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

## センサレス（BEMF）による120度通電方式編

μPD78F0714

資料番号　U18051JJ2V0AN00（第2版）
発行年月　June 2007 NS

〔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5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。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等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。

M8E　02.11

# 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のタイマ／カウンタ機能のシステム例としてPWM出力，コンパレータを使用したセンサレス（BEMF）による120度通電方式のモータ制御を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
・システム概要
・BLDCモータ制御の原理
・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ピュータ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 | 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）による120度通電方式編 | このマニュアル | 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 |
| SM+　システム・シミュレータ | 操作編 | U17246J | U17246E |
| | ユーザ・オープン・インタフェース編 | U17247J | U17247E |
| ID78K0-QB Ver.2.81　統合デバッガ | 操作編 | U16996J | U16996E |
| 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-78K0KX1H　インサーキット・エミュレータ | U17081J | U17081E |

**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-FP3　フラッシュ・メモリ・プログラマ　ユーザーズ・マニュアル | U13502J | U13502E |
| 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

このシステムはブラシレスDCモータ（Brush Less DC Motor：以降BLDCモータ）を120度通電方式で駆動します。

- このシステムは 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，センサレスで BLDC モータを 120 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。

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（サンプル・プログラム）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)
  - 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 W3.60
- RA78K0 アセンブラ W3.70
- DF0714.78K デバイスファイル V1.10

## 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.3.60 C **コンパイラ　各ユーザーズ・マニュアル**
- RA78K0 Ver.3.70 **アセンブラ・パッケージ　各ユーザーズ・マニュアル**

# 第2章　BLDCモータ制御の原理

BLDCモータは固定子部分（ステータ）のコイルが発生する磁界の作用により，永久磁石でできた回転部分（ロータ）が回転します。

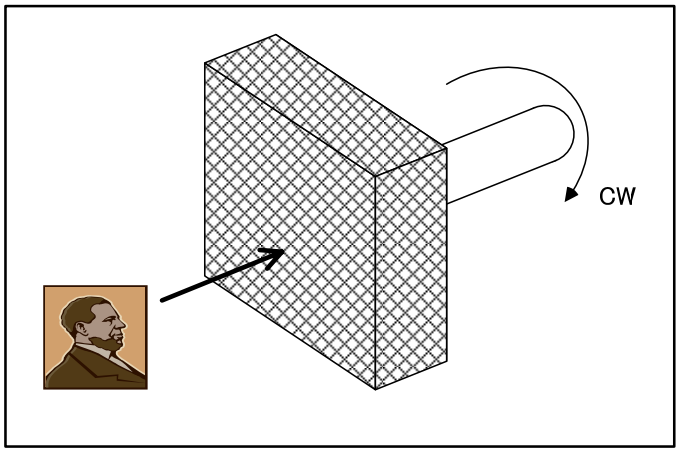
## 2. 1　回転方向の定義

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**



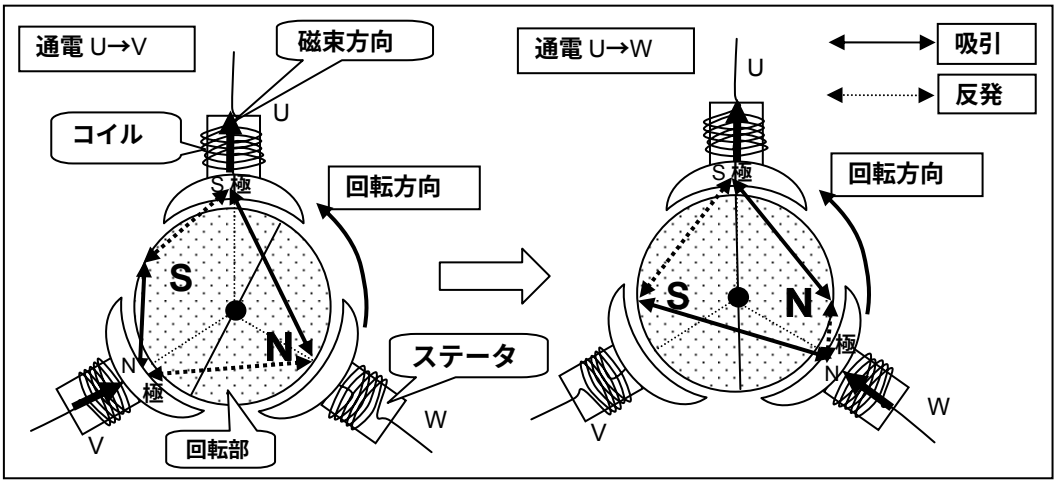

## 2. 2　回転原理

BLDCモータの回転原理を記述します。

次の図に示したBLDCモータは3相2極3スロットY結線でインナーロータ型のSPM（Surface Permanent Magnet：永久磁石を表面に配置した表面磁石構造） です。

ステータの極と回転部永久磁石の磁極との吸引および反発によって発生するマグネット・トルクで回転します。

図 2-2　BLDCモータの回転原理



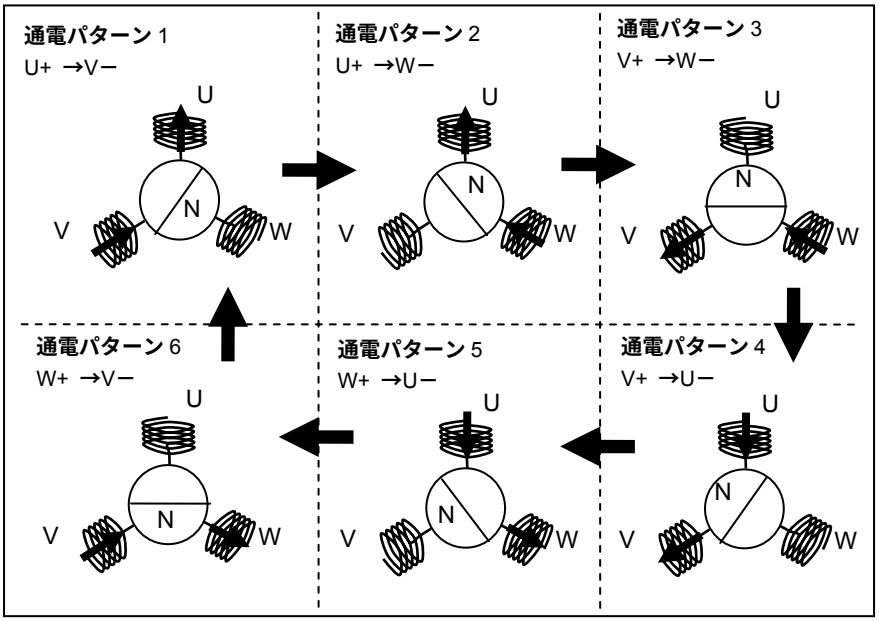

ステータの極はコイルの巻線方向に依存します。

回転部の磁極が逆の場合，回転方向が逆になります。

## 2. 3　通電パターン

次の図に通電パターンとコイルで発生する電流磁束方向（ステータの極）と回転部の磁極の関係を示します。

図 2-3　通電パターンとコイルの磁束方向

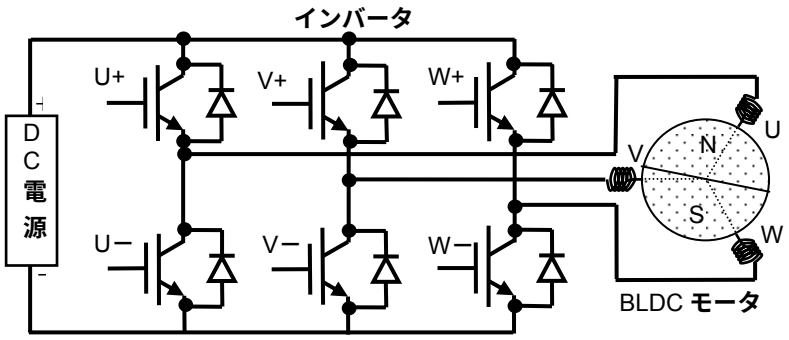



コイルの極は（コイルの）巻き線方向に依存します。

## 2. 4　インバータ

ブラシと整流子のないBLDCモータは，コイルと電流の向きをインバータで切り替えます。
次に3相Y結線BLDCモータとインバータとの結線図を示します。

図 2-4　インバータとBLDCモータ

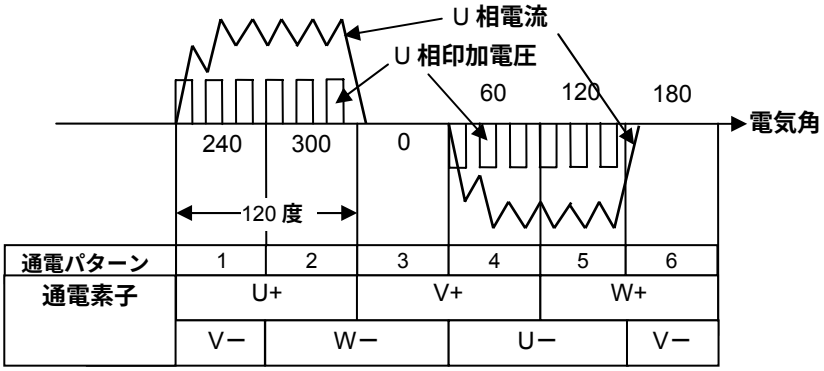



6個のスイッチング素子で通電時間や通電方向を制御します。

## 2. 5　120度通電方式

次の図は2. 3で示した通電パターンで各相に矩形波の電圧を印加したときのU層の状態（電圧，電流）を示したものです。U相には120度の通電期間と60度の無通電期間が繰り返し発生します。
各相の通電期間を120度で行うこの方式を120度通電方式といいます。

図 2-5　120度通電と電流




このシステムは，BLDCモータを120度通電方式で駆動します。

# 2. 6　位置情報

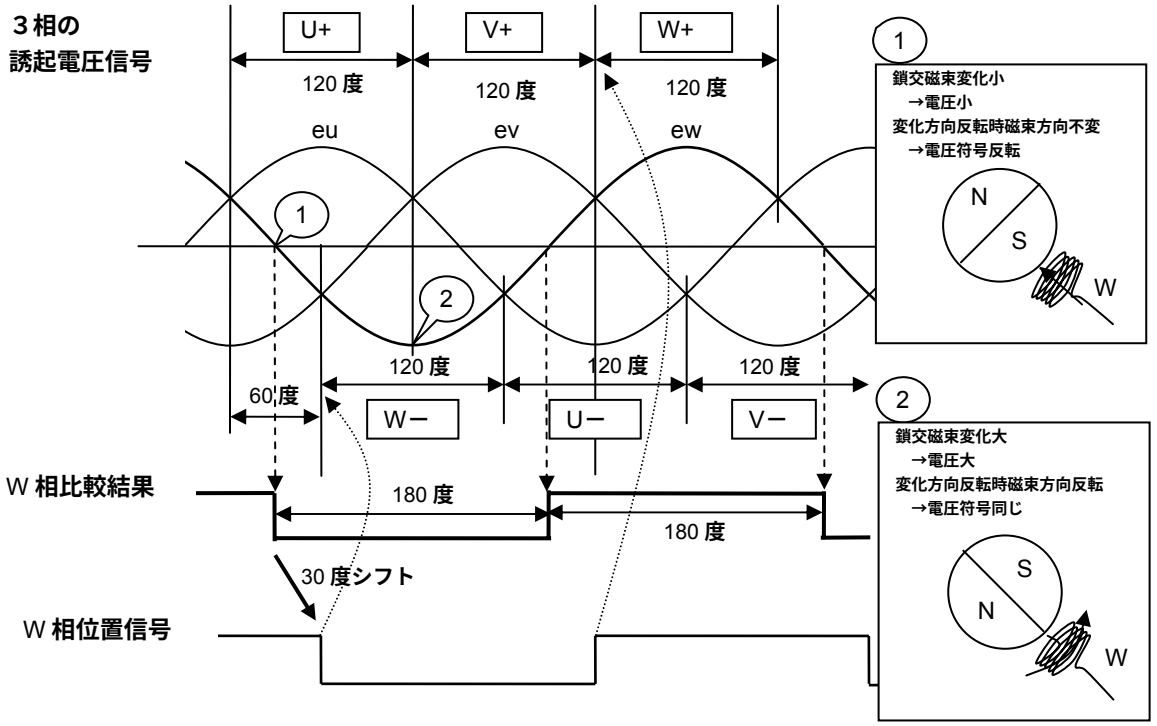
120度通電方式ではインバータの切り換えを行うための位置情報が必要です。

BLDCモータは３相の端子を開放してロータを外部から回転させた場合，各相に正弦波状（eu,ev,ew）が発生します。誘起電圧はロータ磁石の回転でステータコイルに発生する鎖交磁束の時間当たりの変化に比例するので，発生する電圧値はロータの回転位置を表します。

120度通電方式では，３相コイルのうち，二つのコイルに電流を流し，60度ごとに通電するコイルを切り替えて転流するため，通電していない開放相から誘起電圧（逆起電力）の検出が可能です。たとえばU相からV相に通電する区間では，開放相のW相の逆起電力（Back EMF）がゼロクロスするポイントが検出可能です。

次の図に位置推定原理を示します。

**図** 2-6　**位置推定原理**




W相の比較結果を30度シフトした信号がW相の通電切替えタイミングに一致します。

回転部の磁極が4極の場合，W相の比較結果は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（このマニュアル内の角度はすべて電気角で記述しています）。

機械角度：電気角／極対数

極対数：極数／2

## 2. 7　起動方法

誘起電圧による位置検出方法は，停止時や低速域では利用できないため，位置に無関係に通電パターンを切り替える同期始動で起動し，回転速度を上げてから比較結果信号を利用した駆動に切り替えます。

## 2. 8　通電パターン切り替え

各相の比較結果の変化から30度の時間経過後，比較結果に対応する通電パターンを設定します。

## 2. 9　速度検出

各相の比較結果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. 10　電圧制御

モータのコイルに加える電圧は，120度通電の矩形波をスイッチング素子のいずれか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. 11　速度制御

モータのコイルに加える電圧（PWMの通流率）をPID制御で変更することで速度を制御します。

### 2. 11. 1　PID制御

指定された速度と検出した回転速度の偏差でPID制御を行い，PWM電圧制御法の通流率（以降，デューティ比と記述）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），偏差の積分に比例する出力を出す積分動作（Integral action:I動作），偏差の微分に比例する出力を出す微分動作（Derivative action:D動作）からPWMのデューティ比操作量を求めます。

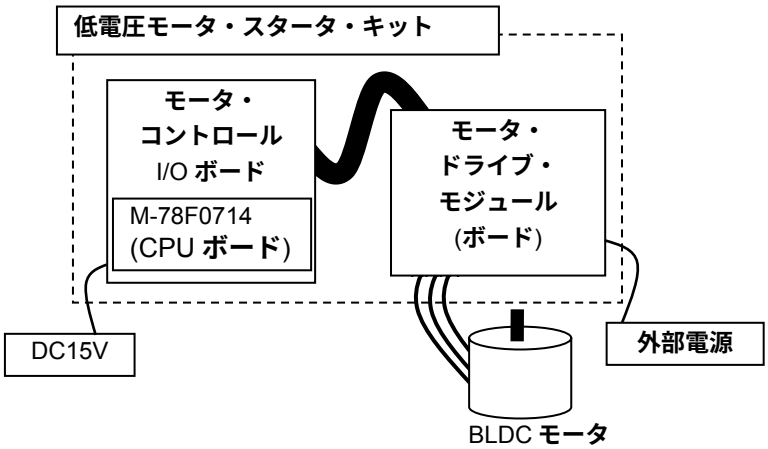
# 第3章　システム概要

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

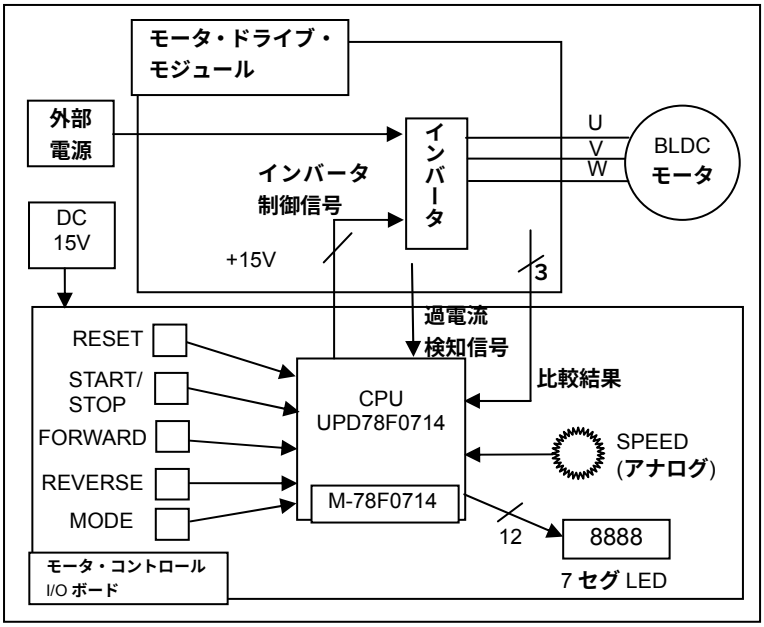



モータを駆動するモータ・ドライブ・モジュール，モータ制御用スイッチを搭載したモータ・コントロールI/Oボード，CPUを搭載したM-78F0714で構成されます。

BLDC モータは 3 相 4 極（2 極対）のモータです。

低電圧モータ・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**<br>**速度表示切り替え** |
| SPEED | R52 | **指定速度変更** |
| 7**セグ**LED | DISP1<br>DISP2<br>DISP3<br>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** 1 **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 |
|---|---|---|
| 過電流 | 0.C. | インバータの電流が異常 |
| 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-TW0TO5 | ３相PWMインバータ切り替え |
| 11 | P01/INTP1 | 比較結果信号（CMPU） |
| 10 | P02/INTP2 | 比較結果信号（CMPV） |
| 9 | P03/INTP3 | 比較結果信号（CMPW） |
| 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-P47 | 7セグLEDへの出力データ |
| 12 | TW0TOFFP/INTP0 | 過電流検知（+5V→0V） |
| 60 | ANI4 | 速度変更（R52） |
| 20 | P53/TI000/INTP5 | タイマ・キャプチャ・トリガ |
| 21 | P54/TI00/TO00 | モータ・ドライブ・モジュール制御 |

注　　M-78F0714 ボード，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

| 機　　能 | 概　　要 |
|---|---|
| 起動（電源供給） | ・LEDに“ SELF ”を1秒表示する<br>・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<br>・LEDに“ 0 ”を表示する<br>・モータはCWで回転を開始する<br>・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<br>・モータの回転速度（rpm）が0になるまでLEDに表示する<br>・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<br>・変化なし |
| | モータ制御中：回転方向を変える<br>・回転方向がCCWの場合，一度停止してからCWになる<br>・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<br>・変化なし |
| | モータ制御中：回転方向を変える<br>・回転方向がCCWの場合，変化しない<br>・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<br>・SPEEDボリュームによる指定速度変更を無効/有効にする<br>・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<br>・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<br>・LEDの表示速度（rpm）が200～3200に変化する<br>・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<br>・モータの回転速度（平均）が指定した速度になる |
| 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<br>・モータ制御を停止する<br>・LEDに“ 0.C. ”を表示する<br>・RESETスイッチ以外の機能を無効にする |
| 無回転 | ・STARTから約1秒経過後も回転していない場合，モータが約0.05秒以上停止した場合に発生する<br>・モータ制御を停止する<br>・LEDに“ FAIL ”を表示する<br>・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<br>・10ビット・インバータ制御用タイマ（TW0UDCなど）<br>・キャリア（変調）周波数は13 kHz対称三角波<br>・キャリア（変調）同期割り込みが76.9 μs間隔で発生する<br>（メインのPID制御でデューティ比が計算され，キャリア同期割り込みで更新される） |
| リアルタイム出力 | ・通電パターン切り替え用<br>・リアルタイム出力ポート（RTBH01, RTBL01など）<br>・16ビット・タイマ・キャプチャ／コンペア・レジスタ01（CR01）<br>（キャリア同期割り込みで通電パターンが変更される） |
| ウェイト処理 | ・タイミング調整用<br>・8ビット・タイマ／イベントカウンタ50（TM50,CR50） |
| 指定速度読み出し | ・可変抵抗値の電圧を指定速度に変換<br>・A/Dコンバータ（ANI4） |
| キャプチャ割り込み | ・速度計測用<br>・割り込み機能（TI000/INTP5） |
| H/W過電流割り込み | ・割り込み機能（INTP0）<br>・モータ・ドライブ・モジュールの過電流発生（LOWアクティブ） |
| フェールセーフ | ・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 | 比較結果の変化検知（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 |
| RESET | リセット発生 | 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

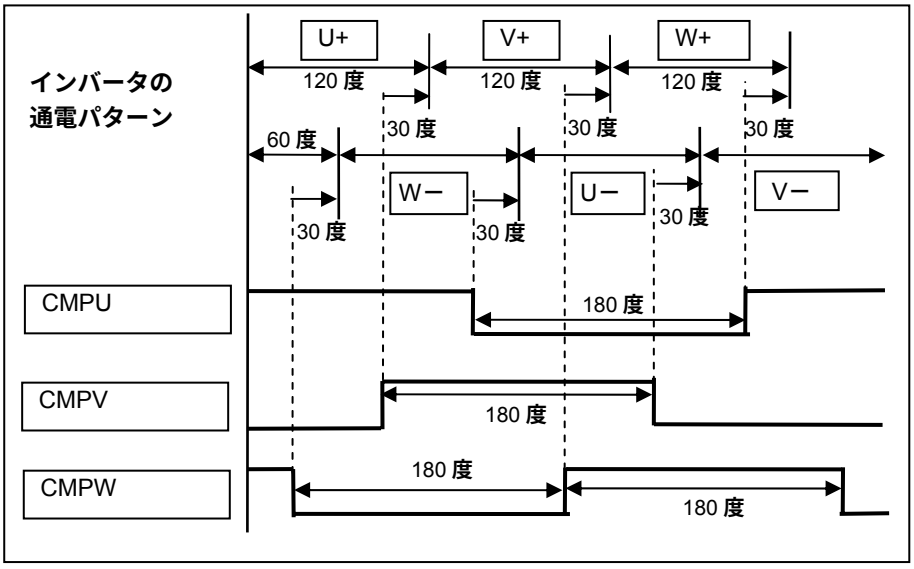
# 第4章　制御プログラム

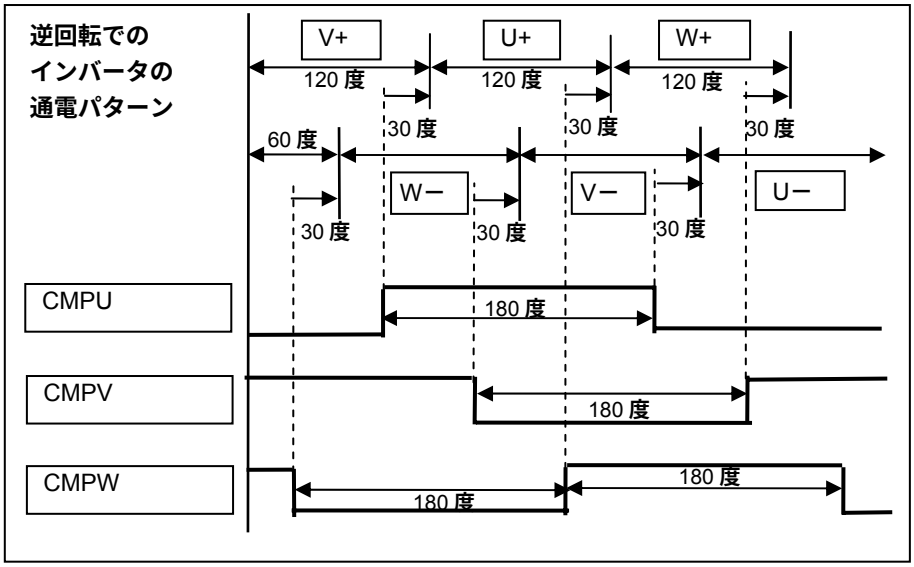
このシステムでは基本的な制御プログラムを用いて実際に制御を行います。

## 4. 1　通電パターン

このシステムの比較信号（CMPU,CMPV,CMPW）と120度通電パターン切り換えのタイミングを図 4-1に示します。

図 4-1　比較信号と通電パターン



BLDCモータのスペックから端子情報を記述します。

### 4.1.1　低電圧インバータ・セット

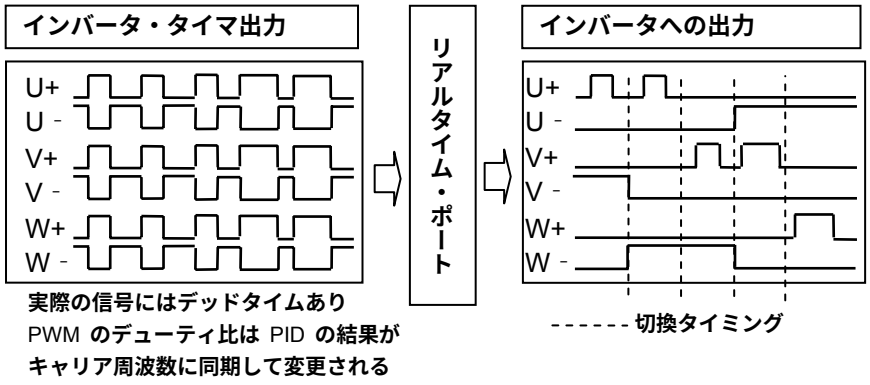
表 4-1　BLDCモータの端子スペック

| カラー | 機　　能 | 備　　考 |
|---|---|---|
| BROWN | MOTOR $\phi$A | U相 |
| RED | MOTOR $\phi$B | V相 |
| ORANGE | MOTOR $\phi$C | W相 |

## 4.2　通電パターン切り替え

通電パターンの切り替えはインバータ・タイマ出力（PWM波形）に同期して行う必要があるため，インバータ・タイマのキャリア同期割り込み（76.9 $\mu$s周期）で比較結果信号変化から30度の時間経過後に対応する通電パターンをリアルタイム・ポートに設定して行います。

図 4-2 通電パターン切り替えタイミング

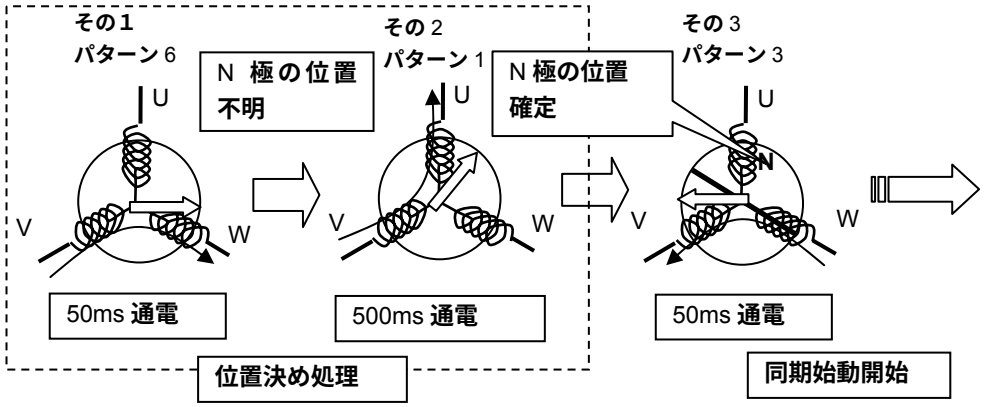



- 実際の信号にはデッドタイムあり
- PWM のデューティ比は PID の結果がキャリア周波数に同期して変更される

# 4. 3　起動方法

特定の二相に通電して強制的に回転子の位置を決め，順番（図 2-3　通電パターンとコイルの磁束方向参照）に通電パターンを切り替え，周期（回転速度）を上昇させます。指定時間経過後，BEMF信号を使った通電パターンによる制御に切り替えます。

図 4-3　起動時の通電パターン切り替え



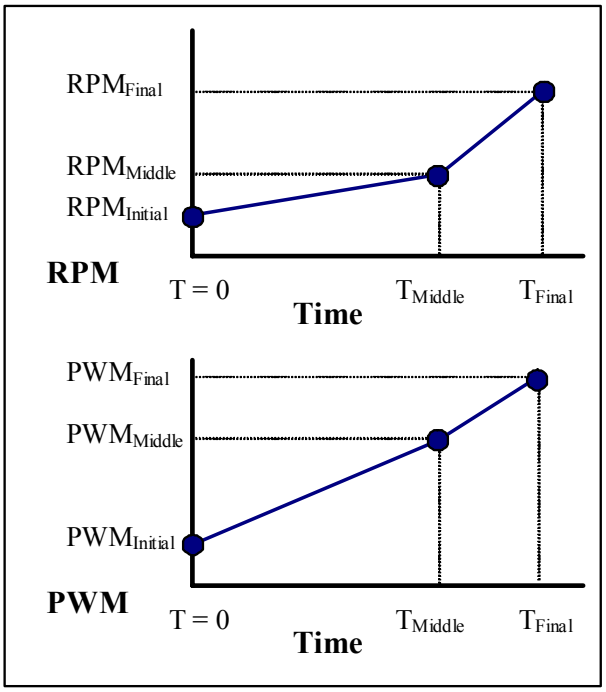

その1：その2で回転子のN極が電流磁束方向の反対側にある場合，
　　回転子が回転しない（N極が移動しない）ため，それを回避する予備動作
その2：回転子のN極を強制的に移動する
その3：電流磁束の方向が回転子のN極位置から60度～120度になる通電パターンから開始

自由度の高い起動を実現するために,下図グラフのような起動特性を実行することが出来ます。起動直後(T=0)，一定時間経過後（$T_{Middle}$），そしてBEMF信号による制御へ移行する経過時間（$T_{Final}$）におけるRPMとPWMの各値をそれぞれ指定できるようにしています。

表 4-3　指定可能なパラメータ

| 経過時間 | 回転数 | PWMduty比 |
|---|---|---|
| T=0 | $RPM_{Initial}$ | $PWM_{Initial}$ |
| $T_{Middle}$ | $RPM_{Middle}$ | $PWM_{Middle}$ |
| $T_{Final}$ | $RPM_{Final}$ | $PWM_{Final}$ |

図 4-4　起動特性グラフ




## 4.4　速度検出

タイマ・キャプチャ機能を用いて比較結果信号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な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つの比較結果信号は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200 rpm～3200 rpmの回転速度範囲をサポートしています。

## 4.5　電圧制御

モータに印加する電圧は，インバータのプラス側を13 kHzのキャリア周波数でチョッパ動作（PWM）させた平均電圧で制御します。

マイナス側はPWMによる相補動作を行わないため，減速時のブレーキ・トルクは発生しません（自然減速）。

# 4. 6　速度制御

150 ms周期にPWMのデューティ比を調整することでモータの回転速度を制御します。

デューティ比の調整量は指定速度とモータの回転速度の差をPID制御で求めています。

## 4. 6. 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PWMのデューティ比（平均電圧）にPID制御を行うことで速度調整を行っています。PWMのデューティ比操作量は，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。

$$MV_n = MV_{n-1} + \Delta MV_n$$
$$\Delta MV_n = Kp(e_n - e_{n-1}) + Ki \times e_n + Kd((e_n - e_{n-1}) - (e_{n-1} - e_{n-2}))$$

$MV_n$: 今回操作量
$MV_{n-1}$: 前回操作量
$\Delta MV_n$: 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
$Kp$：フィードバックゲイン（比例要素）
$Ki$：フィードバックゲイン（積分要素）
$Kd$：フィードバックゲイン（微分要素）

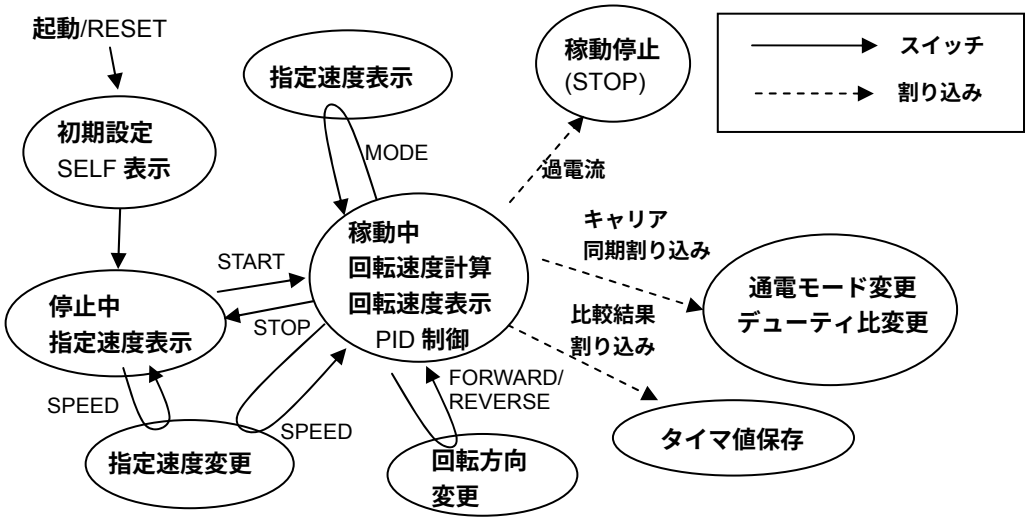
フィードバック・ゲインの最適値はモータ特性や負荷の有無で変化します。

PID制御の周期は最低速度（200 rpm）での速度更新間隔（150 ms）に設定しています。

# 4. 7　モジュール構成

システムの状態遷移を図 4-5に示します。

図 4-5　システムの状態遷移




キャリア同期割り込みは76.9 μsごとに発生します。

# 4. 8　制御プログラムの関数一覧とフローチャート

## 4. 8. 1　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。表 4-4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4. 8. 2　フローチャートを参照してください

表 4-4　関数の機能一覧（1/2）

| 関数名 | 機能概要 |
|---|---|
| void main(void) | メインルーチン<br>起動，初期設定，スイッチ入力待ち無限ループ（起動，停止，PID制御） |
| void int_carrier(void) | キャリア同期割り込み<br>モータ無回転の検出（エラー処理）<br>速度情報の取得<br>PWMのデューティ比変更<br>通電パターン切り替え |
| void int_fault(void) | 過電流割り込み<br>制御停止<br>エラー表示 |
| void system_init(void) | 初期設定<br>内蔵周辺の初期化<br>LEDへの“ SELF ”表示 |
| void system_start(void) | モータ制御開始<br>変数に開始時の値設定<br>内蔵周辺タイマ起動<br>割り込みマスク解除 |
| void system_restart(void) | 反転停止後のモータ制御再開<br>変数に再開時の値設定 |
| void system_stop(void) | モータ制御停止<br>割り込みマスク設定<br>変数に停止時の値設定<br>内蔵周辺タイマ停止 |
| void init_openloop(void) | 初起動制御に必要なパラメータの演算 |
| void INTP0_on(void) | 過電流割り込み許可<br>割り込みマスク解除 |
| void print_error(char) | エラー表示<br>7セグLEDにエラーを表示<br>STOP命令でプログラムを停止 |
| void pwm_pid(void) | 速度のPID制御<br>キャリア同期割り込み回数やタイマの値から速度を計算<br>指定時間ごとに指定速度との差分でPID制御を行い，PWMデューティ比を計算 |
| void get_speed(void) | 指定速度読み出し<br>可変抵抗の電圧をA/D変換した値から指定速度を計算<br>200～3200 rpmの指定速度を変数に設定 |
| unsigned char get_sw(void) | スイッチの読み出し<br>制御スイッチの状態を返す |
| void init_PORT(void) | I/Oポートの初期設定 |
| void init_OSC(void) | CPUクロックの切り替え |

表 4-4　関数の機能一覧（2/2）

| 関数名 | 機能概要 |
|---|---|
| void init_TW0(void) | 10ビット・インバータ・タイマの初期設定 |
| void set_TW0(int, int, int, int) | PWMのデューティ比変更 |
| void start_TW0(void) | PWMの出力開始 |
| void stop_TW0(void) | PWMの出力停止 |
| void init_TM00(void) | 16ビット・タイマ（TM00）初期設定 |
| void start_TM00(void) | 16ビット・タイマ（TM00）起動 |
| void stop_TM00(void) | 16ビット・タイマ（TM00）停止 |
| void init_RTPM01(void) | リアルタイム・ポートの初期設定 |
| void set_RTPM01(unsigned char) | リアルタイム・ポートの設定<br>通電パターン（PWMを出力するポート）を変更 |
| void start_RTPM01(void) | リアルタイム・ポート機能動作 |
| void stop_RTPM01(void) | リアルタイム・ポート機能動作停止 |
| void init_AD(void) | A/D機能初期設定<br>SPEEDボリュームのA/D変換を設定 |
| void start_AD(void) | A/D変換開始 |
| unsigned int get_AD(char) | A/D変換値の読み出し |
| void init_WDTM(void) | ウォッチドッグ・タイマの初期設定 |
| void clear_WDTM(void) | ウォッチドッグ・タイマのカウンタ・クリア |
| void reset_WDTM(void) | ウォッチドッグ・タイマによる内部リセット信号発生 |
| void init_TM50(void) | 8ビット・タイマ（TM50）の初期設定<br>1 msタイマで使用 |
| void start_TM50(void) | 8ビット・タイマ（TM50）起動 |
| void wait_TM50(void) | 8ビット・タイマ（TM50）待ち |
| void speed_print(int) | LEDに速度を表示する<br>モータ制御停止中は指定速度を表示する<br>モータ制御中でMODEスイッチが押されている場合は，モータの回転速度を表示する |
| void INTTW0UD_on(void) | キャリア同期割り込み許可 |
| void INTTW0UD_off(void) | キャリア同期割り込み禁止 |
| void INTP5_on(void) | INTP5割り込み許可 |
| void INTP5_off(void) | INTP5割り込み禁止 |
| void led_print(int, char) | LEDに１～４桁の値を表示する |
| void led_set<br>(unsigned char, unsigned char) | 指定されたLEDに値を出力する |
| void wait(int) | 指定時間（ms）経過待ち |
| char read_BEMF(void) | BEMF信号の値を読み出す |
| void int_speed (void) | INTP5割り込み |

## 4. 8. 2　フローチャート

図 4-6から図 4-50に，各関数のフローチャートを示します。

＜メインルーチン:main( )＞

図 4-6　メイン処理（1/2）

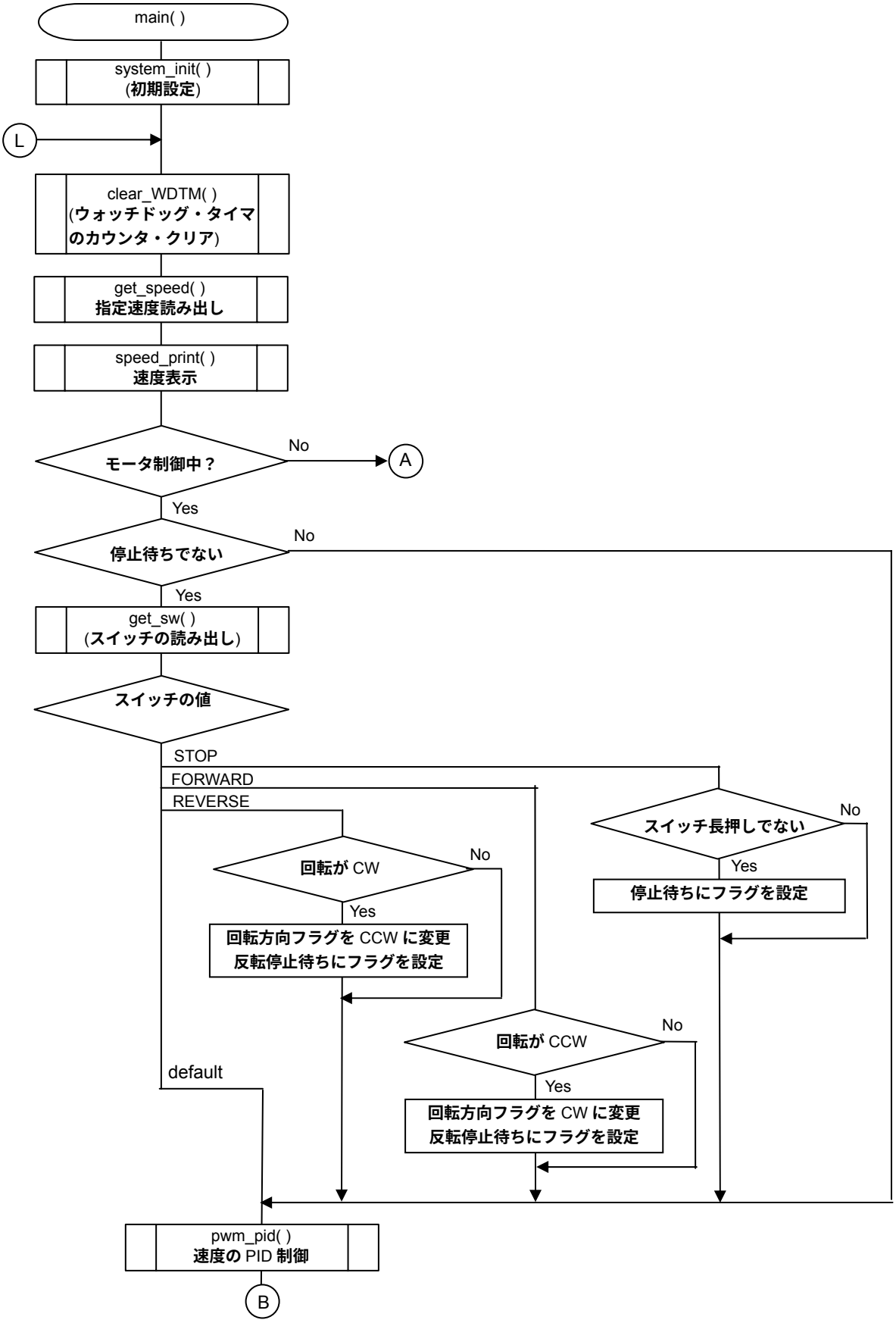
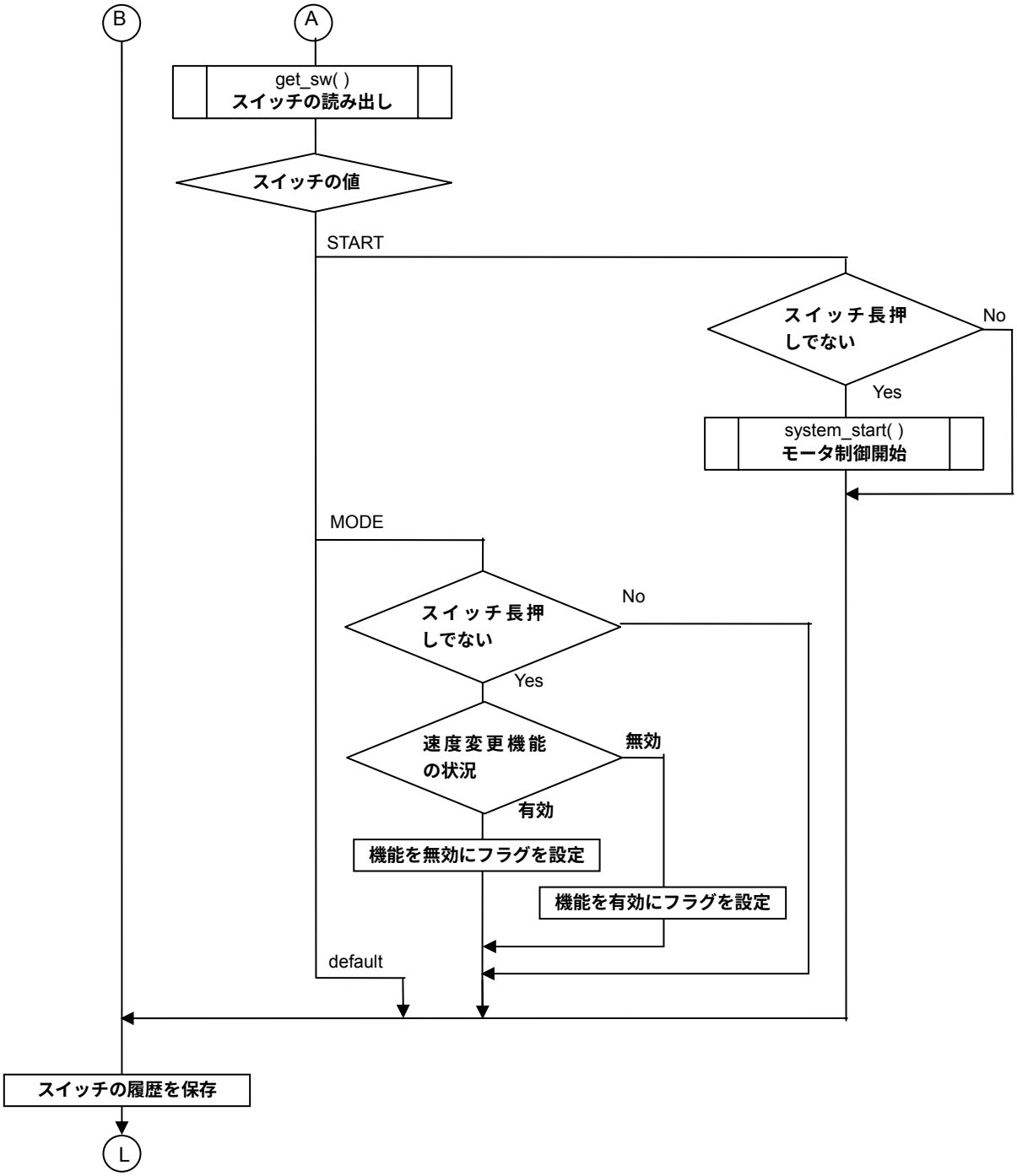


**図 4-6　メイン処理（2/2）**


B
A
get_sw( )
スイッチの読み出し
スイッチの値
START
スイッチ長押しでない
No
Yes
system_start( )
モータ制御開始
MODE
スイッチ長押しでない
No
Yes
速度変更機能の状況
無効
有効
機能を無効にフラグを設定
機能を有効にフラグを設定
default
スイッチの履歴を保存
L

＜キャリア同期割り込み＞

図 4-7　キャリア同期割り込み処理（1/3）

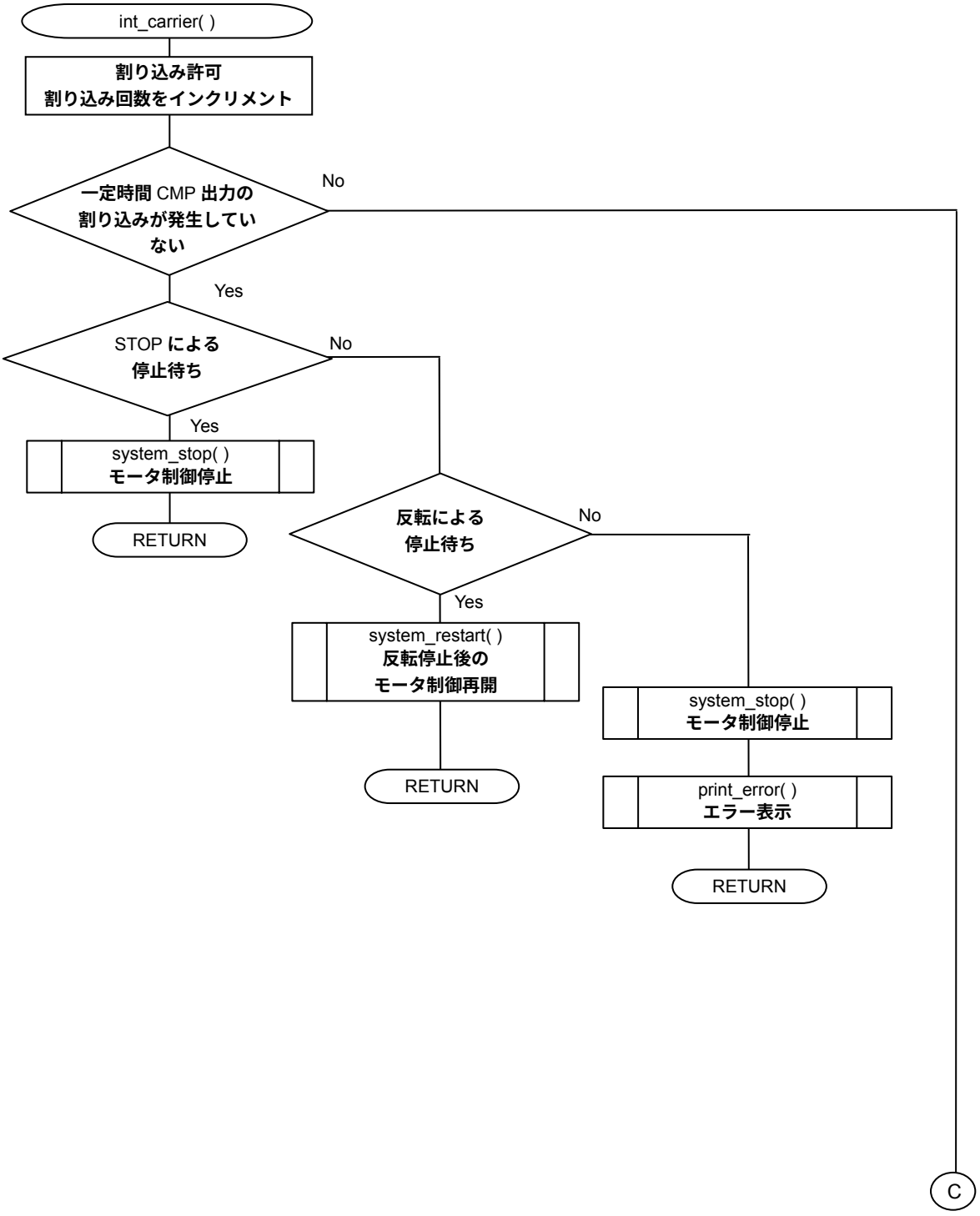


**図 4-7　キャリア同期割り込み処理（2/3）**

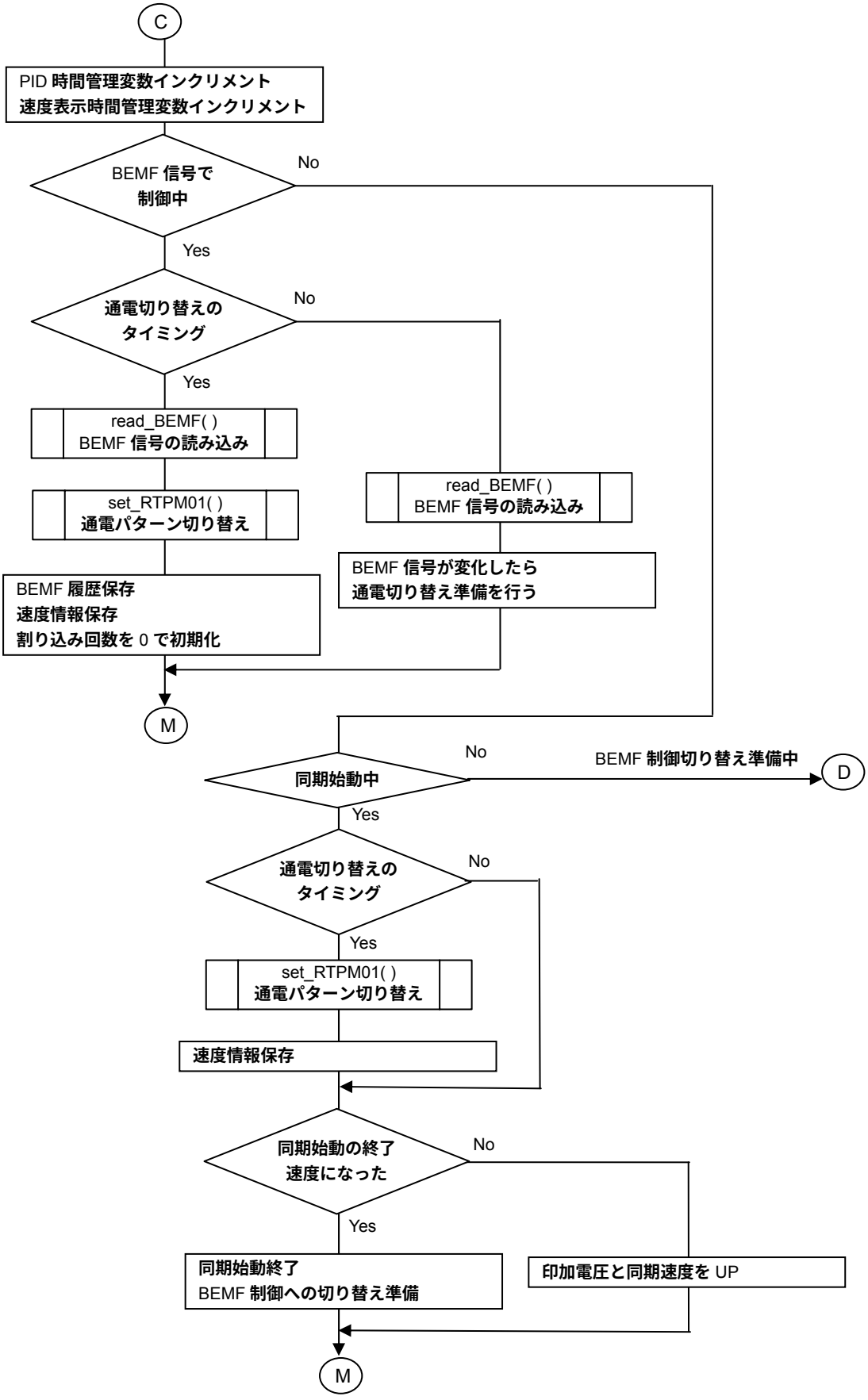
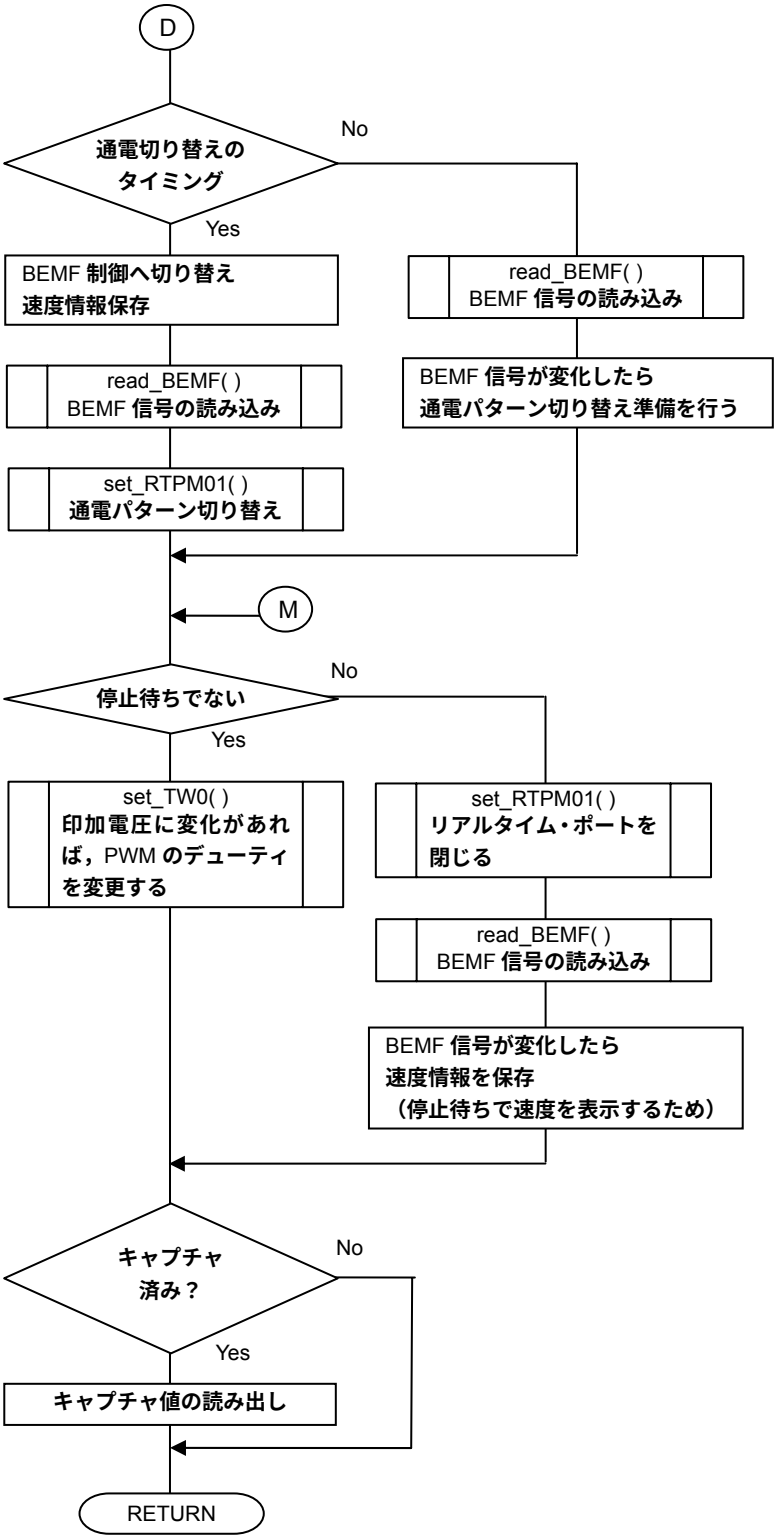
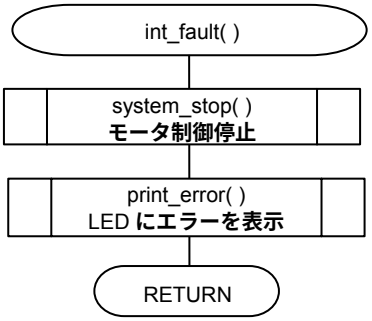


**図 4-7　キャリア同期割り込み処理（3/3）**



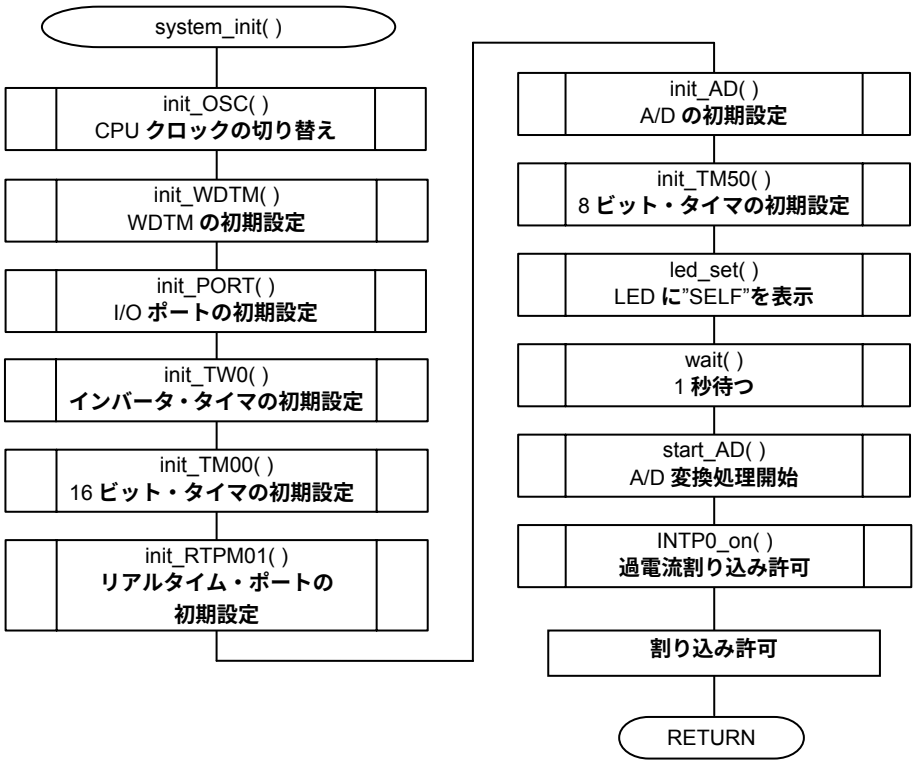
＜過電流割り込み＞

図 4-8　過電流割り込み処理



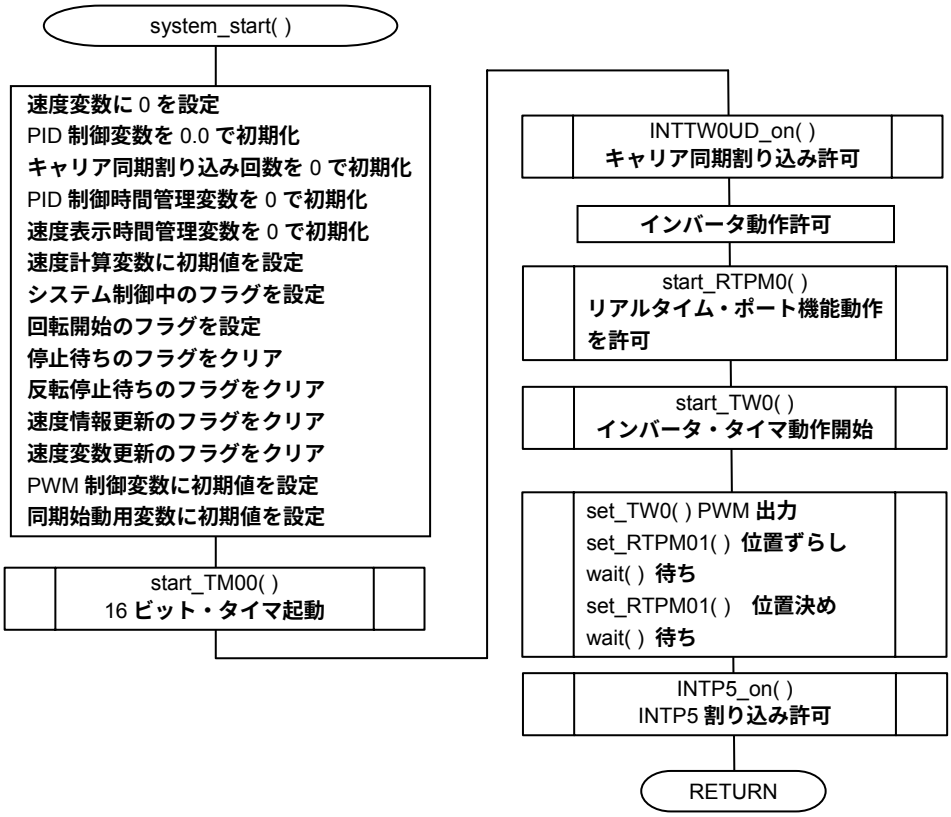

＜初期設定＞

図 4-9　初期設定処理



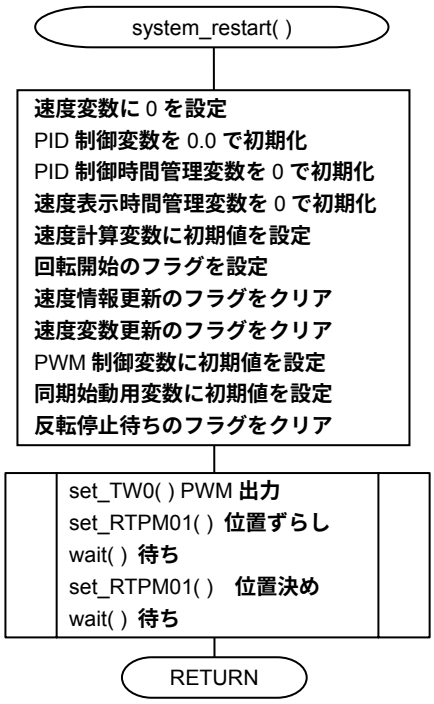
＜モータ制御開始＞

図 4-10　モータ制御開始処理




＜反転停止後のモータ制御再開＞

図 4-11　反転停止後のモータ制御再開処理

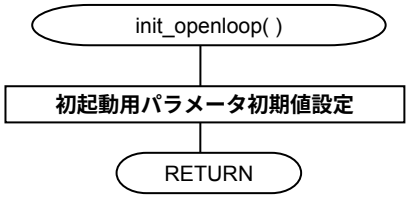


＜モータ制御停止＞

図 4-12　モータ制御停止処理


＜初起動用初期値設定＞

図 4-13　初起動用初期値設定処理

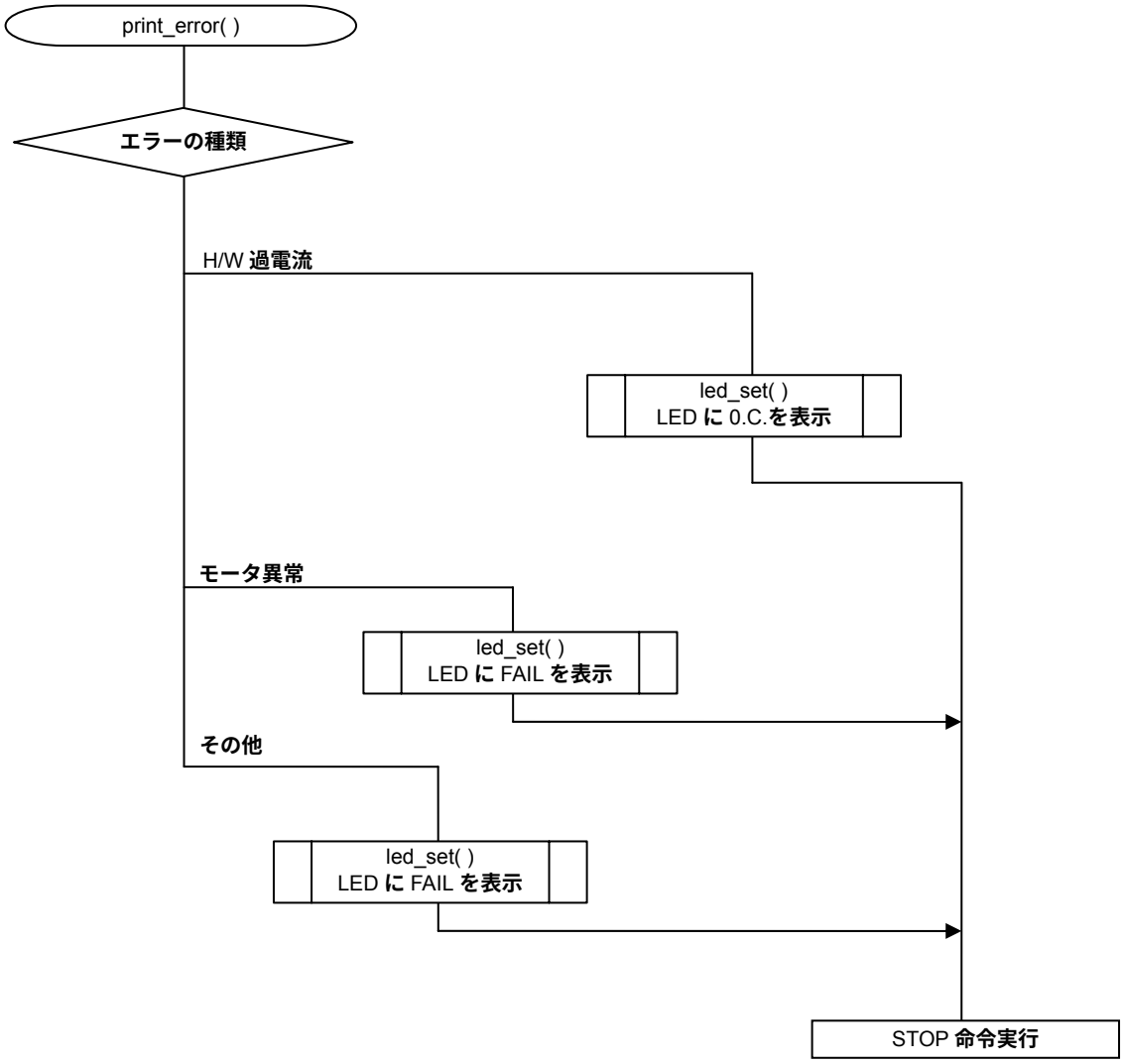



＜過電流割り込み許可＞

図 4-14　過電流割り込み許可処理

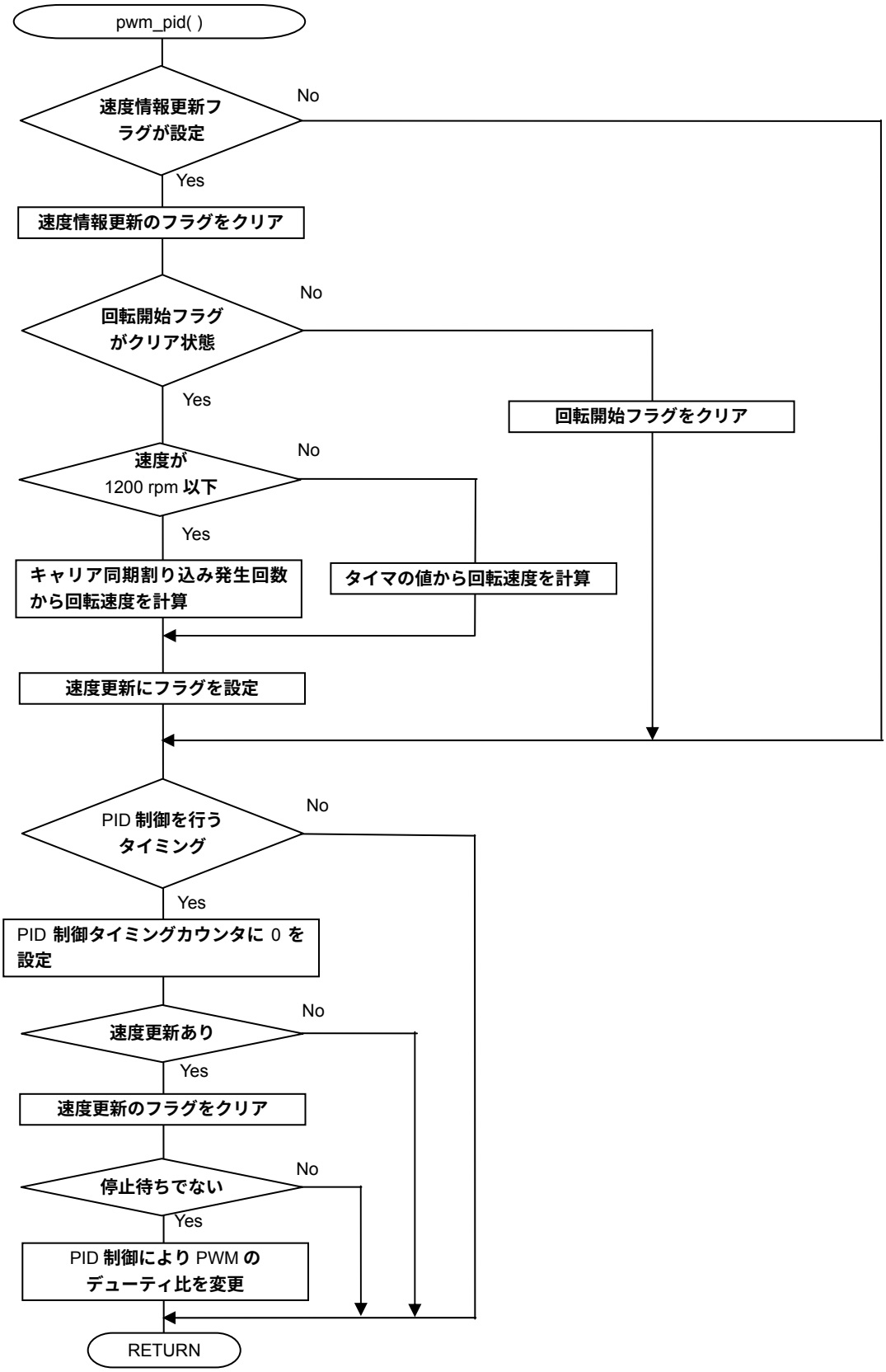
＜エラー表示＞

図 4-15　エラー表示処理



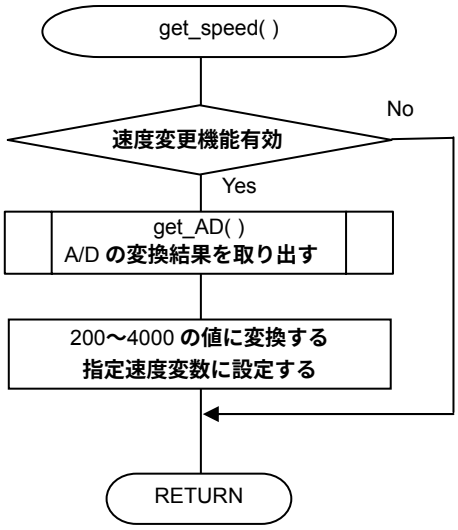
＜速度のPID制御＞

図 4-16　速度のPID制御処理



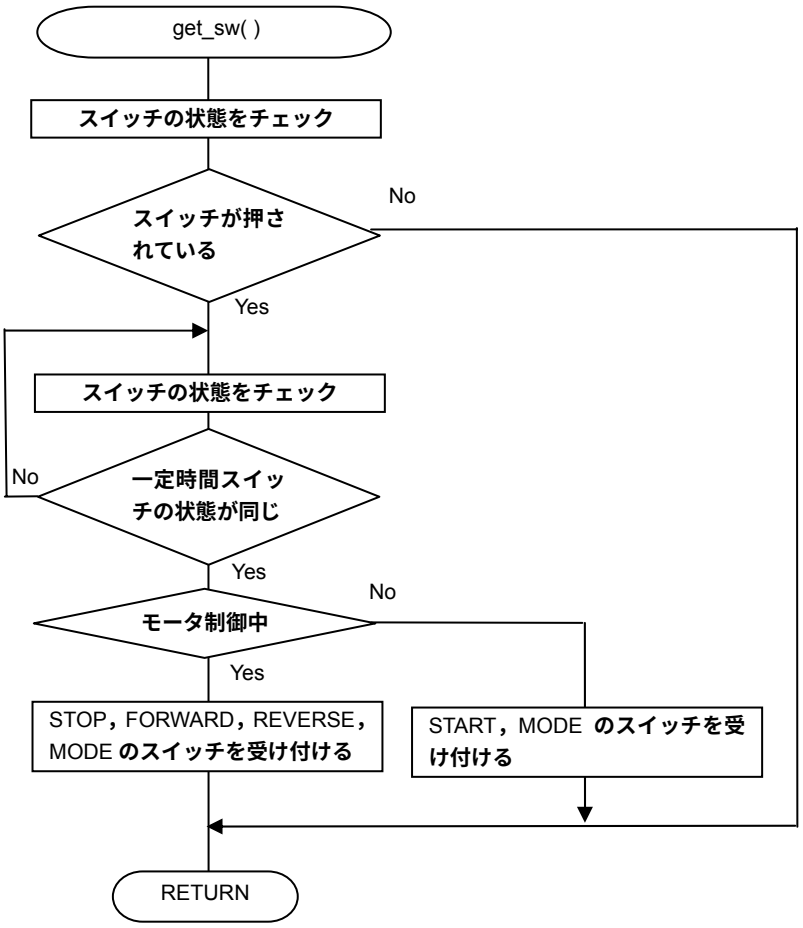
＜指定速度の読み出し＞

図 4-17　指定速度の読み出し処理



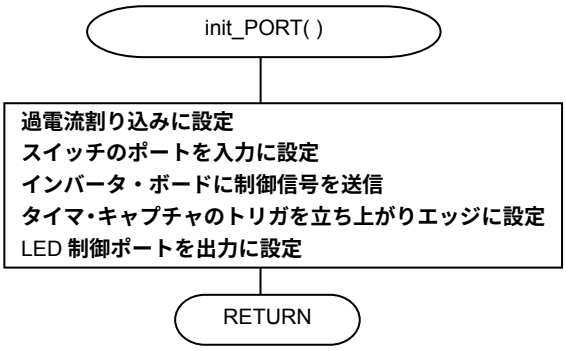

＜スイッチ読み出し＞

図 4-18　スイッチ読み出し処理



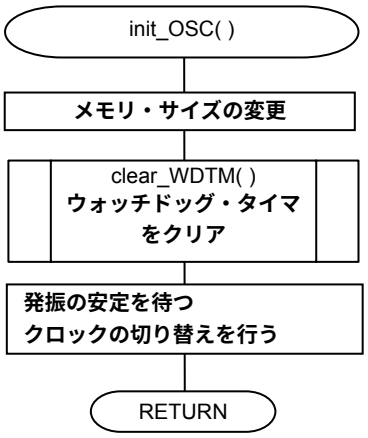
＜I/Oポートの初期設定＞

図 4-19　I/Oポートの初期設定処理




＜CPUクロックの切り替え＞

図 4-20　CPUクロック切り替え処理

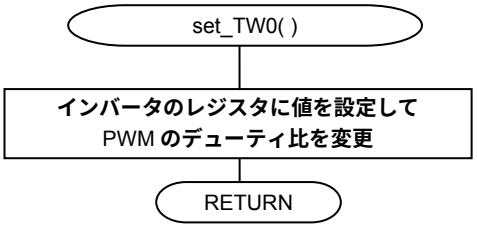



＜10ビット・インバータ・タイマの初期設定＞

図 4-21　10ビット・インバータ・タイマの初期設定処理

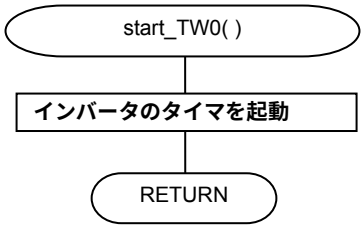
＜PWMのデューティ比変更＞

図 4-22　PWMのデューティ比変更処理



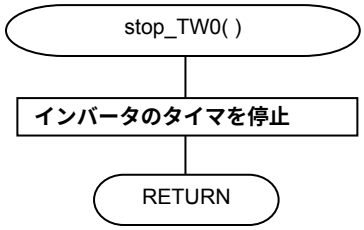

＜PWMの出力開始＞

図 4-23　PWMの出力開始処理



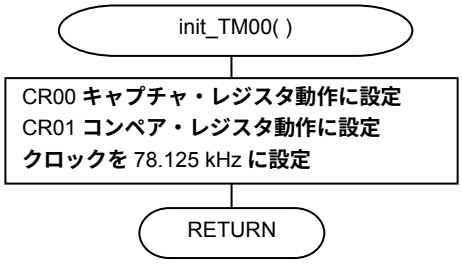

＜PWM出力停止＞

図 4-24　PWMの出力停止処理



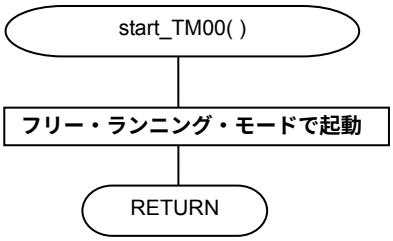

＜16ビット・タイマ（TM00）初期設定＞

図 4-25　16ビット・タイマ（TM00）初期設定処理



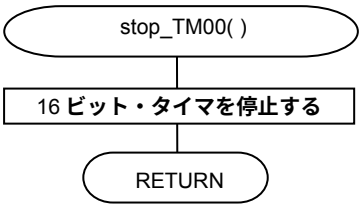
＜16ビット・タイマ（TM00）起動＞

図 4-26　16ビット・タイマ（TM00）起動処理



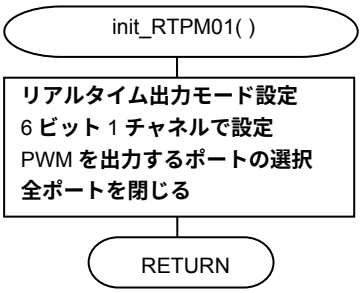

＜16ビット・タイマ（TM00）停止＞

図 4-27　16ビット・タイマ（TM00）停止処理



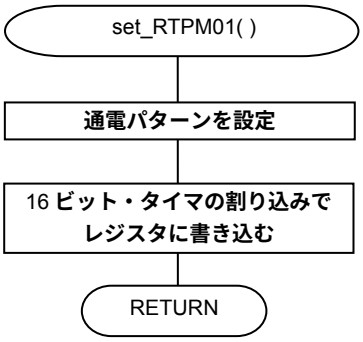

＜リアルタイム・ポートの初期設定＞

図 4-28　リアルタイム・ポートの初期設定処理



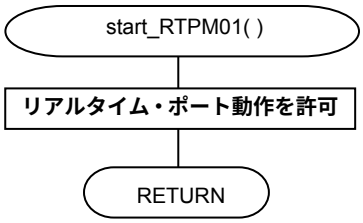
＜リアルタイム・ポートの設定＞

図 4-29　リアルタイム・ポートの設定処理



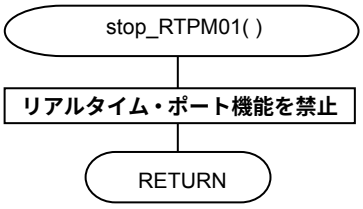

＜リアルタイム・ポート起動＞

図 4-30　リアルタイム・ポート機能動作処理



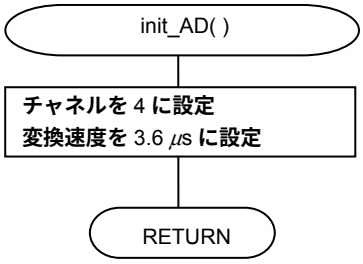

＜リアルタイム・ポート停止＞

図 4-31　リアルタイム・ポート機能動作停止処理



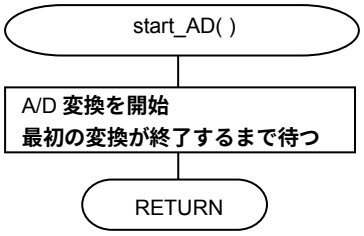
＜A/D機能初期設定＞

図 4-32　A/D機能初期設定処理



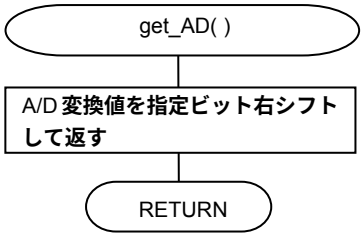

＜A/D 変換開始＞

図 4-33　A/D変換開始処理



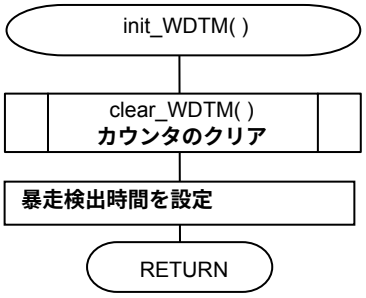

＜A/D 変換値の読み出し＞

図 4-34　A/D変換値の読み出し処理



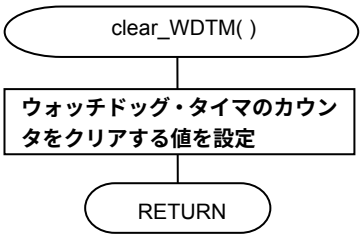

＜ウォッチドッグ・タイマ初期設定＞

図 4-35　ウォッチドック・タイマの初期設定処理



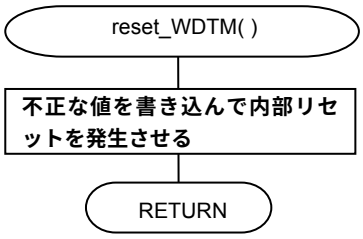
＜ウォッチドッグ・タイマのカウンタ・クリア＞

図 4-36　ウォッチドッグ・タイマのカウンタ・クリア処理



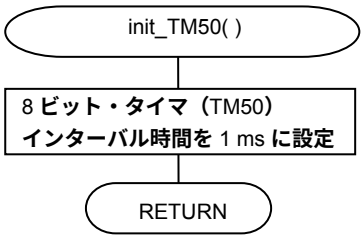

＜ウォッチドッグ・タイマによる内部リセット信号発生＞

図 4-37　ウォッチドッグ・タイマによる内部リセット信号発生処理



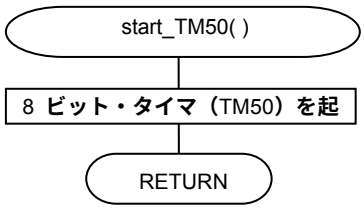

＜8ビット・タイマ（TM50）の初期設定＞

図 4-38　8ビット・タイマ（TM50）の初期設定処理



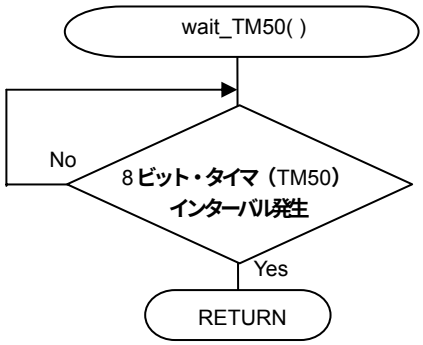

＜8ビット・タイマ（TM50）起動＞

図 4-39　8ビット・タイマ（TM50）起動処理



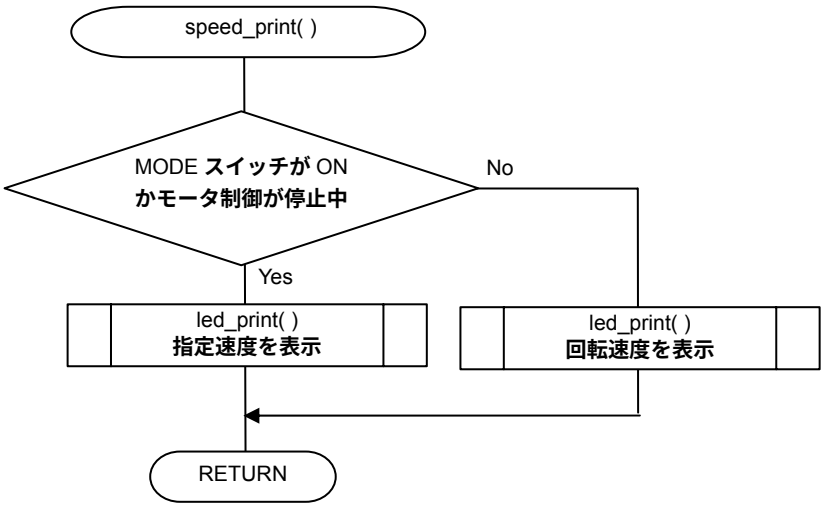
＜8ビット・タイマ（TM50）待ち＞

**図** 4-40　8**ビット・タイマ（**TM50**）待ち処理**



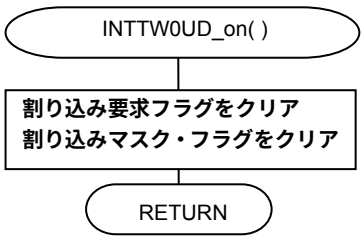

＜LEDに速度を表示する＞

**図** 4-41　LED**に速度を表示する処理**



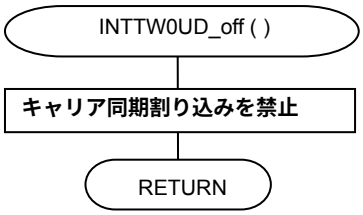

＜キャリア同期割り込み許可＞

**図** 4-42　**キャリア同期割り込み許可処理**



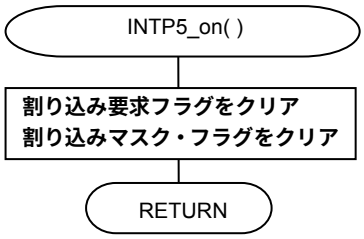
＜キャリア同期割り込み禁止＞

図 4-43　キャリア同期割り込み禁止処理



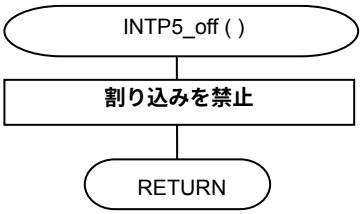

＜INTP5割り込み許可＞

図 4-44　INTP5割り込み許可処理




＜INTP5割り込み禁止＞

図 4-45　INTP5割り込み禁止処理

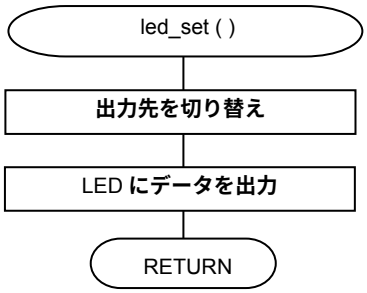


＜LEDに１～４桁の値を表示する＞

図 4-46　LEDに１～４桁の値を表示する処理

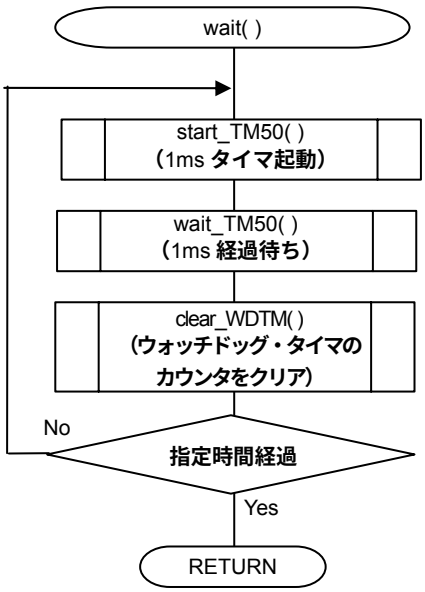

＜指定されたLEDに値を出力する＞

図 4-47　指定されたLEDに値を出力する処理



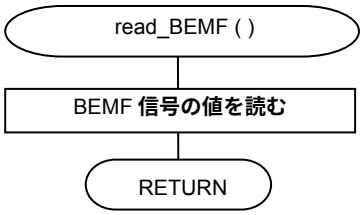
＜指定時間（ms）経過待ち＞

図 4-48　指定時間（ms）経過待ち処理



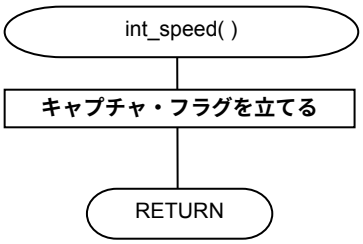

＜BEMF信号の値を読み出す＞

図 4-49　BEMF信号の値を読み出す処理




＜INTP5割り込み＞

図 4-50　INTP5割り込み処理



# 4. 9　制御プログラムの変数・定数一覧

## 4. 9. 1　main.hの#defineで定義している値

| 名　　称 | 意　　味 | 設定値 | 備　　考 |
|---|---|---|---|
| FLG_ON | フラグ値 | 1 | フラグの有効/無効を示す |
| FLG_OFF | フラグ値 | 0 | |
| FLG_START | フラグ値 | 2 | |
| FLG_WAIT | フラグ値 | 4 | |
| CW | 回転方向 | 0 | モータの回転方向を示す |
| CCW | 回転方向 | 1 | |
| 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 |
| 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 |
| 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 | |

## 4.9.2　lib_eu.hの#defineで定義している値

（1/2）

| 名　　称 | 意　　味 | 設定値 | 備　　考 |
|---|---|---|---|
| KP_DEF | フィードバック・ゲイン | 0.075 | PID制御で使用 |
| KI_DEF | フィードバック・ゲイン | 0.001 | |
| KD_DEF | フィードバック・ゲイン | 0.02 | |
| PWM_LIMIT_H | PWMの操作量上限 | 50.0 | |
| PWM_LIMIT_L | PWMの操作量下限 | -50.0 | |
| PWM_BASE_REF | PWMベース | 769 | PWM出力で使用 |
| PWM_F_REF | PWM初期値 | 0 | |
| PWM_F_START | PWM開始値 | 44 | |
| PWM_F_MIN | PWM最小値 | 10 | |
| PWM_F_MAX | PWM最大値 | 769 | |
| 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 | インバータ制御ポート制御レジスタ | 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 |
| 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 |

(2/2)

| 名　　称 | 意　　味 | 設定値 | 備　　考 |
|---|---|---|---|
| LED_0 | LED表示データ | 0xc0 | “0”を表示 |
| LED_1 | | 0xf9 | “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 |
| 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 |
| IMS_DATA | メモリ・サイズ | 0xc8 | メモリサイズ切替設定に使用 |
| WDTM_OFF | 停止 | 0x77 | ウォッチドッグ・タイマ制御用 |
| WDTM_SET | 設定 | 0x67 | |
| WDTE_CLR | クリア | 0xac | |
| WDTE_RESET | リセット | 0x00 | |
| 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 |
| 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 | |
| OFF_SW | 押されていない | 0xf | |

### 4.9.3 変　　数

| 変数名 | 型 | 意　　味 | 備　考 |
|---|---|---|---|
| sys_flag | char | 制御フラグ | 制御状態 |
| 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 |
| speed_flag | char | 速度情報フラグ | 速度計算に必要なデータの有無 |
| ad_flag | char | 速度変更フラグ | 指定速度変更機能を制限 |
| 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 |
| 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 |
| cnt_data<br>cnt_data1<br>cnt_data2<br>cnt_data3 | unsigned int | 速度計算用データ | キャリア同期割り込み回数 |
| sync_flag | char | 同期始動フラグ | 同期始動とBEMF制御を区別 |
| cnt_flag | char | 時間管理 | 30度遅延用 |
| sync_sw | char | 通電パターン | ポインタ |
| sync_tbl[][] | unsigned char | 通電パターン | テーブル |
| sync_cnt | unsigned int | 30度遅延用 | キャリア同期割り込み回数 |
| sync_def | unsigned int | 30度遅延の基準値 | 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 |
| pwm_ff | int | PWM指令値 | PWMの流通率 |
| old_ff | int | PWM指令値履歴 | 1つ前のpwm_ffの値 |
| m_speed | int | 実速度 | モータの回転速度（rpm） |
| led_data[ ] | unsigned char | LED出力データ | LEDに表示する数値 |
| up_flag | char | 実速度計算状態フラグ | 回転速度計算用 |
| old_clk | unsigned int | 前回のクロック値 | |
| new_clk | unsigned int | 最新のクロック値 | |
| kp_ref | float | Pゲイン | 速度制御 |
| ki_ref | float | Iゲイン | |
| kd_ref | float | Dゲイン | |
| mvn | float | 前回の操作量 | |
| en | float | 今回の差分 | |
| en_1 | float | 前回の差分 | |
| en_2 | float | 前々回の差分 | |

# 4. 10　制御プログラムのソースファイル

**プログラムサイズ**　　ROM－ 1EDDh
　　RAM－ 3C0h

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    target  : uPD78F0714 モータ・スタータ・キット

    date    : 2006/12/13
    filename: main.h

    NEC Micro Systems,Ltd
*/

#define FLG_ON          1
#define FLG_OFF         0
#define FLG_START       2
#define FLG_WAIT        4

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

#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 U→V          */
#define P_T2            0x1e            /* 通電パターン 2 U→W          */
#define P_T3            0x1b            /* 通電パターン 3 V→W          */
#define P_T4            0x39            /* 通電パターン 4 V→U          */
#define P_T5            0x2d            /* 通電パターン 5 W→U          */
#define P_T6            0x27            /* 通電パターン 6 W→V          */

extern const unsigned char sync_tbl[2][6];
extern char             sync_flag;
extern char             cnt_flag;
extern char             sync_sw;
extern char             sys_flag;
extern char             stop_wait;
extern char             cw_ccw_flag;
extern char             cw_ccw_wait;
extern char             speed_flag;
extern char             ad_flag;
extern char             clk_flag;
extern char             capture_flag;
extern unsigned int     new_clk;
extern unsigned int     old_clk;
extern unsigned int     bck_clk;
extern unsigned int     sync_cnt;
extern unsigned int     sync_def;
extern char             openloop_dim;
extern unsigned int     limit_tim[2];
extern unsigned int     limit_pwm[2];
extern unsigned int     limit_rpm[2];
extern unsigned char    const_rpm[70];
extern unsigned int     cnt_limit_tim;
extern unsigned int     cnt_limit_pwm;
extern unsigned int     cnt_limit_rpm;
extern unsigned int     cnt_const_rpm;
extern unsigned int     int_cnt;
extern unsigned int     pid_cnt;
extern unsigned int     print_cnt;
extern unsigned int     cnt_data;
extern unsigned int     cnt_data1;
extern unsigned int     cnt_data2;
extern unsigned int     cnt_data3;
extern unsigned int     cnt_dataH;
extern int              pwm_ff;
extern int              old_ff;
extern int              pwm_base;
extern int              m_speed;

extern void             system_init(void);
extern void             speed_print(int);
extern unsigned char    get_sw(void);
extern void             pwm_pid(int);
```

```
extern void             system_start(void);
extern void             system_stop(void);
extern void             system_restart(void);
extern char             read_BEMF(void);
extern void             set_RTPMO1(unsigned char);
extern void             set_TWO(int, int, int, int);
extern void             print_error(char);
extern void             clear_WDTM(void);
extern void             clear_WDTM(void);
```

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    target  : uPD78F0714 モータ・スタータ・キット

    date    : 2006/12/18
    filename: main.c

    NEC Micro Systems,Ltd
*/

#include    "main.h"

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

    system_init();                          /* 初期設定                 */
    while(1){
        clear_WDTM();                       /* ウォッチドッグタイマクリア */
        get_speed();                        /* 指定速度読み出し          */
        speed_print(300);                   /* 速度表示(300ms 間隔)       */
        if(sys_flag != FLG_OFF){            /* 起動中                   */
            if((cw_ccw_wait == FLG_OFF) &&  /* 反転の停止待ちでない      */
               (stop_wait   == FLG_OFF)){   /* 停止待ちでない           */
                sw = get_sw();              /* スイッチチェック          */
                switch(sw){
                case STOP_TR:
                    if(sw != tmp_sw){       /* トグル対処               */
                        stop_wait = FLG_ON; /* 停止待ちを設定           */
                    }
                    break;

                case FORWARD_TR:
                    if(cw_ccw_flag == CCW){ /* CCW で回転中              */
                        cw_ccw_wait = FLG_ON;/* 反転の停止待ちを設定      */
                        cw_ccw_flag = CW;   /* 回転方向を設定           */
                    }
                    break;

                case REVERSE_TR:
                    if(cw_ccw_flag == CW){  /* CW で回転中               */
                        cw_ccw_wait = FLG_ON;/* 反転停止待ちを設定       */
                        cw_ccw_flag = CCW;  /* 回転方向を設定           */
                    }
                    break;

                default:
                    ;
                }
            }
            pwm_pid(150);                   /* PID 制御 150ms 周期        */
        }else{                              /* 停止中                   */
            sw = get_sw();                  /* スイッチチェック          */
            switch(sw){
            case START_TR:                  /* 起動                     */
                if(sw != tmp_sw){           /* トグル対処               */
                    system_start();         /* 起動                     */
                }
                break;

            case MODE_TR:                   /* 速度変更機能             */
                if(sw != tmp_sw){           /* トグル対処               */
                    if(ad_flag == FLG_ON){  /* 機能停止中               */
                        ad_flag = FLG_OFF;  /* 機能復帰                 */
                    }else{
                        ad_flag = FLG_ON;   /* 機能停止                 */
                    }
                }
                break;

            default:
                ;
            }
        }
```

```
        tmp_sw = sw;
    }
}
```

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    target  : uPD78F0714 モータ・スタータ・キット

    date    : 2006/12/20
    filename: motor.c

    NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma ei

#pragma INTERRUPT INTP0     int_fault   rb1  /* 過電流発生    */
#pragma INTERRUPT INTTW0UD  int_carrier rb2  /* キャリア同期割り込み  */


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

/* ---------------------------------------------- */
#define INT_CNT_MAX     650                         /* 50ms 無回転の判定 x100us    */
#define SYNC_END_CNT    325                         /* 200rpm */
#define SYNC_UP_CNT     30
#define V_ADD_CNT       50

#define ISHUNT_AD       5       /* ISHUNT 端子が接続された AD チャンネル番号 */

#define ADDATA_MIN      0
#define ADDATA_100      0x1900
#define ADDATA_200      0x3200
#define ADDATA_300      0x4b00
#define ADDATA_400      0x6400
#define ADDATA_E_2      0x7400    /* 464<<6 */
#define ADDATA_500      0x7d00
#define ADDATA_600      0x9600
#define ADDATA_700      0xaf00
#define ADDATA_800      0xc800
#define ADDATA_900      0xe100
#define ADDATA_1000     0xfa00
#define ADDATA_MAX      0xfcc0

#define CLEAR           0
#define SET             1

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

const unsigned char tr_sw[2][8] = {                 /* 通電パターン */
    {P_OFF, P_T2, P_T4, P_T3, P_T6, P_T1, P_T5, P_OFF},  /* CW         */
    {P_OFF, P_T1, P_T3, P_T2, P_T5, P_T6, P_T4, P_OFF}   /* CCW        */
};

/* ============================================== */
__interrupt void
int_fault(void)
{
    system_stop();                      /* システム停止 */
    print_error(ERROR_OC);              /* エラー処理   */
}


/*
    キャリア同期割り込み
*/
__interrupt void
int_carrier(void)
{
    static char         r_data, o_data;
    static int          cnt;
    int                 tmp_pwm;

    int_cnt++;                                      /* 割り込み回数                 */
    if(int_cnt > INT_CNT_MAX){                      /* 50ms 以上通電切り替え無し    */
        if(stop_wait != FLG_OFF){                   /* STOP での停止待ち             */
            system_stop();
        }else if(cw_ccw_wait != FLG_OFF){           /* 反転で停止待ち               */
            system_restart();
        }else{
            system_stop();
            print_error(ERROR_MOTOR);
        };
        return;
    };
    pid_cnt++;                                      /* PID 制御タイミング     */
```

```
    print_cnt++;                                        /* 速度表示タイミング    */
    if(sync_flag == FLG_OFF){                           /* BEMF 信号で制御中     */
        if(cnt_flag != FLG_OFF){                        /* 通電切り替え待ち      */
            if(--cnt <= 0){                             /* 30 度遅延             */
                cnt_flag = FLG_OFF;
                o_data   = read_BEMF();
                set_RTPMO1(tr_sw[cw_ccw_flag][o_data]);
                cnt_data3  = cnt_data2;
                cnt_data2  = cnt_data1;
                cnt_data1  = int_cnt;
                cnt_data   = cnt_data1 + cnt_data2 + cnt_data3;
                int_cnt    = 0;
                speed_flag = FLG_ON;
            }else{
                if(r_data != read_BEMF()){
                    cnt_flag = FLG_OFF;
                };
            };
        }else{
            r_data = read_BEMF();
            if(o_data != r_data){                       /* BEMF 信号が変化       */
                cnt      = (int)(cnt_data1>>1);
                cnt_flag = FLG_ON;
            };
        };
    }else if(sync_flag == FLG_START){                   /* 同期始動中            */
        int_cnt = 0;
        pid_cnt = 0;
        cnt_limit_tim++;
        cnt_limit_pwm++;
        cnt_limit_rpm++;

        if(--sync_cnt == 0){
            /* 通電パターン切り替え */
            if(++sync_sw > 5){ sync_sw = 0;};
            set_RTPMO1(sync_tbl[cw_ccw_flag][sync_sw]);
            speed_flag = FLG_ON;
            cnt_data3 = cnt_data2;
            cnt_data2 = cnt_data1;
            cnt_data1 = sync_def;
            cnt_data  = cnt_data1 + cnt_data2 + cnt_data3;

            if(cnt_limit_tim >= limit_tim[openloop_dim]){
                cnt_limit_tim = 0;
                openloop_dim++;
                if(openloop_dim >= 2){
                    /* オープンループを抜け PID 制御の前処理へ */
                    sync_flag = FLG_WAIT;
                    o_data    = read_BEMF();
                    cnt_flag  = FLG_OFF;
                };
            }else{
                if(cnt_limit_pwm >= limit_pwm[openloop_dim]){
                    pwm_ff++;
                    cnt_limit_pwm = 0;
                };
                if(cnt_limit_rpm >= limit_rpm[openloop_dim]){
                    sync_def -= const_rpm[cnt_const_rpm];
                    cnt_const_rpm++;
                    cnt_limit_rpm = 0;
                };
            };
            sync_cnt = sync_def;
        };
    }else{                                              /* 切り替わり */
        pid_cnt = 0;
        if(cnt_flag != FLG_OFF){
            if(--cnt <= 0){
                sync_flag = FLG_OFF;
                cnt_flag  = FLG_OFF;
                o_data    = read_BEMF();
                set_RTPMO1(tr_sw[cw_ccw_flag][o_data]);
                int_cnt    = 0;
                speed_flag = FLG_ON;
                cnt_data3  = cnt_data2;
                cnt_data2  = cnt_data1;
                cnt_data1  = int_cnt;
                cnt_data   = cnt_data1 + cnt_data2 + cnt_data3;
            }else{
                if(r_data != read_BEMF()){
                    cnt_flag = FLG_OFF;
                };
            };
        }else{
            r_data = read_BEMF();
            if(o_data != r_data){ /* BEMF が変化した */
                cnt      = (int)(sync_def>>1);
                cnt_flag = FLG_ON;
            };
        };
    };
    if((stop_wait == FLG_OFF) && (cw_ccw_wait == FLG_OFF)){   /* 停止待ちでない       */
        if(pwm_ff != old_ff){                           /* デューティが変化した */
            tmp_pwm = pwm_base - pwm_ff;
```

```
            set_TWO(tmp_pwm, tmp_pwm, tmp_pwm, pwm_base); /* デューティ変更       */
            old_ff = pwm_ff;
        };
    }else{                              /* 停止待ちでも速度表示を続けるための処理 */
        set_RTPM01(P_OFF);                              /* 通電を停止           */
        r_data = read_BEMF();
        if(r_data != o_data){                           /*    */
            o_data     = r_data;
            cnt_data3  = cnt_data2;
            cnt_data2  = cnt_data1;
            cnt_data1  = int_cnt;
            cnt_data   = cnt_data1 + cnt_data2 + cnt_data3;
            if((10 < cnt_data1) && (cnt_data1 < 1200)){
                int_cnt   = 0;
                speed_flag = FLG_ON;
            };
        };
    };
    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;
}
```

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    target  : uPD78F0714 モータ・スタータ・キット

    date    : 2006/12/13
    filename: lib_eu.h

    NEC Micro Systems,Ltd
*/

#define KP_DEF              0.075           /* 0.075    */
#define KI_DEF              0.001           /* 0.001    */
#define KD_DEF              0.020           /* 0.020    */
#define PWM_LIMIT_H         50.0
#define PWM_LIMIT_L        -50.0

#define PWM_BASE_REF        769             /* PWM 周期の半周期                     */
#define PWM_F_REF           0               /* PWM 波形の初期値                     */
#define PWM_F_MIN           10              /* 最小値                               */
#define PWM_F_MAX           769             /* 最大値                               */
#define PWM_DTW_REF         0               /* デッド・タイム                       */

    /*
       m_speed 用定数: 400[rpm] * (0.15[s] / 12.8[us]) / 2
                               ----      ----       ---
                               ↑         ↑          ↑極対数
                               ↑         ↑TM00 カウントクロック 78.125[KHz](1/78.125[KHz])
                               ↑400[rpm]時の周期(1/(400/60))
    */
#define UNIT_RPM            2343750  /* 2count */

#define MIN_SPEED           200
#define MAX_SPEED           3200

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

#define CLEAR               0
#define SET                 1

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

#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  */
#define INTP0               PM00            /* 過電流検知ポート                     */
#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 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_0               0xc0            /* 0-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

#define IMS_DATA            0xc8

#define WDTM_OFF            0x77            /* ウォッチドッグ・タイマ制御用         */
#define WDTM_SET            0x67
#define WDTE_CLR            0xac
#define WDTE_RESET          0x00

#define KEY_WAIT            10              /* チャタリング除去 ms              */
#define SW                  (P7&0xf)        /* スイッチのポート                 */

#define START_SW            0x7             /* 押されているスイッチ判定          */
#define STOP_SW             0x7
#define FORWARD_SW          0xb
#define REVERSE_SW          0xd
#define MODE_SW             0xe

static void         INTPO_on(void);
static void         led_set(unsigned char, unsigned char);
static void         led_print(int, char);
static void         wait(int);
static void         INTTWOUD_on(void);
static void         INTTWOUD_off(void);
static void         init_openloop(void);
static void         init_PORT(void);
static void         init_OSC(void);
static void         init_TWO(void);
static void         start_TWO(void);
static void         stop_TWO(void);
static void         init_TWOO(void);
static void         start_TWOO(void);
static void         stop_TWOO(void);
static void         init_RTPWO1(void);
static void         start_RTPWO1(void);
static void         stop_RTPWO1(void);
static void         init_AD(void);
static void         start_AD(void);
static unsigned int get_AD(char);
static void         init_WDTM(void);
static void         init_TM51(void);
static void         start_TM51(void);
static void         stop_TM51(void);
static void         init_TM50(void);
static void         start_TM50(void);
static void         wait_TM50(void);
static void         INTSROO_on(void);
static void         INTSROO_off(void);
static void         INTP5_on(void);
static void         INTP5_off(void);
```

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    target  : uPD78F0714 モータ・スタータ・キット

    date    : 2006/12/13
    filename: lib_eu.c

    NEC Micro Systems,Ltd
*/

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

#pragma INTERRUPT INTP5         int_speed    rb3

#include "main.h"
#include "lib_eu.h"

const unsigned char sync_tbl[2][6] = {
```

```
                                    /* for LowVoltage  */
    {P_T1, P_T2, P_T3, P_T4, P_T5, P_T6}, /* CCW           */
    {P_T6, P_T5, P_T4, P_T3, P_T2, P_T1}  /* CW            */
};

char                sync_flag;
char                cnt_flag;
char                sync_sw;
unsigned int        sync_cnt;
unsigned int        sync_def;

char                openloop_dim;
unsigned int        limit_tim[2];
unsigned int        limit_pwm[2];
unsigned int        limit_rpm[2];

unsigned char       const_rpm[70];   /* 100[rpm]から 800[rpm]までを想定して配列を決めた。 */

unsigned int        cnt_limit_tim;
unsigned int        cnt_limit_pwm;
unsigned int        cnt_limit_rpm;
unsigned int        cnt_const_rpm;

float               carrier_freq;

unsigned int        t_0, t_mid, t_fin;
unsigned int        pwm_t0, pwm_tmid, pwm_tfin;
unsigned int        rpm_t0, rpm_tmid, rpm_tfin;
unsigned char       rpm_step;

int                 pwm_ff;                         /* アクティブ幅：0 ～ 1000      */
int                 old_ff;                         /* PWM 履歴                      */
int                 pwm_base = PWM_BASE_REF;        /* PWM の半周期：1000 固定       */
int                 speed_ref = 200;                /* 指定速度                      */
int                 m_speed;                        /* モータ回転速度                */
unsigned int        int_cnt;                        /* キャリア同期割り込み回数      */
unsigned int        pid_cnt;                        /* PID 制御周期確認用            */
unsigned int        wait_cnt;                       /* 処理間隔確認用                */
unsigned int        print_cnt;                      /* 速度表示間隔確認用            */
unsigned int        cnt_data1, cnt_data2, cnt_data3; /* キャリア同期割り込み回数履歴 */
unsigned int        cnt_data;
char                cw_ccw_flag = CW;               /* 回転方向                      */
char                start_flag;                     /* 回転開始直後の判定            */
char                speed_flag;                     /* 速度情報の有無                */
char                cw_ccw_wait;                    /* 反転の停止待ち                */
char                sys_flag;                       /* システム稼動状況              */
char                stop_wait;                      /* 停止待ちの有無                */
char                ad_flag     = FLG_OFF;          /* 指定速度変更機能制限          */

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

/*
    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 char         up_flag = FLG_OFF;              /* 速度更新確認フラグ            */
static float        kp_ref = KP_DEF;                /* Kp 初期値                      */
static float        ki_ref = KI_DEF;                /* Ki 初期値                      */
static float        kd_ref = KD_DEF;                /* Kd 初期値                      */
static float        mvn;                            /* 操作量                        */
static float        en, en_1, en_2;                 /* 偏差                          */

/*
    初期設定
*/
void
system_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TM0();
    init_TM00();
    init_RTPM01();
    init_AD();
    init_TM50();
    init_openloop();
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(1000);
    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
system_start(void)
{
    m_speed      = 0;
    en_1         =
    en           =
    mvn          = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 1000;
    cnt_data     = 3000;
    sys_flag     = FLG_ON;
    start_flag   = FLG_ON;
    stop_wait    = FLG_OFF;
    cw_ccw_wait  = FLG_OFF;
    speed_flag   = FLG_OFF;
    up_flag      = FLG_OFF;
    clk_flag     = FLG_OFF;
    capture_flag = FLG_OFF;
    cnt_flag     = FLG_OFF;
    sync_flag    = FLG_START;
    sync_sw      = 1;
    sync_def     = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt     = sync_def;
    openloop_dim = 0;
    cnt_limit_tim = 0;
    cnt_limit_pwm = 0;
    cnt_limit_rpm = 0;
    cnt_const_rpm = 0;
    pwm_ff        = (int)pwm_t0;
    old_ff        = 0;
    start_TM00();
    INTTM00UD_on();       /* キャリア同期割り込みマスク解除       */
    INVERTER_SW  = INV_ON;   /* INVERTER enable        */
    start_RTPM01();
    start_TM0();
    set_TM0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
    INTP5_on();
}

/*
    反転停止からの再起動
*/
void
system_restart(void)
{
    m_speed      = 0;
    mvn          =
    en_1         =
    en           = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 1000;
    cnt_data     = 3000;
    start_flag   = FLG_ON;
    speed_flag   = FLG_OFF;
    up_flag      = FLG_OFF;
    clk_flag     = FLG_OFF;
    capture_flag = FLG_OFF;
    cnt_flag     = FLG_OFF;
    sync_flag    = FLG_START;
    sync_sw      = 1;
    sync_def     = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt     = sync_def;
    openloop_dim = 0;
    cnt_limit_tim = 0;
    cnt_limit_pwm = 0;
    cnt_limit_rpm = 0;
    cnt_const_rpm = 0;
    pwm_ff        = (int)pwm_t0;
    old_ff       = 0;
    cw_ccw_wait  = FLG_OFF;
    set_TM0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
```

```
    wait(400);
}

/*
    システム停止
*/
void
system_stop(void)
{
    INVERTER_SW  = INV_OFF;         /* INVERTER disable        */
    INTTWOUD_off();
    INTP5_off();
    stop_RTPWO1();
    stop_TWO();
    stop_TWOO();
    sys_flag     = FLG_OFF;
    stop_wait    = FLG_OFF;
    cw_ccw_flag  = CW;
    m_speed      = 0;                       /* 速度 0            */
}

/*

*/
void
init_openloop(void)
{
    unsigned char i;
    unsigned int  tmp_rpm;

    rpm_step = 10;

    t_0   = 0;
    t_mid = 2;
    t_fin = 4;

    rpm_t0   = 100;
    rpm_tmid = 300;
    rpm_tfin = 500;

    pwm_t0   = 50;
    pwm_tmid = 55;
    pwm_tfin = 60;

    carrier_freq = (float)((PWM_BASE_REF * 2.0) / (20.0 * 1000 * 1000));

    limit_tim[0] = (unsigned int)((float)(t_mid - t_0) / (float)carrier_freq);
    limit_pwm[0] = (limit_tim[0] / (unsigned int)(pwm_tmid - pwm_t0));
    limit_rpm[0] = (limit_tim[0] / (unsigned int)((rpm_tmid - rpm_t0) / rpm_step));
    limit_tim[1] = (unsigned int)((float)(t_fin - t_mid) / (float)carrier_freq);
    limit_pwm[1] = (limit_tim[1] / (unsigned int)(pwm_tfin - pwm_tmid));
    limit_rpm[1] = (limit_tim[1] / (unsigned int)((rpm_tfin - rpm_tmid) / rpm_step));

    tmp_rpm = rpm_t0;
    for(i = 0; i<=69; i++) {
        const_rpm[i] = (unsigned char)( (float)(5.0 / carrier_freq)
                                        * ( (float)(1.0 / (double)tmp_rpm)
                                          - (float)(1.0 / (double)(tmp_rpm + rpm_step)) )
                                      );
        tmp_rpm += rpm_step;
    };
}

/*
    過電流割り込み許可
*/
static void
INTPO_on(void)
{
    PROL.1 = 0;                         /* 優先順位          */
    EGN.0  = 1;                         /* 立ち下がりエッジ */
    IFOL.1 = CLEAR;                     /* フラグクリア      */
    MKOL.1 = CLEAR;                     /* マスククリア      */
}

/*
    エラー表示
*/
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();
}

/*
    PID 制御
*/
void
pwm_pid(int cy_time)
{
    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){
            up_flag = FLG_ON;       /* 速度更新済み */
            if(sync_flag == FLG_OFF){
                if(clk_flag == FLG_ON){
                    clk_flag = FLG_OFF;
                    DI();
                    old_tmp = (unsigned long)old_clk;
                    new_tmp = (unsigned long)new_clk;
                    EI();
                    if(old_tmp > new_tmp){
                        tmp = (65536 - old_tmp) + new_tmp;
                    }else{
                        tmp = new_tmp - old_tmp;
                    }
                    m_speed = (int)(UNIT_RPM/tmp);
                }else{
                    up_flag = FLG_OFF;
                };
            }else{
                m_speed = (int)(194400/(long)cnt_data);
            };
        }else{
            start_flag = FLG_OFF;   /* 先頭の速度情報は使用しない */
        }
    }

    if(pid_cnt >= (unsigned int)(cy_time*13)){   /* cy_time*1ms 経過                    */
        pid_cnt = 0;
        if(up_flag == FLG_ON){     /* 速度更新あり                          */
            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                  */
                up_flag = FLG_OFF;
                en_2 = en_1;
                en_1 = en;
                en   = (float)(speed_ref - m_speed);
                mvn  = mvn +
                       kp_ref*(en - en_1) +
                       ki_ref*en +
                       kd_ref*((en - en_1) - (en_1 - en_2));
                if(mvn > PWM_LIMIT_H){
                    mvn = PWM_LIMIT_H;
                }else if(mvn < (float)PWM_LIMIT_L){
                    mvn = (float)PWM_LIMIT_L;
                }
                pwm_tmp  = pwm_ff + (int)mvn;
                if(pwm_tmp > PWM_F_MAX){ /* リミッタ                          */
                    pwm_ff = PWM_F_MAX;
                }else if(pwm_tmp < PWM_F_MIN){
                    pwm_ff = PWM_F_MIN;
                }else{
                    pwm_ff = pwm_tmp;
                }
            }
        }
```

```
    }
}

/*
    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
*/

void
get_speed(void)
{
    unsigned int  data;
    unsigned long tmp;

    if(ad_flag == FLG_OFF){
        data = get_AD(2);
        data = ((data - 3) * 12) + MIN_SPEED;
        if(data > MAX_SPEED){
            data = MAX_SPEED;
        }else if(data < MIN_SPEED) {
            data = MIN_SPEED;
        };
        tmp = (UNIT_RPM / (unsigned long)data);
        speed_ref = (int)(UNIT_RPM / tmp);
    }
}

/*
    スイッチ(P7)と UART からの値読み出し

    スイッチの値を読み出す
    一定時間チェックして安定している値を返す


*/
unsigned char
get_sw(void)
{
    unsigned char data;

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

    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);
}

/*
    ポートの設定
*/
static void
init_PORT(void)
{
    INTP0 = IN;                         /* 過電流通知の割り込み */
    SW2 = IN;                           /* START/STOP           */
```

```
    SW3 = IN;                       /* FORWARD              */
    SW4 = IN;                       /* REVERSE              */
    SW5 = IN;                       /* MODE                 */

    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;

    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 */
}

/*
    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 */
    clear_WDTM();
    while(OSTC != 0x1f);            /* 発振安定待ち */
    PCC   = 0x00;                   /* 分周比設定 */
    MCM.0 = 1;                      /* X1 入力クロック */
    VSWC.1 = 1;
}

/*
    インバータタイマ
*/
static void
init_TW0(void)
{
    TCL02    = 0;       /* カウント・クロック:20MHz */
    TCL01    = 0;
    TCL00    = 0;

    IDEV02   = 0;       /* 毎回割り込み発生         */
    IDEV01   = 0;
    IDEV00   = 0;
    TW0N     = 0;
    TW0TRGS  = 0;
    TW0OC    = 0;
    TW0CM3   = PWM_BASE_REF;
    TW0CM0   = PWM_BASE_REF - PWM_F_REF;
    TW0CM1   = PWM_BASE_REF - PWM_F_REF;
    TW0CM2   = PWM_BASE_REF - PWM_F_REF;
    TW0DTIME = PWM_DTM_REF;             /* デッド・タイム                       */
    TW0BFCM3 = PWM_BASE_REF;
    TW0BFCM0 = PWM_BASE_REF - PWM_F_REF;
    TW0BFCM1 = PWM_BASE_REF - PWM_F_REF;
    TW0BFCM2 = PWM_BASE_REF - PWM_F_REF;
    TW0TRGS  = 1;
}

void
set_TW0(int u, int v, int w, int base)
{
    TW0BFCM3 = (unsigned int)base;
    TW0BFCM0 = (unsigned int)u;
    TW0BFCM1 = (unsigned int)v;
    TW0BFCM2 = (unsigned int)w;
}

static void
start_TW0(void)
{
    TW0C.7 = SET;                   /* タイマスタート   */
}

static void
stop_TW0(void)
{
    TW0C.7 = CLEAR;                 /* タイマストップ   */
}

/*
    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
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                  /* 通電パターン設定           */
    CR01   = TM00 + 1;
}

static void
start_RTPM01(void)
{
    RTPC01.7 = SET;                 /* 動作許可 */
}

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

/*
    AD
*/
static void
init_AD(void)
{
    ADS   = 4;                      /* SPEED */
    ADM   = 0x1a;                   /* 3.6us */
    ADCS2 = SET;                    /* 回路動作許可 */
}

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         */
}

static unsigned int
get_AD(char s_bit)
{
    return((((unsigned int)ADCR)>>(s_bit+6))&0x3ff);
}

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

void
clear_WDTM(void)
{
    WDTE = WDTE_CLR;
}


void
reset_WDTM(void)
{
```

```
    WDTE = WDTE_RESET;
}


/*
    8 ビット・タイマ 50
*/
static 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                   */
}

/*
    速度表示
*/
void
speed_print(int ms)
{
    if((MODE_SW == SW) || (sys_flag == FLG_OFF)){
        led_print(speed_ref, ad_flag);
    }else{
        if(print_cnt > (unsigned int)(ms*13)){
            led_print(m_speed, FLG_OFF);
            print_cnt = 0;
        }
    }
}

/*
    キャリア同期割り込み許可
*/
static void
INTTWOUD_on(void)
{
    UDIFW0 = CLEAR;                     /* 要求フラグクリア */
    UDMKW0 = CLEAR;                     /* 割り込み許可     */
}

/*
    キャリア同期割り込み禁止
*/
static void
INTTWOUD_off(void)
{
    UDMKW0 = SET;                       /* 割り込み禁止 */
    UDIFW0 = CLEAR;
}

/*
    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;
}

/*
    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;
}

/*
    BEMF 信号の読み出し
*/
char
read_BEMF(void)
{
    return((char)((P0>>1)&0x7));
}

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

# 付　　録

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す。

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    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    : 2006/12/13
    filename: lib_eu.h

    NEC Micro Systems,Ltd
*/

#define KP_DEF              0.075           /* 0.075    */
#define KI_DEF              0.001           /* 0.001    */
#define KD_DEF              0.020           /* 0.020    */
#define PWM_LIMIT_H         50.0
#define PWM_LIMIT_L        -50.0

#define PWM_BASE_REF        769             /* PWM 周期の半周期                    */
#define PWM_F_REF           0               /* PWM 波形の初期値                    */
#define PWM_F_MIN           10              /* 最小値                              */
#define PWM_F_MAX           769             /* 最大値                              */
#define PWM_DTM_REF         0               /* デッド・タイム                      */

    /*
       m_speed 用定数: 400[rpm] * (0.15[s] / 12.8[us]) / 2
                                  ----      ----       ---
                                  ↑         ↑          ↑極対数
                                  ↑         ↑TM00 カウントクロック 78.125[KHz](1/78.125[KHz])
                                  ↑400[rpm]時の周期(1/(400/60))
    */
#define UNIT_RPM            2343750  /* 2count */

#define MIN_SPEED           200
#define MAX_SPEED           3200

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

#define CLEAR               0
#define SET                 1

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

#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  */
#define INTP0               PM00            /* 過電流検知ポート                    */
#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 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

#define IMS_DATA            0xc8

#define WDTM_OFF            0x77            /* ウォッチドッグ・タイマ制御用         */
#define WDTM_SET            0x67
#define WDTE_CLR            0xac
#define WDTE_RESET          0x00

#define KEY_WAIT            10              /* チャタリング除去 ms                  */
#define SW                  (P7&0xf)        /* スイッチのポート                     */

#define START_SW            0x7             /* 押されているスイッチ判定             */
#define STOP_SW             0x7
#define FORWARD_SW          0xb
#define REVERSE_SW          0xd
#define MODE_SW             0xe

#ifdef GUI
#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_GETSPEED     0x50            /* 実速度読み出し   */
#define MY_ID               0x21            /* 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
#endif

static void         INTP0_on(void);
static void         led_set(unsigned char, unsigned char);
static void         led_print(int, char);
static void         wait(int);
static void         INTTW0UD_on(void);
static void         INTTW0UD_off(void);
static void         init_openloop(void);
static void         init_PORT(void);
static void         init_OSC(void);
static void         init_TW0(void);
static void         start_TW0(void);
static void         stop_TW0(void);
static void         init_TM00(void);
static void         start_TM00(void);
static void         stop_TM00(void);
static void         init_RTPM01(void);
static void         start_RTPM01(void);
static void         stop_RTPM01(void);
static void         init_AD(void);
static void         start_AD(void);
static unsigned int get_AD(char);
static void         init_WDTM(void);
static void         init_TM51(void);
static void         start_TM51(void);
static void         stop_TM51(void);
static void         init_TM50(void);
static void         start_TM50(void);
static void         wait_TM50(void);
static void         INTSR00_on(void);
static void         INTSR00_off(void);
static void         INTP5_on(void);
static void         INTP5_off(void);

#ifdef GUI
static void          init_UART00(void);
static unsigned char get_UART00(void);
static unsigned char wait_UART00(void);
static void          set_UART00(unsigned char);
#endif
```

```
/*
    BLDC モータ 120 度通電方式 位置センサレス(BEMF: CMP)

    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    : 2006/12/13
    filename: lib_eu.c

    NEC Micro Systems,Ltd
*/

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

#pragma INTERRUPT INTP5         int_speed    rb3

#ifdef GUI
#pragma INTERRUPT INTSR00       int_uart00  rb1   /* UART00 コマンド受信用     */
#endif

#include "main.h"
#include "lib_eu.h"

const unsigned char sync_tbl[2][6] = {
                                    /* for LowVoltage  */
    {P_T1, P_T2, P_T3, P_T4, P_T5, P_T6}, /* CCW           */
    {P_T6, P_T5, P_T4, P_T3, P_T2, P_T1}  /* CW            */
};

char                sync_flag;
char                cnt_flag;
char                sync_sw;
unsigned int        sync_cnt;
unsigned int        sync_def;

char                openloop_dim;
unsigned int        limit_tim[2];
unsigned int        limit_pwm[2];
unsigned int        limit_rpm[2];

unsigned char       const_rpm[70];   /* 100[rpm]から 800[rpm]までを想定して配列を決めた。 */

unsigned int        cnt_limit_tim;
unsigned int        cnt_limit_pwm;
unsigned int        cnt_limit_rpm;
unsigned int        cnt_const_rpm;

float               carrier_freq;

unsigned int        t_0, t_mid, t_fin;
unsigned int        pwm_t0, pwm_tmid, pwm_tfin;
unsigned int        rpm_t0, rpm_tmid, rpm_tfin;
unsigned char       rpm_step;

int                 pwm_ff;                         /* アクティブ幅：0 ～ 1000      */
int                 old_ff;                         /* PWM 履歴                      */
int                 pwm_base = PWM_BASE_REF;        /* PWM の半周期：1000 固定       */
int                 speed_ref = 200;                /* 指定速度                      */
int                 m_speed;                        /* モータ回転速度                */
unsigned int        int_cnt;                        /* キャリア同期割り込み回数      */
unsigned int        pid_cnt;                        /* PID 制御周期確認用            */
unsigned int        wait_cnt;                       /* 処理間隔確認用                */
unsigned int        print_cnt;                      /* 速度表示間隔確認用            */
unsigned int        cnt_data1, cnt_data2, cnt_data3; /* キャリア同期割り込み回数履歴 */
unsigned int        cnt_data;
char                cw_ccw_flag = CW;               /* 回転方向                      */
char                start_flag;                     /* 回転開始直後の判定            */
char                speed_flag;                     /* 速度情報の有無                */
char                cw_ccw_wait;                    /* 反転の停止待ち                */
char                sys_flag;                       /* システム稼動状況              */
char                stop_wait;                      /* 停止待ちの有無                */
char                ad_flag     = FLG_OFF;          /* 指定速度変更機能制限          */

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

/*
    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 char         up_flag = FLG_OFF;              /* 速度更新確認フラグ            */
static float        kp_ref = KP_DEF;                /* Kp 初期値                      */
static float        ki_ref = KI_DEF;                /* Ki 初期値                      */
static float        kd_ref = KD_DEF;                /* Kd 初期値                      */
static float        mvn;                            /* 操作量                        */
```

```
static float       en, en_1, en_2;                 /* 偏差                         */
#ifdef GUI
static unsigned char err_flag = FLG_OFF;           /* エラー通知フラグ             */
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 int           kp_tmp, ki_tmp, kd_tmp;       /* Kp,Ki,Kd の通信データ        */
#endif

/*
    初期設定
*/
void
system_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TW0();
    init_TM00();
    init_RTPM01();
    init_AD();
    init_TM50();
    init_openloop();
#ifdef GUI
    init_UART00();
    led_set(0, LED_P);
    led_set(1, LED_C);
    kp_tmp = (int)(kp_ref * 10000);
    ki_tmp = (int)(ki_ref * 10000);
    kd_tmp = (int)(kd_ref * 10000);
#else
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(1000);
#endif
    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
system_start(void)
{
    m_speed      = 0;
    en_1         =
    en           =
    mvn          = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 1000;
    cnt_data     = 3000;
    sys_flag     = FLG_ON;
    start_flag   = FLG_ON;
    stop_wait    = FLG_OFF;
    cw_ccw_wait  = FLG_OFF;
    speed_flag   = FLG_OFF;
    up_flag      = FLG_OFF;
    clk_flag     = FLG_OFF;
    capture_flag = FLG_OFF;
    cnt_flag     = FLG_OFF;
    sync_flag    = FLG_START;
    sync_sw      = 1;
    sync_def     = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt     = sync_def;
    openloop_dim = 0;
    cnt_limit_tim = 0;
    cnt_limit_pwm = 0;
    cnt_limit_rpm = 0;
    cnt_const_rpm = 0;
    pwm_ff       = (int)pwm_t0;
    old_ff       = 0;
    start_TM00();
    INTTWOUD_on();       /* キャリア同期割り込みマスク解除       */
    INVERTER_SW  = INV_ON;   /* INVERTER enable        */
    start_RTPM01();
    start_TW0();
    set_TW0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
    INTP5_on();
```

```
}

/*
    反転停止からの再起動
*/
void
system_restart(void)
{
    m_speed      = 0;
    mvn          =
    en_1         =
    en           = 0.0;
    int_cnt      =
    pid_cnt      =
    print_cnt    = 0;
    cnt_data1    =
    cnt_data2    =
    cnt_data3    = 1000;
    cnt_data     = 3000;
    start_flag   = FLG_ON;
    speed_flag   = FLG_OFF;
    up_flag      = FLG_OFF;
    clk_flag     = FLG_OFF;
    capture_flag = FLG_OFF;
    cnt_flag     = FLG_OFF;
    sync_flag    = FLG_START;
    sync_sw      = 1;
    sync_def     = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt     = sync_def;
    openloop_dim = 0;
    cnt_limit_tim = 0;
    cnt_limit_pwm = 0;
    cnt_limit_rpm = 0;
    cnt_const_rpm = 0;
    pwm_ff        = (int)pwm_t0;
    old_ff       = 0;
    cw_ccw_wait  = FLG_OFF;
    set_TWO(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPWO1(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPWO1(sync_tbl[cw_ccw_flag][0]);
    wait(400);
}

/*
    システム停止
*/
void
system_stop(void)
{
    INVERTER_SW  = INV_OFF;         /* INVERTER disable       */
    INTTWOUD_off();
    INTP5_off();
    stop_RTPWO1();
    stop_TWO();
    stop_TWOO();
    sys_flag     = FLG_OFF;
    stop_wait    = FLG_OFF;
    cw_ccw_flag  = CW;
    m_speed      = 0;                       /* 速度 0           */
}

/*

*/
void
init_openloop(void)
{
    unsigned char i;
    unsigned int  tmp_rpm;

    rpm_step = 10;

    t_0   = 0;
    t_mid = 2;
    t_fin = 4;

    rpm_t0   = 100;
    rpm_tmid = 300;
    rpm_tfin = 500;

    pwm_t0   = 50;
    pwm_tmid = 55;
    pwm_tfin = 60;

    carrier_freq = (float)((PWM_BASE_REF * 2.0) / (20.0 * 1000 * 1000));

    limit_tim[0] = (unsigned int)((float)(t_mid - t_0) / (float)carrier_freq);
    limit_pwm[0] = (limit_tim[0] / (unsigned int)(pwm_tmid - pwm_t0));
    limit_rpm[0] = (limit_tim[0] / (unsigned int)((rpm_tmid - rpm_t0) / rpm_step));
    limit_tim[1] = (unsigned int)((float)(t_fin - t_mid) / (float)carrier_freq);
```

```
    limit_pwm[1] = (limit_tim[1] / (unsigned int)(pwm_tfin - pwm_tmid));
    limit_rpm[1] = (limit_tim[1] / (unsigned int)((rpm_tfin - rpm_tmid) / rpm_step));

    tmp_rpm = rpm_t0;
    for(i = 0; i<=69; i++) {
        const_rpm[i] = (unsigned char)( (float)(5.0 / carrier_freq)
                                        * ( (float)(1.0 / (double)tmp_rpm)
                                          - (float)(1.0 / (double)(tmp_rpm + rpm_step)) )
                                      );
        tmp_rpm += rpm_step;
    };
}

/*
    過電流割り込み許可
*/
static 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     */
}

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

    case ERROR_OC:
#ifdef GUI
        err_flag = MD_ERROR_OC;
#endif
        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:
#ifdef GUI
        err_flag = MD_ERROR_MOTOR;
#endif
        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:
#ifdef GUI
        err_flag = MD_ERROR_OC;
#endif
        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:
#ifdef GUI
        err_flag = MD_ERROR_MOTOR;
#endif
        led_set(0, LED_F);
        led_set(1, LED_A);
        led_set(2, LED_I);
        led_set(3, LED_L);
    }
#ifndef GUI
    STOP();
#endif
}

/*
    PID制御
*/
void
pwm_pid(int cy_time)
{
    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){
            up_flag = FLG_ON;       /* 速度更新済み */
            if(sync_flag == FLG_OFF){
                if(clk_flag == FLG_ON){
                    clk_flag = FLG_OFF;
                    DI();
                    old_tmp = (unsigned long)old_clk;
                    new_tmp = (unsigned long)new_clk;
                    EI();
                    if(old_tmp > new_tmp){
                        tmp = (65536 - old_tmp) + new_tmp;
                    }else{
                        tmp = new_tmp - old_tmp;
                    }
                    m_speed = (int)(UNIT_RPM/tmp);
                }else{
                    up_flag = FLG_OFF;
                };
            }else{
                m_speed = (int)(194400/(long)cnt_data);
            };
        }else{
            start_flag = FLG_OFF;   /* 先頭の速度情報は使用しない */
        }
    }

    if(pid_cnt >= (unsigned int)(cy_time*13)){   /* cy_time*1ms 経過                    */
        pid_cnt = 0;
        if(up_flag == FLG_ON){     /* 速度更新あり                          */
            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                  */
                up_flag = FLG_OFF;
                en_2 = en_1;
                en_1 = en;
                en   = (float)(speed_ref - m_speed);
                mvn  = mvn +
                       kp_ref*(en - en_1) +
                       ki_ref*en +
                       kd_ref*((en - en_1) - (en_1 - en_2));
                if(mvn > PWM_LIMIT_H){
                    mvn = PWM_LIMIT_H;
                }else if(mvn < (float)PWM_LIMIT_L){
                    mvn = (float)PWM_LIMIT_L;
                }
                pwm_tmp  = pwm_ff + (int)mvn;
                if(pwm_tmp > PWM_F_MAX){ /* リミッタ                          */
                    pwm_ff = PWM_F_MAX;
                }else if(pwm_tmp < PWM_F_MIN){
                    pwm_ff = PWM_F_MIN;
                }else{
                    pwm_ff = pwm_tmp;
                }
            }
        }
    }
}

/*
    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
*/

void
get_speed(void)
{
#ifndef GUI
    unsigned int  data;
    unsigned long tmp;

    if(ad_flag == FLG_OFF){
        data = get_AD(2);
        data = ((data - 3) * 12) + MIN_SPEED;
        if(data > MAX_SPEED){
            data = MAX_SPEED;
        }else if(data < MIN_SPEED) {
            data = MIN_SPEED;
        };
        tmp = (UNIT_RPM / (unsigned long)data);
        speed_ref = (int)(UNIT_RPM / tmp);
    }
#endif
}

/*
    スイッチ(P7)と UART からの値読み出し

    スイッチの値を読み出す
    一定時間チェックして安定している値を返す
```

```
*/
unsigned char
get_sw(void)
{
    unsigned char data;

#ifndef GUI
    int  i;
    unsigned char tmp;
    static unsigned char start_stop_flag = FLG_OFF;

    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 */
    }
#else
    int          ss;
    unsigned char hi, lo;

    data = get_UART00();
    switch(data){
    case MD_CMD_GETID:                  /* Get ID */
        set_UART00(data);
        set_UART00(MY_ID);
        break;

    case MD_CMD_GETVER:                 /* Get Version */
        set_UART00(data);
        set_UART00(VER_MAJOR);
        set_UART00(VER_MINOR);
        set_UART00(VER_SEQ);
        break;

    case MD_CMD_START:                  /* Start */
        set_UART00(data);
        data = START_TR;
        break;

    case MD_CMD_STOP:                   /* Stop */
        set_UART00(data);
        data = STOP_TR;
        break;

    case MD_CMD_RESET:                  /* Reset */
        set_UART00(data);
        while(STIF00 != SET);
        reset_WDTM();
        break;

    case MD_CMD_SETSSPEED:              /* Set Setting Speed */
        lo = wait_UART00();
        hi = wait_UART00();
        set_UART00(data);
        ss = ((int)hi<<8) + lo;
        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 = ss;
        break;

    case MD_CMD_GETSSPEED:              /* Get Setting Speed */
        set_UART00(data);
        ss = speed_ref;
        if(cw_ccw_flag == CCW){
            ss = ~ss + 1;
        }
        lo = (unsigned char)((unsigned int)(ss)&0xff);
        hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
        set_UART00(lo);
        set_UART00(hi);
        break;

    case MD_CMD_SETPIDPARAM:            /* PIDのフィードバックゲイン設定 */
        lo = wait_UART00();
        hi = wait_UART00();
        kp_tmp = (((int)(hi))<<8) + lo;
        lo = wait_UART00();
        hi = wait_UART00();
        ki_tmp = (((int)(hi))<<8) + lo;
        lo = wait_UART00();
        hi = wait_UART00();
        kd_tmp = (((int)(hi))<<8) + lo;
        set_UART00(data);
        kp_ref = ((float)kp_tmp)/10000;
        ki_ref = ((float)ki_tmp)/10000;
        kd_ref = ((float)kd_tmp)/10000;
        break;

    case MD_CMD_GETPIDPARAM:            /* PIDのフィードバックゲイン読み出し */
        set_UART00(data);
        kp_tmp = ((int)kp_ref)*10000;
        set_UART00((unsigned char)((unsigned int)(kp_tmp)&0xff));
        set_UART00((unsigned char)(((unsigned int)(kp_tmp)>>8)&0xff));
        ki_tmp = ((int)ki_ref)*10000;
        set_UART00((unsigned char)((unsigned int)(ki_tmp)&0xff));
        set_UART00((unsigned char)(((unsigned int)(ki_tmp)>>8)&0xff));
        kd_tmp = ((int)kd_ref)*10000;
        set_UART00((unsigned char)((unsigned int)(kd_tmp)&0xff));
        set_UART00((unsigned char)((unsigned int)((kd_tmp)>>8)&0xff));
        break;

    case MD_CMD_GETSPEED:               /* 回転速度読み出し */
        if(err_flag != FLG_OFF){
            set_UART00(err_flag);
            err_flag = FLG_OFF;
        }else{
            set_UART00(data);
            if(m_speed == 0){
                set_UART00(0);
                set_UART00(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;
                    }
                }
                set_UART00((unsigned char)((unsigned int)(ss)&0xff));
                set_UART00((unsigned char)(((unsigned int)(ss)>>8)&0xff));
            }
        }
        break;

    default:
        ;
    }
#endif
    return(data);
}

/*
    ポートの設定
*/
static void
init_PORT(void)
{
    INTP0 = IN;                         /* 過電流通知の割り込み */
    SW2 = IN;                           /* START/STOP           */
```

```
    SW3 = IN;                       /* FORWARD              */
    SW4 = IN;                       /* REVERSE              */
    SW5 = IN;                       /* MODE                 */

    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;

    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 */
}

/*
    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 */
    clear_WDTM();
    while(OSTC != 0x1f);            /* 発振安定待ち */
    PCC   = 0x00;                   /* 分周比設定 */
    MCM.0 = 1;                      /* X1 入力クロック */
    VSWC.1 = 1;
}

/*
    インバータタイマ
*/
static void
init_TW0(void)
{
    TCL02    = 0;       /* カウント・クロック:20MHz */
    TCL01    = 0;
    TCL00    = 0;

    IDEV02   = 0;       /* 毎回割り込み発生         */
    IDEV01   = 0;
    IDEV00   = 0;
    TW0M     = 0;
    TW0TRGS  = 0;
    TW0OC    = 0;
    TW0CM3   = PWM_BASE_REF;
    TW0CM0   = PWM_BASE_REF - PWM_F_REF;
    TW0CM1   = PWM_BASE_REF - PWM_F_REF;
    TW0CM2   = PWM_BASE_REF - PWM_F_REF;
    TW0DTIME = PWM_DTM_REF;             /* デッド・タイム                   */
    TW0BFCM3 = PWM_BASE_REF;
    TW0BFCM0 = PWM_BASE_REF - PWM_F_REF;
    TW0BFCM1 = PWM_BASE_REF - PWM_F_REF;
    TW0BFCM2 = PWM_BASE_REF - PWM_F_REF;
    TW0TRGS  = 1;
}

void
set_TW0(int u, int v, int w, int base)
{
    TW0BFCM3 = (unsigned int)base;
    TW0BFCM0 = (unsigned int)u;
    TW0BFCM1 = (unsigned int)v;
    TW0BFCM2 = (unsigned int)w;
}

static void
start_TW0(void)
{
    TW0C.7 = SET;                   /* タイマスタート   */
}

static void
stop_TW0(void)
{
    TW0C.7 = CLEAR;                 /* タイマストップ   */
}

/*
    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
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                  /* 通電パタン設定           */
    CR01   = TM00 + 1;
}

static void
start_RTPM01(void)
{
    RTPC01.7 = SET;                 /* 動作許可 */
}

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

/*
    AD
*/
static void
init_AD(void)
{
    ADS   = 4;                      /* SPEED */
    ADM   = 0x1a;                   /* 3.6us */
    ADCS2 = SET;                    /* 回路動作許可 */
}

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         */
}

static unsigned int
get_AD(char s_bit)
{
    return((((unsigned int)ADCR)>>(s_bit+6))&0x3ff);
}

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

void
clear_WDTM(void)
{
    WDTE = WDTE_CLR;
}


void
reset_WDTM(void)
{
```

```
    WDTE = WDTE_RESET;
}


/*
    8 ビット・タイマ 50
*/
static 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                   */
}

/*
    速度表示
*/
void
speed_print(int ms)
{
    if((MODE_SW == SW) || (sys_flag == FLG_OFF)){
        led_print(speed_ref, ad_flag);
    }else{
        if(print_cnt > (unsigned int)(ms*13)){
            led_print(m_speed, FLG_OFF);
            print_cnt = 0;
        }
    }
}

/*
    キャリア同期割り込み許可
*/
static void
INTTWOUD_on(void)
{
    UDIFW0 = CLEAR;                     /* 要求フラグクリア */
    UDMKW0 = CLEAR;                     /* 割り込み許可     */
}

/*
    キャリア同期割り込み禁止
*/
static void
INTTWOUD_off(void)
{
    UDMKW0 = SET;                       /* 割り込み禁止 */
    UDIFW0 = CLEAR;
}

/*
    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;
}

/*
    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;
}

/*
    BEMF 信号の読み出し
*/
char
read_BEMF(void)
{
    return((char)((P0>>1)&0x7));
}

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

#ifdef GUI
/*
    UART00 の設定
*/
static void
init_UART00(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;
}

/*
    UART00の受信バッファデータを返す
*/
static unsigned char
get_UART00(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
wait_UART00(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
set_UART00(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_uart00(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         */
}
#endif
```

〔メ　モ〕

【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。