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

μPD78F0714によるモータ制御

ホールICによる180度通電方式編

μPD78F0714

資料番号　U18913JJ1V0AN00（第1版）
発行年月　September 2007 NS

© NEC Electronics Corporation 2007

〔メ　モ〕

目次要約

CMOSデバイスの一般的注意事項

① 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。

CMOSデバイスの入力がノイズなどに起因して，V_{IL}（MAX.）からV_{IH}（MIN.）までの領域にとどまるような場合は，誤動作を引き起こす恐れがあります。入力レベルが固定な場合はもちろん，V_{IL}（MAX.）からV_{IH}（MIN.）までの領域を通過する遷移期間中にチャタリングノイズ等が入らないようご使用ください。

② 未使用入力の処理

CMOSデバイスの未使用端子の入力レベルは固定してください。

未使用端子入力については，CMOSデバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してV_{DD}またはGNDに接続することが有効です。

資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

③ 静電気対策

MOSデバイス取り扱いの際は静電気防止を心がけてください。

MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，当社が出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。

また，MOSデバイスを実装したボードについても同様の扱いをしてください。

④ 初期化以前の状態

電源投入時，MOSデバイスの初期状態は不定です。

電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。

リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

⑤ 電源投入切断順序

内部動作および外部インタフェースで異なる電源を使用するデバイスの場合，原則として内部電源を投入した後に外部電源を投入してください。切断の際には，原則として外部電源を切断した後に内部電源を切断してください。逆の電源投入切断順により，内部素子に過電圧が印加され，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。

資料中に「電源投入切断シーケンス」についての記載のある製品については，その内容を守ってください。

⑥ 電源OFF時における入力信号

当該デバイスの電源がOFF状態の時に，入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。

資料中に「電源OFF時における入力信号」についての記載のある製品については，その内容を守ってください。

- 本資料に記載されている内容は2007年9月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品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，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエ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，ソフトウエ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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。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。当社製品をお客様の機器にご使用の際には，当社製品の不具合の結果として，生命，身体および財産に対する損害や社会的損害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を行ってください。
- 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いただく「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ください。

  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，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

  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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E0710J

はじめに

対 象 者　このアプリケーション・ノートは，μPD78F0714の機能を理解し，それらを使用した応用システムを設計するユーザを対象とします。対象製品を次に示します。

- μPD78F0714

目　　的　このアプリケーション・ノートでは，μPD78F0714を利用し，ホールIC付きブラシレスDCモータ（Brush Less DC Motor）を180度通電方式で駆動するシステムについてユーザに理解していただくことを目的としています。

構　　成　このアプリケーション・ノートは大きく分けて次の内容で構成しています。

- 概　　説
- PMSモータ制御の原理
- システム概要
- 制御プログラム

読 み 方　このマニュアルの読者には，電気，論理回路，およびマイクロコントローラに関する一般知識を必要とします。

ハードウエア機能の詳細（特にレジスタ機能とその設定方法など），および電気的特性を知りたいとき

→別冊のμPD78F0714　**ユーザーズ・マニュアル**（U16928J）を参照してください。

命令機能の詳細を理解しようとするとき

→別冊の78K/0**シリーズ　ユーザーズ・マニュアル　命令編**（U12326J）を参照してください。

凡　例　データ表記の重み：左が上位桁，右が下位桁

アクティブ・ロウの表記：$\overline{xxx}$（端子，信号名称に上線）

メモリ・マップのアドレス：上部-上位，下部-下位

注　：本文中に付けた注の説明

注意：気を付けて読んでいただきたい内容

備考：本文の補足説明

数の表記：2進数 … xxxxまたはxxxxB

10進数… xxxx

16進数… xxxxH

2のべき数を示す接頭語（アドレス空間，メモリ容量）：

K（キロ）　…　$2^{10} = 1024$

M（メガ）　…　$2^{20} = 1024^2$

G（ギガ）　…　$2^{30} = 1024^3$

データ・タイプ：ワード　…　32ビット

ハーフワード　…　16ビット

バイト　…　8ビット

開連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

デバイスの関連資料

資　料　名	資料番号	
	和　文	英　文
μPD78F0714　ユーザーズ・マニュアル	U16928J	U16928E
78K/0シリーズ　ユーザーズ・マニュアル　命令編	U12326J	U12326E
μPD78F0714によるインバータ制御　アプリケーション・ノート　ゼロクロス検出による120度通電方式制御編	U17297J	U17297E
μPD78F0714による単相インダクション・モータ制御　アプリケーション・ノート　V/f制御による2相インバータ正弦波駆動編	U17481J	U17481E
μPD78F0714によるモータ制御　アプリケーション・ノート　ホールICによる120度通電方式編	U18774J	U18774E
μPD78F0714によるモータ制御　アプリケーション・ノート　センサレス（BEMF）による120度通電方式編	U18051J	U18051E
μPD78F0714によるモータ制御　アプリケーション・ノート　ホールICによる180度通電方式編	このマニュアル	作成予定
μPD78F0714によるモータ制御　アプリケーション・ノート　センサレス（BEMFのA/D変換）による120度通電方式編	U18912J	作成予定

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

開発ツール（ソフトウエア）の資料（ユーザーズ・マニュアル）

資　料　名		資料番号	
		和　　文	英　　文
RA78K0　Ver.3.80　アセンブラ・パッケージ	操作編	U17199J	U17199E
	言語編	U17198J	U17198E
	構造化アセンブリ言語編	U17197J	U17197E
CC78K0　Ver.3.70　Cコンパイラ	操作編	U17201J	U17201E
	言語編	U17200J	U17200E
ID78K0-QB　Ver.2.94　統合デバッガ	操作編	U18330J	U18330E
PM plus Ver.5.20		U16934J	U16934E

開発ツール（ハードウエア）の資料（ユーザーズ・マニュアル）

資　料　名	資料番号	
	和　　文	英　　文
QB-780714　インサーキット・エミュレータ	U17081J	U17081E
QB-78K0MINI　オンチップ・ディバグ・エミュレータ	U17029J	U17029E
QB-MINI2　プログラミング機能付きオンチップ・デバッグ・エミュレータ	U18371J	U18371E

フラッシュ・メモリ書き込み用の資料

資　料　名	資料番号	
	和　　文	英　　文
PG-FP4　フラッシュ・メモリ・プログラマ　ユーザーズ・マニュアル	U15260J	U15260E

その他の資料

資　料　名	資料番号	
	和　　文	英　　文
SEMICONDUCTOR SELECTION GUIDE　- Products and Packages -	X13769X	
半導体デバイス　実装マニュアル	注	
NEC半導体デバイスの品質水準	C11531J	C11531E
NEC半導体デバイスの信頼性品質管理	C10983J	C10983E
静電気放電（ESD）破壊対策ガイド	C11892J	C11892E
半導体　品質／信頼性ハンドブック	C12769J	－
マイクロコンピュータ関連製品ガイド　社外メーカ編	U11416J	－

注　「半導体デバイス実装マニュアル」のホーム・ページ参照

和文：http://www.necel.com/pkg/ja/jissou/index.html

英文：http://www.necel.com/pkg/en/mount/index.html

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

目　　次

第1章　概　　説

このシステムはホールIC付き永久磁石同期モータ（Permanent Magnet Shynchronous Motor**：以降**PMS**モータ）を**180**度通電方式で駆動します。**

- **このシステム（サンプル・プログラム）は** NEC **エレクトロニクスのモータ・スタータ・キット（**μPD78F0714**）**[注]**を利用し，ホール** IC **付き** PMS **モータを** 180 **度通電方式で駆動します。**
- **制御ゲインは動作環境の特定のモータに合わせて調整しています。モータや制御周期を変更したい場合の動作は保証しておりません。**

注　モータ・スタータ・キット（μPD78F0714**）については，弊社特約店にお問い合わせください。**

1.1　動作環境

このシステムは以下の環境で使用することを前提に作成しています。

- **モータ・スタータ・キット（**μPD78F0714**）ボード一式**
- **低電圧インバータ・セット**

  BLDC**モータ**　PITTMAN(N2311A011) [注]
  - **基準電圧**[V]　：12
  - **無負荷回転速度**[r/min]　：7197
  - **連続トルク**[Nm]　：0.11
  - **最大トルク**[Nm]　：0.23
  - **駆動コイル**　：3 **相（**Y **結線）**
  - **磁極ロータ**　：4 **極（**2 **極対）**
  - **ステータ**　：6 **スロット**
  - **位置センサ**　：**ホール** IC
- PM plus **環境プラットフォーム** V5.20
- CC78K0 **コンパイラ** W3.70
- RA78K0 **アセンブラ** W3.80
- DF0714.78K **デバイス・ファイル** V1.10

注　この動作環境では，PMS**モータの代わりに**BLDC**モータを使用しています。**

1.2　関連マニュアル

開発環境およびボードにつきましては次に示すマニュアルを参照してください。

- **低電圧モータ・スタータ・キット　マニュアル**
- PM plus Ver.5.20 **ユーザーズ・マニュアル**
- CC78K0 Ver.3.70 C **コンパイラ　各ユーザーズ・マニュアル**
- RA78K0 Ver.3.80 **アセンブラ・パッケージ　各ユーザーズ・マニュアル**

第2章　PMSモータ制御の原理

PMSモータは，ステータ（固定子）に巻いたコイルによって回転磁界を発生させ，永久磁石でできたロータ（回転子）との吸引力によって，回転磁界と同じ速度で回転します。

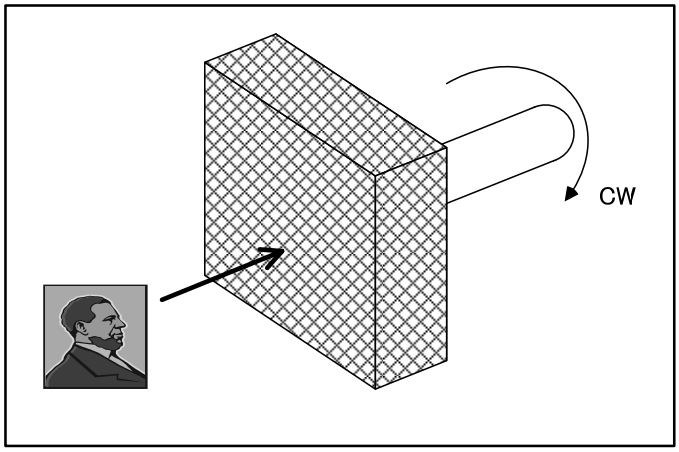
2. 1　回転方向の定義

まず，モータの回転方向についての定義をします。

モータの回転方向は，CW（時計回り）／CCW（反時計回り）があります。

モータが回す対象物の回転方向を基準にしてCW／CCWが決定します。モータの軸がある面を対象物側に向けたときの回転方向を基準にします。したがって，下図のようになります。

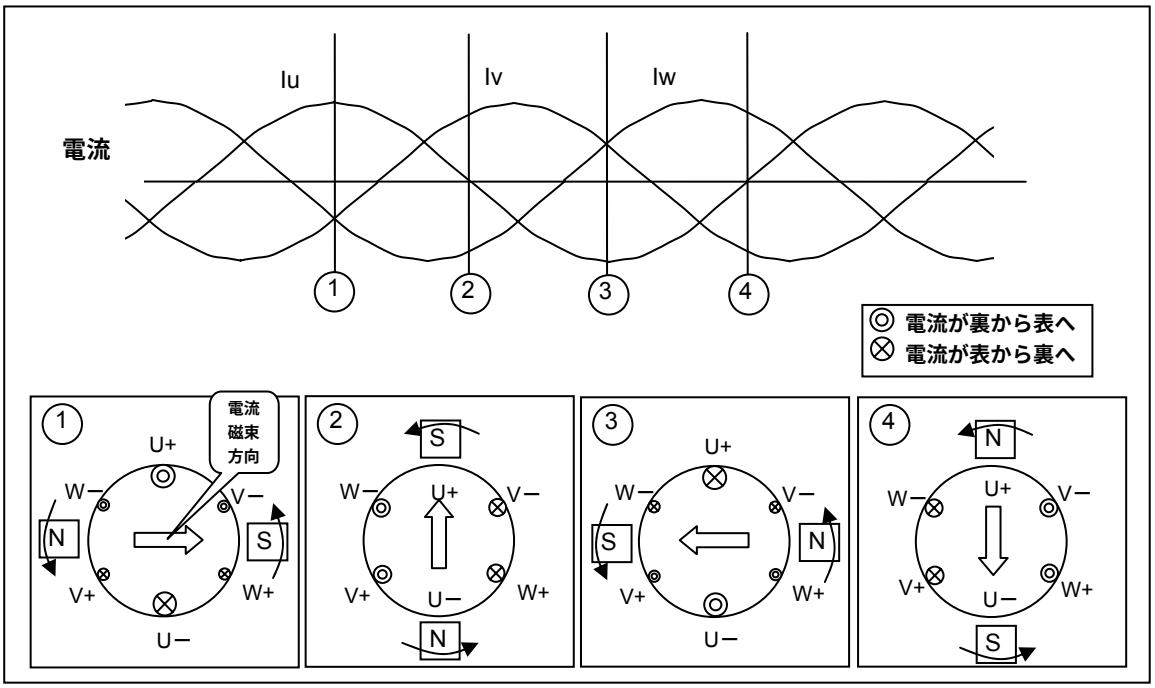
図 2－1　モータの回転方向



2. 2　回転磁界

PMSモータではコイルに交流電流を流すことによって回転磁界を発生させます。次の図に三相巻線による回転磁界の発生原理を記述しました。三相巻線（u,v,w相）は120度間隔で配置し，120度ずつ位相差のある三相巻線電流（Iu,Iv,Iw）を流します。図中の◎の大きさが電流の大きさに比例しています。

図 2−2　三相交流電流と回転磁界の発生原理

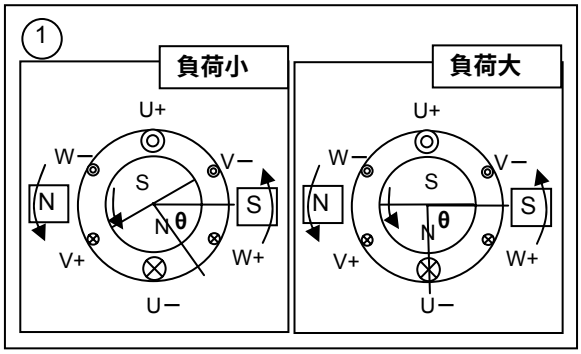


電流磁束方向を*I* ，永久磁石の磁束をϕ，電流磁束と永久磁石の磁束の傾きをθとするとモータのトルクは次の式になります。

$$T = I\phi\sin\theta \quad (1)$$

式（1）からわかるように，無負荷時θは0 °，負荷が最大発生トルクを越えると脱調します。
（θ = 90 ° の時が最大トルク）

図 2−3　回転時のイメージ



また電流の周波数を f，極対数をPとすると速度m（rpm）は次の式になります。

$$m = \frac{60f}{P} \quad (2)$$

回転子の位置を検出しない（オープン・ループ駆動）制御は，トルクを電流（電圧），速度を電流の周波数と別々に制御する必要があります。負荷の変化で式（1）のθも変化することにより，速度のムラ，脱調の発生，そして軽負荷での高電圧では大電流が流れるため，安定した制御が困難です。

そこで，回転子の位置を検出し常にθ=90 °に保つことにより発生トルクが常に最大となるため高効率運転が可能になります。また，電流と磁束の位相を固定したことで，速度とトルクの特性が直流モータと同じ特性となり，印加電圧をV，コイル抵抗をR，誘起電圧をE，電流をI，回転子の回転速度をNとすると，EはNに，TはIに比例するため，その比例定数をそれぞれ，Ke，Ktとすると以下の式が成立します。

$$V = RI + E \qquad (3)$$

$$E = KeN \qquad (4)$$

$$T = KtI \qquad (5)$$

上記より式（6）が求まります。

$$T = \left(Kt \diagup R\right)\left(V - KeN\right) \quad (6)$$

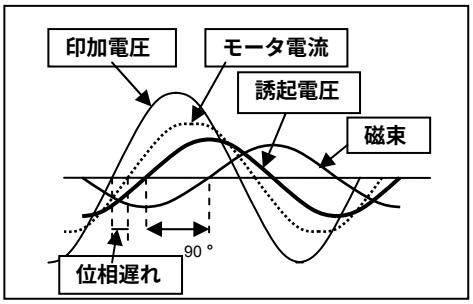
式（6）で V が一定の場合，負荷（T）の増加が回転速度（N）の減少で成り立つことがわかります（脱調しない）。したがって，速度を検出し，印加電圧 V で速度を調整するだけで負荷に最適な電流による高効率運転が可能です。

2. 3　180度通電方式

巻線に180度の正弦波電圧を加えて駆動する方法を180度通電方式といいます。

次の図にU相の瞬時電流波形を示します。

図 2－4　瞬時電圧と電流波形



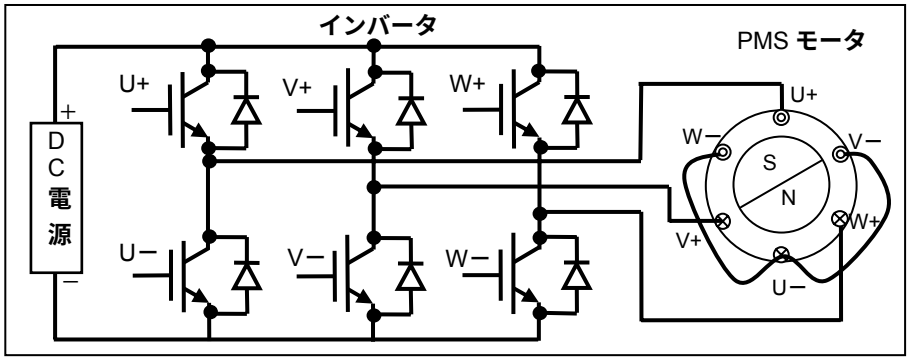
モータに印加される電圧と誘起電圧の差電圧がモータのコイルに加わり，モータ電流が流れます。コイルのインダクタンスによりモータ電流の位相は印加電圧から遅れます。モータのトルクは磁束と電流の位相差が90度の時最大となるので，印加電圧の位相で調整します。

2. 4　インバータ

PMSモータは可変電圧可変周波数制御（VVVFインバータ制御，英語ではVariable Frequency Drive）で直流電源から三相交流を発生させ，その実行電圧と周波数を負荷に応じて可変制御します。

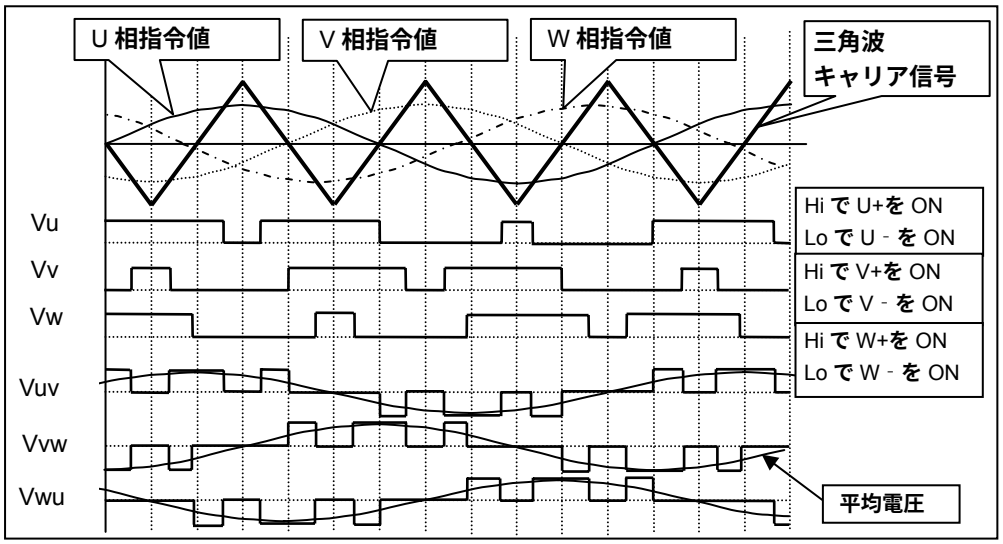
次にPMSモータと三相インバータとの結線図を示します。

図 2−5　三相インバータと PMS モータ



次の図に前記回路を三角波比較PWM法（相変調方式）で作成したタイミングでスイッチング素子をチョッパ動作させることでコイルに三相交流電流を流す例を示します。

図 2−6　三相インバータの出力波形（三角波比較 PWM 法）



動作インバータ出力周波数と同じ指令値（正弦波信号）と，三角波キャリア信号の大小によって，各相のスイッチをON/OFFすることで，相間電圧（Vuv,Vvw,Vwu）が得られます（相間電圧の差で電流の向きと強さが変化）。

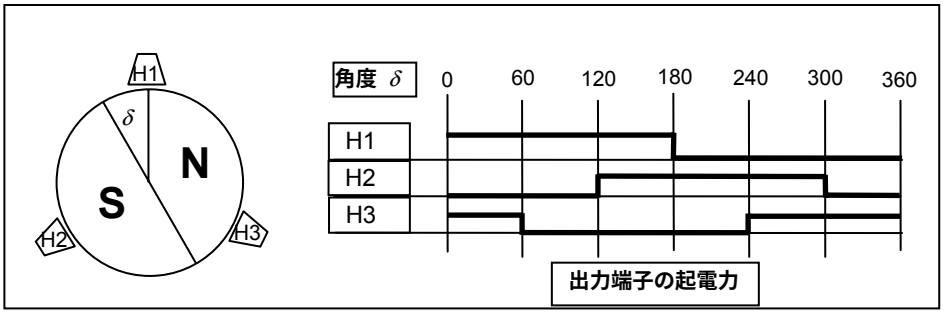
2. 5　位置検出

180度通電方式では電流磁束と永久磁石の磁束の角度を90度に維持するために細かい位置情報が必要です。

このシステムでは120度間隔に配置されたホールIC（6パルス／サイクル）の変化から60度ごとの位置を検出し，60度より細かな位置は回転速度から類推します。

次の図にホールICを利用した磁極位置センサの出力を示します。

図 2−7　ホール IC 出力



各相のホールIC出力が180度ごとに変化するため，三相で60度ごとに位置検出が可能です。

回転部の磁極が4極の場合，1つのホールICは90度ごとに値が変化し，通電パターンの切り替えは30度ごとに行います。

通常，1サイクル（6つの通電パターン）を電気角の360度，モータ軸の1回転を機械角の360度と定義します（このマニュアルではすべて電気角で記述しています）。

機械角：電気角／極対数

極対数：極数／2

2. 6　起動方法

ホールICによる位置情報はホールIC出力の変化と回転速度から回転子の位置を類推することができます。回転子が停止している場合，回転子の位置は最大で60度の誤差があるため，詳細な位置情報を必要としない方法で起動する必要があります。

次に代表的な起動方式を示します。

2.6.1 同期始動方式

コイルに通電して回転子を強制的に位置決めしたのち，180度の正弦波電流を流して同期始動を行い，ホールIC出力の変化と速度で回転子の位置が類推できるところから，180度正弦波駆動に切り替える方法。

低分解能な位置情報の場合，停止状態からの180度通電駆動は印加電圧や電流周波数の調整が難しく回転が不安定になる欠点があります。

2.6.2 120度通電方式

ホールIC出力から120度通電方式の通電パターンで電流を流し，ホールICの変化から速度と回転子の位置が類推できるところから，180度正弦波駆動に切り替える方法。

120度通電方式には，トルクに脈動が発生する欠点がありますが，起動時のみで大きな問題ではないため，主にこの方法が使用されます。

2. 7　速度検出

ホールICの値が変化する時間を計測することで，モータの回転速度を求めます。

2. 8　電圧制御

モータのコイルに加える電圧は，スイッチング素子の導通期間を高い周波数でチョッパ動作させて通流率（平均電圧）を調整するPWM（Pulse Width Modulation）で制御します。

2. 9　速度制御

式（6）からコイルに加える印加電圧で速度を制御します。具体的には，ホールIC出力と回転速度から回転子の位置を推定して，三角波比較PWM法に与える各相の指令値（三角波キャリア信号との比較値）を計算して設定します。

指令値はPID制御で調整を行います。

2.9.1 PID制御

指定された速度と検出した回転速度の偏差を元に計算を行い，指令値を変更します。

PID制御は，偏差に比例する出力を出す比例動作（Proportional action:P動作）と，偏差の積分に比例する出力を出す積分動作（Integral action:I動作）と，偏差の微分に比例する出力を出す微分動作（Derivative action:D動作）とからなります。

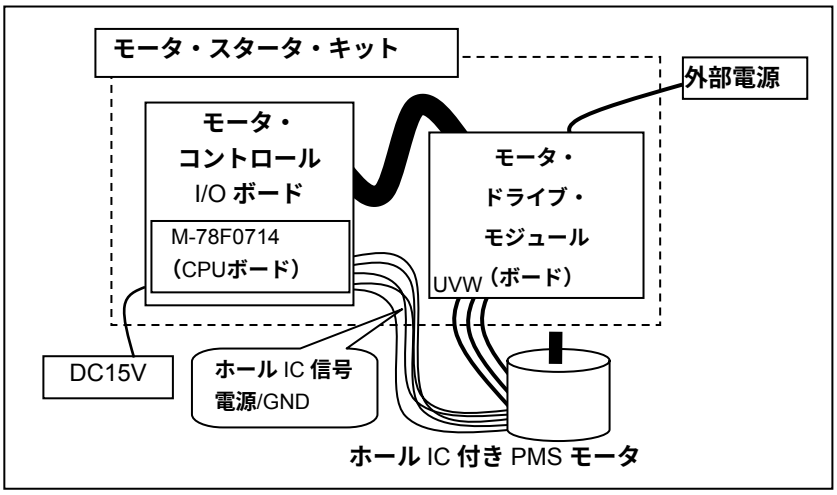
第3章　システム概要

このシステムの概要について記述します。

3. 1　構　　成

このシステムの構成を次の図に示します。

図 3－1　システム構成

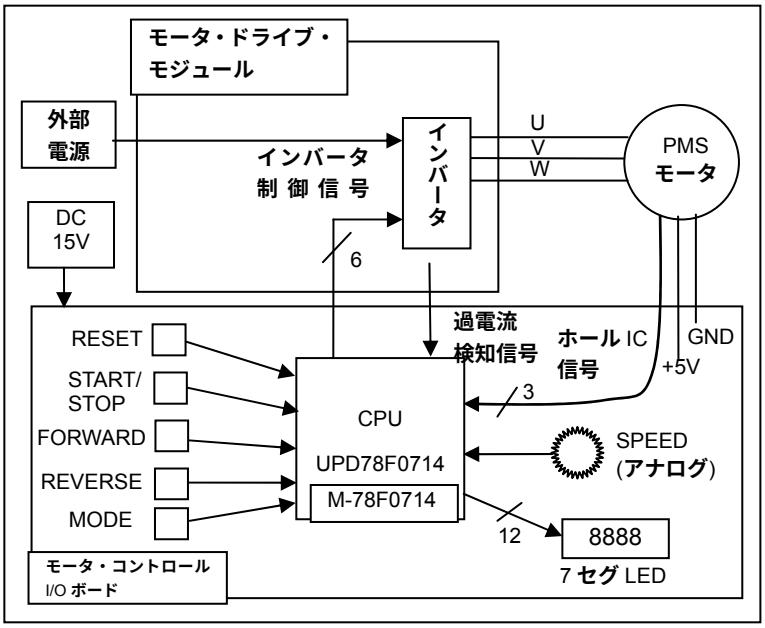


モータを駆動するモータ・ドライブ・モジュール，モータ制御用スイッチを搭載したモータ・コントロールI/Oボード，CPUを搭載したM-78F0714で構成されます。

PMSモータは3相4極（2極対）のホールIC付きモータです。

モータ・スタータ・キットのブロック構成を次の図に示します。

図 3−2　ブロック構成



モータ・コントロールI/Oボード上のスイッチでモータの制御を行います。

3. 2　インタフェース

表 3−1にユーザ・インタフェース一覧を示します。

表 3−1　ユーザ・インタフェース

機能名	部品番号	機　　能
RESET	SW1	リセット
START/STOP	SW2	スタート／ストップ
FORWARD	SW3	回転方向（右回転，時計回り，CW）
REVERSE	SW4	回転方向（左回転，反時計回り，CCW）
MODE	SW5	速度指定切り替え 速度表示切り替え
SPEED	R52	指定速度変更
8セグLED	DISP1 DISP2 DISP3 DISP4	速度の表示（rpm）[注]

注　モータ停止中は常に指定速度を表示します。
指定速度固定時には右下に“ . ”を表示します。
モータ回転中は回転速度を表示します。
モータ回転中は MODE スイッチを押している間，指定速度を表示します。
RESET を押している間はリセット継続，離すとリセット解除です。
リセット解除直後“ SELF ”と 2 秒表示します。

表 3－2にエラー一覧を示します。

表 3－2　エラー一覧

エラー	LEDの表示	状　況
H/W 過電流	0.C.	モータの電流が異常
S/W過電流	S.O.C	モータの電流が異常
ホールIC	HALL	ホールICの値が異常
装置異常	FAIL	モータが回転していない

表 3－3にμPD78F0714端子のインタフェース一覧を示します。

表 3－3　端子のインタフェース

端子番号	端子名	機　能
8	$\overline{\text{RESET}}$	RESET（SW1）[注]
27-32	TW0TO0/RTP10- TW0TO5/RTP15	三相PWMインバータ切り替え
11	P01/INTP1	ホールIC信号（HALL1）
10	P02/INTP2	ホールIC信号（HALL2）
9	P03/INTP3/ADTRG	ホールIC信号（HALL3）
49-52	P64-P67	7セグLED選択（LD_LED0-LD_LED3）
56	P73	START/STOP（SW2）
55	P72	FORWARD（SW3）
54	P71	REVERSE（SW4）
53	P70	MODE（SW5）
41-48	P40/RTP00-P47/RTP07	8セグLEDへの出力データ
12	TW0TOFFP/INTP0/P00	過電流検知（+5V→0V）
60	P24/ANI4	速度変更（R52）
59	P25/ANI5	ISHUNT電流
20	P53/TI000/INTP5	タイマ・キャプチャ・トリガ
21	P54/TI001/TO00	モータ・ドライブ・モジュール制御

注意　2JP7 の 1－2 をショートします。

3.3 機　　能

表 3−4にこのシステムの機能と動作概要を示します。

表 3−4　システムの機能と動作概要　(1/2)

機　　能	概　　要
起動（電源供給）	・LEDに“SELF”を2秒表示する ・SPEEDボリュームの指定速度（rpm）をLEDに表示する
RESETスイッチ	・モータの制御状態に関係なく，システムの再起動を行う
START/STOP スイッチ	モータ制御停止中：モータ制御を開始する ・LEDに“0”を表示する ・モータはCWで回転を開始する ・モータの回転速度（rpm）をLEDに表示する
	モータ制御中：モータ制御を停止する ・モータの回転速度（rpm）が0になるまでLEDに表示する ・SPEEDボリュームの指定速度（rpm）をLEDに表示する
	押し続けてもSTARTとSTOPはトグルしない
FORWARDスイッチ	モータ制御停止中：機能しない ・変化なし
	モータ制御中：回転方向を変える ・回転方向がCCWの場合，一度停止してからCWになる ・回転方向がCWの場合，変化しない
REVERSEスイッチ	モータ制御停止中：機能しない ・変化なし
	モータ制御中：回転方向を変える ・回転方向がCCWの場合，変化しない ・回転方向がCWの場合，一度停止してからCCWになる
MODEスイッチ	モータ制御停止中：速度指定を切り替える ・SPEEDボリュームによる指定速度変更を無効／有効にする ・無効時はLEDの数値右下に“．”が表示される
	モータ制御中：LEDの表示を変える ・押している間，SPEEDボリュームの指定速度（rpm）をLEDに表示する
SPEEDボリューム	モータ制御停止中：指定速度の変更 ・LEDの表示速度（rpm）を表示する ・MODEスイッチで無効に設定されている場合は変化しない
	モータ制御中：指定速度の変更 ・モータの回転速度（平均）が指定した速度になる
H/W過電流	・モータ・ドライブ・モジュールの許容電流超過で発生する ・モータ制御を停止する ・LEDに“0.C.”を表示する ・RESETスイッチ以外の機能を無効にする

注意　複数のスイッチが押された場合の動作は保証していません。

表 3−4　システムの機能と動作概要　(2/2)

機　　能	概　　要
S/W過電流	モータの許容電流超過で発生する モータ制御を停止する LEDに“S. 0.C.”を表示する RESETスイッチ以外の機能を無効にする
無回転	・STARTから約1.25秒モータが回転しない場合，約0.5秒以上モータが停止した場合に発生する ・モータ制御を停止する ・LEDに“FAIL”を表示する ・RESETスイッチ以外の機能を無効にする
ホールIC異常	・ホールICからの値が不正の場合発生する ・モータ制御を停止する ・LEDに“HALL”を表示する ・RESETスイッチ以外の機能を無効にする

注意　複数のスイッチが押された場合の動作は保証していません。

3.4　周辺I/O

このシステムでは次のような周辺I/Oを使用しています。

表 3－5　使用周辺 I/O 一覧

機　　能	周辺I/O機能名（μPD78F0714）
インバータ・タイマ	・PWM出力用 ・10ビット・インバータ制御用タイマ（TW0UDCなど） ・キャリア（変調）周波数は10kHz対称三角波 ・キャリア（変調）同期割り込みが200μs間隔で発生する（キャリアによる割り込み頻度を2回に1回で設定） （メインのPID制御で三相の印加電圧を計算，キャリア同期割り込みで指令値を計算して更新する）
リアルタイム出力	・通電パターン切り替え用 ・リアルタイム出力ポート（RTBH01,RTBL01など） ・16ビット・タイマ・キャプチャ／コンペア・レジスタ01（CR01） （起動時，120度通電方式の通電パターン出力に使用する）
ISHUNT電流値 取得用タイマ	・タイミング調整用 ・８ビット・タイマ／イベントカウンタ51（TM51,CR51） ・1.43 ms間隔で割り込みが発生し，ISHUNT電流測定関数を実施する。
ウエイト処理	・タイミング調整用 ・８ビット・タイマ／イベントカウンタ50（TM50,CR50） ・1ms間隔で割り込み要求フラグが設定される
指定速度読み出し	・可変抵抗値の電圧を指定速度に変換 ・A/Dコンバータ（ANI4）
ISHUNT電流値取得	・モータ動作電流を電圧変換している端子（ANI５）から電圧値を取得
キャプチャ割り込み	・速度計測用 ・割り込み機能（TI000/INTP5）
H/W過電流割り込み	・割り込み機能（INTP0） ・モータ・ドライブ・モジュールの過電流発生（LOWアクティブ）
フェイルセーフ	・ウォッチドッグ・タイマ

3.5　割り込み

このシステムで使用する割り込みの一覧を表 3−6に示します。

表 3−6　使用割り込み一覧

名　　称	機　　能	発生条件
INTP0	過電流発生の検知	外部端子
INTP5	HALL1の変化検知（速度計測）	外部端子
INTTW0UD	キャリア同期割り込み発生	インバータ・タイマ・カウンタのアンダフロー
INTTM51	ISHUNT電流の取得	８ビット・タイマ・カウンタのオーバフロー
RESET	リセット発生	$\overline{\text{RESET}}$端子
WDT	内部リセット発生	プログラム暴走によるウォッチドッグ・タイマ・オーバフロー

備考　INTTM50, INTAD は割り込み要求フラグのポーリングで処理

第4章　制御プログラム

このシステムでは基本的な制御プログラムを用いて実際にモータの速度制御を行います。

4. 1　コンパイラ・オプション

この制御プログラムで制御可能なインバータ２種類をコンパイル時のオプションで切り替えることが可能となっている[注]。

表 4−1　コンパイラ・オプション

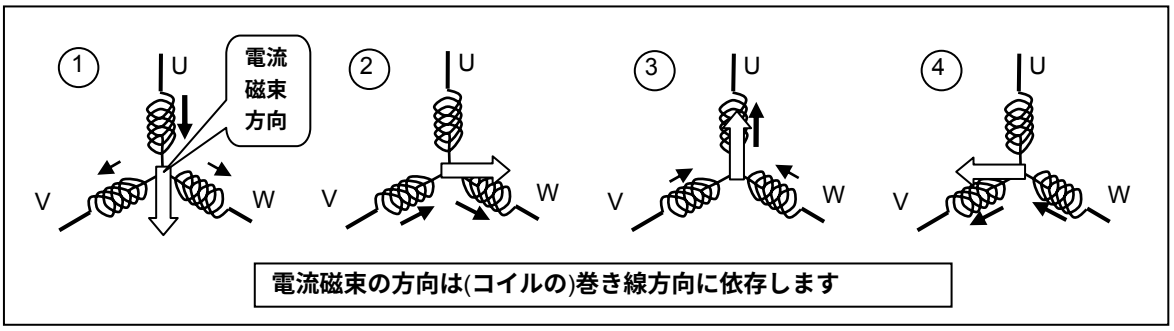
オプション名	機　　能
LOW	PITTMANモータ使用時
なし[注]	その他のモータ使用時

注　この制御プログラムでは，PITTMANモータを使用するので，必ずLOWを設定してください。

4.2　回転磁界

このシステムではBLDCモータ（Brush Less DC Motor）をPMSモータの代わりに使用するため，「図 2−2　三相交流電流と回転磁界の発生原理」は以下のようになります。

図 4−1　集中巻きでの回転磁界

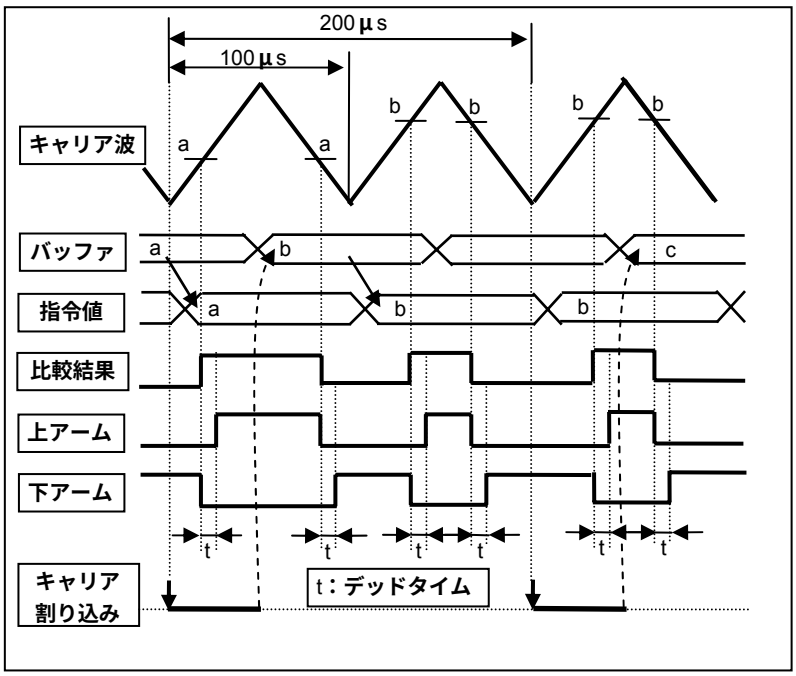


4.3　インバータ

インバータ・タイマの機能を使って，インバータの切り替えを行います。

キャリア同期割り込みはカウンタのアンダフローで毎回発生し，次のインバータ・タイマ出力パターンを設定します。

図 4−2　キャリア同期割り込み発生タイミング

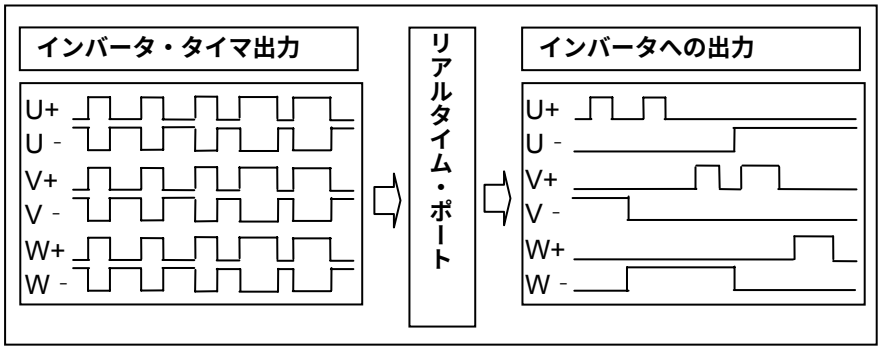


180度通電方式ではリアルタイム・ポートの機能を停止し，インバータ・タイマ出力がインバータへの出力となります。インバータ・タイマ出力の波形はPID制御の演算結果をキャリア同期割り込み（200μsごと）で，ホールICの値とキャリア同期割り込み回数から求めた位置情報を使った演算結果で変更されます。

キャリア波を5kHzにしてキャリア同期割り込みを毎回(200μs)とした場合，使用したモータでは高周波音が発生したため，10kHzのキャリア波2回に1回割り込みを発生させて処理を行いました。

起動時の120度通電方式は上アーム全域非相補動作のPWM制御です。

図 4−3　120 度通電の PWM 出力イメージ



インバータ・タイマ出力の波形はPID制御の演算結果で変更されます。

リアルタイム・ポートでのマスク処理（通電パターン）はキャリア同期割り込み（200μs）ごとにホールICの値で決まります。

4.4　180度通電方式

三角波比較PWM法で使用する指令値を以下の式で作成します。

$$Vu = E_d/2 \cdot M \cdot \cos(\delta + \alpha)$$
$$Vv = E_d/2 \cdot M \cdot \cos(\delta + \alpha - 2\pi/3)$$
$$Vw = -(Vu + Vv)$$

Vu, Vv, Vw : 三相の指令値
E_d : 直流電源電圧、 δ : 回転角、 α : 進角、M : 電圧指令値 $0 < M \leqq 1$

電圧の進角はモータを電圧，電流モデル式であらわすことで求めることができます。
永久磁石同期モータの定常時の電圧方程式およびトルク式は次のようになります。

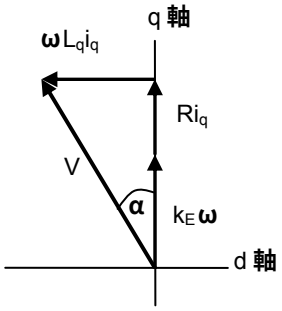
$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R & -\omega L_q \\ \omega L_d & R \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ k_E \omega \end{bmatrix}$$

$$T = P_n k_E i_q + P_n (L_d - L_q) i_d i_q$$

id, iq : 電気子電流のd, q軸成分、v_d, v_q : 電気子電圧のd, q軸成分、
k_E : 誘導起電圧定数、L_d, L_q : d, q軸インダクタンス、ω : 速度、P_n : 極対数

d軸電流が0になるように制御されている場合の電圧ベクトルの軌跡は以下の図になります。

図 4−4　ベクトル図



トルク式から負荷の増加で電流が増加することがわかり，ベクトル図から速度，電流の増加で電圧の進角αが増加することがわかります。

今回のシステム(CPU)ではこれらの計算を随時行う能力が不足しているため，テーブル化などにより条件に対応する値を進角として設定し処理を行っています（プログラム内部にて進角を変更可能）。

4.5　位置検出

ホールICの値から60度ごとの角度を検出し，モータの回転速度とホールICの値変化からの経過時間（200μs間隔のキャリア同期割り込み発生回数）で，さらに詳細な角度を類推します。

120度通電方式の通電パターンとホールICの値からホールICの値と回転子の磁極位置の関係は以下のようになります。

図 4−5　120 度通電方式の通電パターン

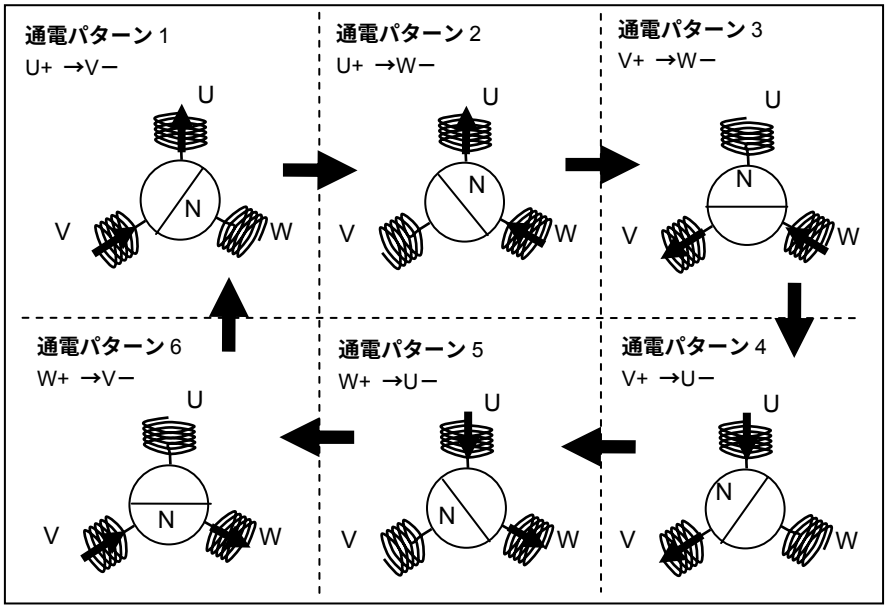


表 4−2　通電パターンとホール IC の関係（モータのスペック表より）

ホールIC ＼ 通電パターン	1	2	3	4	5	6
S1(HALL1)	L	L	H	H	H	L
S2(HALL2)	H	L	L	L	H	H
S3(HALL3)	H	H	H	L	L	L

ホールIC出力の変化と角度の関係を以下に記述します。

BLDCモータのスペックから端子情報と通電パターンとホールICの値を記述します。

4.5.1 低電圧インバータ・セット

表 4−3　BLDC モータの端子スペック

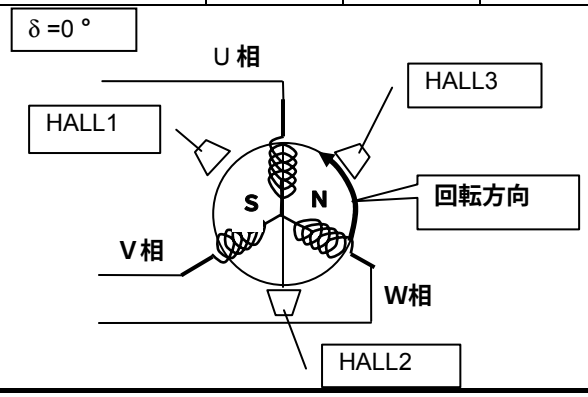
カラー	機　　能	備　　考
BROWN	MOTOR ϕA	U相
RED	MOTOR ϕB	V相
ORANGE	MOTOR ϕC	W相
GREY	SENSOR ϕ1	HALL1
BLUE	SENSOR ϕ2	HALL2
WHITE	SENSOR ϕ3	HALL3
VIOLET	SENSOR Vcc	
BLACK	SENSOR GND	

表 4−4　120 度通電モード時の通電パターンとホール IC の値

通電（正回転）	+A −B	+A −C	+B −C	−B −A	+C −A	+C −B
SENSOR ϕ1	L	L	H	H	H	L
SENSOR ϕ2	H	L	L	L	H	H
SENSOR ϕ3	H	H	H	L	L	L
通電（逆回転）	+B −A	+C −A	+C −B	+A −B	+A −C	+B −C

表 4−5　180 度通電モード時の角度

角度(δ) ホールIC	0	60	120	180	240	300
HALL1	L	L→H	H	H	H→L	L
HALL2	H→L	L	L	L→H	H	H
HALL3	H	H	H→L	L	L	L→H



指令値の計算式

$$Vu = E_d/2 \cdot M \cdot \cos(\theta)$$

$$Vv = E_d/2 \cdot M \cdot \cos(\theta - 2\pi/3)$$

$$Vw = -(Vu + Vv)$$

$$\theta = \delta + \text{電圧進角}$$

4.6　起動方法

このシステムでは120度通電方式で起動します。120度通電方式についての詳細は「モータ制御システム仕様設計報告書　ホールICによる120度通電方式編」を参照してください。

4.7　速度検出

タイマ・キャプチャ機能を用いてホールICの変化時点でのタイマのカウント値を瞬時に保存することで，通常の処理でのカウント値の保存の際に発生する遅延の影響が無い速度計算により精度の高い速度検出を行います。

このシステムのBLDCモータ(3相4極)の場合，1つのホールICは１回転で4回変化しますが，使用するタイマ機能の仕様により立ち上がり側の変化のみ処理を行うため，1/2回転に変化したタイマ値から速度を計算します。

$$N = \frac{60}{s \times n \times 2} = \frac{2343750}{n}$$

N：１分間の回転数(rpm)
s：タイマの分解能($12.8\mu s$)
n：発生回数
2: 変化回数

このシステムでは
PITTMANモータ：　200 rpm～7,200 rpm
の回転速度範囲をサポートしています。

4.8　電圧制御

三角波比較PWM法で使用する電圧指令値で三相の相間電圧を制御します。

4.9　速度制御

三角波比較PWM法の電圧指令値を変更することで速度制御を行います。電圧指令値の調整はPID制御で行います。

4.9.1 PID制御

回転速度と指定速度の差をフィードバックして直流電源電圧指令値にPID制御を行うことで速度調整を行っています。電圧指令値の操作量は，サンプリング方式（離散値）に適した以下の速度形PIDアルゴリズムを使用しています。

$$MV_n = MV_{n-1} + \triangle MV_n$$
$$\triangle MV_n = K_p(e_n - e_{n-1}) + K_i \times e_n + K_d((e_n - e_{n-1}) - (e_{n-1} - e_{n-2}))$$

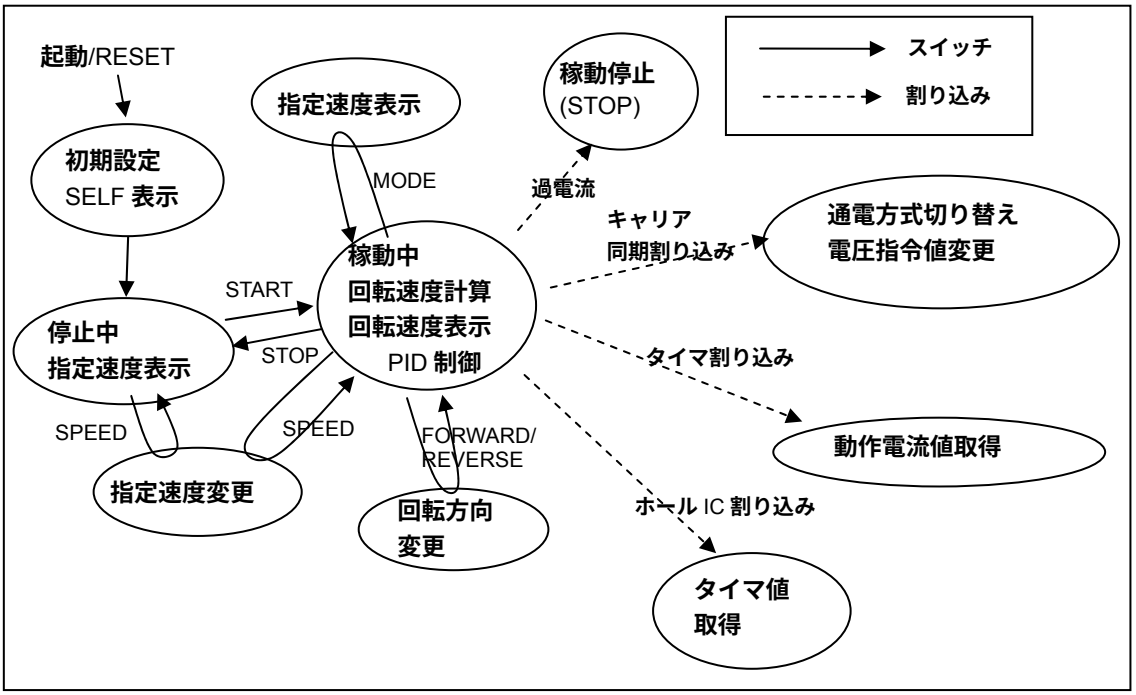
MV_n: 今回操作量
MV_{n-1}: 前回操作量
$\triangle MV_n$: 今回操作量差分
e_n: 今回の偏差（指定速度と実速度の差分）
e_{n-1}: 前回の偏差
e_{n-2}: 前々回の偏差
K_p：フィードバックゲイン（比例要素）
K_i：フィードバックゲイン（積分要素）
K_d：フィードバックゲイン（微分要素）

フィードバック・ゲインの最適値はモータ特性や負荷の有無で変化します。

4.10　モジュール構成

システムの状態遷移図を以下に記述しました。

図 4−6　システムの状態遷移



キャリア同期割り込みは200μsごとに発生します。

ホールICによる割り込みは1回転に2回発生します。

4.11　制御プログラムの関数一覧

制御プログラムは複数の関数で構成されています。以下に各関数の機能概要を示します。より詳細な処理についてはフロー・チャートを参照してください。

4.11.1 ユーザ使用可能関数

表 4−6　ユーザ使用可能関数一覧

関数名	機　　能	目　　的
motor_init()	初期設定	各機能のレジスタ設定，割り込み設定などモータ制御に必要な初期化を行う。
motor_start()	起動指示	モータの回転の開始を指示する。
motor_stop()	停止指示	モータの回転の停止を指示する。
motor_rotation()	回転方向指示	モータの回転方向を指示する。
motor_pid()	PID演算	PID演算を指示する。
motor_pset()	パラメータ設定	モータ制御に必要なパラメータを設定する。

4.11.2 モータ・ライブラリ内部関数

表 4−7　モータ・ライブラリ内部関数一覧

関数名	機　　能	目　　的
system_restart()	再起動処理	反転停止後の再起動を行う。
system_stop()	停止処理	システムの停止を行う。
init_PORT()	ポート設定処理	モータ制御で使用するポートの設定を行う。
init_OSC()	クロック設定処理	CPU動作速度などのクロック設定を行う。
init_TW0()	インバータ設定処理	インバータ・タイマの設定を行う。
start_TW0()	インバータ起動処理	インバータ・タイマの起動を行う。
stop_TW0()	インバータ停止処理	インバータ・タイマの停止を行う。
set_TW0()	PWM設定処理	PWMの設定を行う。
init_TM00()	TM00設定処理	16ビット・タイマの設定を行う。
start_TM00()	TM00起動処理	16ビット・タイマの起動を行う。
stop_TM00()	TM00停止処理	16ビット・タイマの停止を行う。
init_RTPM01()	リアルタイム出力ポート設定処理	リアルタイム出力ポートの設定を行う。
start_RTPM01()	リアルタイム出力ポート起動処理	リアルタイム出力ポートの起動を行う。
stop_RTPM01()	リアルタイム出力ポート出力停止処理	リアルタイム出力ポートの停止を行う。
end_RTPM01()	リアルタイム出力ポート使用停止処理	リアルタイム出力ポートからPWM出力へ移行する。
set_RTPM01()	通電パターン切り替え処理	リアルタイム出力ポートの出力値を設定し，出力切り替え処理を行う。
init_AD()	A/D設定処理	A/Dの設定を行う。
start_AD()	A/D起動処理	A/Dの起動を行う。
init_WDTM()	ウォッチドッグ・タイマ設定処理	ウォッチドッグ・タイマの設定を行う。
init_TM51()	TM51設定処理	8ビット・タイマの設定を行う。
start_TM51()	TM51起動処理	8ビット・タイマの起動を行う。
read_Hall_IC()	ホールICの情報を取得	ホールICの情報をポートから取得する。
INTP0_on()	INTP0許可処理	過電流割り込みの許可を行う。
INTTW0UD_on()	INTTW0UD許可処理	キャリア同期割り込みの許可を行う。
INTTW0UD_off()	INTTW0UD禁止処理	キャリア同期割り込みの禁止を行う。
INTP5_on()	INTP5許可処理	速度情報取得用割り込みの許可を行う。
INTP5_off()	INTP5禁止処理	速度情報取得用割り込みの禁止を行う。
INTTM51_on()	INTTM51許可処理	電流マイナー・ループ用タイマ割り込みの許可を行う。
int_speed()	速度情報取得用割り込み処理	速度情報の更新を促すフラグの設定を行う。
int_fault()	過電流割り込み処理	モータ処理の停止を行う。
int_carrier()	キャリア割り込み(谷)処理	通電パターン切り替え，デューティ比の更新，モータ動作状況診断を行う。
int_TM51()	電流取得用割り込み処理	電流値の取得を行う。
get_coswt()	COS演算処理	COS演算を行う。
set_pwm()	PWM値取得処理	取得したホールICでの最適なPWM値の取得演算を行う。

4.11.3 参考プログラム用関数

表 4－8　参考プログラム関数一覧

関数名	機　　能	目　　的
print_error()	エラー表示	LEDへエラー状態を表示する。
get_sw()	スイッチ読み込み指示	MC-IOボード上のスイッチ情報の読み込みを指示し，読み込んだスイッチに対応した動作情報を返す。
speed_print()	速度表示	LEDへ速度を表示する。
led_print()	表示用データ生成	LEDへデータを生成し渡す。
led_set()	LED表示	LEDへデータを表示する。
wait()	待機	指定時間待機する。
startup_disp()	起動時のLED表示	起動時のLED表示
vol2speed()	速度指示	速度を取得する。
get_vol()	ボリューム読み込み指示	MC-IOボード上のボリューム情報の読み込みを指示する。
init_PORT()	ポートの設定	MC-IOボード上のポート設定を行う。
init_TM50()	TM50設定処理	8ビット・タイマの設定を行う。
start_TM50()	TM50起動処理	8ビット・タイマの起動を行う。
wait_TM50()	TM50待機処理	指定時間の待機を行う。
clear_WDTM()	WDTのクリア処理	ウォッチドッグ・タイマのカウンタをクリアする。

4.12　フロー・チャート

各関数のフロー・チャートを記述しました。

- モータ・ライブラリ

図 4－7　モータ初期設定処理（motor_init 関数）

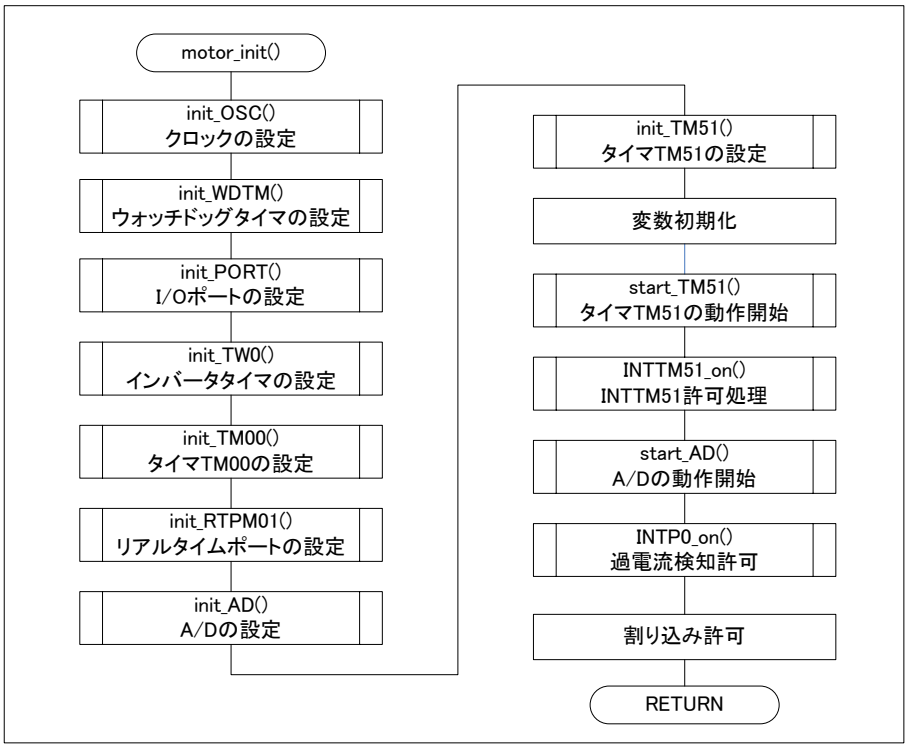


図 4－8　モータ始動処理（motor_start 関数）

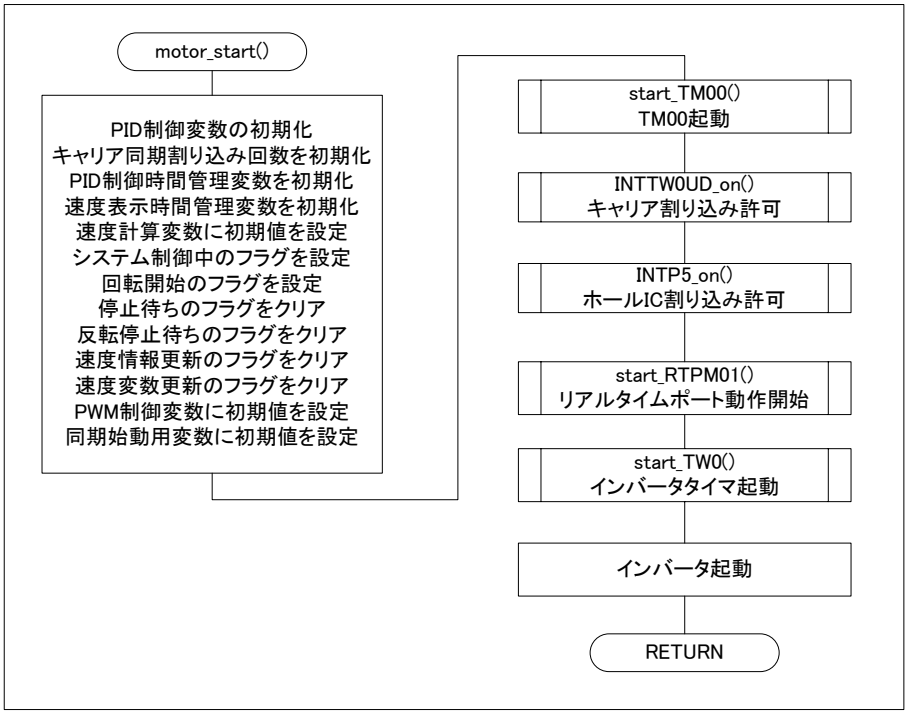


図 4－9　モータ停止処理（motor_stop 関数）

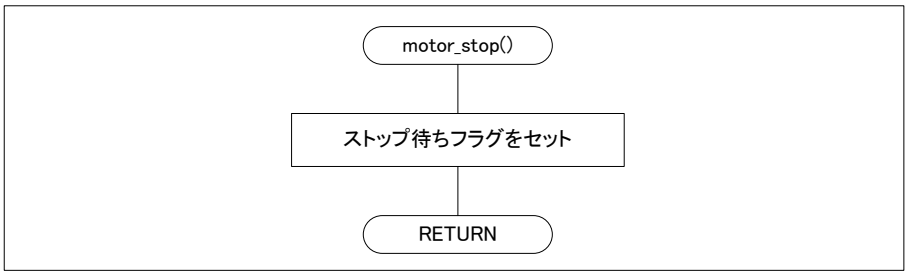


図 4－10　モータ回転方向変更処理（motor_rotation 関数）

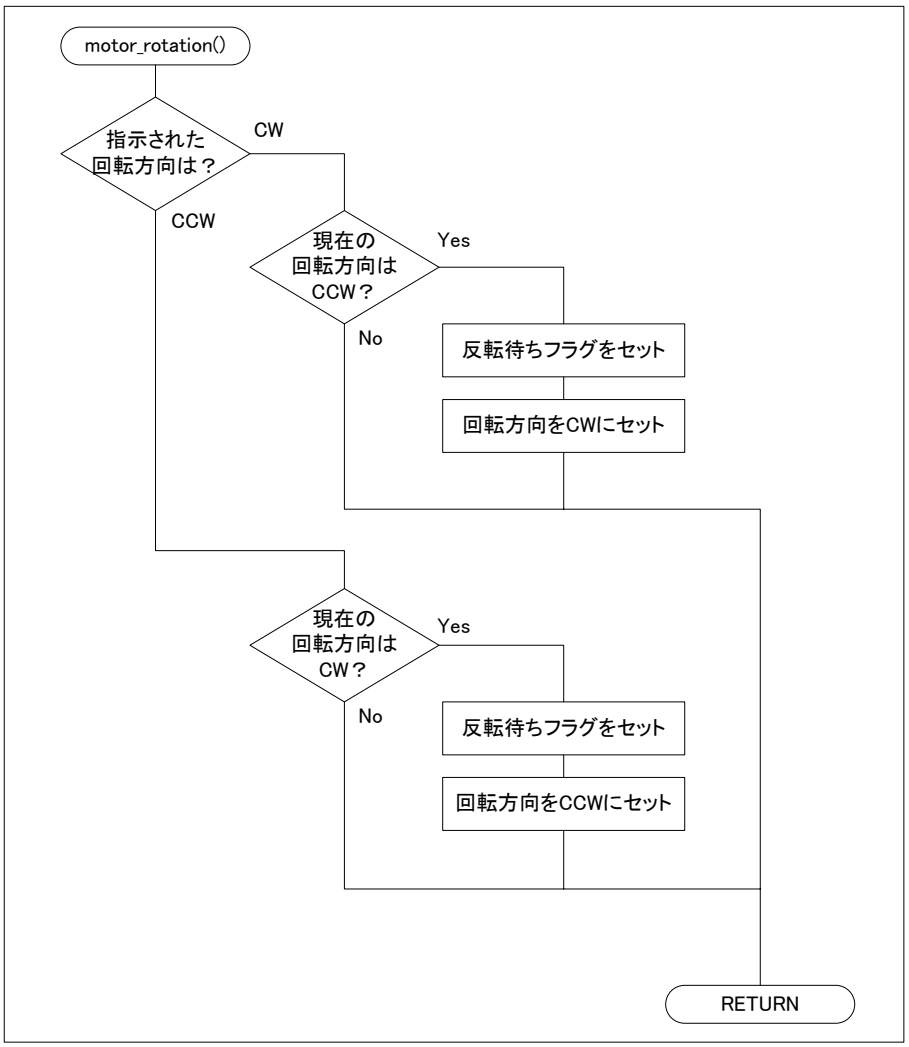


図 4－11　速度の PID 制御処理（motor_pid 関数）

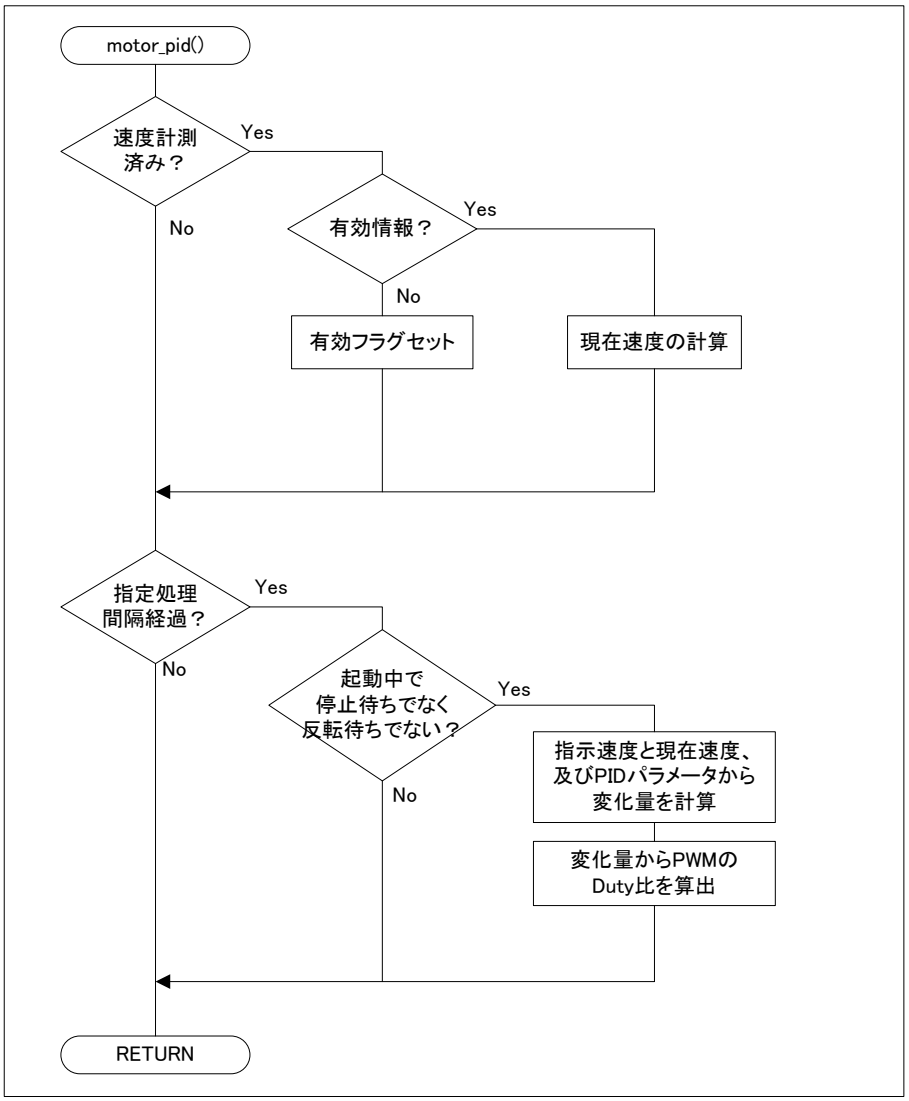


図 4－12　モータ制御パラメータの変更処理（motor_pset 関数）

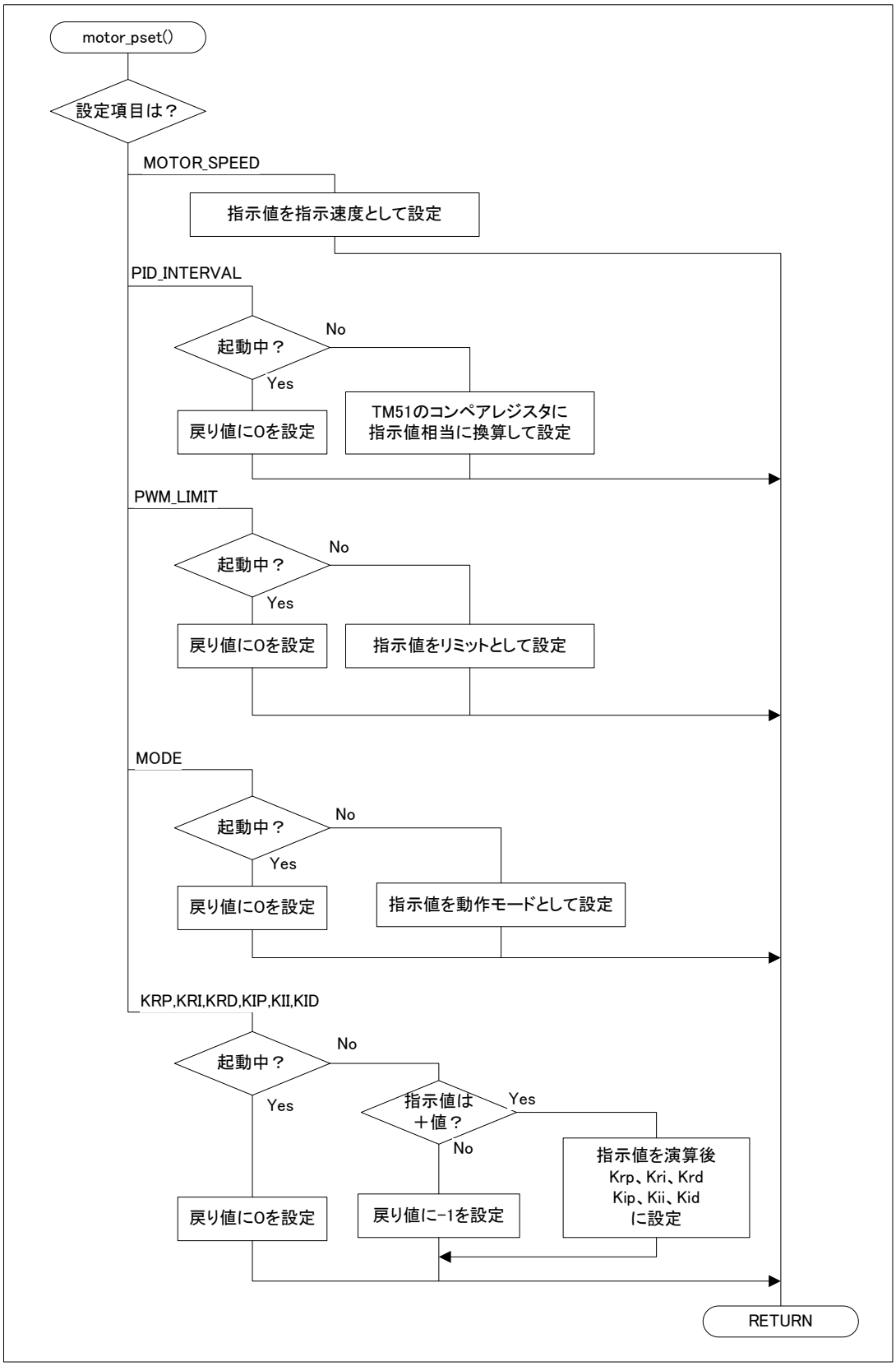


図 4－13　反転停止からの再起動処理（system_restart 関数）

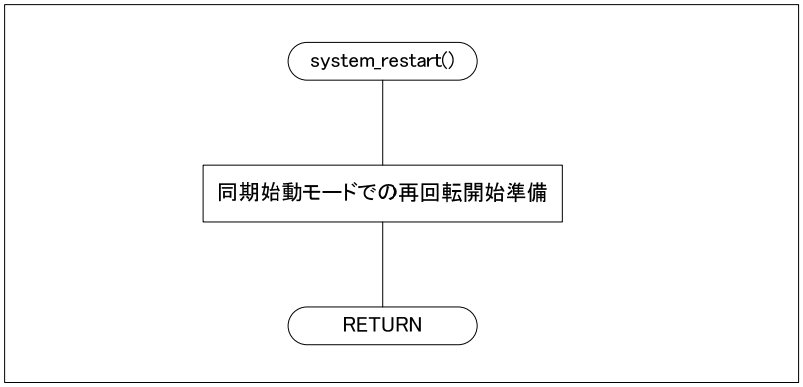


図 4－14　システム停止処理（system_stop 関数）

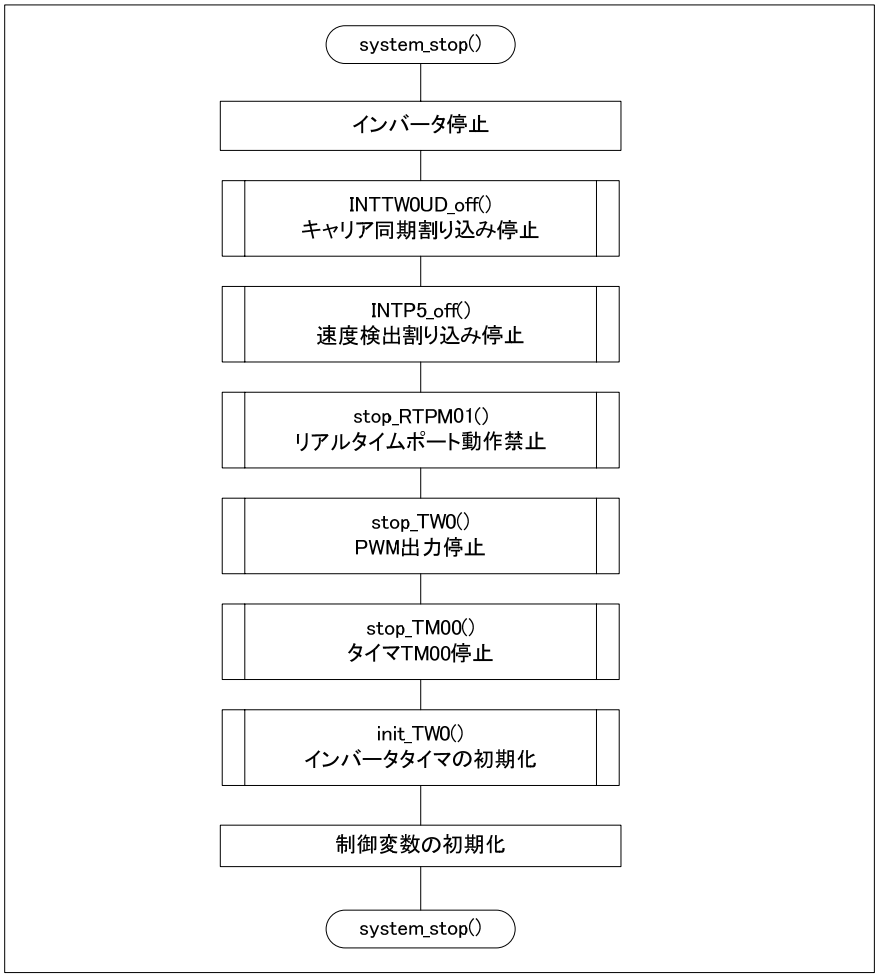


図 4－15　ポートの設定処理（init_PORT 関数）

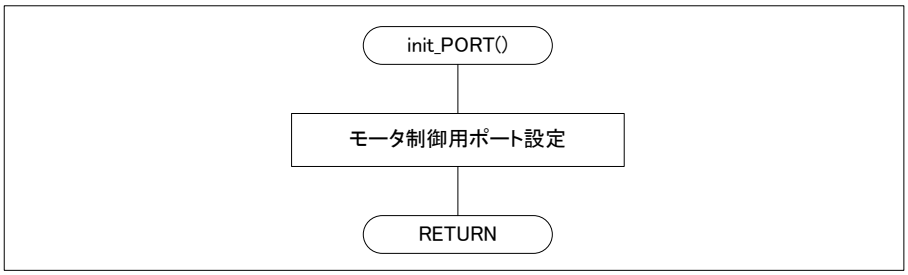


図 4－16　クロック切り替え処理（init_OSC 関数）

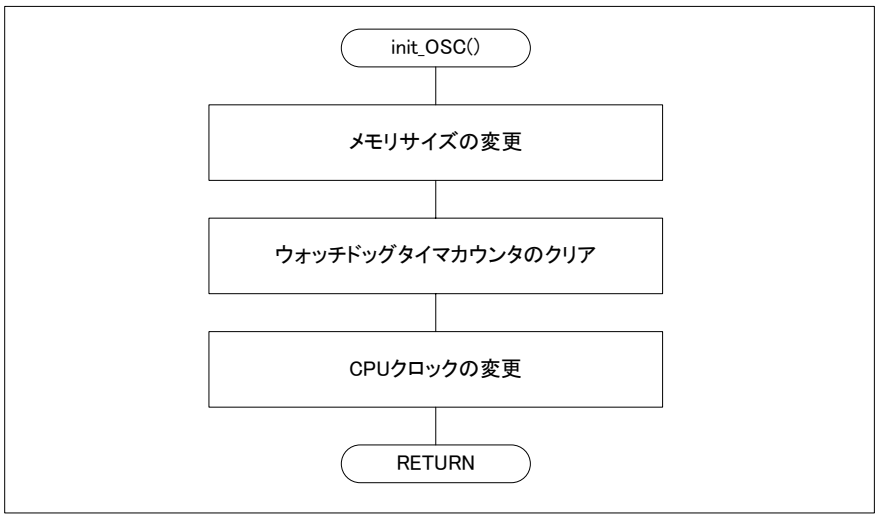


図 4－17　インバータ・タイマ初期設定処理（init_TW0 関数）

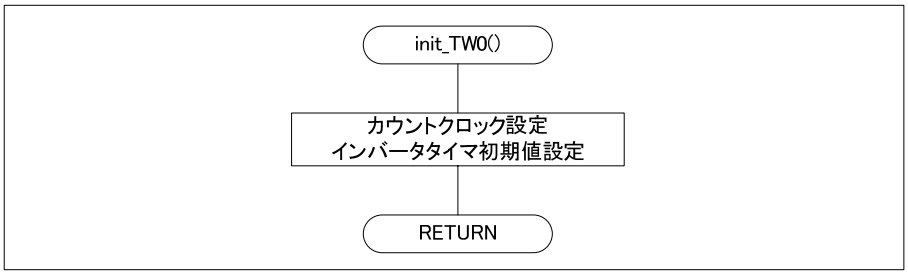


図 4－18　インバータ・タイマ動作開始処理（start_TW0 関数）

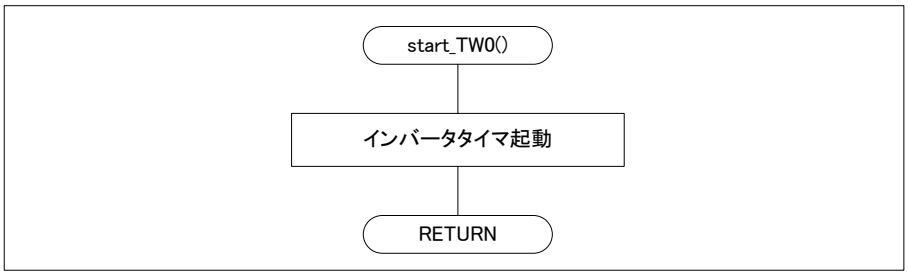


図 4－19　インバータ・タイマ動作停止処理（stop_TW0 関数）

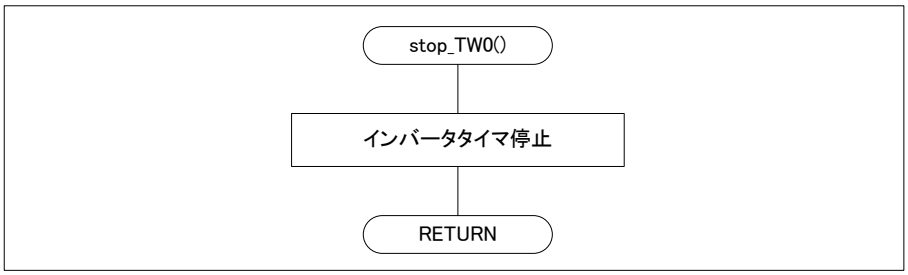


図 4－20　PWM のデューティ比設定処理（set_TW0 関数）

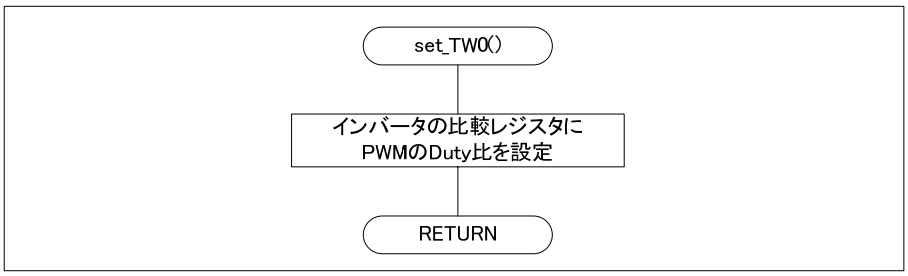


図 4－21　通電パターン切り替え用タイマ初期設定処理（init_TM00 関数）

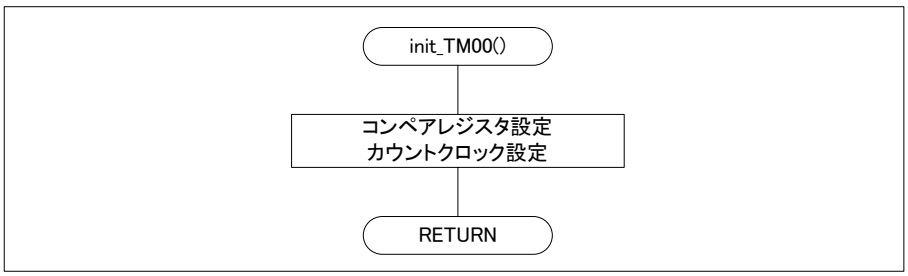


図 4－22　通電パターン切り替え用タイマ起動処理（start_TM00 関数）

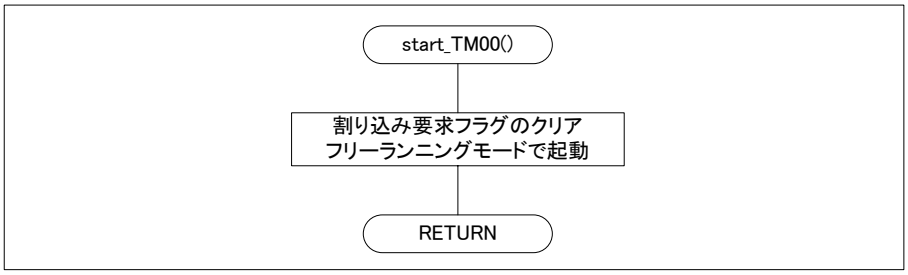


図 4－23　通電パターン切り替え用タイマ停止処理（stop_TM00 関数）

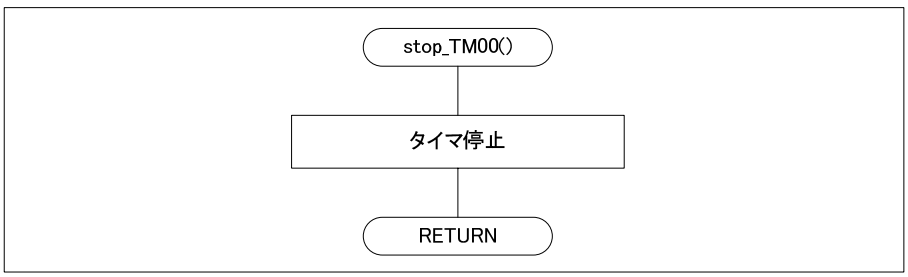


図 4－24　通電パターン切り替え用ポート初期設定処理（init_RTPM01 関数）

図 4－25　通電パターン切り替え用ポート出力許可処理（start_RTPM01 関数）

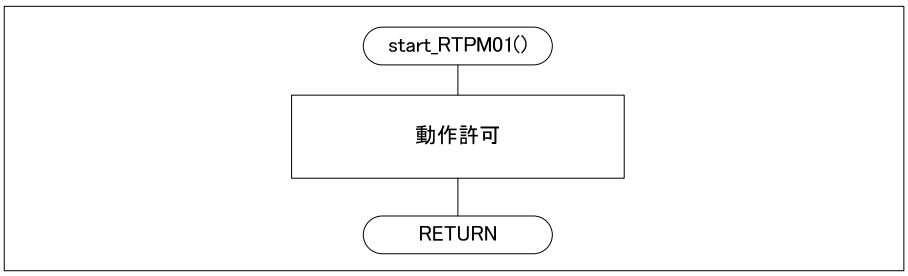


図 4－26　通電パターン切り替え用ポート出力禁止処理（stop_RTPM01 関数）

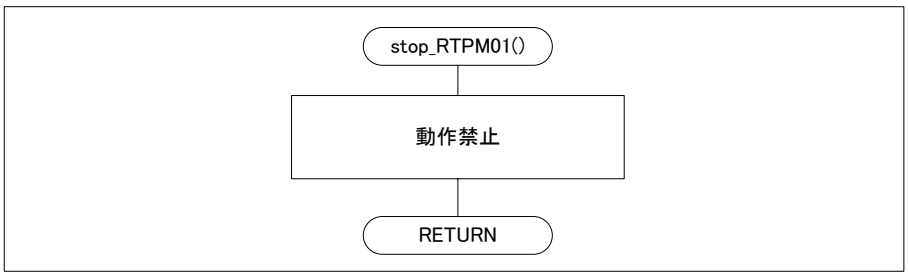


図 4－27　通電パターン切り替え用ポート PWM 出力モード切り替え処理（end_RTPM01 関数）

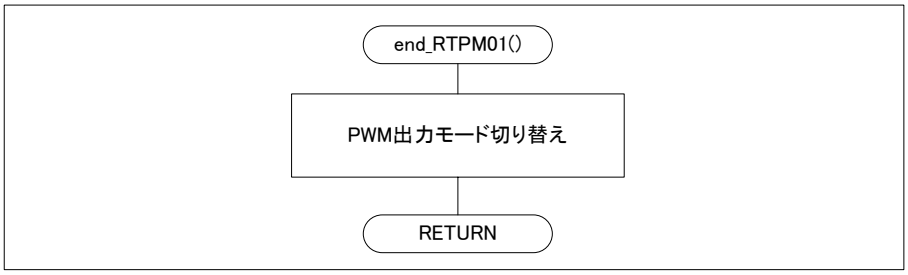


図 4－28　通電パターン設定（set_RTPM01 関数）

図 4－29　A/D 初期設定処理（init_AD 関数）

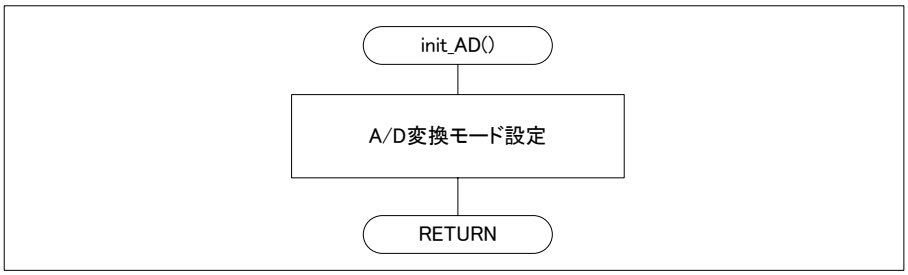


図 4－30　A/D 変換動作開始処理（start_AD 関数）

図 4－31　ウォッチドッグ・タイマの初期設定処理（init_WDTM 関数）

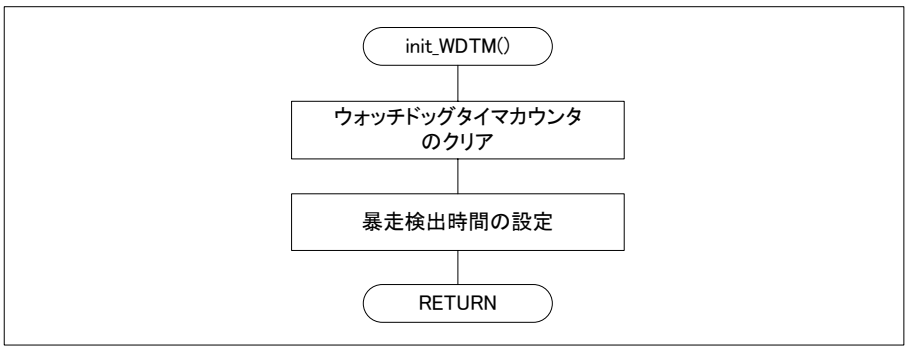


図 4－32　8 ビット・タイマ（TM51）初期設定処理(init_TM51 関数)

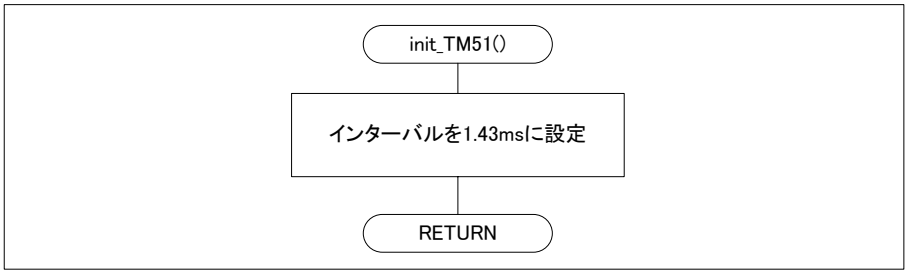


図 4－33　8 ビット・タイマ（TM51）起動処理（start_TM51 関数）

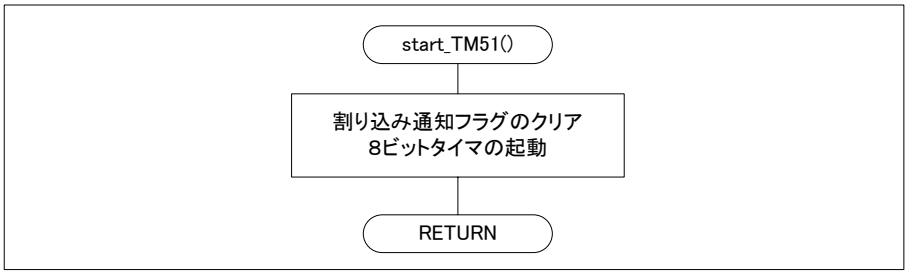


図 4－34　ホール IC の読み出し処理（read_Hall_IC 関数）

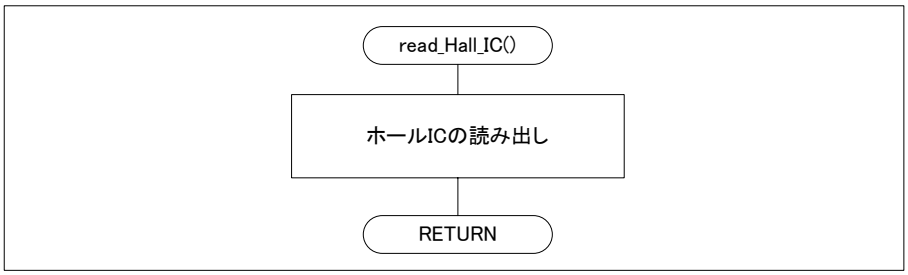


図 4－35　過電流割り込み許可処理（INTP0_on 関数）

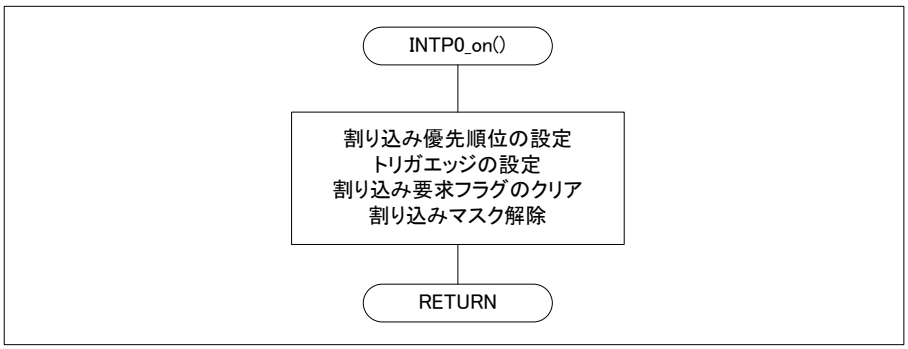


図 4－36　キャリア同期割り込み（谷処理）許可処理（INTTW0UD_on 関数）

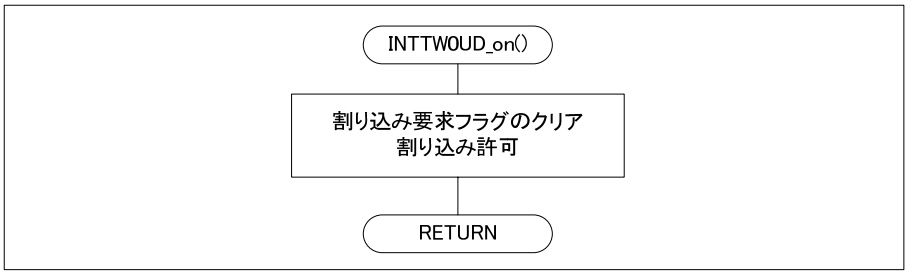


図 4－37　キャリア同期割り込み（谷処理）禁止処理（INTTW0UD_off 関数）

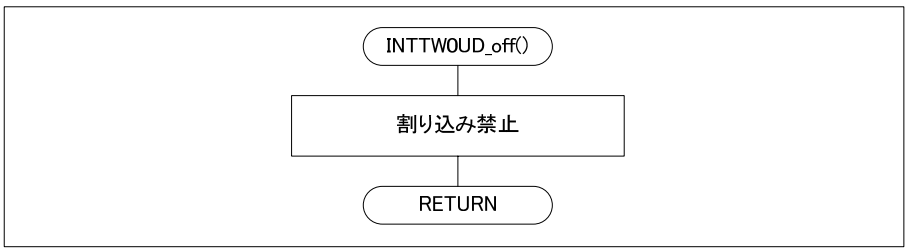


図 4－38　INTP5 割り込み許可処理（INTP5_on 関数）

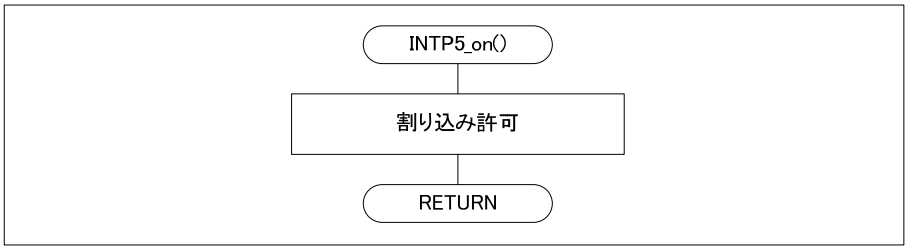


図 4－39　INTP5 割り込み禁止処理（INTP5_off 関数）

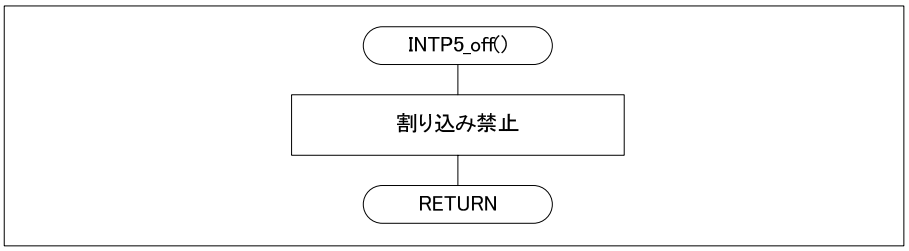


図 4－40　電流値取得許可処理（INTTM51_on 関数）

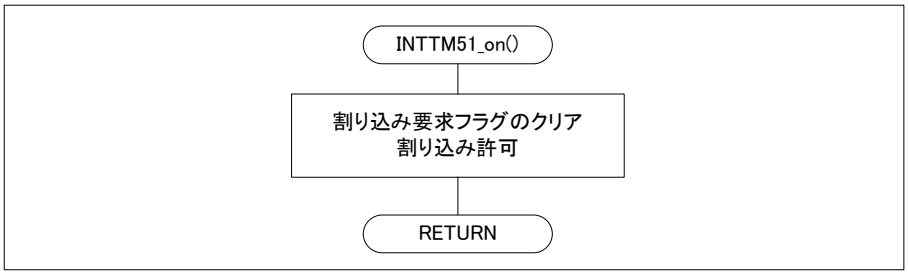


図 4－41　過電流割り込み処理（int_fault 関数）

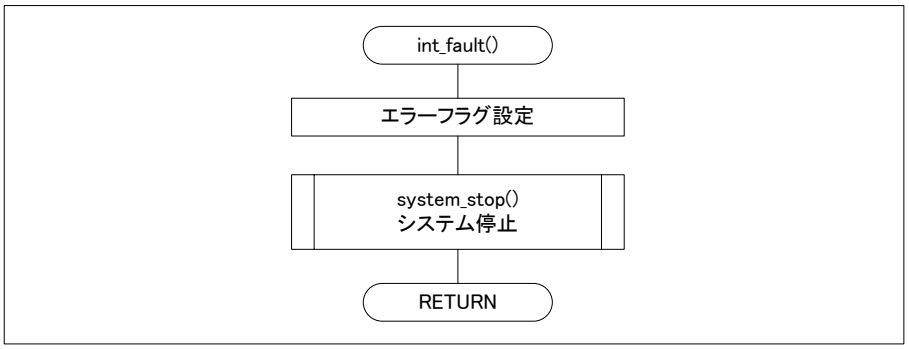


図 4－42　速度取得用割り込み処理（int_speed 関数）

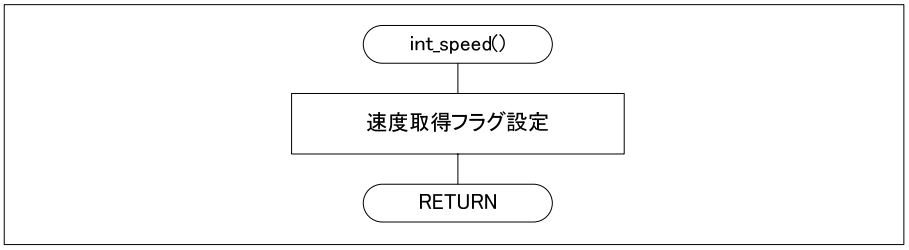


図 4－43　電流取得用割り込み処理（int_TM51 関数）

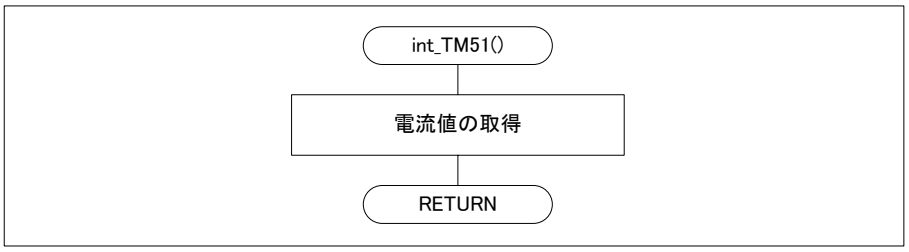


図 4－44　COS 演算処理（get_coswt 関数）

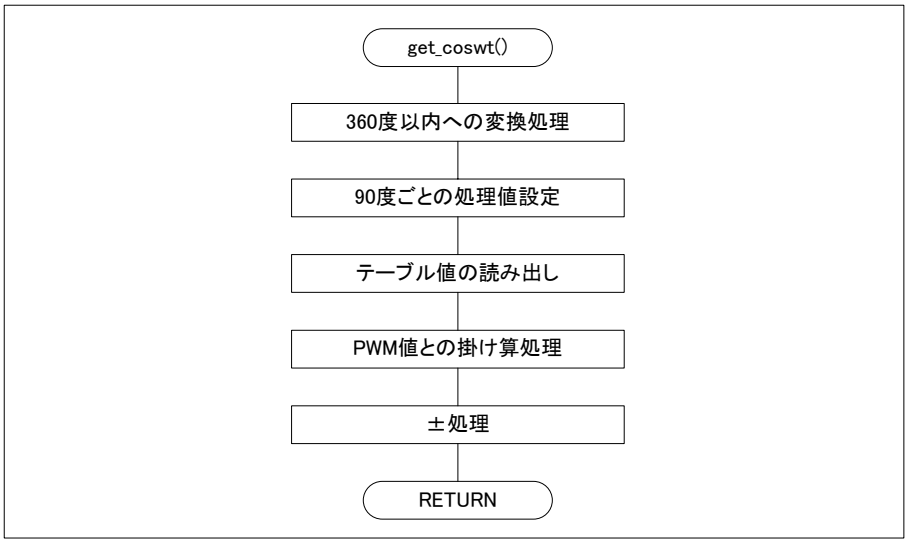


図 4－45　PWM 設定処理（set_pwm 関数）

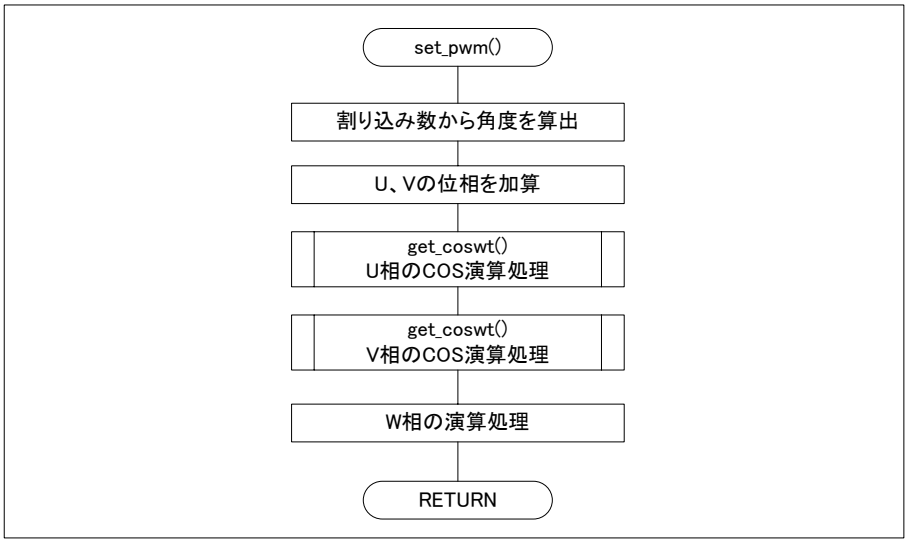


図 4－46　キャリア同期割り込み（谷処理）（1/2）(int_carrier 関数)

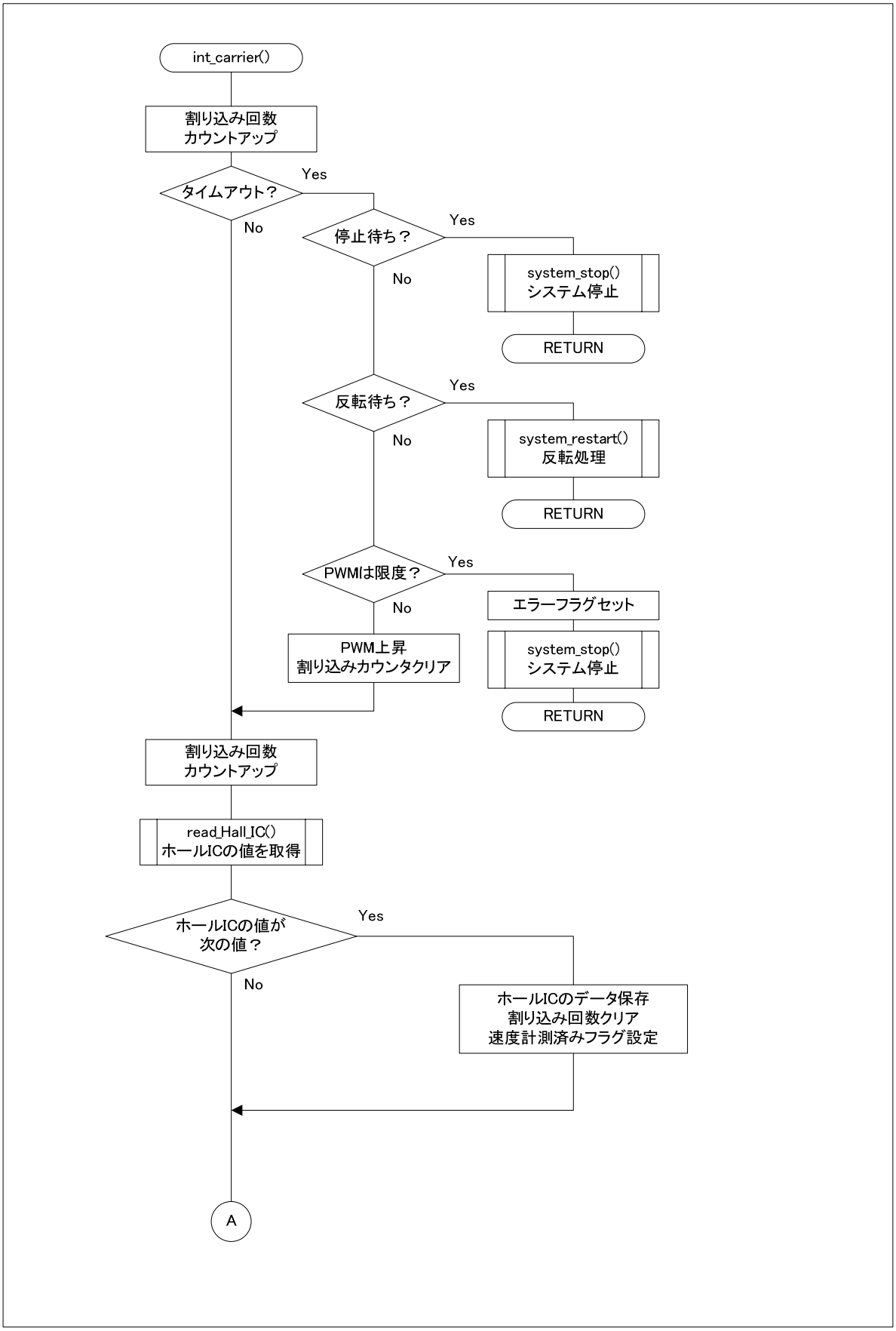
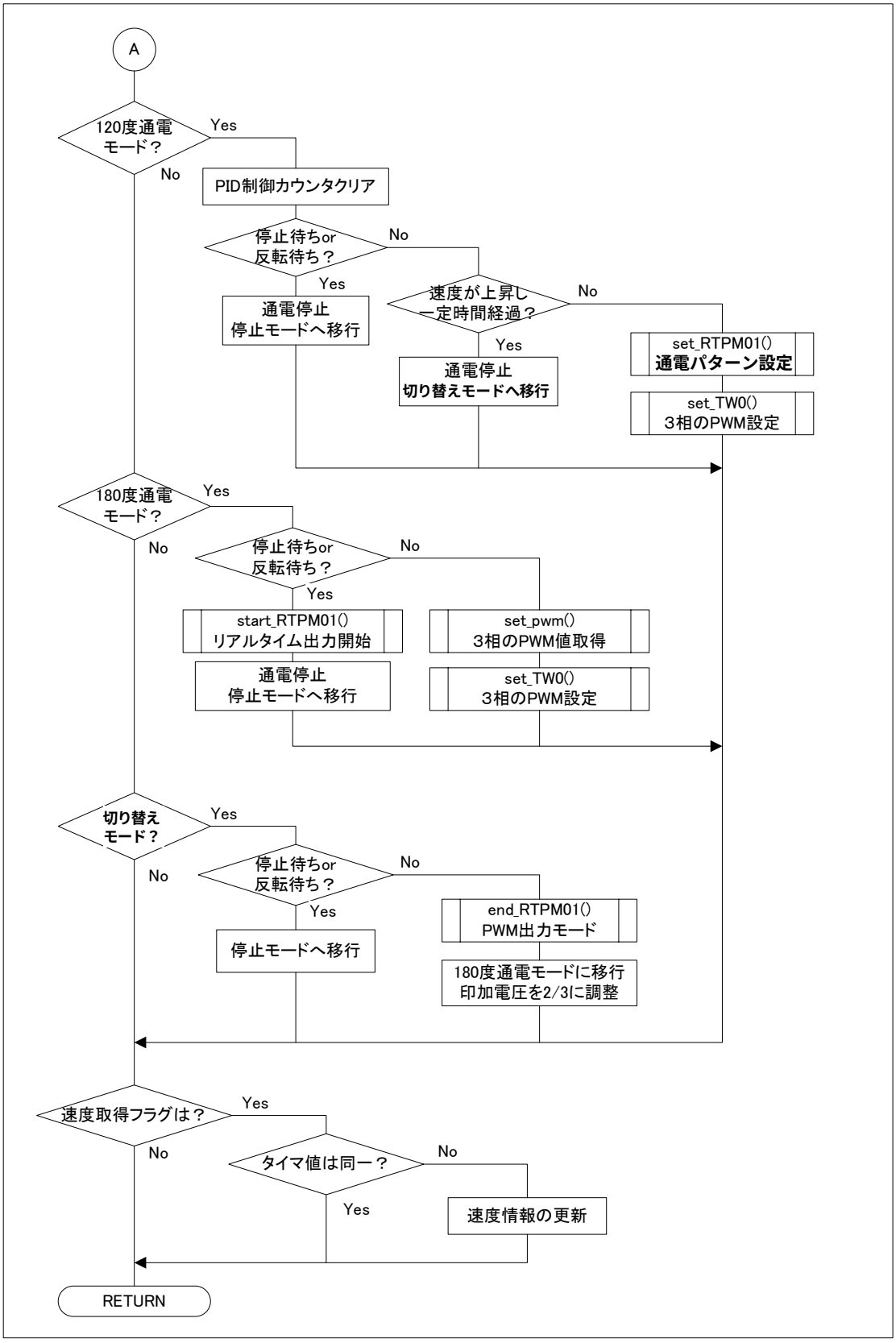


図 4－47　キャリア同期割り込み（谷処理）(2/2) (int_carrier 関数)



4.13　モータ・ライブラリの定数一覧

4.13.1 ユーザ変更可能定数

- 低電圧インバータ版設定

表 4−9　モータ・ライブラリのユーザ変更可能定数（L/V）一覧

名　　称	意　　味	設定値	備　　考
PWM_F_MIN	デューティ比の最小値	5	
PWM_F_MAX	デューティ比の最大値	PWM_BASE_REF	
STARTUP_METHOD_REF	初起動方法	PWM_START	センサレス制御用初期起動プロセスのための設定。GUIでの互換上必要なので残してある。PWM2_REFのみ使用している。
T_KNEE_REF	初起動切り替え時間	0.75	
T_END_REF	初起動終了時間	1.50	
RPM0_REF	初起動時初期回転数	60	
RPM1_REF	初起動切り替え時回転数	100	
RPM2_REF	初起動終了時回転数	200	
I_REF0_REF	初起動時初期電流値	140	
I_REF1_REF	初起動切り替え時電流値	160	
I_REF2_REF	初起動終了時電流値	160	
PWM0_REF	初起動時初期PWM値	100	
PWM1_REF	初起動切り替え時PWM値	100	
PWM2_REF	初起動終了時PWM値	100	
PWM_F_START	開始時PWM値	10	
ADGAIN_REF	A/D変換値の増幅量	1.0	
ADOFFSET_REF	A/D変換値のオフセット量	0	
MAXCURRENT_REF	最大電流値	1024	
MINCURRENT_REF	最小電流値	10	
MAXSPEED_REF	最大回転数	5000	
MINSPEED_REF	最小回転数	300	
I_RATE_MAX_REF	電流単位時間ごと上昇率	900	
RPM_RATE_MAX_REF	回転数単位時間ごと上昇率	2000	
KRP_REF_REF	回転数のKp値	0.05	
KRI_REF_REF	回転数のKi値	0.02	
KRD_REF_REF	回転数のKd値	0.01	
KIP_REF_REF	電流のKp値	0.10	
KII_REF_REF	電流のKi値	0.04	
KID_REF_REF	電流のKd値	0.02	
CLIMIT	過電流検知用電流値	700	

4.13.2 ユーザ参照可能定数

表 4－10　モータ・ライブラリのユーザ参照可能定数一覧

名　　称	意　　味	設定値	備　　考
FLG_ON	フラグ値	1	フラグの有効／無効を示します
FLG_OFF	フラグ値	0	
CW	回転方向	0	モータの回転方向を示します
CCW	回転方向	1	
ERROR_NONE	エラー・フラグ・ビット	0x00	エラー無し
ERROR_HALL	エラー・フラグ・ビット	0x01	ホールICのエラー
ERROR_OC	エラー・フラグ・ビット	0x02	過電流によるエラー
ERROR_MOTOR	エラー・フラグ・ビット	0x04	モータ異常のエラー
ERROR_S_OC	エラー・フラグ・ビット	0x08	過電流によるエラー
MOTOR_SPEED	パラメータ・セット・コード	0x01	モータ回転数の設定
PID_INTERVAL	パラメータ・セット・コード	0x10	PID間隔の設定
MODE	パラメータ・セット・コード	0x12	モード設定
KRP	パラメータ・セット・コード	0x20	回転数のKp設定
KRI	パラメータ・セット・コード	0x21	回転数のKi設定
KRD	パラメータ・セット・コード	0x22	回転数のKd設定
KIP	パラメータ・セット・コード	0x23	電流のKp設定
KII	パラメータ・セット・コード	0x24	電流のKi設定
KID	パラメータ・セット・コード	0x25	電流のKd設定
WDTE_CLR	ウォッチドッグ・タイマ値	0xac	WDTのクリア値
CURRENT_START	初起動モード	0x01	
PWM_START	初起動モード	0x02	
NOTABLE_START	初起動モード	0x03	
SPEED_CMD	動作モード	0x01	
I_CMD	動作モード	0x02	
V_CMD	動作モード	0x03	
AD_ISHUNT	電流検知チャネル	5	
UNIT_RPM	回転数演算用定数	2343750	

4.13.3 内部定数

表 4－11　モータ・ライブラリの内部変数一覧

名　　称	意　　味	設定値	備　　考
FLG_ON	フラグ値	1	フラグの有効／無効を示します
FLG_OFF	フラグ値	0	
FLG_WAIT	フラグ値	2	停止待ち
FLG_SW	フラグ値	3	切り替え待ち
FLG_120	フラグ値	12	120度通電方式
FLG_180	フラグ値	18	180度通電方式
PWM_BASE_REF	PWMベース	1000	PWM出力で使用
PWM_F_REF	PWM初期値	0	
PWM_DTM_REF	デッド・タイム	0	
IN	ポート設定用	1	ポート機能指定に使用
OUT	ポート設定用	0	
CLEAR	レジスタ・ビット設定用	0	レジスタのビット・アクセスで使用
SET	レジスタ・ビット設定用	1	
INV_OFF	インバータ制御用	1/0	低電圧インバータ
INV_ON	インバータ制御用	0/1	
INVERTER_SW	インバータ制御ポート	P54	インバータ制御ポート
INVERTER_SW_MODE	インバータ制御レジスタ	PM54	インバータ制御ポート
INTP0	ポート制御レジスタ	PM00	過電流検知ポート
IMS_DATA	メモリ・サイズ切り替え	0xc8	
WDTM_SET	WDT設定値	0x6f	
P_OFF	通電パターン	0x3f	リアルタイム・ポートの通電
P_STOP	通電パターン	0x15	
P_T1	通電パターン	0x36	
P_T2	通電パターン	0x1e	
P_T3	通電パターン	0x1b	
P_T4	通電パターン	0x39	
P_T5	通電パターン	0x2d	
P_T6	通電パターン	0x27	
INT_CNT_MAX	リミット値	500	無回転判定用（時間）
START_PWM_FF_MAX	リミット値	300	無回転判定用（電圧）
DELTA_CW	進角	0	CW側の進角
DELTA_CCW	進角	0	CCW側の進角
CHANGE_SPEED	リミット値	100	120度通電⇒180度通電への速度
CR01_ADD	カウンタ値	3	通電パターン切り替え待ち時間
PWM_MIDDLE	PWM値	500	PWM_BASE_REF ÷ 2
PWM_x3	PWM値	1500	PWM_MIDDLE × 3
INTERVAL_BASE	基準値	5	1[ms]を生成するための基準
TIME_OUT	基準値	5000	1[s]を生成するための基準

4.14　参考プログラムの定数一覧

4.14.1 内部定数

表 4－12　参考プログラムの内部定数一覧 (1/2)

名　　称	意　　味	設定値	備　　考
AD_VOL	A/Dチャネル	4	MCIOボード上のボリューム
IN	ポート設定用	1	ポート機能指定に使用
OUT	ポート設定用	0	
CLEAR	レジスタ・ビット設定用	0	レジスタのビット・アクセスで使用
SET	レジスタ・ビット設定用	1	
KEY_WAIT	スイッチ観察時間	10	チャタリング除去で使用
SW	スイッチ	(P7&0xf)	スイッチ接続ポート
SW2	ポート制御レジスタ	PM73	START/STOP用ポート
SW3	ポート制御レジスタ	PM72	FORWARD用ポート
SW4	ポート制御レジスタ	PM71	REVERSE用ポート
SW5	ポート制御レジスタ	PM70	MODE用ポート
LD_LED0	ポート制御レジスタ	PM64	LED選択用ポート
LD_LED1	ポート制御レジスタ	PM65	
LD_LED2	ポート制御レジスタ	PM66	
LD_LED3	ポート制御レジスタ	PM67	
LD_DATA	ポート制御レジスタ	PM4	LEDへのデータ出力ポート
START_SW	起動	0x7	スイッチの状態
STOP_SW	停止	0x7	
FORWARD_SW	CW	0xb	
REVERSE_SW	CCW	0xd	
MODE_SW	モード	0xe	
START_TR	状態設定用	0x01	制御開始
STOP_TR	状態設定用	0x02	制御停止
FORWARD_TR	状態設定用	0x04	回転をCWに変更
REVERSE_TR	状態設定用	0x08	回転をCCWに変更
MODE_TR	状態設定用	0x10	MODEスイッチが押された状態
LED_0	LED表示データ	0xc0	“0”を表示
LED_1		0cf9	“1” を表示
LED_2		0xa4	“2” を表示
LED_3		0xb0	“3” を表示
LED_4		0x99	“4” を表示
LED_5		0x92	“5” を表示
LED_6		0x82	“6” を表示
LED_7		0xf8	“7” を表示
LED_8		0x80	“8” を表示
LED_9		0x98	“9” を表示
LED_O		0xc0	“O”の変わりに“0” を表示
LED_I		0xcf	“I” を表示
LED_C		0xc6	“C” を表示

表 4－12　参考プログラムの内部定数一覧 (2/2)

名　　称	意　　味	設定値	備　　考
LED_H	LED表示データ	0x89	“H”を表示
LED_A		0x88	“A”を表示
LED_L		0xc7	“L”を表示
LED_		0xff	“ ”を表示
LED_S		0x92	“S”を表示
LED_E		0x86	“E”を表示
LED_F		0x8e	“F”を表示
LED_P		0x8c	“P”を表示
LED_dot		0x7f	“.”を表示

4.15　モータ・ライブラリの変数一覧

4.15.1 外部公開変数

表 4－13　モータ・ライブラリの外部公開変数一覧 (1/2)

変数名	型	意　　味	備　　考
sys_flag	char	制御フラグ	制御状態
err_flag	char	エラー・フラグ	エラー状態
stop_wait	char	停止指令フラグ	停止指令の有無
cw_ccw_flag	char	回転方向指令フラグ	回転方向状態
cw_ccw_wait	char	反転停止指令フラグ	反転による停止指令の有無
maxed_flags	unsigned char	上限フラグ	単位時間上限を越えた値の場合に立つフラグ
print_cnt	unsigned int	速度表示タイミング	キャリア同期割り込み回数
pwm_ff	int	PWM指令現在値	PWMのデューティ比の現在値
pwm_ff_o	int	PWM指令値	PWMのデューティ比の指示値
m_speed	int	実速度	モータの回転速度(rpm)
krp_ref	float	回転数Kp値	速度制御
kri_ref	float	回転数Ki値	
krd_ref	float	回転数Kd値	
kip_ref	float	電流Kp値	電流マイナー・ループ制御
kii_ref	float	電流Ki値	
kid_ref	float	電流Kd値	
startup_method	unsigned char	同期始動方法	
I_ref	float	電流指令現在値	
I_ref_o	float	電流指令値	
I_measured	float	動作電流値	
speed_ref	int	速度指令現在値	
speed_ref_o	int	速度指令値	
maxspeed	int	最大速度	
minspeed	int	最低速度	
dspeed_ref_max	int	単位時間内速度制御量	
dI_ref_max	float	単位時間内電流制御量	
RPM_rate_max	float	単位時間内速度変化量	

表 4－13　モータ・ライブラリの外部公開変数一覧 (2/2)

変数名	型	意　　味	備　　考
I_rate_max	float	単位時間内電流変化量	
t_knee	float	初起動切り替え時間	
t_end	float	初起動終了時間	
RPM0	float	初起動時初期回転数	
RPM1	float	初起動切り替え時回転数	
RPM2	float	初起動終了時回転数	
I_ref0	float	初起動時初期電流値	
I_ref1	float	初起動切り替え時電流値	
I_ref2	float	初起動終了時電流値	
PWM0	float	初起動時初期PWM値	
PWM1	float	初起動切り替え時PWM値	
PWM2	float	初起動終了時PWM値	
adgain	float	A/D変換値の増幅量	
adoffset	float	A/D変換値オフセット量	
maxcurrent	float	最大電流値	
mincurrent	float	最小電流値	

4.15.2 内部変数

表 4－14　モータ・ライブラリの内部変数一覧

変数名	型	意　　味	備　　考
old_hdata	char	ホールIC履歴	1つ前のホールICの値
speed_flag	char	速度情報フラグ	速度計算に必要なデータの有無
int_cnt	unsigned int	割り込みカウンタ	キャリア同期割り込み回数
pid_cnt	unsigned int	PID制御タイミング	キャリア同期割り込み回数
pwm_base	int	PWMベースクロック値	PWMのキャリア幅
old_ff	int	PWM指令値履歴	1つ前のpwm_ffの値
up_flag	char	実速度計算状態フラグ	回転速度計算用
capture_flag	char	タイマ値取得フラグ	
clk_flag	char	タイマ値演算可能フラグ	
old_clk	unsigned int	前回のクロック値	
new_clk	unsigned int	最新のクロック値	
mvn_f	float	前回の操作量	
startdelay	char	電流検知開始遅延値	
start_flag	char	回転開始フラグ	
cy_time	int	PID制御間隔	
cmd_mode	char	制御モード	
dpwm_ff_max	int	単位時間内PWM制御量	

4.16　参考プログラムの変数一覧

4.16.1 内部変数

表 4－15　参考プログラムの変数一覧

ad_flag	char	速度変更フラグ	指定速度変更機能を制限
led_data[ ]	char	LED出力データ	LEDに表示する数値

4.17　モータ・ライブラリのソース・ファイル

メモリ・サイズ　　ROM ： 23A8h
　　RAM ： 420h

```
/*


    BLDC モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット


    date    : 2007/05/30
    filename: motor.h

    NEC Micro Systems,Ltd
*/

#ifdef LOW
/* ========================================== */
/* For Pittman motor */
#define PWM_F_MIN          5            /* 最小値                          */
#define PWM_F_MAX          PWM_MIDDLE   /* 最大値                          */

/* startup parameters */
#define STARTUP_METHOD_REF PWM_START    /* which method to use: PWM_START, CURRENT_START, or NOTABLE_START */
#define T_KNEE_REF         2.0          /* start of second startup segment */
#define T_END_REF          4.0          /* end of startup second segment */
#define RPM0_REF           100          /* initial startup RPM */
#define RPM1_REF           200          /* midpoint startup RPM */
#define RPM2_REF           300          /* final startup RPM */
#define I_REF0_REF         140          /* initial startup current command */
#define I_REF1_REF         160          /* midpoint  startup current command */
#define I_REF2_REF         160          /* final  startup current command */
#define PWM0_REF           140          /* initial startup voltage setting */
#define PWM1_REF           145          /* midpoint startup voltage setting */
#define PWM2_REF           150          /* final startup voltage setting */

/* original startup NOTABLE_START */
#define SYNC_DEF_REF       500
#define PWM_F_START        150           /* 始動時の初期値                    */

/* a/d gain parameters (for GUI mostly) */
#define ADGAIN_REF         4.0          /* mA = Gain*(A/D - Offset) */
#define ADOFFSET_REF       0            /* */
#define MAXCURRENT_REF     1023         /* mA */
#define MINCURRENT_REF     10           /* mA */
#define MAXSPEED_REF       5000         /* RPM */
#define MINSPEED_REF       300          /* RPM */

/* Setpoint change maximum rates */
/* used for setpoint changes     */
#define I_RATE_MAX_REF     900          /* dI_int/sec maximum */
#define RPM_RATE_MAX_REF   2000         /* RPM/sec */

/* Feedback Gains: current measured in A/D units, voltage in PWM%*10 */
/*  RPM feedback to I command  #define */
#define KRP_DEF_REF        0.05         /* dI_ints/dRPM error */
#define KRI_DEF_REF        0.02         /* dI_ints/RPM error  */
#define KRD_DEF_REF        0.01         /* dI_ints/d(dRPM error) */

/*  Current feedback to PWM command */
#define KIP_DEF_REF        0.025        /* dPWM/dI_error */
#define KII_DEF_REF        0.010        /* dPWM/I_error  */
#define KID_DEF_REF        0.005        /* dPWM/d(derror) */

/* ========================================== */
#else
/* ========================================== */
/* For Oriental motor */
#define PWM_F_MIN          5            /* 最小値                          */
#define PWM_F_MAX          PWM_BASE_REF /* 最大値                          */

/* startup parameters */
#define STARTUP_METHOD_REF PWM_START /* which method to use: PWM_START, CURRENT_START, or NOTABLE_START */
#define T_KNEE_REF         0.75         /* start of second startup segment */
#define T_END_REF          1.5          /* end of startup second segment */
#define RPM0_REF           300          /* initial startup RPM */
#define RPM1_REF           300          /* midpoint startup RPM */
#define RPM2_REF           300          /* final startup RPM */
#define I_REF0_REF         140          /* initial startup current command */
#define I_REF1_REF         160          /* midpoint  startup current command */
#define I_REF2_REF         160          /* final  startup current command */
```

```
#define PWM0_REF          195          /* initial startup voltage setting */
#define PWM1_REF          195          /* midpoint startup voltage setting */
#define PWM2_REF          195          /* final startup voltage setting */

/* original startup NOTABLE_START */
#define SYNC_DEF_REF      500
#define PWM_F_START       70           /* 始動時の初期値                          */

/* a/d gain parameters (for GUI mostly) */
#define ADGAIN_REF        1.0          /* mA = Gain*(A/D - Offset) */
#define ADOFFSET_REF      100          /* */
#define MAXCURRENT_REF    1023         /* mA */
#define MINCURRENT_REF    10           /* mA */
#define MAXSPEED_REF      3000         /* RPM */
#define MINSPEED_REF      300          /* RPM */

/* Setpoint change maximum rates */
/* used for setpoint changes     */
#define I_RATE_MAX_REF    900          /* dI_int/sec maximum */
#define RPM_RATE_MAX_REF  3000         /* RPM/sec */

/* Feedback Gains: current measured in A/D units, voltage in PWM%*10 */
/*  RPM feedback to I command  #define */
#define KRP_DEF_REF       0.4          /* dI_ints/dRPM error */
#define KRI_DEF_REF       0.005        /* dI_ints/RPM error  */
#define KRD_DEF_REF       0.05         /* dI_ints/d(dRPM error) */

/*  Current feedback to PWM command */
#define KIP_DEF_REF       0.9          /* dPWM/dI_error */
#define KII_DEF_REF       0.9          /* dPWM/I_error  */
#define KID_DEF_REF       0            /* dPWM/d(derror) */

/* ======================================== */
#endif

#define CLIMIT          1024      /* 1024 */

#define AD_VOL          4        /* VOL 端子    */
#define AD_ISHUNT       5

/* +++++++++++++++ 以下，変更不可 +++++++++++++++ */

#define CW              0               /* 時計回り                          */
#define CCW             1               /* 反時計周り                        */

#define ERROR_NONE      0x00
#define ERROR_HALL      0x01     /* ホール IC 異常              */
#define ERROR_OC        0x02     /* 過電流                     */
#define ERROR_MOTOR     0x04     /* モータ異常                 */
#define ERROR_S_OC      0x08     /* 過電流（ソフトウェア検出）*/

#define MOTOR_SPEED     0x01     /* 指示回転数 */
#define PID_INTERVAL    0x10     /* PID 制御間隔 */
#define PWM_LIMIT       0x11     /* PWM 変化量リミッタ */
#define MODE            0x12
#define KRP             0x20     /* Kp 値 */
#define KRI             0x21     /* Ki 値 */
#define KRD             0x22     /* Kd 値 */
#define KIP             0x23     /* Kp 値 */
#define KII             0x24     /* Ki 値 */
#define KID             0x25     /* Kd 値 */

#define WDTE_CLR        0xac

#define CURRENT_START   0x01
#define PWM_START       0x02
#define NOTABLE_START   0x03

#define SPEED_CMD       0x01
#define I_CMD           0x02
#define V_CMD           0x03

  /*
    m_speed 用定数:  x[rpm] = ( 60[s] * 78.125[Krps] ) / 6
                                      ------          ---
                                        ↑             ↑カウンタ値取得
                                        ↑             ↑タイミング
                                        ↑             ↑HallIC(6)
                                        ↑TM00 カウントクロック
                                        ↑78.125[KHz]
  */
#define UNIT_RPM            2343750  /* 2count */
#define CARRIRE_DEF_CONST   50000
#define SPEED_CAL_CONST     100000

/* ---------------------------------------- */

extern char             sys_flag;    /* システム稼動状況         */
extern int              speed_ref;   /* 指示回転数               */
extern int              speed_ref_o;
extern int              m_speed;     /* 現在回転数               */
extern char             stop_wait;   /* 停止待ちフラグ           */
extern char             cw_ccw_wait; /* 反転待ちフラグ           */
extern char             cw_ccw_flag; /* 回転方向ステータス       */
```

```
extern char             err_flag;    /* エラー・フラグ             */
extern unsigned char    maxed_flags;
extern int              pwm_ff;
extern int              pwm_ff_o;
extern float            I_ref;
extern float            I_ref_o;
extern float            I_measured;
extern float            kip_ref, kii_ref, kid_ref;
extern float            krp_ref, kri_ref, krd_ref;
extern unsigned char    startup_method;
extern float            t_knee;
extern float            t_end;
extern float            RPM0, RPM1, RPM2;
extern float            I_ref0, I_ref1, I_ref2;
extern float            PWM0, PWM1, PWM2;
extern float            adgain;
extern int              adoffset;
extern float            maxcurrent, mincurrent;
extern float            I_rate_max, RPM_rate_max;
extern float            dI_ref_max;
extern int              dspeed_ref_max;
extern int              maxspeed, minspeed;
extern unsigned int     print_cnt;
extern unsigned int     ad_data_vol;

/* ---------------------------------------- */

void motor_init(void);
void motor_start(void);
void motor_stop(void);
void motor_rotation(char);
void motor_pid(void);
char motor_pset(unsigned char, long);

/* ------------------------------ EOF ---- */
```

```
/*


    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット


    date    : 2007/06/26
    filename: motor.c

    NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma ei
#pragma di

#pragma INTERRUPT INTP0     int_fault    rb1  /* 過電流発生 */
#pragma INTERRUPT INTP5     int_speed    rb1
#pragma INTERRUPT INTTWOUD  int_carrier  rb2  /* キャリア同期割り込み */
#pragma INTERRUPT INTTM51   int_TM51     rb3

/* ------------------------------------------------ */
#include "motor.h"

/* ------------------------------------------------ */
#define IN              1       /* 入力 */
#define OUT             0       /* 出力 */

#define FLG_ON          1       /* 有効 */
#define FLG_OFF         0       /* 無効 */
#define FLG_WAIT        2       /* 切り替わり待ち */
#define FLG_SW          3
#define FLG_120         12      /* 120 度通電方式 */
#define FLG_180         18      /* 180 度通電方式 */

#define START_TR        0x01    /* 押されているスイッチ判定  */
#define STOP_TR         0x02
#define FORWARD_TR      0x04
#define REVERSE_TR      0x08
#define MODE_TR         0x10

#define IMS_DATA        0xc8

#define WDTM_SET        0x6f

#define P_OFF           0x3f    /* リアルタイム出力ポート OFF */
#define P_STOP          0x15
#define P_T1            0x36    /* 通電パターン 1              */
#define P_T2            0x1e    /* 通電パターン 2              */
#define P_T3            0x1b    /* 通電パターン 3              */
#define P_T4            0x39    /* 通電パターン 4              */
#define P_T5            0x2d    /* 通電パターン 5              */
```

```
#define P_T6            0x27    /* 通電パターン 6               */

#define INT_CNT_MAX     500     /* 無回転の判定 x200us       */
#define START_PWM_FF_MAX 300    /* 無回転時の限界値 10%      */

#define DELTA_CW        0       /* CW 側の進角                 */
#define DELTA_CCW       0       /* CCW 側の進角                */

#define CHANGE_SPEED    100     /* 120 通電->180 度通電 の速度 */
#define CHANGE_TIME     2500

#define CLEAR           0
#define SET             1

#define CR01_ADD        3       /* 切り替え待ち時間用カウンタ値     */
                                /* TM00 の設定に依存             */

#define PWM_BASE_REF    1000    /* PWM 周期の半周期              */
#define PWM_F_REF       0       /* PWM 波形の初期値              */
#define PWM_DTM_REF     200     /* デッド・タイム                */

#define PWM_MIDDLE      PWM_BASE_REF / 2
#define PWM_x3          PWM_MIDDLE * 3

#define INTERVAL_BASE   5       /* 1ms = 200[us] * INTERVAL_BASE */
#define TIME_OUT        5000    /* 1s  = 200[us] * TIME_OUT      */
                                /* インバータタイマの設定に依存  */

#define INV_OFF         1       /* For LowVoltage */
#define INV_ON          0

#define INTP0           PM00    /* 過電流検知ポート  */
#define INVERTER_SW     P54     /* インバータ動作制御ポート  */
#define INVERTER_SW_MODE PM54   /* インバータ動作制御ポート  */


/* ------------------------------------------------ */
static const unsigned int sin_tbl[91] = {  /*  sin(theta) x (2^16 - 1) */
    0,  1143,  2287,  3429,  4571,  5711,  6850,  7986,
    9120, 10251, 11380, 12504, 13625, 14742, 15854, 16961,
    18063, 19160, 20251, 21336, 22414, 23485, 24549, 25606,
    26655, 27696, 28728, 29752, 30766, 31771, 32767, 33753,
    34728, 35692, 36646, 37589, 38520, 39439, 40347, 41242,
    42125, 42994, 43851, 44694, 45524, 46340, 47141, 47929,
    48701, 49459, 50202, 50930, 51642, 52338, 53018, 53683,
    54330, 54962, 55576, 56174, 56754, 57318, 57863, 58392,
    58902, 59394, 59869, 60325, 60762, 61182, 61582, 61964,
    62327, 62671, 62996, 63301, 63588, 63855, 64102, 64330,
    64539, 64728, 64897, 65046, 65175, 65285, 65375, 65445,
    65495, 65525, 65535
};


const unsigned char tr_sw[2][8] = {     /* 通電パターン       */
#ifdef LOW
    /*
       PITTMAN N2311A011 12VDC  driveLayer U:brown , V:red,    W:orange
                                Hall-sensor U:grey  , V:blue,   W:white
                                            Hall-1    Hall-2    Hall-3
     */
    {P_OFF, P_T4, P_T6, P_T5, P_T2, P_T3, P_T1, P_OFF},   /* CW              */
    {P_OFF, P_T1, P_T3, P_T2, P_T5, P_T6, P_T4, P_OFF}    /* CCW             */
#else
    /*
       OrientalMOTOR FBLM575W-A driveLayer U:purple, V:blue,   W:gray
                                Hall-sensor U:beige , V:orange, W:pink
                                            Hall-1    Hall-2    Hall-3
     */
    {P_OFF, P_T3, P_T5, P_T4, P_T1, P_T2, P_T6, P_OFF},   /* CW        */
    {P_OFF, P_T6, P_T2, P_T1, P_T4, P_T5, P_T3, P_OFF}    /* CCW       */
#endif
};

const unsigned char next_type[2][8] = {  /* 次のホール IC の値 */
    {8, 3, 6, 2, 5, 1, 4, 8},
    {8, 5, 3, 1, 6, 4, 2, 8}
};

unsigned int        int_cnt;                /* キャリア同期割り込み回数 */
unsigned int        pid_cnt;
unsigned int        start_cnt;
unsigned int        print_cnt;
unsigned int        cnt_data;
unsigned int        cnt_data1;
unsigned int        cnt_data2;
unsigned int        cnt_data3;

int                 m_speed   = 0;
int                 speed_ref = 0;          /* 指定速度             */
int                 speed_ref_o = 0;
float               I_ref;
float               I_ref_o;
float               I_measured;
```

```
char                stop_wait;              /* 停止待ちフラグ       */
char                cw_ccw_wait;            /* 反転待ちフラグ       */

char                cw_ccw_flag;            /* 回転方向ステータス   */
char                sys_flag;               /* システム稼動状況     */
char                err_flag = FLG_OFF;     /* エラー・フラグ         */
static char         start_flag;             /* 回転開始の判定       */
static char         speed_flag;             /* 速度情報の有無       */
static char         hall_new;
static char         hall_old;

static char         clk_flag;
static char         capture_flag;
static unsigned int new_clk, old_clk;       /* 速度測定用タイマ値       */

int                 sin_table_end = 90;
static unsigned int angle;                  /* 角度情報                 */
int                 pwm_ff;                 /* アクティブ幅：0 ～ 500   */
int                 pwm_ff_o;               /* アクティブ幅：0 ～ 500   */
int                 old_ff;                 /* アクティブ幅：0 ～ 500   */
int                 pwm_base = PWM_BASE_REF;/* PWM の半周期：1000 固定    */

unsigned char       startup_method = STARTUP_METHOD_REF;
unsigned char       maxed_flags;            /* bits to specify if on limits: bits 7-6-5 are RPM-Current-Volts respectively */

int                 maxspeed = MAXSPEED_REF;
int                 minspeed = MINSPEED_REF;

/* Setpoint change maximum rates */
float               RPM_rate_max = RPM_RATE_MAX_REF; /* RPM/sec */
float               I_rate_max   = I_RATE_MAX_REF;   /* dI_int/sec maximum */

static int          dpwm_ff_max  = PWM_BASE_REF /10; /* max change in pwm per control cycle */
int                 dspeed_ref_max;                  /* max change in speed ref per control cycle */
float               dI_ref_max;

float               t_knee = T_KNEE_REF;
float               t_end = T_END_REF;
float               RPM0 = RPM0_REF;
float               RPM1 = RPM1_REF;
float               RPM2 = RPM2_REF;
float               I_ref0 = I_REF0_REF;
float               I_ref1 = I_REF1_REF;
float               I_ref2 = I_REF2_REF;
float               PWM0 = PWM0_REF;
float               PWM1 = PWM1_REF;
float               PWM2 = PWM2_REF;

/* RPM feedback to I command */
float               krp_ref = KRP_DEF_REF;  /* dI_ints/dRPM error    */
float               kri_ref = KRI_DEF_REF;  /* dI_ints/RPM error     */
float               krd_ref = KRD_DEF_REF;  /* dI_ints/d(dRPM error) */

/* Current feedback to PWM command */
float               kip_ref = KIP_DEF_REF;  /* dPWM/dI_error  */
float               kii_ref = KII_DEF_REF;  /* dPWM/I_error   */
float               kid_ref = KID_DEF_REF;  /* dPWM/d(derror) */

float               adgain = ADGAIN_REF;
int                 adoffset = ADOFFSET_REF;
float               maxcurrent = MAXCURRENT_REF;
float               mincurrent = MINCURRENT_REF;


/*
    スタティック変数
*/
static unsigned int  cy_time = 150 * INTERVAL_BASE;
static char          cmd_mode = SPEED_CMD;
static char          up_flag;               /* 速度更新通知フラグ       */
static unsigned int  pwm_ff_u, pwm_ff_v, pwm_ff_w;
static float         mvn;                   /* 操作量                   */
static float         en, en_1, en_2;        /* 偏差                     */

/* ------------------------------------------------ */
static void init_PORT(void);
static void init_OSC(void);
static void init_TW0(void);
static void init_TW00(void);
static void init_RTPW01(void);
static void init_AD(void);
static void init_TM51(void);
static void init_WDTM(void);

static void start_AD(void);
static void start_TW0(void);
static void start_TW00(void);
static void start_TM51(void);
static void start_RTPW01(void);

static void stop_TW0(void);
static void stop_TW00(void);
static void stop_RTPW01(void);
```

```
static void INTP0_on(void);
static void INTP5_on(void);
static void INTTM51_on(void);
static void INTTWOUD_on(void);

static void INTP5_off(void);
static void INTTWOUD_off(void);

static void set_TWO(unsigned int, unsigned int, unsigned int, unsigned int);
static void set_RTPWO1(unsigned char);

static void wait(unsigned int);

static void system_restart(void);
static void system_stop(void);

static char read_Hall_IC(void);
static void filters(void);


/* ============================================== */
/* ------------------------------------------------
    ユーザ公開モータ制御関数部
   ------------------------------------------------ */

/* --------------------------------
    モータ制御関係機能の初期設定
  -------------------------------- */
void
motor_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TWO();
    init_TWOO();
    init_RTPWO1();
    init_AD();
    init_TM51();

    dspeed_ref_max = (int)(RPM_rate_max/10.0);
    dI_ref_max     = (float)(I_rate_max/10.0);

    start_TM51();
    INTTM51_on();

    start_AD();             /* AD 変換開始                  */
    INTP0_on();             /* 過電流割り込みマスク解除     */
    EI();                   /* 割り込み許可                 */
}

/* --------------------------------
    システム起動
   -------------------------------- */
void
motor_start(void)
{
    en_1         =
    en           =
    mvn          = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    =
    start_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 500;
    cnt_data     = 1500;
    new_clk      =
    old_clk      =
    m_speed      = 0;
    sys_flag     = FLG_120;
    start_flag   = FLG_ON;
    stop_wait    = FLG_OFF;
    cw_ccw_wait  = FLG_OFF;
    speed_flag   = FLG_OFF;
    clk_flag     = FLG_OFF;
    up_flag      = FLG_OFF;
    capture_flag = FLG_OFF;
    err_flag     = ERROR_NONE;
    pwm_ff       = (int)PWM2;
    I_ref        = I_measured;
    speed_ref    = (int)RPM2;
    hall_old     = read_Hall_IC();

    start_TWOO();
    INTTWOUD_on();        /* キャリア同期割り込みマスク解除  */
    INTP5_on();           /* ホール IC 割り込みマスク解除      */
    start_RTPWO1();
    start_TWO();

    INVERTER_SW = INV_ON;    /* INVERTER enable  */
}
```

```
/* --------------------------------
    モータ停止処理
  -------------------------------- */
void
motor_stop(void) {
    stop_wait = FLG_ON;
}


/* --------------------------------
    モータ方向変更処理
  -------------------------------- */
void
motor_rotation(char dir) {
    switch(dir) {
    case CW:
        if(cw_ccw_flag == CCW) {
            cw_ccw_wait = FLG_ON;
            cw_ccw_flag = CW;
        };
        break;
    default:
        if(cw_ccw_flag == CW) {
            cw_ccw_wait = FLG_ON;
            cw_ccw_flag = CCW;
        };
    };
}

/* --------------------------------
    PID制御
  -------------------------------- */
void
motor_pid(void)
{
    unsigned long tmp;
    unsigned long old_tmp, new_tmp;
    int pwm_tmp;

    if(speed_flag == FLG_ON){           /* 速度計測済み */
        speed_flag = FLG_OFF;
        if(start_flag == FLG_OFF){
            if(clk_flag == FLG_ON){
                clk_flag = FLG_OFF;
                up_flag = FLG_ON;       /* 速度更新済み */
                angle = cnt_data1;
                DI();
                new_tmp = new_clk;
                old_tmp = old_clk;
                EI();
                if(old_tmp > new_tmp){
                    tmp = (unsigned long)((65536 - (long)old_tmp) + (long)new_tmp);
                }else{
                    tmp = new_tmp - old_tmp;
                }
                m_speed = (int)(UNIT_RPM/tmp);
            };
        }else{
            start_flag = FLG_OFF;   /* 先頭の速度情報は使用しない */
        }
    }
    if(pid_cnt >= cy_time){    /* cy_time(ms)経過   */
        pid_cnt = 0;
        if(up_flag == FLG_ON){
            up_flag = FLG_OFF;
            if((sys_flag    != FLG_OFF) &&  /* 起動中                 */
               (stop_wait   != FLG_ON) &&   /* 停止待ちでない         */
               (cw_ccw_wait != FLG_ON)){    /* 反転の停止待ちでない   */
                maxed_flags = 0;    /* reset all flags here */
                switch(cmd_mode) {
                case V_CMD:         /* voltage control: pwm set by GUI */
                    /* no loop on speed */
                    speed_ref = m_speed;
                    /* no loop on current */
                    I_ref = I_measured;
                    /* adjust setpoint (acceleration limits) */
                    if((pwm_ff_o - pwm_ff) > dpwm_ff_max) {
                        pwm_ff += dpwm_ff_max;
                        maxed_flags |= 0x20;  /* on limit, set bit 5 */
                    } else if((pwm_ff_o - pwm_ff) < -dpwm_ff_max) {
                        pwm_ff -= dpwm_ff_max;
                        maxed_flags |= 0x20;  /* on limit, set bit 5 */
                    } else {
                        pwm_ff = pwm_ff_o;
                    };
                    break;

                case I_CMD:         /* current control (no use) */
                case SPEED_CMD:     /* speed control            */
                    /* no loop on current */
                    I_ref = I_measured;
                    /* adjust setpoint (acceleration limits) */
                    if((speed_ref_o - speed_ref) > dspeed_ref_max) {
                        speed_ref += dspeed_ref_max;
```

```
                    maxed_flags |= 0x80;  /* on limit, set bit 7 */
                } else if((speed_ref_o - speed_ref) < -dspeed_ref_max) {
                    speed_ref -= dspeed_ref_max;
                    maxed_flags |= 0x80;  /* on limit, set bit 7 */
                } else {
                    speed_ref = speed_ref_o;
                };
                en_2 = en_1;  /* last last error */
                en_1 = en;    /* last error */
                en   = (float)(speed_ref - m_speed); /* up-to-date error */

                mvn = mvn
                    + krp_ref*(en - en_1)
                    + kri_ref*en
                    + krd_ref*((en - en_1) - (en_1 - en_2));

                pwm_tmp = (int)(mvn);

                if(pwm_tmp > dpwm_ff_max) {
                    pwm_tmp += dpwm_ff_max;
                    maxed_flags |= 0x20;  /* on limit, set bit 5 */
                } else if(pwm_tmp < -dpwm_ff_max) {
                    pwm_tmp -= dpwm_ff_max;
                    maxed_flags |= 0x20;  /* on limit, set bit 5 */
                };

                pwm_ff += pwm_tmp;

                if(pwm_ff > PWM_F_MAX) {
                    pwm_ff = PWM_F_MAX;
                    mvn = 0.0;
                } else if(pwm_ff < PWM_F_MIN) {
                    pwm_ff = PWM_F_MIN;
                    mvn = 0.0;
                };
                break;
            };
            MDB0 = (unsigned int)pwm_ff;
        };
    };
};
}

/*  --------------------------------
    モータパラメータ設定
  -------------------------------- */
char
motor_pset(unsigned char com, long data) {
    char status = 1;

    switch(com) {
    case MOTOR_SPEED:
        speed_ref_o = (int)data;
        break;
    case PID_INTERVAL:
        if(sys_flag == FLG_OFF) {
            cy_time = (unsigned int)((int)data*INTERVAL_BASE);
        } else {
            status = 0;
        };
        break;
    case PWM_LIMIT:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                dpwm_ff_max = (int)data;
            };
        } else {
            status = 0;
        };
        break;
    case MODE:
        if(sys_flag == FLG_OFF) {
            cmd_mode = (char)data;
        } else {
            status = 0;
        };
        break;
    case KRP:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                krp_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    case KRI:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
```

```
            } else {
                kri_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    case KRD:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                krd_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    case KIP:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                kip_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    case KII:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                kii_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    case KID:
        if(sys_flag == FLG_OFF) {
            if(data < 0) {
                status = -1;
            } else {
                kid_ref = (float)data / 1000;
            };
        } else {
            status = 0;
        };
        break;
    default:
        status = -1;
    };

    return(status);
}


/* ================================================ */
/* ------------------------------------------------
    ユーザ非公開モータ制御関数部
   ------------------------------------------------ */
/*
    反転停止からの再起動
*/
void
system_restart(void)
{
    init_TW0();

    en_1         =
    en           =
    mvn          = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    =
    start_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 500;
    cnt_data     = 1500;
    new_clk      =
    old_clk      =
    m_speed      = 0;
    sys_flag     = FLG_120;
    start_flag   = FLG_ON;
    cw_ccw_wait  = FLG_OFF;
    speed_flag   = FLG_OFF;
    clk_flag     = FLG_OFF;
    up_flag      = FLG_OFF;
    capture_flag = FLG_OFF;
    pwm_ff       = (int)PWM2;
    I_ref        = I_measured;
```

```
    speed_ref    = (int)RPM2;

    hall_old     = read_Hall_IC();
}

/*
    システム停止
*/
void
system_stop(void)
{
    pwm_ff = 0;
    INVERTER_SW = INV_OFF;  /* INVERTER disable        */

    INTTWOUD_off();         /* 同期キャリア割り込み停止 */
    INTP5_off();
    stop_RTPM01();          /* リアルタイム・ポート停止 */
    stop_TWO();             /* インバータ・タイマ停止   */
    stop_TWO0();            /* 16 ビットタイマ停止      */

    init_TWO();

    sys_flag     = FLG_OFF;
    stop_wait    = FLG_OFF;
    cw_ccw_flag  = CW;
    m_speed      = 0;       /* 速度 0                   */
}

/*
    ポートの設定
*/
static void
init_PORT(void)
{
    INTP0 = IN;                         /* 過電流通知の割り込み */

    INVERTER_SW_MODE = OUT;             /* INVERTER enable/disable */
    INVERTER_SW      = INV_OFF;         /* INVERTER disable        */

    EGP5 = SET;                         /* 立ち上がりエッジ有効 */
    EGN5 = CLEAR;
}


/*
    クロック切り替え
*/
static void
init_OSC(void)
{
    IMS = IMS_DATA;                     /* メモリ・サイズ切り替え */
    WDTE = WDTE_CLR;
    while(OSTC != 0x1f);                /* 発振安定待ち */
    PCC   = 0x00;                       /* 分周比設定 */
    MCN.0 = 1;                          /* X1 入力クロック */
    VSWC.1 = 1;
}

/*
    インバータタイマ
*/
static void
init_TWO(void)
{
    TCL02    = 0;       /* カウント・クロック:20MHz */
    TCL01    = 0;
    TCL00    = 0;
    IDEV02   = 0;       /* 2 回に 1 回割り込み発生    */
    IDEV01   = 0;
    IDEV00   = 1;
    TWOM     = 0;
    TWOTRGS  = 0;
    TWOOC    = 0;
    TWOCM3   = PWM_BASE_REF;
    TWOCM0   = PWM_BASE_REF - PWM_F_REF;
    TWOCM1   = PWM_BASE_REF - PWM_F_REF;
    TWOCM2   = PWM_BASE_REF - PWM_F_REF;
    TWODTIME = PWM_DTM_REF;                 /* デッド・タイム                       */
    TWOBFCM3 = PWM_BASE_REF;
    TWOBFCM0 = PWM_BASE_REF - PWM_F_REF;
    TWOBFCM1 = PWM_BASE_REF - PWM_F_REF;
    TWOBFCM2 = PWM_BASE_REF - PWM_F_REF;
}

static void
start_TWO(void)
{
    TWOC.7 = SET;                       /* タイマスタート   */
}

static void
stop_TWO(void)
{
    TWOC.7 = CLEAR;                     /* タイマストップ   */
```

```
}

void
set_TWO(unsigned int u, unsigned int v, unsigned int w, unsigned int base)
{
    TWOBFCM3 = base;
    TWOBFCM0 = base - u;
    TWOBFCM1 = base - v;
    TWOBFCM2 = base - w;
}

/*
    16 ビットタイマ

    CR01    リアルタイム出力ポート・トリガ
*/
static void
init_TM00(void)
{
    ES000  = 0;
    ES001  = 0;                     /* TI000 端子の有効エッジ 立下りエッジ      */
    CRC000 = 1;                     /* CR00 キャプチャレジスタ                  */
    CRC001 = 1;                     /* TI000 端子の有効エッジの逆相でキャプチャ */
    CRC002 = 0;                     /* CR01 コンペアレジスタ                    */
    PRM001 = SET;
    PRM000 = CLEAR;                 /* カウント・クロック 78.125kHz */
}

static void
start_TM00(void)
{
    TMIF00 = CLEAR;
    TMC00  = 0x04;                  /* フリーランニングモード   */
}

static void
stop_TM00(void)
{
    TMC00 = CLEAR;                  /* タイマ停止               */
}


/*
    リアルタイムポート
*/
static void
init_RTPM01(void)
{
    RTPM01  = 0x3f;                 /* リアルタイム出力モード   */
    RTPC01  = 0x20;                 /* 6 ビット x1 チャンネル     */
    DCCTL01 = 0xc0;                 /* PWM 変調出力             */
    RTBL01  = 0x3f;                 /* 出力バッファ             */
}

void
start_RTPM01(void)
{
    DCCTL01.7 = SET;                /* リアルタイム出力 */
    RTPC01.7  = SET;                /* 動作許可 */
}

static void
stop_RTPM01(void)
{
    RTPC01.7 = CLEAR;               /* 動作禁止 */
}

void
end_RTPM01(void)
{
    DCCTL01.7 = CLEAR;              /* インバータ・タイマ出力 */
}

void
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                  /* 通電パターン設定           */
    CR01   = TM00 + CR01_ADD;
}


/*
    AD
*/
static void
init_AD(void)
{
    ADCS2 = SET;                    /* 回路動作許可 */
    ADS   = AD_ISHUNT;              /* SHUNT */
    ADM   = 0x04;                   /* 3.6us */
}

static void
start_AD(void)
```

```
{
    ADIF = CLEAR;                       /* 割り込み通知フラグクリア */
    ADCS = SET;                         /* 変換動作許可             */
    while(ADIF != SET);                 /* 割り込み通知待ち         */
}


/*
    ウォッチドッグ・タイマ
*/
static void
init_WDTM(void)
{
    WDTE = WDTE_CLR;
    WDTM = WDTM_SET;
}


/* --------------------------------
    8 ビット・タイマ 51
  -------------------------------- */
static void
init_TM51(void) {
    TMC51 = 0x00;                       /* no PWM, TIMER out disabled */
    TCL51 = 0x05;                       /* 19.53KHz */
    CR51  = 28;                         /* 1.43 ms */
}

static void
start_TM51(void) {
    TMIF51 = CLEAR;                     /* 割り込み通知フラグクリア     */
    TCE51  = SET;                       /* タイマ開始                   */
}


/*
    ホール IC の読み出し
*/
char
read_Hall_IC(void)
{
    return((char)((P0>>1)&0x7));
}


/* ============================================== */
/*
    割り込み許可／禁止処理
 */
/*
    過電流割り込み許可
*/
void
INTP0_on(void)
{
    PR0L.1 = 0;                         /* 優先順位         */
    EGN.0  = 1;                         /* 立ち下がりエッジ */
    IF0L.1 = CLEAR;                     /* フラグクリア     */
    MK0L.1 = CLEAR;                     /* マスククリア     */
}

/*
    キャリア同期割り込み許可
*/
static void
INTTWOUD_on(void)
{
    IF0H.1 = CLEAR;                     /* 要求フラグクリア */
    MK0H.1 = CLEAR;                     /* 割り込み許可     */
}

/*
    キャリア同期割り込み禁止
*/
static void
INTTWOUD_off(void)
{
    MK0H.1 = SET;                       /* 割り込み禁止 */
}

/* --------------------------------
    INTP5 割り込み許可
  -------------------------------- */
static void
INTP5_on(void) {
    PIF5 = CLEAR;
    PMK5 = CLEAR;
}

/* --------------------------------
    INTP5 割り込み禁止
  -------------------------------- */
static void
INTP5_off(void) {
```

```
    PMK5 = SET;
}

/* --------------------------------
    電流マイナー割り込み許可
  -------------------------------- */
static void
INTTM51_on(void) {
    IF1H.0 = CLEAR;
    MK1H.0 = CLEAR;
}


/* ========================================== */
/*
    割り込み処理
*/
/* ---------------------------------------------- */
__interrupt void
int_fault(void)
{
    err_flag = ERROR_OC;
    system_stop();              /* システム停止 */
}


/*
    INTP5
*/
__interrupt void
int_speed(void)
{
    capture_flag = FLG_ON;
}

/*
    キャリア同期割り込み
*/
static unsigned int
get_coswt(unsigned int wt)
{
    unsigned int   n;
    unsigned int   answer;
    unsigned char  dir, pm;


    n = wt / 90;
    if(n > 3){
        wt -= 360;
        n -= 4;
    };

    switch(n) {
    case 0:
        dir = 0xff;
        pm  = 0x00;
        break;
    case 1:
        wt -= 90;
        dir = 0x00;
        pm  = 0xff;
        break;
    case 2:
        wt -= 180;
        dir = 0xff;
        pm  = 0xff;
        break;
    default:
        wt -= 270;
        dir = 0x00;
        pm  = 0x00;
    };

    if(dir == 0) { MDAOL = sin_tbl[wt]; }
    else         { MDAOL = sin_tbl[sin_table_end - wt]; };

    DMUC0 = 0x81;
    while(DMUE == 1);

    if(pm == 0) { answer = PWM_MIDDLE + MDAOH;}
    else        { answer = PWM_MIDDLE - MDAOH;};

    return(answer);
}

static void
set_pwm(void)
{
    unsigned int    p, p_u, p_v;

#ifdef LOW
    if(cw_ccw_flag == CW) {
        p = ((60 * int_cnt)/angle) + DELTA_CW; /* 詳細な角度 進角は固定値 */
        switch(hall_new) {
```

```
        case 4: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

        case 5: /* 60 - 119 */
            p_u = p + 60;
            p_v = p_u + 240;
            break;

        case 1: /* 120 - 179 */
            p_u = p + 120;
            p_v = p_u + 240;
            break;

        case 3: /* 180 - 239 */
            p_u = p + 180;
            p_v = p_u + 240;
            break;

        case 2: /* 240 - 299 */
            p_u = p + 240;
            p_v = p_u + 240;
            break;

        case 6: /* 300 - 359 */
            p_u = p + 300;
            p_v = p_u + 240;
            break;

        default:
            p_u = 90;
            p_v = 90;
        };
    }else{
        p = ((60 * int_cnt)/angle) + DELTA_CCW; /* 詳細な角度 進角は固定値 */
        switch(hall_new){
        case 1: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

        case 5: /* 60 - 119 */
            p_u = p + 60;
            p_v = p_u + 240;
            break;

        case 4: /* 120 - 179 */
            p_u = p + 120;
            p_v = p_u + 240;
            break;

        case 6: /* 180 - 239 */
            p_u = p + 180;
            p_v = p_u + 240;
            break;

        case 2: /* 240 - 299 */
            p_u = p + 240;
            p_v = p_u + 240;
            break;

        case 3: /* 300 - 359 */
            p_u = p + 300;
            p_v = p_u + 240;
            break;

        default:
            p_u = 90;
            p_v = 90;
        };
    };
#else
    if(cw_ccw_flag == CW) {
        p = ((60 * int_cnt)/angle) + DELTA_CW; /* 詳細な角度 進角は固定値 */
        switch(hall_new) {
        case 5: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

        case 1: /* 60 - 119 */
            p_u = p + 60;
            p_v = p_u + 240;
            break;

        case 3: /* 120 - 179 */
            p_u = p + 120;
            p_v = p_u + 240;
            break;

        case 2: /* 180 - 239 */
            p_u = p + 180;
            p_v = p_u + 240;
```

```
            break;

        case 6: /* 240 - 299 */
            p_u = p + 240;
            p_v = p_u + 240;
            break;

        case 4: /* 300 - 359 */
            p_u = p + 300;
            p_v = p_u + 240;
            break;

        default:
            p_u = 90;
            p_v = 90;
        };
    }else{
        p = ((60 * int_cnt)/angle) + DELTA_CCW; /* 詳細な角度 進角は固定値 */
        switch(hall_new){
        case 3: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

        case 1: /* 60 - 119 */
            p_u = p + 60;
            p_v = p_u + 240;
            break;

        case 5: /* 120 - 179 */
            p_u = p + 120;
            p_v = p_u + 240;
            break;

        case 4: /* 180 - 239 */
            p_u = p + 180;
            p_v = p_u + 240;
            break;

        case 6: /* 240 - 299 */
            p_u = p + 240;
            p_v = p_u + 240;
            break;

        case 2: /* 300 - 359 */
            p_u = p + 300;
            p_v = p_u + 240;
            break;

        default:
            p_u = 90;
            p_v = 90;
        };
    };
#endif
    pwm_ff_u = get_coswt(p_u);
    pwm_ff_v = get_coswt(p_v);
    pwm_ff_w = PWM_x3 - (pwm_ff_u + pwm_ff_v);

    WDTE = WDTE_CLR;
}

__interrupt void
int_carrier(void)
{
    unsigned int     tmp_pwm;

    int_cnt++;
    print_cnt++;
    if(int_cnt > INT_CNT_MAX){                          /* オーバーフロー       */
        if(stop_wait != FLG_OFF){                       /* 停止待ち             */
            system_stop();                              /* システム停止         */
            return;
        }else if(cw_ccw_wait != FLG_OFF){               /* 反転の停止待ち       */
            system_restart();                           /* 再起動               */
            return;
        }else if(pwm_ff > START_PWM_FF_MAX){            /* モータが回転しない   */
            system_stop();                              /* システム停止         */
            err_flag = (ERROR_HALL | ERROR_MOTOR);
            return;
        }else{
            pwm_ff++;                                  /* 電圧を上げる         */
            int_cnt = 0;
            start_cnt = 0;
            MDB0 = (unsigned int)pwm_ff;
        }
    }
    pid_cnt++;
    start_cnt++;
    hall_new = read_Hall_IC();
    if(hall_new == next_type[cw_ccw_flag][hall_old]){  /* ホール IC の値が次の値 */
        hall_old  = hall_new;
        cnt_data1 = int_cnt;
        int_cnt   = 0;
```

```
        speed_flag = FLG_ON;
    }
    if(sys_flag == FLG_120){                        /* 120 度通電モード   */
        pid_cnt = 0;                                /* PID 制御を行わない */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            set_RTPM01(P_OFF);                      /* 通電停止 */
            sys_flag = FLG_WAIT;
        }else if((m_speed != 0) && (start_cnt > CHANGE_TIME)){
            set_RTPM01(P_OFF);                      /* 通電停止 */
            sys_flag = FLG_SW;
        }else{
            set_RTPM01(tr_sw[cw_ccw_flag][hall_new]);  /* 通電パターンを設定 */
            if(pwm_ff != old_ff) {
                set_TW0((unsigned int)pwm_ff,
                        (unsigned int)pwm_ff,
                        (unsigned int)pwm_ff,
                        (unsigned int)pwm_base);    /* デューティ変更  */
                old_ff = pwm_ff;
            }
        }
    }else if(sys_flag == FLG_180){                  /* 180 度通電モード */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            start_RTPM01();
            set_RTPM01(P_OFF);                      /* 通電停止 */
            sys_flag = FLG_WAIT;
        }else{
            set_pwm();
            set_TW0(pwm_ff_u,
                    pwm_ff_v,
                    pwm_ff_w,
                    (unsigned int)(pwm_base));
        }
    }else if(sys_flag == FLG_SW){ /* 120 度から 180 度に切り替わった直後 */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            sys_flag = FLG_WAIT;
        }else{
            end_RTPM01();                           /* リアルタイムポート出力終了 */
            tmp_pwm = (unsigned int)(pwm_base >> 1);
            set_TW0(tmp_pwm, tmp_pwm, tmp_pwm, pwm_base);
            sys_flag = FLG_180;
            pwm_ff = (pwm_ff/3)*2;                  /* 印加電圧を 2/3 に調整 */
            MDB0 = (unsigned int)pwm_ff;
        }
    };

    if(capture_flag == FLG_ON){
        if(CR00 != new_clk){
            old_clk      = new_clk;
            new_clk      = CR00;
            clk_flag     = FLG_ON;
            capture_flag = FLG_OFF;
        };
    }

    return;
}


/* --------------------------------
    Interrupt for TM51 overflow
  -------------------------------- */
__interrupt void
int_TM51(void) {
    unsigned int x_filt, reg;

    reg = ADCR;             /* 10 bit A/D reading */
    x_filt = reg>>6;
    I_measured = (float)x_filt;
}
```

4.18　参考プログラムのソース・ファイル

```
/*


    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット


    date    : 2007/05/21
    filename: main_mcio.c

    NEC Micro Systems,Ltd
*/

#pragma sfr

#include    "motor.h"
#include    "sub_mcio.h"

/*
    メイン関数
*/
void
main(void) {
    unsigned char  sw = 0;
    unsigned char  tmp_sw = 0;

    /* システム初期設定 */
    motor_init();
    init_PORT();
    init_TM50();

    /* モータパラメータの設定 */
    motor_pset(PID_INTERVAL, 150);  /* PID 制御間隔 150ms */

    motor_pset(KRP,  50);           /* RPM-Kp の設定。1000 倍した値を指定する。             */
    motor_pset(KRI,  20);           /* RPM-Ki の設定。1000 倍した値を指定する。             */
    motor_pset(KRD,  10);           /* RPM-Kd の設定。1000 倍した値を指定する。             */

    startup_disp();
    while(1){
        clear_WDTM();
        vol2speed();
        speed_print(200);
        sw = get_sw();
        if(sys_flag != FLG_OFF){
            if((cw_ccw_wait == FLG_OFF) &&
               (stop_wait   == FLG_OFF)) {
                if(sw != tmp_sw) {
                    switch(sw){
                    case STOP_TR:    motor_stop();       break;
                    case FORWARD_TR: motor_rotation(CW);  break;
                    case REVERSE_TR: motor_rotation(CCW); break;
                    default: ;
                    };
                };
            };
            motor_pid();
        } else {                                /* 停止中     */
            if(sw != tmp_sw) {
                switch(sw){
                case START_TR: motor_start(); break;
                case MODE_TR:
                    if(ad_flag == FLG_ON){      /* 機能停止中 */
                        ad_flag = FLG_OFF;      /* 機能復帰   */
                    }else{                      /* 機能動作中 */
                        ad_flag = FLG_ON;       /* 機能停止   */
                    };
                    break;
                default: ;
                };
            };
        };
        tmp_sw = sw;
    };
    return;
}
```

```
/*


    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応
```

```
    target  : uPD78F0714 モータ・スタータ・キット


    date    : 2007/05/22
    filename: sub_mcio.h

    NEC Micro Systems,Ltd
*/

#define  CLEAR       0
#define  SET         1

#define  IN          1    /* 入力 */
#define  OUT         0    /* 出力 */

#define  FLG_ON      1    /* 有効 */
#define  FLG_OFF     0    /* 無効 */

#define  KEY_WAIT    10
#define  SW          (P7&0x0f)
#define  SW2         PM73
#define  SW3         PM72
#define  SW4         PM71
#define  SW5         PM70

#define  LD_LED0     PM64
#define  LD_LED1     PM65
#define  LD_LED2     PM66
#define  LD_LED3     PM67

#define  LD_DATA     PM4

#define  START_SW    0x7
#define  STOP_SW     0x7
#define  FORWARD_SW  0xb
#define  REVERSE_SW  0xd
#define  MODE_SW     0xe

#define  START_TR    0x01   /* 押されているスイッチ判定  */
#define  STOP_TR     0x02
#define  FORWARD_TR  0x04
#define  REVERSE_TR  0x08
#define  MODE_TR     0x10

#define  LED_0       0xc0   /* LED 表示用データ  */
#define  LED_1       0xf9
#define  LED_2       0xa4
#define  LED_3       0xb0
#define  LED_4       0x99
#define  LED_5       0x92
#define  LED_6       0x82
#define  LED_7       0xf8
#define  LED_8       0x80
#define  LED_9       0x98
#define  LED_O       0xc0   /* O.C.             */
#define  LED_C       0xc6
#define  LED_I       0xcf   /* I                */
#define  LED_H       0x89   /* HALL             */
#define  LED_A       0x88
#define  LED_L       0xc7
#define  LED_        0xff   /* " "              */
#define  LED_S       0x92   /* SELF             */
#define  LED_E       0x86
#define  LED_F       0x8e
#define  LED_P       0x8c   /* PC               */
#define  LED_dot     0x7f   /* .                */

/* ------------------------------------------------------------ */
extern unsigned char ad_flag;

/* ------------------------------------------------------------ */
extern unsigned char get_sw(void);
extern void          speed_print(unsigned int);

extern void          init_PORT(void);
extern void          init_TM50(void);
extern void          vol2speed(void);
extern void          startup_disp(void);

extern void          clear_WDTM(void);
/* ------------------------------------------------------ EOF -- */
```

```
/*

    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット
```

```
    date    : 2007/05/22
    filename: sub_mcio.c

    NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma stop
#pragma ei
#pragma di

#include "sub_mcio.h"
#include "motor.h"


#define  PRINT_INTERVAL  10     /* 100[us] * 10 = 1[ms] */

/*
    スタティック変数
*/
static const unsigned char led_data[10] = {          /* 数字表示用   */
    LED_0, LED_1, LED_2, LED_3, LED_4,
    LED_5, LED_6, LED_7, LED_8, LED_9
};

static void          led_set(unsigned char, unsigned char);
static void          led_print(int, char);
static void          wait(int);
static void          start_TM50(void);
static void          wait_TM50(void);
static void          print_error(char);
static unsigned int  get_vol(char);

unsigned char        ad_flag;

/* ================================================== */
/*
    エラー表示
*/
static void
print_error(char eno) {
    switch(eno){
    case ERROR_HALL:
        led_set(0, LED_H);
        led_set(1, LED_A);
        led_set(2, LED_L);
        led_set(3, LED_L);
        break;

    case ERROR_OC:
        led_set(0, LED_);
        led_set(1, LED_O & LED_dot);
        led_set(2, LED_C & LED_dot);
        led_set(3, LED_);
        break;

    case ERROR_MOTOR:
        led_set(0, LED_F);
        led_set(1, LED_A);
        led_set(2, LED_I);
        led_set(3, LED_L);
        break;

    case ERROR_S_OC:
        led_set(0, LED_S & LED_dot);
        led_set(1, LED_);
        led_set(2, LED_O & LED_dot);
        led_set(3, LED_C & LED_dot);
        break;

    default:
        led_set(0, LED_F);
        led_set(1, LED_A);
        led_set(2, LED_I);
        led_set(3, LED_L);
    };

    STOP();
}


/*
   MC-10 上のスイッチの読み込み
*/
unsigned char
get_sw(void)
{
    unsigned char data;

    int  i;
    unsigned char tmp;
    static unsigned char start_stop_flag = FLG_OFF;
```

```
    if(err_flag != ERROR_NONE){
        print_error(err_flag);
    };

    data = SW;                          /* スイッチ読み込み                    */
    if(data != 0xf){
        for(i=0;i<KEY_WAIT;){           /* チャタリング除去 要調整             */
            tmp = SW;                   /* スイッチ再読み込み                  */
            if(data == tmp){
                i++;
            }else{
                i = 0;
                data = tmp;
            }
            wait(2);
        }
        if(sys_flag != FLG_OFF){
            switch(data){
            case STOP_SW:
                if(start_stop_flag == FLG_OFF){
                    data = STOP_TR;
                    start_stop_flag = FLG_ON;
                }
                break;

            case FORWARD_SW: data = FORWARD_TR; break;
            case REVERSE_SW: data = REVERSE_TR; break;
            case MODE_SW:    data = MODE_TR;    break;
            default:
                ;
            }
        }else{
            if((data == START_SW) && (start_stop_flag == FLG_OFF)){
                data = START_TR;
                start_stop_flag = FLG_ON;
            }else if(data == MODE_SW){
                data = MODE_TR;
            };
        };
    };
    if(data == 0xf){    /* スイッチが何も押されていない */
        start_stop_flag = FLG_OFF; /* START/STOP のトグル防止 */
    };

    return(data);
}


/*
    速度表示
*/
void
speed_print(unsigned int ms)
{
    if((MODE_SW == SW) || (sys_flag == FLG_OFF)){
        led_print(speed_ref_o, ad_flag);
    }else{
        if(print_cnt > (ms*PRINT_INTERVAL)){
            led_print(m_speed, FLG_OFF);
            print_cnt = 0;
        };
    };
}


/*
    7 セグ LED に値を表示する
*/
static void
led_print(int data, char flag)
{
    unsigned int tmp;
    int flg = FLG_OFF;

    tmp = (unsigned int)(data / 1000);
    if(tmp != 0){                       /* 最上位が 0 でない */
        flg = FLG_ON;                   /* 最上位に数値を表示した */
        led_set(0, led_data[tmp]);
    }else{
        led_set(0, LED_);
    };
    data %= 1000;
    tmp = (unsigned int)(data / 100);
    if((tmp != 0) || (flg == FLG_ON)){      /* 値が 0 でない or すでに数字を表示した */
        flg = FLG_ON;
        led_set(1, led_data[tmp]);
    }else{
        led_set(1, LED_);
    };
    data %= 100;
    tmp = (unsigned int)(data / 10);
    if((tmp != 0) || (flg == FLG_ON)){      /* 値が 0 でない or すでに数字を表示した */
        led_set(2, led_data[tmp]);
    }else{
```

```
        led_set(2, LED_);
    };
    data %= 10;
    if(flag == FLG_OFF){
        led_set(3, led_data[data]);
    }else{
        led_set(3, (unsigned char)(led_data[data])&LED_dot);
    };
}

/*
    7 セグ LED にデータを表示する
    no:   表示する LED の場所
    data: 出力するデータ
*/
static void
led_set(unsigned char no, unsigned char data)
{
    unsigned char p;

    P6 = 0x00;
    P4 = data;

    switch(no){                         /* 表示する場所 */
    case 0:  p = 0x80; break;           /* 左端                                 */
    case 1:  p = 0x40; break;
    case 2:  p = 0x20; break;
    default: p = 0x10; break;           /* 右端                                 */
    };
    P6 = p;
}

/*
    時間調整    ms
*/
static void
wait(int cnt)
{
    int i;

    for(i=0;i<cnt;i++){
        start_TM50();               /* タイマスタート                       */
        wait_TM50();                /* 割り込み要求発生待ち                 */
        clear_WDTM();               /* ウォッチドック・タイマのクリア       */
    };
    return;
}

/*
  スタートアップ表示
*/
void
startup_disp(void) {
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(2000);
}


/*
    速度指定可変抵抗の電圧を読み出し
    指定速度に変換する
*/
#define SHIFT_BIT   2
void
vol2speed(void)
{
    unsigned int  data;
    unsigned long tmp;

    if(ad_flag == FLG_OFF){
        data = get_vol(SHIFT_BIT);
        data = ((data - 3) * 12) + MINSPEED_REF;
        if(data > MAXSPEED_REF){
            data = MAXSPEED_REF;
        }else if(data < MINSPEED_REF) {
            data = MINSPEED_REF;
        };
        tmp = (UNIT_RPM / (unsigned long)data);
        speed_ref_o = (int)(UNIT_RPM / tmp);
    }
}

static unsigned int
get_vol(char s_bit)
{
    unsigned int  data;
    unsigned char ads_backup;

    DI();
    ads_backup = ADS;
    ADS = AD_VOL;
```

```
    ADIF = CLEAR;
    while(ADIF != SET);
    data = ADCR;
    EI();
    ADS = ads_backup;
    ADIF = CLEAR;

    return((data>>(s_bit+6))&0x3ff);
}


/* ---------------------------------------------------------- */
/*
    ポートの設定
*/
void
init_PORT(void) {
    SW2 = IN;                   /* START/STOP              */
    SW3 = IN;                   /* FORWARD                 */
    SW4 = IN;                   /* REVERSE                 */
    SW5 = IN;                   /* MODE                    */

    LD_LED0 = OUT;              /* LED 選択 出力 */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;              /* LED 表示データ 出力 */
}

/*
    8 ビット・タイマ 50
*/
void
init_TM50(void)
{
    TCL50 = 0x06;
    CR50  = 78;                         /* 1ms */
}

static void
start_TM50(void)
{
    TMIF50 = CLEAR;                     /* 割り込み通知フラグクリア     */
    TCE50  = SET;                       /* タイマ開始                   */
}

static void
wait_TM50(void)
{
    while(TMIF50 != SET);               /* 割り込み通知待ち             */
    TCE50  = CLEAR;                     /* タイマ停止                   */
}

void
clear_WDTM(void)
{
    WDTE = WDTE_CLR;
}
```

付録A　プログラム例

別途提供するモータ操作パネル(GUI プログラム)とモータ・コントロール I/O ボード上の RS-232C 端子を接続してこのシステムをコントロールする場合のプログラム例を添付しました。

A.1　GUI用参考プログラムの関数一覧

表 A－1　GUI 用参考プログラム関数一覧

関数名	機能	目的
uart_set()	UART設定	UART機能の設定を行う。
get_uart()	コマンド読み込み指示	UART通信での通信データの読み込みを指示し、読み込んだデータに対応した動作情報を返す。
uart_read()	UARTデータの読み込み	UART受信データの読み込みを指示する。
uart_wait()	UARTデータの受信待機付き読み込み	UART通信の多バイト構成コマンド連続受信用関数。
uart_send()	UARTデータの送信	UART通信の送信を指示する。
int_uart()	UART受信	UART通信の受信を行う割り込み関数。
set_error()	エラー状態格納	UART通信の通信データにエラー状態を格納する。
led_set()	LED表示	LEDへデータを表示する。
startup_disp()	起動時のLED表示	起動時のLED表示
init_PORT()	ポートの設定	MC-IOボード上のポート設定を行う。

A.2　GUI用参考プログラムの定数一覧

A.2.1 内部定数

表 A－2　GUI用参考プログラムの内部定数一覧(1/2)

名称	意味	設定値	備考
IN	ポート設定用	1	ポート機能指定に使用
OUT	ポート設定用	0	
CLEAR	レジスタ・ビット設定用	0	レジスタのビット・アクセスで使用
SET	レジスタ・ビット設定用	1	
LED_0	LED表示データ	0xc0	“0”を表示
LED_1		0cf9	“1”を表示
LED_2		0xa4	“2”を表示
LED_3		0xb0	“3”を表示
LED_4		0x99	“4”を表示
LED_5		0x92	“5”を表示
LED_6		0x82	“6”を表示
LED_7		0xf8	“7”を表示
LED_8		0x80	“8”を表示
LED_9		0x98	“9”を表示
LED_O		0xc0	“O”の代わりに“0”を表示
LED_I		0xcf	“I”を表示
LED_C		0xc6	“C”を表示
LED_H		0x89	“H”を表示
LED_A		0x88	“A”を表示
LED_L		0xc7	“L”を表示
LED_		0xff	“ ”を表示
LED_S		0x92	“S”を表示
LED_E		0x86	“E”を表示
LED_F		0x8e	“F”を表示
LED_P		0x8c	“P”を表示
LED_dot		0x7f	“.”を表示
LD_LED0	ポート制御レジスタ	PM64	LED選択用ポート
LD_LED1		PM65	LEDへのデータ出力ポート
LD_LED2		PM66	
LD_LED3		PM67	
LD_DATA		PM4	
START_TR	状態設定用	0x01	制御開始
STOP_TR		0x02	制御停止
FORWARD_TR		0x04	回転をCWに変更
REVERSE_TR		0x08	回転をCCWに変更
MODE_TR		0x10	MODEスイッチが押された状態
RTS	通信ポート	P11	
CTS		P10	

表 A－2　GUI用参考プログラムの内部定数一覧(2/2)

名称	意味	設定値	備考
RX_BUFF_SIZE	通信バッファ・サイズ	6	
MD_ERROR_HALL	通信情報	0xF0	
MD_ERROR_OC		0xF1	
MD_ERROR_MOTOR		0xF2	
MD_CMD_START	通信コマンド	0x20	
MD_CMD_STOP		0x21	
MD_CMD_RESET		0x2f	
MD_CMD_GETID		0x10	
MD_CMD_GETVER		0x11	
MD_CMD_SETSSPEED		0x30	
MD_CMD_GETSSPEED		0x31	
MD_CMD_SETPIDPARAM		0x40	
MD_CMD_GETPIDPARAM		0x41	
MD_CMD_GETPIDI		0x42	
MD_CMD_GETPIDV		0x43	
MD_CMD_SETPIDI		0x44	
MD_CMD_SETPIDV		0x45	
MD_CMD_GETSPEED		0x50	
MD_CMD_GET_I_SHNT		0x60	
MD_CMD_GET_PWM		0x61	
MD_CMD_GET_I_CMD		0x62	
MD_CMD_SETICMD		0x70	
MD_CMD_SETVCMD		0x71	
MD_CMD_SETSTART		0x72	
MD_CMD_GETSTART		0x73	
MD_CMD_SETLIMIT		0x74	
MD_CMD_GETLIMIT		0x75	
MY_ID		0x02	
VER_MAJOR		0x01	
VER_MINOR		0x00	
VER_SEQ		0x01	

A.3　GUI用参考プログラムの変数一覧

A.3.1 内部変数

表 A－3　GUI用参考プログラムの内部変数一覧

ad_flag	char	速度変更フラグ	指定速度変更機能を制限
led_data[ ]	char	LED出力データ	LEDに表示する数値

A.4　GUI用参考プログラムのソース・プログラム

```
/*

    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット
              コンパイラオプション  GUI:GUI 使用
    date    : 2007/05/22
    filename: main_gui.c

    NEC Micro Systems,Ltd
*/

#pragma sfr

#include    "motor.h"
#include    "sub_gui.h"

/*
    メイン関数
*/
void
main(void) {
    unsigned char  sw = 0;
    unsigned char  tmp_sw = 0;

    /* システム初期設定 */
    motor_init();
    init_PORT();
    uart_set();
    startup_disp();

    /* モータパラメータの設定 */
    motor_pset(PID_INTERVAL, 150);  /* PID 制御間隔 150ms */

    while(1){
        clear_WDTM();
        sw = get_uart();
        if(sys_flag != FLG_OFF) {
            if((cw_ccw_wait == FLG_OFF) &&
               (stop_wait   == FLG_OFF)){
                if(sw != tmp_sw) {
                    switch(sw){
                    case STOP_TR:    motor_stop();        break;
                    case FORWARD_TR: motor_rotation(CW);  break;
                    case REVERSE_TR: motor_rotation(CCW); break;
                    default: ;
                    };
                };
            };
            motor_pid();
        } else {
            if(sw != tmp_sw) {
                switch(sw){
                case START_TR: motor_start(); break;
                default: ;
                };
            };
        };
        tmp_sw = sw;
    };
    return;
}
```

```
/*

    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応
```

```
    target  : uPD78F0714 モータ・スタータ・キット
              コンパイラオプション  GUI:GUI 使用

    date    : 2007/05/22
    filename: sub_gui.h

    NEC Micro Systems,Ltd
*/


#define IN                  1               /* 入力 */
#define OUT                 0               /* 出力 */

#define CLEAR               0
#define SET                 1

#define START_TR    0x01
#define STOP_TR     0x02
#define FORWARD_TR  0x04
#define REVERSE_TR  0x08
#define MODE_TR     0x10

#define FLG_ON          1               /* 有効 */
#define FLG_OFF         0               /* 無効 */

#define MD_ERROR_HALL       0xF0            /* ホール IC 異常     */
#define MD_ERROR_OC         0xF1            /* 過電流             */
#define MD_ERROR_MOTOR      0xF2            /* モータ異常         */

#define RTS                 P11
#define CTS                 P10
#define RX_BUFF_SIZE        6               /* UART00 の受信バッファサイズ   */

#define MD_CMD_START        0x20            /* START              */
#define MD_CMD_STOP         0x21            /* STOP               */
#define MD_CMD_RESET        0x2f            /* RESET              */
#define MD_CMD_GETID        0x10            /* ID 要求             */
#define MD_CMD_GETVER       0x11            /* バージョン要求   */
#define MD_CMD_SETSSPEED    0x30            /* 指定速度変更     */
#define MD_CMD_GETSSPEED    0x31            /* 指定速度読み出し */
#define MD_CMD_SETPIDPARAM  0x40            /* PID 変更           */
#define MD_CMD_GETPIDPARAM  0x41            /* PID 読み出し       */
#define MD_CMD_GETPIDI      0x42
#define MD_CMD_GETPIDV      0x43
#define MD_CMD_SETPIDI      0x44
#define MD_CMD_SETPIDV      0x45
#define MD_CMD_GETSPEED     0x50            /* 実速度読み出し   */

#define MD_CMD_GET_I_SHNT   0x60
#define MD_CMD_GET_PWM      0x61
#define MD_CMD_GET_I_CMD    0x62
#define MD_CMD_SETICMD      0x70
#define MD_CMD_SETVCMD      0x71

#define MD_CMD_SETSTART     0x72
#define       MD_CMD_GETSTART     0x73
#define MD_CMD_SETLIMIT     0x74
#define MD_CMD_GETLIMIT     0x75

#define MY_ID               0x02            /* ファーム識別 ID   */
#define VER_MAJOR           0x01            /* バージョン情報 */
#define VER_MINOR           0x00
#define VER_SEQ             0x01

#define LD_LED0             PM64
#define LD_LED1             PM65
#define LD_LED2             PM66
#define LD_LED3             PM67

#define LD_DATA             PM4

#define LED_0               0xc0            /* LED 表示用データ                    */
#define LED_1               0xf9
#define LED_2               0xa4
#define LED_3               0xb0
#define LED_4               0x99
#define LED_5               0x92
#define LED_6               0x82
#define LED_7               0xf8
#define LED_8               0x80
#define LED_9               0x98
#define LED_O               0xc0            /* O-C                                 */
#define LED_I               0xcf
#define LED_C               0xc6
#define LED_H               0x89            /* HALL                                */
#define LED_A               0x88
#define LED_L               0xc7
#define LED_                0xff
#define LED_S               0x92            /* SELF                                */
#define LED_E               0x86
#define LED_F               0x8e
#define LED_P               0x8c            /* PC                                  */
#define LED_dot             0x7f
```

```
/* ------------------------------------------------------------ */
extern unsigned char get_uart(void);

extern void          clear_WDTN(void);
extern void          uart_set(void);
extern void          init_PORT(void);
extern void          startup_disp(void);

/* ------------------------------------------------------ EOF -- */
```

```
/*


    PMS モータ 180 度通電方式 位置センサ(ホール IC)

    速度推定精度改良版（タイマ値キャプチャ方式）
    モジュール化対応

    target  : uPD78F0714 モータ・スタータ・キット
              コンパイラオプション  GUI：GUI 使用
    date    : 2007/05/22
    filename: sub_gui.c

    NEC Micro Systems,Ltd
*/

#pragma sfr


#pragma INTERRUPT INTSR00       int_uart  rb1   /* UART00 コマンド受信用     */


#include "sub_gui.h"
#include "motor.h"


/*
    スタティック変数
*/
static const unsigned char led_data[10] = {         /* 数字表示用   */
    LED_0, LED_1, LED_2, LED_3, LED_4,
    LED_5, LED_6, LED_7, LED_8, LED_9
};

static unsigned char uart00_data[RX_BUFF_SIZE];     /* UART00 受信バッファ          */
static unsigned char read_p, write_p;               /* バッファポインタ             */
static char          uart_err_flag;

static unsigned char uart_read(void);
static unsigned char uart_wait(void);
static void          uart_send(unsigned char);
static void          led_set(unsigned char, unsigned char);
static void          set_error(char);

#define WDTE_RESET      0x00

/* ============================================================ */
/* ------------------------------------------------------------ */
/*
    UART00 の設定
*/
void
uart_set(void) {
    PM10    = IN;
    PM11    = OUT;
    PM13    = IN;
    PM14    = OUT;
    RTS     = 1;
    P14     = 1;
    BRGC00  = 0x56;                 /* 115200 */
    PS001   = CLEAR;
    PS000   = CLEAR;
    CL00    = SET;                  /* 8bit */
    SL00    = CLEAR;
    POWER00 = SET;
    TXE00   = SET;
    RXE00   = SET;
    STIF00  = SET;
    SRIF00  = CLEAR;
    SRMK00  = CLEAR;                /* 受信割り込み許可 */
    RTS     = 0;
    read_p  = 0;
    write_p = 0;
}

/*
    UART 通信による動作命令の取得
 */
unsigned char
get_uart(void) {
    unsigned char data;
```

```
    int           ss, ref;
    unsigned char hi, lo;
    float         shunt_volt;
    float         shunt_command;
    float         tmp_float;
    int           tmp_int;

    /* convenient macros */
#define GET16to(thevar)            \
        lo = uart_wait();           \
        hi = uart_wait();           \
        thevar = (((int)(hi))<<8) + lo;

#define SETUART(thevar)                              \
        uart_send((char)(((int)(thevar))&0xff));      \
        uart_send((char)((((int)(thevar))>>8)&0xff));

    if(err_flag != ERROR_NONE) {
        set_error(err_flag);
    };

    data = uart_read();
    switch(data){
    case MD_CMD_GETID:                  /* Get ID */
        uart_send(data);
        uart_send(MY_ID);
        break;

    case MD_CMD_GETVER:                 /* Get Version */
        uart_send(data);
        uart_send(VER_MAJOR);
        uart_send(VER_MINOR);
        uart_send(VER_SEQ);
        break;

    case MD_CMD_START:                  /* Start */
        uart_send(data);
        data = START_TR;
        break;

    case MD_CMD_STOP:                   /* Stop */
        uart_send(data);
        data = STOP_TR;
        break;

    case MD_CMD_RESET:                  /* Reset */
        uart_send(data);
        while(STIF00 != SET);
        WDTM = WDTE_RESET;
        break;

    case MD_CMD_SETSSPEED:              /* Set Setting Speed */
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){                  /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CW;
            }
        }
        speed_ref_o = ss;
        motor_pset(MODE, SPEED_CMD);   /* speed control mode */
        break;

    case MD_CMD_SETICMD:
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){                  /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
```

```
                cw_ccw_flag = CW;
            }
        }
        I_ref_o = (float)ss;
        speed_ref = m_speed;            /* open loop on speed */
        motor_pset(MODE, I_CMD);        /* current control mode */
              break;

    case MD_CMD_SETVCMD:
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){                  /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CW;
            }
        }
        pwm_ff_o = ss;
        I_ref = I_measured;
        speed_ref = m_speed;            /* open loop on speed */
        motor_pset(MODE, V_CMD);        /* voltage control mode */
              break;

    case MD_CMD_GETSSPEED:              /* Get Setting Speed */
        uart_send(data);
        ss = speed_ref;
        if(cw_ccw_flag == CCW){
            ss = ~ss + 1;
        }
#if 1
        lo = (unsigned char)((unsigned int)(ss)&0xff);
        hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
        uart_send(lo);
        uart_send(hi);
#else
        SETUART(ss);
#endif
        break;

    case MD_CMD_GET_I_SHNT:             /* Get shunt a/d (current) */
        shunt_volt = I_measured*10.0;   /* 10 bits */
        uart_send(data);
#if 1
        ss = (int)shunt_volt;
        lo = (unsigned char)(ss&0xff);
        hi = (unsigned char)(ss>>8)&0xff;
        uart_send(lo);
        uart_send(hi);
#else
        SETUART(shunt_volt);
#endif
        break;

    case MD_CMD_GET_I_CMD:              /* Get shunt a/d (current) */
        uart_send(data);
        shunt_command = I_ref*10.0;
#if 1
        ss = (int)shunt_command;        /* 10 bits (TEMPORARY)     */
        lo = (unsigned char)(ss&0xff);
        hi = (unsigned char)(ss>>8)&0xff;
        uart_send(lo);
        uart_send(hi);
#else
        SETUART(shunt_command);
#endif
        break;

    case MD_CMD_GET_PWM:                /* Get PWM value */
        uart_send(data);
        ss = pwm_ff;
        if (ss > 1000) ss=1000;         /* overflow prevention for flag space */
        if (ss < 0) ss=0;               /* overflow prevention for flag space */
        lo = (unsigned char)((unsigned int)(ss)&0xff);
        hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
        /*
                note: this is a 10 bit variable, so the upper bits are used
                      to transmit the maxed_flags variable.
        */
        hi |= maxed_flags;              /* use bits 7,6,5 for flags  */
        uart_send(lo);
        uart_send(hi);
        break;
```

```
    case MD_CMD_SETPIDI:
        GET16to(ref);
        kip_ref = (float)ref/1000.0;
        GET16to(ref);
        kii_ref = (float)ref/1000.0;
        GET16to(ref);
        kid_ref = (float)ref/1000.0;
        uart_send(data);
        break;

    case MD_CMD_SETPIDV:
        GET16to(ref);
        krp_ref = (float)ref/1000.0;
        GET16to(ref);
        kri_ref = (float)ref/1000.0;
        GET16to(ref);
        krd_ref = (float)ref/1000.0;
        uart_send(data);
        break;

    case MD_CMD_GETPIDI:            /* PID for current control */
        uart_send(data);
        uart_send((char)(((int)(kip_ref*1000))&0xff));
        uart_send((char)((((int)(kip_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kii_ref*1000))&0xff));
        uart_send((char)((((int)(kii_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kid_ref*1000))&0xff));
        uart_send((char)((((int)(kid_ref*1000))>>8)&0xff));
        break;

    case MD_CMD_GETPIDV:            /* PID for current control */
        uart_send(data);
        uart_send((char)(((int)(krp_ref*1000))&0xff));
        uart_send((char)((((int)(krp_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kri_ref*1000))&0xff));
        uart_send((char)((((int)(kri_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(krd_ref*1000))&0xff));
        uart_send((char)((((int)(krd_ref*1000))>>8)&0xff));
        break;

    case MD_CMD_GETSPEED:               /* 回転速度読み出し */
        if(uart_err_flag != ERROR_NONE){
            uart_send(uart_err_flag);
            err_flag = ERROR_NONE;
            uart_err_flag = ERROR_NONE;
        }else{
            uart_send(data);
            if(m_speed == 0){
                uart_send(0);
                uart_send(0);
            }else{
                ss = m_speed;
                if(cw_ccw_wait == FLG_OFF){
                    if(cw_ccw_flag == CCW){
                        ss = ~ss + 1;
                    }
                }else{  /* 反転停止待ち */
                    if(cw_ccw_flag == CW){
                        ss = ~ss + 1;
                    }
                }
#if 1
                uart_send((unsigned char)((unsigned int)(ss)&0xff));
                uart_send((unsigned char)(((unsigned int)(ss)>>8)&0xff));
#else
                SETUART(ss);
#endif
            }
        }
        break;

    case MD_CMD_SETSTART:             /* open-loop startup parameters defined by gui*/
        GET16to(ref);
        startup_method = (unsigned char)(ref);
        GET16to(ref);
        t_knee = (float)ref/1000.0;
        GET16to(ref);
        t_end = (float)ref/1000.0;
        GET16to(RPM0);
        GET16to(RPM1);
        GET16to(RPM2);
        GET16to(I_ref0);
        GET16to(I_ref1);
        GET16to(I_ref2);
        GET16to(PWM0);
        GET16to(PWM1);
        GET16to(PWM2);
        uart_send(data);
        break;

    case MD_CMD_GETSTART:            /* send startup parameters to gui*/
        uart_send(data);
        uart_send(startup_method);
```

```
        uart_send(0x00);            /* char sent as int for convenience in gui */
        SETUART(t_knee*1000);
        SETUART(t_end*1000);
        SETUART(RPM0);
        SETUART(RPM1);
        SETUART(RPM2);
        SETUART(I_ref0);
        SETUART(I_ref1);
        SETUART(I_ref2);
        SETUART(PWM0);
        SETUART(PWM1);
        SETUART(PWM2);
        break;

    case MD_CMD_SETLIMIT:              /* A/D gains and rate limits */
        GET16to(ref);
        adgain = (float)ref/100.0;
        GET16to(adoffset);
        GET16to(ref);
        maxcurrent = (float)ref;
        GET16to(ref);
        mincurrent = (float)ref;
        GET16to(ref);
        I_rate_max = (float)ref/10.;
        GET16to(maxspeed);
        GET16to(minspeed);
        GET16to(ref);
        RPM_rate_max = (float)ref;

        dI_ref_max = I_rate_max*0.1;       /* dt of 0.1 sec, this is the max change per cycle */
        dspeed_ref_max = (int)(RPM_rate_max*0.1); /* dt of 0.1 sec, this is the max change per cycle */
        uart_send(data);
        break;

    case MD_CMD_GETLIMIT:             /* A/D gains and rate limits */
        uart_send(data);
        SETUART(adgain*100);
        SETUART(adoffset);
        SETUART(maxcurrent);
        SETUART(mincurrent);
        SETUART(I_rate_max*10.);
        SETUART(maxspeed);
        SETUART(minspeed);
        SETUART(RPM_rate_max);
        break;

    default:
        ;
    };

    return(data);
}

/*
    UART00 の受信バッファデータを返す
*/
static unsigned char
uart_read(void) {
    unsigned char data = 0;

    if(read_p != write_p){              /* 受信データ有り           */
        data = uart00_data[read_p++];   /* データを取り出す         */
        read_p %= 6;                    /* 読み出しポインタを更新   */
    }
    return(data);
}

/*
    UART00 の受信待ち
*/
static unsigned char
uart_wait(void) {
    unsigned char data;

    while(read_p == write_p);           /* バッファにデータが入るまで待つ   */
    data = uart00_data[read_p++];       /* データを取り出す                 */
    read_p %= 6;                        /* 読み出しポインタを更新           */
    return(data);
}

/*
    UART00 から送信
*/
static void
uart_send(unsigned char data) {
    while(STIF00 != SET);               /* 送信完了待ち                     */
    STIF00 = CLEAR;
    TXS00 = data;                       /* 送信                             */
}

/*
    UART00 のコマンド受信用
*/
__interrupt void
```

```
int_uart(void) {
    unsigned char tmp;

    tmp = ASIS00;                       /* エラー・ステータス読み出し   */
    uart00_data[write_p++] = RXB00;     /* 受信バッファに格納           */
    write_p %= RX_BUFF_SIZE;            /* 書き込みポインタ更新         */
}


/* ---------------------------------------------------------- */
/*
    エラー表示
*/
static void
set_error(char eno) {

    switch(eno){
    case ERROR_HALL:
        uart_err_flag = MD_ERROR_HALL;
        led_set(0, LED_H);
        led_set(1, LED_A);
        led_set(2, LED_L);
        led_set(3, LED_L);
        break;

    case ERROR_OC:
        uart_err_flag = MD_ERROR_OC;
        led_set(0, LED_);
        led_set(1, LED_O & LED_dot);
        led_set(2, LED_C & LED_dot);
        led_set(3, LED_);
        break;

    case ERROR_MOTOR:
        uart_err_flag = MD_ERROR_MOTOR;
        led_set(0, LED_F);
        led_set(1, LED_A);
        led_set(2, LED_I);
        led_set(3, LED_L);
        break;

    case ERROR_S_OC:
        uart_err_flag = MD_ERROR_OC;
        led_set(0, LED_S & LED_dot);
        led_set(1, LED_);
        led_set(2, LED_O & LED_dot);
        led_set(3, LED_C & LED_dot);
        break;

    default:
        uart_err_flag = MD_ERROR_MOTOR;
        led_set(0, LED_F);
        led_set(1, LED_A);
        led_set(2, LED_I);
        led_set(3, LED_L);
    };
}

/*
    8 セグ LED にデータを表示する
    no:   表示する LED の場所
    data: 出力するデータ
*/
static void
led_set(unsigned char no, unsigned char data) {
    unsigned char p;

    P6 = 0x00;
    P4 = data;

    switch(no){                         /* 表示する場所 */
    case 0:  p = 0x80; break;           /* 左端                         */
    case 1:  p = 0x40; break;
    case 2:  p = 0x20; break;
    default: p = 0x10; break;           /* 右端                         */
    };
    P6 = p;
}

/*
  スタートアップ表示
*/
void
startup_disp(void) {
    led_set(0, LED_P);
    led_set(1, LED_C);
    led_set(2, LED_);
    led_set(3, LED_);
}

/*
    ウォッチドッグタイマ
*/
void
```

```
clear_WDTM(void) {
    WDTE = WDTE_CLR;
}

/* ------------------------------------------------------- */
/*
    ポートの設定
*/
void
init_PORT(void) {
    LD_LED0 = OUT;                  /* LED 選択 出力 */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;                  /* LED 表示データ 出力 */
}
```

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL(アドレス)　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。