドラからの復帰
- マスカブル割り込みの受け付け禁止／再開
- 割り込み制御レジスタの変更／獲得

5.2 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とき，ただちに起動される割り込み処理専用ルーチンであり，タスクとは独立したものとして扱われます。したがって，システム内で最高優先度を持つタスクが実行中であっても，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。

なお，RX850では，割り込みの発生から割り込みハンドラを起動するまでの応答性を考慮し，次の2種類の割り込みハンドラ用インタフェースを提供しています。

- 直接起動割り込みハンドラ
- 間接起動割り込みハンドラ

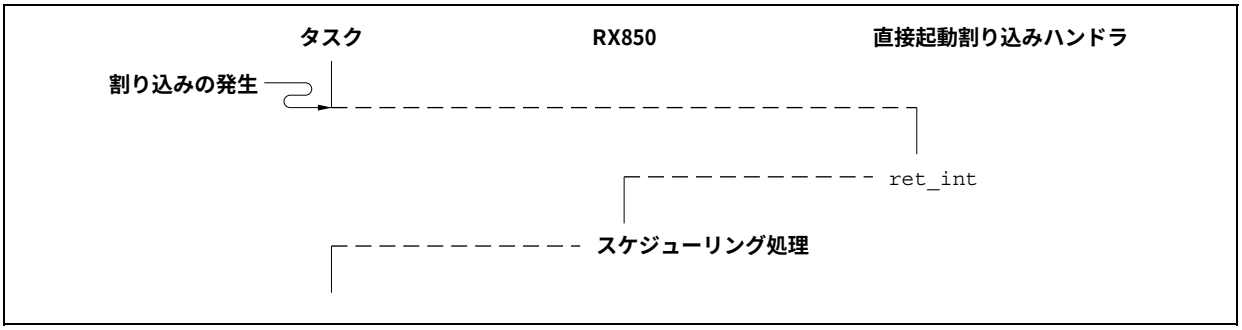
5.3 直接起動割り込みハンドラ

直接起動割り込みハンドラは，割り込みが発生したとき，RX850を介在させることなく起動される割り込み処理専用ルーチンです。

このため，ハードウエアの限界に近い高速な応答性が期待される処理プログラムです。

図5－1に，直接起動割り込みハンドラの動作の流れを示します。

図5－1　直接起動割り込みハンドラの動作の流れ



5. 3. 1　直接起動割り込みハンドラの登録

直接起動割り込みハンドラの登録は，割り込みが発生したときにプロセッサが制御を移すハンドラ・アドレスに直接起動割り込みハンドラを割り付ける，または直接起動割り込みハンドラへの分岐命令を設定することにより行われます。

5. 3. 2　直接起動割り込みハンドラ内での処理

直接起動割り込みハンドラの処理を記述するときには，次に示す注意事項があります。

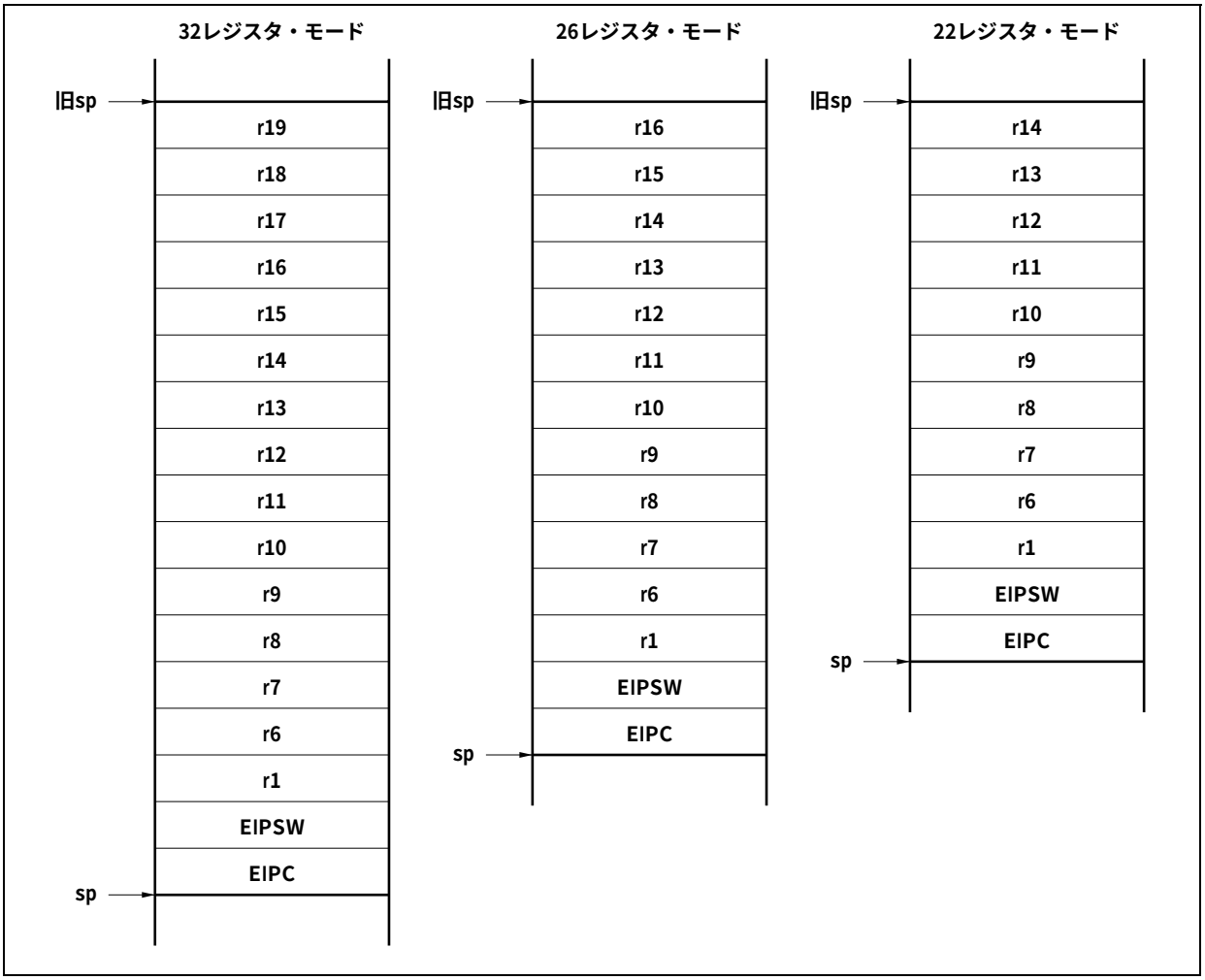
(1) レジスタの退避／復帰

直接起動割り込みハンドラに制御が移ったときの作業用レジスタの内容は，割り込み発生時のものとなります。したがって，直接起動割り込みハンドラ内で作業用レジスタを使用する場合は，直接起動割り込みハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要があります。

これらの処理はアセンブリ言語で記述する必要がありますが，RX850ではアセンブリ言語での記述におけるユーザの負担を軽減するため，直接起動割り込みハンドラ用マクロを用意しています。このマクロ内でレジスタの退避／復帰を行いますので，これを使用してください。このマクロについての詳細は，RX850 **ユーザーズ　マニュアル　インストレーション編**を参照してください。

図5－2に，直接起動割り込みハンドラ内でレジスタの退避処理を記述する場合のレジスタ退避順序を各レジスタ・モードごとに示します。

図5－2　レジスタの退避



(2) スタックの切り替え

直接起動割り込みハンドラに制御が移ったときのスタックは，割り込み発生時のスタックとなります。したがって，割り込みハンドラ用スタックを使用する場合は，直接起動割り込みハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要があります。ただし，直接起動割り込みハンドラ用マクロを使用した場合，マクロ内でスタックの切り替え処理を行っていますので，改めて記述する必要がなくなります。

(3) システム・コールの発行制限

直接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を次に示します。

・タスク管理機能システム・コール

sta_tsk	chg_pri	rot_rdq	rel_wai	get_tid
ref_tsk				

・タスク付属同期機能システム・コール

sus_tsk	rsm_tsk	frsm_tsk	wup_tsk	can_wup

・同期通信機能システム・コール

sig_sem	preq_sem	ref_sem	set_flg	clr_flg
pol_flg	ref_flg	vset_flg1	vclr_flg1	vpol_flg1
vref_flg1	snd_msg	prcv_msg	ref_mbx	

・割り込み管理機能システム・コール

ret_int	ret_wup	dis_int	ena_int	chg_icr
ref_icr				

・メモリ・プール管理機能システム・コール

pget_blf	rel_blf	ref_mpf	pget_blk	rel_blk
ref_mpl				

・時間管理機能システム・コール

act_cyc	ref_cyc

・システム管理機能システム・コール

get_ver	ref_sys

(4) 直接起動割り込みハンドラからの復帰処理

直接起動割り込みハンドラからの復帰処理は，直接起動割り込みハンドラの終了部分で，ret_intまたはret_wupシステム・コールを発行することにより行われます。

・ret_intシステム・コール

直接起動割り込みハンドラから復帰します。

・ret_wupシステム・コール

パラメータで指定されたタスクに起床要求を発行したのち，直接起動割り込みハンドラから復帰します。

なお，RX850では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_semなど）が発行されたときには，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（ret_intまたはret_wupシステム・コールの発行）まで遅延され，一括して行うようにしています。

注意　ret_int，ret_wupシステム・コールでは，外部割り込みコントローラに対する処理終了通知（EOIコマンドの発行）を行っていません。このため，外部割り込み要求によって起動した直接起動割り込みハンドラから復帰する場合は，このシステム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

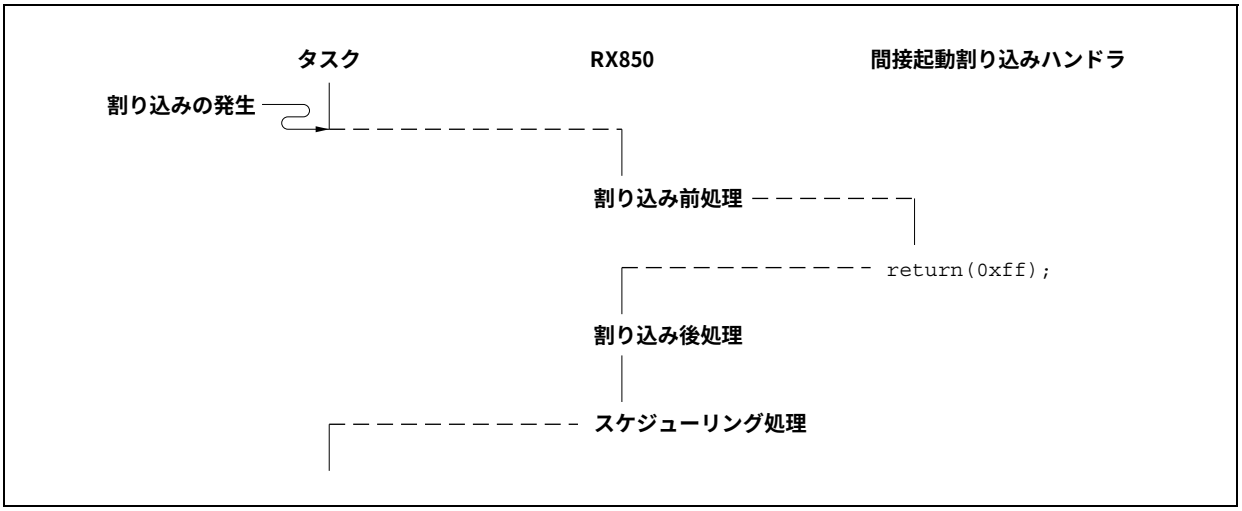
5.4　間接起動割り込みハンドラ

間接起動割り込みハンドラは，割り込みが発生したとき，RX850による割り込み前処理（レジスタの退避処理，スタックの切り替え処理など）を行わせたのち起動される割り込み処理専用ルーチンです。

このため，直接起動割り込みハンドラに比べて応答性の面では劣りますが，RX850による割り込み前処理が行われているため，ハンドラ内での処理が簡素化されるといった利点を持った処理プログラムです。

図5－3に，間接起動割り込みハンドラの動作の流れを示します。

図5－3　間接起動割り込みハンドラの動作の流れ



5.4.1　間接起動割り込みハンドラの登録

RX850では，システムで使用する間接起動割り込みハンドラをコンフィギュレーション時に指定します。つまり，RX850における間接起動割り込みハンドラの登録は，コンフィギュレーション時に指定した情報をもとに登録するスタティックな登録のみで，システム・コールによるダイナミックな登録はできません。

コンフィギュレーション時に指定する間接起動割り込みハンドラ情報を次に示します。

・割り込みレベル

・間接起動割り込みハンドラの起動アドレス

5.4.2　間接起動割り込みハンドラ内での処理

間接起動割り込みハンドラの処理を記述するときには，次に示す注意事項があります。

(1) レジスタの退避／復帰

RX850では，間接起動割り込みハンドラに制御を移すとき，および間接起動割り込みハンドラから復帰するとき，Cコンパイラ（CA850またはCCV850）の関数呼び出し規約に従った，作業用レジスタの退避処理／復帰処理を行っています。したがって，間接起動割り込みハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要がありません。

注意　RX850では，間接起動割り込みハンドラに制御を移すとき，gp，tp，epレジスタの切り替えを行っていません。

(2) スタックの切り替え

RX850では，間接起動割り込みハンドラに制御を移すとき，および間接起動割り込みハンドラから復帰するとき，スタックの切り替え処理を行っています。したがって，間接起動割り込みハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要がありません。

ただし，コンフィギュレーション時に割り込みハンドラ用スタックを定義していない場合は，スタックの切り替え処理が行われないため，割り込み発生時のスタックが使用されることになります。

(3) システム・コールの発行制限

間接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を次に示します。

・タスク管理機能システム・コール

```
sta_tsk     chg_pri     rot_rdq     rel_wai     get_tid
ref_tsk
```

・タスク付属同期機能システム・コール

```
sus_tsk     rsm_tsk     frsm_tsk    wup_tsk     can_wup
```

・同期通信機能システム・コール

```
sig_sem     preq_sem    ref_sem     set_flg     clr_flg
pol_flg     ref_flg     vset_flg1   vclr_flg1   vpol_flg1
vref_flg1   snd_msg     prcv_msg    ref_mbx
```

・割り込み管理機能システム・コール

```
dis_int     ena_int     chg_icr     ref_icr
```

・メモリ・プール管理機能システム・コール

```
pget_blf    rel_blf     ref_mpf     pget_blk    rel_blk
ref_mpl
```

・時間管理機能システム・コール

act_cyc　　ref_cyc

・システム管理機能システム・コール

get_ver　　ref_sys

(4) 間接起動割り込みハンドラからの復帰処理

間接起動割り込みハンドラからの復帰処理は，間接起動割り込みハンドラの終了部分で，return命令を発行することにより行われます。

- return（0xff）命令
  間接起動割り込みハンドラから復帰します。
- return（ID *tskid*）命令
  パラメータで指定されたタスクに起床要求を発行したのち，間接起動割り込みハンドラから復帰します。

なお，RX850では，間接起動割り込みハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_semなど）が発行されたときには，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，間接起動割り込みハンドラからの復帰処理（return命令の発行）まで遅延され，一括して行うようにしています。

注意　return命令では，外部割り込みコントローラに対する処理終了通知（EOIコマンドの発行）が行われません。このため，外部割り込み要求によって起動した間接起動割り込みハンドラから復帰する場合は，return命令を発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

5.5　マスカブル割り込みの受け付け禁止／再開

RX850では，ユーザの処理プログラムからマスカブル割り込みの受け付け状態を操作し，マスカブル割り込みの受け付けを禁止／再開する機能が提供されています。

なお，この機能は次に示すシステム・コールをタスク，またはハンドラ内から発行することにより実現されます。

(1) loc_cpuシステム・コール

マスカブル割り込みの受け付けを禁止したのち，ディスパッチ処理（タスクのスケジューリング処理）も禁止します。

これにより，このシステム・コールの発行からunl_cpuシステム・コールが発行されるまでの間，他タスク，ハンドラに制御が移ることはありません。

(2) unl_cpuシステム・コール

マスカブル割り込みの受け付けを許可したのち，ディスパッチ処理（タスクのスケジューリング処理）も再開します。

これにより，loc_cpuシステム・コール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が許可されるとともに，ディスパッチ処理も再開されます。

(3) dis_intシステム・コール

マスカブル割り込みの受け付けを禁止します。

これにより，このシステム・コールの発行からena_intシステム・コールが発行されるまでの間，ハンドラに制御が移ることはありません。

(4) ena_intシステム・コール

マスカブル割り込みの受け付けを再開します。

これにより，dis_intシステム・コール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が再開されます。

図5－4から図5－6に，通常の場合，loc_cpuシステム・コールを発行した場合，dis_intシステム・コールを発行した場合の制御の流れを示します。

図5－4　通常の場合

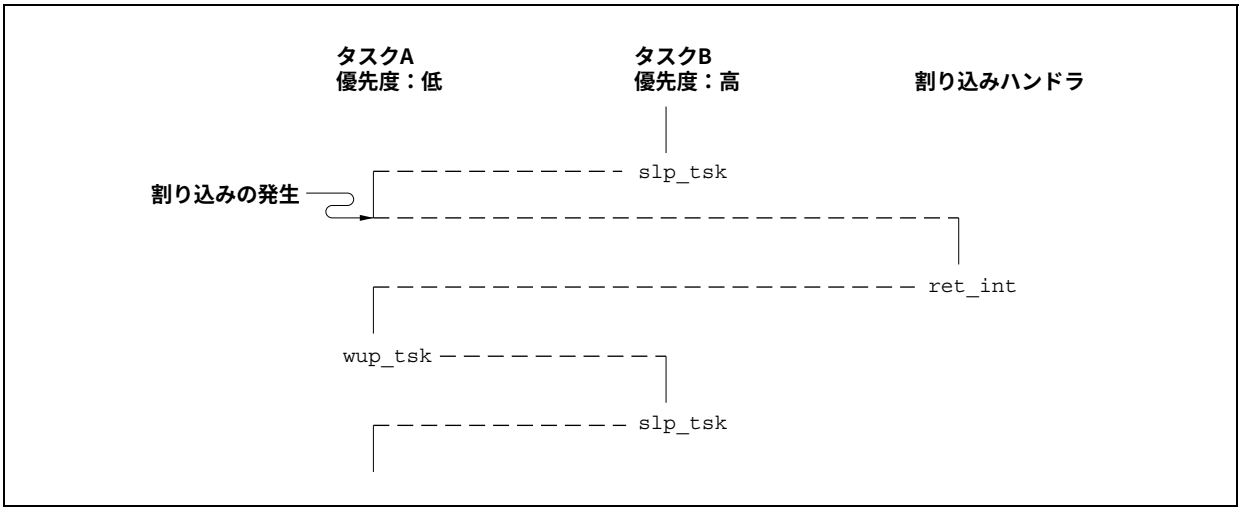


図5－5　loc_cpuシステム・コールを発行した場合

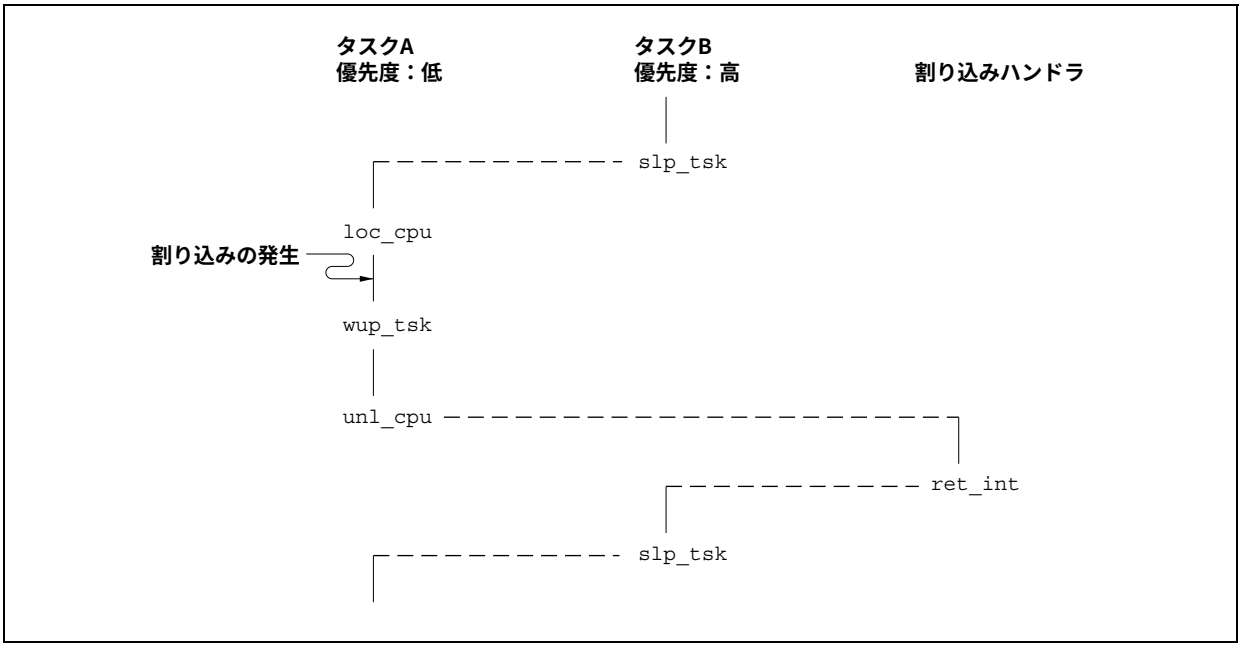
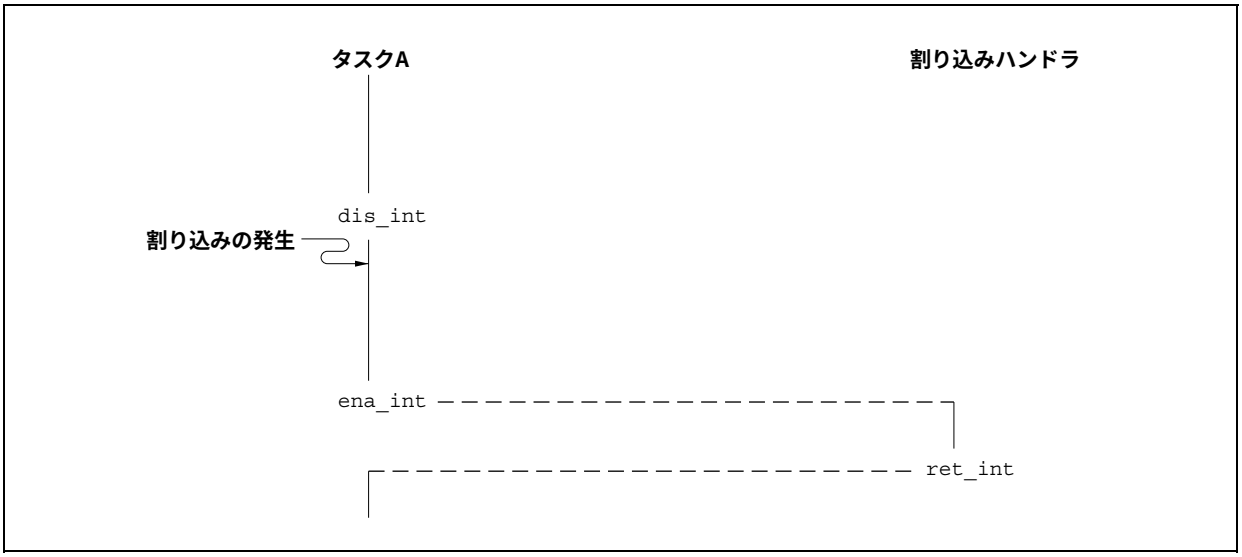


図5－6　dis_intシステム・コールを発行した場合



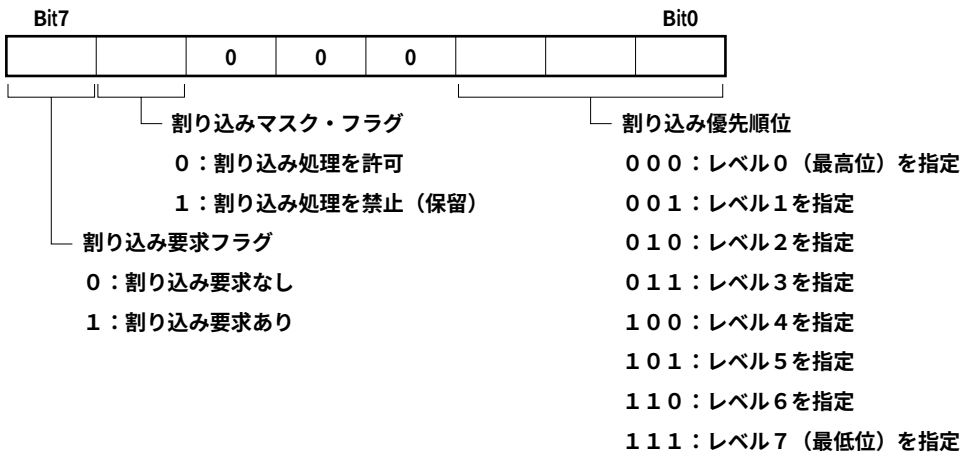
注意　ena_int, dis_intシステム・コールは，発行したタスク内で割り込みの許可／禁止をします。たとえば，タスクAでdis_intシステム・コール発行後，マスカブル割り込みが発生すると，割り込み処理は保留されます。しかし，そのあとディスパッチを起こすようなシステム・コールを発行してタスクBに処理が移ったとき，タスクBでマスカブル割り込みが禁止されていなければ（タスクBでdis_intシステム・コールを発行していなければ）割り込みが受け付けられます。すなわち，タスクAで保留されていた割り込み処理が，タスクBに移った直後に行われることになります。
再びタスクAにディスパッチされると，マスカブル割り込み禁止状態で再開します。

5.6　割り込み制御レジスタの変更／獲得

割り込み制御レジスタの内容の変更／獲得は，chg_icrまたはref_icrシステム・コールを発行することにより行われます。

- chg_icrシステム・コール
  パラメータで指定された割り込み制御レジスタの内容を変更します。
- ref_icrシステム・コール
  パラメータで指定された割り込み制御レジスタの内容を獲得します。
  図5－7に，獲得した割り込み制御レジスタの内容を示します。

図5－7　割り込み制御レジスタの内容



5.7　ノンマスカブル割り込み

ノンマスカブル割り込みは，割り込み優先順位の対象外となっており，すべての割り込みに優先して受け付けられます。また，プロセッサを割り込み禁止状態（PSWのIDフラグを設定）にしても受け付けられる割り込みです。

このため，RX850の処理中，および割り込みハンドラの処理中であっても，ノンマスカブル割り込みは受け付けられることになります。

そこで，RX850では，ノンマスカブル割り込みに対応した割り込みハンドラ内でシステム・コールを発行した場合，その動作を保証していません。

5.8　クロック割り込み

RX850では，ハードウエア（リアルタイム・パルス・ユニットなど）により一定周期で発生するクロック割り込みを利用して，時間管理を行っています。

つまり，クロック割り込みが発生したときには，RX850の時間管理用割り込みハンドラ（クロック・ハンドラ）が起動し，タスクの時間経過待ち，周期起動ハンドラの起動などといった時間に関連した処理が行われます。

なお，時間管理についての詳細は，**第7章　時間管理機能**を参照してください。

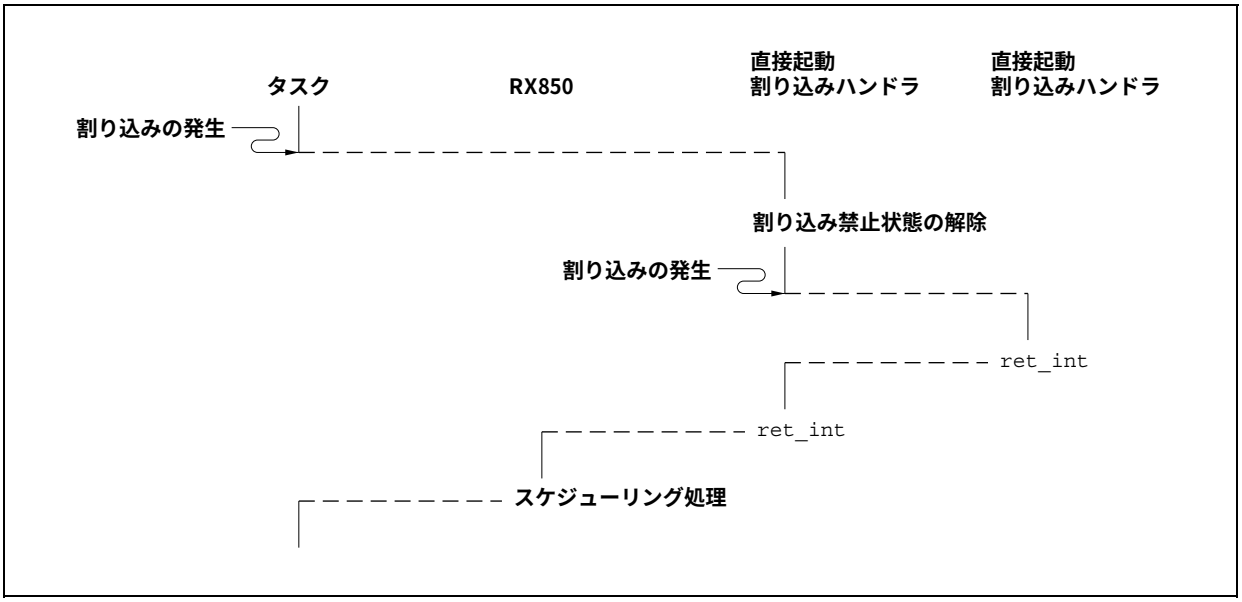
5.9　多重割り込み

RX850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」**と呼びます。

ただし，割り込みハンドラは，割り込み禁止状態（PSWのIDフラグを設定）でその処理が開始されるため，多重割り込みを受け付けるには，割り込みハンドラ内で割り込み禁止状態の解除処理を記述する必要があります。

図5－8に，多重割り込み発生時の動作の流れを示します。

図5－8　多重割り込み発生時の動作の流れ



第6章　メモリ・プール管理機能

この章では，RX850が行うメモリ・プール管理機能について示しています。

6.1　概　　要

RX850では，システム全体を管理するための情報テーブル，各種機能（セマフォ，イベント・フラグなど）を実現するための管理ブロックなどの管理オブジェクトをシステム初期化処理時にスタティックに生成／初期化しています。

また，RX850では，ダイナミックなメモリ・プール管理も行っており，作業用のメモリ領域が必要となったときに獲得し，不要となったときには解放できるという機能が用意されています。ユーザは，このようなダイナミックなメモリ操作を行うことにより，限りあるメモリ領域を効率よく使用することが可能となります。

6.2　管理オブジェクト

RX850がシステム全体を管理するうえで必要となる管理オブジェクト，およびRX850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を次に示します。

なお，これら管理オブジェクトは，コンフィギュレーション時に指定した情報（システム情報，タスク情報など）をもとに，システム初期化処理時に生成／初期化されます。

- システム・ベース・テーブル
- タスク実行権グループ管理ブロック
- タスク管理ブロック
- セマフォ管理ブロック
- イベント・フラグ管理ブロック
- 1ビット・イベント・フラグ管理ブロック
- メールボックス管理ブロック
- 固定長メモリ・プール管理ブロック
- 可変長メモリ・プール管理ブロック
- 周期起動ハンドラ管理ブロック
- メモリ・プール
- タスク用スタック領域
- 割り込みハンドラ用スタック領域

図6－1に，管理オブジェクトの配置イメージを示します。

図6－1　管理オブジェクトの配置イメージ

6.3　固定長メモリ・プール

RX850では，処理プログラム（タスク，ハンドラなど）からメモリ領域に対する要求が行われたときに使用する領域として固定長メモリ・プールを用意しています。

なお，固定長メモリ・プールは，複数の固定長メモリ・ブロックから構成されており，固定長メモリ・プールに対する操作は，固定長メモリ・ブロックを単位として行います。

また，固定長メモリ・プールに対するダイナミックな操作は，次に示す固定長メモリ・プール関連のシステム・コールを用いて行います。

get_blf　：固定長メモリ・ブロックを獲得する
pget_blf：固定長メモリ・ブロックを獲得する（ポーリング）
tget_blf：固定長メモリ・ブロック獲得する（タイムアウトあり）
rel_blf　：固定長メモリ・ブロックを返却する

注意1. 固定長メモリ・ブロックの獲得／返却といった操作は，1ブロックを単位として行われます。

2. 固定長メモリ・プール関連のシステム・コールでは，どの固定長メモリ・プールに対して操作を行うのか（対象固定長メモリ・プール）を指定するとき，ID番号を使用します。
このID番号は，システム情報テーブルに記述された固定長メモリ・プール情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各固定長メモリ・プールごとに順次割り付けたものです。

6.3.1　固定長メモリ・プールの生成

RX850では，システムで使用する固定長メモリ・プールをコンフィギュレーション時に指定します。つまり，RX850における固定長メモリ・プールの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）だけで，システム・コールによるダイナミックな生成はできません。

なお，RX850における固定長メモリ・プールの生成は，固定長メモリ・プールを管理するための領域（管理オブジェクト），および固定長メモリ・プール用の領域を確保したのち，初期化することです。

RX850では，このようにしてメモリ上に確保された領域を固定長メモリ・プールとして認識し，管理対象とします。

コンフィギュレーション時に指定する固定長メモリ・プール情報を次に示します。

- 固定長メモリ・プール名
- 固定長メモリ・プールのサイズ
- 固定長メモリ・プールを割り付けるシステム・メモリの種別（.pool0, .pool1セクション）

6.3.2　固定長メモリ・ブロックの獲得

固定長メモリ・ブロックの獲得は，get_blf, pget_blf, またはtget_blfシステム・コールを発行することにより行われます。

(1) get_blfシステム・コール

パラメータで指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロッ

クを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象固定長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（固定長メモリ・ブロック待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック待ち状態の解除は次の場合に行われ，固定長メモリ・ブロック待ち状態からready状態へと遷移します。

- rel_blfシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に固定長メモリ・ブロック待ち状態を解除した

注意1. タスクを対象固定長メモリ・プールの待ちキューにキューイングするときのキューイング方法は，固定長メモリ・ブロックの獲得要求を行った順（FIFO順）に行われます。

2. RX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。したがって，4バイト以降の内容については不定となります。

(2) pget_blfシステム・コール

パラメータで指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

注意　RX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。したがって，4バイト以降の内容については不定となります。

(3) tget_blfシステム・コール

パラメータで指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象固定長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（固定長メモリ・ブロック待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック待ち状態の解除は次の場合に行われ，固定長メモリ・ブロック待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- rel_blfシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的にメモリ・ブロック待ち状態を解除した

注意1. タスクを対象固定長メモリ・プールの待ちキューにキューイングするときのキューイング方法は，固定長メモリ・ブロックの獲得要求を行った順（FIFO順）に行われます。

2. RX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。したがって，4バイト以降の内容については不定となります。

6.3.3　固定長メモリ・ブロックの返却

固定長メモリ・ブロックの返却は，rel_blfシステム・コールを発行することにより行われます。

(1) rel_blf

パラメータで指定された固定長メモリ・プールに固定長メモリ・ブロックを返却します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却処理は行わず，該当するタスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（固定長メモリ・ブロック待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意1. RX850では，固定長メモリ・ブロックを返却するとき，メモリ・クリアを行っていません。したがって，返却した固定長メモリ・ブロックの内容は不定となります。

2. 固定長メモリ・ブロックの返却は，get_blf, pget_blf, tget_blfシステム・コールを発行したときに指定した固定長メモリ・プールと同一でなければなりません。

6.3.4　固定長メモリ・プール情報の獲得

固定長メモリ・プール情報の獲得は，ref_mpfシステム・コールを発行することにより行われます。

(1) ref_mpfシステム・コール

パラメータで指定された固定長メモリ・プールの固定長メモリ・プール情報（拡張情報，待ちタスクの有無など）を獲得します。

固定長メモリ・プール情報の詳細を次に示します。

- 拡張情報
- 待ちタスクの有無

  FALSE（0）：待ちタスクなし

  数値　　：待ちキューの先頭タスクのID番号
- 空き領域のブロック数

6.4　可変長メモリ・プール

RX850では，処理プログラム（タスク，ハンドラなど）からメモリ領域に対する要求が行われたときに使用する領域として可変長メモリ・プールを用意しています。

なお，可変長メモリ・プールは，複数の可変長メモリ・ブロックから構成されており，可変長メモリ・プールに対する操作は，可変長メモリ・ブロックを単位として行います。

また，可変長メモリ・プールに対するダイナミックな操作は，次に示す可変長メモリ・プール関連のシステム・コールを用いて行います。

get_blk　：可変長メモリ・ブロックを獲得する

pget_blk：可変長メモリ・ブロックを獲得する（ポーリング）

tget_blk：可変長メモリ・ブロックを獲得する（タイムアウトあり）

rel_blk　：可変長メモリ・ブロックを返却する

注意1. **可変長メモリ・ブロックの獲得／返却といった操作は，**1**ブロックを単位として行われます。**

2. **可変長メモリ・プール関連のシステム・コールでは，どの可変長メモリ・プールに対して操作を行うのか（対象可変長メモリ・プール）を指定するとき，**ID**番号を使用します。**
このID**番号は，システム情報テーブルに記述された可変長メモリ・プール情報をもとに，コンフィギュレータ**CF850**が**0x1**から始まる整数値を各可変長メモリ・プールごとに順次割り付けたものです。**

6.4.1　可変長メモリ・プールの生成

RX850**では，システムで使用する可変長メモリ・プールをコンフィギュレーション時に指定します。つまり，**RX850**における可変長メモリ・プールの生成は，コンフィギュレーション時に指定した情報をもとに生成するスタティックな生成（システム初期化処理時の生成）のみで，システム・コールによるダイナミックな生成はできません。**

なお，RX850**における可変長メモリ・プールの生成は，可変長メモリ・プールを管理化するための領域（管理オブジェクト），および可変長メモリ・プール用の領域を確保したのち，初期化することです。**

RX850**では，このようにしてメモリ上に確保された領域を可変長メモリ・プールとして認識し，管理対象とします。**

コンフィギュレーション時に指定する可変長メモリ・プール情報を次に示します。

- **可変長メモリ・プール名**
- **可変長メモリ・プールのサイズ**
- **可変長メモリ・プールを割り付けるシステム・メモリの種別（**.pool0, .pool1**セクション）**

6.4.2　可変長メモリ・ブロックの獲得

可変長メモリ・ブロックの獲得は，get_blk, pget_blk, **または**tget_blk**システム・コールを発行することにより行われます。**

(1) get_blk**システム・コール**

パラメータで指定された可変長メモリ・プールから可変長メモリ・ブロックを獲得します。

ただし，本システム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象可変長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run**状態から**wait**状態（可変長メモリ・ブロック待ち状態）へと遷移させます。**

なお，可変長メモリ・ブロック待ち状態の解除は次の場合に行われ，可変長メモリ・ブロック待ち状態からready**状態へと遷移します。**

- rel_blk**システム・コールが発行された**
- rel_wai**システム・コールが発行され，強制的に可変長メモリ・ブロック待ち状態を解除した**

注意1. **タスクを対象可変長メモリ・プールの待ちキューにキューイングするときのキューイング方法は，可変長メモリ・ブロックの獲得要求を行った順（**FIFO**順）に行われます。**

2. RX850**では，可変長メモリ・ブロックを獲得するとき，先頭**4**バイトのみをクリアしています。したがって，**4**バイト以降の内容については不定となります。**

(2) pget_blkシステム・コール

パラメータで指定された可変長メモリ・プールから可変長メモリ・ブロックを獲得します。

ただし，本システム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

注意　RX850では，可変長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。したがって，4バイト以降の内容については不定となります。

(3) tget_blkシステム・コール

パラメータで指定された可変長メモリ・プールから可変長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象可変長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（可変長メモリ・ブロック待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック待ち状態の解除は次の場合に行われ，可変長メモリ・ブロック待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- rel_blkシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に可変長メモリ・ブロック待ち状態を解除した

注意1. タスクを対象可変長メモリ・プールの待ちキューにキューイングするときのキューイング方法は，可変長メモリ・ブロックの獲得要求を行った順（FIFO順）に行われます。

2. RX850では，可変長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。したがって，4バイト以降の内容については不定となります。

6.4.3　可変長メモリ・ブロックの返却

可変長メモリ・ブロックの返却は，rel_blkシステム・コールを発行することにより行われます。

(1) rel_blk

パラメータで指定された可変長メモリ・プールに可変長メモリ・ブロックを返却します。

ただし，このシステム・コールを発行したとき，対象可変長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，可変長メモリ・ブロックの返却処理は行わず，該当タスク（待ちキューの先頭タスク）に可変長メモリ・ブロックを渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（可変長メモリ・ブロック待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意1. RX850では，可変長メモリ・ブロックを返却するとき，メモリ・クリアを行っていません。したがって，返却した可変長メモリ・ブロックの内容は不定となります。

2. 可変長メモリ・ブロックの返却は，get_blk, pget_blk, tget_blkシステム・コールを発行したときに指定した可変長メモリ・プールと同一でなければなりません。

6.4.4　可変長メモリ・プール情報の獲得

可変長メモリ・プール情報の獲得は，ref_mplシステム・コールを発行することにより行われます。

(1) ref_mplシステム・コール

パラメータで指定された可変長メモリ・プールの可変長メモリ・プール情報（拡張情報，待ちタスクの有無など）を獲得します。

可変長メモリ・プール情報の詳細を次に示します。

- 拡張情報
- 待ちタスクの有無

  FALSE（0）：待ちタスクなし

  数値　　：待ちキューの先頭タスクのID番号
- 空き領域のブロック数

第7章　時間管理機能

この章では，RX850が行う時間管理機能について示しています。

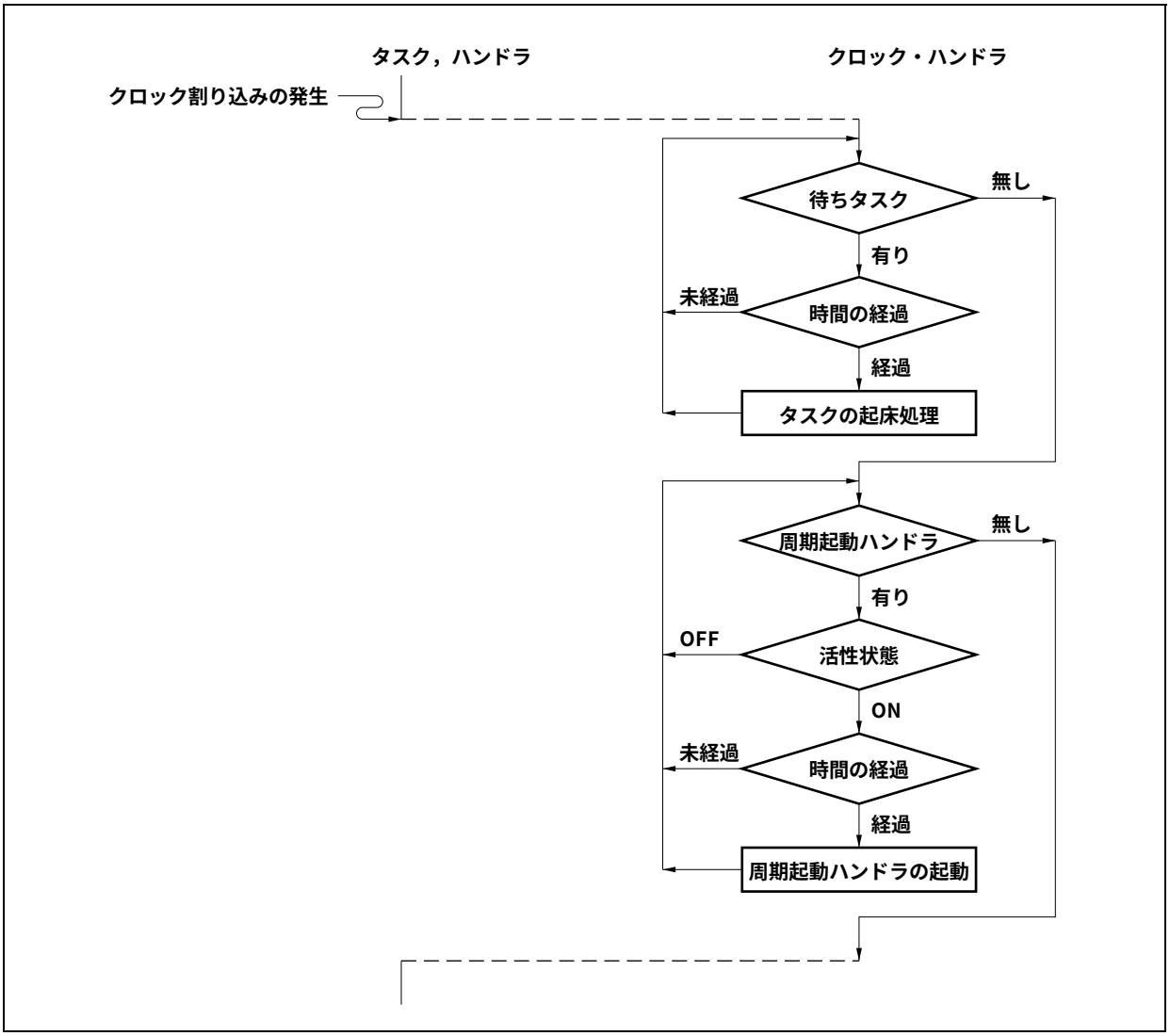
7.1 概　　要

RX850における時間管理は，ハードウエア（リアルタイム・パルス・ユニットなど）により一定周期で発生するクロック割り込みを利用して行います。

つまり，クロック割り込みが発生したときには，RX850の時間管理用割り込みハンドラ（クロック・ハンドラ）が起動し，タスクの時間経過待ち，周期起動ハンドラの起動などの時間に関連した処理が行われます。

図7－1に，クロック・ハンドラで行われるおもな処理の流れを示します。

図7－1　クロック・ハンドラの処理の流れ



7.2　タイマ・オペレーション

リアルタイム処理では，あるタスクの処理を一定時間だけ中断したり，あるハンドラの処理を一定周期で実行したりというような時間と同期した機能（タイマ・オペレーション機能）が必要となります。そこで，RX850では，タイマ・オペレーション機能として，タスクの遅延起床，タイムアウト，周期起動ハンドラの起動といった機能を提供しています。

7.3　タスクの遅延起床

タスクの遅延起床は，一定の時間が経過するまでの間，タスクをrun状態からwait状態（時間経過待ち状態）へと遷移させ，時間が経過したときには，wait状態を解除し，ready状態へと遷移させるものです。

なお，タスクの遅延起床は，dly_tskシステム・コールを発行することにより行われます。

(1) dly_tskシステム・コール

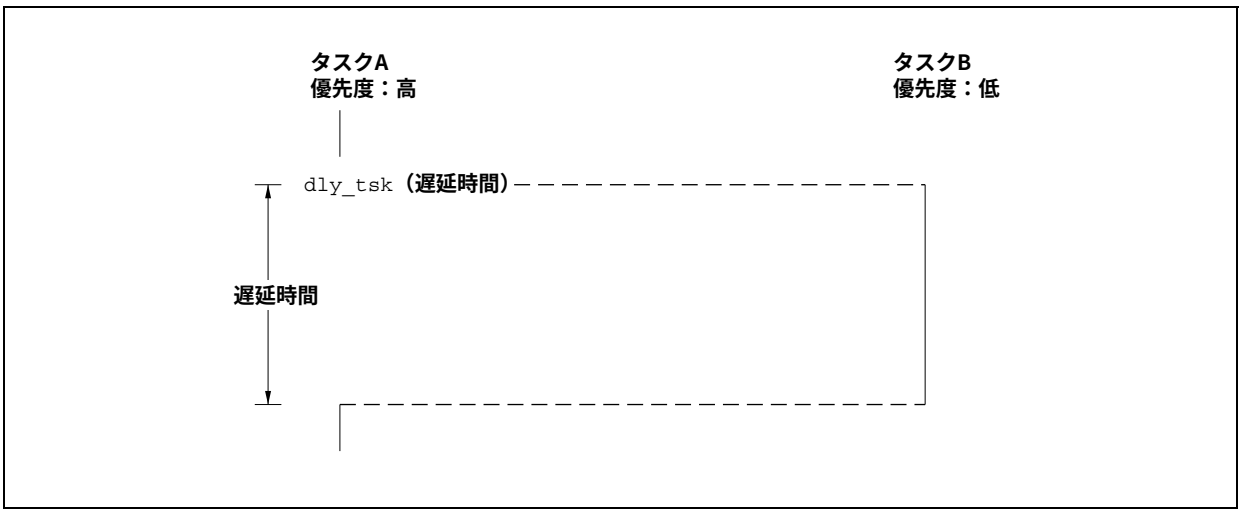
パラメータで指定された遅延時間だけ，自タスクをrun状態からwait状態（時間経過待ち状態）へと遷移させます。

なお，時間経過待ち状態の解除は次の場合に行われ，時間経過待ち状態からready状態へと遷移します。

- パラメータで指定された遅延時間が経過した
- rel_waiシステム・コールが発行され，強制的に時間経過待ち状態を解除した

図7－2に，dly_tskシステム・コールを発行したときの処理の流れを示します。

図7－2　dly_tskシステム・コール発行時の処理の流れ



7.4　タイムアウト

タイムアウトは，要求する条件が即時に成立しなかった場合，一定の時間が経過するまでの間，タスクをrun状態からwait状態（起床待ち状態，資源待ち状態など）へと遷移させ，時間が経過したときには，wait状態を解除し，ready状態へと遷移させるものです。

なお，タイムアウトは，tslp_tsk, twai_sem, twai_flg, vtwai_flg1, trcv_msg, tget_blf, tget_blkシステム・コールを発行することにより行われます。

(1) tslp_tskシステム・コール

自タスクに発行されている起床要求を1回分だけ解除（起床要求カウンタから0x1を減算）します。

ただし，このシステム・コールを発行したとき，自タスクの起床要求カウンタが0x0であった場合には，起床要求の解除（起床要求カウンタの減算処理）は行わず，自タスクをrun状態からwait状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態の解除は次の場合に行われ，起床待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- wup_tskシステム・コールが発行された
- ret_wupシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に起床待ち状態を解除した

(2) twai_semシステム・コール

パラメータ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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は次の場合に行われ，資源待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- sig_semシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に資源待ち状態を解除した

(3) twai_flgシステム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンが設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューにキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は次の場合に行われ，イベント・フラグ待ち状態からready状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- set_flgシステム・コールが発行され，要求する待ち条件を設定した
- rel_waiシステム・コールが発行され，強制的にイベント・フラグ待ち状態を解除した

(4) vtwai_flg1システム・コール

パラメータで指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，自タスクを対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は次の場合に行われ，イベント・フラグ待ち状態からready状態へと遷移します。

・パラメータで指定された待ち時間が経過した

・vset_flg1システム・コールが発行された

・rel_waiシステム・コールが発行され，強制的にイベント・フラグ待ち状態を解除した

(5) trcv_msgシステム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行したとき，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（メッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたのち，run状態からwait状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態の解除は次の場合に行われ，メッセージ待ち状態からready状態へと遷移します。

・パラメータで指定された待ち時間が経過した

・snd_msgシステム・コールが発行された

・rel_waiシステム・コールが発行され，強制的にメッセージ待ち状態を解除した

(6) tget_blfシステム・コール

パラメータで指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象固定長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（固定長メモリ・ブロック待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック待ち状態の解除は次の場合に行われ，固定長メモリ・ブロック待ち状態からready状態へと遷移します。

・パラメータで指定された待ち時間が経過した

・rel_blfシステム・コールが発行さた

・rel_waiシステム・コールが発行され，強制的に固定長メモリ・ブロック待ち状態を解除した

(7) tget_blkシステム・コール

パラメータで指定された可変長メモリ・プールから可変長メモリ・ブロックを獲得します。

ただし，このシステム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象可変長メモリ・

プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（可変長メモリ・ブロック待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック待ち状態の解除は次の場合に行われ，可変長メモリ・ブロック待ち状態からready状態へと遷移します。

- **パラメータで指定した待ち時間が経過した**
- rel_blkシステム・コールが発行された
- rel_waiシステム・コールが発行され，強制的に可変長メモリ・ブロック待ち状態を解除した

7.5　周期起動ハンドラ

周期起動ハンドラは，一定の起動周期に達したとき，ただちに起動される周期処理専用ルーチンであり，タスクとは独立したものとして扱われます。このため，起動周期に達したときには，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，周期起動ハンドラに制御が移ります。

なお，周期起動ハンドラは，ユーザが記述する周期的な処理プログラムの中で，実行開始までのRX850によるオーバヘッドが最も小さな処理プログラムです。

7.5.1　周期起動ハンドラの登録

RX850では，システムで使用する周期起動ハンドラをコンフィギュレーション時に指定します。つまり，RX850における周期起動ハンドラの登録は，コンフィギュレーション時に指定した情報をもとに登録するスタティックな登録（システム初期化処理時の登録）だけで，システム・コールによるダイナミックな登録はできません。

なお，RX850における周期起動ハンドラの登録は，周期起動ハンドラを管理するための領域（管理オブジェクト）を確保したのち，初期化することです。

以下に，コンフィギュレーション時に指定する周期起動ハンドラ情報を示します。

- 周期起動ハンドラ名
- 周期起動ハンドラの起動アドレス
- 周期起動ハンドラの初期活性状態
- 周期起動ハンドラの起動間隔

注意　周期起動ハンドラ関連のシステム・コールでは，どの周期起動ハンドラに対して操作を行うのか（対象周期起動ハンドラ）を指定するとき，指定番号を使用します。
この指定番号は，システム情報テーブルに記述された周期起動ハンドラ情報をもとに，コンフィギュレータCF850が0x1から始まる整数値を各周期起動ハンドラごとに順次割り付けたものです。

7.5.2　周期起動ハンドラの活性状態

周期起動ハンドラの活性状態は，RX850が周期起動ハンドラを起動するかどうかを判定するときの基準の1つです。

なお，活性状態は，コンフィギュレーション時に指定された情報をもとに，システム初期化処理時にいったん設定されますが，RX850では，この活性状態をユーザの処理プログラムからダイナミックに変更する機能を提供しています。

・act_cycシステム・コール

パラメータで指定された周期起動ハンドラの活性状態を変更します。

TCY_OFF　：周期起動ハンドラの活性状態をオフ状態に変更する。

TCY_ON　：周期起動ハンドラの活性状態をオン状態に変更する。

TCY_INI　：周期起動ハンドラをタイマ・キューにキューイングしたのち，周期カウンタを初期化する。

TCY_ULNK：周期起動ハンドラの活性状態をオフ状態に変更したのち，タイマ・キューから外す。

RX850では，周期起動ハンドラがタイマ・キューにキューイングされている間は，活性状態がオフ状態でも，周期カウンタのカウント処理が行われます。このため，act_cycシステム・コールを発行し，活性状態をオフ状態からオン状態に変更した場合，1回目の起動要求が発行されるまでの間隔は，コンフィギュレーション時に指定した起動周期よりも短くなる可能性があります。そこで，1回目の起動要求がコンフィギュレーション時に指定した起動周期で発行されるためには，act_cycシステム・コールを発行するとき，周期起動ハンドラの起動再開TCY_ONのほかに，周期カウンタの初期化TCY_INIをあわせて指定する必要があります。

図7－3，図7－4に，act_cycシステム・コールを発行することにより，周期起動ハンドラの活性状態をオフ状態からオン状態に変更したときの処理の流れを示します。

なお，図7－3，図7－4におけるΔTは，コンフィギュレーション時に指定した周期起動ハンドラの起動周期であるものとし，図7－3におけるΔtとΔTの関係はΔt≦ΔTであるものとします。

図7－3　act_cyc（TCY_ON）発行時の処理の流れ

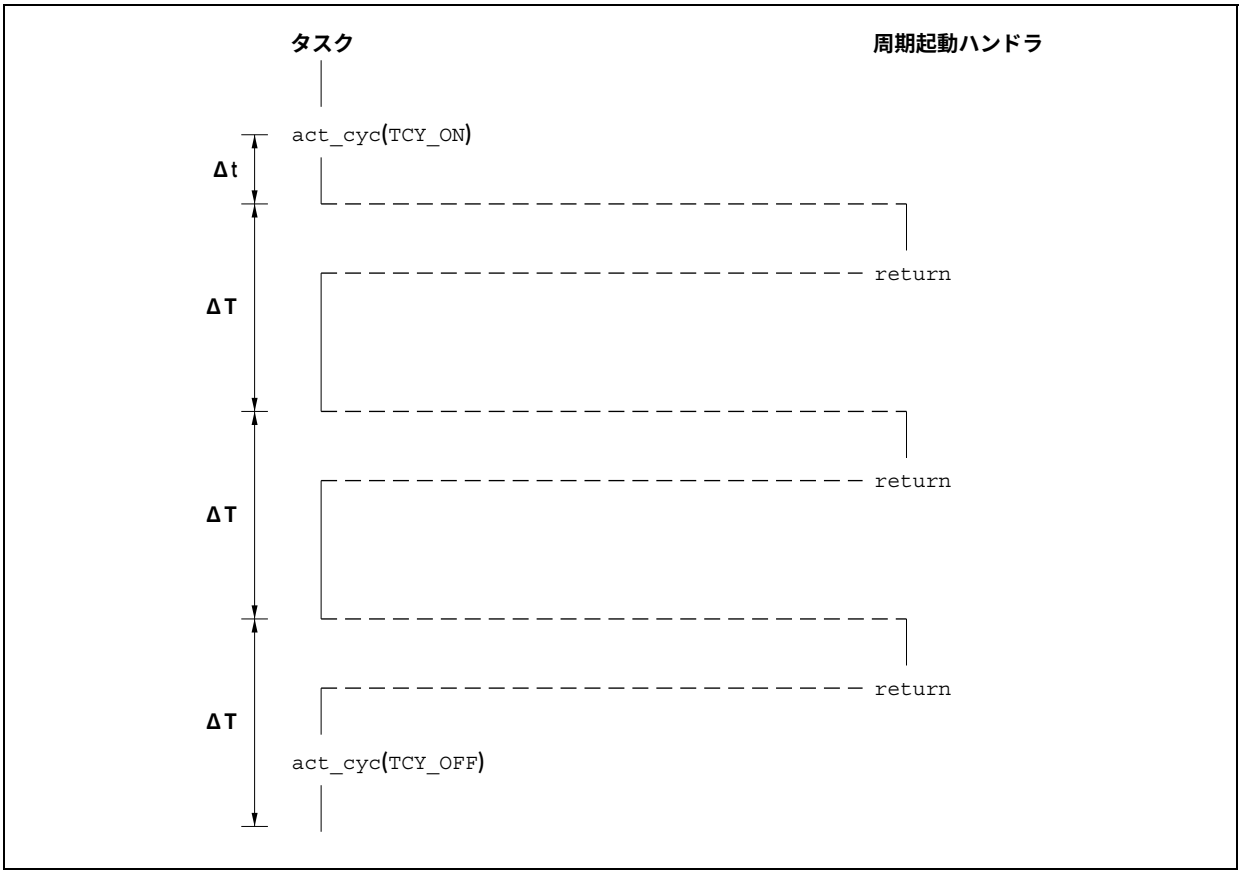
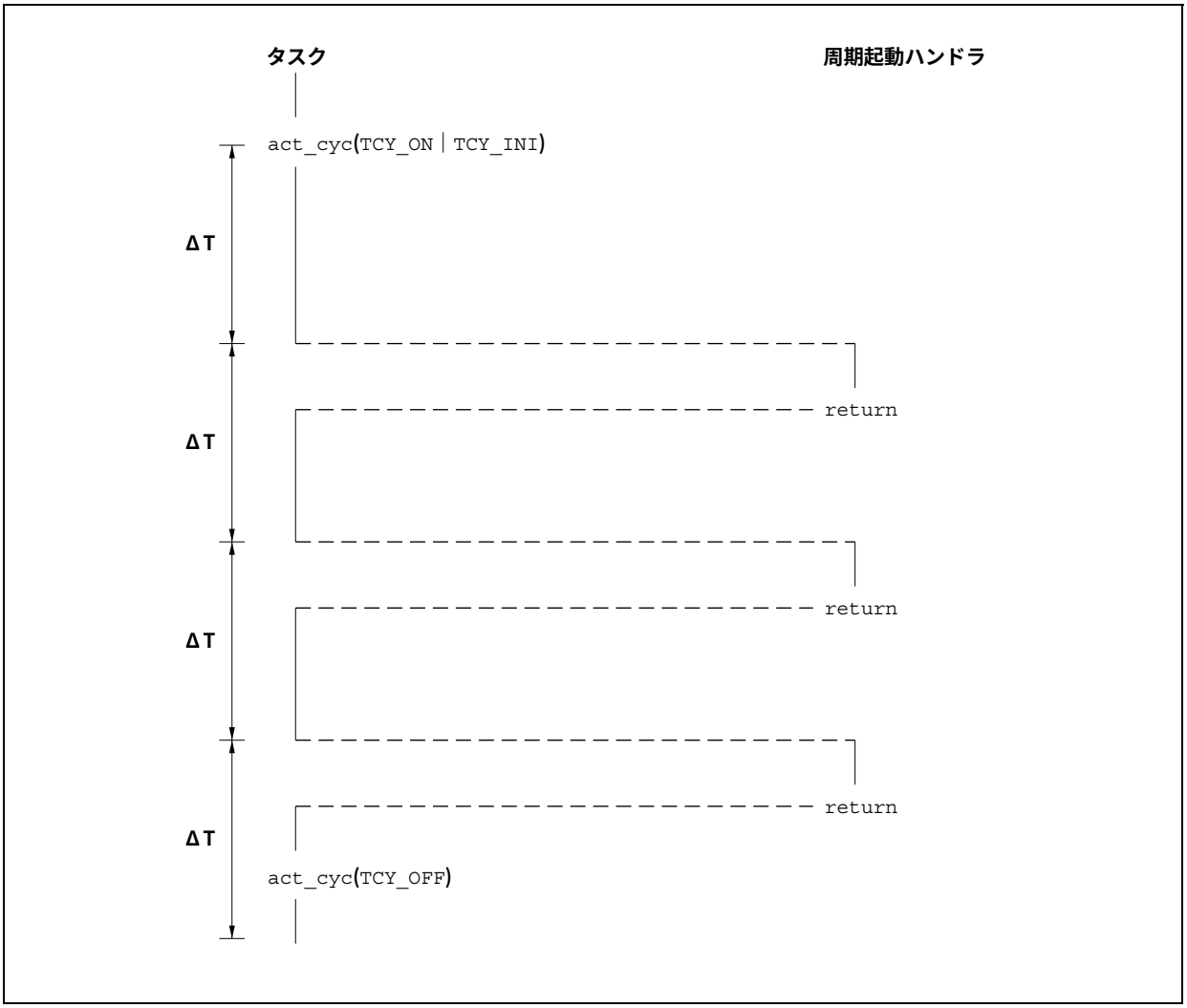


図7ー4　act_cyc（TCY_ON｜TCY_INI）発行時の処理の流れ



7.5.3　周期起動ハンドラ内での処理

RX850では，クロック割り込みが発生してから周期起動ハンドラに制御を移すとき，独自の割り込み前処理を行っています。また，周期起動ハンドラから制御を戻すときにも，独自の割り込み後処理を行っています。

このため，周期起動ハンドラの処理を記述するときには，次に示す注意事項があります。

(1) レジスタの退避／復帰

RX850では，周期起動ハンドラに制御を移すとき，および周期起動ハンドラから復帰するとき，Cコンパイラ（CA850またはCCV850）の関数呼び出し規約に従った，作業用レジスタの退避処理／復帰処理を行っています。したがって，周期起動割り込みハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要がありません。

注意　RX850では，周期起動ハンドラに制御を移すとき，gp, tp, epレジスタの切り替えを行っていません。

(2) スタックの切り替え

RX850では，周期起動ハンドラに制御を移すとき，および周期起動ハンドラから復帰するとき，スタックの切り替え処理を行っています。したがって，周期起動ハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要がありません。

ただし，コンフィギュレーション時に割り込みハンドラ用スタックを定義していない場合は，スタックの切り替え処理が行われないため，割り込み発生時のスタックが使用されることになります。

(3) システム・コールの発行制限

周期起動ハンドラ内で発行可能なシステム・コールの一覧を次に示します。

- タスク管理機能システム・コール

```
sta_tsk     chg_pri     rot_rdq     rel_wai     get_tid
ref_tsk
```

- タスク付属同期機能システム・コール

```
sus_tsk     rsm_tsk     frsm_tsk    wup_tsk     can_wup
```

- 同期通信機能システム・コール

```
sig_sem     preq_sem    ref_sem     set_flg     clr_flg
pol_flg     ref_flg     vset_flg1   vclr_flg1   vpol_flg1
vref_flg1   snd_msg     prcv_msg    ref_mbx
```

- 割り込み管理機能システム・コール

```
dis_int     ena_int     chg_icr     ref_icr
```

- メモリ・プール管理機能システム・コール

```
pget_blf    rel_blf     ref_mpf     pget_blk    rel_blk
ref_mpl
```

- 時間管理機能システム・コール

```
act_cyc     ref_cyc
```

- システム管理機能システム・コール

```
get_ver     ref_sys
```

(4) 周期起動ハンドラからの復帰処理

周期起動ハンドラからの復帰処理は，周期起動ハンドラの終了部分で，return命令を発行することにより行われます。

- return命令

  周期起動ハンドラから復帰します。

なお，RX850では，周期起動ハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_semなど）が発行されたときには，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，周期起動ハンドラからの復帰処理（return命令の発行）まで遅延され，一括して行うようにしています。

7.5.4　周期起動ハンドラ情報の獲得

周期起動ハンドラ情報の獲得は，ref_cycシステム・コールを発行することにより行われます。

(1) ref_cycシステム・コール

パラメータで指定された周期起動ハンドラの周期起動ハンドラ情報（拡張情報，残り時間など）を獲得します。

周期起動ハンドラ情報の詳細を次に示します。

- 拡張情報
- 次に周期起動ハンドラを起動するまでの残り時間
- 現在の活性状態

〔メ　モ〕

第8章　スケジューラ

この章では，RX850が行うスケジューリング処理について示しています。

8.1 概　　要

RX850のスケジューラでは，ダイナミックに変化していくタスクの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，最適なタスクにプロセッサの使用権を与えています。

8.2 駆動方式

RX850のスケジューラは，何らかの事象が発生したときに駆動される事象駆動方式を採用しています。

なお，RX850における「何らかの事象」とは，タスクの状態遷移を引き起こす可能性のあるシステム・コールの発行，ハンドラからの復帰命令の発行，およびクロック割り込みの発生を意味します。

したがって，ユーザの処理プログラム内からタスクの状態遷移を引き起こす可能性のあるシステム・コールが発行されたとき，ハンドラからの復帰命令が発行されたとき，およびクロック割り込みが発生したときには，スケジューラが駆動し，タスクのスケジューリング処理を行います。

スケジューラを駆動するシステム・コールの一覧を次に示します。

- タスク管理機能システム・コール

```
sta_tsk     ext_tsk     ena_dsp     chg_pri     rot_rdq
rel_wai
```

- タスク付属同期機能システム・コール

```
rsm_tsk     frsm_tsk    slp_tsk     tslp_tsk    wup_tsk
```

- 同期通信機能システム・コール

```
sig_sem     wai_sem     twai_sem    set_flg     wai_flg
twai_flg    vset_flg1   vwai_flg1   vtwai_flg1  snd_msg
rcv_msg     trcv_msg
```

- 割り込み管理機能システム・コール

```
ret_int     ret_wup     unl_cpu
```

- メモリ・プール管理機能システム・コール

```
get_blf     tget_blf    rel_blf     get_blk     tget_blk
rel_blk
```

- 時間管理機能システム・コール

```
dly_tsk
```

8.3　スケジューリング方式

RX850のスケジューリング方式は，優先度方式およびFCFS方式を採用しています。

このため，スケジューラは，実行可能な状態（run状態およびready状態）にあるタスクの優先度を参照し，その中から最適なタスクを選び出し，プロセッサの使用権を与えています。

8.3.1　優先度方式

各タスクには，処理を実行するうえでの優先順位を決定する優先度が付けられています。

したがって，スケジューラは，実行可能な状態（run状態およびready状態）にあるタスクの優先度を参照し，その中から最も高い優先度（最高優先度）を持つタスクを選び出し，プロセッサの使用権を与えています。

注意　RX850におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。

8.3.2　FCFS（First Come First Service）方式

RX850では，同一の優先度を複数のタスクに付けることが可能です。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（最高優先度）を持つタスク」が複数存在する場合があります。

そこで，スケジューラは，最高優先度を持つタスクが複数存在する場合には，先に実行可能状態となったタスク（ready状態となってから最も時間が経過しているタスク）を選び出し，プロセッサの使用権を与えています。

8.4　アイドル・ハンドラ

アイドル・ハンドラは，すべてのタスクがrun状態またはready状態でなくなったとき，すなわち，RX850のスケジューリング対象となるタスクがシステム内に1つも存在しなくなったときにスケジューラから起動される処理ルーチンであり，タスクとは独立したものとして扱われます。

なお，アイドル・ハンドラは，V850ファミリが提供するパワー・セーブ機能を有効活用するために用意された処理ルーチンであり，その処理内容については，サンプル・ソース・ファイルを提供しています。

8.4.1　アイドル・ハンドラ内での処理

アイドル・ハンドラでは，V850ファミリのパワー・セーブ機能の設定処理を行います。

V850ファミリが提供するパワー・セーブ機能について次に示します。

(1) HALTモード

クロック・ジェネレータ（発振回路およびPLLシンセサイザ）は動作を継続しますが，プロセッサの動作クロックが停止するモードです。なお，その他の内部周辺機能へのクロック供給は継続され，動作を継続します。

HALTモードは，通常動作モードとの組み合わせによる間欠動作により，システムのトータル消費電力を低減することができます。

専用命令（HALT命令）を発行することにより，HALTモードに移行します。

(2) IDLEモード

クロック・ジェネレータ（発振回路およびPLLシンセサイザ）は動作を継続したままで，内部システム・

クロックの供給を停止させることにより，システム全体を停止させるモードです。

IDLEモードからの解除時，クロック安定時間（発振回路の発振安定時間など）を確保する必要がないため，高速に通常動作モードに移行することができます。

IDLEモードは，消費電力とクロック安定時間に関して，HALTモードとSTOPモードの中間に位置するモードであり，低消費電力，クロック安定時間の削除を実現したい用途に利用することができます。

制御レジスタ（パワー・セーブ・コントロール・レジスタPSC）を設定することにより，IDLEモードに移行します。

(3) STOPモード

クロック・ジェネレータ（発振回路およびPLLシンセサイザ）の動作を停止させることにより，システム全体を停止させるモードです。

STOPモードは，リーク電流のみの超低消費電力状態となります。

制御レジスタ（パワー・セーブ・コントロール・レジスタPSC）を設定することにより，STOPモードに移行します。

8.5　ラウンドロビン方式の実現

RX850では，優先度方式およびFCFS方式のスケジューリングを行っていますが，この方式では，タスクが複数存在した場合，最初にプロセッサの使用権を与えられたタスクが他の状態へ遷移，またはプロセッサの使用権を放棄しないかぎり，他タスクが「同一の優先度でありながらも処理を実行することができない」といった事態が生じてきます。

そこで，RX850では，このような事態を回避するスケジューリング方式（ラウンドロビン方式）を実現するために，rot_rdqなどといったシステム・コールを提供しています。

セマフォを使用してタスク間の排他制御を行った場合のラウンドロビン方式の実現例を次に示します。

（前提条件）

- タスクの優先度

  タスクA = タスクB = タスクC
- タスクの状態

  タスクA：run状態

  タスクB：ready状態

  タスクC：ready状態
- 周期起動ハンドラの属性

  活性状態：ON状態

  起動周期：ΔT（単位：クロック割り込み周期）

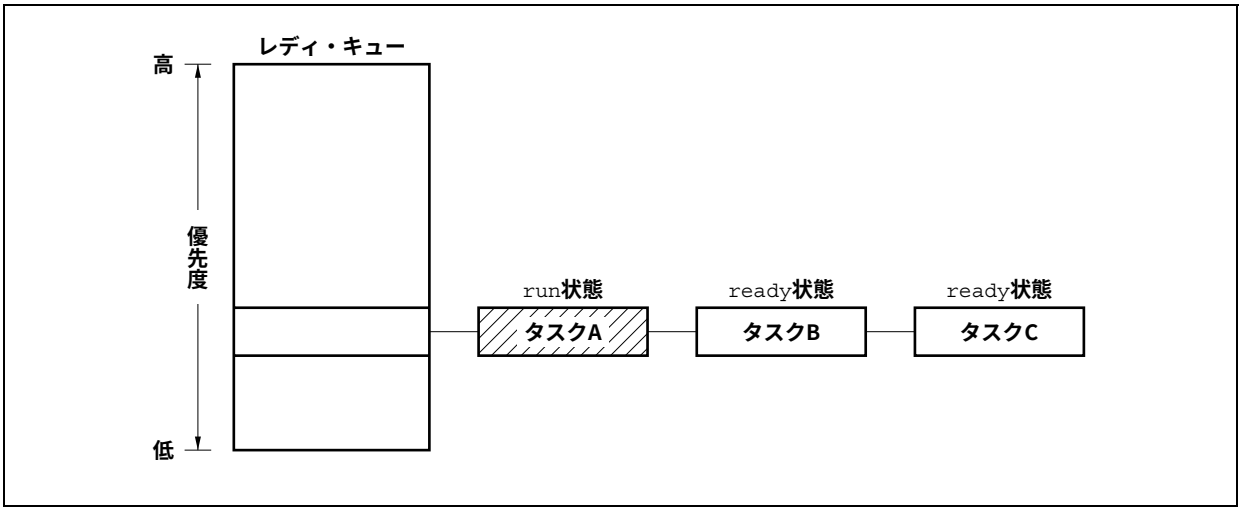
  処理内容：レディ・キューの回転（rot_rdqシステム・コールの発行）

(1) 現在，タスクAが処理を実行しています。

なお，他タスク（タスクB，タスクC）もタスクAと同一の優先度を持ちますが，タスクAが他の状態へ遷移，またはプロセッサの使用権を放棄しないかぎり，処理を実行することはできません。

図8－1に，レディ・キューの状態を示します。

図8－1　レディ・キューの状態

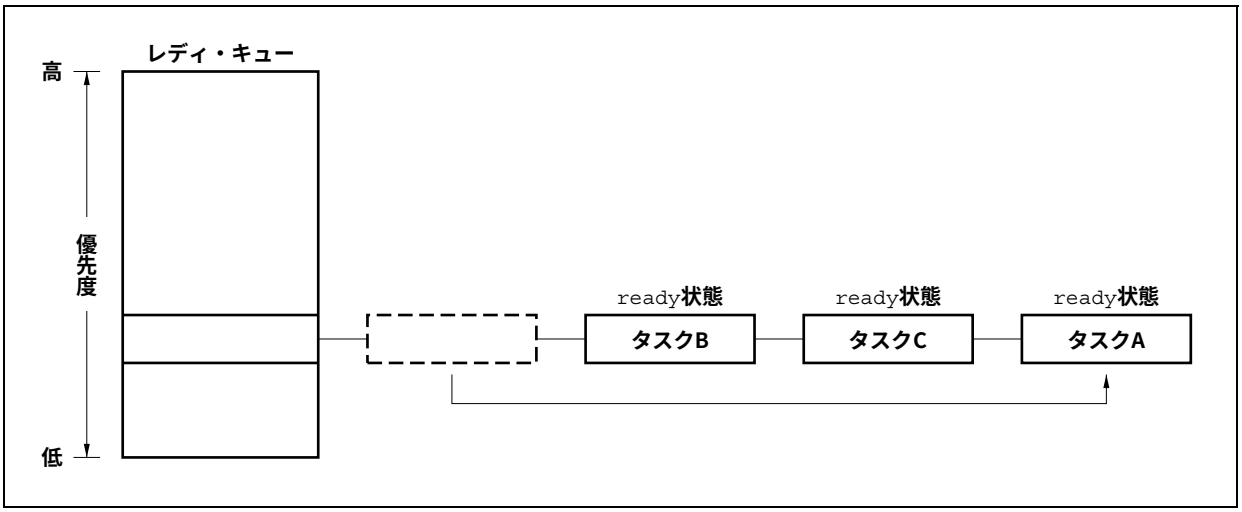


(2) 時間の経過により周期起動ハンドラが起動し，rot_rdqシステム・コールを発行します。

これにより，タスクAは，優先度に応じたレディ・キューの最後尾にキューイングされるとともに，run状態からready状態へと遷移します。

図8－2に，レディ・キューの状態を示します。

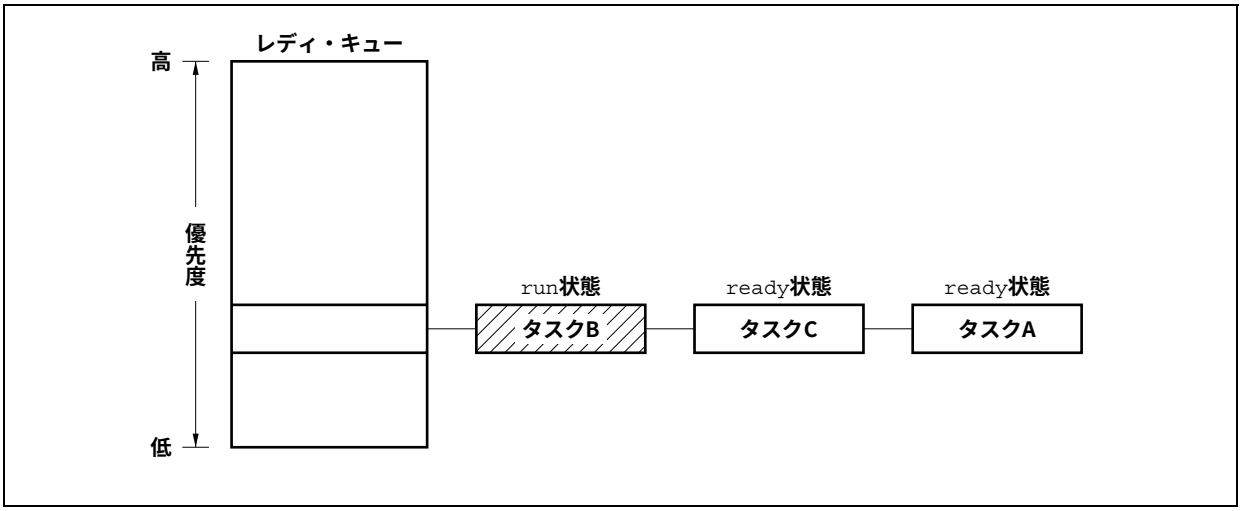
図8－2　レディ・キューの状態



(3) タスクAがready状態へと遷移したことにともない，タスクBがready状態からrun状態へと遷移します。

図8－3に，レディ・キューの状態を示します。

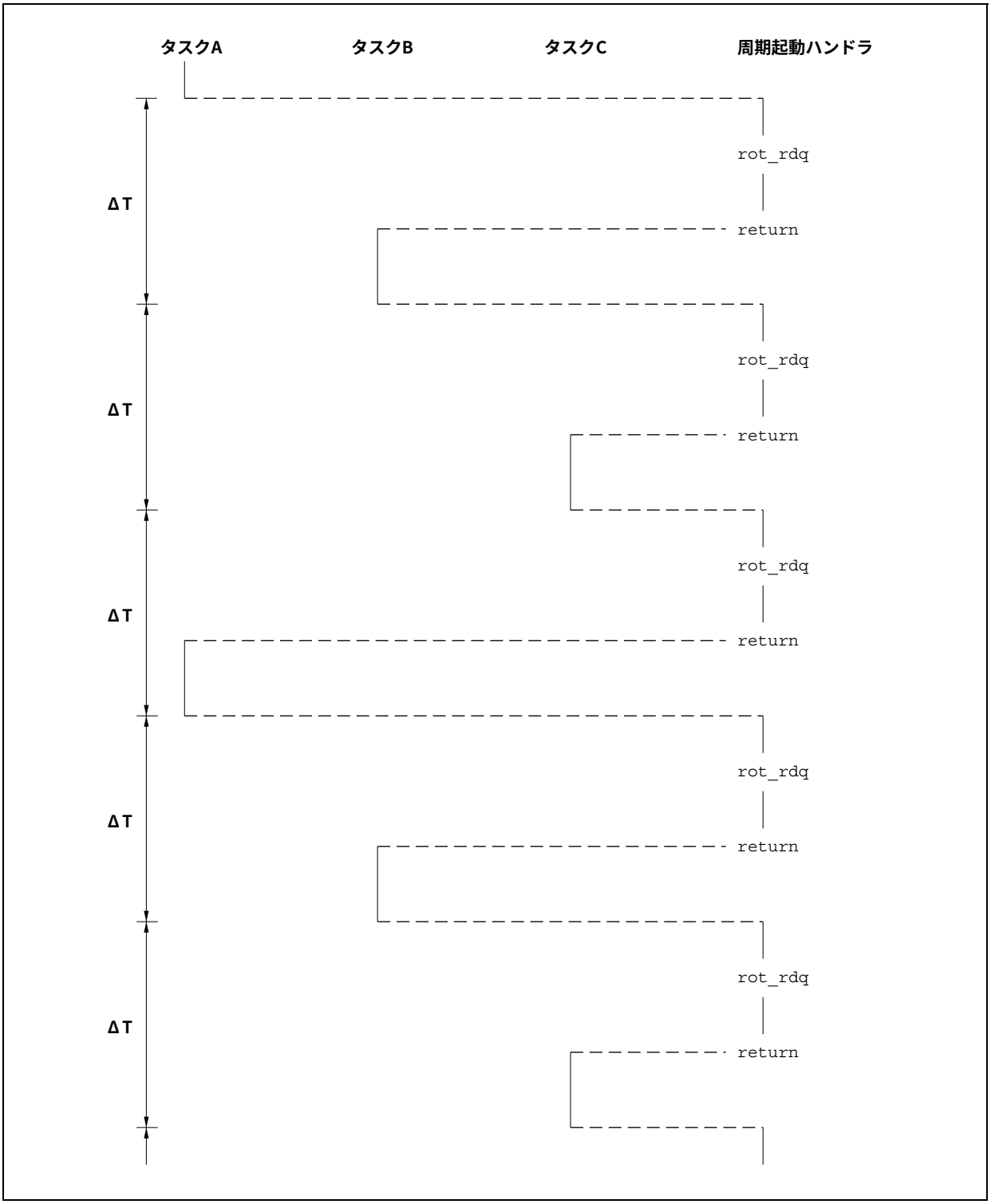
図8－3　レディ・キューの状態



(4) このようにして，一定周期で起動される周期起動ハンドラ内からrot_rdqシステム・コールを発行することにより，一定の時間（ΔT）が経過したときには，必ずタスクが切り替わるスケジューリング方式（ラウンドロビン方式）を実現することができます。

図8－4に，ラウンドロビン方式による処理の流れを示します。

図8－4　ラウンドロビン方式による処理の流れ



8.6　スケジューリングのロック機能

RX850では，ユーザの処理プログラムからスケジューラの駆動を操作し，ディスパッチ処理（タスクのスケジューリング処理）を禁止／再開する機能が提供されています。

なお，この機能は，次に示したシステム・コールをタスク内から発行することにより実現されます。

(1) dis_dspシステム・コール

ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールの発行からena_dspシステム・コールが発行されるまでの間，他タスクに制御が移ることはありません。

(2) ena_dspシステム・コール

ディスパッチ処理（タスクのスケジューリング処理）を再開します。

これにより，dis_dspシステム・コールの発行により禁止されていたディスパッチ処理が再開されます。

(3) loc_cpuシステム・コール

マスカブル割り込みの受け付けを禁止したのち，ディスパッチ処理（タスクのスケジューリング処理）も禁止します。

これにより，このシステム・コールの発行からunl_cpuシステム・コールが発行されるまでの間，他タスク，ハンドラに制御が移ることはありません。

(4) unl_cpuシステム・コール

マスカブル割り込みの受け付けを許可したのち，ディスパッチ処理（タスクのスケジューリング処理）も再開します。

これにより，loc_cpuシステム・コール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が許可されるとともに，ディスパッチ処理も再開されます。

図8－5から図8－7に，通常の場合，dis_dspシステム・コールを発行した場合，loc_cpuシステム・コールを発行した場合の制御の流れを示します。

図8－5　通常の場合

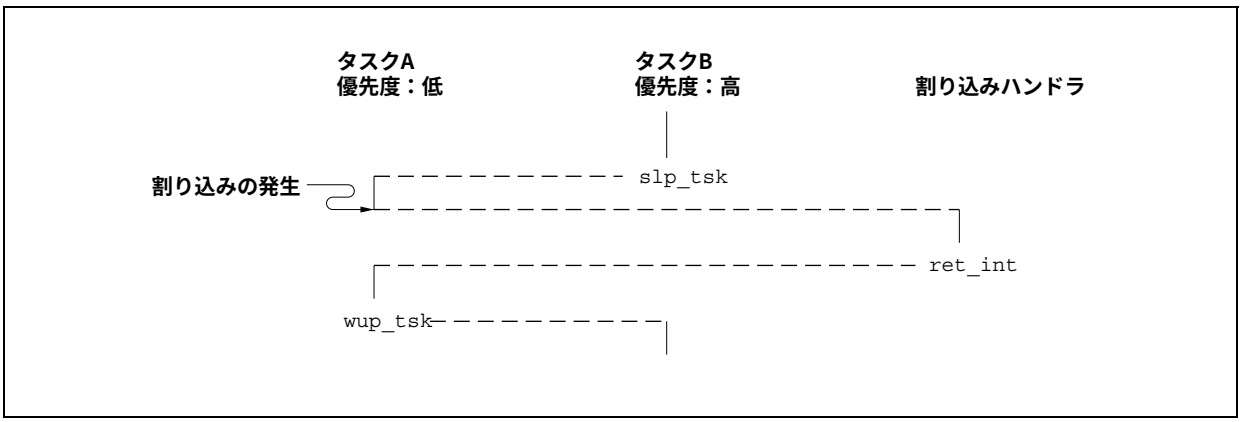


図8－6　dis_dsp システム・コールを発行した場合

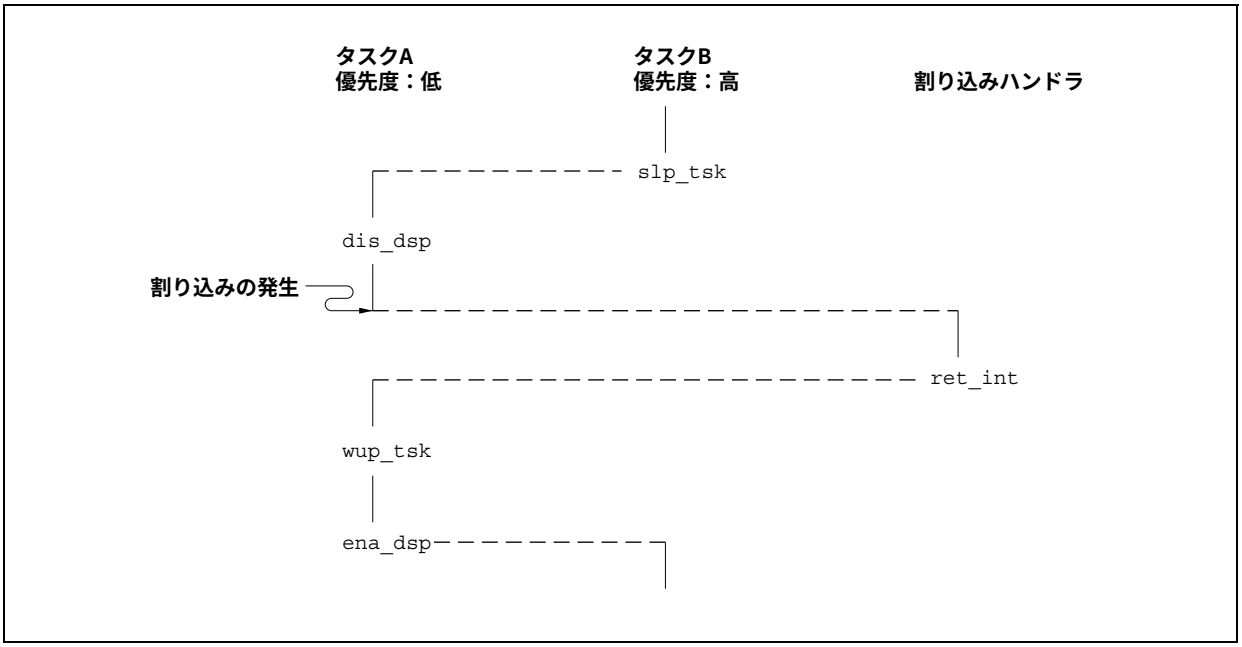
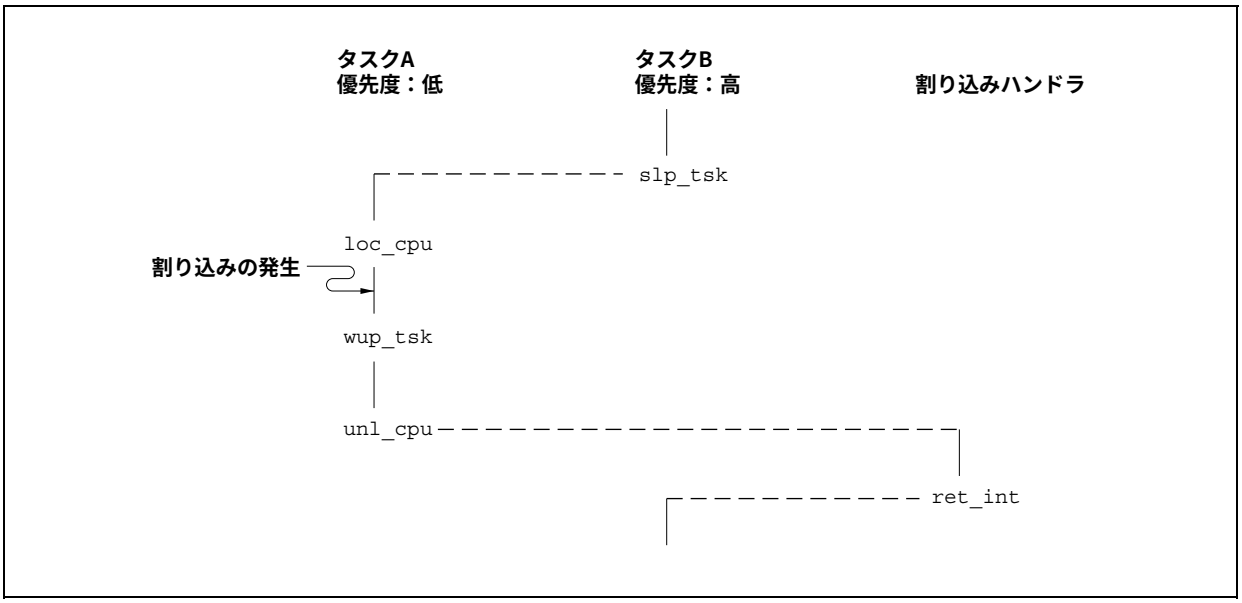


図8－7　loc_cpu システム・コールを発行した場合



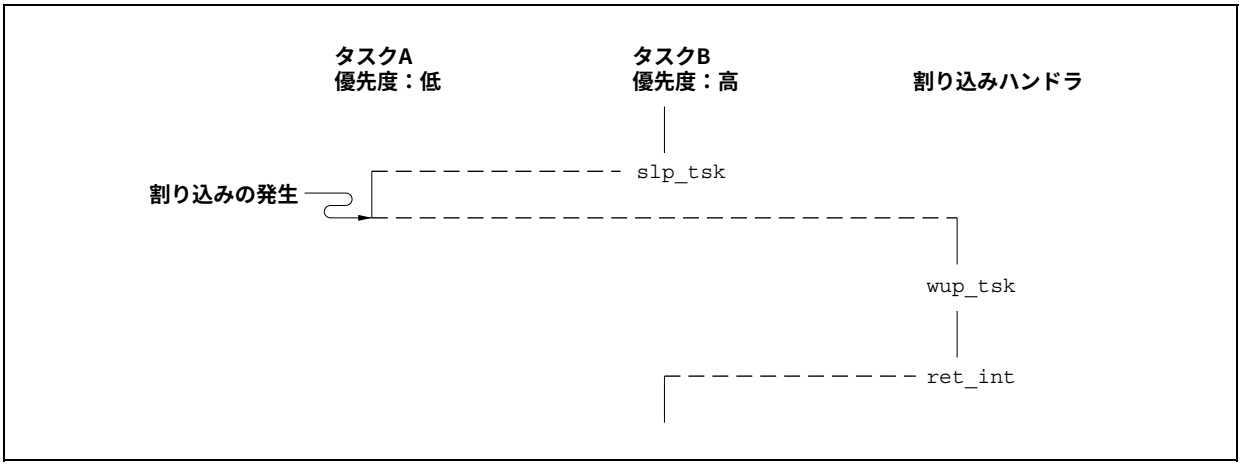
8.7　ハンドラ内でのスケジューリング

RX850では，各種ハンドラ（割り込みハンドラ，周期起動ハンドラ）を高速に終了させる目的で，ハンドラ内の処理が終了するまでの間，スケジューラの駆動を遅延しています。

したがって，ハンドラ内でタスクのスケジューリング処理が必要なシステム・コール（chg_pri, sig_semなど）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，ハンドラからの復帰処理（ret_int, returnなど）まで遅延され，一括して行うようにしています。

図8－8に，ハンドラ内でスケジューリング処理が必要なシステム・コールが発行された場合の制御の流れを示します。

図8－8　wup_tskシステム・コールを発行した場合



〔メ　モ〕

第9章　システム初期化処理

この章では，RX850が行うシステム初期化処理について示しています。

なお，システム初期化処理についての詳細は，RX850　**ユーザーズ・マニュアル　インストレーション編**を参照してください。

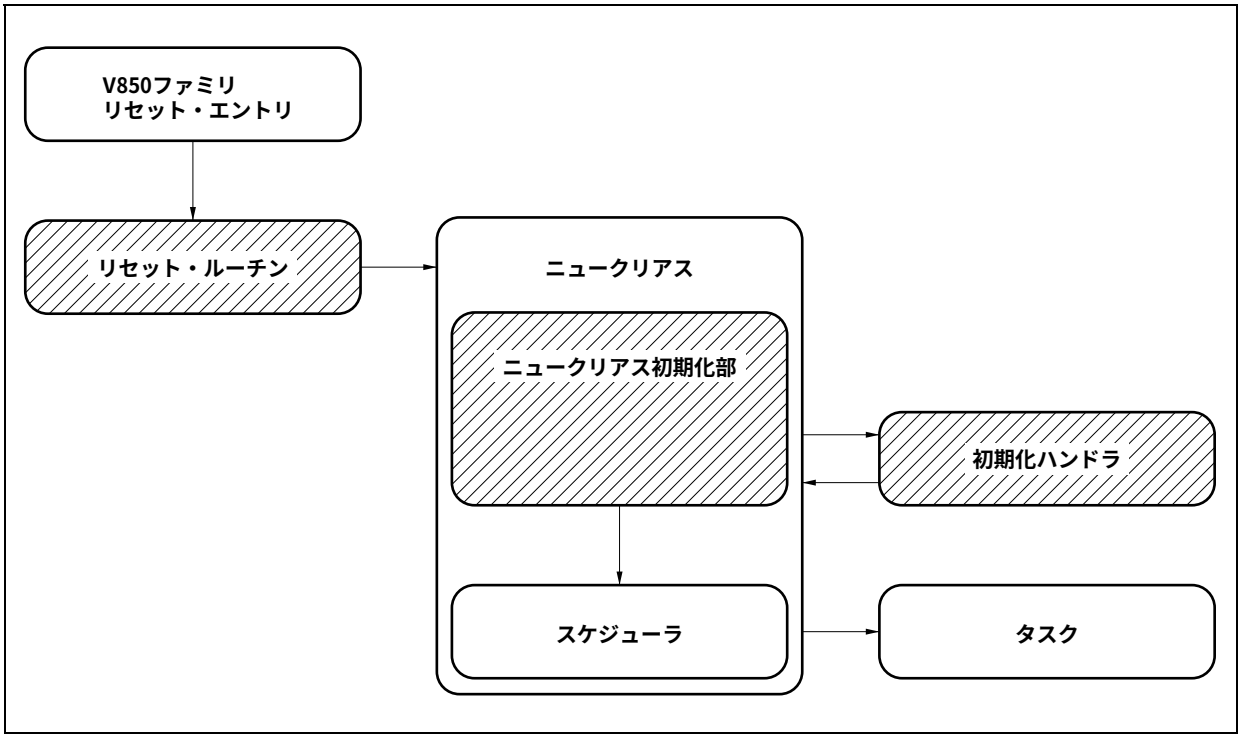
9.1 概　　要

システム初期化処理は，RX850が動作するうえで必要となるハードウエアの初期化処理（リセット・ルーチン），およびソフトウエアの初期化処理（ニュークリアス初期化部，初期化ハンドラ）から構成されています。

したがって，RX850では，システムが起動したとき，最初に行われる処理がシステム初期化処理となります。

図9－1に，システム初期化処理の流れを示します。

図9－1　システム初期化処理の流れ



注意　システム初期化処理は，マスカブル割り込みの受け付けを禁止した状態で処理を実行します。なお，システム初期化処理内でマスカブル割り込みの受け付けを許可した場合，動作の保証がありません。

9. 2　リセット・ルーチン

リセット・ルーチンではシステム初期化処理において最初に処理が実行されます。

リセット・ルーチンではRX850で使用するメモリ以外の初期化処理，たとえばターゲット・システムの初期化やプログラム実行に必要なポインタ（tp, gp, ep）の設定を行います。

リセット・ルーチンでは次に示すような処理を行っています。

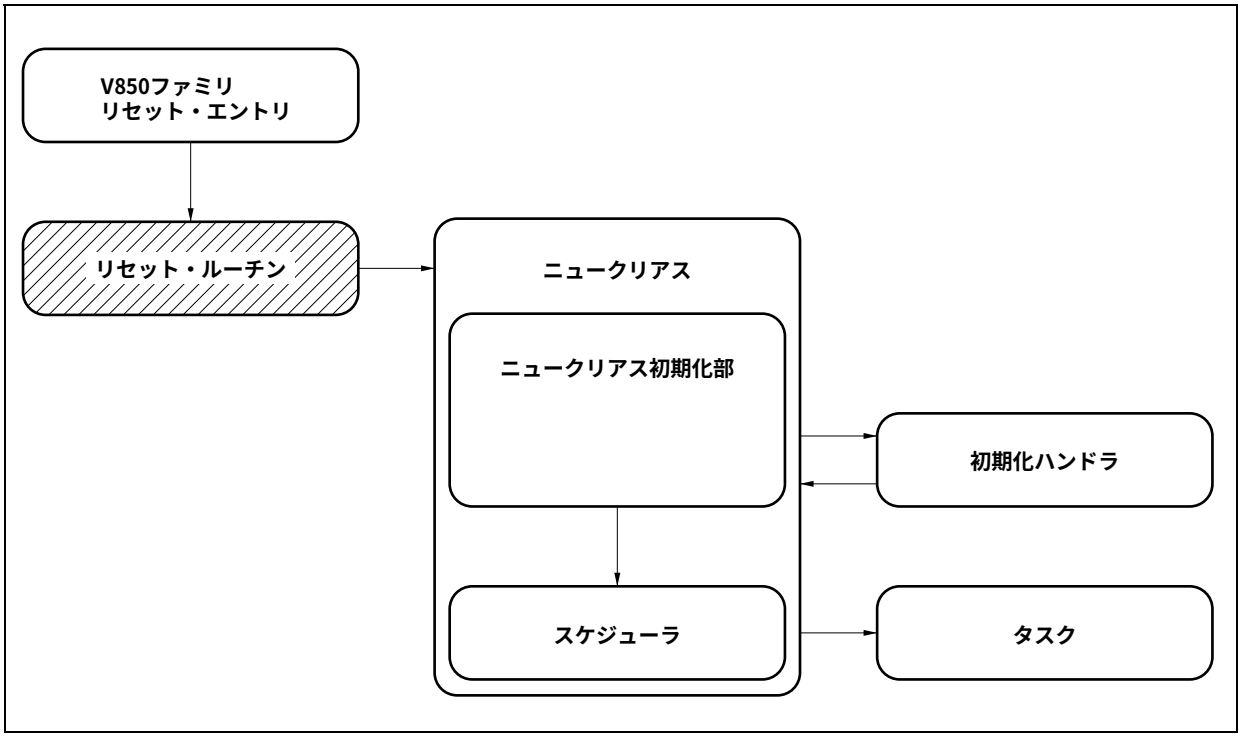
- tp, gp, epレジスタの設定
- バス制御関連の初期化
- 内部ユニット／周辺コントローラの初期化
- 初期値なしデータ領域の初期化
- 初期化データのコピー
- ニュークリアスへの分岐命令

リセット・ルーチンはアセンブリ言語で記述しなければならないため，最低限必要な初期化処理のみを行い，残りはC言語での記述が可能な初期化ハンドラで処理するようにします。

これらの処理内容はユーザのニーズに合わせて書き換えます。

図9－2に，リセット・ルーチンの位置付けを示します。

図9－2　リセット・ルーチンの位置付け



9. 3　ニュークリアス初期化部

ニュークリアス初期化部は，リセット・ルーチンから呼び出される関数であり，システム情報テーブルに記述された情報（システム情報，タスク情報など）をもとに，各種管理オブジェクトを生成／初期化するために用意された関数です。

なお，ニュークリアス初期化部では，次のような処理を行います。

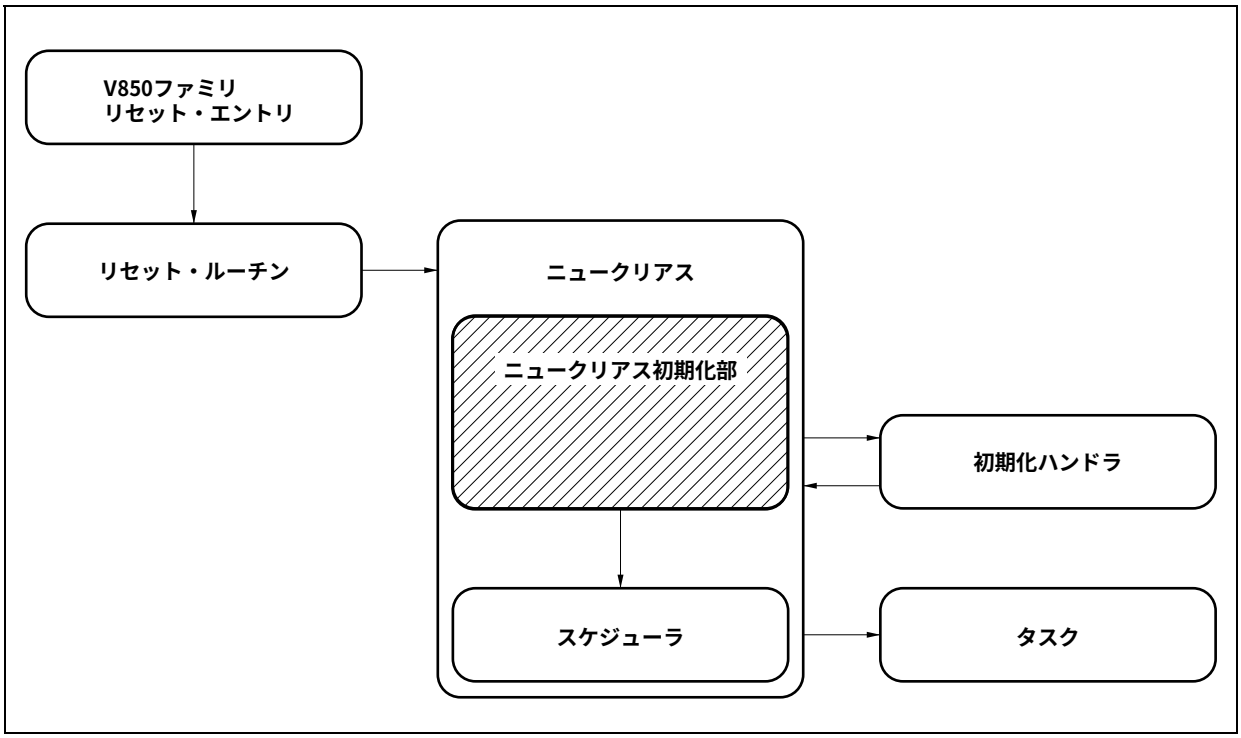
- 管理オブジェクトの生成／初期化
- タスクの起動[注]
- 間接起動割り込みハンドラの登録
- 周期起動割り込みハンドラの登録
- 時間管理用割り込みハンドラ（クロック・ハンドラ）の登録
- 初期化ハンドラの呼び出し
- スケジューラに制御を移す

注　ニュークリアス初期化部で起動されるタスクは，CF定義ファイルでタスクを生成するときに初期状態を`ready`に指定したものです。詳細はRX850　**ユーザーズ・マニュアル　インストレーション編**を参照してください。

注意　ニュークリアス初期化部は，RX850が提供する機能の一部です。このため，ユーザがニュークリアス初期化部を記述する必要はありません。

図9－3に，ニュークリアス初期化部の位置付けを示します。

図9－3　ニュークリアス初期化部の位置付け



9.4　初期化ハンドラ

初期化ハンドラはニュークリアスから呼び出される関数です。リセット・ルーチンからニュークリアスへの分岐命令が発行された後，ニュークリアスの初期化が始まります。ニュークリアスの初期化が終わると初期化ハンドラが呼ばれます。初期化ハンドラの処理が終わると，RX850のスケジューラが起動されOSの動作を開始します。

リセット・ルーチンで行わなかった初期化処理をこの初期化ハンドラで行うようにしてください（9.2　**リセット・ルーチン**参照）。処理例を次に示します。

- 割り込みコントロール・ユニットの初期化
- タイマ・カウンタ・ユニット初期化
- システム・クロック初期化
- ニュークリアスへの分岐命令

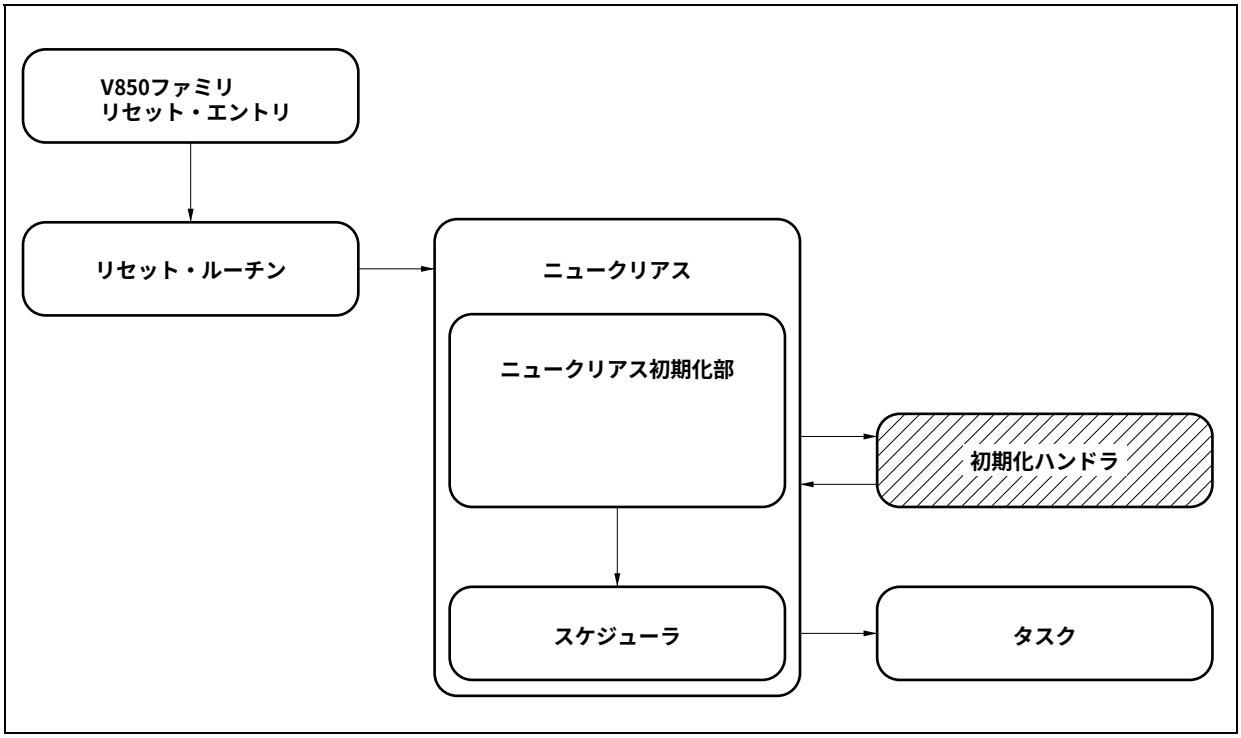
このほかにも初期値なしデータ領域の初期化，初期値ありデータのコピーなどをこの初期化ハンドラで行うことも可能です。

これらの処理内容はユーザのニーズに合わせて書き換えます。

注意　初期化ハンドラ内では割り込みハンドラ，周期起動ハンドラで発行可能なシステム・コールを発行することが可能です。

図9－4に，初期化ハンドラの位置付けを示します。

図9－4　初期化ハンドラの位置付け



第10章　システム・コール

この章では，RX850が提供するシステム・コールについて示しています。

10.1 概　　要

ユーザは，システム・コールを利用することにより，RX850が直接管理している資源（カウンタ，待ちキューなど）を間接的に操作することが可能となります。

なお，RX850では，μ ITRON3.0仕様に準拠した62種類のシステム・コールを提供していますが，これらシステム・コールは，その機能により次に示す7グループに分類することができます。

(1) タスク管理機能システム・コール（10種類）

タスクの状態操作を行うシステム・コールのグループです。

なお，このグループには，タスクを起動／終了する機能，ディスパッチ処理を禁止／再開する機能，タスクの優先度を変更する機能，タスクのレディ・キューを回転する機能，タスクのwait状態を強制的に解除する機能，タスクの状態を参照する機能などが含まれます。

sta_tsk	ext_tsk	ter_tsk	dis_dsp	ena_dsp
chg_pri	rot_rdq	rel_wai	get_tid	ref_tsk

(2) タスク付属同期機能システム・コール（7種類）

タスクに従属した同期操作を行うシステム・コールのグループです。

なお，このグループには，タスクをsuspend状態に移行する／suspend状態のタスクを再開する機能，タスクを起床待ち状態に移行する／起床待ち状態のタスクを起床する機能，タスクの起床要求を無効化する機能などが含まれます。

sus_tsk	rsm_tsk	frsm_tsk	slp_tsk	tslp_tsk
wup_tsk	can_wup			

(3) 同期通信機能システム・コール（22種類）

タスク間の同期（排他制御，待ち合わせ），および，通信を行うシステム・コールのグループです。

なお，このグループには，セマフォを操作する機能，イベント・フラグ，または，1ビット・イベント・フラグを操作する機能，メールボックスを操作する機能などが含まれます。

sig_sem	wai_sem	preq_sem	twai_sem	ref_sem
set_flg	clr_flg	wai_flg	pol_flg	twai_flg
ref_flg	vset_flg1	vclr_flgl	vwai_flgl	vpol_flgl
vtwai_flg1	vref_flg1	snd_msg	rcv_msg	prcv_msg
trcv_msg	ref_mbx			

(4) 割り込み管理機能システム・コール（8種類）

割り込みに依存した処理を行うシステム・コールのグループです。

なお，このグループには，直接起動割り込みハンドラから復帰する機能，マスカブル割り込みの受け付

けを禁止／再開する機能，割り込み制御レジスタの内容を変更／参照する機能などが含まれます。

```
ret_int      ret_wup      loc_cpu      unl_cpu      dis_int
ena_int      chg_icr      ref_icr
```

(5) メモリ・プール管理機能システム・コール（10種類）

メモリ・ブロックの割り当てを行うシステム・コールのグループです。

なお，このグループには，メモリ・ブロックを獲得／返却する機能，メモリ・プールの状態を参照する機能などが含まれます。

```
get_blf      pget_blf     tget_blf     rel_blf      ref_mpf
get_blk      pget_blk     tget_blk     rel_blk      ref_mpl
```

(6) 時間管理機能システム・コール（3種類）

時間に依存した処理を行うシステム・コールのグループです。

なお，このグループには，タスクを時間経過待ち状態に移行する機能，周期起動ハンドラの活性状態を制御する機能，周期起動ハンドラの状態を参照する機能などが含まれます。

```
dly_tsk      act_cyc      ref_cyc
```

(7) システム管理機能システム・コール（2種類）

システムに依存した処理を行うシステム・コールのグループです。

なお，このグループには，バージョン情報を獲得する機能，システムの状態を参照する機能などが含まれます。

```
get_ver      ref_sys
```

10.2　システム・コールの呼び出し

C言語で記述されたタスクまたはハンドラ（割り込みハンドラ，周期起動ハンドラ）からシステム・コールを発行する場合，C言語の関数として呼び出しを行い，そのパラメータは引き数として渡されます。

また，アセンブリ言語で記述されたタスクまたはハンドラからシステム・コールを発行する場合は，使用するCコンパイラ（CA850またはCCV850）の関数呼び出し規約に従ってパラメータおよび戻り番地の設定を行ったのち，jarl命令により呼び出しを行います。

注意　RX850では，システム・コールのプロト・タイプ宣言をstdrx850.hファイルで行っています。このため，タスクまたはハンドラからシステム・コールを発行するときには，このヘッダ・ファイルのインクルード処理（#include〈stdrx850.h〉）を記述する必要があります。

10.3　パラメータのデータ・タイプ

RX850が提供するシステム・コールのパラメータは，μ ITRON3.0仕様に準拠したデータ・タイプを基本に定義されています。

表10－1に，システム・コールを発行するときに指定する各種パラメータのデータ・タイプ一覧を示します。

なお，データ・タイプのマクロ定義は，ヘッダ・ファイルtype.hで行われています。

表10－1　データ・タイプの一覧

マクロ	データ・タイプ	意　味
B	char	符号付き8ビット整数
H	short	符号付き16ビット整数
INT	int	符号付き32ビット整数
W	long	符号付き32ビット整数
UB	unsigned char	符号無し8ビット整数
UH	unsigned short	符号無し16ビット整数
UINT	unsigned int	符号無し32ビット整数
UW	unsigned long	符号無し32ビット整数
VB	char	データ・タイプが一定しない値（8ビット）
VH	short	データ・タイプが一定しない値（16ビット）
VW	long	データ・タイプが一定しない値（32ビット）
＊VP	void	データ・タイプが一定しない値（ポインタ）
（＊FP）()	void	処理プログラムの起動アドレス
BOOL	char	ブール値
ID	char	オブジェクトのID番号
BOOL_ID	char	待ちタスクの有無
HNO	char	周期起動ハンドラの指定番号
ER	long	エラー・コード
PRI	char	タスク，または，メッセージの優先度
TMO	long	待ち時間
CYCTIME	long	周期起動時間間隔（残り時間）
DLYTIME	long	遅延時間

10.4　システム・コールからの戻り値

RX850が提供するシステム・コールの戻り値は，μ ITRON3.0仕様に準拠した戻り値を基本に定義されています。

表10－2に，システム・コールからの戻り値一覧を示します。

なお，戻り値のマクロ定義は，ヘッダ・ファイルerrno.hで行われています。

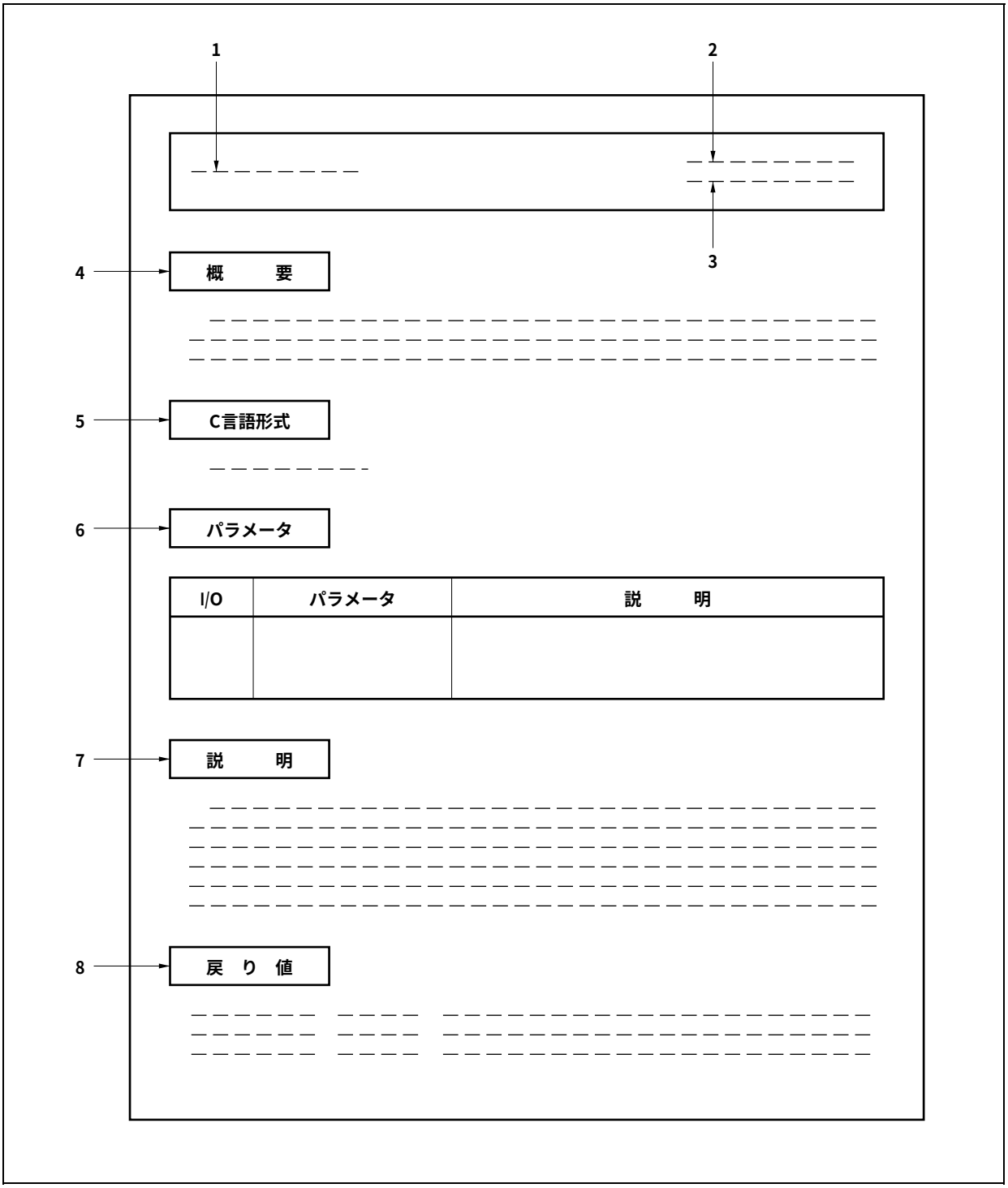
表10－2　戻り値の一覧

マクロ	数　値	意　味
E_OK	0	正常終了
E_OBJ	－63	指定したオブジェクトの状態が不適当である
E_CTX	－69	コンテキスト・エラー
E_QOVR	－73	カウントがオーバフローした
E_TMOUT	－85	タイムアウト／ポーリング失敗
E_RLWAI	－86	wait状態を強制的に解除された

10.5　システム・コール解説

10. 5. 1　タスク管理機能システム・コールから10. 5. 7　システム管理機能システム・コールに，RX850が提供するシステム・コールについて，次の記述フォーマットに従って解説します。

図10－1　システム・コールの記述フォーマット



1　**名称**

システム・コールの名称を示しています。

2　**名称の由来**

システム・コールの名称の由来を示しています。

3　**発行可能な状態**

システム・コールの発行が可能な状態を示しています。

タスク／ハンドラ　　：タスク，ハンドラ（直接起動割り込みハンドラ，間接起動割り込みハンドラ，周期起動ハンドラ）のどちらからも発行可能

タスク　　：タスクからのみ発行可能

直接起動割り込みハンドラ：直接起動割り込みハンドラからのみ発行可能

周期起動ハンドラ　　：周期起動ハンドラからのみ発行可能

4　**概　　要**

システム・コールの機能概要を示しています。

5　C**言語形式**

システム・コールをC**言語で発行するときの記述形式を示しています。**

6　**パラメータ**

システム・コールのパラメータを，次の形式で示しています。

なお，各種マクロ定義は，ヘッダ・ファイルusr.h, option.h**で行われています。**

I/O	パラメータ	説　　明
A	B	C

A：**パラメータの種類**

I … RX850**への入力パラメータ**

O … RX850**からの出力パラメータ**

B：**パラメータのデータ・タイプ**

C：**パラメータの説明**

7　**説　　明**

システム・コールの機能について示しています。

8　**戻　り　値**

システム・コールからの戻り値をマクロと数値で示しています。

なお，戻り値のマクロ定義は，ヘッダ・ファイルerrno.h**で行われています。**

10. 5. 1　タスク管理機能システム・コール

この項では，タスクの状態操作を行うシステム・コールのグループ（タスク管理機能システム・コール）について説明しています。

表10－3に，タスク管理機能システム・コールの一覧を示します。

表10－3　タスク管理機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
sta_tsk	他タスクを起動する	○
ext_tsk	自タスクを終了する	○
ter_tsk	他タスクを強制的に終了する	×
dis_dsp	ディスパッチ処理を禁止する	×
ena_dsp	ディスパッチ処理を再開する	○
chg_pri	タスクの優先度を変更する	○
rot_rdq	タスクのレディ・キューを回転する	○
rel_wai	他タスクのwait状態を強制的に解除する	○
get_tid	タスクのID番号を獲得する	×
ref_tsk	タスク情報を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Start Task

sta_tsk

タスク／ハンドラ

概　　要

他タスクを起動する。

C言語形式

```
ER sta_tsk（ID tskid, INT stacd）；
```

パラメータ

I/O	パラメータ	説　明
I	ID　*tskid*；	タスクのID番号
I	INT　*stacd*；	起動コード

説　　明

*tskid*で指定されたタスクにタスク実行権を与えたのち，dormant状態からready状態へと遷移させます。

ただし，このシステム・コールを発行したとき，対象タスク実行権グループからタスク実行権を獲得することができなかった場合には，対象タスクを対象タスク実行権グループの待ちキューにキューイングしたのち，dormant状態からwait状態（タスク実行権待ち状態）へと遷移させます。

なお，*stacd*には，対象タスクに引き渡す起動コードを指定します。対象タスクは，起動コードを関数パラメータと同様に取り扱うことで操作可能となります。

注意1. **このシステム・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，戻り値としてE_OBJを返しています。**

2. **タスク実行権待ち状態の解除は，rel_waiシステム・コールでは行われません。**

3. **タスクを対象タスク実行権グループの待ちキューにキューイングするときのキューイング方法は，タスクの優先度順に行われます。**

戻　り　値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクがdormant状態でない**

Exit Task

ext_tsk

タスク

概　　要

自タスクを終了する。

C言語形式

```
void ext_tsk () ;
```

パラメータ

なし

説　　明

自タスクのタスク実行権を返却したのち，自タスクをrun状態からdormant状態へと遷移させます。

なお，このシステム・コールを発行したとき，対象タスク実行権グループ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，タスク実行権の返却処理は行わず，該当するタスク（待ちキューの先頭タスク）にタスク実行権を渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（タスク実行権待ち状態）からready状態へと遷移します。

注意1. **このシステム・コールをハンドラまたはディスパッチ禁止状態から発行した場合，動作の保証がありません。**

2. **このシステム・コールでは，自タスクが終了する以前に獲得していた資源（セマフォ・カウント，メモリ・ブロックなど）の解放処理が行われません。したがって，このシステム・コールを発行する前に資源の解放処理を行うことは，ユーザの責任となります。**

3. **タスクをアセンブリ言語で記述した場合，自タスクの終了は次のように記述します。**

```
jr      _ext_tsk
```

戻 り 値

なし

Terminate Task

ter_tsk

タスク

概　　要

他タスクを強制的に終了する。

C言語形式

```
ER ter_tsk（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*tskid*；	タスクのID番号

説　　明

*tskid*で指定されたタスクのタスク実行権を返却したのち，対象タスクを強制的にdormant状態へと遷移させます。

なお，このシステム・コールを発行したとき，対象タスク実行権グループ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，タスク実行権の返却処理は行わず，該当するタスク（待ちキューの先頭タスク）にタスク実行権を渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（タスク実行権待ち状態）からready状態へと遷移します。

注意　このシステム・コールでは，対象タスクが終了する以前に獲得していた資源（セマフォ・カウント，メモリ・ブロックなど）の解放処理が行われません。したがって，このシステム・コールを発行する前に資源の解放処理を行うことは，ユーザの責任となります。

戻　り　値

E_OK	0	正常終了
E_OBJ	－63	対象タスクが自タスクまたはdormant状態である

Disable Dispatch

dis_dsp

タスク

概　　要

ディスパッチ処理を禁止する。

C言語形式

```
ER dis_dsp () ;
```

パラメータ

なし

説　　明

ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールの発行からena_dspシステム・コールが発行されるまでの間，ディスパッチ処理が禁止されます。

なお，RX850では，このシステム・コールの発行からena_dspシステム・コールが発行されるまでの間に，タスクのスケジューリング処理が必要なシステム・コール（chg_pri, sig_semなど）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，ena_dspシステム・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。

注意1. **このシステム・コールでは，禁止要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，ディスパッチ処理が禁止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。**

2. **このシステム・コールの発行からena_dspシステム・コールの発行までの間，自タスクの状態遷移を引き起こす可能性のあるシステム・コール（ext_tsk, wai_semなど）を発行した場合，動作の保証がありません。**

戻 り 値

E_OK	0	正常終了
E_ CTX	－69	loc_cpuシステム・コールを発行後，このシステム・コールを発行した

Enable Dispatch

ena_dsp

タスク

概　　要

ディスパッチ処理を再開する。

C言語形式

```
ER ena_dsp () ;
```

パラメータ

なし

説　　明

dis_dspシステム・コール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）を再開します。

なお，RX850では，dis_dspシステム・コールの発行からこのシステム・コールが発行されるまでの間に，タスクのスケジューリング処理が必要なシステム・コール（chg_pri, sig_semなど）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，このシステム・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。

注意　このシステム・コールでは，再開要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，ディスパッチ処理が再開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。

戻 り 値

E_OK	0	正常終了
E_CTX	－69	loc_cpuシステム・コールを発行後，このシステム・コールを発行した

chg_pri

Change Task Priority

タスク／ハンドラ

概　　要

タスクの優先度を変更する。

C言語形式

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

パラメータ

I/O	パラメータ	説　　明
I	ID　*tskid*；	タスクのID番号
I	PRI　*tskpri*；	タスクの優先度

説　　明

*tskid*で指定されたタスクの優先度を*tskpri*で指定された値に変更します。

ただし，対象タスクがrun状態またはready状態であった場合には，優先度の変更処理とともに，対象タスクを優先度に応じたレディ・キューの最後尾につなぎかえます。

注意1. **このシステム・コールが再発行されるまで，または対象タスクが**dormant**状態へと遷移するまで，*tskpri* で指定された値が有効となります。**

2. RX850**におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。**

戻　り　値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクが**dormant**状態である**

rot_rdq

Rotate Ready Queue

タスク／ハンドラ

概　　要

タスクのレディ・キューを回転する。

C言語形式

```
ER rot_rdq（PRI tskpri）；
```

パラメータ

I/O	パラメータ	説　　明
I	PRI　*tskpri*；	**タスクの優先度**

説　　明

***tskpri*で指定された優先度に応じたレディ・キューの先頭タスクを最後尾につなぎかえます。**

注意1. **このシステム・コールでは，対象優先度のレディ・キューにタスクが１つもキューイングされていない場合には，何も処理は行わず，エラーとしても扱いません。**

2. **このシステム・コールを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可能となります。**

戻　り　値

E_OK　　0　　**正常終了**

Release Wait

rel_wai

タスク／ハンドラ

概　　要

他タスクのwait状態を強制的に解除する。

C言語形式

```
ER rel_wai（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　　*tskid*；	タスクのID番号

説　　明

***tskid*で指定されたタスクのwait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，wait状態からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

なお，このシステム・コールの発行によりwait状態を解除されたタスクには，wait状態へと遷移するきっかけとなったシステム・コール（slp_tsk, wai_semなど）の戻り値としてE_RLWAIが返されます。

注意1. **このシステム・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wait状態またはwait_suspend状態以外の場合には，戻り値としてE_OBJを返しています。**

2. **このシステム・コールでは，タスク実行権待ち状態の解除が行われません。このため，対象タスクがタスク実行権待ち状態の場合には，戻り値としてE_OBJを返しています。**

戻　り　値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクがwait状態またはwait_suspend状態でない**

Get Task Identifier

get_tid

タスク／ハンドラ

概　　要

タスクのID番号を獲得する。

C言語形式

```
ER get_tid（ID *p_tskid）；
```

パラメータ

I/O	パラメータ	説　　明
O	ID　*p_tskid；	ID番号を格納する領域のアドレス

説　　明

自タスクのID番号を*p_tskid*で指定される領域に格納します。

注意　このシステム・コールをハンドラ内から発行した場合，*p_tskid*で指定される領域にはFALSE（0）が格納されます。

戻 り 値

E_OK	0	**正常終了**

ref_tsk

Refer Task Status

タスク／ハンドラ

概　　要

タスク情報を獲得する

C言語形式

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

パラメータ

I/O	パラメータ	説　　明
O	T_RTSK　＊*pk_rtsk*；	タスク情報を格納するパケットの先頭アドレス
I	ID　　*tskid*；	タスクのID番号

タスク情報T_RTSKの構造

```
typedef struct t_rtsk {
        VP      exinf；     /＊ 拡張情報              ＊/
        PRI     tskpri；    /＊ 現在の優先度          ＊/
        UINT    tsksts；    /＊ タスクの状態          ＊/
        UINT    tskwait；   /＊ 待ち要因              ＊/
        ID      wid；       /＊ 対象オブジェクトのID番号 ＊/
} T_RTSK；
```

説　　明

***tskid*で指定されたタスクのタスク情報（拡張情報，現在の優先度など）を*pk_rtsk*で指定されるパケットに格納します。**

タスク情報の詳細を次に示します。

exinf … **拡張情報**

tskpri … **現在の優先度**

tsksts … **タスクの状態**

TTS_RUN（H' 01）：run**状態**

TTS_RDY（H' 02）：ready**状態**

TTS_WAI（H' 04）：wait**状態**

- **起床待ち状態**
- **時間経過待ち状態**
- **イベント・フラグ待ち状態**
- **資源待ち状態**
- **メッセージ待ち状態**
- **固定長メモリ・ブロック待ち状態**
- **可変長メモリ・ブロック待ち状態**

・1ビット・イベント・フラグ待ち状態

TTS_SUS（H' 08）：suspend状態

TTS_WAS（H' 0c）：wait_suspend状態

・起床待ち状態とsuspend状態が複合した状態

・時間経過待ち状態とsuspend状態が複合した状態

・イベント・フラグ待ち状態とsuspend状態が複合した状態

・資源待ち状態とsuspend状態が複合した状態

・メッセージ待ち状態とsuspend状態が複合した状態

・固定長メモリ・ブロック待ち状態とsuspend状態が複合した状態

・可変長メモリ・ブロック待ち状態とsuspend状態が複合した状態

・1ビット・イベント・フラグ待ち状態とsuspend状態が複合した状態

TTS_DMT（H' 10）：dormant状態

TTS_WTX（H' 20）：wait状態

・タスク実行権待ち状態

TTS_WTS（H' 28）：wait_suspend状態

・タスク実行権待ち状態とsuspend状態が複合した状態

tskwait …　wait状態の種類

TTW_SLP（H' 0001）　：起床待ち状態

TTW_DLY（H' 0002）　：時間経過待ち状態

TTW_FLG（H' 0010）　：イベント・フラグ待ち状態

TTW_SEM（H' 0020）　：資源待ち状態

TTW_MBX（H' 0040）　：メッセージ待ち状態

TTW_MPL（H' 1000）　：可変長メモリ・ブロック待ち状態

TTW_MPF（H' 2000）　：固定長メモリ・ブロック待ち状態

TTW_1FLG（H' 4000）：1ビット・イベント・フラグ待ち状態

wid　　…　対象オブジェクト（セマフォ，イベント・フラグなど）のID番号

注意1. tskstsの値がTTS_WAI, TTS_WAS以外の場合，tskwaitの内容は不定となります。

2. tskstsの値がTTS_WTX, またはtskwaitの値がTTW_FLG, TTW_SEM, TTW_MBX, TTW_MPL, TTW_MPF, TTW_1FLG以外の場合，widの内容は不定となります。

戻　り　値

E_OK	0	正常終了

10. 5. 2　タスク付属同期機能システム・コール

この項では，タスクに従属した同期操作を行うシステム・コールのグループ（タスク付属同期機能システム・コール）について説明しています。

表10－4に，タスク付属同期機能システム・コールの一覧を示します。

表10－4　タスク付属同期機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
sus_tsk	他タスクをsuspend状態へ移行する	×
rsm_tsk	suspend状態のタスクを再開する	○
frsm_tsk	suspend状態のタスクを強制的に再開する	○
slp_tsk	自タスクを起床待ち状態へ移行する	○
tslp_tsk	自タスクを起床待ち状態へ移行する（タイムアウトあり）	○
wup_tsk	他タスクを起床する	○
can_wup	タスクの起床要求を無効化する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Suspend Task

sus_tsk

タスク／ハンドラ

概　　要

他タスクをsuspend**状態へ移行する。**

C言語形式

```
ER sus_tsk（ID tskid）；
```

パラメータ

I/O	パラメータ	説　明
I	ID　*tskid*；	**タスクの**ID**番号**

説　　明

***tskid*で説明されたタスクにサスペンド要求を発行（サスペンド要求カウンタに**0x1**を加算）します。**

ただし，このシステム・コールを発行したとき，対象タスクがready**状態または**wait**状態であった場合には，サスペンド要求の発行（サスペンド要求カウンタの加算処理）とともに，対象タスクを**ready**状態から**suspend**状態へ，または**wait**状態から**wait_suspend**状態へと遷移させます。**

注意　**RX850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，このシステム・コールでは，サスペンド要求数が127回を越えるような場合には，サスペンド要求カウンタの加算処理は行わず，戻り値として**E_QOVR**を返しています。**

戻 り 値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクが自タスクまたは**dormant**状態である**
E_QOVR	－73	**サスペンド要求数が127回を越えた**

Resume Task

rsm_tsk

タスク／ハンドラ

概　　要

suspend**状態のタスクを再開する。**

C**言語形式**

```
ER rsm_tsk（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*tskid*；	**タスクの**ID**番号**

説　　明

*tskid***で指定されたタスクに発行されているサスペンド要求を**1**回分だけ解除（サスペンド要求カウンタから**0x1**を減算）します。**

なお，このシステム・コールの発行により対象タスクのサスペンド要求カウンタが0x0**となった場合には，対象タスクを**suspend**状態から**ready**状態へ，または**wait_suspend**状態から**wait**状態へと遷移させます。**

注意　このシステム･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suspend**状態または**wait_suspend**状態以外の場合には，サスペンド要求カウンタの減算処理は行わず，戻り値として**E_OBJ**を返しています。**

戻　り　値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクが**suspend**状態または**wait_suspend**状態でない**

Force Resume Task

frsm_tsk

タスク／ハンドラ

概　　要

suspend状態のタスクを強制的に再開する。

C言語形式

```
ER frsm_tsk（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*tskid*；	タスクのID番号

説　　明

*tskid*で指定されたタスクに発行されているサスペンド要求をすべて解除（サスペンド要求カウンタに0x0を設定）します。

これにより，対象タスクは，suspend状態からready状態へ，またはwait_suspend状態からwait状態へと遷移します。

注意　このシステム・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suspend状態，またはwait_suspend状態以外の場合には，サスペンド要求カウンタの設定処理は行わず，戻り値としてE_OBJを返しています。

戻　り　値

E_OK	0	正常終了
E_OBJ	－63	対象タスクがsuspend状態またはwait_suspend状態でない

Sleep Task

slp_tsk

タスク

概　　要

自タスクを起床待ち状態へ移行する。

C言語形式

```
ER slp_tsk () ;
```

パラメータ

なし

説　　明

自タスクに発行されている起床要求を1回分だけ解除（起床要求カウンタから0x1を減算）します。

ただし，このシステム・コールを発行したとき，自タスクの起床要求カウンタが0x0であった場合には，起床要求の解除（起床要求カウンタの減算処理）は行わず，自タスクをrun状態からwait状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態の解除は，wup_tsk, ret_wup, rel_waiシステム・コールが発行されたときに行われ，起床待ち状態からready状態へと遷移します。

戻 り 値

E_OK	0	正常終了
E_RLWAI	－86	rel_waiシステム・コールにより，起床待ち状態を強制的に解除された

Sleep Task with Timeout

tslp_tsk

タスク

概　　要

自タスクを起床待ち状態へ移行する（タイムアウトあり）。

C言語形式

```
ER tslp_tsk（TMO tmout）；
```

パラメータ

I/O	パラメータ	説　　明
I	TMO　*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_FEVR（－1）　：永久待ち 数値　：待ち時間

説　　明

自タスクに発行されている起床要求を1回分だけ解除（起床要求カウンタから0x1を減算）します。

ただし，このシステム・コールを発行したとき，自タスクの起床要求カウンタが0x0であった場合には，起床要求の解除（起床要求カウンタの減算処理）は行わず，自タスクをrun状態からwait状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはwup_tsk，ret_wup，rel_waiシステム・コールが発行されたときに行われ，起床待ち状態からready状態へと遷移します。

注意　このシステム・コールを発行するとき，待ち時間*tmout*に0x0を指定した場合，動作の保証がありません。

戻 り 値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，起床待ち状態を強制的に解除された

Wakeup Task

wup_tsk

タスク／ハンドラ

概　　要

他タスクを起床する。

C言語形式

```
ER wup_tsk（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*tskid*；	タスクのID番号

説　　明

*tskid*で指定されたタスクに起床要求を発行（起床要求カウンタに0x1を加算）します。

ただし，このシステム・コールを発行したとき，対象タスクがwait状態（起床待ち状態）であった場合には，起床要求の発行（起床要求カウンタの加算処理）は行わず，対象タスクを起床待ち状態からready状態へと遷移させます。

注意　RX850が管理す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，このシステム・コールでは，起床要求数が127回を越えるような場合には，起床要求カウンタの加算処理は行わず，戻り値としてE_QOVRを返しています。

戻 り 値

E_OK	0	正常終了
E_OBJ	－63	対象タスクが自タスクまたはdormant状態である
E_QOVR	－73	起床要求数が127回を越えた

Cancel Wakeup Task

can_wup

タスク／ハンドラ

概　　要

タスクの起床要求を無効化する。

C言語形式

```
ER can_wup（INT ＊p_wupcnt, ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
O	INT　＊*p_wupcnt*；	**起床要求数を格納する領域のアドレス**
I	ID　*tskid*；	**タスクのID番号**

説　　明

***tskid*で指定されたタスクに発行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，このシステム・コールの発行により解除された起床要求数は，*p_wupcnt*で指定される領域に格納されます。

戻 り 値

E_OK	0	**正常終了**
E_OBJ	－63	**対象タスクがdormant状態である**

10. 5. 3　同期通信機能システム・コール

この項では，タスク間の同期（排他制限，待ち合わせ），および通信を行うシステム・コールのグループ（同期通信機能システム・コール）について説明しています。

表10－5に，同期通信機能システム・コールの一覧を示します。

表10－5　同期通信機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
sig_sem	資源を返却する	○
wai_sem	資源を獲得する	○
preq_sem	資源を獲得する（ポーリング）	×
twai_sem	資源を獲得する（タイムアウトあり）	○
ref_sem	セマフォ情報を獲得する	×
set_flg	ビット・パターンを設定する	○
clr_flg	ビット・パターンをクリアする	×
wai_flg	ビット・パターンをチェックする	○
pol_flg	ビット・パターンをチェックする（ポーリング）	×
twai_flg	ビット・パターンをチェックする（タイムアウトあり）	○
ref_flg	イベント・フラグ情報を獲得する	×
vset_flg1	ビットを設定する	○
vclr_flg1	ビットをクリアする	×
vwai_flg1	ビットをチェックする	○
vpol_flg1	ビットをチェックする（ポーリング）	×
vtwai_flg1	ビットをチェックする（タイムアウトあり）	○
vref_flg1	1ビット・イベント・フラグ情報を獲得する	×
snd_msg	メッセージを送信する	○
rcv_msg	メッセージを受信する	○
prcv_msg	メッセージを受信する（ポーリング）	×
trcv_msg	メッセージを受信する（タイムアウトあり）	○
ref_mbx	メールボックス情報を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Signal Semaphore

sig_sem

タスク／ハンドラ

概　　要

資源を返却する。

C言語形式

```
ER sig_sem（ID semid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*semid*；	セマフォのID番号

説　　明

*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，このシステム・コールを発行したとき，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却処理（セマフォ・カウンタの加算処理）は行わず，該当するタスク（待ちキューの先頭タスク）に資源を渡します。

これにより，該当するタスクは，待ちキューから外れ，wait状態（資源待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意　RX850が管理するセマフォ・カウンタは，7ビット幅で構成されています。このため，このシステム・コールでは，資源数が127個を越えるような場合には，セマフォ・カウンタの加算処理は行わず，戻り値としてE_QOVRを返しています。

戻 り 値

E_OK	0	正常終了
E_QOVR	－73	資源数が127個を越えた

Wait on Semaphore

wai_sem

タスク

概　　要

資源を獲得する。

C言語形式

```
ER wai_sem（ID semid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*semid*；	セマフォのID番号

説　　明

*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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は，sig_sem, rel_waiシステム・コールが発行されたときに行われ，資源待ち状態からready状態へと遷移します。

戻　り　値

E_OK	0	正常終了
E_RLWAI	－86	rel_waiシステム・コールにより，資源待ち状態を強制的に解除された

Poll and Request Semaphore

preq_sem

タスク／ハンドラ

概　　要

資源を獲得する（ポーリング）。

C言語形式

```
ER preq_sem（ID semid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*semid*；	セマフォのID番号

説　　明

*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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	対象セマフォの資源数が0x0である

Wait on Semaphore with Timeout

twai_sem

タスク

概　　要

資源を獲得する（タイムアウトあり）。

C言語形式

```
ER twai_sem（ID semid, TMO tmout）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*semid*；	セマフォのID番号
I	TMO　*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　：即時リターン TMO_FEVR（－1）　：永久待ち 数値　：待ち時間

説　　明

*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，このシステム・コールを発行したとき，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはsig_sem, rel_waiシステム・コールが発行されたときに行われ，資源待ち状態からready状態へと遷移します。

戻 り 値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，資源待ち状態を強制的に解除された

ref_sem

Refer Semaphore Status

タスク／ハンドラ

概　　要

セマフォ情報を獲得する。

C言語形式

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

パラメータ

I/O	パラメータ	説　明
O	T_RSEM ＊*pk_rsem*；	セマフォ情報を格納するパケットの先頭アドレス
I	ID *semid*；	セマフォのID番号

セマフォ情報T_RSEMの構造

```
typedef struct t_rsem {
        VP      exinf；  /＊ 拡張情報          ＊/
        BOOL_ID wtsk；   /＊ 待ちタスクの有無 ＊/
        INT     semcnt； /＊ 現在の資源数      ＊/
} T_RSEM；
```

説　　明

***semid*で指定されたセマフォのセマフォ情報（拡張情報，待ちタスクの有無など）を*pk_rsem*で指定されるパケットに格納します。**

セマフォ情報の詳細を次に示します。

- exinf … **拡張情報**
- wtsk … **待ちタスクの有無**
  - FALSE（0）：**待ちタスクなし**
  - **数値**：**待ちキューの先頭タスクのID番号**
- semcnt … **現在の資源数**

戻 り 値

E_OK	0	**正常終了**

set_flg

Set Event Flag

タスク／ハンドラ

概　　要

ビット・パターンを設定する。

C言語形式

```
ER  set_flg（ID flgid, UINT setptn）；
```

パラメータ

I/O	パラメータ	説　明
I	ID　*flgid*；	イベント・フラグのID番号
I	UINT　*setptn*；	設定するビット・パターン（32ビット幅）

説　　明

*flgid*で指定されたイベント・フラグのビット・パターンと*setptn*で指定されたビット・パターンの論理和ORをとり，その結果を対象イベント・フラグに設定します。

なお，このシステム・コールを発行したとき，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待ち条件を満足した場合には，該当するタスクを待ちキューから外します。

これにより，該当するタスクは，wait状態（イベント・フラグ待ち状態）からready状態へ，または，wait_suspend状態からsuspend状態へと遷移します。

例　このシステム・コールを発行するとき，対象イベント・フラグのビット・パターンがB' 1100, *setptn*で指定されたビット・パターンがB' 1010の場合，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

E_OK	0	正常終了

clr_flg

Clear Event Flag

タスク／ハンドラ

概　　要

ビット・パターンをクリアする。

C言語形式

```
ER  clr_flg（ID flgid, UINT clrptn）；
```

パラメータ

I/O	パラメータ	説　明
I	ID　*flgid*；	イベント・フラグのID番号
I	UINT　*clrptn*；	クリアするビット・パターン（32ビット幅）

説　　明

*flgid*で指定されたイベント・フラグのビット・パターンと*clrptn*で指定されたビット・パターンの論理積ANDをとり，その結果を対象イベント・フラグに設定します。

例　このシステム・コールを発行するとき，対象イベント・フラグのビット・パターンがB' 1100, *clrptn*で指定されたビット・パターンがB' 1010の場合，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。

戻 り 値

E_OK	0	正常終了

Wait Event Flag

wai_flg

タスク

概　　要

ビット・パターンをチェックする。

C言語形式

```
ER  wai_flg（UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode）；
```

パラメータ

I/O	パラメータ	説　　明
O	UINT　*p_flgptn；	条件成立時のビット・パターンを格納する領域のアドレス
I	ID　*flgid*；	イベント・フラグのID番号
I	UINT　*waiptn*；	要求するビット・パターン（32ビット幅）
I	UINT　*wfmode*；	待ち条件／条件成立時の指定 TWF_ANDW（0）：AND待ち TWF_ORW（2）：OR待ち TWF_CLR（1）：ビット・パターンをクリアする

説　　明

*waiptn*で指定された要求ビット・パターンと*wfmode*で指定された待ち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どうかをチェックします。

なお，待ち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*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューにキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は，set_flgシステム・コールの発行により待ち条件を満足するようなビット・パターンが設定されたとき，またはrel_waiシステム・コールが発行されたときに行われ，イベント・フラグ待ち状態からready状態へと遷移します。

*wfmode*の指定形式を次に示します。

- *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どうかをチェックします。

- *wfmode* =（TWF_ANDW | TWF_CLR）

  *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どうかをチェックします。

  なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B' 0000を設定）されます。

・*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どうかをチェックします。

・*wfmode* = （TWF_ORW|TWF_CLR）

*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どうかをチェックします。

なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B' 0000を設定）されます。

注意1. RX850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が1タスクに限られています。

このため，すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値としてE_OBJを返しています。

2. rel_waiシステム・コール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場合には，*p_flgptn*で指定される領域の内容は不定となります。

戻　り　値

E_OK	0	正常終了
E_OBJ	－63	すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した
E_RLWAI	－86	rel_waiシステム・コールにより，イベント・フラグ待ち状態を強制的に解除された

pol_flg

Poll Event Flag

タスク／ハンドラ

概　　要

ビット・パターンをチェックする（ポーリング）。

C言語形式

```
ER  pol_flg（UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode）；
```

パラメータ

I/O	パラメータ	説　　明
O	UINT　*p_flgptn；*	条件成立時のビット・パターンを格納する領域のアドレス
I	ID　*flgid；*	イベント・フラグのID番号
I	UINT　*waiptn；*	要求するビット・パターン（32ビット幅）
I	UINT　*wfmode；*	待ち条件／条件成立時の指定 TWF_ANDW（0）　：AND待ち TWF_ORW（2）　：OR待ち TWF_CLR（1）　：ビット・パターンをクリアする

説　　明

*waiptn*で指定された要求ビット・パターンと*wfmode*で指定された待ち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どうかをチェックします。

なお，待ち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*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが待ち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。

*wfmode*の指定形式を次に示します。

- *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どうかをチェックします。

- *wfmode* = （TWF_ANDW | TWF_CLR）

  *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どうかをチェックします。

  なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B'0000を設定）されます。

- *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どうかをチェックします。

・*wfmode* =　（TWF_ORW｜TWF_CLR）

*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どうかをチェックします。

なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B' 0000を設定）されます。

注意　RX850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が1タスクに限られています。

このため，すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値としてE_OBJを返しています。

戻　り　値

E_OK	0	正常終了
E_OBJ	－63	すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した
E_TMOUT	－85	対象イベント・フラグのビット・パターンが待ち条件を満足していない

twai_flg

Wait Event Flag with Timeout

タスク

概　　要

ビット・パターンをチェックする（タイムアウトあり）。

C言語形式

```
ER  twai_flg（UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO tmout）；
```

パラメータ

I/O	パラメータ		説　　明
O	UINT	*p_flgptn*；	条件成立時のビット・パターンを格納する領域のアドレス
I	ID	*flgid*；	イベント・フラグのID番号
I	UINT	*waiptn*；	要求するビット・パターン（32ビット幅）
I	UINT	*wfmode*；	待ち条件／条件成立時の指定 TWF_ANDW（0）　：AND待ち TWF_ORW（2）　：OR待ち TWF_CLR（1）　：ビット・パターンをクリアする
I	TMO	*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　：即時リターン TMO_FEVR（−1）　：永久待ち 数値　：待ち時間

説　　明

*waiptn*で指定された要求ビット・パターンと*wfmode*で指定された待ち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どうかをチェックします。

なお，待ち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*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューにキューイングしたのち，run状態からwait状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはset_flgシステム・コールの発行により待ち条件を満足するようなビット・パターンが設定されたとき，またはrel_waiシステム・コールが発行されたときに行われ，イベント・フラグ待ち状態からready状態へと遷移します。

*wfmode*の指定形式を次に示します。

- *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どうかをチェックします。

- *wfmode* =　（TWF_ANDW | TWF_CLR）

  *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どうかをチェックし

ます。

なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B'0000を設定）されます。

- *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どうかをチェックします。

- *wfmode* = （TWF_ORW|TWF_CLR）

  *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どうかをチェックします。

  なお，待ち条件が満足したときには，対象イベント・フラグのビット・パターンがクリア（B'0000を設定）されます。

注意1. RX850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が1タスクに限られています。

このため，すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値としてE_OBJを返しています。

2. rel_waiシステム・コール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場合には，*p_flgptn*で指定される領域の内容は不定となります。

戻　り　値

E_OK	0	正常終了
E_OBJ	－63	すでに待ちタスクがキューイングされているイベント・フラグに対して，このシステム・コールを発行した
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，イベント・フラグ待ち状態を強制的に解除された

ref_flg

Refer Event Flag Status

タスク／ハンドラ

概　　要

イベント・フラグ情報を獲得する。

C言語形式

```
ER  ref_flg（T_RFLG ＊pk_rflg, ID flgid）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RFLG　＊*pk_rflg*；	**イベント・フラグ情報を格納するパケットの先頭アドレス**
I	ID　　*flgid*；	**イベント・フラグのID番号**

イベント・フラグ情報T_RFLGの構造

```
typedef struct t_rflg {
        VP      exinf；  /＊ 拡張情報              ＊/
        BOOL_ID wtsk；   /＊ 待ちタスクの有無      ＊/
        UINT    flgptn； /＊ 現在のビット・パターン ＊/
} T_RFLG；
```

説　　明

***flgid*で指定されたイベント・フラグのイベント・フラグ情報（拡張情報，待ちタスクの有無など）を*pk_rflg*で指定されるパケットに格納します。**

イベント・フラグ情報の詳細を次に示します。

exinf　… **拡張情報**

wtsk　… **待ちタスクの有無**

FALSE（0）：**待ちタスクなし**

数値　：**待ちキューのタスクのID番号**

flgptn　… **現在のビット・パターン**

戻 り 値

E_OK	0	**正常終了**

Set 1Bit Event Flag

vset_flg1

タスク／ハンドラ

概　　要

ビットを設定する。

C言語形式

```
ER  vset_flg1（ID flgid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　　*flgid*；	1ビット・イベント・フラグのID番号

説　　明

*flgid*で指定された1ビット・イベント・フラグに“1”を設定します。

なお，このシステム・コールを発行したとき，対象1ビット・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た場合には，待ちキューの先頭タスクからビット・クリア指定を行っているタスクまでを待ちキューから外します。

これにより，待ちキューから外れたタスクは，wait状態（1ビット・イベント・フラグ待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

戻　り　値

E_OK	0	正常終了

Clear 1Bit Event Flag

vclr_flg1

タスク／ハンドラ

概　　要

ビットをクリアする。

C言語形式

```
ER  vcrl_flg1（ID flgid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　　*flgid*；	1ビット・イベント・フラグのID番号

説　　明

*flgid*で指定された1ビット・イベント・フラグに“0”を設定します。

注意　このシステム・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，すでにこのシステム・コールが発行され，対象1ビット・イベント・フラグ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。

戻　り　値

E_OK	0	正常終了

Wait 1Bit Event Flag

vwai_flg1

タスク

概　　要

ビットをチェックする。

C言語形式

```
ER  vwai_flg1（ID flgid, UINT wfmode）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*flgid*；	1ビット・イベント・フラグのID番号
I	UINT　*wfmode*；	条件成立時の指定 TWF_NCL（0）：ビットをクリアしない TWF_CLR（1）：ビットをクリアする

説　　明

*flgid*で指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，自タスクを対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（1ビット・イベント・フラグ待ち状態）へと遷移させます。

なお，1ビット・イベント・フラグ待ち状態の解除は，vset_flg1，rel_waiシステム・コールが発行されたときに行われ，1ビット・イベント・フラグ待ち状態からready状態へと遷移します。

戻 り 値

E_OK	0	正常終了
E_RLWAI	－86	rel_waiシステム・コールにより，1ビット・イベント・フラグ待ち状態を強制的に解除された

Poll 1Bit Event Flag

vpol_flg1

タスク／ハンドラ

概　　要

ビットをチェックする（ポーリング）。

C言語形式

```
ER  vpol_flg1（ID flgid, UINT wfmode）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　*flgid*；	1ビット・イベント・フラグのID番号
I	UINT　*wfmode*；	条件成立時の指定 TWF_NCL（0）：ビットをクリアしない TWF_CLR（1）：ビットをクリアする

説　　明

*flgid*で指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，戻り値としてE_TMOUTが返されます。

戻 り 値

E_OK	0	正常終了
E_TMOUT	－85	対象1ビット・イベント・フラグに“1”が設定されていない

Wait 1Bit Event Flag with Timeout

vtwai_flg1

タスク

概　　要

ビットをチェックする（タイムアウトあり）。

C言語形式

ER　vtwai_flg1（ID *flgid*, UINT *wfmode*, TMO *tmout*）；

パラメータ

I/O	パラメータ		説　　明
I	ID	*flgid*；	1ビット・イベント・フラグのID番号
I	UINT	*wfmode*；	条件成立時の指定 TWF_NCL（0）　：ビットをクリアしない TWF_CLR（1）　：ビットをクリアする
I	TMO	*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　　：即時リターン TMO_FEVR（－1）：永久待ち 数値　　：待ち時間

説　　明

*flgid*で指定された1ビット・イベント・フラグに“1”が設定されているかどうかをチェックします。

ただし，このシステム・コールを発行したとき，対象1ビット・イベント・フラグに“1”が設定されていなかった場合には，自タスクを対象1ビット・イベント・フラグの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（1ビット・イベント・フラグ待ち状態）へと遷移させます。

なお，1ビット・イベント・フラグ待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはvset_flag1, rel_waiシステム・コールが発行されたときに行われ，1ビット・イベント・フラグ待ち状態からready状態へと遷移します。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，1ビット・イベント・フラグ待ち状態を強制的に解除された

Refer 1Bit Event Flag Status

vref_flg1

タスク／ハンドラ

概　　要

1ビット・イベント・フラグ情報を獲得する。

C言語形式

```
ER  vref_flg1（T_RFLG ＊pk_rflg, ID flgid）；
```

パラメータ

I/O	パラメータ	説　明
O	T_RFLG　＊*pk_rflg*；	1ビット・イベント・フラグ情報を格納するパケットの先頭アドレス
I	ID　　*flgid*；	1ビット・イベント・フラグのID番号

1ビット・イベント・フラグ情報T_RFLGの構造

```
typedef struct  t_rflg {
        VP      exinf；  /＊ 拡張情報          ＊/
        BOOL_ID wtsk；   /＊ 待ちタスクの有無 ＊/
        UINT    flgptn； /＊ 現在のビット値    ＊/
} T_RFLG；
```

説　　明

*flgid*で指定された1ビット・イベント・フラグの1ビット・イベント・フラグ情報（拡張情報，待ちタスクの有無など）を*pk_rflg*で指定されるパケットに格納します。

1ビット・イベント・フラグ情報の詳細を次に示します。

exinf　…　拡張情報

wtsk　…　待ちタスクの有無

FALSE（0）：待ちタスクなし

数値　　：待ちキューの先頭タスクのID番号

flgptn　…　現在のビット値

戻 り 値

E_OK	0	正常終了

Send Message to Mailbox

snd_msg

タスク／ハンドラ

概　　要

メッセージを送信する。

C言語形式

```
ER  snd_msg（ID mbxid, T_MSG ＊pk_msg）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　　*mbxid*；	メールボックスのID番号
I	T_MSG　　＊*pk_msg*；	メッセージを格納した領域のアドレス

メッセージT_MSGの構造

```
typedef struct t_msg {
        VW      msgrfu；        /＊ メッセージ管理領域 ＊/
        PRI     msgpri；        /＊ メッセージの優先度 ＊/
        VB      msgcont [] ；   /＊ メッセージ本体     ＊/
} T_MSG；
```

説　　明

*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（メッセージ用待ちキューにメッセージをキューイング）します。

ただし，このシステム・コールを発行したとき，対象メールボックスのタスク用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操作は行わず，該当するタスク（タスク用待ちキューの先頭タスク）にメッセージを渡します。

これにより，該当するタスクはタスク用待ちキューから外れ，wait状態（メッセージ待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意1. メッセージを対象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ときのキューイング方法は，対象メールボックス生成時（コンフィギュレーション時）に指定された順（FIFO順，優先度順）に行われます。

2. RX850では，メッセージの先頭4バイト（メッセージ管理領域msgrfu）をメッセージ用待ちキューにキューイングするときのリンク領域として使用しています。このため，メッセージを対象メールボックスに送信する場合は，このシステム・コールを発行する前にmsgrfuに“0x0”を設定する必要があります。

戻 り 値

E_OK	0	正常終了

Receive Message from Mailbox

rcv_msg

タスク

概　　要

メッセージを受信する。

C言語形式

ER　rcv_msg（T_MSG　＊＊*ppk_msg*, ID *mbxid*）；

パラメータ

I/O	パラメータ	説　　明
O	T_MSG　＊＊*ppk_msg*；	**メッセージの先頭アドレスを格納する領域のアドレス**
I	ID　*mbxid*；	**メールボックスのID番号**

説　　明

***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*で指定される領域に格納します。**

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたのち，run状態からwait状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態の解除は，snd_msg, rel_waiシステム・コールが発行されたときに行われ，メッセージ待ち状態からready状態へと遷移します。

戻 り 値

E_OK	0	**正常終了**
E_RLWAI	－86	rel_wai**システム・コールにより，メッセージ待ち状態を強制的に解除された**

Poll and Receive Message from Mailbox

prcv_msg

タスク／ハンドラ

概　　要

メッセージを受信する（ポーリング）。

C言語形式

```
ER  prcv_msg（T_MSG ＊＊ppk_msg, ID mbxid）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_MSG　＊＊*ppk_msg*；	メッセージの先頭アドレスを格納する領域のアドレス
I	ID　*mbxid*；	メールボックスのID番号

説　　明

*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*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	対象メールボックスにメッセージが存在しない

Receive Message from Mailbox with Timeout

trcv_msg

タスク

概　　要

メッセージを受信する（タイムアウトあり）。

C言語形式

```
ER  trcv_msg（T_MSG ＊＊ppk_msg, ID mbxid, TMO tmout）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_MSG　＊＊*ppk_msg*；	メッセージの先頭アドレスを格納する領域のアドレス
I	ID　*mbxid*；	メールボックスのID番号
I	TMO　*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　：即時リターン TMO_FEVR（－1）　：永久待ち 数値　：待ち時間

説　　明

*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*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，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（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたのち，run状態からwait状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはsnd_msg, rel_waiシステム・コールが発行されたときに行われ，メッセージ待ち状態からready状態へと遷移します。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，メッセージ待ち状態を強制的に解除された

ref_mbx

Refer Mailbox Status

タスク／ハンドラ

概　　要

メールボックス情報を獲得する。

C言語形式

```
ER  ref_mbx（T_RMBX ＊pk_rmbx, ID mbxid）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RMBX　＊*pk_rmbx*；	メールボックス情報を格納するパケットの先頭アドレス
I	ID　　*mbxid*；	メールボックスのID番号

メールボックス情報T_RMBXの構造

```
typedef struct  t_rmbx {
        VP      exinf；       /＊ 拡張情報            ＊/
        BOOL_ID wtsk；        /＊ 待ちタスクの有無    ＊/
        T_MSG   ＊＊ppk_msg； /＊ 待ちメッセージの有無＊/
} T_RMBX；
```

説　　明

*mbxid*で指定されたメールボックスのメールボックス情報（拡張情報，待ちタスクの有無など）を*pk_rmbx*で指定されるパケットに格納します。

メールボックス情報の詳細を次に示します。

exinf　… 拡張情報

wtsk　… 待ちタスクの有無

　　FALSE（0）：待ちタスクなし

　　数値　　：待ちキューの先頭タスクのID番号

ppk_msg … 待ちメッセージの有無

　　NADR（－1）：待ちメッセージなし

　　数値　　：待ちキューの先頭メッセージのアドレス

戻　り　値

E_OK	0	正常終了

10. 5. 4　割り込み管理機能システム・コール

この項では，割り込みに依存した処理を行うシステム・コールのグループ（割り込み管理機能システム・コール）について説明しています。

表10－6に，割り込み管理機能システム・コールの一覧を示します。

表10－6　割り込み管理機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
ret_int	直接起動割り込みりハンドラから復帰する	○
ret_wup	他タスクの起床，および，直接起動割り込みハンドラからの復帰を行う	○
loc_cpu	マスカブル割り込みの受け付け，および，ディスパッチ処理を禁止する	×
unl_cpu	マスカブル割り込みの受け付け，および，ディスパッチ処理を再開する	○
dis_int	マスカブル割り込みの受け付けを禁止する	×
ena_int	マスカブル割り込みの受け付けを再開する	×
chg_icr	割り込み制御レジスタの内容を変更する	×
ref_icr	割り込み制御レジスタの内容を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Return from Interrupt Handler

ret_int

直接起動割り込みハンドラ

概　　要

直接起動割り込みハンドラから復帰する。

C言語形式

```
void ret_int () ;
```

パラメータ

なし

説　　明

直接起動割り込みハンドラから復帰します。

なお，RX850では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要なシステム・コール（chg_pri, sig_semなど）が発行された場合，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（このシステム・コールまたはret_wupシステム・コールの発行）まで遅延され，一括して行うようにしています。

注意1. **このシステム・コールでは，外部割り込みコントローラに対する処理終了通知（EOIコマンドの発行など）を行っていません。このため，外部割り込み要求によって起動した直接起動割り込みハンドラから復帰する場合は，このシステム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。**

2. **間接起動割り込みハンドラをC言語で記述した場合，間接起動割り込みハンドラからの復帰は，次のように記述します。**

```
return（0xff）；
```

3. **間接起動割り込みハンドラをアセンブリ言語で記述した場合，間接起動割り込みハンドラからの復帰は，次のように記述します。**

```
mov     0xff, r10
jmp     [lp]
```

戻 り 値

なし

ret_wup

Retrun and Wakeup Task

直接起動割り込みハンドラ

概　　要

他タスクの起床，および直接起動割り込みハンドラからの復帰を行う。

C言語形式

```
void  ret_wup（ID tskid）；
```

パラメータ

I/O	パラメータ	説　　明
I	ID　　*tskid*；	タスクのID番号

説　　明

*tskid*で指定されたタスクに起床要求を発行（起床要求カウンタに0x1を加算）したのち，直接起動割り込みハンドラから復帰します。

ただし，このシステム・コールを発行したとき，対象タスクがwait状態（起床待ち状態）であった場合には，起床要求の発行（起床要求カウンタの加算処理）は行わず，対象タスクを起床待ち状態からready状態へと遷移させます。

なお，RX850では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_semなど）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（ret_intまたはこのシステム・コールの発行）まで遅延され，一括して行うようにしています。

注意1. **このシステム・コールでは，外部割り込みコントローラに対する処理終了通知（EOIコマンドの発行など）を行っていません。このため，外部割り込み要求によって起動した直接起動割り込みハンドラから復帰する場合は，このシステム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。**

2. **このシステム・コールでは，**
   - **対象タスクがrun状態またはdormant状態である**
   - **起床要求数が127回を越えた**

   などのエラーが生じた場合，直接起動割り込みハンドラからの復帰処理のみを行います。
3. **間接起動割り込みハンドラをC言語で記述した場合，他タスクの起床および間接起動割り込みハンドラからの復帰は，次のように記述します。**

   ```
   return（ID tskid）；
   ```
4. **間接起動割り込みハンドラをアセンブリ言語で記述した場合，他タスクの起床および割り込みハンドラからの復帰は，次のように記述します。**

   ```
   mov   tskid, r10
   jmp   [lp]
   ```

戻　り　値

なし

Lock CPU

loc_cpu

タスク

概　　要

マスカブル割り込みの受け付けおよびディスパッチ処理を禁止する。

C言語形式

```
ER  loc_cpu () ;
```

パラメータ

なし

説　　明

マスカブル割り込みの受け付けおよびディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールの発行からunl_cpuシステム・コールが発行されるまでの間，マスカブル割り込みの受け付けおよびディスパッチ処理が禁止されます。

なお，RX850では，このシステムコールの発行からunl_cpuシステム・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ていた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は，unl_cpuシステム・コールが発行されるまで遅延されます。また，タスクのスケジューリング処理が必要なシステム・コール（chg_pri, sig_semなど）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，unl_cpuシステム・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。

注意1. **このシステム・コールでは，禁止要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けおよびディスパッチ処理が禁止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。**

2. **このシステム・コールの発行からunl_cpuシステム・コールの発行までの間，自タスクの状態遷移を引き起こす可能性のあるシステム・コール（ext_tsk, wai_semなど）を発行した場合，動作の保証はありません。**

3. **このシステム・コールの発行からunl_cpuシステム・コールの発行までの間，ena_intシステム・コールを発行した場合，動作の保証がありません。**

戻 り 値

E_OK	0	正常終了

Unlock CPU

unl_cpu

タスク

概　　要

マスカブル割り込みの受け付けおよびディスパッチ処理を再開する。

C言語形式

```
ER  unl_cpu () ;
```

パラメータ

なし

説　　明

loc_cpu**システム・コール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およびディスパッチ処理（タスクのスケジューリング処理）を再開します。**

なお，RX850では，loc_cpu**システム・コールの発行からこのシステム・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ていた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は，このシステム・コールが発行されるまで遅延されます。また，タスクのスケジューリング処理が必要なシステム・コール（**chg_pri, sig_sem**など）が発行された場合，待ちキューのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，このシステム・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。**

注意1. **このシステム・コールでは，再開要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けおよびディスパッチ処理が再開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。**

2. dis_int**システム・コールの発行により禁止されたマスカブル割り込みの受け付けは，このシステム・コールを発行することにより再開されます。**

3. dis_dsp**システム・コールの発行により禁止されたディスパッチ処理は，このシステム・コールを発行することにより再開されます。**

戻　り　値

E_OK	0	**正常終了**

Disable Interrupt

dis_int

タスク／ハンドラ

概　　要

マスカブル割り込みの受け付けを禁止する。

C言語形式

```
ER  dis_int () ;
```

パラメータ

なし

説　　明

マスカブル割り込みの受け付けを禁止します。

これにより，このシステム・コールの発行からena_intシステム・コールが発行されるまでの間，マスカブル割り込みの受け付けが禁止されます。

なお，RX850では，このシステム・コールの発行からena_intシステム・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ていた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は，ena_intシステム・コールが発行されるまで遅延されます。

注意　このシステム・コールでは，禁止要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けが禁止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。

戻 り 値

E_OK	0	正常終了

Enable Interrupt

ena_int

タスク／ハンドラ

概　　要

マスカブル割り込みの受け付けを再開する。

C言語形式

```
ER  ena_int () ;
```

パラメータ

なし

説　　明

dis_int**システム・コール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を再開します。**

なお，RX850**では，**dis_int**システム・コールの発行からこのシステム・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ていた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は，このシステム・コールが発行されるまで遅延されます。**

注意　このシステム・コールでは，再開要求のキューイングが行われません。このため，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けが再開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。

戻 り 値

E_OK	0	**正常終了**

Change Interrupt Control Register

chg_icr

タスク／ハンドラ

概　　要

割り込み制御レジスタの内容を変更する。

C言語形式

```
ER  chg_icr（UINT eintno, UB icrcmd）；
```

パラメータ

I/O	パラメータ	説　　明
I	UINT　*eintno*；	割り込み要求番号
I	UB　*icrcmd*；	割り込み要求フラグの指定 ICR_CLRINT（0x20）：割り込み要求なし 割り込みマスク・フラグの指定 ICR_CLRMSK（0x10）：割り込み処理を許可 ICR_SETMSK（0x40）：割り込み処理を禁止 割り込み優先順位変更の指定 ICR_CHGLVL（0x08）：割り込み優先順位を変更 割り込み優先順位の指定 数値（0〜7）：割り込み優先順位

説　　明

*eintno*で指定された割り込み制御レジスタの内容を*icrcmd*で指定された値に変更します。

*icrcmd*の指定形式を次に示します。

- *icrcmd* = ICR_CLRINT

  割り込み制御レジスタの割り込み要求フラグを“0”に変更します。

- *icrcmd* = ICR_CLRMSK

  割り込み制御レジスタの割り込みマスク・フラグを“0”に変更します。

- *icrcmd* = ICR_SETMSK

  割り込み制御レジスタの割り込みマスク・フラグを“1”に変更します。

- *icrcmd* =　（ICR_CHGLVL|数値）

  割り込み制御レジスタの割り込み優先順位を“数値”で指定された値に変更します。

  なお，数値は“0”がレベル0に，“7”がレベル7に対応しています。

注意　割り込み要求番号*eintno*には，（対象割り込み要求番号の例外コード－0x80）/0x10で算出される値を指定します。

戻　り　値

E_OK	0	正常終了

Refer Interrupt Control Register Status

ref_icr

タスク／ハンドラ

概　　要

割り込み制御レジスタの内容を獲得する。

C言語形式

```
ER  ref_icr（UB ＊p_regptn, UINT eintno）；
```

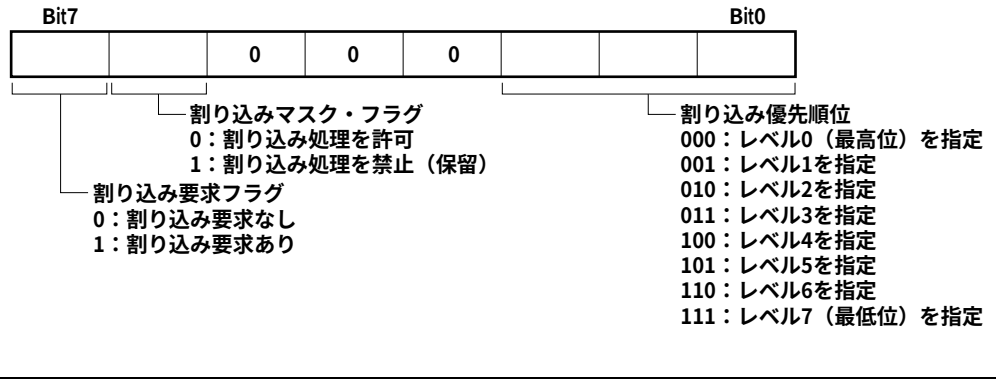
パラメータ

I/O	パラメータ	説　　明
O	UB　＊*p_regptn*；	割り込み制御レジスタの内容を格納する領域のアドレス
I	UINT　*eintno*；	割り込み要求番号

説　　明

*eintno*で指定された割り込み制御レジスタの内容を*p_regptn*で指定される領域に格納します。

獲得した割り込み制御レジスタの内容を次に示します。



注意　割り込み要求番号*eintno*には，（対象割り込み要求番号の例外コード－0x80）/0x10で算出される値を指定します。

戻　り　値

E_OK	0	正常終了

10. 5. 5　メモリ・プール管理機能システム・コール

この項では，メモリ・ブロックの割り当てを行うシステム・コールのグループ（メモリ・プール管理機能システム・コール）について説明しています。

表10－7に，メモリ・プール管理機能システム・コールの一覧を示します。

表10－7　メモリ・プール管理機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
get_blf	固定長メモリ・ブロックを獲得する	○
pget_blf	固定長メモリ・ブロックを獲得する（ポーリング）	×
tget_blf	固定長メモリ・ブロックを獲得する（タイムアウトあり）	○
rel_blf	固定長メモリ・ブロックを返却する	○
ref_mpf	固定長メモリ・プール情報を獲得する	×
get_blk	可変長メモリ・ブロックを獲得する	○
pget_blk	可変長メモリ・ブロックを獲得する（ポーリング）	×
tget_blk	可変長メモリ・ブロックを獲得する（タイムアウトあり）	○
rel_blk	可変長メモリ・ブロックを返却する	○
ref_mpl	可変長メモリ・プール情報を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Get Fixed-size Memory Block

get_blf

タスク

概　　要

固定長メモリ・ブロックを獲得する。

C言語形式

```
ER  get_blf（VP *p_blf, ID mpfid）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　*p_blf；	固定長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　*mpfid；	固定長メモリ・プールのID番号

説　　明

*mpfid*で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを*p_blf*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象固定長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（固定長メモリ・ブロック待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック待ち状態の解除は，rel_blf, rel_waiシステム・コールが発行されたときに行われ，固定長メモリ・ブロック待ち状態からready状態へと遷移します。

注意　RX850では，メールボックスを介して，タスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このためRX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。

戻　り　値

E_OK	0	正常終了
E_RLWAI	－86	rel_waiシステム・コールにより，固定長メモリ・ブロック待ち状態を強制的に解除された

Poll and Get Fixed-size Memory Block

pget_blf

タスク／ハンドラ

概　　要

固定長メモリ・ブロックを獲得する（ポーリング）。

C言語形式

```
ER  pget_blf（VP *p_blf, ID mpfid）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　*p_blf；	固定長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　*mpfid；	固定長メモリ・プールのID番号

説　　明

*mpfid*で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを*p_blf*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

注意　RX850では，メールボックスを介して，タスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このため，RX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	対象固定長メモリ・プールに空き領域が存在しない

Get Fixed-size Memory Block with Timeout

tget_blf

タスク

概　　要

固定長メモリ・ブロックを獲得する（タイムアウトあり）。

C言語形式

```
ER  tget_blf（VP *p_blf, ID mpfid, TMO tmout）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　*p_blf；	固定長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　*mpfid*；	固定長メモリ・プールのID番号
I	TMO　*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　：即時リターン TMO_FEVR（－1）　：永久待ち 数値　：待ち時間

説　　明

*mpfid*で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを*p_blf*で指定される領域に格納します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象固定長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（固定長メモリ・ブロック待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，または，rel_blf, rel_waiシステム・コールが発行されたときに行われ，固定長メモリ・ブロック待ち状態からready状態へと遷移します。

注意　RX850では，メールボックスを介して，タスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このためRX850では，固定長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，固定長メモリ・ブロック待ち状態を強制的に解除された

Release Fixed-size Memory Block

rel_blf

タスク／ハンドラ

概　　要

固定長メモリ・ブロックを返却する。

C言語形式

```
ER  rel_blf（ID mpfid, VP blf）；
```

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番号
I	VP　*blf*；	固定長メモリ・ブロックの先頭アドレス

説　　明

*blf*で指定された固定長メモリ・ブロックを*mpfid*で指定された固定長メモリ・プールに返却します。

ただし，このシステム・コールを発行したとき，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却処理は行わず，該当するタスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。

これにより，該当するタスクは待ちキューから外れ，wait状態（固定長メモリ・ブロック待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意1. RX850では，固定長メモリ・ブロックを返却するとき，メモリ・クリアを行っていません。したがって，返却した固定長メモリ・ブロックの内容は不定となります。

2. 固定長メモリ・ブロックの返却は，get_blf, pget_blf, tget_blfシステム・コールを発行したときに指定した固定長メモリ・プールと同一でなければなりません。

戻　り　値

E_OK	0	正常終了

ref_mpf

Refer Fixed-size Memory Pool Status

タスク／ハンドラ

概　　要

固定長メモリ・プール情報を獲得する。

C言語形式

```
ER  ref_mpf（T_RMPF ＊pk_rmpf, ID mpfid）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RMPF　＊*pk_rmpf*；	固定長メモリ・プール情報を格納するパケットの先頭アドレス
I	ID　　*mpfid*；	固定長メモリ・プールのID番号

固定長メモリ・プール情報T_RMPFの構造

```
typedef struct  t_rmpf {
        VP      exinf；   /＊ 拡張情報            ＊/
        BOOL_ID wtsk；    /＊ 待ちタスクの有無      ＊/
        INT     frbcnt；  /＊ 空き領域のブロック数 ＊/
} T_RMPF；
```

説　　明

***mpfid*で指定された固定長メモリ・プールのメモリ・プール情報（拡張情報，待ちタスクの有無など）を*pk_rmpf*で指定されるパケットに格納します。**

メモリ・プール情報の詳細を次に示します。

exinf　…　**拡張情報**

wtsk　…　**待ちタスクの有無**

FALSE（0）　：**待ちタスクなし**

数値　　：**待ちキューの先頭タスクのID番号**

frbcnt　…　**空き領域のブロック数**

戻 り 値

E_OK	0	**正常終了**

Get Variable-size Memory Block

get_blk

タスク

概　　要

可変長メモリ・ブロックを獲得する。

C言語形式

```
ER  get_blk（VP *p_blk, ID mplid, INT blksz）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　*p_blk；	可変長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　*mplid*；	可変長メモリ・プールのID番号
I	INT　*blksz*；	可変長メモリ・ブロック・サイズ（バイト数）

説　　明

***mplid*で指定された可変長メモリ・プールから*blksz*[注]で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを*p_blk*で指定される領域に格納します。**

可変長メモリ・ブロックには4バイトの管理領域が付加されるため，獲得できる可変長メモリ・ブロックのサイズは可変長メモリ・プールのサイズよりも小さくなります。

このシステム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象可変長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（可変長メモリ・ブロック待ち状態）へと遷移させます。

このシステム・コールを発行したとき，待ちキューにタスクがつながっていた場合には，空き可変長メモリ・ブロックのサイズに関わらず無条件に可変長メモリ・ブロック待ち状態になります。すなわち，タスクの待ちキューのキューイング順はFIFOだけで，優先度順や要求ブロック・サイズ順でのキューイングは行われません。

なお，可変長メモリ・ブロック待ち状態の解除は，rel_blk, rel_waiシステム・コールが発行されたときに行われ，可変長メモリ・ブロック待ち状態からready状態へと遷移します。

注　可変長メモリ・ブロックの獲得は4バイト単位で行われます。したがって，*blksz*には4の倍数を設定してください。それ以外の値を設定しても4バイト単位で可変長メモリ・ブロックを獲得するため，実際に獲得する可変長メモリ・ブロック数は*blksz*設定値と異なってしまいます。

注意1. **RX850では，メールボックスを介してタスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このため，RX850では，可変長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。**

2. **ディスパッチ禁止状態およびハンドラからこのシステム・コールを発行した場合，動作の保証はありません。**

戻　り　値

E_OK	0	**正常終了**
E_RLWAI	－86	rel_wai**システム・コールにより，可変長メモリ・ブロック待ち状態を強制的に解除された**

Poll and Get Variable-size Memory Block

pget_blk

タスク／ハンドラ

概　　要

可変長メモリ・ブロックを獲得する（ポーリング）。

C言語形式

```
ER  pget_blk（VP  *p_blk, ID mplid, INT blksz）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　*p_blk；	可変長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　mplid；	可変長メモリ・プールのID番号
I	INT　blksz；	可変長メモリ・ブロック・サイズ（バイト数）

説　　明

*mplid*で指定された可変長メモリ・プールから*blksz*[注]で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを*p_blk*で指定される領域に格納します。

可変長メモリ・ブロックには4バイトの管理領域が付加されるため，獲得できる可変長メモリ・ブロックのサイズは可変長メモリ・プールのサイズよりも小さくなります。

このシステム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，戻り値としてE_TMOUTが返されます。

このシステム・コールを発行したとき，対象可変長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，無条件にポーリング失敗となります。可変長メモリ・プールに必要とするサイズの空きブロックがある場合でも，より大きいサイズの可変長メモリ・ブロックの獲得を待っているタスクが存在すると，そのタスクによる可変長メモリ・ブロックの獲得が優先されます。

注　可変長メモリ・ブロックの獲得は4バイト単位で行われます。したがって，*blksz*には，4の倍数を設定してください。それ以外の値を設定しても4バイト単位で可変長メモリ・ブロックを獲得するため，実際に獲得する可変長メモリ・ブロック数は*blksz*設定値と異なってしまいます。

注意　RX850では，メールボックスを介して，タスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，メッセージ用待ちキューにキューイングするときのリンク領域として使用されます。このため，RX850では，可変長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。

戻　り　値

E_OK	0	正常終了
E_ TMOUT	－85	対象可変長メモリ・プールに空き領域が存在しない

Get Variable-size Memory Block with Timeout

tget_blk

タスク

概　　要

可変長メモリ・ブロックを獲得する（タイムアウトあり）。

C言語形式

```
ER tget_blk（VP ＊p_blk, ID mplid, INT blksz, TMO tmout）；
```

パラメータ

I/O	パラメータ	説　　明
O	VP　＊*p_blk*；	可変長メモリ・ブロックの先頭アドレスを格納する領域のアドレス
I	ID　*mplid*；	可変長メモリ・プールのID番号
I	INT　*blksz*；	可変長メモリ・ブロック・サイズ（バイト数）
I	TMO　*tmout*；	待ち時間（単位：クロック割り込み周期） TMO_POL（0）　：即時リターン TMO_FEVR（－1）：永久待ち 数値　：待ち時間

説　　明

*mplid*で指定された可変長メモリ・プールから*blksz*[注]で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを*p_blk*で指定される領域に格納します。

可変長メモリ・ブロックには4バイトの管理領域が付加されるため，獲得できる可変長メモリ・ブロックのサイズは可変長メモリ・プールのサイズより小さくなります。

このシステム・コールを発行したとき，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（空き領域が存在しなかった）場合には，自タスクを対象可変長メモリ・プールの待ちキューの最後尾にキューイングしたのち，run状態からwait状態（可変長メモリ・ブロック待ち状態）へと遷移させます。

タスクの待ちキューのキューイング順はFIFO順に行われます。

このシステム・コールを発行したとき，待ちキューにタスクがつながっていた場合には，空き可変長メモリ・ブロックのサイズに関わらず無条件に可変長メモリ・ブロック待ち状態になります。

なお，可変長メモリ・ブロック待ち状態の解除は，*tmout*で指定された待ち時間が経過したとき，またはrel_blk, rel_waiシステム・コールが発行されたときに行われ，可変長メモリ・ブロック待ち状態からready状態へと遷移します。

注　可変長メモリ・ブロックの獲得は4バイト単位で行われます。したがって，*blksz*には，4の倍数を設定してください。それ以外の値を設定しても4バイト単位で可変長メモリ・ブロックを獲得するため，実際に獲得する可変長メモリ・ブロック数は*blksz*設定値と異なってしまいます。

注意　RX850では，メールボックスを介して，タスク間でやり取りされるメッセージ用の領域として，メモリ・ブロックを使用することを推奨しています。また，メッセージの先頭4バイトは，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されます。このため，RX850では，可変長メモリ・ブロックを獲得するとき，先頭4バイトのみをクリアしています。

戻　り　値

E_OK	0	正常終了
E_TMOUT	－85	待ち時間が経過した
E_RLWAI	－86	rel_waiシステム・コールにより，可変長メモリ・ブロック待ち状態を強制的に解除された

Release Variable-size Memory Block

rel_blk

タスク／ハンドラ

概　　要

可変長メモリ・ブロックを返却する。

C言語形式

```
ER rel_blk（ID mplid, VP blk）；
```

パラメータ

I/O	パラメータ	説　明
I	ID　*mplid*；	可変長メモリ・プールのID番号
I	VP　*blk*；	可変長メモリ・ブロックの先頭アドレス

説　　明

*blk*で指定された可変長メモリ・ブロックを*mplid*で指定された可変長メモリ・プールに返却します。可変長メモリ・ブロックの返却によって待ちタスクの要求サイズを満足する連続した空き領域が可変長メモリ・プールに確保できた場合，待ちタスクの待ちが解除され可変長メモリ・ブロックを獲得します。

このシステム・コールを発行したとき，対象可変長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，可変長メモリ・ブロックの返却処理は行わず，該当タスク（待ちキューの先頭タスク）に可変長メモリ・ブロックを渡します。

これにより，該当タスクは，待ちキューから外れ，wait状態（可変長メモリ・ブロック待ち状態）からready状態へ，またはwait_suspend状態からsuspend状態へと遷移します。

注意1. RX850では，可変長メモリ・ブロックを返却するとき，メモリ・クリアを行っていません。したがって，返却した可変長メモリ・ブロックの内容は不定となります。

2. 可変長メモリ・ブロックの返却は，get_blk, pget_blk, tget_blkシステム・コールを発行したときに指定した可変長メモリ・プールと同一でなければなりません。

戻　り　値

E_OK	0	正常終了

ref_mpl

Refer Variable-size Memory Pool Status

タスク／ハンドラ

概　　要

可変長メモリ・プール情報を獲得する。

C言語形式

```
ER ref_mpl（T_RMPL ＊pk_rmpl, ID mplid）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RMPL　＊*pk_rmpl*；	可変長メモリ・プール情報を格納するパケットの先頭アドレス
I	ID　　*mplid*；	可変長メモリ・プールのID番号

可変長メモリ・プール情報T_RMPLの構造

```
typedef struct  t_rmpl {
        VP      exinf；    /＊ 拡張情報                        ＊/
        BOOL_ID wtsk；     /＊ 待ちタスクの有無                  ＊/
        INT     frsz；     /＊ 空き領域の合計サイズ（バイト数）  ＊/
        INT     maxsz；    /＊ 最大の空き領域のサイズ（バイト数）＊/
} T_RMPL；
```

説　　明

*mplid*で指定された可変長メモリ・プールの可変長メモリ・プール情報（拡張情報，待ちタスクの有無など）を*pk_rmpl*で指定されるパケットに格納します。

可変長メモリ・プール情報の詳細を次に示します。

- exinf　…　拡張情報
- wtsk　…　待ちタスクの有無
  - FALSE（0）　：待ちタスクなし
  - 数値　　：待ちキューの先頭タスクのID番号
- frsz　…　空き領域の合計サイズ（バイト数）
- maxsz　…　最大の空き領域のサイズ（バイト数）

戻 り 値

E_OK	0	正常終了

10. 5. 6　時間管理機能システム・コール

この項では，時間に依存した処理を行うシステム・コールのグループ（時間管理機能システム・コール）について説明しています。

表10－8に，時間管理機能システム・コールの一覧を示します。

表10－8　時間管理機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
dly_tsk	自タスクを時間経過待ち状態へ移行する	○
act_cyc	周期起動ハンドラの活性状態を制御する	×
ref_cyc	周期起動ハンドラ情報を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Delay Task

dly_tsk

タスク

概　　要

自タスクを時間経過待ち状態へ移行する。

C言語形式

```
ER dly_tsk（DLYTIME dlytim）；
```

パラメータ

I/O	パラメータ	説　　明
I	DLYTIME　*dlytim*；	遅延時間（単位：クロック割り込み周期）

説　　明

自タスクを*dlytim*で指定された遅延時間だけ，run状態からwait状態（時間経過待ち状態）へと遷移させます。

なお，時間経過待ち状態の解除は，*dlytim*で指定された遅延時間が経過したとき，またはrel_waiシステム・コールが発行されたときに行われ，時間経過待ち状態からready状態へと遷移します。

注意　時間経過待ち状態の解除は，wup_tsk, ret_wup**システム・コールでは行われません。**

戻　り　値

E_OK	0	正常終了
E_RLWAI	－86	rel_waiシステム・コールにより，時間経過待ち状態を強制的に解除された

Activate Cyclic Handler

act_cyc

タスク／ハンドラ

概　　要

周期起動ハンドラの活性状態を制御する。

C言語形式

```
ER act_cyc（HNO cycno, UINT cycact）；
```

パラメータ

I/O	パラメータ	説　　明
I	HNO　*cycno*；	**周期起動ハンドラの指定番号**
I	UINT　*cycact*；	**活性状態／周期カウンタ／キューイングの指定** TCY_OFF　（0）：**オフ状態** TCY_ON　（1）：**オン状態** TCY_INI　（2）：**周期カウンタを初期化する** TCY_ULNK　（4）：**タイマ・キューから外す**

説　　明

***cycno*で指定された周期起動ハンドラの活性状態を*cycact*で指定された状態に変更します。**

***cycact*の指定形式を次に示します。**

- *cycact* = TCY_OFF

  対象周期起動ハンドラの活性状態をオフ状態に変更します。

  これにより，起動周期に達しても，対象周期起動ハンドラは起動されません。

  注意　RX850**では，周期起動ハンドラがタイマ・キューにキューイングされている間は，活性状態がオフ状態でも，周期カウンタのカウント処理が行われます。**

- *cycact* = TCY_ON

  対象周期起動ハンドラの活性状態をオン状態に変更します。

  これにより，起動周期に達したときには，対象周期起動ハンドラが起動されます。

- *cycact* = TCY_INI

  対象周期起動ハンドラをタイマ・キューにキューイングしたのち，周期カウンタを初期化します。

  注意　このシステム・コールを発行したとき，対象周期起動ハンドラがタイマ・キューにキューイングされていた場合には，タイマ・キューのキュー操作は行わず，周期カウンタの初期化処理のみを行います。

- *cycact* =　（TCY_ON｜TCY_INI）

  対象周期起動ハンドラの活性状態をオン状態に変更し，タイマ・キューにキューイングしたのち，周期カウンタを初期化します。

これにより，起動周期に達したときには，対象周期起動ハンドラが起動されます。

- *cycact* = TCY_ULNK

対象周期起動ハンドラの活性状態をオフ状態に変更したのち，タイマ・キューから外します。
これにより，起動周期に達しても，対象周期起動ハンドラは起動されません。

注意　このシステム・コールを発行したとき，対象周期起動ハンドラの活性状態がオフ状態であった場合には，活性状態の状態操作は行わず，タイマ・キューのキュー操作のみを行います。

戻　り　値

E_OK	0	正常終了

Refer Cyclic Handler Status

ref_cyc

タスク／ハンドラ

概　　要

周期起動ハンドラ情報を獲得する。

C言語形式

```
ER ref_cyc（T_RCYC ＊pk_rcyc, HNO cycno）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RCYC　＊pk_rcyc；	周期起動ハンドラ情報を格納するパケットの先頭アドレス
I	HNO　　cycno；	周期起動ハンドラの指定番号

周期起動ハンドラ情報T_RCYCの構造

```
typedef struct  t_rcyc {
        VP      exinf；  /＊ 拡張情報      ＊/
        CYCTIME lfttim； /＊ 残り時間      ＊/
        UINT    cycact； /＊ 現在の活性状態 ＊/
} T_RCYC；
```

説　　明

*cycno*で指定された周期起動ハンドラの周期起動ハンドラ情報（拡張情報，残り時間など）を*pk_rcyc*で指定されるパケットに格納します。

周期起動ハンドラ情報の詳細を次に示します。

exinf　…　拡張情報

lfttim　…　次に周期起動ハンドラを起動するまでの残り時間
（単位：クロック割り込み周期）

cycact　…　現在の活性状態
- TCY_OFF　（0）：活性状態はオフ状態
- TCY_ON　（1）：活性状態はオン状態
- TCY_ULNK　（4）：周期起動ハンドラはタイマ・キューから外れている

注意　cycactの値がTCY_ULNKの場合，lfttimの内容は不定となります。

戻 り 値

E_OK	0	正常終了

10. 5. 7　システム管理機能システム・コール

この項では，システムに依存した処理を行うシステム・コールのグループ（システム管理機能システム・コール）について説明しています。

表10－9に，システム管理機能システム・コールの一覧を示します。

表10－9　システム管理機能システム・コールの一覧

システム・コール	機　　能	schdl[注]
get_ver	RX850のバージョン情報を獲得する	×
ref_sys	システム情報を獲得する	×

注　schdlは，スケジューラの駆動を行うかどうかを示しています。
ただし，スケジューラを駆動した結果，他タスクに制御を移すかどうかは，そのときの状況により異なります。

○：スケジューラの駆動を行う。

×：スケジューラの駆動を行わない。

Get Version Information

get_ver

タスク／ハンドラ

概　　要

RX850のバージョン情報を獲得する。

C言語形式

```
ER get_ver（T_VER ＊pk_ver）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_VER　＊pk_ver；	バージョン情報を格納するパケットの先頭アドレス

バージョン情報T_VERの構造

```
typedef struct t_ver {
        UH      maker；     /＊ OS製造メーカ          ＊/
        UH      id；        /＊ OS形式番号            ＊/
        UH      spver；     /＊ 仕様書バージョン番号  ＊/
        UH      prver；     /＊ OS製品バージョン番号  ＊/
        UH      prno[4]；   /＊ 製品番号／製品管理情報 ＊/
        UH      cpu；       /＊ CPU情報               ＊/
        UH      var；       /＊ バリエーション記述子  ＊/
} T_VER；
```

説　　明

RX850のバージョン情報（OS製造メーカ，OS形式番号など）を*pk_ver*で指定されるパケットに格納します。

バージョン情報の詳細を次に示します。

- maker … OS製造メーカ
  - H'000d：NEC
- id … OS形式番号
  - H'0000：未使用
- spver … 仕様書バージョン番号
  - H'5302：μITRON3.0 Ver3.02
- prver … OS製品バージョン番号
  - H'03xx：RX850 Ver3.xx
- prno[4] … 製品番号／製品管理情報
  - 不定　：出荷製品のシリアル番号（出荷製品ごとに異なる）
- cpu … CPU情報
  - H'0d33：V850ファミリ

var　　… **バリエーション記述子**

H'8000 ：μ ITRON**レベル**S，**シングル・プロセッサ用，仮想記録サポートなし，**MMU**サポートなし，ファイル・サポートなし**

戻　り　値

E_OK	0	**正常終了**

ref_sys

Refer System Status

タスク／ハンドラ

概　　要

システム情報を獲得する。

C言語形式

```
ER ref_sys（T_RSYS ＊pk_rsys）；
```

パラメータ

I/O	パラメータ	説　　明
O	T_RSYS　＊*pk_rsys*；	システム情報を格納するパケットの先頭アドレス

システム情報T_RSYSの構造

```
typedef struct t_rsys {
        INT     sysstat； /＊ システムの状態     ＊/
} T_RSYS；
```

説　　明

動的に変化するシステム情報（システムの状態）の現在値を*pk_rsys*で指定されるパケットに格納します。

システム情報の詳細を次に示します。

sysstat … **システムの状態**

- TTS_TSK （0）：**タスクの処理を実行中。なお，ディスパッチ処理は許可状態。**
- TTS_DDSP（1）：**タスクの処理を実行中。なお，ディスパッチ処理は禁止状態。**
- TTS_LOC （3）：**タスクの処理を実行中。なお，マスカブル割り込みの受け付け，およびディスパッチ処理は禁止状態。**
- TTS_INDP（4）：**ハンドラ（割り込みハンドラ，周期起動ハンドラ）の処理を実行中。**

戻 り 値

E_OK	0	**正常終了**

〔メ　モ〕

付録A　プログラミングのために

この章では，CA850またはCCV850を使用したときの処理プログラムの記述方法について説明しています。

A. 1　概　　要

RX850では，処理プログラムを用途別に次のように区別しています。

なお，これらの処理プログラムは，一般的あるいはRX850を使用するうえでの約束ごとなどにより，それぞれに基本型があります。

(1) タスク

RX850の管理下で実行可能な処理プログラムの最小単位です。

(2) 直接起動割り込みハンドラ

割り込みが発生したとき，RX850を介在させることなく起動される割り込み処理専用ルーチンであり，タスクとは独立したものとして扱われます。このため，割り込みが発生したときには，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，直接起動割り込みハンドラに制御が移ります。

なお，直接起動割り込みハンドラは，RX850を介在させることなく起動される割り込み処理専用ルーチンであるため，ハードウエアの限界に近い高速な応答性が期待される処理プログラムです。

(3) 間接起動割り込みハンドラ

割り込みが発生したとき，RX850による割り込み前処理（レジスタの退避処理，スタックの切り替え処理など）を行わせたのちに起動される割り込み処理専用ルーチンであり，タスクとは独立したものとして扱われます。このため，割り込みが発生したときには，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，間接起動割り込みハンドラに制御が移ります。

なお，間接起動割り込みハンドラは，割り込みが発生したとき，RX850による割り込み前処理を行わせたのちに起動される割り込み処理専用ルーチンであるため，直接起動割り込みハンドラに比べて応答性の面では劣りますが，ハンドラ内での処理が簡素化されるといった利点を持った処理プログラムです。

(4) 周期起動ハンドラ

一定の起動時間に達したとき，ただちに起動される周期処理専用ルーチンであり，タスクとは独立したものとして扱われます。このため，起動時間に達したときには，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，周期起動ハンドラに制御が移ります。

なお，周期起動ハンドラは，ユーザが記述する周期的な処理プログラムの中で，実行開始までのオーバヘッドが最も小さな処理プログラムです。

A. 2　キー・ワード

コンフィギュレータでは，次に示す文字列をキー・ワードとして予約しています。したがって，これらの文字列を他の用途に使用することは禁止されています。

auto	clkhdr	cyc	di	ei
flg	flg1	inthdr	intstk	maxpri
mbx	mpf	no_use	no_wait	pool0
pool1	RX850	rxsers	sem	ser_def
sit_def	TA_MFIFO	TA_MPRI	TCY_ULNK	TCY_OFF
TCY_ON	tsk	tskgrp	TTS_DMT	TTS_RDY
V310				

A. 3　予 約 語

RX850では，次に示す文字列を外部シンボルとして予約しています。したがって，これらの文字列を他の用途に使用することは禁止されています。

_CYC＊	_ID＊	inthdrH	inthdrL	Pool0＊
Pool1＊	RX850＊	Sit＊	SysIntEnt	Timer_Handler
_txcb＊				

備考　＊は，1文字以上の文字列を表しています。

A. 4　タスクの記述方法

A. 4. 1　CA850対応版の場合

タスクをC言語で記述する場合，プラグマ指令による関数宣言を行ったのち，INT型の引き数を1つ持ったvoid型の関数として記述します。

なお，引き数（*stacd*）には，sta_tskシステム・コール発行時に指定された起動コードが設定されます。

図A－1に，CA850を使用したときのタスクの基本型（C言語）を示します。

図A－1　CA850を使用したときのタスクの基本型（C言語）

```
#include  <stdrx850.h>


#pragma  rtos_task   func_task
void
func_task(INT  stacd )
{
            /* タスク  func_task  の本体処理  */
            ..................................
            ..................................
            ..................................

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

注意　プラグマ指令による関数宣言についての詳細は，CA830，CA850　Cコンパイラ・パッケージ　ユーザーズ・マニュアル　C言語編を参照してください。

また，タスクをアセンブリ言語で記述する場合は，CA850の関数呼び出し規約に従った関数として記述します。

なお，引き数（r6レジスタ）には，sta_tskシステム・コール発行時に指定された起動コードが設定されます。

図A－2に，CA850を使用したときのタスクの基本型（アセンブリ言語）を示します。

図A－2　CA850を使用したときのタスクの基本型（アセンブリ言語）

```
#include   <stdrx850.h>


        .text
        .align      4
        .globl     _func_task
_func_task:
        /* タスク func_task の本体処理 */
        ..................................
        ..................................
        ..................................

        /* タスク func_task の終了 */
        jr          _ext_tsk
```

注意1. タスクをアセンブリ言語で記述する場合，ファイル名の拡張子には，“.c”を指定してください。

2. タスクをアセンブリ言語で記述する場合，ファイルの先頭で

```
#include 〈stdrx850.h〉
```

を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CA850の前処理（フロント・エンド）を実行する必要があります。

A. 4. 2　CCV850対応版の場合

タスクをC言語で記述する場合，INT型の引き数を１つ持ったvoid型の関数として記述します。

なお，引き数（*stacd*）には，sta_tskシステム・コール発行時に指定された起動コードが設定されます。

図A－3に，CCV850を使用したときのタスクの基本型（C言語）を示します。

図A－3　CCV850を使用したときのタスクの基本型（C言語）

```
#include  <stdrx850.h>


void
func_task(INT  stacd )
{
            /* タスク func_task の本体処理 */
            ..................................
            ..................................
            ..................................

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

また，タスクをアセンブリ言語で記述する場合は，CCV850の関数呼び出し規約に従った関数として記述します。

なお，引き数（r6レジスタ）には，sta_tskシステム・コール発行時に指定された起動コードが設定されます。

図A－4に，CCV850を使用したときのタスクの基本型（アセンブリ言語）を示します。

図A－4　CCV850を使用したときのタスクの基本型（アセンブリ言語）

```
#include   <stdrx850.h>


        .text
        .align      4
        .globl     _func_task
_func_task：
        /＊ タスク func_task の本体処理 ＊/
        ..................................
        ..................................
        ..................................

        /＊ タスク func_task の終了 ＊/
        jr          _ext_tsk
```

注意1. タスクをアセンブリ言語で記述する場合，ファイル名の拡張子には，“.850”を指定してください。

2. タスクをアセンブリ言語で記述する場合，ファイルの先頭で

   #include 〈stdrx850.h〉

   を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CCV850の前処理（フロント・エンド）を実行する必要があります。

A. 5　直接起動割り込みハンドラの記述方法

直接起動割り込みハンドラは，C言語のみで記述することはできません。アセンブリ言語を使用して記述します。

直接起動割り込みハンドラでは，その処理の前でレジスタの退避を，その処理の後でレジスタの復帰を行う必要があります。RX850では，レジスタの退避／復帰処理を記述したマクロを用意しており，アセンブリ言語でのハンドラ記述におけるユーザの負担を軽減しています。

また，ハンドラの記述方法は，その復帰方法（reti，ret_int，ret_wup）によって異なります。各復帰方法に対応した記述方法および注意事項を次に示します。

A. 5. 1　retiで復帰する場合（CA850対応版）

直接起動割り込みハンドラからretiで復帰する場合は，図A－5のように記述します。

図A－5　直接起動割り込みハンドラからretiで復帰する場合の記述例（CA850）

```
#include  “stdrx850.inc”

        .section“割り込み要因名”
        jr      _inthdr

        .text
.align    4
        .globl  _inthdr
_inthdr：
        RTOS_IntEntry

        ハンドラ本体

        RTOS_IntExit
```

RTOS_IntEntryではRX850に対してハンドラ処理の開始を通知します。RTOS_IntExitはRX850に対してハンドラ処理の終了を通知し，reti命令の発行を行います。これらのマクロにはレジスタの退避／復帰処理は含まれていません。レジスタの退避／復帰は“ハンドラ本体”で行ってください。

また，この記述方法ではスタックの切り替えを行いません。割り込まれたタスクのスタックが使用されます。

なお，システム・コールの発行はできません。システム・コールを発行する割り込みハンドラやタイマ・ハンドラがネストすることもできません。この記述は，I/Oの操作だけを行ってただちに復帰するような，単純な処理を行う場合だけに限られます。

A. 5. 2　retiで復帰する場合（CCV850対応版）

直接起動割り込みハンドラからretiで復帰する場合は，図A－6のように記述します。

図A－6　直接起動割り込みハンドラからretiで復帰する場合の記述例（CCV850）

```
#include  <stdrx850.h>

        .org    0x00000150      -- 割り込みエントリ・アドレス
        jr      _inthdr

        .text
.align    4
        .globl  _inthdr
_inthdr：
        RTOS_IntEntry

        ハンドラ本体

        RTOS_IntExit
```

RTOS_InEntryでは，RX850に対してハンドラ処理の開始を通知します。RTOS_IntExitはRX850に対してハンドラ処理の終了を通知し，reti命令の発行を行います。これらのマクロにはレジスタの退避／復帰処理は含まれていません。レジスタの退避／復帰は“ ハンドラ本体 ”で行ってください。

また，この記述方法ではスタックの切り替えを行いません。割り込まれたタスクのスタックが使用されます。

なお，システム・コールの発行はできません。システム・コールを発行する割り込みハンドラやタイマ・ハンドラがネストすることもできません。この記述はI/Oの操作だけを行ってただちに復帰するような，単純な処理を行う場合だけに限られます。

注意　コンパイル時には，コンパイル・オプションとして“ -D_ _asm_ _”が必要です。

A. 5. 3　ret_int, ret_wupで復帰する場合（CA850対応版）

ret_int，ret_wupでハンドラ復帰する場合は，図A－7のように記述します。

図A－7　直接起動割り込みハンドラからret_int, ret_wupで復帰する場合の記述例（CA850）

```
.set      Reg26, 1          ーー　レジスタ 26 本モード使用時
.set      Reg22, 1          ーー　レジスタ 22 本モード使用時
                            ーー　レジスタ 32 本モード使用時は不要
#include  “stdrx850.inc”

        .section "INTP00"   ーー　割り込み要因名を指定
        jr      _inthdr

        .text
        .globl  _inthdr

        .align 4
_inthdr：
        RTOS_IntEntry
        RTOS_IntPrologue

        .extern _inthdr_body
        jarl    _inthdr_body, lp

        mov   r10, r6

        RTOS_IntEpilogue

        jr       _ret_wup
```

```
#include  <stdrx850.h>

ID inthdr_body(void)
{
            ハンドラ本体

            return tskid; /＊起床させるタスクのID番号＊/
}
```

RTOS_IntEntryでは，RX850に対しハンドラ処理開始の通知を行います。RTOS_IntPrologueでは，テンポラリ・レジスタおよびlpの退避，スタックの切り替えなどを行います。

そのあと，上記以外のレジスタ（r20-r30）の退避を行い，ハンドラ本体部へと移ります。図A－7では，ハンドラ本体であるCの関数“inthdr_body”を呼び出しています。

ハンドラ本体部（RTOS_IntPrologue以降）では，**5．3　直接起動割り込みハンドラ**に示すシステム・コールが発行可能です。割り込みを許可することも可能です。

ハンドラの本体処理が終了したのち，ユーザが退避したレジスタの復帰と割り込みハンドラからの復帰を行い（図A－7では同様に不要），ret_wupでハンドラを終了する場合は，レジスタr6に起床させるタスクID番号をセットします。

図A－7では，inthdr_bodyから復帰するときに戻り値としてタスクのIDを返しており，その値がr10にコピーされます。そのために，r10の値をr6にコピーしています。なお，ret_intで復帰するときはこの作業は必要ありません。

以上の処理が終わったら，マクロRTOS_IntEpilogueを記述し，“jr ret_int”または“jr ret_wup”でハンドラを終了します。なお，RTOS_IntEpilogueと“jr ret_int（またはret_wup）”の間には，処理を記述しないでください。

注意　.section疑似命令についての詳細は，CA850　Cコンパイラ・パッケージ　ユーザーズ・マニュアル　アセンブリ言語編を参照してください。

A. 5. 4　ret_int, ret_wupで復帰する場合（CCV850対応版）

ret_int，ret_wupで割り込みハンドラを終了する場合，図A－8のように記述します。

図A－8　直接起動割り込みハンドラからret_int, ret_wupで復帰する場合の記述例（CCV850）

```
#include  <stdrx850.h>

        .org    0x00000150    ーー　割り込みエントリ・アドレス
        jr      _inthdr

        .text
        .globl  _inthdr

        .align  4
_inthdr：
        RTOS_IntEntry
        RTOS_IntProlague

        .extern _inthdr_body
        jarl    _inthdr_body, lp

        mov     r10, r6

        RTOS_IntEpilogue

        jr      _ret_wup
```

≈　≈

ハンドラ本体が記述されているファイル

```
#include  <stdrx850.h>

ID inthdr_body (void)
{
            ハンドラ本体

            return tskid; /＊起床させるタスクのID番号＊/
}
```

RTOS_InEntryでは，RX850に対してハンドラ処理の開始を通知します。RTOS_IntPrologueではテンポラリ・レジスタおよびIpの退避，スタックの切り替えなどを行います。

その後，上記以外のレジスタ（r20-r30）の待避を行い，ハンドラ本体部へと移ります。図A－8では，ハンドラの本体であるCの関数“inthdr_body”を呼び出しています。

ハンドラ本体部（RTOS_IntPrologue以降）では，**5.3　直接起動割り込みハンドラ**に示すシステム・コールが発行可能です。割り込みを許可することも可能です。

ハンドラの本体処理が終了したのち，ユーザが退避したレジスタの復帰と割り込みハンドラからの復帰を行い（図A－8では，同様に不要），ret_wupでハンドラを終了する場合はレジスタr6に起床させるタスクID番号をセットします。

図A－8では，inthdr_bodyから復帰するときに戻り値としてタスクのIDを返しており，その値がr10にコピーされます。そのために，r10の値をr6にコピーしています。なお，ret_intで復帰するときはこの作業は必要ありません。

以上の作業が終わったら，マクロRTOS_IntEpilogueを記述し，“jr ret_int”または“jr ret_wup”でハンドラを終了します。なお，RTOS_InEpilogueと“jr ret_int（またはret_wup）”の間には，処理を記述しないでください。

注意1. 割り込みが発生したときにプロセッサが処理を移すハンドラ・アドレスに対して直接起動割り込みハンドラへの分岐命令を設定する必要があります。図A－8における.org命令部分がこれに当たります。.org命令に続く値が割り込みのハンドラ・アドレスに相当します。

2. 直接起動割り込みハンドラのアセンブリ言語部分のファイルは，ファイル名の拡張子として“.850”を指定してください。

3. コンパイル時には，コンパイル・オプションとして“-D_ _asm_ _”が必要です。

A. 6　間接起動割り込みハンドラの記述方法

A. 6. 1　CA850対応版の場合

間接起動割り込みハンドラをC言語で記述する場合，引数を持たないINT型の関数として記述します。

図A－9に，CA850を使用したときの間接起動割り込みハンドラの基本型（C言語）を示します。

図A－9　CA850を使用したときの間接起動割り込みハンドラの基本型（C言語）

```
#include  <stdrx850.h>


INT
func_inthdr ( )
{
        /* 間接起動割り込みハンドラ  func_inthdr  の本体処理 */
        .....................................................
        .....................................................
        .....................................................

        /* 外部割り込みコントローラに対する処理終了通知 */
        .....................................................

        /* 間接起動割り込みハンドラ  func_inthdr  からの復帰 */
        return(0xff);
}
```

注意　間接起動割り込みハンドラは，RX850の割り込み前処理から呼び出されるサブルーチンです。このため，間接起動割り込みハンドラを記述する場合，割り込みが発生したときにプロセッサが制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要がありません。

また，間接起動割り込みハンドラをアセンブリ言語で記述する場合は，CA850の関数呼び出し規約に従った関数として記述します。

図A－10に，CA850を使用したときの間接起動割り込みハンドラの基本型（アセンブリ言語）を示します。

図A－10　CA850を使用したときの間接起動割り込みハンドラの基本型（アセンブリ言語）

```
#include  <stdrx850.h>


        .text
        .align      4
        .globl      _func_inthdr
_func_inthdr：
        /＊ 間接起動割り込みハンドラ  func_inthdr  の本体処理 ＊/
        ..................................................
        ..................................................
        ..................................................

        /＊ 外部割り込みコントローラに対する処理終了通知 ＊/
        ..................................................

        /＊ 間接起動割り込みハンドラ  func_inthdr  からの復帰 ＊/
        mov         0xff, r10
        jmp         [lp]
```

注意1. **間接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には，“.c”を指定してください。**

2. **間接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイルの先頭で**

   `#include  〈stdrx850.h〉`

   を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CA850の前処理（フロント・エンド）を実行する必要があります。

3. **間接起動割り込みハンドラは，RX850の割り込み前処理から呼び出されるサブルーチンです。このため，間接起動割り込みハンドラを記述する場合，割り込みが発生したときにプロセッサが制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要がありません。**

A. 6. 2　CCV850対応版の場合

間接起動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INT型の関数として記述します。

図A－11に，CCV850を使用したときの間接起動割り込みハンドラの基本型（C言語）を示します。

図A－11　CCV850を使用したときの間接起動割り込みハンドラの基本型（C言語）

```
#include  <stdrx850.h>


INT
func_inthdr ( )
{
        /＊ 間接起動割り込みハンドラ  func_inthdr  の本体処理 ＊/
        ..................................................
        ..................................................
        ..................................................

        /＊ 外部割り込みコントローラに対する処理終了通知 ＊/
        ..................................................

        /＊ 間接起動割り込みハンドラ  func_inthdr  からの復帰 ＊/
        return(0xff);
}
```

注意　間接起動割り込みハンドラは，RX850の割り込み前処理から呼び出されるサブルーチンです。このため，間接起動割り込みハンドラを記述する場合，割り込みが発生したときにプロセッサが制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要がありません。

また，間接起動割り込みハンドラをアセンブリ言語で記述する場合は，CCV850の関数呼び出し規約に従った関数として記述します。

図A－12に，CCV850を使用したときの間接起動割り込みハンドラの基本型（アセンブリ言語）を示します。

図A－12　CCV850を使用したときの間接起動割り込みハンドラの基本型（アセンブリ言語）

```
#include  <stdrx850.h>


        .text
        .align      4
        .globl      _func_inthdr
_func_inthdr：
        /＊ 間接起動割り込みハンドラ  func_inthdr  の本体処理 ＊/
        ..................................................
        ..................................................
        ..................................................

        /＊ 外部割り込みコントローラに対する処理終了通知 ＊/
        ..................................................

        /＊ 間接起動割り込みハンドラ  func_inthdr  からの復帰 ＊/
        mov         0xff, r10
         jmp          [lp]
```

注意1. **間接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には，“**.850**”を指定してください。**

2. **間接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイルの先頭で**

   #include 〈stdrx850.h〉

   を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CCV850**の前処理（フロント・エンド）を実行する必要があります。**

3. **間接起動割り込みハンドラは，**RX850**の割り込み前処理から呼び出されるサブルーチンです。このため，間接起動割り込みハンドラを記述する場合，割り込みが発生したときにプロセッサが制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要がありません。**

A. 7　周期起動ハンドラの記述方法

A. 7. 1　CA850対応版の場合

周期起動ハンドラをC言語で記述する場合，引数を持たないvoid型の関数として記述します。

図A－13に，CA850を使用したときの周期起動ハンドラの基本型（C言語）を示します。

図A－13　CA850を使用したときの周期起動ハンドラの基本型（C言語）

```
#include  <stdrx850.h>


void
func_cychdr( )
{
        /* 周期起動ハンドラ  func_cychdr  の本体処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 周期起動ハンドラ  func_cychdr  からの復帰 */
        return;
}
```

注意　周期起動ハンドラは，RX850**の時間管理用割り込みハンドラ（クロック・ハンドラ）から呼び出されるサブルーチンです。**

また，周期起動ハンドラをアセンブリ言語で記述する場合は，CA850の関数呼び出し規約に従った関数として記述します。

図A－14に，CA850を使用したときの周期起動ハンドラの基本型（アセンブリ言語）を示します。

図A－14　CA850を使用したときの周期起動ハンドラの基本型（アセンブリ言語）

```
#include  <stdrx850.h>


        .text
        .align      4
        .globl      _func_cychdr
_func_cychdr：
        /＊ 周期起動ハンドラ  func_cychdr  の本体処理 ＊/
        ..........................................
        ..........................................
        ..........................................

        /＊ 周期起動ハンドラ  func_cychdr  からの復帰 ＊/
         jmp        [lp]
```

注意1. 周期起動ハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には，“.c”を指定してください。

2. 周期起動ハンドラをアセンブリ言語で記述する場合，ファイルの先頭で

   #include 〈stdrx850.h〉

   を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CA850の前処理（フロント・エンド）を実行する必要があります。

3. 周期起動ハンドラは，RX850の時間管理用割り込みハンドラ（クロック・ハンドラ）から呼び出されるサブルーチンです。

A. 7. 2　CCV850対応版の場合

周期起動ハンドラをC言語で記述する場合，引き数を持たないvoid型の関数として記述します。

図A－15に，CCV850を使用したときの周期起動ハンドラの基本型（C言語）を示します。

図A－15　CCV850を使用したときの周期起動ハンドラの基本型（C言語）

```
#include  <stdrx850.h>


void
func_cychdr( )
{
        /* 周期起動ハンドラ  func_cychdr  の本体処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 周期起動ハンドラ  func_cychdr  からの復帰 */
        return;
}
```

注意　周期起動ハンドラは，RX850の時間管理用割り込みハンドラ（クロック・ハンドラ）から呼び出されるサブルーチンです。

また，周期起動ハンドラをアセンブリ言語で記述する場合は，CCV850の関数呼び出し規約に従った関数として記述します。

図A－16に，CCV850を使用したときの周期起動ハンドラの基本型（アセンブリ言語）を示します。

図A－16　CCV850を使用したときの周期起動ハンドラの基本型（アセンブリ言語）

```
#include  <stdrx850.h>


        .text
        .align      4
        .globl      _func_cychdr
_func_cychdr：
        /＊ 周期起動ハンドラ  func_cychdr  の本体処理 ＊/
        ..........................................
        ..........................................
        ..........................................

        /＊ 周期起動ハンドラ  func_cychdr  からの復帰 ＊/
        jmp         [lp]
```

注意1. **周期起動ハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には，“.850”を指定してください。**

2. **周期起動ハンドラをアセンブリ言語で記述する場合，ファイルの先頭で**

   `#include 〈stdrx850.h〉`

   を記述します。したがって，オブジェクト・ファイルへの変換を行うときには，CCV850**の前処理（フロント・エンド）を実行する必要があります。**

3. **周期起動ハンドラは，**RX850**の時間管理用割り込みハンドラ（クロック・ハンドラ）から呼び出されるサブルーチンです。**

★ A. 8　制限事項と注意事項

A. 8. 1　V850E**を使用するとき**

RX850**は**V850E（V850E/MS1，V850E/MA1，V850E/IA1**など）でも使用できますが，一部制限事項があります。**V850E**アーキテクチャには，**CALLT**というほかの**V850**ファミリにはない独自の命令が存在します。この命令では**CTPSW**と**CTPC**というレジスタを使用しますが，**RX850**ではスケジューラなどでこの**2**つのレジスタを退避・復帰する処理を行っていません。したがって，プログラム中にこの命令が使用されていると動作が保証できないため，**CALLT**命令の使用を制限します。

アセンブリ言語でタスク・ハンドラを記述する場合は，CALLT命令を使用しないでください。またC言語でタスクや割り込みハンドラを記述したファイルをコンパイルするときは，CALLT命令を使用しないようにするオプションを指定してください。コンパイル・オプションについては，各コンパイラのマニュアルを参照してください。

A. 8. 2　.sitセクションと.pool0セクションの配置について

RX850では.sitセクションと.pool0セクションを配置できるアドレスに制限があります。これらのセクションに対するロード，ストアはr0（0番地）相対1命令で行われており，配置可能な範囲は±32 Kバイトに限られます。つまりこれらのセクションが配置できる範囲は次のようになります。

- 0xffff8000～0xffffffff（ただし0xfffff000～0xffffffffは内蔵周辺I/O領域）
- 0x0～0x7fff（ただし0x0から数百バイトは割り込みハンドラ・アドレス領域）

.sitセクションはROM化が可能なため，V850ファミリの内蔵ROM領域にあたる0x0～0x7fffに配置し，.pool0セクションはRAM領域に配置する必要があるため，V850ファミリの内蔵RAM領域である0xffffe000～0xffffefffに配置することを推奨しています。

ただし，アプリケーションのリンク時に指定するアドレスとして，実際にROM, RAMが搭載されているアドレス（物理アドレス）を指定すると，範囲外となってリンク・エラーになるCPUがあります。たとえばV850E/MS1において，内蔵RAM領域の物理アドレスは0x3ffe0000ですが，この近辺のアドレスを使って.pool0セクションを配置しようとするとリンク・エラーになります。こういったケースの場合，物理アドレスのイメージ（ミラー・イメージ）を使用してリンクしてください。なお，物理アドレスのイメージについての詳細は，各マニュアルのハードウエア編を参照してください。

A. 8. 3　システム・コールの呼び出し可能な範囲について

RX850のシステム・コールは，jarl命令を使用して呼び出されます。jarl命令はアーキテクチャの仕様上22ビット・ディスプレースメント範囲内のアクセスに限られるため，呼び出し側と呼び出される側でこの範囲を越えるとアクセスができなくなります。22ビットのうち，上位1ビットは符号ビットなので，正負それぞれ21ビットが届く限界となります。つまり0x1fffff（2097151）を越えないように配置してください。

〔メ　モ〕

付録B　総合索引

B. 1　50音で始まる語句の索引

【ら行】

【わ行】

B. 2　数字，アルファベットで始まる語句の索引

【S】

【T】

【U】

【V】

【W】

【数字】

〔メ　モ〕

★

付録C　改版履歴

これまでの改版履歴を次に示します。なお，適用箇所は各版での章を示します。

版数	内　　容	適用箇所
第2版	実行環境の動作対象CPUの記述を変更	第1章　概　　説
	開発環境のハードウエア環境とソフトウエア環境の記述を変更	
	システム構築手順図のコンフィギュレータ（GUI部）を削除	
	制限事項と注意事項の記述を追加	付録A　プログラミングのために

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【営業関係お問い合わせ先】

第一販売事業部

東　京　(03)3798−6106, 6107, 6108
大　阪　(06)6945−3178, 3200, 3208, 3212
広　島　(082)242−5504
仙　台　(022)267−8740
郡　山　(024)923−5591
千　葉　(043)238−8116

第二販売事業部

東　京　(03)3798−6110, 6111, 6112
立　川　(042)526−5981, 6167
松　本　(0263)35−1662
静　岡　(054)254−4794
金　沢　(076)232−7303
松　山　(089)945−4149

第三販売事業部

東　京　(03)3798−6151, 6155, 6586, 1622, 1623, 6156
水　戸　(029)226−1702
前　橋　(027)243−6060
鳥　取　(0857)27−5313
太　田　(0276)46−4014
名古屋　(052)222−2170, 2190
福　岡　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

NECエレクトロンデバイスの情報がインターネットでご覧になれます。　URL（アドレス）　http://www.ic.nec.co.jp/

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］RX850ユーザーズ・マニュアル　基礎編

（U13430JJ2V1UM00（第2版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは

NEC販売員，特約店販売員，その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気(株) NEC エレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ