

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のですが、誤りが無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、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。

M3T-MR30 V.3.30

ユーザーズマニュアル

M16C/60,30,20,10 シリーズ用リアルタイムOS

Microsoft、MS、MS-DOS、Windows、WindowsNT は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

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。

Solaris は、米国 Sun Microsystems, Inc. の登録商標です。

Sun、SunOS は、米国およびその他の国における米国 Sun Microsystems, Inc. の登録商標です。

IBM、AT は、米国 International Business Machines Corporation の登録商標です。

PC-9800 は、日本電気株式会社の商標です。

SPARC、SPARCstation は、米国 SPARC International, Inc. の登録商標です。

HP9000、HP-UX は、米国 Hewlett-Packard Company の米国およびその他の国における登録商標です。

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注意ください。

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。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を負いません。
- 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 マイコンツール部	
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

MR30 は M16C/60,30,20,10 シリーズ用のリアルタイム・オペレーティングシステム¹です。MR30 は μ ITRON 仕様²に準拠しています。

本マニュアルは MR30 を使用したプログラムの作成手順および作成上の注意事項について説明します。各システムコールの詳細な使用方法については「MR30 リファレンスマニュアル」を参照してください。

MR30 を使うために必要なこと

MR30 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。

- M16C/60,30,20,10 シリーズ用 C コンパイラ M3T-NC30WA
これらの製品をあわせて御使用頂ければ、より効率の良いプログラム開発がおこなえます。

ドキュメント一覧

MR30 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)
MR30 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)
MR30 のシステムコールの使用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。MR30 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 μ ITRON 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、 μ ITRON 仕様の著作権は同氏に属しています。MR30 は同氏に承認を得て、 μ ITRON 仕様に基づき製作されたものです。

MEMO

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 MR30 のねらい	4
2.2 TRON 仕様と MR30	6
2.3 MR30 の特長	8
第 3 章 MR30 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 システムコール	16
3.2.1 システムコール処理	17
3.2.2 システムコールにおけるタスクの指定方法	18
3.3 タスク	19
3.3.1 タスクの状態	19
3.3.2 タスクの優先度とレディキュー	23
3.3.3 タスクコントロールブロック (TCB)	24
3.4 ハンドラ	26
3.4.1 タスクとハンドラの違い	26
3.4.2 ハンドラ専用のシステムコール	28
3.5 MR30 カーネルの構成	29
3.5.1 モジュール構成	29
3.5.2 モジュール概要	30
3.5.3 タスク管理機能	31
3.5.4 タスク付属同期機能	34
3.5.5 同期・通信機能 (イベントフラグ)	36
3.5.6 同期・通信機能 (セマフォ)	38
3.5.7 同期・通信機能 (メールボックス)	40
3.5.8 割り込み管理機能	42
3.5.9 メモリプール管理機能	43
固定長メモリプール管理機能	43
可変長メモリプール管理機能	44
3.5.10 時間管理機能	46
3.5.11 バージョン管理機能	49
3.5.12 タスク、ハンドラから発行できるシステムコール一覧	50
第 4 章 アプリケーション作成手順概要	53
4.1 概要	54
4.2 開発手順例	56
4.2.1 アプリケーションプログラムのコーディング	56
4.2.2 コンフィグレーションファイル作成	58
4.2.3 コンフィグレータ実行	59
4.2.4 システム生成	59
4.2.5 ROM 書き込み	59

第 5 章 アプリケーション作成手順詳細	61
5.1 C 言語によるコーディング方法	62
5.1.1 タスクの記述方法	62
5.1.2 OS 依存割り込みハンドラの記述方法	64
5.1.3 OS 独立割り込みハンドラの記述方法	65
5.1.4 周期起動ハンドラ、アラームハンドラの記述方法	66
5.2 アセンブリ言語によるコーディング方法	67
5.2.1 タスクの記述方法	67
5.2.2 OS 依存割り込みハンドラの記述方法	69
5.2.3 OS 独立割り込みハンドラ記述方法	70
5.2.4 周期起動ハンドラ、アラームハンドラの記述方法	71
5.3 INT 命令の使用について	72
5.4 レジスタバンクについて	72
5.5 割り込みについて	73
5.5.1 割り込みハンドラの種類	73
5.5.2 ノンマスカブル割り込みについて	73
5.5.3 割り込み制御方法	74
5.6 ディスパッチ遅延について	76
5.7 初期起動タスクについて	78
5.8 MR30 スタートアッププログラムの修正方法	79
5.8.1 C 言語用スタートアッププログラム (crt0mr.a30)	80
5.9 メモリ配置方法	86
5.9.1 start.a30 のセクションの配置	87
5.9.2 crt0mr.a30 のセクション配置	88
5.10 機種別の注意事項	90
5.10.1 M16C/62 グループをご使用の場合	90
5.10.2 M16C/6N グループをご使用の場合	90
第 6 章 コンフィグレータの使用法	91
6.1 コンフィグレーションファイルの作成方法	92
6.1.1 コンフィグレーションファイル内の表現形式	92
6.1.2 コンフィグレーションファイルの定義項目	95
【システム定義】	95
【システムクロック定義】	96
【最大項目数定義】	98
【タスク定義】	99
【イベントフラグ定義】	100
【セマフォ定義】	101
【メールボックス定義】	102
【固定長メモリプール定義】	102
【可変長メモリプール定義】	103
【周期起動ハンドラ定義】	104
【アラームハンドラ定義】	105
【割り込みベクタ定義】	106
6.1.3 コンフィグレーションファイル例	108
6.2 コンフィグレータの実行	110
6.2.1 コンフィグレータ概要	110
6.2.2 コンフィグレータの環境設定	112
6.2.3 コンフィグレータ起動方法	113
6.2.4 makefile 生成機能	114
6.2.5 コンフィグレータ実行上の注意	115
6.2.6 コンフィグレータのエラーと対処方法	116
エラーメッセージ	116
警告メッセージ	117
その他のメッセージ	118
6.3 MAKEFILE の編集	119

第 7 章 アプリケーション作成の手引き	121
7.1 ハンドラからのシステムコールの処理手順.....	122
7.1.1 タスク実行中に割り込んだハンドラからのシステムコール.....	123
7.1.2 システムコール処理中に割り込んだハンドラからのシステムコール.....	124
7.1.3 ハンドラ実行中に割り込んだハンドラからのシステムコール.....	125
7.2 システムの使用する RAM 容量の計算方法.....	126
7.3 スタックについて.....	127
7.3.1 システムスタックとユーザースタック.....	127
第 8 章 サンプルプログラムの説明	129
8.1 概要.....	130
8.2 ソースプログラム.....	131
8.3 コンフィグレーションファイル.....	133
第 9 章 別 ROM 化について	135
9.1 別 ROM 化の方法.....	136
索引	143

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.....	15
図 3.9	システムコール.....	16
図 3.10	システムコールの処理の流れ.....	17
図 3.11	タスクの識別.....	18
図 3.12	タスクの状態.....	19
図 3.13	MR30 のタスク状態遷移図.....	20
図 3.14	レディーキュー (実行待ち状態).....	23
図 3.15	タスクコントロールブロック.....	25
図 3.16	周期起動ハンドラ、アラームハンドラの起動.....	27
図 3.17	MR30 の構成.....	29
図 3.18	タスクのリセット.....	31
図 3.19	優先度の変更.....	32
図 3.20	rot_rdq システムコールによるレディキューの操作.....	32
図 3.21	タスクの強制待ちと再開.....	34
図 3.22	起床要求の蓄積.....	35
図 3.23	起床要求のキャンセル.....	35
図 3.24	イベントフラグによるタスクの実行制御.....	37
図 3.25	セマフォによる排他制御.....	38
図 3.26	セマフォカウンタ.....	38
図 3.27	セマフォによるタスクの実行制御.....	39
図 3.28	メールボックス.....	40
図 3.29	メッセージの意味.....	40
図 3.30	メッセージキューのサイズ.....	41
図 3.31	割り込み処理の流れ.....	42
図 3.32	メモリプール管理.....	43
図 3.33	pget_blk 処理.....	44
図 3.34	rel_blk 処理.....	45
図 3.35	dly_tsk システムコール.....	46
図 3.36	タイムアウト処理.....	46
図 3.37	周期起動ハンドラ.....	47
図 3.38	周期起動ハンドラ:活性状態 TCY_ON を指定.....	47
図 3.39	周期起動ハンドラ:活性状態 TCY_INI_ON を指定.....	48
図 4.1	MR30 システム生成詳細フロー.....	55
図 4.2	プログラム例.....	57
図 4.3	コンフィグレーションファイル例.....	58
図 4.4	コンフィグレータ実行.....	59
図 4.5	システム生成.....	59
図 5.1	C 言語で記述したタスクの例.....	62

図 5.2	C 言語で記述した無限ループタスクの例	63
図 5.3	C 言語で記述した割り込みハンドラの例	64
図 5.4	OS 独立割り込みハンドラの例	65
図 5.5	C 言語で記述した周期起動ハンドラの例	66
図 5.6	アセンブリ言語で記述した無限ループタスクの例	67
図 5.7	アセンブリ言語で記述した ext_tsk で終了するタスクの例	67
図 5.8	OS 依存割り込みハンドラの例	69
図 5.9	特定レベルの OS 独立割り込みハンドラの例	70
図 5.10	アセンブリ言語で記述したハンドラの例	71
図 5.11	割り込みハンドラの IPL	73
図 5.12	タスクからのみ発行できるシステムコール内での割り込み制御	74
図 5.13	タスク独立部から発行できるシステムコール内での割り込み制御	75
図 5.14	C 言語用スタートアッププログラム (crt0mr.a30)	84
図 5.15	C 言語スタートアッププログラムのセクション配置	89
図 6.1	コンフィグレータ動作概要	111
図 7.1	タスク実行中に割り込んだ割り込みハンドラからのシステムコール処理手順	123
図 7.2	システムコール処理中に割り込んだ割り込みハンドラからのシステムコール処理手順	124
図 7.3	多重割り込みハンドラからのシステムコール処理手順	125
図 7.4	システムスタックとユーザースタック	127
図 8.1	LED 点灯の様子	130
図 9.1	ROM 分割	138
図 9.2	メモリマップ	140

表目次

表 2.1	MR30 概略仕様.....	7
表 3.1	ハンドラから発行できるシステムコール	28
表 3.2	タスク、ハンドラから発行できるシステムコール一覧.....	50
表 5.1	C 言語における変数の扱い	63
表 5.2	割り込み番号の割り当て	72
表 5.3	dis_dsp,loc_cpu に関する割り込み、ディスパッチの状態遷移	77
表 6.1	数値表現例	92
表 6.2	演算子.....	93
表 6.3	M16C/60 での固定ベクタ割り込み要因とベクタ番号との対応	107
表 7.1	MR_RAM セクションのサイズ算出方法	126
表 8.1	サンプルプログラムの関数一覧.....	130

第 1 章

ユーザーズマニュアルの構成

MR30 ユーザーズマニュアルは、9 つの章から構成されています。

- 第 2 章 概要
MR30 の目的や、概略の機能、位置づけなどを説明します。
- 第 3 章 MR30 入門
MR30 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
MR30 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
MR30 を使用してアプリケーションプログラムを作成する場合の開発手順を詳細に説明します。
- 第 6 章 コンフィグレータの使用法
コンフィグレーションファイルの記述方法、および、コンフィグレータの使用法を詳細に説明します。
- 第 7 章 アプリケーション作成の手引き
MR30 を使用してアプリケーションプログラムを作成する際に知っておいたほうがよい事項や、注意事項について説明します。
- 第 8 章 サンプルプログラムの説明
製品にソースファイル形式で含まれている MR30 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 9 章 別 ROM 化について
別 ROM 化の方法について説明します。

第 2 章

概要

2.1 MR30 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り、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 OS³という新しい手法の導入が必要になってきます。

そこで、弊社はこの要求に答えるべく 16 ビットマイクロプロセッサ M16C/60 シリーズ用にリアルタイム OS MR30 を開発しました。MR30 を導入することにより以下のような効果があります。

1. ソフトウェアの再利用が容易になります。

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

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

2. チームプログラミングがおこないやすくなります。

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

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

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

4. タイマ制御が簡単になります。

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

³ OS : Operating System

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

5. ソフトウェアの保守性が向上します。

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

6. ソフトウェアの信頼性が向上します。

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

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

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

2.2 TRON 仕様と MR30

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

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

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

1. システムコールのタイムアウト機能がありません。
2. タスク、セマフォ等のオブジェクトは、システム作成時のみに生成⁴することができます。システム起動後に生成⁵することはできません。
3. メモリプールは固定長のみで、可変長のメモリプールは扱えません。
4. システムコール例外管理機能および CPU 例外管理機能がありません。

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

MR30 は、この μITRON 仕様⁶に従って 16 ビットマイクロプロセッサ M16C/60 シリーズ用に開発された、リアルタイム・オペレーティングシステムです。

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

MR30 は、μITRON 仕様 V.3.0 で規定されたシステムコールのうち、レベル R、レベル S の全部と、レベル E の一部をインプリメントしています。

MR30 の概略仕様を表 2.1 に示します。

⁴ 静的オブジェクト生成

⁵ 動的オブジェクト生成

⁶ MR30 は μITRON 仕様 V.3.0 に準拠しています。

表 2.1 MR30 概略仕様

項目	仕様
ターゲットマイクロプロセッサ	M16C/60,30,20,10 シリーズ
最大タスク数	255
タスクの優先度数	255
最大イベントフラグ数	255
イベントフラグの幅	16 ビット
最大セマフォ数	255
セマフォの形式	計数型
最大メールボックス数	255
メッセージサイズ	16 ビット or 32 ビット
メールボックスのバッファサイズ	0 バイト以上
最大固定長メモリプール数	255
最大可変長メモリプール数	1
システムコール数	61
OS 核コードサイズ OS 核データサイズ	約 1.0K ~ 10.0K バイト 最小 16 バイト、1 タスクあたり（スタックを除く）11 バイト増加。なお、タイムアウト機能を使用した場合は、15 バイト増加
OS 核記述言語	C 言語、アセンブリ言語

2.3 MR30 の特長

MR30 は以下に示す特長を持っています。

1. μ ITRON 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

MR30 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 μ ITRON 仕様に基づいて開発されました。

μ ITRON 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 MR30 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

MR30 は M16C シリーズオブジェクトライブラリ形式で供給されています。

したがって、リンカージェディタ LN30 のもつ機能により、数ある MR30 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 NC30 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 NC30 を用いて、MR30 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 MR30 の機能を呼び出すためのインターフェースライブラリがソフトウェアディスクに添付されています。

5. 上流工程ツール "コンフィグレータ" により、容易な開発手順

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

第 3 章

MR30 入門

3.1 リアルタイム OS の考え方

本節では、リアルタイム OS の基本概念について説明します。

3.1.1 リアルタイム OS の必要性

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

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

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

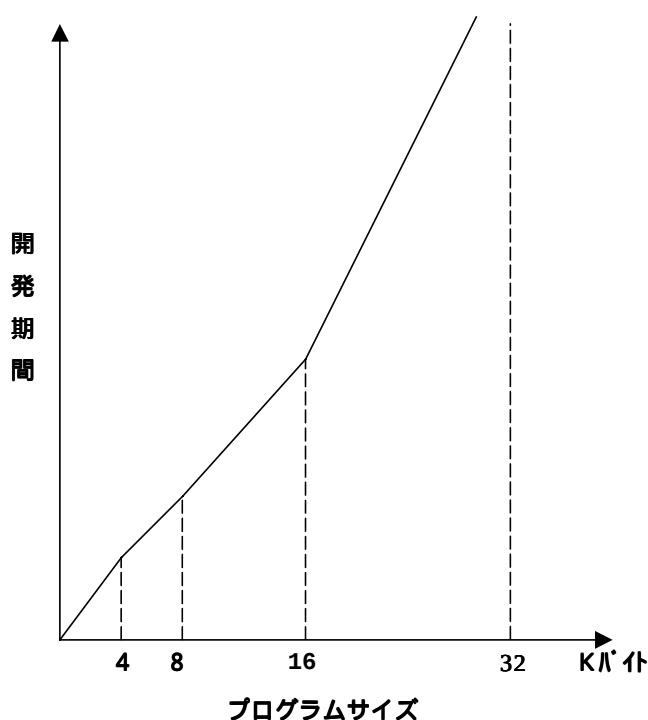


図 3.1 プログラムサイズと開発期間

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

⁷ ROM 詰めが必要ないことを前提とします

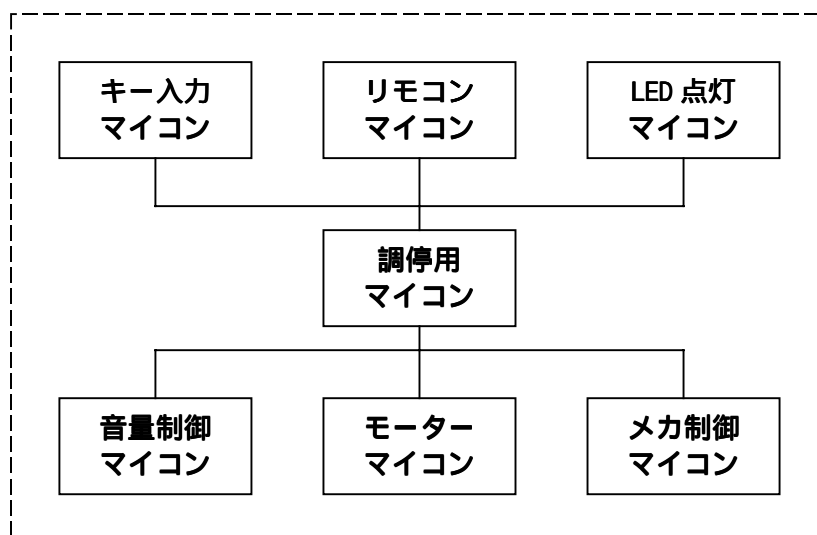


図 3.2 マイコンを多く使ったシステム例 (オーディオ機器)

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

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

この反面以下のような欠点があります。

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

そこでそれぞれのマイクロコンピュータで動作しているプログラムを、1つのマイクロコンピュータでソフトウェア的に、別々のマイクロコンピュータで動作しているように見せることのできるリアルタイム OS を採用すれば、上記の利点を残したままで欠点をすべて無くすることができます。

図 3.3に、図 3.2に示したシステムにリアルタイム OS を導入した場合のシステム例を示します。

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

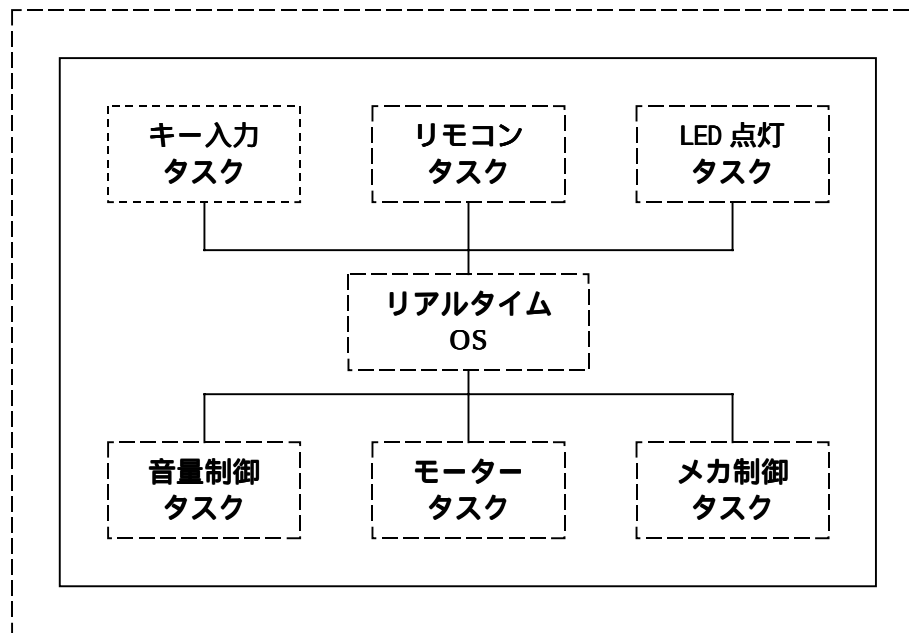


図 3.3 リアルタイム OS の導入システム例 (オーディオ機器)

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

3.1.2 リアルタイム OS の動作原理

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

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

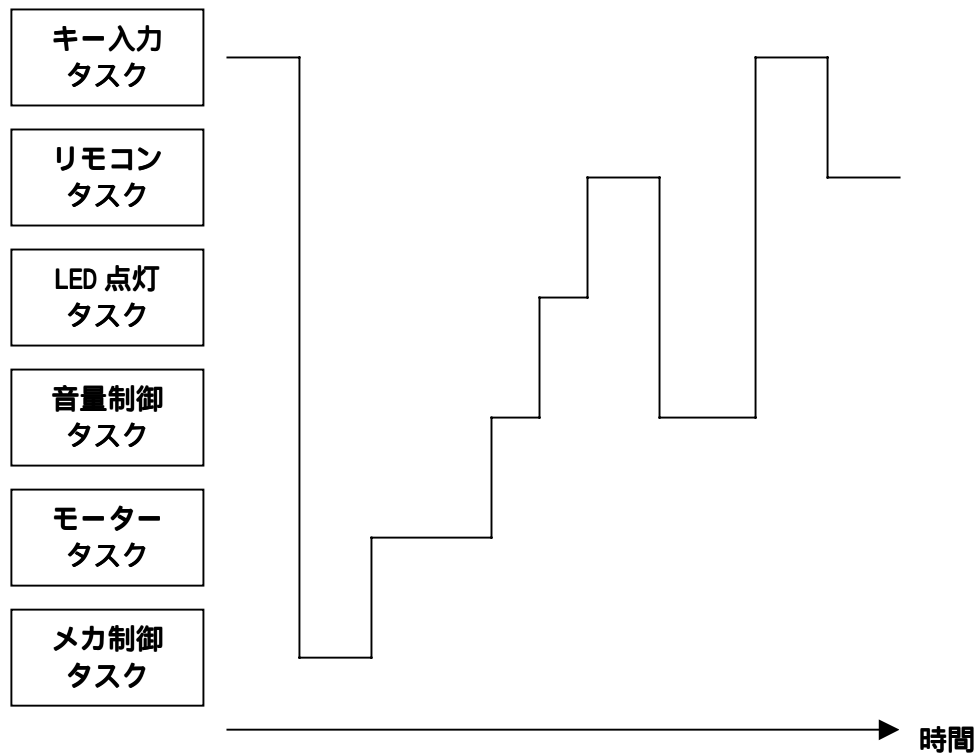


図 3.4 タスクの時分割動作

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

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

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

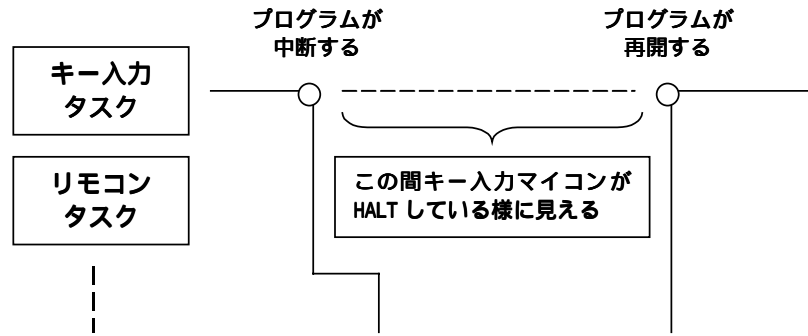


図 3.5 タスクの中断と再開

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

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

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

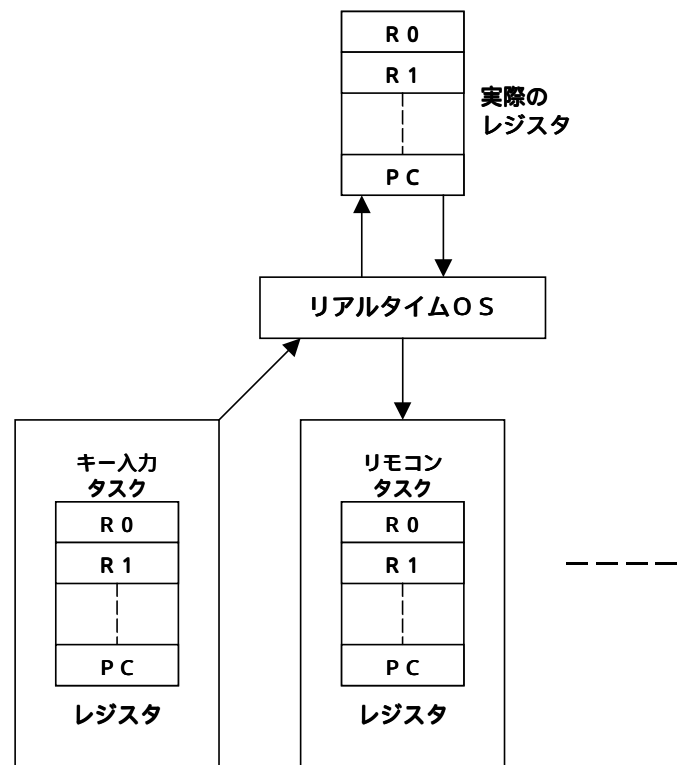


図 3.6 タスクの切り替え

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

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

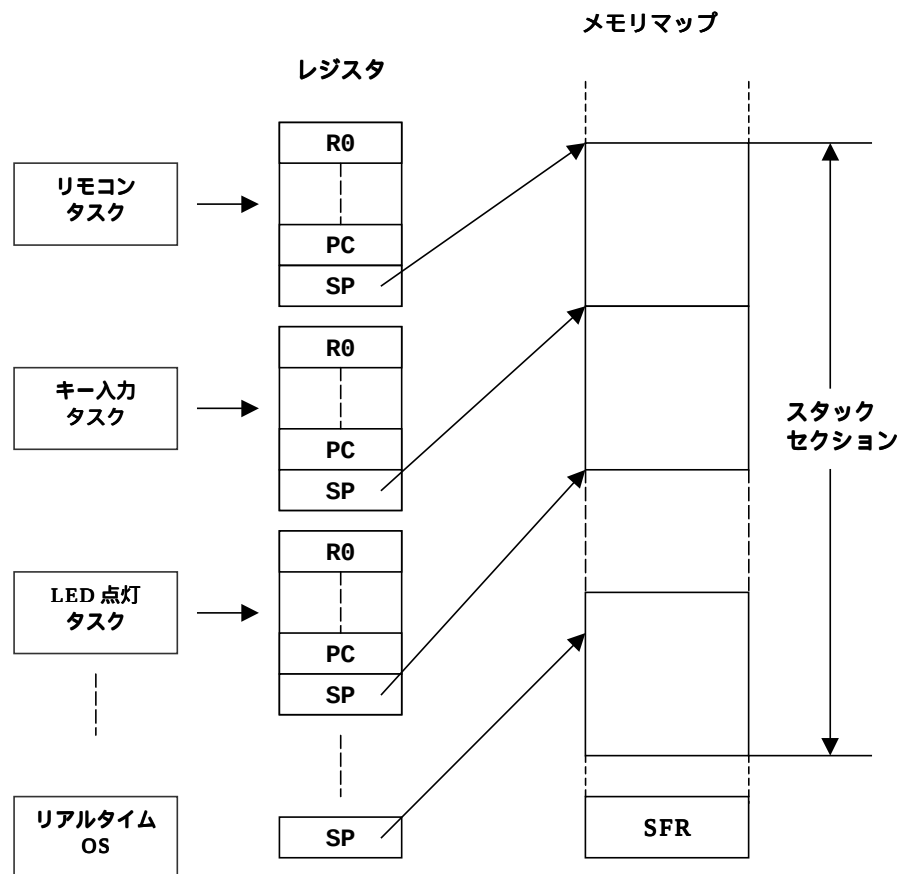


図 3.7 タスクのレジスタ領域

図 3.8は各タスクのレジスタおよびスタック領域を詳細に説明したものです。MR30 では各タスクのレジスタは図 3.8に示すようにスタック領域の中に格納され管理されています。図 3.8は、レジスタ格納後の状態を示しています。

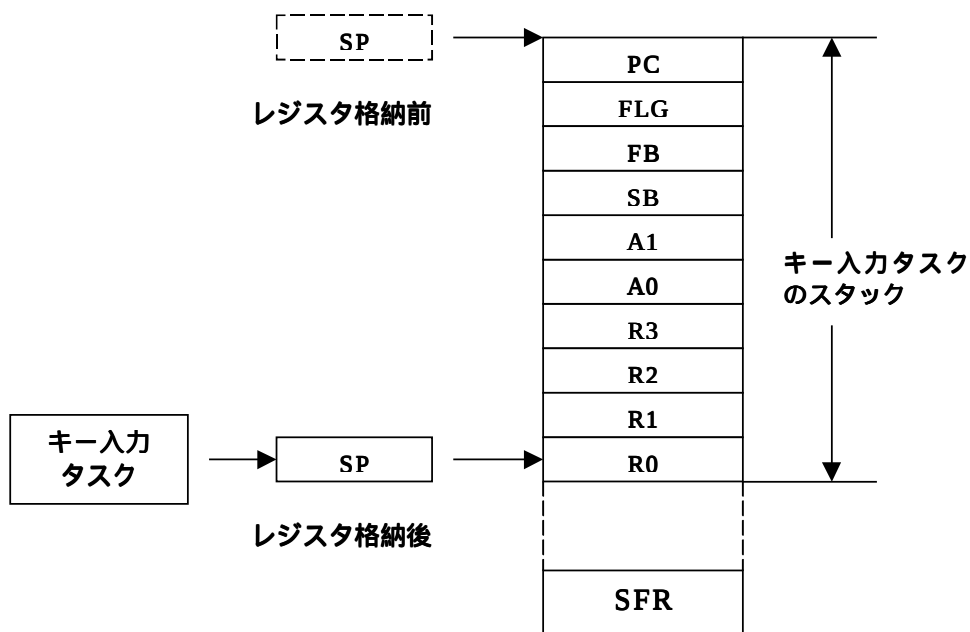


図 3.8 実際のレジスタとスタック領域の管理

3.2 システムコール

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

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



図 3.9 システムコール

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

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

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

```
sta_tsk #ID_main, #3
```

3.2.1 システムコール処理

システムコールが発行されると以下の手順により処理がおこなわれます。⁹

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

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

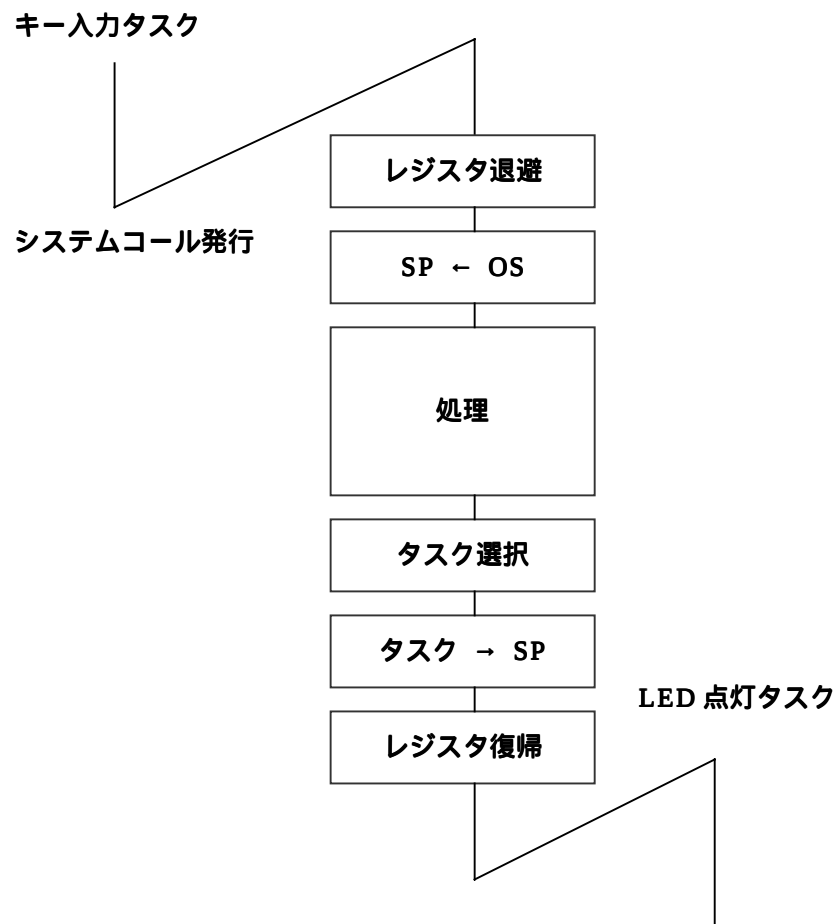


図 3.10 システムコールの処理の流れ

⁹ タスク切り替えの発生しないシステムコールはこの限りではありません。

3.2.2 システムコールにおけるタスクの指定方法

各タスクの識別は、リアルタイム OS MR30 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 MR30 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 MR30 に付属しているプログラム"コンフィグレータ cfg30"が自動的におこないます。図 3.11 にその様子を示します。

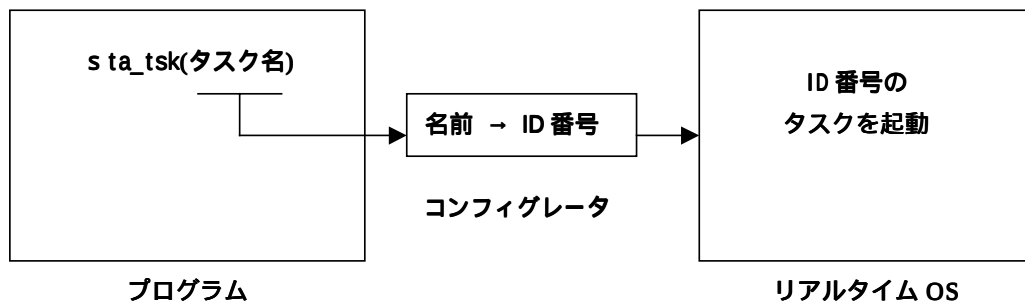


図 3.11 タスクの識別

これによりタスクの指定を以下のようにおこなえます。

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

この例では、関数名"task()"もしくはシンボル名"task:"のタスクを起動するように指定しています。

なお、タスクの名前から 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 MR30 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。

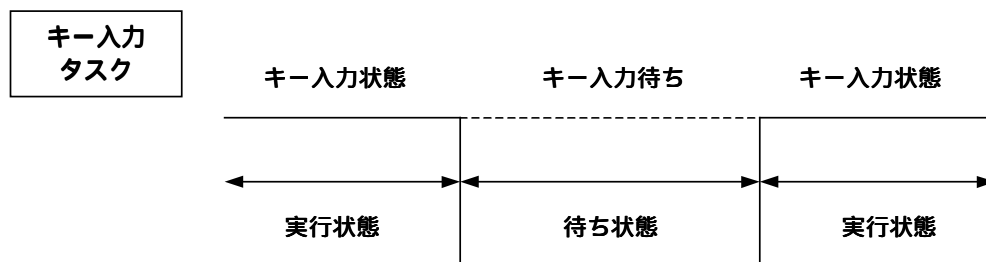


図 3.12 タスクの状態

MR30 では実行状態、待ち状態を含め以下の 6 つの状態を管理しています。

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

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

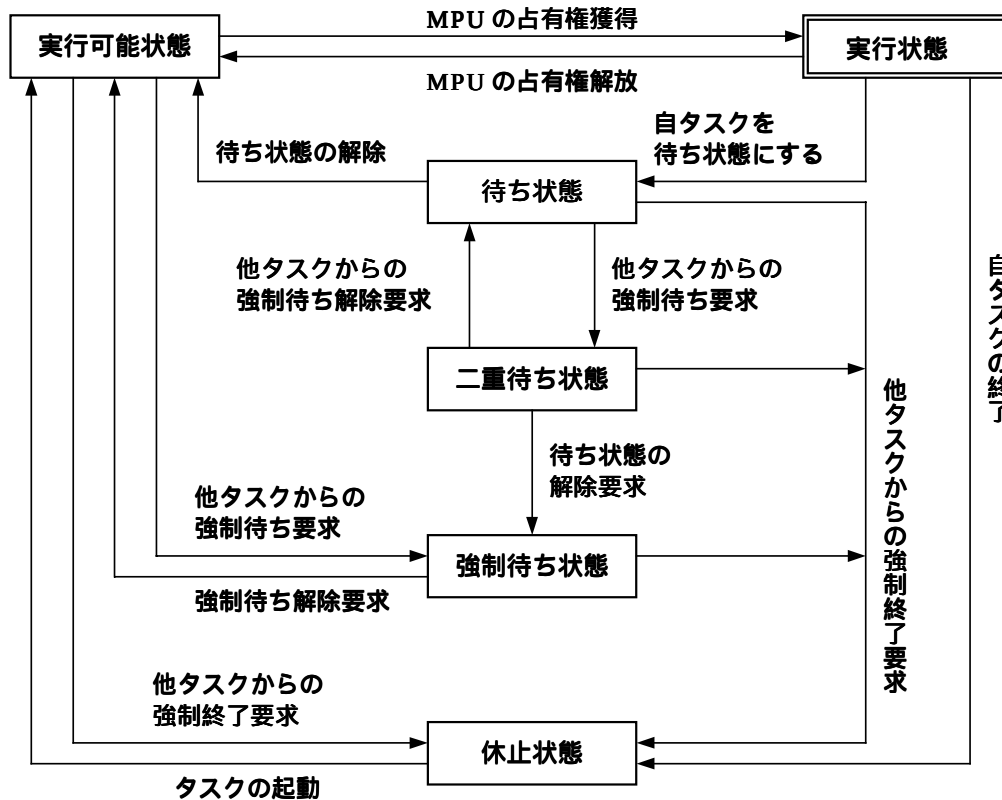


図 3.13 MR30 のタスク状態遷移図

1. 実行状態 (RUN 状態)

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

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

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

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

2. 実行可能状態 (READY 状態)

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

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

¹⁰ ext_tsk システムコールによる

¹¹ dly_tsk, slp_tsk, tslp_tsk, wai_flg, twai_flg, wai_sem, twai_sem, rcv_msg, trcv_msg システムコールによる

¹² chg_pri システムコールによる

¹³ ichg_pri システムコールによる

¹⁴ レディーキューについては次節参照

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

3. 待ち状態 (WAIT 状態)

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

実行待ち状態に移行するには以下の方法があります。

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

4. 強制待ち状態 (SUSPEND 状態)

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

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

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

5. 二重待ち状態 (WAIT-SUSPEND 状態)

待ち状態にあるタスクに強制待ちの要求が出された場合、タスクは二重待ち状態になります。
`wai_flg`、`wai_sem`、`rcv_msg` システムコールによる要求待ちで待ち状態にあるタスクに対して強制待

¹⁵ `ext_tsk` システムコールによる

¹⁶ `dly_tsk`、`slp_tsk`、`tslp_tsk`、`wai_flg`、`twai_flg`、`wai_sem`、`twai_sem`、`rcv_msg`、`trcv_msg` システムコールによる

¹⁷ `chg_pri` システムコールによる

¹⁸ `icg_pri` システムコールによる

¹⁹ `twai_flg`、`twai_sem`、`trcv_msg` システムコールも含まれます。

²⁰ ハンドラから `isus_tsk` システムコールにより実行タスクを強制待ち状態にする場合は、実行状態から直接強制待ち状態に移行されます。例外的にこの場合のみ実行状態から強制待ち状態に移行する場合はあることに注意してください。

²¹ 待ち状態にあるタスクに対して強制待ち要求をおこなうと二重待ち状態になります。

ち要求が出された場合、そのタスクは要求待ち行列から外されず単にそのタスクが二重待ち状態に移行するだけです。

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

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

6. 休止状態 (DORMANT 状態)

通常は、MR30 システムに登録されているが起動していない状態です。この状態になるには以下の 2 つの場合があります。

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

²² Rsm_tsk, irsm_tsk システムコール

²³ ext_tsk システムコール

²⁴ ter_tsk システムコール

3.3.2 タスクの優先度とレディキュー

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

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

MR30 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 3.14²⁵にレディキューの構造を示します。レディキューは優先度ごとに管理され、タスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。²⁶

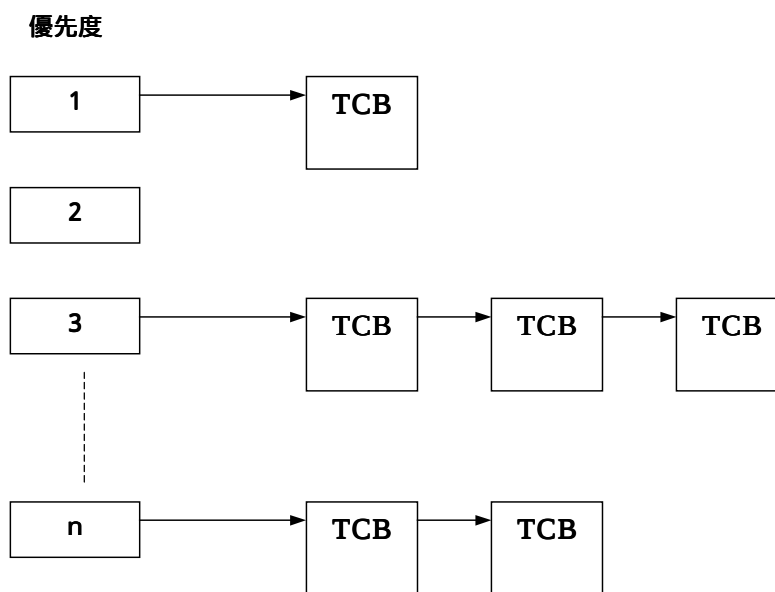


図 3.14 レディーキュー（実行待ち状態）

²⁵ TCB:タスクコントロールブロックについては次節で述べます。

²⁶ 実行状態のタスクはレディキューにつながれたままです。

3.3.3 タスクコントロールブロック (TCB)

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

MR30 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
- タスクのレジスタ情報など²⁷を格納したスタック領域のポインタ (現在の SP レジスタの値)
- 起床要求カウンタ
タスクの起床要求カウンタを蓄積する領域
- タイムアウトカウンタ、待ちフラグパターン
タスクがタイムアウト待ち状態である時は、残りの待ち時間が格納され、フラグ待ち状態であれば、フラグの待ちパターンがこの領域に格納されます。
- フラグ待ちモード
イベントフラグ待ちの時の待ちモード
- タイマキュー接続ポインタ
タイムアウト機能を使用した場合に使用する領域です。タイマキューを構成する時に使用するタスクの接続用ポインタを格納する領域です。
- フラグ待ちパターン
タイムアウト機能を使用した場合に使用する領域です。
タイムアウト機能付きのイベントフラグ待ちのシステムコール (twai_flg)を使用した場合に、フラグ待ちパターンが格納されます。なお、この領域は、イベントフラグを使用しない場合は、確保されません。

タスクコントロールブロックを図 3.15に示します。

²⁷ これをタスクコンテキストと呼びます。

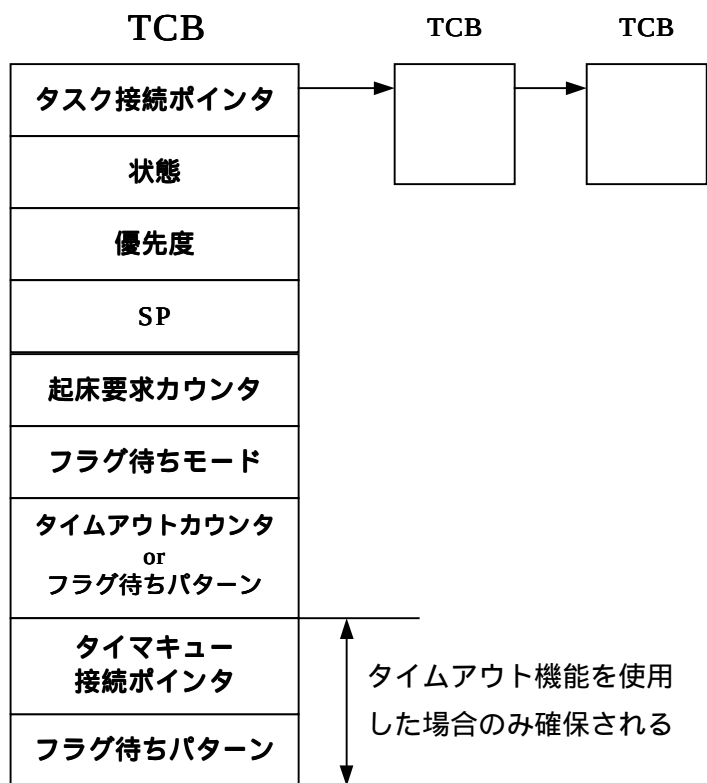


図 3.15 タスクコントロールブロック

3.4 ハンドラ

3.4.1 タスクとハンドラの違い

タスクは MR30 が実行制御するプログラムの単位ですから、タスクはそれぞれ独立したコンテキスト (プログラムカウンタ、スタックポインタ、ステータスレジスタ、その他のレジスタ) を持っています。したがってあるタスクからあるタスクに実行を移す²⁸にはこのコンテキストを切り替える必要があります。この処理には時間を要します。

割り込みなどの処理は、高速に応答する必要がありますので MR30 にはコンテキストを切り替えずに処理する機能があります。すなわち割り込まれたタスクのコンテキスト (レジスタ) をそのまま使用してプログラムを動作させることができます。このプログラムのことをハンドラと呼びます。ハンドラは割り込まれたタスクのコンテキスト (レジスタ) をそのまま使用しますので必ずハンドラの先頭で割り込まれたタスクのコンテキストをメモリに退避し、タスクに復帰するときにはその退避したコンテキストをもとに戻さなくてはなりません。

また、アセンブリ言語で記述する場合、割り込みハンドラからの復帰には `ret_int` システムコールを通常は使用してください。(5.2.2節を参照)ただし割り込みハンドラ処理内で MR30 のシステムコールを使用していない場合は、`reit` 命令で復帰してもかまいません。(5.2.3節を参照してください。)

ハンドラには以下のものがあります。

1. 割り込みハンドラ

ハードウェア割り込みにより起動されるプログラムを割り込みハンドラと呼びます。割り込みハンドラの起動には MR30 は全く関与しません。したがって割り込みハンドラの入り口アドレスを割り込みベクターテーブルに直接書き込みます。

割り込みハンドラには、OS 独立割り込み、OS 依存割り込みの 2 種類があります。各割り込みについては、5.5節を参照して下さい。

2. 周期起動ハンドラ

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

3. アラームハンドラ

アラームハンドラは、指定した時刻に起動されるハンドラです。

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

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

²⁸ このことをディスパッチもしくはタスク切り替えと呼びます。

²⁹ `act_cyc` システムコール

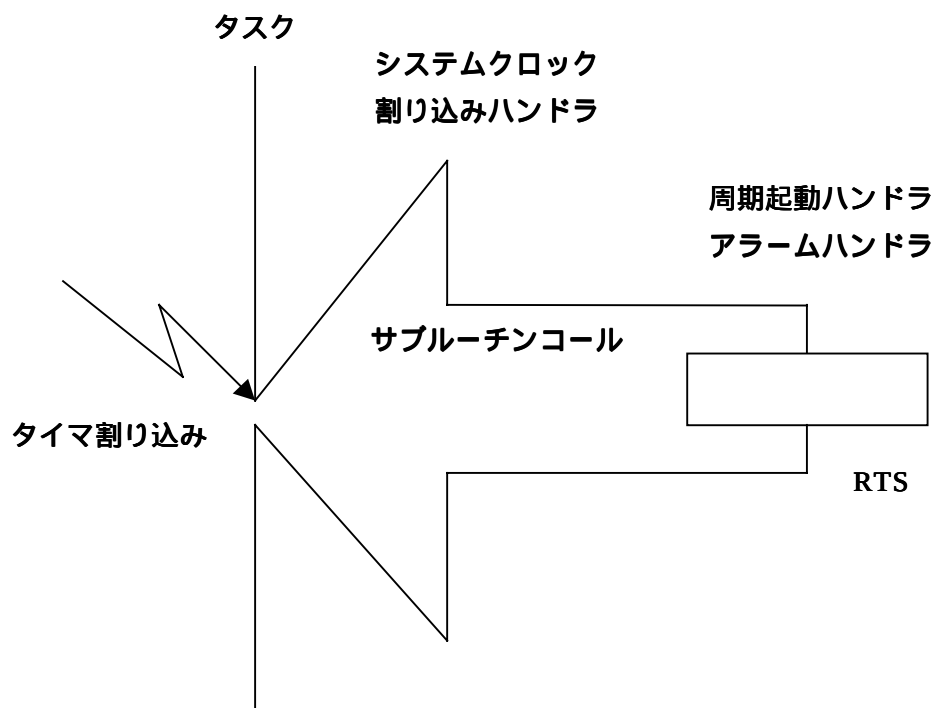


図 3.16 周期起動ハンドラ、アラームハンドラの起動

3.4.2 ハンドラ専用のシステムコール

MR30 では以下に示すシステムコールはハンドラからのみ発行できます。

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

表 3.1 ハンドラから発行できるシステムコール

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

³⁰ `#pragma INTHANDLER` で割り込みハンドラを指定した場合（C 言語）このシステムコールを記述する必要はありません。

3.5 MR30 カーネルの構成

3.5.1 モジュール構成

MR30 カーネルは、図 3.17に示すモジュールから構成されています。これらの個々のモジュールはそれぞれのモジュールの機能を実現する関数群より構成されています。

MR30 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 LN30 の機能によりリンクします。ただし、スケジューラとタスク管理の一部および時間管理の一部は必須機能関数ですので常時リンクされます。

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

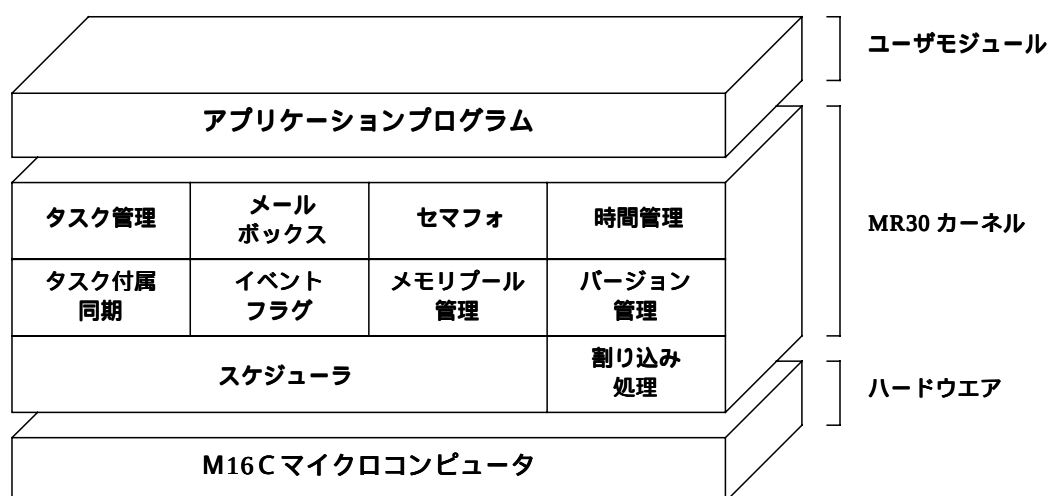


図 3.17 MR30 の構成

³¹ 詳細は第3.5.10節を参照。

3.5.2 モジュール概要

MR30 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
MR30 カーネルで使用するシステムタイマの設定、タイムアウトの処理、ユーザーの作成したアラームハンドラ³²、周期起動ハンドラ³³の起動をおこないます。
- バージョン管理
MR30 カーネルのバージョン番号等の情報を報告します。
- 同期・通信
タスク間の同期をとったりタスク間の通信をおこなうための機能です。以下の 3 つの機能モジュールが用意されています。
 - ◆ イベントフラグ
MR30 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どうかを制御します。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
MR30 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どうかを制御します。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
タスク間のデータの通信をデータの先頭アドレスを渡すことによりおこないます。
- メモリプール管理
タスクまたはハンドラが使用するメモリ領域の動的な獲得および解放を行います。

³² 指定時刻に一回のみ起動されるハンドラです。

³³ 周期的に起動されるハンドラです。

3.5.3 タスク管理機能

タスク管理機能とは、タスクの起動・終了・優先度の変更等のタスク操作をおこなう機能です。MR30 カーネルが提供するタスク管理機能のシステムコールには、次のものがあります。

- タスクを起動する (sta_tsk)
あるタスクから、他タスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態もしくは実行状態に移行します。
- ハンドラからタスクを起動する (ista_tsk)
ハンドラからタスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態に移行します。
- 自タスクを終了する (ext_tsk)
自タスクを終了するとタスクの状態が休止状態になります。これにより再起動されるまではこのタスクは実行しません。
- 他タスクを強制的に終了させる (ter_tsk)
休止状態以外の他のタスクを強制的に終了させ休止状態にします。
タスクを強制的に終了させた後に再度そのタスクを起動するとそのタスクがリセットしたように振る舞います。(図 3.18参照)

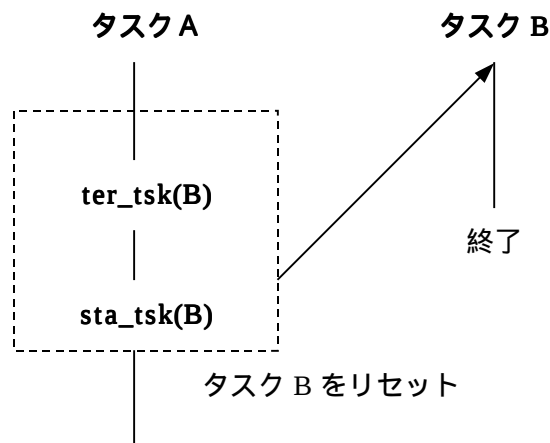


図 3.18 タスクのリセット

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

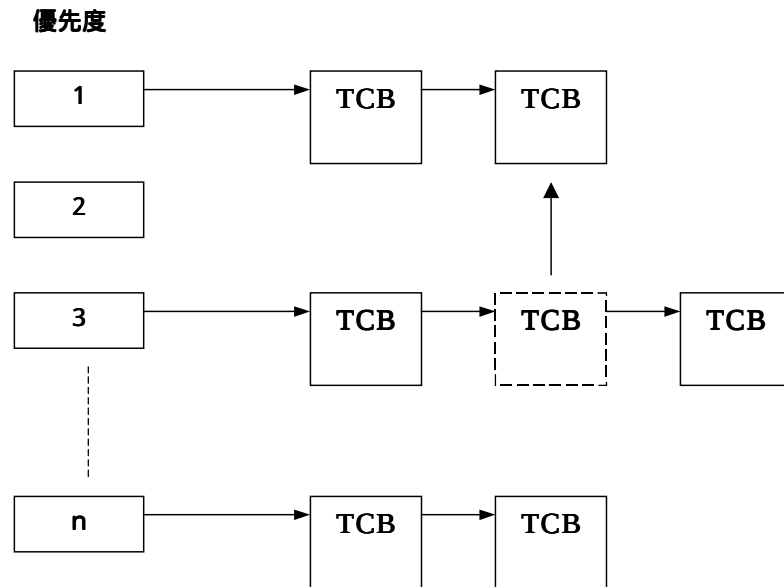


図 3.19 優先度の変更

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

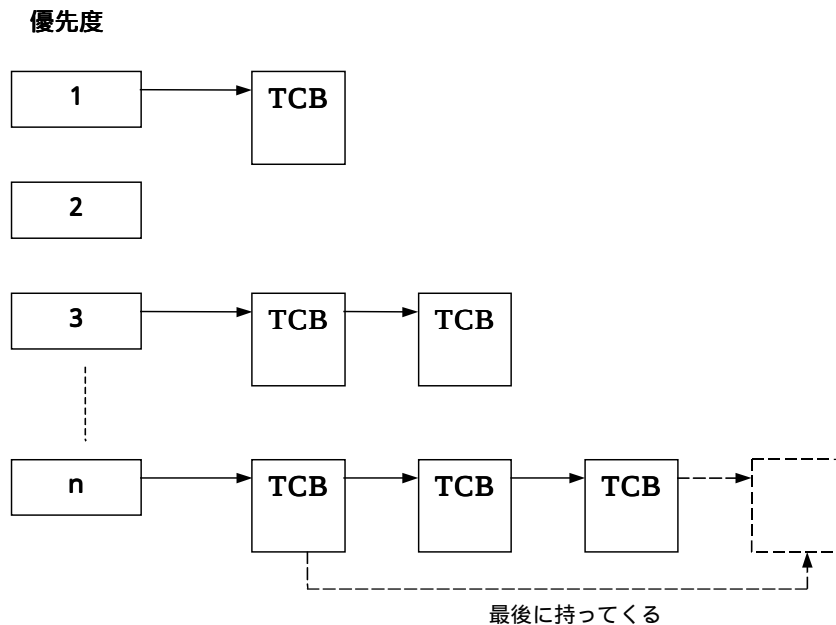


図 3.20 rot_rdq システムコールによるレディキューの操作

- タスクの待ち状態を強制解除する (rel_wai, irel_wai)
タスクの待ち状態を強制的に解除します。解除される待ち状態は以下の条件により待ちに入ったタスクです。
 - ◆ タイムアウト待ち状態
 - ◆ slp_tsk システムコールによる (+タイムアウト有)待ち状態
 - ◆ イベントフラグ (+タイムアウト有)待ち状態
 - ◆ セマフォ(+タイムアウト有) 待ち状態
 - ◆ メッセージ (+タイムアウト有)待ち状態

- 自タスクの ID を得る (`get_tid`)
自タスクの ID 番号を得ます。ハンドラから発行した場合は、ID 番号の代わりに 0(ゼロ)が得られます。
- タスクの状態を参照する (`ref_tsk`)
対象タスクの状態およびその優先度等を参照します。

3.5.4 タスク付属同期機能

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

MR30 カーネルが提供するタスク付属同期システムコールには次のものがあります。

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

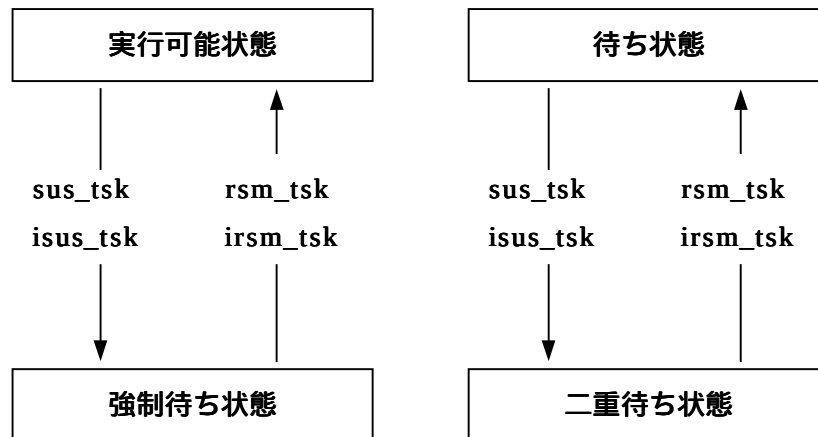


図 3.21 タスクの強制待ちと再開

- タスクを待ち状態に移行する (`slp_tsk`, `tslp_tsk`)
- 待ち状態のタスクを起床する (`wup_tsk`, `iwup_tsk`)
`slp_tsk`, `tslp_tsk` システムコールにより待ち状態に入ったタスクを起床させます。
`slp_tsk`, `tslp_tsk` システムコール以外の条件で待ち状態にあるタスクは起床できません。³⁴
`slp_tsk`, `tslp_tsk` システムコール以外の条件で待ちに入ったタスクや休止状態を除く他の状態のタスクに対して `wup_tsk`, `iwup_tsk` システムコールにより起床要求をおこなうと、この起床要求だけが蓄積されます。
したがって、例えば実行状態のタスクに対して起床要求をおこなうと、この起床要求が一時的に記憶されます。そして、その実行状態のタスクが `slp_tsk`, `tslp_tsk` システムコールにより待ち状態に入ろうとした時、蓄積された起床要求が有効になり、待ち状態にならずに再び実行を続けます。（図 3.22 参照）
- タスクの起床要求を無効にする (`can_wup`)
蓄積された起床要求をクリアします。（図 3.23 参照）

³⁴ 待ち状態であっても以下の条件で待っているタスクは起床されませんのでご注意ください。

- ◆ イベントフラグ待ち状態
- ◆ セマフォ待ち状態
- ◆ メッセージ待ち状態
- ◆ タイムアウト待ち状態

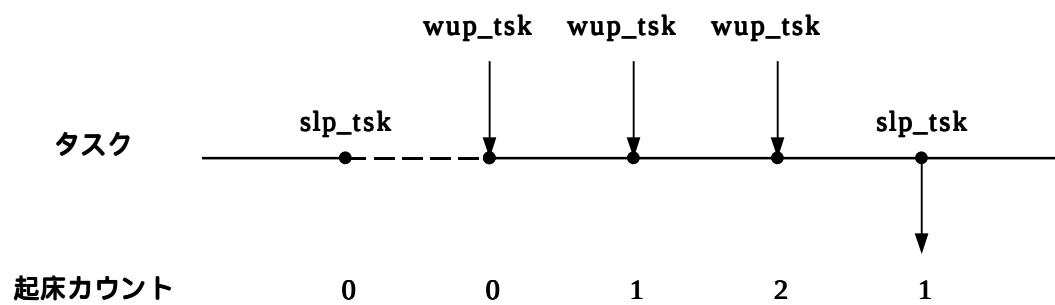


図 3.22 起床要求の蓄積

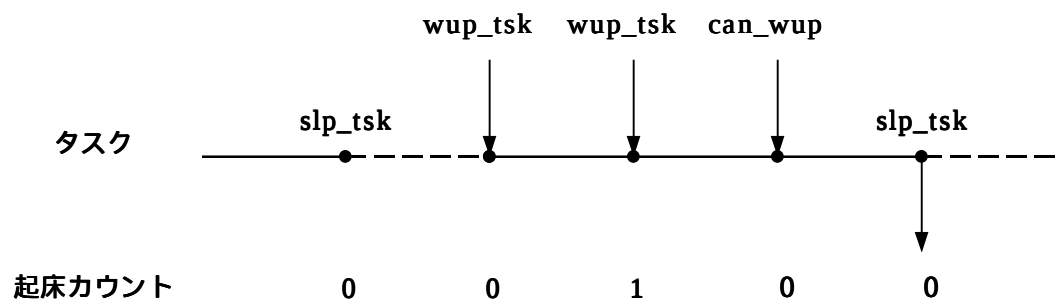


図 3.23 起床要求のキャンセル

3.5.5 同期・通信機能 (イベントフラグ)

イベントフラグは複数のタスクの実行の同期をとるための MR30 内部に持つ機構です。イベントフラグは、フラグ待ちパターンと 16 ビットのビットパターンによりタスクの実行制御をおこないます。タスクは、設定したフラグ待ちの条件が満たされるまで待ちます。

MR30 カーネルが提供するイベントフラグのシステムコールには次のものがあります。

- イベントフラグをセットする (set_flg, iset_flg)
イベントフラグをセットします。これにより、このイベントフラグの待ちパターンを待っていたタスクは待ち解除されます。
- イベントフラグをクリアする (clr_flg)
イベントフラグをクリアします。
- イベントフラグを待つ (wai_flg, twai_flg)
イベントフラグがあるパターンにセットされるのを待ちます。イベントフラグを待つ時のモードは、以下に示す 3 種類があります。
 - ◆ AND 待ち
指定されたビットが全てセットされるのを待ちます。
 - ◆ OR 待ち
指定されたビットの内いずれか 1 ビットがセットされるのを待ちます。
 - ◆ クリア指定
AND 待ちか OR 待ちの条件が満たされた場合にフラグをクリアします。
- イベントフラグを得る (pol_flg)
イベントフラグがあるパターンになっているか否かを調べます。このシステムコールではタスクは待ち状態に移行しません。
- イベントフラグの状態を得る (ref_flg)
対象イベントフラグのビットパターンや待ちタスクの有無を参照します。

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

イベントフラグは複数のタスクを一度に起床できるという特徴があります。

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

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

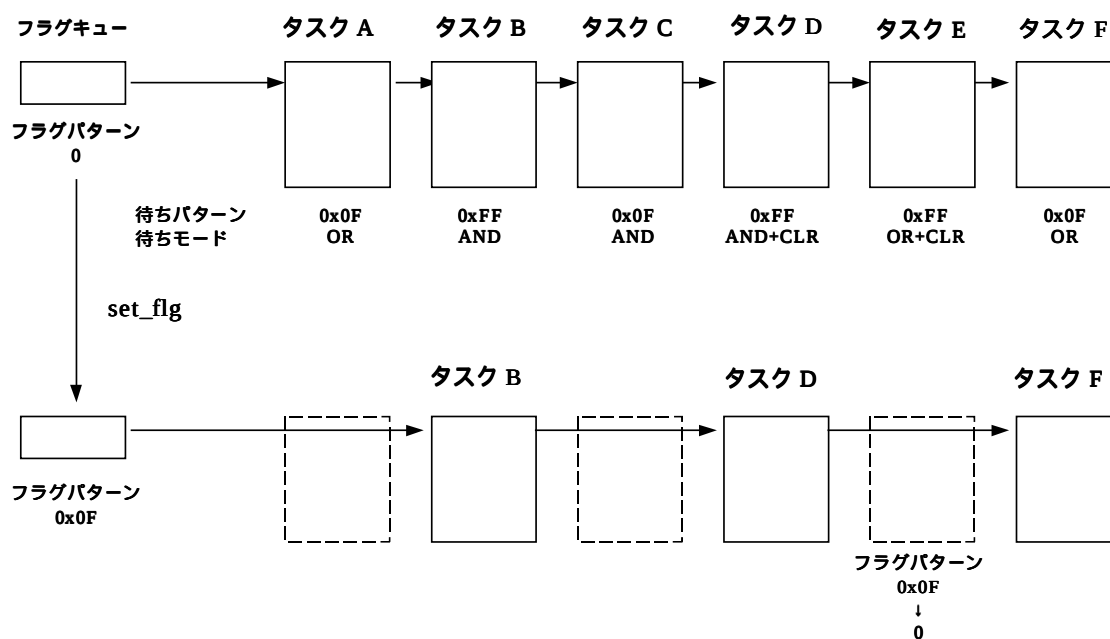


図 3.24 イベントフラグによるタスクの実行制御

3.5.6 同期・通信機能 (セマフォ)

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

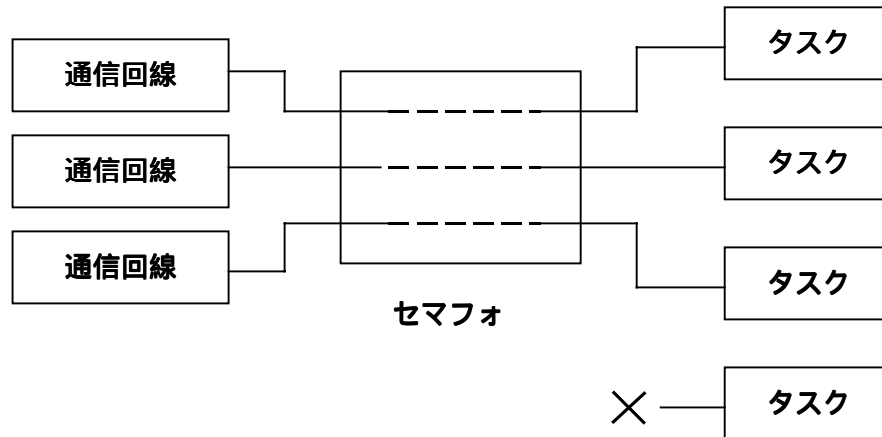


図 3.25 セマフォによる排他制御

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

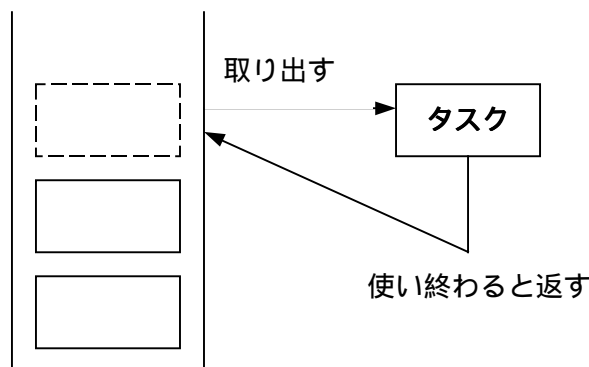


図 3.26 セマフォカウンタ

MR30 カーネルが提供するセマフォ同期のシステムコールには次のものがあります。

- セマフォに対する信号操作 (sig_sem, isig_sem)
セマフォに信号をおくります。すなわち、セマフォを待っているタスクがあればそのタスクを起床し、なければセマフォカウンタを 1 増やします。
- セマフォ獲得操作 (wai_sem, twai_sem)
セマフォを待ちます。セマフォカウンタが 0 であればセマフォを得ることができませんので待ち状態になります。
- セマフォ獲得操作 (preq_sem)
セマフォを得ます。得るべきセマフォがなければ待ち状態に入らずにエラーコードをかえします。
- セマフォの状態を参照する (ref_sem)
対象セマフォの状態を参照します。対象セマフォのカウント値や待ちタスクの有無を参照します。wai_sem、sig_sem システムコールを用いたタスクの実行制御の例を図 3.27に示します。

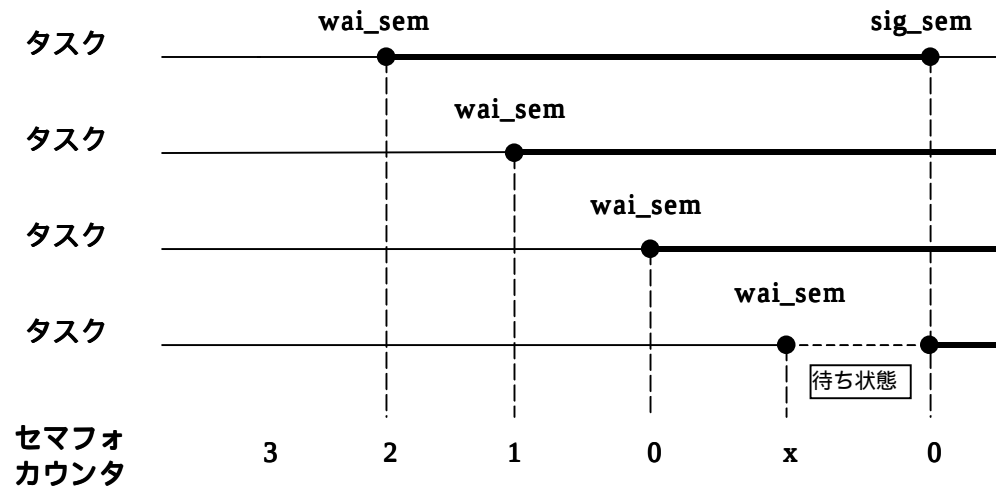


図 3.27 セマフォによるタスクの実行制御

3.5.7 同期・通信機能 (メールボックス)

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 3.28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。

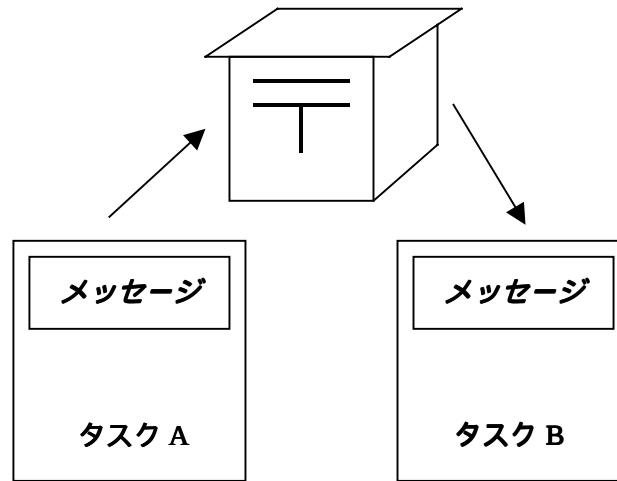


図 3.28 メールボックス

このメールボックスに投函できるメッセージは 16 ビットあるいは 32 ビットのデータです³⁵。

MR30 では、このデータをメッセージパケットの先頭アドレスとして使用することを標準としています³⁶、単なるデータとして使用することも可能です³⁷。

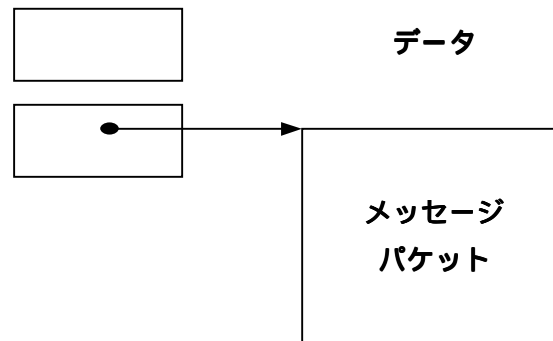


図 3.29 メッセージの意味

メールボックスにはメッセージを蓄積する機能があります。蓄積されたメッセージは FIFO³⁸でメッセージが取り出されます。ただし、メールボックスに蓄積できるメッセージの数には制限があります。

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

³⁵ メッセージサイズの指定は、コンフィグレーションファイルのシステム定義で行います。

³⁶ ITRON 仕様では、メッセージパケットの先頭アドレスとして使用することを標準としています。

³⁷ この場合、システムコールの引数であるデータにキャストして、ポインタに型変換しなければなりません

³⁸ ファーストインファーストアウト

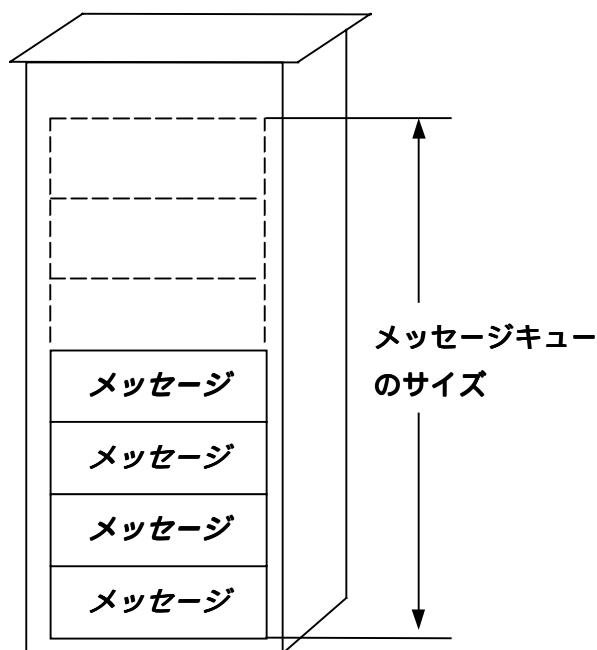


図 3.30 メッセージキューのサイズ

MR30 カーネルが提供するメールボックスのシステムコールには次のものがあります。

- メッセージを送信します (snd_msg, isnd_msg)
メッセージを送信します。すなわち、メッセージをメールボックスに投函します。
- メッセージを受信します (rcv_msg, trcv_msg)
メッセージを受信します。すなわち、メッセージをメ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い場合は、投函されるまで待ち状態になります。
- メッセージを受信します (prcv_msg)
メッセージを受信します。rcv_msg システムコールと異なるのは、メ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)
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。

3.5.8 割り込み管理機能

割り込み管理機能は外部割り込みの発生に対して、実時間で処理をおこなう機能を提供します。MR30 カーネルが提供する割り込み管理システムコールには次のものがあります。

- 割り込みハンドラから復帰します (ret_int)
ret_int システムコールは、割り込みハンドラから復帰するとき、必要ならばスケジューラを起動し、タスク切り替えをおこないます。
本機能は、C 言語を用いた場合³⁹、ハンドラ関数の終了時に自動で呼び出されます。従って、この場合、本システムコールを呼び出す必要はありません。
- 割り込みおよびタスクのディスパッチを禁止します (loc_cpu)
loc_cpu システムコールは、OS 依存の外部割り込みとタスクのディスパッチを禁止します。
- 割り込みおよびタスクのディスパッチを許可します (unl_cpu)
unl_cpu システムコールは、OS 依存の外部割り込みとタスクのディスパッチを許可します。従って、loc_cpu システムコールによる割り込み、およびディスパッチの禁止状態がこのシステムコールの発行により解除されます。

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

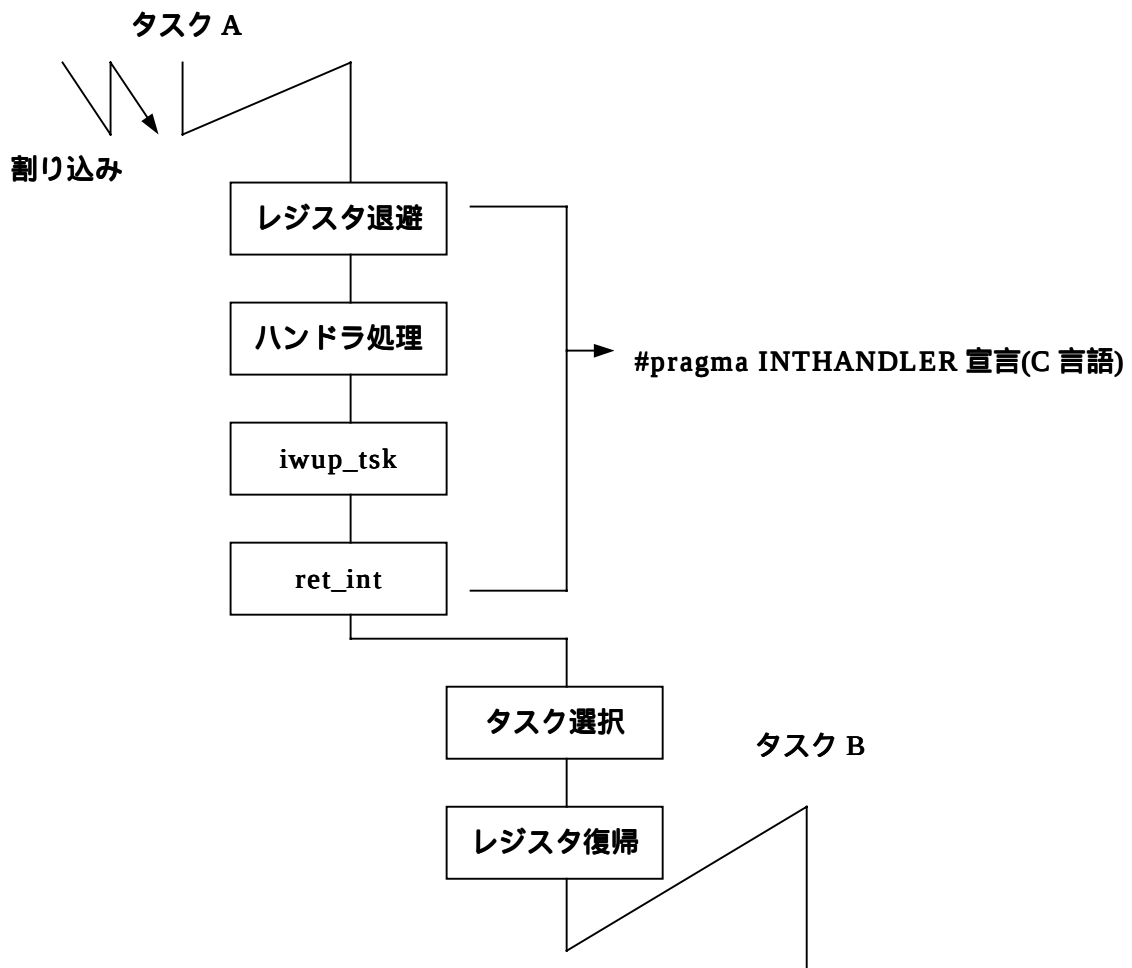


図 3.31 割り込み処理の流れ

³⁹ 割り込みハンドラを、#pragma INTHANDLER で指定した場合

3.5.9 メモリプール管理機能

メモリプール管理機能はシステムのメモリ空間 (RAM 空間) を動的に管理する機能を提供します。

特定のメモリ領域 (メモリプール) を管理し、そのメモリプールからタスクあるいはハンドラの必要とするメモリブロックを動的に確保し、また不用になったメモリブロックをメモリプールに解放します。

MR30 では、固定長と可変長の 2 つのメモリプール管理機能をサポートしています。

固定長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが決まっていることを固定長といいます。

獲得するメモリブロックサイズは、コンフィグレーションファイルで指定します。

MR30 カーネルが提供する固定長メモリプール管理システムコールには次のものがあります。

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

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

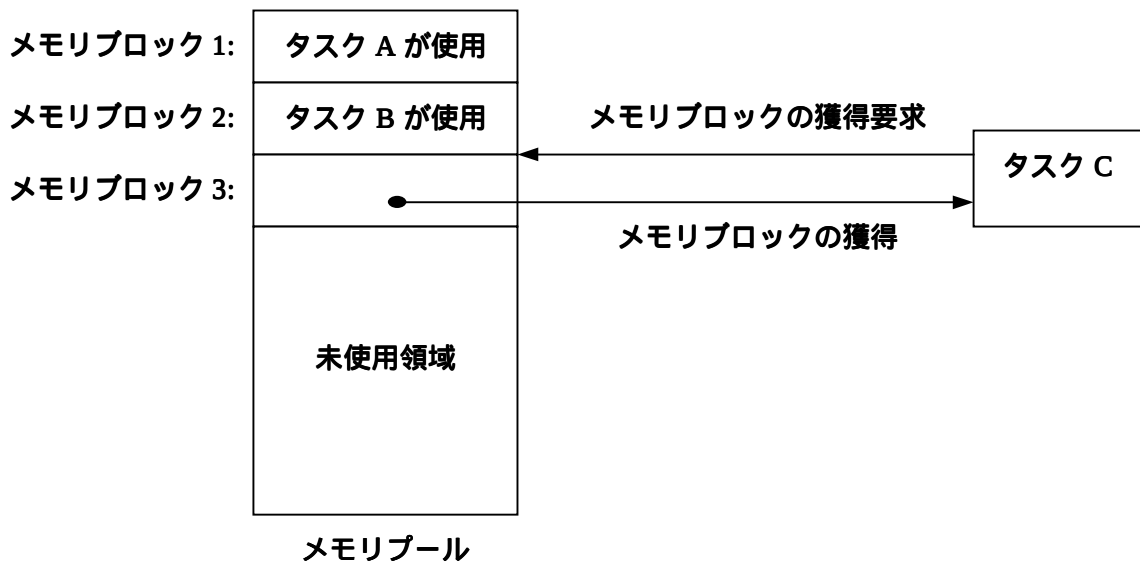


図 3.32 メモリプール管理

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

可変長メモリプール管理機能

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

MR30 では、メモリを 4 種類の固定長ブロックサイズで管理しています。

4 種類の各々のサイズは、ユーザーが獲得するメモリブロックの最大サイズから MR30 が計算します。メモリブロックの最大サイズは、コンフィグレーションファイルで指定します。

例

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

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

ユーザーが要求したメモリは、指定サイズをもとに MR30 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。

MR30 カーネルが提供する可変長メモリプール管理システムコールには次のものがあります。

- メモリブロックを獲得する (pget_blk)

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

4 種類のブロックサイズ(下記 a,b,c,d) は下記の計算式から算出されます。

$$\begin{aligned} a &= (((\text{max_memsize} + (X - 1)) / (X \times 8)) + 1) \times X \\ b &= a \times 2 \\ c &= a \times 4 \\ d &= a \times 8 \end{aligned}$$

max_memsize: コンフィグレーションファイルで指定した値

X: ブロック管理用データサイズ (8 バイト)

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

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

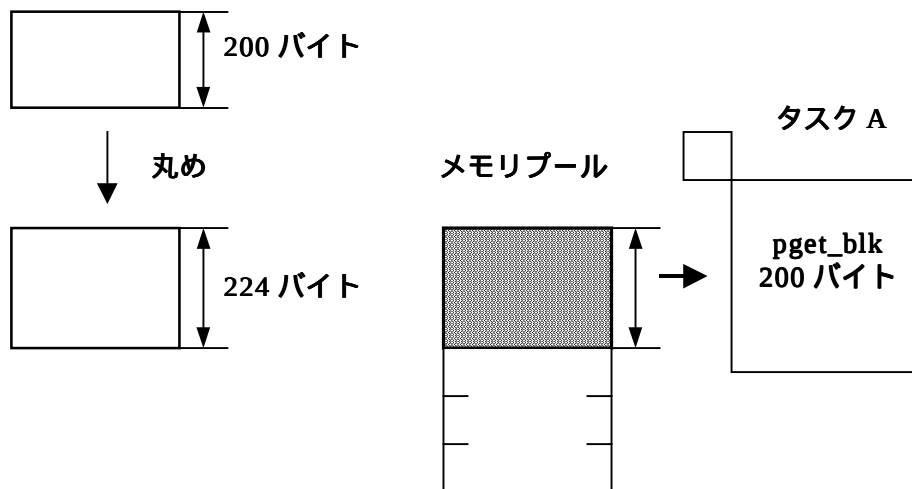


図 3.33 pget_blk 処理

- メモリブロックを解放する (rel_blk)
pget_blk で獲得したメモリブロックを解放します。

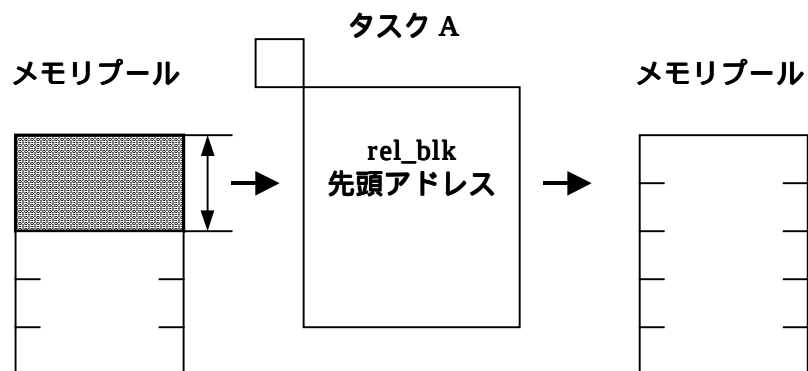


図 3.34 rel_blk 処理

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

3.5.10 時間管理機能

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

MR30 カーネルはシステムタイマとして M16C マイクロコンピュータの持つハードウェアタイマを一つ占有します。どのタイマを使用するかはコンフィグレーションファイルで設定します。

MR30 カーネルが提供する時間管理システムコールには次のものがあります。

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

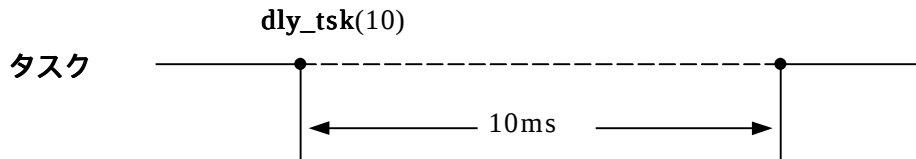


図 3.35 dly_tsk システムコール

- 待ち状態にタイムアウト値を指定すれば、一定時間待ち状態に移行します
タスクを待ち状態に移行するシステムコール⁴²にタイムアウトを指定することができます。システムコール名は、tslp_tsk、twai_flg、twai_sem、trcv_msg です。タイムアウトの指定時間が経過するまでに待ち解除条件が満たされない場合、エラーコード E_TMOUT を返し、待ち状態が解除されます。待ち解除条件が満たされた場合は、エラーコードは E_OK を返します。（図 3.36参照）
タイムアウトの基準時間は、MR30 のシステムクロックの時間を基準としています。

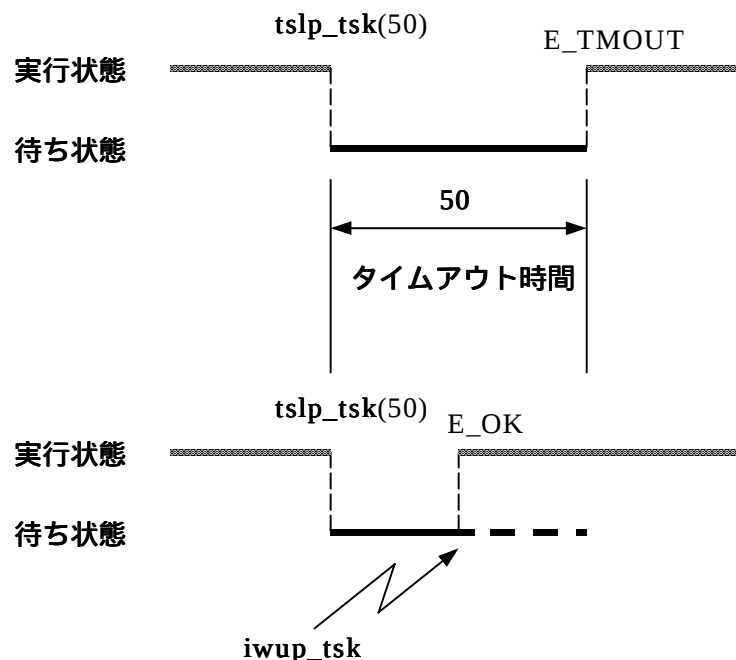


図 3.36 タイムアウト処理

⁴⁰ get_tim システムコール

⁴¹ set_tim システムコール

⁴² 強制待ち状態を除きます。

- システム時刻を設定する (set_tim)
- システム時刻の値を読みだす (get_tim)
システム時刻はリセット後のシステムクロック割り込みの発生回数をカウントし、48 ビットのデータで表します。
- 周期起動ハンドラの活性制御をおこなう (act_cyc)
周期起動ハンドラは一定間隔で起動するプログラムです。(図 3.37参照) システムクロック割り込みの発生回数をカウントすることによって、一定時間で起動します。周期起動ハンドラは、システムコールで活性状態を指定することによって制御されます。例えば、TCY_ON を指定して活性状態を OFF から ON に変更したり (図 3.38参照)、TCY_INI_ON を指定して、ハンドラのカウント値を初期化する (図 3.39参照) ことができます。
- 周期起動ハンドラの状態を参照する (ref_cyc)
対象周期起動ハンドラの活性状態やつぎの起動までの残り時間を参照します。
- アラームハンドラの状態を参照する (ref_alm)
対象アラームハンドラの起動までの残り時間を参照します。

なお、システムタイマは必須機能ではありません。したがって下記のシステムコールおよび時間管理機能を使用しなければ、タイマを MR30 用に占有する必要がありません。

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

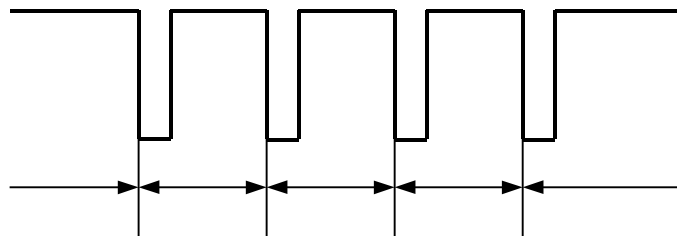


図 3.37 周期起動ハンドラ

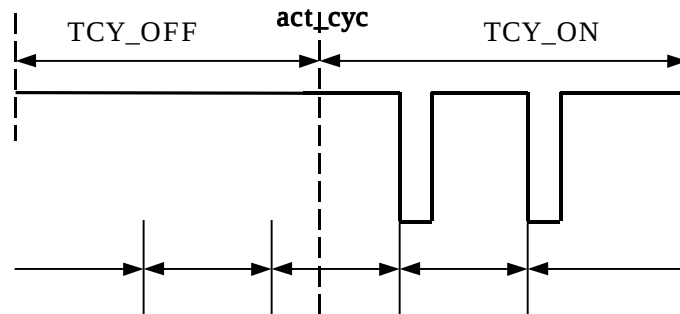


図 3.38 周期起動ハンドラ:活性状態 TCY_ON を指定

⁴³ set_tim, get_tim システムコール

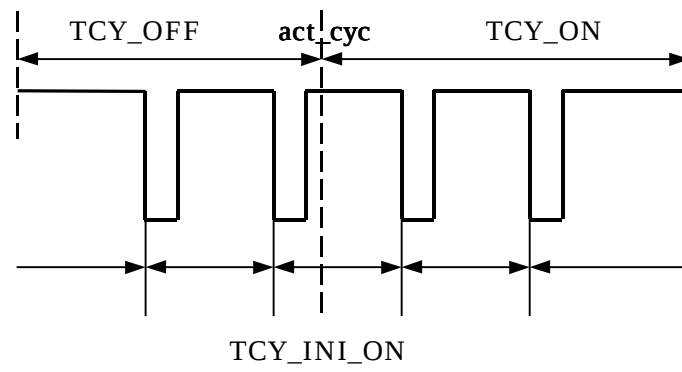


図 3.39 周期起動ハンドラ:活性状態 `TCY_INI_ON` を指定

3.5.11 バージョン管理機能

MR30 のバージョンを `get_ver` システムコールにより得ることができます。

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

- **メーカー名**
メーカーを示す番号
- **形式番号**
製品識別番号
- **仕様書バージョン**
μITRON 仕様であることを表す番号とこの製品のもとになった μITRON 仕様書のバージョン番号
- **製品バージョン**
MR30 のバージョン番号
- **製品管理情報**
製品のリリース番号、リリース日等の情報
- **MPU 情報**
M16C/60,30,20,10 シリーズマイクロコンピュータを示す番号
- **バリエーション記述子**
MR30 で利用できる機能番号

3.5.12 タスク、ハンドラから発行できるシステムコール一覧

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

表 3.2にその一覧を示します。

表 3.2 タスク、ハンドラから発行できるシステムコール一覧

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

システムコール	タスク	割り込みハンドラ	周期起動ハンドラ	アラームハンドラ
sig_sem	○	×	×	×
isig_sem	×	○	○	○
wai_sem	○	×	×	×
twai_sem	○	×	×	×
preq_sem	○	○	○	○
ref_sem	○	○	○	○
snd_msg	○	×	×	×
isnd_msg	×	○	○	○
rcv_msg	○	×	×	×
trcv_msg	○	×	×	×
prcv_msg	○	○	○	○
ref_mbx	○	○	○	○
pget_blf	○	○	○	○
rel_blf	○	○	○	○
ref_mpf	○	○	○	○
pget_blk	○	×	×	×
rel_blk	○	×	×	×
ref_mpl	○	○	○	○
ret_int	×	○ ⁴⁴	×	×
loc_cpu	○	×	×	×
unl_cpu	○	×	×	×
set_tim	○	○	○	○
get_tim	○	○	○	○
act_cyc	○	○	○	○
ref_cyc	○	○	○	○
ref_alm	○	○	○	○
get_ver	○	○	○	○
vrst_msg	○	×	×	×
vrst_blf	○	×	×	×
vrst_blk	○	×	×	×

⁴⁴ C 言語を用いて記述された割り込みハンドラからは発行できません。

第 4 章

アプリケーション作成手順概要

4.1 概要

MR30 のアプリケーションプログラムは一般的に以下に示す手順で開発します。

1. アプリケーションプログラムのコーディング

C 言語もしくはアセンブリ言語を用いてアプリケーションプログラムをコーディングします。
この時サンプルスタートアッププログラム "crt0mr.a30"もしくは "start.a30"、およびセクション定義ファイル "c_sec.inc"もしくは "asm_sec.inc"を環境変数"LIB30" の示すディレクトリからカレントディレクトリにコピーしておいてください。⁴⁵また必要があればスタートアッププログラム、セクション定義ファイルを修正してください。

2. コンフィグレーションファイル作成

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

3. コンフィグレータ実行

コンフィグレーションファイルからシステムデータ定義ファイル (sys_rom.inc、sys_ram.inc)、インクルードファイル (mr30.inc、id.h)およびシステム生成手順記述ファイル (makefile) を作成します。

4. システム生成

make⁴⁶コマンドを実行してシステムを生成します。

5. ROM 書き込み

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

図 4.1にシステム生成の詳細フローを示します。

⁴⁵ 標準のスタートアッププログラム"crt0mr.a30","start.a30"、およびセクション定義ファイル"c_sec.inc","asm_sec.inc"は環境変数"LIB30"で示されたディレクトリにあります。

⁴⁶ 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。

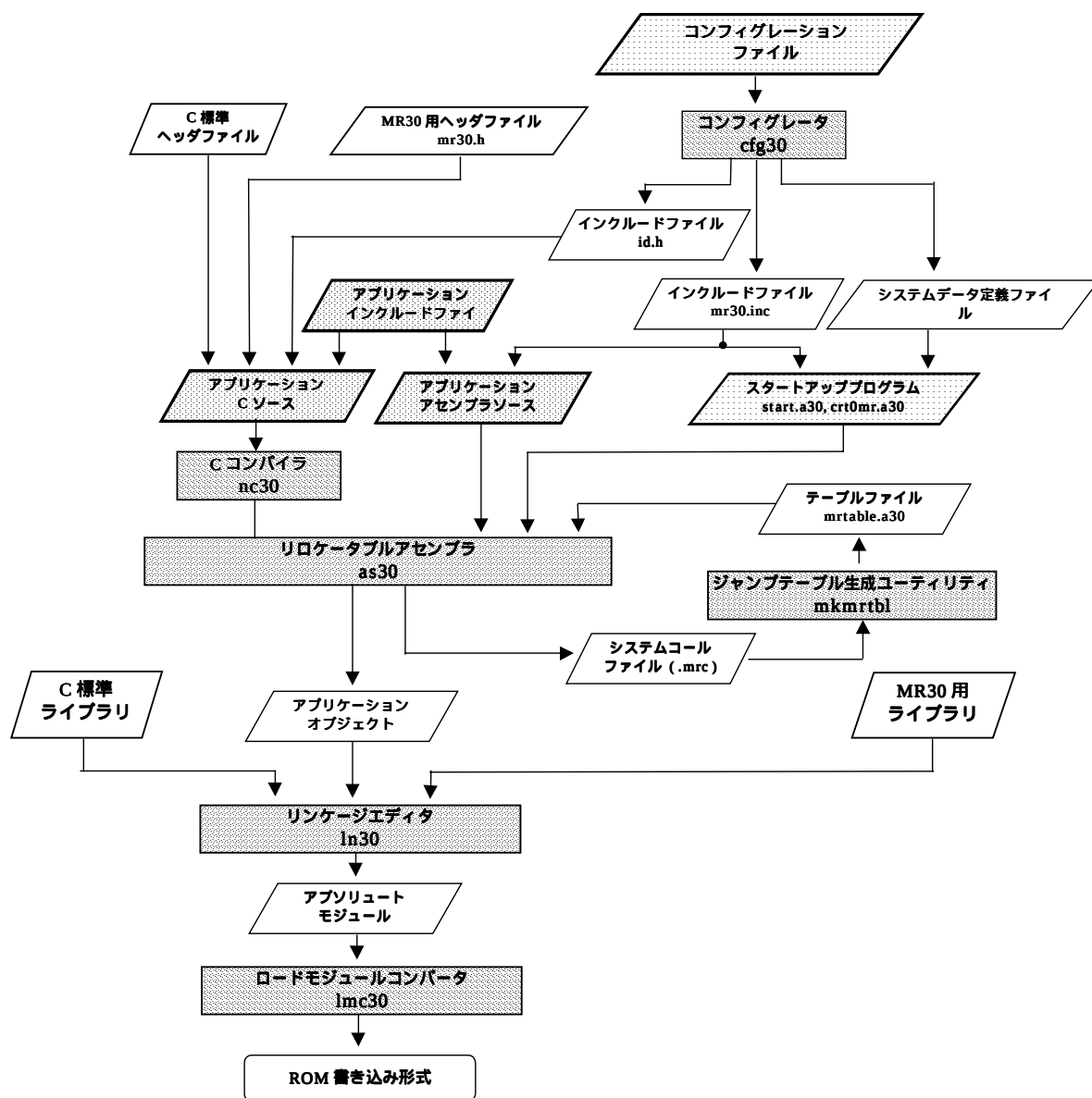


図 4.1 MR30 システム生成詳細フロー

4.2 開発手順例

この節では MR30 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にレーザー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 MR30 ライブラリのなかの以下の機能を利用します。

- sta_tsk()
タスク起動をおこないます。引き数は起動すべきタスクを選択するための ID 番号を与えます。ID 番号はコンフィグレータの生成するファイル"id.h"をインクルードすることにより名前 (文字列)でタスクを指定することができます。⁴⁷
- wai_flg()
イベントフラグが立つまで待ちます。この例ではセントロニクスインターフェースから1ページ分データがバッファに溜まるのを待つために使用しています。
- wup_tsk()
指定タスクを待ち状態から起床します。プリンタエンジンタスクを起動するために使用しています。
- slp_tsk()
タスクを実行状態から待ち状態にします。この例ではプリンタエンジンタスクをイメージ展開待ちにするため使用しています。
- iset_flg()
イベントフラグを立てます。この例では、1ページ分のデータ入力完了をイメージ展開タスクに知らせるために使用しています。

また、この時スタートアッププログラム"crt0mr.a30"、セクション定義ファイル"c_sec.inc"をカレントディレクトリにコピーしておいてください。例えば、

```
% cp $LIB30/crt0mr.a30 .
% cp $LIB30/c_sec.inc .
```

⁴⁷ すなわち、コンフィグレータがコンフィグレーションファイルに記述されている情報をもとに ID 番号を名前 (文字列) に置き換えます。

```
#include <mr30.h>
#include "id.h"

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

        printf("ビットマップ展開処理\n");
        wup_tsk(ID_printer); /* プリンタエンジンタスク起床 */
    }
}
void printer() /* プリンタエンジンタスク */
{
    while(1){
        slp_tsk();
        printf("プリンタエンジン動作\n");
    }
}
void sent_in() /* セントロニクスインターフェースハンドラ */
{
    /* セントロニクスインターフェースからの入力処理 */
    if ( /* 1ページ入力完了 */ )
        iset_flg(ID_pagein,setptn);
}
```

図 4.2 プログラム例

4.2.2 コンフィグレーションファイル作成

タスクのエントリーアドレスやスタックサイズなどを定義したコンフィグレーションファイルを作成します。

図 4.3にレーザービームプリンタシュミレーションプログラムのコンフィグレーションファイル (ファイル名 "lbp.cfg") を示します。

```
// System Definition
system{
    stack_size          = 1024;
    priority             = 5;
    system_IPL          = 4;
};
//System Clock Definition
clock{
    mpu_clock           = 20MHz;
    timer               = A0;
    IPL                 = 4;
    unit_time           = 10ms;
    initial_time        = 0:0:0;
};
//Task Definition
task[1]{
    entry_address       = main();
    stack_size          = 512;
    priority            = 1;
    initial_start       = ON;
};
task[2]{
    entry_address       = image();
    stack_size          = 512;
    priority            = 2;
};
task[3]{
    entry_address       = printer();
    stack_size          = 512;
    priority            = 4;
};
task[4]{
    entry_address       = idle();
    stack_size          = 256;
    priority            = 5;
};
//Eventflag Definition
flag[1]{
    name                = pagein;
};
//Interrupt Vector Definition
interrupt_vector[0x23]{
    os_int              = YES;
    entry_address       = sent_in();
};
```

図 4.3 コンフィグレーションファイル例

4.2.3 コンフィグレータ実行

コンフィグレータ `cfg30` を実行して、コンフィグレーションファイルからシステムデータ定義ファイル (`sys_rom.inc`, `sys_ram.inc`)、インクルードファイル (`mr30.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> cfg30 -mv lbp.cfg

MR30 system configurator V.3.30.01
Copyright 2003 RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED.
MR30 version ==> V.3.30 Release 1

A>
```

図 4.4 コンフィグレータ実行

4.2.4 システム生成

`make` コマンド⁴⁸を実行してシステムを生成します。

```
A> nmake -f makefile
as30 -F -Dtest=1 crt0mr.a30
nc30 -c task.c
ln30 @ln30.sub

A>
```

図 4.5 システム生成

4.2.5 ROM 書き込み

ロードモジュールコンバータ `lmc30` により、アブソリュートモジュールファイルを ROM 書き込み形式に変換し、ROM に書き込みます。もしくは、デバッガに読み込ませてデバッグを行います。

⁴⁸ `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があります。MR30 では、UNIX 標準もしくは UNIX 準拠の `make` コマンドのみ使用できます。MS-DOS を使用する場合は、UNIX 互換の `make` コマンド（例えば、マイクロソフト社製 C コンパイラ付属の `nmake` コマンド）を使用してください。UNIX 互換の `make` コマンド対応状況については、リリースノートを参照下さい。本章では、UNIX 互換の `nmake` 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 MR30 に登録するにはコンフィグレーションファイルに関数名を記述します。例えば関数名 "task()" をタスク ID 番号 3 で登録するには以下のようにおこないます。

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

2. ファイル先頭で必ずシステムディレクトリのなかの "mr30.h" とカレントディレクトリ内の "id.h" をインクルードしてください。すなわちファイルの先頭で以下の 2 行を必ず記述してください。

```
#include <mr30.h>  
#include "id.h"
```

3. タスク開始関数の戻り値はありません。したがって、void 型で宣言してください。

4. スタティック宣言をおこなった関数はタスクとして登録できません。

5. タスク開始関数の出口で ext_tsk()を記述する必要はありません。⁴⁹タスク開始関数から呼び出した関数でタスクを終了する場合は、ext_tsk()を記述してください。

6. システムコール関数の extern 宣言およびプロトタイプ宣言は記述しないでください。

7. タスクの開始関数を無限ループで記述することも可能です。

```
#include <mr30.h>  
#include "id.h"  
void task(void)  
{  
    /* 処理 */  
}
```

図 5.1 C 言語で記述したタスクの例

⁴⁹ MR30 では、#pragma TASK 宣言を行うことで、自動的に ext_tsk()で終了します。関数の途中で return 文により戻る場合も同様に ext_tsk()で終了処理をおこないます。

```

#include <mr30.h>

#include "id.h"

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

```

図 5.2 C 言語で記述した無限ループタスクの例

8. タスクを指定する場合はタスクの関数名に "ID_" を追加した文字列で指定してください⁵⁰。

```
wup_tsk(ID_main);
```

9. 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、コンフィグレーションファイルで定義したそれぞれの名前に "ID_" を追加した文字列で指定してください。

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

```

flag[1]{
    name    = abc;
};

```

このイベントフラグを指定するには以下のようにおこなってください。

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

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

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

11. タスクを `ter_tsk()` システムコールなどで終了した後で `sta_tsk()` システムコールで再起動した場合は、タスク自身は初期状態から開始しますが⁵¹ 外部変数、スタティック変数はタスクの開始にともなう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 MR30 が立ち上がる前に起動されるスタートアッププログラム (`crt0mr.a30`、`start.a30`) でのみおこないます。

12. MR30 システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。

13. 変数の記憶クラスについて

C 言語の変数は MR30 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

⁵⁰ コンフィグレータがタスクの ID 番号をタスクを指定するための文字列に変換するためのファイル "id.h" を生成します。すなわち、タスクの開始関数名に "ID_" を付加した文字列をそのタスクの ID 番号に変換するための `#define` 宣言を "id.h" で行います。

⁵¹ 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 OS 依存割り込みハンドラの記述方法

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

1. OS 依存割り込みハンドラは関数として記述します。⁵²
2. 割り込みハンドラ開始関数の戻り値および引き数は、必ず void 型で宣言してください。
3. ファイルの先頭で必ずシステムディレクトリのなかの"mr30.h" とカレントディレクトリ内の"id.h"をインクルードしてください。
4. 関数の最後に `ret_int` システムコールは記述しないで下さい。⁵³また、割り込みハンドラ関数から呼び出した関数のなかで割り込みハンドラを終了することはできません。
5. システムコール関数の `extern` 宣言およびプロトタイプ宣言は記述しないでください。
6. 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 <mr30.h>
#include "id.h"
void inthand(void)
{
    /* 処理 */
    iwup_tsk(ID_main);
}
```

図 5.3 C 言語で記述した割り込みハンドラの例

⁵² 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。

⁵³ OS 依存割り込みハンドラを `#pragma INTHANDLER` で宣言すると、自動的に `ret_int` システムコールを生成します。

5.1.3 OS 独立割り込みハンドラの記述方法

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

1. 割り込みハンドラ開始関数の戻り値および引き数は、必ず void 型で宣言してください。
2. OS 独立割り込みハンドラからは、システムコールは発行できません。
(注)システムコールを発行した場合は不正動作をするので十分注意して下さい。
3. スタティック宣言をおこなった関数は割り込みハンドラとしては登録できません。
4. OS 独立割り込みハンドラの中で多重割り込みを許可する場合は、必ず、OS 独立割り込みハンドラの割り込み優先レベルを、他の OS 依存割り込みハンドラの割り込み優先レベルより高くしてください。⁵⁴

```
#include <mr30.h>
#include "id.h"
void inthand(void)
{
    /* 処理 */
}
```

図 5.4 OS 独立割り込みハンドラの例

⁵⁴ OS 独立割り込みハンドラの割り込みレベル優先レベルを、OS 依存割り込みハンドラの割り込み優先レベルより低くしたい場合は、OS 独立割り込みハンドラを OS 依存割り込みハンドラの記述に変更してください。

5.1.4 周期起動ハンドラ、アラームハンドラの記述方法

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

1. 周期起動ハンドラおよびアラームハンドラは関数として記述します。⁵⁵
2. 関数の戻り値および引き数を、void 型で宣言してください。
3. ファイルの先頭で必ずシステムディレクトリのなかの"mr30.h"とカレントディレクトリ内の"id.h"をインクルードしてください。
4. スタティック宣言をおこなった関数は周期起動ハンドラおよびアラームハンドラとしては登録できません。
5. システムコール関数の extern 宣言およびプロトタイプ宣言は記述しないでください。
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 <mr30.h>
#include "id.h"

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

図 5.5 C 言語で記述した周期起動ハンドラの例

⁵⁵ ハンドラと関数名との対応は、コンフィグレーションファイルにより行います。

5.2 アセンブリ言語によるコーディング方法

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

5.2.1 タスクの記述方法

アセンブリ言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. ファイルの先頭で必ず "mr30.inc"をインクルードしてください。
2. タスクの開始アドレスを示すシンボルは外部シンボル宣言⁵⁶をおこなってください。
3. タスクは無限ループか ext_tsk システムコールで終了してください。

```
.INCLUDE mr30.inc ----- (1)
.GLB      task      ----- (2)

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

図 5.6 アセンブリ言語で記述した無限ループタスクの例

```
.INCLUDE mr30.inc
.GLB      task

task:
    ; 処理
    ext_tsk
```

図 5.7 アセンブリ言語で記述した ext_tsk で終了するタスクの例

4. タスク起動時のレジスタの初期値は、R0、PC、SB、FLG レジスタ以外は 0(ゼロ)です。
5. タスクを指定する場合はタスクの開始シンボル名に"ID_"を追加した文字列で指定してください。

⁵⁷

```
wup_tsk   #ID_task
```

6. 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、コンフィグレーションファイルで定義したそれぞれの名前に"ID_"を追加した文字列で指定してください。

例えば、コンフィグレーションファイルで以下のようにセマフォを定義した場合、

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

このセマフォを指定するには以下のようにおこなってください。

```
sig_sem   #ID_abc
```

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

```
act_cyc   #ID_cyc, #TCY_ON
```

⁵⁶ .GLB 疑似命令を使用してください。

⁵⁷ コンフィグレータがタスクの ID 番号を、タスクを指定するための文字列に変換するための命令をファイル"mr30.inc"に生成します。すなわち、タスクの開始シンボル名に"ID_"を付加した文字列をそのタスク ID 番号に変換するための EQU 宣言を"mr30.inc"でおこないます。

8. MR30 システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。⁵⁸

⁵⁸ このタスク ID 番号とタスク(プログラム)との対応づけは、コンフィグレーションファイルにより行います。

5.2.2 OS 依存割り込みハンドラの記述方法

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

1. ファイルの先頭で必ず"mr30.inc"をインクルードしてください。
2. 割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）⁵⁹をおこなってください。
3. ハンドラ内で使用するレジスタは、ハンドラの入口でセーブし、使用後復帰して下さい。
4. ret_int システムコールにて復帰してください。また、割り込みハンドラ関数から呼び出した関数のなかで割り込みハンドラを終了することはできません。

```

        .INCLUDE mr30.inc          -----(1)
        .GLB    inth              -----(2)

inth:
    ; 使用レジスタ退避              -----(3)
    iwup_tsk   #ID_task1
    :
    ;      処      理
    :

    ; 使用レジスタ復帰              -----(3)
    ret_int                                         -----(4)

```

図 5.8 OS 依存割り込みハンドラの例

⁵⁹ .GLB 疑似命令を使用してください。

5.2.3 OS 独立割り込みハンドラ記述方法

1. 割り込みハンドラの開始アドレスを示すシンボルは外部宣言 (グローバル宣言) して下さい。
2. ハンドラ内で使用するレジスタは入り口でセーブし、使用後復帰して下さい。
3. REIT 命令で終了してください。
4. OS 独立割り込みハンドラからは、システムコールは発行できません。
(注)システムコールを発行した場合は不正動作をするので十分注意して下さい。
5. OS 独立割り込みハンドラ内で多重割り込みを許可する場合は、OS 独立割り込みハンドラの割り込み優先レベルは、他の OS 依存割り込みハンドラの割り込み優先レベルより必ず高くして下さい。⁶⁰

```

        .GLB      inthand                ----- (1)

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

```

図 5.9 特定レベルの OS 独立割り込みハンドラの例

⁶⁰ OS 独立割り込みハンドラの割り込み優先レベルを、OS 依存割り込みハンドラの割り込み優先レベルより低くしたい場合は、OS 独立割り込みハンドラを OS 依存割り込みハンドラの記述に変更してください。

5.2.4 周期起動ハンドラ、アラームハンドラの記述方法

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

1. ファイルの先頭で必ず"mr30.inc"をインクルードしてください。
2. ハンドラの開始アドレスを示すシンボルは外部宣言 (グローバル宣言)⁶¹をおこなってください。
3. 周期起動ハンドラ、アラームハンドラは全て RTS 命令 (サブルーチンリターン命令) にて復帰してください。

例えば、

```
.INCLUDE      mr30.inc      ----- (1)
.GLB         cychand       ----- (2)

cychand:
    :
    ; ハンドラ処理
    :

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

図 5.10 アセンブリ言語で記述したハンドラの例

⁶¹ .GLB 疑似命令を使用してください。

5.3 INT 命令の使用について

MR30 では、INT 命令の割り込み番号を表 5.2に示すようにシステムコール発行のため予約しています。そのため、ユーザー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、32 から 47 以外の割り込みを使用して下さい。

表 5.2 割り込み番号の割り当て

割り込み番号	使用するシステムコール
32	タスクからのみ発行可能なシステムコール
33	タスク独立部からのみ発行可能なシステムコール タスク、タスク独立部の両方から発行可能なシステムコール
34	ret_int システムコール
35	dis_dsp システムコール
36	loc_cpu システムコール
37	ext_tsk システムコール
38	タスクからのみ発行可能なシステムコール
39	タスク独立部からのみ発行可能なシステムコール タスク、タスク独立部の両方から発行可能なシステムコール
40	拡張システムコール
41 から 47	拡張のため予約

5.4 レジスタバンクについて

MR30 では、タスク起動時のコンテキストは、レジスタバンク 0 を使用しています。

カーネル処理中にレジスタバンク切り替えは行いません。

プログラム誤動作の原因となりますので、以下の点にご注意ください。

- タスク内では、レジスタバンク切り替えは行わないで下さい。
- レジスタバンク切り替えを指定している割り込み同士が、多重に割り込まないようにして下さい。

5.5 割り込みについて

5.5.1 割り込みハンドラの種類

MR30 の割り込みハンドラには、OS 依存割り込みハンドラと OS 独立割り込みハンドラを定義しています。それぞれの割り込みハンドラの定義を以下に示します。

- OS 依存割り込みハンドラ
 - 以下の 2 つの条件のうち、どちらか一方を満たすものを OS 依存割り込みハンドラとして定義します。
 - ◆ システムコールを発行する割り込みハンドラ
 - ◆ システムコールを発行する割り込みハンドラが多重ではいる割り込みハンドラ
- OS 独立割り込みハンドラ
 - 以下の 2 つの条件の両方満たすものを OS 独立割り込みハンドラとして定義します。
 - ◆ システムコールを発行しない割り込みハンドラ
 - ◆ システムコールを発行する割り込みハンドラ (システムクロック割り込みハンドラ) が多重割り込みではいないもの

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

図 5.11 に、OS 割り込み禁止レベルを 3 にした場合の、OS 依存割り込みハンドラと OS 独立割り込みハンドラの関係を示します。

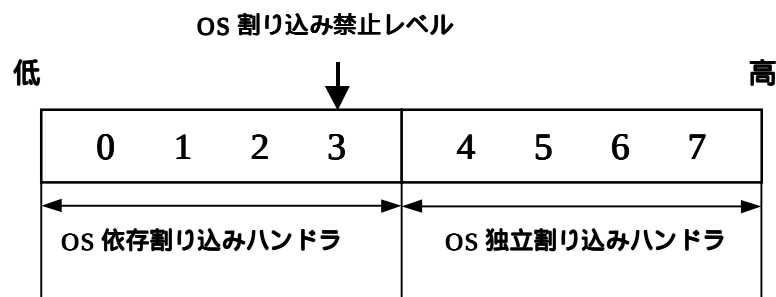


図 5.11 割り込みハンドラの IPL

5.5.2 ノンマスカブル割り込みについて

NMI 割り込みおよび監視タイマ割り込みは、必ず、OS 独立割り込みにしてください。OS 依存割り込みにした場合、プログラム誤動作の原因となりますので、ご注意ください。

⁶² system.IPL は、コンフィグレーションファイルで設定します。

5.5.3 割り込み制御方法

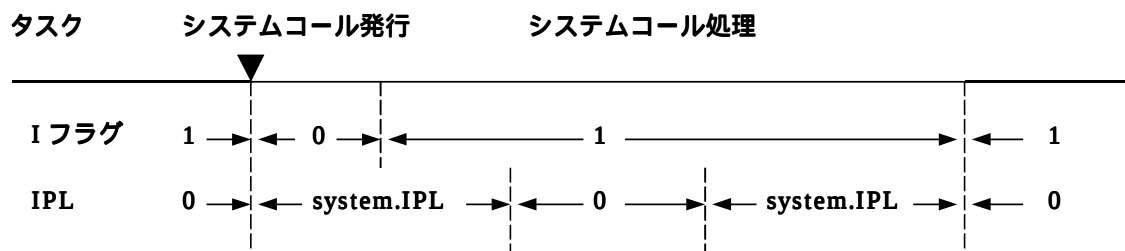
システムコール内の割り込み禁止/許可の制御は、IPL の操作により行っています。

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

図 5.12 に、システムコール内での割り込み許可フラグと IPL の状態を示します。

- タスクからのみ発行できるシステムコールの場合

- ・システムコール発行前の I フラグが 1 の場合



- ・システムコール発行前の I フラグが 0 の場合

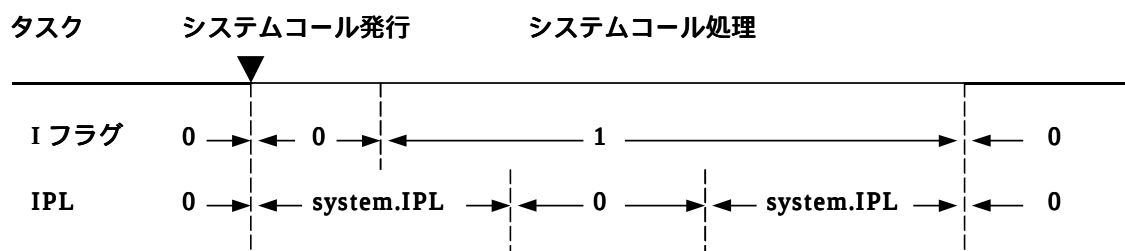
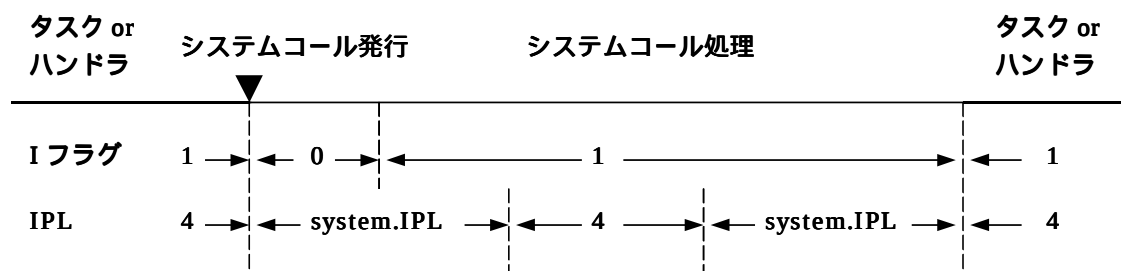


図 5.12 タスクからのみ発行できるシステムコール内での割り込み制御

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

・システムコール発行前の I フラグが 1 の場合



・システムコール発行前の I フラグが 0 の場合



図 5.13 タスク独立部から発行できるシステムコール内での割り込み制御

図 5.12、図 5.13に示すように割り込み許可フラグおよび IPL は、システムコール内で変化します。そのため、ユーザーアプリケーション内で割り込みを禁止したい場合、割り込み許可フラグおよび IPL の操作によって割り込みを禁止にする方法をとらないで下さい。

割り込みの制御は、以下に示す二つの方法で実現して下さい。

1. 禁止にしたい割り込みの割り込み制御レジスタ (SFR)を変更する。

2. loc_cpu ~ unl_cpu を使用する。

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

5.6 ディスパッチ遅延について

MR30 では、ディスパッチ遅延に関するシステムコールが 4 つあります。

- `dis_dsp`
- `ena_dsp`
- `loc_cpu`
- `unl_cpu`

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

1. ディスパッチ遅延中の実行タスクがプリエンプトされる場合

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

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

2. ディスパッチ遅延中の `isus_tsk`, `irsm_tsk`

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

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

3. ディスパッチ遅延中の `rot_rdq`, `irotd_rdq`

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

4. 注意事項

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

上記の状態遷移をまとめると表 5.3 のようになります。

表 5.3 dis_dsp,loc_cpu に関する割り込み、ディスパッチの状態遷移

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

5.7 初期起動タスクについて

MR30 では、システム起動時に READY 状態からスタートするタスクを指定できます。この指定はコンフィグレーションファイルで設定を行います。

設定方法の詳細については、100ページを参照して下さい。

5.8 MR30 スタートアッププログラムの修正方法

MR30 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.a30`
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.a30`
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.a30"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。

- リセット後のプロセッサの初期化
- C 言語の変数の初期化 (`crt0mr.a30` のみ)
- システムタイマの設定
- MR30 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 "LIB30"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。

なお、必要があれば以下の示す箇所を修正、あるいは追加して下さい。

- プロセッサモードレジスタの設定
プロセッサモードレジスタに、システムに合わせたプロセッサモードを設定して下さい。
(`crt0mr.a30` の 102 行目)
- ユーザーに必要な初期化プログラムの追加
ユーザーに必要な初期化プログラムを追加する場合は、C 言語用スタートアッププログラム
(`crt0mr.a30`)の 190 行目に追加して下さい。
標準入出力関数を使用しない場合は `crt0mr.a30` の 191-192 行目をコメントにしてください。

5.8.1 C 言語用スタートアッププログラム (crt0mr.a30)

図 5.14に C 言語用スタートアッププログラム crt0mr.a30 を示します。

```

1 ; *****
2 ;
3 ;     MR30 start up program for C language
4 ;     COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
5 ;     AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
6 ;
7 ; *****
8 ; $Id: crt0mr.a30,v 1.44 2003/08/22 13:00:34 muraki Exp $
9
10      .list    OFF
11      .include      c_sec.inc
12      .include      mr30.inc
13      .include      sys_rom.inc
14      .include      sys_ram.inc
15      .list    ON
16
17 ;-----
18 ; SBDATA area definition
19 ;-----
20      .glb      __SB__
21      .SB      __SB__
22
23 ;=====
24 ; Initialize Macro declaration
25 ;-----
26 N_BZERO .macro TOP_,SECT_
27      mov.b    #00H, R0L
28      mov.w    #(TOP_ & 0FFFFH), A1
29      mov.w    #sizeof SECT_, R3
30      sstr.b
31      .endm
32
33 N_BCOPY .macro FROM_,TO_,SECT_
34      mov.w    #(FROM_ & 0FFFFH), A0
35      mov.b    #(FROM_>>16), R1H
36      mov.w    #TO_, A1
37      mov.w    #sizeof SECT_, R3
38      smovf.b
39      .endm
40
41 BZERO .macro TOP_,SECT_
42      push.w   #sizeof SECT_ >> 16
43      push.w   #sizeof SECT_ & 0ffffh
44      pusha    TOP_>>16
45      pusha    TOP_ & 0ffffh
46
47      .glb     _bzero
48      jsr.a    _bzero
49      .endm
50 ;
51
52 BCOPY .macro FROM_,TO_,SECT_
53      push.w   #sizeof SECT_ >> 16
54      push.w   #sizeof SECT_ & 0ffffh
55      pusha    TO_>>16
56      pusha    TO_ & 0ffffh
57      pusha    FROM_>>16
58      pusha    FROM_ & 0ffffh
59
60      .glb     _bcopy
61      jsr.a    _bcopy
62      .endm
63
64 ;=====
65 ; Interrupt section start
66 ;-----
67      .glb     __SYS_INITIAL
68      .section      MR_KERNEL, CODE, ALIGN
69 __SYS_INITIAL:
70 ;-----
71 ; after reset, this program will start
72 ;-----
73      ldc      #(__Sys_Sp&0FFFFH), ISP ; set initial ISP
74
75      mov.b    #2H, 0AH

```

```

76      mov.b  #00, PMOD                ; Set Processor Mode Register
77      mov.b  #0H, 0AH
78      ldc    #00H, FLG
79      ldc    #(__Sys_Sp&0FFFFH), fb
80      ldc    #__SB__, sb
81
82 ; +-----+
83 ; |      MR RAM DATA 0(zero) clear          |
84 ; +-----+
85      N_BZERO MR_RAM_top, MR_RAM
86
87 ; +-----+
88 ; |      ISSUE SYSTEM CALL DATA INITIALIZE  |
89 ; +-----+
90 .if    __MR_NODBG == 1
91 .else
92      mov.b  #0, __DBG_MODE
93 .endif
94
95 ;=====
96 ; NEAR area initialize.
97 ;-----
98 ; bss zero clear
99 ;-----
100      N_BZERO bss_SE_top, bss_SE
101      N_BZERO bss_SO_top, bss_SO
102
103      N_BZERO bss_NE_top, bss_NE
104      N_BZERO bss_NO_top, bss_NO
105
106 ;-----
107 ; initialize data section
108 ;-----
109      N_BCOPY data_SEI_top, data_SE_top, data_SE
110      N_BCOPY data_SOI_top, data_SO_top, data_SO
111      N_BCOPY data_NEI_top, data_NE_top, data_NE
112      N_BCOPY data_NOI_top, data_NO_top, data_NO
113
114 ;=====
115 ; FAR area initialize.
116 ;-----
117 ; bss zero clear
118 ;-----
119      BZERO   bss_FE_top, bss_FE
120      BZERO   bss_F0_top, bss_F0
121
122 ;-----
123 ; Copy edata_E(0) section from edata_EI(0I) section
124 ;-----
125      BCOPY   data_FEI_top, data_FE_top, data_FE
126      BCOPY   data_FOI_top, data_F0_top, data_F0
127
128      ldc     #(__Sys_Sp&0FFFFH),    sp
129      ldc     #(__Sys_Sp&0FFFFH),    fb
130
131 ;=====
132 ; Initialize standard I/O
133 ;-----
134 ; .glb    _init
135 ; jsr.a   _init
136
137 ;-----
138 ; Set System IPL
139 ; and
140 ; Set Interrupt Vector
141 ;-----
142      mov.b  #0, R0L
143      mov.b  #__SYS_IPL, R0H
144      ldc    R0, FLG                ; set system IPL
145      ldc    #((__INT_VECTOR>>16)&0FFFFH), INTBH
146      ldc    #(__INT_VECTOR&0FFFFH), INTBL
147
148 .IF USE_TIMER
149 ; +-----+
150 ; |      System timer interrupt setting      |
151 ; +-----+
152      mov.b  #stmr_mod_val, stmr_mod_reg    ;set timer mode
153 ;      mov.b  #1H, 0AH
154 ;      bset   6, 07H
155      mov.b  #stmr_int_IPL, stmr_int_reg    ;set timer IPL

```

```

156 ;      bclr      6,07H
157 ;      mov.b     #0,0AH
158 ;      mov.w     #stmr_cnt,stmr_ctr_reg      ;set interval count
159 ;      or.b      #stmr_bit+1,stmr_start      ;system timer start
160 .ENDIF
161
162 ; +-----+
163 ; |      System timer initialize      |
164 ; +-----+
165 .IF      USE_SYSTEM_TIME
166     MOV.W     #__D_Sys_TIME_L, __Sys_time+4
167     MOV.W     #__D_Sys_TIME_M, __Sys_time+2
168     MOV.W     #__D_Sys_TIME_H, __Sys_time
169 .ENDIF
170
171 ; +-----+
172 ; |      User Initial Routine ( if there are )      |
173 ; +-----+
174 ;
175
176
177 ;      jmp      __MR_INIT      ; for Separate ROM
178
179 ; +-----+
180 ; |      Initalization of System Data Area      |
181 ; +-----+
182 ;      .section      MR_KERNEL, CODE, ALIGN
183
184     .glb      __init_sys, __init_tsk, __END_INIT
185 __MR_INIT:
186     jsr.w     __init_sys
187     jsr.w     __init_tsk
188
189 .if      __MR_TIMEOUT
190     .glb      __init_tout
191     jsr.w     __init_tout
192 .endif
193
194 .if      __NUM_FLG
195     .glb      __init_flg
196     jsr.w     __init_flg
197 .endif
198
199 .if      __NUM_SEM
200     .glb      __init_sem
201     jsr.w     __init_sem
202 .endif
203
204 .if      __NUM_MBX
205     .glb      __init_mbx
206     jsr.w     __init_mbx
207 .endif
208
209 .IF      ALARM_HANDLER
210     .glb      __init_alh
211     jsr.w     __init_alh
212 .ENDIF
213
214 .IF      CYCLIC_HANDLER
215     .glb      __init_cyh
216     jsr.w     __init_cyh
217 .ENDIF
218
219 ; Fixed Memory Pool
220 .if      __NUM_MPL
221     .glb      __init_mpl
222     jsr.w     __init_mpl
223 .endif
224
225 ; Variable Memory Pool
226 .IF      __MR_HEAPSIZE
227     .glb      __init_memblk, __init_vmpl
228
229     jsr.w     __init_vmpl
230     jsr.w     __init_memblk
231 .ENDIF
232
233 .if      __MR_NODBG == 1
234 .else
235     or.b      #80H, __DBG_MODE

```

```

236 .endif
237
238 __END_INIT:
239 ; +-----+
240 ; |      Start initial active task      |
241 ; +-----+
242     mov.w    #1, A0
243 __start_task:
244     lde.b    __D_INIT_START-1[A0], A1
245     jeq      __start_end
246     mov.w    A0, R1          ;move counter to R1
247     mov.w    A1, A0          ;set task id
248     mov.w    #0, R2          ;set start code
249     mov.w    #TFN_ISTA_TSK, R0 ;set func code
250     .ASSERT 'ista_tsk' >> ..file@.mrc
251     int      #33
252     mov.w    R1, A0
253     inc.w    A0
254     jmp.B    __start_task
255 __start_end:
256     .glb     __rdyq_search
257     jmp.W    __rdyq_search
258
259 ; +-----+
260 ; |      Define Dummy      |
261 ; +-----+
262     .glb     __SYS_DMY_INH
263 __SYS_DMY_INH:
264     reit
265
266 ; +-----+
267 ; |      exit() function      |
268 ; +-----+
269     .glb     _exit, $exit
270 _exit:
271 $exit:
272     jmp      _exit
273
274 ; +-----+
275 ; | Syscall exit routine(trace error code) |
276 ; +-----+
277     .glb     __sys_end
278 __sys_end:
279 .if      __MR_NODBG == 1
280 .else
281     .glb     __SYS_TRC
282     mov.w    R0, __SYS_TRC
283     mov.w    R1, __SYS_TRC
284     mov.w    R2, __SYS_TRC
285     mov.w    R3, __SYS_TRC
286     nop
287     nop
288 .endif
289     REIT
290
291 .if USE_TIMER
292 ; +-----+
293 ; |      System clock interrupt handler      |
294 ; +-----+
295     .SECTION      MR_KERNEL, CODE, ALIGN
296     .glb          __SYS_STMR_INH, __SYS_TIMEOUT
297     .glb          __DBG_MODE, __SYS_ISS
298 __SYS_STMR_INH:
299 ; process issue system call
300 .if      __MR_NODBG == 1
301 .else
302     tst.b    #1H, __DBG_MODE
303     jz       __proc_timer
304     pushm    R0, R1, R2, R3, A0
305     mov.w    __SYS_ISS, R0
306     mov.w    __SYS_ISS+2, R1
307     mov.w    __SYS_ISS+4, R2
308     mov.w    __SYS_ISS+6, R3
309     mov.w    __SYS_ISS+8, A0
310     cmp.w    #108, R0
311     jeq      __INT_33
312     cmp.w    #110, R0
313     jeq      __INT_33
314     cmp.w    #64, R0
315     jnc      __INT_33      ; if(R0 < 64) jmp __INT_33

```

```

316      cmp.w    #68,R0
317      jnc      __INT_33OR39      ; if((R0 >= 64)&&(R0 < 68)) jmp __INT_33OR39
318      cmp.w    #76,R0
319      jnc      __INT_33          ; if((R0 >= 68)&&(R0 < 76)) jmp __INT_33
320      jmp.b     __INT_39
321 __INT_33OR39:
322 .if      FAR_MSG_MODEL
323      int      #39
324 .else
325      int      #33
326 .endif
327      jmp.b     __END_ISS_SYS
328 __INT_39:
329      int      #39
330      jmp.b     __END_ISS_SYS
331 __INT_33:
332      int      #33
333 __END_ISS_SYS:
334      mov.w     R0,__SYS_ISS      ; error code
335      mov.w     R1,__SYS_ISS+2    ;
336      mov.w     R2,__SYS_ISS+4    ;
337      mov.w     R3,__SYS_ISS+6    ;
338      and.b     #0FEH,__DBG_MODE
339      popm      R0,R1,R2,R3,A0
340 .endif
341
342 __proc_timer:
343      pushm     R3,R0
344      stc       FLG,R0
345      mov.w     R0,__user_FLG      ;save user IPL
346      mov.b     #__SYS_IPL,R0H     ;change to system IPL
347      ldc       R0,FLG
348      fset      I
349
350      mov.w     __user_FLG,R3
351      and.w     #0ff00H,R3
352
353 ; System timer interrupt handler
354      _STMR_hdr
355      popm      R3,R0
356      ret_int
357 .endif
358
359      .end

```

図 5.14 C 言語用スタートアッププログラム (crt0mr.a30)

以下に標準スタートアッププログラム crt0mr.a30 の内容について説明します。

1. セクション定義ファイルを組み込みます。 [図 5.14の 12 行目]
2. MR30 用インクルードファイルを組み込みます。 [図 5.14の 13 行目]
3. システム ROM 領域定義ファイルを組み込みます。 [図 5.14の 14 行目]
4. システム RAM 領域定義ファイルを組み込みます。 [図 5.14の 15 行目]
5. リセット直後に起動される初期化プログラム __SYS_INITIAL です。 [図 5.14の 98 行目-246 行目]
 - ◆ システムスタックポインタの設定 [図 5.14の 99 行目]
 - ◆ プロセッサモードレジスタの設定 [図 5.14の 101 行目-103 行目]
 - ◆ FLG、SB、FB レジスタの設定 [図 5.14の 104 行目-108 行目]
 - ◆ C 言語の初期設定をおこないます。 [図 5.14の 128 行目-154 行目]
 - ◆ OS 割り込み禁止レベルの設定 [図 5.14の 163 行目-165 行目]
 - ◆ 割り込みベクタテーブルのアドレス設定 [図 5.14の 166 行目]
 - ◆ MR30 のシステムクロック割り込みの設定をおこないます。 [図 5.14の 171 行目-176 行目]
 - ◆ 標準入出力関数の初期化 [図 5.14の 191 行目-192 行目]
標準入出力関数を使用しない場合は、この行をコメントアウトしてください。
 - ◆ MR30 のシステム時刻の初期設定をおこないます。 [図 5.14の 181 行目-185 行目]

6. 必要があればアプリケーション固有の初期設定をおこないます。 [図 5.14の 190 行目]
7. MR30 が使用する RAM データの初期化をおこないます。 [図 5.14の 197 行目-241 行目]
8. スタートアップの終了を示すビットをセットします。 [図 5.14の 244 行目]
9. 初期起動タスクを起動します。 [図 5.14の 251 行目-255 行目]
10. システムクロックの割り込みハンドラです。 [図 5.14の 285 行目-296 行目]

5.9 メモリ配置方法

アプリケーションプログラムのデータのメモリ配置方法について説明します。
メモリ配置を設定するためには、MR30 が提供しているセクションファイルで設定します。
MR30 では、以下に示す 2 種類のセクションファイルが用意されています。

- `asm_sec.inc`
アセンブリ言語で、アプリケーション開発を行った場合に使用します。
各セクションの詳細については、87ページを参照して下さい。
- `c_sec.inc`
C 言語で、アプリケーション開発を行った場合に使用します。
`c_sec.inc` は、"`asm_sec.inc`"に C コンパイラ NC30 が生成するセクションを追加したものです。
各セクションの詳細については、87ページを参照して下さい。

ユーザーシステムに合わせて、セクション配置、開始アドレスの設定を変更して下さい。
セクションファイルの変更方法を以下に示します。

例

プログラムセクションの先頭をF0000H番地からF1000H番地に変更したい場合

```
.section      program
.org      0F0000H ; この部分を F1000H に修正

↓

.section      program
.org      0F1000H ;
```

5.9.1 start.a30 のセクションの配置

アセンブリ言語用サンプルスタートアッププログラム"start.a30"のセクション配置は、"asm_sec.inc"で定義しています。

セクションの再配置が必要な場合は、"asm_sec.inc"を編集してください。

以下に、サンプルセクション定義ファイル"asm_sec.inc"で定義している各セクションについて説明します。

- **MR_RAM_DBG セクション**
MR30 のデバッグ機能に必要な RAM データが入ったセクションです。
このセクションは、必ず、内蔵 RAM 領域に配置してください。
- **MR_RAM セクション**
MR30 のシステム管理データで、アブソリュートアドレッシングで参照する RAM データが入るセクションです。
このセクションは、必ず 0 ~ 0FFFFFFH(near 領域)以内に配置して下さい。
- **stack セクション**
各タスクのユーザスタック、およびシステムスタックのセクションです。
このセクションは、必ず 0 ~ 0FFFFFFH(near 領域)以内に配置して下さい。
- **MR_HEAP セクション**
可変長メモリプールが格納されるセクションです。
- **MR_KERNEL セクション**
MR30 カーネルプログラムを格納するセクションです。
- **MR_CIF セクション**
MR30 用 C 言語インターフェイスライブラリを格納するセクションです。
- **MR_ROM セクション**
MR30 カーネルが参照するタスクの開始番地などのデータを格納するセクションです。
- **program セクション**
ユーザプログラムを格納するセクションです。
このセクションは MR30 カーネルは全く使用していません。したがって、ユーザーで自由に使用することができます。
- **INTERRUPT_VECTOR セクション**
- **FIX_INTERRUPT_VECTOR セクション**
割り込みベクタを格納するセクションです。このセクションの開始番地は使用する M16C ファミリ機種により異なります。サンプルスタートアッププログラムの番地は M16C/60 シリーズ用のものです。他のグループを使用する場合は変更してください。

5.9.2 crt0mr.a30 のセクション配置

C 言語用サンプルスタートアッププログラム"crt0mr.a30"のセクション配置は、"c_sec.inc"で定義しています。セクションの再配置が必要な場合は、"c_sec.inc"を編集してください。

サンプルセクション定義ファイル"c_sec.inc"で定義しているセクションは、アセンブリ言語用スタートアッププログラムのセクション配置"asm_sec.inc"に、以下のセクションを定義したものです。

- data_SE セクション
- bss_SE セクション
- data_SO セクション
- bss_SO セクション
- data_NE セクション
- bss_NE セクション
- data_NO セクション
- bss_NO セクション
- rom_NE セクション
- rom_NO セクション
- data_FE セクション
- bss_FE セクション
- data_FO セクション
- bss_FO セクション
- rom_FE セクション
- rom_FO セクション
- data_SEI セクション
- data_SOI セクション
- data_NEI セクション
- data_NOI セクション
- data_FEI セクション
- data_FOI セクション

これらのセクションは、NC30 が生成するセクションです。アセンブリ言語用セクションファイルでは、このセクションは定義されていません。

詳細は、NC30 のマニュアルを参照してください。

サンプルスタートアッププログラムのセクションの配置を図 5.15に示します。

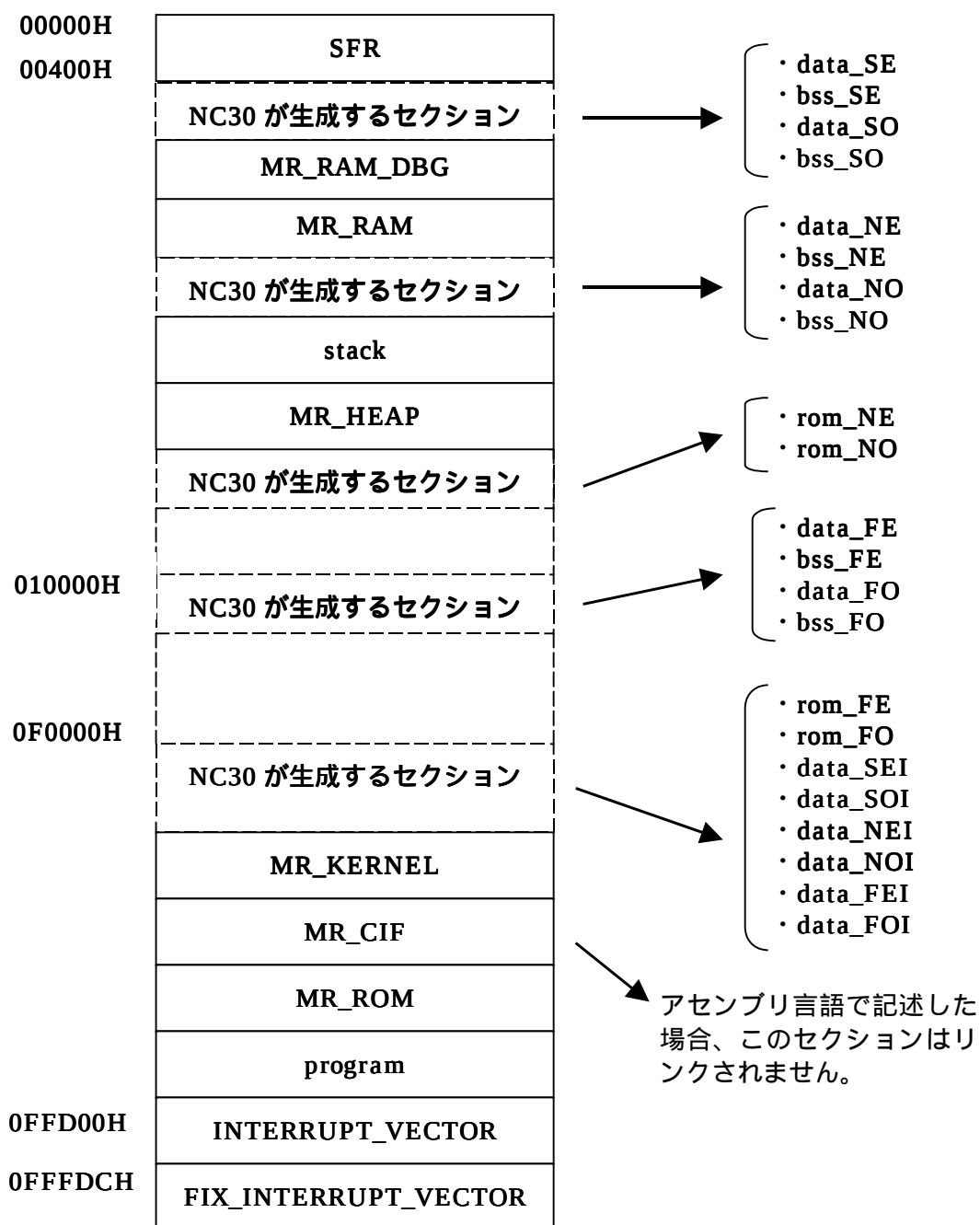


図 5.15 C 言語スタートアッププログラムのセクション配置

5.10 機種別の注意事項

5.10.1 M16C/62 グループをご使用の場合

- メモリ拡張機能をメモリ空間拡張モード 1(メモリ空間=1.2M)でご使用の場合
MR30 のカーネル(MR_KERNEL セクション)を 30000H から FFFFFH 番地以内に配置して下さい。
- メモリ拡張機能をメモリ空間拡張モード 2(メモリ空間=4M)でご使用の場合
MR30 のカーネル(MR_KERNEL セクション)を 3C0000H から 3FFFFFFH 番地以内に配置して下さい。

5.10.2 M16C/6N グループをご使用の場合

スタートアッププログラム中の下記に示す処理を MR30 のシステムタイマ設定箇所を追加して下さい。
(MR30 のシステムタイマ設定箇所は、"インストール先ディレクトリ¥LIB30"ディレクトリ内のスタートアッププログラム crt0mr.a30 では 160 行目、start.a30 では、73 行目になります。)

なお、周辺機能クロック選択レジスタ(25E 番地---0 ビット目)で 1 分周モードを設定する場合は下記の処理の追加は必要ありません。

```

;+-----+
;| System timer interrupt setting |
;+-----+
      mov.b      #stmr_mod_val,stmr_mod_reg      ; set timer mode
; mov.b      #1H,0AH
; bset      6,07H
      mov.b      #stmr_int_IPL,stmr_int_reg      ; set timer IPL
; bclr      6,07H
; mov.b      #0,0AH
      mov.w      #stmr_cnt_stmr_ctr_reg          ; set interval count

      mov.b      stmr_mod_reg,R0L                <---- 追加
      and.b      #0C0H,R0L                      <---- 追加
      jnz        __MR_SYSTIME_END                <---- 追加
      mov.w      #stmr_cnt/2,stmr_ctr_reg        <---- 追加
__MR_SYSTIME_END:                             <---- 追加

      or.b       #stmr_bit+1,stmr_start

```

第 6 章

コンフィグレータの使用方法

6.1 コンフィグレーションファイルの作成方法

アプリケーションプログラムのコーディング、スタートアッププログラムの修正が終わると、そのアプリケーションプログラムを MR30 システムに登録する必要があります。これをおこなうのがコンフィグレーションファイルです。

6.1.1 コンフィグレーションファイル内の表現形式

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

コメント文

'/'から行の終わりまではコメント文とみなし、処理の対象になりません。

文の終わり

';'で文を終わります。

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。⁶³

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'O'もしくは'o'を付加します。

4. 2 進数

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

表 6.1 数値表現例

16 進数	0xf12
	0Xf12
	0a12h
	0a12H
	12h
	12H
10 進数	32
8 進数	017
	17o
	17O
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 6.2に示します。

⁶³ 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 6.2 演算子

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

数値の例を以下に示します。

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

シンボル

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

シンボルの例を以下に示します。

- _TASK1
- IDLE3

関数名

関数名は数字、英大文字、英小文字、'_' (アンダースコア)、'\$' (ドル記号)より構成される数字以外の文字で始まり、'()'で終わる文字列で表されます。

C 言語で記述した関数名の例を以下に示します。

- main()
- func()

アセンブリ言語で記述した場合はモジュールの先頭ラベルを関数名とします。

周波数

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

周波数の例を以下に示します。

- 16MHz
- 8.1234MHz

なお、周波数は'.'で始まってはいけません。

時間

時間は数字と'.' (ピリオド) から構成され'ms'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。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。

- 10ms
- 10.5ms

なお時間は'.' (ピリオド)で始まってはいけません。

時刻

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

- 23 : 0x02 : 100B

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

6.1.2 コンフィグレーションファイルの定義項目

コンフィグレーションファイルでは以下の項目⁶⁴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
- タスク定義
- イベントフラグ定義
- セマフォ定義
- メールボックス定義
- 固定長メモリープール定義
- 可変長メモリープール定義
- 周期起動ハンドラ定義
- アラームハンドラ定義
- 割り込みベクタ定義

【システム定義】

<< 形式 >>

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

<< 内容 >>

1. システムスタックサイズ (バイト)

【 定義形式 】 数値

【 定義範囲 】 1 以上

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

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

MR30 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。⁶⁵

⁶⁴ 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。

⁶⁵ MR30 の優先度は、値が大きいほど優先度は低くなります。

3. メッセージサイズ

【 定義形式 】 数値

【 定義範囲 】 16 or 32

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

4. OS 割り込み禁止レベル

【 定義形式 】 数値

【 定義範囲 】 0 ~ 7

システムコール内での IPL の値、すなわち OS 割り込み禁止レベルを設定します。⁶⁶

5. タイムアウト

【 定義形式 】 シンボル

【 定義範囲 】 YES or NO

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

6. タスクポーズ

【 定義形式 】 シンボル

【 定義範囲 】 YES or NO

PD30 の OS デバッグ機能であるタスクポーズ機能をご使用になる場合は、YES を設定し、使用していない場合は、NO を設定してください。

【システムクロック定義】

<< 形式 >>

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

<< 内容 >>

1. MPU のクロック (MHz)

【 定義形式 】 周波数

【 定義範囲 】 なし

M16C の MPU 動作クロックの周波数を MHz 単位で定義します。

⁶⁶ 0 を定義した場合は、システムクロック割り込みハンドラや OS 依存割り込みハンドラが全く使用できなくなります。

2. システムクロックに使用するタイマ

【 定義形式 】 シンボル

- 【 定義範囲 】
- ・ M16C/60 シリーズ
A0 ~ A4, B0 ~ B5, OTHER, NOTIMER
 - ・ M16C/30 シリーズ
A0 ~ A2, B1 ~ B2, OTHER, NOTIMER
 - ・ M16C/20 シリーズ
A0 ~ A7, B0 ~ B5, X0 ~ X2, OTHER, NOTIMER
 - ・ M16C/10 シリーズ
OTHER, NOTIMER

システムクロックに使用するハードウェアタイマを定義します。

上記のシリーズごとのタイマの設定範囲はコンフィグレータではチェックしていませんので、マイコンにないタイマを指定しないように注意する必要があります。

M16C/10 シリーズでタイマーを指定する場合は OTHER を指定して、使用するタイマの設定をスタートアップで行ってください。

システムクロックを使用しない場合は、"NOTIMER"を定義します。

3. システムクロック割り込み優先レベル

【 定義形式 】 数値

【 定義範囲 】 1 ~ (システム定義の OS 割り込み禁止レベル)

システムクロック用タイマ割り込みの優先レベルを定義します。1 ~ OS 割り込み禁止レベルまでの値を設定して下さい。

システムクロックの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられません。

4. システムクロックの単位時間 (ms)

【 定義形式 】 時間

【 定義範囲 】 $\frac{1}{\text{MPUのクロック}(mpu_clock)}$ ~ $\frac{32 \times 65535}{\text{MPUのクロック}(mpu_clock)}$

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

最小値はユーザーズマニュアルの計算式で計算できますが、計算値が 0.001ms より小さくなる場合は、最小値は 0.001ms となります。コンフィグレーションファイル中で 0.001ms より小さい値を設定すると、コンフィグレータが以下のエラーを返します。

```
cfg30 Error : illegal clock.unit_time --> <0> near line YY (smp.cfg)
```

なお、コンフィグレーションファイルで時間間隔を設定する場合には、OS のシステムクロック割り込みハンドラの処理時間よりも大きくする必要があります。この時間は、マイコンの種類や動作条件に依存します。

リファレンスマニュアルに、システムクロック割り込みの処理時間の計算方法を記載していますので、その計算値を参考にしてください。

5. システム時刻の初期値

【 定義形式 】 時刻

【 定義範囲 】 0:0:0 ~ 0x7FFF:0xFFFF:0xFFFF

システム時刻の初期値を定義します。システム時刻を用いた機能 (set_tim、get_tim、アラームハンドラ)を使用しない場合は、この項目を設定する必要はありません。本定義をおこなわないことにより

自動的にシステムクロック割り込みハンドラの処理が最適化されます。ただし、デフォルトコンフィグレーションファイルでデフォルト値を定義した場合、最適化は行いませんので注意してください。

【最大項目数定義】

この定義は、別 ROM 化⁶⁷を行う場合のみ設定する項目です。

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。

<< 形式 >>

```
// Max Definition
maxdefine{
    max_task      = 最大タスク定義数;
    max_flag      = 最大イベントフラグ定義数;
    max_mbx = 最大メールボックス定義数;
    max_sem = 最大セマフォ定義数;
    max_mpl = 最大固定長メモリブール定義数;
    max_cyh = 最大周期起動ハンドラ定義数;
    max_alh = 最大アラームハンドラ定義数;
};
```

<< 内容 >>

1. 最大タスク定義数

【 定義形式 】数値

【 定義範囲 】1 ~ 255

タスク定義の最大数を定義します。

2. 最大イベントフラグ定義数

【 定義形式 】数値

【 定義範囲 】1 ~ 255

イベントフラグ定義の最大数を定義します。

3. 最大メールボックス定義数

【 定義形式 】数値

【 定義範囲 】1 ~ 255

メールボックス定義の最大数を定義します。

4. 最大セマフォ定義数

【 定義形式 】数値

【 定義範囲 】1 ~ 255

セマフォ定義の最大数を定義します。

⁶⁷ 別 ROM 化の詳細については、136ページを参照してください。

5. 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 ~ 255

固定長メモリプール定義の最大数を定義します。

6. 最大周期起動ハンドラ定義数

【 定義形式 】 数値

【 定義範囲 】 1 ~ 255

周期起動ハンドラ定義の最大数を定義します。

7. 最大アラームハンドラ定義数

【 定義形式 】 数値

【 定義範囲 】 1 ~ 255

アラームハンドラ定義の最大数を定義します。

【タスク定義】

<< 形式 >>

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

ID 番号は 1 ~ 255 の範囲でなければなりません。ID 番号は省略可能です。省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

タスク ID 番号ごとに以下の定義をおこないます。

1. タスク開始アドレス

【 定義形式 】 シンボル、または、関数名

【 定義範囲 】 なし

タスクの入り口アドレスを定義します。C 言語で記述したときはその関数名の最後に ()をつけるか、先頭に_をつけます。

なお、ここで定義した関数名は、id.h ファイルに以下の宣言文が出力されます。

```
#pragma TASK 関数名
```

2. ユーザースタックサイズ (バイト)

【 定義形式 】 数値

【 定義範囲 】 6 以上

タスクごとのユーザースタックサイズを定義します。ユーザースタックとは、各々のタスクが使用するスタック領域です。MR30 ではユーザー用のスタック領域をタスクごとに割り当てる必要があります。最低で 6 バイトが必要です。

3. タスクの初期優先度

【 定義形式 】 数値

【 定義範囲 】 1 ~ (システム定義の優先度の最大値)

タスクの起動時の優先度を定義します。

MR30 の優先度は、値が小さいほど、優先度としては高くなります。

4. 使用するレジスタ

【 定義形式 】 シンボル [,シンボル,.....]

【 定義範囲 】 R0,R1,R2,R3,A0,A1,SB,FB から選択

タスクで使用するレジスタを定義します。MR30 では、ここで定義されたレジスタをコンテキストとして扱います。タスク起動時に、タスク起動コードが R1 レジスタに設定されますので、R1 レジスタは、必ず指定して下さい。ただし、タスクをアセンブリ言語で記述する場合のみ使用レジスタが選択可能です。C 言語で記述する場合、全レジスタを選択してください。

なお、レジスタを選択する場合、各タスクで使用しているシステムコールのパラメータを格納するレジスタについては、全て選択してください。

MR30 カーネル内では、レジスタバンク切り替えは行いません。

本定義を省略した場合、全レジスタが選択されたものとします。

5. 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。

初期起動タスクのタスク起動コードは 0 です。

【イベントフラグ定義】

この定義は、イベントフラグ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は 1 ~ 255 の範囲でなければなりません。ID 番号は省略可能です。省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

イベントフラグ ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でイベントフラグを指定する時の名前を定義します。

【セマフォ定義】

この定義は、セマフォ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

セマフォ ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でセマフォを指定する時の名前を定義します。

2. セマフォカウンタ初期値

【 定義形式 】数値

【 定義範囲 】0 ～ 32767

セマフォカウンタの初期値を定義します。

【メールボックス定義】

この定義は、メールボックス機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

メールボックス ID 番号ごとに以下の項目の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

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。

2. 最大メッセージ数

【 定義形式 】数値

【 定義範囲 】0 ～ 16383

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

【固定長メモリプール定義】

この定義は、固定長メモリプール機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は、1～255 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

メモリプール ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメモリプールを指定する時の名前を指定します。

2. 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 (asm_sec.inc あるいは c_sec.inc) にて配置を行って下さい。

定義しない場合は、MR_RAM セクションに配置します。

3. ブロック数

【 定義形式 】数値

【 定義範囲 】1 ~ 16

メモリプールのブロック総数を定義します。

4. サイズ(バイト)

【 定義形式 】数値

【 定義範囲 】1 ~ 65535

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

【可変長メモリプール定義】

この定義は、可変長メモリプール機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は、1 を指定して下さい。

MR_HEAP セクションに配置されます。

<< 内容 >>

1. 確保するメモリブロックサイズの最大値 (バイト)

【 定義形式 】数値

【 定義範囲 】1 ~ 65520

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

2. メモリプールのサイズ (バイト)

【 定義形式 】 数値

【 定義範囲 】 16 ~ 524288

メモリプールのサイズを指定します。

MR30 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。この 4 種類のブロックサイズ(下記 a,b,c,d)は、下記の計算式から算出されます。

$$\begin{aligned} a &= ((\text{max_memsize} + (X - 1) / (X \times 8)) + 1) \times X \\ b &= a \times 2 \\ c &= a \times 4 \\ d &= a \times 8 \end{aligned}$$

X: ブロック管理用データサイズ(1 ブロックあたり 8 バイト)

可変長メモリプールは、メモリブロックを管理する領域として 1 ブロックあたり 8 バイトの領域が必要となります。このため、max_memsize で指定したサイズのメモリブロックを獲得するためには、max_memsize+8 の結果が収まる上記 a,b,c,d いずれかのブロックサイズ以上の値をメモリプールのサイズに指定しなければなりません。

【周期起動ハンドラ定義】

この定義は、周期起動ハンドラ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は 1 ~ 255 の範囲でなければなりません。ID 番号は省略可能です。省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

周期起動ハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 周期間隔

【 定義形式 】 数値

【 定義範囲 】 1 ~ 32767

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

2. モード

【 定義形式 】 シンボル

【 定義範囲 】 TCY_OFF または TCY_ON

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

- ◆ TCY_OFF

act_cyc システムコールを発行することで動作を始めるモード。

◆ TCY_ON

システムの立ち上げと同時に動作を始めるモード。

3. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

周期起動ハンドラの開始アドレスを定義します。

なお、ここで定義した関数名は、id.h ファイルに以下の宣言文が出力されます。

```
#pragma CYCHANDLER 関数名
```

【アラームハンドラ定義】

この定義は、アラームハンドラ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

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

ID 番号は 1 ~ 255 の範囲でなければなりません。ID 番号は省略可能です。省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

アラームハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 起動時刻

【 定義形式 】時刻

【 定義範囲 】0:0:0 ~ 0x7FFF:0xFFFF:0xFFFF

アラームハンドラの起動時刻を定義します。

2. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

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

```
#pragma ALMHANDLER 関数名
```

【割り込みベクタ定義】

<< 形式 >>

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

割り込みベクタ番号は 0～63、247～255 の範囲まで記述できます。

ただし、そのベクタ番号が有効か否かは使用しているマイクロコンピュータに依存します。

コンフィグレータは、ここで指定した割り込みの割り込み制御レジスタ (IPL 等) や、割り込み要因等の初期設定のコードは生成しません。初期設定はスタートアップファイル中もしくは、開発されるアプリケーションにあわせて作成していただく必要があります。

【注意】

- コンフィグレーションファイルからレジスタバンク 1 の指定は行えません。

C 言語ソースファイル内の "id.h" のインクルード命令の後で、"#pragma INTERRUPT/B" 宣言を記述してください。

```
例) #include <mr30.h>
     #include "id.h"
     #pragma INTERRUPT/B OS 独立割り込みハンドラ関数名
```

- C 言語でレジスタバンク 1 のレジスタを使った OS 依存割り込みハンドラの記述はできません。

アセンブリ言語のみ記述することができます。アセンブリ言語で記述する場合、割り込みハンドラの入口と出口を以下に示すように記述してください。

(ret_int システムコールを発行する前に必ず B フラグをクリアしてください。)

```
例) interrupt:
      fset      B
          :
      fclr      B
      ret_int
```

MR30 カーネル内では、レジスタバンク切り替えは行いません。

NMI 割り込み、監視タイマ割り込みは、OS 依存割り込みで使用しないでください。

<< 内容 >>

1. OS 依存割り込みハンドラ

【 定義形式 】 シンボル

【 定義範囲 】 YES または NO

ハンドラが OS 依存割り込みハンドラかどうかを定義します。OS 依存割り込みハンドラであれば YES を、OS 独立割り込みハンドラであれば NO を定義して下さい。

YES を定義した場合、id.h ファイルに以下の宣言文を出力します。

```
#pragma INTHANDLER 関数名
```

また、NO を定義した場合、id.h ファイルに以下の宣言文を出力します。

```
#pragma INTERRUPT 関数名
```

2. 開始アドレス

【 定義形式 】 シンボルまたは関数名

【 定義範囲 】 なし

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

以下に固定ベクタの割り込み要因とベクタ番号を以下に示します。可変ベクタについてはご使用になっているマイコンのハードウェアマニュアルを参照してください。

表 6.3 M16C/60 での固定ベクタ割り込み要因とベクタ番号との対応

割り込み要因	割り込みベクタ番号	セクション名
未定義命令	247	FIX_INTERRUPT_VECTOR
オーバーフロー	248	FIX_INTERRUPT_VECTOR
BRK 命令	249	FIX_INTERRUPT_VECTOR
アドレス一致	250	FIX_INTERRUPT_VECTOR
シングルステップ	251	FIX_INTERRUPT_VECTOR
監視タイマ	252	FIX_INTERRUPT_VECTOR
DBC(使用禁止)	253	FIX_INTERRUPT_VECTOR
NMI	254	FIX_INTERRUPT_VECTOR
リセット	255	FIX_INTERRUPT_VECTOR

6.1.3 コンフィグレーションファイル例

以下にコンフィグレーションファイルの例を示します。

```

1 //*****
2 //
3 //      Copyright 2003 RENESAS TECHNOLOGY CORPORATION
4 //      AND RENESAS SOLUTIONS CORPORATION
5 //
6 //      MR30 System Configuration File.
7 //
8 //*****
9 // System Definition
10 system{
11     stack_size           = 0x20;
12     priority             = 3;
13     message_size         = 32;
14     system_IPL           = 6;
15     timeout              = NO;
16 };
17 //System Clock Definition
18 clock{
19     mpu_clock             = 10MHz;
20     timer                 = A0;
21     IPL                   = 5;
22     unit_time             = 0.5ms;          // ms
23     initial_time          = 1:0x10:0xffff;
24 };
25 //Task Definition
26 task[1]{
27     entry_address         = _task1;
28     stack_size            = 0x20;
29     priority              = 1;
30     context               = R0, R1, R2, A0;
31     initial_start         = ON;
32 };
33 task[2]{
34     entry_address         = _task2;
35     stack_size            = 512;
36     priority              = 2;
37     context               = R0, R1, R2, A0;
38     initial_start         = OFF;
39 };
40 task[3]{
41     entry_address         = _task3;
42     stack_size            = 512;
43     priority              = 3;
44     context               = R0, R1, R3, A0;
45     initial_start         = OFF;
46 };
47 //
48 flag[1]{
49     name                  = flg1;
50 };
51 //
52 semaphore[1]{
53     name                  = sem1;
54     initial_count         = 1;
55 };
56 //
57 mailbox[1]{
58     name                  = mbx1;
59     buffer_size           = 3;
60 };
61 //
62 memorypool[1]{
63     name                  = mpl1;
64     section               = FIX_MEM;
65     num_block             = 5;
66     siz_block             = 100;
67 };
68 //
69 variable_memorypool[1]{
70     max_memsize           = 400;
71     heap_size             = 1600;
72 };
73 //
74 cyclic_hand[1]{
75     interval_counter      = 0xff;

```

```
76         mode          = TCY_OFF;
77         entry_address  = _cyh1;
78     };
79 //
80 alarm_hand[1]{
81     time          = 1:0xff:0xffff;
82     entry_address = _alh1;
83 };
84 interrupt_vector[6]{
85     os_int        = YES;
86     entry_address = _intr;
87     };
```

6.2 コンフィグレータの実行

6.2.1 コンフィグレータ概要

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

1. コンフィグレータの実行には以下の入力ファイルが必要です。

- コンフィグレーションファイル (XXXX.cfg)
システムの初期設定項目を記述したファイルです。カレントディレクトリに作成します。
- デフォルトコンフィグレーションファイル (default.cfg)
コンフィグレーションファイルで値の設定を省略した場合にこのファイルに書き込まれている値を設定します環境変数 "LIB30"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⁶⁸ (makefile.ews, makefile.dos, makefile, Makefile)
makefile⁶⁹を生成する場合にテンプレートのファイルとして使用するファイルです。(第 6.2.4 項参照)
- mr30.inc テンプレートファイル (mr30.inc)
インクルードファイル mr30.inc のテンプレートとなるファイルです。
環境変数 "LIB30"で示されるディレクトリに存在します。
- MR30 バージョンファイル (version)
MR30 のバージョンを記述したファイルです。環境変数 "LIB30" 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 MR30 のバージョン情報を出します。

2. コンフィグレータの実行によって以下のファイルが出力されます。

コンフィグレータが出力したファイルには、ユーザーのデータ定義を行わないで下さい。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ーの定義したデータは失われます。

- システムデータ定義ファイル (sys_rom.inc)
システムの設定を定義しているファイルです。
- インクルードファイル (mr30.inc)
mr30.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。

⁶⁸ EWS 版では makefile.ews, DOS 版では makefile.dos を使用します。

⁶⁹ この makefile は、UNIX 標準もしくは準拠の make コマンドで処理可能なシステム生成手順記述ファイルです。

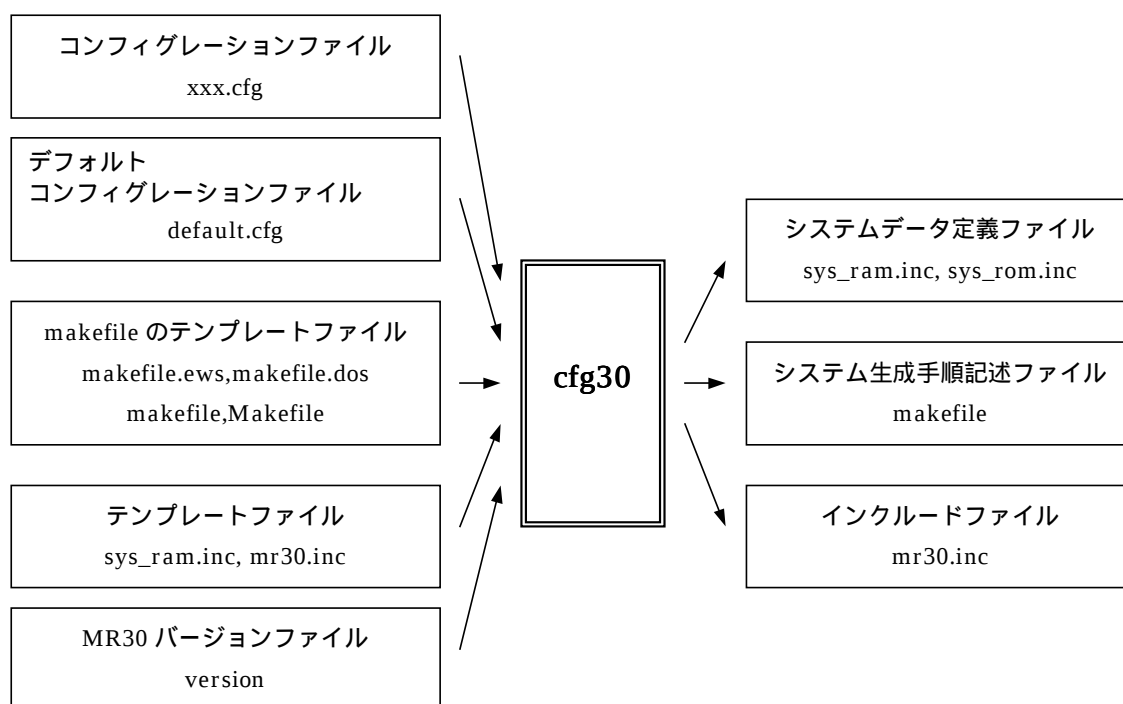


図 6.1 コンフィグレータ動作概要

6.2.2 コンフィグレータの環境設定

コンフィグレータを実行するにあたって環境変数 "LIB30"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 "LIB30" 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)
- mr30.inc のテンプレートファイル (mr30.inc)
- セクション定義ファイル (c_sec.inc または asm_sec.inc)
- スタートアップファイル (crt0mr.a30 または start.a30)
- makefile のテンプレートファイル (makefile.ews または makefile.dos)
- MR30 バージョンファイル (version)

6.2.3 コンフィグレータ起動方法

コンフィグレータは以下の形式で起動します。

```
A> cfg30 [-vmV] コンフィグレーションファイル名
```

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

コマンドオプション

-v オプション

コマンドのオプションの説明と詳細なバージョンを表示します。

-V オプション

コマンドが生成するファイルの作成状況を表示します。

-m オプション

UNIX 標準もしくは準拠のシステム生成手順記述ファイル (makefile)を作成します。指定がない場合は makefile を作成しません。⁷⁰

また、カレントディレクトリにスタートアップファイル (crt0mr.a30 または start.a30)とセクション定義ファイル (c_sec.inc、asm_sec.inc) がない場合は、環境変数 LIB30 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。

⁷⁰ UNIX 標準もしくは準拠の"makefile"には、"clean"ターゲットによりワークファイルを削除する機能があります。すなわち、make コマンドで生成されたオブジェクトファイルなどを削除するには以下のように行います。

```
> make clean
```

6.2.4 makefile 生成機能

コンフィグレータは以下の手順で makefile を生成します。

1. ソースファイルの依存関係を調べます。

コンフィグレータはカレントディレクトリの拡張子が.c と.a30 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'.a30'を使用してください。
- ◆ ソースファイルの依存関係には、#if やコメント行に記述して無効化した#include 文に指定したファイルも含まれます。

2. makefile ヘ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"LIB30"で示されるディレクトリの"makefile.ews"または"makefile.dos"を、テンプレートファイルとして、カレントディレクトリに"makefile"を作成します。

6.2.5 コンフィグレータ実行上の注意

以下にコンフィグレータ実行上の注意点を示します。

- コンフィグレータを再度かけ直した場合は、必ず `make clean` を実行するかオブジェクトファイル (拡張子.r30) 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 `cfg30` では、UNIX 標準、もしくは準拠の `makefile` しか生成しません。

6.2.6 コンフィグレータのエラーと対処方法

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

エラーメッセージ

cfg30 Error : syntax error near line xxx (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。

cfg30 Error : not enough memory

メモリが足りません。

cfg30 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。

cfg30 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。

cfg30 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。

cfg30 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。

cfg30 Error : can't open version file

環境変数"LIB30"の示すディレクトリの下に MR30 バージョンファイル"version"がありません。

cfg30 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 "LIB30"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。

cfg30 Error : can't open configuration file <xxxxcfg>

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。

cfg30 Error : illegal XXXX --> <xx> near line xxx (xxxx.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。

cfg30 Error : Unknown XXXX --> <xx> near line xxx (xxxx.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。

cfg30 Error : too big XXXX's ID number --> <xxx> (xxxx.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。

cfg30 Error : too big task[x]'s priority --> <xxx> near line xxx (xxxx.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。

cfg30 Error : too big IPL --> <xxx> near line xxx (xxxx.cfg)

システムクロック定義項目のシステムクロック割り込み優先レベルがシステム定義項目の system IPL 値を越えています。

cfg30 Error : system timer's vector <x>conflict near line xxx

システムクロックの割り込みベクタに、別の割り込みが定義されています。割り込みベクタ番号を確認して下さい。

cfg30 Error : XXXX is not defined (xxxx.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。

cfg30 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。

cfg30 Error : double definition <XXXX> near line xxx (xxxx.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。

cfg30 Error : double definition XXXX[x] near line xxx (default.cfg)**cfg30 Error : double definition XXXX[x] near line xxx (xxxx.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。

cfg30 Error : you must define XXXX near line xxx (xxxx.cfg)

XXXX は、省略できない項目です。

cfg30 Error : you must define SYMBOL near line xxx (xxxx.cfg)

省略できないシンボルです。

cfg30 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.a30" または "crt0mr.a30" が、カレントディレクトリに必要です。

cfg30 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。

cfg30 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。

cfg30 Error : zero divide error near line xxx (xxxx.cfg)

演算式で 0(ゼロ) 除算が発生しました。

cfg30 Error : task[X].stack_size must set XX or more near line xxx (xxxx.cfg)

タスクのスタックサイズを XX バイト以上のサイズをセットしてください。

cfg30 Error : "R0" must exist in task[x].context near line xxx (xxxx.cfg)

タスクのコンテキスト選択項目では、必ず、R0 レジスタを選択してください。

cfg30 Error : can't define address match interrupt definition for Task Pause Function near line xxx (xxxx.cfg)

タスクポーズ機能に必要な割り込みベクタに別の割り込みがコンフィグレーションファイルに定義されています。

cfg30 Error : Set system.timer [system.timeout = YES] near line xxx (xxxx.cfg)

system.timeout = YES の設定に関わらず、clock 定義において timer 項目が NOTIMER になっています。timer 項目でタイマを設定してください。

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。

cfg30 Warning : system is not defined (xxxx.cfg)**cfg30 Warning : system.XXXX is not defined (xxxx.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。

cfg30 Warning : system.message_size is not defined (xxxx.cfg)

システム定義中のメッセージサイズ定義項目が省略されています。メールボックス機能のメッセージサイズ (16 または 32) を指定して下さい。

cfg30 Warning : task[x].XXXX is not defined near line xxx (xxxx.cfg)

ID 番号 x のタスク定義項目 XXXX が省略されています。

cfg30 Warning : Already definition XXXX near line xxx (xxxx.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。

cfg30 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。

cfg30 Warning : interrupt_vector[x]'s default is not defined near line xxx (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。

cfg30 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。

cfg30 Warning : system.stack_size is an uneven number near line xxx

cfg30 Warning : task[x].stack_size is an uneven number near line xxx

スタックサイズは、偶数サイズを指定してください。

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。

cfg30 Error : xxxx (line xxx): 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。

cfg30 Warning : xxxx (line xxx): can't find <XXXX>

cfg30 Warning : xxxx (line xxx): 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。

cfg30 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。

6.3 makefile の編集

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

1. NC30 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。

2. AS30 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。

3. LN30 コマンドオプション

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

4. ライブラリの指定

ライブラリの指定は、"LIBS" に定義します。

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

独自に makefile を作成する場合は、下記 4 項目を必ず、メイクファイルに記述してください。

1. MR30 ライブラリの指定

メールボックス機能のメッセージサイズにより、リンクする MR30 ライブラリが異なります。32 ビットのメッセージサイズを使用している場合、MR30 ライブラリは、mr30lm.lib と c30mrlm.lib を指定しなければなりません。また、16 ビットのメッセージサイズを使用している場合は、mr30.lib と c30mr.lib を指定する必要があります。

2. コンパイルオプションの指定

メールボックス機能関連のシステムコールを使用しているファイルをコンパイルする時にコンパイルオプションの指定に注意してください。32 ビットのメッセージサイズを使用している場合、コンパイルオプションには、"-Dfar_msg=1"を指定してください。16 ビットの場合は、このコンパイルオプションの指定は必要ありません。

3. アセンブルオプションの指定

システムコールを発行しているアセンブリ言語で記述したソースファイルをアセンブルする場合には、必ず、アセンブルオプション"-F"を指定してください。

4. リンクを行う前の処理

リンクを行う前には、必ず、以下に示す 2 つの処理を以下に示す順序で実行してください。

1. mkmrtbl

2. as30 mrtable.a30

mkmrtbl は、MR30 が用意しているユーティリティです。mkmrtbl はコンフィグレータ cfg30 を起動したディレクトリ (mkmrtbl はカレントディレクトリの cfg30 が出力した mr30.inc、vector.tpl を読み込んで処理します) で起動してください。また、コンパイラ、アセンブラの生成するシステムコールファイル (XXX.mrc) が出力されるディレクトリ (r30 ファイルが出力されているディレクトリと同じ) がカレントディレクトリでない場合は、その出力先ディレクトリを mkmrtbl の引数に指定して下さい。

例) mkmrtbl outputdir

注)デバグ PD30 でシステムコール発行機能を使用する場合は、mkmrtbl 起動時の引数に\$(LIB30) MR30 インストール先ディレクトリの lib30) を追記してください。

mkmrtbl を実行すれば、mrtable.a30 ファイルがカレントディレクトリに生成されます。

上記に示した処理を実行した後、mrtable.r30 ファイルを含めてリンクを実行してください。

第 7 章

アプリケーション作成の手引き

7.1 ハンドラからのシステムコールの処理手順

ハンドラ⁷¹からのシステムコール発行はタスクからのシステムコールと異なり、システムコール発行時にタスク切り替えは発生しません。タスク切り替えが発生するのはハンドラからの復帰時です。

ハンドラからのシステムコール処理手順は大きく分けて以下の 3 通りがあります。

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

⁷¹ OS 独立割り込みハンドラからはシステムコールは発行できませんので、ここで述べているハンドラは OS 独立割り込みハンドラを含みません。

7.1.1 タスク実行中に割り込んだハンドラからのシステムコール

スケジューリング (タスク切り替え) は `ret_int` システムコールによりおこなわれます。⁷² (図 7.1 参照)

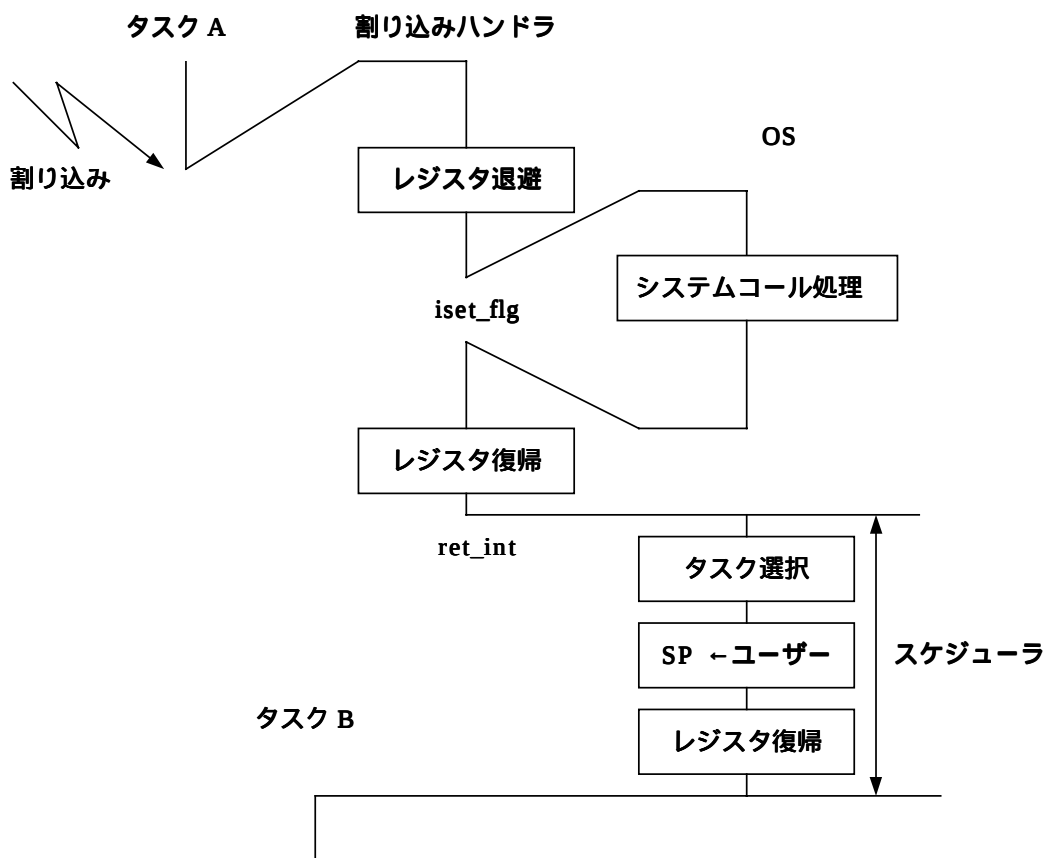


図 7.1 タスク実行中に割り込んだ割り込みハンドラからのシステムコール処理手順

⁷² C 言語で OS 依存割り込みハンドラを記述する場合 (`#pragma INTHANDLER` 指定時) `ret_int` システムコールは自動的に発行されます。

7.1.2 システムコール処理中に割り込んだハンドラからのシステムコール

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

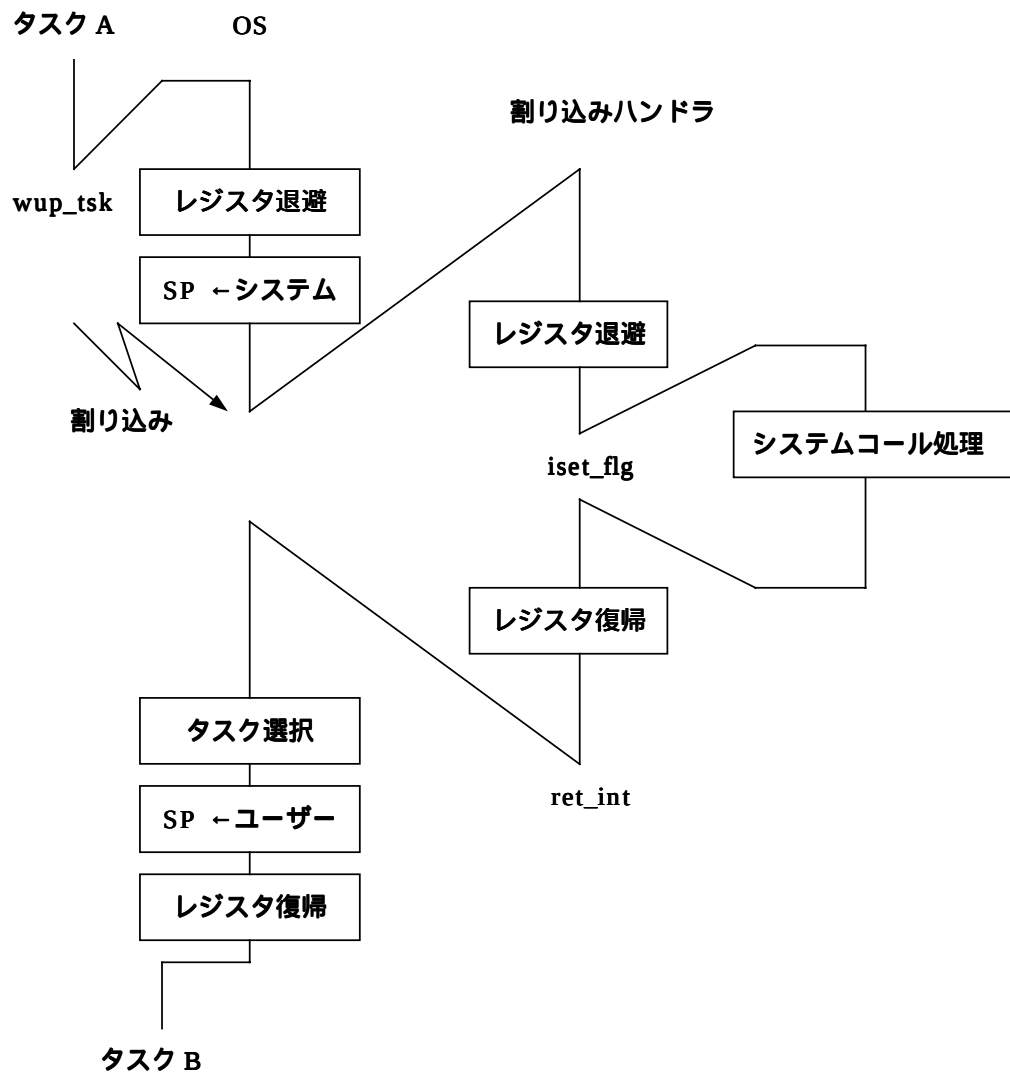


図 7.2 システムコール処理中に割り込んだ割り込みハンドラからのシステムコール処理手順

7.1.3 ハンドラ実行中に割り込んだハンドラからのシステムコール

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

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

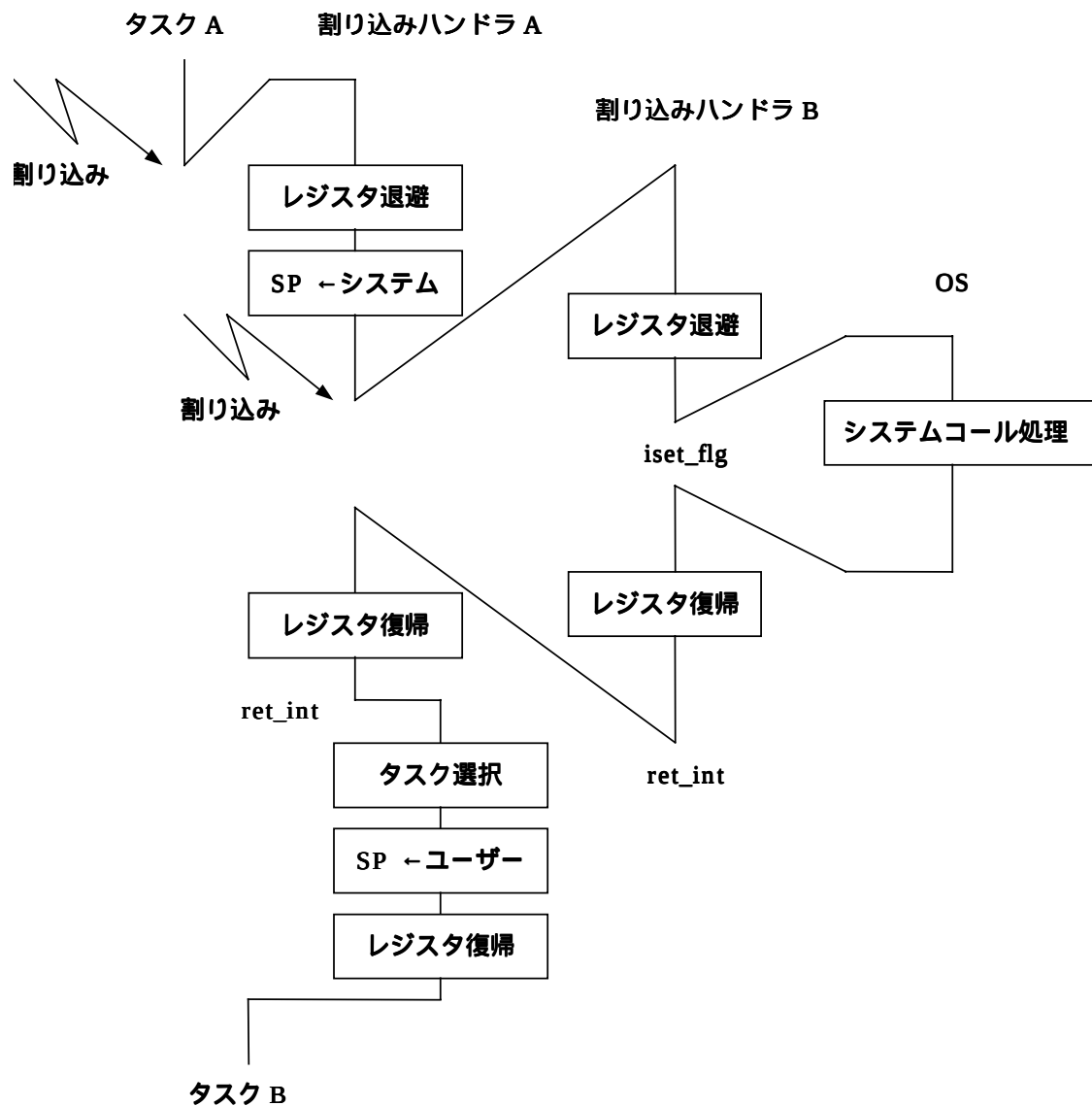


図 7.3 多重割り込みハンドラからのシステムコール処理手順

7.2 システムの使用する RAM 容量の計算方法

MR30 カーネルがタスクなどを管理するための RAM は、MR_RAM セクションにあります。

MR_RAM セクションにおいて MR30 が使用する RAM 容量は表 7.1 で計算することができます。ただし、システムおよびタスクの使用するスタックは含まれていません。

スタックサイズの計算はリファレンスマニュアルを参照して下さい。

表 7.1 MR_RAM セクションのサイズ算出方法

領域名	バイト数
システム作業領域	4+優先度数
システム時刻管理領域	6
タスク管理領域	11 × タスク数(タイムアウトなし)
	15 × タスク数(タイムアウトあり)
タイマキュー管理領域	6
周期起動ハンドラ管理領域	3 × 周期起動ハンドラ数
アラームハンドラ管理領域	1
イベントフラグ管理領域	4 × イベントフラグ数+1+ (イベントフラグ数-1)/8
セマフォ管理領域	3 × セマフォ数
固定長メモリプール管理領域	2 × 固定長メモリプール数
可変長メモリプール管理領域	44(メモリプール領域は含みません)
メールボックス管理領域	7 × メールボックス数 +メールボックスバッファサイズ × 2 (16 ビットの場合)or +メールボックスバッファサイズ × 4 (32 ビットの場合)

MR30 を使用する上で、最低限必要な RAM サイズは、16 バイトとなります。

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

なお、MR_RAM セクションの他に MR_RAM_DBG セクションがあります。このセクションは、OS デバッグ機能を使用するために必要なセクションです。このセクションの RAM 容量は、タスクポーズ機能を使用する場合は 17 バイト、タスクポーズ機能を使用しない場合は、13 バイトになります。

7.3 スタックについて

7.3.1 システムスタックとユーザースタック

MR30 のスタックにはシステムスタックとユーザースタックがあります。

- ユーザースタック
タスクごとに 1 つずつ存在するスタックです。したがって MR30 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
MR30 内部 (システムコール処理中) に使用されるスタックです。MR30 ではシステムコールをタスクが発行するとスタックをユーザースタックからシステムスタックに切り替えます。(図 7.4を参照して下さい。)
システムスタックは、割り込みスタックを使用します。

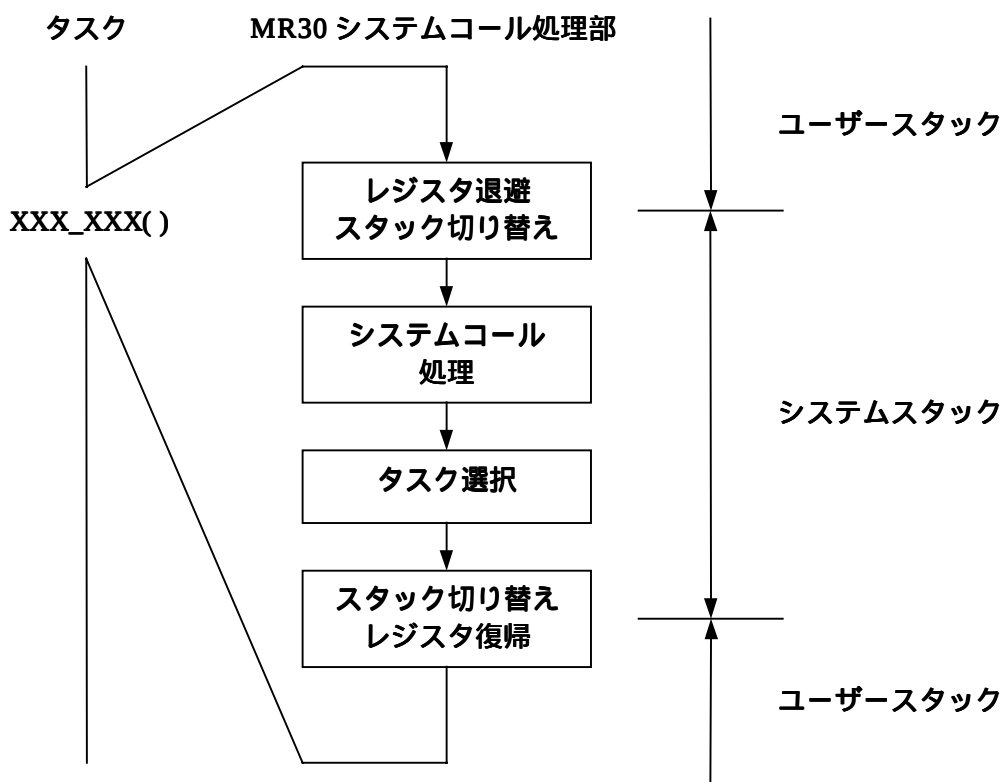


図 7.4 システムスタックとユーザースタック

また、ベクタ番号が 0 ~ 31、247 ~ 255 の割り込み発生時には、ユーザースタックからシステムスタックに切り替えます。したがって、割り込みハンドラで使用するスタックは全てシステムスタックを使用します。

第 8 章

サンプルプログラムの説明

8.1 概要

MR30 の応用例として、M16C/60 シリーズの各ポート (P0~P8) に接続した LED を点灯させるプログラムを示します。この応用例では各ポートの制御を、一つ一つの独立した関数により行なっています。表 2.1に各関数の一覧を示します。

表 8.1 サンプルプログラムの関数一覧

関数名	種類	ID 番号	優先度	機能
main()	タスク	1	1	task2 から順次 task4 まで起床させます。
task2()	タスク	2	2	port7 の入出力を制御します。
task3()	タスク	3	3	port8 の入出力を制御します。
task4()	タスク	4	4	port9 の入出力を制御します。
cyh1()	ハンドラ			port10 の出力データを変更します。

main タスクは、まずポート 7,8,9,10 を出力モードに設定します。そして、task2 ~ task4 を起動します。

task2 は、初期値 0xff をポート 7 に設定し (ポートをすべて点灯させる。)システムクロックが 25 回カウントするまで待ち状態に移行します。その後ポート 7 に 0x01 の値を設定し、LED を点灯させます。そして再びシステムクロックが 25 回カウントするまで待ち状態に移行し、ポート 0 のデータ値を 1 ビットシフトし点灯させるということを 8 回繰り返します。(例 0 0 0 0 0 0 0 1 → 0 0 0 0 0 0 1 0 → 0 0 0 0 0 1 0 0) これを無限ループにします。

図 8.1にポートの点灯の様子を示します。

task3 ~ 4 は、待ち時間が task2 と違うのみで他は同様の処理を行ないます。

変数 p10 とポート 10 の値は周期起動ハンドラ cyh1()の中で変更されます。周期起動ハンドラは変数 pt10 の値を 1 ビットずつシフトさせ、その値をポート 10 に転送し、LED にその値を点灯させています。

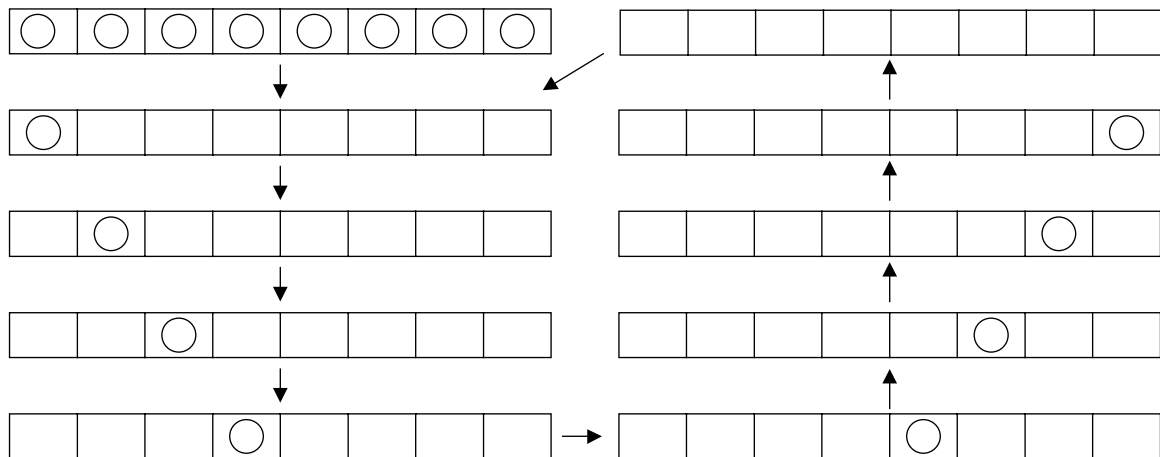


図 8.1 LED 点灯の様子

8.2 ソースプログラム

```

1  /*****
2
3      COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
4      AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
5
6          sample program
7          $Id: demo.c,v 1.7 2003/08/22 12:59:49 muraki Exp $
8
9  *****/
10
11 #include <stdio.h>
12 #include <mr30.h>
13
14 #include "id.h"
15
16 #pragma ADDRESS PD7      3efH
17 #pragma ADDRESS PD8      3f2H
18 #pragma ADDRESS PD10     3f6H
19
20 #pragma ADDRESS P7       3edH
21 #pragma ADDRESS P8       3f0H
22 #pragma ADDRESS P9       3f1H
23 #pragma ADDRESS P10      3f4H
24
25 char PD7,PD8,PD10;
26 char P7,P8,P9,P10;
27
28 char pt7,pt8,pt9,pt10;
29
30 void main()
31 {
32     #pragma ASM
33         bset    2,0AH
34         mov.w   #0FFH,3f3H
35         bclr    2,0AH
36     #pragma ENDASM
37     PD7 = PD8 = PD10 = 0xff;
38
39     sta_tsk(ID_task2,1);
40     sta_tsk(ID_task3,1);
41     sta_tsk(ID_task4,1);
42 }
43
44 void task2()
45 {
46     int    k;
47
48     P7 = 0xff;
49
50     while(1){
51         dly_tsk(25);
52         pt7 = 0x01;
53         P7 = pt7;
54         for(k=1; k<=8; k++){
55             dly_tsk(25);
56             pt7 <= 1;
57             P7 = pt7;
58         }
59     }
60 }
61
62 void task3()
63 {
64     int k;
65     P8 = 0xff;
66
67     while(1){
68         dly_tsk(50);
69         pt8 = 0x01;
70         P8 = pt8;
71         for(k=1; k<=8; k++){
72             dly_tsk(50);
73             pt8 <= 1;
74             P8 = pt8;
75         }
76     }

```

```
77     }
78 }
79
80 void task4()
81 {
82     int k;
83     P9 = 0xff;
84
85     while(1){
86         dly_tsk(100);
87         pt9 = 0x01;
88         P9 = pt9;
89         for(k=1;k<=8;k++){
90             dly_tsk(100);
91             pt9 <= 1;
92             P9 = pt9;
93         }
94     }
95 }
96
97 void cyh1()
98 {
99     if(pt10 == 0)
100         pt10 = 0x01;
101     P10 = pt10;
102     pt10 <= 1;
103 }
```

8.3 コンフィグレーションファイル

```

1//*****
2//
3//      COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
4//      AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
5//
6//      MR30 System Configuration File.
7//      $Id: smp.cfg,v 1.8 2003/08/22 12:59:49 muraki Exp $
8//
9//*****
10
11// System Definition
12system{
13    stack_size          = 1024;
14    priority            = 10;
15    system_IPL          = 4;
16    message_size        = 16;
17    timeout             = NO;
18    task_pause          = NO;
19};
20//System Clock Definition
21clock{
22    mpu_clock           = 10MHz;
23    timer               = A0;
24    IPL                 = 4;
25    unit_time           = 100ms;          // ms
26    initial_time        = 0:0:0;
27};
28//Task Definition
29task[1]{
30    entry_address       =    main();
31    stack_size          =    100;
32    priority            =    1;
33    initial_start       =    ON;
34};
35task[2]{
36    entry_address       =    task2();
37    stack_size          =    100;
38    priority            =    2;
39};
40task[3]{
41    entry_address       =    task3();
42    stack_size          =    100;
43    priority            =    3;
44};
45task[4]{
46    entry_address       =    task4();
47    stack_size          =    100;
48    priority            =    4;
49};
50// Cyclic Handler Definition
51cyclic_hand[1]{
52    interval_counter    =    150;
53    mode                =    TCY_ON;
54    entry_address       =    cyh1();
55};

```


第 9 章

別 ROM 化について

9.1 別 ROM 化の方法

本章では、MR30 のカーネルとアプリケーションプログラムを別の ROM に配置する方法を示します。

図 9.1は、2 つの異なるアプリケーション間の共通部分とカーネル部分はカーネル ROM に、アプリケーション部分はそれぞれ別の ROM に配置した場合の例を示しています。

この例をもとに、ROM 分割の方法を示します。

4. システム構成

アプリケーションプログラムのシステム構成を行います。

ここでは、2 つのアプリケーションプログラムのシステム構成が以下に示すものとして説明を行います。

	アプリケーション 1	アプリケーション 2
タスク数	4	5
イベントフラグ数	1	3
セマフォ数	4	2
メールボックス数	3	5
固定長メモリプール数	3	1
周期起動ハンドラ数	3	3

5. コンフィグレーションファイル作成

システム構成を行なった結果をもとに、コンフィグレーションファイルを作成します。この時、以下に示す定義部で注意が必要です。

- maxdefine 定義

この maxdefine 定義部で設定する値は、2 つのアプリケーション間の各定義について定義数の大きい方を指定します。よって、この 2 つのアプリケーションにおいて各項目の値は、一致していなければなりません。

```

例
    maxdefine{
        max_task      = 5;
        max_flag      = 3;
        max_sem       = 4;
        max_mbx       = 5;
        max_mpl       = 3;
        max_cyh       = 3;
    };

```

なお、maxdefine 定義には、可変長メモリプールに対する指定は用意しておりませんので、どちらかのアプリケーションにおいて可変長メモリプール機能をご使用になっている場合は、2 つのコンフィグレーションファイルにおいて、可変長メモリプールに対する定義を同じ指定内容で行ってください。

- system 定義

この system 定義の以下に示す項目を 2 つのアプリケーションにおいて共通にする必要があります。

- ◆ message_size
- ◆ timeout
- ◆ task_pause
- ◆ priority

- clock 定義

- ◆ initial_time

この項目に設定する値は、2 つのアプリケーション間で異なっても問題ありませんが、1 つのアプリケーションではこの項目を定義し、もう 1 つのアプリケーションでは定義を省略するといったことはしないでください。必ず、この項目の設定および省略は 2 つのアプリケーション間で統一してください。

- task 定義

- ◆ initial_start

この項目には、システム起動後最初に起動されるタスクのみを"ON"に指定し、その他のすべてのタスクには"OFF"を指定してください。

- variable_memorypool 定義

この定義の設定および省略は、2 つのアプリケーション間で統一してください。1 つのアプリケーションではこの定義の設定を行い、もう 1 つのアプリケーションではこの定義を省略するといったことはしないでください。

その他の定義については、2 つのコンフィグレーションファイル間で異なっても問題ありません。

6. プロセッサモードレジスタの変更

スタートアッププログラムのプロセッサモードレジスタをシステムに応じて変更します。

7. アプリケーションプログラム作成

2 つのアプリケーションプログラムを作成します。

8. スタートアッププログラムのセクション名変更

サンプルスタートアップファイル(start.a30、crt0mr.a30)では、スタートアッププログラムは MR_KERNEL セクションとして定義していますが、これを MR_KERNEL とは別のセクション名に変更して下さい。

例

```
.section MR_KERNEL, CODE, ALIGN
↓ (変更)
.section MR_STARTUP, CODE, ALIGN
```

9. 各セクション配置設定

カーネル ROM とアプリケーション ROM に配置するプログラムを以下に示します。

●カーネル ROM に配置するプログラム

- ◆ MR30 のカーネル (MR_KERNEL セクション)
- ◆ 2 つのアプリケーション間の共通プログラム (program セクション)

この例では ID=1 のタスクが共通プログラムとします。共通プログラムはアプリケーション ROM に配置しても問題ありません。なお、共通プログラムをカーネル ROM に配置する場合は、以下に示すシステムコールは発行できませんので、ご注意ください。

can_wup、get_tid、pget_blf、pget_blk、pol_flg、
prcv_msg、rcv_msg、trcv_msg、twai_flg、wai_flg

共通プログラムから上記のシステムコールを発行したい場合は、アプリケーション ROM に配置してください。

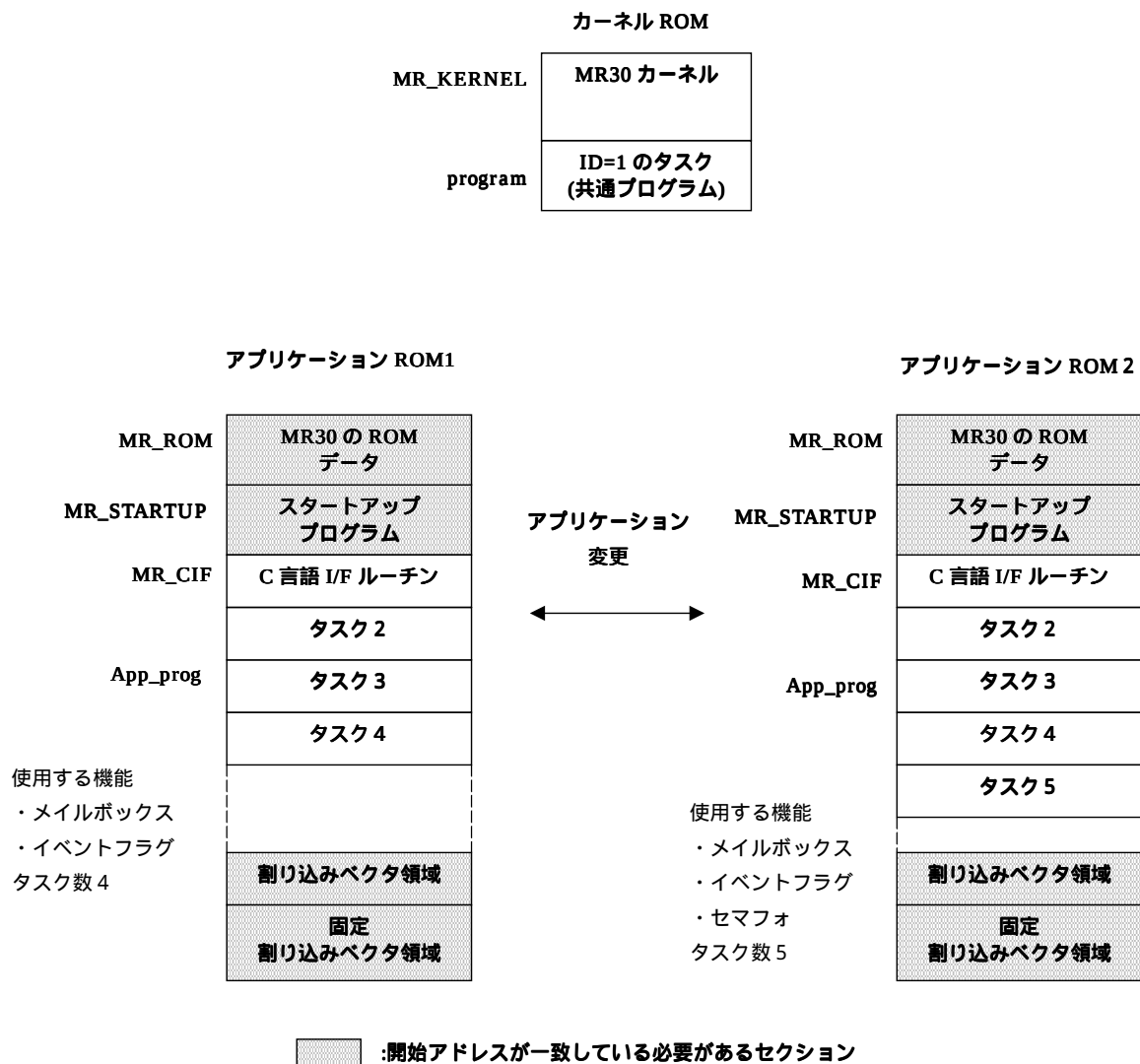


図 9.1 ROM 分割

- アプリケーション ROM に配置するプログラム
 - ◆ スタートアッププログラム
 - ◆ MR30 の ROM データ (MR_ROM セクション)
 - ◆ C 言語 I/F ルーチン (MR_CIF セクション)
 - ◆ アプリケーションプログラム (app_prog セクション)
 - ◆ 割り込みベクタ領域 (INTERRUPT_VECTOR セクション)
 - ◆ 固定割り込みベクタ領域 (FIX_INTERRUPT_VECTOR セクション)
- 各プログラムの配置方法を以下に示します。
 - ◆ ユーザープログラムのセクション名変更

C 言語でアプリケーションプログラムを記述している場合は、以下に示すように #pragma SECTION を使ってアプリケーション ROM に配置するプログラムのセクション名を変更します。NC30 では、ユーザープログラムのセクション名は、設定がなければ program セクションになります。そのため、アプリケーション ROM に配置するタスクを別のセクション名に変更しなければなりません。⁷³

```
#pragma SECTION program app_prog/* プログラムのセクションを変更 */
/* task2,task3 のセクション名は app_progに変わります。 */
void task2(void){
    :
}

void task3(void){
    :
}
```

- ◆ セクションの配置設定

セクションファイル (c_sec.inc、asm_sec.inc)を変更して、各セクションのアドレス設定を行ないます。この時、以下に示すセクションの開始アドレスは、2 つのアプリケーション間で一致していなければなりません。

 - スタートアッププログラム
 - MR30 の RAM データ (MR_RAM、MR_RAM_DBG セクション)
 - MR_HEAP セクション
 - MR30 カーネル (MR_KERNEL セクション)
 - MR30 の ROM データ (MR_ROM セクション)
 - 割り込みベクタ領域 (INTERRUPT_VECTOR セクション)

⁷³ カーネル ROM に配置するタスクのセクションは、変更する必要はありません。

以下に、セクションファイルの設定例を示します。

```
.section MR_RAM_DBG, DATA      ; MR30のRAMデータ
.org 500H                      ; 2つのアプリケーション間で共通アドレス
.section MR_RAM, DATA         ; MR30のRAMデータ
.org 600H                      ; 2つのアプリケーション間で共通アドレス
:
.section MR_HEAP, DATA        ; MR_HEAPセクション
.org 10000H                    ; 2つのアプリケーション間で共通アドレス
:
.section MR_ROM, ROMDATA       ; MR30のROMデータ
.org 0e0000H                  ; 2つのアプリケーション間で共通アドレス
:
.section MR_STARTUP, CODE      ; スタートアッププログラム
.org 0e1000H                  ; 2つのアプリケーション間で共通アドレス

.section MR_CIF, CODE          ; C言語 I/F ルーチン
:
.section app_prog, CODE        ; ユーザープログラム
:
.section INTERRUPT_VECTOR     ; 可変割り込みベクタ
.org 0efd00H                  ; 2つのアプリケーション間で共通アドレス

.section MR_KERNEL, CODE      ; MR30のカーネル
.org 0f0000H                  ; 2つのアプリケーション間で共通アドレス
:
.section FIX_INTERRUPT_VECTOR ; 固定割り込みベクタ
.org 0fffdch                  ; 2つのアプリケーション間で共通アドレス
```

図 9.2に示すメモリマップになります。

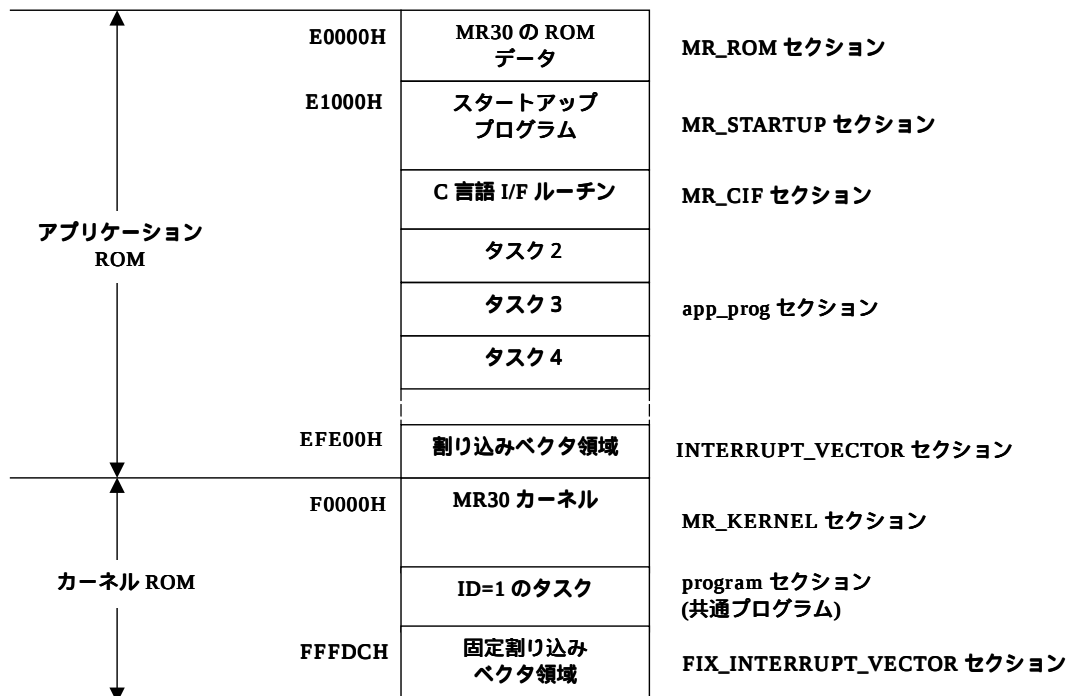


図 9.2 メモリマップ

10. コンフィグレータ `cfg30` を実行します。

11. 全システムコール名を記述した `mrc` ファイルを作成してください。(コンパイルを行うとワークディレクトリに拡張子 `mrc` というファイルが生成されます。これを参考にして作成ください。)なお、`cfg30` が生成するメイクファイルでは、`make clean` を実行すると `mrc` ファイルを削除しますので、ご注意ください。

12. システム生成

`make` コマンドを実行し、システム生成します。⁷⁴

13. 7から12の処理をアプリケーション 2 についても行うことで、アプリケーション 2 のシステムが生成できます。

以上に述べた方法で、別 ROM 化が行なえます。

⁷⁴ カレントディレクトリに `mrtable.a30` ファイルがない場合は、`make` を実行して生成します。

索引

A

act_cyc 47
 AND 待ち 36
 AS30 119
 asm_sec.inc 54, 86, 87

C

c_sec.inc 54, 86, 88
 can_wup 34
 cfg30 18, 59
 chg_pri 31
 clr_flg 36
 crt0mr.a30 54, 79, 80, 88
 C コンパイラ NC30 8

D

default.cfg 110
 dis_dsp 76
 dly_tsk 46
 DORMANT 状態 22

E

ena_dsp 76
 ext_tsk 31

F

FIX_INTERRUPT_VECTOR 87

G

get_tid 33
 get_tim 47
 get_ver 49

I

ichg_pri 28, 31
 id.h 54, 62
 INT 72
 INTERRUPT_VECTOR 87
 IPL 73, 74
 irel_wai 28, 32
 irot_rdq 28, 32, 76
 irsm_tsk 28, 34, 76
 iset_flg 28, 36
 isig_sem 28, 38
 isnd_msg 28, 41
 ista_tsk 28, 31
 isus_tsk 28, 34, 76
 ITRON 仕様 6
 iwup_tsk 28, 34

L

LIB30 112
 LMC30 59
 LN30 119
 loc_cp 76
 loc_cpu 42, 75, 76

M

make 59
 makefile 54, 110, 114, 119
 Makefile 110
 makefile.dos 110
 makefile.ews 110
MPU 情報 49
 MPU のクロック 96
 MR_CIF 87
 MR_HEAP 87, 103
 MR_KERNEL 87
 MR_RAM 87, 103, 126
 MR_RAM_DBG 87
 MR_ROM 87

MR30	6, 8
mr30.h	62
mr30.inc	54, 67, 69, 71, 110
MR30 概略仕様	7

N

NC30.....	119
-----------	-----

O

OR 待ち	36
OS 依存割り込みハンドラ	64, 69, 73, 106
OS 割り込み禁止レベル	74, 85, 96
OS 独立割り込みハンドラ	65, 70, 73

P

pget_blf.....	43
pget_blk.....	44
pol_flg.....	36
prcv_msg.....	41
preq_sem.....	38
program.....	87

R

READY 状態.....	20
ref_alm	47
ref_cyc.....	47
ref_flg	36
ref_mbx.....	41
ref_mpf	43
ref_mpl	45
ref_sem	38
ref_tsk.....	33
REIT	70
rel_blf	43
rel_blk.....	45
rel_wai.....	32
ret_int.....	28, 42, 64
ROM 書き込み形式ファイル.....	54
rot_rdq.....	32, 76
rsm_tsk.....	34
RTS.....	71
RUN 状態.....	20

S

set_flg.....	36
set_tim.....	47
SFR.....	75
sig_sem	38
slp_tsk	34, 76
sta_tsk	31
stack	87
start.a30	54, 79, 87
sus_tsk.....	34
SUSPEND 状態.....	21
sys_ram.inc	54
sys_rom.inc.....	54, 110
system.IPL	73, 74

T

TCB.....	24
TCY_INI_ON.....	47
TCY_OFF.....	105
TCY_ON	47, 105
ter_tsk.....	31
trcv_msg.....	41, 46, 96
TRON 仕様.....	6
tslp_tsk	34, 46, 96
TSS.....	32
twai_flg.....	36, 46, 96
twai_sem.....	38, 46, 96

U

unl_cpu	42, 75, 76
---------------	------------

V

version	110
---------------	-----

W

wai_flg	36
wai_sem	38, 76
WAIT-SUSPEND 状態.....	21
WAIT 状態.....	21
wup_tsk	34

M

μITRON 仕様.....	6
μITRON 仕様 V.2.0.....	6
μITRON 仕様 V.3.0.....	6

あ

アラームハンドラ	26, 46, 66, 71
アラームハンドラ定義.....	105

い

イベントフラグ	36
イベントフラグ定義	100
イベントフラグ待ち行列	21

か

カーネル	29
可変長メモリプール	44
可変長メモリプール定義	103
関数名.....	93

き

起動時刻	105
休止状態	22
強制待ち状態	21, 34

く

クリア指定.....	36
------------	----

け

形式番号.....49

こ

固定長メモリプール.....43
固定長メモリプール定義.....102
固定ベクタ割り込み要因.....107
コンテキスト.....26, 100
コンフィグレーションファイル.....54, 92, 95, 108, 110
コンフィグレータ.....18, 59, 110, 112, 113, 115

さ

最大アラームハンドラ定義数.....99
最大イベントフラグ定義数.....98
周期起動ハンドラ定義.....104
最大項目数定義.....98
最大固定長メモリプール定義数.....99
最大周期起動ハンドラ定義数.....99
最大セマフォ定義数.....98
最大タスク定義数.....98
最大メールボックス定義数.....98
最大メッセージ数.....102

し

時間.....93
時間管理.....46
時刻.....94
システムクロック.....97
システムクロック割り込み.....85
システムクロック割り込みハンドラ.....26, 73
システムクロック割り込み優先レベル.....97
システムクロック定義.....96
システムコール.....16
システムコール処理.....17
システムコール発行.....122
システム時刻.....97
システムスタック.....127
システムスタックサイズ.....95
システムスタックポインタ.....85
システムタイマ.....46, 79, 85
システムデータ定義ファイル.....110
システム起動.....78
システム生成手順記述ファイル.....54
システム定義.....95
実行可能状態.....20
実行状態.....20
周期間隔.....104
周期起動ハンドラ.....26, 46, 66, 71
周波数.....93
仕様書バージョン.....49
初期起動状態.....100
初期起動タスク.....78
シンボル.....93

す

スケジューラ.....30, 42
スタートアッププログラム.....79, 80, 87

スタックサイズ.....126

せ

製品管理情報.....49
製品バージョン.....49
セクションファイル.....86
セクション定義ファイル.....54
セマフォ.....38
セマフォカウンタ.....38, 101
セマフォ定義.....101
セマフォ待ち行列.....21

そ

ソフトウェア割り込み.....72

た

タイムアウト.....46, 96
多重割り込み.....70
タスク.....19
タスク、ハンドラから発行できるシステムコール.....50
タスク開始アドレス.....99
タスク管理.....31
タスクコントロールブロック.....24
タスク定義.....99
タスクの ID 番号.....18
タスクの状態.....19
タスクの初期優先度.....100
タスクの記述方法.....62
タスクの識別.....18
タスク付属同期機能.....34
タスクポーズ.....96
タスクの優先度.....23
タスク切り替え.....13
単位時間.....97

て

ディスパッチ.....13
ディスパッチ遅延.....76
デフォルトコンフィグレーションファイル.....110

に

二重待ち状態.....21, 34

は

バージョンファイル.....110
バージョン管理.....49
バリエーション記述子.....49
ハンドラ.....26
ハンドラ専用のシステムコール.....28

ふ

プロセッサの初期化.....79
プロセッサモードレジスタ.....85

へ

ベクタ番号	107, 127
別 ROM 化	98, 136

ま

待ち状態	21
------------	----

め

メールボックス	40
メールボックス定義	102
メールボックス待ち行列	21
メーカー名	49
メッセージキュー	40
メッセージサイズ	96
メッセージパケット	40
メモリプール管理	43
メモリ配置	86

ゆ

ユーザースタック	127
ユーザースタックサイズ	100
優先度	95

ら

ラウンドロビンスケジューリング	32
ラム	54

り

リアルタイム OS	4
リアルタイム OS	10
リアルタイム OS の動作原理	13
リンケーজেديتا LN30	8

れ

レジスタバンク	72, 106
レジスタバンク切り替え	100, 106
レディキュー	23

わ

割り込み管理	42
割り込み禁止/許可	74
割り込み許可フラグ	74
割り込み制御レジスタ	75
割り込みハンドラ	26, 73
割り込みベクタ定義	106
割り込みベクタテーブル	85

M3T-MR30 V.3.30 ユーザーズマニュアル

Rev. 1.00
03.09.16
RJJ10J0133-0100Z

COPYRIGHT ©2003 RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED

M3T-MR30 V.3.30 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

RJJ10J0133-0100Z