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。

お客様各位

資料中の「三菱電機」、「三菱XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って株式会社日立製作所及び三菱電機株式会社のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。
従いまして、本資料中には「三菱電機」、「三菱電機株式会社」、「三菱半導体」、「三菱XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス　テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

注:「高周波・光素子事業、パワーデバイス事業については三菱電機にて引き続き事業運営を行います。」

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

User's Manual

7900シリーズ用 リアルタイムOS

MR79 V.2.20　ユーザーズマニュアル

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2000.12

《安全設計に関するお願い》

三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

《本資料ご利用に際しての留意事項》

- 本資料は、お客様が用途に応じた適切な三菱半導体製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について三菱電機株式会社・三菱電機セミコンダクタシステム株式会社が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。三菱半導体製品のご購入に当たりましては、事前に三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へ最新の情報をご確認頂きますとともに、三菱電機半導体情報ホームページ（http://www.semicon.melco.co.jp/）および三菱開発ツールホームページ（http://www.tool-spt.mesc.co.jp/）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社はその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へご照会ください。
- 本資料の転載、複製については、文書による三菱電機株式会社・三菱電機セミコンダクタシステム株式会社の事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

電子メールの場合：インストーラが生成する以下のテキストファイルに必要事項を記入の上、開発ツールサポート窓口support@tool.mesc.co.jpまで送信ください。

Windows 98/95/Windows NT 4.0版：\SUPPORT\製品名\SUPPORT.TXT
EWS版：/support/製品名/toolinfo.txt

FAXの場合：各ユーザーズマニュアル及びユーザーズマニュアルの最後に添付されている「技術サポート連絡書」に必要事項を記入の上、開発ツールサポート窓口まで送信ください。FAX送信先は「技術サポート連絡書」に記載してあります。

はじめに

MR79 は 7900 シリーズ用のリアルタイム・オペレーティングシステム[1]です。MR79 はμITRON 仕様[2]に準拠しています。

本マニュアルはMR79を使用したプログラムの作成手順および作成上の注意事項について説明します。各システムコールの詳細な使用方法については「MR79 リファレンスマニュアル」を参照してください。

MR79 を使うために必要なこと

MR79 を使用したプログラムを作成するには弊社下記製品を別途御購入して頂く必要があります。

- 7900 用 C コンパイラ NC79WA

  これらの製品をあわせて御使用頂ければ、より効率の良いプログラム開発がおこなえます。

ドキュメント一覧

MR79 に添付されているドキュメントは以下の３種類あります。

- リリースノート

  ソフトウェアの概要やユーザーズマニュアル、リファレンスマニュアルの訂正などを記載したドキュメントです。

- ユーザーズマニュアル (PDF ファイル)

  MR79 を使用したプログラムの作成手順や作成上の注意事項を記載したドキュメントです。

- リファレンスマニュアル (PDF ファイル)

  MR79 のシステムコールの使用方法や使用例を記述したドキュメントです。

  本マニュアルを読む前に必ずリリースノートをお読みください。

ソフトウェアの使用権

ソフトウェアの使用権はソフトウェア使用権許諾契約書に基づきます。MR79 はお客様の製品開発の目的でのみ使用できます。その他の目的での使用はできませんのでご注意ください。

また、本マニュアルによってソフトウェアの使用権の実施に対する保証及び使用権の実施の許諾をおこなうものではありません。

---

[1] 以降リアルタイム OS と略します。

[2] μITRON 仕様は、東京大学理学部坂村健博士とその研究室により考案されたものです。したがって、μITRON 仕様の著作権は同氏に属しています。MR79 は同氏に承認を得て、μITRON 仕様に基づき製作されたものです。

目次

図目次

表目次

第 1 章　　ユーザーズマニュアルの構成

MR79 ユーザーズマニュアルは、8 つの章から構成されています。

- 第 2 章 概要
  MR79 の目的や、概略の機能、位置づけなどを説明します。

- 第 3 章 MR79 入門
  MR79 を使用する上で必要となる考え方や用語などを説明します。

- 第 4 章 アプリケーション作成手順概要
  MR79 を使用してアプリケーションプログラムを作成する場合の開発手順の概要を説明します。

- 第 5 章 アプリケーション作成手順詳細
  MR79 を使用してアプリケーションプログラムを作成する場合の開発手順を詳細に説明します。

- 第 6 章 コンフィグレータの使用方法
  コンフィグレーションファイルの記述方法、および、コンフィグレータの使用方法を詳細に説明します。

- 第 7 章 アプリケーション作成の手引き
  MR79 を使用してアプリケーションプログラムを作成する際に知っておいたほうがよい事項や、注意事項について説明します。

- 第 8 章 サンプルプログラムの説明
  製品にソースファイル形式で含まれている MR79 サンプルアプリケーションプログラムについて説明します。

第 2 章　　概要

2.1 MR79 のねらい

近年マイクロコンピュータの急激な進歩にともない、マイクロコンピュータ応用製品の機能が複雑化してきています。これにともない、マイクロコンピュータのプログラムサイズが大きくなってきています。また製品開発競争が激化しマイクロコンピュータ応用製品を短期間に開発しなければなりません。すなわち、マイクロコンピュータのソフトウェアを開発している技術者は今までより大きなプログラムを今までより短期間で開発することが要求されてきます。そこでこの困難な要求を解決するためには以下のことを考えていかなければなりません。

1. ソフトウェアの再利用性を高めて、 開発すべきソフトウェアの量を削減する。

このためにはソフトウェアをできるだけ機能単位で独立したモジュールに分割して再利用できるようにする方法があります。すなわち、汎用サブルーチン集などを多く蓄積してそれをプログラム開発時に使用します。ただこの方法では、時間やタイミングに依存したプログラムは再利用するのは困難です。ところが実際の応用プログラムは時間やタイミングに依存したプログラムがかなりの部分を占めていてこのような手法で再利用できるプログラムはあまり多くありません。

2. チームプログラミングを推進し、1 つのソフトウェアを何名かの技術者でおこなうようにする。

チームプログラミングをおこなうには色々な問題があります。1 つはデバッグ作業をおこなうにあたり、チームプログラミングをおこなっている技術者全員のソフトウェアがデバッグできる状態にないとデバッグに入れません。また、チーム内の意志統一を十分におこなう必要があります。

3. ソフトウェアの生産効率を向上させ、技術者 1 名あたりの開発可能量を増加させる。

このためは 1 つは技術者の教育をおこない技術者のスキルアップをはかる方法があります。また、構造化記述アセンブラや C コンパイラなどを用いることによりより簡単にプログラムを作成できるようにする方法があります。また、ソフトウェアのモジュール化を推進してデバッグの効率を向上させる方法等があります。

しかし、このような問題を解決するには従来の手法では限界があります。そこでリアルタイム OS[3] という新しい手法の導入が必要になってきます。

そこで、弊社はこの要求に答えるべく 16 ビットマイクロプロセッサ 7900 用にリアルタイム OS MR79 を開発しました。MR79 を導入することにより以下のような効果があります。

1. ソフトウェアの再利用が容易になります。

リアルタイム OS を導入することにより、タイミングをリアルタイム OS を介してとることにより、タイミングに依存したプログラムが再利用できるようになります。

また、プログラムをタスクというモジュールに分割しますので、自然と構造化プログラミングをおこなうようになります。すなわち再利用可能なプログラムを自然に作成するようになります。

2. チームプログラミングがおこないやすくなります。

リアルタイム OS を導入することにより、プログラムがタスクという機能単位のモジュールに分割されますので、タスク単位で開発をおこなう技術者を振り分け開発からタスク単位でデバッグまでできるようになります。とくにリアルタイム OS を導入すると、プログラムが全てでき上がっていなくてもタスクさえ出来ていればその部分のデバッグを初めることが容易にできます。またタスク単位で技術者を割り振ることができますので、作業分担が容易におこなえます。

---

[3] OS：Operating System

3. ソフトウェアの独立性が高くなり、 プログラムをデバックしやすくなります。

リアルタイム OS を導入することにより、プログラムをタスクという独立した小さなモジュールに分割できますので、プログラムをデバッグする際ほとんどはその小さなモジュールに着目するだけでデバッグすることができます。

4. タイマ制御が簡単になります。

従来例えば、10ms ごとにある処理を動作させるためには、マイクロコンピュータのタイマ機能を用いて定期的に割り込みを発生させて処理させていました。ところが、マイクロコンピュータのタイマの数には限りがありますのでタイマが足らなくなったら 1 本のタイマを複数の処理に使用するなどの手法を用いて解決していました。

ところがリアルタイム OS を導入することにより、リアルタイム OS の時間管理機能を使用して一定時間毎にある処理をさせるというプログラムを、マイクロコンピュータのタイマ機能を特に意識せずに作成することができます。また、同時にプログラマから見たとき疑似的にマイクロコンピュータに無限本数のタイマが搭載されたようにプログラムを作成することができます。

5. ソフトウェアの保守性が向上します。

リアルタイム OS を導入することにより開発したソフトウェアが小さなタスクと呼ばれるプログラムの集合で構成されます。これにより開発完了後保守をおこなう場合、小さなタスクだけを保守すればよくなり保守性が向上します。

6. ソフトウェアの信頼性が向上します。

リアルタイム OS を導入することにより、プログラムの評価、試験などがタスクという小さなモジュール単位でおこなえますので評価、試験が容易になりひいては信頼性が向上します。

7. マイクロコンピュータの性能を最大限生かすことができます。これにより応用製品の性能向上が望めます。

リアルタイム OS を導入することにより、入出力待ちなどのマイクロコンピュータのむだな動作を減少させることができます。これによりマイクロコンピュータの能力を最大限に引き出すことができます。ひいては応用製品の性能向上につながります。

2.2 TRON 仕様と MR79

TRON 仕様とは The Realtime Operating system Nucleus 仕様の略で、リアルタイム・オペレーティングシステムの核となる部分の仕様を意味します。TRON 仕様の設計を中心とした TRON プロジェクトは、東京大学理学部 坂村健博士を中心として進められています。

この TRON プロジェクトで推進されているものの 1 つに ITRON 仕様があります。ITRON 仕様は Industrial TRON 仕様の略で、産業用組み込みシステムをターゲットとしたリアルタイム・オペレーティングシステムの仕様です。

ITRON 仕様は広い分野の応用に十分対応できるように数多くの機能を有しています。このため ITRON システムは比較的大きなメモリ容量と処理能力を必要とします。μITRON 仕様 V.2.0 は、この ITRON 仕様を処理速度を向上させるため仕様を整理し、必要十分な機能のみにサブセット化されたものです。μITRON 仕様 V.2.0 は以下の項目において ITRON 仕様のサブセットになっています。

1. **システムコールのタイムアウト機能がありません。**

2. **タスク、セマフォ等のオブジェクトは、システム作成時のみに生成[4]することができます。システム起動後に生成[5]することはできません。**

3. **メモリプールは固定長のみで、可変長のメモリプールは扱えません。**

4. **システムコール例外管理機能および CPU 例外管理機能がありません。**

現在、μITRON 仕様 V.3.0 が規定されています。このμITRON 仕様 V.3.0 はμITRON 仕様 V.2.0 と ITRON 仕様を統合し、接続機能を追加したものです。

MR79 は、このμITRON 仕様[6]に従って 16 ビットマイクロプロセッサ 7900 シリーズ用に開発された、リアルタイム・オペレーティングシステムです。

μITRON 仕様 V.3.0 では、各システムコールは、レベル R、レベル S、レベル E、レベル C に分けられています。

MR79 は、μITRON 仕様 V.3.0 で規定されたシステムコールのうち、レベル R、レベル S の全部と、レベル E の一部をインプリメントしています。

MR79 の概略仕様を表 2.1に示します。

---

[4] 静的オブジェクト生成
[5] 動的オブジェクト生成
[6] MR79 V.2.00 はμITRON 仕様 V3.0 に準拠しています。

表 2.1 MR79 概略仕様

項目	仕様
ターゲットマイクロプロセッサ	7900 シリーズ
最大タスク数	124
タスクの優先度数	124
最大イベントフラグ数	126
イベントフラグの幅	16 ビット
最大セマフォ数	126
セマフォの形式	計数型
最大メイルボックス数	126
メッセージサイズ	16 ビット or 24 ビット
メイルボックスのバッファサイズ	2^n 単位で 16K 個まで選択可能
最大固定長メモリプール数	126
最大可変長メモリプール数	1
システムコール数	61
OS 核コードサイズ	約 2K～12K バイト(LM モデル) 約 2K～12K バイト(SM モデル) 約 2K～14K バイト(LMI モデル) 約 2K～14K バイト(SMI モデル)
OS 核データサイズ	最小 17 バイト(LM,SM モデル) 18 バイト(LMI,SMI モデル) 1 タスクあたり（スタックを除く）11 バイト増加。なお、タイムアウト機能を使用した場合は、15 バイト増加
OS 核記述言語	C 言語、アセンブリ言語

2.3 MR79 の特長

MR79 は以下に示す特長を持っています。

1. μITRON 仕様に準拠したリアルタイム・オペレーティングシステム

MR79 は ITRON 仕様をワンチップマイクロコンピュータでも実装できるように機能を整理したμITRON 仕様に基づいて開発されました。

μITRON 仕様は ITRON 仕様のサブセットですので ITRON 教科書として出版されている文献や ITRON セミナー等で得た知識をほとんどそのまま役立てることができます。また、ITRON 仕様に準拠したリアルタイム OS を用いて開発したアプリケーションプログラムを MR79 に移行するのは比較的容易に行えます。

2. 高速処理を実現

マイコンのアーキテクチャを活用し、高速処理を実現しています。

3. 必要モジュールのみを自動選択することにより常に最小サイズのシステムを構築

MR79 は 7900 シリーズオブジェクトライブラリ形式で供給されています。

したがって、リンケージエディタ LN79 のもつ機能により、数ある MR79 の機能モジュールのなかで使用しているモジュールのみを自動選択してシステムを生成します。このため、常に最小サイズのシステムが自動的に生成されます。

4. C コンパイラ NC79 を用いて C 言語でアプリケーションプログラムが開発可能

C コンパイラ NC79 を用いて、MR79 のアプリケーションプログラムを C 言語で開発できます。また C 言語から MR79 の機能を呼び出すためのインターフェースライブラリがソフトウェアディスクに添付されています。

5. 上流工程ツール "コンフィグレータ" により、容易な開発手順

ROM 書き込み形式ファイルまでの作成を簡単な定義のみでおこなえるコンフィグレータを装備しています。これにより、どんなライブラリを結合する必要があるかなどを特に気にする必要はありません。

第 3 章　　MR79 入門

3.1 リアルタイム OS の考え方

本節では、リアルタイム OS の基本概念について説明します。

3.1.1 リアルタイム OS の必要性

近年半導体技術の進歩にともなってシングルチップマイクロコンピュータ（マイコン)の ROM 容量が増大してきています。

このような大 ROM 容量のマイクロコンピュータの出現によりそのプログラム開発が従来の方法では困難になってきています。図 3.1にプログラムサイズと開発期間（開発の困難さ）との関係を示します。この図 3.1はあくまでイメージ図ですが、プログラムのサイズが大きくなるに従い開発期間が指数関数的に長くなってきます。

例えば 32K バイトのプログラムを 1 個開発するより、8K バイトのプログラムを 4 個開発する方が簡単です。[7]

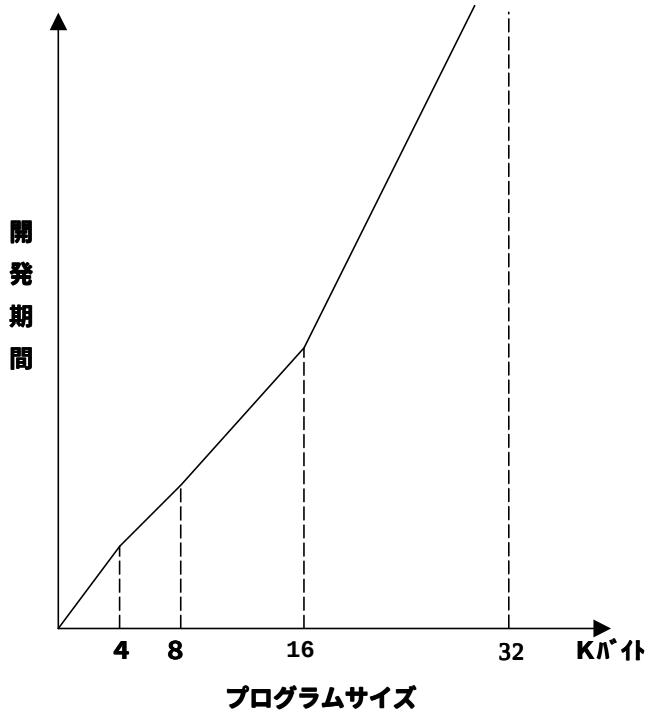


図 3.1 プログラムサイズと開発期間

そこで大きなプログラムを短期間に簡単に開発するための手法が必要になってきます。この方法として小さな ROM 容量のマイクロコンピュータを多く使う方法があります。たとえば、図 3.2にオーディオ機器システムを複数のマイクロコンピュータで構成した例を示します。

[7] ROM 詰めの必要がないことを前提とします

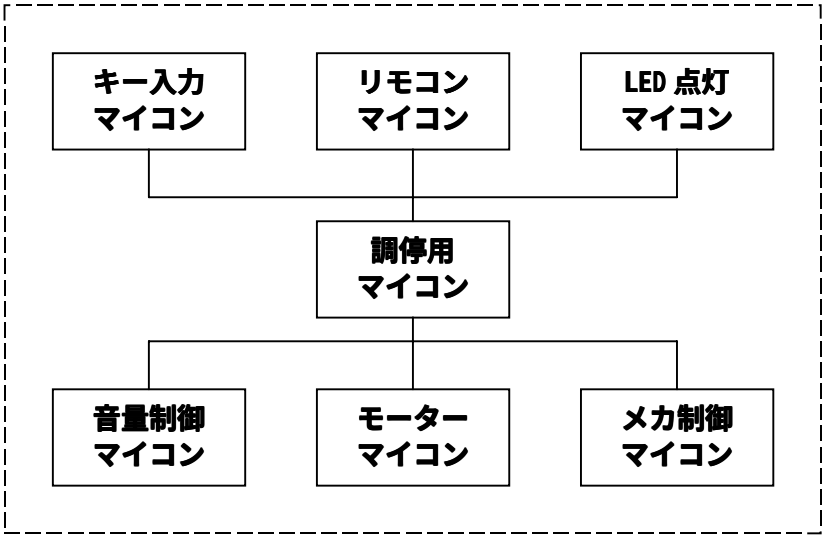


図 3.2　マイコンを多く使ったシステム例（オーディオ機器）

このように機能単位で別々のマイクロコンピュータを用いることは以下の利点があります。

1. **ひとつひとつのプログラムが小さくなり、プログラム開発が容易になる。**
2. **一度開発したソフトウェアを再利用することが非常に容易になる。**[8]
3. **完全に機能ごとにプログラムが分離するので複数の技術者でプログラム開発が容易にできる。**

この反面以下のような欠点があります。

1. **部品点数が多くなり製品の原価を上昇させる。**
2. **ハードウェア設計が複雑になる。**
3. **製品の物理的サイズが大きくなる。**

そこでそれぞれのマイクロコンピュータで動作しているプログラムを、1 つのマイクロコンピュータでソフトウェア的に、別々のマイクロコンピュータで動作しているように見せることのできるリアルタイム OS を採用すれば、上記の利点を残したままで欠点をすべて無くすことができます。
図 3.3に、図 3.2に示したシステムにリアルタイム OS を導入した場合のシステム例を示します。

[8] 例えば、図 3.2において、リモコンマイコンを別の製品にそのまま使用することができる。

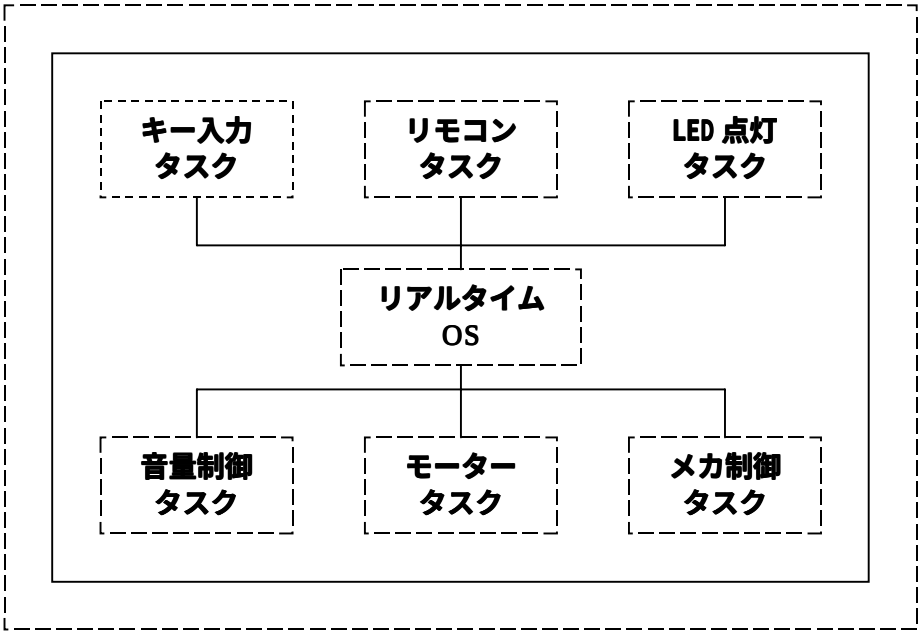


図 3.3　リアルタイム OS の導入システム例（オーディオ機器）

すなわちリアルタイム OS とは 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せるソフトウェアです。複数のマイクロコンピュータに相当するひとつひとつのプログラムをリアルタイム OS 用語でタスクと呼びます。

3.1.2 リアルタイム OS の動作原理

リアルタイム OS は 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せることのできるソフトウェアです。では 1 個のマイクロコンピュータをどのようにして複数あるように見せかけるのでしょうか？

それは、図 3.4に示すようにそれぞれのタスクを時分割で動作させるからです。つまり実行するタスクを一定時間ごとに切り替えて、複数のタスクが同時に実行しているように見せるのです。

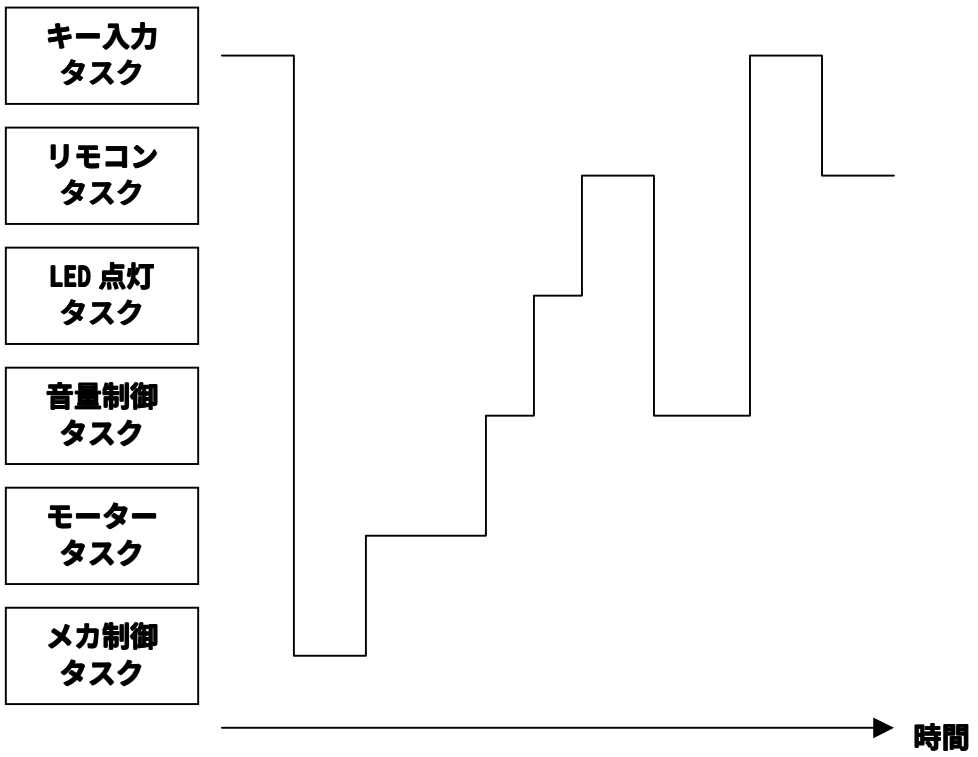


図 3.4 タスクの時分割動作

このようにタスクを一定時間ごとに切り替えて実行しています。このタスクを切り替えることをリアルタイム OS 用語でディスパッチと呼ぶこともあります。タスク切り替え (ディスパッチ)が発生する要因として以下のものがあります。

- 自分自身で切り替えを要求する。
- 割り込みなどの外的要因で切り替わる。

タスク切り替えが発生し、再度、そのタスクを実行するときには、中断していたところから再開します。 (図 3.5参照)

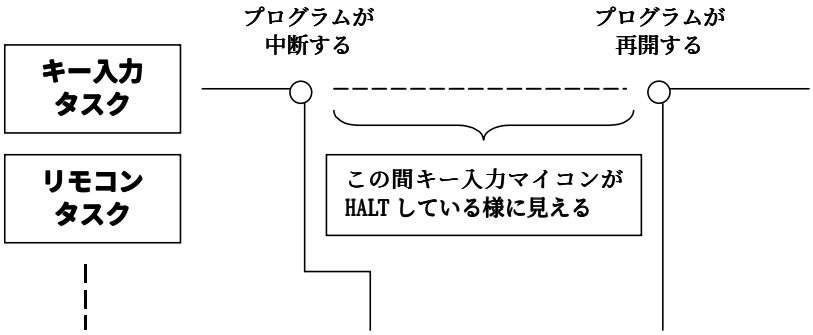


図 3.5　タスクの中断と再開

図 3.5においてキー入力タスクは、 他のタスクに実行制御が移っている間、プログラマから見ればプログラムが中断しそのマイコンが HALT しているようにみえます。

タスクの実行は、中断した時点のレジスタ内容を復帰することにより、中断した時点の状態で再開されます。すなわちタスクの切り替えとは、現在実行中のタスクのレジスタの内容をそのタスクを管理するメモリ領域に退避し、切り替えるタスクのレジスタ内容を復帰することです。

すなわちリアルタイム OS を実現するには、タスクごとにレジスタを管理し、切り換えが発生する度にそのレジスタ内容を入れ換えることにより複数のマイクロコンピュータが存在しているように見せてやればよいということになります。 (図 3.6参照)

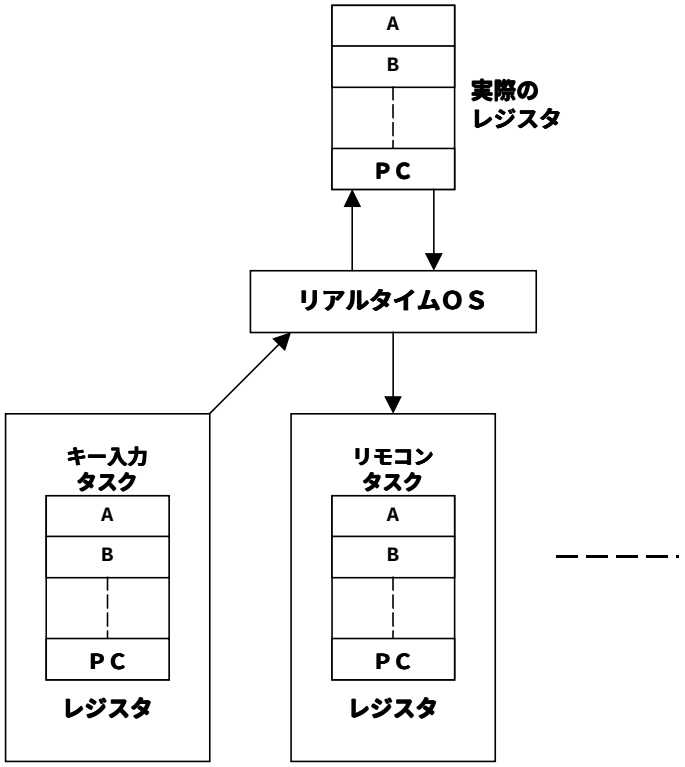


図 3.6　タスクの切り替え

図 3.7は各タスクのレジスタをどのように管理しているか具体的に示したものです。

実際にはタスクごとに持つ必要のあるのはレジスタだけでなく、スタック領域もタスクごとに持つ

必要があります。

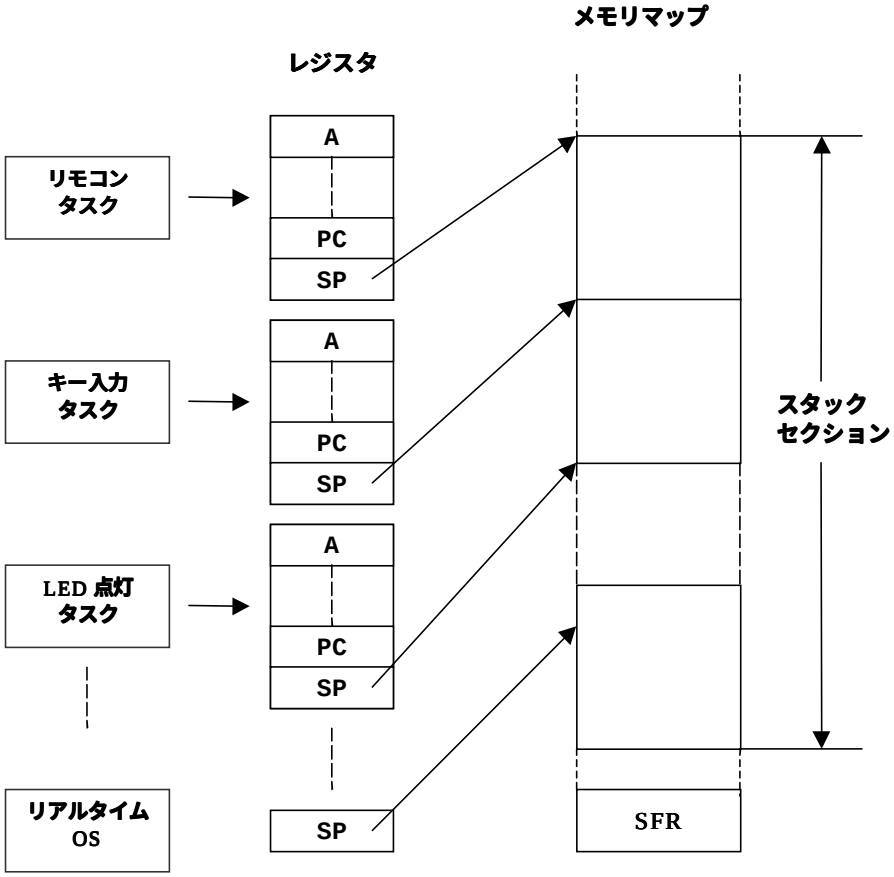


図 3.7　タスクのレジスタ領域

図 3.8は各タスクのレジスタおよびスタック領域を詳細に説明したものです。MR79 では各タスクのレジスタは図 3.8に示すようにスタック領域の中に格納され管理されています。図 3.8は、レジスタ格納後の状態を示しています。

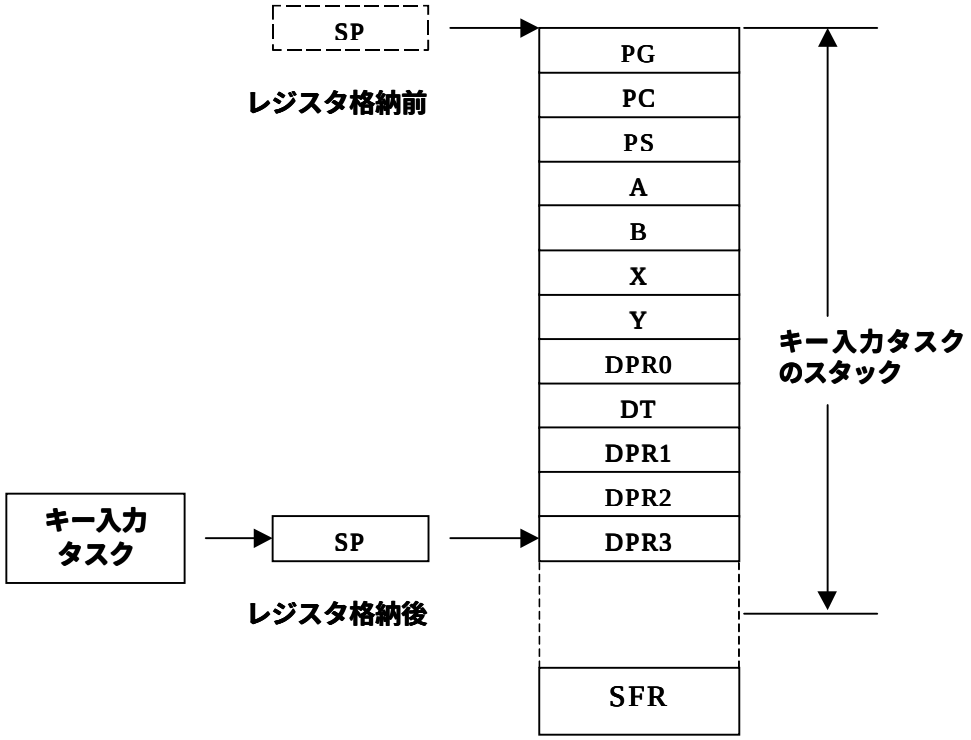


図 3.8　実際のレジスタとスタック領域の管理

3.2 システムコール

リアルタイム OS をプログラマはプログラム中でどのように使用するのでしょうか?

これにはリアルタイム OS の機能をプログラムから何らかの形で呼び出すことが必要です。こリアルタイム OS の機能を呼び出すことをシステムコールといいます。すなわちシステムコールのにより、タスクの起動などの処理を行なうことができます (図 3.9参照)。

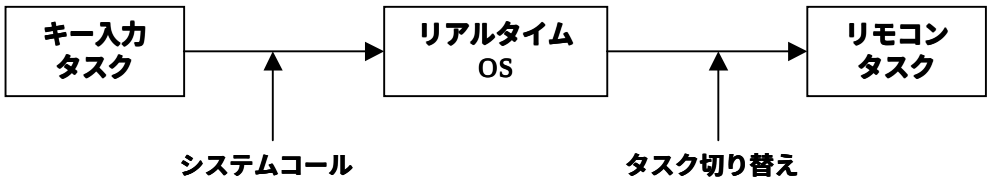


図 3.9 システムコール

このシステムコールは、C 言語で応用プログラムを記述する場合は関数呼び出しで実現します。すなわち、

```
sta_tsk(ID_main,3);
```

またアセンブリ言語で応用プログラムを記述する場合はアセンブラマクロ呼び出しにより実現します。すなわち、

```
sta_tsk ID_main,3
```

3.2.1 システムコール処理

システムコールが発行されると以下の手順により処理がおこなわれます。[9]

1. 現レジスタ内容を退避します。
2. スタックポインタをタスクのものからリアルタイム OS(システム)のものへ切り替えます。
3. システムコール要求にしたがった処理を行います。
4. 次に実行するタスクの選択をおこないます。
5. スタックポインタをタスクのものに切り替えます。
6. レジスタ内容を復帰してタスクの実行を再開します。

システムコールが発生してからタスク切り替えまでの処理の流れを図 3.10に示します。

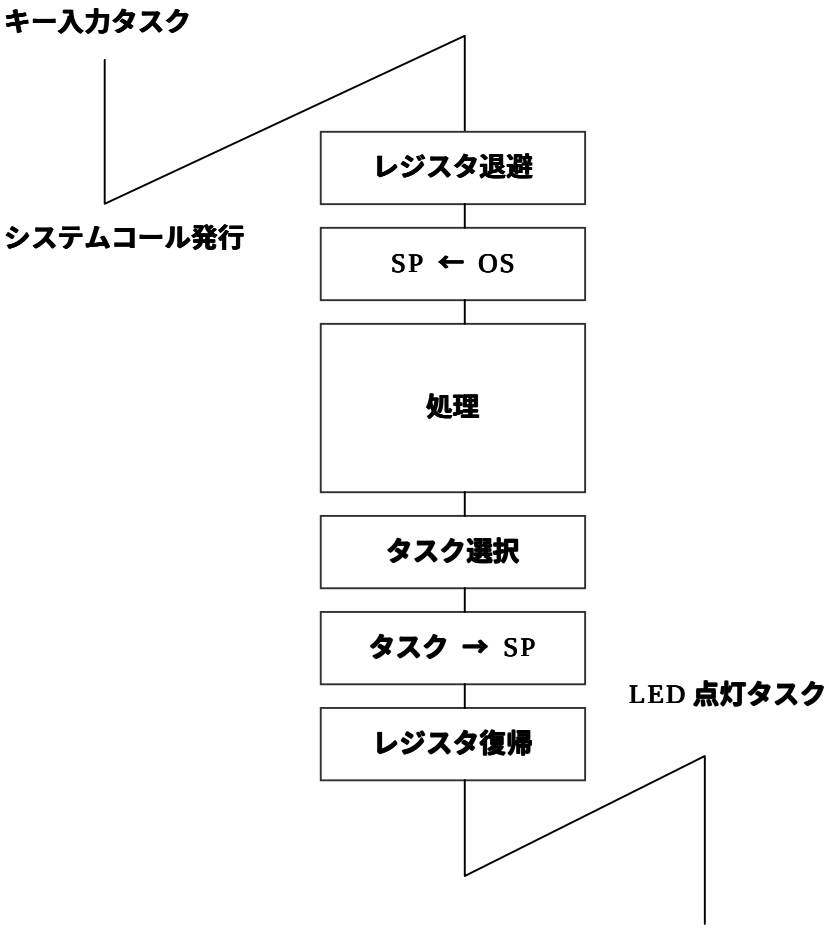


図 3.10 システムコールの処理の流れ

[9] タスク切り替えの発生しないシステムコールはこの限りではありません。

3.2.2 システムコールにおけるタスクの指定方法

各タスクの識別は、リアルタイム OS MR79 の内部では ID 番号でおこないます。
すなわち、"タスク ID 番号 1 のタスクを起動する"などというように管理されています。
しかし、プログラム中にタスクの番号を直接書き込むと非常に可読性の低いプログラムになってしまいます。たとえば、

```
sta_tsk(2,1);
```

とプログラム中に記述するとプログラマは絶えず ID 番号の 2 番のタスクは何かを知っている必要があります。また、他人がこのプログラムを見たときに ID 番号の 2 番のタスクが何かが一目では分かりません。
そこで MR79 ではタスクの識別をそのタスクの名前（関数もしくはシンボルの名前）で指定し、その名前からタスクの ID 番号への変換を MR79 に付属しているプログラム"コンフィグレータ cfg79"が自動的におこないます。図 3.11にその様子を示します。

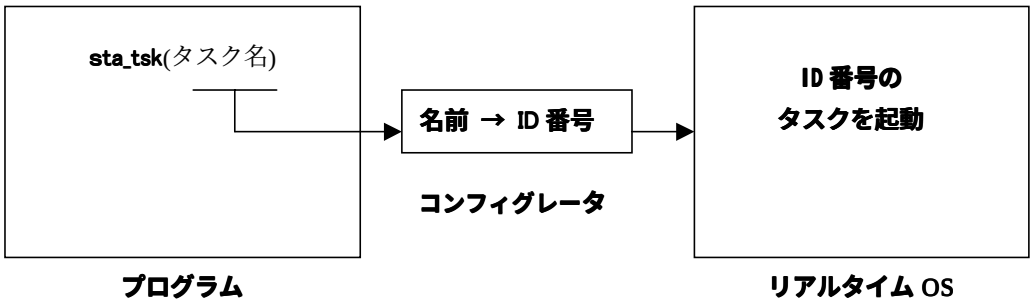


図 3.11 タスクの識別

これによりタスクの指定を以下のようにおこなえます。

```
sta_tsk(ID_task,1);
```

この例では、関数名"task()"もしくはシンボル名"task:"のタスクを起動するように指定しています。
なお、タスクの名前か ID ら番号への変換は、プログラムを生成するときにおこないます。したがって、この機能による処理速度の低下はありません。

3.3 タスク

本節ではタスクをリアルタイム OS MR79 がどのように管理しているかを説明します。

3.3.1 タスクの状態

リアルタイム OS ではタスクを実行するべきか否かを、タスクの状態を管理することにより制御しています。例えば、図 3.12にキー入力タスクの実行制御と状態の関係を示します。キー入力が発生した場合はそのタスクを実行しなければなりません。すなわち、キー入力タスクが実行状態となります。またキー入力を待っているときはタスクを実行する必要はありません。すなわち、キー入力タスクは待ち状態になっています。

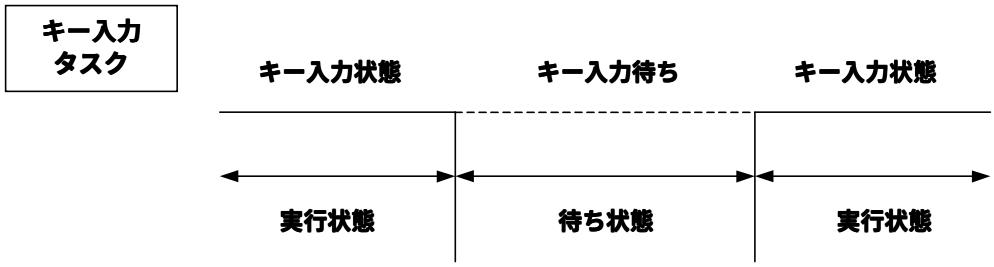


図 3.12 タスクの状態

MR79 では実行状態、待ち状態を含め以下の 6 つの状態を管理しています。

1. **実行状態 (RUN 状態)**
2. **実行可能状態 (READY 状態)**
3. **待ち状態 (WAIT 状態)**
4. **強制待ち状態 (SUSPEND 状態)**
5. **二重待ち状態 (WAIT-SUSPEND 状態)**
6. **休止状態 (DORMANT 状態)**

タスクは上記の 6 つの状態を遷移していきます。図 3.13に、タスクの状態遷移図を示します。

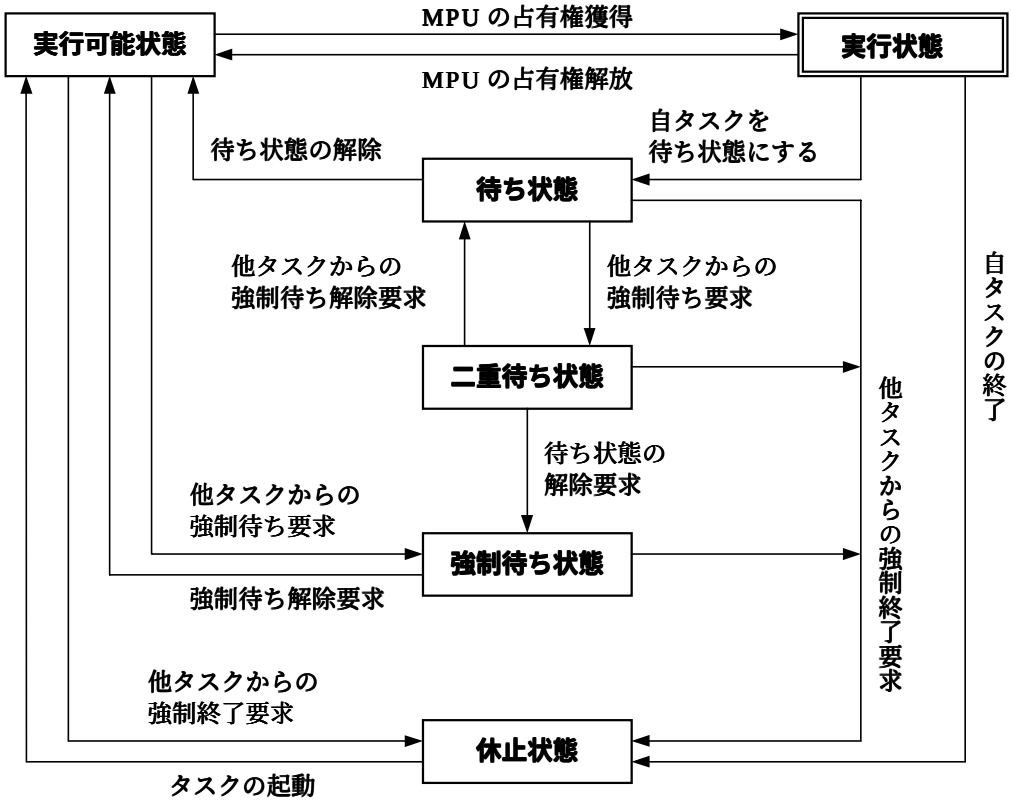


図 3.13 MR79 のタスク状態遷移図

1. 実行状態 (RUN 状態)

タスクが、まさに現在実行中の状態を実行状態といいます。マイクロコンピュータは１つしかないのですから当然実行状態にあるのは常に１つだけです。

現在実行状態のタスクが他の状態に移行するには、以下の事象のうちどれかが発生した場合です。

- 自分で自タスクを正常終了させた場合[10]
- 自分で待ち状態に入った場合[11]
- 割り込み等の事象の発生により、その割り込みハンドラによって自タスクより優先度の高いタスクが実行可能状態になった場合
- 自タスクの優先度を変更することにより他の実行可能状態のタスクが自タスクより優先度が高くなった場合[12]
- 割り込み等の事象の発生により自タスクもしくは他の実行可能状態のタスクの優先度が変更され、そのため他の実行可能状態のタスクが自タスクより優先度が高くなった場合[13]

上記の事象が発生すると再スケジュールされて実行状態と実行可能状態にあるタスクのなかで最も優先度の高いタスクが実行状態に移され、そのタスクのプログラムが実行されます。

[10] ext_tsk システムコールによる
[11] dly_tsk, slp_tsk, tslp_tsk, wai_flg, twai_flg, wai_sem, twai_sem, rcv_msg, trcv_msg システムコールによる
[12] chg_pri システムコールによる
[13] ichg_pri システムコールによる

2. 実行可能状態 (READY 状態)

タスクが実行される条件は整っているが、そのタスクより優先度の高いタスクもしくは同一優先度のタスクが実行されているために実行できずに実行待ち状態になっている状態を実行可能状態といいます。

実行可能状態であるタスクで、レディキュー[14]では 2 番目に実行される可能性のあるタスクが実行状態になるのは、以下の事象の内いずれかが発生した場合です。

- 実行状態のタスクが自分で正常終了した場合[15]
- 実行状態のタスクが自分で待ち状態に入った場合[16]
- 実行状態のタスクが自分で優先度を変更することにより実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[17]
- 割り込み等の事象の発生により実行状態のタスクの優先度が変更され、そのため実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[18]

3. 待ち状態 (WAIT 状態)

実行状態のタスクが自分自身を待ち状態に移行させる要求を出すことにより、タスクは実行状態から待ち状態に移行することができます。待ち状態は通常入出力装置の入出力動作完了待ちや他のタスクの処理待ちなどの状態として使用されます。

実行待ち状態に移行するには以下の方法があります。

- slp_tsk システムコールにより単純に待ち状態に移行します。この場合、他のタスクから明示的に待ち状態から解除されないと実行可能状態に移行しません。
- dly_tsk システムコールにより一定時間待ち状態に移行します。この場合、指定時間経過するかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- wai_flg、wai_sem、rcv_msg システムコールにより要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- tslp_tsk、twai_flg、twai_sem、trcv_msg システムコールは、slp_tsk、wai_flg、wai_sem、rcv_msg システムコールにタイムアウトを指定したシステムコールです。各システムコールの要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは、指定時間が経過した場合、実行可能状態に移行します。
- タスクが wai_flg、wai_sem、rcv_msg システムコール[19]により要求待ちで待ち状態になると、その要求事項により次の待ち行列のいずれかにつながります。

- イベントフラグ待ち行列 (Event Flag Queue)
- セマフォ待ち行列 (Semaphore Queue)
- メイルボックス待ち行列 (Mailbox Queue)

4. 強制待ち状態 (SUSPEND 状態)

実行状態のタスクから sus_tsk システムコールが発行される、もしくはハンドラから

---

[14] レディーキューについては次節参照
[15] ext_tsk システムコールによる
[16] dly_tsk, slp_tsk, tslp_tsk, wai_fg, twai_flg, wai_sem, twai_sem, rcv_msg, trcv_msg システムコールによる
[17] chg_pri システムコールによる
[18] icg_pri システムコールによる
[19] twai_flg,twai_sem,trcv_msg システムコールも含まれます。

isus_tsk システムコールが発行されると、システムコールにより指定された実行可能なタスクもしくは実行中のタスクは強制待ち状態になります。なお待ち状態のタスクが指定された場合は二重待ち状態になります。

強制待ち状態は入出力エラー等の発生により実行可能なタスクもしくは実行中のタスク[20]が処理を一時的に中断させるためにスケジューリングから外された状態です。すなわち実行可能状態のタスクに対して強制待ち要求が出された場合、そのタスクは実行待ち行列から外されます。

なお、強制待ち要求のキューイングは行いません。したがって強制待ち要求は実行状態、実行可能状態、待ち状態[21]にあるタスクにのみ行えます。すでに強制待ち状態にあるタスクに強制待ち要求した場合には、エラーコードが返されます。

5. 二重待ち状態 (WAIT-SUSPEND 状態)

待ち状態にあるタスクに強制待ちの要求が出された場合、タスクは二重待ち状態になります。wai_flg、wai_sem、rcv_msg システムコールによる要求待ちで待ち状態にあるタスクに対して強制待ち要求が出された場合、そのタスクは要求待ち行列から外されず単にそのタスクが二重待ち状態に移行するだけです。

また、二重待ち状態のタスクは待ち条件が解除されると強制待ち状態になります。待ち条件が解除されるには以下の場合が考えられます。

- wup_tsk、iwup_tsk システムコールにより起床する場合
- dly_tsk、tslp_tsk システムコールにより待ち状態になったタスクが時間経過により起床される場合
- wai_flg、wai_sem、rcv_msg、twai_flg、twai_sem、trcv_msg システムコールにより待ち状態になったタスクの要求が満たされた場合
- rel_wai、irel_wai システムコールにより待ち状態が強制解除される場合二重待ち状態のタスクに強制待ち解除要求[22]がだされると待ち状態になります。なお、強制待ち状態にあるタスクが自分自身を待ち状態にする要求は出せないため、強制待ち状態から二重待ち状態への移行は発生しません。

6. 休止状態 (DORMANT 状態)

通常は、MR79 システムに登録されているが起動していない状態です。この状態になるには以下の 2 つの場合があります。

- タスクが起動をかけられるのを待っている場合
- タスクが正常終了[23]もしくは強制終了[24]により終了した場合

[20] ハンドラから isus_tsk システムコールにより実行タスクを強制待ち状態にする場合は、実行状態から直接強制待ち状態に移行されます。例外的にこの場合のみ実行状態から強制待ち状態に移行する場合はあることに注意してください。
[21] 待ち状態にあるタスクに対して強制待ち要求をおこなうと二重待ち状態になります。
[22] rsm_tsk, irsm_tsk システムコール
[23] ext_tsk システムコール
[24] ter_tsk システムコール

3.3.2 タスクの優先度とレディキュー

リアルタイム OS では実行したいタスクが同時にいくつも発生することがあります。

このときにどのタスクを実行するかを判断することが必要になります。そこでタスクに実行の優先度をつけ、優先度の高いタスクから実行するようにします。すなわち、処理を素早くおこなう必要のあるタスクの優先度を高くしておけば実行したいときに素早く実行することができるようになります。

MR79 では同一の優先度を複数のタスクに与えることができます。そこで、実行可能になったタスクの実行順を制御するためにタスクの待ち行列 (レディキュー)を生成します。

図 3.14[25]にレディキューの構造を示します。レディキューは優先度ごとに管理され、タスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。[26]

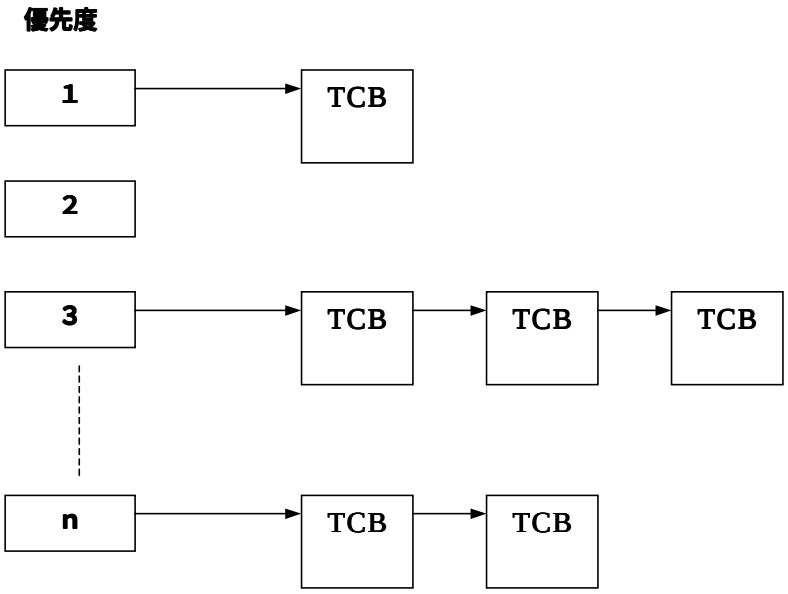


図 3.14 レディーキュー（実行待ち状態）

[25] TCB:タスクコントロールブロックについては次節で述べます。
[26] 実行状態のタスクはレディキューにつながれたままです。

3.3.3 タスクコントロールブロック (TCB)

タスクコントロールブロック (TCB)とは、リアルタイム OS がそれぞれのタスクの状態や優先度などを管理するデータブロックのことを言います。

MR79 ではタスクの以下の情報をタスクコントロールブロックとして管理しています。

- タスク接続ポインタ
  レディキューなどを構成するときに使用するタスク接続用ポインタ
- タスクの優先度
- タスクのレジスタ情報など[27]を格納したスタック領域のポインタ (現在の SP レジスタの値)
- タスクの状態
- 起床要求カウンタ
  タスクの起床要求カウンタを蓄積する領域
- タイムアウトカウンタ、待ちフラグパターン
  タスクがタイムアウト待ち状態である時は、残りの待ち時間が格納され、フラグ待ち状態であれば、フラグの待ちパターンがこの領域に格納されます。
- フラグ待ちモード
  イベントフラグ待ちの時の待ちモード
- タイマキュー接続ポインタ
  タイムアウト機能を使用した場合に使用する領域です。タイマキューを構成する時に使用するタスクの接続用ポインタを格納する領域です。
- フラグ待ちパターン
  タイムアウト機能を使用した場合に使用する領域です。
  タイムアウト機能付きのイベントフラグ待ちのシステムコール (twai_flg)を使用した場合に、フラグ待ちパターンが格納されます。なお、この領域は、イベントフラグを使用しない場合は、確保されません。

タスクコントロールブロックを図 3.15に示します。

[27] これをタスクコンテキストと呼びます。

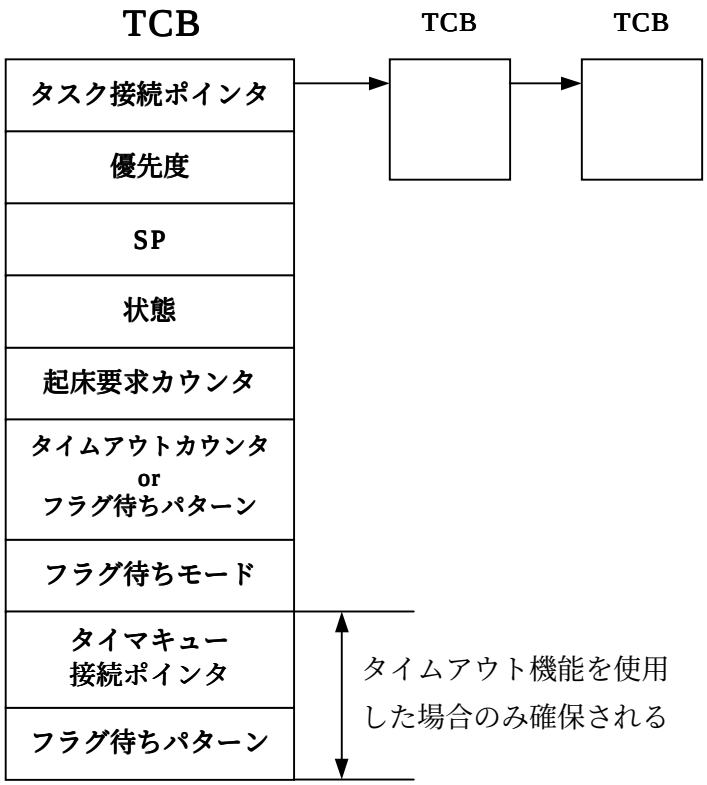


図 3.15 タスクコントロールブロック

3.4 ハンドラ

3.4.1 タスクとハンドラの違い

タスクはMR79が実行制御するプログラムの単位ですから、タスクはそれぞれ独立したコンテキスト(プログラムカウンタ、スタックポインタ、ステータスレジスタ、その他のレジスタ)を持っています。したがってあるタスクからあるタスクに実行を移す[28]にはこのコンテキストを切り替える必要があります。この処理には時間を要します。

割り込みなどの処理は、高速に応答する必要がありますのでMR79にはコンテキストを切り替えずに処理する機能があります。すなわち割り込まれたタスクのコンテキスト（レジスタ）をそのまま使用してプログラムを動作させることができます。このプログラムのことをハンドラと呼びます。ハンドラは割り込まれたタスクのコンテキスト（レジスタ)をそのまま使用しますので必ずハンドラの先頭で割り込まれたタスクのコンテキストをメモリに退避し、タスクに復帰するときにはその退避したコンテキストをもとに戻さなくてはなりません。

また、アセンブリ言語で記述する場合、割り込みハンドラからの復帰にはret_intシステムコールを通常は使用してください。(5.2.2節を参照)ただし割り込みハンドラ処理内でMR79のシステムコールを使用していない場合は、rti命令で復帰してもかまいません。(5.2.3節を参照してください。)

ハンドラには以下のものがあります。

1. **割り込みハンドラ**

   ハードウェア割り込みにより起動されるプログラムを割り込みハンドラと呼びます。割り込みハンドラの起動にはMR79は全く関与しません。したがって割り込みハンドラの入り口アドレスを割り込みベクターテーブルに直接書き込みます。

   割り込みハンドラには、OS独立割り込み、OS依存割り込みの2種類があります。各割り込みについては、5.3節を参照してください。

2. **周期起動ハンドラ**

   周期起動ハンドラはあらかじめ設定された時間毎に周期的に起動されるプログラムです。設定された周期起動ハンドラを無効にするか有効にするかは周期起動ハンドラの活性状態の変更[29]によりおこないます。

3. **アラームハンドラ**

   アラームハンドラは、 指定した時刻に起動されるハンドラです。

   なお、set_timシステムコールにより、すでに実行されたアラームハンドラの起動時刻よりも前の時刻にシステムクロックの値を戻しても、そのアラームハンドラが再び起動されることはありません。また、まだ起動されていないアラームハンドラの起動時刻よりも後の時刻を指定した場合は、すべてのアラームハンドラが起動されません。

周期起動ハンドラとアラームハンドラはシステムクロック割り込み（タイマ割り込み)ハンドラからサブルーチンコールで呼び出されます（図 3.16参照)。したがって、周期起動ハンドラ、アラームハンドラはシステムクロック割り込みハンドラの一部として動作します。なお、周期起動ハンドラ、アラームハンドラが呼び出されるときは、システムクロック割り込みの割り込み優先レベルの状態で実行されます。

[28] このことをディスパッチもしくはタスク切り替えと呼びます。
[29] act_cycシステムコール

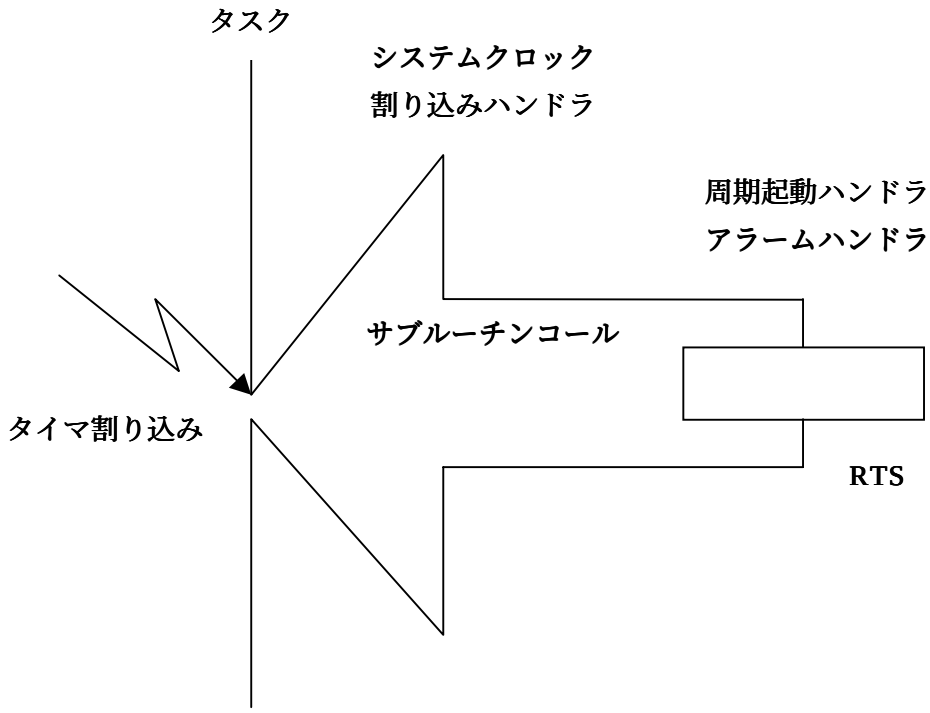


図 3.16 周期起動ハンドラ、アラームハンドラの起動

3.4.2 ハンドラ専用のシステムコール

MR79 では以下に示すシステムコールはハンドラからのみ発行できます。

なお、ret_int システムコールは、割り込みハンドラ専用です[30]。したがって周期起動ハンドラ、アラームハンドラからは発行できません。

表 3.1 ハンドラから発行できるシステムコール

システムコール名		機能
ichg_pri	Change Task Priorit	タスクの優先度を変更する
irot_rdq	Rotate Ready Queue	タスクのレディキューを回転する
irel_wai	Release Task Wait	タスクの待ち状態を強制解除する
isus_tsk	Suspend Task	タスクを強制待ち状態へ移行する
irsm_tsk	Resume Task	強制待ち状態のタスクを再開する
iwup_tsk	Wakeup Task	待ち状態のタスクを起床する
iset_flg	Set EventFlag	イベントフラグをセットする
isig_sem	Signal Semaphore	セマフォに対する信号操作
isnd_msg	Send Message to Mailbox	メッセージを送信する
ista_tsk	Start Task	タスクを起動する
ret_int	Return from Interrupt Handler	割込みハンドラから復帰する

[30] #pragma INTHANDLER で割り込みハンドラが指定されている場合（C 言語）、このシステムコールを記述する必要はありません。

3.5 MR79 カーネルの構成

3.5.1 モジュール構成

MR79 カーネルは、図 3.17に示すモジュールから構成されています。これらの個々のモジュールはそれぞれのモジュールの機能を実現する関数群より構成されています。

MR79 カーネルはライブラリ形式で提供されシステム生成時に必要な機能のみがリンクされます。すなわちこれらのモジュールを構成する関数群の中で使用している関数のみをリンケージエディタ LN79 の機能によりリンクします。ただし、スケジューラとタスク管理の一部および時間管理の一部は必須機能関数ですので常時リンクされます。

アプリケーションプログラムはユーザーが作成するプログラムで、タスク・割り込みハンドラ・アラームハンドラおよび周期起動ハンドラ[31]から構成されます。

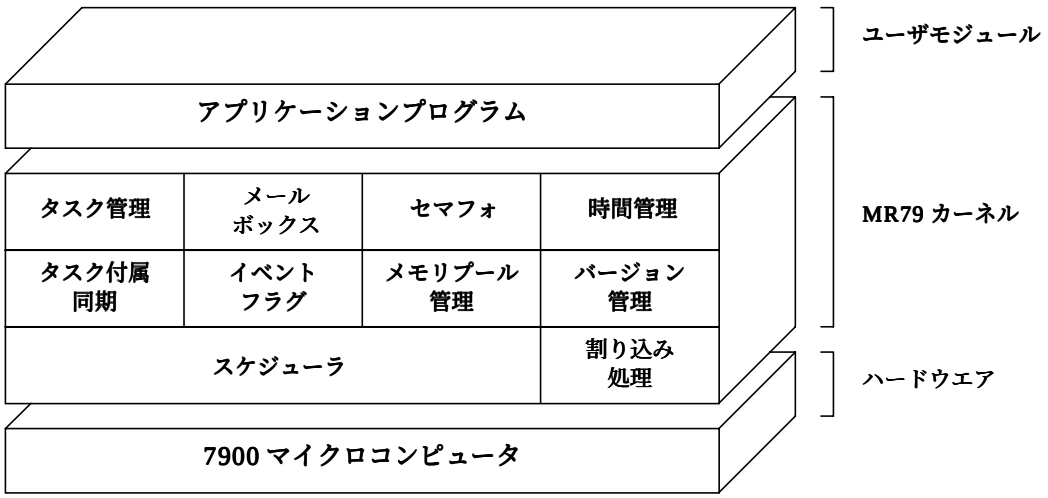


図 3.17 MR79 の構成

[31] 詳細は第3.5.10節を参照。

3.5.2 モジュール概要

MR79 カーネルを構成する各モジュールの概要を説明します。

- スケジューラ
  タスクの持つ優先度に基づいて、タスクの処理待ち行列を形成し、その待ち行列の先頭にある優先度の高い (優先度の値の小さい)タスクの処理を実行するよう制御をおこないます。

- タスク管理
  実行・実行可能・待ち・強制待ち等のタスク状態の管理をおこないます。

- タスク付属同期
  他タスクからタスクの状態を変化させることによりタスク間の同期をとります。

- 割り込み管理
  割り込みハンドラからの復帰処理をおこないます。

- 時間管理
  MR79 カーネルで使用するシステムタイマの設定、タイムアウトの処理、ユーザーの作成したアラームハンドラ[32]、周期起動ハンドラ[33]の起動をおこないます。

- バージョン管理
  MR79 カーネルのバージョン番号等の情報を報告します。

- 同期・通信
  タスク間の同期をとったりタスク間の通信をおこなうための機能です。以下の 3 つの機能モジュールが用意されています。
  - ◆ イベントフラグ
    MR79 内部で管理されているフラグが立っているか否かによりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ◆ セマフォ
    MR79 内部で管理されているセマフォカウンタ値によりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ◆ メイルボックス
    タスク間のデータの通信をデータの先頭アドレスを渡すことによりおこないます。

- メモリプール管理
  タスクまたはハンドラが使用するメモリ領域の動的な獲得および解放を行います。

[32] 指定時刻に一回のみ起動されるハンドラです。
[33] 周期的に起動されるハンドラです。

3.5.3 タスク管理機能

タスク管理機能とは、タスクの起動・終了・優先度の変更等のタスク操作をおこなう機能です。MR79 カーネルが提供するタスク管理機能のシステムコールには、次のものがあります。

- タスクを起動する（sta_tsk）
  あるタスクから、他タスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態もしくは実行状態に移行します。

- ハンドラからタスクを起動する（ista_tsk）
  ハンドラからタスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態に移行します。

- 自タスクを終了する（ext_tsk）
  自タスクを終了するとタスクの状態が休止状態になります。これにより再起動されるまではこのタスクは実行しません。

- 他タスクを強制的に終了させる（ter_tsk）
  休止状態以外の他のタスクを強制的に終了させ休止状態にします。
  タスクを強制的に終了させた後に再度そのタスクを起動するとそのタスクがリセットしたように振る舞います。（図 3.18参照）

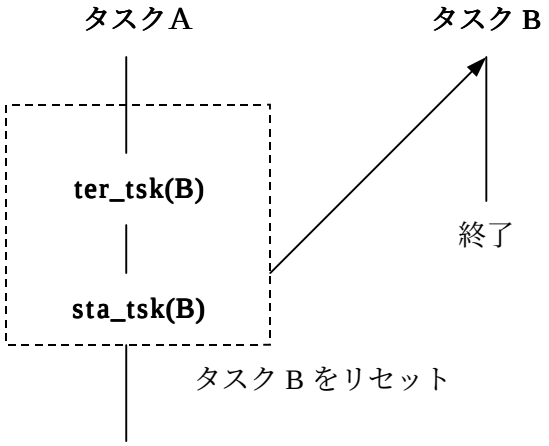


図 3.18 タスクのリセット

- タスクの優先度を変更する（chg_pri，ichg_pri）
  タスクの優先度を変更するとそのタスクが実行可能状態もしくは実行状態であるときは、レディキューも更新されます。（図 3.19参照）

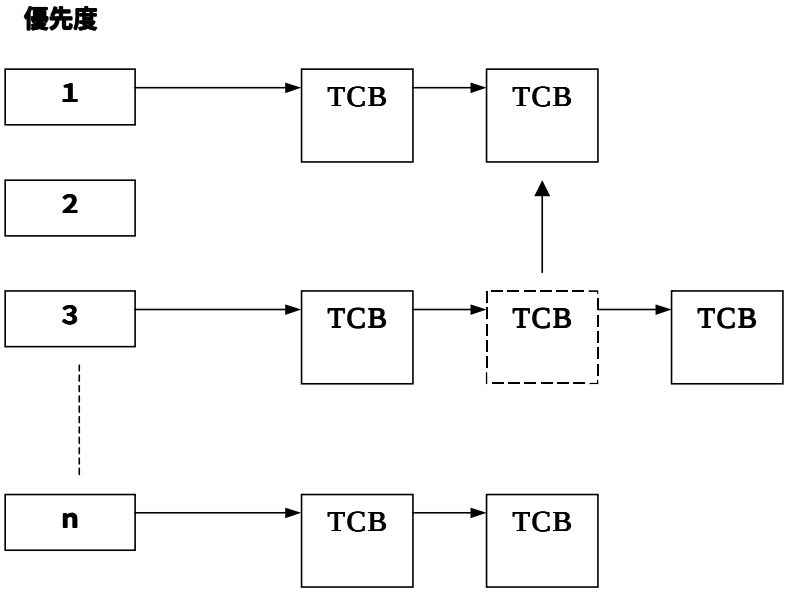


図 3.19 優先度の変更

- タスクの実行待ち行列を回転する（rot_rdq，irot_rdq）
  本システムコールにより TSS(タイムシェアリングシステム) を実現することができます。すなわち、一定周期でレディキューを回転すれば、TSS で必要なラウンドロビンスケジューリングを実現することができます。 (図 3.20参照)

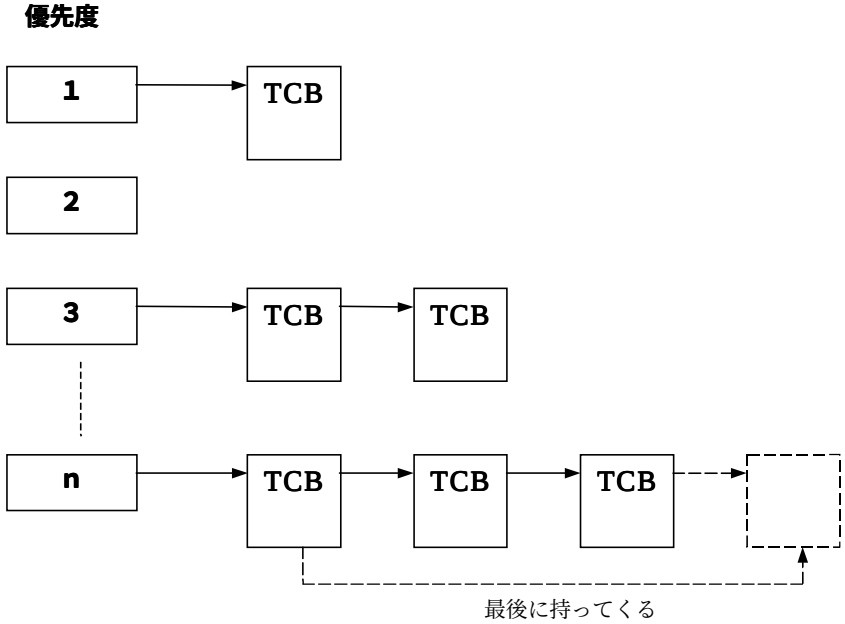


図 3.20 rot_rdq システムコールによるレディキューの操作

- タスクの待ち状態を強制解除する（rel_wai，irel_wai）
  タスクの待ち状態を強制的に解除します。解除される待ち状態は以下の条件により待ちに入ったタスクです。

- タイムアウト待ち状態
- slp_tsk システムコールによる (+タイムアウト有)待ち状態
- イベントフラグ (+タイムアウト有)待ち状態
- セマフォ(+タイムアウト有) 待ち状態
- メッセージ (+タイムアウト有)待ち状態

- 自タスクの ID を得る (get_tid)
  自タスクの ID 番号を得ます。ハンドラから発行した場合は、ID 番号の替わりに 0(ゼロ)が得られます。

- タスクの状態を参照する (ref_tsk)
  対象タスクの状態およびその優先度等を参照します。

3.5.4 タスク付属同期機能

タスク付属同期機能とは、タスク間の同期をとるためにタスクを待ち状態（もしくは強制待ち状態・二重待ち状態)にしたり、待ち状態になったタスクを起床させたりする機能です。

MR79 カーネルが提供するタスク付属同期システムコールには次のものがあります。

- タスクを強制待ち状態に移行する（sus_tsk，isus_tsk)
- 強制待ち状態のタスクを再開する（rsm_tsk，irsm_tsk)

  タスクの実行を強制的に待たせたり、実行を再開したりします。実行可能状態のタスクを強制待ちすれば強制待ち状態になり、待ち状態のタスクを強制待ちすれば二重待ち状態になります。（図 3.21参照)

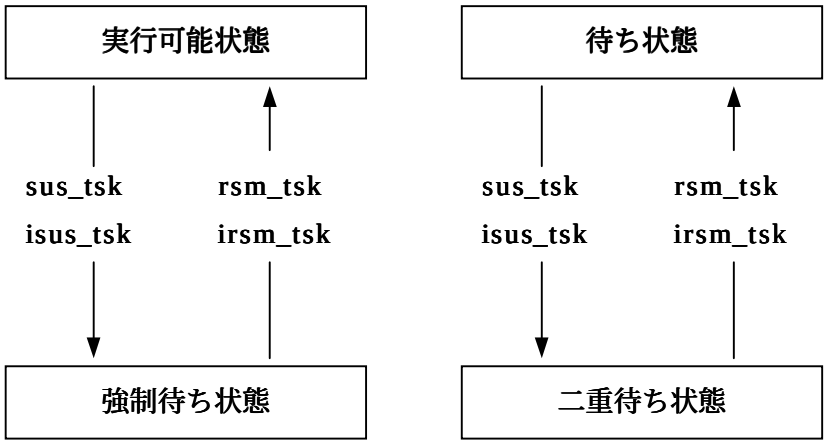


図 3.21 タスクの強制待ちと再開

- タスクを待ち状態に移行する（slp_tsk,tslp_tsk)
- 待ち状態のタスクを起床する（wup_tsk，iwup_tsk)

  slp_tsk、tslp_tsk システムコールにより待ち状態に入ったタスクを起床させます。

  slp_tsk、tslp_tsk システムコール以外の条件で待ち状態にあるタスクは起床できません。[34]

  slp_tsk、tslp_tsk システムコール以外の条件で待ちに入ったタスクや休止状態を除く他の状態のタスクに対して wup_tsk、iwup_tsk システムコールにより起床要求をおこなうと、この起床要求だけが蓄積されます。

  したがって、例えば実行状態のタスクに対して起床要求をおこなうと、この起床要求が一時的に記憶されます。そして、その実行状態のタスクが slp_tsk、tslp_tsk システムコールにより待ち状態に入ろうとした時、蓄積された起床要求が有効になり、待ち状態にならずに再び実行を続けます。（図 3.22参照)
- タスクの起床要求を無効にする（can_wup)

  蓄積された起床要求をクリアします。（図 3.23参照)

[34] 待ち状態であっても以下の条件で待っているタスクは起床されませんのでご注意ください。
- イベントフラグ待ち状態
- セマフォ待ち状態
- メッセージ待ち状態
- タイムアウト待ち状態

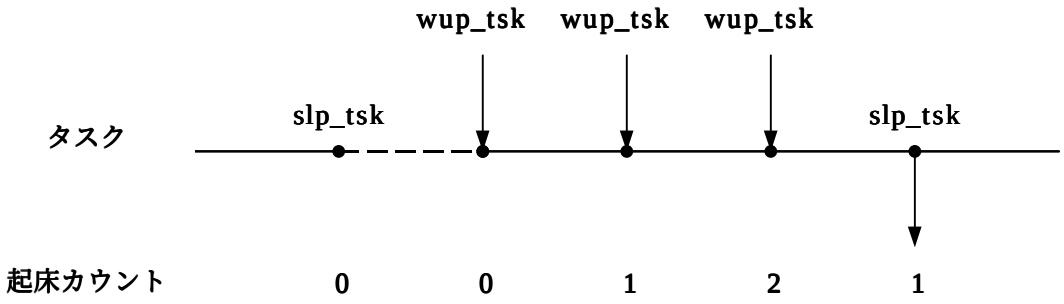


図 3.22 起床要求の蓄積

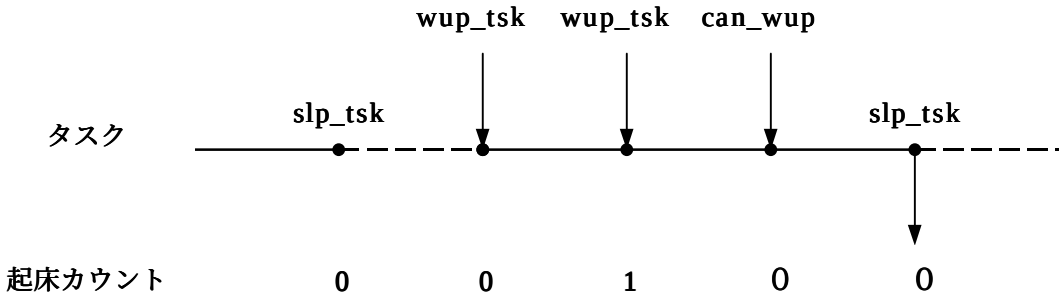


図 3.23 起床要求のキャンセル

3.5.5 同期・通信機能 (イベントフラグ)

イベントフラグは複数のタスクの実行の同期をとるための MR79 内部に持つ機構です。イベントフラグは、フラグ待ちパターンと 16 ビットのビットパターンによりタスクの実行制御をおこないます。タスクは、設定したフラグ待ちの条件が満たされるまで待ちます。

MR79 カーネルが提供するイベントフラグのシステムコールには次のものがあります。

- イベントフラグをセットする (set_flg, iset_flg)
  イベントフラグをセットします。これにより、このイベントフラグの待ちパターンを待っていたタスクは待ち解除されます。

- イベントフラグをクリアする (clr_flg)
  イベントフラグをクリアします。

- イベントフラグを待つ (wai_flg, twai_flg)
  イベントフラグがあるパターンにセットされるのを待ちます。イベントフラグを待つ時のモードは、 以下に示す 3 種類があります。
  - ♦ AND 待ち
    指定されたビットが全てセットされるのを待ちます。
  - ♦ OR 待ち
    指定されたビットの内いずれか 1 ビットがセットされるのを待ちます。
  - ♦ クリア指定
    AND 待ちか OR 待ちの条件が満たされた場合にフラグをクリアします。

- イベントフラグを得る (pol_flg)
  イベントフラグがあるパターンになっているか否かを調べます。このシステムコールではタスクは待ち状態に移行しません。

- イベントフラグの状態を得る (ref_flg)
  対象イベントフラグのビットパターンや待ちタスクの有無を参照します。

図 3.24に wai_flg と set_flg システムコールを使用したイベントフラグによるタスクの実行制御の例を示します。

イベントフラグは複数のタスクを一度に起床できるという特徴があります。

図 3.24では、タスク A からタスク F までの 6 個のタスクがつながっています。

そして、set_flg システムコールによって、フラグパターンを 0x0F にすると、待ち条件にあっているタスクがキューの前から順にはずされていきます。この図で待ち条件を満たすタスクはタスク A、タスク C、タスク E、タスク F です。このうち、タスク A、タスク C、タスク E はキューからはずされますが、タスク E は、クリア指定で待っていますので、タスク E がキューからはずされた時点でフラグがクリアされます。したがって、タスク F はキューからはずされません。

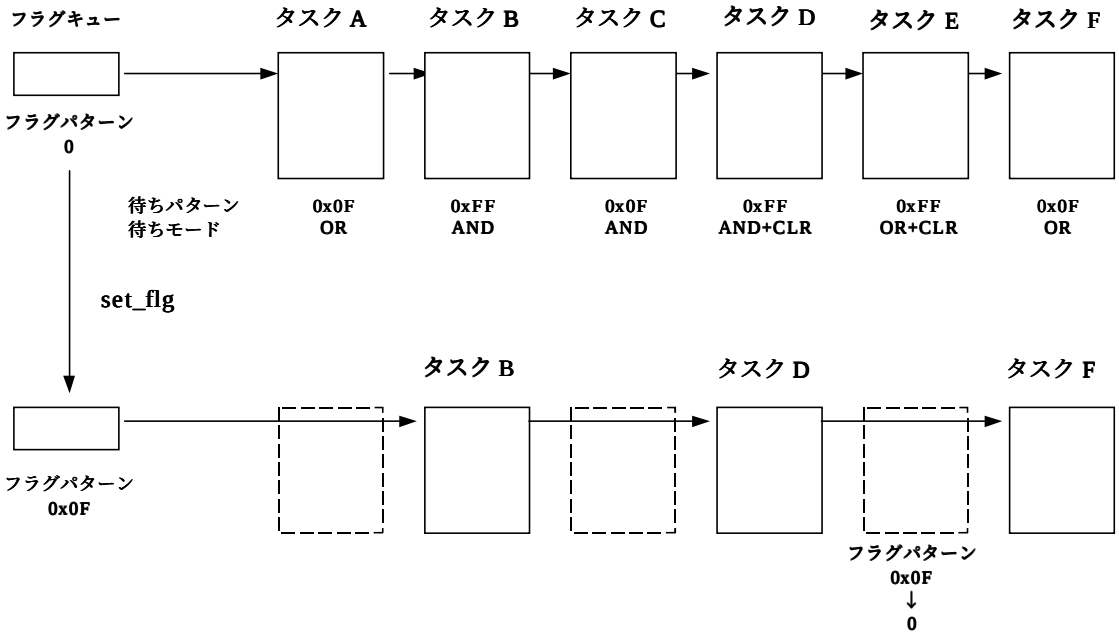


図 3.24 イベントフラグによるタスクの実行制御

3.5.6 同期・通信機能 (セマフォ)

セマフォは複数のタスクで共有する装置などの資源の競合を防ぐための機能です。例えば図 3.25に示すような場合、すなわち通信回線が３本しかないシステムに４つのタスクが回線を獲得しようと競合した場合に、通信回線を競合することなくタスクに接続することがセマフォを用いるとできます。

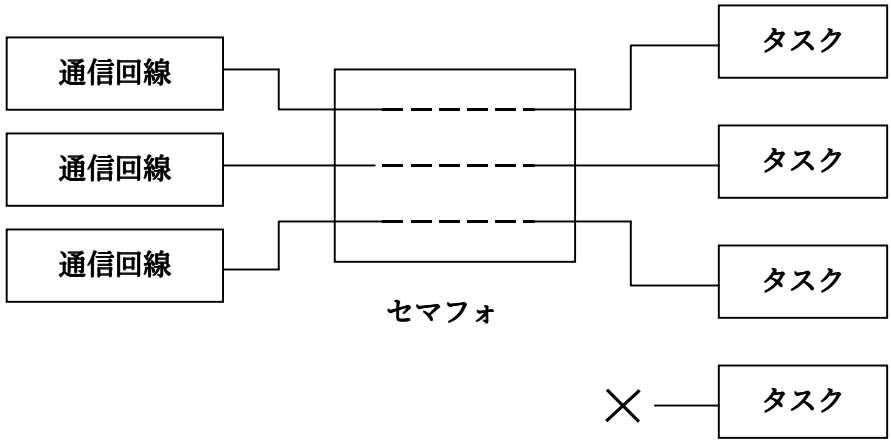


図 3.25 セマフォによる排他制御

セマフォは内部にセマフォカウンタと呼ばれる計数値を持っており、そのセマフォカウンタに基づきセマフォを獲得・解放をおこなうことによって資源の競合を防ぎます。 (図 3.26参照)

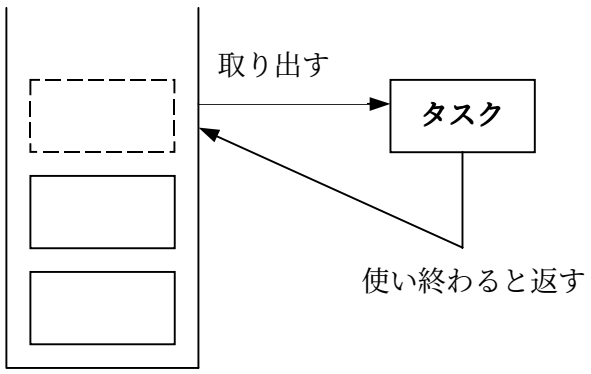


図 3.26 セマフォカウンタ

MR79 カーネルが提供するセマフォ同期のシステムコールには次のものがあります。

- セマフォに対する信号操作 (sig_sem, isig_sem)
  セマフォに信号をおくります。すなわち、セマフォを待っているタスクがあればそのタスクを起床し、なければセマフォカウンタを１増やします。

- セマフォ獲得操作 (wai_sem, twai_sem)
  セマフォを待ちます。セマフォカウンタが０であればセマフォを得ることができませんので待ち状態になります。

- セマフォ獲得操作（preq_sem）
  セマフォを得ます。得るべきセマフォがなければ待ち状態に入らずにエラーコードをかえします。

- セマフォの状態を参照する（ref_sem）
  対象セマフォの状態を参照します。対象セマフォのカウント値や待ちタスクの有無を参照します。

  wai_sem、sig_sem システムコールを用いたタスクの実行制御の例を図 3.27に示します。

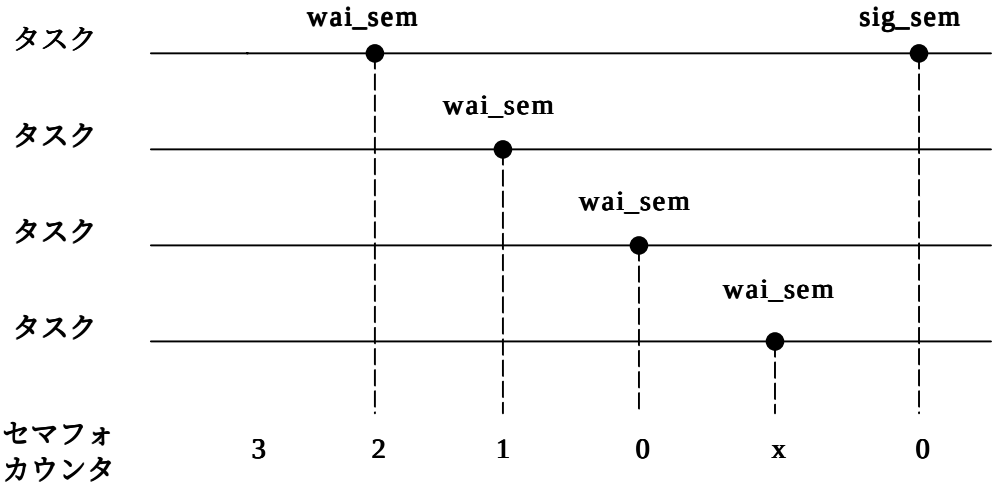


図 3.27 セマフォによるタスクの実行制御

3.5.7 同期・通信機能 (メイルボックス)

メイルボックスとはタスク間でデータの通信をおこなう機構です。例えば、図 3.28においてタスクAがメッセージをメイルボックスに投函しタスクBがそのメッセージをメイルボックスから取り出すことができます。

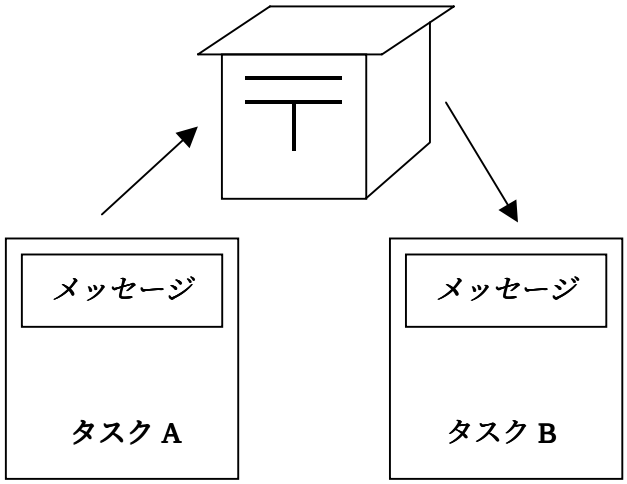


図 3.28 メイルボックス

このメイルボックスに投函できるメッセージは 16 ビットあるいは 24 ビットのデータです[35]。

MR79 では、このデータをメッセージパケットの先頭アドレスとして使用することを標準としていますが[36]、単なるデータとして使用することも可能です[37]。

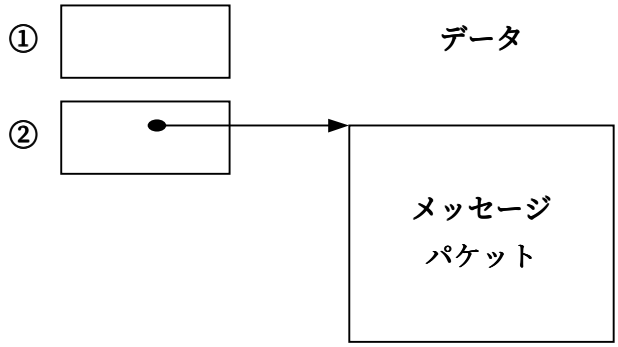


図 3.29 メッセージの意味

メイルボックスにはメッセージを蓄積する機能があります。蓄積されたメッセージは FIFO[38]でメッセージが取り出されます。ただし、メイルボックスに蓄積できるメッセージの数には制限があります。

このメイルボックスに蓄積できるメッセージの最大数をメッセージキューのサイズと呼びます。(図 3.30参照)

---

[35] メッセージサイズの指定は、コンフィグレーションファイルのシステム定義で行います。
[36] ITRON 仕様では、メッセージパケットの先頭アドレスとして使用することを標準としています。
[37] この場合、システムコールの引数であるデータにキャストして、ポインタに型変換しなければなりません
[38] ファーストインファーストアウト

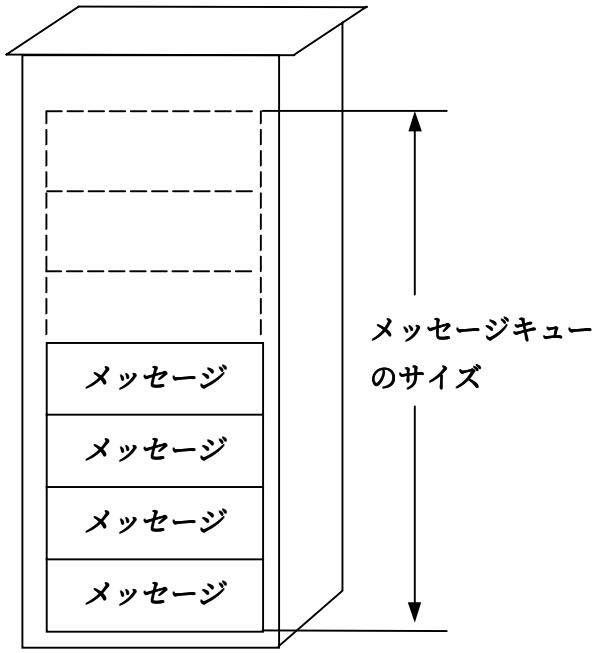


図 3.30 メッセージキューのサイズ

MR79 カーネルが提供するメイルボックスのシステムコールには次のものがあります。

- メッセージを送信します（snd_msg，isnd_msg）
  メッセージを送信します。すなわち、メッセージをメイルボックスに投函します。

- メッセージを受信します（rcv_msg，trcv_msg）
  メッセージを受信します。すなわち、メッセージをメイルボックスから取り出します。このときメッセージがメイルボックスに投函されていなければ、投函されるまで待ち状態になります。

- メッセージを受信します（prcv_msg）
  メッセージを受信します。rcv_msg システムコールと異なるのは、メイルボックスにメッセージがない場合は待ち状態にならずにエラーコードを返すところです。

- メイルボックスの状態を参照します（ref_mbx）
  対象メイルボックスにメッセージが入るのを待っているタスクの有無やメイルボックスに入っている先頭のメッセージを参照します。

3.5.8 割り込み管理機能

割り込み管理機能は外部割り込みの発生に対して、実時間で処理をおこなう機能を提供します。MR79 カーネルが提供する割り込み管理システムコールには次のものがあります。

- 割り込みハンドラから復帰します (ret_int)
  ret_int システムコールは、割り込みハンドラから復帰するとき、必要ならばスケジューラを起動し、タスク切り替えをおこないます。
  本機能は、C 言語を用いた場合[39]、ハンドラ関数の終了時に自動で呼び出されます。従って、この場合、本システムコールを呼び出す必要はありません。

- 割り込みおよびタスクのディスパッチを禁止します (loc_cpu)
  loc_cpu システムコールは、OS 依存の外部割り込みとタスクのディスパッチを禁止します。

- 割り込みおよびタスクのディスパッチを許可します (unl_cpu)
  unl_cpu システムコールは、OS 依存の外部割り込みとタスクのディスパッチを許可します。従って、loc_cpu システムコールによる割り込み、およびディスパッチの禁止状態がこのシステムコールの発行により解除されます。

図 3.31に割り込み処理の流れをしめします。なお、タスク選択からレジスタ復帰までの処理をスケジューラと呼びます。

[39] 割り込みハンドラが、#pragrma INTHANDLER で指定された場合

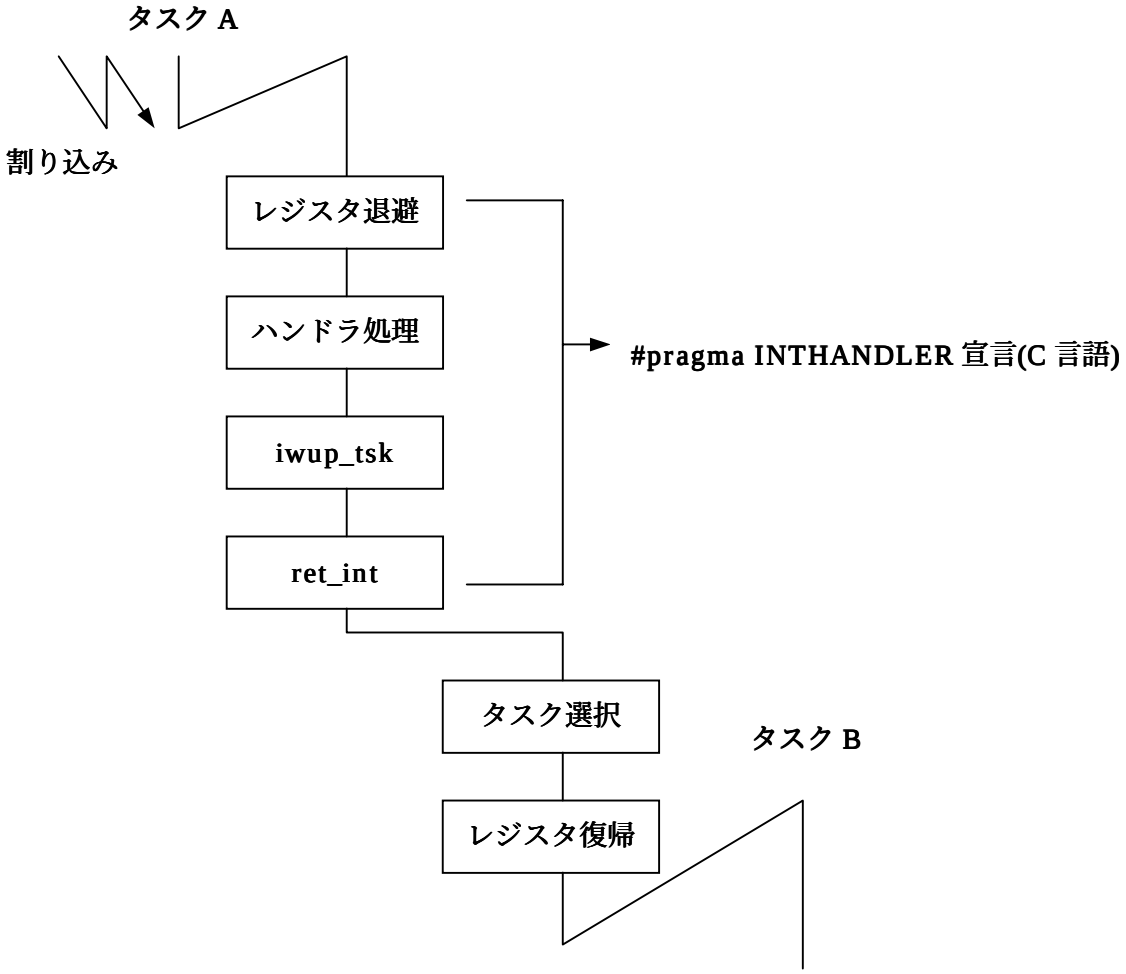


図 3.31 割り込み処理の流れ

3.5.9 メモリプール管理機能

メモリプール管理機能はシステムのメモリ空間 (RAM 空間) を動的に管理する機能を提供します。
特定のメモリ領域 (メモリプール)を管理し、そのメモリプールからタスクあるいはハンドラの必要とするメモリブロックを動的に確保し、また不用になったメモリブロックをメモリプールに解放します。
MR79 では、固定長と可変長の 2 つのメモリプール管理機能をサポートしています。

固定長メモリプール管理機能
メモリプールから獲得できるメモリブロックサイズが決まっていることを固定長といいます。
獲得するメモリブロックサイズは、コンフィグレーションファイルで指定します。
MR79 カーネルが提供する固定長メモリプール管理システムコールには次のものがあります。

- メモリブロックを獲得する (pget_blf)
- メモリブロックを解放する (rel_blf)

図 3.32で示されるようにタスクCからメモリブロックの獲得要求がだされるとメモリプールからメモリブロック 3 がタスク C に渡されます。メモリブロック 1、メモリブロック 2 はそれぞれタスク A、タスク B により使用されていると仮定しています。

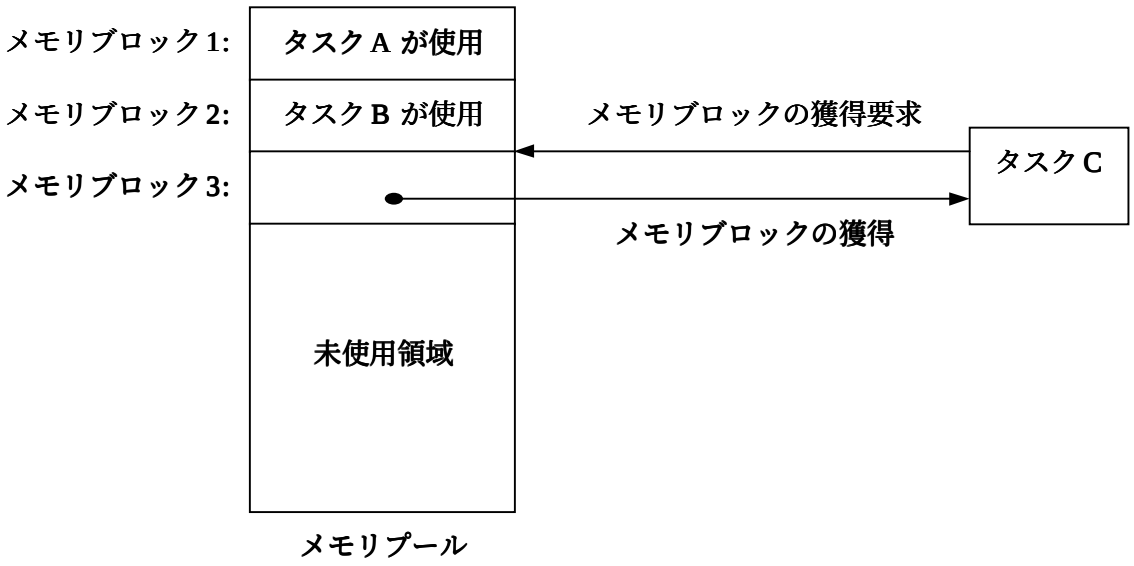


図 3.32 メモリプール管理

- メモリプールの状態を参照する (ref_mpf)
  対象メモリプールの空きブロック数やブロックサイズを参照します。

可変長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが任意に指定可能なことを可変長といいます。

MR79 では、メモリを 4 種類の固定長ブロックサイズで管理しています。

4 種類の各々のサイズは、ユーザーが獲得するメモリブロックの最大サイズから MR79 が計算します。メモリブロックの最大サイズは、コンフィグレーションファイルで指定します。

例

```
variable_memorypool[]{
    max_memsize = 400; <---- 最大獲得サイズ
    heap_size = 5000;
};
```

上記のように、可変長メモリプール定義を行った場合、4 種類の固定長ブロックサイズは、max_memsize 定義値から 56,112,224,448 となります。

ユーザーが要求したメモリは、指定サイズをもとに MR79 が計算を行い 4 種類の固定長メモリブロックサイズの中から最適なサイズを選択し、メモリを割り当てます。この 4 種類以外のサイズのメモリブロックを割り当てることはありません。

MR79 カーネルが提供する可変長メモリプール管理システムコールには次のものがあります。

- メモリブロックを獲得する (pget_blk)

  ユーザーが指定したブロックサイズは、4 種類のブロックサイズのうちから最適なブロックサイズに丸めて、丸めたサイズ分のメモリをメモリプールから獲得します。

  例えば、ユーザーが 200 バイトのメモリを要求した場合 224 バイトに丸めて、224 バイト分のメモリを獲得します。

  メモリが獲得できた場合、獲得したメモリの先頭アドレスとエラーコード E_OK を返します。 獲得できなかった場合には、エラーコード E_TMOUT を返します。

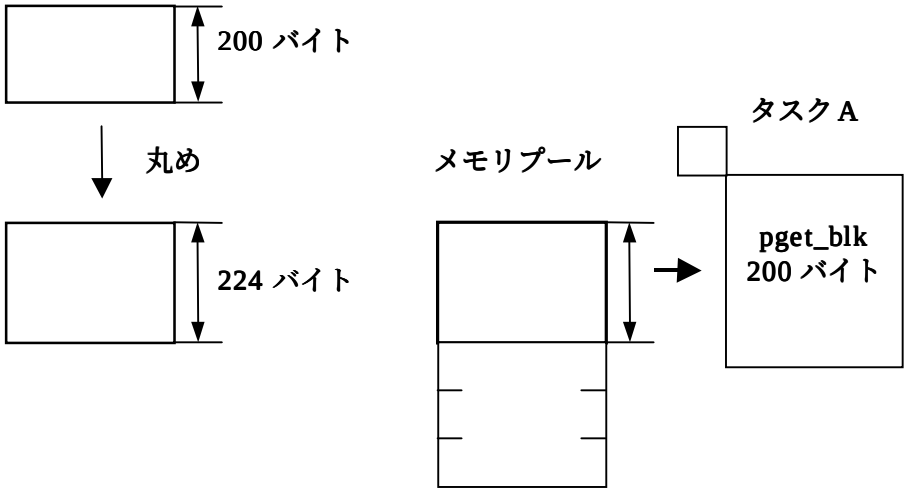


図 3.33 pget_blk 処理

- メモリブロックを解放する (rel_blk)

  pget_blk で獲得したメモリブロックを解放します。

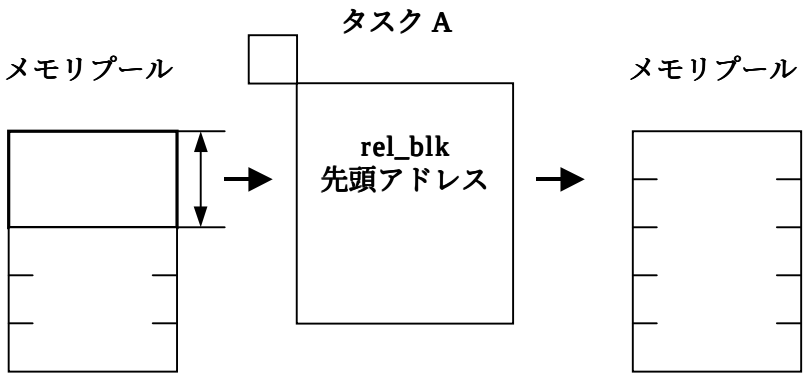


図 3.34 rel_blk 処理

- メモリプールの状態を参照する（ref_mpl）
  メモリプールの空き領域の合計サイズやすぐに獲得できる最大の空き領域のサイズを参照します。

3.5.10 時間管理機能

時間管理機能はシステムの時刻を管理し、時刻の読みだし[40]、時刻の設定[41]機能、タイムアウトの処理や特定時刻に起動するアラームハンドラや定期的に起動する周期起動ハンドラの機能を提供します。

MR79 カーネルはシステムタイマとして 7900 マイクロコンピュータの持つハードウェアタイマを一つ占有します。どのタイマを使用するかはコンフィグレーションファイルで設定します。

MR79 カーネルが提供する時間管理システムコールには次のものがあります。

- タスクを一定時間待ち状態に移行します (dly_tsk)

  タスクを一定時間待たせます。図 3.35に dly_tsk システムコールにより 10ms 間タスクの実行を待たせる例を示します。

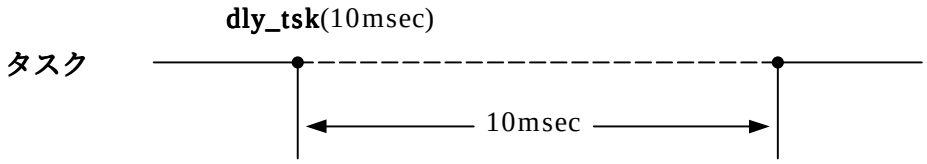


図 3.35 dly_tsk システムコール

- 待ち状態にタイムアウト値を指定すれば、一定時間待ち状態に移行します

  タスクを待ち状態に移行するシステムコール[42]にタイムアウトを指定することができます。システムコール名は、tslp_tsk、twai_flg、twai_sem、trcv_msg です。タイムアウトの指定時間が経過するまでに待ち解除条件が満たされない場合、エラーコード E_TMOUT を返し、待ち状態が解除されます。待ち解除条件が満たされた場合は、エラーコードは E_OK を返します。(図 3.36参照)

  タイムアウトの基準時間は、MR79 のシステムクロックの時間を基準としています。

[40] get_tim システムコール
[41] set_tim システムコール
[42] 強制待ち状態を除きます。

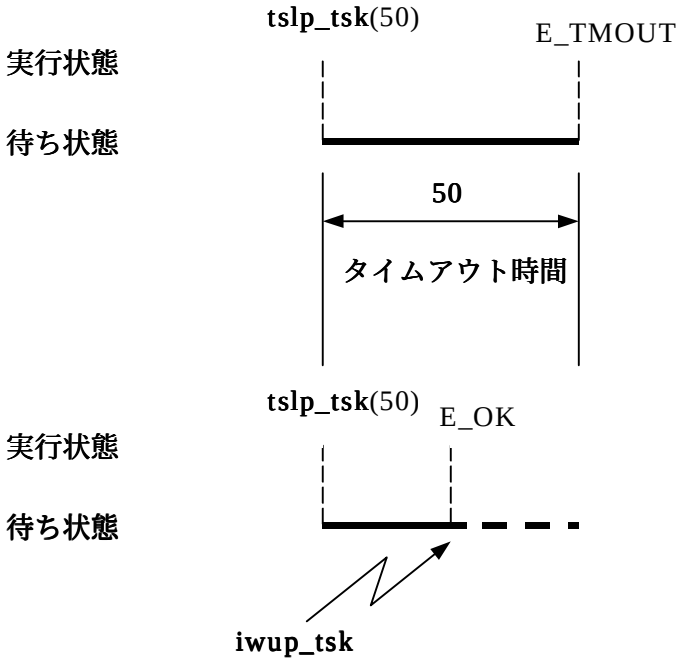


図 3.36 タイムアウト処理

- システム時刻を設定する (set_tim)

- システム時刻の値を読みだす (get_tim)
  システム時刻はリセット後のシステムクロック割り込みの発生回数をカウントし、48 ビットのデータで表します。

- 周期起動ハンドラの活性制御をおこなう (act_cyc)
  周期起動ハンドラは一定間隔で起動するプログラムです。(図 3.37参照) システムクロック割り込みの発生回数をカウントすることによって、一定時間で起動します。周期起動ハンドラは、システムコールで活性状態を指定することによって制御されます。例えば、TCY_ON を指定して活性状態を OFF から ON に変更したり (図 3.38参照) 、TCY_INI_ON を指定して、ハンドラのカウント値を初期化する (図 3.39参照) ことができます。

- 周期起動ハンドラの状態を参照する (ref_cyc)
  対象周期起動ハンドラの活性状態やつぎの起動までの残り時間を参照します。

- アラームハンドラの状態を参照する (ref_alm)
  対象アラームハンドラの起動までの残り時間を参照します。

なお、システムタイマは必須機能ではありません。したがって下記のシステムコールおよび時間管理機能を使用しなければ、タイマを MR79 用に占有する必要がありません。

1. システムクロックの設定、読みだし[43]
2. 周期起動ハンドラ
3. アラームハンドラ
4. dly_tsk システムコール
5. タイムアウト機能を使用したシステムコール

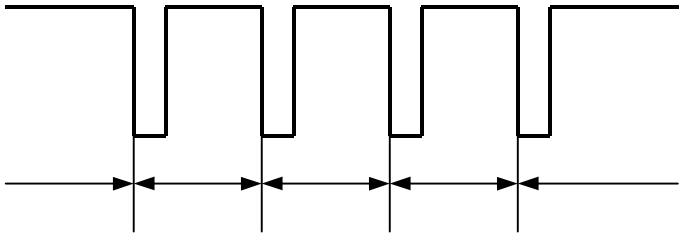


図 3.37 周期起動ハンドラ

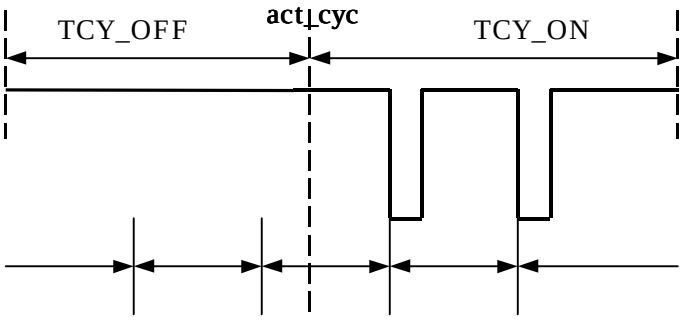


図 3.38 周期起動ハンドラ:活性状態 TCY_ON を指定

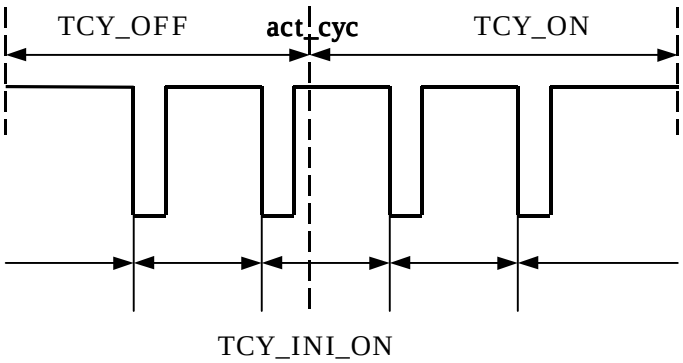


図 3.39 周期起動ハンドラ:活性状態 TCY_INI_ON を指定

[43] set_tim, get_tim システムコール

3.5.11 バージョン管理機能

MR79 のバージョンを get_ver システムコールにより得ることができます。

このバージョンは TRON 仕様で標準化された形式で得ることができます。このシステムコールでは以下の情報を得ることができます。

- **メーカー名**
  三菱電機株式会社を示す番号

- **形式番号**
  製品識別番号

- **仕様書バージョン**
  μITRON 仕様であることを表す番号とこの製品のもとになったμITRON 仕様書のバージョン番号

- **製品バージョン**
  MR79 のバージョン番号

- **製品管理情報**
  製品のリリース番号、リリース日等の情報

- **MPU 情報**
  7900 シリーズマイクロコンピュータを示す番号

- **バリエーション記述子**
  MR79 で利用できる機能番号

3.5.12 カーネルライブラリモデル

MR79 には以下の 4 種類のカーネルライブラリモデルが存在します。

- **LM モデル（ロングメッセージモデル）**
  メールボックス機能で扱うメッセージの長さが 24 ビットです。

- **SM モデル（ショートメッセージモデル）**
  メールボックス機能で扱うメッセージの長さが 16 ビットです。

- **LMI モデル（ロングメッセージモデル）**
  メッセージの長さは 24 ビット長で、OS 独立割り込み[44]に対して割り込み禁止時間が短いモデルです。ただし、各システムコールの処理時間や OS 依存割り込み[45]に対する処理時間は LM モデルより大きくなります。

- **SMI モデル（ショートメッセージモデル）**
  メッセージの長さは 16 ビット長で、OS 独立割り込みに対して割り込み禁止時間が短いモデルです。ただし、各システムコールの処理時間や OS 依存割り込みに対する処理時間は SM モデルより大きくなります。

これらのモデルのどれを使用するかは、コンフィグレーションファイルのシステム定義[46]に設定する内容により決まります。

[44] 5.3 節を参照
[45] 5.3 節を参照
[46] 6.1.2 項を参照

3.5.13 タスク、ハンドラから発行できるシステムコール一覧

システムコールにはタスクから発行できるものとハンドラから発行できるもの、その両方から発行できるものがあります。

表 3.2にその一覧を示します。

表 3.2 タスク、ハンドラから発行できるシステムコール一覧

システムコール	タスク	割り込みハンドラ	周期起動ハンドラ	アラームハンドラ
sta_tsk	○	×	×	×
ista_tsk	×	○	○	○
ext_tsk	○	×	×	×
ter_tsk	○	×	×	×
dis_dsp	○	×	×	×
ena_dsp	○	×	×	×
chg_pri	○	×	×	×
ichg_pri	×	○	○	○
rot_rdq	○	×	×	×
irot_rdq	×	○	○	○
rel_wai	○	×	×	×
irel_wai	×	○	○	○
get_tid	○	○	○	○
ref_tsk	○	○	○	○
sus_tsk	○	×	×	×
isus_tsk	×	○	○	○
rsm_tsk	○	×	×	×
irsm_tsk	×	○	○	○
slp_tsk	○	×	×	×
tslp_tsk	○	×	×	×
dly_tsk	○	×	×	×
wup_tsk	○	×	×	×
iwup_tsk	×	○	○	○
can_wup	○	○	○	○
set_flg	○	×	×	×
iset_flg	×	○	○	○
clr_flg	○	○	○	○
wai_flg	○	×	×	×
twai_flg	○	×	×	×
pol_flg	○	○	○	○
ref_flg	○	○	○	○

システムコール	タスク	割り込みハンドラ	周期起動ハンドラ	アラームハンドラ
sig_sem	○	×	×	×
isig_sem	×	○	○	○
wai_sem	○	×	×	×
twai_sem	○	×	×	×
preq_sem	○	○	○	○
ref_sem	○	○	○	○
snd_msg	○	×	×	×
isnd_msg	×	○	○	○
rcv_msg	○	×	×	×
trcv_msg	○	×	×	×
prcv_msg	○	○	○	○
ref_mbx	○	○	○	○
pget_blf	○	○	○	○
rel_blf	○	○	○	○
ref_mpf	○	○	○	○
pget_blk	○	×	×	×
rel_blk	○	×	×	×
ref_mpl	○	○	○	○
ret_int	○	○[47]	×	×
loc_cpu	○	×	×	×
unl_cpu	○	×	×	×
set_tim	○	○	○	○
get_tim	○	○	○	○
act_cyc	○	○	○	○
ref_cyc	○	○	○	○
ref_alm	○	○	○	○
get_ver	○	○	○	○
vrst_msg	○	×	×	×
vrst_blf	○	×	×	×
vrst_blk	○	×	×	×

[47] C 言語を用いて記述された割り込みハンドラからは発行できません。

第 4 章　　アプリケーション作成手順概要

4.1 概要

MR79 のアプリケーションプログラムは一般的に以下に示す手順で開発します。

1. **アプリケーションプログラムのコーディング**
   C 言語もしくはアセンブリ言語を用いてアプリケーションプログラムをコーディングします。
   この時サンプルスタートアッププログラム "crt0mr.a79"もしくは "start.a79"、およびセクション定義ファイル "c_sec.inc"もしくは "asm_sec.inc"を環境変数"LIB79" の示すディレクトリからカレントディレクトリにコピーしておいてください。[48]また必要があればスタートアッププログラム、セクション定義ファイルを修正してください。

2. **コンフィグレーションファイル作成**
   タスクのエントリーアドレスやスタックサイズなどを定義したコンフィグレーションファイルをエディタを用いて作成します。

3. **コンフィグレータ実行**
   コンフィグレーションファイルからシステムデータ定義ファイル (sys_rom.inc、sys_ram.inc)、インクルードファイル (mr79.inc、id.h)およびシステム生成手順記述ファイル (makefile) を作成します。

4. **システム生成**
   make[49]コマンドを実行してシステムを生成します。

5. ROM **書き込み**
   作成された ROM 書き込み形式ファイルにより、ROM に書き込みます。もしくはデバッガに読み込ませてデバッグを行います。

図 4.1にシステム生成の詳細フローを示します。

[48] 標準のスタートアッププログラム"crt0mr.a79","start.a79"、およびセクション定義ファイル"c_sec.inc","asm_sec.inc“ は環境変数"LIB79"で示されたディレクトリにあります。
[49] Make コマンドは、UNIX 標準もしくは準拠のコマンドのみ使用可能です。

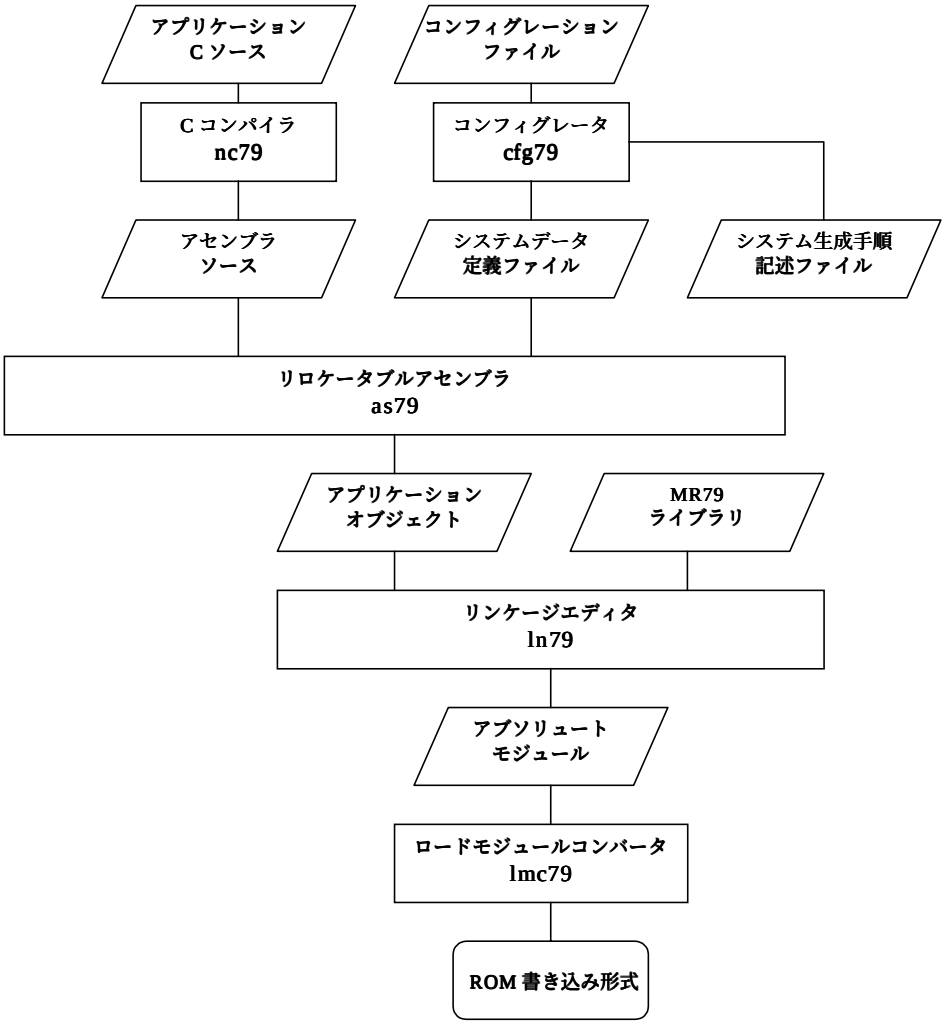


図 4.1　MR79 システム生成詳細フロー

4.2 開発手順例

この節では MR79 のアプリケーション例をもとに開発手順の概要について説明します。

4.2.1 アプリケーションプログラムのコーディング

図 4.2にレーザビームプリンタの動作をシミュレーションするプログラムを示します。このレーザービームプリンタのシミュレーションプログラムを記述したファイルの名前を"lbp.c"とします。このプログラムは以下の３つのタスクと１つの割り込みハンドラから構成されます。

- メインタスク
- イメージ展開タスク
- プリンタエンジンタスク
- セントロニクスインターフェース割り込みハンドラ

このプログラムでは MR79 ライブラリのなかの以下の機能を利用します。

- sta_tsk()
  タスク起動をおこないます。引き数は起動すべきタスクを選択するための ID 番号を与えます。ＩＤ番号はコンフィグレータの生成するファイル"id.h"をインクルードすることにより名前 (文字列)でタスクを指定することができます。[50]
- wai_flg()
  イベントフラグが立つまで待ちます。この例ではセントロニクスインターフェースから１ページ分データがバッファに溜まるのを待つために使用しています。
- wup_tsk()
  指定タスクを待ち状態から起床します。プリンタエンジンタスクを起動するために使用しています。
- slp_tsk()
  タスクを実行状態から待ち状態にします。この例ではプリンタエンジンタスクをイメージ展開待ちにするため使用しています。
- iset_flg()
  イベントフラグを立てます。この例では、１ページ分のデータ入力完了をイメージ展開タスクに知らせるために使用しています。
  また、この時スタートアッププログラム"crt0mr.a79"、セクション定義ファイル"c_sec.inc"をカレントディレクトリにコピーしておいてください。例えば、

```
% cp $LIB79/crt0mr.a79 .
% cp $LIB79/c_sec.inc .
```

[50] すなわち、コンフィグレータがコンフィグレーションファイルに記述されている情報をもとに ID 番号を名前（文字列）に置き換えます。

```
 #include "id.h"

 void main() /* main task */
 {
    printf("LBP シミュレーション開始¥n");
    sta_tsk(ID_image,1);    /* イメージ展開タスク起動 */
    sta_tsk(ID_printer,1);  /* プリンタエンジンタスク起動 */
}
void image() /* イメージ展開タスク */
{
    while(1){
            wai_flg(&flgptn,ID_pagein,waiptn,TWF_ANDW+TWF_CLR);/* 1ページ入力
待ち */

            printf("ビットマップ展開処理¥n");
            wup_tsk(ID_printer);    /* プリンタエンジンタスク起床 */
    }
}
void printer() /* プリンタエンジンタスク */
{
    while(1){
            slp_tsk();
            printf("プリンタエンジン動作¥n");
    }
}
void sent_in() /* セントロニクスインターフェースハンドラ */
{


    /* セントロニクスインターフェースからの入力処理 */
    if ( /* 1ページ入力完了 */ )
            iset_flg(ID_pagein,setptn);
}
```

図 4.2　プログラム例

4.2.2 コンフィグレーションファイル作成

タスクのエントリーアドレスやスタックサイズなどを定義したコンフィグレーションファイルを作成します。図 4.3にレーザービームプリンタシュミレーションプログラムのコンフィグレーションファイル（ファイル名 "lbp.cfg"）を示します。

```
// System Definition
system{
    stack_size              = 1024;
    priority                = 4;
    system_IPL              = 4;
};
//System Clock Definition
clock{
    mpu_clock               = 20MHz;
    timer                   = A0;
    IPL                     = 4;
    unit_time               = 10ms;
    initial_time            = 0:0:0;
};
//Task Definition
task[1]{
    entry_address           = main();
    stack_size              = 512;
    priority                = 1;
    initial_start           = ON;
};
task[2]{
    entry_address           = image();
    stack_size              = 512;
    priority                = 2;
};
task[3]{
    entry_address           = printer();
    stack_size              = 512;
    priority                = 4;
};
//Eventflag Definition
flag[1]{
     name                   = pagein;
};
//Interrupt Vector Definition
interrupt_vector[9]{
    os_int                  = YES;
    entry_address           = sent_in();
};
```

図 4.3 コンフィグレーションファイル例

4.2.3 コンフィグレータ実行

コンフィグレータ cfg79 を実行して、コンフィグレーションファイルからシステムデータ定義ファイル（sys_rom.inc，sys_ram.inc）、インクルードファイル（mr79.inc，id.h）
およびシステム生成手順記述ファイル（makefile）を作成します。

```
A> cfg79  -mv  lbp.cfg

MR79 system configurator V.2.00.00
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION,
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION,
All Rights Reserved
MR79 version ==> V.2.00 Release 1

A>
```

図 4.4　コンフィグレータ実行

4.2.4 システム生成

make コマンド[51]を実行してシステムを生成します。

```
A> make -f  makefile
as79 -F -Dtest=1 crt0mr.a79
nc79 -c task.c
ln79 @ln79.sub

A>
```

図 4.5　システム生成

4.2.5 ROM 書き込み

ロードモジュールコンバータ LMC79 により、アブソリュートモジュールファイルを ROM 書き込み形式に変換し、ROM に書き込みます。もしくは、デバッガに読み込ませてデバッグをおこないます。

[51] make コマンドは MS-DOS 標準のものと、UNIX 標準もしくは準拠のものとがあります。MR79 では、UNIX 標準もしくは UNIX 準拠の make コマンドのみ使用できます。MS-DOS を使用する場合は、UNIX 互換の make コマンドを使用してください。UNIX 互換の make コマンド対応状況については、リリースノートを参照ください。本章では、UNIX 互換の make コマンドを実行する場合を例として説明します。

第 5 章　　アプリケーション作成手順詳細

5.1 C 言語によるコーディング方法

本節では、C 言語を用いてアプリケーションプログラムを記述する方法について述べます。

5.1.1 タスクの記述方法

C 言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **タスクは関数として記述します。**
   そのタスクを MR79 に登録するにはコンフィグレーションファイルに関数名を記述します。例えば関数名 "task()" をタスク ID 番号 3 で登録するには以下のようにおこないます。

```
task[3]{
        entry_address   = task();
        stack_size      = 100;
        priority        = 3;
};
```

2. **ファイル先頭で必ずカレントディレクトリ内の "id.h" をインクルードしてください。すなわちファイルの先頭で以下の 1 行を必ず記述してください。**

```
#include "id.h"
```

3. **タスク開始関数の戻り値はありません。したがって、void 型で宣言してください。**
4. **スタティック宣言をおこなった関数はタスクとして登録できません。**
5. **タスク開始関数の出口で ext_tsk()を記述する必要はありません。[52]タスク開始関数から呼び出した関数でタスクを終了する場合は、ext_tsk システムコールを発行してください。**
6. **タスクの開始関数を無限ループで記述することも可能です。**

```
#include "id.h"

void task(void)
{
        /* 処理 */

}
```

図 5.1 C 言語で記述したタスクの例

[52] MR79 では、#pragma TASK 宣言をおこなうことで、自動的に ext_tsk()で終了します。関数の途中で return 文により戻る場合も同様に ext_tsk()で終了処理をおこないます。

```
#include "id.h"

void task(void)
{
        for(;;){
                /* 処理 */

        }
}
```

図 5.2　C 言語で記述した無限ループタスクの例

7. **タスクを指定する場合はタスクの関数名に** "ID_" **を追加した文字列で指定してください**[53]**。**

```
wup_tsk(ID_main);
```

8. **イベントフラグ、セマフォ、メイルボックスを指定する場合は、コンフィグレーションファイルで定義したそれぞれの名前に**"ID_"**を追加した文字列で指定してください。**

例えば、コンフィグレーションファイルで以下のようにイベントフラグを定義した場合は、

```
flag[1]{
        name    = abc;
};
```

このイベントフラグを指定するには以下のようにおこなってください。

```
set_flg(ID_abc,&setptn);
```

9. **周期起動ハンドラ、アラームハンドラを指定する場合は、そのハンドラの開始関数名に** "ID_"**を追加した文字列で指定してください。例えば、周期起動ハンドラ** "cyc()" **を指定する場合は、以下のようにおこなってください。**

```
act_cyc(ID_cyc,TCY_ON);
```

10. **タスクを** ter_tsk() **システムコールなどで終了した後で** sta_tsk() **システムコールで再起動した場合は、タスク自身は初期状態から開始しますが**[54]**外部変数、スタティック変数はタスクの開始にともなっての初期化はされません。外部変数、スタティック変数の初期化は** MR79 **が立ち上がる前に起動されるスタートアッププログラム** (crt0mr.a79、start.a79) **でのみおこないます。**

11. MR79 **システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。**

12. **変数の記憶クラスについて**

C 言語の変数は MR79 から見て表 5.1に示す扱いになります。

[53] コンフィグレータがタスクの ID 番号をタスクを指定するための文字列に変換するためのファイル"id.h"を生成します。すなわち、タスクの開始関数名に"ID_"を付加した文字列をそのタスクの ID 番号に変換するための#define 宣言を"id.h"で行います。

[54] タスクの開始関数から初期優先度でなおかつ起床カウントがクリアされた状態で開始します。

表 5.1　C 言語における変数の扱い

変数の記憶クラス	扱い
グローバル変数	すべてのタスクの共有変数
関数外のスタティック変数	同一ファイル内のタスクの共有変数
オート変数 レジスタ変数 関数内のスタティック変数	タスク固有の変数

5.1.2 OS 依存割り込みハンドラの記述方法

C 言語を用いて OS 依存割り込みハンドラを記述する場合、以下の点に注意してください。

1. **OS 依存割り込みハンドラは関数として記述します。**[55]
2. **割り込みハンドラ開始関数の戻り値および引き数は、必ず** void **型で宣言してください。**
3. **ファイルの先頭で必ずカレントディレクトリ内の**"id.h"**をインクルードしてください。**
4. **関数の最後に** ret_int **システムコールは記述しないでください。**[56]
5. **スタティック宣言をおこなった関数は割り込みハンドラとしては登録できません。**

```
#include "id.h"

void inthand(void)

{
        /* 処理 */

        iwup_tsk(ID_main);
}
```

図 5.3 C 言語で記述した割り込みハンドラの例

[55] ハンドラと関数名との対応はコンフィグレーションファイルにより行います。
[56] OS 依存割り込みハンドラを#pragma INTHANDLER で宣言すると、自動的に ret_int システムコールを生成します。

5.1.3 OS 独立割り込みハンドラの記述方法

C 言語を用いて OS 独立割り込みハンドラを記述する場合、以下の点に注意してください。

1. **割り込みハンドラ開始関数の戻り値および引き数は、必ず** void **型で宣言してください。**
2. OS **独立割り込みハンドラからは、システムコールは発行できません。**
   (注)システムコールを発行した場合は不正動作をするので十分注意してください。
3. **スタティック宣言をおこなった関数は割り込みハンドラとしては登録できません。**
4. OS **独立割り込みハンドラの中で多重割り込みを許可する場合は、必ず、**OS **独立割り込みハンドラの割り込み優先レベルを、他の** OS **依存割り込みハンドラの割り込み優先レベルより高くしてください。**[57]

```
#include "id.h"

void inthand(void)
{
        /* 処理 */
}
```

図 5.4 OS 独立割り込みハンドラの例

[57] OS 独立割り込みハンドラの割り込みレベル優先レベルを、OS 依存割り込みハンドラの割り込み優先レベルより低くしたい場合は、OS 独立割り込みハンドラを OS 依存割り込みハンドラの記述に変更してください。

5.1.4 周期起動ハンドラ、アラームハンドラの記述方法

C 言語を用いて周期起動ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **周期起動ハンドラおよびアラームハンドラは関数として記述します。**[58]
2. **関数の戻り値および引き数を、**void **型で宣言してください。**
3. **ファイルの先頭で必ずカレントディレクトリ内の** "id.h"**をインクルードしてください。**
4. **スタティック宣言をおこなった関数は周期起動ハンドラおよびアラームハンドラとしては登録できません。**
5. **周期起動ハンドラおよびアラームハンドラはシステムクロックの割り込みハンドラからサブルーチン呼び出しにより起動されます。**

```
#include "id.h"

void cychand(void)
{
        /* 処理 */
}
```

図 5.5 C 言語で記述した周期起動ハンドラの例

[58] ハンドラと関数名との対応は、コンフィグレーションファイルにより行います。

5.2 アセンブリ言語によるコーディング方法

本節では、アセンブリ言語を用いてアプリケーションを記述する方法について述べます。

5.2.1 タスクの記述方法

アセンブリ言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず** "mr79.inc"**をインクルードしてください。**
2. **タスクの開始アドレスを示すシンボルは外部シンボル宣言**[59]**をおこなってください。**
3. **タスクは無限ループか** ext_tsk **システムコールで終了してください。**

```
        .INCLUDE  mr79.inc ----- (1)
        .GLB      task     ----- (2)

task:
                ; 処理
        jmp  task          ----- (3)
```

図 5.6 アセンブリ言語で記述した無限ループタスクの例

```
        .INCLUDE  mr79.inc
        .GLB      task

task:
                ; 処理
        ext_tsk
```

図 5.7 アセンブリ言語で記述した ext_tsk で終了するタスクの例

4. **タスク起動時のレジスタの初期値は、**DT**、**PS**、**PG**、**PC **レジスタ以外は** 0 **で初期化します。**
5. **タスクを指定する場合はタスクの開始シンボル名に**"ID_"**を追加した文字列で指定してください。**[60]

```
        wup_tsk  ID_task
```

6. **イベントフラグ、 セマフォ、メイルボックスを指定する場合は、コンフィグレーションファイルで定義したそれぞれの名前に**"ID_"**を追加した文字列で指定してください。**

   例えば、 コンフィグレーションファイルで以下のようにセマフォを定義した場合、

```
semaphore[1]{
        name            = abc;
};
```

   このセマフォを指定するには以下のようにおこなってください。

```
        sig_sem  ID_abc
```

7. **周期起動ハンドラ、アラームハンドラを指定する場合は、そのハンドラの開始シンボル名**

---

[59] .GLB 疑似命令を使用してください。

[60] コンフィグレータがタスクの ID 番号を、タスクを指定するための文字列に変換するための命令をファイル"mr79.inc"に生成します。すなわち、タスクの開始シンボル名に"ID_"を付加した文字列をそのタスク ID 番号に変換するための.EQU 宣言を"mr79.inc"でおこないます。

に"ID_"を追加した文字列で指定してください。例えば、周期起動ハンドラ"cyc"を指定する場合は、以下のようにおこなってください。

```
act_cyc  ID_cyc,TCY_ON
```

8. MR79 **システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。**[61]

[61] このタスク ID 番号とタスク(プログラム)との対応づけは、コンフィグレーションファイルにより行います。

5.2.2 OS 依存割り込みハンドラの記述方法

アセンブリ言語を用いて OS 依存割り込みハンドラを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず**"mr79.inc"**をインクルードしてください。**
2. **割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）**[62]**をおこなってください。**
3. **割り込みハンドラ内で** DT **レジスタを変更しないでください。変更した場合は、システムコール発行前に** _SDT **に戻してください。**

```
（例）   ldt   #__SDT
        irsm_tsk  2
```

4. **割り込みハンドラは０バンク内に記述してください。**
5. **割り込み禁止時間短縮機能を使用しない場合、割り込みハンドラの先頭で** SaveRegs **マクロと** IntEntry **マクロを必ず呼び出してください。**
6. **割り込み禁止時間短縮機能を使用する場合、割り込みハンドラの先頭で** change_IPL **マクロ、**cli **命令、**SaveRegs **マクロ、**IntEntry **マクロを順次必ず呼び出してください。**
7. ret_int **システムコールにて復帰してください。**

```
        .INCLUDE  mr79.inc              ------(1)
        .GLB     inth                   ------(2)

inth:
        SaveRegs                        ------(5)
        IntEntry
                :
                処    理
                :
        ret_int                         ------(7)
```

図 5.8　割り込み禁止時間短縮機能を使用しない OS 依存割り込みハンドラの例

```
        .INCLUDE  mr79.inc              ------(1)
        .GLB     inth                   ------(2)

inth:
        change_IPL                      ------(6)
        cli
        SaveRegs
        IntEntry
                :
                処    理
                :
        ret_int                         ------(7)
```

図 5.9　割り込み禁止時間短縮機能を使用する OS 依存割り込みハンドラの例

[62] .GLB 疑似命令を使用してください。

5.2.3 OS 独立割り込みハンドラ記述方法

1. **割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）してください。**
2. **ハンドラ内で使用するレジスタは入り口でセーブし、使用後復帰してください。**
3. RTI **命令で終了してください。**
4. OS **独立割り込みハンドラからは、システムコールは発行できません。**
   (注)システムコールを発行した場合は不正動作をするので十分注意してください。

5. OS **独立割り込みハンドラ内で多重割り込みを許可する場合は、**OS **独立割り込みハンドラの割り込み優先レベルは、他の** OS **依存割り込みハンドラの割り込み優先レベルより必ず高くしてください。**[63]

```
        .GLB    inthand                 ----- (1)

inthand:
        ; 使用レジスタ退避              ----- (2)
        ; 割り込み処理
        ; 使用レジスタ復帰              ----- (2)
        RTI                             ----- (3)
```

図 5.10 特定レベルの OS 独立割り込みハンドラの例

[63] OS 独立割り込みハンドラの割り込み優先レベルを、OS 依存割り込みハンドラの割り込み優先レベルより低くしたい場合は、OS 独立割り込みハンドラを OS 依存割り込みハンドラの記述に変更してください。

5.2.4 周期起動ハンドラ、アラームハンドラの記述方法

アセンブリ言語を用いて周期起動ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **ファイルの先頭で必ず**"mr79.inc"**をインクルードしてください。**
2. **ハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）**[64]**をおこなってください。**
3. **周期起動ハンドラ、アラームハンドラは全て** RTS **命令（サブルーチンリターン命令）にて復帰してください。**
4. **周期起動ハンドラ、アラームハンドラ内で** DT **レジスタを変更しないでください。変更した場合は、システムコール発行前および** RTS **命令の前に** _SDT **に戻してください。**

```
（例）   ldt    #__SDT
        irsm_tsk   2
```

5. **周期起動ハンドラ、アラームハンドラは、０バンク内に記述してください。**

```
        .INCLUDE        mr79.inc        ----- (1)
        .GLB            cychand         ----- (2)

cychand:
                :
                ハンドラ処理
                :

        rts                             ----- (3)
```

図 5.11 アセンブリ言語で記述したハンドラの例

[64] .GLB 疑似命令を使用してください。

5.3 割り込みについて

MR79 の割り込みハンドラには、OS 依存割り込みハンドラと OS 独立割り込みハンドラを定義しています。それぞれの割り込みハンドラの定義を以下に示します。

- OS 依存割り込みハンドラ
  以下の 2 つの条件のうち、どちらか一方を満たすものを OS 依存割り込みハンドラとして定義します。
  - ♦ システムコールを発行する割り込みハンドラ[65]
  - ♦ システムコールを発行する割り込みハンドラが多重ではいる割り込みハンドラ

  OS 依存割り込みハンドラの IPL 値は、OS 割り込み禁止レベル（system.IPL)以下（IPL=0 ～ system.IPL)[66]にする必要があります。

- OS 独立割り込みハンドラ
  以下の 2 つの条件の両方を満たすものを OS 独立割り込みハンドラとして定義します。
  - ♦ システムコールを発行しない割り込みハンドラ
  - ♦ システムコールを発行する割り込みハンドラ、および、システムクロック割り込みハンドラが多重割り込みではいらないもの
  - ♦ 割り込みを禁止できない NMI（ノンマスカブル割り込み）[67]

  OS 独立割り込みハンドラの IPL 値は、（system.IPL+1）～ 7 にする必要があります。すなわち、OS 独立割り込みハンドラの IPL 値を OS 割り込み禁止レベル以下に設定できません。

図 5.12に、OS 割り込み禁止レベルを 3 にした場合の、OS 依存割り込みハンドラと OS 独立割り込みハンドラの関係を示します。

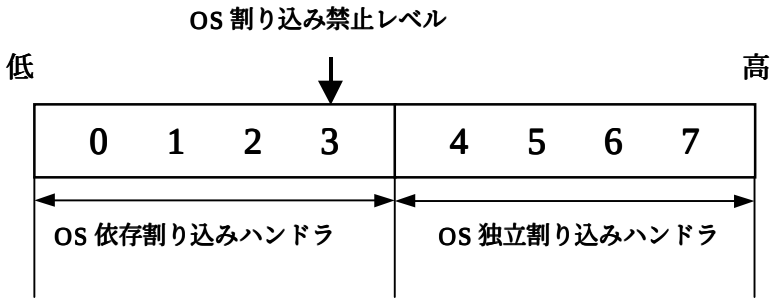


図 5.12 割り込みハンドラの IPL

[65] コンフィグレーションファイルの割り込みベクタ定義で os_int=YES にすると、割込みハンドラの最後で自動的に ret_int システムコールが発行されます。
[66] system.IPL は、コンフィグレーションファイルで設定します。
[67] NMI でシステムコールを発行した場合は正常に動作しません。

5.3.1 割り込み制御方法

割り込み禁止時間短縮機能を使用しない場合（LM, SM モデル)、MR79 の内部では、割り込み禁止制御を I フラグのみでおこないます。

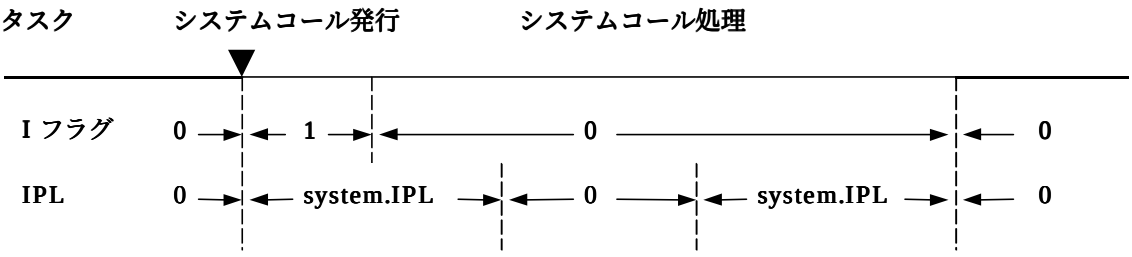
割り込み禁止時間短縮機能を使用する場合（LMI, SMI モデル)、システムコール内の割り込み禁止/許可の制御は、IPL の操作によりおこなっています。

システムコール内での IPL 値は、OS 割り込み禁止レベル (system.IPL)にして、OS 依存割り込みハンドラの割り込みを禁止しています。全ての割り込みを許可できる箇所では、システムコール発行時の IPL 値に戻します。

図 5.13に、LMI,SMI モデルの場合のシステムコール内での割り込み禁止フラグと IPL の状態を示します。

- タスクからのみ発行できるシステムコールの場合

・システムコール発行前の I フラグが 0(割り込み許可)の場合



・システムコール発行前の I フラグが 1（割り込み禁止）の場合

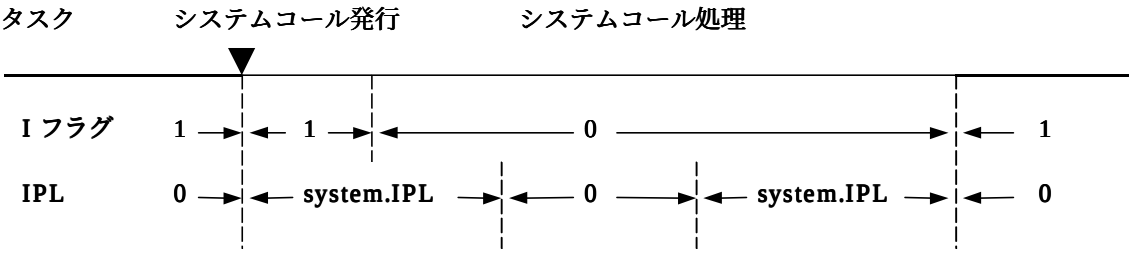
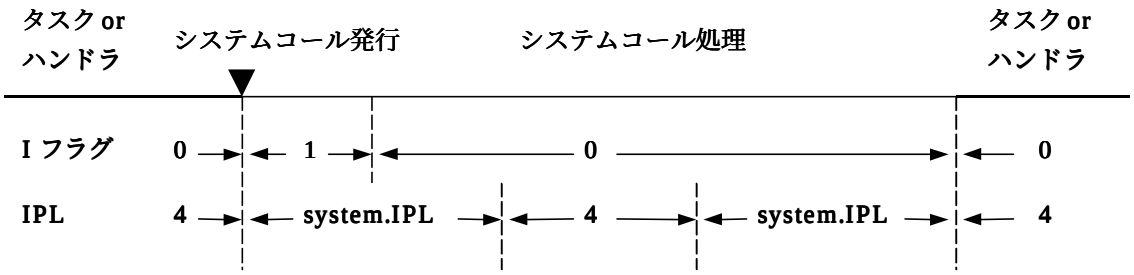


図 5.13 タスクからのみ発行できるシステムコール内での割り込み制御

- タスク独立部からのみ発行できるシステムコール、もしくは、タスク独立部とタスクの両方から発行できるシステムコールの場合

・システムコール発行前のI フラグが0（割り込み許可）の場合



・システムコール発行前のI フラグが1（割り込み禁止）の場合

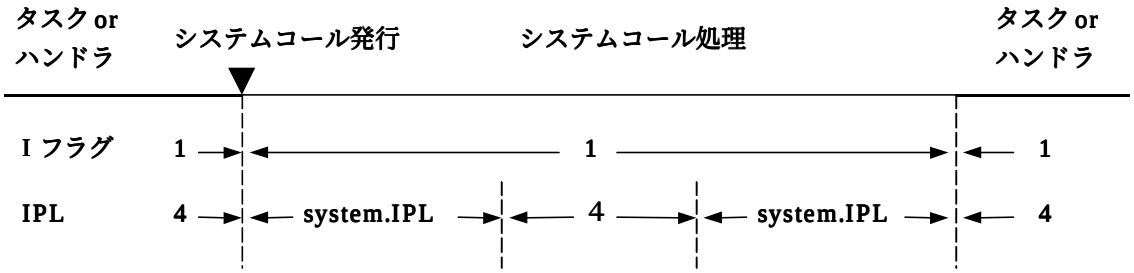


図 5.14 タスクからのみ発行できるシステムコール内での割り込み制御

図 5.13、図 5.14に示すように割り込み許可フラグおよび IPL は、システムコール内で変化します。そのため、ユーザーアプリケーション内で割り込みを禁止したい場合、割り込み許可フラグおよび IPL の操作によって割り込みを禁止にする方法はお奨めできません。

割り込みの制御は、以下に示す二つの方法をお奨めします。

1. **禁止にしたい割り込みの割り込み制御レジスタ**（SFR）**を変更する。**
2. loc_cpu **～** unl_cpu **を使用する。**

loc_cpu システムコールにより、制御できる割り込みは、OS 依存割り込みのみです。OS 独立割り込みを制御する場合には、1による方法でおこなってください。

5.4 ディスパッチ遅延について

MR79 では、ディスパッチ遅延に関するシステムコールが 4 つあります。

- dis_dsp
- ena_dsp
- loc_cpu
- unl_cpu

これらのシステムコールを使用し、一時的にディスパッチを遅延した場合のタスクの扱いについて以下に記述します。

1. **ディスパッチ遅延中の実行タスクがプリエンプトされる場合**

   ディスパッチが禁止されている間は、実行中のタスクがプリエンプトされるべき状況となっても、新たに実行すべき状態となったタスクにはディスパッチされません。実行するべきタスクへのディスパッチは、ディスパッチ禁止状態が解除されるまで遅延されます。ディスパッチ遅延中は、

   - 実行中のタスクは RUN 状態であり、レディキューにつながれている。
   - ディスパッチ禁止解除後に実行するタスクは、READY 状態であり、(タスクがつながれている中で)最高優先度のレディキューにつながれている。

2. **ディスパッチ遅延中の** isus_tsk,irsm_tsk

   また、ディスパッチ禁止状態で起動された割り込みハンドラから、実行中のタスク(dis_dsp を発行したタスク) に対して isus_tsk を発行し SUSPEND 状態へ移行させようとした場合、タスク状態の遷移はディスパッチ禁止状態が解除されるまで遅延されます。ディスパッチ遅延中は、

   - 実行中のタスクの状態の扱いは、OS 内部では、ディスパッチ遅延解除後の状態として扱います。そのため、実行中のタスクに対して発行された isus_tsk では、実行中のタスクをレディキューからはずし、SUSPEND 状態に移行します。エラーコードは E_OK を返します。この後、実行中のタスクに対して irsm_tsk が発行されると、実行中のタスクをレディキューにつなぎ、エラーコードは E_OK を返します。ただし、ディスパッチ遅延が解除されるまでタスクの切り替えは起こりません。
   - ディスパッチ禁止解除後に実行するタスクは、レディキューにつながれています。

3. **ディスパッチ遅延中の** rot_rdq, irot_rdq

   ディスパッチ遅延中に、rot_rdq(TPRI_RUN=0)を発行した場合、自タスクの持つ優先度のレディキューを回転させます。また、irot_rdq(TPRI_RUN=0)を発行した場合、実行中のタスクが持つ優先度のレディキューが回転します。この場合、実行中のタスクは該当レディキューにはつながれていない場合があります。 (ディスパッチ遅延中に、実行タスクに対し isus_tsk が発行された場合など。)

4．**注意事項**

- dis_dsp,loc_cpu により、ディスパッチが禁止されている状態で、自タスクを待ち状態に移す可能性のあるシステムコール (slp_tsk,wai_sem など)は発行できません。
- loc_cpu により割り込みおよびディスパッチを禁止した状態で ena_dsp,dis_dsp は発行できません。
- dis_dsp を何回か発行して、その後、ena_dsp を 1 回発行しただけでディスパッチ禁止状態は解除されます。

  上記の状態遷移をまとめると表 5.2のようになります。

表 5.2　dis_dsp,loc_cpu に関する割り込み、ディスパッチの状態遷移

状態番号	状態の内容		dis_dsp を実行	ena_dsp を実行	loc_cpu を実行	unl_cpu を実行
	割り込み	ディスパッチ				
1	許可	許可	→ 2	→ 1	→ 3	→ 1
2	許可	禁止	→ 2	→ 1	→ 3	→ 1
3	禁止	禁止	×	×	→ 3	→ 1

5.5 初期起動タスクについて

MR79 では、システム起動時に READY 状態からスタートするタスクを指定できます。この指定はコンフィグレーションファイルで設定を行います。

設定方法の詳細については、99ページを参照してください。

5.6 MR79 スタートアッププログラムの修正方法

MR79 には、以下に示す 2 種類のスタートアッププログラムが用意されています。

- start.a79
  アセンブリ言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。

- crt0mr.a79
  C 言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。"start.a79"に C 言語の初期化ルーチンを追加したものです。

スタートアッププログラムは以下のようなことをおこなっています。

- リセット後のプロセッサの初期化
- C 言語の変数の初期化 (crt0mr.a79 のみ)
- システムタイマの設定
- MR79 のデータ領域の初期化

このスタートアッププログラムは、カレントディレクトリにない場合には cfg79 を –m オプションを用いて起動したとき、環境変数 "LIB79"の示すディレクトリからカレントディレクトリへコピーされます。

なお、必要があれば以下の示す箇所を修正、あるいは追加してください。

- プロセッサモードレジスタの設定
  プロセッサモードレジスタ(5EH,5FH 番地)に、システムに合わせたプロセッサモードを設定してください。(crt0mr.a79 の 77-82 行目) 5FH 番地の設定は、DPR4 本モードの場合は 79 行目、DPR1 本モードの場合は 81 行目を変更します。

- リセット直後に必要な設定
  CS の設定など、CPU のリセット直後に必要な設定があれば記述してください。(crt0mr.a79 の 86 行目)

- ユーザーで必要な初期化プログラムの追加
  ユーザーに必要な初期化プログラムを追加する場合は、C 言語用スタートアッププログラム (crt0mr.a79)の 192 行目に追加してください。また、ユーザーの初期化プログラム内で、DT, DPR, PSW 等のレジスタ値を変更した場合、ユーザーの初期化プログラムに入る前の値に戻してください。
  標準入出力関数を使用する場合は crt0mr.a79 の 169-170 行目のコメント記号をはずしてください。

5.6.1 C 言語用スタートアッププログラム (crt0mr.a79)

図 5.15に C 言語用スタートアッププログラム crt0mr.a79 を示します。

```
 1 ;*****************************************************************
 2 ;
 3 ;    crt0mr.a79 : MR79 startup program (for C langage)
 4 ;
 5 ;    Copyright 1998-1999 MITSUBISHI ELECTRIC CORPORATION
 6 ;    and MITSUBISHI ELECTRIC SEMICONDUCTOR SOFTWARE CORPORATION
 7 ;    All Rights Reserved.
 8 ;
 9 ;    $Id: crt0mr.a79,v 1.1 2000/11/17 09:47:21 muraki Exp $
10 ;
11 ;*****************************************************************
12 C_inc        .EQU    0
13
14      .INCLUDE        mr79.inc
15 ;
16 ;=======================
17 ; Section Arrangement
18 ;=======================
19      .INCLUDE        c_sec.inc
20
21 ;=======================
22 ; MR79 data definition
23 ;=======================
24      .INCLUDE        sys_rom.inc
25      .INCLUDE        sys_ram.inc
26
27 ;============================
28 ; initialize MACRO definition
29 ;============================
30 BZERO        .macro  _TOP,_SECT
31      ldad    E,#SIZEOF _SECT
32      phb                             ; push Eregister(high)
33      pha                             ; push Eregister(low)
34      ldad    E,#STARTOF _TOP
35      phb
36      pha
37      .glb    _bzero
38      jsrl    _bzero
39      adds    #8
40      .endm
41
42 BCOPY        .macro  _FROM,_TO,_SECT
43      ldad    E,#SIZEOF _SECT
44      phb
45      pha
46      ldad    E,#STARTOF _TO
47      phb
48      pha
49      ldad    E,#STARTOF _FROM
50      phb
51      pha
52      .glb    _bcopy
53      jsrl    _bcopy
54      adds    #12
55      .endm
56
57 ;========================
58 ; Statrt up MR79 routine
59 ;========================
```

```
 60 ; after reset,this program will start
 61 ;
 62
 63     .SECTION        interrupt
 64     .GLB    __SYS_INITIAL
 65 .IF  __DPR_MODE == 4
 66     .DP0    OFF
 67 .ELSE
 68     .DP     OFF
 69 .ENDIF
 70 __SYS_INITIAL:
 71 __DT .equ    00H
 72     .GLB    __DP1
 73     .GLB    __DP2
 74     .GLB    __DP3
 75
 76     ldt     #__SDT
 77     movmb   DT:5eH,#0A2H            ; set processor mode register
 78 .IF  __DPR_MODE == 4
 79     movmb   DT:5fH,#82H             ; case DPR0-3 mode
 80 .ELSE
 81     movmb   DT:5fH,#80H             ; case DPR mode
 82 .ENDIF
 83 ;---------------------
 84 ; Initialize CS Area
 85 ;---------------------
 86
 87     .GLB    __SYS_END
 88 __SYS_END:
 89     jmpl    __INITIAL_START
 90 ;
 91 ; MR79 start
 92 ;
 93     .SECTION        MR79_KERNEL
 94     .GLB    __INITIAL_START,__END_INIT
 95 __INITIAL_START:
 96
 97     clp     m,x,d
 98     lda     A,LG:__D_Sys_SP
 99     tas
100
101 ;---------------------------------------------------------------------
102 ; If you use a chip-select wait controller, a DRAM control unit and
103 ; others, add the settings of these functions here before initializing
104 ; bss and data sections.
105 ;---------------------------------------------------------------------
106
107     movmb   DT:__DBG_MODE,#0        ; Debug mode initialize
108     ldt     #__DT                   ; Initialize data bank register
109
110 ;============================
111 ; Initialize NC79 section
112 ;============================
113     ;================
114     ;       NEAR
115     ;================
116     ;---------------------------------------
117     ;       Zero clear for data area
118     ;---------------------------------------
119     BZERO   bss_NE,bss_NE
120     BZERO   bss_NO,bss_NO
121     ;---------------------------------------
122     ;       Initialize for data area
123     ;---------------------------------------
124     BCOPY   data_INE,data_NE,data_NE
```

```
125     BCOPY   data_INO,data_NO,data_NO
126 .IF  __DPR_MODE == 4
127     ;---------------------------------------
128     ;       Zero clear for data area
129     ;---------------------------------------
130     ldd     (1,2,3),#__DP1,#__DP2,#__DP3
131     BZERO   bss_DP1E,bss_DP1E
132     BZERO   bss_DP1O,bss_DP1O
133     BZERO   bss_DP2E,bss_DP2E
134     BZERO   bss_DP2O,bss_DP2O
135     BZERO   bss_DP3E,bss_DP3E
136     BZERO   bss_DP3O,bss_DP3O
137     ;---------------------------------------
138     ;       Initialize for data area
139     ;---------------------------------------
140     BCOPY   data_IDP1E,data_DP1E,data_DP1E
141     BCOPY   data_IDP1O,data_DP1O,data_DP1O
142     BCOPY   data_IDP2E,data_DP2E,data_DP2E
143     BCOPY   data_IDP2O,data_DP2O,data_DP2O
144     BCOPY   data_IDP3E,data_DP3E,data_DP3E
145     BCOPY   data_IDP3O,data_DP3O,data_DP3O
146 .ENDIF
147     ;================
148     ;       FAR
149     ;================
150     ;---------------------------------------
151     ;       Zero clear for data area
152     ;---------------------------------------
153     BZERO   bss_FE,bss_FE
154     BZERO   bss_FO,bss_FO
155     ;---------------------------------------
156     ;       Initialize for data area
157     ;---------------------------------------
158     BCOPY   data_IFE,data_FE,data_FE
159     BCOPY   data_IFO,data_FO,data_FO
160
161 ;======================
162 ; Initialize UART1
163 ;======================
164 ;
165 ;    .GLB    _init
166 ;    jsrl    _init
167     ldt     #__SDT
168
169 .IF USE_TIMER
170 ;=========================
171 ; SYSTEM CLOCK initialize
172 ;=========================
173 tm_strt             .EQU    40h
174 ; set Timer mode.
175     movmb   DT:stmr_mod_reg,#01000000B
176 ; set Timer counter.
177     lda.W   A,#stmr_cnt
178     sta     A,stmr_ctr_reg
179 ; set Timer interrupt register.
180     movmb   stmr_int_reg,#stmr_int_IPL
181 ; start Timer.
182     lda     A, tm_strt
183     oramb   tm_strt,#stmr_bit
184 .ENDIF
185
186 ;====================================
187 ; user's initialization if there are
188 ;====================================
```

```
189
190 .IF  __DPR_MODE == 4
191     .DP0    OFF
192 .ELSE
193     .DP     OFF
194 .ENDIF
195     .DT     __SDT
196
197 ;============================
198 ; Initialize MR79_RAM section
199 ;============================
200     _INIT_MR79
201
202 ;============
203 ; start task
204 ;============
205
206     sep     x,m
207     ldx.B   #1
208 __start_task:
209     lda.B   A,LG:__D_INIT_START-1,X
210     beq     __start_end
211     inx.B
212     phx
213     tay.B                           ; Y = start task ID
214     ldx.B   #0                      ; X = start code
215     .IF     INT_DISABLE_MODEL
216     .GLB    __sys_ista_tsk_i
217     jsrl    __sys_ista_tsk_i
218     .ELSE
219     .GLB    __sys_ista_tsk
220     jsrl    __sys_ista_tsk
221     .ENDIF
222     plx
223     bra     __start_task
224 __start_end:
225     oram.B  __DBG_MODE,#80H
226
227 __END_INIT:
228     .IF     INT_DISABLE_MODEL
229     .GLB    __SYS_EXIT_I
230     jmpl    __SYS_EXIT_I
231     .ELSE
232     .GLB    __SYS_EXIT
233     jmpl    __SYS_EXIT
234     .ENDIF
235
236
237 .IF USE_TIMER
238 ;================================================
239 ;
240 ; __SYS_STMR_INH --- System clock interrupt handler
241 ;
242     .SECTION        interrupt
243 .IF  __DPR_MODE == 4
244     .DP0            OFF
245 .ELSE
246     .DP             OFF
247 .ENDIF
248     .DT             __SDT
249     .GLB            __SYS_TIMEOUT, __SYS_TIMEOUT_TOUT
250     .GLB            __SYS_STMR_INH
251 __SYS_STMR_INH:
252 .IF INT_DISABLE_MODEL
253       change_IPL
```

```
254     cli
255 .ENDIF
256     SaveRegs        ; Save Registers
257     _IntEntry       ; Interrupt Entry
258     _STMR_Int       ; System timer interrupt
259 .ENDIF
260
261     .END
262
263 ;*****************************************************************
264 ;
265 ;       Copyright 1998-1999 MITSUBISHI ELECTRIC CORPORATION
266 ;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
267 ;       All Rights Reserved.
268 ;
269 ;*****************************************************************
```

図 5.15 C 言語用スタートアッププログラム (crt0mr.a79)

以下に標準スタートアッププログラム crt0mr.a79 の内容について説明します。

1. MR79 用インクルードファイルを組み込みます。 [図 5.15の 14]
2. セクション定義ファイルを組み込みます。 [図 5.15の 19]
3. システム ROM 領域定義ファイルを組み込みます。 [図 5.15の 24]
4. システム RAM 領域定義ファイルを組み込みます。 [図 5.15の 25]
5. リセット直後に起動される初期化プログラム__SYS_INITIAL です。 [図 5.15の 76-227]
   - DT レジスタの初期化 [図 5.15の 76]
   - プロセッサモードレジスタの設定 [図 5.15の 78-82]
   - システムスタックポインタの設定 [図 5.15の 98-99]
   - C 言語の初期設定をおこないます。 [図 5.15の 107-159]
     72-74 行目は DPR4 本モードを使用しない場合も削除しないでください。
   - MR79 のシステムクロック割り込みの設定をおこないます。 [図 5.15の 169-184]
6. 必要であればアプリケーション固有の初期設定を記述します。 [図 5.15の 192]
7. MR79 が使用する RAM データの初期化をおこないます。 [図 5.15の 200]
8. 初期起動タスクを起動します。 [図 5.15の 205-224]
9. システムクロックの割り込みハンドラです。 [図 5.15の 237-259]

5.7 メモリ配置方法

アプリケーションプログラムのデータのメモリ配置方法について説明します。
メモリ配置を設定するためには、MR79 が提供しているセクションファイルで設定します。
MR79 では、以下に示す 2 種類のセクションファイルが用意されています。

- asm_sec.inc
  アセンブリ言語で、アプリケーション開発を行った場合に使用します。
  各セクションの詳細については、87ページを参照してください。

- c_sec.inc
  C 言語で、アプリケーション開発を行った場合に使用します。
  c_sec.inc は、"asm_sec.inc"に C コンパイラ NC79 が生成するセクションを追加したものです。
  各セクションの詳細については、88ページを参照してください。

ユーザーシステムに合わせて、セクション配置、開始アドレスの設定を変更してください。

プログラムセクションの変更方法を以下に示します。

例
プログラムセクションの先頭を A000H 番地から B000H 番地に変更したい場合

```
.section        program
.org    A000H   ; この部分を B000H に修正

        ↓

.section        program

.org    B000H   ;
```

5.7.1 start.a79 のセクションの配置

アセンブリ言語用サンプルスタートアッププログラム"start.a79"のセクション配置は、"asm_sec.inc"で定義しています。
セクションの再配置が必要な場合は、"asm_sec.inc"を編集してください。
以下に、サンプルセクション定義ファイル"asm_sec.inc"で定義している各セクションについて説明します。

- _SFR セクション
  スペシャルファンクションレジスタ（SFR）の存在するセクション。このセクションのサイズや位置については、使用しているマイコンのユーザーズマニュアルを参照してください。

- MR79_RAM セクション
  MR79 のシステム管理データで、アブソリュートアドレシングで参照する RAM データが入るセクションです。

- data セクション
  ユーザープログラムのデータを入れるセクション。このセクションは、MR79 カーネルは全く使用していません。したがって、ユーザーで自由に使用することができます。

- stack セクション
  各タスクのユーザースタック、およびシステムスタックのセクションです。
  このセクションは必ず 0 バンク内に配置してください。

- mr_heap セクション
  可変長メモリプールが格納されるセクションです。

- interrupt セクション
  割り込みハンドラおよび、スタートアップルーチンの一部を格納するセクションです。
  このセクションは必ず 0 バンク内に配置してください。

- MR79_KERNEL セクション、MR79_KERNEL_F セクション
  MR79 カーネルプログラムを格納するセクションです。

- MR79_ROM セクション
  MR79 カーネルが参照するタスクの開始番地などのデータを格納するセクションです。

- program セクション
  ユーザープログラムを格納するセクションです。
  このセクションは MR79 カーネルは全く使用していません。したがって、ユーザーで自由に使用することができます。

- INTERRUPT_VECTOR セクション
  割り込みベクタを格納するセクションです。このセクションの開始番地は使用する 7900 シリーズ機種により異なります。サンプルスタートアッププログラムの番地は 7920 用のものです。他のグループを使用する場合は変更してください。

5.7.2 crt0mr.a79 のセクション配置

C 言語用サンプルスタートアッププログラム"crt0mr.a79"のセクション配置は、"c_sec.inc"で定義しています。
セクションの再配置が必要な場合は、"c_sec.inc"を編集してください。
サンプルセクション定義ファイル"c_sec.inc"で定義しているセクションは、アセンブリ言語用スタートアッププログラムのセクション配置"asm_sec.inc"に、以下のセクションを定義したものです。

- data_NE セクション
- bss_NE セクション
- data_N0 セクション
- bss_N0 セクション
- program_N セクション
- rom_NE セクション
- rom_N0 セクション
- data_INE セクション
- data_IN0 セクション
- data_IDP1E セクション
- data_IDP10 セクション
- data_IDP2E セクション
- data_IDP20 セクション
- data_IDP3E セクション
- data_IDP30 セクション
- rom_FE セクション
- rom_F0 セクション
- bss_FE セクション
- bss_F0 セクション
- program_F セクション
- rom_FE セクション
- rom_F0 セクション
- data_IFE セクション
- data_IF0 セクション

これらのセクションは、NC79 が生成するセクションです。詳細は、NC79 のマニュアルを参照してください。

- MR79_INTERFACE セクション
  MR79 C 言語インターフェースライブラリを格納するセクションです。

サンプルスタートアッププログラムのセクションの配置を図 5.16に示します。

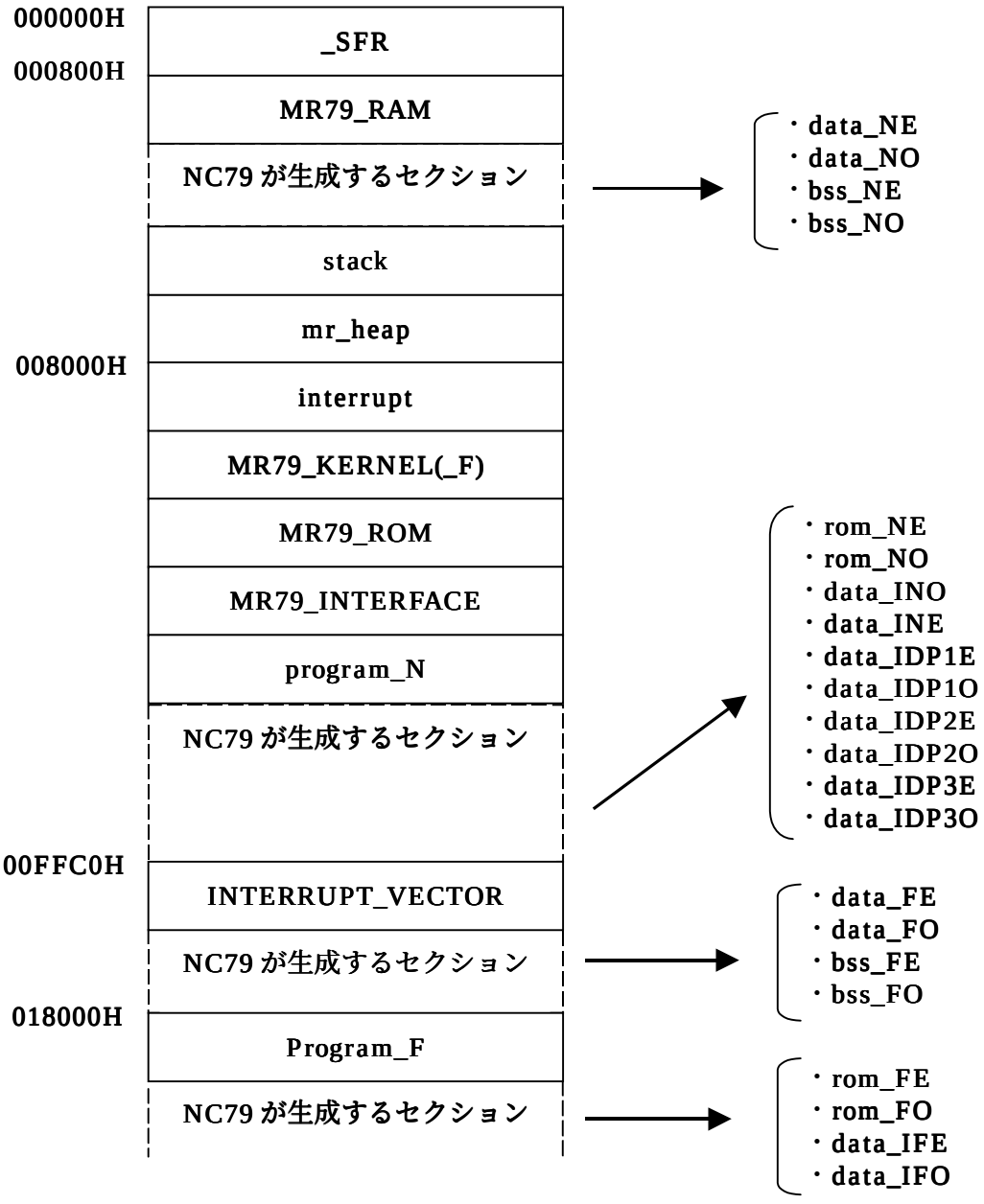


図 5.16 C 言語スタートアッププログラムのセクション配置

第 6 章　　コンフィグレータの使用方法

6.1 コンフィグレーションファイルの作成方法

アプリケーションプログラムのコーディング、スタートアッププログラムの修正が終わると、そのアプリケーションプログラムを MR79 システムに登録する必要があります。これをおこなうのがコンフィグレーションファイルです。

6.1.1 コンフィグレーションファイル内の表現形式

この節ではコンフィグレーションファイル内における定義データの表現形式について説明します。

コメント文

'//'から行の終わりまではコメント文とみなし、処理の対象になりません。

文の終わり

';'で文を終ります。

数値

数値は以下の形式で入力できます。

1. 16 進数

数値の先頭に'0x'か'0X'を付加します。または、数値の最後に'h'か'H'を付加します。後者の場合でかつ先頭が英文字 (A～F)で始まる場合は先頭に必ず'0'を付加してください。なおここで使用する数値表現で英文字 (A～F)は大文字・小文字を識別しません。[68]

2. 10 進数

23 のように整数のみで表します。ただし'0'で始めることはできません。

3. 8 進数

数値の先頭に'0'を付加するか数値の最後に'0'もしくは'o'を付加します。

4. 2 進数

数値の最後に'B'または'b'を付加します。ただし'0'で始めることはできません。

表 6.1 数値表現例

16 進数	0xf12 0Xf12 0a12h 0a12H 12h 12H
10 進数	32
8 進数	017 17o 170
2 進数	101110b 101010B

[68] 数値表現内の'A'～'F','a'～'f'を除いて全ての文字は、大文字・小文字の区別を行います。

また数値内に演算子を記述できます。使用できる演算子を表 6.2に示します。

表 6.2　演算子

演算子	優先度	演算方向
()	高	左から右
- (単項マイナス)		右から左
* / %		左から右
+ - (二項マイナス)	低	左から右

数値の例を以下に示します。

- 123
- 123 + 0x23
- (23 / 4 + 3) * 2
- 100B + 0aH

シンボル

シンボルは数字、英大文字、英小文字、'_'（アンダースコア)、'？'より構成される数字以外の文字で始まる文字列で表されます。
シンボルの例を以下に示します。

- _TASK1
- IDLE3

関数名

関数名は数字、英大文字、英小文字、'_' (アンダースコア)、'$'(ドル記号)より構成される数字以外の文字で始まり、' ()'で終わる文字列で表されます。
C 言語で記述した関数名の例を以下に示します。

- main()
- func()

アセンブリ言語で記述した場合はモジュールの先頭ラベルを関数名とします。

周波数

周波数は数字と'.'(ピリオド) から構成され'MHz'で終わる文字列で表されます。小数点以下は 6 桁が有効です。なお周波数は 10 進数のみで記述可能です。
周波数の例を以下に示します。

- 16MHz
- 8.1234MHz

なお、周波数は'.'で始まってはいけません。

時間

時間は数字と'. ' (ピリオド) から構成され'ms'または's'で終わる文字列で表されます。有効桁数は'ms'の場合小数点以下 3 桁です。's'の場合は小数点以下 6 桁です。なお時間は 10 進数のみで記述可能です。

時間の例を以下に示します。

- 0.23s
- 10ms
- 10.5ms

なお時間は'.'(ピリオド)で始まってはいけません。

時刻

時刻は 3 つの 16 ビットの数値を':'でつないだ形で表現される 48 ビットのデータです。たとえば、

- 23 : 0x02 : 100B

なお、3 つの数字の内、上位の 1 つもしくは上位の 2 つを省略した場合はその位置の数字は 0 とみなされます。すなわち、'12' は'0: 0: 12'と等価です。

6.1.2 コンフィグレーションファイルの定義項目

コンフィグレーションファイルでは以下の項目[69]の定義をおこないます。

- システム定義
- システムクロック定義
- タスク定義
- イベントフラグ定義
- セマフォ定義
- メイルボックス定義
- 固定長メモリープール定義
- 可変長メモリープール定義
- 周期起動ハンドラ定義
- アラームハンドラ定義
- 割り込みベクタ定義

【システム定義】

<< 形式 >>

```
// System Definition
system{
    stack_size      = システムスタックサイズ ;
    priority        = 優先度の最大値 ;
    message_size    = メッセージサイズ ;
    interrupt_model = 割り込み制御方法
    system_IPL      = OS割り込み禁止レベル ;
    timeout         = タイムアウト ;
};
```

<< 内容 >>

1. システムスタックサイズ（バイト数）

【 定義形式 】数値

【 定義範囲 】1 以上

【 省略時 】デフォルト値を使用

システムコール処理および割り込み処理で使用するスタックサイズの合計を定義します。

[69] タスク定義以外の項目は、省略することができます。省略した場合にはデフォルトコンフィグレーションファイルの定義が参照されます。

2. 優先度の最大値（最低優先度の値）

【 定義形式 】数値

【 定義範囲 】1 ～ 124

【 省略時 】デフォルト値を使用

MR79 のアプリケーションプログラムの使用する優先度の最大値を定義します。すなわち使用している優先度の最も大きい値を設定してください。[70]

3. メッセージサイズ（ビット幅）

【 定義形式 】数値

【 定義範囲 】16 or 24

【 省略時 】デフォルト値を使用

メイルボックスのメッセージサイズを指定します。メッセージが 16 ビットのデータの場合は、16 を指定し、24 ビットのデータの場合は、32 を指定してください。指定を省略した場合は、16 が設定されます。

4. 割り込み制御方法

【 定義形式 】シンボル

【 定義範囲 】SHORT または STANDARD

【 省略時 】デフォルト値を使用

割り込み制御方法を指定します。I フラグによる割り込み制御（LM,SM モデル）の場合は "STANDARD" を、IPL による制御（LMI,SMI モデル）の場合は "SHORT" を設定します。

5. OS 割り込み禁止レベル

【 定義形式 】数値

【 定義範囲 】0 ～ 7

【 省略時 】デフォルト値を使用

システムコール内での IPL の値、すなわち OS 割り込み禁止レベルを設定します。[71] 割り込み制御方法を "SHORT" とした時のみ、この設定が有効となり、割り込み制御方法を "STANDARD" を選択した場合は、この項目を設定する必要はありません。もし設定したとしても、設定値は無視されます。

[70] MR79 の優先度は、値が大きいほど優先度は低くなります。
[71] 0 を定義した場合は、システムクロック割り込みハンドラや OS 依存割り込みハンドラが全く使用できなくなります。

6. タイムアウト

【 定義形式 】シンボル

【 定義範囲 】YES or NO

【 省略時 】デフォルト値を使用

tslp_tsk、twai_flg、twai_sem、trcv_msg を使用している場合は、YES を設定し、使用していない場合は、NO を設定してください。

【システムクロック定義】

<< 形式 >>

```
// System Clock Definition
clock{
    mpu_clock       = MPUのクロック;
    timer           = システムクロックに使用するタイマ;
    IPL             = システムクロック割り込み優先レベル;
    unit_time       = システムクロックの単位時間;
    initial_time    = システム時刻の初期値;
};
```

<< 内容 >>

1. MPU のクロック(MHz 単位)

【 定義形式 】周波数

【 定義範囲 】なし

【 省略時 】デフォルト値を使用

7900 の MPU 動作クロックの周波数を MHz 単位で定義します。

2. システムクロックに使用するタイマ

【 定義形式 】シンボル

【 定義範囲 】A0, A1, A2, A3, A4, B0, B1, B2, OTHER, NOTIMER

【 省略時 】デフォルト値を使用

システムクロックに使用するハードウェアタイマを定義します。
タイマ A（A0～A4）を指定した場合は、タイマ A クロック分周指定レジスタ（45H 番地）には 0 または 1 以外の値を設定しないでください。リセット時は 0 です。0 または 1 以外の値に設定しなければならない場合は、システムクロックにはタイマ B（B0～B2）を使用してください。
システムクロックを使用しない場合は、"NOTIMER"を定義します。

3. システムクロック割り込み優先レベル

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義の OS 割り込み禁止レベル)

【 省略時 】デフォルト値を使用

システムクロック用タイマ割り込みの優先レベルを定義します。1 ～ OS 割り込み禁止レベルまでの値を設定してください。
システムクロックの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられません。

4. システムクロックの単位時間(ms 単位)

【 定義形式 】時間

【 定義範囲 】$\frac{16}{\text{MPUのクロック}(mpu\_clock)\times 1000} \sim \frac{16\times 65535}{\text{MPUのクロック}(mpu\_clock)\times 1000}$

システムクロックの単位時間（システムクロック割り込み発生間隔)を ms 単位で定義します。

5. システム時刻の初期値

【 定義形式 】時刻

【 定義範囲 】0 : 0 : 0 ～ 0x7FFF : 0xFFFF : 0xFFFF

【 省略時 】デフォルト値を使用、あるいは不要

システム時刻の初期値を定義します。システム時刻を用いた機能（set_tim、get_tim、アラームハンドラ)を使用しない場合は、この項目を設定する必要はありません。本定義をおこなわないことにより自動的にシステムクロック割り込みハンドラの処理が最適化されます。ただし、デフォルトコンフィグレーションファイルでデフォルト値を定義した場合、最適化は行いませんので注意してください。

【タスク定義】

<< 形式 >>

```
 // Tasks Definition
task[ID番号]{
    entry_address   = タスクの開始アドレス;
    stack_size      = タスクのユーザースタックサイズ;
    priority        = タスクの初期優先度;
    initial_start   = 初期起動状態;
};
            :
            :
```

ID 番号は 1～124 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

タスク ID 番号ごとに以下の定義をおこないます。

1. タスク開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

【 省略時 】エラーを出力

タスクの入り口アドレスを定義します。C 言語で記述したときはその関数名の最後に ()をつけるか、先頭に_をつけます。
なお、ここで定義した関数名は、id.h ファイルに以下の宣言文が出力されます。

```
#pragma TASK 関数名
```

2. ユーザースタックサイズ（バイト数）

【 定義形式 】数値

【 定義範囲 】24 以上

【 省略時 】デフォルト値を使用

タスクごとのユーザースタックサイズを定義します。ユーザースタックとは、各々のタスクが使用するスタック領域です。MR79 ではユーザー用のスタック領域をタスクごとに割り当てる必要があり最低で 24 バイトが必要です。

3. タスクの初期優先度

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義の優先度の最大値)

【 省略時 】デフォルト値を使用

タスクの起動時の優先度を定義します。
MR79 の優先度は、値が小さいほど、優先度としては高くなります。

4. 初期起動状態

【 定義形式 】シンボル

【 定義範囲 】ON or OFF

【 省略時 】デフォルト値を使用

タスクの初期起動状態を定義します。
ON を指定すると、システムの初期起動時に READY 状態になります。

【イベントフラグ定義】

<< 形式 >>

```
// Eventflag Definition
flag[ID番号]{
    name            = 名前;
};
            ⋮
            ⋮
```

ID 番号は 1～124 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

イベントフラグ ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

【 省略時 】デフォルト値を使用

プログラム中でイベントフラグを指定する時の名前を定義します。

【セマフォ定義】

<< 形式 >>

```
// Semaphore Definition
semaphore[ID番号]{
    name            = 名前;
    initial_count   = セマフォのカウンタ初期値;
};
            ⋮
            ⋮
```

ID 番号は 1～124 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

セマフォ ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

【 省略時 】エラーを出力

プログラム中でセマフォを指定する時の名前を定義します。

2. セマフォカウンタ初期値

【 定義形式 】数値

【 定義範囲 】0 ～ 32767

【 省略時 】デフォルト値を使用

セマフォカウンタの初期値を定義します。

【メイルボックス定義】

<< 形式 >>

```
// Mailbox Definition
mailbox[ID番号]{
    name            = 名前;
    buffer_size     = メイルボックスの最大メッセージ数;
};
            :
            :
```

ID 番号は 1～124 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

メイルボックス ID 番号ごとに以下の項目の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

【 省略時 】デフォルト値を使用

プログラム中でメイルボックスを指定する時の名前を定義します。

2. 最大メッセージ数（個数）

【 定義形式 】数値

【 定義範囲 】2^0 ～ 2^{14} （2^n の数値のみ定義可能）

【 省略時 】デフォルト値を使用

メイルボックスに蓄えることのできる最大メッセージ数を定義します。これを越えてメッセージを蓄えようとするとエラーがかえってきます。

【固定長メモリプール定義】

<< 形式 >>

```
// Fixed Memorypool Definition
memorypool[ID番号]{
    name            = 名前;
    base_address    = メモリプールのベースアドレス
    section         = セクション名;
    num_block       = メモリプールのブロック数;
    siz_block       = メモリプールのブロックサイズ;
};
```

ID 番号は、1～126 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

メモリプール ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

【 省略時 】エラーを出力

プログラム中でメモリプールを指定する時の名前を指定します。

2. メモリプールのベースアドレス

【 定義形式 】シンボルまたは数値

【 定義範囲 】なし

【 省略時 】下記のセクション名を使用

メモリプールとして使用する RAM 領域のベースアドレスを定義します。この定義をおこなうと下記のセクションは定義できません。

3. セクション名

【 定義形式 】シンボル

【 定義範囲 】なし

【 省略時 】デフォルト値を使用

メモリプールを配置するセクションの名前を定義します。ここで定義したセクションは、必ず、セクションファイル (asm_sec.inc あるいは c_sec.inc)にて配置をおこなってください。
この定義をおこなうと上記のベースアドレスは定義できません。

4. ブロック数

【 定義形式 】数値

【 定義範囲 】1～16

【 省略時 】デフォルト値を使用

メモリプールのブロック総数を定義します。

5. サイズ（バイト数）

【 定義形式 】数値

【 定義範囲 】1 以上

【 省略時 】デフォルト値を使用

メモリプールの 1 ブロック当たりのサイズを定義します。この定義によりメモリプールとして使用する RAM 容量は、 (ブロック数)×(サイズ)バイトです。

【可変長メモリプール定義】

<< 形式 >>

```
// Variable-Size Memorypool Definition
variable_memorypool[ID番号]{
    max_memsize     = 確保するメモリブロックサイズの最大値;
    heap_size       = メモリプールのサイズ;
};
```

ID 番号は、1 を指定してください。
セクションファイルで mr_heap セクションに指定されている領域に、メモリ確保されます。

<< 内容 >>

1. 確保するメモリブロックサイズの最大値（バイト数）

【 定義形式 】数値

【 定義範囲 】1 ～ 65520

【 省略時 】デフォルト値を使用

アプリケーションプログラム中で、確保しているメモリブロックサイズの最大値を指定します。

2. メモリプールのサイズ（バイト数）

【 定義形式 】数値

【 定義範囲 】16 ～ 524288

【 省略時 】デフォルト値を使用

メモリプールのサイズを指定します。

【周期起動ハンドラ定義】

<< 形式 >>

```
// Cyclic Handler Definition
 cyclic_hand[ID番号]{
    interval_counter        = 周期起動ハンドラの周期間隔;
    mode                    = 周期起動ハンドラのモード;
    entry_address           = 周期起動ハンドラの開始アドレス;
 };
            :
            :
```

ID 番号は 1 ～ 126 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

周期起動ハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 周期間隔

【 定義形式 】数値

【 定義範囲 】1 ～ 32767

周期起動ハンドラの周期起動間隔を定義します。ここで定義する時間の単位はシステムクロック定義項目において定義したシステムクロックの単位時間です。例えば、システムクロックの単位時間が 10ms のときに、10 秒間隔で周期起動しようとすると、この値を 1000 に設定します。

2. モード

【 定義形式 】シンボル

【 定義範囲 】TCY_OFF または TCY_ON

【 省略時 】デフォルト値を使用

周期起動ハンドラの初期モードを定義します。ここでは以下の２つのモードの何れかを定義します。

- TCY_OFF
  act_cyc システムコールを発行することで動作を始めるモード。
- TCY_ON
  システムの立ち上げと同時に動作を始めるモード。

3. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

【 省略時 】エラーを出力

周期起動ハンドラの開始アドレスを定義します。
なお、ここで定義した関数名は、id.h ファイルに以下の宣言文が出力されます。

```
#pragma CYCHANDLER 関数名
```

【アラームハンドラ定義】

<< 形式 >>

```
// Alarm Handler Definition
alarm_hand[ID番号]{
    time            = アラームハンドラの起動時刻;
    entry_address   = アラームハンドラの開始アドレス;
};
            ⋮
            ⋮
```

ID 番号は 1～126 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

アラームハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 起動時刻

【 定義形式 】時刻

【 定義範囲 】0 : 0 : 0 ～ 0x7FFF : 0xFFFF : 0xFFFF

【 省略時 】デフォルト値を使用

アラームハンドラの起動時刻を定義します。

2. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

【 省略時 】エラーを出力

アラームハンドラの開始アドレスを定義します。なお、ここで定義した関数名は、id.h ファイルに以下の宣言文が出力されます。

```
#pragma ALMHANDLER 関数名
```

【割り込みベクタ定義】

<< 形式 >>

```
// Interrupt Vector Definition
interrupt_vector[ベクタ番号]{
    os_int          = OS依存割り込みハンドラ;
    entry_address   = 割り込みハンドラの開始アドレス;
};
            ⋮
            ⋮
```

割り込みベクタ番号は 1～60 の範囲まで記述できます。
ただし、そのベクタ番号が有効か否かは使用しているマイクロコンピュータに依存します。
7920 グループの場合の割り込み要因と割り込みベクタ番号は表 6.3に示す対応になります。
また、コンフィグレータは、ここで指定した割り込みの割り込み制御レジスタ（IPL 等）や、割り込み要因などの初期設定のコードは生成しません。初期設定はスタートアップファイル中、もしくは、開発するアプリケーション中に記述していただく必要があります。

<< 内容 >>

1. OS 依存割り込みハンドラ

【 定義形式 】シンボル

【 定義範囲 】YES または NO

【 省略時 】エラーを出力

ハンドラが OS 依存割り込みハンドラかどうかを定義します。OS 依存割り込みハンドラで

あれば YES を、OS 独立割り込みハンドラであれば NO を定義してください。
YES を定義した場合、id.h ファイルに以下の宣言文を出力します。

```
#pragma INTHANDLER 関数名
```

また、NO を定義した場合、id.h ファイルに以下の宣言文を出力します。

```
#pragma INTERRUPT 関数名
```

2. 開始アドレス

【 定義形式 】シンボルまたは関数名

【 定義範囲 】なし

【 省略時 】エラーを出力

割り込みハンドラの入口アドレスを定義します。C 言語で記述した時はその関数名の最後に () をつけるか先頭に_をつけます。

表 6.3　7920 での割り込み要因とベクタ番号との対応

割り込み要因	割り込みベクタ番号	セクション名
リセット	1	INTERRUPT_VECTOR
零除算	2	INTERRUPT_VECTOR
BRK 命令（使用禁止）	3	INTERRUPT_VECTOR
DBC（使用禁止）	4	INTERRUPT_VECTOR
監視タイマ	5	INTERRUPT_VECTOR
NMI	6	INTERRUPT_VECTOR
INT0 外部割り込み	7	INTERRUPT_VECTOR
INT1 外部割り込み	8	INTERRUPT_VECTOR
INT2 外部割り込み	9	INTERRUPT_VECTOR
タイマ A0	10	INTERRUPT_VECTOR
タイマ A1	11	INTERRUPT_VECTOR
タイマ A2	12	INTERRUPT_VECTOR
タイマ A3	13	INTERRUPT_VECTOR
タイマ A4	14	INTERRUPT_VECTOR
タイマ B0	15	INTERRUPT_VECTOR
タイマ B1	16	INTERRUPT_VECTOR
タイマ B2	17	INTERRUPT_VECTOR
UART0 受信	18	INTERRUPT_VECTOR
UART0 送信	19	INTERRUPT_VECTOR
UART1 受信	20	INTERRUPT_VECTOR
UART1 送信	21	INTERRUPT_VECTOR
A-D 変換	22	INTERRUPT_VECTOR
INT3 外部割り込み	23	INTERRUPT_VECTOR
INT4 外部割り込み	24	INTERRUPT_VECTOR
予約領域	25	INTERRUPT_VECTOR
予約領域	26	INTERRUPT_VECTOR
アドレス一致検出	27	INTERRUPT_VECTOR
予約領域	28	INTERRUPT_VECTOR
DMA0	29	INTERRUPT_VECTOR
DMA1	30	INTERRUPT_VECTOR
DMA2	31	INTERRUPT_VECTOR
DMA3	32	INTERRUPT_VECTOR

6.1.3 コンフィグレーションファイル例

以下にコンフィグレーションファイルの例を示します。

```
 1 //*****************************************************************
 2 //
 3 //    Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
 4 //    AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
 5 //
 6 //    MR79 System Configuration File.
 7 //
 8 //*****************************************************************
 9 // System Definition
10 system{
11             stack_size              = 0x200;
12             priority                = 3;
13             message_size            = 24;
14             interrupt_model         = STANDARD;
15             system_IPL              = 6;
16             timeout                 = NO;
17 };
18 //System Clock Definition
19 clock{
20             mpu_clock               = 20MHz;
21             timer                   = A0;
22             IPL                     = 5;
23             unit_time               = 5ms;         // ms
24             initial_time            = 1:0x10:0xffff;
25 };
26 //Task Definition
27 task[1]{
28             entry_address           = task1();
29             stack_size              = 0x100;
30             priority                = 1;
31             initial_start           = ON;
32 };
33 task[2]{
34             entry_address           = _task2;
35             stack_size              = 100;
36             priority                = 2;
37             initial_start           = OFF;
38 };
39 task[3]{
40             entry_address           = task3;
41             stack_size              = 0100;
42             priority                = 3;
43             initial_start           = OFF;
44 };
45 //
46 flag[1]{
47             name            = flg1;
48 };
49 //
50 semaphore[1]{
51             name            = sem1;
52             initial_count   = 1;
53 };
54 //
55 mailbox[1]{
56             name            = mbx1;
57             buffer_size     = 0x10;
58 };
59 //
```

```
60 memorypool[1]{
61             name            = mpl1;
62             base_address    = 0x2000;
63             num_block       = 5;
64             siz_block       = 100;
65 };
66 //
67 variable_memorypool[1]{
68             max_memsize     = 400;
69             heap_size       = 1600;
70 };
71 //
72 cyclic_hand[1]{
73             interval_counter        = 0xff;
74             mode                    = TCY_OFF;
75             entry_address           = _cyh1;
76 };
77 //
78 alarm_hand[1]{
79             time            = 1:0xff:0xffff;
80             entry_address   = _alh1;
81 };
82 interrupt_vector[15]{
83             os_int          = YES;
84             entry_address   = _intr;
85     };
```

6.2 コンフィグレータの実行

6.2.1 コンフィグレータ概要

コンフィグレータはコンフィグレーションファイルで定義した内容をアセンブリ言語のインクルードファイル等に変換するツールです。コンフィグレータの動作概要を図 6.1に示します。

1. コンフィグレータの実行には以下の入力ファイルが必要です。

- コンフィグレーションファイル (XXXX.cfg)
  システムの初期設定項目を記述したファイルです。カレントディレクトリに作成します。

- デフォルトコンフィグレーションファイル (default.cfg)
  コンフィグレーションファイルで値の設定を省略した場合にこのファイルに書き込まれている値を設定します環境変数 "LIB79"で示されるディレクトリ、もしくは、カレントディレクトリに置きます。両方のディレクトリに存在する場合は、カレントディレクトリのファイルが優先されます。

- makefile のテンプレートファイル[72] (makefile.ews, makefile.dos, makefile, Makefile)
  makefile[73]を生成する場合にテンプレートのファイルとして使用するファイルです。 (第 6.2.4 項参照)

- mr79.inc テンプレートファイル (mr79.inc)
  インクルードファイル mr79.inc のテンプレートとなるファイルです。
  環境変数 "LIB79"で示されるディレクトリに存在します。

- MR79 バージョンファイル (version)
  MR79 のバージョンを記述したファイルです。環境変数 "LIB79" で示されるディレクトリに存在します。コンフィグレータはこのファイルを読み込み、起動メッセージに MR79 のバージョン情報を出力します。

2. コンフィグレータの実行によって以下のファイルが出力されます。

コンフィグレータが出力したファイルには、ユーザーのデータ定義を行わないでください。データ定義を行った後で、コンフィグレータを起動するとユーザーの定義したデータは失われます。

- システムデータ定義ファイル (sys_rom.inc)
  システムの設定を定義しているファイルです。

- インクルードファイル (mr79.inc)
  mr79.inc はアセンブリ言語用のインクルードファイルです。

- システム生成手順記述ファイル (makefile)
  システムを自動生成するためのファイルです。

---

[72] EWS 版では makefile.ews,DOS 版では makefile.dos を使用します。
[73] この makefile は、UNIX 標準もしくは準拠の make コマンドで処理可能なシステム生成手順記述ファイルです。

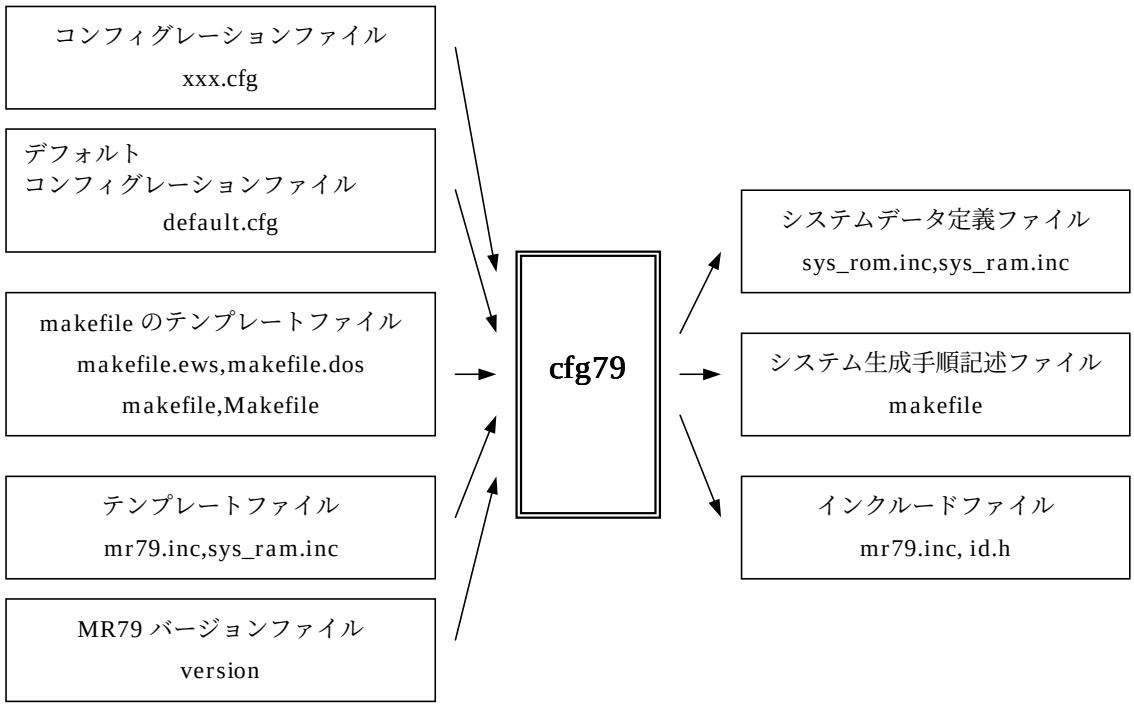


図 6.1　コンフィグレータ動作概要

6.2.2 コンフィグレータの環境設定

コンフィグレータを実行するにあたって環境変数 "LIB79"が正しく設定されているかを確認してください。

環境変数 "LIB79" で示すディレクトリ下には以下のファイルがないと正常に実行できません。

- デフォルトコンフィグレーションファイル (default.cfg)
  カレントディレクトリにコピーして使用することもできます。その場合はカレントディレクトリのファイルを優先して使用します。

- システム RAM 領域定義データベースファイル (sys_ram.inc)
- mr79.inc のテンプレートファイル (mr79.inc)
- セクション定義ファイル (c_sec.inc または asm_sec.inc)
- スタートアップファイル (crt0mr.a79 または start.a79)
- makefile のテンプレートファイル (makefile.ews または makefile.dos)
- MR79 バージョンファイル (version)

6.2.3 コンフィグレータ起動方法

コンフィグレータは以下の形式で起動します。

```
A> cfg79 [-vmV] コンフィグレーションファイル名
```

コンフィグレーションファイル名は、 通常拡張子（.cfg) かまたは拡張子（.cfg) を除いたファイル名を指定します。

コマンドオプション

-v オプション

コマンドのオプションの説明と詳細なバージョンを表示します。

-V オプション

コマンドが生成するファイルの作成状況を表示します。

-m オプション

UNIX 標準もしくは準拠のシステム生成手順記述ファイル（makefile)を作成します。指定がない場合は makefile を作成しません。[74]

また、カレントディレクトリにスタートアップファイル（crt0mr.a79 または start.a79) とセクション定義ファイル（c_sec.inc、asm_sec.inc) がない場合は、環境変数 LIB79 が示すディレクトリにある、サンプルスタートアップファイルとセクション定義ファイルをカレントディレクトリにコピーします。

[74] UNIX 標準もしくは準拠の"makefile"には、"clean"ターゲットによりワークファイルを削除する機能があります。すなわち、make コマンドで生成されたオブジェクトファイルなどを削除するには以下のように行います。
> make clean

6.2.4 makefile 生成機能

コンフィグレータは以下の手順で makefile を生成します。

1. **ソースファイルの依存関係を調べます。**

   コンフィグレータはカレントディレクトリの拡張子が.c と.a79 のファイルをそれぞれ、C 言語とアセンブリ言語として、それらがインクルードするファイルなどの依存関係を調べます。

   したがって、ソースファイルを記述する場合には次の 2 点に注意してください。

   - ソースファイルはカレントディレクトリに置かなければなりません。
   - ソースファイルの拡張子はＣ言語では'.c'を、アセンブリ言語では'.a79'を使用してください。
   - ソースファイルの依存関係には、#if やコメント行に記述して無効化した#include 文に指定したファイルも含みます。

2. makefile **へファイルの依存関係を書き込みます。**

   カレントディレクトリの "makefile"、または"Makefile"または、環境変数"LIB79"で示されるディレクトリの"makefile.ews"または"makefile.dos"を、テンプレートファイルとして、カレントディレクトリに"makefile"を作成します。

6.2.5 コンフィグレータ実行上の注意

以下にコンフィグレータ実行上の注意点を示します。

- コンフィグレータを再度かけ直した場合は、必ず make clean を実行するかオブジェクトファイル（拡張子.r79）を全て消去してから make コマンドを実行してください。リンク時等にエラーが発生する場合があります。
- コンフィグレーションファイルのシステム定義における "メッセージサイズ"、"DP レジスタモード"、"タイムアウト" のいずれかを変更した場合、"-m" オプションを使用してコンフィグレータを起動してください。
- スタートアッププログラム名、およびセクション定義ファイル名は変更しないでください。変更した場合、コンフィグレータ実行時にエラーが発生します。
- コンフィグレータ cfg79 では、UNIX 標準、もしくは準拠の makefile しか生成しません。

6.2.6 コンフィグレータのエラーと対処方法

以下のメッセージが表示された場合はコンフィグレータが正常に終了していませんのでコンフィグレーションファイルを修正の上、再度コンフィグレータを実行してください。

エラーメッセージ

cfg79 Error : syntax error near line 16 (test.cfg)
コンフィグレーションファイルに文法エラーがあります。

cfg79 Error : not enough memory
メモリが足りません。

cfg79 Error : illegal option --> <x>
コンフィグレータのコマンドオプションに誤りがあります。

cfg79 Error : illegal argument --> <xx>
コンフィグレータの起動形式に誤りがあります。

cfg79 Error : can't write open <XXXX>
XXXX ファイルが作成できません。ディレクトリの属性やディスクの残り容量を確認してください。

cfg79 Error : can't open <XXXX>
XXXX ファイルにアクセスできません。XXXX ファイルの属性や、存在を確認してください。

cfg79 Error : can't open version file
環境変数"LIB79"の示すディレクトリの下に MR79 バージョンファイル"version"がありません。

cfg79 Error : can't open default configuration file
デフォルトコンフィグレーションファイルがアクセスできません。環境変数 "LIB79"の示すディレクトリ、またはカレントディレクトリに"default.cfg"が必要です。

cfg79 Error : can't open configuration file <test.cfg>
コンフィグレーションファイルがアクセスできません。コンフィグレータの起動形式を確認の上、正しいファイル名を指定してください。

cfg79 Error : illegal XXXX --> <xx> near line 212 (test.cfg)
定義項目 XXXX の数値または ID 番号が間違っています。定義範囲を確認してください。

cfg79 Error : Unknown XXXX --> <xx> near line 23 (test.cfg)
定義項目 XXXX のシンボル定義が間違っています。定義範囲を確認してください。

cfg79 Error : too big XXXX's ID number --> <6> (test.cfg)
XXXX 定義の ID 番号に、定義したオブジェクトの総数を超える値が設定されています。ID 番号がオブジェクトの総数を超えることはありません。

cfg79 Error : too big task[x]'s priority --> <16> near line 36 (test.cfg)
ID 番号 x のタスク定義項目の初期優先度が、システム定義項目の優先度値を越えています。

cfg79 Error : too big IPL --> <5> near line 28 (test.cfg)
システムクロック定義項目のシステムクロック割り込み優先レベルがシステム定義項目の system IPL 値を越えています。

cfg79 Error : system timer's vector <x>conflict near line XXXX (test.cfg)
システムクロックの割り込みベクタに、 別の割り込みが定義されています。 割り込みベクタ番号を確認してください。

cfg79 Error : XXXX is not defined (test.cfg)
コンフィグレーションファイルで XXXX の項目の定義が必要です。

cfg79 Error : system's default is not defined
デフォルトコンフィグレーションファイルで定義が必要な項目です。

cfg79 Error : double definition <XXXX> near line 17 (test.cfg)
項目 XXXX は既に定義されています。確認の上、重複定義を削除してください。

cfg79 Error : double definition XXXX[x] near line 77 (default.cfg)

cfg79 Error : double definition XXXX[x] (test.cfg)
項目 XXXX において ID 番号 x は既に登録されています。ID 番号を変更するか重複定義を削除してください。

cfg79 Error : you must define XXXX near line 107 (test.cfg)
XXXX は、省略できない項目です。

cfg79 Error : you must define SYMBOL near line 30 (test.cfg)
省略できないシンボルです。

cfg79 Error : start-up-file (XXXX) not found
カレントディレクトリにスタートアップファイル XXXX が見つかりません。スタートアップファイル"start.a79"または "crt0mr.a79"が、カレントディレクトリに必要です。

cfg79 Error : bad start-up-file(XXXX)
カレントディレクトリに不要なスタートアップファイルがあります。

cfg79 Error : no source file
カレントディレクトリにソースファイルがありません。

cfg79 Error : zero divide error near line 28 (test.cfg)
演算式で 0(ゼロ) 除算が発生しました。

cfg79 Error : can’t define both “base_address” keywood and “section” keywo od near line 77 (test.cfg)
固定長メモリプール定義で、base_address と section の両方を定義しています。どちらか一方にしてください。

警告メッセージ

以下のメッセージは警告ですので、内容が理解できていれば無視してもかまいません。

cfg79 Warning : system is not defined (test.cfg)

cfg79 Warning : system.XXXX is not defined (test.cfg)
コンフィグレーションファイルでシステム定義またはシステム定義項目 XXXX が省略されています。

cfg79 Warning : system.message_size is not defined (test.cfg)
システム定義中のメッセージサイズ定義項目が省略されています。メイルボックス機能のメッセージサイズ (16 または 32)を指定してください。

cfg79 Warning : task[x].XXXX is not defined near line 32 (test.cfg)
ID 番号 x のタスク定義項目 XXXX が省略されています。

cfg79 Warning : Already definition XXXX near line 20 (test.cfg)
XXXX は既に定義されています。定義内容は無視されます。確認の上、重複定義を削除してください。

cfg79 Warning : specified variable memorypool.max_memsize 120 (test.cfg)
可変長メモリプール定義項目の最大メモリサイズが 120 より、小さいためその値を 120 に設定しました。

cfg79 Warning : interrupt_vector[x]'s default is not defined (default.cfg)
デフォルトコンフィグレーションファイルでベクタ番号 x の割り込みベクタ定義が抜けています。

cfg79 Warning : interrupt_vector[x]'s default is not defined near line 213 (test.cfg)
コンフィグレーションファイルのベクタ番号 x の割り込みベクタは、デフォルトコンフィグレーションファイルに定義されていません。

cfg79 Warning : Initial Start Task not defined
コンフィグレーションファイルで、初期起動タスクの定義がないためタスク ID 番号 1 のタスクを初期起動タスクとして定義しました。

cfg79 Warning : dpr_mode is no more supported near line xxx (test.cfg)
コンフィグレーションファイルの定義項目 dpr_mode の設定は不要になりました。定義されている場合でも、本ワーニングを出力して、なにも設定しません。

その他のメッセージ

以下のメッセージは makefile を生成する場合にのみ出力される警告メッセージです。コンフィグレータは要因となった部分を読み飛ばして makefile を生成します。

test.c(line 11): include format error.
ファイル読み込みの書式が間違っています。正しい書式に書き直してください。

cfg79 Warning : test.c(line 12): can't find <XXXX>

cfg79 Warning : test.c(line 13): can't find "XXXX"

インクルードファイル XXXX が見つかりません。ファイル名および存在を確認してください。

cfg79 Warning : over character number of including path-name

インクルードファイルのパス名が 255 文字を超えています。

cfg79 Warning : aa.h : include nest over

インクルードファイルのネストが 8 段を超えています。

6.3 makefile の編集

コンフィグレータが生成した、makefile を編集し、コンパイルオプションやライブラリなどを設定します。以下にその設定方法を示します。

1. NC79 コマンドオプション

C コンパイラのコマンドオプションは"CFLAGS"に定義します。"-c"オプションは必ず指定してください。

2. AS79 コマンドオプション

アセンブラのコマンドオプションは"ASFLAGS"に定義します。"-DC_inc=0"は必ず指定してください。

3. LN79 コマンドオプション

リンカのコマンドオプションは"LDFLAGS"に定義します。特に指定しなければならないオプションはありません。

4. ライブラリの指定

ライブラリの指定は、"LIBS" に定義します。

コンフィグレータがコンフィグレーションファイルとカレントディレクトリのソースファイルから必要なライブラリを"LIBS"に定義します。必要に応じて追加、削除をおこなってください。

6.4 make 実行時のワーニング

周期起動ハンドラまたはアラームハンドラを使用したプログラムを make 実行すると、

mr79.inc XXX Warning (asp79): Destination address may be changed

というワーニングが出力されますが、無視してください。

第 7 章　　アプリケーション作成の手引き

7.1 ハンドラからのシステムコールの処理手順

ハンドラ[75]からのシステムコール発行はタスクからのシステムコールと異なり、システムコール発行時にタスク切り替えは発生しません。タスク切り替えが発生するのはハンドラからの復帰時です。

ハンドラからのシステムコール処理手順は大きく分けて以下の３通りがあります。

1. **タスク実行中に割り込んだハンドラからのシステムコール**
2. **システムコール処理中に割り込んだハンドラからのシステムコール**
3. **ハンドラ実行中に割り込んだ（多重割り込み）ハンドラからのシステムコール**

[75] OS 独立割り込みハンドラからはシステムコールは発行できませんので、ここで述べているハンドラは OS 独立割り込みハンドラを含みません。

7.1.1 タスク実行中に割り込んだハンドラからのシステムコール

スケジューリング (タスク切り替え)は ret_int システムコールによりおこなわれます。[76] (図 7.1 参照)

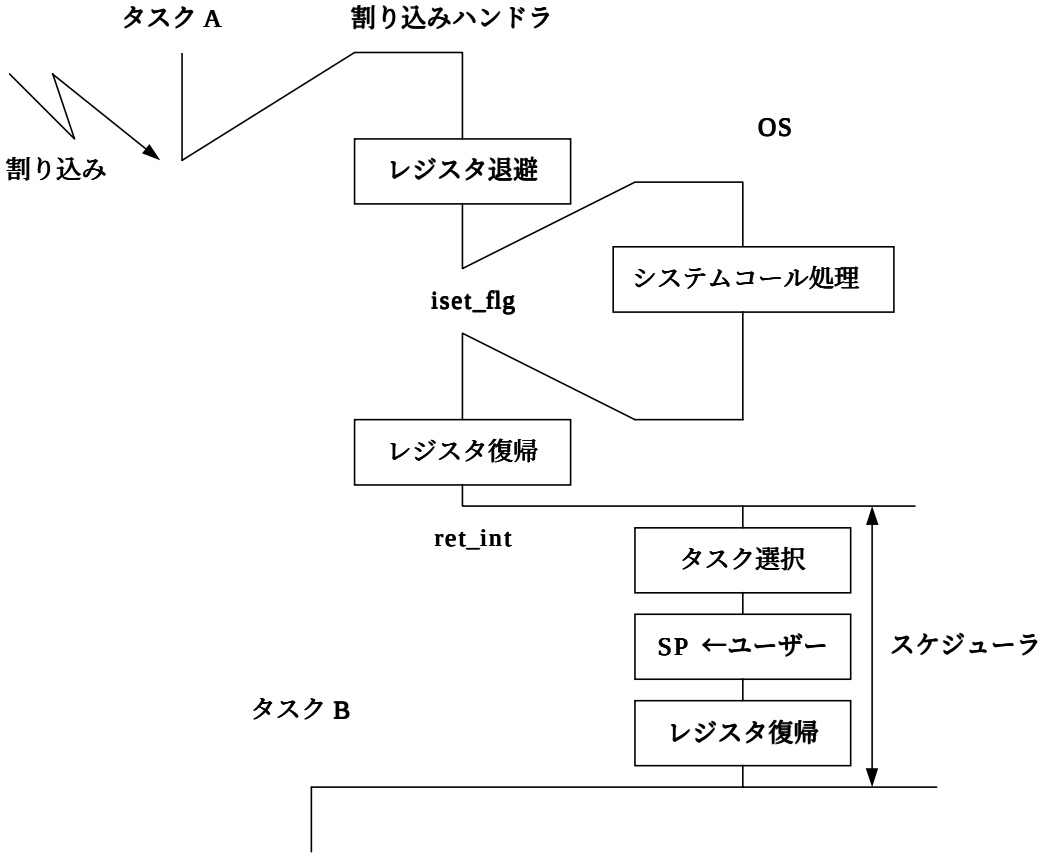


図 7.1 タスク実行中に割り込んだ割り込みハンドラからのシステムコール処理手順

[76] C 言語で OS 依存割り込みハンドラを記述する場合（#pragma INTHANDLER 指定時)、ret_int システムコールは自動的に発行されます。

7.1.2 システムコール処理中に割り込んだハンドラからのシステムコール

スケジューリング (タスク切り替え)は割り込まれたシステムコール処理に戻った後におこなわれます。 (図 7.2参照)

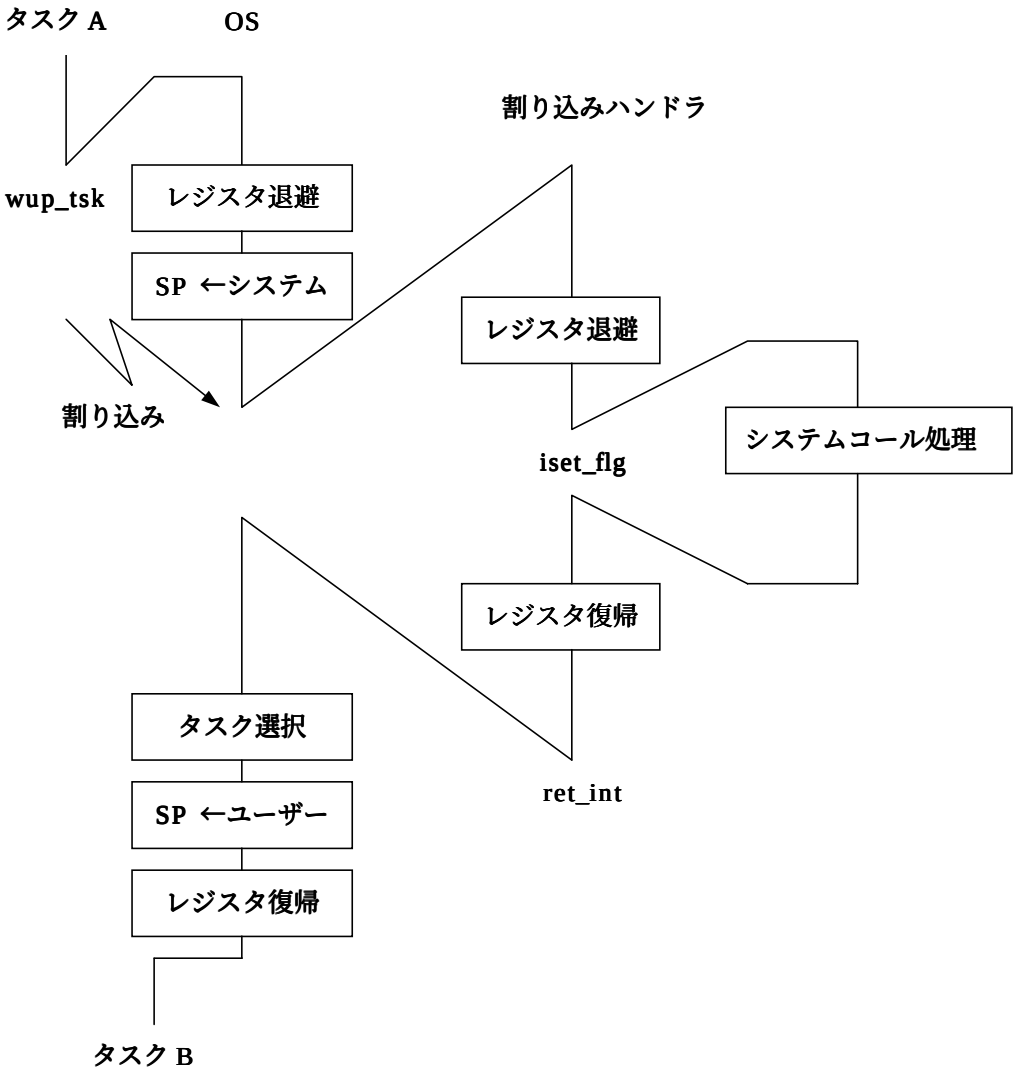


図 7.2 システムコール処理中に割り込んだ割り込みハンドラからのシステムコール処理手順

7.1.3 ハンドラ実行中に割り込んだハンドラからのシステムコール

ハンドラ（以後ハンドラ A と呼びます。)実行中に割り込みが発生した場合を考えます。ハンドラ A 実行中に割り込んだハンドラ（以後ハンドラ B と呼びます。)が、発行したシステムコールによりタスク切り替えが必要になった場合は、ハンドラ B から復帰するシステムコール（ret_int システムコール)では、ハンドラ A に戻るだけでタスク切り替えはおこないません。

ハンドラ A からの ret_int システムコールによりタスク切り替えが行われます。（図 7.3参照)

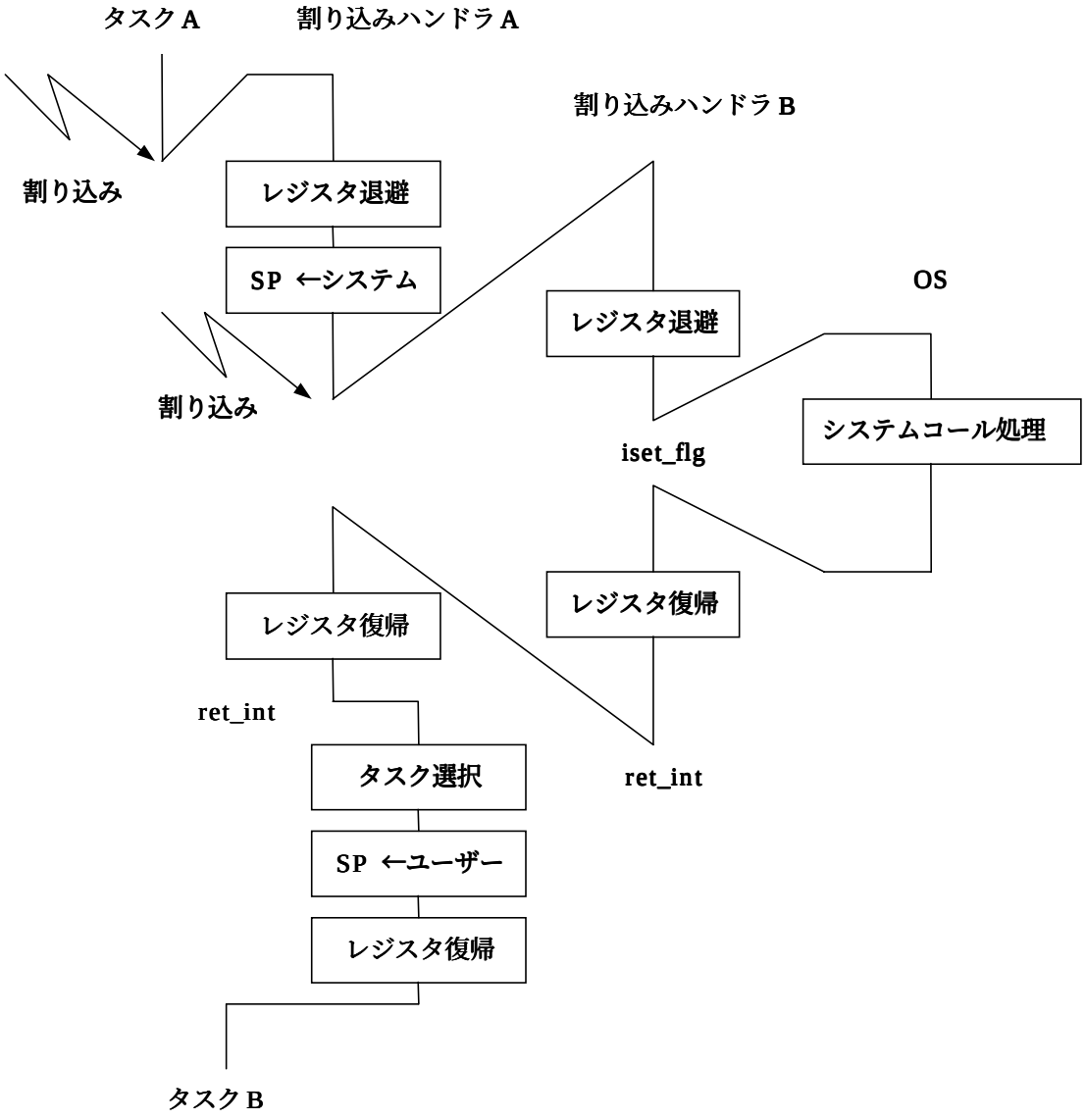


図 7.3 多重割り込みハンドラからのシステムコール処理手順

7.2 システムの使用する RAM 容量の計算方法

MR79 カーネルがタスクなどを管理するための RAM は、MR79_RAM セクションにあります。

MR79_RAM セクションにおいて MR79 が使用する RAM 容量は表 7.1で計算することができます。ただし、システムおよびタスクの使用するスタックは含まれていません。

スタックサイズの計算はリファレンスマニュアルを参照してください。

表 7.1 MR_RAM セクションのサイズ算出方法

領域名	バイト数
システム作業領域	5+優先度数（LM,SM モデル） 6+優先度数（LMI,SMI モデル）
システム時刻管理領域	6
タスク管理領域	11×タスク数(タイムアウトなし)
	15×タスク数(タイムアウトあり)
タイマキュー管理領域	6
周期起動ハンドラ管理領域	3×周期起動ハンドラ数
アラームハンドラ管理領域	1
イベントフラグ管理領域	5×イベントフラグ数
セマフォ管理領域	3×セマフォ数
固定長メモリプール管理領域	3×固定長メモリプール数
可変長メモリプール管理領域	ヒープサイズ分+49
メイルボックス管理領域	7×メイルボックス数 +メイルボックスバッファサイズ×2 (16 ビットの場合)or +メイルボックスバッファサイズ×3 (24 ビットの場合)

MR79 を使用する上で、最低限必要な RAM サイズは、LM モデルで 17 バイト、LMI モデルで 18 バイトとなります。

また、1 タスク増加につき、タイムアウト機能を使用しない場合、11 バイト必要になり、タイムアウト機能を使用した場合は、15 バイト必要になります

7.3 スタックについて

7.3.1 システムスタックとユーザースタック

MR79 のスタックにはシステムスタックとユーザースタックがあります。

- ユーザースタック
  タスクごとに１つずつ存在するスタックです。したがって MR79 を用いてアプリケーションを記述する場合はタスクごとのスタック領域を確保する必要があります。

- システムスタック
  MR79 内部（システムコール処理中）に使用されるスタックです。MR79 ではシステムコールをタスクが発行するとスタックをユーザースタックからシステムスタックに切り替えます。(図 7.4を参照してください。)

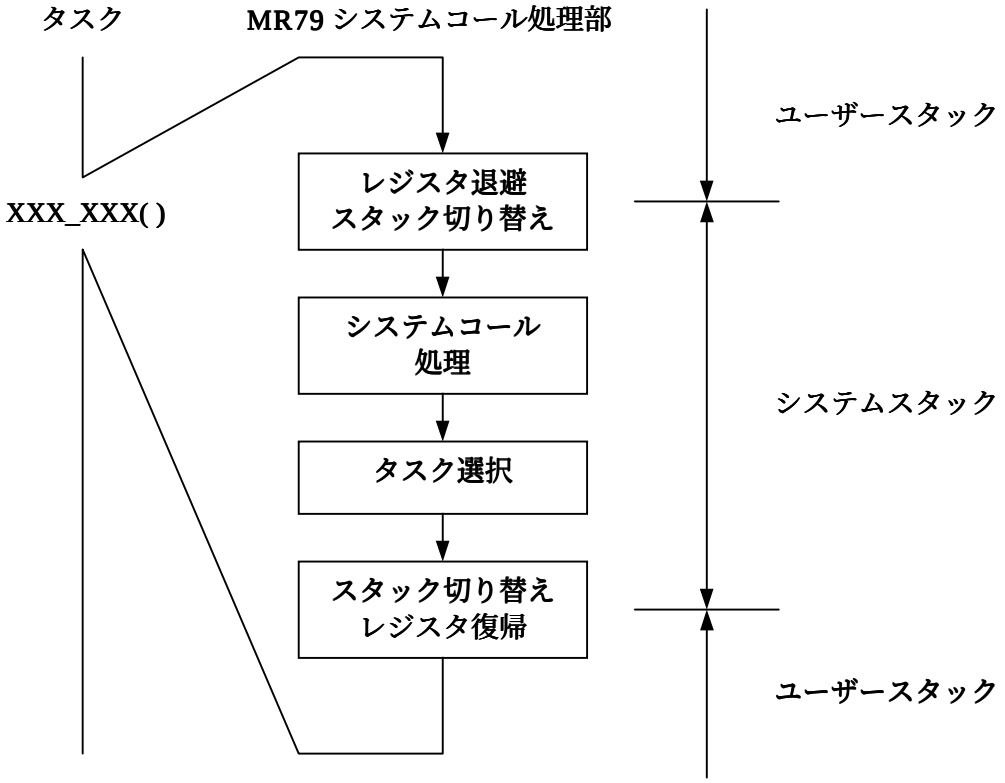


図 7.4　システムスタックとユーザースタック

また、割り込み発生時には、ユーザースタックからシステムスタックに切り替えます。したがって、割り込みハンドラで使用するスタックは全てシステムスタックを使用します。

7.4 MR79 V.2.00 への移行の手引き

MR79 V.1.10 から MR79 V.2.00 への移行方法について説明します。

7.4.1 スタートアッププログラムの変更

スタートアッププログラム(crt0mr.a79 又は start.a79)は、V.1.10 のものをそのまま使用できません。V.2.00 に含まれているスタートアッププログラムを使用してください。

また従来のスタートアッププログラムにユーザー独自の初期化などのプログラムの追加変更をおこなった場合は、V.2.00 に含まれているスタートアッププログラムにも追加変更をおこなってください。

7.4.2 セクションファイルの変更

セクションファイル(c_sec.inc 又は asm_sec.inc)は、 V.1.10 の MR79_KERNEL セクションの直後に、MR79_KERNEL_F というセクションを追加してください。

7.4.3 コンフィグレーションファイルの変更

V.1.10 のコンフィグレーションファイルを V.2.00 で使用するには、システム定義に "timeout" 項目を追加設定してください。

7.4.4 makefile の変更

makefile は V.1.10 のものをそのまま使用できません。

V.1.10 の makefile がカレントディレクトリにある場合、バックアップをとった後に削除してください。次にコンフィグレータを -m オプションを使用して起動して、V.2.00 の makefile をカレントディレクトリに作成してください。V.1.10 の makefile のバックアップを参照し、ユーザー記述部分を V.2.00 の makefile に追加してください。

第 8 章　　サンプルプログラムの説明

8.1 概要

MR79 の応用例として、7900 の各ポート (P0～P8) に接続した LED を点灯させるプログラムを示します。この応用例では各ポートの制御を、一つ一つの独立した関数により行なっています。表 2.1に各関数の一覧を示します。

表 8.1 サンプルプログラムの関数一覧

関数名	種類	ID 番号	優先度	機能
main()	タスク	1	1	task2 から順次 task8 まで起床させ、task6 と task7 を強制待ち状態にし、その後休止状態に移行します。
task2()	タスク	2	2	port0 の入出力を制御します。
task3()	タスク	3	3	port1 の入出力を制御します。
task4()	タスク	4	4	port2 の入出力を制御します。
task5()	タスク	5	5	port3 の入出力を制御します。
task6()	タスク	6	6	port4 の入出力を制御します。
task7()	タスク	7	7	port5 の入出力を制御します。
task8()	タスク	8	8	port7～port8 の入出力を制御します。
cyh1()	ハンドラ			port7 の出力データを変更します。
cyh2()	ハンドラ			port8 の出力データを変更します。
alh1()	ハンドラ			task7 の強制待ち状態を解除します
intr()	ハンドラ			task6 の強制待ち状態を解除します。

main タスクは、まずポート 0～5、ポート 7～8 を出力モードに設定します。ポート 6 は外部割り込みポートとして使用するため入力ポートに設定します。次に、変数 pt7～pt8 に 0x01 の値を設定し、変数 pcnt7～pcnt8 に初期値 1 を与えます。変数 pt7～pt8、変数 pcnt7～pcnt8 は、ポート 7～8 の値をハンドラで変更する場合に使用します。そして、task2 ～ task8 まで順次起動させ、その後休止状態に移行します。

task2 は、初期値 0xff をポート 0 に設定し (ポートをすべて点灯させる。)システムクロックが 15 回カウントするまで待ち状態に移行します。その後ポート 0 に 0x01 の値を設定し、LED を点灯させます。そして再びシステムクロックが 15 回カウントするまで待ち状態に移行し、ポート 0 のデータ値を 1 ビットシフトし点灯させるということを 8 回繰り返します。 (例 ００００００００１→００００００１０→００００１００) これを無限ループにします。

図 8.1にポート 0 の点灯の様子を示します。

task3～5 は、待ち時間が task2 と違うのみで他は同様の処理を行ないます。task7 は、main タスクにより強制待ち状態に移行し、アラームハンドラ alh() により解除されるまでプログラムは実行されません。解放された後は task2 と待ち時間が違うのみで他は同様の処理をおこないます。taks6 は、main タスクにより強制待ち状態に移行し、外部割り込みハンドラ intr() により解除されるまでプログラムは実行されません。解放された後は task2 と待ち時間が違うのみで他は同様の処理をおこないます。

task8 は、ポート 7～8 に初期値 pt7～8=1 に設定し直します。これも無限ループとしています。

変数 pcnt7～8 とポート 7～8 の値は周期起動ハンドラ cyh1()～cyh2()の中で変更されます。周期起動ハンドラは変数 pcnt7～8 の値を 1 ずつ増加させ、ポート 7～8 の値を 1 ビットずつシフトしていきます。

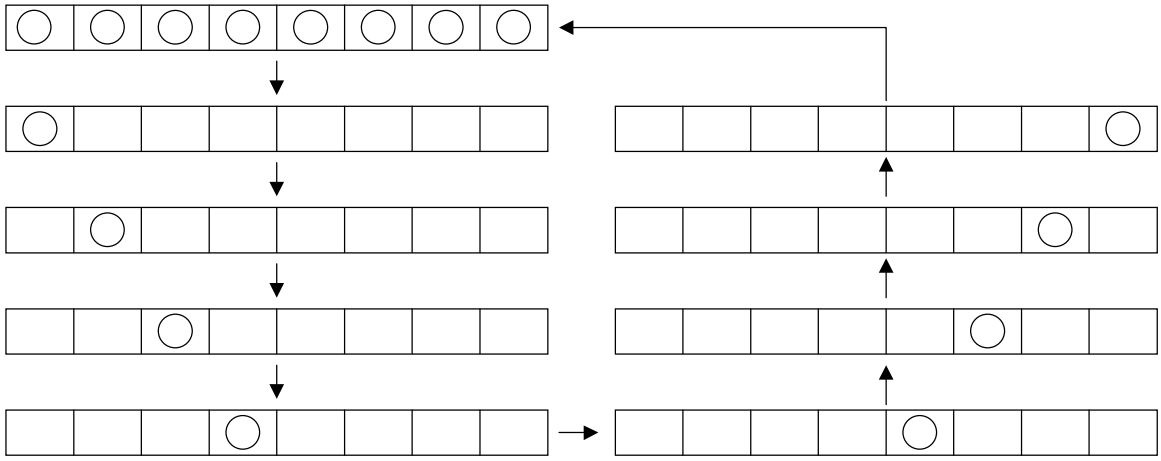


図 8.1　LED 点灯の様子

8.2 ソースプログラム

```
 1 /************************************************************************/
 2 /*                  MR79 smaple program                                 */
 3 /*                                                                      */
 4 /*  Copyright 1998 MITSUBISHI ELECTRIC CORPORATION                      */
 5 /*  AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION           */
 6 /*  All Rights Reserved                                                 */
 7 /*                                                                      */
 8 /*  $Id: demo.c,v 1.1 98/03/24 16:11:22 kuwata Exp $                    */
 9 /*                                                                     */
10 /************************************************************************/
11
12 #include <stdio.h>
13 #include "id.h"
14
15 #pragma EQU dir0 = 0x04
16 #pragma EQU dir1 = 0x05
17 #pragma EQU dir2 = 0x08
18 #pragma EQU dir3 = 0x09
19 #pragma EQU dir4 = 0x0c
20 #pragma EQU dir5 = 0x0d
21 #pragma EQU dir6 = 0x10
22 #pragma EQU dir7 = 0x11
23 #pragma EQU dir8 = 0x14
24 #pragma EQU port0 = 0x02
25 #pragma EQU port1 = 0x03
26 #pragma EQU port2 = 0x06
27 #pragma EQU port3 = 0x07
28 #pragma EQU port4 = 0x0a
29 #pragma EQU port5 = 0x0b
30 #pragma EQU port6 = 0x0e
31 #pragma EQU port7 = 0x0f
32 #pragma EQU port8 = 0x12
33
34 #pragma EQU intipl = 0x7d
35
36
37 char dir0,dir1,dir2,dir3,dir4,dir5,dir6,dir7,dir8;
38 char port0,port1,port2,port3,port4,port5,port6,port7,port8;
39 char pt0,pt1,pt2,pt3,pt4,pt5,pt6,pt7,pt8;
40 short pcnt7,pcnt8;
41 char intipl;
42
43 void  main(){
44
45     UINT    stacd;
46
47     dir0 = dir1 = dir2 = dir3 = dir4 = dir5 = dir7 = dir8 = 0xff;
48     dir6 = 0x00;
49
50     intipl = 0x1e;
51
52     pt7 = pt8 = 0x01;
53     pcnt7 = pcnt8 = 1;
54
55
56     sta_tsk(ID_task2,stacd);
57     sta_tsk(ID_task3,stacd);
58     sta_tsk(ID_task4,stacd);
59     sta_tsk(ID_task5,stacd);
60     sta_tsk(ID_task6,stacd);
61     sta_tsk(ID_task7,stacd);
```

```
 62     sta_tsk(ID_task8,stacd);
 63     sta_tsk(ID_idle,stacd);
 64
 65     sus_tsk(ID_task6);
 66     sus_tsk(ID_task7);
 67
 68 }
 69
 70 void task2(){
 71     int     k;
 72     port0 = 0xff;
 73
 74     while(1){
 75             dly_tsk(15);
 76             pt0 = 0x01;
 77             port0 = pt0;
 78             for(k=1;k<=8;k++){
 79                     dly_tsk(15);
 80                     pt0 = (pt0<<1);
 81                     port0 = pt0;
 82             }
 83     }
 84 }
 85
 86 void  task3(){
 87     int k;
 88     port1 = 0xff;
 89
 90     while(1){
 91             dly_tsk(25);
 92             pt1 = 0x01;
 93             port1 = pt1;
 94             for(k=1;k<=8;k++){
 95                     dly_tsk(25);
 96                     pt1 = (pt1<<1);
 97                     port1 = pt1;
 98             }
 99     }
100 }
101
102 void task4(){
103     int k;
104     port2 = 0xff;
105
106     while(1){
107             dly_tsk(45);
108             pt2 = 0x01;
109             port2 = pt2;
110             for(k=1;k<=8;k++){
111                     dly_tsk(45);
112                     pt2 = (pt2<<1);
113                     port2 = pt2;
114             }
115     }
116 }
117
118 void task5(){
119     int k;
120     port3 = 0xff;
121
122     while(1){
123             dly_tsk(65);
124             pt3 = 0x01;
125             port3 = pt3;
```

```
126             for(k=1;k<=8;k++){
127                     dly_tsk(65);
128                     pt3 = (pt3<<1);
129                     port3 = pt3;
130             }
131     }
132 }
133
134 void task6(){
135     int k;
136     port4 = 0xff;
137
138     while(1){
139             dly_tsk(75);
140             pt4 = 0x01;
141             port4 = pt4;
142             for(k=1;k<=8;k++){
143                     dly_tsk(75);
144                     pt4 = (pt4<<1);
145                     port4 = pt4;
146             }
147     }
148 }
149
150 void task7(){
151     int k;
152     port5 = 0xff;
153
154     while(1){
155             dly_tsk(85);
156             pt5 = 0x01;
157             port5 = pt5;
158             for(k=1;k<=8;k++){
159                     dly_tsk(85);
160                     pt5 = (pt5<<1);
161                     port5 = pt5;
162             }
163     }
164 }
165
166 void task8(){
167     while(1){
168             port7 = pt7;
169             port8 = pt8;
170             if(pcnt7 >= 9 ){pt7 = 1; pcnt7 = 1; }
171             if(pcnt8 >= 9 ){pt8 = 1; pcnt8 = 1; }
172     }
173 }
174
175 void cyh1(){
176     pcnt7 += 1;
177     pt7 = (pt7<<1);
178 }
179
180 void cyh2(){
181     pcnt8 += 1;
182     pt8 = (pt8<<1);
183 }
184
185 void alh1(){
186     irsm_tsk(ID_task7);
187 }
188
189 void intr()
190 {
```

```
191     irsm_tsk(ID_task6);
192
193 }
```

8.3 コンフィグレーションファイル

```
 1 //************************************************************************
 2 //
 3 //  Copyright 1998 MITSUBISHI ELECTRIC CORPORATION
 4 //  AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
 5 //
 6 //  MR79 System Configuration File.
 7 //  $Id: test.cfg,v 1.1 98/03/24 16:11:23 kuwata Exp $
 8 //
 9 //************************************************************************
10
11 //System Definition
12 system{
13        stack_size           =       1024;
14        priority             =       10;
15        message_size         =       16;
16        interrupt_model              =       STANDARD;
17        system_IPL           =       0;
18  };
19 //System Clock Definition
20 clock{
21        mpu_clock            =       16MHz;
22        timer                        =       A0;
23        IPL                  =       5;
24        unit_time            =       10ms;   // ms
25        initial_time         =       0:0:0;
26 };
27 //Task Definition
28 task[1]{
29     entry_address           =       main();
30     stack_size              =       512;
31     priority                =       1;
32     initial_start           =       ON;
33 };
34 task[2]{
35     entry_address           =       task2();
36     stack_size              =       512;
37     priority                =       2;
38 };
39 task[3]{
40     entry_address           =       task3();
41     stack_size              =       512;
42     priority                =       3;
43 };
44 task[4]{
45     entry_address           =       task4();
46     stack_size              =       512;
47     priority                =       4;
48 };
49 task[5]{
50     entry_address           =       task5();
51     stack_size              =       512;
52     priority                =       5;
53 };
54 task[6]{
55     entry_address           =       task6();
56     stack_size              =       512;
57     priority                =       6;
58 };
59 task[7]{
60     entry_address           =       task7();
61     stack_size              =       512;
```

```
62      priority                =       7;
63  };
64  task[8]{
65      entry_address           =       task8();
66      stack_size              =       512;
67      priority                =       8;
68  };
69  task[9]{
70      entry_address           =       idle();
71      stack_size              =       512;
72      priority                =       10;
73  };
74  //Cyclic Handler Definition
75  cyclic_hand[1]{
76      interval_counter        =       45;
77      mode                    =       TCY_ON;
78      entry_address           =       cyh1();
79  };
80  cyclic_hand[2]{
81      interval_counter        =       500;
82      mode                    =       TCY_ON;
83      entry_address           =       cyh2();
84  };
85  //Alarm Handler Definition
86  alarm_hand[1]{
87      time                    =       0:0:1000;
88      entry_address           =       alh1();
89  };
90  //Interrupt vector Definition
91  interrupt_vector[7]{
92      os_int                  =       YES;
93      entry_address           =       intr();
94  };
```

索引

L

M

N

O

P

R

S

T

U

V

W

M

あ

い

か

き

く

け

こ

さ

し

す

せ

た

て

に

は

ふ

へ

ま

め

ゆ

ら

り

れ

わ

技術サポート連絡書

年　月　日（合計　　枚）

三菱電機セミコンダクタシステム株式会社
マイコンソフトツール部
開発ツールサポート窓口行

［電子メール］ support@tool.mesc.co.jp
［大阪地区］ FAX：06-6398-6191
［東京地区］ FAX：03-5783-7339
［中部地区］ FAX：052-221-7318
［九州地区］ FAX：092-452-1427

※インストーラが生成する以下のテキストファイルもサポート連絡書としてご利用できます。
Windows 98/95/Windows NT 4.0版の場合：\SUPPORT\製品名\SUPPORT.TXT
EWS版の場合：/support/製品名/toolinfo.txt

ご連絡先	製品情報
会社名：	ソフトウェア：
部署名：	バージョン番号：V.
担当者名：	ライセンスID：
電話番号：	－　－　－　－
FAX番号：	ホストマシン：
電子メール：	OS：　　V.
通信欄：	

MEMO

MR79 V.2.20ユーザーズマニュアル

第1版：2000年12月16日発行
資料番号：MSD-MR79-U-001216

三菱電機株式会社
三菱電機セミコンダクタシステム株式会社

7900 シリーズ用　リアルタイム OS
MR79 V.2.20　ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668