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。

User's Manual

RENESAS

M3T-MR32R/4 V.4.00

ユーザーズマニュアル

M32R ファミリ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2004.12

- Microsoft、MS-DOS、Windows**および**Windows NT**は、米国**Microsoft Corporation**の米国およびその他の国における登録商標です。**
- HP-UX**は、米国**Hewlett-Packard Company**のオペレーティングシステムの名称です。**
- Sun 、Java **およびすべての**Java**関連の商標およびロゴは、米国およびその他の国における米国**Sun Microsystems, Inc.**の商標または登録商標です。**
- UNIX**は、**X/Open Company Limited**が独占的にライセンスしている米国ならびに他の国における登録商標です。**
- IBM**および**AT**は、米国**International Business Machines Corporation**の登録商標です。**
- HP 9000**は、米国**Hewlett-Packard Company**の商品名称です。**
- SPARC**および**SPARCstation**は、米国**SPARC International, Inc.**の登録商標です。**
- Intel, Pentium**は、米国**Intel Corporation**の登録商標です。**
- Adobe**および**Acrobat**は、**Adobe Systems Incorporated**（アドビシステムズ社）の登録商標です。**
- Netscape**および**Netscape Navigator**は、米国およびその他の諸国の**Netscape Communications Corporation**社の登録商標です。**
- **その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。**

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

- **本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズ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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズ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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。**
- **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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズ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。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、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、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又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、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズ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株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又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

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、ツール技術サポート窓口 support_tool@renesas.com **まで送信ください。**

¥SUPPORT¥**製品名**¥SUPPORT.TXT

株式会社ルネサス ソリューションズ
ツール技術サポート窓口 support_tool@renesas.com
ユーザ登録窓口 regist_tool@renesas.com
ホームページ http://www.renesas.com/jp/tools

はじめに

M3T-MR32R/4(**以下** MR32R **と略す**)は M32R **ファミリ**用のリアルタイム・オペレーティングシステム[1]です。MR32R は**μ**ITRON 仕様[2]に準拠しています。

本マニュアルは MR32R 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。各サービスコールの詳細な使用方法については「MR32R リファレンスマニュアル」を参照してください。

MR32R を使うために必要なこと

MR32R 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。

- M32R **ファミリ**クロスツールキット M3T-CC32R(**以下** CC32R **と略す**)

ドキュメント一覧

MR32R 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）

  MR32R 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）

  MR32R のサービス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。MR32R 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行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。**MR32R** 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

第 1 章　ユーザーズマニュアルの構成 1

第 2 章　概要 3

2.1　MR32R のねらい 4

2.2　TRON 仕様と MR32R 6

2.3　MR32R の特長 8

第 3 章　MR32R 入門 9

3.1　リアルタイム OS の考え方 10
3.1.1　リアルタイム OS の必要性 10
3.1.2　リアルタイム OS の動作原理 13

3.2　サービスコール 17
3.2.1　サービスコールの呼び出し 17
3.2.2　サービスコール処理 18
3.2.3　サービスコールにおけるタスクの指定方法 19

3.3　タスク 20
3.3.1　タスクの状態 20
3.3.2　タスクの優先度とレディキュー 25
3.3.3　タスクの優先度と待ち行列 26
3.3.4　タスクコントロールブロック (TCB) 27

3.4　システムの状態 30
3.4.1　タスクコンテキストと非タスクコンテキスト 30
3.4.2　ディスパッチ禁止/許可状態 31
3.4.3　CPU ロック/ロック解除状態 32
3.4.4　ディスパッチ禁止状態と CPU ロック状態 32

3.5　MR32R カーネルの構成 33
3.5.1　モジュール構成 33
3.5.2　モジュール概要 34

3.5.3 タスク管理機能 36
3.5.4 タスク付属同期機能 39
3.5.5 タスク例外機能 43
3.5.6 同期・通信機能 (セマフォ) 45
3.5.7 同期・通信機能 (イベントフラグ) 47
3.5.8 同期・通信機能 (データキュー) 49
3.5.9 同期・通信機能 (メールボックス) 51
3.5.10 拡張同期・通信機能 (メッセージバッファ) 53
3.5.11 拡張同期・通信機能 (ランデヴ) 58
3.5.12 メモリプール管理機能 62
3.5.13 時間管理機能 67
3.5.14 周期ハンドラ機能 69
3.5.15 アラームハンドラ機能 71
3.5.16 システム状態管理機能 72
3.5.17 割り込み管理機能 74
3.5.18 システム構成管理機能 74
3.5.19 拡張機能 75

3.6 サービスコール発行コンテキスト一覧 76

第 4 章 アプリケーション作成手順概要 81

4.1 概要 82

4.2 開発手順例 84
4.2.1 アプリケーションプログラムのコーディング 84
4.2.2 コンフィギュレーションファイル作成 86
4.2.3 コンフィギュレータ実行 92
4.2.4 システム生成 92
4.2.5 ROM 書き込み 92

第 5 章 アプリケーション作成手順詳細 94

5.1 C 言語によるコーディング方法 95
5.1.1 タスクの記述方法 95
5.1.2 割り込みハンドラの記述方法 98
5.1.3 周期起動ハンドラ、アラームハンドラの記述方法 99
5.1.4 タスク例外の記述方法 100

第 6 章 アプリケーション作成時の注意事項 101

6.1 初期起動タスクについて 102

6.2 動的生成・削除機能について 102

6.3 TRAP 命令の使用について 103

6.4 割り込みについて 104
6.4.1 割り込み制御方法 104
6.4.2 割り込みハンドラの処理手順 105
6.4.3 ハンドラ実行時の割り込みの受付について 106
6.4.4 多重割り込みの許可方法 106

6.5 OS デバッグ機能使用手順 107
6.5.1 OS デバッグ機能使用手順 107
6.5.2 OS デバッグ機能使用時の注意事項 107

6.6 システムクロックの設定について 109
6.6.1 システムクロック処理の登録について 109
6.6.2 タイマの自動設定について 109

6.7 ベースレジスタ機能の使用方法 111

6.8 メモリ配置 112
6.8.1 メモリ配置に関する注意事項 114
6.8.2 カーネル領域の 16MB を超える空間への配置について 115

第 7 章 コンフィギュレータの使用方法 117

7.1 コンフィグレーションファイル内の表現形式 118

7.2 コンフィグレーションファイルの定義項目 121
7.2.1 システム定義 123
7.2.2 動的生成用ユーザスタック領域定義(内蔵 RAM) 125
7.2.3 動的生成用ユーザスタック領域定義(外部 RAM) 126
7.2.4 動的生成用データキュー領域定義(内蔵 RAM) 127
7.2.5 動的生成用データキュー領域定義(外部 RAM) 128
7.2.6 動的生成用メッセージバッファ領域定義(内蔵 RAM) 129
7.2.7 動的生成用メッセージバッファ領域定義(外部 RAM) 130
7.2.8 動的生成用固定長メモリプール領域定義(内蔵 RAM) 131
7.2.9 動的生成用固定長メモリプール領域定義(外部 RAM) 132
7.2.10 動的生成用可変長メモリプール領域定義(内蔵 RAM) 133
7.2.11 動的生成用可変長メモリプール領域定義(外部 RAM) 134
7.2.12 システムクロック定義 135
7.2.13 最大項目数定義 137
7.2.14 タスク定義 140
7.2.15 セマフォ定義 143
7.2.16 イベントフラグ定義 145
7.2.17 データキュー定義 147
7.2.18 メールボックス定義 149
7.2.19 メッセージバッファ定義 151
7.2.20 ランデヴ定義 153
7.2.21 固定長メモリプール定義 155
7.2.22 可変長メモリプール定義 157
7.2.23 周期ハンドラ定義 159

7.2.24 アラームハンドラ定義 ... 161
7.2.25 使用サービスコール定義 ... 162
7.2.26 割り込みベクタ定義 ... 163

7.3 コンフィグレーションファイル例 ... 164

7.4 コンフィグレータの実行 ... 169
7.4.1 コンフィグレータ概要 ... 169
7.4.2 コンフィグレータの環境設定 ... 171
7.4.3 コンフィグレータ起動方法 ... 172
7.4.4 makefile 生成機能 ... 173
7.4.5 コンフィグレータ実行上の注意 ... 174
7.4.6 コンフィグレータのエラーと対処方法 ... 175

第 8 章 各設定ファイルのカスタマイズ方法 ... 179

8.1 割り込み制御プログラムについて ... 180
8.1.1 割り込み制御プログラムの処理内容 ... 180
8.1.2 割り込み制御プログラム例 ... 182

8.2 MR32R スタートアッププログラムのカスタマイズ方法 ... 184
8.2.1 ROM から RAM へのデータ転送について ... 185
8.2.2 ROM から RAM への転送を止める方法について ... 186
8.2.3 C 言語用スタートアッププログラム(crt0mr.ms) ... 187

8.3 セクションファイルのカスタマイズ ... 192

8.4 makefile の編集 ... 193

第 9 章 アプリケーション作成の手引き ... 195

9.1 ハンドラからのサービスコールの処理手順 ... 196
9.1.1 タスク実行中に割り込んだハンドラからのサービスコール ... 196
9.1.2 サービスコール処理中に割り込んだハンドラからのサービスコール ... 197
9.1.3 ハンドラ実行中に割り込んだハンドラからのサービスコール ... 198

9.2 システムの使用する RAM 容量の計算方法 ... 199

9.3 スタックについて ... 201
9.3.1 システムスタックとユーザスタック ... 201

第 10 章 付録 ... 203

10.1 サンプルプログラム ... 204
10.1.1 サンプルプログラム概要 ... 204
10.1.2 サンプルプログラムソース ... 205
10.1.3 コンフィギュレーションファイル ... 206

図目次

図 3.1 プログラムサイズと開発期間 10
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) 11
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) 12
図 3.4 タスクの時分割動作 13
図 3.5 タスクの中断と再開 14
図 3.6 タスクの切り替え 14
図 3.7 タスクのレジスタ領域 15
図 3.8 実際のレジスタとスタック領域の管理 16
図 3.9 サービスコール 17
図 3.10 サービスコールの処理の流れ 18
図 3.11 タスクの識別 19
図 3.12 タスクの状態 20
図 3.13 MR32R のタスク状態遷移図 21
図 3.14 レディーキュー 25
図 3.15 TA_TPRI 属性の待ち行列 26
図 3.16 TA_TFIFO 属性の待ち行列 26
図 3.17 タスクコントロールブロック 29
図 3.18 周期起動ハンドラ、アラームハンドラの起動 31
図 3.19 MR32R の構成 33
図 3.20 タスクのリセット 37
図 3.21 優先度の変更 37
図 3.22 待ちキューのつなぎ換え 38
図 3.23 起床要求の蓄積 39
図 3.24 起床要求のキャンセル 40
図 3.25 タスクの強制待ちと再開 40
図 3.26 タスクの強制待ちと強制再開 41
図 3.27 DLY_TSK サービスコール 42
図 3.28 タスク例外処理の動作例 43
図 3.29 セマフォによる排他制御 45
図 3.30 セマフォカウンタ 45
図 3.31 セマフォによるタスクの実行制御 46
図 3.32 イベントフラグによるタスクの実行制御 48
図 3.33 データキュー 49
図 3.34 メールボックス 51
図 3.35 メッセージキュー 52
図 3.36 メッセージバッファ 53
図 3.37 メッセージの送信例 54
図 3.38 メッセージの送信 55
図 3.39 メッセージの受信 56

図 3.40　ランデヴ 58
図 3.41　複数のランデヴ 59
図 3.42　ランデヴの回送 60
図 3.43　ランデヴの返答 61
図 3.44　固定長メモリプールの獲得 63
図 3.45　GET_MPL 処理 64
図 3.46　REL_MPL 処理 65
図 3.47　メモリブロックの解放 66
図 3.48　タイムアウト処理 67
図 3.49　起動位相を保存する場合の動作 69
図 3.50　起動位相を保存しない場合の動作 69
図 3.51　アラームハンドラの動作 71
図 3.52　ROT_RDQ サービスコールによるレディキューの操作 72
図 4.1　MR32R システム生成詳細フロー 83
図 4.2　プログラム例 85
図 4.3　GUI コンフィギュレータ 86
図 4.4　コンフィギュレーションファイルの生成 86
図 4.5　コンフィギュレーションファイル出力例 91
図 4.6　コンフィギュレータ実行 92
図 4.7　システム生成 92
図 5.1　C 言語で記述したタスクの例 95
図 5.2　C 言語で記述した無限ループタスクの例 96
図 5.3　C 言語で記述した割り込みハンドラの例 98
図 5.4　C 言語で記述した周期起動ハンドラの例 99
図 5.5　C 言語で記述したタスク例外の例 100
図 6.1　サービスコール内での割り込み制御 104
図 6.2　割り込みハンドラの処理手順 105
図 6.3　OS カーネルの 16MB 超領域への配置 115
図 7.1　コンフィグレータ動作概要 170
図 8.1　割り込み制御プログラムのスタックの状況 181
図 8.2　C 言語用スタートアッププログラム 191
図 8.3　サンプルセクションファイルのメモリ配置 192
図 9.1　タスク実行中に割り込んだ割り込みハンドラからのサービスコール処理手順 196
図 9.2　サービスコール処理中に割り込んだ割り込みハンドラからのサービスコール処理手順 197
図 9.3　多重割り込みハンドラからのサービスコール処理手順 198
図 9.4　システムスタックとユーザスタック 201

表目次

表 2-1　MR32R 概略仕様 7
表 3-1　　タスクコンテキストと非タスクコンテキスト 30
表 3-2　CPU ロック状態で使用可能なサービスコール 32
表 3-3　DIS_DSP,LOC_CPU に関する CPU ロック、ディスパッチ禁止状態遷移 32
表 3-4　　タスク、ハンドラから発行できるサービスコール一覧 76
表 5-1　C 言語における変数の扱い 97
表 6-1　動的生成・削除サービスコール一覧 102
表 7-1　　数値表現例 118
表 7-2　　演算子 119
表 7-3　　使用マイコンとテンプレートファイル名との対応 136
表 8-1　　割り込み制御プログラムの概要 180
表 9-1　MR_RAM セクションのサイズ算出方法 199
表 9-2　MR_ROM セクションのサイズ算出方法 200
表 10-1　　サンプルプログラムの関数一覧 204

第 1 章 ユーザーズマニュアルの構成

MR32R ユーザーズマニュアルは、10 の章から構成されています。

- 第 2 章　概要

  MR32R の目的や、概略の機能、位置づけなどを説明します。

- 第 3 章　MR32R 入門

  MR32R 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

  MR32R 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

  MR32R 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 アプリケーション作成時の注意事項

  MR32R 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注意事項を説明します。

- 第 7 章　コンフィギュレータの使用方法

  コンフィギュレーションファイルの記述方法、および、コンフィギュレータの使用方法を詳細に説明します。

- 第 8 章　各設定ファイルのカスタマイズ方法

  スタートアップファイル、割り込み制御プログラム、セクションファイルについて説明します。

- 第 9 章　アプリケーション作成の手引き

  MR32R 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 10 章　付録

  製品にソースファイル形式で含まれている MR32R 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 MR32R 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。**

   このためには 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 32 ビットマイクロプロセッサ M32R **ファミリ**用にリアルタイム OS MR32R を開発しました。MR32R を導入することにより以下のような効果があります。

4. **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。

5. **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始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。

6. **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

[3] OS：Operating System

けでデバッグすることができます。

7. タイマ制御が簡単になります。

従来例えば、10mSec 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。

8. 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。

9. 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。

10. 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 MR32R

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、1989 年に仕様が規定されたもので、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。

1993 年には、μITRON 仕様 V.2.0 と ITRON 仕様を統合し、接続機能を追加したμITRON 仕様 V.3.0 が規定されました。

さらに、1999 年には、互換性の強化を図ったμITRON 仕様 V.4.0 が規定されました。

MR32R は、μITRON 仕様 V.4.0 に準拠した 32 ビットマイクロプロセッサ M32R ファミリ用に開発された、リアルタイム・オペレーティングシステムです。μITRON 仕様 V.4.0 では、ソフトウェアの移植性を確保するための試みとしてスタンダードプロファイルを規定していますが、MR32R は、スタンダードプロファイルのうち、静的 API を除くすべてのサービスコールをインプリメントしています。

MR32R の概略仕様を表 2-1に示します。

表 2-1　MR32R 概略仕様

項目	仕様
ターゲットマイクロプロセッサ	M32R ファミリ
最大タスク数	1024
タスクの優先度数	255
最大イベントフラグ数	1024
イベントフラグの幅	32 ビット
最大セマフォ数	1024
セマフォの形式	計数型
最大メールボックス数	1024
メッセージサイズ	32 ビット
メールボックスのバッファサイズ	4 バイト以上
最大データキュー数	1024
最大メッセージバッファ数	1024
最大ランデヴ用ポート数	1024
最大固定長メモリプール数	1024
最大可変長メモリプール数	1024
最大周期ハンドラ数	1024
最大アラームハンドラ数	1024
サービスコール数	181
OS 核記述言語	C 言語、アセンブリ言語

2.3 MR32R の特長

MR32R 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

   MR32R 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 MR32R 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

   MR32R は M32R ファミリ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のもつ機能により、数ある MR32R 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 CC32R を用いて、MR32R 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 MR32R 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添付されています。

5. 上流工程ツール〝コンフィギュレータ〟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ギュ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 MR32R 入門

本章では、MR32R を使用する上で必要な知識や考え方を説明し、MR32R が持つ機能の使用方法を説明します。

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。[4]

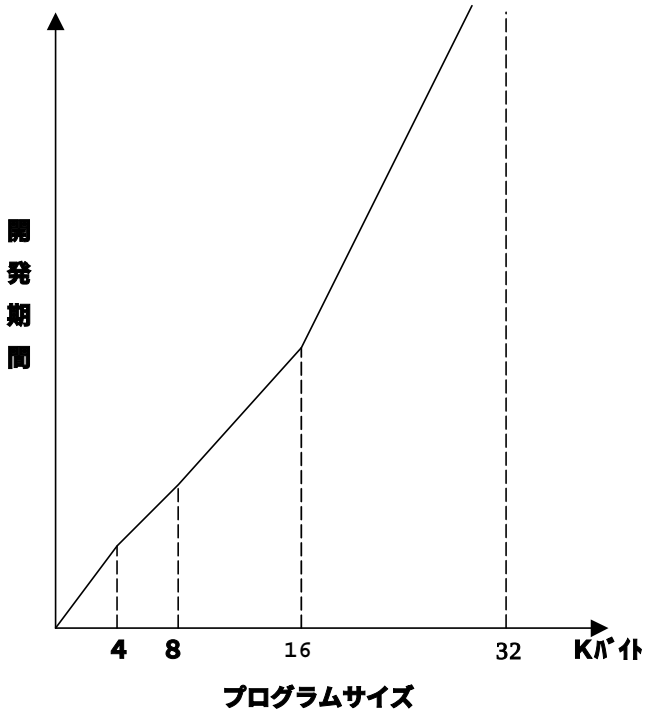


図 3.1　プログラムサイズと開発期間

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。

[4] ROM 詰めの必要がないことを前提とします

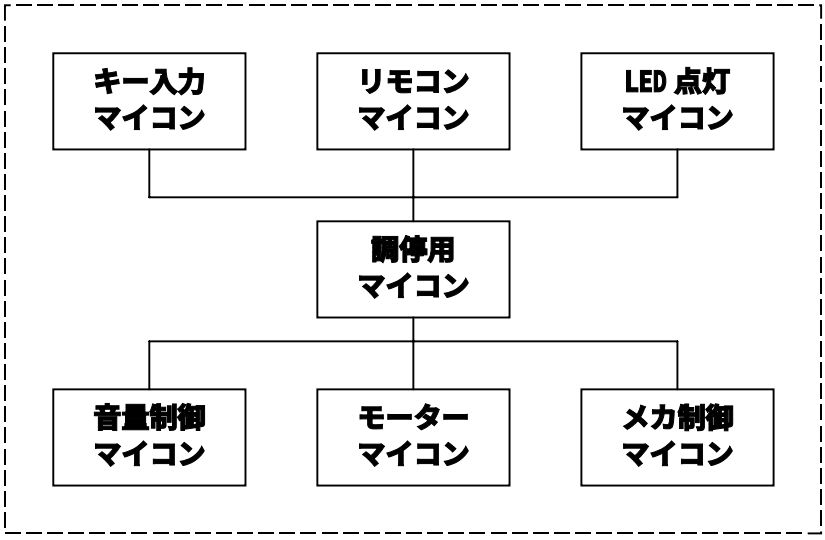


図 3.2　マイコンを多く使ったシステム例（オーディオ機器）

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。[5]
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。

[5] 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。

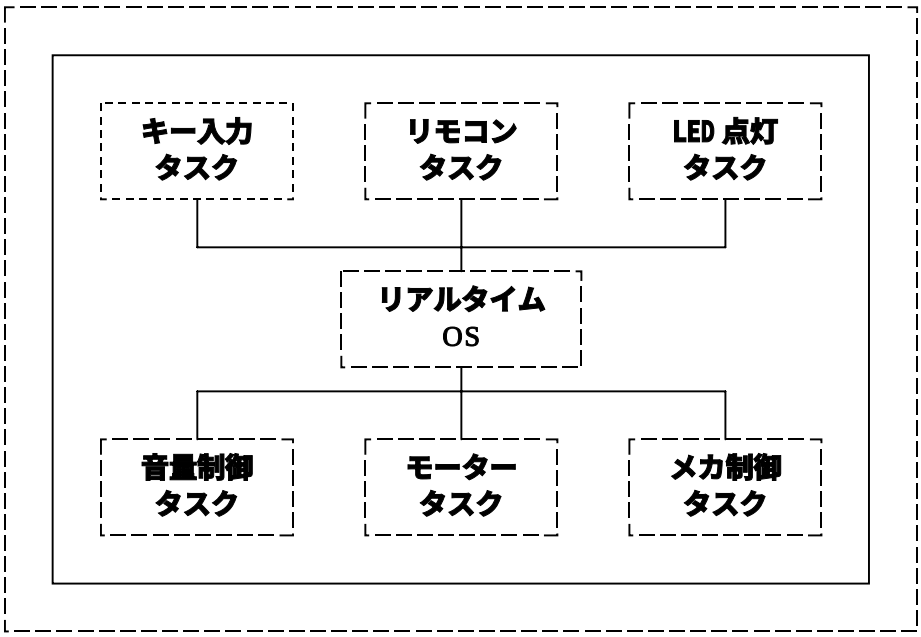


図 3.3　リアルタイム OS の導入システム例（オーディオ機器）

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。

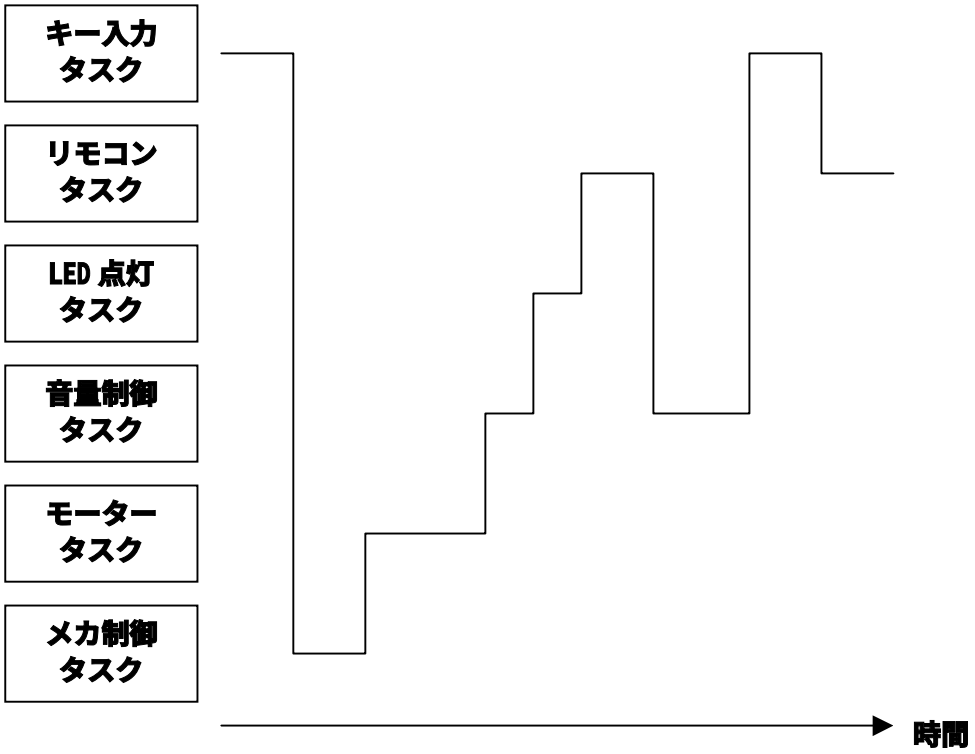


図 3.4　タスクの時分割動作

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)

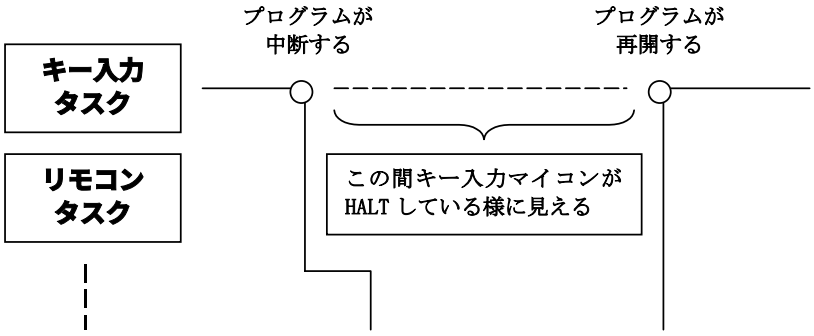


図 3.5　タスクの中断と再開

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）。

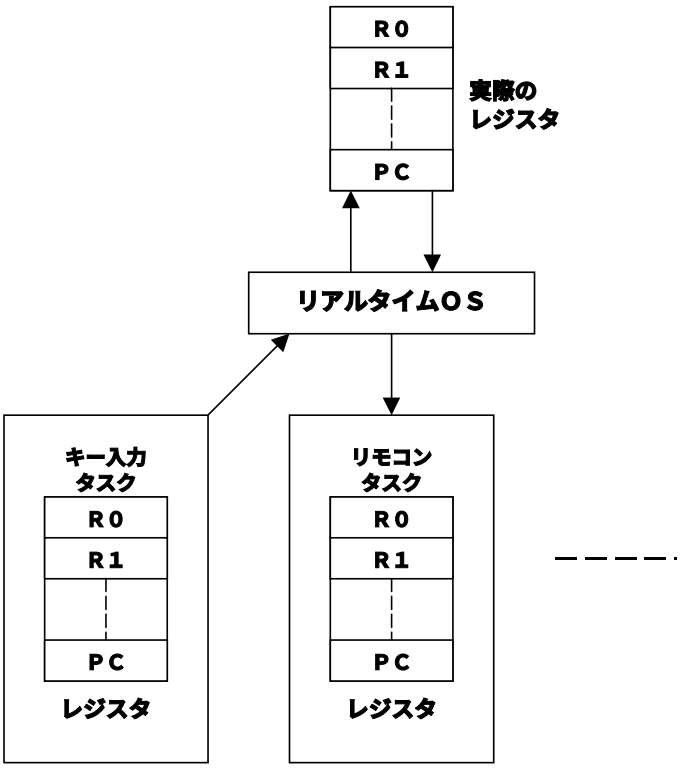


図 3.6　タスクの切り替え

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。

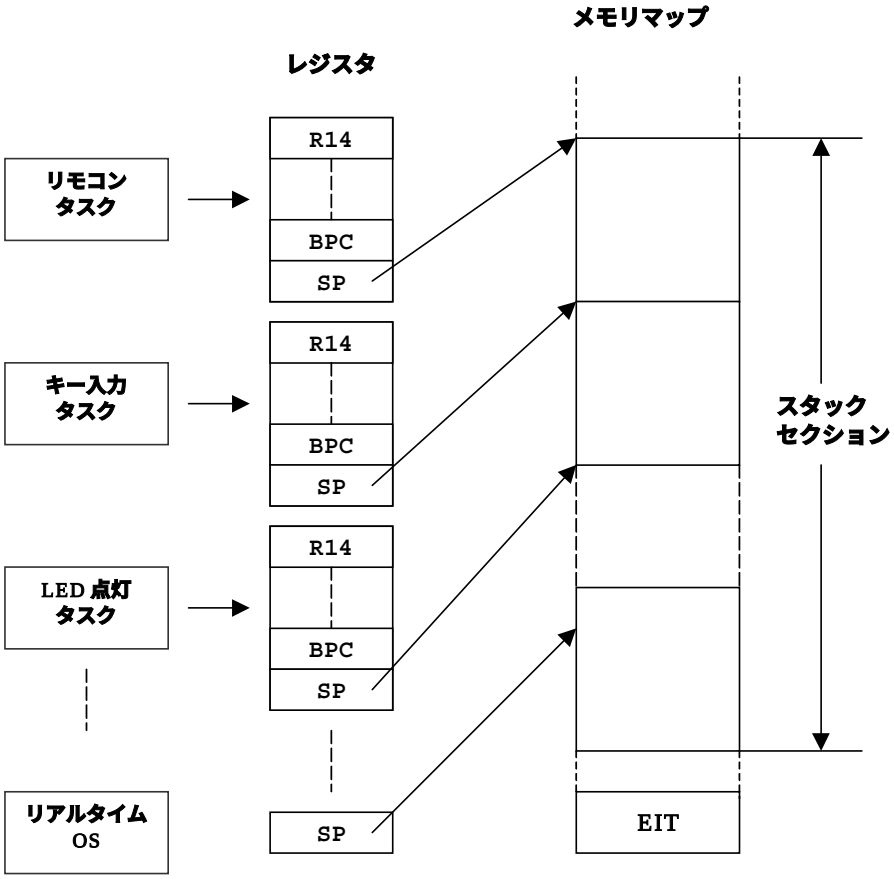


図 3.7　タスクのレジスタ領域

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。MR32R 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。

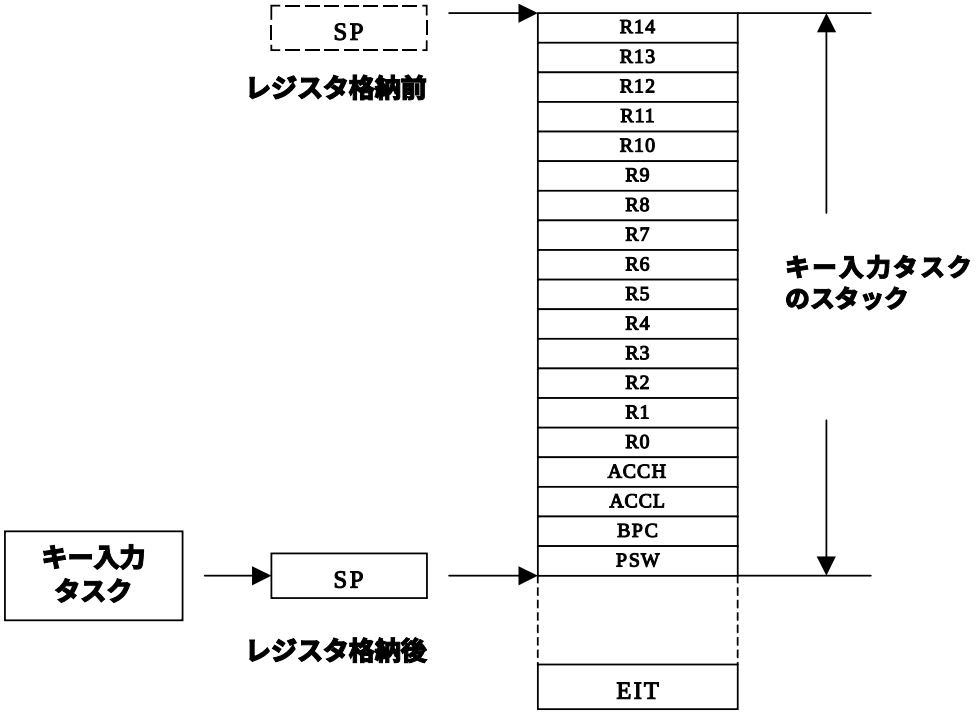


図 3.8　実際のレジスタとスタック領域の管理

3.2 サービスコール

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。このリアルタイム OS の機能を呼び出すことをサービスコール[6]といいます。すなわちサービスコールのにより、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)。

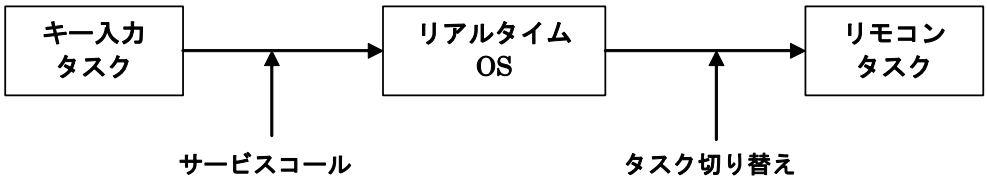


図 3.9　サービスコール

このサービスコールは、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　サービスコールの呼び出し

μITRON 仕様 V.4.0 では、非タスクコンテキスト専用のサービスコールは、先頭に"i"をつけるようになっています。例えば、"act_tsk"サービスコールは非タスクコンテキストから発行する場合は、先頭に"i"を付けて"iact_tsk"として呼び出します。

MR32R においても ITRON 仕様で決められたとおり、非タスクコンテキスト専用のサービスコールは、先頭に"i"をつけてサービスコールを発行することが可能です。また、MR32R では、"i"つきサービスコールと"i"なしサービスコールでは同じルーチンを使用しており、両者に違いはないため、非タスクコンテキストからも"i"なしサービスコールを使用することができます。

例えば、μITRON 仕様では、タスクを起動する際、非タスクコンテキストからは、"iact_tsk"を使用することになっていますが、MR32R では、"iact_tsk"と"act_tsk"のいずれも使用することができます。

しかし、他のμITRON 仕様 OS との互換性を保つため、非タスクコンテキスト専用のサービスコールを使用する場合は、先頭に"i"をつけて使用することを推奨します。

[6] μITRON 仕様 V.3.0 では、システムコールと呼んでいましたが、μITRON 仕様 V.4.0 では、ソフトウェア部品への対応を考慮してサービスコールと呼ぶようになりました。

3.2.2 サービスコール処理

サービス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。[7]

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. サービス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。

サービス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。

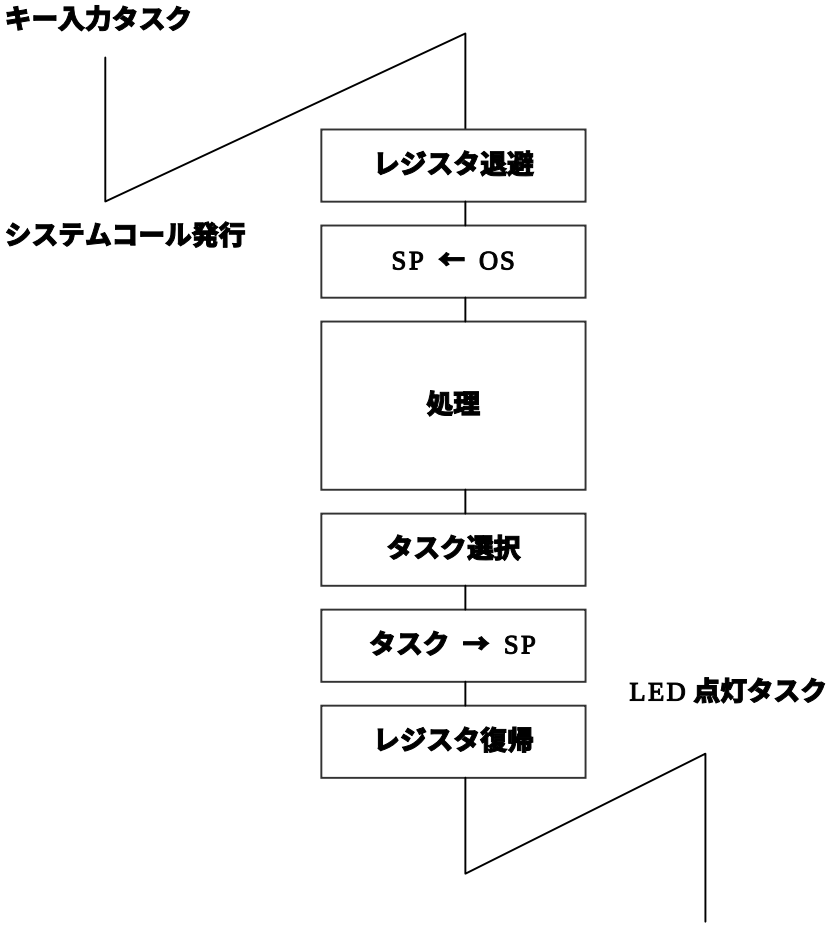


図 3.10 サービスコールの処理の流れ

[7] 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。

3.2.3 サービス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 MR32R 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 MR32R 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 MR32R に付属しているプログラム"コンフィギュレータ cfg32r"が自動的におこないます。図 3.11にその様子を示します。

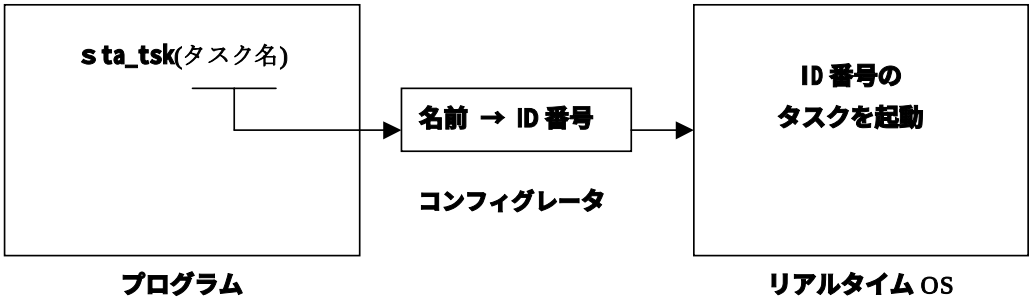


図 3.11 タスクの識別

これによりタスクの指定を以下のように行えます。

```
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、タスクの名前から ID 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 MR32R 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。

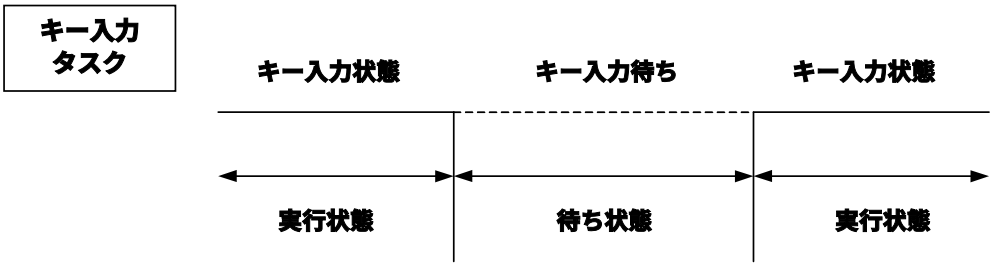


図 3.12 タスクの状態

MR32R では実行状態、待ち状態を含め以下の 7 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)
7. 未登録状態(NON-EXISTENT 状態)

タスクは上記の 7 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。

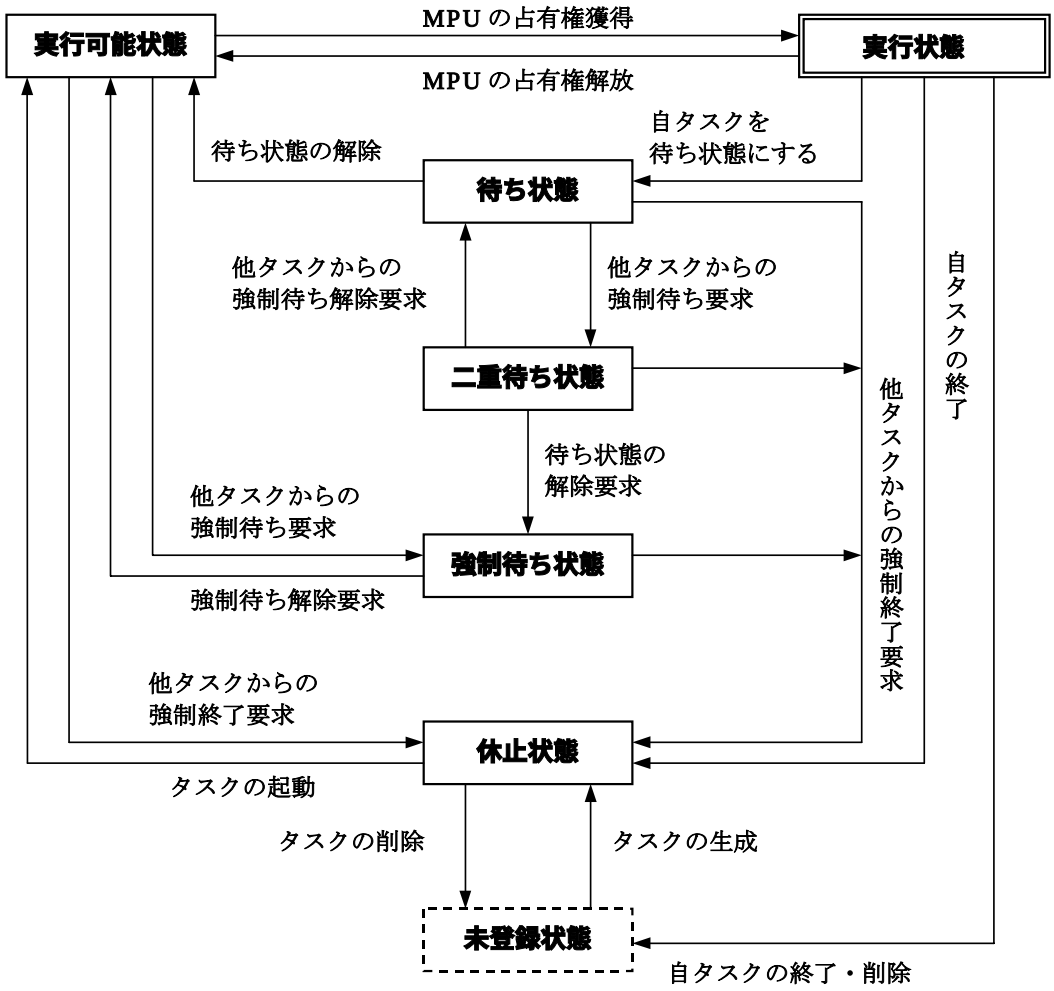


図 3.13 MR32R のタスク状態遷移図

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[8]
- 自分で待ち状態に入った場合
- 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[9]
- 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[10]

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。

[8] ext_tsk,exd_tsk システムコールによる
[9] chg_pri システムコールによる
[10] 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[11]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
- 実行状態のタスクが自分で待ち状態に入った場合
- 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[12]
- 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[13]

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 サービス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、tslp_tsk サービス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、snd_dtq、rcv_dtq、rcv_mbx、snd_mbf、rcv_mbf、cal_por、acp_por、get_mpf、get_mpl サービス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、tsnd_dtq、trcv_mdtq、trcv_mbx、tsnd_mbf、trcv_mbf、tcal_por、tacp_por、tget_mpf、tget_mpl サービスコールは、slp_tsk、wai_flg、wai_sem、rcv_dtq、rcv_mbx、snd_mbf、rcv_mbf、cal_por、acp_por、get_mpf、get_mpl サービスコールにタイムアウトを指定したサービスコールです。各サービス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、snd_dtq、rcv_dtq、rcv_mbx、snd_mbf、rcv_mbf、cal_por、acp_por、get_mpf、get_mpl サービスコール[14]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。

[11] レディーキューについては次節参照
[12] chg_pri システムコールによる
[13] icg_pri システムコールによる
[14]twai_flg,twai_sem,trcv_msg システムコールも含まれます。

- イベントフラグ待ち行列
- セマフォ待ち行列
- メールボックス待ち行列
- データ送信待ち行列
- データ受信待ち行列
- メッセージバッファ送信待ち行列
- メッセージバッファ受信待ち行列
- ランデヴ用ポート呼出待ち行列
- ランデヴ用ポート受付待ち行列
- 固定長メモリブロック獲得待ち行列
- 可変長メモリブロック獲得待ち行列

4. 強制待ち状態（SUSPEND 状態）

実行状態のタスクから sus_tsk サービスコールが発行される、もしくはハンドラから isus_tsk サービスコールが発行されると、サービス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[15]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。

強制待ち要求のキューイング数の最大値は 32767 です。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、強制待ち要求はキューイングされ、そのネスト数が、32767 の場合に sus_tsk あるいは、isus_tsk サービスコールを発行するとエラーコード E_QOVR が返ります。

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、snd_dtq、rcv_dtq、rcv_mbx、snd_mbf、rcv_mbf、cal_por、acp_por、get_mpf、get_mpl サービス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。

[15] 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した場合は、実行状態から直接強制待ち状態に移行されます。この場合のみ、例外的に実行状態から強制待ち状態に移行しますので注意してください。

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 サービスコールにより起床する場合
- dly_tsk、tslp_tsk サービス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、snd_dtq、rcv_dtq、rcv_mbx、snd_mbf、rcv_mbf、cal_por、acp_por、get_mpf、get_mpl サービス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 サービス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が出されると待ち状態になります。ただし、強制待ち要求ネスト数が2以上の二重待ち状態のタスクにsus_tsk,isus_tskサービスコールを発行しても強制待ち要求ネスト数が１減じられ、強制待ち状態は解除されません。

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、MR32R 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[16]もしくは強制終了[17]により終了した場合

7. 未登録状態(NON-EXISTENT 状態)

タスクが生成される前、またはタスクが削除された後の MR32R システムに登録されていない状態です。タスクを生成[18]すると MR32R に登録され、休止状態に移行します。この状態になるには以下の２つの場合があります。

- 他タスクにより、休止状態のタスクが削除された場合[19]
- 実行状態のタスクが自タスクを終了し、さらに削除した場合[20]

[16] ext_tsk システムコール
[17] ter_tsk システムコール
[18] cre_tsk システムコール
[19] del_tsk システムコール
[20] exd_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。

MR32R 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 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。[21]

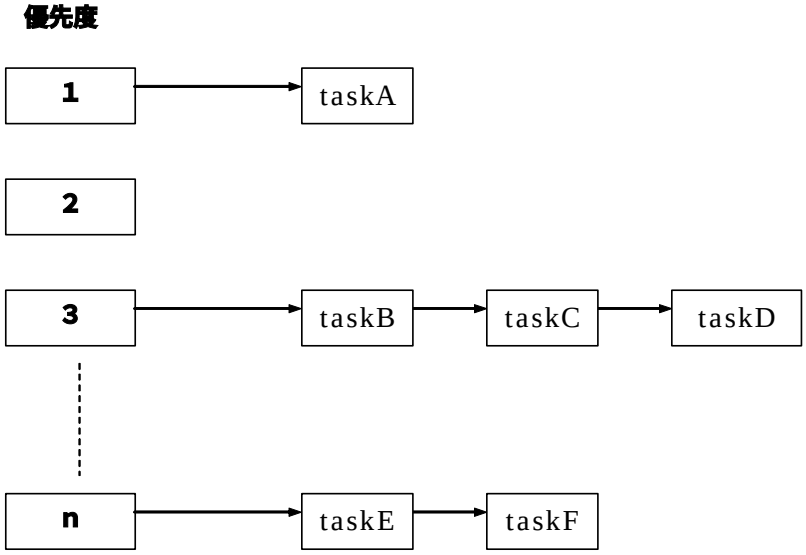


図 3.14 レディーキュー

[21] 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 タスクの優先度と待ち行列

μITRON V.4.0 仕様のスタンダードプロファイルでは、各オブジェクトの待ち方にタスクの優先度順に待ち行列をつなぐ(TA_TPRI 属性)、FIFO 順に待ち行列につなぐ(TA_TFIFO)の２種類をサポートすることになっています。MR32R でも、この２種類の待ち方をサポートしています[22]。

図 3.15、図 3.16 にタスクが、"taskD"、"taskC"、"taskA"、"taskB"の順で待ち行列につながれたときの様子を示します。

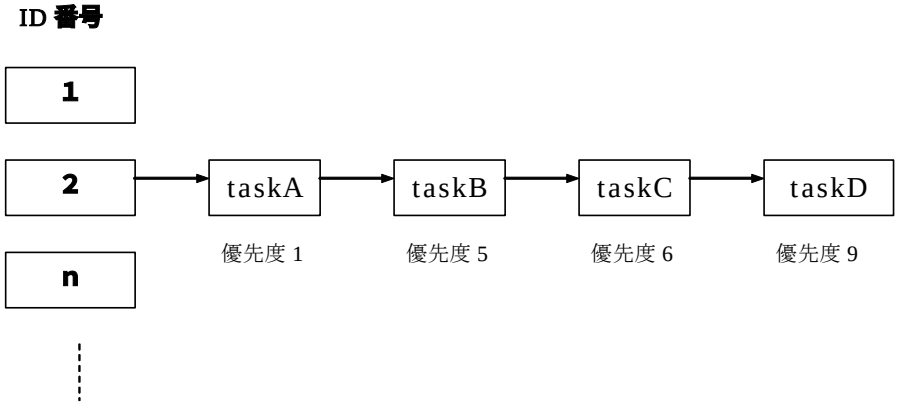


図 3.15 TA_TPRI 属性の待ち行列

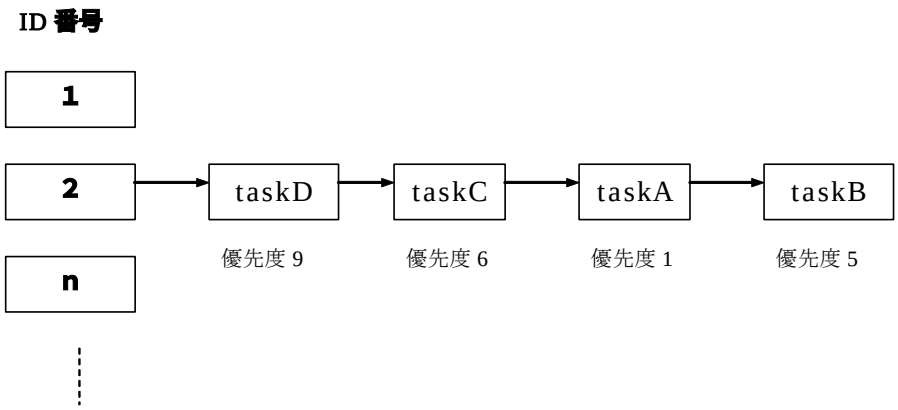


図 3.16 TA_TFIFO 属性の待ち行列

[22] 可変長メモリプール機能は、TA_TFIFO 属性のみのサポートです。

3.3.4 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。

MR32R 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

- タスクの状態

- タスクの優先度

- タスク属性

  タスクのスタック領域が内蔵 RAM あるいは外部 RAM かの情報をこの領域に格納します。

- タスクの拡張情報

  タスク生成時に設定する、タスクの拡張情報がこの領域に格納されます。

- タスクのレジスタ情報など[23]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）

- 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

- 起動要求カウンタ

  タスクの起動要求カウンタを蓄積する領域

- 強制待ち要求ネスト数

  強制待ち要求ネスト数を蓄積する領域

- メモリブロックサイズ

  タスクが可変長メモリブロック待ち状態にある場合、この領域に要求するメモリブロックサイズを格納します。

- ランデヴ待ちビットパターン

  ランデヴ呼出である場合に、呼出側選択条件ビットパターンを、また、受付待ち状態である場合に、受付選択条件ビット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

[23] これをタスクコンテキストと呼びます。

- フラグ待ちパターン

  タスクがイベントフラグ待ち状態の場合、フラグ待ちパターンが格納されます。

- 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。

- タイムアウトカウンタ

  タスクがタイムアウト待ち状態である場合に、残りの待ち時間が格納されます。

- 保留例外要因

  保留例外要因を格納します。

- タスク例外関連フラグ

  タスク例外の許可・禁止やタスク例外要求の有無を格納します。

- TCB 領域操作フラグ

  サービスコール中に一時的に割り込みを許可した際にフラグをクリアし、TCB 領域を操作した際にセットします。

- スタックベースアドレス

  スタック領域のベースアドレスを格納します。

タスクコントロールブロックを図 3.17に示します。

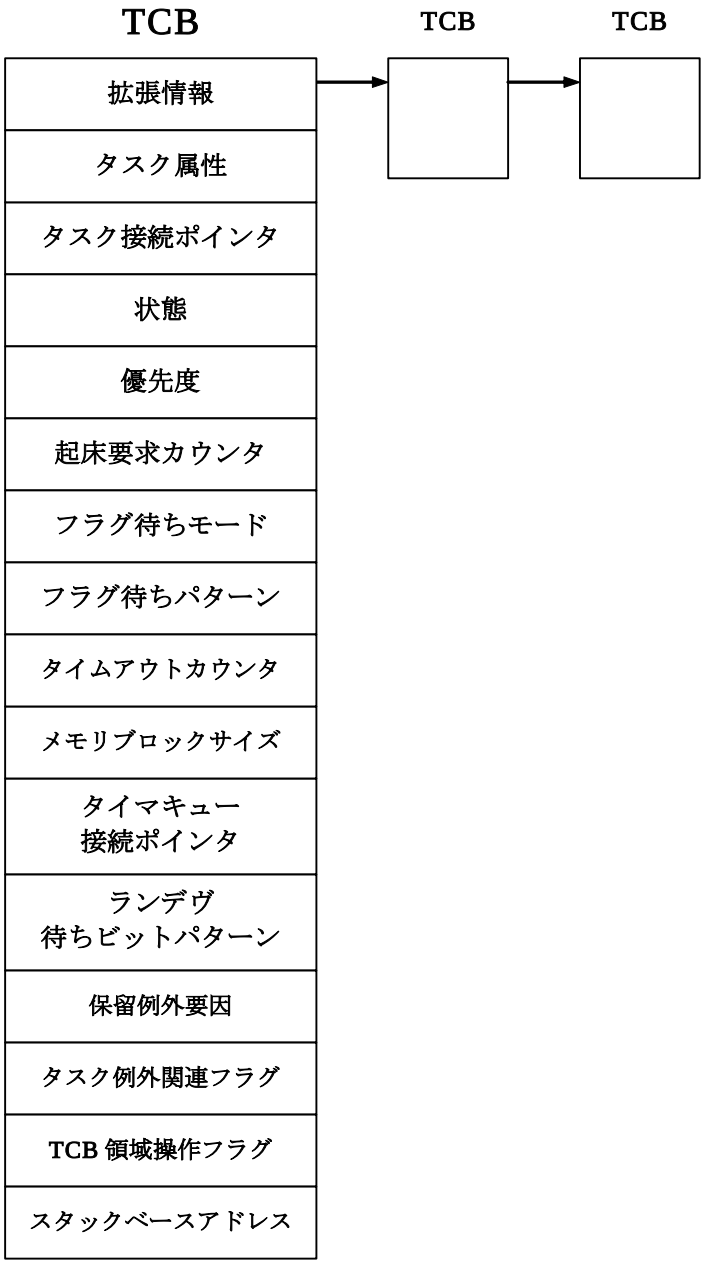


図 3.17 タスクコントロールブロック

3.4 システムの状態

3.4.1 タスクコンテキストと非タスクコンテキスト

システムは、「タスクコンテキスト」か「非タスクコンテキスト」のいずれかのコンテキスト状態で実行します。タスクコンテキストと非タスクコンテキストの違いを表 2-1に示します。

表 3-1 タスクコンテキストと非タスクコンテキスト

	タスクコンテキスト	非タスクコンテキスト
呼び出し可能なサービスコール	タスクコンテキストから呼び出せるもの	非タスクコンテキストから呼び出せるもの
タスクスケジューリング	キューの状態が変化し、ディスパッチ禁止状態、CPU ロック状態のいずれでもない場合に発生	発生しない
スタック	ユーザスタックスタック	システムスタック

非タスクコンテキストで実行される処理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は、割り込み入り口処理を行った後に、割り込み処理を行います。割り込み入り口処理は、MR32R カーネル内で行いますので、ユーザは、コンフィギュレーションファイルに登録するだけでかまいません。割り込み入り口処理については、5.6.1 を参照してください。

   システムクロック割り込みハンドラ(isig_tim)も割り込みハンドラに含まれます。

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 sta_cyc(ista_cyc)や stp_cyc(istp_cyc)サービスコールによりおこないます。

   また、周期ハンドラ起動時刻は、set_tim(iset_tim)による、時刻変化の影響を受けません。)

3. **アラームハンドラ**

   アラームハンドラは、 指定した相対時刻経過後に起動されるハンドラです。起動時刻は、sta_alm(ista_alm)設定時の時刻に対する相対時刻で決定され、set_tim(iset_tim)による、時刻変化の影響を受けません。)

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 3.18参照)。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。

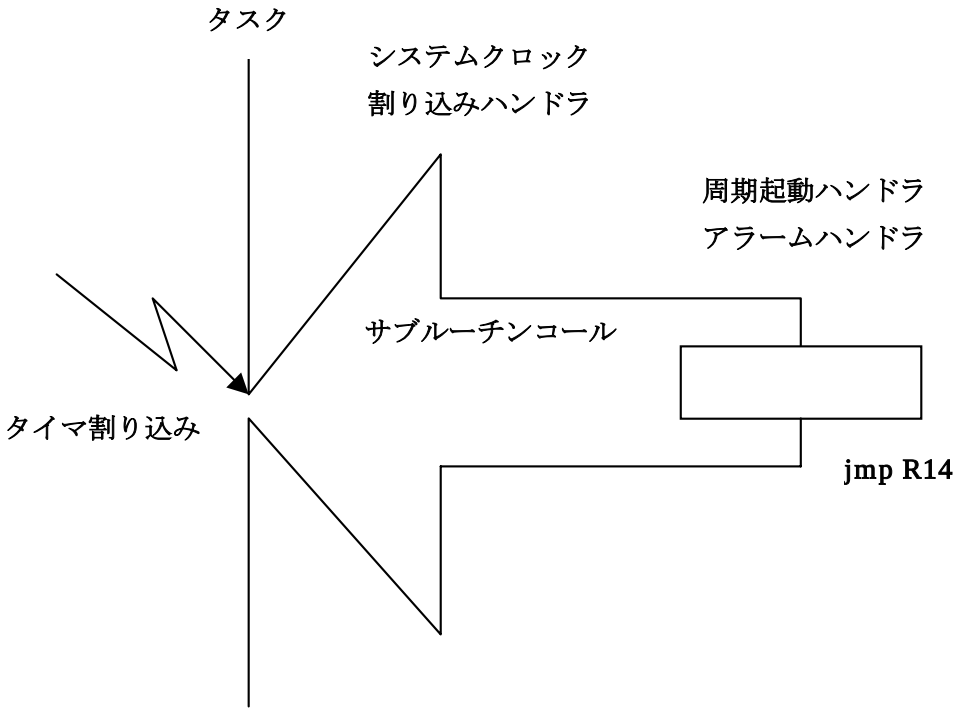


図 3.18 周期起動ハンドラ、アラームハンドラの起動

3.4.2 ディスパッチ禁止/許可状態

システムは、ディスパッチ許可状態、ディスパッチ禁止状態のいずれかの状態をとります。ディスパッチ禁止状態では、タスクスケジューリングが行われません。また、サービスコール発行タスクが待ち状態に移行するようなサービスコールも呼び出すことはできません。[24]

ディスパッチ禁止状態へは、dis_dsp サービスコール、ディスパッチ許可状態へは ena_dsp サービスコールの発行により遷移することができます。また、sns_dsp サービスコールによりディスパッチ禁止状態がどうか知ることができます。

[24] MR32R は、ディスパッチ禁止状態から発行できないサービスコールを発行してもエラーは返しませんが、その場合の動作は保証しません。

3.4.3 CPU ロック/ロック解除状態

システムは、CPU ロック状態が CPU ロック解除状態のいずれかの状態をとります。CPU ロック状態では、すべての外部割り込みの受付が禁止され、タスクスケジューリングも行われません。

CPU ロック状態へは loc_cpu(iclo_cpu)サービスコール、CPU ロック解除状態へは unl_cpu(iunl_cpu)サービスコール発行により遷移します。また、sns_loc サービスコールによって CPU ロック状態かどうか調べることができます。

CPU ロック状態から発行できるサービスコールは表 3-2のように制限されます。[25]

表 3-2 CPU ロック状態で使用可能なサービスコール

loc_cpu	iloc_cpu	unl_cpu	iunl_cpu
ext_tsk	exd_tsk	sns_tex	sns_ctx
sns_loc	sns_dsp	sns_dpn	

3.4.4 ディスパッチ禁止状態と CPU ロック状態

μITRON V.4.0 仕様では、ディスパッチ禁止状態と CPU ロック状態が明確に区別されるようになりました。従って、ディスパッチ禁止状態で unl_cpu サービスコールを発行したとしても、ディスパッチ禁止状態のまま変化せず、タスクスケジューリングは行われません。状態遷移をまとめると表 3-3のようになります。

表 3-3 dis_dsp,loc_cpu に関する CPU ロック、ディスパッチ禁止状態遷移

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
	CPU ロック状態	ディスパッチ禁止状態				
1	○	×	→ 2	→ 1	→ 1	→ 3
2	○	○	→ 2	→ 1	→ 2	→ 4
3	×	×	→ 4	→ 3	→ 1	→ 3
4	×	○	→ 4	→ 3	→ 2	→ 4

[25] MR32R は、CPU ロック状態から発行できないサービスコールを発行してもエラーは返しませんが、その場合の動作は保証しません。

3.5 MR32R カーネルの構成

3.5.1 モジュール構成

MR32R カーネルは、図 3.19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。

MR32R 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[26]から構成されます。

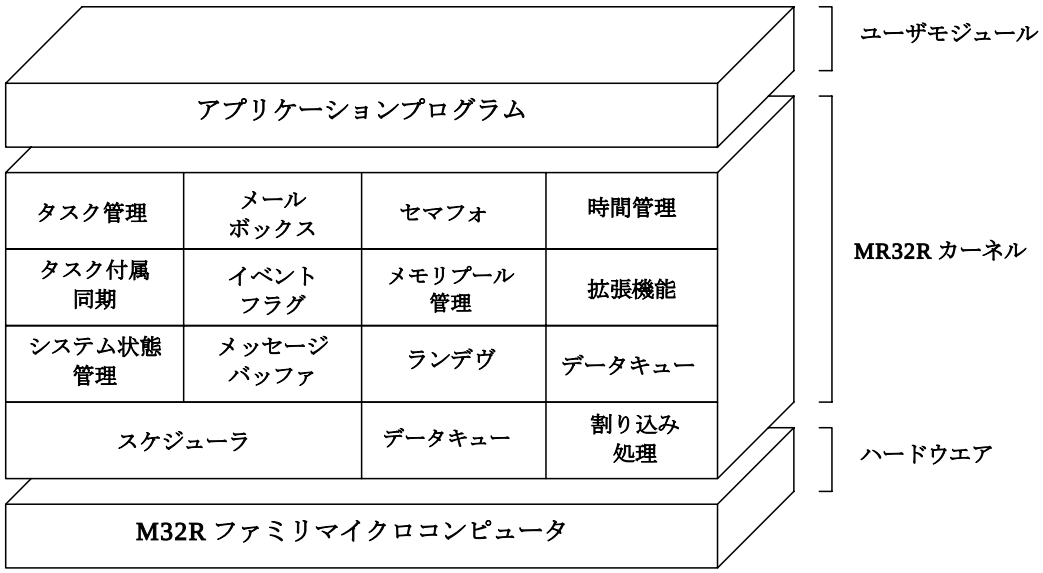


図 3.19 MR32R の構成

[26] 詳細は第3.5.13節を参照。

3.5.2 モジュール概要

MR32R 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

  MR32R カーネルで使用するシステムクロックの設定、タイムアウトの処理、ユーザの作成したアラームハンドラ[27]、周期起動ハンドラ[28]の起動をおこないます。

- システム構成管理

  システムの動的な情報を報告します。

- 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 4 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
    MR32R 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
    MR32R 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。
  - ♦ メールボックス
    タスク間のデータ通信をメッセージの先頭アドレスを渡すことにより行います。
  - ♦ データキュー
    タスク間のデータ通信を 1 ワードのデータを受け渡しすることにより行います。

- 拡張同期・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 2 つの機能モジュールが用意されています。

  - ♦ メッセージバッファ

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

タスク間の同期と通信を同時に行う機能で、メールボックスとは異なり、可変長のメッセージ内容をコピーして送受信する機能です。

- ランデヴ

  タスク間の待ち合わせを行う機能です。

- メモリプール管理機能

  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。

- 拡張機能

  OS 資源のリセット機能を提供します。

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。
MR32R カーネルが提供するタスク管理機能のサービスコールには、次のものがあります。

- タスクを生成する（cre_tsk）

  あるタスクから指定された ID のタスクを生成します。

- タスクを生成する（acre_tsk）

  あるタスクからタスクを生成します。生成するタスク ID 番号は、カーネルが自動的に割り当てます。

- タスクを削除する（del_tsk）

  あるタスクから指定された ID のタスクを削除します。

- タスクを起動する（act_tsk,iact_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。
  本サービスコールでは、sta_tsk(ista_tsk)と違い、起動要求は蓄積しますが、起動コードを指定することはできません。

- タスクを起動する（sta_tsk,ista_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。
  本サービスコールは、act_tsk(iact_tsk)と違い、起動要求は蓄積しませんが、起動コードを指定することができます。

- 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、このタスクは実行しません。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。その際、自タスクは、リセットされたように振る舞います。
  本サービスコールは、タスク終了時に明示的に記述されていなくても、タスクからリターンする際に自動的に呼び出されます。

- 自タスクを終了させ、削除する（exd_tsk）

  自タスクを正常終了させ、さらに自タスクを削除します。すなわち、自タスクを未登録状態に移行し、スタック領域を解放します。

- 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。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。
  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 3.20参照）

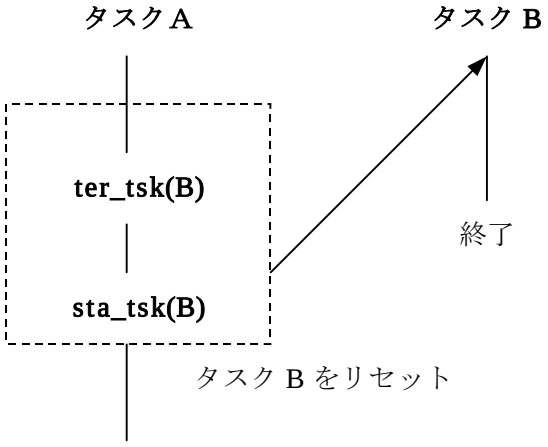
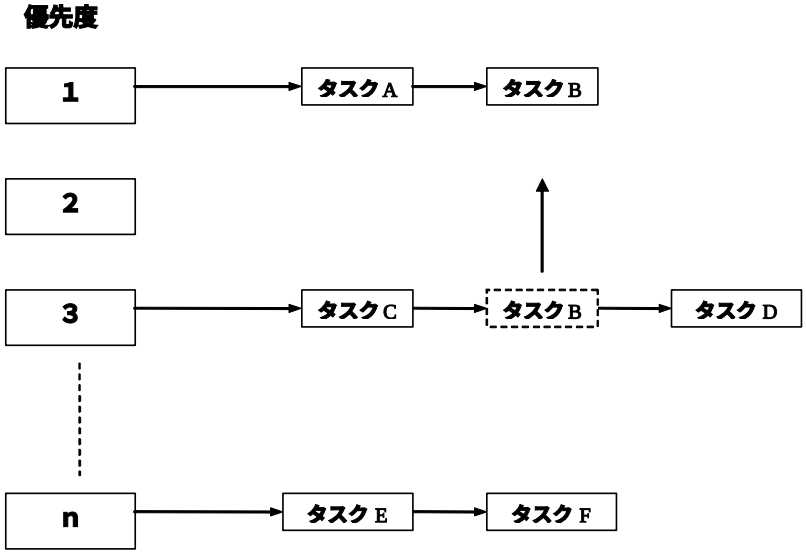


図 3.20 タスクのリセット

- タスクの優先度を変更する（chg_pri, ichg_pri）

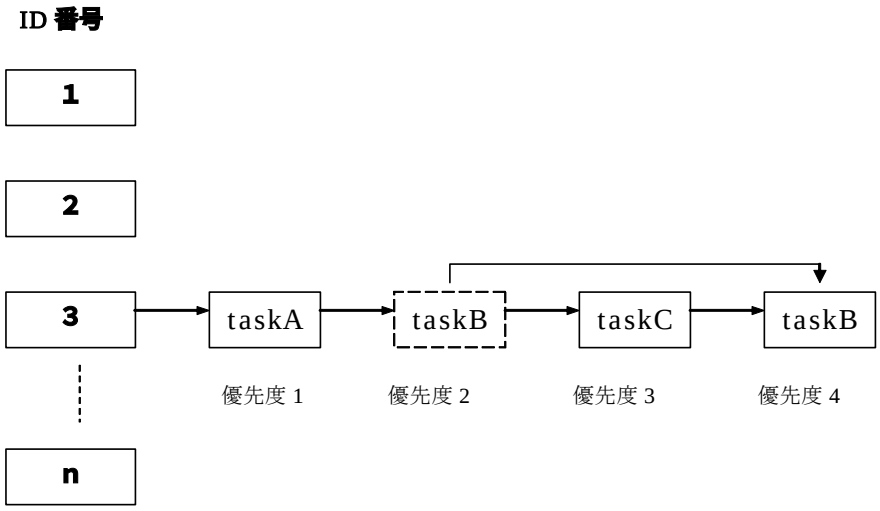
  タスクの優先度を変更するとそのタスクが実行可能状態もしくは実行状態であるときは、レディキューも更新されます。（図 3.21参照）

  また、対象タスクが TA_TPRI 属性を持つオブジェクトの待ち行列につながれている場合は、待ち行列も更新されます。(図 3.22参照)



タスク B の優先度を 3 から 1 に変更した場合

図 3.21 優先度の変更



タスク B の優先度を 2 から 4 に変更した場合

図 3.22 待ちキューのつなぎ換え

- タスクの優先度を取得する（get_pri, iget_pri）

  タスクの優先度を取得します。

- タスクの状態を参照する(簡易版)（ref_tst,iref_tst）

  対象タスクの状態を参照します。

- タスクの状態を参照する（ref_tsk,iref_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。

MR32R カーネルが提供するタスク付属同期サービスコールには次のものがあります。

- 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 サービス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 サービス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。

  slp_tsk、tslp_tsk サービス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[29]のタスクに対して wup_tsk、iwup_tsk サービス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 サービス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 3.23参照)

- 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 3.24参照）

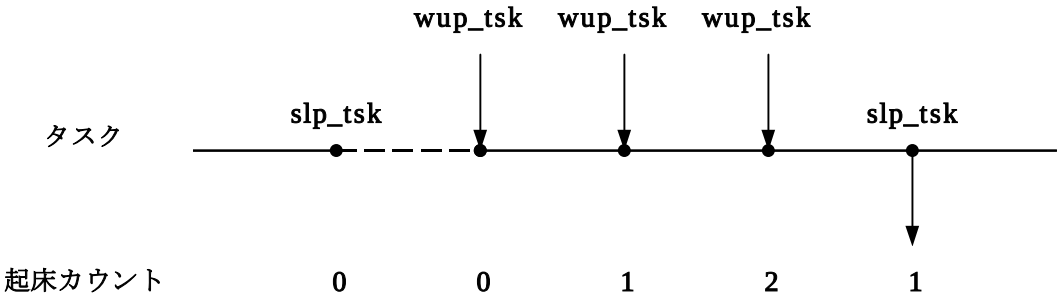


図 3.23 起床要求の蓄積

[29] 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、データ送信待ち状態、データ受信待ち状態、タイムアウト待ち状態、メッセージバッファ送信待ち、メッセージバッファ受信待ち、固定長メモリプール獲得待ち、可変長メモリプール獲得待ち、ランデヴ受付待ち、ランデヴ呼び出し待ち、ランデヴ終了待ち

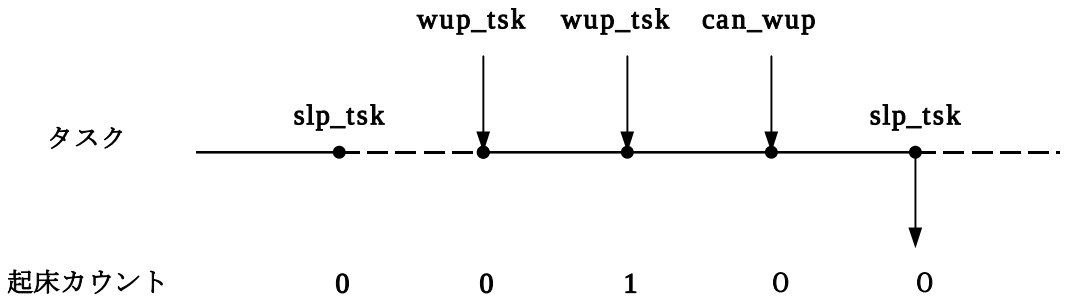


図 3.24 起床要求のキャンセル

- 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。ただし、強制待ち要求はネストされるため、強制待ち要求ネスト数が、２以上のタスクに対して、rsm_tsk(irsm_tsk)を発行しても、強制待ち要求ネスト数が１減じられるだけで強制待ち状態は変化しません。（図 3.25参照）

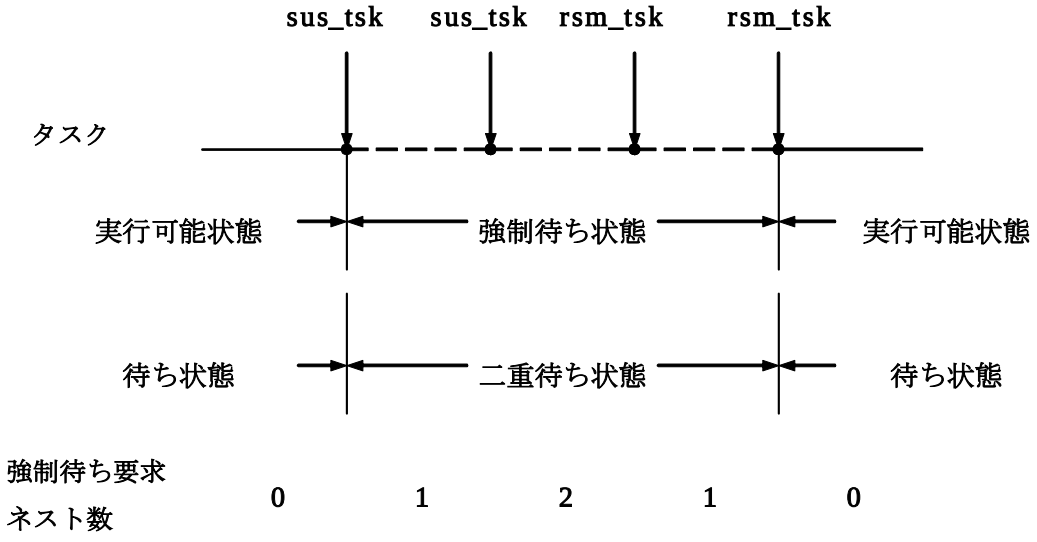


図 3.25 タスクの強制待ちと再開

- 強制待ち状態のタスクを強制再開する（frsm_tsk, ifrsm_tsk）

  タスクの強制待ち状態を解除し、タスクの実行を再開します。強制待ち要求ネスト数が、２以上のタスクに対して本サービスコールを発行した場合は、強制待ち要求のネスト数をすべてクリアし、タスクの実行を強制的に再開します。(図 3.26参照)

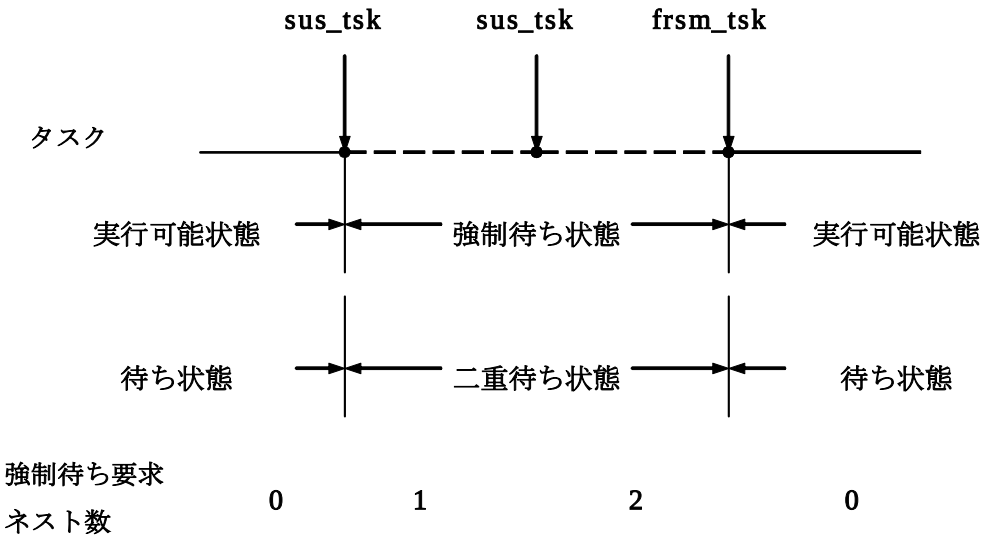


図 3.26 タスクの強制待ちと強制再開

- 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 サービスコールによる（+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
  - データ送信（+タイムアウト有)待ち状態
  - データ受信（+タイムアウト有)待ち状態
  - メッセージバッファ送信（+タイムアウト有)待ち状態
  - メッセージバッファ受信（+タイムアウト有)待ち状態
  - ランデヴ呼出（+タイムアウト有)待ち状態
  - ランデヴ受付（+タイムアウト有)待ち状態
  - ランデヴ終了（+タイムアウト有)待ち状態
  - 固定長、可変長メモリブロック（+タイムアウト有)獲得待ち状態

- 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 3.27に dly_tsk サービスコールにより 10msec 間タスクの実行を待たせる例を示します。タイムアウト値として指定する単位は、タイムティック数ではなく ms 単位で指定します。

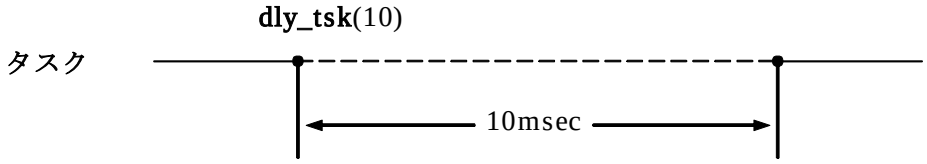


図 3.27 dly_tsk サービスコール

3.5.5 タスク例外機能

タスク例外機能とは、タスクに発生した例外事象の処理を、タスク本体の処理とは非同期に行うための機能で、一般的に「シグナル」と呼ばれている機能に類似しています。

タスクは、①タスク例外が禁止され、②保留例外要因はクリアされ、③タスク例外処理要求がない状態で起動します。タスク例外処理は、①タスク例外が許可されている(ena_tex)②保留例外要因がでない(ras_tex,iras_tex によりタスク例外処理要求が発生した)③タスクが実行可能状態である④非タスクコンテキストが実行されていないという４つの条件を同時に満たした場合に実行されます。

タスク例外処理ルーチンには、引数として保留例外要因と拡張情報が渡されます。

```
void texrtn(TEXPTN texptn, VP_INT exinf)
{
        タスク例外処理
}
```

MR32R でタスク例外を使用する場合、タスクのコンテキスト保存領域として余分に 76 バイト必要とします。

図 3.28 にタスク例外処理の動作例を示します。

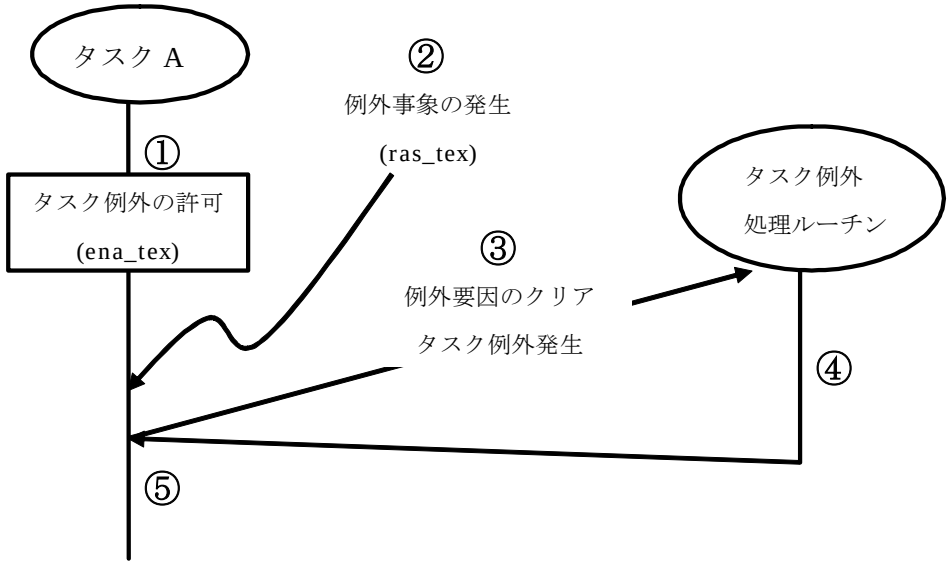


図 3.28 タスク例外処理の動作例

【図の解説】

① タスク A がタスク例外を許可します。

② タスク A 実行中に ras_tex サービスコールによってタスク A に例外事象が要求されます。

③ タスク A にタスク切り替えが発生した時、タスク A 本体の処理は実行されず、タスク A のタスク例外処理ルーチンが起動されます。このときタスク例外は禁止状態となり、保留例外要因もクリアされます。

④ タスク例外処理ルーチンを実行します。

⑤ タスク例外処理ルーチンからリターンすると、タスク A 本体処理の実行を再開します。

- タスク例外を定義する（def_tex）

  タスク例外を定義します。

- タスク例外の実行を処理要求する（ras_tex,iras_tex）

  タスク例外の実行を要求します。その際に、保留例外要因を指定することができ、これをタスク例外処理ルーチンの引数として渡します。

- タスク例外を許可する（ena_tex）

  タスク例外を許可します。

- タスク例外を禁止する（dis_tex）

  タスク例外を禁止します。

- タスク例外許可状態を調べる（sns_tex）

  タスク例外が許可されているかどうか調べます。

- タスク例外を参照する（ref_tex,iref_tex）

  タスク例外処理の状態、保留例外要因を参照します。

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 3.29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。

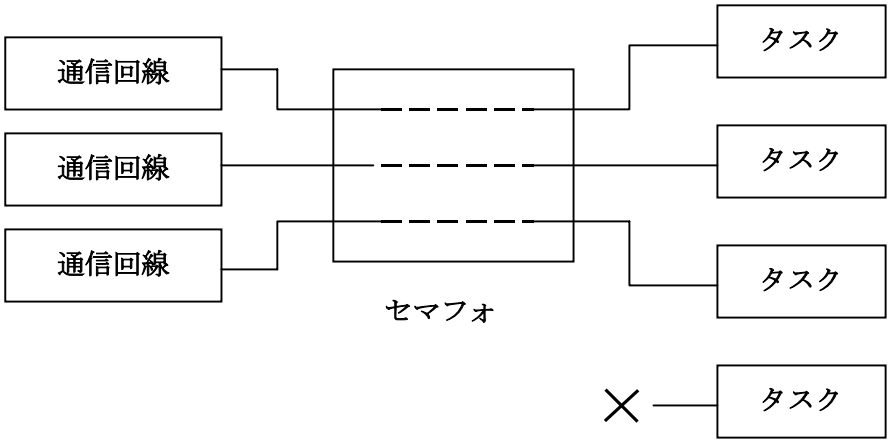


図 3.29 セマフォによる排他制御

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の獲得・解放をおこなうことによって資源の競合を防ぎます。（図 3.30参照）

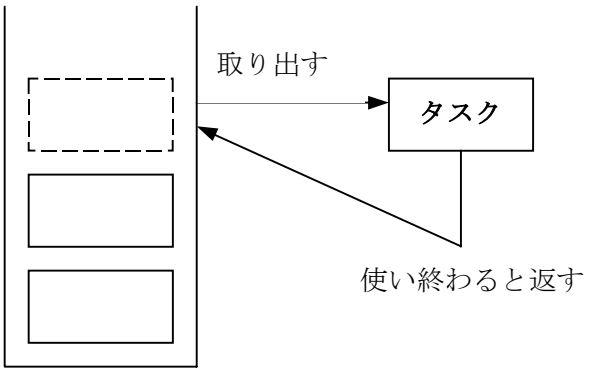


図 3.30 セマフォカウンタ

MR32R カーネルが提供するセマフォ同期のサービスコールには次のものがあります。

- セマフォを生成する（cre_sem）

  あるタスクから指定された ID のセマフォを生成します。

- セマフォを生成する（acre_sem）

  あるタスクからセマフォを生成します。ID 番号はカーネルが自動的に割り当てます。

- セマフォを削除する（del_sem）

  あるタスクから指定された ID のセマフォを削除します。

- 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（pol_sem,ipol_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,iref_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 サービスコールを用いたタスクの実行制御の例を図 3.31に示します。

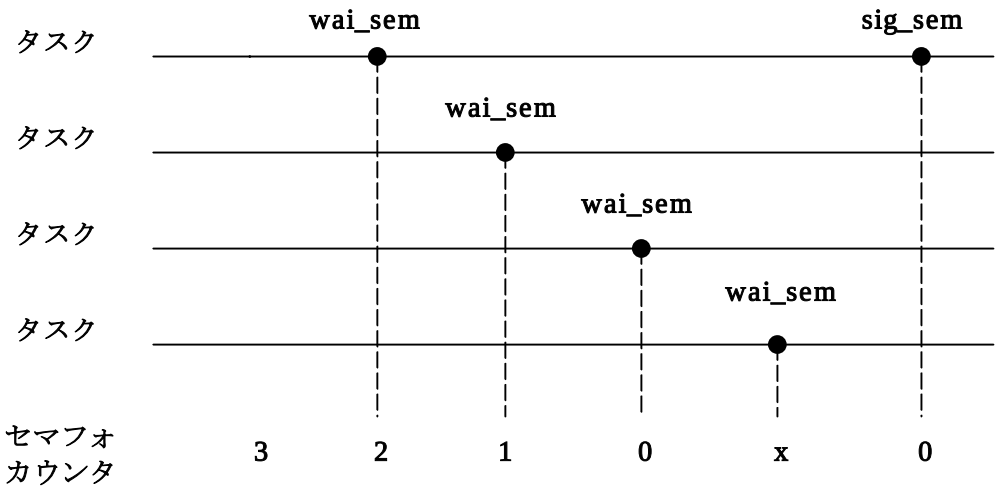


図 3.31 セマフォによるタスクの実行制御

3.5.7 同期・通信機能 (イベントフラグ)

イベントフラグは複数タスクの実行同期をとるための MR32R 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 32 ビットのイベントフラグ(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。

ひとつのイベントフラグの待ち行列に複数の待ちタスクの接続を許可するかどうかをイベントフラグ属性 TA_WSGL、TA_WMUL を指定することにより決定することができます。

また、イベントフラグ属性に TA_CLR を指定することにより、イベントフラグが待ち条件を満たした場合イベントフラグのビットパターンをすべてクリアすることができます。

MR32R カーネルが提供するイベントフラグのサービスコールには次のものがあります。

- イベントフラグを生成する (cre_flg)

  あるタスクから指定された ID のイベントフラグを生成します。

- イベントフラグを生成する (acre_flg)

  あるタスクからイベントフラグを生成します。ID 番号はカーネルが自動的に割り当てます。

- イベントフラグを削除する (del_flg)

  あるタスクから指定された ID のイベントフラグを削除します。

- 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。

- イベントフラグを得る (pol_flg,ipol_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。このサービス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,iref_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 3.32に wai_flg と set_flg サービス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 3.32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 サービスコールによって、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。

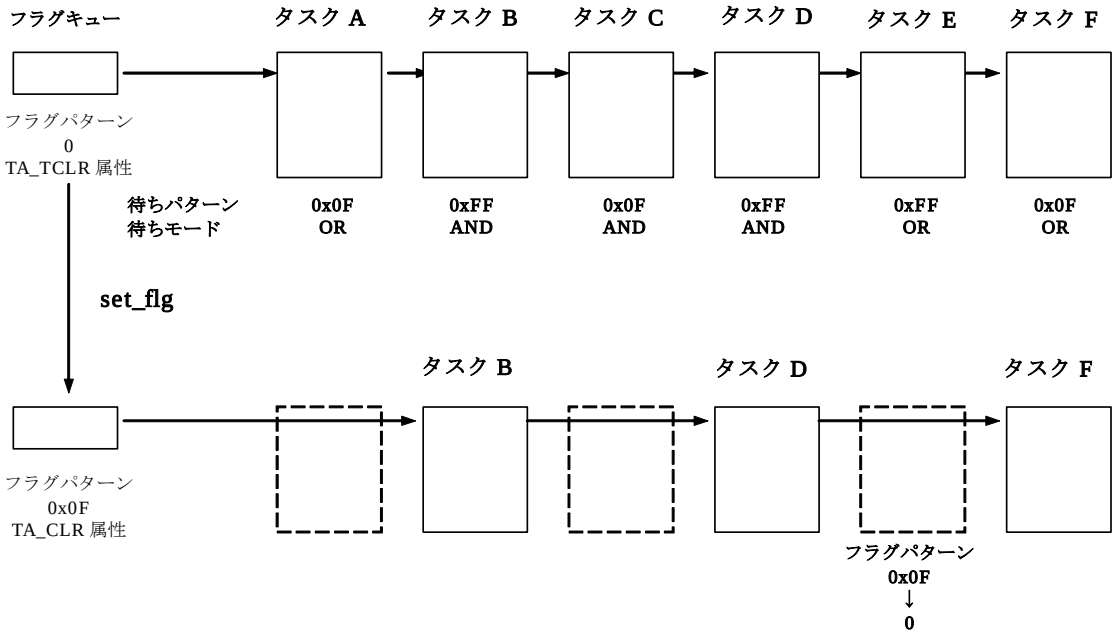


図 3.32 イベントフラグによるタスクの実行制御

3.5.8 同期・通信機能 (データキュー)

データキューとはタスク間で１ワードデータの通信をおこなう機構です。例えば、図 3.33においてタスクＡがデータをデータキューに送信しタスクＢがそのデータをデータキューから受信することができます。

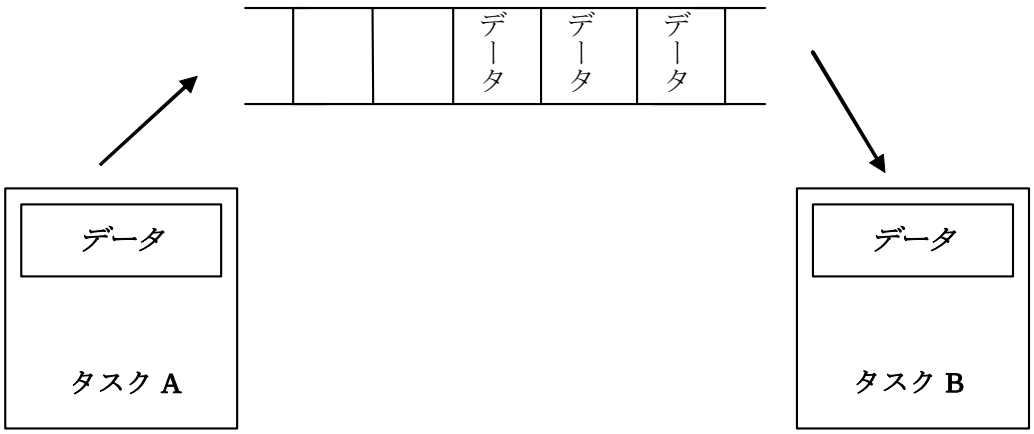


図 3.33 データキュー

このデータキューに送信できるデータ幅は 32 ビットのデータです[30]。

データキューにはデータを蓄積する機能があります。蓄積されたデータは FIFO[31]でデータが取り出されます。ただし、データキューに蓄積できるデータの数には制限があります。データキューがデータで一杯になった状態で、データを送信した場合は、サービスコール発行タスクはデータ送信待ち状態に移行します。

MR32R カーネルが提供するデータキューのサービスコールには次のものがあります。

- データキューを生成します（cre_dtq）

  あるタスクから指定された ID のデータキューを生成します。

- データキューを生成します（acre_dtq）

  あるタスクからデータキューを生成します。ID 番号はカーネルが自動的に割り当てます。

- データキューを削除します（del_dtq ）

  あるタスクから指定された ID のデータキューを削除します。

- データを送信します（snd_dtq, tsnd_dtq）

  データを送信します。すなわち、データをデータキューに送信します。送信データキューがデータで一杯の場合は、データ送信待ち状態に移行します。

[30] 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。
[31] ファーストインファーストアウト

- データを送信します（psnd_dtq, ipsnd_dtq）

  データを送信します。すなわち、データをデータキューに送信します。送信データキューがデータで一杯の場合は、データ送信待ち状態に移行せず、エラーコードを返します。

- データを受信します（rcv_dtq, trcv_dtq）

  データを受信します。すなわち、データをデータキューから取り出します。このときデータキューにデータがなければ、データが送信されるまで待ち状態になります。

- データを受信します（prcv_dtq,iprcv_dtq）

  データを受信します。rcv_dtq サービスコールと異なるのは、データキューにデータがない場合は待ち状態に移行せずにエラーコードを返すところです。

- データキューの状態を参照します（ref_dtq,iref_dtq）

  対象データキューにデータが入るのを待っているタスクの有無やデータキューに入っている先頭のデータを参照します。

3.5.9 同期・通信機能 (メールボックス)

メー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 3.34においてタスクＡがメッセージをメールボックスに投函しタスクＢがそのメッセージをメールボックスから取り出すことができます。メールボックスを用いた通信は、メッセージの先頭アドレスの受け渡しによって実現されているため、メッセージサイズに依存せずに高速に行われます。

カーネルは、メッセージキューをリンクリストで管理します。アプリケーション側でリンクリストに用いるためのヘッダ領域を用意しなければいけません。これをメッセージヘッダと呼びます。メッセージヘッダと実際にアプリケーションが使用するメッセージを格納する領域をメッセージパケットと呼びます。カーネルはメッセージヘッダの内容を書き換えて管理しています。アプリケーションからはメッセージヘッダを書き換えることはできません。メッセージキューの状態を図 3.35 に示します。 メッセージヘッダのデータ型は以下の通り定義しています。

T_MSG: メールボックスのメッセージヘッダ

T_MSG_PRI: メールボックスの優先度付きメッセージヘッダ

メッセージキューに入れることのできるメッセージのサイズは、アプリケーション側でヘッダ領域を確保するため、制限はありません。また、送信するためにタスクが待ち状態になることはありません。

メッセージに優先度を設定し、優先度の高いメッセージから受信することができます。さらに、メッセージ待ち状態のタスクがメッセージを受信する際、優先度の高いタスクからメッセージを受信することもできます。

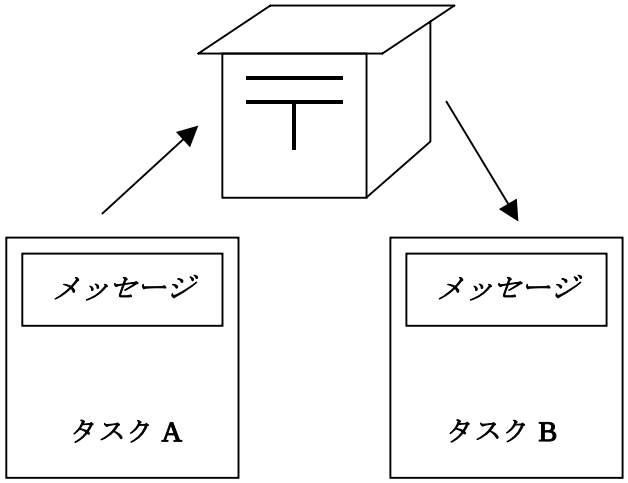


図 3.34 メールボックス

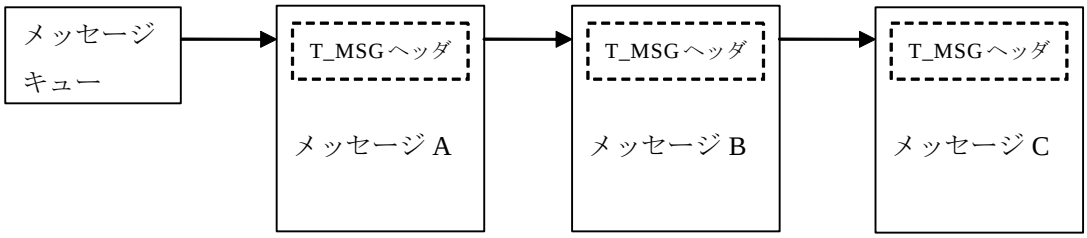


図 3.35 メッセージキュー

MR32R カーネルが提供するメールボックスのサービスコールには次のものがあります。

- メールボックスを生成します（cre_mbx）

  あるタスクから指定された ID のメールボックスを生成します。

- メールボックスを生成します（acre_mbx）

  あるタスクからメールボックスを生成します。ID 番号はカーネルが自動的に割り当てます。

- メールボックスを削除します（del_mbx）

  あるタスクから指定された ID のメールボックスを削除します。

- メッセージを送信します（snd_mbx, isnd_mbx）

  メッセージを送信します。すなわち、メッセージをメールボックスに投函します。

- メッセージを受信します（rcv_mbx, trcv_mbx）

  メッセージを受信します。すなわち、メッセージをメールボックスから取り出します。このときメッセージがメー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_mbx,iprcv_mbx）

  メッセージを受信します。rcv_msg サービスコールと異なるのは、メー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。

- メールボックスの状態を参照します（ref_mbx,iref_mbx）

  対象メー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ー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。

3.5.10 拡張同期・通信機能 (メッセージバッファ)

メッセージバッファは、メールボックスの機能と同様にメッセージを渡すことにより同期と通信を同時に行う機能です。メールボックス機能と異なる点は、メッセージ内容そのもののコピーが、送信相手のタスクに送信されます。

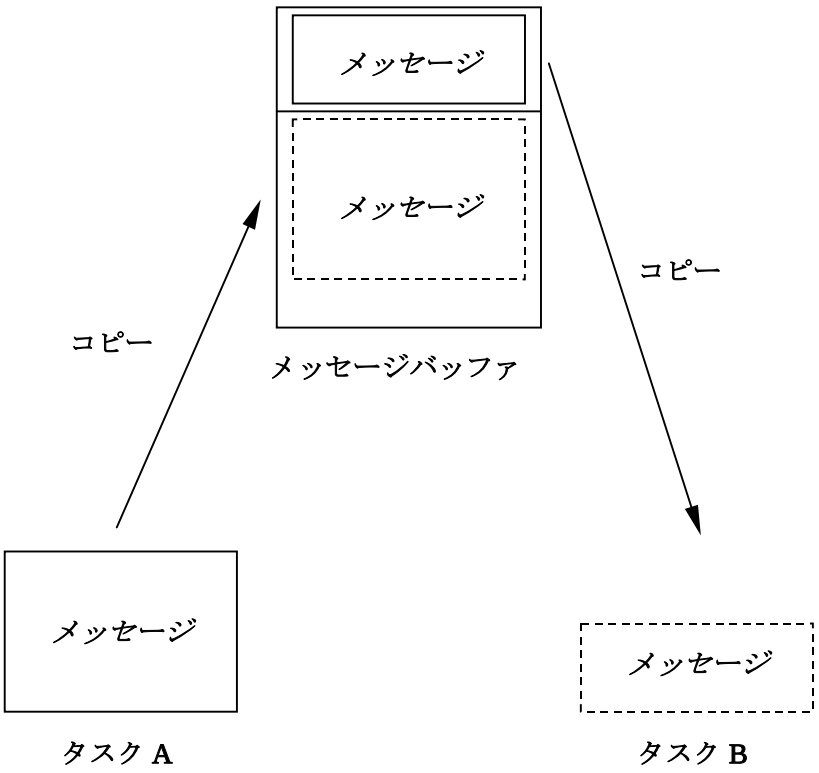


図 3.36 メッセージバッファ

図 3.36が示すようにメッセージバッファには、メールボックスと同様にメッセージを蓄積します。蓄積されたメッセージは、FIFO 順でメッセージが取り出されます。

メッセージバッファ機能を使用し、メッセージを送信する場合、MR32R はバッファにまず、4 バイトのメッセージのサイズ情報を書き込んだ後、メッセージを書き出すようになっています。直前に送信されたメッセージの終端が 4 バイト境界にない場合は、次の 4 バイト境界よりメッセージのサイズ情報を書き出します。

そのため、ひとつのメッセージを送信すると、「アライメント調整分+サイズ情報(4 バイト)+メッセージサイズ」分だけバッファを使用します。

なお、受信したメッセージにメッセージのサイズ情報は含まれません。

［例］

メッセージ終端が4バイト境界にないメッセージAを送信後､12バイトのメッセージBを送信した場合、次の 4 バイト境界にサイズ情報が書き込まれた後、メッセージ B が書き込まれます。

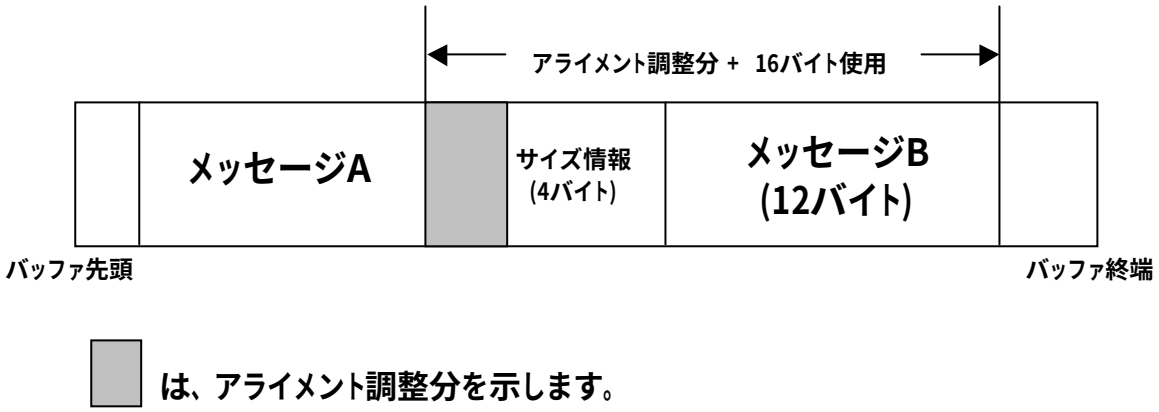


図 3.37 メッセージの送信例

MR32R カーネルが提供するメッセージバッファのサービスコールには、次のものがあります。

- メッセージバッファを生成する（cre_mbf）

  あるタスクから指定された ID のメッセージバッファを生成します。

- メッセージバッファを生成する（acre_mbf）

  あるタスクからメッセージバッファを生成します｡ID 番号はカーネルが自動的に割り当てます。

- メッセージバッファを削除する（del_mbf）

  あるタスクから指定された ID のメッセージバッファを削除します。

- メッセージを送信する（snd_mbf、tsnd_mbf）

  メッセージバッファにメッセージを送信します。すなわち、メッセージをメッセージバッファにコピーします｡メッセージバッファでは､転送するメッセージのサイズは一定ではありません｡メッセージバッファの空きサイズによっては、メッセージを転送できたり、できなくなる場合が出てきます。メッセージバッファの空きサイズが足りない場合は、送信待ち状態になります。（図 3.38参照）

  既にメッセージ送信待ち状態のタスクが存在する場合は､メッセージバッファに空き領域があったとしても送信待ち状態に移行します。

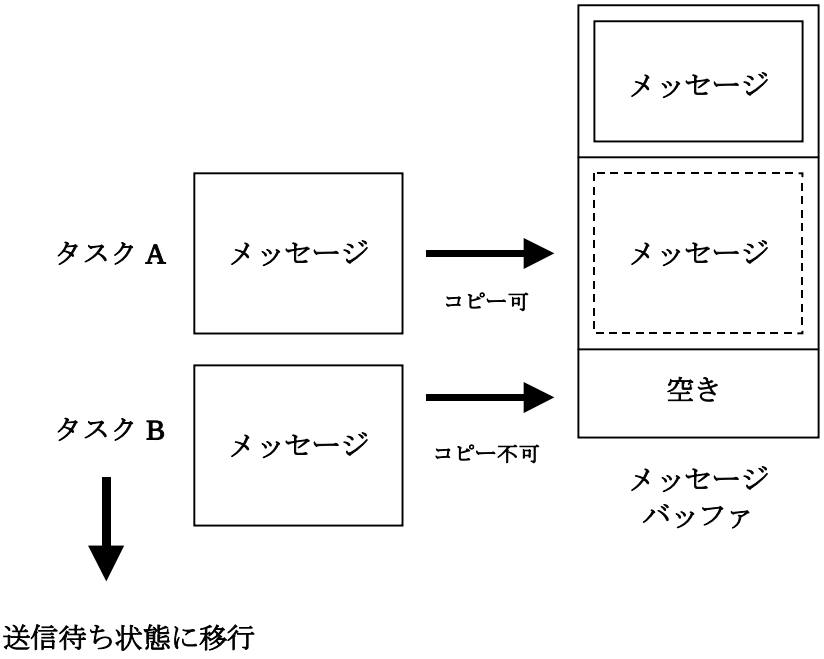


図 3.38 メッセージの送信

- メッセージを送信する（psnd_mbf）

  メッセージバッファにメッセージを送信します。snd_mbf、tsnd_mbf と異なるのは、メッセージバッファに空きサイズが足りない場合、送信待ち状態にはならず、エラーコードを返すところです[32]。

[32] エラーコードは、E_TMOUT を返します。

- メッセージを受信する（rcv_mbf、trcv_mbf）

  メッセージを受信します。メッセージをメッセージバッファから取り出します。メッセージバッファにメッセージがない場合は受信待ち状態に移行します。

  メッセージをメッセージバッファから受信すると、メッセージバッファの空きサイズが大きくなります。送信待ちタスクがある場合、その待ちタスクが送信するメッセージのサイズよりも空きサイズの方が大きい場合は、メッセージをメッセージバッファに送信し、送信待ち状態から実行(RUN)状態あるいは実行可能(READY)状態に移行します。

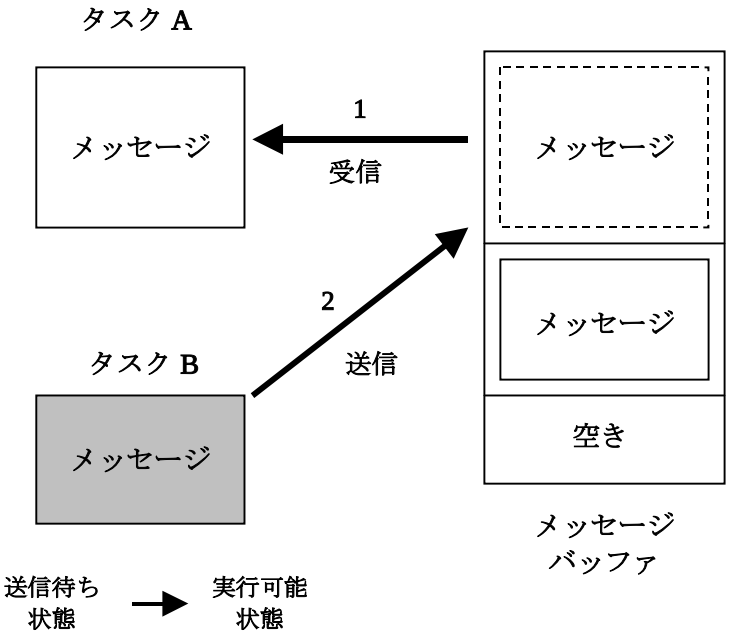


図 3.39 メッセージの受信

状態は、図 3.39では、タスク A が実行(RUN)状態、タスク B が送信待ち状態中で、

1. **タスク A がメッセージバッファからメッセージを受信します。**

   メッセージバッファからメッセージを受信することで、メッセージバッファの空き領域が大きくなります。

2. **タスク B は、メッセージバッファにメッセージを送信します。**

   メッセージバッファの空き領域が足りないため、送信待ち状態中であったタスク B は、タスク A のメッセージ受信により、メッセージバッファにメッセージを送信します。タスク B の状態は、送信待ち状態から実行可能状態に移行します。

- メッセージを受信する（prcv_mbf）

  メッセージを受信します。rcv_mbf、trcv_mbf と異なるところは、メッセージバッファにメッセージがない場合は、受信待ち状態に移行せずにエラーコードを返すところです[33]。

- メッセージバッファの状態を参照する（ref_mbf,iref_mbf）

  対象メッセージバッファに対する送信待ちタスクの有無、受信待ちタスクの有無、次に受信する

[33] エラーコード E_TMOUT を返します。

メッセージのサイズなどを参照します。

3.5.11 拡張同期・通信機能 (ランデヴ)

ランデヴ機能は、ポートと呼ばれる窓口を介してタスク(呼出)とタスク(受付)が待ち合わせを行い、待ち合わせ(ランデヴ)が成立すると、互いのメッセージの交換を行う機能です。(待ち合わせ(ランデヴ)の成立とは、呼出側のビットパターンと受付側のビットパターンとの論理積の結果が 0 以外であれば、ランデヴ成立となり、待ち合わせが成立したことになります。)

本機能は、リアルタイム OS 上でのサーバ・クライアント形式の実現に有用です。

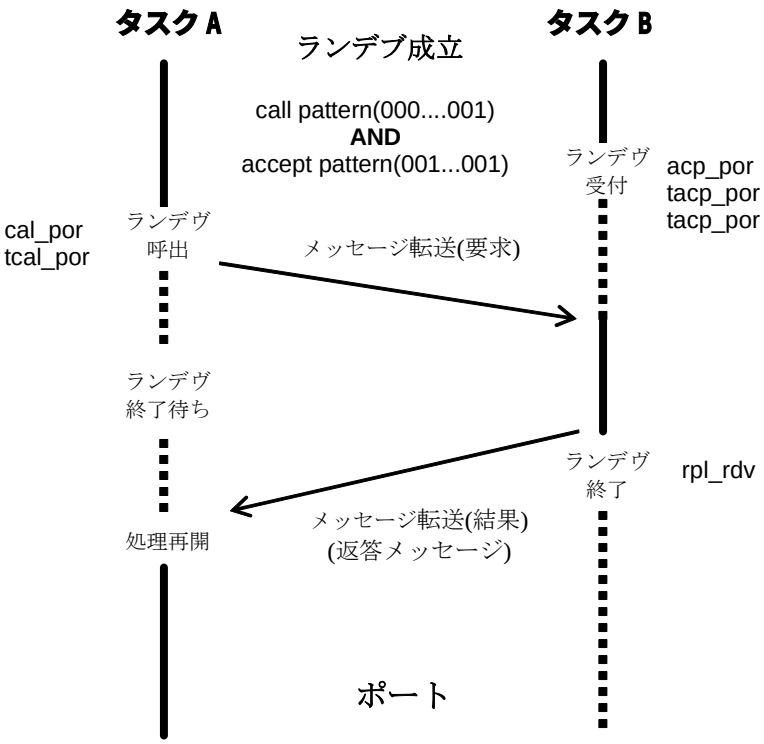


図 3.40 ランデヴ

MR32R カーネルが提供するランデヴのサービスコールには次のものがあります。

- ランデヴ用ポートを生成します（cre_por）

  あるタスクから指定された ID のランデヴ用ポートを生成します。

- ランデヴ用ポートを生成します（acre_por）

  あるタスクからランデヴ用ポートを生成します。ID 番号はカーネルが自動的に割り当てます。

- ランデヴ用ポートを削除します（del_por）

  あるタスクから指定された ID のランデヴ用ポートを削除します。

- ポートに対するランデヴの呼出（cal_por、tcal_por）

  cal_por、tcal_por サービスコールを発行し、指定したポートの受付待ち状態のタスクとランデヴが成立するかどうかをチェックします。ランデヴが成立すれば、受付待ち状態のタスクに対して、メッセージを転送し、受付待ち状態から実行可能状態に移行します。

  指定したポートに受付待ちタスクがない場合や受付待ちタスクがあってもランデヴ成立条件が

満たされない場合には、cal_por、tcal_por を発行したタスクは、呼出待ち状態となり、呼出待ち行列につながれます。

tcal_por が呼び出されてから、tmout が経過してもランデヴが終了しない場合[34]は処理をキャンセルしてタイムアウトします。

- ポートに対するランデヴの受付（acp_por、tacp_por）

acp_por、tacp_por サービスコールを発行し、指定したポートの呼出待ち状態のタスクとランデヴが成立するかどうかをチェックします。ランデヴが成立すれば、呼出待ち状態であったタスクは、呼出待ち行列からはずれ、ランデヴ終了待ち状態へ移行します。なお、ランデヴ受付側タスク(acp_por を発行し他タスク)が、同時に複数のランデヴを行うこともできます。図 3.41は、異なるポートでの複数のランデヴを表わしています。また、同一ポートに対しても複数のランデヴは行うことはできます。

指定したポートに呼出待ちタスクがない場合や呼出待ちタスクがあってもランデヴ成立条件が満たされない場合には、acp_por、tacp_por を発行したタスクは、受付待ち状態となり、受付待ち行列につながれます。

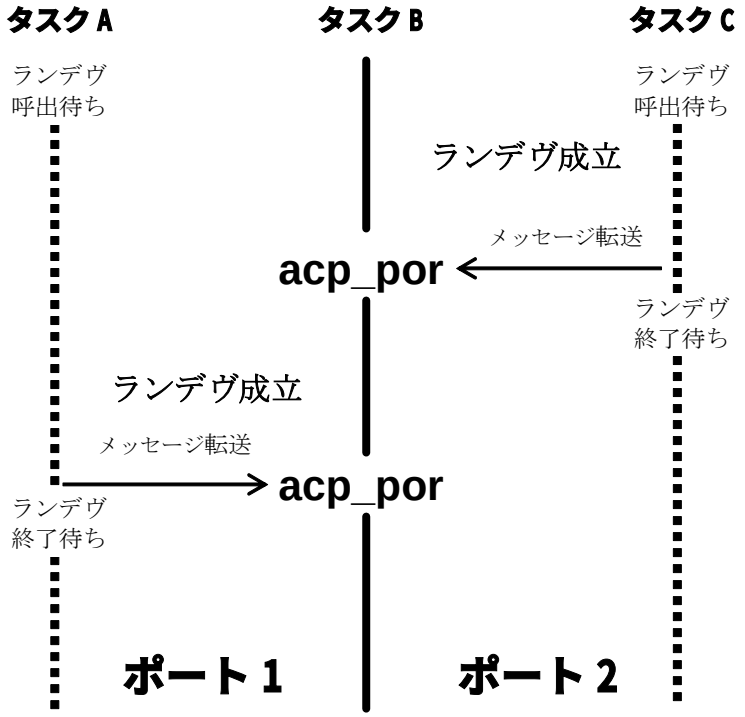


図 3.41 複数のランデヴ

- ポートに対するランデヴの受付（pacp_por）

acp_por、tacp_por サービスコールと同様の処理を行いますが、指定したポートに呼出待ちタスクがない場合や呼出待ちタスクがあってもランデヴ成立条件が満たされない場合は、受付待ち状態には移行せず、エラーコードを返します。

[34] タイムアウトの対象は、ランデヴ成立までの時間ではなくランデヴ終了までの時間となる点に注意してください。

- ポートに対するランデヴの回送（fwd_por）

  本サービスコール(fwd_por)を発行すると、現在、ランデヴ中の呼出側タスクとのランデヴ状態を解除し、指定した別のポートに対してランデヴ呼出を行います。(本サービスコールは、acp_por、tacp_por サービスコールを実行した後の状態で発行しなければなりません。)この後の処理は、cal_por、tcal_por サービスコールの処理と同様の処理を行います。

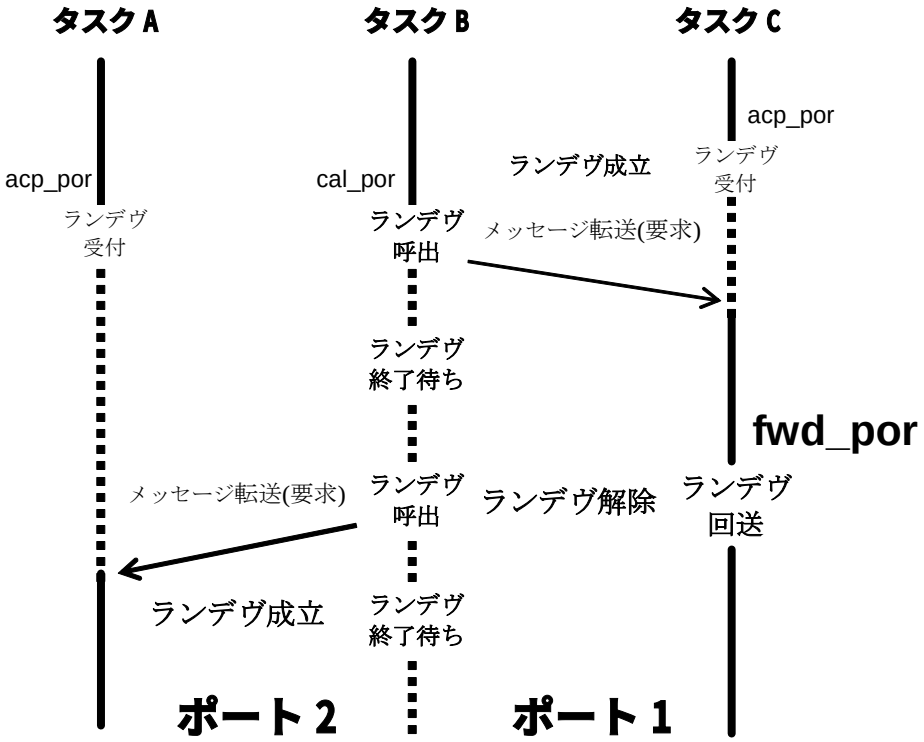


図 3.42 ランデヴの回送

- ランデヴの返答（rpl_rdv）

  本サービスコール(rpl_rdv)を発行すると呼出側タスクに対して返答メッセージを返し、ランデヴを終了します。本サービスコールは、acp_por、tacp_por サービスコール発行後の状態でなければなりません。返答メッセージを受け取った呼出側タスクは、ランデヴ終了待ち状態から実行可能状態に移行します。

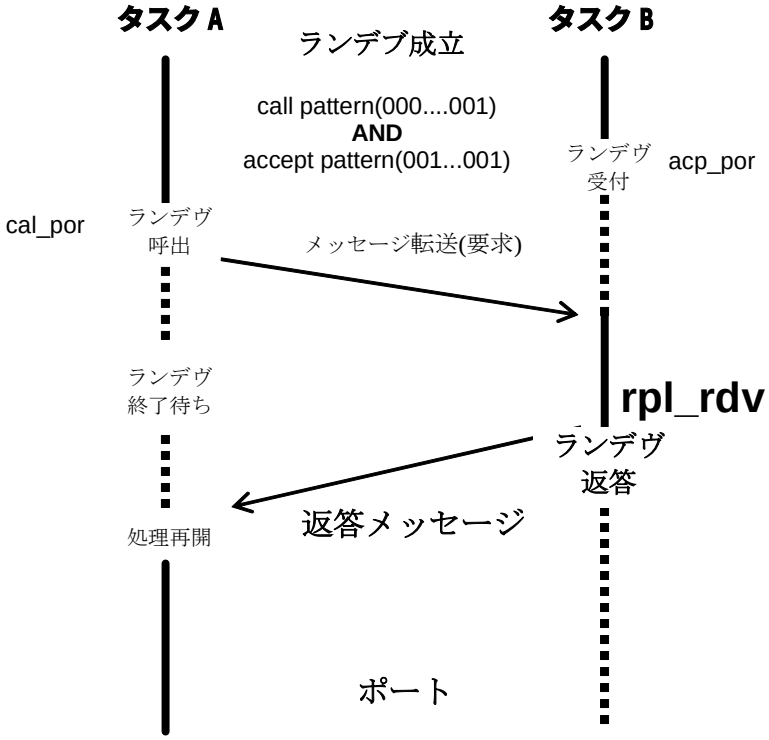


図 3.43 ランデヴの返答

- ランデヴ用ポートの参照（ref_por,iref_por）

  指定した ID のランデヴ用ポートに対する呼び出し待ちタスクの有無や受付待ちタスクの有無を参照します。

- ランデヴ状態の参照（ref_rdv,iref_rdv）

  指定したランデヴを参照します。具体的には、ランデヴを呼び出したタスク ID を調べます。

3.5.12 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。

MR32R 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、cre_mpf サービスコールあるいはコンフィギュレーションファイルで指定します。

MR32R カーネルが提供する固定長メモリプール管理サービスコールには次のものがあります。

- 固定長メモリプールを生成する（cre_mpf）

  あるタスクから指定された ID の固定長メモリプールを生成します。

- 固定長メモリプールを生成する（acre_mpf）

  あるタスクから固定長メモリプールを生成します。ID 番号はカーネルが自動的に割り当てます。

- 固定長メモリプールを削除する（del_mpf）

  あるタスクから指定された ID の固定長メモリプールを生成します。

- メモリブロックを獲得する（get_mpf、tget_mpf）

  指定された ID の固定長メモリプールからメモリブロックを獲得します。空きメモリブロックが、指定された固定長メモリプールにない場合、このサービスコールを発行したタスクは待ち状態に移行し、待ち行列につながれます。

- メモリブロックを獲得する（pget_mpf,ipget_mpf）

  指定された ID の固定長メモリプールからメモリブロックを獲得します。get_mpf、tget_mpf と異なるのは、空きメモリブロックがメモリプールにない場合は、待ち状態に移行せず、エラーコードを返します。

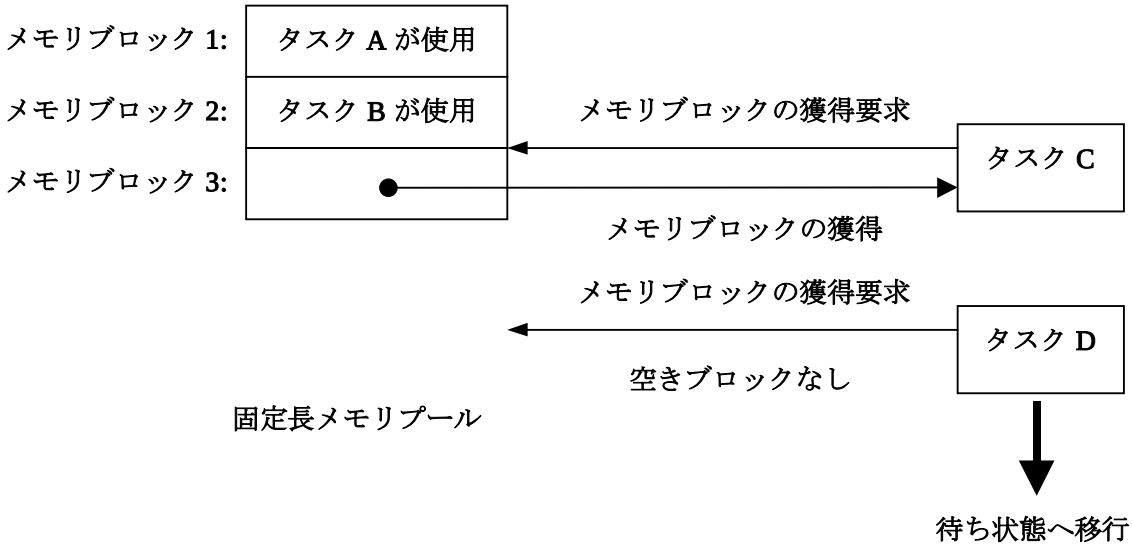


図 3.44 固定長メモリプールの獲得

- メモリブロックを解放する（rel_mpf,irel_mpf）

  獲得しているメモリブロックを解放します。指定された固定長メモリプールに対する待ち状態のタスクがある場合には、待ち行列の先頭につながれたタスクに解放したメモリブロックを割り当てます。この時のタスクの状態は、待ち状態から実行可能(READY)状態に移行します。また、待ち状態のタスクがない場合は、メモリプールにメモリブロックを返却します。

- メモリプールの状態を参照する（ref_mpf,iref_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。

MR32R 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、ユーザが獲得するメモリブロックの最大サイズから MR32R が計算します。メモリブロックの最大サイズは、cre_mpl サービスコールあるいはコンフィギュレーションファイルで指定します。

例

```
variable_memorypool[]{

        name            = mpl1;
        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 60,120,240,480 となります。

ユーザが要求したメモリは、指定サイズをもとに MR32R 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。

4 種類のブロックサイズは(下記 abcd)は、下記の計算式から算出されます。

```
a = (((max_memsize + (12-1))/96)+1)*12
b = a*2
c = a*4
d = a*8
```

MR32R カーネルが提供する可変長メモリプール管理サービスコールには次のものがあります。

- 可変長メモリプールを生成する（cre_mpl）

  あるタスクから指定された ID の可変長メモリプールを生成します。

- 可変長メモリプールを生成する（acre_mpl）

  あるタスクから可変長メモリプールを生成します。ID 番号はカーネルが自動的に割り当てます。

- 可変長メモリプールを生成する（del_mpl）

  あるタスクから指定された ID の可変長メモリプールを削除します。

- メモリブロックを獲得する（get_mpl、tget_mpl）

  メモリブロックを可変長メモリプールから獲得します。ユーザ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。指定されたサイズのメモリブロックがメモリプールにない場合、このサービスコールを発行したタスクは、待ち状態に移行し、待ち行列につながれます。

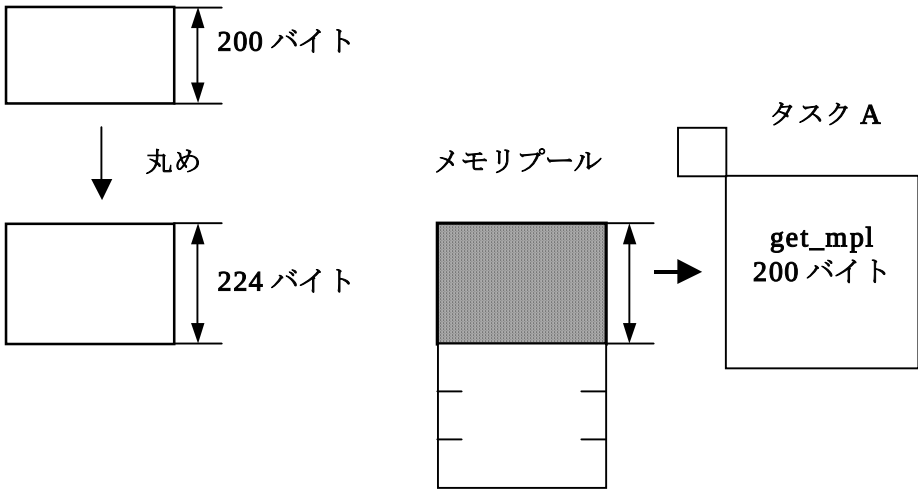


図 3.45 get_mpl 処理

図 3.45では、タスク A が 200 バイトのメモリブロックを要求している例です。MR32R では、200 バイトのサイズを 224 バイトに丸め、224 バイトのメモリブロックをメモリプールから切り出し、タスク A に割り当てている様子を示しています。

既にメモリプール獲得待ちタスクが存在する場合は、空きメモリが十分存在しても可変長メモリ獲得待ち状態に移行します。

- メモリブロックを獲得する（pget_mpl）

  メモリブロックを可変長メモリプールから獲得します。get_mpl、tget_mpl サービスコールと異なるところは、指定されたサイズのメモリブロックがメモリプールにない場合は、待ち状態に移行せず、エラーコードを返します。

- メモリブロックを解放する（rel_mpl）

  get_mpl、tget_mpl あるいは tget_mpl で獲得したメモリブロックを解放します。指定されたメモリプールのメモリブロックを待っているタスクがない場合は、メモリプールにメモリブロックを返却します。

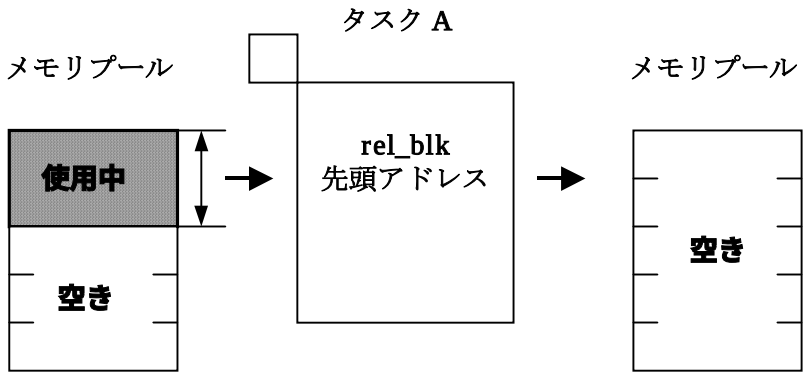


図 3.46 rel_mpl 処理

次に、メモリブロック解放時に対象メモリプールのメモリブロックを待っているタスクがある場合は、待ち行列の先頭タスクから要求サイズを調べ、条件を満足[35]すれば要求サイズ分のメモリを切り出し、タスクに割り当てて、待ち状態から実行可能(READY)状態へ移行します。さらに、次に接続されているタスクの要求サイズと解放したメモリの残りのサイズとを比較していきます。もし、要求を満足しないタスクがあれば、その時点で、メモリブロック割り当てを終了します。

[35]ここでの条件の満足とは、メモリの要求サイズが解放するよりも小さい場合をいいます。

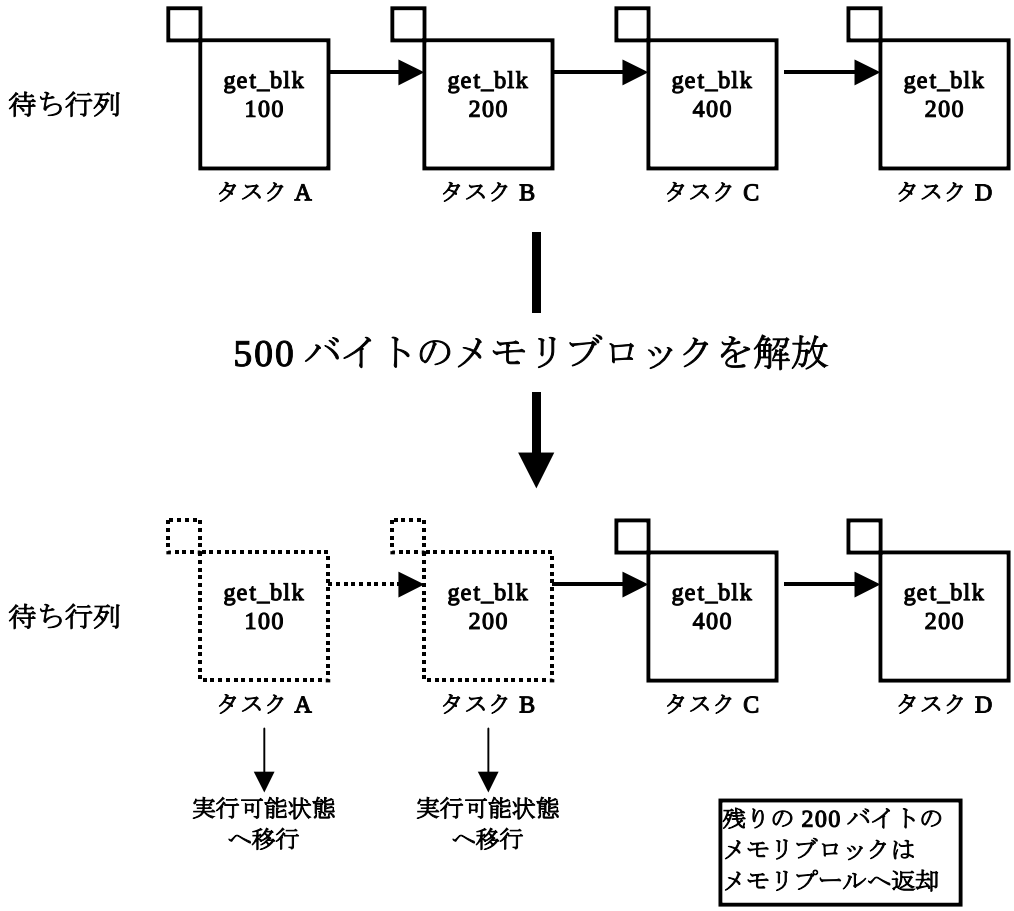


図 3.47 メモリブロックの解放

- メモリプールの状態を参照する（ref_mpl,iref_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。

3.5.13 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[36]、時刻の設定[37]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。

MR32R カーネルはシステムクロックとしてタイマを一つ必要とします。MR32R カーネルが提供する時間管理サービスコールには次のものがあります。なお、システムクロックは必須機能ではありません。したがって下記のサービスコールおよび時間管理機能を使用しなければ、タイマを MR32R 用に占有する必要がありません。

- 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サービスコール[38]にタイムアウトを指定することができます。サービスコール名は、tslp_tsk、twai_flg、twai_sem、tsnd_dtq、trcv_dtq、trcv_mbx、tsnd_mbf、trcv_mbf、tcal_por、tacp_por、tget_mpf、tget_mpl 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。

  タイムアウトの時間単位は、ms です。

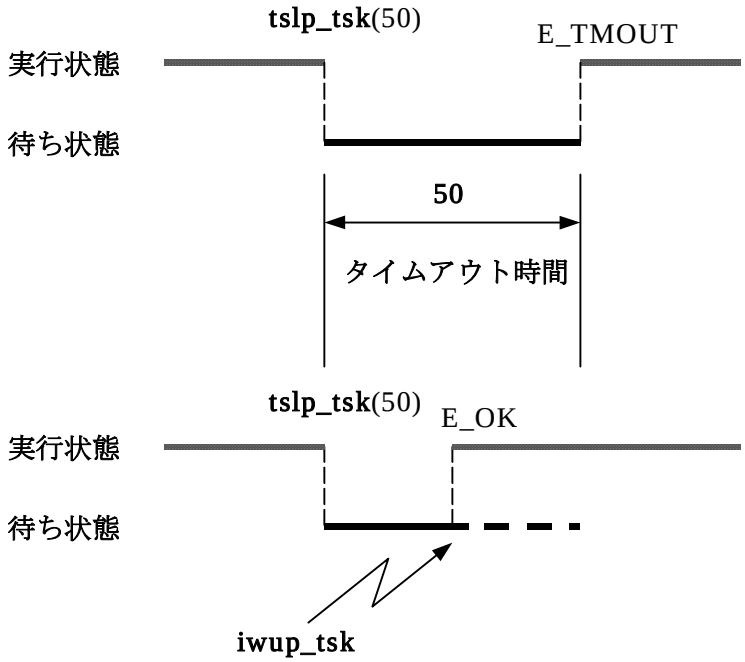


図 3.48 タイムアウト処理

MR32R では、μITRON 仕様の規定通り、指定されたタイムアウト値分以上の時間が経過してからタイムアウト処理を行うことを保証します。具体的には以下のタイミングでタイムアウト処理を行います。

1. タイムアウト値が 0 の場合(dly_tsk の場合のみ)

[36] get_tim システムコール
[37] set_tim システムコール
[38] 強制待ち状態を除きます。

サービスコール発行後の最初のタイムティックでタイムアウトします。

2. タイムアウト値が、タイムティック間隔の倍数である場合
(タイムアウト値/タイムティック間隔)+1 回目のタイムティックでタイムアウトします。例えば、タイムティック間隔が 10ms でタイムアウト値に 40ms を指定した場合、5 回目のタイムティックでタイムアウトします。また、タイムティック間隔が 0.5ms でタイムアウト値に 10ms を指定した場合、21 回目のタイムティックでタイムアウトします。
3. タイムアウト値が、タイムティック間隔の倍数でない場合
(タイムアウト値/タイムティック間隔)+2 回目のタイムティックでタイムアウトします。例えば、タイムティック間隔が 10ms でタイムアウト値に 35ms を指定した場合、5 回目のタイムティックでタイムアウトします。

- 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時からの経過時間を 48 ビットのデータで表します。時間の単位は ms です。

3.5.14 周期ハンドラ機能

周期ハンドラは、指定した起動位相経過後、起動周期ごとに起動されるタイムイベントハンドラです。

周期ハンドラの起動には、起動位相を保存する方法と起動位相を保存しない方法があります。起動位相を保存する場合は、周期ハンドラの生成時点を基準に周期ハンドラを起動します。起動位相を保存しない場合は、周期ハンドラの動作開始時点を基準に周期ハンドラを起動します。図 3.49、図 3.50に周期ハンドラの動作例を示します。

タイムティック間隔より、起動周期が短い場合、タイムティック供給(isig_tim 相当の処理)毎に、前回のタイムティック供給時より、本来起動されるべき時刻が経過した回数分だけ周期ハンドラを起動します。例えば、タイムティック間隔が 10ms、起動周期が 3ms で、タイムティック供給時に周期ハンドラ動作が開始された場合、最初のタイムティックで 3 回周期ハンドラが起動されることになります。

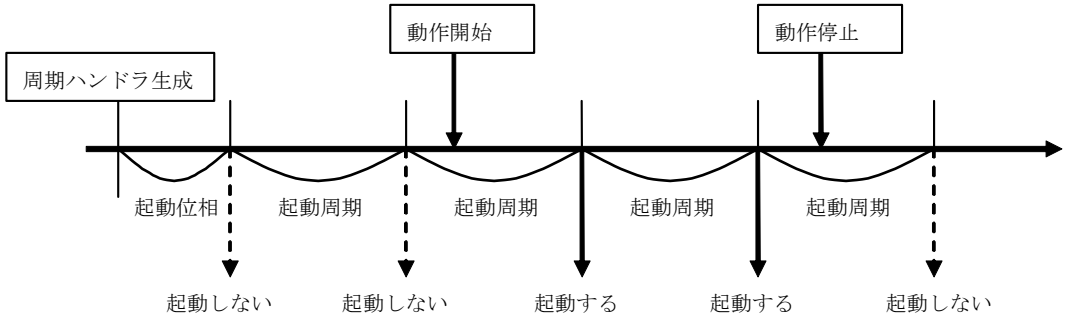


図 3.49 起動位相を保存する場合の動作

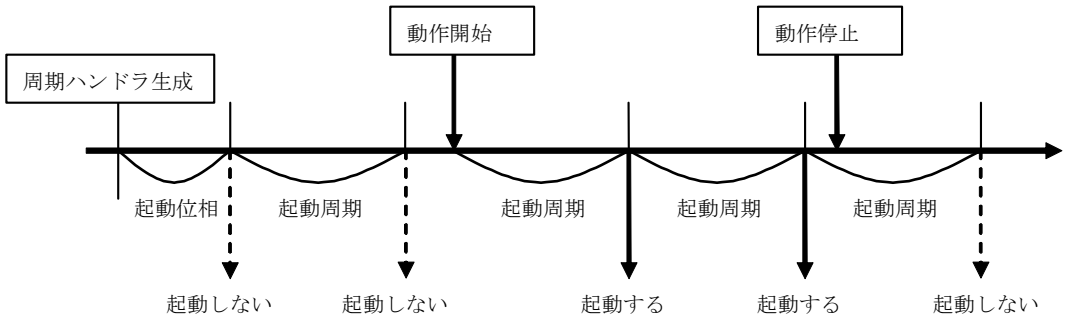


図 3.50 起動位相を保存しない場合の動作

- 周期ハンドラを生成する（cre_cyc）

  あるタスクから指定された ID の周期ハンドラを生成します。

- 周期ハンドラを生成する（acre_cyc）

  あるタスクから周期ハンドラを生成します。ID 番号はカーネルが自動的に割り当てます。

- 周期ハンドラを削除する (del_cyc)

  指定された ID の周期ハンドラを削除します。。

- 周期ハンドラの動作を開始する(sta_cyc,ista_cyc)

  指定された ID の周期ハンドラの動作を開始します。

- 周期ハンドラの動作を停止する(stp_cyc,istp_cyc)

  指定された ID の周期ハンドラの動作を停止します。

- 周期ハンドラの状態を参照する (ref_cyc,iref_cyc)

  周期ハンドラの状態を参照します。対象周期ハンドラの動作状態と次の起動までの残り時間を調べます。

3.5.15 アラームハンドラ機能

アラームハンドラは、指定した時刻になると 1 度だけ起動されるタイムイベントハンドラです。

アラームハンドラを用いることにより、時刻に依存した処理を行うことができます。時刻の指定は、相対時間です。図 3.51 動作例をに示します。

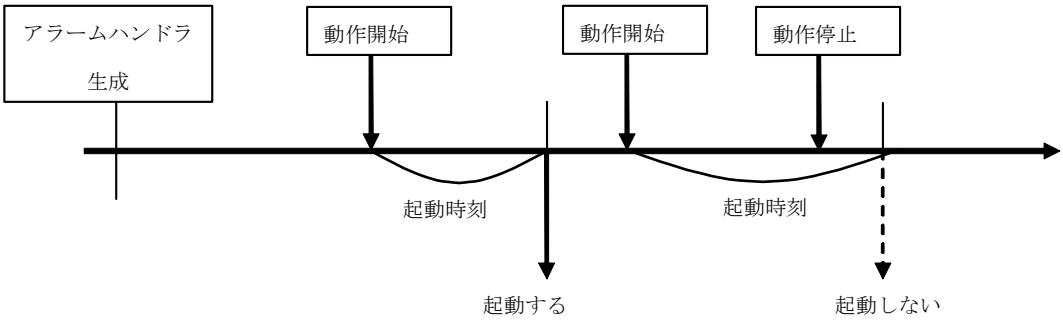


図 3.51 アラームハンドラの動作

- アラームハンドラを生成する（cre_cyc）

  あるタスクから指定された ID のアラームハンドラを生成します。

- アラームハンドラを生成する（acre_cyc）

  あるタスクからアラームハンドラを生成します。ID 番号はカーネルが自動的に割り当てます。

- アラームハンドラを削除する（del_cyc）

  指定された ID のアラームハンドラを削除します。。

- アラームハンドラの動作を開始する(sta_cyc,ista_cyc)

  指定された ID のアラームハンドラの動作を開始します。

- アラームハンドラの動作を停止する(stp_cyc,istp_cyc)

  指定された ID のアラームハンドラの動作を停止します。

- アラームハンドラの状態を参照する（ref_cyc,iref_cyc）

  アラームハンドラの状態を参照します。対象アラームハンドラの動作状態と起動までの残り時間を調べます。

3.5.16 システム状態管理機能

- タスクの実行待ち行列を回転する（rot_rdq , irot_rdq）

  本サービスコールにより 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 3.52参照）

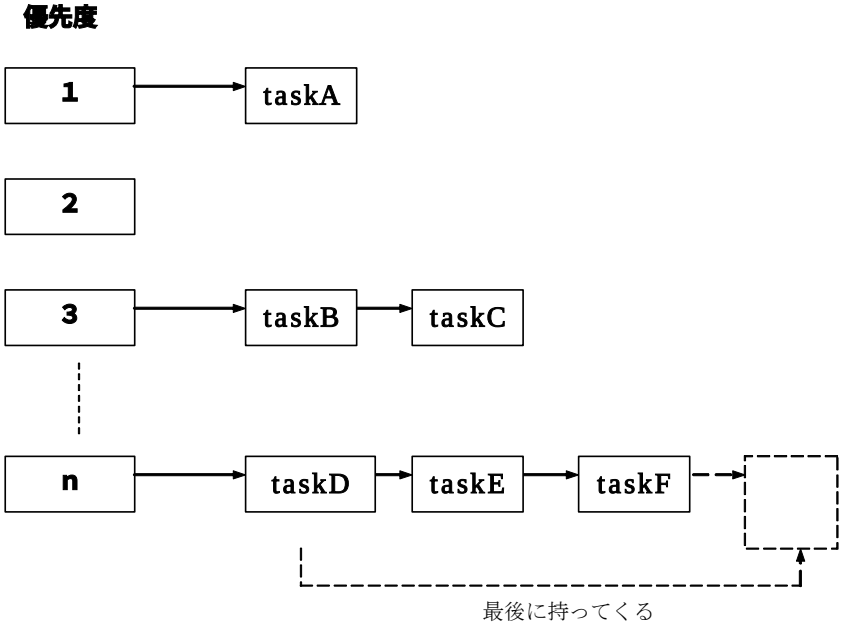


図 3.52 rot_rdq サービスコールによるレディキューの操作

- 自タスクの ID を得る（get_tid,iget_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 TSK_NONE(=0=)が得られます。

- CPU ロック状態に移行する（loc_cpu,iloc_cpu）

  CPU ロック状態に移行します。

- CPU ロック状態を解除する（unl_cpu,iunl_cpu）

  CPU ロック状態を解除します。

- ディスパッチ禁止状態に移行する（dis_dsp）

  ディスパッチ禁止状態に移行します。

- ディスパッチ禁止状態を解除する（ena_dsp）

  ディスパッチ禁止状態を解除します。

- コンテキスト状態を得ます（sns_ctx）

  コンテキスト状態を得ます。

- CPU ロック状態を得ます（sns_loc）

  CPU ロック状態を得ます。

- ディスパッチ禁止状態を得ます（sns_dsp）

  ディスパッチ禁止状態を得ます。

- ディスパッチ保留状態を得ます（sns_dpn）

  ディスパッチ保留状態を得ます。

3.5.17 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。
MR32R カーネルが提供する割り込み管理サービスコールには次のものがあります。

- 割り込みハンドラを定義します（def_inh,idef_inh）

  割り込みハンドラを指定された割り込みベクタに定義します。

3.5.18 システム構成管理機能

MR32R のバージョン情報を参照する機能です。

- MR32R のバージョンを得る(ref_ver,iref_ver)

  MR32R のバージョンを ref_ver サービス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。

3.5.19 拡張機能

拡張機能は、μITRON V.4.0 仕様外の機能です。

- データキューを初期化する(vrst_dtq)

  データキューを初期化します。送信待ちのタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- メールボックスを初期化する(vrst_mbx)

  メールボックスを初期化します。

- メッセージバッファを初期化する(vrst_mbf)

  メッセージバッファを初期化します。送信待ちのタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- 固定長メモリプールを初期化する(vrst_mpf)

  固定長メモリプールを初期化します。待ち状態のタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- 可変長メモリプールを初期化する(vrst_mpl)

  可変長メモリプールを初期化します。待ち状態のタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

3.6 サービスコール発行コンテキスト一覧

サービスコールにはタスクから発行できるものと非タスクコンテキスト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 3-4にその一覧を示します。

表 3-4　タスク、ハンドラから発行できるサービスコール一覧

サービスコール	タスクコンテキスト	非タスクコンテキスト
acre_tsk	○	×
cre_tsk	○	×
del_tsk	○	×
act_tsk	○	×
iact_tsk	×	○
can_act	○	×
ican_act	×	○
sta_tsk	○	×
ista_tsk	×	○
ext_tsk	○	×
exd_tsk	○	×
ter_tsk	○	×
chg_pri	○	×
ichg_pri	×	○
get_pri	○	×
iget_pri	×	○
ref_tsk	○	×
iref_tsk	×	○
ref_tst	○	×
iref_tst	×	○
slp_tsk	○	×
tslp_tsk	○	×
wup_tsk	○	×
iwup_tsk	×	○
can_wup	○	×
ican_wup	×	○
rel_wai	○	×
irel_wai	×	○
sus_tsk	○	×
isus_tsk	×	○
rsm_tsk	○	×
irsm_tsk	×	○
frsm_tsk	○	×
ifrsm_tsk	×	○
dly_tsk	○	×

サービスコール	タスクコンテキスト	非タスクコンテキスト
def_tex	○	×
ras_tex	○	×
iras_tex	×	○
dis_tex	○	×
ena_tex	○	×
sns_tex	○	×
ref_tex	○	×
iref_tex	×	○
cre_sem	○	×
acre_sem	○	×
del_sem	○	×
sig_sem	○	×
isig_sem	×	○
wai_sem	○	×
twai_sem	○	×
pol_sem	○	×
ipol_sem	×	○
ref_sem	○	×
iref_sem	×	○
cre_flg	○	×
acre_flg	○	×
del_flg	○	×
set_flg	○	×
iset_flg	×	○
clr_flg	○	×
iclr_flg	×	○
wai_flg	○	×
twai_flg	○	×
pol_flg	○	×
ipol_flg	×	○
ref_flg	○	×
iref_flg	×	○
cre_dtq	○	×
acre_dtq	○	×
del_dtq	○	×
snd_dtq	○	×

サービスコール	タスクコンテキスト	非タスクコンテキスト
tsnd_dtq	○	×
psnd_dtq	○	×
ipsnd_dtq	×	○
fsnd_dtq	○	×
ifsnd_dtq	×	○
rcv_dtq	○	×
trcv_dtq	○	×
prcv_dtq	○	×
iprcv_dtq	×	○
ref_dtq	○	×
iref_dtq	×	○
cre_mbx	○	×
acre_mbx	○	×
del_mbx	○	×
snd_mbx	○	×
isnd_mbx	×	○
rcv_mbx	○	×
trcv_mbx	○	×
prcv_mbx	○	×
iprcv_mbx	×	○
ref_mbx	○	×
iref_mbx	×	○
cre_mbf	○	×
acre_mbf	○	×
del_mbf	○	×
snd_mbf	○	×
tsnd_mbf	○	×
psnd_mbf	○	×
rcv_mbf	○	×
trcv_mbf	○	×
prcv_mbf	○	×
ref_mbf	○	×

サービスコール	タスクコンテキスト	非タスクコンテキスト
iref_mbf	×	○
cre_por	○	×
acre_por	○	×
del_por	○	×
cal_por	○	×
tcal_por	○	×
acp_por	○	×
tacp_por	○	×
pacp_por	○	×
fwd_por	○	×
rpl_rdv	○	×
ref_por	○	×
iref_por	×	○
ref_rdv	○	×
iref_rdv	×	○
cre_mpf	○	×
acre_mpf	○	×
del_mpf	○	×
get_mpf	○	×
tget_mpf	○	×
pget_mpf	○	×
rel_mpf	○	×
irel_mpf	×	○
ref_mpf	○	×
iref_mpf	×	○
ipget_mpf	×	○
cre_mpl	○	×
acre_mpl	○	×
del_mpl	○	×
get_mpl	○	×
tget_mpl	○	×
pget_mpl	○	×
rel_mpl	○	×
ref_mpl	○	×
iref_mpl	×	○
set_tim	○	×
iset_tim	×	○
get_tim	○	×
iget_tim	×	○
cre_cyc	○	×
acre_cyc	○	×
del_cyc	○	×

サービスコール	タスクコンテキスト	非タスクコンテキスト
sta_cyc	○	×
ista_cyc	×	○
stp_cyc	○	×
istp_cyc	×	○
ref_cyc	○	×
iref_cyc	×	○
cre_alm	○	×
acre_alm	○	×
sta_alm	○	×
ista_alm	×	○
stp_alm	○	×
istp_alm	×	○
del_alm	○	×
ref_alm	○	×
iref_alm	×	○
rot_rdq	○	×
irot_rdq	×	○
get_tid	○	×
iget_tid	×	○
loc_cpu	○	×
iloc_cpu	×	○
unl_cpu	○	×
iunl_cpu	×	○
dis_dsp	○	×
ena_dsp	○	×
sns_ctx	○	○
sns_loc	○	○
sns_dsp	○	○
sns_dpn	○	○
def_inh	○	×
idef_inh	×	○
ref_ver	○	×
iref_ver	×	○
vrst_mpf	○	×
vrst_mpl	○	×
vrst_mbx	○	×
vrst_mbf	○	×
vrst_dtq	○	×

第 4 章 アプリケーション作成手順概要

本章では MR32R を用いてアプリケーションプログラムを開発する手順の概要について説明します。

4.1 概要

MR32R 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。
2. **割り込み制御用プログラムの作成**
   割り込みコントローラ、タイマなどに依存する部分の処理のプログラムを作成します。
3. **コンフィギュ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ギュレーションファイルを作成します。作成は、GUI コンフィギュレータを使用して生成することもできます。
4. **コンフィギュレータ実行**
   コンフィギュ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（mr32r.inc、kernel_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。
5. **システム生成**
   make[39]コマンドを実行してシステムを生成します。
6. **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。

[39] 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。

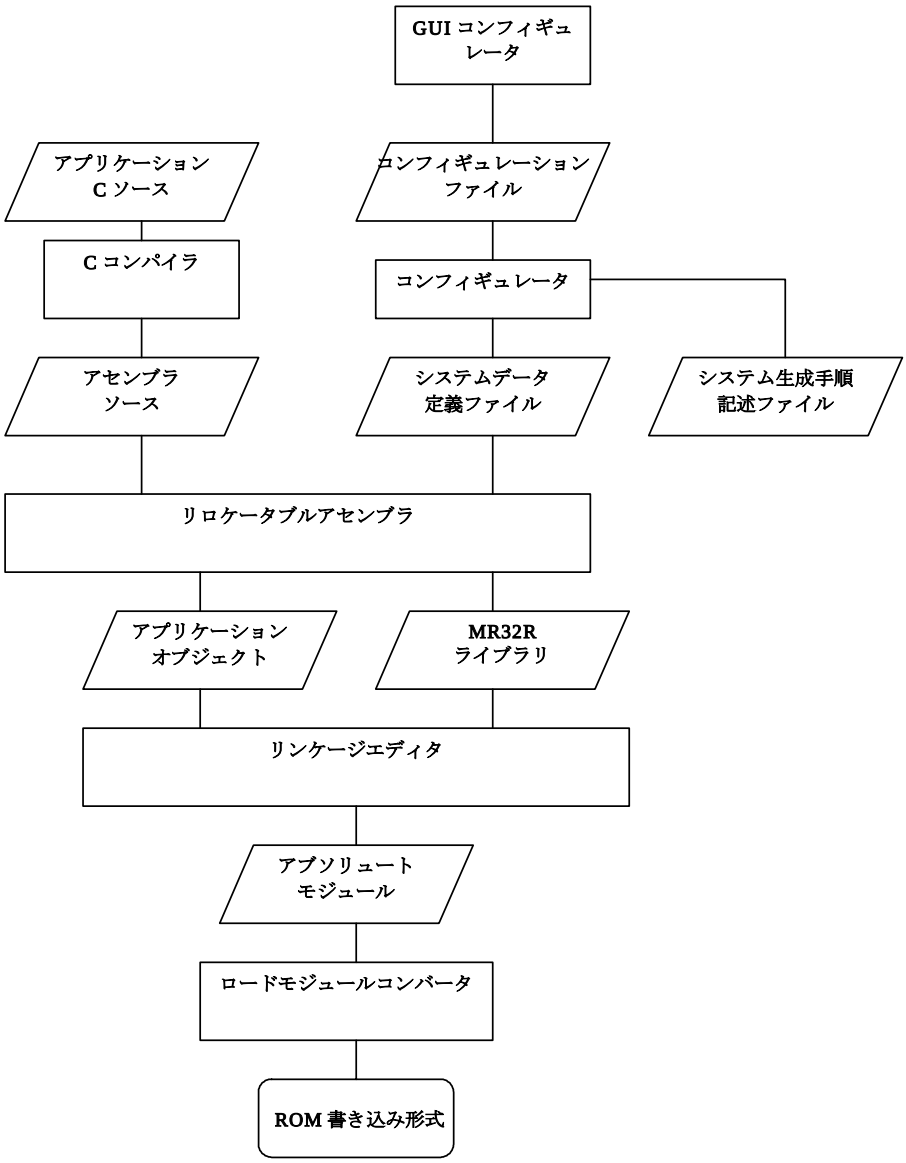


図 4.1　MR32R システム生成詳細フロー

4.2 開発手順例

この節では MR32R 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 MR32R ライブラリ中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番号はコンフィギュ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。[40]

- 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。

[40] 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。

```
 1
 2
 3 #include <itron.h>
 4 #include <kernel.h>
 5 #include "kernel_id.h"
 6
 7
 8  void main() /* main task */
 9  {
10        printf("LBP シミュレーション開始¥n");
11        sta_tsk(ID_idle,1);     /* アイドルタスク起動 */
12        sta_tsk(ID_image,1);    /* イメージ展開タスク起動 */
13        sta_tsk(ID_printer,1);  /* プリンタエンジンタスク起動 */
14 }
15 void image() /* イメージ展開タスク */
16 {
17        while(1){
18                wai_flg(&flgptn,ID_pagein,waiptn,TWF_ANDW);/* 1ページ入力待ち */
19
20                printf("ビットマップ展開処理¥n");
21                wup_tsk(ID_printer);    /* プリンタエンジンタスク起床 */
22        }
23 }
24 void printer() /* プリンタエンジンタスク */
25 {
26        while(1){
27                slp_tsk();
28                printf("プリンタエンジン動作¥n");
29        }
30 }
31 void sent_in() /* セントロニクスインターフェースハンドラ */
32 {
33
34
35        /* セントロニクスインターフェースからの入力処理 */
36        if ( /* 1ページ入力完了 */ )
37                iset_flg(ID_pagein,setptn);
38 }
```

図 4.2　プログラム例

4.2.2 コンフィギュレーションファイル作成

図 4.3 に示す GUI コンフィギュレータを使用してコンフィギュレーションファイルを作成します。GUI コンフィギュレータを使用して各設定が終了したら、コマンドラインコンフィギュレータの実行に必要なコンフィギュレーションファイルを図 4.4 のダイアログで生成するコンフィギュレーションファイル名を指定した後、開始ボタンを押して生成します。

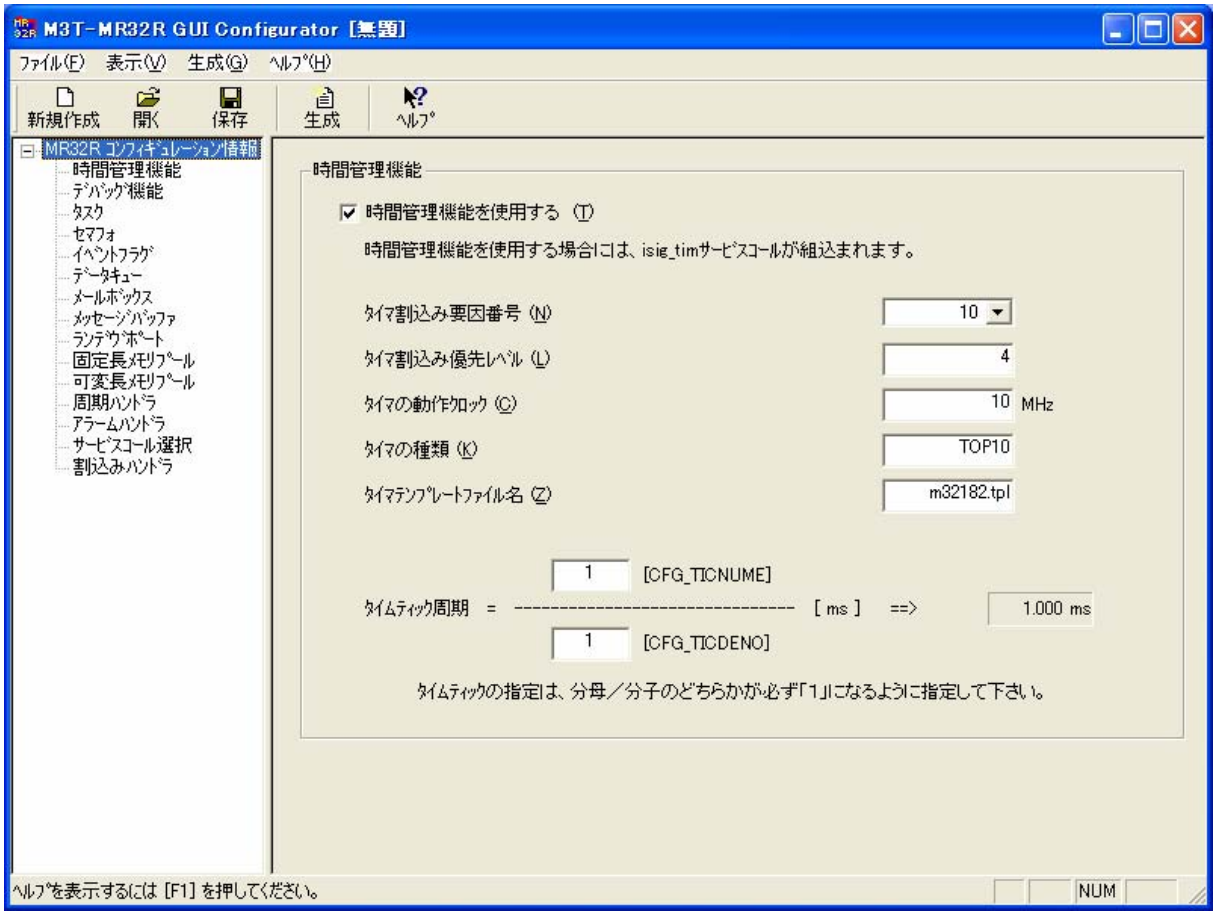


図 4.3 GUI コンフィギュレータ

図 4.4 コンフィギュレーションファイルの生成

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ギュレーションファイルを作成

します。図 4.5 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ギュレーションファイル（ファイル名 "lbp.cfg"）を示します。

```
 1 ////////////////////////////////////////////////////////////////////////////
 2 //
 3 //    kernel.cfg : building file for MR32R Ver.4.00
 4 //
 5 //    Generated by M3T-MR32R GUI Configurator at 2004/11/29 1:40:18
 6 //
 7 ////////////////////////////////////////////////////////////////////////////
 8
 9 // system definition
10 system{
11         stack_size      = 4096;
12         priority        = 6;
13         debug    = NO;
14         debug_buffer    = 4096;
15         message_pri      = 1;
16         tick_deno        = 1;
17         tick_nume        = 1;
18 };
19
20 // dynamic generation definition
21 // for task generation
22 int_memstk{
23         all_memsize     = 0;
24         max_memsize     = 0;
25 };
26 ext_memstk{
27         all_memsize     = 0;
28         max_memsize     = 0;
29 };
30
31 // for dataqueue generation
32 int_memdtq{
33         all_memsize     = 0;
34         max_memsize     = 0;
35 };
36 ext_memdtq{
37         all_memsize     = 0;
38         max_memsize     = 0;
39 };
40
41 // for message buffer generation
42 int_memmbf{
43         all_memsize     = 0;
44         max_memsize     = 0;
45 };
46 ext_memmbf{
47         all_memsize     = 0;
48         max_memsize     = 0;
49 };
50
51 // for fixed-size memorypool generation
52 int_memmpf{
53         all_memsize     = 0;
54         max_memsize     = 0;
55 };
56 ext_memmpf{
57         all_memsize     = 0;
58         max_memsize     = 0;
59 };
60
61 // for variable-size memorypool generation
62 int_memmpl{
63         all_memsize     = 0;
64         max_memsize     = 0;
65 };
66 ext_memmpl{
67         all_memsize     = 0;
68         max_memsize     = 0;
69 };
70
   71
```

```
 72 // max definition
 73 maxdefine{
 74         max_task        = 4;
 75         max_flag        = 1;
 76         max_sem = 0;
 77         max_dtq = 0;
 78         max_mbx = 0;
 79         max_mbf = 0;
 80         max_por = 0;
 81         max_mpf = 0;
 82         max_mpl = 0;
 83         max_cyh = 0;
 84         max_alh = 0;
 85         max_int = 65;
 86 };
 87
 88 // system clock definition
 89 clock{
 90         timer_clock     = 33.000000MHz;
 91         timer   = MFT00;
 92         IPL     = 4;
 93         file_name       = m32102.tpl;
 94 };
 95
 96 task[]{
 97         entry_address   = main();
 98         name    = ID_main;
 99         stack_size      = 256;
100         stack_area      = INTERNAL;
101         priority        = 1;
102         initial_start   = OFF;
103         exinf   = 0x0;
104 };
105 task[]{
106         entry_address   = image();
107         name    = ID_image;
108         stack_size      = 256;
109         stack_area      = INTERNAL;
110         priority        = 2;
111         initial_start   = OFF;
112         exinf   = 0x0;
113 };
114 task[]{
115         entry_address   = printer();
116         name    = ID_printer;
117         stack_size      = 256;
118         stack_area      = INTERNAL;
119         priority        = 3;
120         initial_start   = OFF;
121         exinf   = 0x0;
122 };
123 task[]{
124         entry_address   = idle();
125         name    = ID_idle;
126         stack_size      = 256;
127         stack_area      = INTERNAL;
128         priority        = 6;
129         initial_start   = OFF;
130         exinf   = 0x0;
131 };
132
133 flag[]{
134         name    = ID_pagein;
135         initial_pattern = 0x00000000;
136         wait_queue      = TA_TFIFO;
137         clear_attribute = YES;
138         wait_multi      = TA_WMUL;
139 };
140
141
142
143
144
145
146
147
148
149
150 interrupt_vector[16]    =  __sys_timer;
151 interrupt_vector[23]    =  sent_in();
```

```
152
153 // Service Call definition
154 systemcall{
155         cre_tsk =       NO;
156         acre_tsk =      NO;
157         del_tsk =       NO;
158         act_tsk =       YES;
159         iact_tsk =      YES;
160         can_act =       NO;
161         ican_act =      NO;
162         sta_tsk =       YES;
163         ista_tsk =      YES;
164         ext_tsk =       YES;
165         exd_tsk =       NO;
166         ter_tsk =       YES;
167         chg_pri =       NO;
168         ichg_pri =      NO;
169         get_pri =       NO;
170         iget_pri =      NO;
171         ref_tsk =       NO;
172         iref_tsk =      NO;
173         ref_tst =       NO;
174         iref_tst =      NO;
175
176         slp_tsk =       YES;
177         tslp_tsk =      YES;
178         wup_tsk =       YES;
179         iwup_tsk =      YES;
180         can_wup =       NO;
181         ican_wup =      NO;
182         rel_wai =       NO;
183         irel_wai =      NO;
184         sus_tsk =       NO;
185         isus_tsk =      NO;
186         rsm_tsk =       NO;
187         irsm_tsk =      NO;
188         frsm_tsk =      NO;
189         ifrsm_tsk =     NO;
190         dly_tsk =       YES;
191
192         def_tex =       NO;
193         ena_tex =       NO;
194         dis_tex =       NO;
195         ras_tex =       NO;
196         iras_tex =      NO;
197         sns_tex =       NO;
198         ref_tex =       NO;
199         iref_tex =      NO;
200
201         cre_sem =       NO;
202         acre_sem =      NO;
203         del_sem =       NO;
204         sig_sem =       NO;
205         isig_sem =      NO;
206         wai_sem =       NO;
207         twai_sem =      NO;
208         pol_sem =       NO;
209         ipol_sem =      NO;
210         ref_sem =       NO;
211         iref_sem =      NO;
212
213         cre_flg =       YES;
214         acre_flg =      YES;
215         del_flg =       YES;
216         set_flg =       YES;
217         iset_flg =      YES;
218         clr_flg =       YES;
219         iclr_flg =      YES;
220         wai_flg =       YES;
221         pol_flg =       YES;
222         ipol_flg =      YES;
223         twai_flg =      YES;
224         ref_flg =       YES;
225         iref_flg =      YES;
226
227         cre_dtq =       NO;
228         acre_dtq =      NO;
229         del_dtq =       NO;
230         snd_dtq =       NO;
231         tsnd_dtq =      NO;
```

```
232         psnd_dtq =      NO;
233         ipsnd_dtq =     NO;
234         fsnd_dtq =      NO;
235         ifsnd_dtq =     NO;
236         rcv_dtq =       NO;
237         trcv_dtq =      NO;
238         prcv_dtq =      NO;
239         iprcv_dtq =     NO;
240         ref_dtq =       NO;
241         iref_dtq =      NO;
242
243         cre_mbx =       NO;
244         acre_mbx =      NO;
245         del_mbx =       NO;
246         snd_mbx =       NO;
247         isnd_mbx =      NO;
248         rcv_mbx =       NO;
249         trcv_mbx =      NO;
250         prcv_mbx =      NO;
251         iprcv_mbx =     NO;
252         ref_mbx =       NO;
253         iref_mbx =      NO;
254
255         cre_por =       NO;
256         acre_por =      NO;
257         del_por =       NO;
258         cal_por =       NO;
259         tcal_por =      NO;
260         acp_por =       NO;
261         tacp_por =      NO;
262         pacp_por =      NO;
263         fwd_por =       NO;
264         rpl_rdv =       NO;
265         ref_por =       NO;
266         iref_por =      NO;
267         ref_rdv =       NO;
268         iref_rdv =      NO;
269
270         cre_mbf =       NO;
271         acre_mbf =      NO;
272         del_mbf =       NO;
273         snd_mbf =       NO;
274         tsnd_mbf =      NO;
275         psnd_mbf =      NO;
276         rcv_mbf =       NO;
277         prcv_mbf =      NO;
278         trcv_mbf =      NO;
279         ref_mbf =       NO;
280         iref_mbf =      NO;
281
282         cre_mpf =       NO;
283         acre_mpf =      NO;
284         del_mpf =       NO;
285         get_mpf =       NO;
286         tget_mpf =      NO;
287         pget_mpf =      NO;
288         ipget_mpf =     NO;
289         rel_mpf =       NO;
290         irel_mpf =      NO;
291         ref_mpf =       NO;
292         iref_mpf =      NO;
293
294         cre_mpl =       NO;
295         acre_mpl =      NO;
296         del_mpl =       NO;
297         get_mpl =       NO;
298         tget_mpl =      NO;
299         pget_mpl =      NO;
300         rel_mpl =       NO;
301         ref_mpl =       NO;
302         iref_mpl =      NO;
303
304         set_tim =       NO;
305         iset_tim =      NO;
306         get_tim =       NO;
307         iget_tim =      NO;
308
309         cre_cyc =       NO;
310         acre_cyc =      NO;
311         del_cyc =       NO;
```

```
312         sta_cyc =       NO;
313         ista_cyc =      NO;
314         stp_cyc =       NO;
315         istp_cyc =      NO;
316         ref_cyc =       NO;
317         iref_cyc =      NO;
318
319         cre_alm =       NO;
320         acre_alm =      NO;
321         del_alm =       NO;
322         sta_alm =       NO;
323         ista_alm =      NO;
324         stp_alm =       NO;
325         istp_alm =      NO;
326         ref_alm =       NO;
327         iref_alm =      NO;
328
329         rot_rdq =       NO;
330         irot_rdq =      NO;
331         get_tid =       NO;
332         iget_tid =      NO;
333         loc_cpu =       NO;
334         iloc_cpu =      NO;
335         unl_cpu =       NO;
336         iunl_cpu =      NO;
337         dis_dsp =       NO;
338         ena_dsp =       NO;
339         sns_ctx =       NO;
340         sns_loc =       NO;
341         sns_dsp =       NO;
342         sns_dpn =       NO;
343
344         def_inh =       NO;
345         idef_inh =      NO;
346
347         ref_ver =       NO;
348         iref_ver =      NO;
349
350         vrst_dtq =      NO;
351         vrst_mpf =      NO;
352         vrst_mpl =      NO;
353         vrst_mbf =      NO;
354         vrst_mbx =      NO;
355
356 };
357
358
359 //
360 // End of Configuration
361 //;
```

図 4.5　コンフィギュレーションファイル出力例

4.2.3 コンフィギュレータ実行

コンフィギュレータ cfg32r を実行して、コンフィギュ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(mr32r.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> cfg32r  -mv  lbp.cfg
MR32R system configurator V.4.00.00 (for CC32R)
COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED.
MR32R version ==> V.4.00 Release 00
A>
```

図 4.6　コンフィギュレータ実行

4.2.4 システム生成

make コマンド[41]を実行してシステムを生成します。

```
C> make -f  makefile
as32R -g crt0mr.ms
cc32R -c task.c
lnk32R lnk32R.sub

C>
```

図 4.7　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 LMC32R(**M3T-CC32R 対応**)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。

[41] 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。**MR32R** 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。Windows 版を使用する場合は、UNIX 互換の make コマンド（GNU make コマンドなど）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下さい。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 MR32R に登録するにはコンフィギュ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();
        name            = ID_task;
        stack_size      = 100;
        priority        = 3;
};
```

2. **ファイル先頭で必ずシステムディレクトリのなかの“itron.h”、”kernel.h”とカレントディレクトリ内の“kernel_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３行を必ず記述してください。**

```
#include <itron.h>
#include <kernel.h>
#include “kernel_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は、ext_tsk()を記述する必要はありません。カーネルが自動的に ext_tsk を呼び出します。**
6. **タスク開始関数を無限ループで記述することも可能です。**

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”

void task(VP_INT stacd)
{

        /* 処理 */

}
```

図 5.1　C 言語で記述したタスクの例

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(VP_INT stacd)
{
        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コンフィギュレーションファイルの"name"あるいは、GUI コンフィギュレータで"ID 名称"に記述した文字列で指定してください[42]。**

```
wup_tsk(ID_main);
```

8. **イベントフラグ、セマフォ、メールボックスを指定する場合は、コンフィギュレーションファイルで定義したそれぞれの"name"あるいは、GUI コンフィギュレータの"ID 名称"に指定した文字列で指定してください。**

例えば、コンフィギュ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    = ID_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。

[42] 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"kernel_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"kernel_id.h"で行います。

```
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、コンフィギュレーションファイルの"name"あるいは、GUI コンフィギュレータで"ID 名称"に記述した文字列で指定してください。**

```
sta_cyc(ID_cyc);
```

10. **タスクを** ter_tsk() **サービスコールなどで終了した後で** sta_tsk(),act_tsk()**サービス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[43]が、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** MR32R **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でのみおこないます。**

11. MR32R **システム起動時に起動されるタスクは、コンフィギュ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 MR32R から見て表 5-1に示す扱いになります。

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

[43] 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。

5.1.2 割り込みハンドラの記述方法

C 言語を用いて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. **割り込みハンドラは関数として記述します。**[44]
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. **ファイル先頭で必ずシステムディレクトリのなかの** “itron.h” 、”kernel.h”**とカレントディレクトリ内の** “kernel_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３行を必ず記述してください。**
4. **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 <itron.h>
#include <kernel.h>
#include “kernel_id.h”

void int_handler(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

[44] 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。

5.1.3 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。**[45]
2. **関数の戻り値を、**void **型で宣言してください。**
3. **関数の引数は、拡張情報が** VP_INT **型で渡されます。**
4. **ファイル先頭で必ずシステムディレクトリのなかの** “itron.h” 、”kernel.h”**とカレントディレクトリ内の** “kernel_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３行を必ず記述してください。**
5. **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。**
6. **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 <itron.h>
#include <kernel.h>
#include “kernel_id.h”

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

図 5.4　C 言語で記述した周期起動ハンドラの例

[45] 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。

5.1.4 タスク例外の記述方法

C 言語を用いてタスク例外を記述する場合、以下の点に注意してください。

1. **例外ハンドラは関数として記述します。**
2. **関数の戻り値を、**void **型で宣言してください。**
3. **関数の引数は、保留例外要因が** TEXPTN **型、タスク拡張情報が** VP_INT **型で渡されます。**
4. **ファイル先頭で必ずシステムディレクトリのなかの** “itron.h” 、”kernel.h”**とカレントディレクトリ内の** “kernel_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３行を必ず記述してください。**
5. **スタティック宣言をおこなった関数はタスク例外として登録できません。**
6. **タスク例外の終了は、**return **で終了します。**
7. **タスク例外処理中は、タスクコンテキストから発行可能なサービスコールを発行することができます。**

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”


void tex1 (TEXPTN texptn, VP_INT exinf)
{
    /* 処理 */
        sig_sem(ID_sem1);

}
```

図 5.5　C 言語で記述したタスク例外の例

第 6 章 アプリケーション作成時の注意事項

6.1 初期起動タスクについて

MR32R 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ギュ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、141ページを参照して下さい。

6.2 動的生成・削除機能について

MR32R では動的生成機能(cre_tsk,cre_mpf,cre_mpl など)を使用する際、可変長メモリプール機能と同様に４つのサイズのメモリブロックのうち最適なサイズを割り当てるようにしています。対象となるサービスコールをに示します。

表 6-1　動的生成・削除サービスコール一覧

サービスコール	用途
cre_tsk,del_tsk,exd_tsk	スタック領域
cre_dtq,del_dtq	データキュー領域
cre_mbf,del_mbf	メッセージバッファ領域
cre_mpf,del_mpf	固定長メモリプール領域
cre_mpl,del_mpl	可変長メモリプール領域

4 つのメモリブロックのサイズ(a,b,c,d)も可変長メモリプールと同様に、以下のように求めることができます。以下のサイズは、管理情報を含むため実際にはこれより 12 バイト少ない値になります。

```
A =(((max_memsize +(12-1))/96)+1)*12
b =a*2
c =a*4
d =a*8
```

以下に例を挙げます。

【例】

- **ユーザプログラム内で"__MR_EXT__"を指定して生成したタスク**

  タスク A　:　256 バイトのスタック
  タスク B　:　1024 バイトのスタック

この場合、ID=3 のタスクのスタックサイズが最大となるので max_memsize = 1024 以上を指定します。コンフィギュレーションファイルの ext_memstk 定義を max_memsize = 1024 とした場合、4 つのメモリブロックのサイズは上記計算式より以下のようになります。

種類	サイズ	実使用可能サイズ
a	132	120
b	264	252
c	528	516
d	1056	1040

(差分の 12 バイトは管理領域として使用)

タスク A のスタックサイズは、256 バイトとなりますのでサイズ c(=528)のブロックが割り当てられ、タスク B のスタックサイズ(1024 バイト)は d(=1056)バイト割り当てられるようになります。
したがって、要求サイズ、実使用サイズ、無駄なサイズの関係は以下のようになります。

	要求サイズ	実使用サイズ	無駄なサイズ
タスク A	256	528	272
タスク B	1024	1056	32
Total	1280	1584	304

従って、コンフィギュレーションファイルには、下記のように記載すればメモリ不足となる事はありません。

- **コンフィギュレーションファイルの記述**

```
ext_memstk{
        max_memsize     = 1024;
        all_memsize     = 1584;
};
```

しかし、この場合、無駄な領域が 304 バイトできます。
無駄な領域を減らすためには、max_memsize を調整する必要があります。例えば、max_memsize=1048 とすれば、a=144 b=288 c=576 d=1152 となり、

	要求サイズ	実使用サイズ	無駄なサイズ
タスク A	256	288	32
タスク B	1024	1152	128
Total	1280	1440	160

と無駄なサイズを減らすことが可能です。

本方式では、フラグメンテーションは、発生しにくくなっていますが、全く発生しないわけではありません。そのため、空き領域のサイズは十分あっても連続した空き領域がないため、メモリが割り当てられないこともあるので注意してください。

6.3 TRAP 命令の使用について

MR32R では、TRAP7 をサービスコール発行のため使用しています。また、TRAP8～12 は、将来の機能拡張のため予約領域として扱っています。
ユーザアプリケーションでソフトウェア割り込みを使用する場合は、7～12 以外の TRAP エントリを使用して下さい。

6.4 割り込みについて

6.4.1 割り込み制御方法

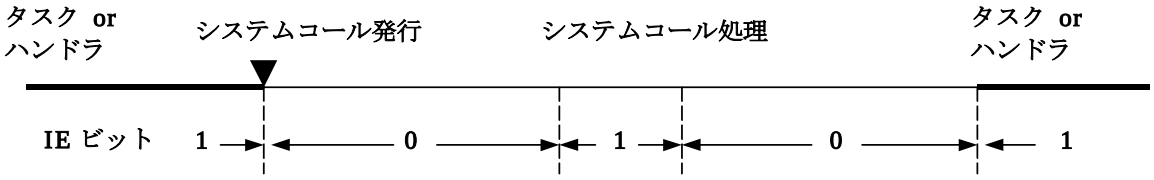
サービスコール内の割り込み禁止/許可の制御は、PSW の IE ビット操作により行っています。

サービスコール内での IE ビットは、クリアされており、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、サービスコール発行時の IE ビットに戻します。

図 6.1に、サービスコール内での IE ビットの状態を示します。

- タスクからのみ発行できるサービスコールの場合

・システムコール発行前の IE ビットが 1 の場合



・システムコール発行前の IE ビットが 0 の場合

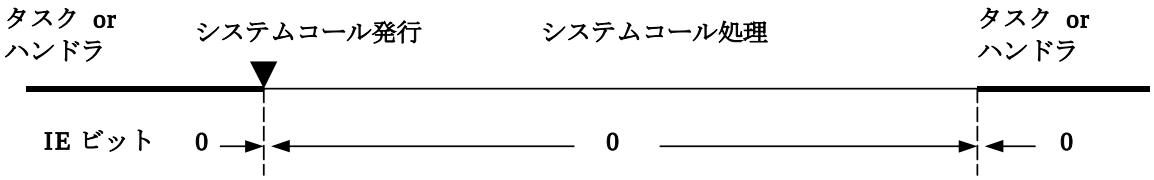


図 6.1　サービスコール内での割り込み制御

図 6.1に示すように IE ビットは、サービスコール内で変化します。そのため、ユーザアプリケーション内で割り込みを禁止したい場合、IE ビット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を変更する。**
2. loc_cpu ～ unl_cpu **を使用する。**

6.4.2 割り込みハンドラの処理手順

MR32R では、図 6.2のような手順で割り込みハンドラの処理を行います。

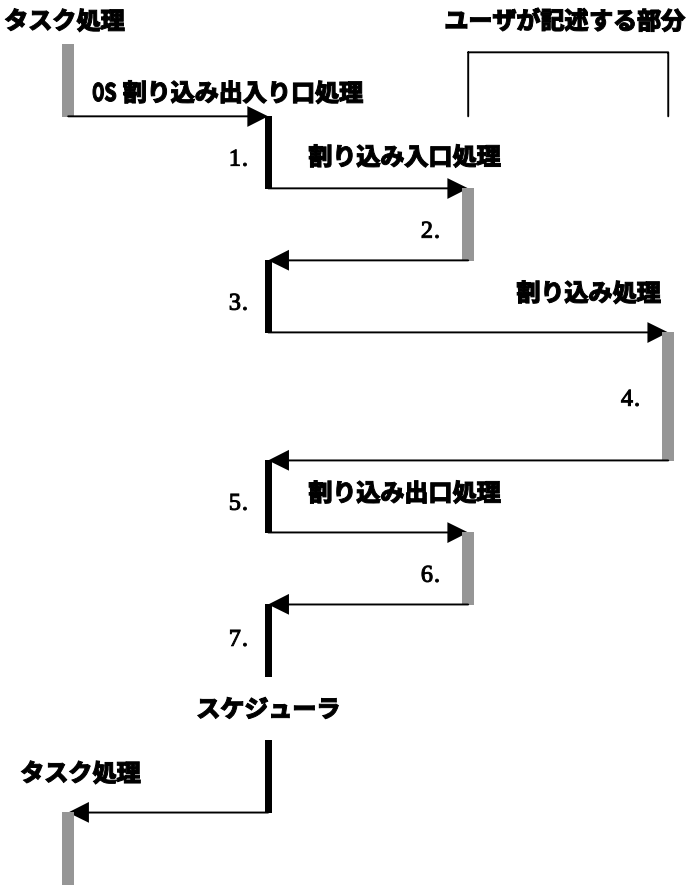


図 6.2 割り込みハンドラの処理手順

1. レジスタを保存し、ユーザの割り込み入り口処理ルーチンをコールする。
2. ユーザの割り込み入り口処理ルーチン
   - 割り込みレベルを読み、スタックに退避します。
   - 割り込み要因を読み、割り込みのジャンプ先アドレスをスタックに退避します。
3. スタック上に積まれた割り込みハンドラのアドレスをコールする。
4. 割り込み処理ルーチン
5. ユーザの割り込み出口処理ルーチンをコールする。
6. 割り込み出口処理
   - 割り込みレベルを元に戻します。
7. レジスタを復帰し、スケジューラへジャンプする。

上記処理の2と6は、割込み制御プログラム(ipl.ms)内でユーザが記述する必要があります。(詳細は、8.1割り込み制御プログラムについてを参照ください。)その他の部分については、OS で処理を行います。

6.4.3 ハンドラ実行時の割り込みの受付について

ハンドラ実行時の割り込み可否の状態は以下のようになっています。

- 周期起動ハンドラ、アラームハンドラ
  周期起動ハンドラ・アラームハンドラは、割り込み許可状態で起動されます。
- 割り込みハンドラ
  割り込みハンドラは、デフォルトでは割り込み禁止状態で起動されます。多重割り込みを許可する場合は、「6.4.4 多重割り込みの許可方法」を参照ください。

6.4.4 多重割り込みの許可方法

多重割り込みを許可するには下記に示す処理を行ってください。

- すべての割り込みに対して多重割り込みを許可する場合

  1. 割り込み入口処理の 2.のルーチン内で、割り込み許可ビットをセットします。

  2. 割り込み出口処理の 6.のルーチン内で、割り込み許可ビットをクリアします。

- ある特定の割り込みだけ多重割り込みを許可する場合

  1. 多重割り込みを許可する割り込みハンドラの先頭で割り込み許可ビットをセットします。

  2. 多重割り込みを許可する割り込みハンドラの最後で割り込み許可ビットをクリアします。

6.5 OS デバッグ機能使用手順

OS デバッグ機能(タスクトレース、サービスコールトレース、サービスコール発行)を使用するための手順および注意事項を以下に示します。

6.5.1 OS デバッグ機能使用手順

1. コンフィギュレーションファイルのシステム定義項目を編集します。
   定義方法についての詳細は、第 7 章　コンフィギュレータの使用方法を参照してください。
2. フックルーチンのライブラリ mrdbg.lib をリンクするようにします。
3. バッファ領域のセクション(MR_dbg_RAM)を任意の領域に配置します。
4. サービスコール発行機能を使用する場合は、発行するサービスコールもコンフィギュレーションファイルの使用サービスコール定義で"YES"を指定します
5. 割り込みハンドラの定義でシステムクロック割り込みを__sys_timer としていたものを__Dbg_sys_timer に変更してください。
   (__Dbg_sys_timer は、サービスコール発行機能用のフックルーチンです)

6.5.2 OS デバッグ機能使用時の注意事項

- サービスコール発行機能で発行可能なサービスコールは、デバッガのマニュアルを参照してください。
- フックルーチンをリンクすることにより、OS の処理時間・割り込み禁止時間が長くなります。
- サービスコールトレース、タスクトレース機能を制御するために"__Dbg_mode"という 1 バイトのデータで OS とデバッガとやりとりを行っています。標準では、スタートアップファイル内で、トレースを行わないように OS がデータを書き換えています。そのため、プログラムをダウンロードし、デバッガの MR トレースウインドウを開いても、トレースができません。OS がデータを書き換えてから MR トレースウインドウを開くか、いったん OS がデータを書き換えたところでブレークしてから再度実行してください。デフォルトでトレースを ON にするには、スタートアップファイルを以下のように変更してから使用してください。

[変更後]

```
;       seth    R11,#high(__REL_BASE11)
;       or3     R11,R11,#low(__REL_BASE11)
;       seth    R12,#high(__REL_BASE12)
;       or3     R12,R12,#low(__REL_BASE12)
;       seth    R13,#high(__REL_BASE13)
;       or3     R13,R13,#low(__REL_BASE13)


.AIF    ¥&__Dbg_flg gt 0
ld24    r1,#__Dbg_mode
ldi     r2,#0           ←R2 レジスタの設定値を変更する
stb     r2,@r1            この例では"AFTER"モードに設定
.AENDI
```

[変更前]

```
;       seth    R11,#high(__REL_BASE11)
;       or3     R11,R11,#low(__REL_BASE11)
;       seth    R12,#high(__REL_BASE12)
;       or3     R12,R12,#low(__REL_BASE12)
;       seth    R13,#high(__REL_BASE13)
;       or3     R13,R13,#low(__REL_BASE13)


        .AIF    ¥&__Dbg_flg gt 0
        ld24    r1,#__Dbg_mode
;       ldi     r2,#0
        ldi     r2,#4
```

6.6 システムクロックの設定について

6.6.1 システムクロック処理の登録について

MR32R では、システムクロックを使用する場合、システムクロック処理(__sys_timer というラベルのアセンブラルーチン)を割り込みハンドラとして登録する必要があります。
登録は、通常の割り込みハンドラと同様にコンフィギュレーションファイルで定義します。

例えば、割り込み要因番号 0 から 63 までの 64 の割り込み要因が存在し、割り込み要因番号 16 をシステムクロックとする場合以下のようにコンフィギュレーションファイルに記述します。

```
【例】interrupt_vector[16]      =       __sys_timer;
```

システムクロック処理に関する注意事項

- **テンプレートファイルとして M32182.tpl を指定した場合**
  テンプレートファイルとして m32182.tpl を指定した場合、タイマ割り込みは、同じ割り込みエントリを使用しますので、どのタイマ割り込みが入ったか判定するルーチンは別途ユーザ側で用意する必要があります。コンフィギュレーションファイルの割り込み定義で設定したルーチンがその判定ルーチンにあたります。システムクロック割り込みが発生した場合、この判定ルーチンでシステムクロック割り込みが発生したか判定し、システムクロック処理(__sys_timer)を呼び出すようにします。
  (__sys_timer を呼び出す際は、「bra 命令」ではなく「bl 命令」で呼び出すようにしてください。)

6.6.2 タイマの自動設定について

タイマの自動設定の仕組み

MR32R では、いくつかのマイコンについては、タイマの初期化や、ICU の設定などを、コンフィギュレータによって自動的に出力できるようになっています。

その仕組みの概要は次のとおりです。まずコンフィギュレーションファイルに、テンプレートファイル・タイマの種類・割り込み間隔・入力周波数を設定します。コンフィギュレータは、環境変数 LIB32R にあるテンプレートファイルをコンフィギュレータの実行ディレクトリに"timer.inc"というファイル名でコピーすると同時に、タイマの種類・割り込み間隔・入力周波数などをシンボル定義し、"mr32r.inc"に出力します。"timer.inc"は、ipl.ms ファイルでインクルードされるようになっており、"mr32r.inc"に出力されたシンボル定義をもとにタイマの設定を行います。

タイマの自動設定機能に関する注意事項

- **タイマの自動設定機能を使用する場合**
  コンフィギュレータの起動オプションに"-i"オプションをつけて実行する必要があります。
  コンフィギュレータを実行したディレクトリにすでにタイマ設定ファイル"timer.inc"がコピーされている場合は、あらたにコピーされることはありません。
  タイマの自動設定において、タイマカウンタ設定値のオーバーフローのチェックを行っていません。
- **タイマの自動設定機能を使用しない場合**
  TM を使用せずにコマンドラインで使用し、"-m"オプションをつけて makefile を自動的に生成する場合は、割り込み制御プログラムを作成してからコンフィギュレータを実行してください。割り込み制御プログラムを作成せずに"-m"オプション付きで実行した場合、"timer.inc"をインクルードするように設定された割り込み制御プログラムがコピーされてしまい、"timer.inc"がないためにコンフィギュレータがエラーを出力します。

(ユーザが作成する際の割り込み制御プログラムは、サンプルプログラムが格納されているディレクトリにあるものを参考にしてください。)

6.7 ベースレジスタ機能の使用方法

CC32R V.3.00 でサポートされたベースレジスタ機能を使用する場合、下記の手順で使用します。(ベースレジスタ機能の詳細については、「CC32R ユーザーズマニュアル<<C コンパイラ編>>」拡張機能リファレンスを参照ください。)

1. ベースアドレスを決定します。
2. アクセス制御ファイルを作成します。
3. **ベースシンボルの定義を行います。**

   ベースシンボルの定義は、OS のスタートアップルーチンで行います。

   【例】R11=0x00FC8000,R12=0x00F88000,R13=未使用に設定する場合

   OS に添付されているスタートアップルーチン(crt0mr.ms または start.ms)の 48 行目から 66 行目を下記のように変更します。

```
        .global  __REL_BASE11
        .global  __REL_BASE12
        .global  __REL_BASE13
__REL_BASE11:    .equ     0x00FC8000        ← ①ベースシンボルを定義します。
__REL_BASE12:    .equ     0x00F88000
__REL_BASE13:    .equ     0

__START:
        seth     r1,#high(__Sys_Sp)
        or3      r1,r1,#low(__Sys_Sp)
        addi     r1,#-4
        mvtc     r1,SPI               ;SPI initialize
        mvtc     r1,SPU
        ldi      r0,#-1
        st       r0,@r1
```

(注意)

- タスク・割り込みハンドラ起動時のベースレジスタの設定は、OS カーネルが行いますので、スタートアップルーチンから C 言語で記述された関数(I/O の初期設定関数など)を呼び出さない場合、ベースレジスタの設定は不要です。
- **ベースレジスタとして使用しないレジスタがある場合、ベースレジスタ機能をまったく使用しない場合であっても、R11～R13 までのベースシンボルが定義されている必要があります。ただし、その値はどのような値でもかまいません。**

4. コンパイル・リンクを行います。

6.8 メモリ配置

ここでは、MR32R で使用するセクションについて説明します。

● C 言語使用時に使用するセクション

- **P セクション**

  ユーザのアプリケーションプログラムが配置されます。

- **B セクション**

  初期値なしのデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **C セクション**

  定数データが配置されます。

- **D セクション**

  初期値ありのデータが配置されます。
  このセクションは、ROM から RAM へ転送して使用します。

● MR32R で使用するセクション

- **SYS_STACK セクション**

  システムスタック領域です。MR32R カーネル内部、割り込みハンドラなどで使用します。
  このセクションは、RAM 上に配置する必要があります。

- **INT_USR_STACK セクション**

  内蔵 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **EXT_USR_STACK セクション**

  外部 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **MR_KERNEL・MR_KERNEL2 セクション**

  MR32R カーネルが配置されます。

- **MR_RAM セクション**

  MR32R が使用する内蔵 RAM のデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **MR_ROM セクション**

  MR32R が使用する固定データが配置されます。
  動的生成機能を使用する場合は、ROM から RAM へ転送して使用します。

- **MR_HEAP セクション**

  MR32R が使用する内蔵 RAM のヒープ領域です。データキュー、メッセージバッファ、可変長メモリプール、固定長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

- **EXT_MR_HEAP セクション**

  MR32R が使用する外部 RAM のヒープ領域です。データキュー、メッセージバッファ、可変長メモリプール、固定長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

● 割り込みベクタ関連のセクション

- **Int_Vector セクション**

- **EIT_Vector セクション**

  EIT ベクタ領域を格納するセクションです。

- **RESET_VECT セクション**

  リセットベクタ領域です。

- **INTERRUPT_VECT セクション**

  コンフィギュレータにより出力された割り込みベクタテーブルが格納されているセクションです。
  def_inh サービスコール使用時は、ROM から RAM へ転送して使用します。

6.8.1 メモリ配置に関する注意事項

MR32R のカーネル領域やデータ領域は 16MB を超える空間に配置可能です。セクション配置については、下記の制限があります。

- EVB(EIT ベクタベースレジスタ)をサポートしていないもしくは、16MB を超える領域に EVB を設定できないマイコンでは、6.8.2に示す処理が必要です。

6.8.2 カーネル領域の 16MB を超える空間への配置について

MR32R では、サービスコールの呼び出しに TRAP 命令を使用しています。しかし、TRAP は、4 バイトのエントリしかないため、直接 16MB を超える空間にジャンプする事ができません。従って、EVB(EIT ベクタベースレジスタ)をサポートしていないもしくは、16MB を超える領域に EVB を設定できないマイコンでは、①いったん 16MB 空間内にジャンプし、そこから 16MB を越える空間に jmp,jl 命令によってジャンプする必要があります。また、これらの命令を使用するにはレジスタを 1 つ使用しますので、②このレジスタの退避・復帰処理が必要となります。

MR32R カーネル領域(MR_KERNEL,MR_KERNEL2 セクション)を 16MB を越える空間に配置するためには、上記①②の処理が必要となります。H'2000000 番地以降に MR32R カーネル領域を配置した場合の TRAP7 使用サービスコール処理の例を下図に示します。

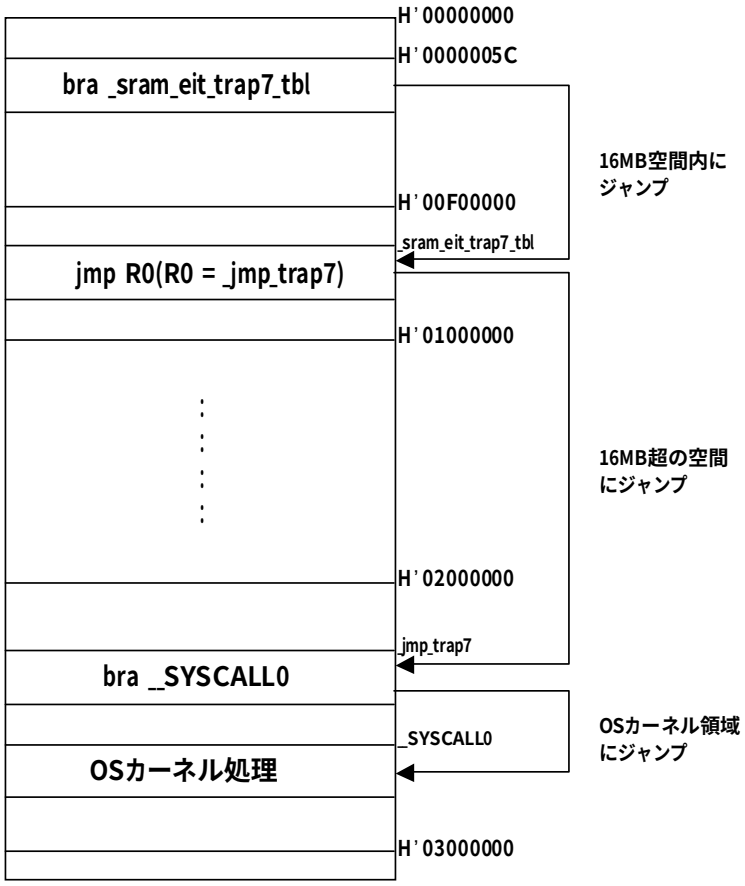


図 6.3 OS カーネルの 16MB 超領域への配置

以下に詳細手順を示します。

(1)スタートアップルーチンを変更し、TRAP,EI ベクタのジャンプ先を変更します。

```
.global _sram_eit_trap7_tbl
                :
bra     _sram_eit_trap7_tbl             ; TRAP7
                :
.section        Int_Vector,code,align=4
st      r0,@-r15
seth    r0,#high(_jmp_ei)
or3     r0,r0,#low(_jmp_ei)
jmp     r0
```

(2)スタートアップルーチンに 2 段目ジャンプ用のコードを追記します。このセクションは、16MB 空間内に配置します。

```
.section        ROM_MR_EIT,code,align=4
.section        MR_EIT,code,align=4
.export _sram_eit_trap7_tbl
_sram_eit_trap7_tbl:
st      r0,@-r15
seth    r0,#high(_jmp_trap7)
or3     r0,r0,#low(_jmp_trap7)
jmp     r0
```

(3)スタートアップルーチンに 3 段目ジャンプ用のコードを追記します。このセクションは、OS カーネル領域と同じセクション(MR_KERNEL)とします。

```
.section        MR_KERNEL
_jmp_trap7:
.AIF    ¥&__Dbg_flg eq 0
.global __SYSCALL0
ld      r0,@r15+
bra     __SYSCALL0
.AELSE
.global __Dbg_entry0
ld      r0,@r15+
bra __Dbg_entry0
.AENDI

_jmp_ei:
.global __int_entry
ld      r0,@r15+
bra __int_entry
```

第 7 章 コンフィギュレータの使用方法

7.1 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。[46]

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 7-1　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

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 7-2に示します。

[46] 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。

表 7-2　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
- (234 + 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。

7.2 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[47]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
- 最大項目定義
- 動的生成用ユーザスタック領域定義
- 動的生成用データキュー領域定義
- 動的生成用メッセージバッファ領域定義
- 動的生成用固定長メモリプール領域定義
- 動的生成用可変長メモリプール領域定義
- タスク定義
- イベントフラグ定義
- セマフォ定義
- データキュー定義
- メールボックス定義
- メッセージバッファ定義
- ランデヴ定義
- 固定長メモリプール定義
- 可変長メモリプール定義
- 周期ハンドラ定義

[47] 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。

- アラームハンドラ定義
- 割り込みベクタ定義

7.2.1 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 ;
    debug           = デバッグ機能;
    debug_buffer    = デバッグ機能で使用するバッファのサイズ;
    message_pri     = メッセージ優先度の最大値;
    tick_deno       = タイムティックの分母;
    tick_nume       = タイムティックの分子;
 };
```

<< 内容 >>

1. システムスタックサイズ

【 定義形式 】数値

【 定義範囲 】76 以上

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

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 ～ 255

MR32R 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。[48]

3. デバッグ機能

【 定義形式 】シンボル

【 定義範囲 】YES or NO

OS デバッグ機能に対応したデバッガ(PD32RM など)で MR トレース機能、サービスコール発行機能を使用する場合は YES、使用しない場合は、NO を指定します。YES を指定することにより、OS デバッグ機能を実現するためのフックルーチンを呼び出します。

[48] **MR32R** の優先度は、値が大きいほど優先度は低くなります。

4. デバッグ機能に使用するバッファサイズ

【 定義形式 】数値

【 定義範囲 】0 以上(4 の倍数)

デバッグ機能で使用するバッファのサイズを指定します。指定したサイズは 4 **の倍数に丸められます。**4096 **バイト確保した場合、少なくとも約** 60 **回分のタスク切り替えがトレースできます。**

5. メッセージ優先度の最大値

【 定義形式 】数値

【 定義範囲 】1 ～ 255

メールボックス機能で使用するメッセージ優先度の最大値を指定します。MR32R **は、この値は関知しません。**

6. タイムティックの分母

【 定義形式 】数値

【 定義範囲 】1 ～ 1000

タイムティック分母の値を指定します。タイムティック分母と分子のどちらかを必ず１にしなければいけません。
システムクロック割込の間隔は、次のような式で計算します。

システムクロック割込間隔(単位 ms) = tick_nume / tick_deno

例えば、システムクロックの周期間隔を 10ms **とする場合は、**tick_deno =1,tick_nume = 10 **とします。**
システムクロックの周期間隔が 0.1ms **の場合は、**tick_deno = 10,tick_nume = 1 **とします。**
tick_deno = 5, tick_nume = 4 **と分母、分子のいずれも１でない場合はエラーとなります。**

7. タイムティックの分子

【 定義形式 】数値

【 定義範囲 】1 ～ 1000

タイムティック分子の値を指定します。タイムティック分母と分子のどちらかを必ず１にしなければいけません。

7.2.2 動的生成用ユーザスタック領域定義(内蔵 RAM)

<< 形式 >>

```
int_memstk{
    max_memsize     = 生成するタスクのスタックサイズの最大値;
    all_memsize     = タスク生成用ユーザスタック領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するタスクのスタックサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_tsk、acre_tsk サービスコールの発行により生成するタスクの中でスタックサイズが最も大きい値を指定します。ここでの指定は、スタックに使用する領域を内蔵 RAM に指定したタスクの中で、最も大きいサイズを指定してください。

2. タスク生成用ユーザスタック領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

タスク生成用ユーザスタック領域のサイズを指定してください。
ここで指定されたサイズの領域は、タスク生成時に必要となるタスクのスタック領域(内蔵 RAM)を確保するために必要となります。
cre_tsk、acre_tsk サービスコールで指定されたスタックサイズは、このタスク生成用ユーザスタック領域から指定サイズ分のメモリを確保します。

7.2.3 動的生成用ユーザスタック領域定義(外部 RAM)

<< 形式 >>

```
ext_memstk{
    max_memsize     = 生成するタスクのスタックサイズの最大値;
    all_memsize     = タスク生成用ユーザスタック領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するタスクのスタックサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_tsk、acre_tsk サービスコールの発行により生成するタスクの中でスタックサイズが最も大きい値を指定します。ここでの指定は、スタックに使用する領域を外部 RAM に指定したタスクの中で、最も大きいサイズを指定してください。

2. ユーザスタック領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

タスク生成用ユーザスタック領域のサイズを指定してください。
ここで指定されたサイズの領域は、タスク生成時にタスクのスタック領域(外部 RAM)を確保するために必要となります。
cre_tsk、acre_tsk サービスコールで指定されたスタックサイズは、このタスク生成用ユーザスタック領域から指定サイズ分のメモリを確保します。

7.2.4 動的生成用データキュー領域定義(内蔵 RAM)

<< 形式 >>

```
int_memdtq{
    max_memsize     = 生成するデータキューサイズの最大値;
    all_memsize     = 動的生成用データキュー領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するデータキューサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_dtq、acre_dtq サービスコールの発行により生成するデータキューの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定したデータキューの中で、最も大きいサイズを指定してください。

2. 動的生成用データキュー領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用データキュー領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_dtq,acre_dtq サービスコール発行時にデータキュー領域(内蔵 RAM)を確保するために必要となります。
cre_dtq、acre_dtq サービスコールで指定されたデータキューのサイズは、この動的生成用データキュー領域から指定サイズ分のメモリを確保します。

7.2.5 動的生成用データキュー領域定義(外部 RAM)

<< 形式 >>

```
ext_memdtq{
    max_memsize     = 生成するデータキューサイズの最大値;
    all_memsize     = データキュー領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するデータキューサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_dtq、acre_dtq サービスコールの発行により生成するデータキューの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定したデータキューの中で、最も大きいサイズを指定してください。

2. データキュー領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用データキュー領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_dtq、acre_dtq サービスコール発行時にデータキュー領域(外部 RAM)を確保するために必要となります。
cre_dtq、acre_dtq サービスコールで指定されたデータキューのサイズは、この動的生成用データキュー領域から指定サイズ分のメモリを確保します。

7.2.6 動的生成用メッセージバッファ領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmbf{
    max_memsize     = 生成するメッセージバッファサイズの最大値;
    all_memsize     = 動的生成用メッセージバッファ領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメッセージバッファサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbf、acre_mbf サービスコールの発行により生成するメッセージバッファの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定したメッセージバッファの中で、最も大きいサイズを指定してください。

2. メッセージバッファ領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メッセージバッファ領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbf、acre_mbf サービスコール発行時にメッセージバッファ領域(内蔵 RAM)を確保するために必要となります。
cre_mbf、acre_mbf サービスコールで指定されたメッセージバッファのサイズは、この動的生成用メッセージバッファ領域から指定サイズ分のメモリを確保します。

7.2.7 動的生成用メッセージバッファ領域定義(外部 RAM)

<< 形式 >>

```
ext_memmbf{
    max_memsize     = 生成するメッセージバッファサイズの最大値;
    all_memsize     = 動的生成用メッセージバッファ領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメッセージバッファサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbf、acre_mbf サービスコールの発行により生成するメッセージバッファの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定したメッセージバッファの中で、最も大きいサイズを指定してください。

2. メッセージバッファ領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メッセージバッファ領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbf、acre_mbf サービスコール発行時にメッセージバッファ領域(外部 RAM)を確保するために必要となります。
cre_mbf、acre_mbf サービスコールで指定されたメッセージバッファのサイズは、この動的生成用メッセージバッファ領域から指定サイズ分のメモリを確保します。

7.2.8 動的生成用固定長メモリプール領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmpf{
    max_memsize     = 生成する固定長メモリプール領域の最大値;
    all_memsize     = 動的生成用固定長メモリプール領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する固定長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpf、acre_mpf サービスコールの発行により生成する固定長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定した固定長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用固定長メモリプール領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用固定長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpf、acre_mpf サービスコール発行時に固定長メモリプール領域(内蔵 RAM)を確保するために必要となります。
cre_mpf、acre_mpf サービスコールで指定された固定長メモリプールのサイズは、この動的生成用固定長メモリプール領域から指定サイズ分のメモリを確保します。

7.2.9 動的生成用固定長メモリプール領域定義(外部 RAM)

<< 形式 >>

```
ext_memmpf{
    max_memsize     = 生成する固定長メモリプール領域の最大値;
    all_memsize     = 動的生成用固定長メモリプール領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する固定長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpf、acre_mpf サービスコールの発行により生成する固定長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定した固定長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用固定長メモリプール領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用固定長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpf、acre_mpf サービスコール発行時に固定長メモリプール領域(外部 RAM)を確保するために必要となります。
cre_mpf、acre_mpf サービスコールで指定された固定長メモリプールのサイズは、この動的生成用固定長メモリプール領域から指定サイズ分のメモリを確保します。

7.2.10 動的生成用可変長メモリプール領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmpl{
    max_memsize     = 生成する可変長メモリプール領域の最大値;
    all_memsize     = 動的生成用可変長メモリプール領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する可変長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpl、acre_mpl サービスコールの発行により生成する可変長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定した可変長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用可変長メモリプール領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用可変長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpl、acre_mpl サービスコール発行時に可変長メモリプール領域(内蔵 RAM)を確保するために必要となります。
cre_mpl、acre_mpl サービスコールで指定された可変長メモリプールのサイズは、この動的生成用可変長メモリプール領域から指定サイズ分のメモリを確保します。

7.2.11 動的生成用可変長メモリプール領域定義(外部 RAM)

<< 形式 >>

```
ext_memmpl{
    max_memsize     = 生成する可変長メモリプール領域の最大値;
    all_memsize     = 動的生成用可変長メモリプール領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する可変長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpl、acre_mpl サービスコールの発行により生成する可変長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定した可変長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用可変長メモリプール領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用可変長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpl、acre_mpl サービスコール発行時に可変長メモリプール領域(外部 RAM)を確保するために必要となります。
cre_mpl、acre_mpl サービスコールで指定された可変長メモリプールのサイズは、この動的生成用可変長メモリプール領域から指定サイズ分のメモリを確保します。

7.2.12 システムクロック定義

<< 形式 >>

```
// System Clock Definition
clock{
    timer_clock     = タイマのクロック;
    timer           = タイマモード;
    IPL             = システムクロック割り込み優先レベル;
    file_name       = タイマ設定用テンプレートファイル名;
};
```

<< 内容 >>

1. タイマのクロック

【 定義形式 】周波数（MHz）

【 定義範囲 】なし

タイマに供給されるクロックの周波数を MHz 単位で定義します。

2. タイマモード

【 定義形式 】シンボル

【 定義範囲 】OTHER, NOTIMER, その他シンボル

タイマのモードを OTHER、NOTIMER、その他のシンボルで定義します。
システムクロックを使用しない場合は、”NOTIMER”を定義します。ここで定義されたシンボルは、”mr32r.inc”ファイルにシンボルとして定義され、”timer.inc”のタイマの初期設定部分でマクロ展開される際に使用されます。
ユーザが ipl.ms 相当のファイルを作成した場合には、OTHER を指定します。

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 】0 ～ 6

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。
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. テンプレートファイル名

【 定義形式 】シンボル

【 定義範囲 】なし

使用するテンプレートファイル名を指定します。指定したファイル名のファイルがコンフィグレータ実行ディレクトリに"timer.inc"というファイル名でコピーされます。標準添付のテンプレートファイルとマイコン・周辺機能との対応は、以下のとおりです。また、下表の定義可能なシンボルは、各マイコン・周辺のタイマの名称と同じです。

表 7-3　使用マイコンとテンプレートファイル名との対応

使用マイコンまたは周辺	テンプレートファイル名	定義可能なシンボル
M32182,M32192	M32182.tpl	TOP0,TOP1,TOP2, …TOP10
M32104	M32104.tpl	MFT00,MFT01,MFT02
M32102	m32102.tpl	MFT00,MFT01…MFT13

7.2.13 最大項目数定義

ここでの設定は、複数のアプリケーションの中で、各定義の最大数を定義します[49]。

<< 形式 >>

```
// Max Definition
maxdefine{
    max_task        = 最大タスク定義数;
    max_sem         = 最大セマフォ定義数;
    max_flag        = 最大イベントフラグ定義数;
    max_dtq         = 最大データキュー定義数;
    max_mbx         = 最大メールボックス定義数;
    max_mbf         = 最大メッセージバッファ定義数;
    max_por         = 最大ランデヴポート定義数;
    max_mpf         = 最大固定長メモリプール定義数;
    max_mpl         = 最大可変長メモリプール定義数;
    max_cyh         = 最大周期ハンドラ定義数;
    max_alh         = 最大アラームハンドラ定義数;
    max_int         = 最大割り込みハンドラ定義数;
};
```

<< 内容 >>

1. 最大タスク定義数

   【 定義形式 】数値

   【 定義範囲 】1 ～ 1024

   タスク定義の最大数を定義します。

2. 最大セマフォ定義数

   【 定義形式 】数値

   【 定義範囲 】1 ～ 1024

   セマフォ定義の最大数を定義します。

3. 最大イベントフラグ定義数

   【 定義形式 】数値

   【 定義範囲 】1 ～ 1024

   イベントフラグ定義の最大数を定義します。

[49] タスクやオブジェクトを生成するサービスコール(cre_xxx)をご使用の場合、必ず、生成するタスクやオブジェクトの最大数を定義してください。

4. 最大データキュー定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

データキュー定義の最大数を定義します。

5. 最大メールボックス定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

メールボックス定義の最大数を定義します。

6. 最大メッセージバッファ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

メッセージバッファ定義の最大数を定義します。

7. 最大ランデヴポート定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

ランデヴポート定義の最大数を定義します。

8. 最大固定長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

固定長メモリプール定義の最大数を定義します。

9. 最大可変長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

可変長メモリプール定義の最大数を定義します。

10. 最大周期ハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

周期ハンドラ定義の最大数を定義します。

11. 最大アラームハンドラ定義

【 定義形式 】数値

【 定義範囲 】1 ～ 1024

アラームハンドラ定義の最大数を定義します。

12. 最大割り込みハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

割り込みハンドラ定義の最大数を定義します。コンフィグレータによってここに指定した値の分だけ割り込みベクタテーブルが確保されます。

7.2.14 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;
    name            = タスクのID名称;
    stack_size      = タスクのユーザスタックサイズ;
    stack_area      = スタックの配置;
    stack_section   = スタックのセクション名;
    priority        = タスクの初期優先度;
    initial_start   = 初期起動状態;
    exinf           = タスク拡張情報;
    texaddr         = タスク例外開始アドレス;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。また個々で指定する ID 番号は、最大タスク定義数より小さく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

タスクの入り口アドレスを定義します。

2. タスク ID 名称

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

タスクの ID 名称を定義します。タスク ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

3. ユーザスタックサイズ

【 定義形式 】数値

【 定義範囲 】76 以上

タスクごとのユーザスタックサイズを定義します。ユーザスタックとは、各々のタスクが使用す

るスタック領域です。MR32R ではユーザ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 76 バイトが必要です。

また、本定義によるタスクに対してタスク例外を定義する場合は、さらに 80 バイト分のスタックが必要になります。ここで指定した値は、4 の倍数に丸めて割り当てられます。

4. スタックの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

ユーザスタックの配置を指定します。ユーザスタックを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合(INT_USR_STACK セクション)
__MR_INT を指定してください。
★ 外部 RAM に配置する場合(EXT_USR_STACK セクション)
__MR_EXT を指定してください

本項目を省略した場合は、__MR_INT が設定されます。
なお、5の項目 stack_section の指定を行う場合は、本項目は指定できません。

5. スタックのセクション名

【 定義形式 】シンボル

【 定義範囲 】なし

タスクのスタックを配置するセクション名を指定します。ここで定義したセクションは、必ず、セクションファイルにて配置を行って下さい。

なお、4の項目 stack_area の指定を行う場合、本項目の指定はできません。

6. 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）

タスクの起動時の優先度を定義します。
MR32R 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。

7. 初期起動状態

【 定義形式 】シンボル

【 定義範囲 】ON or OFF

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。

8. タスク拡張情報

【 定義形式 】数値

【 定義範囲 】1～0xFFFFFFFF

タスクの拡張情報を定義します。変数名などの文字列を指定することはできません。ここで定義された値は、タスク起動時、タスク例外起動時に引数として渡されます。

9. タスク例外エントリアドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

タスク例外を定義する場合はそのエントリアドレスをここで定義します。

7.2.15 セマフォ定義

<< 形式 >>

```
// Semaphore Definition
semaphore[ID番号]{
    name            = 名前;
    max_count       = セマフォのカウンタ最大値;
    initial_count   = セマフォのカウンタ初期値;
    wait_queue      = セマフォの待ちタイプ;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。また個々で指定する ID 番号は、最大セマフォ定義数より小さく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

   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。セマフォ ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. **セマフォカウンタ最大値**

   【 定義形式 】数値

   【 定義範囲 】0 ～ 0xFFFF

   セマフォカウンタの最大値を定義します。

3. **セマフォカウンタ初期値**

   【 定義形式 】数値

   【 定義範囲 】0 ～ セマフォカウンタ最大値

   セマフォカウンタの初期値を定義します。

4. セマフォの待ちタイプ

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

セマフォの待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

7.2.16 イベントフラグ定義

<< 形式 >>

```
// Event Flag Definition
flag[ID番号]{
    name            = 名前;
    init_pattern    = イベントフラグの初期ビットパターン;
    wait_queue      = イベントフラグの待ちタイプ;
    wait_multi      = 複数待ちの許可属性;
    clear_attribute = クリア属性;

};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 また個々で指定する ID 番号は、最大イベントフラグ定義数より小さくなければなりません。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

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。イベントフラグ ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. 初期ビットパターン

【 定義形式 】数値

【 定義範囲 】0 ～ 0xFFFFFFFF

イベントフラグビットパターンの初期値を設定します。

3. イベントフラグの待ちタイプ

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

イベントフラグの待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

4. 複数待ちの許可属性

【 定義形式 】シンボル

【 定義範囲 】TA_WMUL or TA_WSGL

複数待ちの許可、不許可に関するイベントフラグ属性を設定します。属性が TA_WMUL(複数タスクの待ちを許可する)場合は"TA_MUL"、属性が TA_WSGL(複数タスクの待ちを許さない)場合は"TA_WSGL"を指定します。

5. クリア属性

【 定義形式 】シンボル

【 定義範囲 】YES or NO

イベントフラグのクリア属性に関する設定を行います。TA_CLR 属性を付加する場合は、"YES"、付加しない場合は"NO"を指定します。

7.2.17 データキュー定義

<< 形式 >>

```
// Mailbox Definition
dataqueue[ID番号]{
    name            = 名前;
    dtq_area        = データキューの配置;
    buffer_size     = データキューの最大データ数;
    wait_queue      = データキューの待ちタイプ;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 また個々で指定する ID 番号は、最大データキュー定義数より小さくなければなりません。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 >>

データキューID 番号ごとに以下の項目の定義をおこないます。

6. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でデータキューを指定する時の名前を定義します。データキューID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

7. データキューの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

データキューの配置を指定します。データキューを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合(MR_HEAP セクション)
__MR_INT を指定してください。

★ 外部 RAM に配置する場合(EXT_MR_HEAP セクション)
__MR_EXT を指定してください。

本項目を省略した場合は、__MR_INT が設定されます。

8. 最大データ数

【 定義形式 】数値

【 定義範囲 】0 以上

データキューに蓄えることのできる最大データ数を定義します。これを越えてデータを蓄えようとするとエラーがかえってきます。

9. データキューの待ちタイプ

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

データキューの待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

7.2.18 メールボックス定義

<< 形式 >>

```
// Mailbox Definition
mailbox[ID番号]{
    name            = 名前;
    wait_queue      = メッセージの待ちタイプ;
    message_queue   = メッセージキューへのつなぎ方;
    maxpri          = メッセージの最大優先度;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 また個々で指定する ID 番号は、最大メールボックス定義数より小さくなければなりません。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 >>

メールボックス 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

プログラム中でメールボックスを指定する時の名前を定義します。メールボックス ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. メッセージを受信する順序

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

メールボックスメッセージの待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

3. メッセージキューへのつなぎ方

【 定義形式 】シンボル

【 定義範囲 】TA_MFIFO or TA_MPRI

メールボックスメッセージキュータイプに関する属性を設定します。属性が TA_MFIFO(メッセージキューは FIFO 順)の場合は"TA_MFIFO"、属性が TA_MPRI(メッセージキューは優先度順)の場合は"TA_MPRI"を指定します。

4. メッセージの最大優先度

【 定義形式 】数値

【 定義範囲 】1～255

メッセージの最大優先度を指定します。システム定義のメッセージ優先度の最大値より大きな値を指定することはできません。MR32R は、この値は関知しません。

7.2.19 メッセージバッファ定義

<< 形式 >>

```
 // MessageBuffer Definition
message_buffer[ID番号]{
    name            = 名前;
    mbf_area        = メッセージバッファの配置;
    buffer_size     = メッセージバッファのサイズ;
    max_msgsz       = 最大メッセージサイズ;
    wait_queue      = メッセージの待ちタイプ;
};
            :
            :
```

ID 番号は 1～32765 の範囲でなければなりません。ID 番号は省略可能です。 また個々で指定する ID 番号は、最大メッセージバッファ定義数より小さくなければなりません。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 >>

メッセージバッファ 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

プログラム中でメッセージバッファを指定する時の名前を定義します。メッセージバッファ ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. メッセージバッファの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

メッセージバッファの配置を指定します。メッセージバッファを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合(MR_HEAP セクション)
__MR_INT を指定してください。

★ 外部 RAM に配置する場合(EXT_MR_HEAP セクション)
__MR_EXT を指定してください。

本項目を省略した場合は、__MR_INT が設定されます。

3. メッセージバッファサイズ

【 定義形式 】数値

【 定義範囲 】0 以上

メッセージバッファのサイズを指定します。
なお、ここで指定した値は、4 の倍数に丸めて割り当てられます。

4. 最大メッセージサイズ

【 定義形式 】数値

【 定義範囲 】0 以上

最大メッセージサイズを指定します。MR32R は、この値は関知しません。

5. メッセージの待ちタイプ

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

データキューの待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

7.2.20 ランデヴ定義

<< 形式 >>

```
// Rendezvous Definition
rendezvous[ID番号]{
    name            = 名前;
    wait_queue      = メッセージの待ちタイプ;
    call_msgsz      = 呼び出しメッセージの最大サイズ;
    rply_msgsz      = 応答メッセージの最大サイズ;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 また個々で指定する ID 番号は、最大ランデヴポート定義数より小さくなければなりません。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 >>

ランデヴ用ポート 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

プログラム中でランデヴ用ポートを指定する時の名前を定義します。ランデヴポート ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. **呼び出し待ちタイプ**

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

ランデヴ呼び出し待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

3. **呼び出しメッセージ最大サイズ**

【 定義形式 】数値

【 定義範囲 】0 以上

呼び出しメッセージの最大サイズを指定します。MR32R は、この値は関知しません。

4. 応答メッセージ最大サイズ

【 定義形式 】数値

【 定義範囲 】0 以上

応答メッセージの最大サイズを指定します。MR32R は、この値は関知しません。

7.2.21 固定長メモリプール定義

<< 形式 >>

```
// Fixed Memory Pool Definition
memorypool[ID番号]{
    name            = 名前;
    mpf_area        = 固定長メモリプールの配置;
    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;
    wait_queue      = メモリ獲得待ちタイプ;
};
            :
            :
```

ID 番号は、1～1024 の範囲でなければなりません。ID 番号は、省略可能です。 また個々で指定する ID 番号は、最大固定長メモリプール定義数より小さくなければなりません。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

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。固定長メモリプール ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. 固定長メモリプールの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

固定長メモリプールの配置を指定します。固定長メモリプールを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合(MR_HEAP セクション)
__MR_INT を指定してください。

★ 外部 RAM に配置する場合(EXT_MR_HEAP セクション)
__MR_EXT を指定してください。

本項目を省略した場合は、__MR_INT が設定されます。
なお、3の項目 section の指定を行う場合、本項目は指定できません。

3. セクション名

【 定義形式 】シンボル

【 定義範囲 】なし

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にて配置を行って下さい。

なお、2の項目 mpf_area の指定を行う場合、本項目は指定できません。

4. ブロック数

【 定義形式 】数値

【 定義範囲 】1～1024

メモリプールのブロック総数を定義します。

5. サイズ

【 定義形式 】数値

【 定義範囲 】1 以上

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))+(((ブロック数-1)/8) +1)バイトです。

6. メモリ獲得待ちタイプ

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO or TA_TPRI

メモリ獲得待ちタイプに関する属性を設定します。属性が TA_TFIFO(待ち行列は FIFO 順)の場合は"TA_TFIFO"、属性が TA_TPRI(待ち行列は優先度順)の場合は"TA_TPRI"を指定します。

7.2.22 可変長メモリプール定義

<< 形式 >>

```
// Variable-Size Memory Pool Definition
 variable_memorypool[ID番号]{
    name            = 名前;
    mpl_area        = 可変長メモリプールの配置;
    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、1～1024 の範囲でなければなりません。ID 番号は、省略可能です。 また個々で指定する ID 番号は、最大可変長メモリプール定義数より小さくなければなりません。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 >>

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

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。可変長メモリプール ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. 確保するメモリブロックサイズの最大値

【 定義形式 】数値

【 定義範囲 】0 以上

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。

7. 可変長メモリプールの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

可変長メモリプールの配置を指定します。可変長メモリプールを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合(MR_HEAP セクション)
__MR_INT を指定してください。

★ 外部 RAM に配置する場合(EXT_MR_HEAP セクション)
__MR_EXT を指定してください。

本項目を省略した場合は、__MR_INT が設定されます。

8. メモリプールのサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

メモリプールのサイズを指定します。ここで、指定された数値は、4 の倍数に丸めて、割り当てられます。

7.2.23 周期ハンドラ定義

<< 形式 >>

```
// Cyclic Handlar Definition
 cyclic_hand[ID番号]{
    name                    = 名前;
    interval_counter        = 周期ハンドラの周期間隔;
    start                   = 周期ハンドラの起動状態;
    phsatr                  = 周期ハンドラの起動位相状態;
    phs_counter             = 周期ハンドラの起動位相;
    entry_address           = 周期ハンドラの開始アドレス;
    exinf                   = 周期ハンドラの拡張情報;
 };
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 ID 番号は、省略可能です。 また個々で指定する ID 番号は、最大周期ハンドラ定義数より小さくなければなりません。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 >>

周期ハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中で周期ハンドラを指定する時の名前を定義します。周期ハンドラ ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

2. 周期間隔

【 定義形式 】数値

【 定義範囲 】1 ～ 0xFFFFFFFF

周期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 ms 単位です。例えば、10 秒間隔で周期起動しようとすると、この値を 10000 に設定します。

3. 起動状態

【 定義形式 】シンボル

【 定義範囲 】ON または OFF

周期ハンドラの動作状態を定義します。TA_STA 属性(周期ハンドラは起動状態)を付加する場合は、"ON"、付加しない場合は、"OFF"を指定します。

4. 起動位相状態

【 定義形式 】シンボル

【 定義範囲 】ON または OFF

周期ハンドラの動作状態を定義します。TA_PHS 属性(起動位相を保存する)を付加する場合は、"ON"、付加しない場合は、"OFF"を指定します。

5. 起動位相

【 定義形式 】数値

【 定義範囲 】0 ～ 0xFFFFFFFF

周期ハンドラの起動位相を定義します。起動位相が 0 の場合は、最初のタイムティックで周期ハンドラが起動します。起動位相に起動周期以上の値が指定された場合であってもコンフィギュレータはエラーを返しません。[50]

6. 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

周期ハンドラの開始アドレスを定義します。

7. 拡張情報

【 定義形式 】数値

【 定義範囲 】0 ～ 0xFFFFFFFF

周期ハンドラの拡張情報を定義します。変数名などの文字列を指定することはできません。ここで定義された値は、周期ハンドラ起動時に引数として渡されます。

[50] GUI コンフィギュレータでは、HI シリーズとの互換性のため、起動位相>=起動周期を指定できません。起動位相>=起動周期となる値を指定する場合は、直接コンフィギュレーションファイルを編集してください。

7.2.24 アラームハンドラ定義

<< 形式 >>

```
// Alarm Handlar Definition
    alarm_hand[ID番号] {
    name            = 名前;
    entry_address   = アラームハンドラの開始アドレス;
    exinf           = アラームハンドラの拡張情報;
};
            :
            :
```

ID 番号は 1～1024 の範囲でなければなりません。ID 番号は省略可能です。 ID 番号は、省略可能です。 また個々で指定する ID 番号は、最大アラームハンドラ定義数より小さくなければなりません。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 >>

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. **名前**

   【 定義形式 】シンボル

   【 定義範囲 】なし

   プログラム中でアラームハンドラを指定する時の名前を定義します。アラームハンドラ ID 番号がここに定義された名称で define されるよう"kernel_id.h"に出力されます。定義される文字列には、"ID_"は付加されず、ここに定義した文字列がそのまま出力されます。

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

   アラームハンドラの開始アドレスを定義します。

3. **拡張情報**

   【 定義形式 】数値

   【 定義範囲 】0 ～ 0xFFFFFFFF

   アラームハンドラの拡張情報を定義します。変数名などの文字列を指定することはできません。ここで定義された値は、アラームハンドラ起動時に引数として渡されます。

7.2.25 使用サービスコール定義

<< 形式 >>

```
// Systemcall Definition
    systemcall{
            サービスコール名          = YES or NO;
};
            :
            :
```

ユーザプログラム中で使用されているサービスコール名を設定します。

<< 内容 >>

3. 使用サービスコール

【 定義形式 】シンボル

【 定義範囲 】YES or NO

以下のサービスコール名を定義できます。

acre_tsk	cre_tsk	del_tsk	act_tsk	iact_tsk	can_act
ican_act	sta_tsk	ista_tsk	ext_tsk	exd_tsk	ter_tsk
chg_pri	ichg_pri	get_pri	iget_pri	ref_tsk	iref_tsk
ref_tst	iref_tst	slp_tsk	tslp_tsk	wup_tsk	iwup_tsk
can_wup	ican_wup	rel_wai	irel_wai	sus_tsk	isus_tsk
rsm_tsk	irsm_tsk	frsm_tsk	ifrsm_tsk	dly_tsk	def_tex
ras_tex	iras_tex	dis_tex	ena_tex	sns_tex	ref_tex
iref_tex	cre_sem	acre_sem	del_sem	sig_sem	isig_sem
wai_sem	twai_sem	pol_sem	ipol_sem	ref_sem	iref_sem
cre_flg	acre_flg	del_flg	set_flg	iset_flg	clr_flg
iclr_flg	wai_flg	twai_flg	pol_flg	ipol_flg	ref_flg
iref_flg	cre_dtq	acre_dtq	del_dtq	snd_dtq	tsnd_dtq
psnd_dtq	ipsnd_dtq	fsnd_dtq	ifsnd_dtq	rcv_dtq	trcv_dtq
prcv_dtq	iprcv_dtq	ref_dtq	iref_dtq	cre_mbx	acre_mbx
del_mbx	snd_mbx	isnd_mbx	rcv_mbx	trcv_mbx	prcv_mbx
iprcv_mbx	ref_mbx	iref_mbx	cre_mbf	acre_mbf	del_mbf
snd_mbf	tsnd_mbf	psnd_mbf	rcv_mbf	trcv_mbf	prcv_mbf
ref_mbf	iref_mbf	cre_por	acre_por	del_por	cal_por
tcal_por	acp_por	tacp_por	pacp_por	fwd_por	rpl_rdv
ref_por	iref_por	ref_rdv	iref_rdv	cre_mpf	acre_mpf
del_mpf	get_mpf	tget_mpf	pget_mpf	rel_mpf	irel_mpf
ref_mpf	iref_mpf	cre_mpl	acre_mpl	del_mpl	get_mpl
tget_mpl	pget_mpl	rel_mpl	ref_mpl	iref_mpl	set_tim
iset_tim	get_tim	iget_tim	cre_cyc	acre_cyc	del_cyc
st_cyc	ista_cyc	stp_cyc	istp_cyc	ref_cyc	iref_cyc
cre_alm	acre_alm	sta_alm	ista_alm	stp_alm	istp_alm
rot_rdq	irot_rdq	get_tid	iget_tid	loc_cpu	iloc_cpu
unl_cpu	iunl_cpu	dis_dsp	ena_dsp	sns_ctx	sns_loc
sns_dsp	sns_dpn	ref_sys	iref_sys	idef_inh	ref_ver
iref_ver	vrst_mpf	vrst_mpl	vrst_mbx	vrst_mbf	vrst_dtq

7.2.26 割り込みベクタ定義

<< 形式 >>

```
// Interrupt Vector Definition
interrupt_vector[割り込み要因番号]     = 割り込みハンドラの開始アドレス;
            :
            :
```

<< 内容 >>

5. 割り込みハンドラの開始アドレス

【 定義形式 】シンボルまたは関数名

【 定義範囲 】なし

割り込みハンドラの開始アドレスを指定します。[]内には、割り込み要因番号を指定してください。コンフィグレータによって割り込みハンドラ開始アドレスが、割り込みベクタテーブルの該当する割り込み要因番号の箇所に出力されます。

システムクロックを使用する場合、システムクロック割り込みハンドラ(__sys_timer)の定義が必要です。

【例】interrupt_vector[10]　　=　　__sys_timer;

7.3 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 //    kernel.cfg : building file for MR32R Ver.4.00
 4 //
 5 //    Generated by M3T-MR32R GUI Configurator at 2004/10/18 15:30:43
 6 //
 7 ////////////////////////////////////////////////////////////////////////////////
 8
 9 // system definition
10 system{
11         stack_size      = 1024;
12         priority        = 1;
13         debug    = NO;
14         debug_buffer    = 4096;
15         message_pri      = 1;
16         tick_deno        = 1;
17         tick_nume        = 1;
18 };
19
20 // dynamic generation definition
21 // for task generation
22 int_memstk{
23         all_memsize     = 0;
24         max_memsize     = 0;
25 };
26 ext_memstk{
27         all_memsize     = 0;
28         max_memsize     = 0;
29 };
30
31 // for dataqueue generation
32 int_memdtq{
33         all_memsize     = 0;
34         max_memsize     = 0;
35 };
36 ext_memdtq{
37         all_memsize     = 0;
38         max_memsize     = 0;
39 };
40
41 // for message buffer generation
42 int_memmbf{
43         all_memsize     = 0;
44         max_memsize     = 0;
45 };
46 ext_memmbf{
47         all_memsize     = 0;
48         max_memsize     = 0;
49 };
50
51 // for fixed-size memorypool generation
52 int_memmpf{
53         all_memsize     = 256;
54         max_memsize     = 256;
55 };
56 ext_memmpf{
57         all_memsize     = 1024;
58         max_memsize     = 256;
59 };
60
61 // for variable-size memorypool generation
62 int_memmpl{
63         all_memsize     = 0;
64         max_memsize     = 0;
65 };
66 ext_memmpl{
67         all_memsize     = 0;
68         max_memsize     = 0;
69 };
70
71
72 // max definition
73 maxdefine{
74         max_task        = 1;
75         max_flag        = 0;
```

```
 76         max_sem = 0;
 77         max_dtq = 0;
 78         max_mbx = 0;
 79         max_mbf = 0;
 80         max_por = 0;
 81         max_mpf = 64;
 82         max_mpl = 0;
 83         max_cyh = 0;
 84         max_alh = 0;
 85         max_int = 65;
 86 };
 87
 88 // system clock definition
 89 clock{
 90         timer_clock     = 27.000000MHz;
 91         timer   = MFT00;
 92         IPL     = 4;
 93         file_name       = m32104.tpl;
 94 };
 95
 96 task[]{
 97         entry_address   = task1();
 98         name    = ID_task1;
 99         stack_size      = 512;
100         stack_area      = INTERNAL;
101         priority        = 1;
102         initial_start   = ON;
103         exinf   = 0x0;
104 };
105
106
107
108
109
110
111
112
113
114
115
116 interrupt_vector[16]    =  __sys_timer;
117
118 // Service Call definition
119 systemcall{
120         cre_tsk =       NO;
121         acre_tsk =      NO;
122         del_tsk =       NO;
123         act_tsk =       YES;
124         iact_tsk =      YES;
125         can_act =       NO;
126         ican_act =      NO;
127         sta_tsk =       YES;
128         ista_tsk =      YES;
129         ext_tsk =       YES;
130         exd_tsk =       NO;
131         ter_tsk =       YES;
132         chg_pri =       YES;
133         ichg_pri =      YES;
134         get_pri =       NO;
135         iget_pri =      NO;
136         ref_tsk =       YES;
137         iref_tsk =      YES;
138         ref_tst =       YES;
139         iref_tst =      YES;
140
141         slp_tsk =       YES;
142         tslp_tsk =      YES;
143         wup_tsk =       YES;
144         iwup_tsk =      YES;
145         can_wup =       NO;
146         ican_wup =      NO;
147         rel_wai =       YES;
148         irel_wai =      YES;
149         sus_tsk =       YES;
150         isus_tsk =      YES;
151         rsm_tsk =       YES;
152         irsm_tsk =      YES;
153         frsm_tsk =      NO;
154         ifrsm_tsk =     NO;
155         dly_tsk =       NO;
```

```
156
157         def_tex =       NO;
158         ena_tex =       NO;
159         dis_tex =       NO;
160         ras_tex =       NO;
161         iras_tex =      NO;
162         sns_tex =       NO;
163         ref_tex =       NO;
164         iref_tex =      NO;
165
166         cre_sem =       NO;
167         acre_sem =      NO;
168         del_sem =       NO;
169         sig_sem =       NO;
170         isig_sem =      NO;
171         wai_sem =       NO;
172         twai_sem =      NO;
173         pol_sem =       NO;
174         ipol_sem =      NO;
175         ref_sem =       NO;
176         iref_sem =      NO;
177
178         cre_flg =       NO;
179         acre_flg =      NO;
180         del_flg =       NO;
181         set_flg =       NO;
182         iset_flg =      NO;
183         clr_flg =       NO;
184         iclr_flg =      NO;
185         wai_flg =       NO;
186         pol_flg =       NO;
187         ipol_flg =      NO;
188         twai_flg =      NO;
189         ref_flg =       NO;
190         iref_flg =      NO;
191
192         cre_dtq =       NO;
193         acre_dtq =      NO;
194         del_dtq =       NO;
195         snd_dtq =       NO;
196         tsnd_dtq =      NO;
197         psnd_dtq =      NO;
198         ipsnd_dtq =     NO;
199         fsnd_dtq =      NO;
200         ifsnd_dtq =     NO;
201         rcv_dtq =       NO;
202         trcv_dtq =      NO;
203         prcv_dtq =      NO;
204         iprcv_dtq =     NO;
205         ref_dtq =       NO;
206         iref_dtq =      NO;
207
208         cre_mbx =       NO;
209         acre_mbx =      NO;
210         del_mbx =       NO;
211         snd_mbx =       NO;
212         isnd_mbx =      NO;
213         rcv_mbx =       NO;
214         trcv_mbx =      NO;
215         prcv_mbx =      NO;
216         iprcv_mbx =     NO;
217         ref_mbx =       NO;
218         iref_mbx =      NO;
219
220         cre_por =       NO;
221         acre_por =      NO;
222         del_por =       NO;
223         cal_por =       NO;
224         tcal_por =      NO;
225         acp_por =       NO;
226         tacp_por =      NO;
227         pacp_por =      NO;
228         fwd_por =       NO;
229         rpl_rdv =       NO;
230         ref_por =       NO;
231         iref_por =      NO;
232         ref_rdv =       NO;
233         iref_rdv =      NO;
234
235         cre_mbf =       NO;
```

```
236         acre_mbf =      NO;
237         del_mbf =       NO;
238         snd_mbf =       NO;
239         tsnd_mbf =      NO;
240         psnd_mbf =      NO;
241         rcv_mbf =       NO;
242         prcv_mbf =      NO;
243         trcv_mbf =      NO;
244         ref_mbf =       NO;
245         iref_mbf =      NO;
246
247         cre_mpf =       YES;
248         acre_mpf =      YES;
249         del_mpf =       YES;
250         get_mpf =       YES;
251         tget_mpf =      YES;
252         pget_mpf =      YES;
253         ipget_mpf =     YES;
254         rel_mpf =       YES;
255         irel_mpf =      YES;
256         ref_mpf =       YES;
257         iref_mpf =      YES;
258
259         cre_mpl =       NO;
260         acre_mpl =      NO;
261         del_mpl =       NO;
262         get_mpl =       NO;
263         tget_mpl =      NO;
264         pget_mpl =      NO;
265         rel_mpl =       NO;
266         ref_mpl =       NO;
267         iref_mpl =      NO;
268
269         set_tim =       NO;
270         iset_tim =      NO;
271         get_tim =       NO;
272         iget_tim =      NO;
273
274         cre_cyc =       NO;
275         acre_cyc =      NO;
276         del_cyc =       NO;
277         sta_cyc =       NO;
278         ista_cyc =      NO;
279         stp_cyc =       NO;
280         istp_cyc =      NO;
281         ref_cyc =       NO;
282         iref_cyc =      NO;
283
284         cre_alm =       NO;
285         acre_alm =      NO;
286         del_alm =       NO;
287         sta_alm =       NO;
288         ista_alm =      NO;
289         stp_alm =       NO;
290         istp_alm =      NO;
291         ref_alm =       NO;
292         iref_alm =      NO;
293
294         rot_rdq =       NO;
295         irot_rdq =      NO;
296         get_tid =       NO;
297         iget_tid =      NO;
298         loc_cpu =       NO;
299         iloc_cpu =      NO;
300         unl_cpu =       NO;
301         iunl_cpu =      NO;
302         dis_dsp =       NO;
303         ena_dsp =       NO;
304         sns_ctx =       NO;
305         sns_loc =       NO;
306         sns_dsp =       NO;
307         sns_dpn =       NO;
308
309         def_inh =       NO;
310         idef_inh =      NO;
311
312         ref_ver =       NO;
313         iref_ver =      NO;
314
315         vrst_dtq =      NO;
```

```
316         vrst_mpf =      NO;
317         vrst_mpl =      NO;
318         vrst_mbf =      NO;
319         vrst_mbx =      NO;
320
321 };
322
323
324 //
325 // End of Configuration
326 //
```

7.4 コンフィグレータの実行

7.4.1 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。

3. 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 "LIB32R"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[51]（makefile.dos, makefile, Makefile）

  makefile[52]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7.4.4項参照）

- mr32r.inc テンプレートファイル（mr32r.inc）

  インクルードファイル mr32r.inc のテンプレートとなるファイルです。
  環境変数 "LIB32R"で示されるディレクトリに存在します。

- タイマ設定用テンプレートファイル(xxx.tpl)

  コンフィグレーションファイルで指定したファイル名のタイマ設定用テンプレートを環境変数"LIB32R"ディレクトリからコンフィグレータを実行したディレクトリに"timer.inc"というファイル名でコピーします。この"timer.inc"は、ipl.ms(または ipl.s)でインクルードされています。

- MR32R バージョンファイル（version）

  MR32R のバージョンを記述したファイルです。環境変数 "LIB32R" 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 MR32R のバージョン情報を出力します。

4. 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、ユーザのデータ定義を行わないで下さい。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。

[51] EWS 版では makefile.ews,DOS 版では makefile.dos を使用します。
[52] 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。

- インクルードファイル（mr32r.inc）

  mr32r.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。

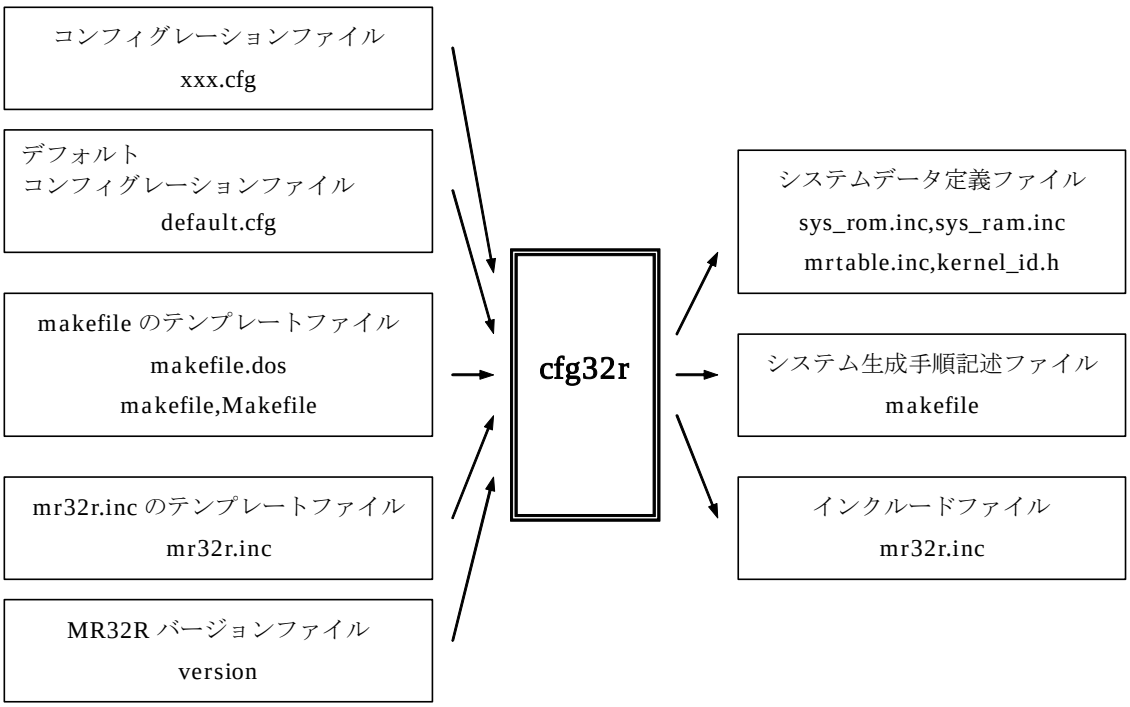


図 7.1　コンフィグレータ動作概要

7.4.2 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"LIB32R"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"LIB32R"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）

- mr32r.inc のテンプレートファイル（mr32r.inc）

- セクション定義ファイル（section）

- makefile のテンプレートファイル（makefile.dos）

- MR32R バージョンファイル（version）

- スタートアップファイル (crt0mr.ms,starut.ms)

- 割込み制御プログラム(ipl.ms)

- タイマ設定用テンプレートファイル(xxx.tpl)

7.4.3 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。

```
cfg32r [-vmVRi] 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。

-i オプション

タイマの自動設定用ファイル(“timer.inc”)を生成します。カレントディレクトリにすでに存在するときは、生成しません。

-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。[53]

また、カレントディレクトリにスタートアップファイルとセクション定義ファイル がない場合は、環境変数 LIB32R 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。

[53] 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

7.4.4 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.s あるいは.ms 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'.s'あるいは'.ms'を使用してください。

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"LIB32R"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。

7.4.5 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.mo)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。
- 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 cfg32r では、UNIX 標準、もしくは準拠の makefile しか生成しません。

7.4.6 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

cfg32r 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。

cfg32r Error : not enough memory
メモリが足りません。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r Error : can't open version file
環境変数"LIB32R"の示すディレクトリの下に MR32R バージョンファイル"version"がありません。

cfg32r 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 "LIB32R"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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r Error : too big task[x]'s priority --> <16> near line 100 (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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r Error : double definition XXXX[x] near line 77 (default.cfg)

cfg32r Error : double definition XXXX[x] (test.cfg) near line 77 (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。

cfg32r Error : you must define XXXX near line 107 (test.cfg)

XXXX は、省略できない項目です。

cfg32r Error : you must define SYMBOL near line 30 (test.cfg)

省略できないシンボルです。

cfg32r 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.s"または "crt0mr.ms"が、カレントディレクトリに必要です。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r Error : “max_memsize” must set XX or less in YYYY definition near line 28 (test.cfg)

YYYY 定義の"max_memsize"の項目に XX バイト以下のサイズをセットしてください。

cfg32r Error : can’t define both “stack_area” keyword and “stack_section” keyword in task definnition near line 28 (test.cfg)

タスク定義において"stack_area"と"stack_section"の両方が定義されています。どちらか片方だけを定義してください。

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。

cfg32r Warning : system is not defined (test.cfg)

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r 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。

cfg32r Warning : XXX is less than real definition number

最大項目数定義の"XXX"で指定した値が実際にコンフィグレーションファイルで定義した値に置き換えます。

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。

cfg32r Error : 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。

cfg32r Warning : test.c(line 12): can't find <XXXX>

cfg32r 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下さい。

cfg32r 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。

第 8 章 各設定ファイルのカスタマイズ方法

8.1 割り込み制御プログラムについて

MR32R では、「図 6.2　　割り込みハンドラの処理手順」で示すように割り込みコントローラの設定やタイマの設定に関係する部分は、OS カーネル内には、含まれていません。
これらの設定に関しては、"ipl.ms"ファイルで行います。MR32R では、「6.6.2タイマの自動設定について」にあるようにいくつかのマイコンや周辺機能に対応したテンプレートがあり、そのテンプレートを使用することができます。
しかし、使用しているマイコンに対応したテンプレートがない場合は、"ipl.ms"ファイルをユーザが作成する必要があります。

8.1.1 割り込み制御プログラムの処理内容

割り込み処理プログラムでは、表 8-1に示すラベルのアセンブラルーチンを記述します。それぞれのアセンブラルーチンでどのような処理を記述する必要があるのか以下に説明します。

表 8-1　割り込み制御プログラムの概要

ラベル	概要	レジスタの保証
__sys_timer_init	タイマの初期化を行います	不要
__SAVE_IPL_to_STACK	割り込みステータスをスタックに退避します。	必要
__RESTORE_IPL_from_STACK	割り込みステータスをスタックに退避します。	必要
__set_ipl	起動時の割り込み優先レベルを設定します。	不要
__Int_Dummy	ダミーの割り込みルーチンです。	必要
__SYS_DUMMY	ダミーのサービスコールルーチンです。	不要

- __sys_timer_init

  このルーチンは、スタートアップルーチン内で呼び出され、システム時刻の設定とシステムクロックの初期化を行います。
  システム時刻の設定部分は、マイコンや周辺機能に依存せず、次節からのサンプルプログラム内のⅠの部分のように記述します。
  タイマの初期化は、使用するタイマに応じて任意の間隔で割り込みが発生するように記述します。
  ユーザがタイマ初期化を記述する際は、コンフィギュレーションファイルのシステムクロック定義の項目"timer"に OTHER を指定します。
  そのときに "timer_clock"，"IPL"，"file_name"に指定されている値は無視されます。

- __SAVE_IPL_to_STACK

  割り込みステータスと割り込みハンドラの入り口アドレスを図 8.1にようにスタックに積んで OS 処理に戻ります。
  このルーチンは、割り込みが発生した時に、図 6.2で示す OS 割り込み出入り口処理から呼び出されます。
  割り込みハンドラの入り口アドレスは、コンフィギュレータによって出力されるアドレステーブルから読み出します。
  アドレステーブルは、INTERRUPT_VECTOR または.INTERRUPT_VECTOR セクション(先頭ラベルが__INT_VECTOR で定義されています。)に割り込み要因０からコンフィギュレーションファイルの最大項目定義の max_int で指定した数の分だけ用意されます。例えば、「max_int = 63;」を指定した場合、割り込み要因番号 0 から割り込み要因番号 62 までの 63 のエントリが用意されます。
  また、割り込みハンドラのアドレスは、コンフィギュレーションファイルの割り込みハンドラ定義で記述した割り込み要因番号のエントリに対応する割り込みハンドラの入り口アドレスが出力さ

れます。例えば、「interrupt_vector[16] = __sys_timer;」と指定した場合、割り込み要因番号 16 のエントリに__sys_timer のラベルが埋め込まれます。
従って、__INT_VECTOR[割り込み要因番号]を読み出すことによって割り込み要因番号に対応する割り込みハンドラの入り口アドレスを取得することができます。

- __RESTORE_IPL_from_STACK
  図 8.1にようにスタックに積まれている割り込みステータスを復帰し、OS 処理に戻ります。
  このルーチンは、割り込みが発生した時に、図 6.2で示す OS 割り込み出入り口処理から呼び出されます。

- __set_ipl
  割り込みマスクレベルを設定します。このルーチンは、スタートアップルーチンから呼び出されます。

- __Int_Dummy[54]
  割り込みハンドラとして登録されていない割り込みが発生した場合、このルーチンが呼び出されます。

- __SYS_DUMMY
  コンフィギュレーションファイルのサービスコール定義で使用しないと定義したサービスコールを使用した場合、このルーチンが呼び出されます。

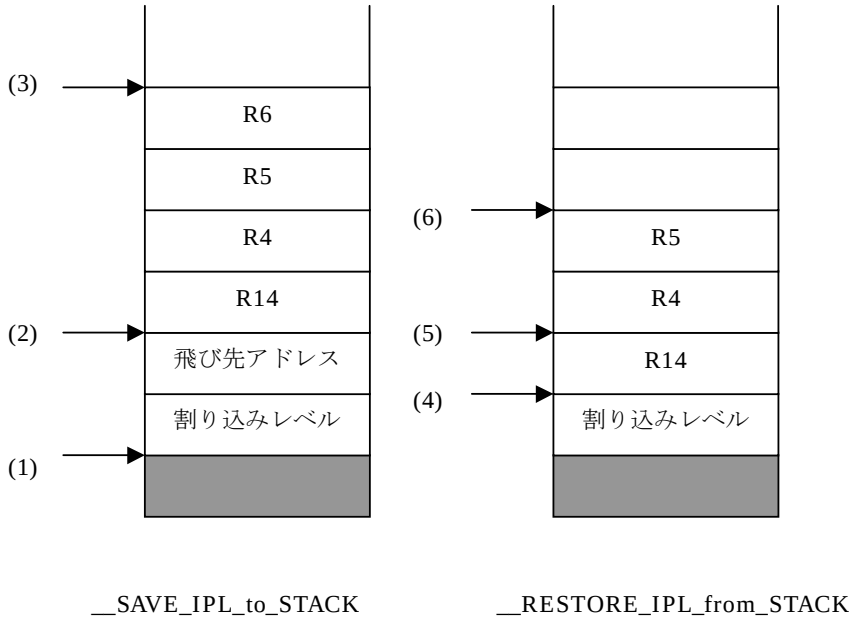


図 8.1　割り込み制御プログラムのスタックの状況

[54] 割り込みハンドラとして登録されていない割り込み要因番号の割り込みは、割り込みハンドラアドレステーブルに__Int_Dummy が埋め込まれます。

8.1.2 割り込み制御プログラム例

```
 1 ;************************************************************************
 2 ;
 3 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 4 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 5 ;
 6 ;
 7 ;   $Id: ipl.s,v 1.2 2001/04/18 06:24:29 inui Exp $
 8 ;
 9 ;************************************************************************
10
11        .include        "mr32r.inc"
12        .section        MR_KERNEL,code
13
14 ICUISTS:        .equ    0x00EFF004
15 ICUIMASK:       .equ    0x00EFF01C
16 MFTCR:          .equ    0x00EFC000
17 MFTMOD:         .equ    0x00EFC100
18 MFTRLD:         .equ    0x00EFC10C
19 ICUCR:          .equ    0x00EFF23C
20 __write_bit:    .equ    0x8000
21 __ena_bit:      .equ    0x0080
22 __timer_div:    .equ    0x3
23 __timer_count:  .equ    0x6596
24
25        .global __sys_timer_init,__stmr_ipl,__stmr_src,__stmr_cnt
26        .global __D_Sys_TIME_H,__Sys_time,__stmr_ctr
27        .global __D_Sys_TIME_L
28
29 ;タイマを使用する場合、タイマの初期化をここで行います。
30 ;このルーチンは、スタートアッププログラム(crt0mr.msまたは、start.ms)
31 ;から呼ばれます。
32        .AIF    &¥USE_TIMER
33 __sys_timer_init:
34
35        st      r14,@-r15
36
37 ;OSのシステムクロックに使用するタイマの初期設定を行います。
38 ;MFT   control register setting
39        seth    r1,#shigh(MFTCR)
40        ld      r0,@(low(MFTCR),r1)
41        ld24    r2,#__write_bit
42        or      r0,r2
43        st      r0,@(low(MFTCR),r1)
44 ;MFT   mode register setting
45        seth    r1,#shigh(MFTMOD)
46        ld24    r0,#0x8000
47        or3     r0,r0,#__timer_div
48        st      r0,@(low(MFTMOD),r1)
49 ;MFT   reload register setting
50        seth    r1,#shigh(MFTRLD)
51        ld24    r0,#__timer_count
52        st      r0,@(low(MFTRLD),r1)
53
54 ;MFT   interrupt control register setting(IEN=1, ILEVEL=3)
55        seth    r1,#shigh(ICUCR)
56        ldi     r0,#0x1100
57        or3     r0,r0,#3
58        st      r0,@(low(ICUCR),r1)
59
60 ;MFT   control register setting
61        seth    r1,#shigh(MFTCR)
62        ld      r0,@(low(MFTCR),r1)
63        ld24    r2,#(__write_bit | __ena_bit)
64        or      r0,r2
65        st      r0,@(low(MFTCR),r1)
66
67        ld      r14,@r15+
68        jmp     r14
69        .aendi
70
71 ;割り込み制御レジスタを退避し、割り込みの飛びさきアドレスをスタック上に
72 ;積み、OSカーネルに戻ります。
73 ;割り込み入り口処理でOSカーネルから呼び出されますのでこのラベル名は
```

```
74 ;変更しないでください。
75 ;
76         .global __SAVE_IPL_to_STACK,__INT_VECTOR
77  __SAVE_IPL_to_STACK:
78
79 ;割り込み制御レジスタをスタックに退避し、割り込み要因番号を読みます。
80         addi    r15,#-8                                  ------------(1)
81         st      r14,@-r15                                ------------(2)
82         st      r4,@-r15
83         st      r5,@-r15
84         st      r6,@-r15
85         seth    r4,#shigh(ICUISTS) )                     ------------(3)
86         ld      r5,@(low(ICUISTS),r4)
87         st      r5,@(D'20,r15)
88         srli    r5,#20
89
90 ;__INT_VECTORからテーブル検索し、割り込み飛び先アドレスをスタックに積みます。
91 ;(このテーブルは、コンフィギュレータによって、sys_rom.incファイルにINTERRUPT_VECTOR
92 ;セクションとして出力され、任意のアドレスに配置することができます。)
93         ld24    r6,#__INT_VECTOR
94         add     r6,r5
95         ld      r6,@r6
96         st      r6,@(16,r15)                             ------------(3)
97         ld      r6,@r15+
98         ld      r5,@r15+
99         ld      r4,@r15+
100        ld      r14,@r15+
101        jmp     r14                                      ------------(2)
102
103 ;割り込み制御レジスタを復帰し、OSカーネルに戻ります。
104 ;割り込み出口処理でOSカーネルから呼び出されますのでこのラベル名は
105 ;変更しないでください。
106        .global __RESTORE_IPL_from_STACK
107 __RESTORE_IPL_from_STACK:                               ------------(5)
108        st      r14,@-r15
109        st      r4,@-r15
110        st      r5,@-r15
111        ld      r5,@(12,r15)                             ------------(6)
112        seth    r4,#shigh(ICUIMASK)
113        st      r5,@(low(ICUIMASK),r4)
114        ld      r5,@r15+
115        ld      r4,@r15+
116        ld      r14,@r15+                                ------------(5)
117        addi    r15,#4
118        jmp     r14                                      ------------(4)
119
120 ;割り込み優先レベルの初期設定を行います。
121 ;スタートアップファイルから呼び出されます。
122        .global __set_ipl
123 __set_ipl:
124        ldi     r5,#0x07
125        seth    r4,#shigh(ICUIMASK)
126        sth     r5,@(low(ICUIMASK),r4)
127        jmp     r14
128
129 ;ダミーの割り込みルーチンです。
130 ;コンフィギュレーションファイルに割り込みハンドラ定義されていない
131 ;割り込みが入った場合、このルーチンが呼び出されます。
132
133        .global __Int_Dummy
134 __Int_Dummy:
135        jmp     r14
136
137 ;コンフィギュレーションファイルのサービスコール定義で"NO"を
138 ;指定したにもかかわらず、ユーザプログラム中でそのサービスコールを
139 ;使用した場合にこのルーチンが呼び出されます。
140        .global __SYS_DUMMY
141 __SYS_DUMMY
142        jmp     r14
143
```

8.2 MR32R スタートアッププログラムのカスタマイズ方法

MR32R 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.ms

  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.ms

  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.ms”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行っています。

- スタックポインタの設定
- リセット後のプロセッサの初期化
- EIT ベクタエントリ、プログラム領域などの外部 ROM から内蔵 DRAM への転送
- C 言語の変数の初期化(crt0mr.ms のみ)
- システムクロックの設定
- MR32R 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、環境変数 ”LIB32R”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して下さい[55]。なお、必要があればスタートアップファイルを修正、あるいは追加して下さい。

[55] カレントディレクトリにスタートアッププログラムがない場合、コンフィグレータ cfg32r を”-m”オプション付きで実行すると、環境変数``LIB32R''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。

8.2.1 ROM から RAM へのデータ転送について

MR32R では、"mr32r.inc"ファイルで定義されている"DOWNLOAD"マクロを利用し、ROM から RAM への転送処理を行うことができます。製品添付のスタートアップルーチンでは、初期値ありのデータ領域と MR_ROM(または.MR_ROM)セクションの転送処理を行っています。

ダウンロード用のマクロを使用した場合のセクションファイル(section)の変更方法・記述方法については次節で説明します。

```
DOWNLOAD    reg1,reg2,reg3,reg4,section,ROM_section
```

reg1,reg2,reg3 には、マクロ内で使用するレジスタを指定します。
section には、ダウンロードするセクション名を指定し,ROM_section には、前述の section に指定したセクション名の頭に ROM_をつけたものを指定します。
以下に R0,R1,R2,R3 レジスタを使用し、DATA セクションを転送する場合のマクロ記述を示します[56]。

```
DOWNLOAD    R0,R1,R2,R3,DATA,ROM_DATA
```

[56] この場合、ROM_DATA セクションは、リンカによって自動的に生成されるセクションで、スタートアップファイルでセクションの定義を行う必要があります。

8.2.2 ROM から RAM への転送を止める方法について

ROM から RAM に転送する必要のないセクションは、転送処理を解除します。転送する必要のあるセクション、転送する必要のないセクションは下記のとおりです。

転送する必要のあるセクション

- D　　初期値ありのデータ領域

転送する必要のないセクション

- P　　ユーザプログラム領域
- C　　定数データ領域
- MR_KERNEL　　OS カーネル領域

転送が必要かどうかは条件による場合

- MR_ROM　　OS データ領域
  タスクや OS 資源の動的生成削除を使用する場合は、ROM から RAM への転送処理が必要になります。
- INTERRUPT_VECTOR　　割り込みベクタテーブル領域
  アドレスは位置が固定され、そのアドレスが ROM 領域に割り込みベクタテーブルを配置しなければならない場合、転送処理を削除しなければいけません。
- EIT_Vector　　EIT ベクタエントリ領域
  EIT ベクタ領域が ROM の場合、転送処理を削除しなければいけません。
- Int_Vector　　割り込みベクタエントリ領域
  Int_Vector ベクタ領域が ROM の場合、転送処理を削除しなければいけません。

転送処理を解除するには以下の手順で行います。

1. スタートアッププログラムの不要となったセクションを削除します。(削除するセクションは、ROM_P,ROM_C など転送しないセクション名の頭に ROM_がついたセクションです。)
2. スタートアップファイルに記述されている転送しないセクションの部分の DOWNLOAD マクロを削除します。
3. セクションファイル(section)で転送用の記述をしてある部分を変更します。(変更方法については、CC32R のマニュアルを参照してください。

8.2.3 C 言語用スタートアッププログラム(crt0mr.ms)

```
 1 ************************************************************************
 2 ;
 3 ;   MR32R start up program for C language (for CC32R)
 4 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 ;
 7 ;
 8 ; $Id: crt0mr.ms,v 1.6 2003/05/29 01:15:54 inui Exp $
 9 ;
10 ;************************************************************************
11 ;OSを組み込むのに必要なファイルをインクルードします。
12 ;この部分は変更しないでください。
13
14         .include        "mr32r.inc"
15         .include        "sys_rom.inc"
16         .include        "sys_ram.inc"
17         .include        "mrtable.inc"
18
19         .global         __Sys_Sp,__sys_timer,__int_entry,__START
20         .global         __D_INIT_START,__sys_timer_init
21         .global         __SYSCALL0
22         .global         __set_ipl
23         .section        B,data
24         .section        D,data
25         .section        P,code
26         .section        C,data
27         .section        ROM_P,code
28         .section        ROM_C,data
29         .section        ROM_D,data
30         .section        ROM_INTERRUPT_VECTOR,data
31         .section        INTERRUPT_VECTOR,data
32         .section        MR_HEAP,data,align=4
33         .section        EXT_MR_HEAP,data,align=4
34         .section        INT_USR_STACK,data,align=4
35         .section        EXT_USR_STACK,data,align=4
36         .section        EXT_MR_RAM,data,align=4
37         .section        MR_KERNEL2,code,align=4
38         .section        MR_KERNEL,code,align=4
39         .section        ROM_MR_KERNEL2,code,align=4
40         .section        ROM_MR_KERNEL,code,align=4
41         .section        ROM_MR_ROM,data,align=4
42         .section        ROM_MR_EIT,code,align=4
43         .section        MR_ROM,data,align=4
44         .section        OS_DEBUG,code,align=4
45         .section        ROM_OS_DEBUG,code,align=4
46         .section        START_UP,code,align=4
47
48 ;スタートアップファイルのセクション定義します。
49
50         .section        START_UP,code,align=4
51 NULL:   .EQU    0
52
53 ;ベースレジスタ機能を使用する場合、各シンボルに値を設定してください。
54 ;ベースレジスタ機能を使用しない場合、使用しないレジスタについても
55 ;ダミーで値を設定する必要があります。
56
57         .global __REL_BASE11
58         .global __REL_BASE12
59         .global __REL_BASE13
60         .global __REL_BASE
61 __REL_BASE11:   .equ    0
62 __REL_BASE12:   .equ    0
63 __REL_BASE:     .equ    0
64 __REL_BASE13:   .equ    __REL_BASE
65
66 ;ここからスタートアッププログラムを開始します。
67
68 __START:
69
70 ;システムスタックポインタの設定を行います。
71
72         seth    r1,#high(__Sys_Sp)
73         or3     r1,r1,#low(__Sys_Sp)
```

```
 74        addi    r1,#-4
 75        mvtc    r1,SPI                ;SPI initialize
 76        mvtc    r1,SPU
 77        ldi     r0,#-1
 78        st      r0,@r1
 79        ldi     r0,#NULL
 80        mvtc    r0,PSW                ;PSW initialize
 81
 82 ;ベースレジスタ機能を使用し、スタートアップルーチンからC言語で記述された関数を呼び
 83 ;出す場合、R11～R13レジスタにベースレジスタの設定を行います。
 84
 85 ;       seth    R11,#high(__REL_BASE11)
 86 ;       or3     R11,R11,#low(__REL_BASE11)
 87 ;       seth    R12,#high(__REL_BASE12)
 88 ;       or3     R12,R12,#low(__REL_BASE12)
 89 ;       seth    R13,#high(__REL_BASE13)
 90 ;       or3     R13,R13,#low(__REL_BASE13)
 91
 92        .AIF    ¥&__Dbg_flg gt 0
 93        seth    R1,#high(__Dbg_mode)
 94        or3     R1,R1,#low(__Dbg_mode)
 95 ;       ldi     r2,#0
 96        ldi     r2,#4
 97        stb     r2,@r1
 98        .AENDI
 99
100 ;マイコンの動作モードの設定などを行います。
101 ;
102 ;       Description below,if you need
103 ;       (ex. Master/Slavemode initialize,Power management initialize)
104 ;
105
106 ;初期値なしのデータをゼロクリアします。
107 ;このマクロは、"mr32r.inc"で定義されており、ユーザも自由に使用することができます。
108
109       RAM_CLEAR r0,r1,r2,B
110
111 ;外部ROM領域から内蔵RAM領域へ転送します。
112 ;このマクロもユーザが自由に使用することができます。
113
114       DOWNLOAD r0,r1,r2,r3,D,ROM_D
115
116 ;C言語の標準関数の初期化を行います。
117 ;
118
119 ;Initialize standard library
120 ;
121
122 ;       seth    R0,#high($_init_stdio)
123 ;       or3     R0,R0,#low($_init_stdio)
124 ;       jl      R0
125 ;       seth    R0,#high($_init_mem)
126 ;       or3     R0,R0,#low($_init_mem)
127 ;       jl      R0;     .global __init_mem,__init_stdio
128
129       .global __OS_INIT
130
131 ;OSの初期化を行います。
132 __OS_INIT:
133
134       RAM_CLEAR r0,r1,r2,MR_RAM
135       DOWNLOAD r0,r1,r2,r3,MR_ROM,ROM_MR_ROM
136       DOWNLOAD r0,r1,r2,r3,INTERRUPT_VECTOR,ROM_INTERRUPT_VECTOR
137 ;
138 ;Initialize OS system area
139 ;
140       .global __MR_INIT
141       seth R0,#high(__MR_INIT)
142       or3     R0,R0,#low(__MR_INIT)
143       jl      R0
144
145 ;OSデバッグ機能使用時の初期化をします。;
146
147       .AIF    ¥&__Dbg_flg gt 0
148       seth    R1,#high(__Dbg_buffer_start)
149       or3     R1,R1,#low(__Dbg_buffer_start)
150       seth    R0,#high(__Dbg_addr)
```

```
151         or3     R0,R0,#low(__Dbg_addr)
152
153         st      r1,@r0
154         seth    R1,#high(__Dbg_cnt)
155         or3     R1,R1,#low(__Dbg_cnt)
156         ldi     r0,#0
157         st      r0,@r1
158         seth    R1,#high(__Dbg_mode)
159         or3     R1,R1,#low(__Dbg_mode)
160
161         ldb     r0,@r1
162         ldi     r2,#1
163         or      r0,r2
164         stb     r0,@r1
165         .aendi
166
167 ;'ipl.ms'内の__set_iplを呼び出し、割り込み優先レベルの初期設定を
168 ;行います。
169
170         seth    R0,#high(__set_ipl)
171         or3     R0,R0,#low(__set_ipl)
172         jl      R0
173
174 ;タイマを使用する場合、'ipl.ms'内の__sys_timer_initを呼び出し、
175 ;タイマの初期化を行います。
176
177         .AIF    ¥&USE_TIMER gt 0
178         seth R0,#high(__sys_timer_init)
179         or3     R0,R0,#low(__sys_timer_init)
180         jl      R0
181         .AENDI
182
183
184 ;
185 ;Start task
186 ;
187
188 ;OSカーネルを起動します。
189 ;
190 ;Start task
191 ;
192         .global __MR_START
193         seth R0,#high(__MR_START)
194         or3     R0,R0,#low(__MR_START)
195         jl      R0
196
197 ;この部分は変更しないでください。
198
199
200         .AIF    ¥&__Dbg_flg eq 0
201         .section        MR_KERNEL,code,align=4
202         .align  4
203         .global __Dbg_idle
204         .global __Dbg_RUNtsk,__Dbg_int_entry,__Dbg_int_exit
205         .global __Dbg_ent_handler,__Dbg_ext_handler,__Dbg_int_exit2
206         .global __Dbg_sys_exit,__Dbg_sys_exit2,__Dbg_sys_timer
207 __Dbg_idle:
208 __Dbg_RUNtsk:
209 __Dbg_int_entry:
210 __Dbg_int_exit:
211 __Dbg_int_exit2:
212 __Dbg_ent_handler:
213 __Dbg_ext_handler:
214 __Dbg_sys_exit:
215 __Dbg_sys_exit2:
216 __Dbg_sys_timer:
217         .AENDI
218
219
220         .AIF    ¥&__Dbg_flg gt 0
221         .section        MR_Dbg_RAM,data
222         .align  4
223         .global __Dbg_mode
224         .global __Dbg_buffer_start
225         .global __Dbg_buffer_end
226         .global __Dbg_cnt
227         .global __Dbg_addr
228         .global __Dbg_sys_iss
229 __Dbg_buffer_start:
```

```
230         .res.b  __Dbg_buffer_size
231 __Dbg_buffer_end:
232 __Dbg_addr:     .res.w  1
233 __Dbg_cnt:      .res.w  1
234 __Dbg_sys_iss: .res.w  7
235 __Dbg_mode:     .res.b  1
236         .aelse
237         .section        MR_Dbg_RAM,data
238         .align  4
239         .global __Dbg_mode
240         .global __Dbg_buffer_start
241         .global __Dbg_buffer_end
242         .global __Dbg_cnt
243         .global __Dbg_addr
244         .global __Dbg_sys_iss
245 __Dbg_sys_iss:
246 __Dbg_buffer_start:
247 __Dbg_buffer_end:
248 __Dbg_addr:
249 __Dbg_cnt:
250 __Dbg_mode:
251         .aendi
252 ;***************  EIT Vector AREA  ********************
253         .section        EIT_Vector,CODE,ALIGN=4
254         st      R0,@-R15                ;SBI    H'20
255         nop
256         seth    R0,#high(SBI_hdr)
257         or3     R0,R0,#low(SBI_hdr)
258         jmp     R0
259         nop
260
261         st      R0,@-R15                ;RIE    H'30
262         nop
263         seth    R0,#high(RIE_hdr)
264         or3     R0,R0,#low(RIE_hdr)
265         jmp     R0
266         nop
267
268         st      R0,@-R15                ;AE     H'40
269         nop
270         seth    R0,#high(AE_hdr)
271         or3     R0,R0,#low(AE_hdr)
272         jmp     R0
273         nop
274
275         rte                             ;TRAP0
276         nop
277         rte                             ;TRAP1
278         nop
279         rte                             ;TRAP2
280         nop
281         rte                             ;TRAP3
282         nop
283         rte                             ;TRAP4
284         nop
285         rte                             ;TRAP5
286         nop
287         rte                             ;TRAP6
288         nop
289         .global __Dbg_entry0
290         .AIF    ¥&__Dbg_flg eq 0
291         bra     __SYSCALL0              ;TRAP7
292         .AELSE
293         bra     __Dbg_entry0    ;TRAP7
294         .AENDI
295         rte                             ;TRAP8
296         nop
297         rte                             ;TRAP9
298         nop
299         rte                             ;TRAP10
300         nop
301         rte                             ;TRAP11
302         nop
303         rte                             ;TRAP12
304         nop
305         rte                             ;TRAP13
306         nop
307         rte                             ;TRAP14
308         nop
309         rte                             ;TRAP15
```

```
310         nop
311
312         .section        Int_Vector,code,align=4
313         bra     __int_entry
314         .AIF    ¥&__Dbg_flg eq 0
315         bra     __int_entry             ;TRAP7
316         .AELSE
317         .global __Dbg_int_entry
318         bra     __Dbg_int_entry
319         .AENDI
320
321         .SECTION        RESET_VECT,code,align=4
322         seth    R0,#high(__START)
323         or3     R0,R0,#low(__START)
324         jmp     R0
325
326         .section        MR_KERNEL,code,align=4
327 SBI_hdr:
328         ld      R0,@R15+
329         bra     SBI_hdr
330 RIE_hdr:
331         ld      R0,@R15+
332         bra     RIE_hdr
333 AE_hdr:
334         ld      R0,@R15+
335         bra     AE_hdr
336
```

図 8.2　C 言語用スタートアッププログラム

8.3 セクションファイルのカスタマイズ

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、MR32R が提供しているセクションファイル(section)で設定します。
このファイルは、リンカのセクション配置オプションとしてリンカに渡されます。
ユーザの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下さい。[57]
通常は、-SEC の後にセクション名を記述していきます。外部 ROM から内蔵 DRAM に転送する場合は、転送先セクション名の頭に@をつけ、転送元のセクション名はそのまま記述します。
次に例を示します。なお、図中の矢印は、矢印方向へ(ROM から RAM へ)転送することを表します。

```
-SEC
RESET_VECTOR,EIT_Vector=10,Int_Vector=80,MR_KERNEL=100,MR_KERNEL2,START_UP,OS_DEBUG,P,D,C,MR_ROM,INTERRUPT_VECTOR,MR_RAM=0F00000,@MR_ROM,@INTERRUPT_VECTOR,MR_Dbg_RAM,SYS_STACK,INT_USR_STACK,MR_HEAP,B,D,EXT_MR_RAM=1000000,EXT_USR_STACK,EXT_MR_HEAP
```

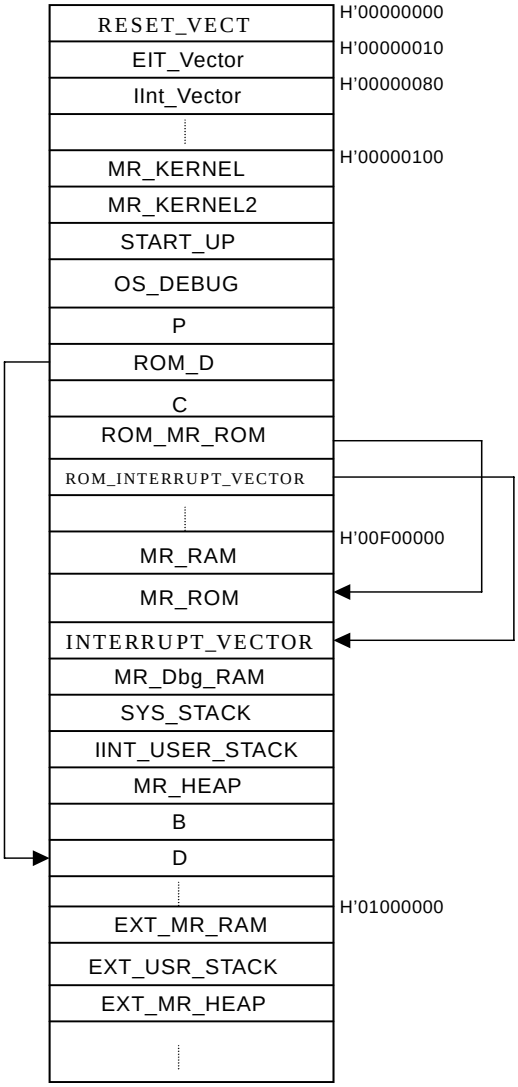


図 8.3　サンプルセクションファイルのメモリ配置

[57] 詳細は、CC32R ユーザーズマニュアル セクション結合機能をご覧ください。

8.4 makefile の編集

コンフィギュレータが生成した、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. cc32R 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下さい。

4. as32R 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。

5. lnk32R 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。

6. ライブラリの指定

ライブラリの指定は、まず、ライブラリのパス名を'echo "-L ライブラリのパス" >> lnk32R.sub'と記述します。

次に指定するライブラリを'echo "-l ライブラリのパス" >> lnk32R.sub'と記述し、ライブラリを指定します。

以下に、smp32r ディレクトリにある smp.lib を指定して例を示します。

```
lnk32r.sub:makefile
        echo -o $(PROGRAM) > lnk32R.sub
        echo -L $(LIB32R) >> lnk32R.sub
        echo -l c32Rmr.lib >> lnk32R.sub
        echo -l mr32R.lib >> lnk32R.sub
  # この後にライブラリの指定を行います。
  # ライブラリのパスを指定します。
        echo -L c:¥mtool¥mr32r¥smp32r >> lnk32R.sub
        echo -l smp.lib >> lnk32R.sub
```

第 9 章 アプリケーション作成の手引き

9.1 ハンドラからのサービスコールの処理手順

ハンドラからのサービスコール発行はタスクからのサービスコールと異なり、サービス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。

ハンドラからのサービス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サービスコール

7. サービスコール処理中に割り込んだハンドラからのサービスコール

8. ハンドラ実行中に割り込んだ（多重割り込み）ハンドラからのサービスコール

9.1.1 タスク実行中に割り込んだハンドラからのサービスコール

スケジューリング（タスク切り替え)は ret_int サービスコールによりおこなわれます。(図 9.1参照)

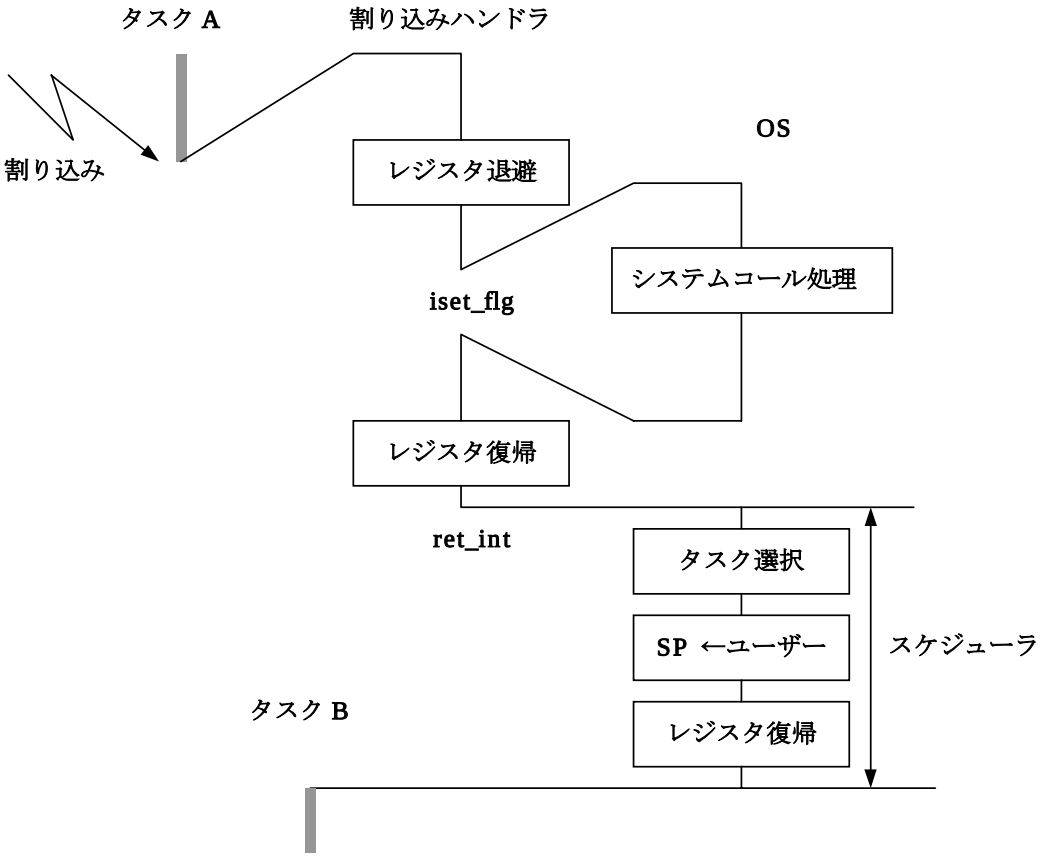


図 9.1　タスク実行中に割り込んだ割り込みハンドラからのサービスコール処理手順

9.1.2 サービスコール処理中に割り込んだハンドラからのサービスコール

スケジューリング（タスク切り替え）は割り込まれたサービスコール処理に戻った後におこなわれます。（図 9.2参照）

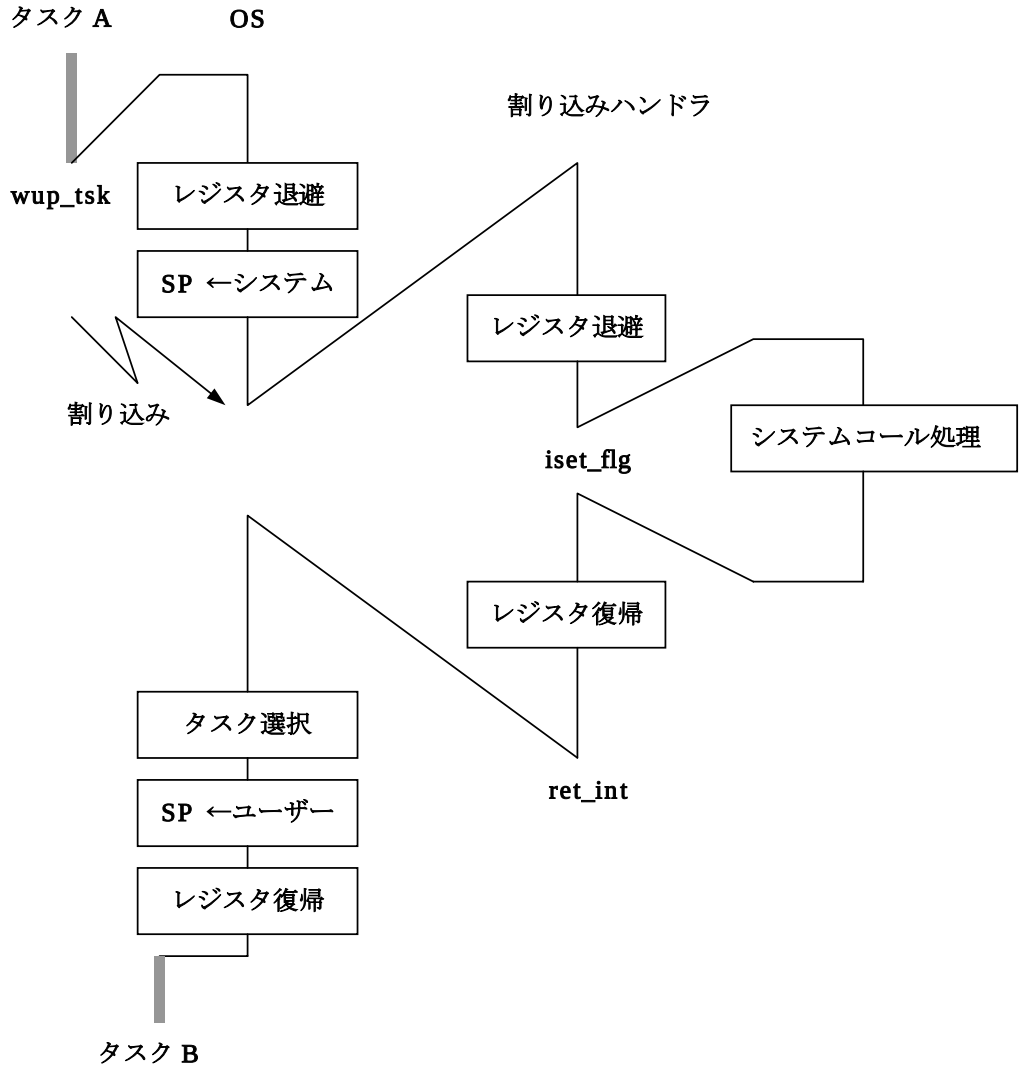


図 9.2　サービスコール処理中に割り込んだ割り込みハンドラからのサービスコール処理手順

9.1.3 ハンドラ実行中に割り込んだハンドラからのサービスコール

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、発行したサービス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 から復帰するサービスコール（ret_int サービスコール）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 サービスコールによりタスク切り替えが行われます。（図 9.3参照）

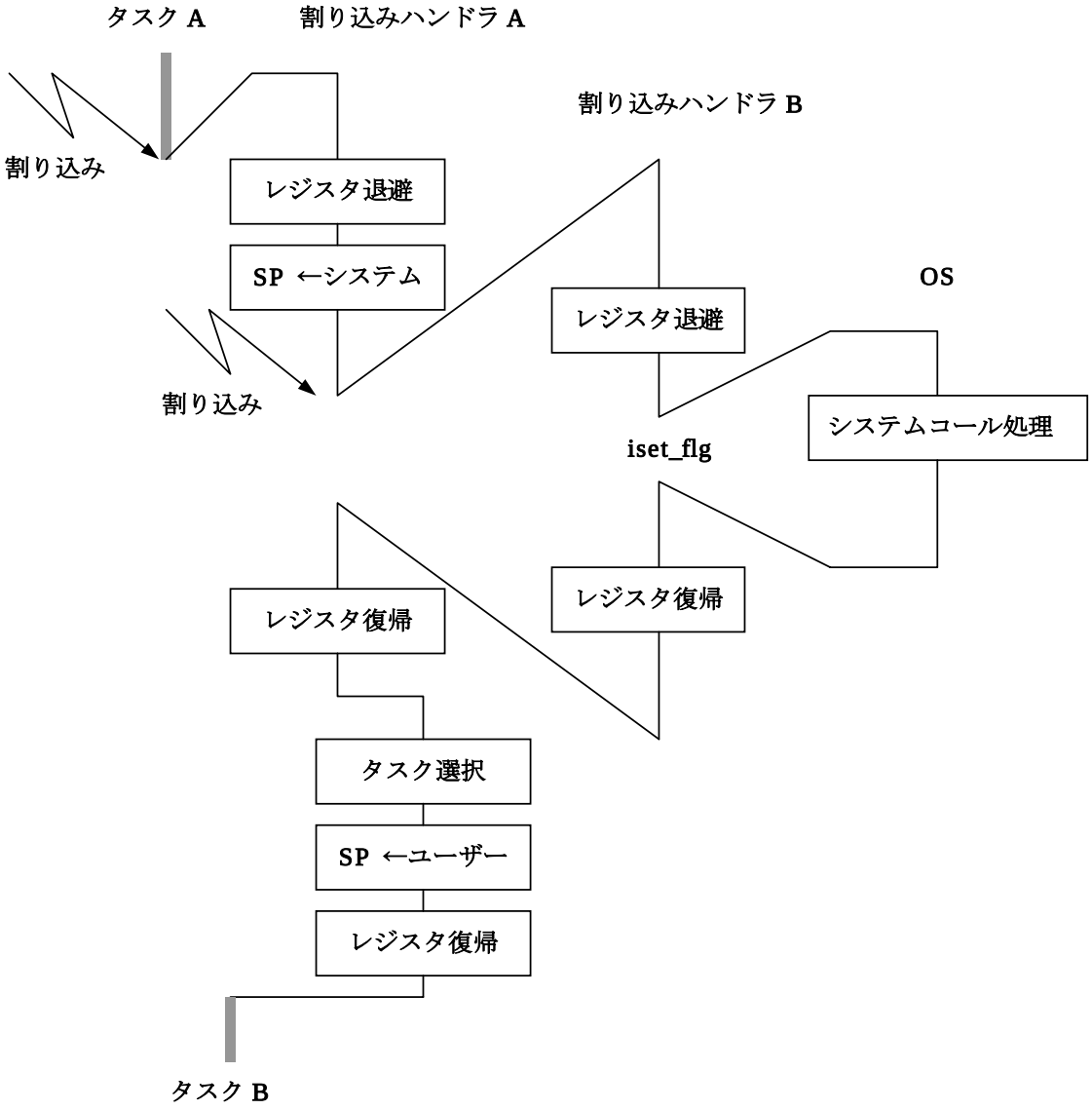


図 9.3　多重割り込みハンドラからのサービスコール処理手順

9.2 システムの使用する RAM 容量の計算方法

MR32R カーネルがタスクなどを管理するためのデータは、MR_RAM セクションに配置されます。オブジェクトの動的生成機能を使用する場合、MR_ROM セクションを RAM に転送して使用する必要がありますので MR_ROM セクションと MR_RAM セクションの使用サイズの合計が RAM 使用量になります。MR_RAM セクションにおいて MR32R が使用する RAM 容量は次の表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。

また、従来、MR_ROM セクションに配置されていた割り込みハンドラアドレステーブルは、INTERRUPT_VECTOR セクションに配置されるようになりましたのでご注意ください。このセクションは、def_int を使用する場合は、RAM 領域に転送する必要があります。

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下さい。

表 9-1　MR_RAM セクションのサイズ算出方法

領域名	バイト数(各領域で 4 バイトのアライメント調整が必要です)
システム管理領域	24+4×(優先度数+最大タスク数[58]×2+最大フラグ数[58]+最大セマフォ数[58]+最大メールボックス数[58]+最大データキュー数[58]×2+最大固定長メモリプール数[58]+最大可変長メモリプール数[58]+最大メッセージバッファ数[58]×2+最大ランデヴポート数[58]×2+2)
タスク管理領域	42×最大タスク数[58]
イベントフラグ管理領域	7×最大イベントフラグ数[58]
セマフォ管理領域	3×最大セマフォ数[58]
データキュー管理領域	13×最大データキュー数[58]
メッセージバッファ管理領域	17×最大メッセージバッファ数[58]
メッセージバッファ領域	メッセージバッファのバッファサイズの合計
ランデヴ用ポート管理領域	3×最大ランデヴ用ポート数[58]
メールボックス管理領域	9×最大メールボックス数[58]
固定長メモリプール管理領域	7×最大固定長メモリプール数[58]
可変長メモリプール管理領域	86×最大可変長メモリプール数[58]
周期起動ハンドラ管理領域	9×最大周期起動ハンドラ数[58]
アラームハンドラ管理領域	7×最大アラームハンドラ数[58]
タスク管理領域(動的生成時に)[59]	68(内蔵 RAM 使用時) 68(外部 RAM 使用時)
データキュー管理領域(動的生成時)[59]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)
メッセージバッファ管理領域(動的生成時)[59]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)
固定長メモリプール管理領域(動的生成時)[59]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)
可変長メモリプール管理領域(動的生成時)[59]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)

(注)　システム管理領域は、必ず確保されます。また、その他の領域については、最大項目数定義の該当オブジェクトの最大値を 0 に指定した場合・記述を省略し、デフォルト値を使用する場合は、確保されません。

たとえば、max_sem = 0;と定義し、セマフォを定義していない場合は、セマフォ管理領域はメモリを消費しません。

[58] コンフィグレーションファイルの最大項目数定義で指定した値です。この項目で値を指定していない場合は、コンフィグレーションファイルで静的に定義した値を指します。

[59] 動的生成機能を使用した場合(cre_tsk, del_tsk, cre_dtq など)に確保されます。

表 9-2　MR_ROM セクションのサイズ算出方法

領域名	バイト数(各領域で 4 バイトのアライメント調整が必要です)
システム管理領域	(初期起動タスク数+1)×2
タスク管理領域	24×最大タスク数[58]
イベントフラグ管理領域	5×最大イベントフラグ数[58] +(最大イベントフラグ数[58]-1)/8+8
セマフォ管理領域	5×最大セマフォ数[58]+(最大セマフォ数[58]-1)/8+8
データキュー管理領域	9×最大データキュー数[58]+(最大データキュー数[58]-1)/8+8
メッセージバッファ管理領域	9×最大メッセージバッファ数[58]+(最大メッセージバッファ数[58]-1)/8+8
ランデヴ用ポート管理領域	(最大ランデヴ用ポート数[58]-1)/8+8
メールボックス管理領域	1×最大メールボックス数[58]+(最大メールボックス数[58]-1)/8+8
固定長メモリプール管理領域	15×最大固定長メモリプール数[58]+(最大固定長メモリプール数[58]-1)/8+8
可変長メモリプール管理領域	17×最大可変長メモリプール数[58]
周期起動ハンドラ管理領域	17×最大周期起動ハンドラ数[58]
アラームハンドラ管理領域	8×最大アラームハンドラ数[58]
システム構成管理領域	20(ref_ver を使用する場合に確保されます。)
サービスコールテーブル	444

(注)　システム管理領域・システム時刻管理領域は、必ず確保されます。また、その他の領域については、最大項目数定義の該当オブジェクトの最大値を 0 に指定した場合・記述を省略し、デフォルト値を使用する場合は、確保されません。

　たとえば、max_sem = 0;と定義し、セマフォを定義していない場合は、セマフォ管理領域はメモリを消費しません。

表 9.3　INTERRUPT_VECTOR セクションのサイズ算出方法

領域名	バイト数
割り込みハンドラアドレステーブル	(最大割り込みハンドラ定義数[60])×4

[60] 最大項目数定義で指定した、最大割り込み要因数です。

9.3 スタックについて

9.3.1 システムスタックとユーザスタック

MR32R のスタックにはシステムスタックとユーザスタックがあります。

- ユーザスタック

  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 MR32R 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

  MR32R 内部（サービスコール処理中）に使用されるスタックです。MR32R ではサービスコールをタスクが発行するとスタックをユーザスタックからシステムスタックに切り替えます。（図 9.4　システムスタックとユーザスタックを参照して下さい。）
  システムスタックは、割り込みスタック(SPI)を使用します。

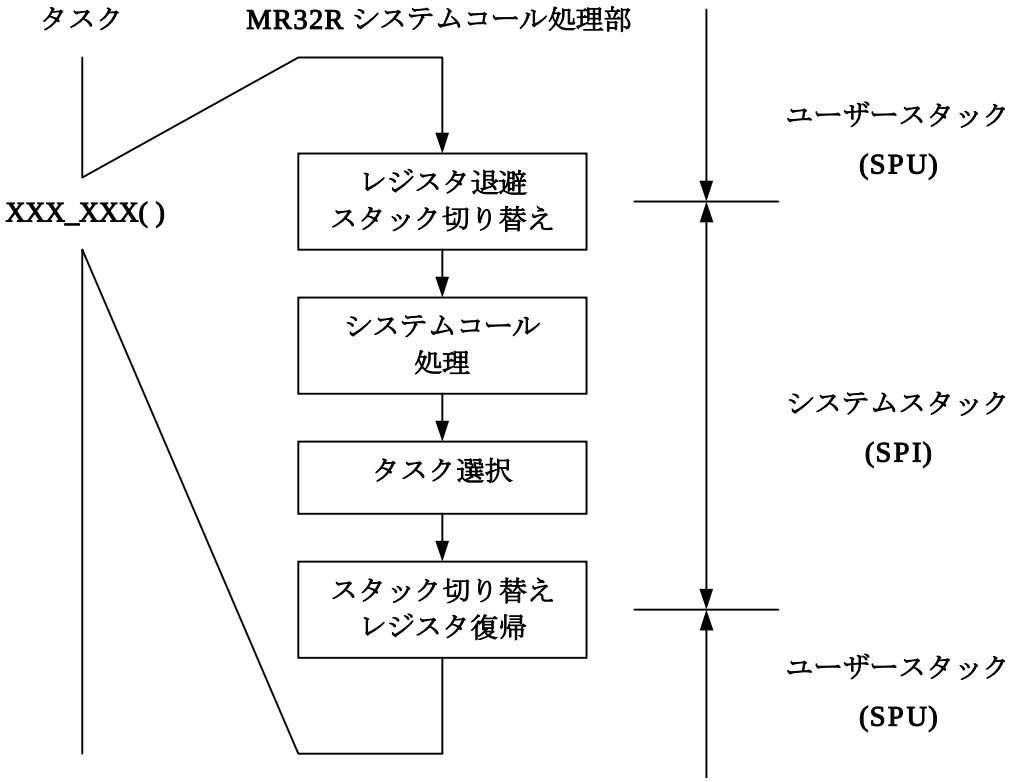


図 9.4　システムスタックとユーザスタック

第 10 章　付録

10.1サンプルプログラム

10.1.1　サンプルプログラム概要

MR32R の応用例として、タスク間で交互に標準出力に文字列を出力するプログラムを示します。

表 10-1 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	task1、task2 を起動させます。
task1()	タスク	2	2	“task1 running”を出力します。
task2()	タスク	3	3	“task2 running”を出力します。
cyh1()	ハンドラ	1		task1()を起床します。

以下に、処理内容を説明します。

- main タスクは、task1、task2、cyh1 を起動し、自タスクを終了させます。

- task1 は、次の順で動作します。

  1. セマフォを獲得します。
  2. 起床待ちに移行します。
  3. “task1 running”を出力します。
  4. セマフォを解放します。

- task2 は、次の順で動作します。

  1. セマフォを獲得します。
  2. “task2 running”を出力します。
  3. セマフォを解放します。

- cyh1 は、100ms 毎に起動し、task1 を起床します。

10.1.2 サンプルプログラムソース

```
 1 /***********************************************************************
 2 *                      MR32R  smaple program
 3 *
 4 * COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 * AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 *
 7 *
 8 *     $Id: smp.c,v 1.3 2003/05/12 10:27:29 inui Exp $
 9 ***********************************************************************/
10
11 #include <itron.h>
12 #include <kernel.h>
13 #include "kernel_id.h"
14 #include <stdio.h>
15
16
17 void main(void)
18 {
19         sta_tsk(ID_task1,0);
20         sta_tsk(ID_task2,0);
21         sta_cyc(ID_cyh1);
22 }
23 void task1(void)
24 {
25         while(1){
26                 wai_sem(ID_sem1);
27                 slp_tsk();
28                 printf("task1 running¥n");
29                 sig_sem(ID_sem1);
30         }
31 }
32
33 void task2(void)
34 {
35         while(1){
36                 wai_sem(ID_sem1);
37                 printf("task2 running¥n");
38                 sig_sem(ID_sem1);
39         }
40 }
41
42 void cyh1( VP_INT exinf )
43 {
44         iwup_tsk(ID_task1);
45 }
```

10.1.3 コンフィギュレーションファイル

```
 1 //************************************************************************
 2 //
 3 //   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 4 //   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 5 //
 6 //   MR32R System Configuration File.
 7 //   smp.cfg
 8 //      $Id: smp.cfg,v 1.3 2003/05/12 10:27:29 inui Exp $
 9 //************************************************************************
10
11 system{
12         stack_size      = 0x1000;
13         priority        = 4;
14         debug           = NO;
15         debug_buffer    = 0;
16         tick_nume       = 1;
17         tick_deno       = 1;
18         message_pri     = 255;
19 };
20 //
21 maxdefine{
22         max_task        = 3;
23         max_int         = 32;
24         max_alh =1;
25 };
26 //
27 clock{
28         timer_clock     = 27MHz;
29         IPL             = 2;
30         timer           = MFT00;
31         file_name       = m32104.tpl;
32 };
33 //
34 task[]{
35         entry_address   = main();
36         name            = ID_main;
37         stack_size      = 0x1000;
38         stack_area      = INTERNAL;
39         priority        = 1;
40         initial_start   = ON;
41 };
42 task[]{
43         entry_address   = task1();
44         name            = ID_task1;
45         stack_size      = 0x1000;
46         stack_area      = INTERNAL;
47         priority        = 2;
48 };
49 task[]{
50         entry_address   = task2();
51         name            = ID_task2;
52         stack_area      = INTERNAL;
53         stack_size      = 0x1000;
54         priority        = 3;
55 };
56 semaphore[]{
57         name            = ID_sem1;
58         max_count       = 1;
59         initial_count   = 1;
60         wait_queue      = TA_TPRI;
61 };
62
63 cyclic_hand [1] {
64         name                    = ID_cyh1;
65         interval_counter        = 100;
66         start                   = OFF;
67         phsatr                  = OFF;
68         phs_counter             = 0;
69         entry_address           = cyh1();
70         exinf                   = 1;
71 };
72
73 systemcall{
74         sta_tsk         = YES;
75         slp_tsk         = YES;
76         wup_tsk         = YES;
```

```
77        iwup_tsk        = YES;
78        wai_sem         = YES;
79        sig_sem         = YES;
80        ext_tsk         = YES;
81        sta_cyc         = YES;
82        stp_cyc         = YES;
83        exd_tsk         = NO;
84 };
85 interrupt_vector[16]   = __sys_timer;
86 //
87 // End of Configuraton
88
```

M32R ファミリ用リアルタイム OS
ユーザーズマニュアル
M3T-MR32R/4

発行年月日　2004 年 12 月 01 日 Rev.1.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
〒100-0004　東京都千代田区大手町 2-6-2

編集　　株式会社 ルネサス ソリューションズ　第一応用技術部

M3T-MR32R/4 V.4.00
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

RJJ10J0760-0100Z