お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

アプリケーション・ノート

RENESAS

保守／廃止

RX78K/Ⅲ リアルタイムOS

μPD78320
μPD78322

資料番号　EEA-611A（第2版）
発行年月　November 1992 P

アプリケーション・ノート

NEC

保守／廃止

RX78K/Ⅲ リアルタイムOS

μPD78320
μPD78322

保守／廃止

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

ＩＴＲＯＮは、Industrial TRON の略称です。

ＭＳ－ＤＯＳ™は、米国マイクロソフト社の商標です。

保守／廃止

○文書による当社の承諾なしに本資料の転載複製を禁じます。
○本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的所有権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
○当社は品質，信頼性の向上に努めていますが，半導体製品はある確率で故障が発生します。当社半導体製品の故障により結果として，人身事故，火災事故，社会的な損害等を生じさせない冗長設計，延焼対策設計，誤動作防止設計等安全設計に十分ご注意願います。
○当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定して頂く「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の上ご使用願います。

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

特別水準：輸送機器（自動車，列車，船舶等），交通用信号機器，防災／防犯装置，各種安全装置，生命維持を直接の目的としない医療機器

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，生命維持のための装置またはシステム等

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。当社製品を上記の「標準水準」の用途以外でご使用をお考えのお客様は，必ず事前に当社販売窓口までご相談頂きますようお願い致します。
○この製品は耐放射線設計をしておりません。

M4 94.11

○文書による当社の承諾なしに本資料の転載複製を禁じます。
○この製品を使用したことにより，第三者の工業所有権等にかかわる問題が発生した場合，当社製品の構造製法に直接かかわるもの以外につきましては，当社はその責を負いませんのでご了承ください。
○当社は，航空宇宙機器，海底中継器，原子力制御システム，生命維持のための医療用機器などに推奨できる製品を標準的には用意しておりません。当社製品をこれらの用途にご使用をお考えのお客様，および，『標準』品質水準品を当社が意図した用途以外にご使用をお考えのお客様は，事前に販売窓口までご連絡頂きますようお願い致します。

当社推奨の用途例

標準：コンピュータ，OA機器，通信機器，計測機器，工作機械，産業用ロボット，AV機器，家電等

特別：輸送機器（列車，自動車等），交通信号機器，防災／防犯装置等

○この製品は耐放射線設計をしておりません。

M4 92.6

保守／廃止

保守／廃止

目　次

保守／廃止

図の目次

保守／廃止

表の目次

1. 概要

ＲＸ７８Ｋ／Ⅲは、１６／８ビット・シングルチップ・マイクロコンピュータ「μPD78320/322, μPD78327/328, μPD78330/334」用に開発されたリアルタイムＯＳです。

本アプリケーション・ノートは、ＲＸ７８Ｋ／Ⅲを使用してリアルタイム・システム設計をされる読者を対象に、以下のような疑問に答えられるよう作成したものです。

・リアルタイムＯＳの機能と使用方法を理解していただく。
・実際の設計手順をどうすれば良いか。
・Ｃコンパイラ（CC78K3）、アセンブラ（RA78K3）での開発において、どのような点に注意したら良いか。

本書は主に、リアルタイムＯＳを始めて使用されるプログラマを対象としています。
以下、本書の進行について説明します。

第２章：リアルタイム・システムの例題として簡単な（単純な）例を取り上げ、システム機能とハードウェア、及び開発ツールの設定と説明を行ないます。

第３章：リアルタイムＯＳの概要と、ＲＸ７８Ｋ／Ⅲ機能の詳細について解説します。

第４章：例題システムを使って具体的に設計作業を進めていきます。
設計段階の各部について、できるだけ一般的・汎用的な考察を行なっています。

第５章：コンパイル～リンク作業の注意点と、例題システムの具体例について解説します。

付録　：Ｉ／Ｏインタフェースとプログラム・リストを掲載しています。

2. システム設計

本章では、ターゲット・システム・ハードウェアとシステム機能、及び開発ツールについて説明します。

2.1 システム設計

ステッピング・モータをインターバル・タイマを使って定速度運転させるという簡単なシステムです。

ＥＢボード（EB-78320-PC)を使用したディバグを目的としていますが、μPD78322によるＲＯＭ化をも前提としています。

システムＩ／Ｏと、システム機能を示します。

1）Ｉ／Ｏ

・ステッピング・モータ（１個）
　ポート０に接続、励磁信号を周期的に与えることによりステップ動作させる。
・ＬＥＤ（１個）
　ポート３０に接続、ステッピング・モータの回転速度をモニタする。
・チャタレス・キー（３個）
　ポート２１～２３に接続、モータ制動のための指令キーとする。

2）システム機能

・モータ機能
　キー指令による設定速度で定速度回転し、段階的に５レベルの速度設定・運転を行なう。
・ＬＥＤ機能
　モータ回転速度を点滅スピードにより表示。
　モータ停止中、ＬＥＤは消灯。
・キー機能
　運転キーにより、運転の開始／停止を行なう。
　運転中のＵＰキーにより、回転速度を上昇させる。
　運転中のＤＯＷＮキーにより、回転速度を下降させる。

Ｉ／Ｏインタフェース詳細とＩ／Ｏ部回路図については、付録を参照ください。

保守／廃止

2.2 開発ツール

1）Cコンパイラ（ＣＣ７８Ｋ３）：
タスク部のプログラミングに使用しています。
第５章で注意点について説明しています。

2）アセンブラ（ＲＡ７８Ｋ３）：
リセット・ルーチン等、一部で使用しています。
第５章でＣプログラムとのリンケージに関する注意点について説明しています。

3）リアルタイムＯＳ（ＲＸ７８Ｋ／Ⅲ）：
第３章で機能と構成について、
第４章でリアルタイム・システムの設計手順とＯＳ機能の使用例を、
第５章でリンケージに関する注意点について解説しています。

3. リアルタイムOS

3.1 リアルタイムOSとは

3.1.1 マルチ・タスク・システムとリアルタイム・システム

制御用システム上でのマルチ・タスク・システム導入の大きな意義の一つに、仕事の分担があります。

制御用システムをシングル・タスクで構築する場合、システム機能の大きさや入／出力の複雑さによりますが、難解な保守性の悪いプログラムになります。そこで、システム全体の機能を小さな仕事の単位でタスクとして細分化し、制御系システムのリアルタイム性を損なうことなく容易に実現できれば、生産性と保守性の向上が期待できます。

複数のタスクが同時に（あるタイミングでＣＰＵの実行権やシステム資源を分配しながら）動いているシステムをマルチ・タスク・システム、その機構をマルチ・タスクＯＳと言います。マルチ・タスク・システムの各タスクは、それぞれ独立したプログラムと考えることができます。

マルチ・タスク・システムにリアルタイム性を備え、タスク間やシステム外部との同期をリアルタイムに取ることができるシステムをリアルタイム・システム、その機構をリアルタイムＯＳといいます。

リアルタイム・システムの各タスクは、そのリアルタイムの必要上、それぞれ密接なつながりを持っています。しかし、それだけにタスクの機能（仕事）をできるだけ独立なものにし、そのインタフェースを単純な方法で実現することが重要になります。そのことが、生産性と保守性の向上につながります。

3.1.2 マルチ・タスクOS機構とタスクの状態

マルチ・タスクＯＳの中枢機構は、スケジューラと呼ばれるタスクへのＣＰＵ時間の分配機構です。この機構によって複数のタスク（仕事）が、（見かけ上）並行動作することができます。スケジューラは、マルチ・タスクＯＳ、リアルタイムＯＳに共通の機構です。

ある時点での動作中のタスクは一つだけです。この動作中のタスクをＲＵＮタスクといいます。それに対して、ＣＰＵの空き時間を待っているタスクをＲＥＡＤＹタスクといいます。

スケジューラとは、ＲＵＮタスクとＲＥＡＤＹタスクを合わせた中から、ある取り決めにより１つのタスクを選びだし、ＲＵＮタスクとする機構をいいます。選ばれたタスクがＲＥＡＤＹタスクだった場合は、ＲＵＮタスクの切り替えが起こることになり、これをタスクがディスパッチする（またはタスクのディスパッチング）といいます。

スケジューラは、ＲＵＮタスクの中断時、全てのレジスタの内容を保存し、次に再びＲＵＮタスクとして走らせる時に備えます。逆に、新たにスケジューラにより選択されたタスク

に対しては、全てのレジスタをこのタスクが最後に中断された時点の内容に戻してやります。

スケジューラにより保存／復元されるタスク別のレジスタ内容を、本書では以後、タスク・コンテキストと呼びます。

ＲＵＮタスクの選択方式には、ラウンド・ロビン方式や優先度方式がありますが、本節での詳述は避け、第２節「ＲＸ７８Ｋ／Ⅲ」で解説します。

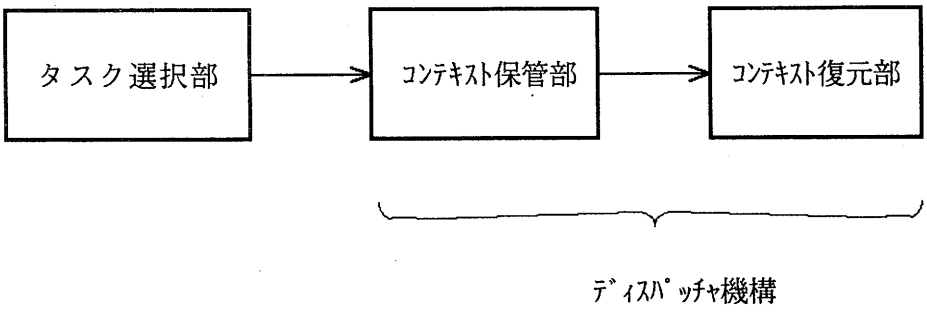


図3.1-1　スケジューラ機構

図3.1-1にスケジューラの機構を示しました。図中にあるように、スケジューラとディスパッチャという言葉は紛らわしいため、本書では以降、以下のように呼びます。

機構の名称：タスクの切り替えのみを指す場合 …… ディスパッチャ
　　タスク選択部を含む場合 …… スケジューラ

状態の名称 …… タスク・ディスパッチ

コンテキストの切り替えのために消費するＣＰＵ時間とコンテキストの保存エリアはシングル・タスクでは発生しないマルチ・タスク・システム導入により発生した時間上、メモリ（ＲＡＭ）資源上のオーバ・ヘッドとなります。
特にスケジューラが消費するＣＰＵ時間(スケジューリング時間)は、ＯＳの一つの評価基準となっています。

3.1.3 タスク間の同期

先にタスクのＲＵＮ状態とＲＥＡＤＹ状態について説明しました。

ＲＥＡＤＹ状態の意味は、スケジューラによりＣＰＵの使用権さえ与えられれば、仕事の開始、または続行ができる状態です。リアルタイム・システムでは、さらに他のタスクやシステム外部との同期を取る必要上、”待つ”という状態が必要になります。

この”待ち”の状態を、ＷＡＩＴ状態といいます。

”待ち”状態のタスクは、スケジューラのＣＰＵ割り当て対象から除外され、”待ちの解除”によりＲＥＡＤＹタスクとして復帰します。

リアルタイムＯＳの同期機構は、この”待ち”と”待ちの解除”の制御機構です。

”待ち”と”待ちの解除”によるタスク間の同期の状態には、次の２種類があります。

・資源要求／開放
・仕事待ち／依頼

１）資源要求／開放

資源対象の例として、Ｉ／Ｏに対するアクセス権を考えます。

あるＩ／Ｏへのアクセスを２つのタスクが行なう場合、通常、一方のタスクのアクセス中、他方のタスクは、そのＩ／Ｏへのアクセスを控えなければなりません。

そのための方法として、資源要求／開放の同期手段が使用されます。

Ｉ／Ｏアクセスを行なうタスクは資源要求によって、資源を獲得します。資源を獲得することができるのは１タスクだけです。そのため、資源要求時、他方が資源を獲得していた場合、要求タスクは”待ち”状態にされます。

Ｉ／Ｏアクセスを終了するタスクは資源開放を行ないます。このとき、他方が資源”待ち”であった場合、待ちの解除”が行なわれ、資源が与えられます。

このような資源管理は、リアルタイム・システムに限らず、マルチ・タスク・システムにおいても重要な課題となります。ただし、マルチ・タスク・システム上でのＩ／Ｏへのアクセス手段は、ライブラリやユーティリティとしてシステムが提供するため、ユーザ（タスク）側で意識することはほとんどありません。

２）仕事待ち／依頼

仕事待ち／依頼の簡単な例として、３つの出力装置を制御するシステムを考えます。

タスク分割は、３つの出力装置のコントローラ・タスクと、システム全体を取りまとめるメイン・タスクの４分割にします。

このシステムのタスク間のインタフェースを図3.1-2に示します。

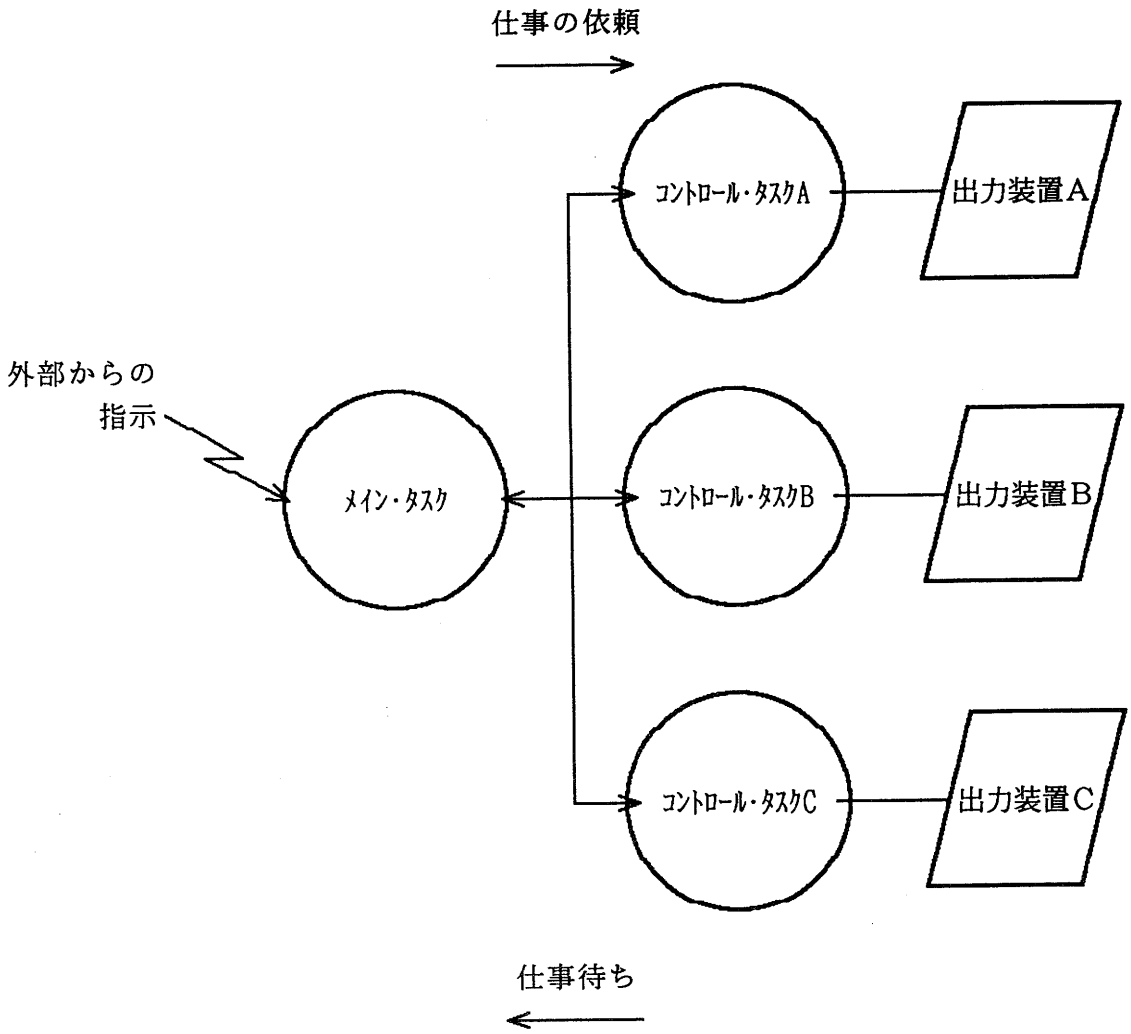


図3.1-2　仕事の依頼と待ちの例

メイン・タスクは、外部からの指示を解読し、その指示装置を受け持つコントロール・タスクへ仕事を依頼します。

A，B，Cのコントロール・タスクは通常、メイン・タスクからの仕事の”待ち”状態です。仕事の依頼が来ると、”待ちの解除”が行なわれ、担当装置への出力制御を行ないます。出力が終了すると、またメイン・タスクに対し仕事を要求します。すぐに仕事がなければ”待ち”状態になります。

メイン・タスクも通常は、外部からの指示待ち状態です。

仕事待ち／依頼はタスク間同期の基本的な考え方です。同期の状態によっては、”信号待ち／通知”、”メッセージ通信”、”事象待ち／通知”などと呼ぶこともあります。

リアルタイム・システムの実現において、タスク分割とタスク間の同期の解決は重大ポイントとなります。そのため、リアルタイムＯＳは、数種類の同期手段を提供します。それによって、様々な同期の状態を実現することができます。

同期手段は、タスクにとって外部関数の形態で提供され、資源の状態と仕事の依頼状況、及び”待ち”と”待ちの解除”がリアルタイムＯＳによって管理・制御されます。
リアルタイムＯＳによって提供されるこれらの関数をシステムコールといいます。

－”仕事”についての補足－
マルチ・タスクやリアルタイム・システム上では、しばしば”タスク＝仕事”と表現します。これは、タスクの処理を高所（システム全体から見てということです）からとらえた表現です。
それに対して、タスク間の同期の状態として”仕事のやり取り”、”仕事の指示”という表現があります。
どちらの”仕事”も重要な概念であり、他に適切な表現がないため、本書でもそのまま使用しています。

3.1.4 キュー構造とリアルタイム・システムの実現

リアルタイムＯＳの同期機構は、”待ち”と”待ちの解除”の制御機構であることについて説明しました。
最後に、リアルタイム性（非同期性）について考えます。
リアルタイム・システムでは、（主に、外部からの入力：割り込みによって）”待ち”と”待ちの解除”要求が非同期に発生し、リアルタイムに処理しなければなりません。
次に挙げるような状態です。

・ある時点で一つの資源を複数のタスクが要求した場合の制御
・現在の仕事が終わっていないのに、次の仕事の依頼があった
・複数のタスクが一つのタスクから仕事を待つ場合の制御

これらの状態を他の言葉で言い換えると、「”待ち”と”待ちの解除”が、どのような順序で発生しても対処しなければならない」ということになります。そのため、同期機構はキュー構造によって実現されます。
キュー構造は、先入れ先出しの構造です（以降ＦＩＦＯと略します）。
以下、仕事待ち／依頼の場合の同期機構について説明します。
この場合、２つのキューによって、リアルタイム性が実現されます。タスクが仕事を待つためのキューと、仕事がタスクによる要求を待つためのキューです。
図3.1-3に仕事待ち／依頼のキュー状態を示します。

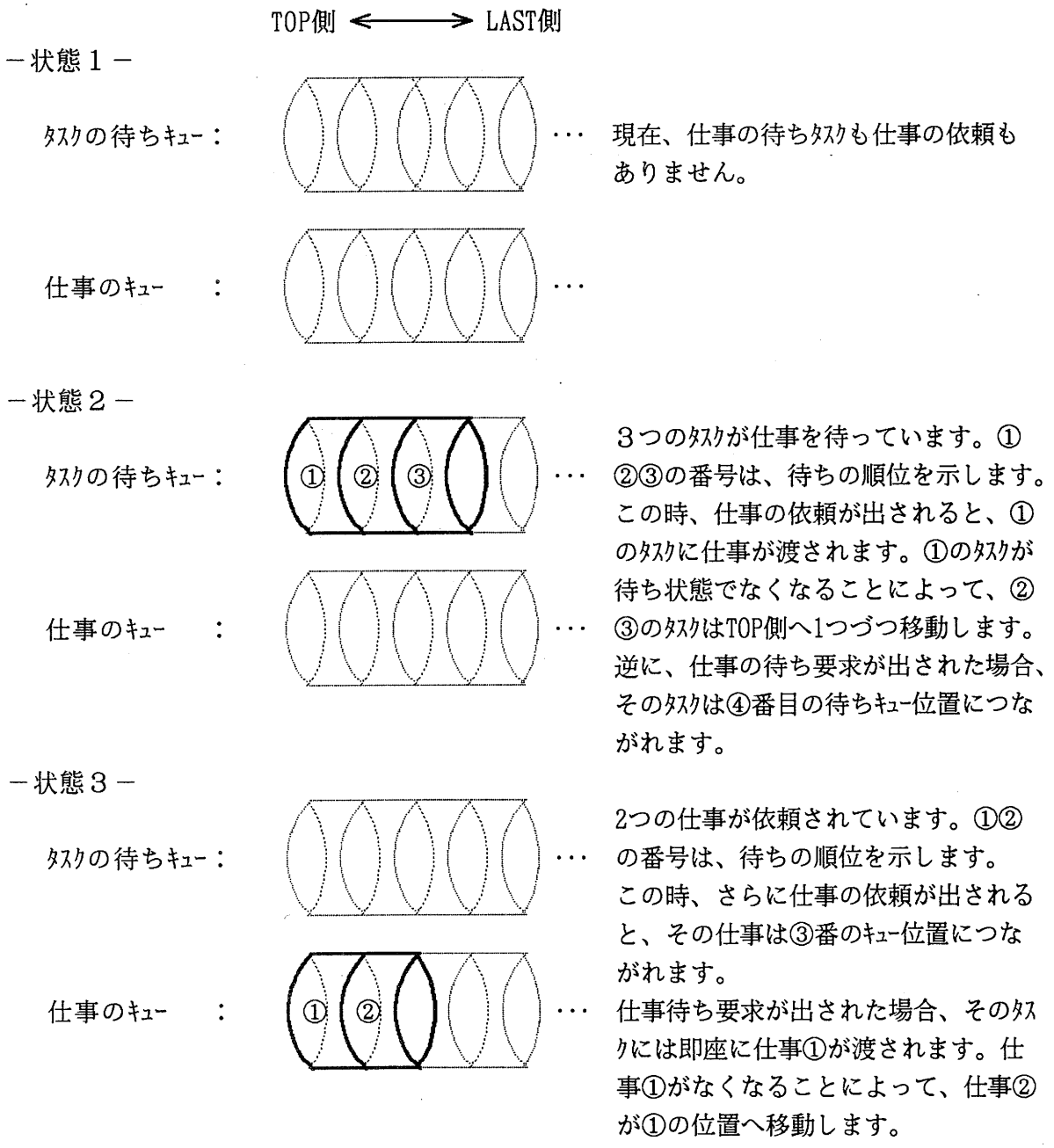


図3.1-3　同期機構のキューの状態

資源管理のための機構の場合もタスクの待ちキューは同様で、仕事のキューの代わりに資源の状態が管理されます（資源管理の機構とその状態については、「3.2 ＲＸ７８Ｋ／Ⅲ」で解説します）。

最後に、タスクの状態遷移を、ＲＸ７８Ｋ／Ⅲの場合について説明します。

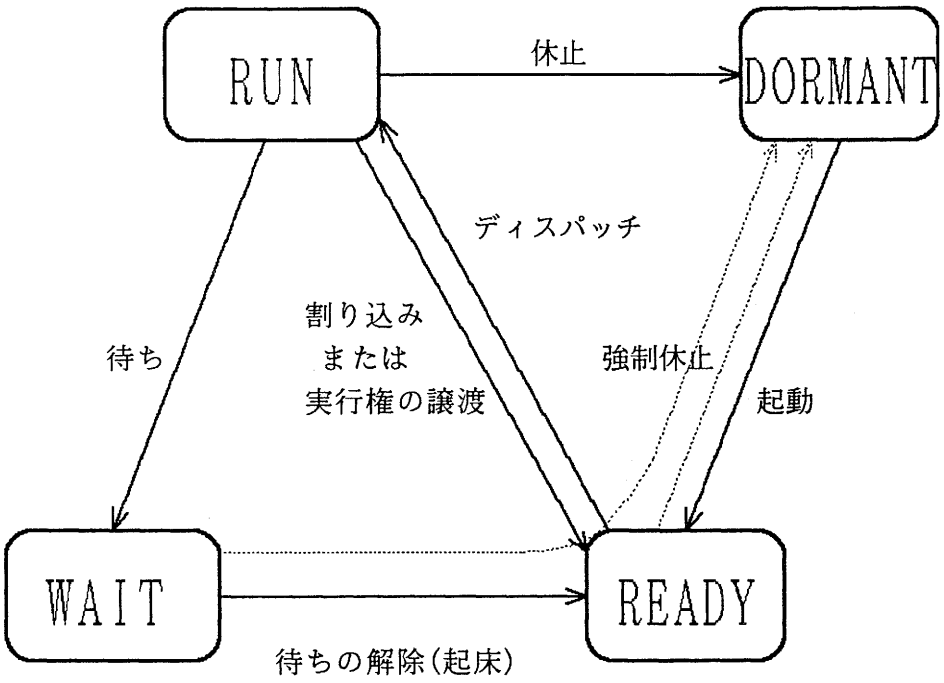


図3.1-4　ＲＸ７８Ｋ／Ⅲのタスクの状態と状態の遷移

ＷＡＩＴ状態へはＲＵＮ状態からしか遷移しません。このことは、自分自身が”待ち”を要求した時にのみＷＡＩＴになることを示しています。他のタスクから強制的にＷＡＩＴにされることはありません。

ＲＵＮ→ＲＥＡＤＹの状態遷移は、割り込み、または実行権の譲渡という特殊なシステムコールの他、タスク優先度の関係により、自身より高優先度のタスクがＲＥＡＤＹタスクとして出現したときに起こります（タスク優先度については第２節で解説します）。

ＤＯＲＭＡＮＴは、動く許可をリアルタイムＯＳからもらっていないタスクの状態です。

したがって、”休止”はシステムにとって自身が不要になった場合のタスクの死を意味します（ＷＡＩＴへの遷移と同じく自身で死を要求します）。

ただし、ＲＵＮタスクから”起動”要求が出されると生き返ることができます。この復活は、生前の状態をすっかり忘れた初期状態です。

システムのリセット直後の状態では全タスクがＤＯＲＭＡＮＴ状態です。まず１つのタスク（イニシャル・タスクといいます）がリアルタイムＯＳにより起動され、他のタスクはイニシャル・タスクから起動することによってシステムの運転がスタートします。

補足．ＲＸ７８Ｋ／Ⅲでは、ＷＡＩＴまたはＲＥＡＤＹ状態のタスクを強制的に休止させることもできます。

3.1.5 割り込み制御

リアルタイム・システムの実現に割り込みは欠くことのできない要素です。

それは、システム外部からの仕事の発生は、主に割り込みによってシステムに知らされるからです。

割り込み処理プログラムを割り込みハンドラと呼び、ハンドラとタスク間の同期手段がリアルタイムＯＳによりサポートされます。

3.2 RX78K/Ⅲ

3.2.1 構成と概要

ＲＸ７８Ｋ／Ⅲは制御機器組み込み用リアルタイムＯＳです。
リアルタイムＯＳは外部及び内部の処理要求に対し、即時処理を行なう実時間での処理速度が要求されます。
ＲＸ７８Ｋ／Ⅲは次に挙げる特徴を有します。

- リアルタイム性に優れている。
- ＲＯＭ化による機器組み込みを前提として考えられている。
- ターゲット・システムにとって不必要なＯＳのシステム機能を取り外すことができるため、使用目的に最適のＯＳを構築することができる。
- 高級言語Ｃでの開発をサポートしている。
- μＩＴＲＯＮ仕様に準拠。

ＲＸ７８Ｋ／Ⅲを用いたシステムの構成を示します。

1）ニュークリアス（核）
ＯＳの中枢機構とシステムコールのかたまりです。

2）システム初期化用リセット・ルーチン
システム自身と5）の管理ブロックの初期化、及び最初のタスク・ディスパッチを発生させます。

3）システムコール・エントリ・テーブル
システムコールの構成をＯＳに知らせるとともに、ニュークリアスへのジャンプ・テーブルの役割を持っています。

4）初期化情報テーブル（コンフィギュレーション・テーブル）
システム情報の記述テーブルで、システム・リセット・ルーチンにより使用されます。

5）オブジェクト管理用制御ブロック
タスク及び同期状態を管理するための制御ブロックで、以下に挙げるものがあります。
- タスク状態の管理ブロック
  タスク・コントロール・ブロック（ＴＣＢ）
- 同期機構のためのキュー・エリア
  イベントフラグ（ＥＶＴ）

セマフォ（ＳＥＭ）
メイルボックス（ＭＢＯＸ）
・メモリプール（後述）

６）Ｃインタフェース・ライブラリ
Ｃ言語のためのシステム機能の（システムコール）呼び出し関数です。

７）ユーザ・プログラム
タスク、割り込みハンドラ、及びリセット・ルーチンです。

１）２）によりリアルタイムＯＳが形成され、６）のＣインタフェース・ライブラリと共に提供されます。３）４）５）７）はユーザの作成部分です。
３）４）５）の作成にはコンフィギュレータが用意されています。コンフィギュレータの使用方法とその設定内容については第５章で解説します。

3.2.2 タスク・スケジューリング方式

ＲＸ７８Ｋ／Ⅲのタスク・スケジューリングは事象駆動による優先度方式です。

ここで事象とは、システムコールの発行あるいはタイマ割り込みの発生をいいます（タイマ割り込みについては、第３．２．４節「タスク付属の同期機構とタイマ・オペレーション」に後述）。優先度は、スケジューラがＲＵＮ状態にすべきタスクを選択するための、各タスクに付けられた優先順位のことです。タスク優先度は１６段階の設定が可能です。

ＲＸ７８Ｋ／Ⅲでは、タスク優先度それぞれに対して同期機構と同様のキューが設けられています。このキューをＲＥＡＤＹキューと呼びます。全てのＲＥＡＤＹタスクとＲＵＮタスクは、それぞれの属する優先度ＲＥＡＤＹキューにつながっていて、スケジューラによるＲＵＮタスクとしての選択を待ちます。

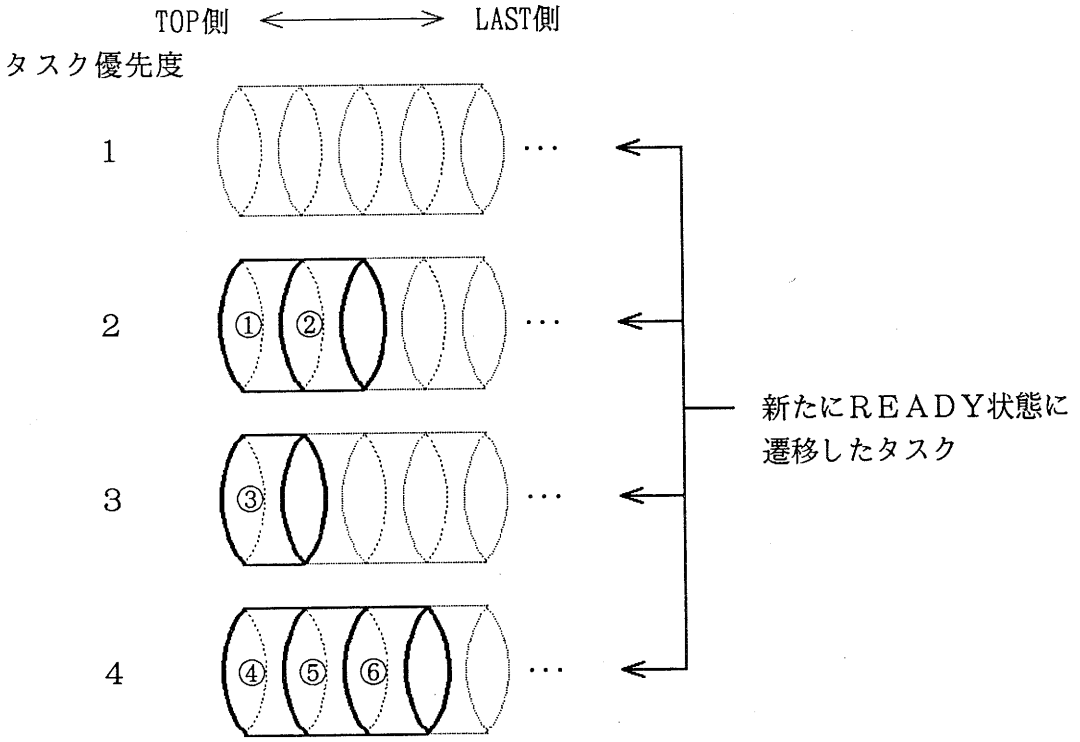


図3.2-1　ＲＥＡＤＹキューのキューイング方式

図3.2-1に、ある時点のＲＥＡＤＹキューの状態を模擬的に優先度を４段階に縮小して示しました。図中の①～⑥は、スケジューラによる選択優先順位を表わしています。最高順位のタスク①がＲＵＮタスクです。

次に、システムコールの発行による（ここではＲＵＮタスクからの）ＯＳの処理手順を示します。

１）そのシステムコール機能の実行

２）スケジューラによる、ＲＥＡＤＹキュー中最高順位タスクの選択

システムコール発行タスク(ＲＵＮタスク)は、ＲＥＡＤＹキューの最高順位タスクです。したがって、次に示す２つのケースを除き、引き続きＲＵＮタスクとしての権利を得ます。

・システムコール発行タスクがＷＡＩＴ状態になった場合
・システムコール発行タスクより高優先度のタスクが起床(または起動)された場合

システムコールによりＷＡＩＴ状態になった場合、そのタスク（ＲＵＮタスク）はＲＥＡＤＹキューからはずされます。逆に起床(または起動)されたタスクは、そのタスクの属する優先度ＲＥＡＤＹキューの最後尾につながれます。

ＲＥＡＤＹキューにつながるタスクが一つもなくなった場合、割り込みが発生するまでＣＰＵはＨＡＬＴ状態になります。この場合、割り込み処理プログラムからシステムコールを発行する（または、タイマ・オペレーション機構により）ことで、その割り込み要求に応じたタスクを起床し、システムを再開させます。

補足． ＲＸ７８Ｋ／Ⅲは、タスクの優先度を動的に変更することができ、そのためのシステムコールが用意されています。
また、この節では、タスクからのシステムコール発行について述べましたが、ＲＸ７８Ｋ／Ⅲでは、割り込み処理プログラム（割り込みハンドラ）からのシステムコール発行も可能です。
この点については、第３．２．８節で解説します。

3.2.3 同期・通信機構

ＲＸ７８Ｋ／Ⅲのタスク間同期・通信機構には、セマフォ、メイルボックス、イベントフラグの３種類があります。

セマフォは資源の排他制御と、情報の伝達が不要な仕事の受け渡しに、メイルボックスは情報の伝達を伴う仕事の受け渡しに、イベントフラグは事象発生の通知手段としてそれぞれ適した機構です。

これらは、タスクとは独立した同期・通信機構であり、仕事や資源はタスク間の直接のやり取りではなく、機構を介して行なわれます。また機構の登録は、セマフォＡ，Ｂとメイルボックス Ａ，Ｂ，Ｃ、及びイベントフラグＡといったように必要に応じて任意に設定でき、実際の仕事のやり取りは”セマフォＡに仕事を依頼する”、”セマフォＡに仕事を要求する”といった具合に行ないます。

ＲＸ７８Ｋ／Ⅲにはこの他にタスク付属の同期機構(タスク間の直接のやり取り)がサポートされています。タスク付属の同期機構については次節「タスク付属の同期機構とタイマ・オペレーション」で解説します。

1）セマフォ

セマフォは資源数と待ちキューにより構成され、P命令[wai_sem]とV命令[sig_sem]によりタスク間の同期を取る機構です。（[]中にシステムコール名を示します。以降同様。）

P命令はセマフォに対する資源要求を意味します。セマフォ資源が1以上なら、資源が与えられセマフォ資源はデクリメントされます。セマフォ資源が0の場合、渡すべき資源がないことになり、タスクは資源が得られるまで待ちキューにつながれます。

V命令は資源開放を意味します。待ちタスクが存在する場合、セマフォの資源数は0のまま変化せず、待ちキューの先頭タスクに（FIFOにより）資源が渡されます。逆に待ちタスクが存在しなかった場合、セマフォ資源がインクリメントされるだけです。

P命令、V命令の獲得／開放資源数は1に固定で、タスクは1回のP／V命令で1つの資源しか獲得／開放できません。

セマフォの状態には、図3.2-2に示す3つの状態があります。

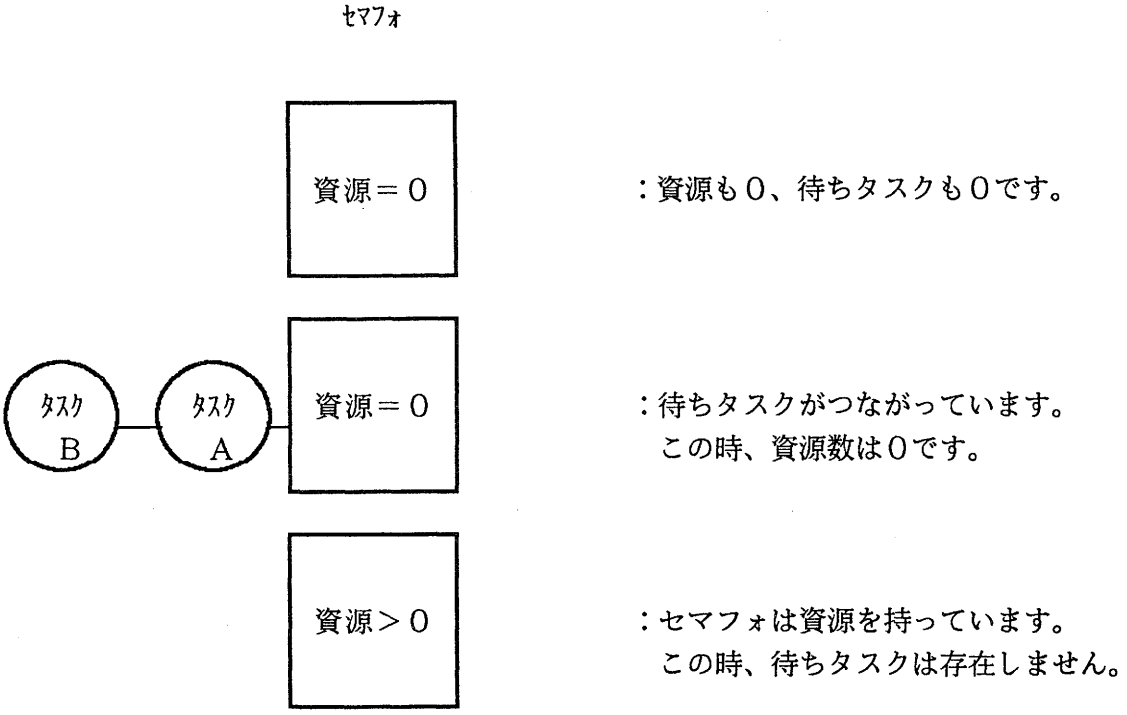


図3.2-2　セマフォの状態

一つの資源をセマフォで管理する場合、資源数の初期値を1にしておきます（コンフィギュレーション時に、初期値を1とします）。

この場合、Ｐ命令による資源要求はいつ、いかなるタスクからでも発行可能ですが、Ｖ命令による資源開放は資源獲得中のタスクから、資源開放の意味でのみ発行します。

セマフォに複数資源の管理をさせる機会はめったにありません。２つの資源をセマフォで管理する場合、セマフォは資源の有無の情報しか管理できないため、２つの資源を区別することができないからです。逆に複数の資源が全く同等の場合には有効です。

例：全く同等の２つのＩ／Ｏがあり、Ｉ／Ｏの選択もタスク側で意識しないでよい場合のＩ／Ｏアクセス権管理

セマフォを仕事の待ち／依頼に使用する場合、セマフォ資源数を現在依頼されている仕事の数と考えます。したがって、セマフォ資源数の初期値は０で、Ｐ命令により仕事の要求を、Ｖ命令により仕事の依頼を行ないます。

セマフォには情報の伝達能力はありませんから、双方のタスクが仕事の内容を知っている必要があります。したがって通常１対１に同期付けを行ない、２つのタスク間の仕事待ち／依頼専用に１つのセマフォを使用します。

セマフォは２５５個の資源の管理能力、あるいは２５５の仕事をためこむことができます。

２）メイルボックス

メイルボックスは、メッセージ・キューとタスクの待ちキューにより構成され、メッセージの送受信命令[snd_msg,rcv_msg]によりタスク間の同期を取る、情報の伝達を伴う仕事の受渡しに用いる機構です。

受信命令により、メッセージ・キューにメッセージがあれば、先頭のメッセージが（ＦＩＦＯにより）受信タスクに渡されます。メッセージがなかった場合、受信命令の発行タスクはタスクの待ちキューにつながれメッセージの送信を待ちます。

送信命令により、待ちタスクが存在すれば、待ちキューの先頭タスクに（ＦＩＦＯにより）メッセージが渡されます。待ちタスクがなかった場合、メッセージはメッセージ・キューにつながれ、受信命令を待ちます。

メイルボックスには、図3.2-3に示す３つの状態が存在します。

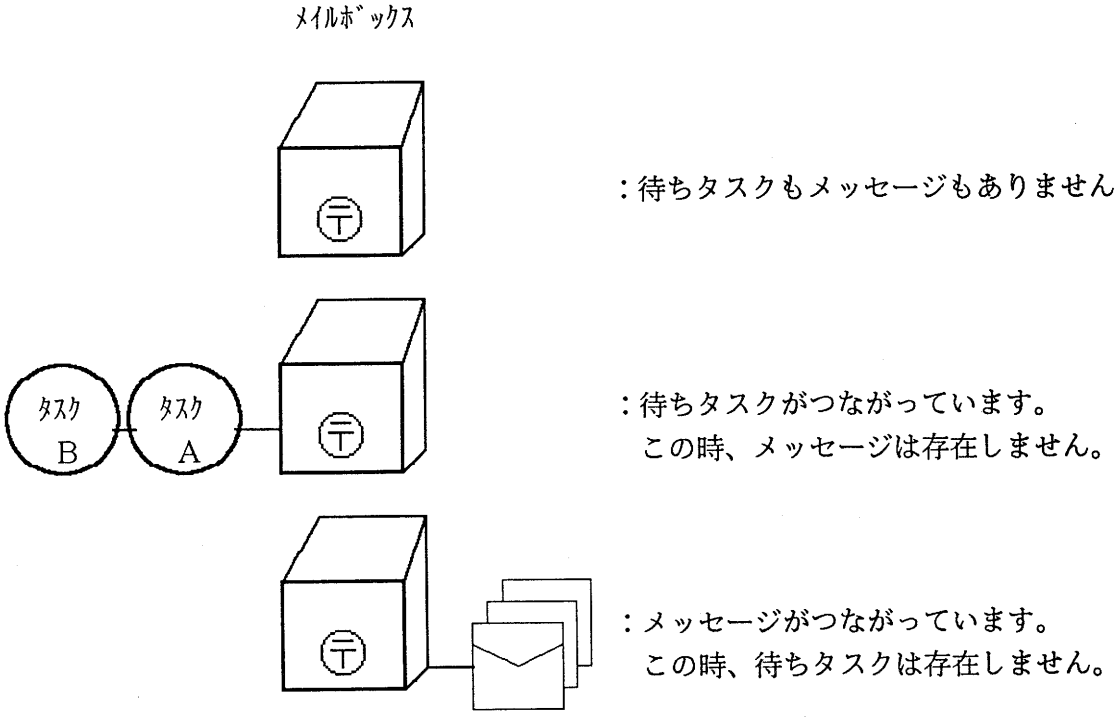


図3.2-3　メイルボックスの状態

送受信されるのは、メッセージそのものではなくメッセージの格納先頭アドレスです。そのため、メッセージ文の保護が必要になります。
メッセージの所有権は送信までは送信タスクにありますが、受信後は受信タスクのものになります。したがって、送信タスクは受信タスクが不要になるまで使用したメッセージのエリアを使用することができません。

メッセージ格納エリアをプロテクト管理するために通常、メモリブロック（３．２．７節参照）を使用します。

メイルボックスはタスクの待ちキューを持っていますが、通常１（受信側）対N（送信側）の同期付けを使用します。これは、郵便受け（メイルボックス）が、普通、誰か１人の所有物であるのと同じ理由です。

３）イベントフラグ

イベントフラグは１ビットのフラグとタスクの待ちキューにより構成され、フラグに対する待ち[wai_flg]、セット[set_flg]、クリア[clr_flg]、またはクリア付きの待ち命令[cwai_flg]によりタスク間の同期を取る、１ビットのフラグをイベントの状態に見立てたイベント待ち／通知の機構です。

待ち命令により、フラグ＝１の場合、即座にイベントの発生が通知されます。フラグ＝０

の場合、命令発行タスクは待ちキューにつながれ、イベントの発生を待ちます。待ち命令によりフラグの状態は変化しません。

セット命令はイベントの発生を通知します。すでにフラグ＝１だった場合、この命令は無視されます。フラグ＝０だった場合、キューにつながっているタスクのＷＡＩＴ状態が先頭から順番に（ＦＩＦＯで）解除され、イベントの発生が通知されていきます。ただし、クリア付きで待っているタスクがあれば、そのタスクまででＷＡＩＴ状態の解除はストップします。フラグの状態はクリア付きの待ちタスクがなければ、最終的に１、存在した場合０となります。

クリア命令は単にイベントフラグをクリアします。元のフラグの状態に関わりなくフラグ＝０にされます。

クリア付き待ち命令は待ち命令と同様ですが、イベントの通知受け付け時にフラグをクリアすることで、次のイベント発生に備えます。

イベントフラグ機構にも、３つの状態が存在します。

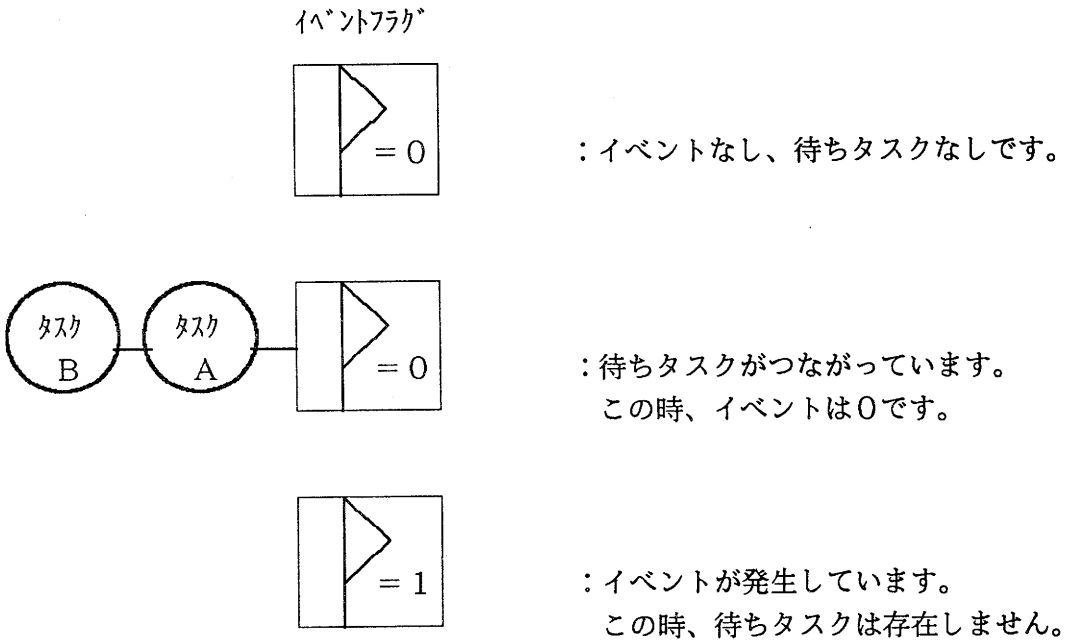


図3.2-4　イベントフラグの状態

セット命令とクリア付き待ち命令を使って１対１に同期を取る場合、セマフォの仕事待ち／依頼型の同期とほぼ同じ意味合いになります。セマフォの場合と異なる点は、セマフォが仕事を蓄えることができるのに対し、イベントフラグでは、続けざまのセット命令は１回のセットの意味しか持たない点です。

セット命令、待ち命令とクリア命令を使った場合も意味合いは同様ですが、待ち命令とク

リア命令を別にすることによりこの間のセット命令を無視することができます。
この場合、クリア命令の発行タイミングでイベントフラグを有効にするという意味を持ちます。

イベントフラグはタスクの待ちキューを持っているため、１（セット側）対Ｎ（待ち側）の同期を用いることによって複数タスクの同時起床が可能です。しかし、この場合どのタスクにいつフラグをクリアさせるかが問題となります。

Ｎ対１の同期付けも使用できます。これは複数のイベントを一本にまとめたことになり、イベントの種類は別の方法によって通知しなければなりません。

3.2.4 タスク付属の同期機構とタイマ・オペレーション

タスク付属の同期機構は、タスクが個別に起床要求カウンタと待ち時間カウンタを持つことにより実現され、２つのカウンタが各タスクの所有物である点でタスク独立の機構とは異なります。２つのカウンタは次の意味を持ちます。

・起床要求カウンタ
　セマフォの資源数に相当します。セマフォとは異なり仕事待ち／依頼型の同期付けだけに利用できます。またカウンタがタスクの所有物であるため１（待ち側：カウンタの所有者）対Ｎ（依頼側）の同期付けだけが可能です。
・待ち時間カウンタ
　タスク付属の同期機構での待ち方は２種類あります。他のタスクからの仕事の依頼を時間無制限で待つ方法と時間制限付きで待つ方法です。
　このカウンタは時間制限付きで待つ場合のタイム・リミットの指定に使います。

タスク付属の同期機構で実現できるタスク間の同期の形態は、タスク独立機構と比べて限定されますが、タスク付属機構＝ＯＳの標準機構のため軽快な機構であるといえます。

１）起床待ち／起床

待ち命令[slp_tsk]は他のタスクに対する仕事待ちを意味します。この時、自分自身の起床要求カウンタ＞０だった場合、他のタスクからすでに仕事の依頼が出されていたことになります。この場合待ち状態には移行せず、即座に仕事が与えられ、起床要求カウンタはデクリメントされます。起床要求カウンタ＝０だった場合、仕事の依頼を待ちます。
起床命令[wup_tsk]は他のタスクへの仕事の依頼を意味します。この時、依頼先タスクがslp_tskによる仕事待ちだった場合、依頼先タスクの待ち状態は解除され仕事が渡されます。依頼先タスクが待ち状態でなかった場合、依頼先の起床要求カウンタがインクリメントされ

ます。

起床待ち／起床に関して、タスクの状態は次の３状態があります。

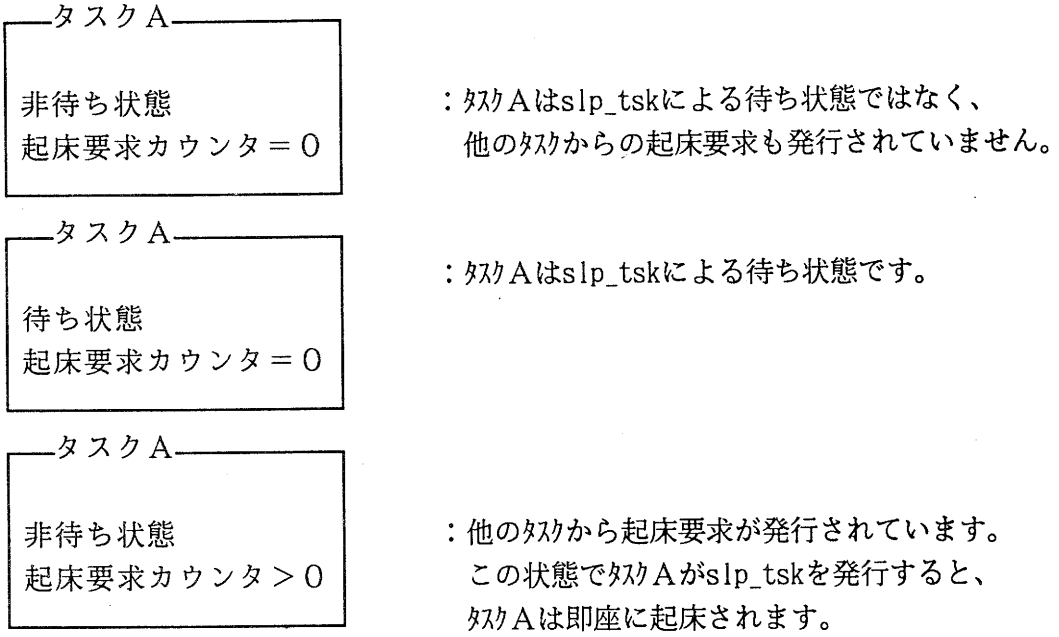


図3.2-5　起床待ち／起床に関するタスクの状態

２）タイマ・オペレーション

「起床待ち／起床」に待ち時間の制限を加えた機構です。この場合、待ち命令にはslp_tskではなくwai_tskを使用し待ち時間を指定します。ＷＡＩＴ状態の解除は２種類あります。待ち時間の経過と、他のタスクからの起床要求[wup_tsk]によるものです。どちらによって起床されたかは、wai_tskの関数値として返されます。

また、この機構を使用するにはインターバル・タイマの設定とスタート、及びベクタ・テーブルへのタイマ・ハンドラの登録が必要です。

（本書の例題システムで使用しているため詳細は第５章に後述）

待ち命令[wai_tsk]による待ち時間の指定は、インターバル・タイマの１インターバルを１カウントとしたカウント値で指定します。カウント値は、自タスクの待ち時間カウンタにセットされ、ＷＡＩＴ状態になります。その後、インターバル・タイマの割り込み周期でタイマ割り込みが発生し、カウンタがデクリメントされていきます。カウンタが０になった時点で待ちタスクは起床され、wai_tskの関数値としてタイムアウトが返されます。

タイムアウトになる前に、他のタスクから待ちタスクに対して起床要求が発行されると、この時点で待ちタスクは起床され、関数値として起床された旨が返されます。

待ち命令の発行時点で起床要求カウンタ（自タスクの）が０でなかった場合、待ち状態に

はならず、即座に関数値として起床された旨が返されます。

起床命令[wup_tsk]は、他のタスクの時間待ち状態を解除するために発行します。起床対象タスクが待ち状態の（待ち時間カウンタ＞０の）場合、対象タスクは起床されます。対象タスクが待ち状態でなかった（待ち時間カウンタ＝０）場合、対象タスクの起床要求カウンタがインクリメントされます。

待ち時間カウンタと起床要求カウンタの組み合わせは、図3.2-6に示す３種類の状態が存在します。

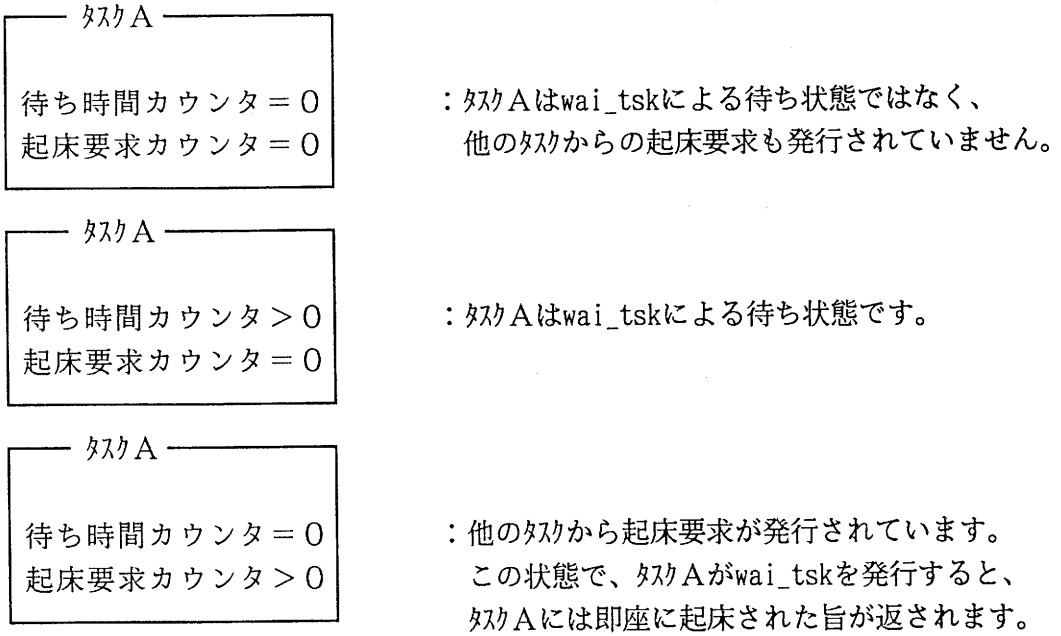


図3.2-6　タイマ・オペレーションに関するタスクの状態

タイマオペレーションの利用方法の１つは、セマフォ同等の仕事待ち／依頼型の同期付けに、待ち時間のタイムアウトによるエラー処理を行なう場合です。

もう１つは、タスクの周期起床に利用する方法です。この場合、起床命令は周期起床の停止に使用することができます。

考慮すべき点は、設定したインターバル・タイマの周期でタイマ・ハンドラが起動されるため、ＯＳのオーバ・ヘッドが増大する点です。インターバルの設定値には十分注意しなければなりません。またタイマ・オペレーションを使用しないのであれば、この機構をリンケージから除外します。

3.2.5 疑似TSS

ＴＳＳ（タイム・シェアリング・システム）は、１タスクの最大連続実行時間（タイム・スライス）を設定し、ＲＵＮタスクがタイム・スライスを使い切った時点で、強制的にタスク・ディスパッチを発生させＣＰＵの実行権を取り上げてしまう機構です。

ＴＳＳは、コンパイラやエディタの使用が主目的であるようなシステムにおいて、各端末に平等にＣＰＵの実行権を分配するための機構で、制御システムにおける必要性の少ない機構です。

そのためＲＸ７８Ｋ／Ⅲでは、負荷の少ない方法で疑似ＴＳＳと呼べるシステムを実現するためのシステムコールがサポートされています。

１）ＣＰＵ実行権の放棄命令[rot_rdq(0)]

自タスクの実行を中断し、同一優先度のＲＥＡＤＹタスクにＣＰＵ実行権を譲ります。これは、自タスクのところどころにＣＰＵの放棄命令を入れておくことで、自分自身でＣＰＵの専有状態を回避制御するのに使用します。

２）ＲＥＡＤＹキューの回転命令[rot_rdq]

この命令の発行により、低優先度のＲＥＡＤＹキューを回転させます。

この場合、低優先度のタスク群のＣＰＵの専有状態を高優先度タスクによりコントロールすることを意味します。

外部信号とこの命令の発行タイミングを同期させれば、外部信号による仕事の切り替えができます。

3.2.6 問い合わせ機能

同期機構は待ち状態の制御機構であり、待ちの形態と解除の方法により５つの機構が提供されます。

しかし、これらの機構から待ちの要素を除き、状態のみをアクセスしたい場合があります。例えば、「セマフォに現在仕事が届いていればその仕事をさせ、届いていなければ別の仕事をさせたい」という場合です。そのために問い合わせ機能がサポートされています。

問い合わせ命令は、各同期機構の待ち命令に代わる命令です。

機構に仕事やメッセージ、イベントや、起床要求が届いていれば、問い合わせ命令は待ち命令と全く同様の結果となります。仕事やメッセージが届いていなかった場合、（ＷＡＩＴ状態にはならず）エラー・リターンが返ります。どちらの場合もタスクのディスパッチは発生せず、後続の処理を続けることができます。

3.2.7 メモリ管理機能

キュー構造を使った固定長のメモリ管理機能がサポートされます。
メモリの獲得／開放対象をメモリブロック、同じサイズのメモリブロックの集まりをメモリプールといいます。

メモリ管理機能を使用するには、システムに対して、メモリプール内のメモリブロック数とブロックのサイズを登録しておかなければなりません（コンフィギュレータにより登録します）。メモリプールは複数登録することができます。
各メモリプールはキュー構造によって実現され、システムのリセット時にメモリブロックがキューイングされ、各メモリプールの全メモリブロックが使われるのを待っている状態になります。

獲得命令[pget_blk]は、メモリプールに対してメモリブロックを要求します。システムは、メモリプール・キュー先頭のメモリブロックを与えます。ただし、キューが空で、メモリブロックが全て使用中だった場合、エラーが返されます。
開放命令[rel_blk]は、メモリプールに対してメモリブロックを返却します。システムは、返却されたメモリブロックをキューの最後尾にキューイングします。
メモリ管理機能はリアルタイム・システムでのメッセージの保護になくてはならない機能です。

（第４章３節「資源管理」で詳述します）

3.2.8 割り込みに対するサポート

割り込みのリアルタイム・システムでの重要性はいうまでもありません。
ＲＸ７８Ｋ／Ⅲでは、割り込み処理ルーチンを割り込みハンドラと呼び、タスクとは独立した扱いになります。
それはハンドラの呼び出しにＯＳが介入しないということによります。しかし、割り込みの発生による事象または仕事の発生は、タスクに通知しなければなりません。そのため、割り込みハンドラ専用の同期用システムコールが用意されています。通常のタスクからのシステムコールとの違いはスケジューラを起動しない点です。
リアルタイムのレスポンスを追求すると、割り込みによる事象または仕事の発生により、すぐさま特定タスクを走らせたいという要求が生まれます。そのため、ハンドラの終了専用のシステムコールが用意されています。このシステムコールの機能は、ハンドラの終了をシステムに通知することと、スケジューラを起動することです。

ハンドラの終了後即座に目的のタスク（ハンドラから仕事を通知したタスク）を走らせるには、そのタスクの優先度を他より高く設定しておきます。

割り込みハンドラの設定は、通常ベクタ・テーブルをＲＯＭ（０番地～）に置き、固定としますが、ＲＸ７８Ｋ／Ⅲではベクタ・テーブルを内部ＲＡＭに置き、タスクからのベクタ・テーブル操作により、ハンドラの入替えを行なうなどの動的アクセスをサポートしており、そのためのシステムコールが用意されています。

静的配置か、動的アクセスかは選択することができます。

補足．ＲＸ７８Ｋ／Ⅲには、タスクの起床とハンドラの終了通知を同時に実行するシステムコールも用意されています。

－割り込みレスポンス－

リアルタイム・システムの割り込みに対するレスポンスとは、割り込みが発生してから、割り込みにより発生した仕事の最終実行タスクまでの、情報の伝達速度の良否のことです。

そのためには、

a)　割り込みをできるだけ速く受け付ける

b)　ハンドラの処理時間を短くする

c)　目的タスクにできるだけ速くスイッチする

の３点が問題となります。

c)は、ハンドラの終了によりスケジューラが呼び出されますから、目的タスクにすぐさまスイッチするかどうかは、ユーザのシステム設計次第ということになります。

b)は、ハンドラ自身の処理速度と、ハンドラで発行するシステムコールの処理時間が問題となります。

a)は、割り込み発生時、割り込み禁止状態だった場合、割り込みは割り込み許可が発行されるまでの間保留されるため、割り込み禁止の最大時間が問題となります。

第１は、ＯＳ自身の割り込み禁止部で、これは、割り込みの発生によりシステム情報が破壊されるのを防ぐためのものです。

第２に、割り込み処理中の割り込みを受付けるかどうかで、必要であれば多重割り込み処理により、低優先度の割り込み処理中の高優先度割り込みを受付けます。

特に、ＯＳ自身の最大割り込み禁止時間は、スケジューリング時間同様、ＯＳの性能評価基準の一つとなります。

保守／廃止

3.2.9 システムコール機能一覧 ……表3.2-1

1）タスク関連

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
sta_tsk	タスクの起動	対象ﾀｽｸ DORMANT→READY	○	○	
ext_tsk	自タスクの休止	自ﾀｽｸ→DOMANT	○	○	
ter_tsk	他タスクの強制休止	指定ﾀｽｸ→DOMANT	○	○	
chg_pri	タスク優先度の変更	なし	○	○	
rot_rdq(0)	実行権の譲渡	自ﾀｽｸ→READY(ｷｭｰ･ﾗｽﾄへ)	○	○	
rot_rdq	ＲＥＡＤＹキューの回転	指定優先度READYｷｭｰを回転 （READYｷｭｰ･ﾄｯﾌﾟのﾀｽｸをﾗｽﾄへ）	○	○	
tsk_sts	タスクの状態を見る	なし	○	○	
slp_tsk	起床待ち	起床要求ｶｳﾝﾄ=0:自ﾀｽｸ→WAIT >0:起床要求ｶｳﾝﾄ--	○	○	
wai_tsk	時間待ち	起床要求ｶｳﾝﾄ=0:自ﾀｽｸ→WAIT >0:起床要求ｶｳﾝﾄ--	○	○	
wup_tsk	起床／時間待ちﾀｽｸを起床	対象ﾀｽｸ=起床待ち／時間待ち: 対象ﾀｽｸ→READY その他　　:起床要求ｶｳﾝﾄ++	○	○	
can_wup	起床要求カウントのクリア	対象ﾀｽｸの起床要求ｶｳﾝﾄ←0	○	○	

*1 スケジューラの起動
*2 タスク部からの使用の可否

2）同期・通信他

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
set_flg	イベントの通知	待ちﾀｽｸあり:全待ちﾀｽｸ→READY *4 ﾌﾗｸﾞ ← 1 待ちﾀｽｸなし:ﾌﾗｸﾞ ← 1	○	○	
clr_flg	フラグのクリア	ﾌﾗｸﾞ ← 0	○	○	
wai_flg	イベント待ち	ﾌﾗｸﾞ=0 :自ﾀｽｸ→WAIT ﾌﾗｸﾞ=1 :なし	○	○	
cwai_flg	イベント待ち（排他）	ﾌﾗｸﾞ=1 :ﾌﾗｸﾞ←0 ﾌﾗｸﾞ=0 :自ﾀｽｸ→WAIT *4	○	○	
sig_sem	資源開放（Ｖ命令）	待ちﾀｽｸあり:待ちﾀｽｸ(TOP)→READY なし:ﾘｿｰｽ++	○	○	
wai_sem	資源待ち（Ｐ命令）	ﾘｿｰｽ>0 :ﾘｿｰｽ-- ﾘｿｰｽ=0 :自ﾀｽｸ→WAIT	○	○	
snd_msg	メッセージ送信	待ちﾀｽｸあり:待ちﾀｽｸ(TOP)→READY なし:ﾒｯｾｰｼﾞ・ｷｭｰへｷｭｰｲﾝｸﾞ	○	○	
rcv_msg	メッセージ受信	ﾒｯｾｰｼﾞあり:ｷｭｰ・ﾄｯﾌﾟのﾒｯｾｰｼﾞ受信 なし:自ﾀｽｸ→WAIT	○	○	

*4 set_flgにより待ちが解除されるのは、待ちキュー先頭からcwai_flgで待っているタスクまでです。この場合最終的なフラグの状態は0になります。

3）問い合わせ機能

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
wai_tsk(0)	起床要求の問い合わせ	起床要求ｶｳﾝﾄ>0:起床要求ｶｳﾝﾄ--	○	○	
pol_flg	イベントの問い合わせ	なし	○	○	
cpol_flg	イベントの問い合わせ （クリア付き）	ﾌﾗｸﾞ=1:ﾌﾗｸﾞ←0	○	○	
preq_sem	資源獲得	ﾘｿｰｽ>0:ﾘｿｰｽ--	○	○	
prcv_msg	メッセージ受信	ﾒｯｾｰｼﾞあり:ｷｭｰ･ﾄｯﾌﾟのﾒｯｾｰｼﾞ受信	○	○	

4）メモリ管理機能

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
pget_blk	メモリブロック獲得	ﾌﾞﾛｯｸあり:ｷｭｰ･ﾄｯﾌﾟのﾌﾞﾛｯｸ獲得	○	○	
rel_blk	メモリブロック返却	ﾌﾞﾛｯｸをﾒﾓﾘﾌﾟｰﾙへｷｭｰｲﾝｸﾞ	○	○	

5）割り込みハンドラ専用システムコール

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
ichg_pri	タスク優先度の変更	chg_priと同じ			○
irot_rdq	ＲＥＡＤＹキューの回転	rot_rdqと同じ			○
iwup_tsk	待ちタスクの起床	wup_tskと同じ			○
iset_flg	イベントの通知	set_flgと同じ			○
isig_sem	資源開放（Ｖ命令）	sig_semと同じ			○
isnd_msg	メッセージ送信	snd_msgと同じ			○
ret_int	割り込み処理の終了	なし	○		○
ret_wup	割り込み処理復帰と タスクの起床	wup_tskと同じ	○		○

6）その他

ｼｽﾃﾑｺｰﾙ名	機　能	状態の遷移	*1	*2	*3
def_int	割り込みベクタ・テーブル の動的設定／解除	なし	○	○	
get_ver	ＯＳのバージョンを得る	なし	○	○	

3.2.10 システムコール・インタフェース一覧 ……表3.2-2

1）Cインタフェース

(1/2)

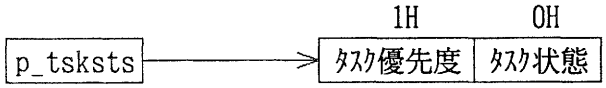
Cﾌﾟﾛﾄﾀｲﾌﾟ	IDNo.	ｴﾝﾄﾘ･ｱﾄﾞﾚｽ
char sta_tsk(unsigned short tskid);	0	40H
void ext_tsk();	1	40H
char ter_tsk(unsigned short tskid);	2	40H
char chg_pri(unsigned short tskid, char tskpri);	3	40H
char rot_rdq(char tskpri);	4	40H
char tsk_sts(unsigned short *p_tsksts, unsigned short tskid);	5	40H
char slp_tsk();	6	40H
char wai_tsk(unsigned short tmout);	7	40H
char wup_tsk(unsigned short tskid);	8	40H
char can_wup(unsigned short *p_wupcnt, unsigned short tskid);	9	40H
char set_flg(unsigned short flgid);	10	40H
char clr_flg(unsigned short flgid);	11	40H
char wai_flg(unsigned short flgid);	12	40H
char cwai_flg(unsigned short flgid);	13	40H
char pol_flg(unsigned short flgid);	14	40H
char cpol_flg(unsigned short flgid);	15	40H
char sig_sem(unsigned short semid);	16	40H
char wai_sem(unsigned short semid);	17	40H
char preq_sem(unsigned short semid);	18	40H
char snd_msg(unsigned short mbxid, void *pk_msg);	19	40H
char rcv_msg(void **ppk_msg, unsigned short mbxid);	20	40H
char prcv_msg(void **ppk_msg, unsigned short mbxid);	21	40H
char pget_blk(void **p_blk, unsigned short mplid);	22	40H
char rel_blk(unsigned short mplid, void *blk);	23	40H
char def_int(char intno, unsigned short inthdr);	24	40H
char get_ver(void **pk_ver);	25	40H

Cﾌﾟﾛﾄﾀｲﾌﾟ	IDNo.	ｴﾝﾄﾘ･ｱﾄﾞﾚｽ
char ichg_pri(unsigned short tskid, char tskpri);	-	42H
char irot_rdq(char tskpri);	-	44H
char iwup_tsk(unsigned short tskid);	-	46H
char iset_flg(unsigned short flgid);	-	48H
char isig_sem(unsigned short semid);	-	4AH
char isnd_msg(unsigned short mbxid, void *pk_msg);	-	4CH

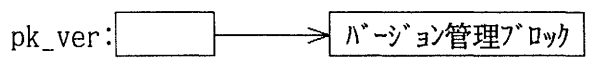
＊ IDNo.とｴﾝﾄﾘ･ｱﾄﾞﾚｽはｱｾﾝﾌﾞﾗからのｼｽﾃﾑｺｰﾙ発行のために示されていますが、
Cｿｰｽからの発行では意識する必要はありません。

－　パラメータ解説　－

tskid　　：ﾀｽｸID（TCB先頭ｱﾄﾞﾚｽ）
tskpri　：ﾀｽｸ優先度
p_tsksts：ﾀｽｸ状態を格納するｴﾘｱのｱﾄﾞﾚｽ



tmout　　：ﾀｲﾑｱｳﾄ指定値
p_wupcnt：起床要求ｶｳﾝﾀを格納するｴﾘｱのｱﾄﾞﾚｽ
flgid　　：ｲﾍﾞﾝﾄﾌﾗｸﾞID（EVT先頭ｱﾄﾞﾚｽ）
semid　　：ｾﾏﾌｫID（SEM先頭ｱﾄﾞﾚｽ）
mbxid　　：ﾒｲﾙﾎﾞｯｸｽID（MBOX先頭ｱﾄﾞﾚｽ）
pk_msg　：送信ﾒｯｾｰｼﾞの先頭ｱﾄﾞﾚｽ
ppk_msg：受信ﾒｯｾｰｼﾞの先頭ｱﾄﾞﾚｽを格納するｴﾘｱのｱﾄﾞﾚｽ
mplid　　：ﾒﾓﾘﾌﾟｰﾙID（ﾒﾓﾘﾌﾟｰﾙ先頭ｱﾄﾞﾚｽ）
p_blk　　：ﾒﾓﾘﾌﾞﾛｯｸの先頭ｱﾄﾞﾚｽを格納するｴﾘｱのｱﾄﾞﾚｽ
blk　　：ﾒﾓﾘﾌﾞﾛｯｸの先頭ｱﾄﾞﾚｽ
intno　　：割り込み番号（ﾍﾞｸﾀ･ﾃｰﾌﾞﾙ･ｱﾄﾞﾚｽ）
inthdr　：割り込みﾊﾝﾄﾞﾗ･ｱﾄﾞﾚｽ
pk_ver　：ﾊﾞｰｼﾞｮﾝ管理ﾌﾞﾛｯｸの先頭ｱﾄﾞﾚｽを格納するｴﾘｱのｱﾄﾞﾚｽ



2）アセンブラ・インタフェース

・タスク用システムコール

```
PUSH    第2ﾊﾟﾗﾒｰﾀ
PUSH    第1ﾊﾟﾗﾒｰﾀ
PUSH    IDNo.
CALLT   [40H]
POP     IDNo.
POP     第1ﾊﾟﾗﾒｰﾀ
POP     第2ﾊﾟﾗﾒｰﾀ
```

・割り込みハンドラ用システムコール

ｼｽﾃﾑｺｰﾙ	ｲﾝﾀﾌｪｰｽ

ICHG_PRI	MOVW DE,tskid
	MOV A,tskpri
	CALLT [42H]

IROT_RDQ	MOV A,tskpri CALL [44H]
IWUP_TSK	MOVW DE,tskid CALLT [46H]
ISET_FLG	MOVW HL,flgid CALLT [48H]
ISIG_SEM	MOVW HL,semid CALLT [4AH]

ISND_MSG	MOVW HL,mbxid
	MOVW DE,pk_msg
	CALLT [4CH]

RET_INT	BR !RET_INT
RET_WUP	MOVW DE,tskid BR !RET_WUP

3）関数値

sta_tsk	E_OK:正常終了、	E_NODMT :対象ﾀｽｸがDORMANTでない
ter_tsk	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT
chg_pri	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT
wai_tsk	E_OK:他ﾀｽｸによる起床、	E_TMOUT :ﾀｲﾑｱｳﾄ
wup_tsk	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT、
		E_QOVR :起床要求ｶｳﾝﾄのｵｰﾊﾞﾌﾛｰ
can_wup	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT
wai_tsk(0)	E_OK:起床要求あり、	E_TMOUT :起床要求なし
pol_flg	E_OK:ｲﾍﾞﾝﾄあり、	E_PLFAIL:ｲﾍﾞﾝﾄなし
cpol_flg	E_OK:ｲﾍﾞﾝﾄあり、	E_PLFAIL:ｲﾍﾞﾝﾄなし
sig_sem	E_OK:正常終了、	E_QOVR :ﾘｿｰｽのｵｰﾊﾞﾌﾛｰ
preq_sem	E_OK:資源獲得、	E_PLFAIL:資源なし
prcv_msg	E_OK:ﾒｯｾｰｼﾞ受信、	E_PLFAIL:ﾒｯｾｰｼﾞなし
pget_blk	E_OK:ﾌﾞﾛｯｸ獲得、	E_PLFAIL:ﾒﾓﾘﾌﾞﾛｯｸなし
def_int	E_OK:正常終了、	E_CTX :ｺﾝﾃｷｽﾄ･ｴﾗｰ
ichg_pri	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT
iwup_tsk	E_OK:正常終了、	E_DMT :対象ﾀｽｸがDORMANT、
		E_QOVR :起床要求ｶｳﾝﾄのｵｰﾊﾞﾌﾛｰ
isig_sem	E_OK:正常終了、	E_QOVR :ﾘｿｰｽのｵｰﾊﾞﾌﾛｰ

＊ 他のシステムコールは関数値としてE_OKのみが返ります。
＊＊ アセンブラでの関数値の出力レジスタはＣレジスタです。

```
E_OK    =       0
E_NODMT =       1
E_DMT   =       2
E_QOVR  =       3
E_TMOUT =       4
E_PLFAIL=       5
E_CTX   =       6
```

4. 機能/インタフェース設計

本章では、例題システムについて、システム機能とシステムＩ／Ｏ設計終了後からコーディング直前までの設計段階を、実際の設計手順に従って解説します。

4.1 タスク分割

先に、制御系システムでのマルチ・タスク化の意義について触れました。
最終的にはリアルタイム・システムを実現しなければなりませんが、最初からリアルタイム性を意識したタスク分割を行なうことには、かなり無理があります（特にシステム機能が大きい場合）。
そこで本書では、まずシステムのマルチ・タスク化（モジュール分割）を行ない、モジュール間のインタフェースを決定した上で、最終的にリアルタイム・システムを実現させる方法を取ります。

以下に、システムの条件となるシステムＩ／Ｏとシステム機能、及び基本方式を示します。

1）Ｉ／Ｏ
- ステッピング・モータ（１個）
  ポート０に接続、励磁信号を周期的に与えることによりステップ動作させる。
- ＬＥＤ（１個）
  ポート３０に接続、ステッピング・モータの回転速度をモニタする。
- チャタレス・キー（３個）
  ポート２１～２３に接続、モータ制動のための指令キーとする。

2）システム機能
- モータ機能
  キー指令による設定速度で定速度回転し、段階的に５レベルの速度設定・運転を行なう。
- ＬＥＤ機能
  モータ回転速度を点滅スピードにより表示。
  モータ停止中、ＬＥＤは消灯。
- キー機能
  運転キーにより、運転の開始／停止を行なう。
  運転中のＵＰキーにより、回転速度を上昇させる。
  運転中のＤＯＷＮキーにより、回転速度を下降させる。

３）基本方式

タイマ・オペレーションのタスクの周期起床を使って、モータ駆動のためのステップ・パルスを発生させる。

ＬＥＤの点滅周期はモータ駆動パルスを流用し、さらに何分周かすることにより得る。

システム機能をＩ／Ｏ別に書きましたが、基本方式に対して問題がなければ、Ｉ／Ｏ別の機能分割が、最もよいモジュール分割の方法となります。

一つにはＩ／Ｏに対するアクセスを１つのモジュールに極所化することができ、Ｉ／Ｏアクセス権という資源問題が限定される点が挙げられます。しかも、機能としてのまとまりは非常に良いことが期待できます。

ここでいうモジュール分割は、制御対象によるシステムの分割です。

モジュールの対象として、Ｉ／Ｏの他に、ソフトウェア的なものでも、ある種のコントローラや、事象に関する制御部（ログなど）なども考えられます。

モジュールは、機能としてまとまっていることが条件です。

まとまりの良さにより、各モジュールは他をあまり意識することなく構築可能となり、かつ、モジュール間のインタフェースも単純に実現可能となります。

タスク分割の方法としては、様々なやり方があると考えられます。ここではまず、システム機能をモジュール単位に分割することで、システムのマルチ・タスク化を行ないます。

モジュール分割による方法は、そのまとまりの良さから、各モジュールの設計作業を早期から同時進行できるという効果を生みます。

第２段階として、モジュール間のインタフェースを決定します。これは、できるだけ単純に、仕事を依頼するといった形で実現します。

最後に、各モジュール内部のインタフェース設計と、資源問題の解決を行ないます。必要であればモジュール内部のタスク分割と、タスク間で共通に使用するような関数のシステム・ツールとしてのユーティリティ化を行ないます。

以下に、本システムのモジュール分割と、モジュール機能一覧を示します。

ﾓｼﾞｭｰﾙ	機　　能
システム制御部	・キーによる制動指示受け付け ・システムの状態掌握 ・Ｉ／Ｏ制御部への制動指示
Ｉ／Ｏ制御部	・モータ制御 ・ＬＥＤ制御
ｲﾆｼｬﾙ･ｾｯﾄ･ｱｯﾌﾟ	・インターバル・タイマとキー割り込みのマスク解除 ・他の全タスクを起動

表4.1-1　モジュール機能一覧

システム制御部からＩ／Ｏ制御部への制動指示情報は、次の２つとします。

・運転の開始／停止／速度変更かの制動指示情報
・回転速度のレベル

Ｉ／Ｏ制御部は、システム制御部からの指示に従ってモータとＬＥＤを制御するだけです。
ＬＥＤとモータの制御部を分割しなかったのは、ＬＥＤがモータのモニタという付属的な位置にあるためです。

4.2 タスク間インタフェース設計

本節では、各モジュール内部とモジュール間のインタフェースを設計していきます。

4.2.1 インタフェースの決定条件

インタフェースを決定するための条件となるのは、システムのリアルタイムに関する条件と資源問題です。

この時点で考慮する資源は、Ｉ／Ｏなどのシステムの最重要資源だけで、タスク間の共通のメモリ資源など細かい点についてはここでは考慮しません。

以下に、システムのリアルタイム条件と、現時点での資源問題を示します。

－リアルタイム条件－

・モータの安定定速度回転を得る
・ＬＥＤをモータの回転と同期して点滅させる
・運転の開始／停止／ＵＰ／ＤＯＷＮ指示に迅速に対応する
・キー入力エラーによってモータの安定性に影響を与えない

－資源問題－

・モータ・アクセス権
・ＬＥＤアクセス権

リセット後と、停止後の運転再開時のモータ回転をどうするかという問題がありますが、ここで次のように決定します。

リセット時：　３段回目のスピードで運転
再開時　　：　停止前のスピードで運転を再開

キー入力エラーは次に示す３種類です。

・運転停止中のＵＰまたはＤＯＷＮキー押下
・最高速運転中のＵＰキー押下
・最低速運転中のＤＯＷＮキー押下

キー入力エラーは無視しますが、その処理によりできるだけ（または全く）モータの安定性を失わないよう考慮します。

資源問題は、出力Ｉ／Ｏの操作権だけです。

最後に制限問題があります。使用できるＲＯＭエリア・サイズが小さい場合、使用するシステムコールを制限することを考えなければなりません。

今回は例題システムであり、また、μPD78322の16kバイトの内蔵ＲＯＭ上にリアルタイムＯＳの全システムコールを含めても十分納まるため、システムコールの制限は行なっていません。ただし、最終的に使用しなかったシステムコールはリンケージから除外します。（第５章でリンクの方法を説明します）

4.2.2 I/O制御部

Ｉ／Ｏ制御部は、その制御状態でみると次の２機能を持ちます。

・定速運転機能
・運転速度の制動機能

定速運転機能は、システムのメイン・パス機能となります。そのため、まず定速運転機能を設計し、次に制動機能を付け足す作業を行ないます。

１）定速運転機能

定速運転機能の機能要求は、次の２点です。

・モータを定速回転させる
・ＬＥＤによりモータ回転速度を点滅表示する

リアルタイムの条件となるのは、次の２点です。

・モータ回転速度の安定
・ＬＥＤ点滅のモータへの同期

モータを定速度回転させるためには、モータへ与える励磁信号の出力間隔を一定に保たなければなりません。

出力周期を作るにはインターバル・タイマを利用することになります。今回は、ベクタ割り込みやマクロ・サービスを使って割り込み処理ルーチンから信号を出力する方法ではなく、リアルタイムＯＳのタスクの周期起床を使用し、モータ駆動タスクとして信号出力させます。

周期起床機能を利用することによって、インターバル・タイマの設定や、割り込みマスク・フラグの制御をタスク側で気にする必要がなくなります。
また、タイマ・オペレーションはリアルタイムＯＳの標準的な機構であるため、他のデバイスへの移植効率が高くなります。

ＬＥＤの点滅は、モータ駆動タスクから励磁信号の出力周期で信号をもらい、その時間間隔をｍ倍することで点灯時間を、ｎ倍することで消灯時間を得ます。

以上により、Ｉ／Ｏ制御部の定速運転機構のインタフェースが完成します。

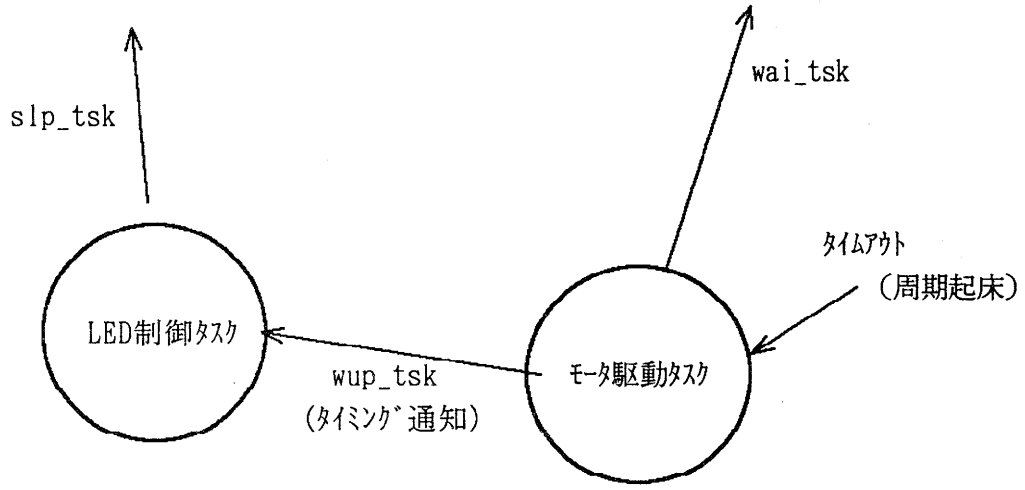


図4.2-1　メイン・パスのインタフェース

モータ駆動タスクからＬＥＤ制御タスクへの周期信号にｗｕｐ＿ｔｓｋを使用しましたが、セマフォを使用することもできます。

機能要求とリアルタイム条件を満足するには、モータ駆動タスクはタイマの１インターバル内に全ての仕事を終え、ｗａｉ＿ｔｓｋを発行する必要があります。また、ＬＥＤ制御タスクはモータ駆動タスクからのタイミング通知より先に、待ち命令を発行する必要があります。

従って、２つのタスクの１サイクルの持ち時間は、次の通りになります。

モータ駆動タスク　：インターバル・タイマの１周期
ＬＥＤ制御タスク　：モータへの励磁信号周期
（＝インターバル・タイマの数周期）

２つのタスクの優先度は、次の様にします。

タスク優先度：　モータ駆動タスク　≧　ＬＥＤ制御タスク

このことにより、ｗｕｐ＿ｔｓｋに伴うスケジューラの起動で、引き続きモータ駆動タスクが実行権を得られます。ＬＥＤ制御タスクは、モータ駆動タスクのＷＡＩＴ状態のときに動くことになり、その分、モータ駆動タスクの持ち時間が多くなります。

モータ駆動タスクとＬＥＤ制御タスク自体の処理時間をどちらも１００μｓ、システムコール１回とタイマ・ハンドラの処理時間を１５０μｓとすると以下のことがいえます。

インターバル・タイマの最小設定周期　＝　１５０×３＋１００μｓ
（＝モータ駆動タスクが動ける最小時間）

励磁信号の最小出力周期　＝　１５０×４＋１００＋１００μｓ
（＝メイン・パスが動ける最小時間）

この数値はメイン・パスが動作することのできるぎりぎりの値です。そのため、他に低優先度のタスクがある場合、ある程度の余裕が必要になります。

（定速運転のタイミング考証は第４節で行ないます）

２）制動機能

次に、モータ回転速度の制動時のインタフェースを考えます。

キー指令による運転速度の変更、または運転の開始／停止指示は、システム制御部からメッセージとして受け取ります。ここで、最高速運転中のＵＰキー押下などのキー入力エラーについては、システム制御部で排除されていることとします。

モジュール間の仕事の分担をどうするかという問題ですが、モジュール分割の考えを導入した場合、モジュール間の相互依存はできるだけ排除します。システム制御部は、キー指令への対応とシステム状態の掌握が担当であり、キー入力エラーなどで他のモジュールへ迷惑を及ぼさないようにします。その反面、システム制御部が掌握すべき情報は”運転中か、停止中であるか”ということと、”現在何段階目の速度が選ばれているか”という点だけで、モータ駆動タスクの待ち時間情報などは扱うべきでありません。

システム制御部からの情報は、次の２つです。

・運転の開始／停止／速度変更かの制動指示情報
・回転速度のレベル

Ｉ／Ｏ制御部は、５段階の速度レベルに応じた、ｗａｉ＿ｔｓｋシステムコールへの待ち時間の設定値テーブルを持っている必要があります。

モータ駆動タスクはモータの回転中、時間待ち状態であるため、システム制御部からのメッセージを直接受け取ることができません。そのため、システム制御部からのメッセージのレシーバ・タスクが必要となります（以後、以下の理由によりメイン・タスクと呼びます）。

Ｉ／Ｏ制御部とシステム制御部間のインタフェースをメッセージの送受信だけでなく、モータ駆動タスクへの起床要求付きにすればよいではないかという疑問を抱かれるかも知れません。しかし、この場合、起床要求はモータの回転の停止を意味し、他のモジュール（システム制御部）からの発行は好ましくありません。モジュール間のインタフェースでこういった機能の侵略をしてしまうと、（モータ制御機能の増大といった）システム機能の変更に耐えられなくなるからです。

メイン・タスクはシステム制御部からのメッセージを受信し、その制動指示により、以下の処理を行ないます。

ａ）速度変更の場合

Ｉ：モータ駆動タスクへの起床要求により、運転停止を指示する。

モータ駆動タスクは、運転停止後、新しい待ちカウントを受け取るためにメッセージ待ちを発行する。

Ⅱ：指示速度レベルをｗａｉ＿ｔｓｋの待ちカウントに変換し、モータ駆動タスクへメッセージ送信する。

モータ駆動タスクは、新しい待ちカウントの受信により、運転を再開する。
この間、ＬＥＤ制御タスクは、モータ駆動タスクからの信号がこないために待ち状態のままで、モータ駆動の再開により、必然的にＬＥＤの点滅周期が変更される。

ｂ）停止の場合

Ｉ：モータ駆動タスクへの起床要求により、モータとＬＥＤ点滅を停止させる。

モータ駆動タスクは、運転停止後、次回の運転開始のために、待ちカウント受け取りのためのメッセージ待ちを発行する。
ＬＥＤ制御タスクは、モータ駆動タスクからの信号待ちのままである。

Ⅱ：ＬＥＤを強制消灯する。

（ＬＥＤ制御タスクはモータ駆動タスクからの信号待ち状態であるため、メイン・タスクから直接ＬＥＤを操作します。）

ｃ）運転開始の場合

Ｉ：指示速度レベルをｗａｉ＿ｔｓｋの待ちカウントに変換し、モータ駆動タスクへメッセージ送信する。

モータ駆動タスクは、待ちカウントの受信により、運転を再開する。
ＬＥＤ制御タスクは、モータ駆動タスクからの信号通知の再開により、点滅を開始する。

リセット時の運転開始処理と、停止後の運転再開処理を同じにするため、Ｉ／Ｏ制御部の各タスクの初期起動時の状態を、次の通りにします。

ＬＥＤ制御タスク　：モータ駆動タスクからの信号待ち
モータ駆動タスク　：待ちカウントの指示メッセージ待ち
メイン・タスク　　：システム制御部からの制動指示待ち

以下、Ｉ／Ｏ制御部各タスクの処理概要フローを示します。

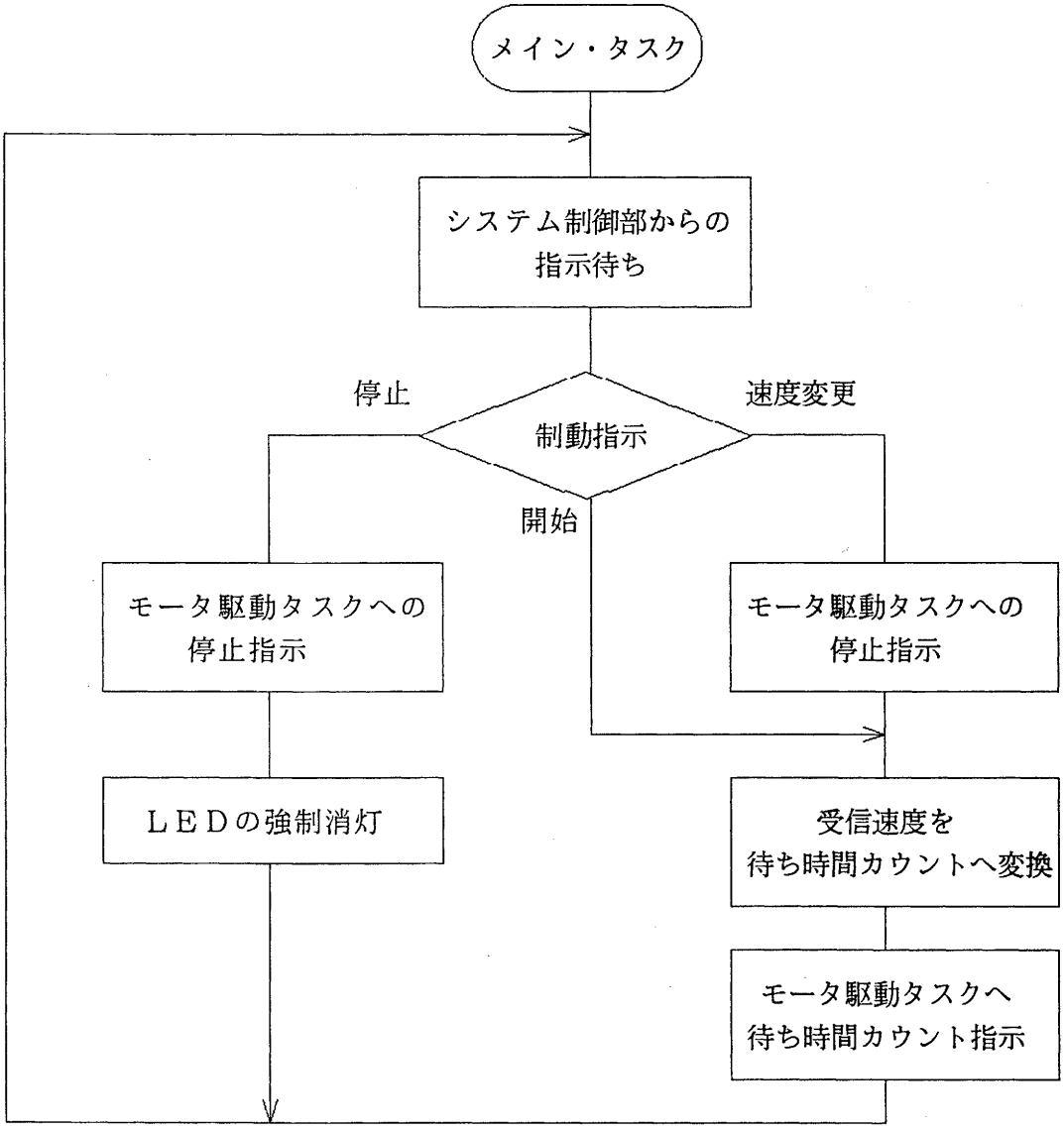


図4.2-2　Ｉ／Ｏ制御部メイン・タスク処理フロー・チャート

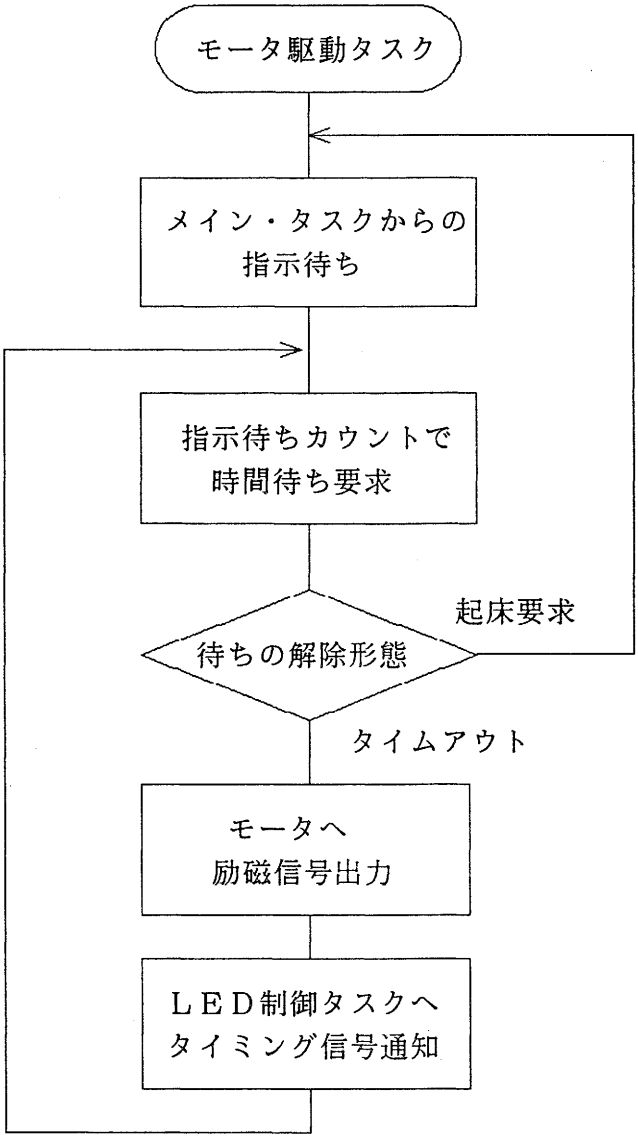


図4.2-3　Ｉ／Ｏ制御部
モータ駆動タスク処理
フロー・チャート

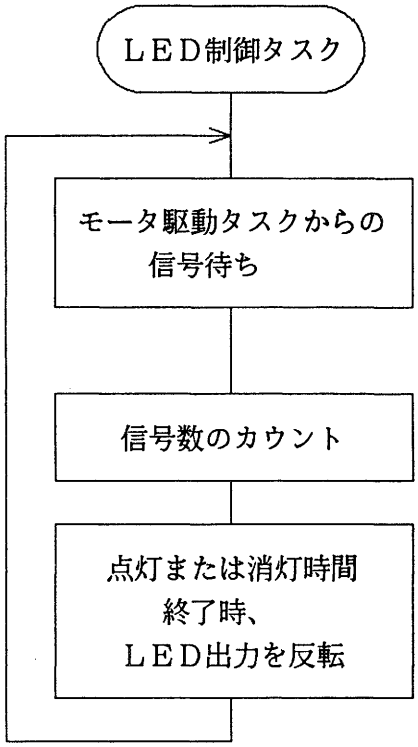


図4.2-4　Ｉ／Ｏ制御部
ＬＥＤ制御タスク処理
フロー・チャート

次に、タスク間インタフェースを示します。

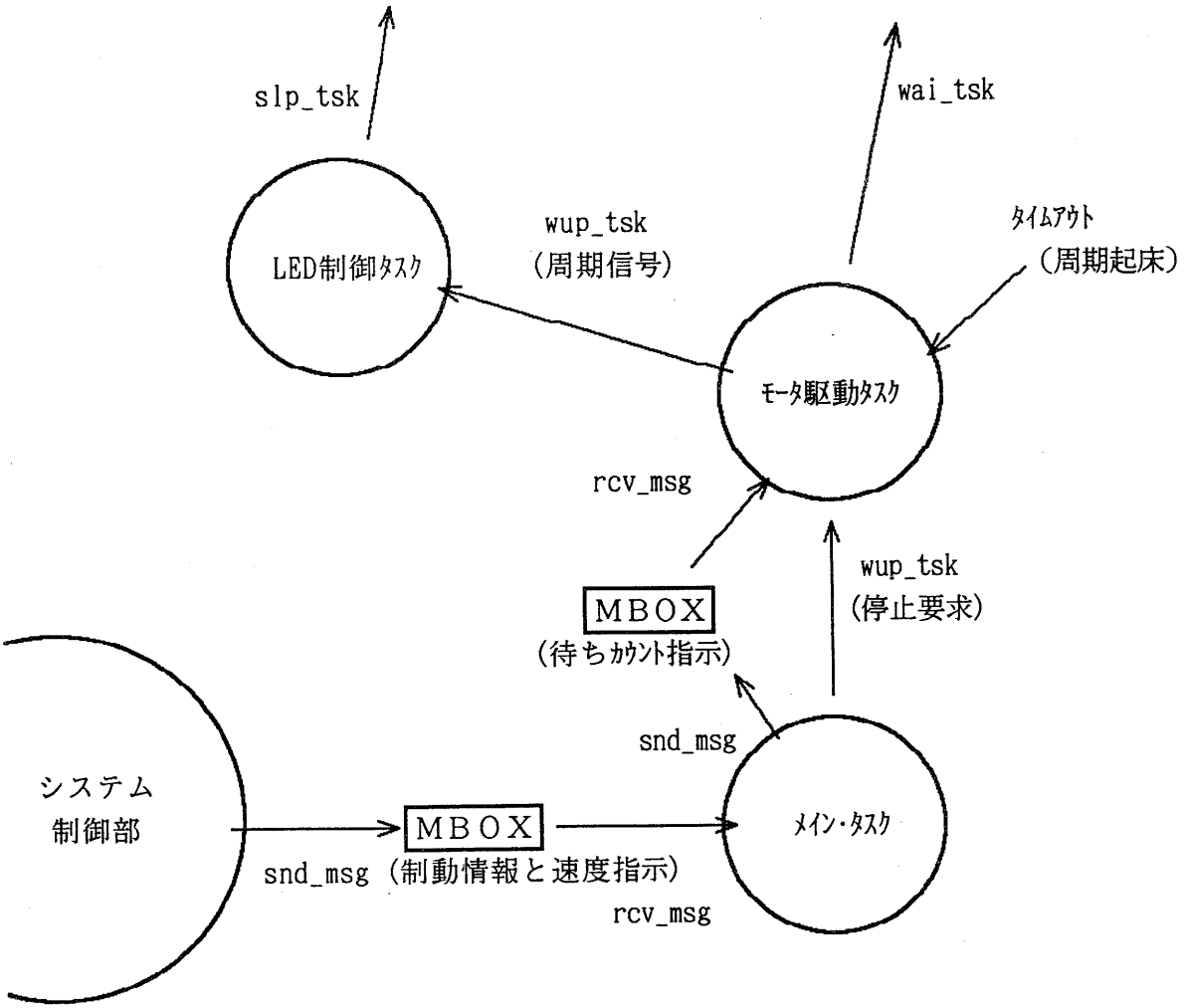


図4.2-5　Ｉ／Ｏ制御部のタスク間インタフェース

メイン・タスクからモータ駆動タスクへの待ちカウント指示には、ｓｎｄ＿ｍｓｇを使用しました。２タスク間の共通スタティック・エリアを使って待ちカウントを指示し、セマフォにより駆動開始を指示するという方法もありますが、その場合、待ちカウント設定と駆動開始指示に時間のずれがあるため、リアルタイム・システムでは良い方法とはいえません。

メイン・タスクの優先度の設定には、メイン・パス（モータ駆動タスクとＬＥＤ制御タスク）に対してメイン・タスクが割り込まないよう注意してやる必要があります。そのため、タスク優先度を以下に決定します。

タスク優先度：　モータ駆動タスク　＞　ＬＥＤ制御タスク　＞　メイン・タスク

4.2.3 システム制御部

システム制御部の機能は、次の４点です。

1）キーによる制動指示の受け付け
2）キー入力エラーの排除
3）システム状態の掌握
　　掌握項目――┬―運転中か停止中か
　　　　　　　└―現在選択されている回転数のレベル
4）Ｉ／Ｏ制御部への制動指示
　　指示項目――┬―運転開始／停止／速度変更かの制動指示情報
　　　　　　　└―選択速度レベル

選択速度レベルは、システム制御部で管理しなければならない情報であるため、システム制御部が一元管理します。そのため、Ｉ／Ｏ制御部は基本的には速度情報を記憶しておく必要はなく、運転開始／速度変更時に指示される速度レベルを使用します。停止の場合、速度レベル情報は不要です。

リアルタイムの条件となるのは、次の２点です。

・キー入力を速やかに受け付け、Ｉ／Ｏ制御部へ指示を行なう
・キー入力エラー時のＩ／Ｏ制御部への影響をできる限り小さくする

制動指示キー押下情報の伝達は、ベクタ割り込みにより割り込みハンドラが起動され、さらにシステム制御タスクへ通知します。ここで、キーの同時押下による、割り込みの同時発生をどう扱うかが問題となります。

今回、割り込みの後者を無視する方法を取りますが、無視する期間となる”同時”のとらえ方は４種類考えられます。

ａ）全く同時（ＣＰＵにとって）
ｂ）割り込みハンドラの処理中に、次の割り込みが発生
ｃ）システム制御タスクが起床され情報を参照する前に、次の割り込みが発生
ｄ）割り込みによるシステムの制御が全て終了する前に、次の割り込みが発生

ａ）の場合の割り込みの処理順序は、割り込み受付けの優先順位により高優先度のものが先に受付けられます。

今回、運転キーを優先させるため、以下に示す優先順位を設定しています。
（ポート２１＝運転キー、ポート２２＝ＵＰキー、ポート２３＝ＤＯＷＮキーとすることで、優先順位が決定されます。）

運転キー　＞　ＵＰキー　＞　ＤＯＷＮキー

同時割り込みを無視する期間としては、ＶＤＴコンソール等と同様に、「処理終了までの間キーの受付けをプロテクトする」という考えから、ｄ）を選びます。

キーの同時押下の後者の無視をどこでやらせるかについては３種類考えられます。

ｅ）ハードウェア機構によりキーの同時押下を抑止
ｆ）ハンドラにより後者のキーを無視
ｇ）タスクにより後者のキーを無視

どの方法でも悪くないと思いますが、ソフトウェアによる方法としてｆ）を選びます。
キーを無視する期間はシステムの制動が終了するまでです。そのため、システム制御タスクからハンドラに対してキー受付け許可のタイミングを知らせる必要があり、両者の間の共通変数を使用します。
また、キー種別の通知手段としてはメイルボックスが考えられますが、ここではすでにキー受付け許可のための共通変数を使用しているために、このエリアを併用します。
つまり、この変数の状態を

０　：　キー受付け許可状態
１　：　運転開始／停止要求、未処理状態
２　：　速度ＵＰ要求、未処理状態
４　：　速度ＤＯＷＮ要求、未処理状態

として、ハンドラによるキー種別のセットと、システム制御タスクへの通知は０の場合にのみ行ない、他の場合無視します。また、システム制御タスクから割り込みハンドラへのキー受付け許可は、この変数を０にすることにより行ないます。
同期手段としては、タスク付属同期機構を用います（セマフォやイベントフラグでも可能です）。
また、Ｉ／Ｏ制御部同様、システム制御タスクも、リセット時の運転開始処理と、停止後の運転再開処理を同じにします。そのため、初期起動時の状態はキー指令待ちとし、イニシャル・タスクから最初の運転開始指示を出させます。
システム制御タスク、割り込みハンドラ、及びイニシャル・タスクの処理フローを図4.2-6,7,8に示します。

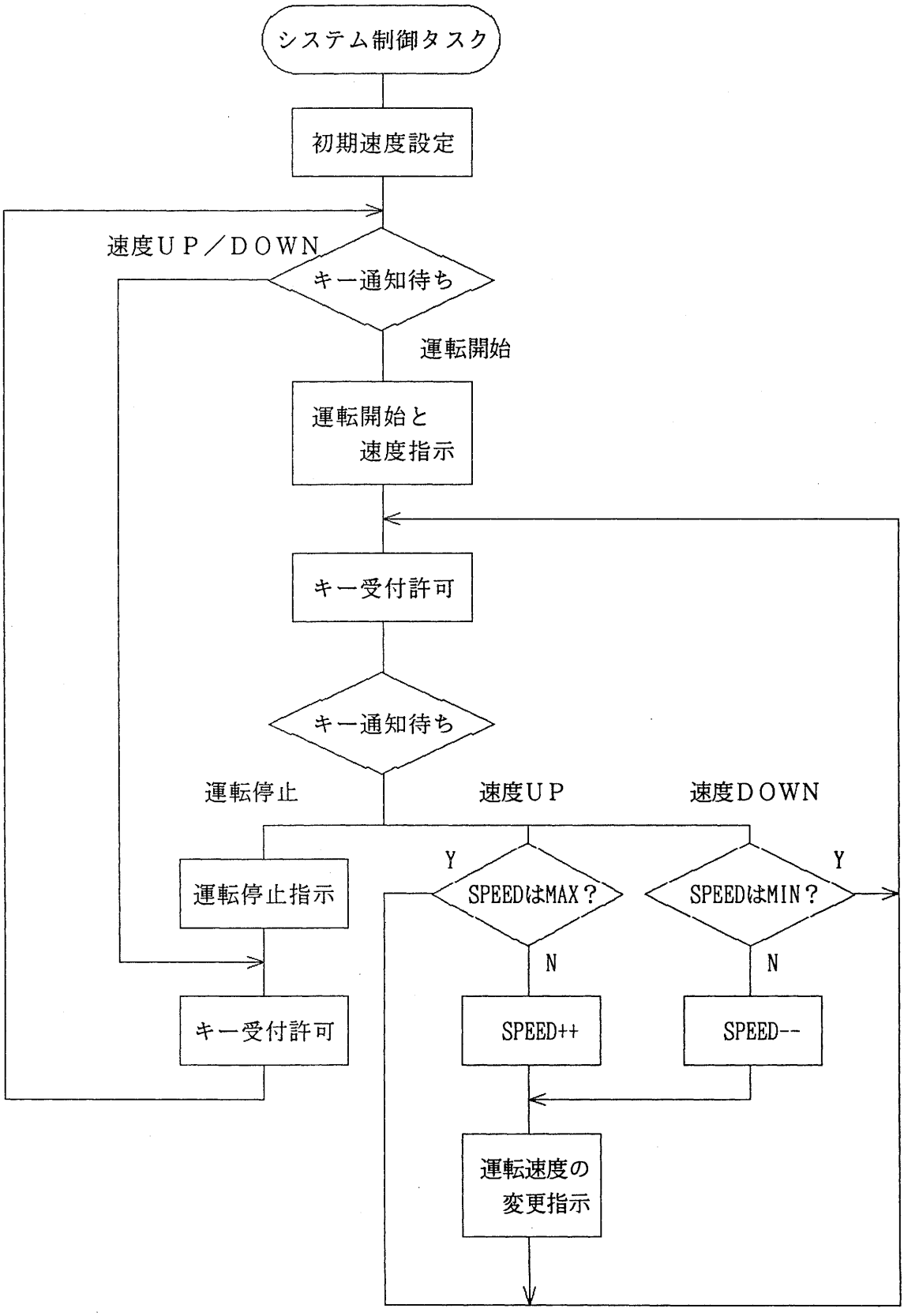


図4.2-6　システム制御タスク処理フロー・チャート

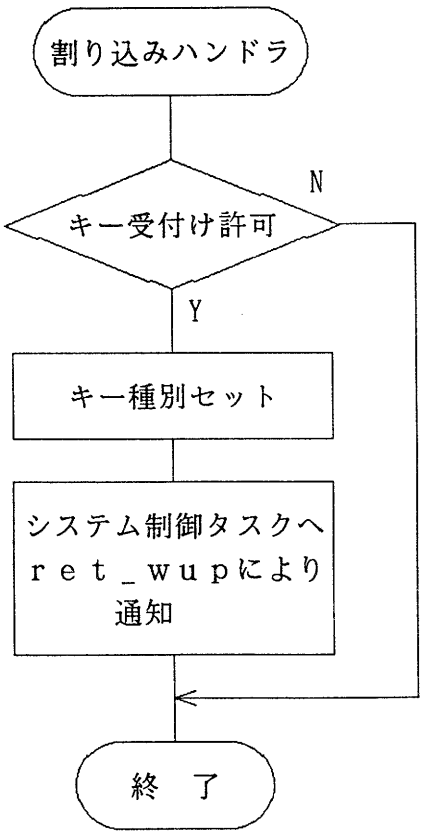


図4.2-7　割り込みハンドラ処理
フロー・チャート

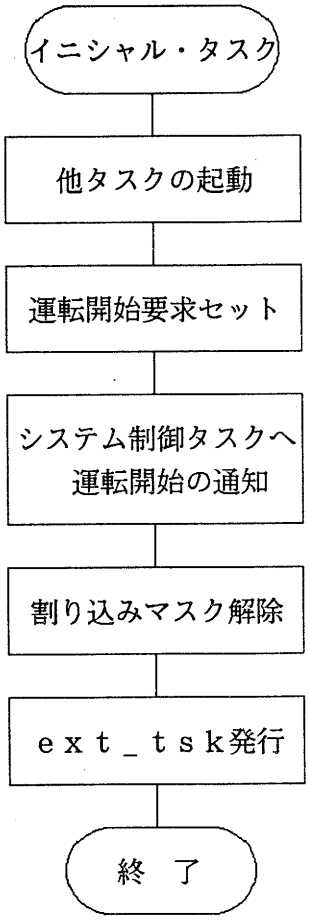


図4.2-8　イニシャル・タスク処理
フロー・チャート

システム制御タスクの優先度は、Ｉ／Ｏ制御部より低位にしました。システム制御タスクの方を高位にすると、Ｉ／Ｏ制御部への制動指示をより早く出すことができます。しかしここでは、速度変更時の処理時間の多少の優劣はあまり問題にならず、それよりキー入力エラー時のメイン・パスへの影響を最小に抑えようと考えたためです。

イニシャル・タスクの優先度は普通、最高位に設定します。最高位にすることでイニシャル・タスクは、ｅｘｔ＿ｔｓｋまで他のタスクにＣＰＵの実行権を譲ることなく一人で走ることができます。そのため他のタスクは、イニシャル・タスクとの競合を意識しないで済みます。

次に、キー入力エラー発生時の、メイン・パス（モータとＬＥＤの運行）に対する影響を考えます。

システム制御タスクの優先度を低位にしたため、システム制御タスクがメイン・パスに対して割り込むことはありません（ＣＰＵ時間を消費するという意味で）。しかし、割り込みハンドラはメイン・パスに対して割り込みます。そのため、このときモータへの信号出力間隔に乱れが起こる可能性があります。

モータの安定性の条件が厳しい場合、割り込みマスク・フラグの操作により、エラー割り込み自体を抑止する必要があるかもしれません。

（付録のソース・リストでマクロ MSKCTL の定義によりマスク・フラグ制御のコードを出力できるようにしています。参照ください。）

最後に、多重割り込み制御は、今回行なっていません。キー受付け方式が先着順である点、またインターバル・タイマによるタイマ・ハンドラの起動と、キー割り込みの受付けに優劣をつける必要性がなかったため、割り込みハンドラはＤＩ状態で走らせています。

システム制御部のタスク間インタフェースを示します。

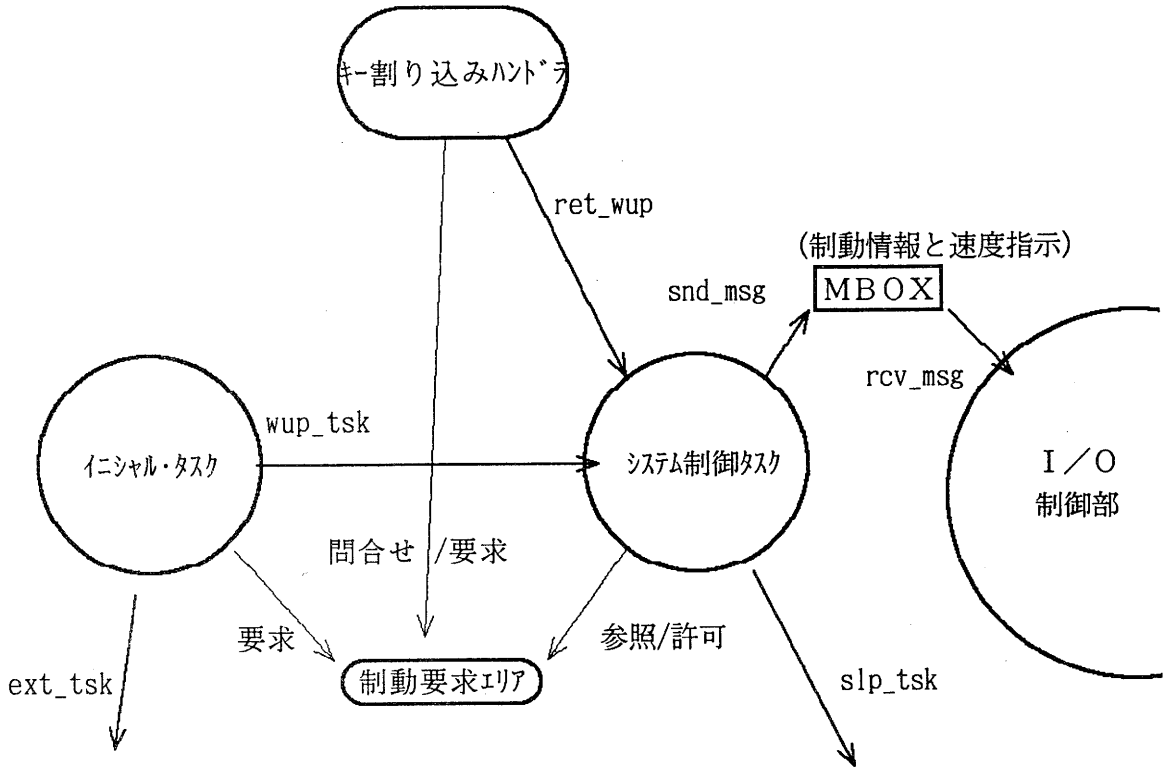


図4.2-9　システム制御部、割り込みハンドラ、イニシャル・タスク間のインタフェース

節のまとめとして、システム全体のタスク機能一覧を示します。

タスク	優先度	機　能
I/O制御部ﾒｲﾝ	3	・制動指示の受付け ・速度レベルのｲﾝﾀｰﾊﾞﾙ･ﾀｲﾏ･ｶｳﾝﾄへの変換 ・Ｉ／Ｏ駆動タスクへの制動指示 ・停止時のＬＥＤの強制消灯
ﾓｰﾀ駆動	1	・モータ駆動 ・ＬＥＤ制御ﾀｽｸへの周期信号出力
LED制御	2	・ＬＥＤの点滅制御
ｼｽﾃﾑ制御	4	・キーによる制動指示受付け ・キー入力エラー処理 ・システムの状態掌握 ・Ｉ／Ｏ制御部への制動指示
ｲﾆｼｬﾙ･ﾀｽｸ	1	・他タスクの起動 ・割り込みマスク解除 ・初期運転の指示
(割り込みﾊﾝﾄﾞﾗ)	－	・キー種別の通知 ・同時キー押下の抑止
(ｼｽﾃﾑ･ﾘｾｯﾄ･ﾙｰﾁﾝ)	－	・ベクタ・テーブル設定 ・システムＩ／Ｏ設定 ・Ｃスタティック変数の初期値ロード ・ＯＳのリセット・ルーチン呼出し

備考．ＲＸ７８Ｋ／Ⅲは１６段階のタスク優先度設定が可能ですが、ここでは４段階のみ使用しています。

表4.2-1　タスク機能一覧

4.3 資源考察

タスク間インタフェース設計により、システム機能をリアルタイム・システムとして構築することができました。

本節では、リアルタイム・システムとしての残る条件、資源問題について考察します。

資源問題は、複数タスク間で、ある資源を共有する場合に発生します。その場合、時分割に”常に１つのタスクのみがその資源のアクセス権を得られるように”コントロールする必要があります。

ＲＸ７８Ｋ／Ⅲ上で問題となる資源には、次に挙げるものがあります。

・Ｉ／Ｏアクセス権
・広域変数
・ユーティリティ関数
・メモリプール

それに対して、資源管理の方法には次のものがあります。

・セマフォまたは他のキュー構造によるもの
・タイミング
・１タスクの独占
・状態コードによる方法

１）Ｉ／Ｏアクセス権

本システムのＩ／Ｏは、制動キー、モータ、ＬＥＤの３種です。制動キーは、割り込みハンドラと１対１に対応付けされるために問題になりません。また、モータは、モータ駆動タスクにアクセス権を独占させているため、これも問題になりません。

最後に、ＬＥＤは、通常ＬＥＤ制御タスクがアクセス権を持ちますが、停止要求によりアクセス権をＩ／Ｏ制御メイン・タスクに移動させ、このときＬＥＤを強制消灯しています。この方法はタイミングによるアクセス権の移動方法です。

タイミングによる資源管理は、セマフォによる方法などに比べて軽快な方法ですが、反面システムの状態（タイミング）に頼っているだけに、不安定な管理手法です。

タイミングの状態が不安定な場合は、この方法を用いることはできません。また将来、状態の変更があり得る場合には、セマフォによる確実な管理を考える必要があります。

２）広域変数

システムで使用した広域変数は以下に示す２つです。

・ＬＥＤ制御タスクとＩ／Ｏ制御メイン・タスクで使用する点滅のためのカウンタ
・キー割り込みハンドラとシステム制御タスク間の制動要求エリア

前者は、停止後の運転再開時、点灯時間のカウントを０からやり直させるため、Ｉ／Ｏ制御メイン・タスクによってカウンタを０クリアさせています。ＬＥＤのアクセス権同様、タイミングによる変数へのアクセス権の移動を行なっています。

制動要求エリアは、ハンドラからシステム制御タスクへキーの種別を伝えるための広域変数で、状態コードの意味をも持たせています。具体的には以下の意味を持ちます。

０の場合　　：ハンドラからの制動要求が可能
０でない場合：ハンドラからの制動要求をシステム制御タスクが受付けていない
　　またはその制動の処理中

変数が０でないときハンドラにはアクセス権がなく、システム制御タスクの制動処理終了時に０クリアされることにより、ハンドラへアクセス権が与えられます。

システムの状態をモニタするような状態変数に対するアクセス権の管理には、通常セマフォを用いますが、今回使用するチャンスがありませんでした。

３）ユーティリティ関数

ユーティリティ関数は、複数タスクからＣＡＬＬすることのできるツール関数のことで、ユーザ自作のものの他にＣのライブラリ関数も含みます。ここで資源問題が発生するのは、関数がスタティック変数またはＩ／Ｏアクセスを行なう場合です。

リアルタイム・システム上で、ユーティリティがスタティック変数を使用していると、この変数は、広域変数と同様の資源問題を発生させます。

そのため、通常ユーティリティでは、タスク・ディスパッチにより保障されるコンテキスト（レジスタとスタック）以外の資源を使用しないようにします。
（このような関数をリエントラントな関数といいます）

Ｉ／Ｏアクセスを行なう関数の場合、そのＩ／Ｏへのアクセス権という資源問題が発生します。したがって、セマフォによって関数の使用権を管理するか、関数の利用を１つのタスクに限定または独占させます。

4）メモリプール

メモリプールは、固定長のメモリブロックを資源として、キュー構造によりメモリブロック資源を管理する機構です。そのため、メモリプールの持つメモリブロック数が、同時に与えることのできる資源数となります。

本システムでは、次の２タスク間のメッセージの格納エリアにメモリブロックを使用しています。

・システム制御タスクとＩ／Ｏ制御メイン・タスク間
・Ｉ／Ｏ制御メイン・タスクとモータ駆動タスク間

メッセージの形式はそれぞれ異なるため、２つのメモリプールを使用しています。

第３章で説明しましたが、ＲＸ７８Ｋ／Ⅲのメッセージ転送方式はアドレス方式のため、メッセージそのものが送受信されるわけではありません。また、メッセージ・エリアの使用権は、メッセージ受信により送信タスクから受信タスクへ移動します（細かくいうとメッセージ送信によりＯＳの管理下におかれ、受信により受信タスクのものになります）。この時点で、送信側タスクは次回のメッセージ作成のためのエリアを失い、新たなメッセージ作成を行なうには別のエリアが必要になります。そのためメッセージ・エリアにメモリブロックを使用します。

メッセージ・エリア（メモリブロック）獲得は送信側タスクが、返却は受信側タスクが次のメッセージを受け取る前に行ないます。通常、送信側からは受信タスクがメモリブロックを返却するタイミングが分からない（または意識したくない）ため、２つ以上のメモリブロック資源が必要になります。

受信側タスクが獲得しているメモリブロックは常に１つ以下です。送信側では、受信タスクがメッセージ受信しなければメッセージ・キューにつながれることにより、２つ以上獲得している状態が考えられます（正確にはメッセージ・キューにつながれているメッセージは、ＯＳの管理下に入ります）。

そのため、受信タスクの優先度が送信タスクより高い場合、メッセージ・エリアとして使用するメモリプールの資源数（ブロック数）は2、逆の場合２以上必要となります。

今回の場合、２ヶ所の送受信とも、受信タスク側が高優先度のためメモリブロック数は２としています。

4.4 タイミング考証

本節ではシステムのきわどいタイミング、または重要な処理に注目して、タイミング・ブロック図を用いて考証を行ないます。

（本節では資源管理のためのシステムコールを省いて表わしています。また便宜上、ｗａｉ＿ｔｓｋシステムコールの待ちカウントを全て２で表わしています。）

１）メイン・パス

定速度運転中のタイミング・ブロック図を示します。

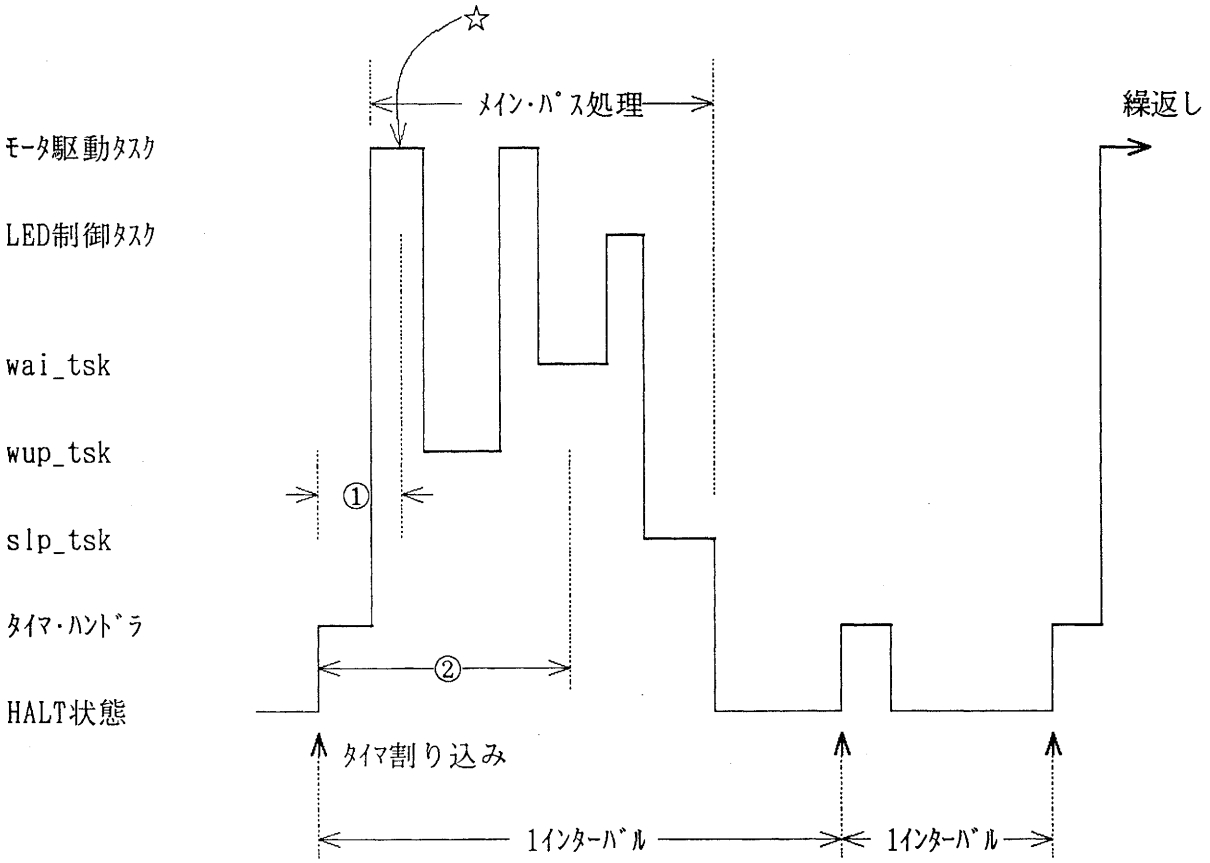


図4.4-1　メイン・パスのタイミング・ブロック図

タイムアウトによる起床により、モータ駆動タスクは励磁信号を出力し、次にＬＥＤ制御タスクに対してｗｕｐ＿ｔｓｋを発行します。このときＬＥＤ制御タスクの起床待ちは解除され、スケジューラが起動されますが、モータ駆動タスクの方が高優先度のため、引き続き実行権を得られます。モータ駆動タスクがｗａｉ＿ｔｓｋを発行すると、ＲＥＡＤＹタスクはＬＥＤ制御タスクだけとなり、ＬＥＤ制御タスクが走ります。

ＬＥＤ制御タスクがモータ駆動タスクからの起床待ちになると、ＲＥＡＤＹタスクは１つもなくなり、ＣＰＵはＨＡＬＴ状態に入ります。
２回のタイマ割り込みにより再びモータ駆動タスクが起床され、以上の繰返しとなります。

図中の☆印はモータへの信号出力点を示します。従って、図中①は、タイマ割り込み発生からモータへ信号が出力されるまでの時間です。この①の処理時間が一定であることにより、信号出力周期が正確に２インターバルになります。
②は、タイマ割り込み発生からｗａｉ＿ｔｓｋのＯＳによる受付けの完了するまでの時間（ｗａｉ＿ｔｓｋに伴うタスク・ディスパッチの処理直前まで）です。この時間が１インターバルより長いと、モータ駆動タスクの起床周期は３インターバルになってしまいます。
さらに、１インターバルをタイマ・ハンドラの処理時間より短くすると、タイマ・ハンドラだけでＣＰＵを１００％消費してしまい、タスクは動くことができなくなってしまいます。

２）制動時
ＵＰまたはＤＯＷＮキーによる速度変更時のブロック図を示します。

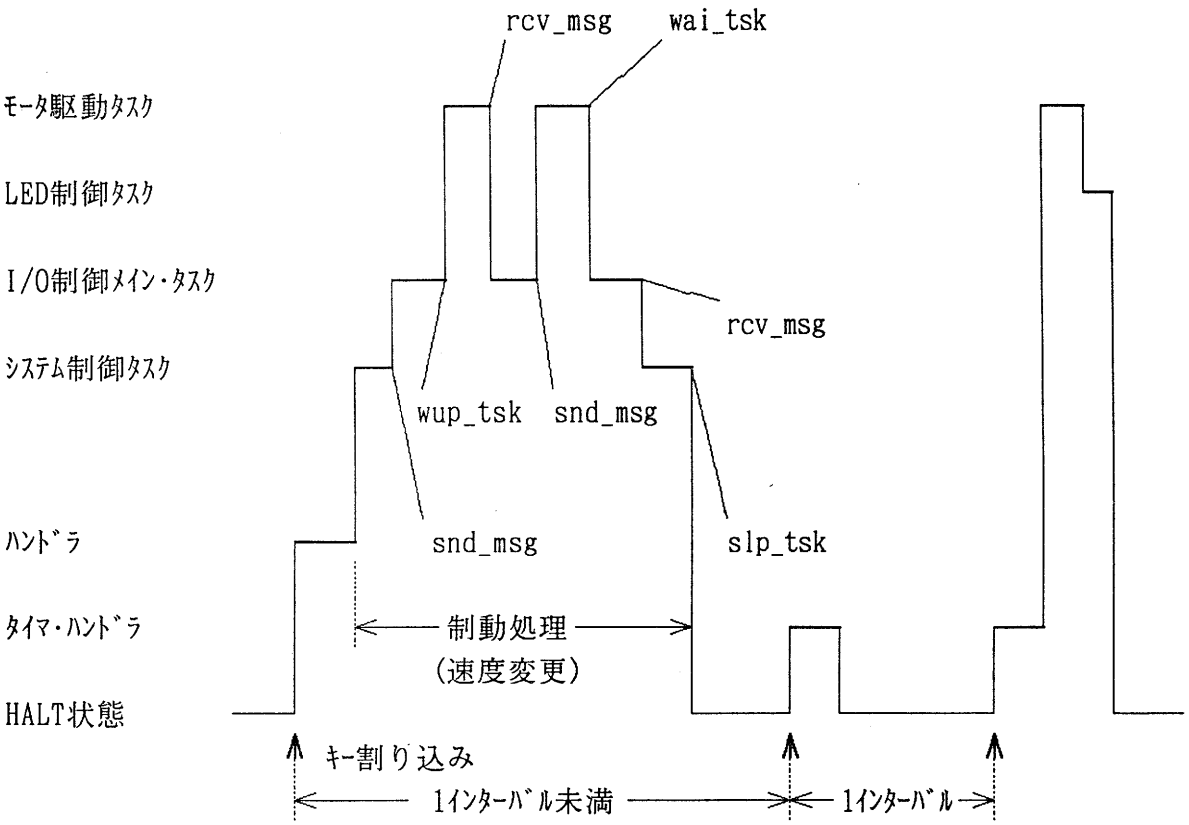


図4.4-2　制動処理のタイミング・ブロック図（その１）

このブロック図は、メイン・パスのＨＡＬＴ状態でキー割り込みが起こり、制動処理の間中、１回もタイマ割り込みが発生しなかった場合のブロック図です。

キー割り込みの発生により、ハンドラからシステム制御タスクを介して、Ｉ／Ｏ制御メイン・タスクへ制動情報が伝わります。Ｉ／Ｏ制御メイン・タスクは、モータ駆動タスクへｗｕｐ＿ｔｓｋにより一旦停止を指示します。そして、ｓｎｄ＿ｍｓｇによって新しい待ちカウントを指示し、運転を再開させます。

ここでモータ駆動タスクは、モータへ信号を出力しません。それはｗａｉ＿ｔｓｋ発行後、次のタイマ割り込みの発生までの時間が一定ではないためです。

その後、高優先度タスクから時間待ち、あるいは制動指示待ちに入っていき、再びメイン・パスのサイクルが繰り返されます。

次に、メイン・パス処理中にキー割り込みが発生した場合のブロック図を示します。

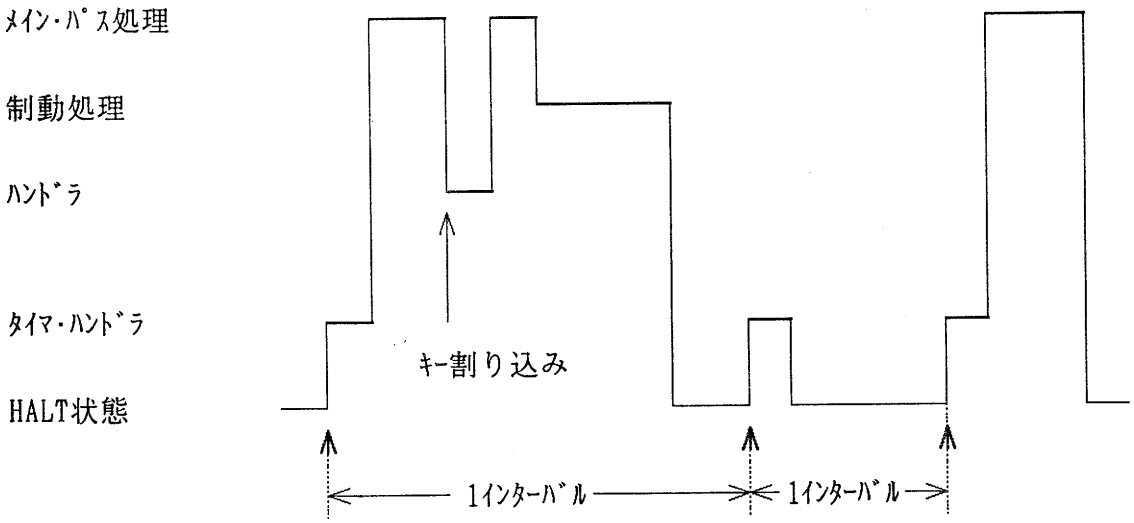


図4.4-3　制動処理のタイミング・ブロック図（その２）

ハンドラはメイン・パスに対して割り込みますが、制動処理はメイン・パス処理より低優先度のため、メイン・パス処理の終了まで待たされることになります。

制動処理中にタイマ割り込みが起こった場合のブロック図は、タイマ・オペレーションのシステムコール（ｗｕｐ＿ｔｓｋとｗａｉ＿ｔｓｋ）の発行前後で異なります。

ｗｕｐ＿ｔｓｋ発行前にタイマ割り込みが発生した場合、タイマ割り込み発生時、モータ駆動タスクはまだ時間待ち状態のため、タイムアウトが起こる可能性があります（その回のタイマ・ハンドラ処理で、モータ駆動タスクの待ちカウントが０になった場合）。タイムアウトが起こると、優先度の関係によりメイン・パス処理がまるまる実行された後、制動処理が再開されます。

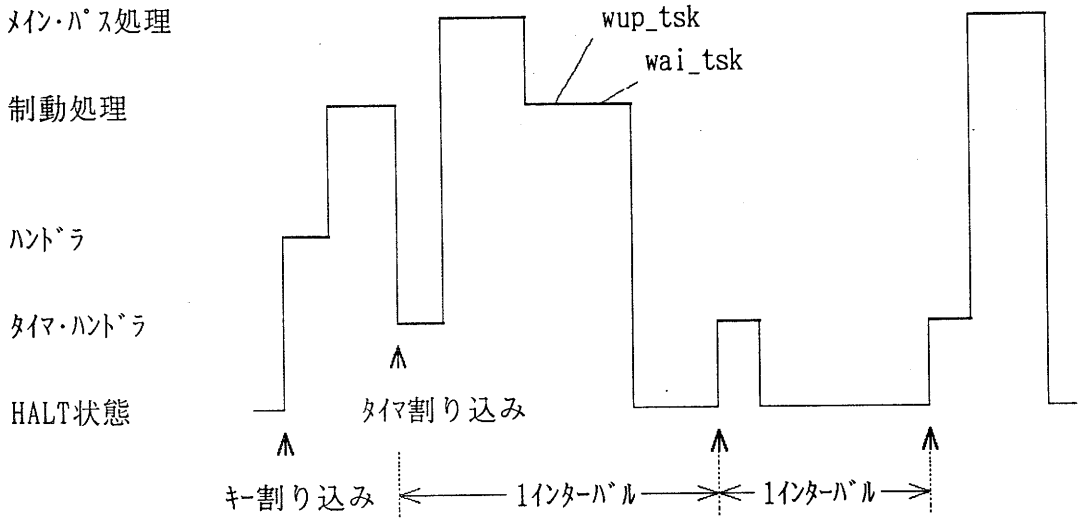


図4.4-4　制動処理のタイミング・ブロック図（その３）

ｗｕｐ＿ｔｓｋ後、ｗａｉ＿ｔｓｋ前に、タイマ割り込みが発生した場合、モータ駆動タスクはすでに、時間待ち状態ではありません。そのため、タイマ・ハンドラによるＣＰＵタイムの消費が起こるだけで制動処理が継続されます。

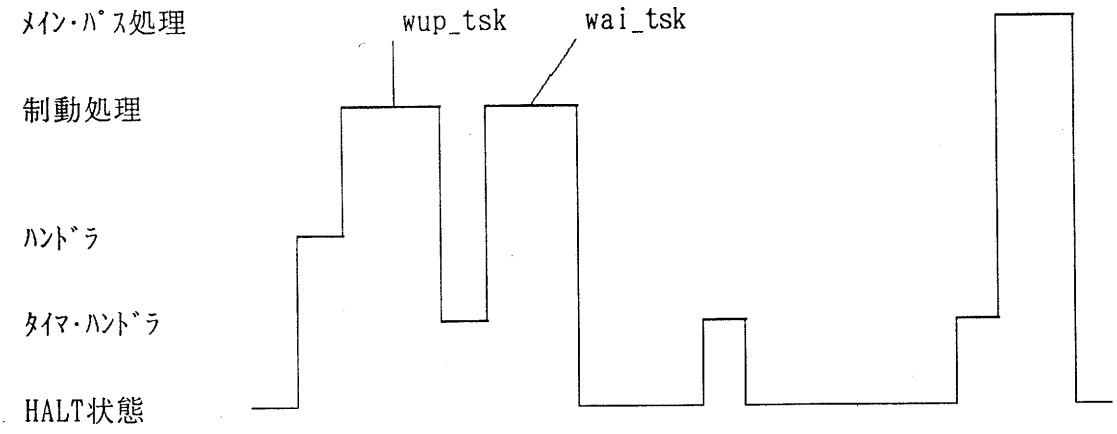


図4.4-5　制動処理のタイミング・ブロック図（その４）

ｗａｉ＿ｔｓｋ発行後にタイマ割り込みが発生した場合、モータ駆動タスクはすでに、速度変更後の待ち状態に入っています。そのため、タイマ・ハンドラにより１回目の待ちカウントのデクリメントが行なわれます。

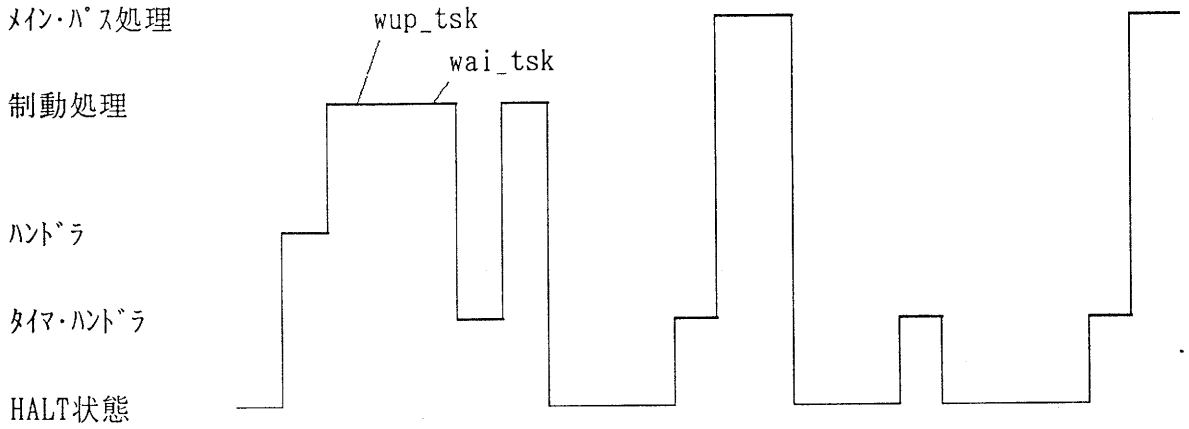


図4.4-6　制動処理のタイミング・ブロック図（その5）

3）キー入力エラー時

最高速度運転中のＵＰキー押下によるエラー処理のブロック図を示します。

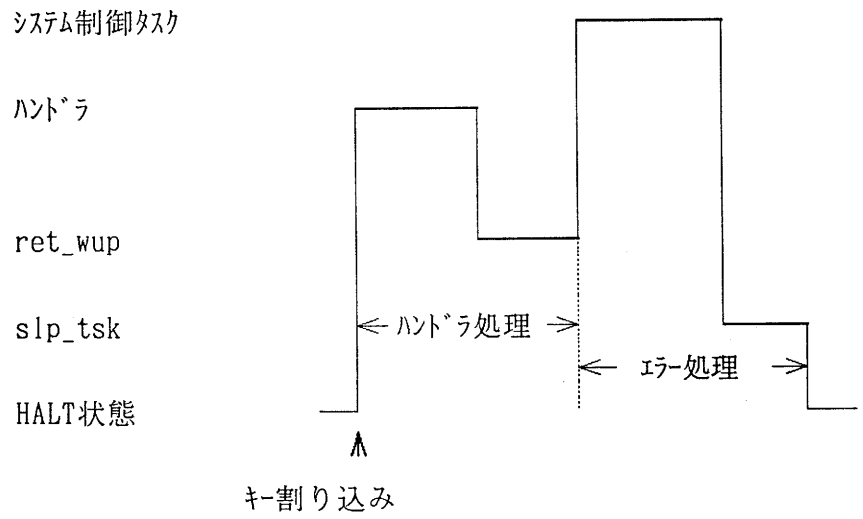


図4.4-7　キー入力エラー処理のブロック図

ハンドラからキー押下情報がシステム制御タスクに通知されますが、システム制御タスクはこれを無視します。

エラー処理によるメイン・パスへの影響は排除したいところですが、図4.4-1の区間①で

キー割り込みが起こった場合、下図に示すように、信号出力に乱れが起こります。

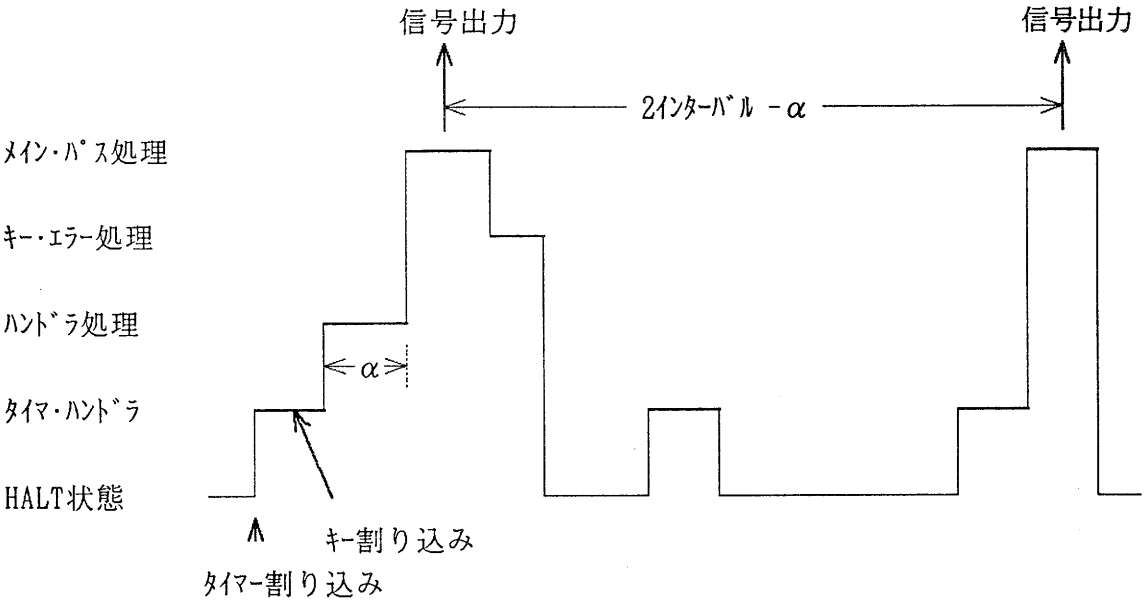


図4.4-8　キー入力エラーによるモータへの信号の乱れ

このときの信号出力の乱れを防ぐには、キー入力エラーによる割り込みの発生そのものを、割り込みマスク・フラグ等によりプロテクトするしかありません。

4.5 データ・エリア設計

本節ではリアルタイムＯＳ、オブジェクト管理ブロック、タスク、及び割り込みハンドラの各部について、スタックとスタティックのＲＡＭ消費サイズを検討し、システム全体のデータ・エリア試算を行ないます。

1）リアルタイムＯＳ部

ＯＳ自身のユーザ（タスク）側からは見えない作業領域で、ＲＥＡＤＹキュー、時間待ちキューやＯＳ自身の変数領域、及びスタック・エリアとして使用されます。

このＯＳの作業領域は番地が固定されており、以下に示す８０バイトがＯＳの予約エリアとなります。

０ＦＣ８０Ｈ　～　０ＦＣＣＦＨ

また、今回は使用していませんが、ベクタ・テーブルを内部ＲＯＭ（０番地～）に置き、かつ動的に割り込みハンドラの設定／解除を行なう場合（２段ジャンプによる方法です．２段目のジャンプ・テーブルが内部ＲＡＭに置かれることにより、ハンドラの動的設定／解除を行なうことができます）、以下の１２８バイトがさらにＯＳの予約エリアになります。

０ＦＣＤ０Ｈ　～　０ＦＤ４ＦＨ

2）オブジェクト管理用制御ブロック

リアルタイムＯＳがオブジェクト（タスクの状態、同期機構、及びメモリプール）を管理するための管理ブロックです。以下のものがあります。

タスク制御ブロック（ＴＣＢ）	：	１８バイト
セマフォ（ＳＥＭ）	：	６バイト
メイルボックス（ＭＢＯＸ）	：	８バイト
イベントフラグ（ＥＶＴ）	：	６バイト
メモリプール	：	４＋(ﾒﾓﾘﾌﾞﾛｯｸ･ｻｲｽﾞ)×(ﾒﾓﾘﾌﾞﾛｯｸ数)バイト

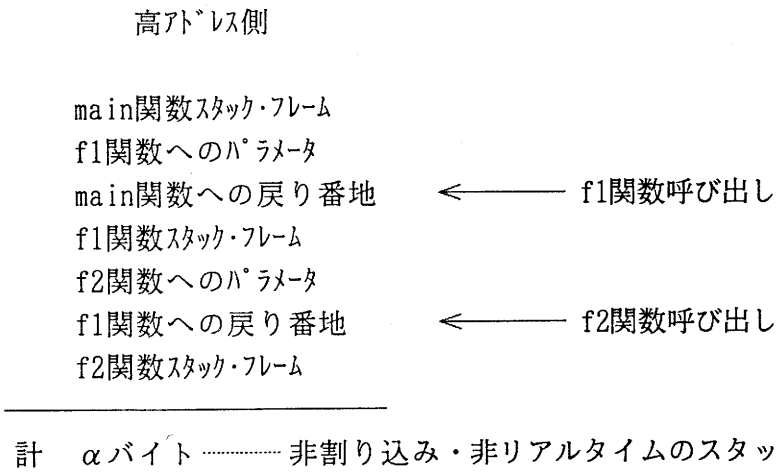
本システムではタスク数５本、ＳＥＭ＝０、ＭＢＯＸ＝２、ＥＶＴ＝０、またメモリプール＝２で、各メモリプールのメモリブロック・サイズとブロック数は、（６，２）、（４，２）を使用しています。

3）スタック・エリア

スタック消費サイズの正確な試算が通常困難なのは、C等のライブラリの消費サイズが分からない点と、割り込みの非同期性によるものです。リアルタイム・システムではさらに、ＯＳのスタック消費量が分からない点が加えられます。
スタック消費サイズの試算は、シングル・タスクの場合、以下のように行ないます。

(1) ｍａｉｎ関数のスタック・フレーム・サイズを試算
(2) 使用全関数のスタック・フレーム・サイズを試算
(3) 関数ＣＡＬＬの全てのケースを検討し、最大消費点を探します。

関数ｆ１を呼び出し、さらにｆ１から関数ｆ２を呼び出した時点が最大消費点とすると、そのときのスタックの内容は以下のようになります。

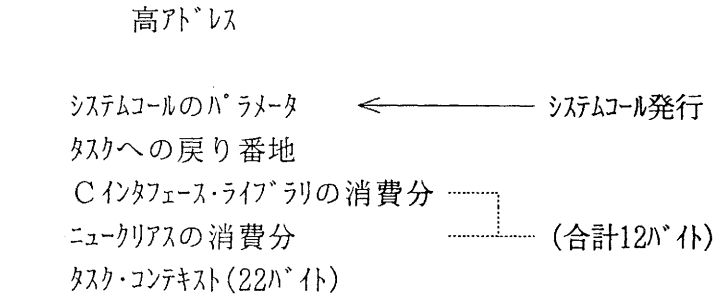


シングル・タスク・システムとは異なり、リアルタイムＯＳ下ではスタック・エリアはタスク個別に用意しなければなりません。
リアルタイム・システムの場合、ｍａｉｎ関数を各タスクに置き換えて試算を行ないます。ただし、ここで考慮する呼び出し関数からシステムコールを除外して考えます。

次に、システムコール呼び出しによるスタック消費を考えます。
Ｃのソースからシステムコール呼び出しを行なった場合、関数ＣＡＬＬのネストは以下のようになります。

タスク ⟶ Ｃｲﾝﾀﾌｪｰｽ･ﾗｲﾌﾞﾗﾘ ⟶ ＯＳﾆｭｰｸﾘｱｽ

システムコールによりタスク・ディスパッチが発生した場合、タスク・コンテキストのスタック退避直後のスタック内容は以下のようになります。



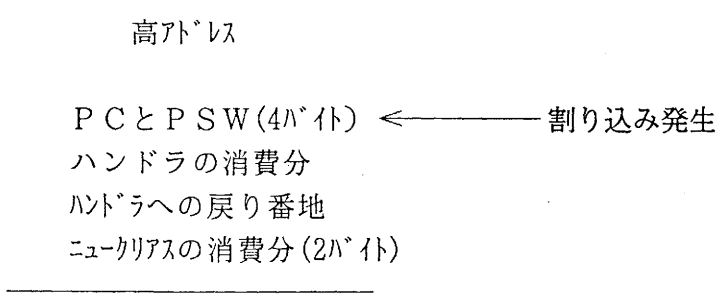
α＋βが割り込み未考慮のスタックの最大消費量と考えることができます。
（非割り込み・非リアルタイムのスタック消費の最大点でのシステムコール発行を行なっていなければ、消費サイズはα＋βより小さくなります。）

最後は、割り込みについて考えます。

第１に考えなければならない点は、割り込みハンドラでそのままタスクのスタック・エリアを使用するかどうかです。ハンドラのスタック領域には、専用領域を確保する方法や１レジスタ・バンクを与える方法も考えられますが、ここでは、タスクのスタック・エリアをそのまま使用する場合について考えます。

第２に、多重割り込みを行なっているかどうかです。多重割り込みを行なっている場合、その最悪の多重割り込みのケースが問題となります。

１回の割り込みで消費するスタック・サイズは、ハンドラで必ずシステムコールを発行すると考えた場合、以下のようになります。



本システムの場合多重割り込みは行なっていませんが、タイマ・オペレーションを使用しているため、タイマ・ハンドラの割り込み処理中のキー割り込みの発生は考えられます。したがって、本システムの割り込みの多重度は２です。

本システムの各タスクのスタック・エリア試算値は以下になります。

$\alpha + \beta + \gamma \times 2 \quad = \quad 128$バイト

$\alpha = 32$バイト、$\beta = 40$バイト、$\gamma = 28$バイト

・各ﾀｽｸのｽﾀｯｸ･ﾌﾚｰﾑの使用上限値 = 32ﾊﾞｲﾄに限定
（auto変数の使用ﾊﾞｲﾄ数の上限値 = 26ﾊﾞｲﾄになります．次章第６節参照．）
・タスクからのｼｽﾃﾑｺｰﾙ以外の関数CALLは行なっていない
・ｼｽﾃﾑｺｰﾙのﾊﾟﾗﾒｰﾀによるｽﾀｯｸ消費は最大4ﾊﾞｲﾄ
・割り込みﾊﾝﾄﾞﾗのｽﾀｯｸ消費の上限値 = 20ﾊﾞｲﾄに限定

補足．今回は使用しませんでしたが、ＲＸ７８Ｋ／Ⅲではレジスタ・バンクをタスク別に割り当てることができます（最大６タスク）。その場合タスク・コンテキストのスタック消費は６バイトとなります。

4）データ・エリア試算

以上による本システムの全データ・エリア試算を示します。

ＲＡＭ領域		
ＯＳ作業領域		８０
オブジェクト管理領域	５タスク ０セマフォ ２メイルボックス ０イベントフラグ メモリプール１ メモリプール２	１８×５ ６×０ ８×２ ６×０ ４＋６×２ ４＋４×２
スタック・エリア	５タスク	１８６×５
タスク、ハンドラ個別のスタティック・エリア　**		０
広域変数エリア		３
－合　計－		１１４７バイト

表4.5-1　ＲＡＭ領域試算表

** このシステムでは、タスク内のローカル変数にレジスタ変数とａｕｔｏ変数のみ使用しています。

5. コンパイル、アセンブル及びリンケージ

5.1 オブジェクト／ファイル分割

本システムのオブジェクト／ファイル分割を示します。

ファイル名	内　容
COMMON．INC	システム共通情報の定義
REFQUE．INC	オブジェクト管理用制御ブロックの参照宣言
LIB．INC	システムコール参照宣言
MDL_IO．C	Ｉ／Ｏ制御部モジュール
MDL_SYS．C	システム制御部モジュール
TSK_I．C	イニシャル・タスク
RESET．ASM	システム・リセット・ルーチン
HANDLER．ASM	割り込みハンドラ
ROM．ASM	Ｃスタティック変数初期値の解決用シンボル定義
CF1．ASM	コンフィギュレーション・テーブル
CF2．ASM	オブジェクト管理用制御ブロック
SYSTBL．ASM	システムコール・エントリ・テーブル

表5.1-1　オブジェクト分割表

ファイル名の拡張子は、以下に統一しています。

ＩＮＣ　：　Ｃソースのインクルード・ファイル
Ｃ　　：　Ｃソース
ＡＳＭ　：　アセンブラ・ソース

ＣＣ７８Ｋ３の提供ファイルを以下に示します。

ＣＳＴＡＲＴＲ．ＡＳＭ（ＲＥＳＥＴ．ＡＳＭ中に展開しています）
ＲＯＭ．ＡＳＭ

ＣＣ７８Ｋ３の提供ファイルは主にＣスタティック変数の初期値の解決のためのものです。
ＲＸ７８Ｋ／Ⅲにはコンフィギュレータが付属しています。次に示す３ファイルはコンフィギュレータの出力ファイルです。

ＣＦ１．ＡＳＭ
ＣＦ２．ＡＳＭ
ＳＹＳＴＢＬ．ＡＳＭ

Ｃスタティック変数の初期値の解決方法については第５節で、コンフィギュレータの出力データについては第７節で解説します。

5.2 開発環境

本システム設計の開発環境を示します。

1）ディレクトリ構成

```
A:¥ ┬── COMMAND.COM
    ├── CONFIG.SYS      : PRINT.SYSとﾌﾛﾝﾄｴﾝﾄﾞ・ﾌﾟﾛｾｯｻのﾃﾞﾊﾞｲｽ登録
    ├── AUTOEXEC.BAT    : 環境の設定
    ├── BAT¥            : ｺﾝﾊﾟｲﾗ、ｱｾﾝﾌﾞﾗ、ﾘﾝｶ、ｵﾌﾞｼﾞｪｸﾄ・ｺﾝﾊﾞｰﾀ実行用ﾊﾞｯﾁ・ﾌｧｲﾙ
    ├── 78K3¥           : ｺﾝﾊﾟｲﾗ、ｱｾﾝﾌﾞﾗ、ﾘﾝｶ、ｵﾌﾞｼﾞｪｸﾄ・ｺﾝﾊﾞｰﾀ実行ﾌｧｲﾙ
    │     ├── INC¥     : Cﾍｯﾀﾞ・ﾌｧｲﾙ群
    │     └── LIB¥     : Cﾗｲﾌﾞﾗﾘ・ﾌｧｲﾙとﾘｱﾙﾀｲﾑOSﾗｲﾌﾞﾗﾘ
    ├── MSDOS¥          : MSDOS外部ｺﾏﾝﾄﾞ
    ├── EDITOR¥         : ｴﾃﾞｨﾀ
    └── FEP¥            : ﾌﾛﾝﾄｴﾝﾄﾞ・ﾌﾟﾛｾｯｻ

B:¥ ──────────────────  : ｿｰｽ・ﾌｧｲﾙと出力ｵﾌﾞｼﾞｪｸﾄ、及びﾘﾝｸ・ﾊﾟﾗﾒｰﾀ・ﾌｧｲﾙとﾘﾝｸ・ﾃﾞｨﾚｸﾃｨﾌﾞ・ﾌｧｲﾙ
```

2）環境の設定

ＡＵＴＯＥＸＥＣ．ＢＡＴファイル内容を示します。

```
PATH A:¥;A:¥BAT;A:¥78K3;A:¥MSDOS;A:¥EDITOR ……①
VERIFY ON
BREAK ON ……②
SET INC78K3=A:¥78K3¥INC ……③
SET LIB78K3=A:¥78K3¥LIB ……④
SET TMP=B:¥ ……⑤
B: ……⑥
```

①ＤＯＳコマンドのディレクトリ・サーチ・パスを指定します。
②ＣＴＲＬ＾Ｃによりコンパイラ等の実行を中断することができます。
③Ｃソース中の　＜　＞　で指定したインクルード・ファイルのディレクトリを指定します。
④リンカにライブラリ・ファイルの所在ディレクトリを指示します。
⑤コンパイラ、リンカ等の使用するテンポラリ・ファイルの作成ディレクトリをＢドライブ、ルート・ディレクトリに指定します。
⑥カレント・ドライブをＢ、カレント・ディレクトリをＢドライブ、ルートにします。

5.3 コンパイル

1）Ｃの制限機能

リアルタイム・システムでは、Ｃの機能の内、次に挙げるものは使用することができません。

・レジスタ変数のｓａｄｄｒ領域への拡張（－ＱＲオプション）

このコンパイル・オプションは、レジスタ変数をＶＰ，ＲＰ３以外にｓａｄｄｒ領域に拡張して、合計１０個（１０ｗｏｒｄ)のレジスタ変数の使用を可能とするためのものです。このｓａｄｄｒ領域は、配置固定の１６バイトのエリアで、各関数で共通に使用されるため、リアルタイム・システム下では使用できません。

例えば、タスクＡ、Ｂが共に－ＱＲオプションを使用して３つのレジスタ変数を定義すると、３番目の変数は同一のｓａｄｄｒ領域に割付けられることになります。そのため、この３番目の変数はローカル変数であるにもかかわらず、（第４章３節で解説した）広域変数と同様の資源問題を引き起こすためです。

１タスクだけが３番目以降のレジスタ変数を使用するならば問題ありませんが、変則的な方法です。

また、ユーティリティ関数で３番目のレジスタ変数を使用すると、リエントラントでなくなってしまいます。注意してください。

・ｎｏｒｅｃ関数の使用

通常の関数の引数の受渡しにはスタックを使用しますが、ｎｏｒｅｃ関数はＶＰ，ＲＰ３とｓａｄｄｒ領域を使用します。

引数が２個（２ｗｏｒｄ）までならかまいませんが、３個以上使用すると、ｓａｄｄｒ領域を使用するため、－ＱＲオプションと同様の問題が発生します。注意を要する機能です。

２）コンパイル・オプション

今回、使用したコンパイル・オプションを示します。ＤＯＳのバッチ・ファイルを使用しています。コマンド中の％１はバッチ・コマンドの引数で、コマンド中にファイル名を展開させるためのものです。

```
ＣＣ７８Ｋ３ －Ｃ３２０ －ＮＣＡ －Ｒ －ＱＣ －Ｇ －ＳＡ －Ｅ
               ①      ②     ③    ④    ⑤    ⑥    ⑦

             －Ｖ ％１．Ｃ
              ⑧    ⑨
```

①デバイス種別指定　：ＥＢボードでのテスト時に３２０、ＲＯＭ化時に３２２を指定します。ソース中の＃ｐｒａｇｍａ指令でも指定できますが、今回はコマンド行で指定しました。

②シンボル名ケース指定：シンボル名の大／小文字を区別します。
Ｃライブラリが－ＮＣＡで作られている（小文字を使用している）ため－ＮＣＡを指定しなければなりません。

③ＲＯＭ化指定　：この指定によりコンパイラはスタティック変数の初期値解決用のコードを出力してくれます。

④最適化指定　：－ＱＣは、ｃｈａｒタイプ変数の演算処理を最適化します。
－ＱＣを指定しなければｃｈａｒの演算はｗｏｒｄ拡張されます。
－ＱＲは、レジスタ変数をｓａｄｄｒ領域を使用して最大２０バイトまで許すオプションですが、前記の通り使用することはできません。

⑤ディバグ情報出力指定：ＥＢボードでシンボリック・ディバグを行なう場合指定します。

⑥アセンブラ・ソース・モジュール・ファイル作成指定
：アセンブル展開リストを”％１．ＡＳＭ”に出力します。

⑦エラー・リスト・ファイル作成指定
　　：エラー・リストを”％１．ＥＣＣ”に出力します。
⑧実行状態表示指定　　：コンパイルの実行状況をコンソールに表示させます。
⑨コンパイル対象　　：コンパイル対象ソースを指定します。

以上のオプション中②－ＮＣＡ、③－Ｒは省略することができます。
また、オブジェクト・モジュール・ファイル作成指定（－Ｏ）は省略時解釈により指定を略しています。

5.4 アセンブル

今回使用したアセンブル・オプションを示します。コンパイル同様バッチ・ファイルを使用しており、ファイル名を引数として渡します。

```
ＲＡ７８Ｋ３ －Ｃ３２０ －Ｇ －ＮＣＡ ％１．ＡＳＭ
              ①     ②     ③      ④
```

①デバイス種別指定　　：コンパイル同様ソース中で指定することもできます。
②ディバグ情報出力指定：コンパイル同様です。
③シンボル名ケース指定：Ｃの出力オブジェクトとリンケージをとる必要上、－ＮＣＡに統一する必要があります。
④アセンブル対象　　：アセンブル対象ソースを指定します。

コンパイルと違って－ＮＣＡオプションは省略することができません。注意してください。
オブジェクト・モジュール・ファイル出力指定（－Ｏ）は省略時解釈により指定を略しています。

5.5 Cスタティック変数初期値のロード

汎用システムの場合、初期値の解決はプログラム・ローダによって行なわれますが、本システム下ではユーザ自身がリセット時に解決しなければなりません。そのため、サンプル・プログラムが提供されています。

本節ではスタティック変数のロードの方法を説明します。

ＣＣ７８Ｋ３のスタティック変数は４種類にわけられ、セグメント名が異なります（ｃｏｎｓｔ指定の変数を除きます）。

	－セグメント名－	
	変数ｴﾘｱ(DSEG)	初期値ﾘｽﾄ(CSEG)
(1) ｓｒｅｇ変数、初期値リストあり	@@INIS	@@R_INIS
(2) ｓｒｅｇ変数、初期値リストなし	@@DATS	@@R_DATS
(3) 通常変数、初期値あり	@@INIT	@@R_INIT
(4) 通常変数、初期値なし	@@DATA	@@R_DATA

初期値リストが指定されていない(2),(4)の初期値リストはＡＬＬ０で、０保障を実現することができます。

ＣＣ７８Ｋ３提供のサンプル・プログラムはこの８つのセグメントの先頭と最後尾のアドレスを得、変数エリアに初期値をロードするためのものです。そのためリンケージ・パラメータのオブジェクトの指定は以下の本システム例のように、Ｃのオブジェクトを挟み込むように指定する必要があります。

```
RESET.REL TSK_I.REL MDL_IO.REL MDL_SYS.REL ROM_REL
```

実際の初期値のロードを０保障までやるか、初期値明示のものだけにするか、または初期値を使用しないかは選択することができます。

（初期値のロード手順については、付録のプログラム・リストを参照ください）

5.6 Cとアセンブラのリンク

Cからのアセンブラ関数呼出しに関する注意事項について解説します。
Cからシステムコールをｃａｌｌする例を示します。

```
extern char rcv_msg(void ** ppk_msg, unsigned short mbxid);

extern short mbox1;                                    ……①
#define MBOX1 ((unsigned short) (& mbox1))             ……②

struct MBLK1 {
             unsigned short dummy;
             char  msgtxt[80];
             };

void tsk_a()
        {
        struct MBLK1 *pk_msg;

        rcv_msg(& pk_msg, MBOX1);
        }
```

＊ _mbox1 は、ｺﾝﾌｨｷﾞｭﾚｰﾀに指定したﾒｲﾙﾎﾞｯｸｽIDのｱｾﾝﾌﾞﾗ･ｼﾝﾎﾞﾙです。
①、②によってｱｾﾝﾌﾞﾗ･ｼﾝﾎﾞﾙをunsigned short型の定数に変換しています。

これに対するアセンブラ展開を示します。

```
        extrn   _rcv_msg
        extrn   _mbox1                                 ……③
        public  _tsk_a
;
_tsk_a: push    hl                                     ……④
        movw    ax,sp
        subw    ax,#02H                                ……⑤
        movw    hl,ax
        movw    sp,ax
;
```

```
        movw    up,#_mbox1
        push    up
        movw    ax,hl
        push    ax
        call    !_rcv_msg
        movw    ax,sp
        addw    ax,#04H
        movw    sp,ax
;
        movw    ax,hl
        addw    ax,#02H
        movw    sp,ax
        pop     hl
        ret
```

⑥
⑦
⑧
⑨

③ tsk_aのアセンブラ名は_tsk_aとなります。同様に、外部との連絡レーベル名には全てアンダー・ライン（_）が付きます。
④ 関数内でauto変数を使用していればｈｌレジスタ・ペアを、レジスタ変数を使用していればそのレジスタが退避されます。
⑤ auto変数のためのスタック・フレームをｈｌレジスタ・ペアをベース・レジスタとして生成しています。
⑥ 関数の引き数を最終引数から順番にスタックへ積んでいきます。
⑦ ｓｐを元に戻します（スタック上の引き数の破棄）。
⑧ auto変数のスタック・フレームを削除します。
⑨ ｈｌ，ｖｐ，ｒｐ３の内使用したものを復帰します。

アセンブラ側でも、③④⑥⑦⑨の規則に則って記述しなければなりません。特に、ｈｌ，ｖｐ，ｒｐ３を破壊しないように、またタスク間で共通に利用する関数の場合リエントラントにしなければなりません。

5.7 リアルタイムOSとのリンク

ＯＳとのリンクは、第１に、リセット時のタスク情報とオブジェクト管理用制御ブロック情報の授受があります。これらの情報はコンフィギュレータによって生成します。第２に、システムコールに関するもの、第３に、特殊なものとして、タイマ・ハンドラとメモリ・バンクに関するものがあります。

本節では、これら各々について解説します。

１）コンフィギュレータの生成情報

コンフギュレータにより生成する情報は以下に示す２つです。

・システム初期化情報……コンフィギュレーション・テーブルとオブジェクト管理用制御ブロックの生成

タスク・コントロール・ブロック（ＴＣＢ）とメモリプールの初期化情報、及びイベントフラグ（ＥＶＴ）、セマフォ（ＳＥＭ）、メイルボックス（ＭＢＯＸ）の構成を指定します。

初期化情報は、コンフィギュレーション・テーブルとして生成され、かつ各管理ブロックのエリアが確保されます。

・システムコール情報……システムコール・エントリ・テーブルの生成

システムで使用するシステムコールを指定します。

ここで指定しなかったシステムコールはリンケージから除外されます。

（割り込みハンドラ専用のシステムコールを除きます）

以下に、本システムのコンフィギュレータの指定値を示します。

１．１）システム初期化情報

コンフィギュレータの出力結果については付録Ｂ．CF1.ASM、及びCF2.ASMに掲載しています。

```
**********************************************************
*                                                        *
*              --- System Information ---                 *
*                                                        *
**********************************************************
(1)       Kernel_location_address          ?SYSRT  ----------①
(2)       Initial_task_ID                  _tid_i  ----------②
```

① リアルタイムＯＳの初期化ルーチン名 "?SYSRT" を指定します。
② Ｃｿｰｽから参照する必要があるためｱﾝﾀﾞｰ･ﾗｲﾝ付きの名前を指定します。

```
************************************************************
*                                                          *
*              --- Memorypool Information ---              *
*                                                          *
************************************************************
(0)       Memory_pool_count              2        ----------①
(1)       Memory_pool_No.1
            Memory_pool_ID               _mpl1    ----------②
            Memory_block_count           2        ----------③
            Memory_block_size            6        ----------④
(2)       Memory_pool_No.2
            Memory_pool_ID               _mpl2
            Memory_block_count           2
            Memory_block_size            4
```

① システムで使用するメモリプール数を指定します。
②③④ メモリプール各々についてﾒﾓﾘﾌﾟｰﾙID、ﾒﾓﾘﾌﾟｰﾙの持つﾌﾞﾛｯｸ数、及びﾌﾞﾛｯｸのｻｲｽﾞを指定します。

メモリブロックをメイルボックスへのメッセージ・エリアとして使用する場合、ブロックの先頭２バイトはメッセージのキューイングのためにシステムが使用するため、２バイト大きく指定しなければなりません。

```
************************************************************
*                                                          *
*                 --- Task Information ---                 *
*                                                          *
************************************************************
(0)       Task_count                     5        ----------①
(1)       Task_No.1
            Task_ID                      _tid_i   ----------②
            Task_priority                1        ----------③
            Reg_bank_ID                  6        ----------④
            Task_start_address           _tsk_i   ----------⑤
            Initail_stack_pointer        7100H    ----------⑥
```

```
        Memory_bank_ID              0       ·····⑦
(2)     Task_No.2
        Task_ID                     _tid_im
        Task_priority               3
        Reg_bank_ID                 6
        Task_start_address          _tsk_im
        Initail_stack_pointer       7200H
        Memory_bank_ID              0
(3)     Task_No.3
        Task_ID                     _tid_id
        Task_priority               1
        Reg_bank_ID                 6
        Task_start_address          _tsk_id
        Initail_stack_pointer       7300H
        Memory_bank_ID              0
(4)     Task_No.4
        Task_ID                     _tid_il
        Task_priority               2
        Reg_bank_ID                 6
        Task_start_address          _tsk_il
        Initail_stack_pointer       7400H
        Memory_bank_ID              0
(5)     Task_No.5
        Task_ID                     _tid_cm
        Task_priority               4
        Reg_bank_ID                 6
        Task_start_address          _tsk_cm
        Initail_stack_pointer       7500H
        Memory_bank_ID              0
```

① システムの全ﾀｽｸ数を指定します。
② ﾀｽｸIDをｱﾝﾀﾞｰ･ﾗｲﾝ付きの名前で指定します。
③ ﾀｽｸの初期優先度(1-16,1が最高優先度)を指定します。
④ ﾀｽｸの使用するﾚｼﾞｽﾀ･ﾊﾞﾝｸ(0-6､0-5は1ﾀｽｸ専有,6は共用)を指定します。
⑤ Ｃの関数としてﾀｽｸの記述を行なった場合、ｱﾝﾀﾞｰ･ﾗｲﾝ付きの関数名を指定します。
⑥ スタック・ポインタの初期値を指定します。
⑦ ﾒﾓﾘ･ﾊﾞﾝｸIDを指定します（使用しない場合0を指定）。

```
************************************************************
*                                                          *
*              --- Eventflag Information ---               *
*                                                          *
************************************************************
(0)       Eventflag_count                  0      ----------①
```

① 使用するイベントフラグ数を指定します。

```
************************************************************
*                                                          *
*              --- Semaphore Information ---               *
*                                                          *
************************************************************
(0)       Semaphore_count                  0      ----------①
```

① 使用するセマフォ数を指定します。

```
************************************************************
*                                                          *
*              --- Mailbox Information ---                 *
*                                                          *
************************************************************
(0)       Mailbox_count                    2      ----------①
(1)       Mailbox_No.1
            Mailbox_ID                     _mbox_im ---------②
(2)       Mailbox_No.2
            Mailbox_ID                     _mbox_id
```

① 使用するメイルボックス数を指定します。
② ﾒｲﾙﾎﾞｯｸｽIDをｱﾝﾀﾞｰ･ﾗｲﾝ付きの名前で指定します。

１．２）システムコール情報

コンフィギュレータの出力結果については、付録Ｂ．SYSTBL.ASMに掲載しています。

```
*********************************************************
*                                                       *
*          --- Task Related to System Call ---         *
*                                                       *
*********************************************************
(1)       STA_TSK                          Yes
(2)       EXT_TSK                          Yes
(3)       TER_TSK                          No
(4)       CHG_PRI                          No
(5)       ROT_RDQ                          No
(6)       TSK_STS                          No
(7)       SLP_TSK                          Yes
(8)       WAI_TSK                          Yes
(9)       WUP_TSK                          Yes
(10)      CAN_WUP                          No

*********************************************************
*                                                       *
*       --- Eventflag Related to System Call ---        *
*                                                       *
*********************************************************
(1)       SET_FLG                          No
(2)       CLR_FLG                          No
(3)       WAI_FLG                          No
(4)       CWAI_FLG                         No
(5)       POL_FLG                          No
(6)       CPOL_FLG                         No

*********************************************************
*                                                       *
*      --- Semaphore Related to System Call ---         *
*                                                       *
*********************************************************
(1)       SIG_SEM                          No
(2)       WAI_SEM                          No
(3)       PREG_SEM                         No
```

```
*********************************************************
*                                                       *
*       --- Message Related to System Call ---         *
*                                                       *
*********************************************************
(1)       SND_MSG                           Yes
(2)       RCV_MSG                           Yes
(3)       PRCV_MSG                          No

*********************************************************
*                                                       *
*       --- Memory Related to System Call ---          *
*                                                       *
*********************************************************
(1)       PGET_BLK                          Yes
(2)       REL_BLK                           Yes

*********************************************************
*                                                       *
*      --- Interrupt Related to System Call ---        *
*                                                       *
*********************************************************
(1)       DEF_INT                           No

*********************************************************
*                                                       *
*      --- Another Related to System Call ---          *
*                                                       *
*********************************************************
(1)       GET_VER                           No
```

2）システムコールとのリンク

システムコールはCALLTテーブルを介して呼び出されます。以下に例を示します。

```
                EXTRN   BRPROC
                EXTRN   ICHG_PRI,IROT_RDQ,IWUP_TSK
                EXTRN   ISET_FLG,ISIG_SEM,ISND_MSG
@@CALLTO        CSEG    AT 40H
                DW      BRPROC          ;40H
                DW      ICHG_PRI        ;42H
                DW      IROT_RDQ        ;44H
                DW      IWUP_TSK        ;46H
                DW      ISET_FLG        ;48H
                DW      ISIG_SEM        ;4AH
                DW      ISND_MSG        ;4CH
```

この例は割り込みハンドラ用のシステムコールの全てを使用する場合のものです。
使用しないハンドラ用のシステムコールはCALLT定義とEXTRN定義をしないことでリンケージ除外することができます。

40H番地のBRPROCは、タスクからのシステムコールのためのもので必須です。

またタスクのプログラム記述にＣを使用した場合、Ｃインタフェース・ライブラリとリンクをとる必要があります（次節で解説します）。

最後に、RET_INTとRET_WUPシステムコールは例外的にCALLTを使用しません。これらとのリンクのためには割り込みハンドラから直接EXTRN宣言を行ないます（本システムでRET_WUPを使用しています．付録Ｃ．HANDLER.ASMを参照ください．）。

3）リセット動作

システムのリセット・ルーチンではSFR等の設定とＣ変数の初期値の解決の後、規定の方法でＯＳの初期化ルーチンにジャンプしなければなりません。
ジャンプの方法については次項で示します。

4）タイマ・オペレーション

タイマ・オペレーションを使用するためには、インターバル・タイマの設定とタイマ・ハンドラのベクタ・テーブル登録が必要です。

タイマはＴＭ１、コンペア・レジスタにはＣＭ１０を、割り込み要求にＩＮＴＣＭ１０を

使用します。
ＩＮＴＣＭ１０のベクタ・テーブルに、タイマ・ハンドラの入口名 ?TMDSP を指定します。
以下に、本システムの例を示します。

```
        EXTRN   SYS_INF         ;CONFIGURATION TABLE
        EXTRN   ?TMDSP          ;TIMER HANDLER
@VCTR1  CSEG    AT 00H
        DW      RES_RTN         ;RESET
@VCTR3  CSEG    AT 1EH
        DW      ?TMDSP          ;INTCM10
@@CODE  CSEG
RES_RTN:
        MOV     PRM,#10H        ;TM1 COUNT CLOCK
        MOV     RPUM,#20H       ;INTERVAL TIMER MODE
        MOVW    CM10,#1F3H      ;1ms INTERVAL
        MOV     TMC,#80H        ;TM1 START

        MOVW    HL,#SYS_INF
        MOVW    AX,[HL]
        BR      AX              ;to realtime OS
        END
```

タイマ・オペレーションを使用するには、上記以外に割り込みマスク・フラグCMMK10の解除をしなければなりません。
ＯＳの初期化ルーチンへジャンプするまでスタックを使用しない場合、ＳＰレジスタの初期設定は不要です。OSの初期化ルーチンは自身のスタック・エリアを使用します。

５）メモリ・バンク

ＲＸ７８Ｋ／Ⅲはメモリ・バンクに対応しており、タスク別に、使用するメモリ・バンクを切替えることができます。
メモリ・バンクに関するＯＳのサポートは、タスク別のメモリ・バンクＩＤの記憶とバンク切替えの指示です。
タスク別のメモリ・バンクＩＤの設定は、コンフィギュレータで行ないます。メモリ・バンクを使用しないのであれば、０を設定します。
メモリ・バンク切替えの指示は、スケジューラにより関数ＣＡＬＬの形で行なわれます。

@ＢＮＫＳＴ：Ａレジスタで指示されるメモリ・バンクへの切替えを行ない、
Ａレジスタ内容（メモリ・バンクＩＤ）を記憶します。

@ＢＮＫＲＤ：記憶していたメモリ・バンクＩＤをＡレジスタへ返します。

この２つの関数は、ユーザが作成しなければなりません。
メモリ・バンクを使用しない場合、２つの関数はＲＥＴ命令だけを実行させます。

5.8 リンク

今回使用したリンケージ・リストを示します。
コンパイル同様、バッチ・ファイルを使用しており、かつコマンドの文字列長が２５５を越えるため、リンク・パラメータ・ファイルを使用しています。

＜バッチ・ファイル＞

```
LK78K3 -FSYSTEM.PLK
          ①
```

＜リンク・パラメータ・ファイル＞

```
-BURXROM.LIB -BLIBRARY.LIB -BCL320.LIB -DSYSTEM.DR -G -KP -PSYSTEM.MAP
    ②           ③            ④          ⑤      ⑥ ⑦     ⑧

-OSYSTEM.LNK RESET.REL CF1.REL CF2.REL SYSTBL.REL TSK_I.REL MDL_IO.REL
    ⑨                                  ⑩

MDL_SYS.REL HANDLER.REL ROM.REL
```

①ﾘﾝｸ･ﾊﾟﾗﾒｰﾀ･ﾌｧｲﾙ指定
②③④ﾗｲﾌﾞﾗﾘ･ﾌｧｲﾙ指定
⑤ﾘﾝｸ･ﾃﾞｨﾚｸﾃｨﾌﾞ･ﾌｧｲﾙ指定
⑥ﾃﾞｨﾊﾞｸﾞ情報出力指定
⑦PUBLICｼﾝﾎﾞﾙ出力指定
⑧ﾘﾝｸ･ﾘｽﾄ･ﾌｧｲﾙ出力指定
⑨ﾛｰﾄﾞ･ﾓｼﾞｭｰﾙ･ﾌｧｲﾙ出力指定
⑩ｵﾌﾞｼﾞｪｸﾄ･ﾘｽﾄ

スタック解決用シンボル生成指定オプション（－Ｓ）は、シングル・タスク・システムでは有効ですが、タスク別にスタック・エリアを用意しなければならないリアルタイム・システムでは使用することはできません。

リンク・リスト・ファイルの情報指定オプションの内、次の２つは、省略時解釈により指定を略しています。

－ＫＭ　：　ロケーション・マップの出力

－ＫＤ　：　ﾘﾝｸ･ﾃﾞｨﾚｸﾃｨﾌﾞ･ﾌｧｲﾙ内容の出力

以下、使用したライブラリ・ファイルと、リンク・ディレクティブ・ファイルについて解説します。

１）ライブラリ・ファイル

３つのライブラリ・ファイルとリンクします。

ＣＬ３２０　．ＬＩＢ	：Ｃライブラリ関数
７８Ｋ３ＲＯＭ．ＬＩＢ	：ＯＳニュークリアス
ＩＮＴＦＡＣＥ．ＬＩＢ	：システムコールのＣインタフェース・ライブ

Ｃライブラリ関数は、ROM化時にはCL322.LIBを使用します。
また、DEF_INFで動的な割り込みハンドラの定義/削除を行なう場合78K3ROM.LIBの代わりに78K3RAM.LIBとリンクします。

２）リンク・ディレクティブ・ファイル

リンク・ディレクティブ・ファイルはロケーションをリンカに指示するためのものです。
リンク・ディレクティブ・ファイルの指定がない場合、以下の手順でロケーションが決定されます。

ａ）絶対番地配置指定（ＡＴ指定等によります）セグメントを配置

ｂ）省略時のＲＯＭ／ＲＡＭ領域内にセグメントの出現順にリロケータブル・セグメントを配置

省略時のメモリ領域はデバイスによって異なります。
今回、ディバグとＲＯＭ化時の使用デバイスが異なる点、また、ＯＳの予約ＲＡＭエリアとＥＢボードのモニタ・エリア、及びスタック・エリア(7000H-7500H)への配置を回避するためにリンク・ディレクティブ・ファイルを使用しています。

```
MEMORY VCT1 : (00000H,00002H) ┐
MEMORY VCT2 : (00004H,0003AH) ┘──①
MEMORY CLT  : (00040H,00040H) ──②
MEMORY ROM  : (00080H,03F80H) ──③
MEMORY RAM  : (04000H,03000H) ──⑤
```

```
MEMORY SDR : (0FE20H,00060H)  ------⑥
MERGE @VCTR1 : = VCT1         ------
MERGE @VCTR2 : = VCT2         ------①
MERGE @VCTR3 : = VCT2         ------
MERGE @@CALLTO : = CLT        ------②
MERGE URX78K3 : AT(80H)       ------④
MERGE @@INIS : = SDR          ------
MERGE @@DATS : = SDR          ------⑥
```

① ＥＢボードのモニタ・エリア（ＮＭＩベクタとブレーク命令ベクタ）を避けるために、ベクタ領域を　２分割して領域を確保しています。
@VCTR1はリセット・ベクタの、
@VCTR2はキー割り込みベクタの、
@VCTR3はINTCM10ベクタのセグメント名です。

② CALLTテーブル・エリアを定義しています。
@@CALLTOはシステムコールのCALLTテーブル・セグメント名です。

③ ＲＯＭ領域をμＰＤ７８３２２の内部ＲＯＭ中、ベクタ・エリアとCALLTエリアを避けて再定義しています。

④ リアルタイムＯＳはコード・セグメント名に ＵＲＸ７８Ｋ を使用しています。
それに対して、タスク、割り込みハンドラ、リセット・ルーチンはセグメント名にＣのコード・セグメント名 @@CODEを使用しています。
この指定によりＯＳ部とその他のロケーションを明確に分離することができます。

⑤ ＲＡＭ領域を外部ＲＡＭ中、4000H～6FFFHに再定義しています。
ＯＳの予約領域（0FC80Hから50Hﾊﾞｲﾄ）を外しているのは、ＯＳは予約領域の確保を行なわないためです。

⑥ ＲＡＭ領域からｓａｄｄｒ領域が分離してしまったため、Ｃのｓｒｅｇ変数の配置エリアとしてｓａｄｄｒ領域を定義・配置します。

ＲＸ７８Ｋ／Ⅲを使用する場合、⑤のＲＡＭ領域の再定義は必須となります。

付録A. I/Oインタフェース

1）ポート割り付け

ポート		機能名称	入出力	端　子　機　能	ｱｸﾃｨﾌﾞ･ﾚﾍﾞﾙ
0	0	ポート0	出　力	ステッピング・モータ励磁信号	Ｈｉｇｈ
	1				
	2				
	3				
2	1	ＩＮＴＰ０	入　力	チャタレス・キー（運転キー）	Ｌｏｗ ＊1
	2	ＩＮＴＰ１		チャタレス・キー（ＵＰキー）	
	3	ＩＮＴＰ２		チャタレス・キー（ＤＯＷＮキー）	
3	0	ＴｘＤ	入出力	ＬＥＤ	Ｈｉｇｈ

表A-1

＊１：外部割り込みモード・レジスタの設定によります。

2）割り込み制御フラグ関連

割り込み要求信号	＊2 ベクタ・テーブル アドレス	ディフォールト 優先順位	割り込み制御レジスタのフラグ名称	
			割り込みマスク・フラグ・レジスタ	割り込み要求フラグ・レジスタ
ＩＮＴＰ０	０００８Ｈ	１	ＰＭＫ０	ＰＩＦ０
ＩＮＴＰ１	０００ＡＨ	２	ＰＭＫ１	ＰＩＦ１
ＩＮＴＰ２	０００ＣＨ	３	ＰＭＫ２	ＰＩＦ２
ＩＮＴＣＭ１０	００１ＥＨ	１２	ＣＭＭＫ１０	ＣＭＩＦ１０

表A-2

＊２：ＴＰＦ＝０とし、ベクタ・テーブルを０番地～へ配置しています。

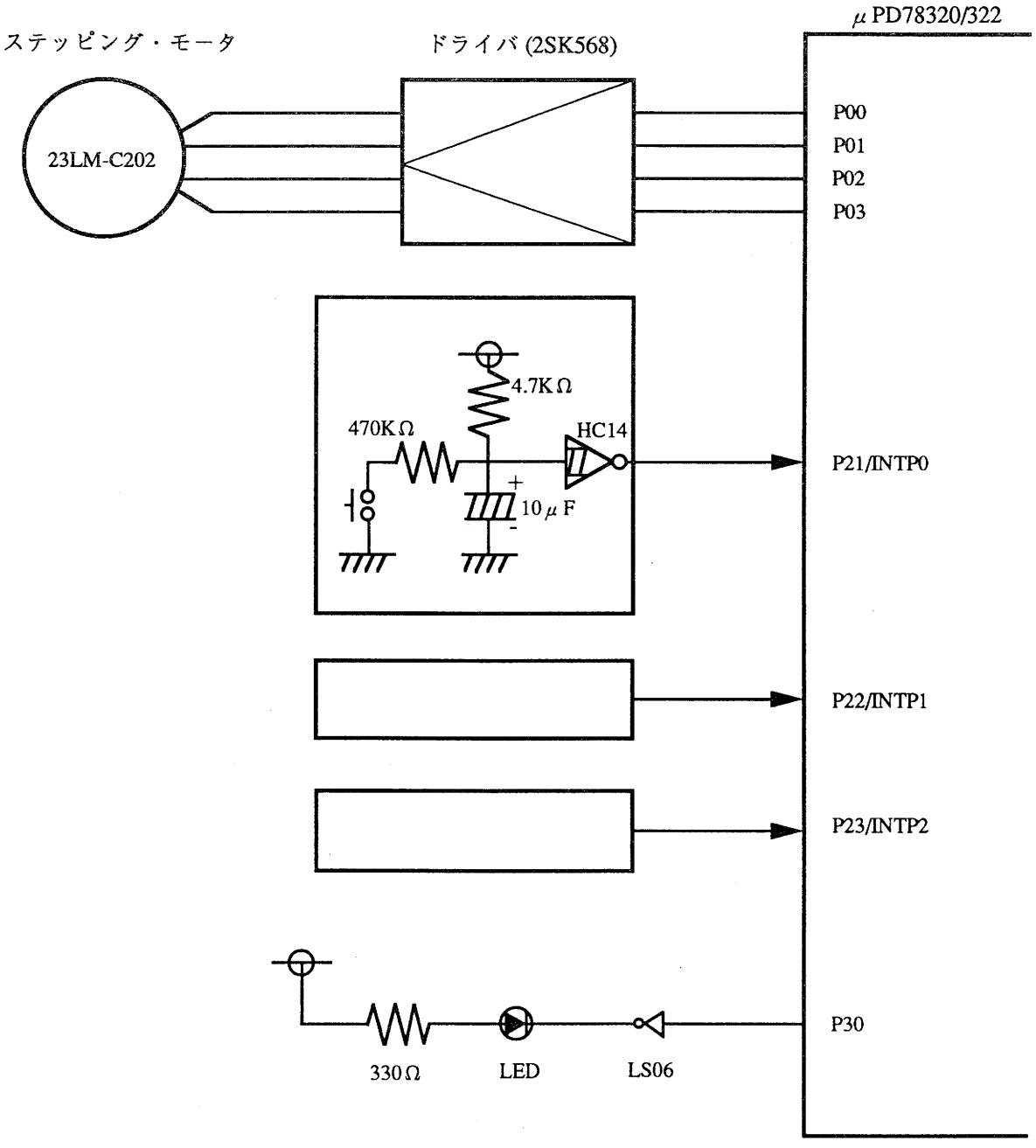
3）特殊機能レジスタ設定

ｱﾄﾞﾚｽ	特殊機能ﾚｼﾞｽﾀの名称	略号	R/W	操作単位	ﾘｾｯﾄ時	設定内容
FF62H	ﾎﾟｰﾄ・ﾘｰﾄﾞ・ｺﾝﾄﾛｰﾙ・ﾚｼﾞｽﾀ	PRDC	W	ﾋﾞｯﾄ	00H	0000 0001B
FF20H	ﾎﾟｰﾄ0ﾓｰﾄﾞ・ﾚｼﾞｽﾀ	PM0	W	ﾊﾞｲﾄ	FFH	00H
FF23H	ﾎﾟｰﾄ3ﾓｰﾄﾞ・ﾚｼﾞｽﾀ	PM3	W	ﾊﾞｲﾄ	xxx1 1111B	xxx1 1110B
FFB2H	ﾌﾟﾘｽｹｰﾗ・ﾓｰﾄﾞ・ﾚｼﾞｽﾀ	PRM	W	ﾊﾞｲﾄ	00H	10H
FFBFH	RPUﾓｰﾄﾞ・ﾚｼﾞｽﾀ	RPUM	W	ﾊﾞｲﾄ	00H	20H
FF7CH	ｺﾝﾍﾟｱ・ﾚｼﾞｽﾀ10	CM10	W	ﾜｰﾄﾞ	不　定	09 C3H
FFB0H	ﾀｲﾏ・ｺﾝﾄﾛｰﾙ・ﾚｼﾞｽﾀ	TMC	W	ﾊﾞｲﾄ	00H	80H
FF00H	ﾎﾟｰﾄ0	P0	W	ﾊﾞｲﾄ	不　定	0000 ****B
FF03H	ﾎﾟｰﾄ3	P3	R/W	ﾋﾞｯﾄ	不　定	0000 000*B
FFE4H	割り込みﾏｽｸ・ﾌﾗｸﾞ・ﾚｼﾞｽﾀ 0L	MK0L	W	ﾋﾞｯﾄ	FFH	1111 **01B
FFE5H	割り込みﾏｽｸ・ﾌﾗｸﾞ・ﾚｼﾞｽﾀ 0H	MK0H	W	ﾋﾞｯﾄ	FFH	1110 1111B
FFE0H	割り込み要求ﾌﾗｸﾞ・ﾚｼﾞｽﾀ 0L	IF0L	W	ﾋﾞｯﾄ	00H	0000 **00B
FFC6H	ﾌﾟﾛｸﾞﾗﾏﾌﾞﾙ・ｳｴｲﾄ制御ﾚｼﾞｽﾀ	PWC	W	ﾊﾞｲﾄ	22H	00H
FFC9H	ﾌｪｯﾁ・ｻｲｸﾙ・ｺﾝﾄﾛｰﾙ・ﾚｼﾞｽﾀ	FCC	W	ﾊﾞｲﾄ	00H	03H

表A-3

＊３：R/W、操作単位の項目は、本システムでの操作を示しています。
＊４：設定内容の＊は、操作ビットを示します。
＊５：ＦＣＣの設定は、μPD78322の場合にのみ03Hにし、高速フェッチを行ないます。
各レジスタのフォーマットについては、
「μPD78322　ユーザーズ・マニュアル」(IEU-619)を参照してください。

4）回路図



図A－1

付録B．構成記述部プログラム・リスト

1）ＣＦ１．ＡＳＭ……コンフィギュレーション・テーブル

```
        NAME    CF1
;
        PUBLIC  SYS_INF
;
        EXTRN   ?SYSRT,_tid_i,_mpl1,_mpl2,_tid_im,_tid_id,_tid_il,_tid_cm,_tsk_i
        EXTRN   _tsk_im,_tsk_id,_tsk_il,_tsk_cm,_mbox_im,_mbox_id
;
;*************************************************************
;*                                                           *
;*        --- SYSTEM INITIALIZE INFORMATION TABLE ---        *
;*                                                           *
;*************************************************************
;
@@CODE  CSEG
SYS_INF:
NUC_LOC:        DW      ?SYSRT
;
INIT_ID:        DW      _tid_i
;
MP_CNT: DB      02H
MP_ID0: DW      _mpl1
B_CNT0: DB      02H
B_SIZ0: DB      06H
;
MP_ID1: DW      _mpl2
B_CNT1: DB      02H
B_SIZ1: DB      04H
;
T_CNT:  DB      05H
T_ID0:  DW      _tid_i
T_PRI0: DB      01H
T_RBK0: DB      06H
T_ADR0: DW      _tsk_i
T_SP0:  DW      7100H
T_MBK0: DB      00H
;
```

```
T_ID1:  DW      _tid_im
T_PRI1: DB      03H
T_RBK1: DB      06H
T_ADR1: DW      _tsk_im
T_SP1:  DW      7200H
T_MBK1: DB      00H
;
T_ID2:  DW      _tid_id
T_PRI2: DB      01H
T_RBK2: DB      06H
T_ADR2: DW      _tsk_id
T_SP2:  DW      7300H
T_MBK2: DB      00H
;
T_ID3:  DW      _tid_il
T_PRI3: DB      02H
T_RBK3: DB      06H
T_ADR3: DW      _tsk_il
T_SP3:  DW      7400H
T_MBK3: DB      00H
;
T_ID4:  DW      _tid_cm
T_PRI4: DB      04H
T_RBK4: DB      06H
T_ADR4: DW      _tsk_cm
T_SP4:  DW      7500H
T_MBK4: DB      00H
;
E_CNT:  DB      00H
;
S_CNT:  DB      00H
;
M_CNT:  DB      02H
M_ID0:  DW      _mbox_im
;
M_ID1:  DW      _mbox_id
;
        ENDS
;
;
        END
```

2）ＣＦ２．ＡＳＭ・・・・・・・・・・・・・・・オブジェクト管理用制御ブロック

```
        NAME    CF2
;
        PUBLIC  _tid_i,_tid_im,_tid_id,_tid_il,_tid_cm,_mbox_im,_mbox_id,_mpl1
        PUBLIC  _mpl2
;
;*************************************************************
;*                                                           *
;*     --- SYSTEM INITIALIZE INFORMATION DATA TABLE ---      *
;*                                                           *
;*************************************************************
;
SYS_DAT DSEG
_tid_i:
                DS      18
;
_tid_im:
                DS      18
;
_tid_id:
                DS      18
;
_tid_il:
                DS      18
;
_tid_cm:
                DS      18
;
_mbox_im:
                DS      8
;
_mbox_id:
                DS      8
;
_mpl1:
                DS      16
;
_mpl2:
                DS      12
;
```

```
        ENDS
;
;
        END
```

3）ＳＹＳＴＢＬ．ＡＳＭ……システムコール・エントリ・テーブル

```
        NAME    sysent
;
        PUBLIC  ENT_TBL
;
        EXTRN   STA_TSK,EXT_TSK,SLP_TSK,WAI_TSK,WUP_TSK,SND_MSG,RCV_MSG,PGET_BLK
        EXTRN   REL_BLK
;
;*************************************************************
;*                                                           *
;*              --- SYSTEM CALL ENTRY TABLE ---              *
;*                                                           *
;*************************************************************
;
@@TBL   CSEG
ENT_TBL:
        DW      STA_TSK
        DW      EXT_TSK
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      SLP_TSK
        DW      WAI_TSK
        DW      WUP_TSK
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
        DW      0000H
```

```
        DW      0000H
        DW      SND_MSG
        DW      RCV_MSG
        DW      0000H
        DW      PGET_BLK
        DW      REL_BLK
        DW      0000H
        DW      0000H
;
        ENDS
;
;
        END
```

付録C．タスク、ハンドラ部プログラム・リスト

1）COMMON．INC……システム共通情報の定義

```
#pragma sfr


#define forever 1
#define on      1
#define off     0


/* MOTOR DRIVE SPEED LEVEL DEFINE */
#define speedlvl        5
#define maxlvl          (speedlvl-1)
#define minlvl          0


/* MASK FLAG REGISTER */
#define CMMK10  MK0H.4
#define PMK0    MK0L.1
#define PMK1    MK0L.2
#define PMK2    MK0L.3


/* INTERRUPT REQUEST FLAG REGISTER */
#define PIF1    IF0L.2
#define PIF2    IF0L.3


/* PORT REGISTER BIT */
#define P30     P3.0


/* MESSAGE FUNC. CODE for I/O CONTROL MAIN TASK */
#define fc_start        1
#define fc_stop         2
#define fc_speed        3
```

```
/* CONTROL SWITCH EVENT REQUEST */
#define start           1
#define stop            1
#define fastup          2
#define slowdown        4


/* message block for tsk_im */
struct  MBLK1   {
                unsigned short  que;
                unsigned short  fc;
                unsigned short  speed;
                };


/* message block for tsk_id */
struct  MBLK2   {
                unsigned short  que;
                unsigned short  count;
                };
```

2）ＲＥＦＱＵＥ．ＩＮＣ……オブジェクト管理用制御ブロックの参照宣言

```
/*                                      */
/*  RX78K3 interface block reference    */
/*                                      */

/* TASK CONTROL BLOCK */

extern  short   tid_i;
extern  short   tid_im;
extern  short   tid_id;
extern  short   tid_il;
extern  short   tid_cm;

#define TID_I   ((unsigned short) (& tid_i))
#define TID_IM  ((unsigned short) (& tid_im))
#define TID_ID  ((unsigned short) (& tid_id))
#define TID_IL  ((unsigned short) (& tid_il))
#define TID_CM  ((unsigned short) (& tid_cm))
```

```
/* INTERFACE QUEUE */

extern  short   mbox_im;
extern  short   mbox_id;

#define MBOX_IM ((unsigned short) (& mbox_im))
#define MBOX_ID ((unsigned short) (& mbox_id))


/* MEMORY POOL */

extern  short   mpl1;
extern  short   mpl2;

#define MPL1    ((unsigned short) (& mpl1))
#define MPL2    ((unsigned short) (& mpl2))
```

3）ＬＩＢ．ＩＮＣ……システムコール参照宣言

```
/*  RX78K3 system call reference */

extern  char    sta_tsk();
extern  void    ext_tsk();
extern  char    ter_tsk();
extern  char    chg_pri();
extern  char    rot_rdq();
extern  char    tsk_sts();
extern  char    slp_tsk();
extern  char    wai_tsk();
extern  char    wup_tsk();
extern  char    can_wup();

extern  char    set_flg();
extern  char    clr_flg();
extern  char    wai_flg();
extern  char    cwai_flg();
extern  char    pol_flg();
extern  char    cpol_flg();
```

```
extern  char    sig_sem();
extern  char    wai_sem();
extern  char    preq_sem();
extern  char    snd_msg();
extern  char    rcv_msg();
extern  char    prcv_msg();

extern  char    pget_blk();
extern  char    rel_blk();

extern  char    def_int();

extern  char    get_ver();


/*  RX78K3 system call RETURN CODE */
#define E_OK            0
#define E_NODMT         1
#define E_DMT           2
#define E_QOVR          3
#define E_TMOUT         4
#define E_PLFAIL        5
#define E_CTX           6
```

４）ＭＤＬ＿ＩＯ．Ｃ……Ｉ／Ｏ制御部モジュール

```
#include "common.inc"
#include "refque.inc"
#include "lib.inc"


/* LED blinking count */
#define turn_c          120
#define putout_c        60


sreg unsigned short led_cnt; /*LED status counter*/
```


```
/*****************************************************************/
/*                                                               */
/*     STEPPING MOTOR CONTROL SYSTEM                             */
/*                                                               */
/*        I/o control module  Main task                          */
/*                                                               */
/*****************************************************************/
void tsk_im()
        {
        struct MBLK1 *p_msgr;
        struct MBLK2 *p_msgs;
        static const unsigned short waitcnt[speedlvl] =
                                    {32,16,8,4,2};


        do {
           rcv_msg(& p_msgr,MBOX_IM);

           switch(p_msgr->fc) {
              case fc_speed: wup_tsk(TID_ID);

              case fc_start: pget_blk(& p_msgs,MPL2);
                             p_msgs->count = waitcnt[p_msgr->speed];
                             snd_msg(MBOX_ID,p_msgs);
                             break;

              case fc_stop:  wup_tsk(TID_ID);
                             P30=off; led_cnt=0;
              }

           rel_blk(MPL1,p_msgr);
           } while(forever);
        }
```

```
/*****************************************************************/
/*                                                               */
/*     STEPPING MOTOR CONTROL SYSTEM                             */
/*                                                               */
/*        I/o control module  motor Drive task                   */
/*                                                               */
/*****************************************************************/
void tsk_id()
        {
        struct MBLK2 *p_msg;

        static const unsigned char signal[] =
                              {0xC,0x6,0x3,0x9};
                             /*{0x8,0xC,0x4,0x6,0x2,0x3,0x1,0x9};*/
                             /*{0x8,0x4,0x2,0x1};                 */
        register unsigned short c = 0;


        do {
           rcv_msg(& p_msg,MBOX_ID);

           do {
              if (wai_tsk(p_msg->count) == E_OK) break;

              P0 = signal[c++];
              if (c >= sizeof signal) c=0;
              wup_tsk(TID_IL);
              } while (forever);

           rel_blk(MPL2,p_msg);
           } while (forever);
        }
```

```
/*****************************************************************/
/*                                                               */
/*     STEPPING MOTOR CONTROL SYSTEM                             */
/*                                                               */
/*        I/o control module  Led control task                   */
/*                                                               */
/*****************************************************************/
void tsk_il()
        {
        led_cnt=0;
        do {
           slp_tsk();

           led_cnt++;
           if (P30) {
              if (led_cnt == turn_c) {P30=off; led_cnt=0;}}
           else {
              if (led_cnt == putout_c) {P30=on; led_cnt=0;}}
           } while (forever);
        }
```

5）ＭＤＬ＿ＳＹＳ．Ｃ・・・・・・・・・・・システム制御部モジュール

```
/*#define MSKCTL*/

#include "common.inc"
#include "refque.inc"
#include "lib.inc"

/* EVENT REQUEST BYTE */
sreg unsigned char evt_req;

/*****************************************************************/
/*                                                               */
/*     STEPPING MOTOR CONTROL SYSTEM                             */
/*                                                               */
/*        system Control module Main task                        */
/*                                                               */
/*****************************************************************/
void tsk_cm()
```

```
{
struct MBLK1 *p_msg;
register unsigned short speed;


speed = 2;
do {
   slp_tsk();

   if (evt_req == start) {
      pget_blk(& p_msg,MPL1);
      p_msg->fc    = fc_start;
      p_msg->speed = speed;
      snd_msg(MBOX_IM,p_msg);

      do {
         evt_req = 0;
         slp_tsk();

         if (evt_req == stop) {
            pget_blk(& p_msg,MPL1);
            p_msg->fc  = fc_stop;
            snd_msg(MBOX_IM,p_msg);

            break;
            }
#ifdef MSKCTL
         if (evt_req == fastup) {
#else
         if (speed != maxlvl && evt_req == fastup) {
#endif
            speed++;

            pget_blk(& p_msg,MPL1);
            p_msg->fc    = fc_speed;
            p_msg->speed = speed;
            snd_msg(MBOX_IM,p_msg);
#ifdef MSKCTL
            if (speed == maxlvl) PMK1 = 1;
            PIF2 = 0;
            PMK2 = 0;
```

```
#endif
                    }
#ifdef MSKCTL
                if (evt_req == slowdown) {
#else
                if (speed != minlvl && evt_req == slowdown) {
#endif
                    speed--;

                    pget_blk(& p_msg,MPL1);
                    p_msg->fc    = fc_speed;
                    p_msg->speed = speed;
                    snd_msg(MBOX_IM,p_msg);
#ifdef MSKCTL
                    if (speed == minlvl) PMK2 = 1;
                    PIF1 = 0;
                    PMK1 = 0;
#endif
                    }

                } while (forever);
            }

          evt_req = 0;
          } while (forever);

       }
```

6）ＴＳＫ＿Ｉ．Ｃ……イニシャル・タスク

```
#include "common.inc"
#include "refque.inc"
#include "lib.inc"


/* EVENT REQUEST BYTE */
extern sreg unsigned char evt_req;


/*****************************************************************/
```

```
/*                                                              */
/*     STEPPING MOTOR CONTROL SYSTEM                            */
/*                                                              */
/*        INITIALIZE TASK                                       */
/*                                                              */
/****************************************************************/
void tsk_i()
        {
        sta_tsk(TID_ID);
        sta_tsk(TID_IL);
        sta_tsk(TID_IM);
        sta_tsk(TID_CM);

        evt_req = start;

        wup_tsk(TID_CM);

        CMMK10 = 0;
        PMK0   = 0;
        PMK1   = 0;
        PMK2   = 0;

        ext_tsk();
        }
```

7）HANDLER．ASM……割り込みハンドラ

```
$       TITLE   ('KEY INTERRUPT HANDLER')

        NAME    @K_HDLR

        EXTRN   RET_WUP

        EXTRN   _evt_req
        EXTRN   _tid_cm

        PUBLIC  K_H1,K_H2,K_H3

@@CODE  CSEG
```

```
;/*********************************************************************/
;/*                                                                   */
;/* STEPPING MOTOR CONTROL SYSTEM                                     */
;/*                                                                   */
;/*    KEY INTERRUPT HANDLER                                          */
;/*                                                                   */
;/*********************************************************************/
K_H1:
        PUSH    AX

        MOV     A,#1
K_H:
        CMP     _evt_req,#0
        BNZ     $N_DONE
        MOV     _evt_req,A

        POP     AX
        PUSH    RP0,RP1,RP2,RP3,RP4,RP5,RP6,RP7
        MOVW    DE,#_tid_cm
        BR      !RET_WUP

N_DONE:
        POP     AX
        RETI

K_H2:
        PUSH    AX

        MOV     A,#2
        BR      $K_H

K_H3:
        PUSH    AX

        MOV     A,#4
        BR      $K_H

        END
```

付録D. リセット・ルーチン部プログラム・リスト

1) RESET. ASM……システム・リセット・ルーチン

```
$       TITLE   ('SETUP ROUTINE')

        NAME    @RESET

;$SET(P322)     ;set if device:322
$RESET(P322)    ;reset if device:320

        PUBLIC  @BNKRD,@BNKST

        EXTRN   SYS_INF

        EXTRN   ?TMDSP

        EXTRN   BRPROC

        EXTRN   K_H1,K_H2,K_H3


        EXTRN   _?R_INIT,_?R_DATA,_?R_INIS,_?R_DATS
        EXTRN   _?INIT,_?DATA,_?INIS,_?DATS



;/*****************************************************************/
;/*                                                               */
;/* VECTOR TABLE                                                  */
;/*                                                               */
;/*                                                               */
;/*                                                               */
;/*****************************************************************/
@VCTR1  CSEG    AT 0000H
        DW      RES_RTN         ;RESET

@VCTR2  CSEG    AT 0008H
        DW      K_H1,K_H2,K_H3  ;INTP0,1,2

@VCTR3  CSEG    AT 001EH
```

```
        DW      ?TMDSP          ;INTCM10

;/*****************************************************************/
;/*                                                               */
;/* SYSTEM CALL CALLT TABLE                                       */
;/*                                                               */
;/*                                                               */
;/*                                                               */
;/*****************************************************************/
@@CALLTO        CSEG    AT 40H
                DW      BRPROC



@@CODE  CSEG

;/*****************************************************************/
;/*                                                               */
;/* RESET ROUTINE                                                 */
;/*                                                               */
;/*                                                               */
;/*                                                               */
;/*****************************************************************/
RES_RTN:
;
;SFR SET
;
$IF(P322)
        MOV     FCC,#03H        ;FAST FETCH CYCLE MODE
$ENDIF
        MOV     PWC,#0          ;NO WAIT

        SET1    PRDC.0          ;PIN ACC. MODE
        MOV     PM0,#0          ;P0  <- OUTPUT MODE
        MOV     PM3,#1EH        ;P30 <- OUTPUT MODE


        MOV     PRM,#10H        ;TM1 COUNT CLOCK
        MOV     RPUM,#20H       ;INTERVAL TIMER MODE
        MOVW    CM10,#1F3H      ;1ms INTERVAL
        MOV     TMC,#80H        ;TM1 START
```

```
;
;INITIAL DATA SET
;
        MOVW    DE,#_@INIT
        MOVW    HL,#_@R_INIT
        MOVW    UP,#_?R_INIT
LINIT1:
        CMPW    HL,UP
        BE      $LINIT2
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIT1
LINIT2:
        MOVW    DE,#_@DATA
        MOVW    HL,#_@R_DATA
        MOVW    UP,#_?R_DATA
LDATA1:
        CMPW    HL,UP
        BE      $LDATA2
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LDATA1
LDATA2:
        MOVW    DE,#_@INIS
        MOVW    HL,#_@R_INIS
        MOVW    UP,#_?R_INIS
LINIS1:
        CMPW    HL,UP
        BE      $LINIS2
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIS1
LINIS2:
        MOVW    DE,#_@DATS
        MOVW    HL,#_@R_DATS
        MOVW    UP,#_?R_DATS
LDATS1:
        CMPW    HL,UP
        BE      $LDATS2
        MOV     A,[HL+]
```

```
        MOV     [DE+],A
        BR      $LDATS1
LDATS2:


        MOVW    HL,#SYS_INF
        MOVW    AX,[HL]
        BR      AX

;/*****************************************************************/
;/*                                                               */
;/* MEMORY BANK READ/SET FUNCTION                                 */
;/*                                                               */
;/*                                                               */
;/*                                                               */
;/*****************************************************************/
@BNKRD:
@BNKST:
        RET

;/*****************************************************************/
;/*                                                               */
;/* C STATIC SEGMENT TOP DEFINE                                   */
;/*                                                               */
;/*                                                               */
;/*                                                               */
;/*****************************************************************/


@@R_INIT        CSEG
_@R_INIT:
@@R_DATA        CSEG
_@R_DATA:
@@R_INIS        CSEG
_@R_INIS:
@@R_DATS        CSEG
_@R_DATS:
@@INIT          DSEG
_@INIT:
@@DATA          DSEG
_@DATA:
```

```
@@INIS          DSEG    SADDRP
_@INIS:
@@DATS          DSEG    SADDRP
_@DATS:

        END
```

2）ＲＯＭ．ＡＳＭ・・・・・・・・・・・・・・・Ｃスタティック変数の初期値解決用シンボル定義

```
; Copyright (C) NEC Corporation 1989
;
; All rights reserved by NEC Corporation. This program must be used solely
; for the purpose for which it was furnished by NEC Corporation. No Part
; of this program may be reproduced or disclosed to others, in any form,
; without the prior written permission of NEC Corporation.
;ROM BOTTOM AREA
;
        NAME    @rom
;
        PUBLIC  _?R_INIT,_?R_DATA,_?R_INIS,_?R_DATS
        PUBLIC  _?INIT,_?DATA,_?INIS,_?DATS
;
@@R_INIT        CSEG
_?R_INIT:
@@R_DATA        CSEG
_?R_DATA:
@@R_INIS        CSEG
_?R_INIS:
@@R_DATS        CSEG
_?R_DATS:
@@INIT          DSEG
_?INIT:
@@DATA          DSEG
_?DATA:
@@INIS          DSEG    SADDRP
_?INIS:
@@DATS          DSEG    SADDRP
_?DATS:
;
        END
```

保守／廃止

お問い合わせは，最寄りのNECへ

本　　社	〒108-01 東京都港区芝五丁目7番1号 (NEC本社ビル)
コンシューマ半導体販売事業部 ＯＡ半導体販売事業部 インダストリ半導体販売事業部	〒108-01 東京都港区芝五丁目7番1号 (NEC本社ビル) 東京 (03)3454-1111
中部支社半導体販売部	〒460 名古屋市中区栄四丁目14番5号 (松下中日ビル) 名古屋(052)242-2755
関西支社半導体販売部	〒540 大阪市中央区城見一丁目4番24号 (NEC関西ビル) 大　阪 (06)945-3178 大　阪 (06)945-3200 大　阪 (06)945-3208

札幌支社	(011)231-0161	立川支社	(0425)26-0911
仙台支社	(022)261-5511	立川支店	(043)227-9084
盛岡支店	(0196)51-4344	静岡支社	(054)255-2211
山形支店	(0236)23-5511	沼津支店	(0559)63-4455
郡山支店	(0249)23-5511	浜松支店	(053)452-2711
いわき支店	(0246)21-5511	金沢支社	(0762)23-1621
長岡支店	(0258)36-2155	福井支店	(0776)22-1866
水戸支店	(0292)26-1717	富山支店	(0764)31-8461
横浜支社	(045)324-5511	京都支社	(075)344-7824
熊谷支店	(0273)26-1255	神戸支社	(078)332-3311
大宮支店	(0276)46-4011	広島支社	(082)242-5504
宇都宮支社	(0286)21-2281	鳥取支店	(0857)27-5311
小山支店	(0285)24-5011	岡山支店	(086)225-4455
長野支社	(0262)35-1444	高松支社	(0878)36-1200
松本支店	(0263)35-1666	新居浜支店	(0897)32-5001
諏訪支店	(0266)53-5350	松山支店	(0899)45-4111
甲府支社	(0552)24-4141	福岡支社	(092)271-7700
大宮支社	(048)641-1411	北九州支店	(093)541-2887

（技術お問い合わせ先）

半導体応用技術本部 マイクロコンピュータ技術部	〒210 川崎市川崎区駅前本町15番5号 (十五番館)	川　崎	(044)246-3921
半導体応用技術本部 中部応用システム技術部	〒460 名古屋市中区栄四丁目14番5号 (松下中日ビル)	名古屋	(052)242-2762
半導体応用技術本部 西日本応用システム技術部	〒540 大阪市中央区城見一丁目4番24号 (NEC関西ビル)	大　阪	(06)945-3383

半導体応用技術本部
インフォメーションセンター
FAX(044)548-7900
（FAXで対応させていただいております）