To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

应用笔记

μPD78F0714 应用电机控制

霍尔传感器 180° 换相方式

μPD78F0714

文档编号. U18913CA1V0AN00 (第 1 版)
发行日期 2009 年 5 月 N

日本印刷

[备忘录]

CMOS 设备注意事项

① 输入引脚处的电压波形

输入噪音或一个反射波引起的波形失真可能导致错误发生。如果由于噪音等的影响使CMOS设备的输入电压范围保持在VIL(MAX)和VIH(MIN)之间，设备可能发生错误。在输入电平固定时以及输入电平从VIL(MAX)过渡到VIH (MIN)时的传输期间，要防止散射噪声影响设备。

② 未使用的输入引脚的处理

CMOS设备的输入端保持开路可能导致误操作。如果一个输入引脚未被连接，则由于噪音等原因可能会产生内部输入电平，从而导致误操作。CMOS设备的操作特性与Bipolar或NMOS设备不同。CMOS设备的输入电平必须借助上拉或下拉电路固定在高电平或低电平。每一个未使用引脚都应该通过附加电阻连接到VDD或GND。如果有可能尽量定义为输出引脚。对未使用引脚的处理因设备而异，必须遵循与设备相关的规定和说明。

③ ESD防护措施

如果MOS设备周围有强电场，将会击穿氧化栅极，从而影响设备的运行。因此必须采取措施，尽可能防止静电产生。一旦有静电，必须立即释放。对于环境必须有适当的控制。如果空气干燥，应当使用增湿器。建议避免使用容易产生静电的绝缘体。半导体设备的存放和运输必须使用抗静电容器、抗静电屏蔽袋或导电材料容器。所有的测试和测量工具包括工作台和工作面必须良好接地。操作员应当佩戴静电消除手带以保证良好接地。不能用手直接接触半导体设备。对于装配有半导体设备的PW板也应采取类似的静电防范措施。

④ 初始化之前的状态

在上电时MOS设备的初始状态是不确定的。在刚刚上电之后，具有复位功能的MOS设备并没有被初始化。因此上电不能保证输出引脚的电平，I/O设置和寄存器的内容。设备在收到复位信号后才进行初始化。具有复位功能的设备在上电后必须立即进行复位操作。

⑤ 电源开关顺序

在一个设备的内部操作和外部接口使用不同的电源的情况下，按照规定，应先在接通内部电源之后再接通外部电源。当关闭电源时，按照规定，先关闭外部电源再关闭内部电源。如果电源开关顺序颠倒，可能会导致设备的内部组件过电压，产生异常电流，从而引起内部组件的误操作和性能的退化。
对于每个设备电源的正确开关顺序必须依据设备的规范说明分别进行判断。

⑥ 电源关闭状态下的输入信号

不要向没有加电的设备输入信号或提供I/O上拉电源。因为输入信号或提供I/O上拉电源将引起电流注入，从而引起设备的误操作，并产生异常电流，从而使内部组件退化。
每个设备电源关闭时的信号输入必须依据设备的规范说明分别进行判断。

- **本文档所登载的内容有效期截止至 2009 年 5 月，信息先于产品的生产周期发布。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。**
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。否则因本文档所登载内容引发的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

  “特殊等级”仅适用于为特定用途而根据用户指定的质量保证程序所开发的日电电子产品。另外，各种日电电子产品的推荐用途取决于其质量等级，详见如下。用户在选用本公司的产品时，请事先确认产品的质量等级。

  “标准等级”： 计算机，办公自动化设备，通信设备，测试和测量设备，音频·视频设备，家电，加工机械以及产业用机器人。

  “专业等级”： 运输设备（汽车、火车、船舶等），交通用信号控制设备，防灾装置，防止犯罪装置，各种安全装置以及医疗设备（不包括专门为维持生命而设计的设备）。

  “特殊等级： 航空器械，宇航设备，海底中继设备，原子能控制系统，为了维持生命的医疗设备、用于维持生命的装置或系统等。

  除在本公司半导体产品的数据表或数据手册等资料中另有特别规定以外，本公司半导体产品的质量等级均为“标准等级”。如果用户希望在本公司设计意图以外使用本公司半导体产品，务必事先与本公司销售代表联系以确认本公司是否同意为该项应用提供支持。

（注）

（1） 本声明中的“本公司”是指日本电气电子株式会社（NEC Electronics Corporation）及其控股公司。

（2） 本声明中的“本公司产品”是指所有由日本电气电子株式会社开发或制造的产品或为日本电气电子株式会社（定义如上）开发或制造的产品。

M5 02.11-1

前言

目标读者

本应用笔记旨在使用户了解 μPD78F0714 的功能及设计开发应用系统。 下面为所应用的产品

- μPD78F0714

目的

本应用笔记旨在帮助用户理解使用μPD78F0714，并通过 180°换相方式来驱动具有霍尔传感器的无刷电机(BLDCM)的系统。

组织

本应用笔记包含以下内容。

- 概述
- PMSM 控制原则
- 系统介绍
- 控制程序

如何阅读本手册

在阅读本应用笔记前，读者应掌握电子工程、逻辑电路和微控制器等电子工程方面的基础知识。

有关硬件功能（尤其是寄存器功能和设置功能等）和电气特性的详细内容

→ 请参考 **μPD78F0714** 用户手册**(U16928E)**

有关指令功能的详细内容

→ 请参考 78K/0 系列指令用户手册 **(U12326E)**

规则

数据规则:	数据的高位部分在左边，低位部分在右边
有效低电平表示法:	$\overline{xxx}$ (在引脚和信号名称上加划一条线)
存储空间地址:	高位地址在上部，低位地址在下方
注:	文中用**注**标注的相关术语的脚注
注意事项:	需要特别关注的信息
备注:	补充信息
数值的表示:	二进制 ... xxxx 或 xxxxB
	十进制 ... xxxx
	十六进制 ... xxxxH
前缀表示 2 的乘幂 (地址空间, 存储器容量)	
	K (K): $2^{10} = 1,024$
	M (兆): $2^{20} = 1,024^2$
	G (G): $2^{30} = 1,024^3$
数据类型:	字长: 32 位
	半字长: 16 位
	字节: 8 位

相关文档

本手册中提到的相关文档可能包括有初稿版本。 但是，初稿版本没有特别注明。

设备相关文档

文档名称	文档编号
μPD78F0714 用户手册	U16928E
78K/0 系列指令用户手册	U12326E
过零检测进行μPD78F0714 120° 换相方式变频控制应用笔记	U17297E
由 V/f 控制，μPD78F0714 应用于单相感应电机控制两相正弦波变频驱动应用笔记	U17481E
μPD78F0714 120° 换相方式进行变频控制应用笔记	U18774E
μPD78F0714 无感 120° 换相方式进行变频控制应用笔记	U18051E
μPD78F0714 180° 换相方式进行变频控制应用笔记	本手册
μPD78F0714 无感 (BEMF 的 A/D 转换) 120° 换相方式进行变频控制应用笔记	U18912E

开发工具相关文档 (软件) (用户手册)

文档名称		文档编号
RA78K0 Ver. 3.80 汇编包	操作篇	U17199E
	语言篇	U17198E
	结构化汇编语言篇	U17197E
CC78K0 Ver. 3.70 C 编译器	操作篇	U17201E
	语言篇	U17200E
ID78K0-QB Ver. 2.94 集成调试器	操作篇	U18330E
PM+ Ver. 5.20		U16934E

开发工具相关文档 (硬件) (用户手册)

文档名称	文档编号
QB-780714 片上仿真器	U17081E
QB-78K0MINI 片上调试仿真器	U17029E
QB-MINI2 具有编程功能的片上调试仿真器	U18371E

Flash存储器编程相关文档

文档名称	文档编号
PG-FP4 存储器编程器用户手册	U15260E

注意事项: 以上列出的相关文档可能会在无任何声明条件下修改。 开发设计时，请使用各文档的最新版本。

其它相关文档

文档名称	文档编号
半导体选择指南 - 产品和封装	X13769X
半导体设备装配手册	**注**
NEC 半导体设备质量等级	C11531E
NEC 半导体设备可靠性/质量控制系统	C10983E
半导体设备防静电 ESD 指南	C11892E

注: 可参阅“半导体设备装配手册”网站 (http：//www.necel.com/pkg/en/mount/index.html)。

注意事项: 以上列出的相关文档可能会在无任何声明条件下修改。 开发设计时，请使用各文档的最新版本。

目录

第1章　概要信息

该系统使用 180° 控制方式来驱动安装有霍尔传感器的永磁同步电动机（下文叫做“PMSM”）。

- 该系统（示例程序）使用日电电子电动机入门套件（μPD78F0714）[注] 以及使用 180° 控制方式来驱动安装有霍尔传感器的 PMSM 。
- 根据以下指定的工作环境中的电动机而调整控制增益。在更改电动机或控制周期时，无法保证操作正确。

注　关于电动机入门套件（μPD78F0714），请联系日电电子销售代表。

1.1 工作环境

该系统的工作环境如下假设。

- 电动机入门套件（μPD78F0714）
- 低压变频器
  BLDCM PITTMAN （N2311A011）[注]
  - 参考电压[V]：　12
  - 空载转速[r/min]：　7197
  - 连续转矩[Nm]：　0.11
  - 最大转矩[Nm]：　0.23
  - 激励线圈：　3 相 （Y 连接）
  - 转子磁极：　4 磁极 （2 对磁极）
  - 定子：　6 节流
  - 位置传感器：　霍尔传感器
- PM plus 环境平台 V5.20
- CC78K0 编译器 W3.70
- RA78K0 汇编器 W3.80
- DF0714.78K 设备文件 V1.10

注　在该工作环境下，使用 BLDCM 来代替 PMSM。

1.2 相关手册

关于开发环境和电路板，参见以下手册。

- 低压电动机入门套件手册
- PM plus V5.20 用户手册
- CC78K0 V3.70 C 编译器的所有用户手册
- RA78K0 V3.80 汇编器包的所有用户手册

第2章　PMSM 控制原理

通过永磁转子和定子周围的线圈产生的旋转磁场之间的吸引力，PMSM 按照旋转磁场的速度旋转。

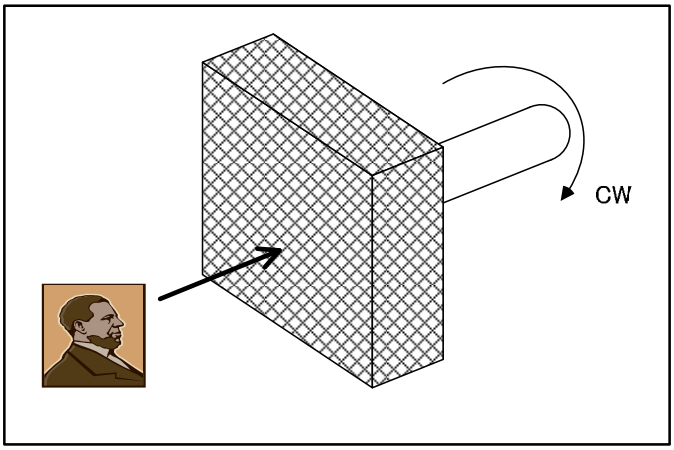
2.1 旋转方向定义

首先，定义电动机的旋转方向。

电动机的旋转方向为 CW（顺时针）或 CCW（逆时针）。

基于电动机驱动对象的旋转方向来确定 CW 或 CCW。如下所示，是在电动机轴所在的表面面向对象时，来定义旋转方向的。

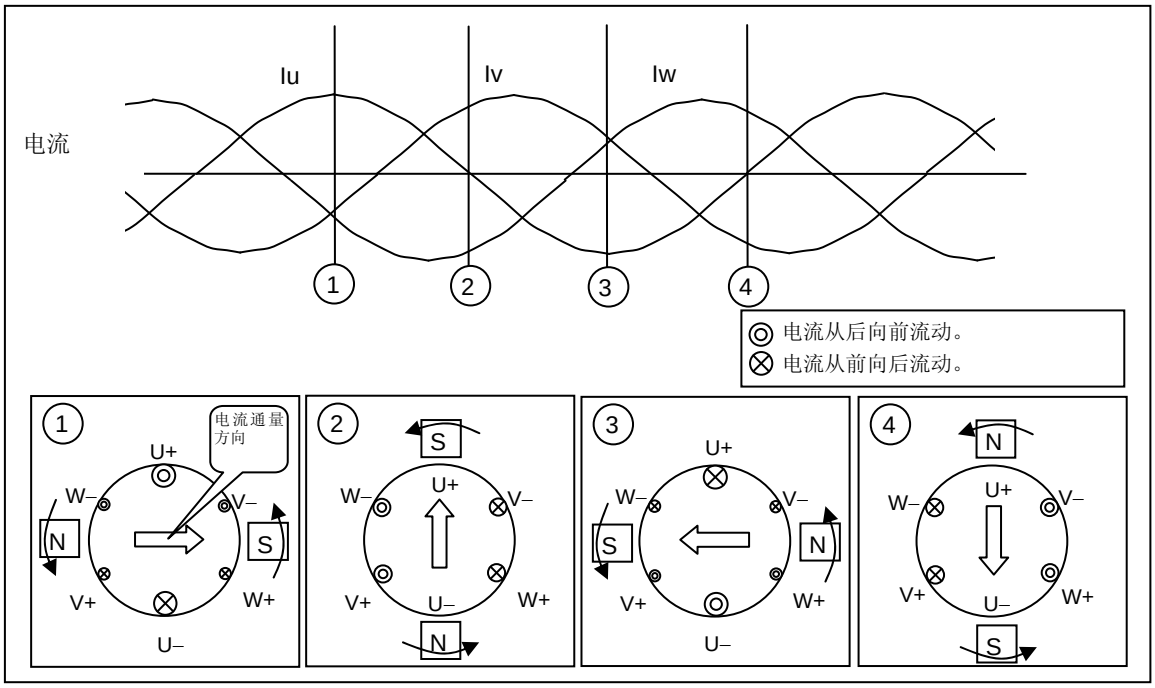
图 2-1. 电动机旋转方向



2.2 旋转磁场

对于 PMSM，旋转磁场由流过线圈的交变电流产生。下图是 3 相绕组产生旋转磁场的原理。3 相绕组（u 相、v 相和 w 相）间隔 120 度放置，并且 3 相绕组电流（Iu、Iv 和 Iw）之间有 120 度的相位差。图中的双圈（“◎”）与电流大小成比例。

图 2-2. 3 相交变电流和产生旋转磁场的原理

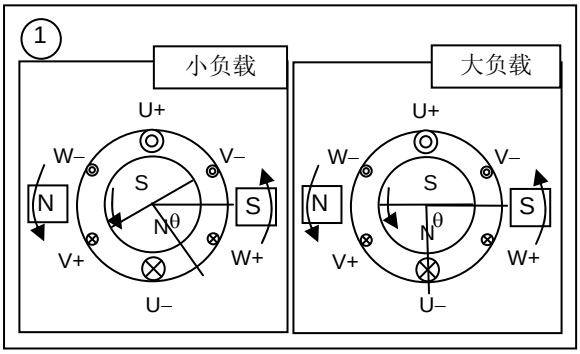


如果电流通量方向为“*I*”，永磁通量为“ϕ”，而电流通量和永磁通量的斜率为“θ”，电动机转矩可以通过以下表达式表示。

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

通过表达式（1）可知，当没有负载时θ为 0°，而当负载超过最大转矩（当θ = 90°时，产生最大转矩）时同步丢失。

图 2-3. 旋转图



如果电流频率为“f”，极对数为“P”，速度“m”（rpm）可以通过以下表达式表示。

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

对于不检测转子位置的控制（开环激励），转矩和电流（电压）以及速度和电流频率必须单独控制。如果负载改变，表达式（1）中的θ 也改变，导致速度变化，同步丢失以及在低负载下高电压下产生的大电流，从而使稳定控制变得困难。

通过检测转子位置并总是保持θ 处于 90 度，可以实现高效操作，因为产生的转矩总是最大。同时，通固定电流和通量相位，速度和转矩的特性将与 DC 电动机的特性一致，并且如果电压为“V”，线圈电阻为“R”，感应电压为“E”，电流为“I”，转子转速为“N”，“E”将与“N”成比例而“T”与“I”成比例。因此，如果比例系数为 Ke 和 Kt，可以得出以下表达式。

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

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

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

表达式（6）可以从以上表达式得出。

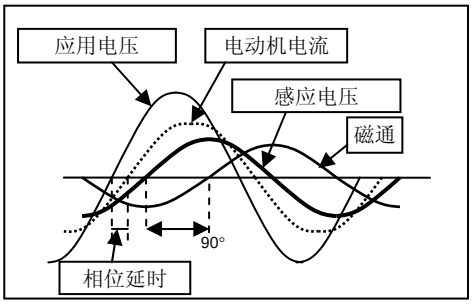
$$T = \left(Kt/R\right)\left(V - KeN\right) \quad (6)$$

如果在表达式（6）中 V 为常数，转速（N）的减少将导致负载（T）的增加（同步未丢失）。因此，通过检测速度并使用电压 V 来调整速度，可以实现适合于负载的高效电流控制。

2.3 180° 激励方法

通过对绕组应用正弦波电压来激励电动机的方法叫做 180° 激励方法。
下图表示相位 U 的瞬时电流波形。。

图 2-4. 瞬时电压和电流波形



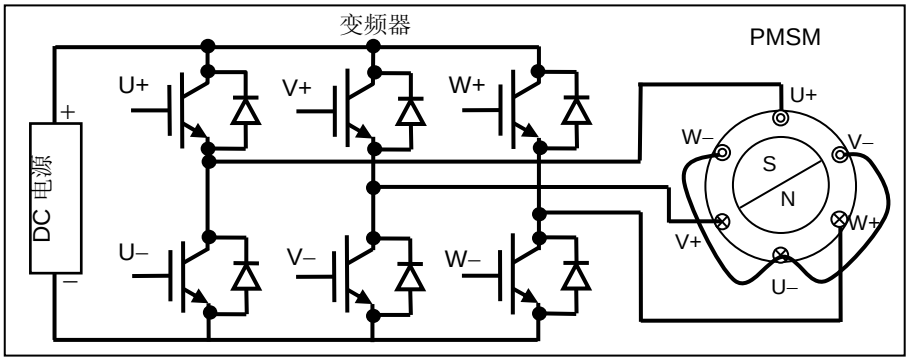
应用到电动机的电压和感应电压的差被应用到电动机线圈，产生电动机电流。由于存在线圈电感，电动机电流的相位与应用电压有相位差。因为当磁通和电流的相位差为 90 度时电动机转矩最大，可以通过应用电压的相位来调整该延时。

2.4 变频器

PMSM 通过可变电压、可变频率控制（VVVF 变频控制或可变频率激励）从 DC 电源产生 3 相交变电流，并根据负载改变控制电压和频率。

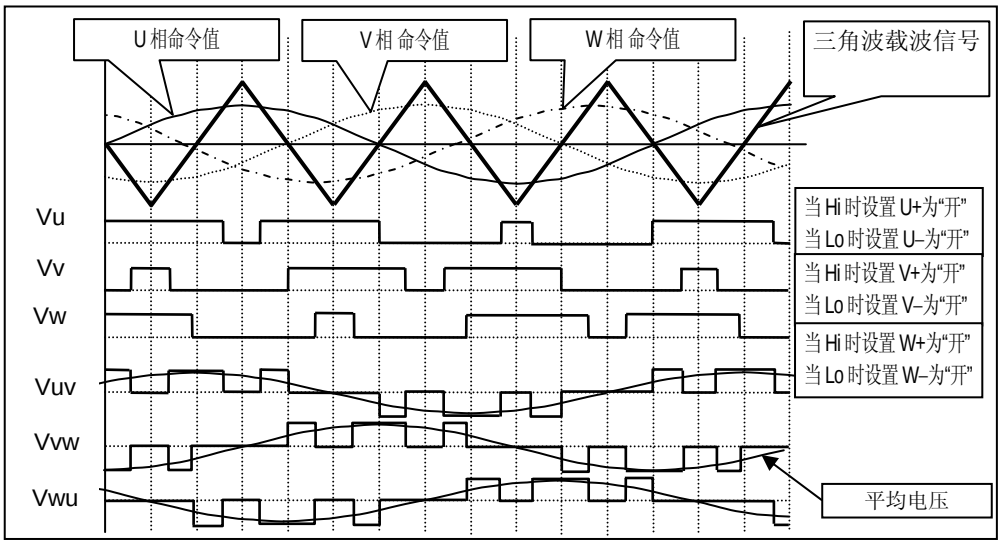
下图表示 PMSM 和 3 相变频器之间的连接。

图 2-5. 3 相变频器和 PMSM



下图表示一个示例，其中当上面提到的电路通过三角波比较 PWM 方法（脉冲调整方法）产生时执行开关元件的斩波操作时，线圈 中流过 3 相交变电流。

图 2-6. 3 相变频器输出波形（三角波比较 PWM 方法）



按照命令值（正弦波信号）切换每相的开关为“开”或“关”，可以得到相电压（Vuv，Vvw，Vwu），命令值与操作变频器频率和三角波载波信号的大小一致。（电流的方向和大小根据相电压的差改变。）

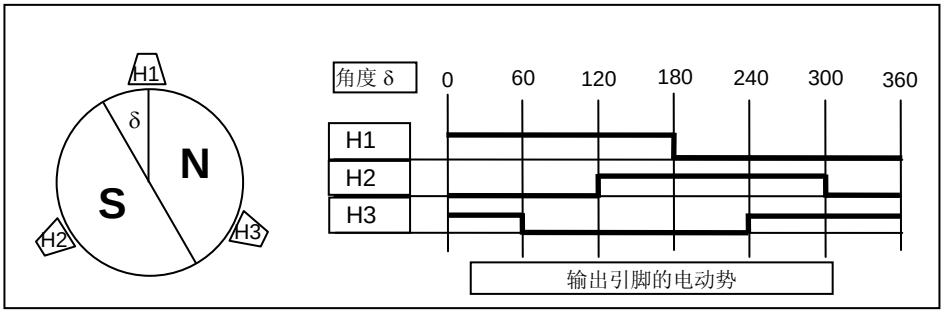
2.5 位置检测

对于 180°激励方法，需要详细的位置信息以保持电流通量和永磁通量之间的夹角为 90 度。

在该系统中，霍尔传感器间隔 120 度放置，通过霍尔传感器的变化（每圈六个脉冲），每 60 度检测霍尔传感器位置，并且可以从转速推断出小于 60 度的位置。

下图表示使用霍尔传感器的磁极位置传感器的输出示例。

图 2-7. 霍尔传感器输出



对于三相来说，霍尔传感器的位置可以每 60 度检测，因为每相的霍尔传感器输出每 180 度改变一次。

当转子具有四个磁极时，霍尔传感器的值每 90 度改变并且每 30 度切换激励模式。

通常，一个周期（六个激励模式）定义为 360 度电角度，电动机轴的一次旋转定义为 360 度机械角（本文档中的所有角都是电角，除非特别说明）。

机械角： 电角 / 极对数

极对数： 极数/2

2.6 启动方法

使用霍尔传感器的位置信息，可以从霍尔传感器的输出和转速推断出转子的位置。当转子停止时，转子位置的误差可能达到 60 度，所以电动机必须通过使用不需要详细位置信息的方法来启动。

典型的启动方法如下所示。

2.6.1 同步启动方法

当通过 180 度正弦波电流执行同步启动后，切换为 180 度正弦波驱动，在通过强制激励线圈以定位转子位置后，可以从霍尔传感器输出的改变和转速推断出来转子位置。

当位置信息的分辨率较低时，很难调整应用电压和电流频率，并且对于 180 度激励方式，从停止状态启动时的旋转不稳定。

2.6.2 120° 激励方法

使用 120°激励方法来输出电流，在可以通过霍尔传感器的输出变化和转速推断出转子位置后，切换为 180 度正弦波驱动。

主要使用 120°激励方法，因为只在启动发动机时发生转矩脉动，所以问题并不严重。

2.7 速度检测

电动机的转速通过计算霍尔传感器值改变的时间来得出。

2.8 电压控制

由 PWM（脉冲宽度调制）控制应用到电动机线圈的电压，PWM 通过以很高的频率做斩波操作的开关元件的传导周期来调整传导速率（平均电压）。

2.9 速度控制

基于表达式（6），通过控制应用到线圈的电压来控制速度。具体来说，从霍尔传感器输出和转速估计转子位置，并且计算并设置应用到三角波比较 PWM 方法每相的命令值（与三角波载波信号的比较值）。

命令值由 PID 控制调整。

2.9.1 PID 控制

PID 控制通过计算改变命令值，计算基于指定的速度和检测的转速之间的偏差。

PID 控制动作由产生与偏差成比例输出的比例（P）动作、产生与偏差的积分成比例输出的积分（I）动作以及产生与偏差的微分成比例输出的微分（D）动作组成。

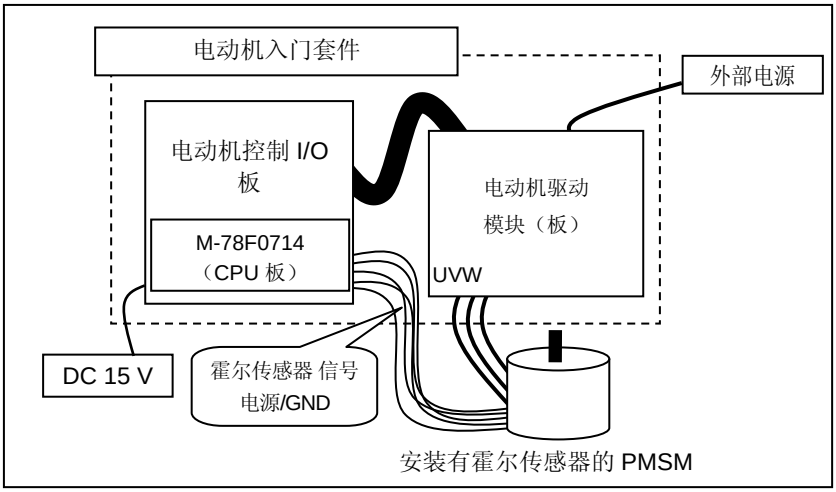
第3章　系统概述

本章介绍系统的概述。

3.1 配置

下图表示该系统的配置。

图 3-1. 系统配置

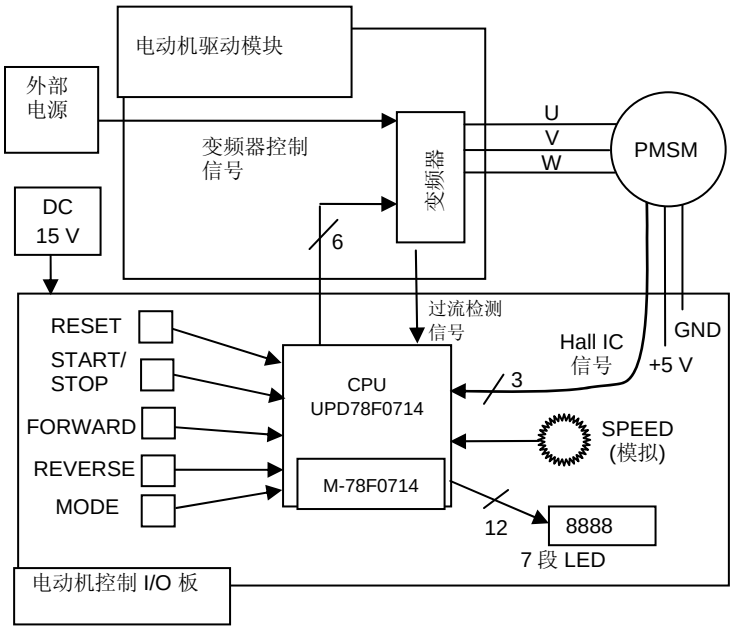


该系统由驱动电动机的电动机驱动模块、安装有控制电动机的开关的电动机控制 I/O 板和带有 CPU 的 M-78F0714 组成。

三相四极（两对磁极）PMSM，并且安装有霍尔传感器。

电动机入门套件的框图结构如下所示。

图 3-2. 框图结构



电动机由电动机控制 I/O 板上的开关控制。

3.2 接口

表 3-1 列出了用户接口函数。

表 3-1. 用户接口

函数名	部件编号	功能
RESET	SW1	复位
START/STOP	SW2	启动 / 停止
FORWARD	SW3	旋转方向（向右旋转，顺时针，CW）
REVERSE	SW4	旋转方向（向左旋转，逆时针，CCW）
MODE	SW5	切换速度指定 切换速度显示
SPEED	R52	更改指定的速度
8 段 LED	DISP1 DISP2 DISP3 DISP4	显示速度（rpm）[注]

注　电动机停止时，显示指定的速度。
指定的速度固定时，在右下角显示一个点（“.”）。
电动机旋转时，显示转速。
电动机旋转时，按下 MODE 开关时，显示指定的速度。
当按下 RESET 按钮时，不断复位；当释放 RESET 按钮时，复位释放。
复位释放后的 2 秒钟内显示“SELF”。

表 3-2 列出了错误。

表 3-2 错误

错误	LED 显示	情况
硬件过流	0.C.	电动机电流异常。
软件过流	S.0.C	电动机电流异常。
霍尔传感器	HALL	霍尔传感器值异常。
系统故障	FAIL	电动机未旋转。

表 3-3 列出了μPD78F0714 引脚的接口。

表 3-3. 引脚接口

引脚编号	引脚名称	功能
8	$\overline{\text{RESET}}$	RESET （SW1）[注]
27 到 32	TW0TO0/RTP10 到 TW0TO5/RTP15	3 相 PWM 变频器选择
11	P01/INTP1	霍尔传感器 信号 （HALL1）
10	P02/INTP2	霍尔传感器 信号 （HALL2）
9	P03/INTP3/ADTRG	霍尔传感器 信号 （HALL3）
49 到 52	P64 到 P67	7 段 LED 选择 （LD_LED0 到 LD_LED3）
56	P73	START/STOP （SW2）
55	P72	FORWARD （SW3）
54	P71	REVERSE （SW4）
53	P70	MODE （SW5）
41 到 48	P40/RTP00 到 P47/RTP07	输出数据到 8 段 LED
12	TW0TOFFP/INTP0/P00	过流检测 （+5 V → 0 V）
60	P24/ANI4	速度更改（R52）
59	P25/ANI5	ISHUNT 电流
20	P53/TI000/INTP5	定时器捕获触发
21	P54/TI001/TO00	电动机激励模块控制

注　　将 2JP7 的 1-2 短接。

3.3 功能

表 3-4 列出了该系统的功能和操作概要。

表 3-4. 系统功能和操作概要（1 / 2）

功能	概要
启动 （电源）	• “SELF”显示在 LED 上 2 秒钟。 • SPEED 控制器指定的速度（rpm）显示在 LED 上。
RESET 开关	• 无论电动机控制的状态，系统都复位。
START/STOP 开关	电动机控制停止期间：电动机控制被启动。
	• “0”显示在 LED 上。 • 电动机开始顺时针旋转。 • 电动机旋转速度（rpm）显示在 LED 上。
	电动机被控制期间：电动机控制被停止。
	• 电动机旋转速度（rpm）显示在 LED 上直到达到 0。 • SPEED 控制器指定的速度（rpm）显示在 LED 上。
	即使该开关被按下，START 和 STOP 也不会锁定。
FORWARD 开关	电动机控制停止期间：无功能。
	• 不变。
	电动机被控制期间：改变旋转方向。
	• 如果旋转方向为 CCW，停止后马上改为 CW。 • 如果旋转方向为 CW，不变。
REVERSE 开关	电动机控制停止期间：无功能。
	• 不变。
	电动机被控制期间：改变旋转方向。
	• 如果旋转方向为 CCW，不变。 • 如果旋转方向为 CW，停止后马上改为 CCW。
MODE 开关	电动机控制停止期间：切换为速度指定。
	• 禁止或使能通过 SPEED 控制器设定速度。 • 当禁止时，在 LED 值的右下角显示一个点（“.”）。
	电动机控制期间： 改变 LED 的显示。
	• 按下开关时，SPEED 控制器指定的速度（rpm）显示在 LED 上。
SPEED 控制器	电动机控制停止期间：更改指定的速度。
	• 在 LED 上显示速度 （rpm）。 • 如果通过 MODE 开关禁止，不变。
	电动机被控制期间：更改指定的速度。
	• 电动机以指定的速度（平均）旋转。
硬件过流	• 如果出现超过电动机驱动模块允许的电流的情况。 • 停止电动机控制。 • 在 LED 上显示“0.C.”。 • 禁止 RESET 开关以外的功能。
软件过流	如果超过电动机允许的电流的情况时。 停止电动机控制。 在 LED 上显示“S.0.C.”。 禁止 RESET 开关以外的功能。

注意事项　如果同时按下两个或更多开关，无法保证操作。

表 3-4. 系统功能和操作概要（2 / 2）

功能	概要
不旋转	• 如果电动机从 START 开始 1.25 秒没有旋转以及如果电动机在旋转期间停止 0.5 秒或更长。 • 停止电动机控制。 • 在 LED 上显示“FAIL”。 • 禁止 RESET 开关以外的功能。
霍尔传感器 异常	• 霍尔传感器的值异常时发生。 • 停止电动机控制。 •在 LED 上显示“HALL”。 • 禁止 RESET 开关以外的功能。

注意事项　如果同时按下两个或更多开关，无法保证操作。

3.4 外围 I/O

该系统使用以下外围 I/O。

表 3-5. 外围 I/O

功能	外围 I/O 功能名称（μPD78F0714）
变频定时器	• 用于 PWM 输出 • 10 位变频控制定时器（TW0UDC 等等） • 载波（调制）频率为 10 kHz（对称三角波）。 • 载波（调制）同步中断以 200 μs 的间隔产生（载波的中断频率每两次被设置为一次）。 （通过由主 PID 控制计算三相应用电压和由载波同步中断计算命令值来更新）
实时输出	• 用于更改激励模式 • 实时输出端口（RTBH01、 RTBL01 等等） • 16 位定时器捕获 / 比较寄存器 01 （CR01） （用于启动时的 120°励磁方法的激励模式输出。）
获取 ISHUNT 电流值的定时器	• 用于时序调整 • 8 位定时器 / 事件计数器 51 （TM51，CR51） • 中断以 1.43 ms 的间隔产生并且执行 ISHUNT 电流测量功能。
等待处理	• 用于时序调整 • 8 位定时器 / 事件计数器 50 （TM50，CR50） • 中断请求标志以 1ms 的间隔设置。
读取指定的速度	• 将变阻器的电压转换为速度指定。 • A/D 转换器 （ANI4）
获取 ISHUNT 的电流值	• 转换电动机工作电流为电压，通过引脚（ANI5）获取电压值。
捕获中断	• 用于速度计算 • 中断功能（TI000/INTP5）
硬件过流中断	• 中断功能（INTP0） • 电动机驱动模块中发生过流（低有效）
故障安全	• 看门狗定时器

3.5 中断

表 3-6 表示该系统中使用的中断。

表 3-6. 使用的中断

名称	功能	发生环境
INTP0	过流检测	外部引脚
INTP5	HALL1 更改的检测（速度计算）	外部引脚
INTTW0UD	发生载波同步中断	变频定时器计数器的下溢
INTTM51	获取 ISHUNT 电流值	8 位定时器计数器的溢出
RESET	发生复位	$\overline{\text{RESET}}$ 引脚
WDT	发生内部复位	由于程序失控，看门狗定时器溢出

备注 通过轮询 INTTM50 和 INTAD 的中断请求标志来处理。

第4章　控制程序

该系统使用一个基本控制程序来实际控制电动机的速度。

4.1 编译器选项

可以通过使用编译时的选项来切换该程序可控制的两种变频器[注]。

表 4-1. 编译器选项

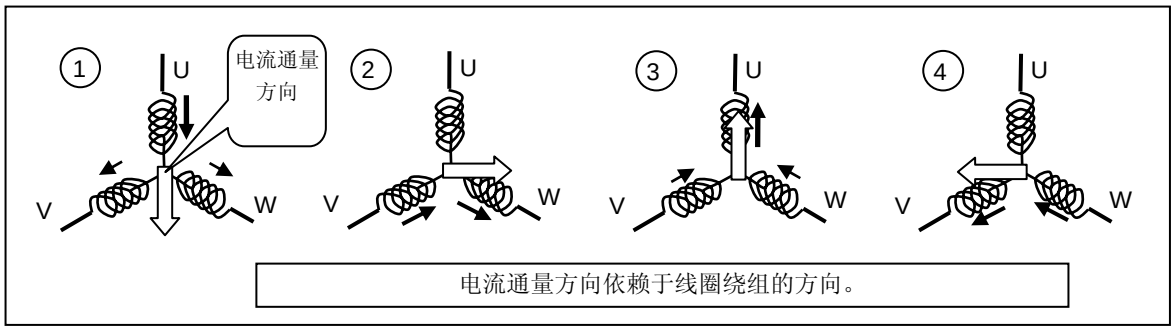
选项名称	功能
LOW	使用 PITTMAN 电动机时
None [注]	使用其它电动机时

注　　必须设置 LOW，因为在该控制程序中，使用 PITTMAN 电动机。

4.2 旋转磁场

图 2-2 三相交变电流和产生旋转磁场的原理按照下面改变，因为该系统使用 BLDCM （无刷 DC 电动机）而不是 PMSM。

图 4-1. 集中绕组的旋转磁场

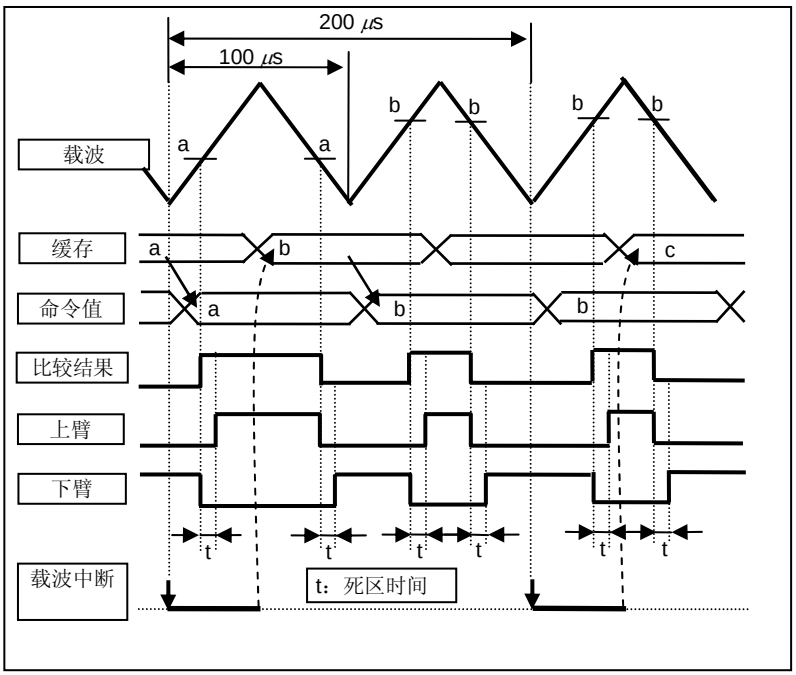


4.3 变频器

变频定时器功能用来开关变频器。

每次计数器下溢发生时，产生载波同步中断，并且设置下一个变频定时器输出模式。

图 4-2. 载波同步中断发生的时序

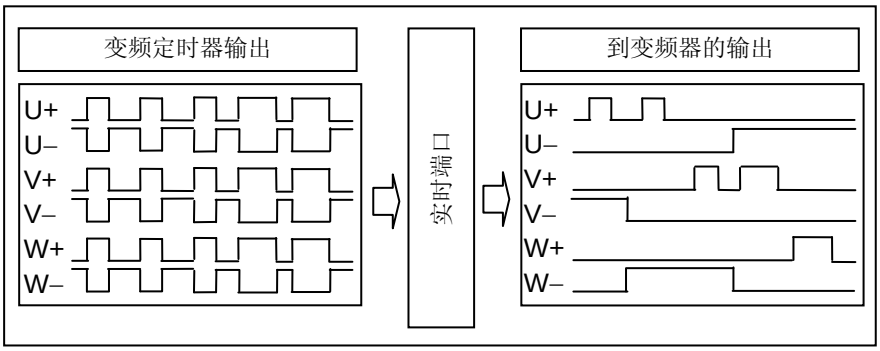


对于 180°励磁方法，停止使用实时端口功能并且变频定时器输出为到变频器的输出。使用从霍尔传感器值和载波同步中断次数计算得到的位置信息进行 PID 控制操作，根据 PID 控制操作的结果，在载波同步中断（每隔 200 μs）中更改变频定时器输出的波形。

因为在使用的电动机中产生高频声音，所以对 10KHz 的载波做分频，每两次产生一次中断，即将载波设置为 5 kHz 以及载波同步中断被设置为每次（200 μs）发生的情况。

对于启动时的 120°励磁方法，整个上臂区域执行非互补操作的 PWM 控制。

图 4-3. 120°励磁方法的 PWM 输出图



变频定时器输出的波形将根据 PID 控制操作的结果而改变。

在每个载波同步中断（200 μs）中，实时端口的屏蔽处理（激励模式）由霍尔传感器的值确定。

4.4　180° 励磁方法

三角波比较 PWM 方法使用的命令值如下得出。

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

Vu，*Vv*，*Vw*：三个相位的命令值
E_d：DC 电源电压，δ：旋转角，α：超前角，*M*：电压命令值 $0 < M \leq 1$

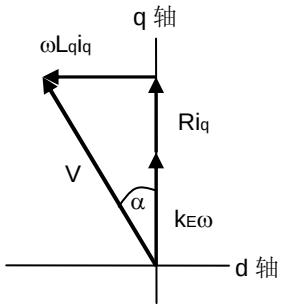
电压超前角可以通过使用电动机的电压和电流模型表达式来得出。
永磁同步电动机静止时的电压方程和转矩表达式如下所示。

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

$T = P_n k_E i_q + P_n(L_d - L_q) i_d i_q$
id，*iq*：电枢电流的 *d* 和 *q* 轴分量，v_d，v_q：电枢电压的 *d* 和 *q* 轴分量，
k_E：感应电压系数，L_d，L_q：*d* 和 *q* 轴电感，ω：速度，P_n：极对数

下图表示 d 轴电流控制为 0 时的电压向量轨迹。

图 4-4. 向量图



可以从转矩表达式理解负载的增加导致了电流的增加，从向量图理解速度和电流的增加导致了超前角α的增加。
通过查询超前角表格来处理相应情况的值，因为该系统（CPU）没有能力执行这些计算。（超前角可以在程序中更改。）

4.5 位置检测

角度每 60 度根据霍尔传感器的数值来检测，并且根据经过的时间来推测更加详细的角度（以 200 μs 间隔发生的载波同步中断的个数），因为电动机转速和霍尔传感器的值在改变。

根据 120°励磁方法的激励模式和霍尔传感器的值，霍尔传感器的值和转子的磁极位置之间的关系如下所示。

图 4-5. 120° 励磁方法的激励模式

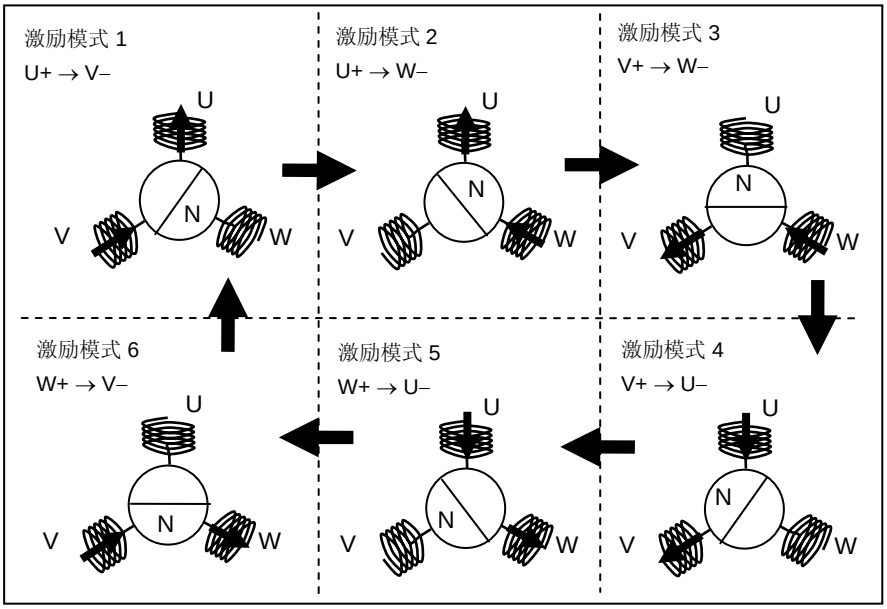


表 4-2. 激励模式和霍尔传感器之间的关系（根据电动机规范表）

激励模式 / 霍尔传感器	1	2	3	4	5	6
S1 （HALL1）	L	L	H	H	H	L
S2 （HALL2）	H	L	L	L	H	H
S3 （HALL3）	H	H	H	L	L	L

霍尔传感器和角度改变之间的关系如下所示。

引脚信息、激励模式和霍尔传感器值根据 BLDCM 规范来描述。

4.5.1　低压变频器设置

表 4-3. BLDCM 引脚规范

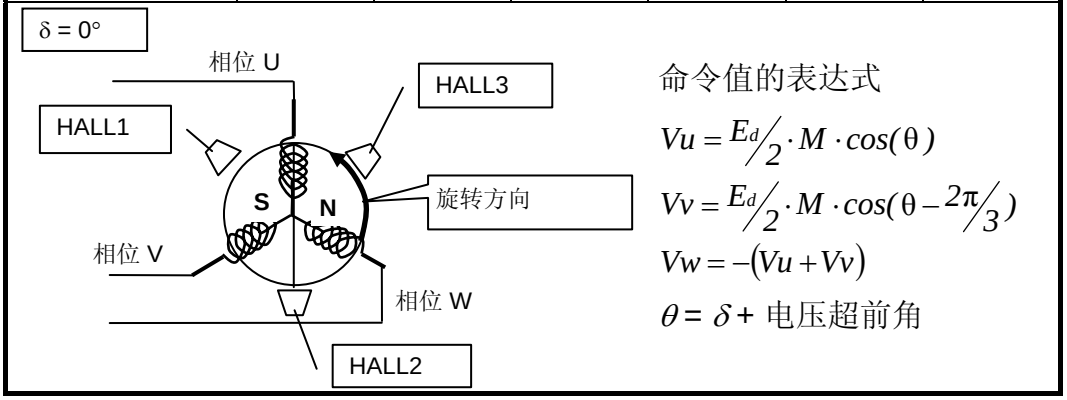
颜色	功能	备注
BROWN	MOTOR ϕA	相位 U
RED	MOTOR ϕB	相位 V
ORANGE	MOTOR ϕC	相位 W
GREY	SENSOR ϕ1	HALL1
BLUE	SENSOR ϕ2	HALL2
WHITE	SENSOR ϕ3	HALL3
VIOLET	SENSOR Vcc	
BLACK	SENSOR GND	

表 4-4. 120°励磁模式中的激励模式和霍尔传感器 值

激励 （正向旋转）	+A –B	+A –C	+B –C	–B –A	+C –A	+C –B
SENSOR ϕ1	L	L	H	H	H	L
SENSOR ϕ2	H	L	L	L	H	H
SENSOR ϕ3	H	H	H	L	L	L
激励 （反向旋转）	+B –A	+C –A	+C –B	+A –B	+A –C	+B –C

表 4-5. 180° 励磁模式中的角度

角度(δ) / 霍尔传感器	0	60	120	180	240	300
HALL1	L	L → H	H	H	H → L	L
HALL2	H → L	L	L	L → H	H	H
HALL3	H	H	H → L	L	L	L → H



4.6 启动方法

该系统使用 120°励磁方法启动电动机。关于 120°励磁方法的详细情况，参见**电动机控制系统规范设计报告霍尔传感器 120°励磁方法。**

4.7 速度检测

通过使用定时器捕获功能在霍尔传感器值改变时保存定时器计数值，速度计算不受在正常处理中保存计数值时发生延时的影响，可实现高精度的速度检测。

在该系统的 BLDCM 情况下（三相，四极），每转霍尔传感器变化四次；然而，由于要使用的定时器功能的要求，只处理上升沿的变化，所以根据在半圈期间改变的定时器值来计算速度。

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

N：每分钟旋转次数（*rpm*）

s：定时器的分辨率（12.8 μs）

n：定时器的值

2：极对数

对于 PITTMAN 电动机，该系统支持 200 到 7,200 rpm 的分辨率速度范围。

4.8 电压控制

三角波比较 PWM 方法使用的电压命令值用来控制三相的相电压。

4.9 速度控制

通过更改三角波比较 PWM 方法的电压命令值，控制速度。电压命令值由 PID 控制调整。

4.9.1 PID 控制

通过反馈实际转速和指定速度之间的偏差并在 DC 电源电压命令值上执行 PID 控制操作，来调整速度。电压命令值的操作变量使用以下适用于采样方法（离散值）的速度类型 PID 算法。

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

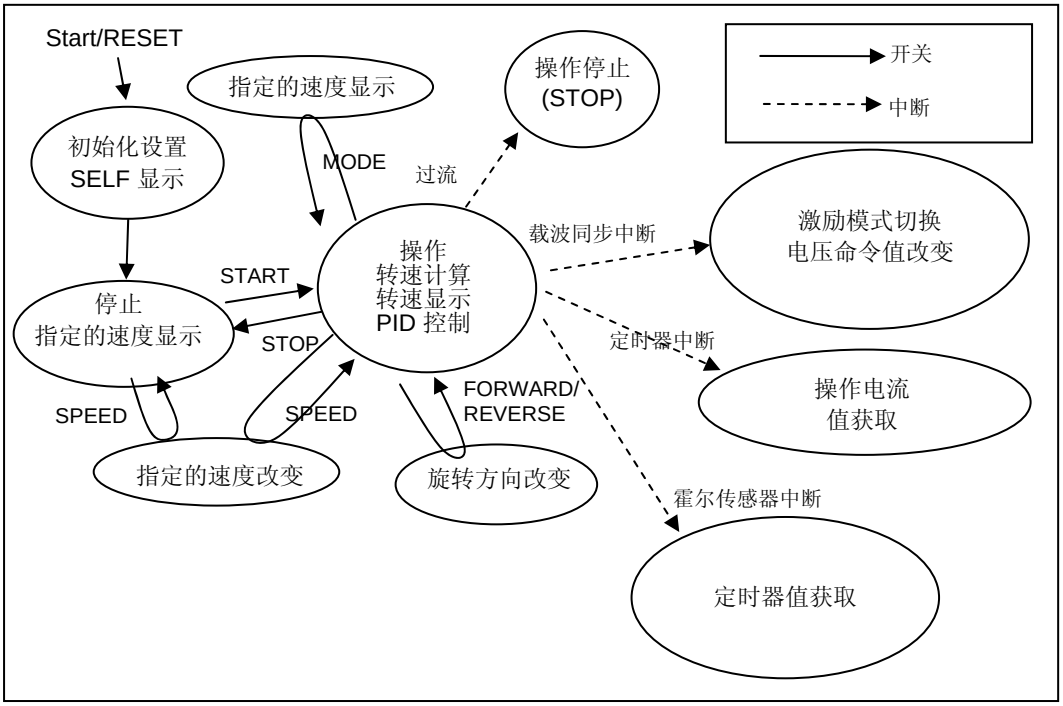
MV_n：　当前操作的变量
MV_{n-1}：　之前操作的变量
ΔMV_n：　当前和之前操作的变量之间的偏差
e_n：　当前偏差（指定速度和实际速度之间的偏差）
e_{n-1}：　之前的偏差
e_{n-2}：　之前偏差之前的偏差
Kp：　反馈增益（比例环节）
Ki：　反馈增益（积分环节）
Kd：　反馈增益（微分环节）

反馈增益的最优值根据电动机特性和负载的有无而改变。

4.10 模块配置

系统的状态转换图如下所示。

图 4-6. 系统的状态转换



载波同步中断每 200 μs 发生。
霍尔传感器中断每次旋转发生两次。

4.11 控制程序函数列表

控制程序由多个函数组成。下表列出了这些函数和它们的特征。关于处理的详细情况，参见流程图。

4.11.1 用户可以使用的函数

表 4-6. 用户可以使用的函数

函数名	功能	用途
motor_init()	初始设置	执行电动机控制需要的初始化，例如每个函数的寄存器设置和中断设置。
motor_start()	启动指令	指示电动机开始旋转。
motor_stop()	停止指令	指示电动机停止旋转。
motor_rotation()	旋转方向指令	指示电动机旋转的方向。
motor_pid()	PID 操作	指示 PID 操作.
motor_pset()	参数设置	设置电动机控制需要的参数。

4.11.2 电动机库内部函数

表 4-7. 电动机库内部函数

函数名	功能	用途
system_restart()	重启处理	停止反向旋转后重启。
system_stop()	停止处理	停止系统。
init_PORT()	端口设置处理	设置电动机控制使用的端口。
init_OSC()	时钟设置处理	为 CPU 工作速度执行时钟设置。
init_TW0()	变频器设置处理	设置变频定时器。
start_TW0()	变频器启动处理	启动变频定时器。
stop_TW0()	变频器停止处理	停止变频定时器。
set_TW0()	PWM 设置处理	设置 PWM。
init_TM00()	TM00 设置处理	设置 16 位定时器。
start_TM00()	TM00 启动处理	启动 16 位定时器。
stop_TM00()	TM00 停止处理	停止 16 位定时器。
init_RTPM01()	实时输出端口设置处理	设置实时输出端口。
start_RTPM01()	实时输出端口启动处理	启动实时输出端口。
stop_RTPM01()	实时输出端口输出停止处理	停止实时输出端口。
end_RTPM01()	实时输出端口使用停止处理	从实时输出端口切换为 PWM 输出。
set_RTPM01()	激励模式切换处理	设置实时输出端口输出值并执行输出切换处理。
init_AD()	A/D 设置处理	设置 A/D 转换。
start_AD()	A/D 启动处理	启动 A/D 转换。
init_WDTM()	看门狗定时器设置处理	设置看门狗定时器。
init_TM51()	TM51 设置处理	设置 8 位定时器。
start_TM51()	TM51 启动处理	启动 8 位定时器。
read_霍尔_IC()	霍尔传感器 信息获取	从端口获取 霍尔传感器信息。
INTP0_on()	INTP0 允许处理	允许过流中断。
INTTW0UD_on()	INTTW0UD 允许处理	允许载波同步中断。
INTTW0UD_off()	INTTW0UD 禁止处理	禁止载波同步中断。
INTP5_on()	INTP5 允许处理	允许速度信息获取中断。
INTP5_off()	INTP5 禁止处理	禁止速度信息获取中断。
INTTM51_on()	INTTM51 允许处理	允许对电流内环的定时器中断。
int_speed()	速度信息获取的中断服务程序	设置促进更新速度信息的标志。
int_fault()	过流中断服务程序	停止电动机处理。
int_carrier()	载波中断（波谷）服务程序	切换激励模式，更新占空比并诊断电动机工作状态。
int_TM51()	电流获取中断服务	获取电流值。
get_coswt()	COS 操作处理	执行 COS 操作。
set_pwm()	处理 PWM 值	根据获取的霍尔传感器信号得到最优的 PWM 值的操作。

4.11.3 参考程序函数

表 4-8. 参考程序函数

函数名	功能	用途
print_error()	错误显示	用 LED 显示错误状态。
get_sw()	开关读取指令	读取 MC-IO 板上开关指示信息并返回读取对应开关的操作信息。
speed_print()	速度显示	用 LED 显示速度。
led_print()	显示数据产生	产生并传递数据到 LED。
led_set()	LED 显示	用 LED 显示数据。
wait()	等待	等待指定的时间。
startup_disp()	启动期间的 LED 显示	启动期间的 LED 显示
vol2speed()	速度指令	获取速度。
get_vol()	调控器读取指令	指示 MC-IO 板上调控器信息的读取。
init_PORT()	端口设置	执行 MC-IO 板上的端口设置。
init_TM50()	TM50 设置处理	设置 8 位定时器。
start_TM50()	TM50 启动处理	启动 8 位定时器。
wait_TM50()	TM50 等待处理	等待指定的时间。
clear_WDTM()	WDT 清除处理	清除看门狗定时器计数器。

4.12 流程图

每个函数的流程图如下所示。

- 电动机库

图 4-7. 电动机初始设置处理（motor_init 函数）

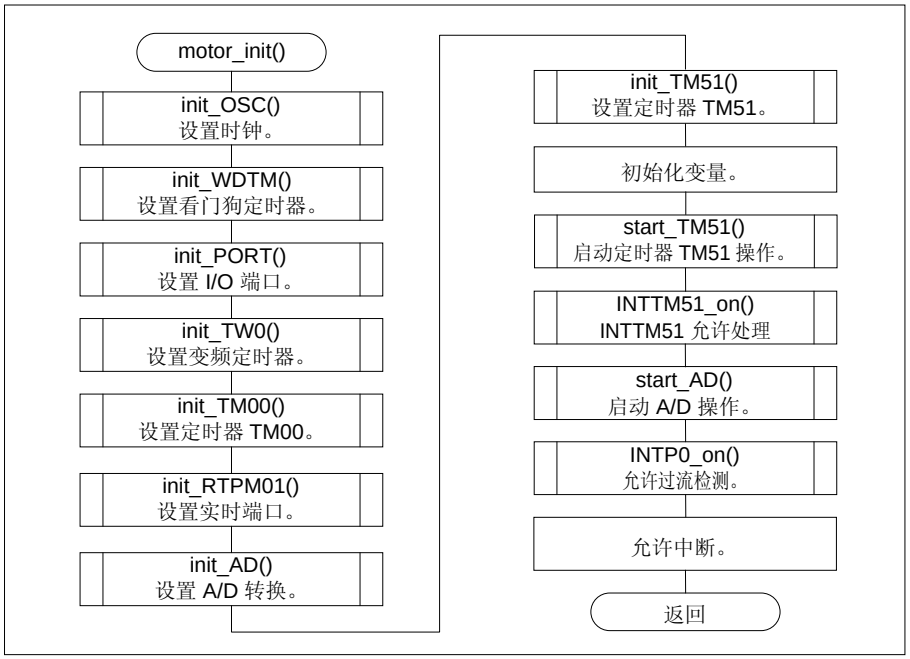


图 4-8. 电动机启动处理（motor_start 函数）

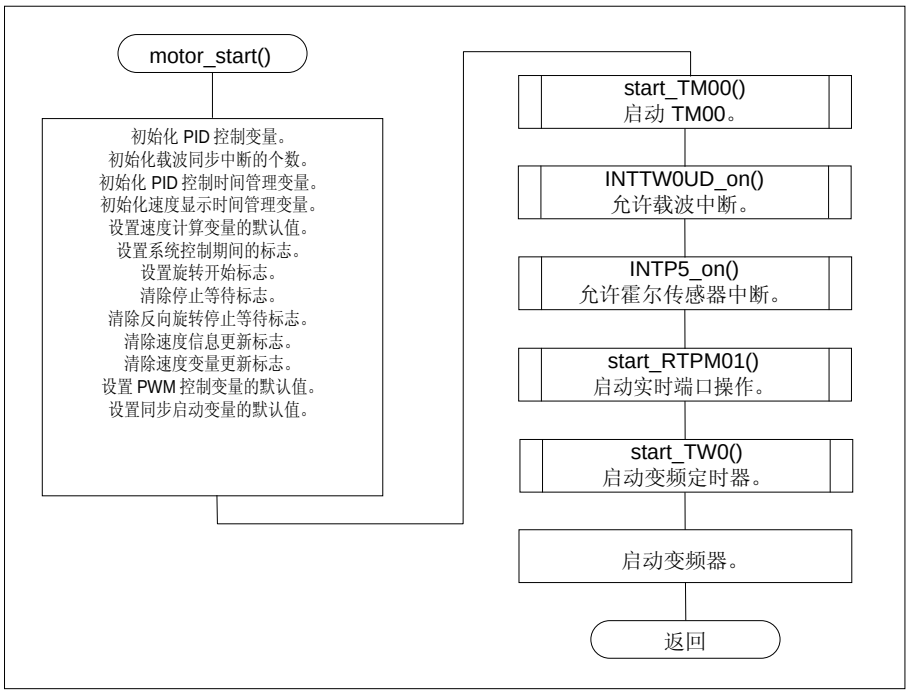


图 4-9. 电动机停止处理（motor_stop 函数）

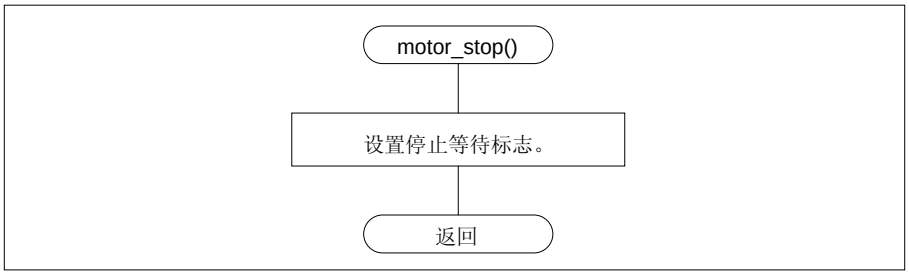


图 4-10. 电动机旋转方向更改处理（motor_rotation 函数）

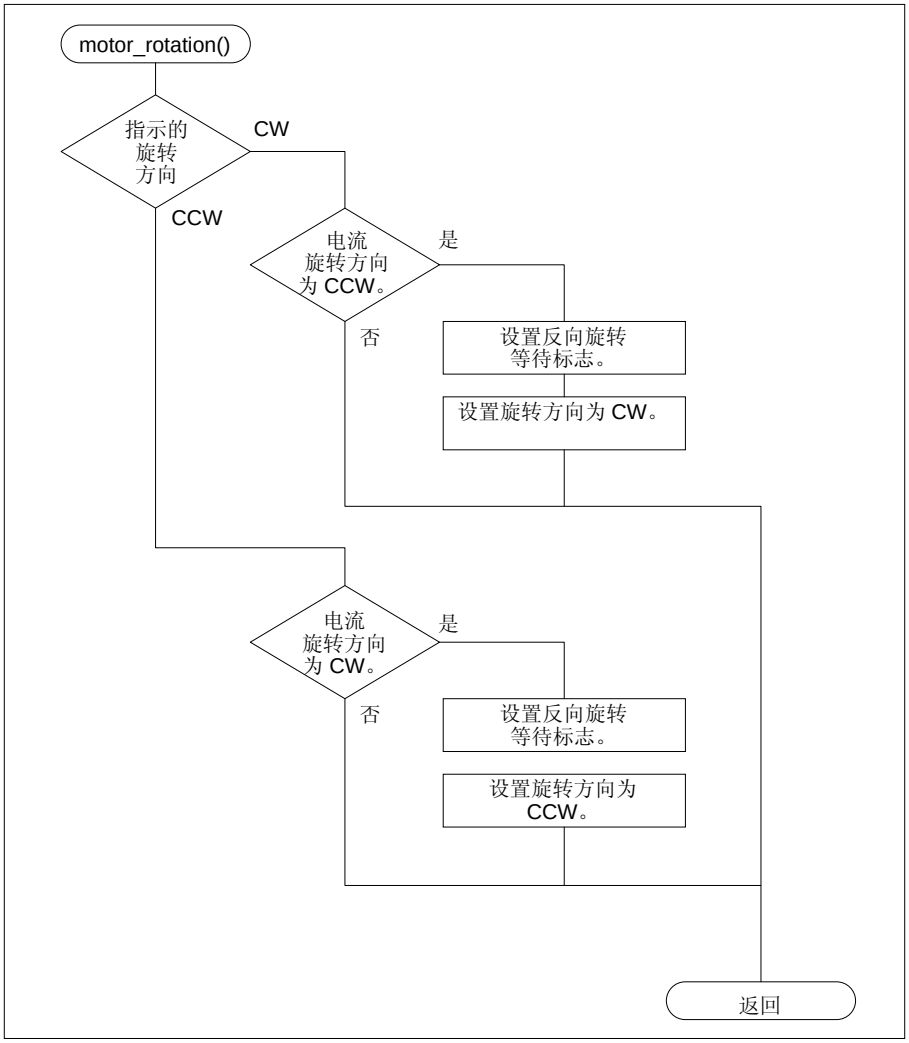


图 4-11. 速度 PID 控制处理（motor_pid 函数）

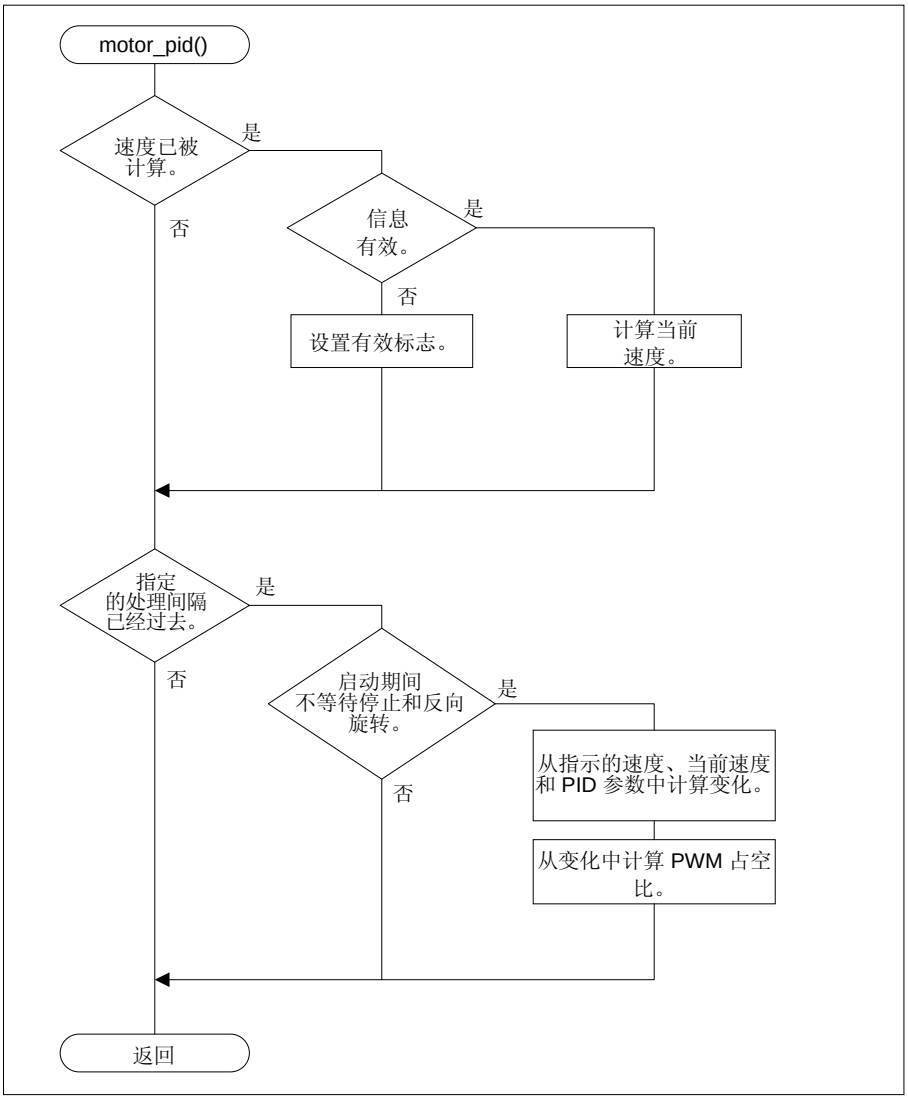


图 4-12. 电动机控制参数更改处理（motor_pset 函数）

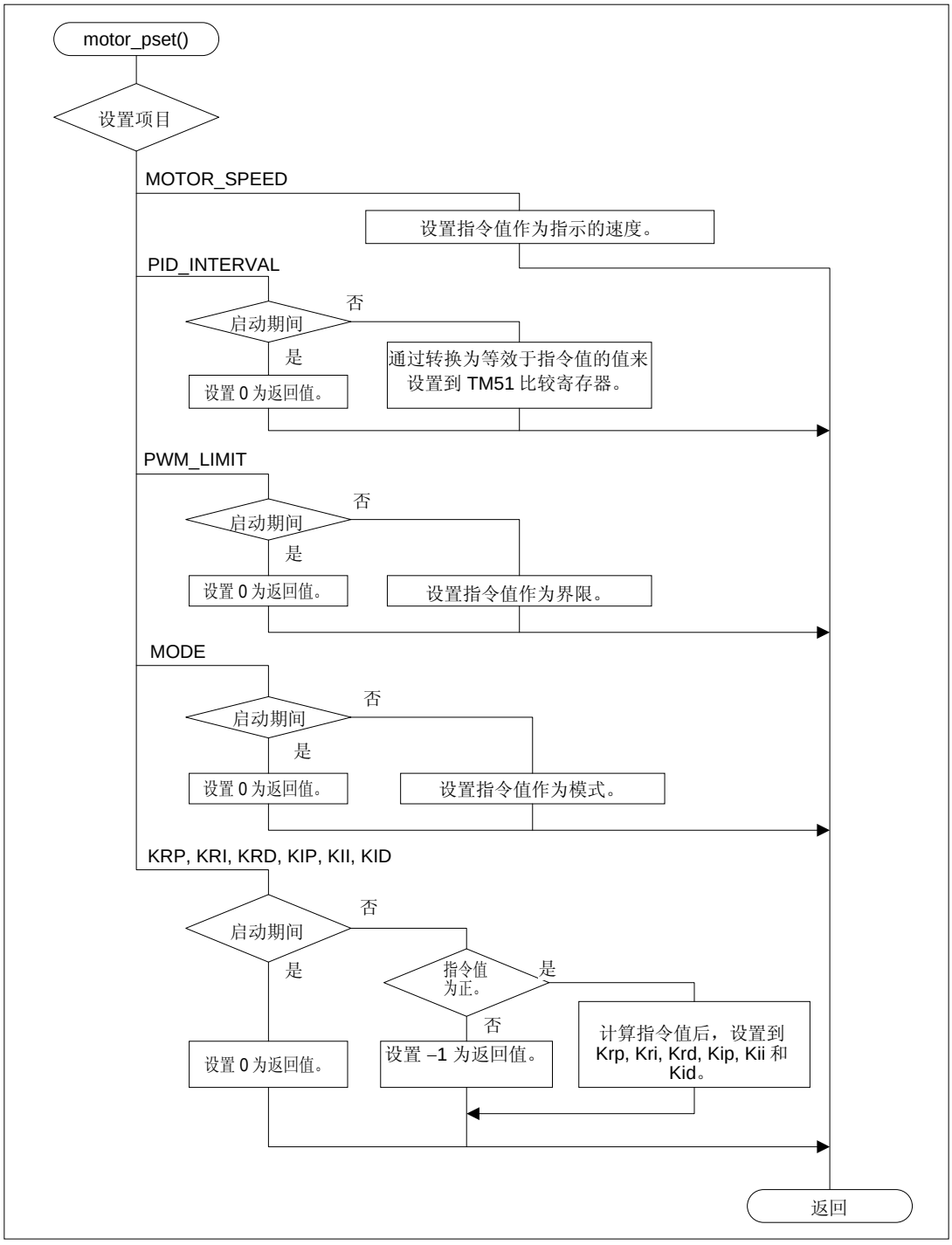


图 4-13. 从反向旋转停止的重启处理（system_restart 函数）

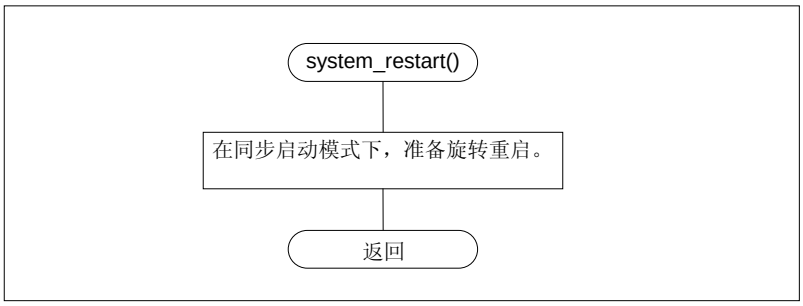


图 4-14. 系统停止处理（system_stop 函数）

图 4-15. 端口设置处理（init_PORT 函数）

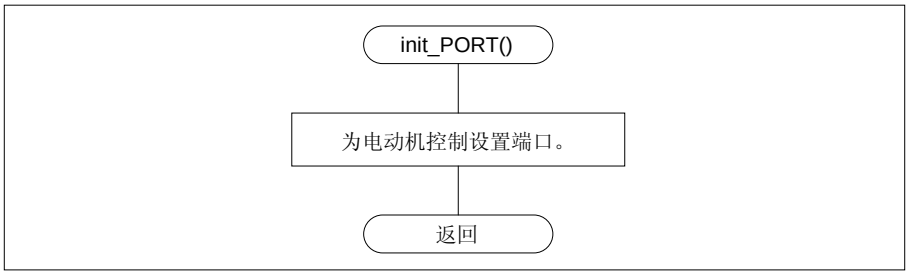


图 4-16. 时钟切换处理（init_OSC 函数）

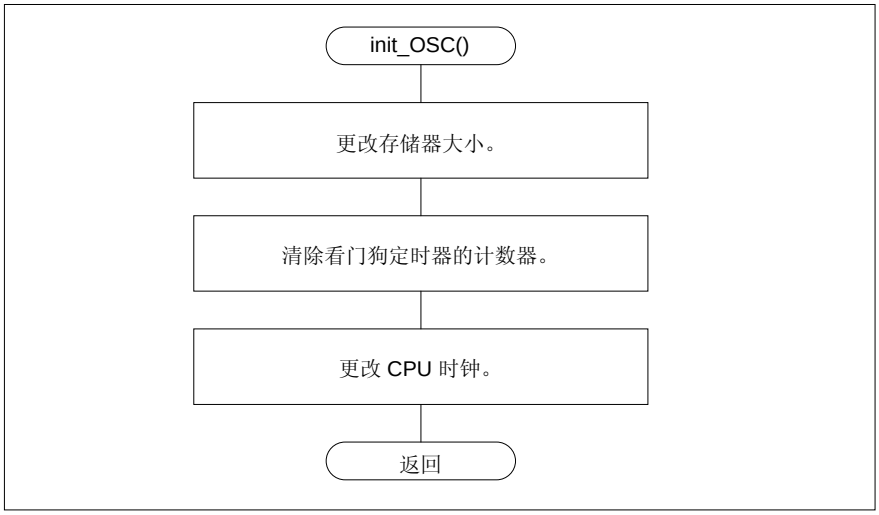


图 4-17. 变频定时器初始设置处理（init_TW0 函数）

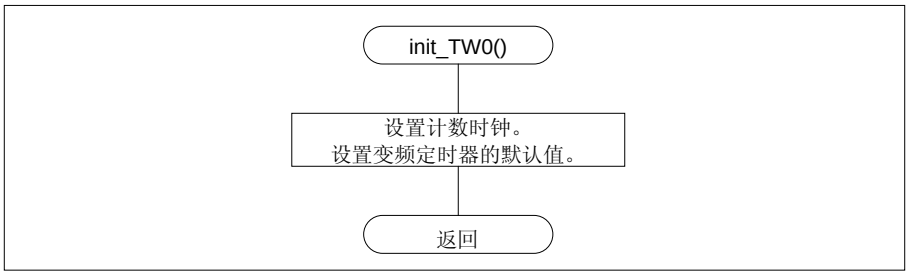


图 4-18. 变频定时器操作启动处理（start_TW0 函数）

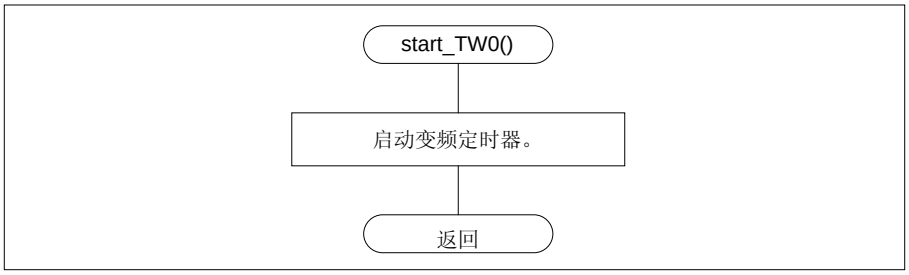


图 4-19. 变频定时器操作停止处理（stop_TW0 函数）

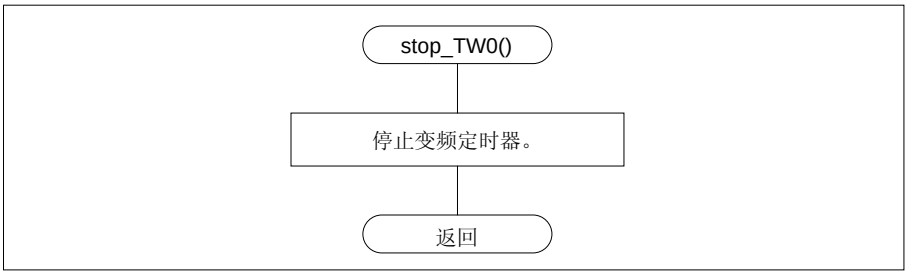


图 4-20. PWM 占空比设置处理（set_TW0 函数）

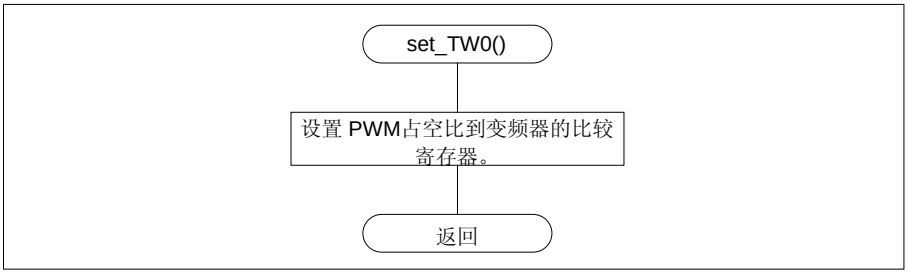


图 4-21. 激励模式切换的定时器初始设置处理（init_TM00 函数）

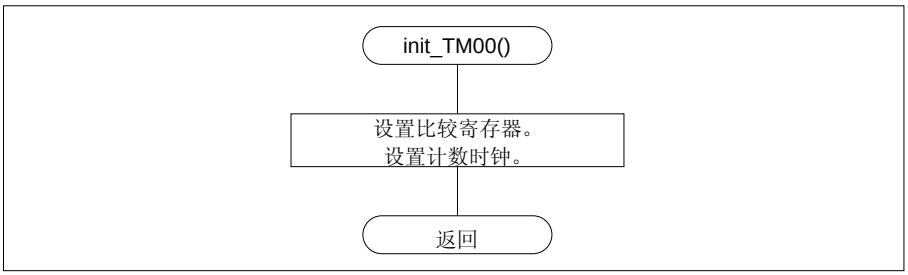


图 4-22. 激励模式切换的定时器启动处理（start_TM00 函数）

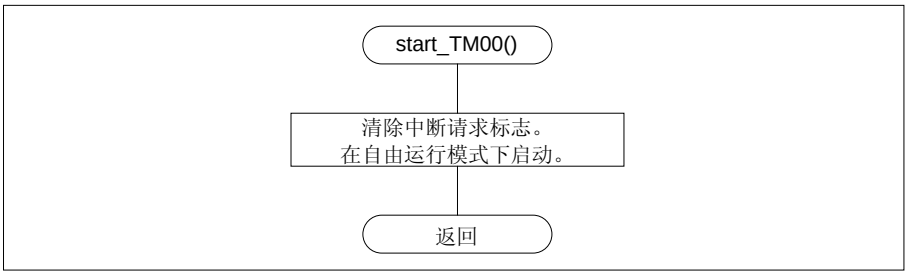


图 4-23. 激励模式切换的定时器停止处理（stop_TM00 函数）

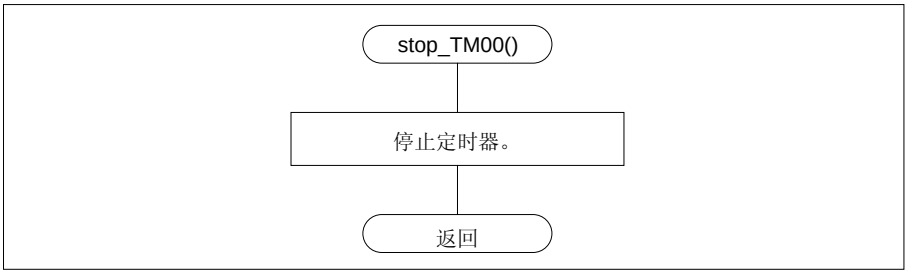


图 4-24. 激励模式切换的端口初始设置处理（init_RTPM01 函数）

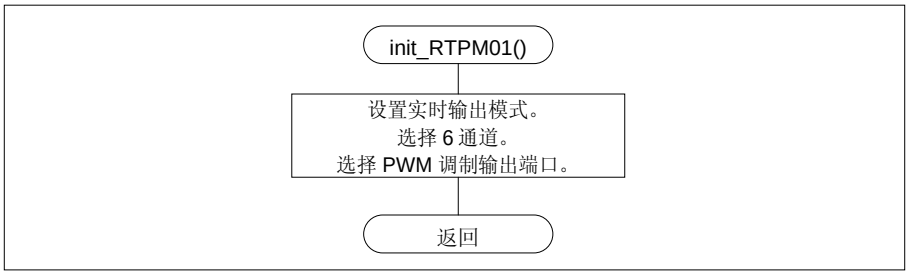


图 4-25. 激励模式切换的端口输出允许处理（start_RTPM01 函数）

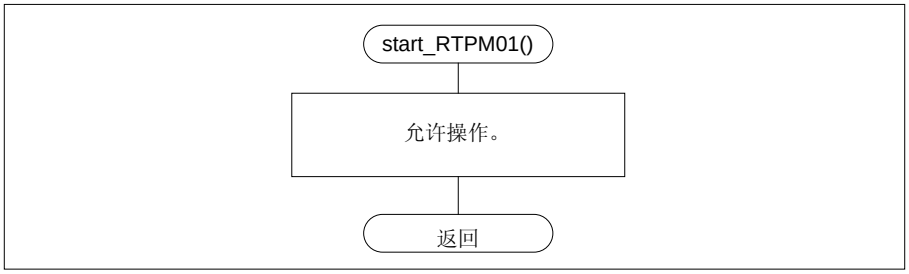


图 4-26. 激励模式切换的端口输出禁止处理（stop_RTPM01 函数）

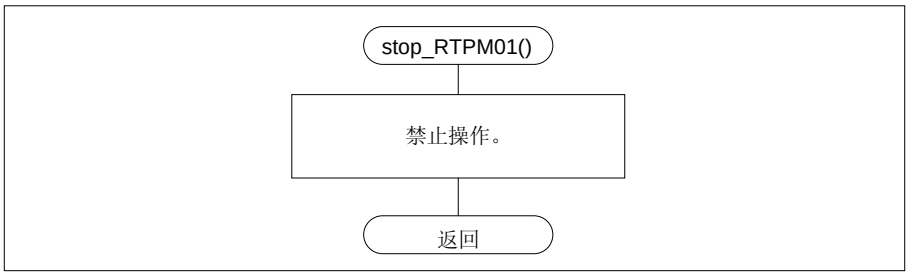


图 4-27. 激励模式切换的端口 PWM 输出模式切换处理（end_RTPM01 函数）

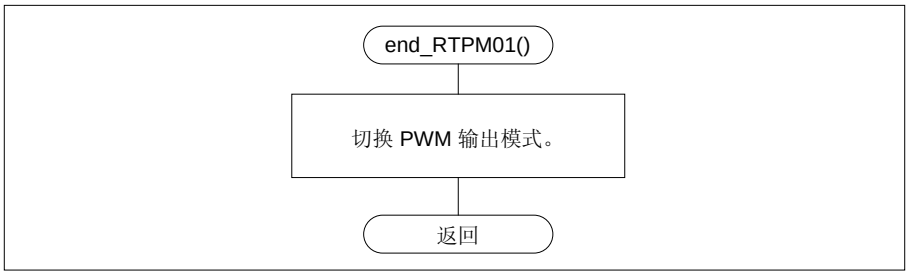


图 4-28. 激励模式设置（set_RTPM01 函数）

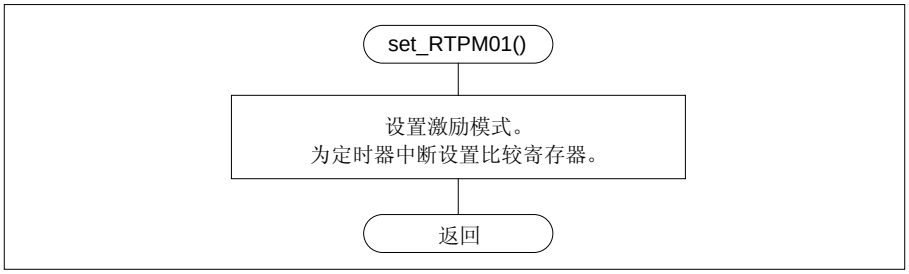


图 4-29. A/D 初始设置处理（init_AD 函数）

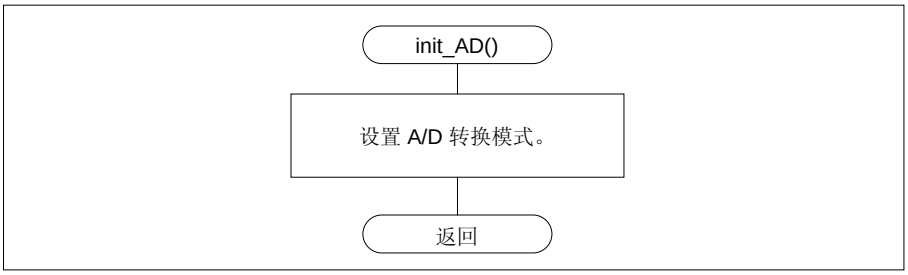


图 4-30. A/D 转换操作启动处理（start_AD 函数）

图 4-31. 看门狗定时器初始设置处理（init_WDTM 函数）

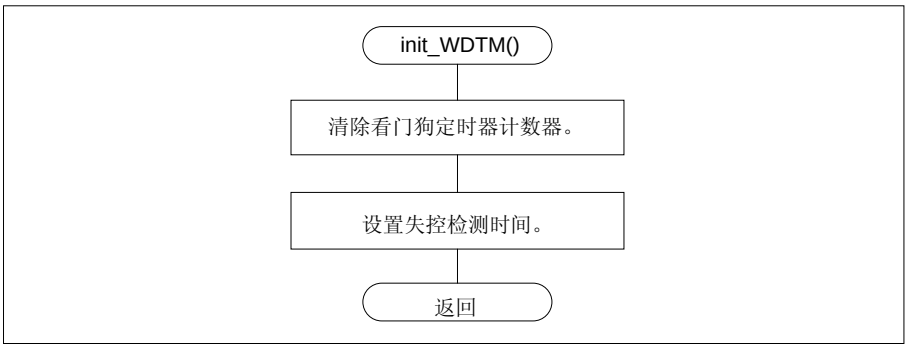


图 4-32. 8 位定时器（TM51）初始设置处理（init_TM51 函数）

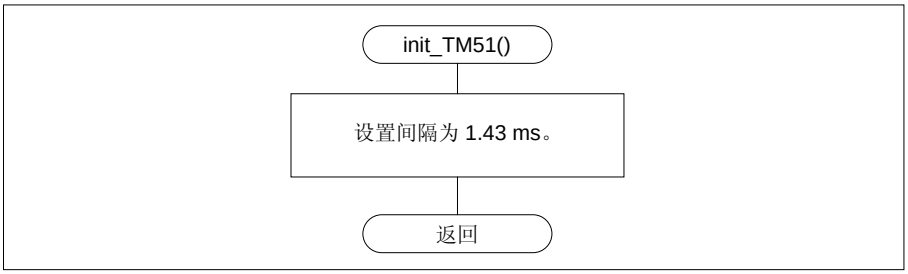


图 4-33. 8 位定时器（TM51）启动处理（start_TM51 函数）

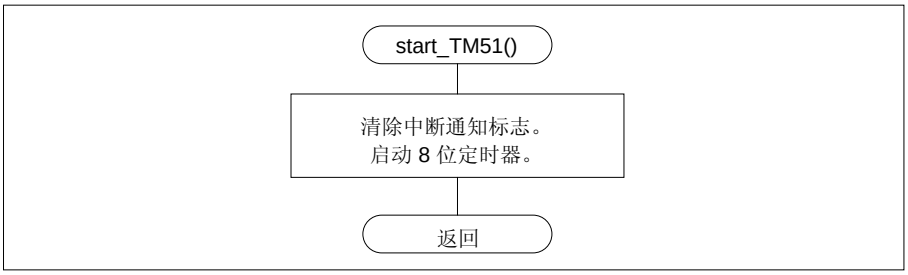


图 4-34. 霍尔传感器读取处理（read_Hall_IC 函数）

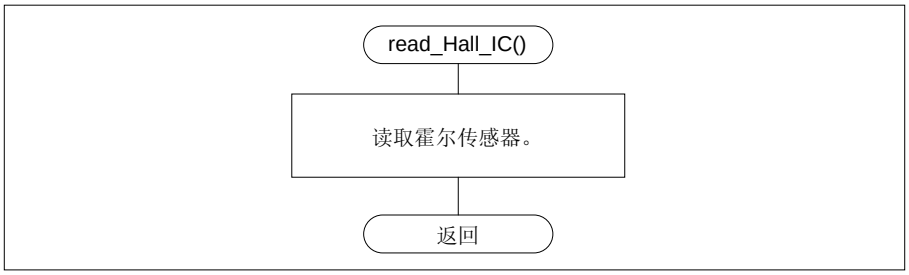


图 4-35. 过流中断允许处理（INTP0_on 函数）

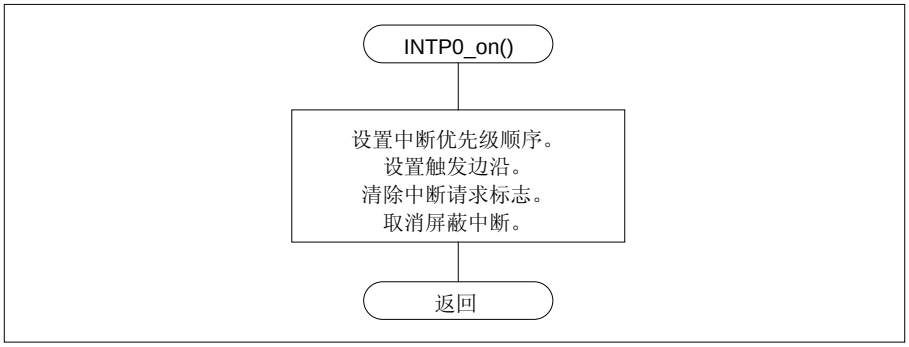


图 4-36. 载波同步中断（波谷处理）允许处理（INTTW0UD_on 函数）

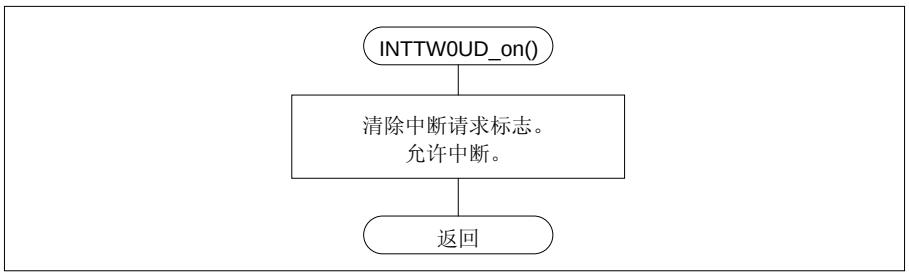


图 4-37. 载波同步中断（波谷处理）禁止处理 （INTTW0UD_off 函数）

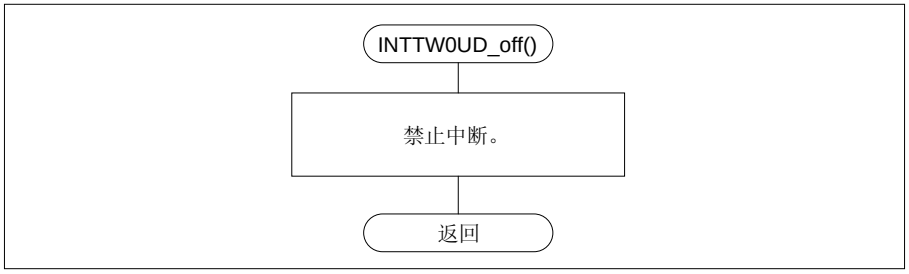


图 4-38. INTP5 中断允许处理（INTP5_on 函数）

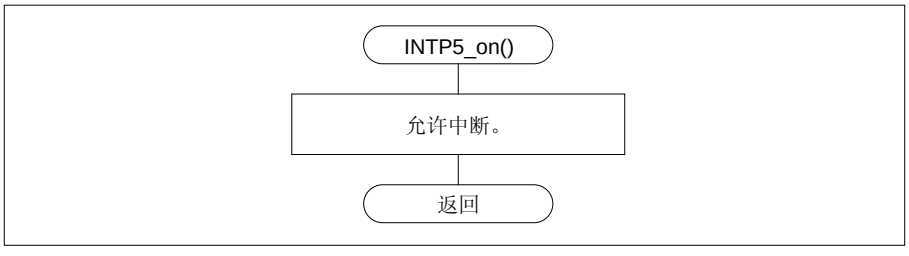


图 4-39. INTP5 中断禁止处理（INTP5_off 函数）

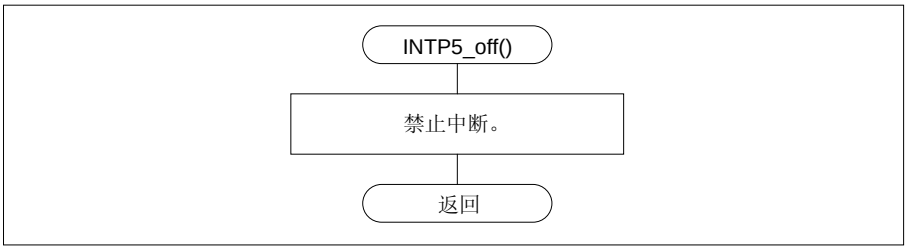


图 4-40. 电流值获取允许处理（INTTM51_on 函数）

图 4-41. 过流中断服务程序（int_fault 函数）

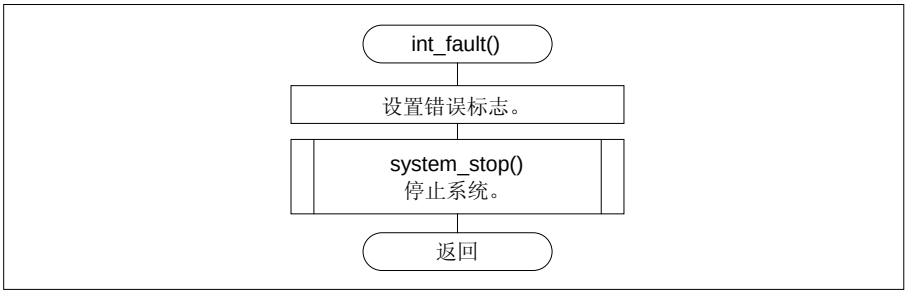


图 4-42. 速度获取中断服务程序（int_speed 函数）

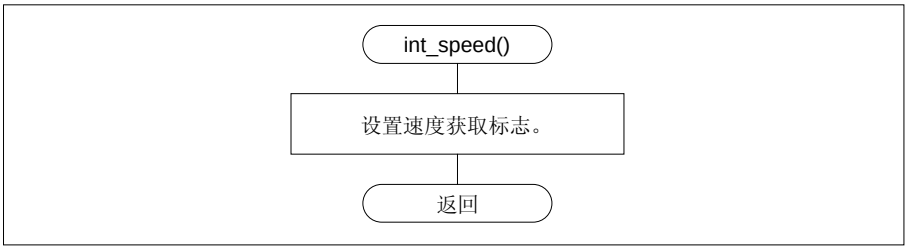


图 4-43. 电流获取中断服务程序（int_TM51 函数）

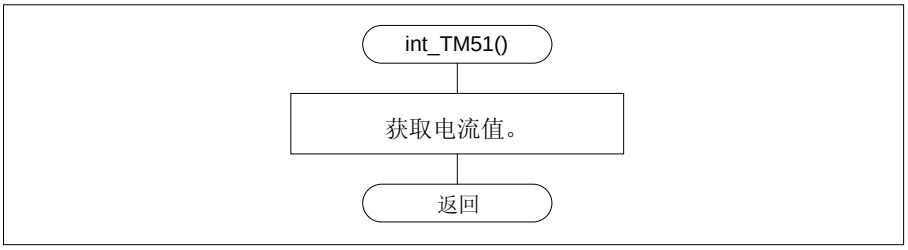


图 4-44. COS 操作处理（get_coswt 函数）

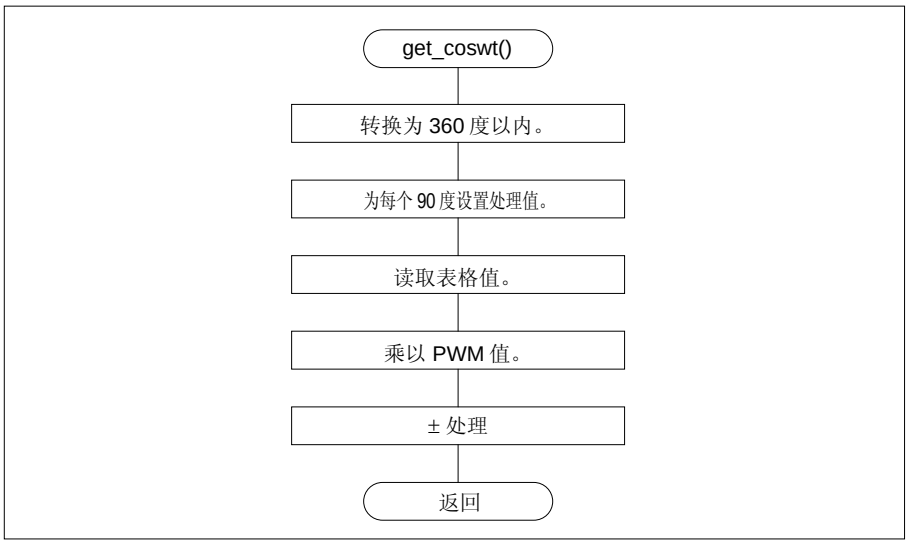


图 4-45. PWM 设置处理（set_pwm 函数）

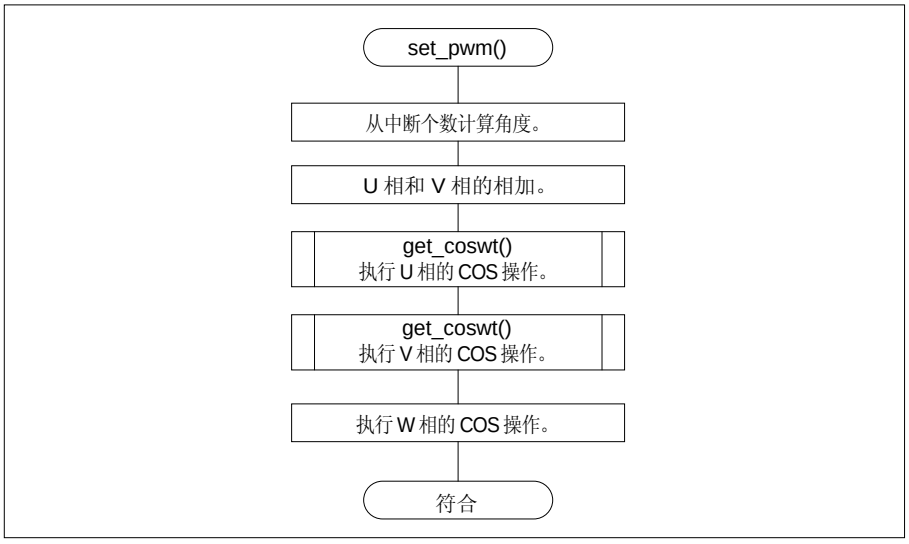


图 4-46. 载波同步中断（波谷处理）（1 / 2）（int_carrier 函数）

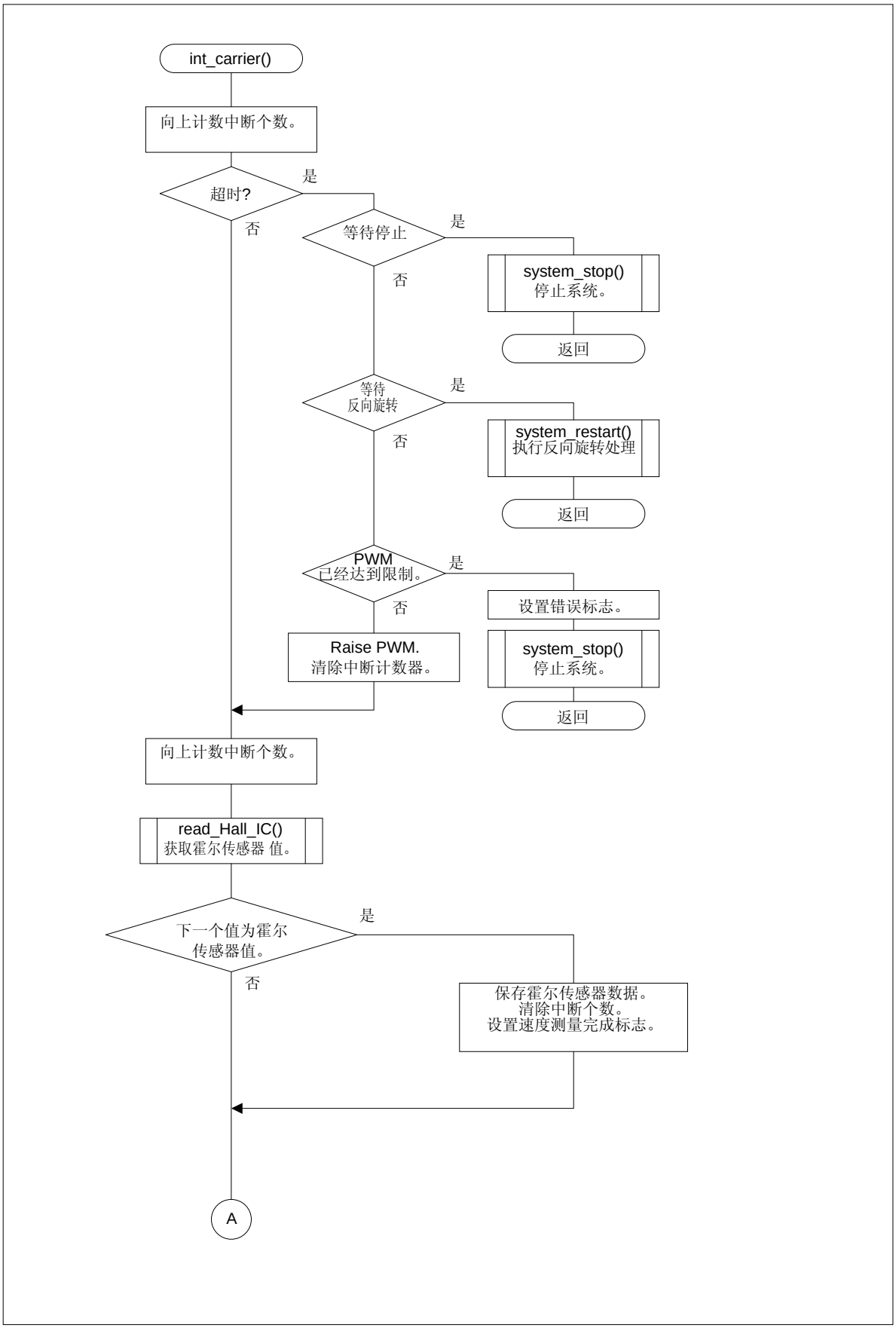
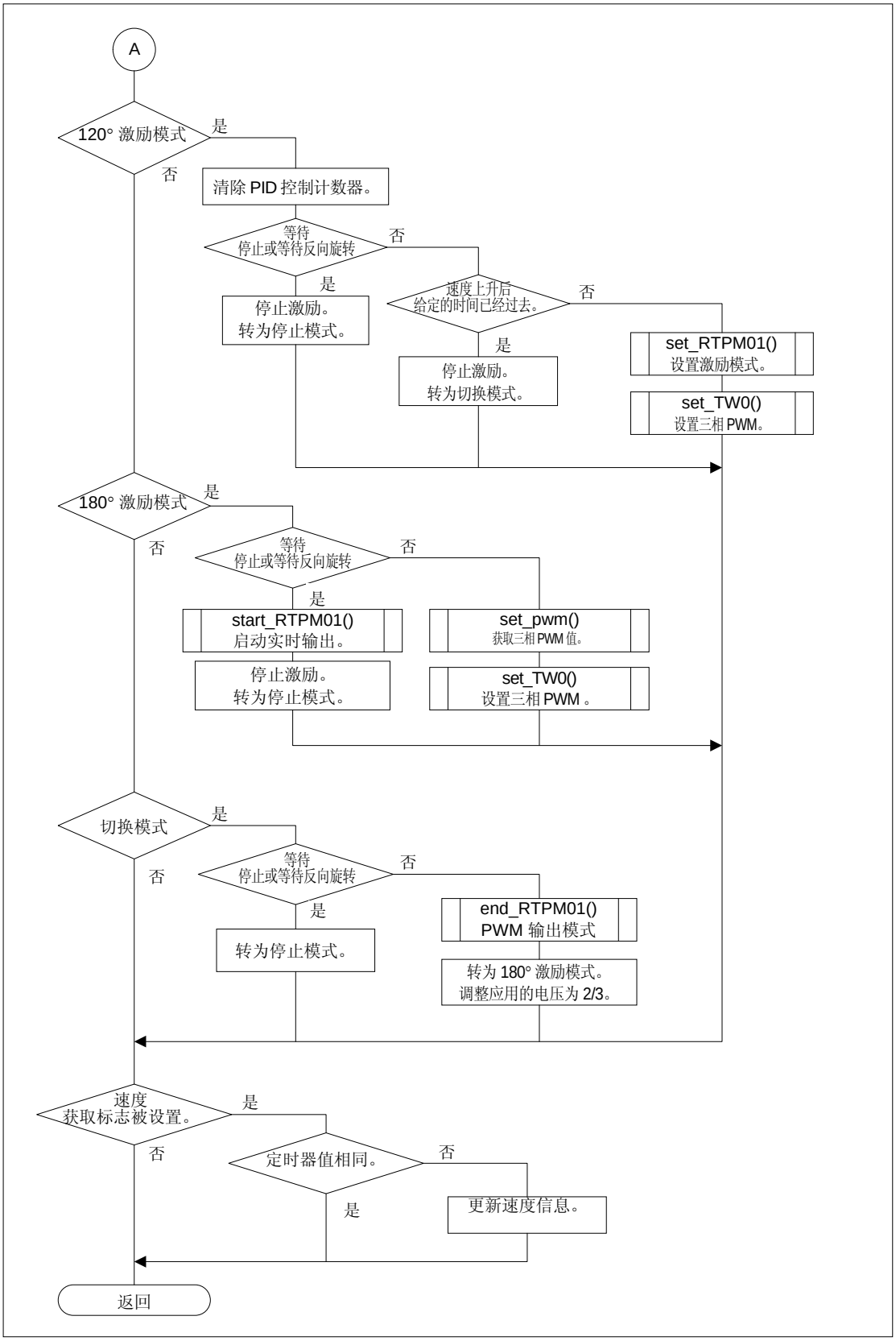


图 4-47. 载波同步中断（波谷处理）（2 / 2）（int_carrier 函数）



4.13 电动机库常量列表

4.13.1 用户可更改的常量

- 低压变频器设置

表 4-9. 用户可更改的电动机库常量（L / V）

名称	含义	设置值	备注
PWM_F_MIN	占空比的最小值	5	
PWM_F_MAX	占空比的最大值	PWM_BASE_REF	
STARTUP_METHOD_REF	初次启动方法	PWM_START	对于无传感器控制的初次启动处理的设置。 其余设置，需要与 GUI 兼容。 只使用 PWM2_REF。
T_KNEE_REF	初次启动切换时间	0.75	
T_END_REF	初次启动结束时间	1.50	
RPM0_REF	初次启动期间的初始转数	60	
RPM1_REF	初次启动切换期间的转数	100	
RPM2_REF	初次启动结束时的转数	200	
I_REF0_REF	初次启动期间的初始电流值	140	
I_REF1_REF	初次启动切换期间的电流值	160	
I_REF2_REF	初次启动结束时的电流值	160	
PWM0_REF	初次启动期间的初始 PWM 值	100	
PWM1_REF	初次启动切换期间的 PWM 值	100	
PWM2_REF	初次启动结束时的 PWM 值	100	
PWM_F_START	启动时的 PWM 值	10	
ADGAIN_REF	A/D 转换值的放大量	1.0	
ADOFFSET_REF	A/D 转换值的偏移量	0	
MAXCURRENT_REF	最大电流值	1024	
MINCURRENT_REF	最小电流值	10	
MAXSPEED_REF	最大转数	5000	
MINSPEED_REF	最小转数	300	
I_RATE_MAX_REF	电流单位时间的上升率	900	
RPM_RATE_MAX_REF	转数单位时间的上升率	2000	
KRP_REF_REF	转数的 Kp 值	0.05	
KRI_REF_REF	转数的 Ki 值	0.02	
KRD_REF_REF	转数的 Kd 值	0.01	
KIP_REF_REF	电流的 Kp 值	0.10	
KII_REF_REF	电流的 Ki 值	0.04	
KID_REF_REF	电流的 Kd 值	0.02	
CLIMIT	过流检测的电流值	700	

4.13.2 用户可引用的常量

表 4-10. 用户可引用的电动机库常量

名称	含义	设置值	备注
FLG_ON	标志值	1	表明标志是否有效。
FLG_OFF	标志值	0	
CW	旋转方向	0	表明电动机的旋转方向。
CCW	旋转方向	1	
ERROR_NONE	错误标志位	0x00	无错误
ERROR_HALL	错误标志位	0x01	霍尔传感器错误
ERROR_OC	错误标志位	0x02	过流引起的错误
ERROR_MOTOR	错误标志位	0x04	电动机故障引起的错误
ERROR_S_OC	错误标志位	0x08	过流引起的错误
MOTOR_SPEED	参数设置代码	0x01	设置电动机的转数。
PID_INTERVAL	参数设置代码	0x10	设置 PID 间隔。
MODE	参数设置代码	0x12	设置模式。
KRP	参数设置代码	0x20	设置转数的 Kp 值。
KRI	参数设置代码	0x21	设置转数的 Ki 值。
KRD	参数设置代码	0x22	设置转数的 Kd 值。
KIP	参数设置代码	0x23	设置电流的 Kp 值。
KII	参数设置代码	0x24	设置电流的 Ki 值。
KID	参数设置代码	0x25	设置电流的 Kd 值。
WDTE_CLR	看门狗定时器值	0xac	WDT 清除值
CURRENT_START	初次启动模式	0x01	
PWM_START	初次启动模式	0x02	
NOTABLE_START	初次启动模式	0x03	
SPEED_CMD	操作模式	0x01	
I_CMD	操作模式	0x02	
V_CMD	操作模式	0x03	
AD_ISHUNT	电流检测通道	5	
UNIT_RPM	计算转数的常量	2343750	

4.13.3 内部常量

表 4-11. 电动机库内部常量

名称	含义	设置值	备注
FLG_ON	标志值	1	表明标志是否有效。
FLG_OFF	标志值	0	
FLG_WAIT	标志值	2	等待停止。
FLG_SW	标志值	3	等待切换。
FLG_120	标志值	12	120° 励磁方法
FLG_180	标志值	18	180° 励磁方法
PWM_BASE_REF	PWM 基数	1000	用于 PWM 输出
PWM_F_REF	PWM 默认值	0	
PWM_DTM_REF	死区时间	0	
IN	用于端口设置	1	用来指定端口功能
OUT	用于端口设置	0	
CLEAR	用于寄存器位设置	0	用来访问寄存器的位
SET	用于寄存器位设置	1	
INV_OFF	用于变频控制	1/0	低压变频器
INV_ON	用于变频控制	0/1	
INVERTER_SW	变频控制端口	P54	变频控制端口
INVERTER_SW_MODE	变频控制端口	PM54	变频控制端口
INTP0	端口控制寄存器	PM00	过流检测端口
IMS_DATA	存储器大小切换	0xc8	
WDTM_SET	WDT 设置值	0x6f	
P_OFF	激励模式	0x3f	实时端口的激励
P_STOP	激励模式	0x15	
P_T1	激励模式	0x36	
P_T2	激励模式	0x1e	
P_T3	激励模式	0x1b	
P_T4	激励模式	0x39	
P_T5	激励模式	0x2d	
P_T6	激励模式	0x27	
INT_CNT_MAX	界限值	500	用于判断没有旋转（时间）
START_PWM_FF_MAX	界限值	300	用于判断没有旋转（电压）
DELTA_CW	超前角：	0	CW 端的超前角
DELTA_CCW	超前角：	0	CCW 端的超前角
CHANGE_SPEED	界限值	100	从 120° 激励到 180°激励的切换速度
CR01_ADD	计数器值	3	激励模式切换等待时间
PWM_MIDDLE	PWM 值	500	PWM_BASE_REF ÷ 2
PWM_x3	PWM 值	1500	PWM_MIDDLE × 3
INTERVAL_BASE	参考值	5	产生 1 [ms]的参考
TIME_OUT	参考值	5000	产生 1 [s]的参考

4.14 参考程序常量列表

4.14.1 内部常量

表 4-12. 参考程序内部常量（1 / 2）

名称	含义	设置值	备注
AD_VOL	A/D 通道	4	MCIO 板上的调控器
IN	用于端口设置	1	用来指定端口功能
OUT	用于端口设置	0	
CLEAR	用于寄存器位设置	0	用来访问寄存器的位
SET	用于寄存器位设置	1	
KEY_WAIT	切换观察时间	10	用来消除抖动
SW	开关	（P7&0xf）	切换连接端口
SW2	端口控制寄存器	PM73	START/STOP 的端口
SW3	端口控制寄存器	PM72	FORWARD 的端口
SW4	端口控制寄存器	PM71	REVERSE 的端口
SW5	端口控制寄存器	PM70	MODE 的端口
LD_LED0	端口控制寄存器	PM64	用于 LED 选择的端口
LD_LED1	端口控制寄存器	PM65	
LD_LED2	端口控制寄存器	PM66	
LD_LED3	端口控制寄存器	PM67	
LD_DATA	端口控制寄存器	PM4	向 LED 输出数据的端口
START_SW	启动	0x7	开关状态
STOP_SW	停止	0x7	
FORWARD_SW	CW	0xb	
REVERSE_SW	CCW	0xd	
MODE_SW	模式	0xe	
START_TR	用于状态设置	0x01	开始控制。
STOP_TR	用于状态设置	0x02	停止控制。
FORWARD_TR	用于状态设置	0x04	更改旋转为 CW。
REVERSE_TR	用于状态设置	0x08	更改旋转为 CCW。
MODE_TR	用于状态设置	0x10	MODE 开关被按下的状态
LED_0	LED 显示数据	0xc0	显示“0”。
LED_1		0cf9	显示“1”。
LED_2		0xa4	显示“2”。
LED_3		0xb0	显示“3”。
LED_4		0x99	显示“4”。
LED_5		0x92	显示“5”。
LED_6		0x82	显示“6”。
LED_7		0xf8	显示“7”。
LED_8		0x80	显示“8”。
LED_9		0x98	显示“9”。
LED_O		0xc0	显示“0”而不是“O”。
LED_I		0xcf	显示“I”。
LED_C		0xc6	显示“C”。
LED_H		0x89	显示“H”。

表 4-12. 参考程序内部常量（2 / 2）

名称	含义	设置值	备注
LED_A	LED 显示数据	0x88	显示“A”。
LED_L		0xc7	显示“L”。
LED_		0xff	显示“ ”。
LED_S		0x92	显示“S”。
LED_E		0x86	显示“E”。
LED_F		0x8e	显示“F”。
LED_P		0x8c	显示“P”。
LED_dot		0x7f	显示“.”。

4.15 电动机库变量列表

4.15.1 外部公开变量

表 4-13. 外部公开的电动机库变量（1 / 2）

变量名	类型	含义	备注
sys_flag	char	控制标志	控制状态
err_flag	char	错误标志	错误状态
stop_wait	char	停止命令标志	表明是否发布停止命令。
cw_ccw_flag	char	旋转方向命令标志	旋转方向状态
cw_ccw_wait	char	反向停止命令标志	表明是否发布反向停止的命令。
maxed_flags	unsigned char	上限标志	当单位时间的上限被超过时设置的标志
print_cnt	unsigned int	速度显示时序	载波同步中断个数
pwm_ff	int	当前 PWM 命令值	当前 PWM 占空比值
pwm_ff_o	int	PWM 命令值	PWM 占空比命令值
m_speed	int	实际速度	电动机的转速（rpm）
krp_ref	float	转数的 Kp 值	速度控制
kri_ref	float	转数的 Ki 值	
krd_ref	float	转数的 Kd 值	
kip_ref	float	电流的 Kp 值	电流内环控制
kii_ref	float	电流的 Ki 值	
kid_ref	float	电流的 Kd 值	
startup_method	unsigned char	同步启动方法	
I_ref	float	电流命令的电流值	
I_ref_o	float	电流命令值	
I_measured	float	工作电流值	
speed_ref	int	当前速度命令值	
speed_ref_o	int	速度命令值	
maxspeed	int	最大速度	
minspeed	int	最小速度	
dspeed_ref_max	int	单位时间内控制的速度变量	
dI_ref_max	float	单位时间内控制的电流变量	
RPM_rate_max	float	单位时间内的速度变化	

表 4-13. 外部公开的电动机库变量（2 / 2）

变量名	类型	含义	备注
I_rate_max	float	单位时间内的电流变化	
t_knee	float	初次启动切换时间	
t_end	float	初次启动结束时间	
RPM0	float	初次启动期间的初始转数	
RPM1	float	初次启动切换期间的转数	
RPM2	float	初次启动结束时的转数	
I_ref0	float	初次启动期间的初始电流值	
I_ref1	float	初次启动切换期间的电流值	
I_ref2	float	初次启动结束时的电流值	
PWM0	float	初次启动期间的初始 PWM 值	
PWM1	float	初次启动切换期间的 PWM 值	
PWM2	float	初次启动结束时的 PWM 值	
adgain	float	A/D 转换值的放大量	
adoffset	float	A/D 转换值的偏移量	
maxcurrent	float	最大电流值	
mincurrent	float	最小电流值	

4.15.2 内部变量

表 4-14. 电动机库内部变量

变量名	类型	含义	备注
old_hdata	char	霍尔传感器历史	之前的霍尔传感器值
speed_flag	char	速度信息标志	表明计算速度需要的获取的数据是否存在。
int_cnt	unsigned int	中断计数器	载波同步中断个数
pid_cnt	unsigned int	PID 控制时序	载波同步中断个数
pwm_base	int	PWM 基数时钟值	PWM 载波宽度
old_ff	int	PWM 命令值历史	前面的 pwm_ff 值
up_flag	char	实际速度计算状态标志	用于计算旋转速度
capture_flag	char	定时器值获取标志	
clk_flag	char	定时器值计算允许标志	
old_clk	unsigned int	前面的时钟值	
new_clk	unsigned int	最新的时钟值	
mvn_f	float	前面的操作的总和	
startdelay	char	电流检测开始延迟值	
start_flag	char	旋转开始标志	
cy_time	int	PID 控制间隔	
cmd_mode	char	控制模式	
dpwm_ff_max	int	单位时间内控制的 PWM 变量	

4.16 参考程序变量列表

4.16.1 内部变量

表 4-15. 参考程序变量

ad_flag	char	速度更改标志	限制指定的速度更改函数。
led_data[ ]	char	LED 输出数据	用 LED 显示的值

4.17 电动机库源文件

存储器大小 ROM： 23A8h
RAM： 420h

```
/*


   BLDCM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标 ： uPD78F0714 电动机入门套件


   日期 ： 2007/05/30
   文件名 ： motor.h

   NEC Micro Systems,Ltd
*/

#ifdef LOW
/* ================================================ */
/* 用于 Pittman 电动机 */
#define PWM_F_MIN        5                          /* 最大值                      */
#define PWM_F_MAX        PWM_MIDDLE            /* 最小值                      */

/* 启动参数 */
#define STARTUP_METHOD_REF PWM_START  /* 使用的方法： PWM_START，CURRENT_START 或 NOTABLE_START */
#define T_KNEE_REF       2.0                        /* 第二个启动段的开始 */
#define T_END_REF        4.0                        /* 第二个启动段的结束 */
#define RPM0_REF         100                        /* 初始启动 RPM */
#define RPM1_REF         200                        /* 中点启动 RPM */
#define RPM2_REF         300                        /* 最后启动 RPM */
#define I_REF0_REF       140                        /* 初始启动电流命令 */
#define I_REF1_REF       160                        /* 中点启动电流命令 */
#define I_REF2_REF       160                        /* 最后启动电流命令 */
#define PWM0_REF         140                        /* 初始启动电压设置 */
#define PWM1_REF         145                        /* 中点启动电压设置 */
#define PWM2_REF         150                        /* 最后启动电压设置 */

/* 最初启动 NOTABLE_START */
#define SYNC_DEF_REF     500
#define PWM_F_START      150                        /* 开始时的默认值                  */

/* a/d 增益参数 （主要用于 GUI ） */
#define ADGAIN_REF       4.0                        /* mA = Gain*(A/D - Offset) */
#define ADOFFSET_REF     0                          /* */
#define MAXCURRENT_REF   1023                  /* mA */
#define MINCURRENT_REF   10                         /* mA */
#define MAXSPEED_REF     5000                       /* RPM */
#define MINSPEED_REF     300                        /* RPM */

/* 设定值最大变化率 */
/* 用于设定值变化 */
#define I_RATE_MAX_REF   900                        /* dI_int/sec 最大值 */
#define RPM_RATE_MAX_REF 2000                  /* RPM/sec */

/* 反馈增益： A/D 单元内测量的电流，PWM%*10 内的电压*/
/* 反馈到 I 命令的 RPM #define */
#define KRP_DEF_REF      0.05                       /* dI_ints/dRPM 误差 */
#define KRI_DEF_REF      0.02                       /* dI_ints/RPM 误差 */
#define KRD_DEF_REF      0.01                       /* dI_ints/d(dRPM 误差) */

/* 反馈到 PWM 命令的电流 */
#define KIP_DEF_REF      0.025                      /* dPWM/dI_error */
#define KII_DEF_REF      0.010                      /* dPWM/I_error */
#define KID_DEF_REF      0.005                      /* dPWM/d(derror) */

/* ================================================ */
#else
/* ================================================ */
/* 用于最初的电动机 */
#define PWM_F_MIN        5                          /* 最大值                      */
#define PWM_F_MAX        PWM_BASE_REF       /* 最小值                      */

/* 启动参数 */
#define STARTUP_METHOD_REF PWM_START /* 使用的方法： PWM_START，CURRENT_START 或 NOTABLE_START */
#define T_KNEE_REF       0.75                       /* 第二个启动段的开始 */
#define T_END_REF        1.5                        /* 第二个启动段的结束 */
#define RPM0_REF         300                        /* 初始启动 RPM */
#define RPM1_REF         300                        /* 中点启动 RPM */
#define RPM2_REF         300                        /* 最后启动 RPM */
#define I_REF0_REF       140                        /* 初始启动电流命令 */
#define I_REF1_REF       160                        /* 中点启动电流命令 */
#define I_REF2_REF       160                        /* 最后启动电流命令 */
#define PWM0_REF         195                        /* 初始启动电压设置 */
#define PWM1_REF         195                        /* 中点启动电压设置 */
#define PWM2_REF         195                        /* 最后启动电压设置 */

/* 最初启动 NOTABLE_START */
#define SYNC_DEF_REF     500
#define PWM_F_START      70                         /* 开始时的默认值                  */

/* a/d 增益参数（主要用于 GUI） */
#define ADGAIN_REF       1.0                        /* mA = Gain*(A/D - Offset) */
#define ADOFFSET_REF     100                        /* */
#define MAXCURRENT_REF   1023                  /* mA */
#define MINCURRENT_REF   10                         /* mA */
#define MAXSPEED_REF     3000                       /* RPM */
#define MINSPEED_REF     300                        /* RPM */
```

```
/* 设定值最大变化率 */
/* 用于设定值变化     */
#define I_RATE_MAX_REF     900                          /* dI_int/sec 最大值 */
#define RPM_RATE_MAX_REF   3000                        /* RPM/sec */

/* 反馈增益： A/D 单元内测量的电流，PWM%*10 内的电压 */
/*  反馈到 I 命令的 RPM  #define */
#define KRP_DEF_REF        0.4                          /* dI_ints/dRPM 误差 */
#define KRI_DEF_REF        0.005                        /* dI_ints/RPM 误差  */
#define KRD_DEF_REF        0.05                         /* dI_ints/d(dRPM 误差) */

/*  反馈到 PWM 命令的电流 */
#define KIP_DEF_REF        0.9                          /* dPWM/dI_error */
#define KII_DEF_REF        0.9                          /* dPWM/I_error  */
#define KID_DEF_REF        0                            /* dPWM/d(derror) */

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

#define CLIMIT          1024                            /* 1024 */

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

/* +++++++++++++++ 以下内容不能更改。 +++++++++++++++ */

#define CW              0                                /* 顺时针                    */
#define CCW             1                                /* 逆时针                    */

#define ERROR_NONE      0x00
#define ERROR_霍尔      0x01                            /* 霍尔传感器 故障           */
#define ERROR_OC        0x02                             /* 过流                      */
#define ERROR_MOTOR     0x04                            /* 电动机故障              */
#define ERROR_S_OC      0x08                            /* 过流 （软件检测)        */

#define MOTOR_SPEED     0x01                            /* 指示的转数 */
#define PID_INTERVAL    0x10                            /* PID 控制间隔 */
#define PWM_LIMIT       0x11                            /* PWM 变化限制 */
#define MODE            0x12
#define KRP             0x20                            /* Kp 值 */
#define KRI             0x21                            /* Ki 值 */
#define KRD             0x22                            /* Kd 值 */
#define KIP             0x23                             /* Kp 值 */
#define KII             0x24                            /* Ki 值 */
#define KID             0x25                            /* Kd 值 */

#define WDTE_CLR        0xac

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

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

 /*
   m_speed 常数：         x[rpm] = ( 60[s] * 78.125[Krps] ) / 6
                            ------          ---
                              ↑             ↑计数器值获取
                              ↑             ↑时序
                              ↑             ↑霍尔传感器(6)
                              ↑TM00 计数时钟
                              ↑78.125[KHz]
 */
#define UNIT_RPM            2343750                     /* 2 个 */
#define CARRIRE_DEF_CONST   50000
#define SPEED_CAL_CONST     100000

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

extern char             sys_flag;                       /* 系统操作状态          */
extern int              speed_ref;                      /* 指示的转数 */
extern int              speed_ref_o;
extern int              m_speed;                        /* 当前的转数    */
extern char             stop_wait;                      /* 调整等待标志                */
extern char             cw_ccw_wait;                   /* 反向等待标志             */
extern char             cw_ccw_flag;                   /* 旋转方向状态       */
extern char             err_flag;                       /* 错误标志                    */
extern unsigned char    maxed_flags;
extern int              pwm_ff;
extern int              pwm_ff_o;
extern float            I_ref;
extern float            I_ref_o;
extern float            I_measured;
extern float            kip_ref, kii_ref, kid_ref;
extern float            krp_ref, kri_ref, krd_ref;
extern unsigned char    startup_method;
extern float            t_knee;
extern float            t_end;
extern float            RPM0, RPM1, RPM2;
extern float            I_ref0, I_ref1, I_ref2;
extern float            PWM0, PWM1, PWM2;
extern float            adgain;
extern int              adoffset;
extern float            maxcurrent, mincurrent;
extern float            I_rate_max, RPM_rate_max;
extern float            dI_ref_max;
extern int              dspeed_ref_max;
extern int              maxspeed, minspeed;
extern unsigned int     print_cnt;
extern unsigned int     ad_data_vol;

/* ---------------------------------------- */
void motor_init(void);
void motor_start(void);
void motor_stop(void);
void motor_rotation(char);
```

```
void motor_pid(void);
char motor_pset(unsigned char, long);

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

```
/*


   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标　：　uPD78F0714 电动机入门套件


   日期　：　2007/06/26
   文件名　：　motor.c

   NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma ei
#pragma di

#pragma INTERRUPT INTP0     int_fault    rb1          /* 过流发生     */
#pragma INTERRUPT INTP5     int_speed    rb1
#pragma INTERRUPT INTTW0UD  int_carrier  rb2  /* 载波同步中断 */
#pragma INTERRUPT INTTM51   int_TM51     rb3

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

/* ------------------------------------------------ */
#define IN              1                                   /* 输入 */
#define OUT             0                                   /* 输出 */

#define FLG_ON          1                                   /* 有效 */
#define FLG_OFF         0                                   /* 无效 */
#define FLG_WAIT        2                                   /* 等待切换 */
#define FLG_SW          3
#define FLG_120         12                                  /* 120 度励磁方法 */
#define FLG_180         18                                  /* 180 度励磁方法 */

#define START_TR        0x01                                /* 识别按下的开关  */
#define STOP_TR         0x02
#define FORWARD_TR      0x04
#define REVERSE_TR      0x08
#define MODE_TR         0x10

#define IMS_DATA        0xc8

#define WDTM_SET        0x6f

#define P_OFF           0x3f                                /* 实时输出端口关闭 */
#define P_STOP          0x15
#define P_T1            0x36                                /* 激励模式 1      */
#define P_T2            0x1e                                /* 激励模式 2      */
#define P_T3            0x1b                                /* 激励模式 3      */
#define P_T4            0x39                                /* 激励模式 4      */
#define P_T5            0x2d                                /* 激励模式 5      */
#define P_T6            0x27                                /* 激励模式 6      */

#define INT_CNT_MAX     500                                 /* 识别未旋转：  x200 us */
#define START_PWM_FF_MAX 300                                /* 无旋转期间的界限值：  10%    */

#define DELTA_CW        0                                   /* CW 端的超前角            */
#define DELTA_CCW       0                                   /* CCW 端的超前角           */

#define CHANGE_SPEED    100                                 /* 从 120 度激励到 180 度激励的切换速度 */
#define CHANGE_TIME     2500

#define CLEAR           0
#define SET             1

#define CR01_ADD        3                                   /* 切换等待时间计数器值 */
                                                                /* 依赖于 TM00 设置     */

#define PWM_BASE_REF    1000                                /* PWM 周期的一半      */
#define PWM_F_REF       0                                   /* 默认 PWM 波形值     */
#define PWM_DTM_REF     200                                 /* 死区时间            */

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

#define INTERVAL_BASE   5                                   /* 1ms = 200[us] * INTERVAL_BASE       */
#define TIME_OUT        5000                                /* 1s  = 200[us] * TIME_OUT            */
                                                                /* 依赖于变频定时器设置                */

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

#define INTP0           PM00                                /* 过流检测端口      */
#define INVERTER_SW     P54                                 /* 变频操作控制端口 */
#define INVERTER_SW_MODE PM54                               /* 变频操作控制端口 */


/* ------------------------------------------------ */
static const unsigned int sin_tbl[91] = {  /*  sin(theta) x (2^16 - 1) */
    0,  1143,  2287,  3429,  4571,  5711,  6850,  7986,
    9120, 10251, 11380, 12504, 13625, 14742, 15854, 16961,
```

```
    18063, 19160, 20251, 21336, 22414, 23485, 24549, 25606,
    26655, 27696, 28728, 29752, 30766, 31771, 32767, 33753,
    34728, 35692, 36646, 37589, 38520, 39439, 40347, 41242,
    42125, 42994, 43851, 44694, 45524, 46340, 47141, 47929,
    48701, 49459, 50202, 50930, 51642, 52338, 53018, 53683,
    54330, 54962, 55576, 56174, 56754, 57318, 57863, 58392,
    58902, 59394, 59869, 60325, 60762, 61182, 61582, 61964,
    62327, 62671, 62996, 63301, 63588, 63855, 64102, 64330,
    64539, 64728, 64897, 65046, 65175, 65285, 65375, 65445,
    65495, 65525, 65535
};


const unsigned char tr_sw[2][8] = {     /* 激励模式       */
#ifdef LOW
    /*
      PITTMAN N2311A011 12VDC  driveLayer U：棕色，V：红色，W：橙色
                        霍尔传感器 U：灰色，V：蓝色，  W：白色
                                霍尔-1    霍尔-2    霍尔-3
     */
    {P_OFF, P_T4, P_T6, P_T5, P_T2, P_T3, P_T1, P_OFF},   /* CW             */
    {P_OFF, P_T1, P_T3, P_T2, P_T5, P_T6, P_T4, P_OFF}    /* CCW            */
#else
    /*
      OrientalMOTOR FBLM575W-A driveLayer U：紫色，V：蓝色，W：灰色
                        霍尔传感器 U：米色，V：橙色，W：粉红色
                                霍尔-1    霍尔-2    霍尔-3
     */
    {P_OFF, P_T3, P_T5, P_T4, P_T1, P_T2, P_T6, P_OFF},   /* CW        */
    {P_OFF, P_T6, P_T2, P_T1, P_T4, P_T5, P_T3, P_OFF}    /* CCW       */
#endif
};

const unsigned char next_type[2][8] = {  /* 下一个 霍尔传感器 值 */
    {8, 3, 6, 2, 5, 1, 4, 8},
    {8, 5, 3, 1, 6, 4, 2, 8}
};

unsigned int       int_cnt;                                  /* 载波同步中断的个数 */
unsigned int       pid_cnt;
unsigned int       start_cnt;
unsigned int       print_cnt;
unsigned int       cnt_data;
unsigned int       cnt_data1;
unsigned int       cnt_data2;
unsigned int       cnt_data3;

int               m_speed   = 0;
int               speed_ref = 0;                             /* 指定的速度                   */
int               speed_ref_o = 0;
float              I_ref;
float              I_ref_o;
float              I_measured;

char               stop_wait;                                /* 停止等待标志                   */
char               cw_ccw_wait;                             /* 方向等待标志                 */

char               cw_ccw_flag;                             /* 旋转方向状态           */
char               sys_flag;                                 /* 系统操作状态             */
char               err_flag = FLG_OFF;                      /* 错误标志                     */
static char        start_flag;                              /* 旋转开始的标识     */
static char        speed_flag;                              /* 速度信息存在 / 不存在 */
static char        霍尔_new;
static char        霍尔_old;

static char        clk_flag;
static char        capture_flag;
static unsigned int new_clk, old_clk;                  /* 用于速度测量的定时器值     */

int               sin_table_end = 90;
static unsigned int angle;                                 /* 角度信息                   */
int               pwm_ff;                                   /* 有效宽度：  0 到 500             */
int               pwm_ff_o;                                 /* 有效宽度：  0 到 500             */
int               old_ff;                                   /* 有效宽度：  0 到 500             */
int               pwm_base = PWM_BASE_REF;       /*  PWM 周期的一半：  固定为 1000     */

unsigned char      startup_method = STARTUP_METHOD_REF;
unsigned char      maxed_flags;                        /* 如果达到界限，指定的位：  第 7-6-5 位分别为 RPM-电流-电压 */

int                maxspeed = MAXSPEED_REF;
int                minspeed = MINSPEED_REF;

/* 设定值最大变化率 */
float              RPM_rate_max = RPM_RATE_MAX_REF;   /* RPM/sec */
float              I_rate_max   = I_RATE_MAX_REF;              /* dI_int/sec 最大值 */

static int         dpwm_ff_max  = PWM_BASE_REF /10;    /* 每个控制周期内 pwm 的最大变化 */
int               dspeed_ref_max;                                /* 每个控制周期内速度的最大变化 */
float              dI_ref_max;

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

/* 反馈到 I 命令的 RPM */
float              krp_ref = KRP_DEF_REF;              /* dI_ints/dRPM 误差    */
float              kri_ref = KRI_DEF_REF;               /* dI_ints/RPM 误差     *
float              krd_ref = KRD_DEF_REF;              /* dI_ints/d(dRPM 误差) */
```

```
/* 反馈到 PWM 命令的电流 */
float               kip_ref = KIP_DEF_REF;                  /* dPWM/dI_error  */
float               kii_ref = KII_DEF_REF;                  /* dPWM/I_error   */
float               kid_ref = KID_DEF_REF;                  /* dPWM/d(derror) */
float               adgain = ADGAIN_REF;
int                adoffset = ADOFFSET_REF;
float               maxcurrent = MAXCURRENT_REF;
float               mincurrent = MINCURRENT_REF;


/*
    静态变量
*/
static unsigned int  cy_time = 150 * INTERVAL_BASE;
static char          cmd_mode = SPEED_CMD;
static char          up_flag;                                /* 速度更新通知标志 */
static unsigned int  pwm_ff_u, pwm_ff_v, pwm_ff_w;
static float         mvn;                                    /* 修改的变量          */
static float         en, en_1, en_2;                         /* 偏差                */

/* ------------------------------------------------- */
static void init_PORT(void);
static void init_OSC(void);
static void init_TW0(void);
static void init_TM00(void);
static void init_RTPM01(void);
static void init_AD(void);
static void init_TM51(void);
static void init_WDTM(void);

static void start_AD(void);
static void start_TW0(void);
static void start_TM00(void);
static void start_TM51(void);
static void start_RTPM01(void);

static void stop_TW0(void);
static void stop_TM00(void);
static void stop_RTPM01(void);

static void INTP0_on(void);
static void INTP5_on(void);
static void INTTM51_on(void);
static void INTTW0UD_on(void);

static void INTP5_off(void);
static void INTTW0UD_off(void);

static void set_TW0(unsigned int, unsigned int, unsigned int, unsigned int);
static void set_RTPM01(unsigned char);

static void wait(unsigned int);

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

static char read_霍尔_IC(void);
static void filters(void);


/* ======================================================== */
/* ---------------------------------------------------------
    对用户公开的电动机控制函数段
   --------------------------------------------------------- */

/* ---------------------------------------------------------
   电动机控制相关的函数的初始设置
   --------------------------------------------------------- */
void
motor_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TW0();
    init_TM00();
    init_RTPM01();
    init_AD();
    init_TM51();

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

    start_TM51();
    INTTM51_on();

    start_AD();                                             /* 启动 A/D 转换        */
    INTP0_on();                                             /* 取消屏蔽过流中断 */
    EI();                                                   /* 允许中断            */
}

/* ---------------------------------
    系统启动
   --------------------------------- */
void
motor_start(void)
{
    en_1        =
    en          =
    mvn         = 0.0;
    int_cnt     =
    pid_cnt     =
    print_cnt   =
    start_cnt   = 0;
    cnt_data1   =
    cnt_data2   =
```

```
    cnt_data3    = 500;
    cnt_data     = 1500;
    new_clk      =
    old_clk      =
    m_speed      = 0;
    sys_flag     = FLG_120;
    start_flag   = FLG_ON;
    stop_wait    = FLG_OFF;
    cw_ccw_wait  = FLG_OFF;
    speed_flag   = FLG_OFF;
    clk_flag     = FLG_OFF;
    up_flag      = FLG_OFF;
    capture_flag = FLG_OFF;
    err_flag     = ERROR_NONE;
    pwm_ff       = (int)PWM2;
    I_ref        = I_measured;
    speed_ref    = (int)RPM2;
    霍尔_old     = read_霍尔_IC();

    start_TM00();
    INTTW0UD_on();                                  /* 取消屏蔽载波同步中断 */
    INTP5_on();                                     /* 取消屏蔽 霍尔传感器 中断              */
    start_RTPM01();
    start_TW0();

    INVERTER_SW = INV_ON;                           /* INVERTER 允许  */
}

/* ---------------------------------
    电动机停止处理
   --------------------------------- */
void
motor_stop(void) {
    stop_wait = FLG_ON;
}


/* ------------------------------------------------
    电动机旋转方向改变处理
   ------------------------------------------------ */
void
motor_rotation(char dir) {
    switch(dir) {
    case CW:
        if(cw_ccw_flag == CCW) {
            cw_ccw_wait = FLG_ON;
            cw_ccw_flag = CW;
        };
        break;
    default:
        if(cw_ccw_flag == CW) {
            cw_ccw_wait = FLG_ON;
            cw_ccw_flag = CCW;
        };
    };
}

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

    if(speed_flag == FLG_ON){                       /* 速度已经被测量 */
        speed_flag = FLG_OFF;
        if(start_flag == FLG_OFF){
            if(clk_flag == FLG_ON){
                clk_flag = FLG_OFF;
                up_flag = FLG_ON;                   /* 速度已经被更新 */
                angle = cnt_data1;
                DI();
                new_tmp = new_clk;
                old_tmp = old_clk;
                EI();
                if(old_tmp > new_tmp){
                    tmp = (unsigned long)((65536 - (long)old_tmp) + (long)new_tmp);
                }else{
                    tmp = new_tmp - old_tmp;
                }
                m_speed = (int)(UNIT_RPM/tmp);
            };
        }else{
            start_flag = FLG_OFF;                   /* 第一个速度信息未被使用 */
        }
    }
    if(pid_cnt >= cy_time){                         /* cy_time (ms) 已经过去 */
        pid_cnt = 0;
        if(up_flag == FLG_ON){
            up_flag = FLG_OFF;
            if((sys_flag    != FLG_OFF) &&          /* 被启动                */
               (stop_wait   != FLG_ON) &&           /* 不等待停止            */
               (cw_ccw_wait != FLG_ON)){            /* 不等待方向旋转的停止 */
                maxed_flags = 0;                    /* 在这里复位所有标志  */
                switch(cmd_mode) {
                case V_CMD:                         /* 电压控制：  由 GUI 设置的 pwm  */
                    /* 速度上没有闭环 */
                    speed_ref = m_speed;
                    /* 电流上没有闭环 */
                    I_ref = I_measured;
                    /* 调整设置值 (加速度限制) */
                    if((pwm_ff_o - pwm_ff) > dpwm_ff_max) {
                        pwm_ff += dpwm_ff_max;
                        maxed_flags |= 0x20;        /* 达到界限，设置第 5 位 */
```

```
                } else if((pwm_ff_o - pwm_ff) < -dpwm_ff_max) {
                    pwm_ff -= dpwm_ff_max;
                    maxed_flags |= 0x20;                    /* 达到界限，设置第 5 位 */
                } else {
                    pwm_ff = pwm_ff_o;
                };
                break;

            case I_CMD:                                     /* 电流控制 (未使用) */
            case SPEED_CMD:                                 /* 速度控制          */
                /* 电流上没有闭环 */
                I_ref = I_measured;
                /* 调整设置值 (加速度限制)*/
                if((speed_ref_o - speed_ref) > dspeed_ref_max) {
                    speed_ref += dspeed_ref_max;
                    maxed_flags |= 0x80;                    /* 达到界限，设置第 7 位 */
                } else if((speed_ref_o - speed_ref) < -dspeed_ref_max) {
                    speed_ref -= dspeed_ref_max;
                    maxed_flags |= 0x80;                    /* 达到界限，设置第 7 位 */
                } else {
                    speed_ref = speed_ref_o;
                };
                en_2 = en_1;                                /* 上上个误差 */
                en_1 = en;                                  /* 上个误差 */
                en   = (float)(speed_ref - m_speed);     /* 最近的误差 */

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

                pwm_tmp = (int)(mvn);

                if(pwm_tmp > dpwm_ff_max) {
                    pwm_tmp += dpwm_ff_max;
                    maxed_flags |= 0x20;                    /* 达到界限，设置第 5 位 */
                } else if(pwm_tmp < -dpwm_ff_max) {
                    pwm_tmp -= dpwm_ff_max;
                    maxed_flags |= 0x20;                    /* 达到界限，设置第 5 位 */
                };

                pwm_ff += pwm_tmp;

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

/* --------------------------------
   电动机参数设置
   -------------------------------- */
char
motor_pset(unsigned char com, long data) {
    char status = 1;

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

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

    return(status);
}


/* ============================================================ */
/* ------------------------------------------------------------
    未对用户公开的电动机控制函数
   ------------------------------------------------------------ */
/*
    从反向旋转停止重启
*/
void
system_restart(void)
{
    init_TW0();

    en_1        =
    en          =
    mvn         = 0.0;
    int_cnt     =
    pid_cnt     =
    print_cnt   =
    start_cnt   = 0;
    cnt_data1   =
    cnt_data2   =
    cnt_data3   = 500;
    cnt_data    = 1500;
    new_clk     =
    old_clk     =
    m_speed     = 0;
    sys_flag    = FLG_120;
    start_flag  = FLG_ON;
    cw_ccw_wait  = FLG_OFF;
    speed_flag  = FLG_OFF;
    clk_flag    = FLG_OFF;
    up_flag     = FLG_OFF;
    capture_flag = FLG_OFF;
    pwm_ff      = (int)PWM2;
    I_ref       = I_measured;
    speed_ref   = (int)RPM2;

    霍尔_old    = read_霍尔_IC();
}

/*
    系统停止
*/
void
system_stop(void)
{
    pwm_ff = 0;
    INVERTER_SW = INV_OFF;                  /* INVERTER 禁止                */

    INTTW0UD_off();                         /* 停止同步载波中断 */
    INTP5_off();
    stop_RTPM01();                          /* 停止实时端口                 */
    stop_TW0();                             /* 停止变频器定时器             */
```

```
    stop_TM00();                                    /* 停止 16 位定时器                */

    init_TW0();

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

/*
    端口设置
*/
static void
init_PORT(void)
{
    INTP0 = IN;                                     /* 过流中断通知 */

    INVERTER_SW_MODE = OUT;                         /* INVERTER 允许 / 禁止            */
    INVERTER_SW      = INV_OFF;                     /* INVERTER 禁止                   */

    EGP5 = SET;                                     /* 上升沿有效                      */
    EGN5 = CLEAR;
}


/*
    时钟切换
*/
static void
init_OSC(void)
{
    IMS = IMS_DATA;                                 /* 切换存储器大小                  */
    WDTE = WDTE_CLR;
    while(OSTC != 0x1f);                            /* 等待振荡稳定 */
    PCC   = 0x00;                                   /* 设置分频比率                    */
    MCM.0 = 1;                                      /* X1 输入时钟                     */
    VSWC.1 = 1;
}

/*
变频器定时器
*/
static void
init_TW0(void)
{
    TCL02    = 0;                                   /* 计数时钟： 20 MHz               */
    TCL01    = 0;
    TCL00    = 0;
    IDEV02   = 0;                                   /* 每两次产生一个中断 */
    IDEV01   = 0;
    IDEV00   = 1;
    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;                         /* 死区时间                        */
    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;
}

static void
start_TW0(void)
{
    TW0C.7 = SET;                                   /* 启动定时器  */
}

static void
stop_TW0(void)
{
    TW0C.7 = CLEAR;                                 /* 停止定时器   */
}

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

/*
    16 位定时器

    CR01    实时输出端口触发
*/
static void
init_TM00(void)
{
    ES000  = 0;
    ES001  = 0;                                     /* TI000 引脚的有效沿： 下降沿     */
    CRC000 = 1;                                     /* CR00 捕获寄存器                 */
      CRC001 = 1;                                   /* TI000 引脚的有效沿的反向相位时捕获 */
    CRC002 = 0;                                     /* CR01 比较寄存器                 */
      PRM001 = SET;
      PRM000 = CLEAR;                               /* 计数时钟： 78.125 kHz */
}

static void
start_TM00(void)
{
    TMIF00 = CLEAR;
    TMC00  = 0x04;                                  /* 自由运行模式       */
}
```

```
static void
stop_TM00(void)
{
    TMC00 = CLEAR;                          /* 停止定时器          */
}


/*
    实时端口
*/
static void
init_RTPM01(void)
{
    RTPM01  = 0x3f;                         /* 实时输出模式    */
    RTPC01  = 0x20;                         /* 6 位 x 1 通道       */
    DCCTL01 = 0xc0;                         /* PWM 调制输出    */
    RTBL01  = 0x3f;                         /* 输出缓存          */
}

void
start_RTPM01(void)
{
    DCCTL01.7 = SET;                        /* 实时输出        */
    RTPC01.7  = SET;                        /* 允许操作        */
}

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

void
end_RTPM01(void)
{
    DCCTL01.7 = CLEAR;                      /* 输出变频器定时器   */
}

void
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                          /* 设置激励模式  */
    CR01   = TM00 + CR01_ADD;
}


/*
    AD
*/
static void
init_AD(void)
{
    ADCS2 = SET;                            /* 允许电路操作 */
    ADS   = AD_ISHUNT;                      /* SHUNT */
    ADM   = 0x04;                           /* 3.6us */
}

static void
start_AD(void)
{
    ADIF = CLEAR;                           /* 清除中断通知标志 */
    ADCS = SET;                             /* 允许转换操作       */
    while(ADIF != SET);                     /* 等待中断通知   */
}


/*
看门狗定时器
*/
static void
init_WDTM(void)
{
    WDTE = WDTE_CLR;
    WDTM = WDTM_SET;
}


/* --------------------------------
    8 位定时器 51
    -------------------------------- */
static void
init_TM51(void) {
    TMC51 = 0x00;                           /* 无 PWM，TIMER 输出禁止 */
    TCL51 = 0x05;                           /* 19.53KHz */
    CR51  = 28;                             /* 1.43 ms */
}

static void
start_TM51(void) {
    TMIF51 = CLEAR;                         /* 清除中断通知标志 */
    TCE51  = SET;                           /* 启动定时器                  */
}


/*
    读取 霍尔传感器
*/
char
read_霍尔_IC(void)
{
    return((char)((P0>>1)&0x7));
}


/* ================================================== */
/*
    中断允许 / 禁止处理
 */
/*
    过流中断允许
*/
void
```

```
INTP0_on(void)
{
    PR0L.1 = 0;                                          /* 优先级     */
    EGN.0  = 1;                                          /* 下降沿 */
    IF0L.1 = CLEAR;                                     /* 清除标志  */
    MK0L.1 = CLEAR;                                     /* 取消屏蔽      */
}

/*
    载波同步中断允许
*/
static void
INTTW0UD_on(void)
{
    IF0H.1 = CLEAR;                                      /* 清除请求标志 */
    MK0H.1 = CLEAR;                                     /* 允许中断   */
}

/*
    载波同步中断禁止
*/
static void
INTTW0UD_off(void)
{
    MK0H.1 = SET;                                        /* 禁止中断 */
}

/* ---------------------------------
    INTP5 中断允许
   --------------------------------- */
static void
INTP5_on(void) {
    PIF5 = CLEAR;
    PMK5 = CLEAR;
}

/* ---------------------------------
    INTP5 中断禁止
   --------------------------------- */
static void
INTP5_off(void) {
    PMK5 = SET;
}

/* ---------------------------------
    电流过小中断允许
   --------------------------------- */
static void
INTTM51_on(void) {
    IF1H.0 = CLEAR;
    MK1H.0 = CLEAR;
}


/* ======================================================== */
/*
    中断服务程序
*/
/* ------------------------------------------------ */
__interrupt void
int_fault(void)
{
    err_flag = ERROR_OC;
    system_stop();                                       /* 停止系统 */
}


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

/*
    载波同步中断
*/
static unsigned int
get_coswt(unsigned int wt)
{
    unsigned int   n;
    unsigned int   answer;
    unsigned char  dir, pm;


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

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

```
    if(dir == 0) { MDA0L = sin_tbl[wt]; }
    else         { MDA0L = sin_tbl[sin_table_end - wt]; };

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

    if(pm == 0) { answer = PWM_MIDDLE + MDA0H;}
    else        { answer = PWM_MIDDLE - MDA0H;};

    return(answer);
}

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

#ifdef LOW
    if(cw_ccw_flag == CW) {
        p = ((60 * int_cnt)/angle) + DELTA_CW;          /* 角度细节：超前角固定 */
        switch(霍尔_new) {
        case 4: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

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

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

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

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

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

        default:
            p_u = 90;
            p_v = 90;
        };
    }else{
        p = ((60 * int_cnt)/angle) + DELTA_CCW;       /* 角度细节：超前角固定 */
        switch(霍尔_new){
        case 1: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

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

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

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

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

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

        default:
            p_u = 90;
            p_v = 90;
        };
    };
#else
    if(cw_ccw_flag == CW) {
        p = ((60 * int_cnt)/angle) + DELTA_CW;          /* 角度细节：超前角固定 */
        switch(霍尔_new) {
        case 5: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

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

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

        case 2: /* 180 - 239 */
```

```
            p_u = p + 180;
            p_v = p_u + 240;
            break;

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

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

        default:
            p_u = 90;
            p_v = 90;
        };
    }else{
        p = ((60 * int_cnt)/angle) + DELTA_CCW;        /* 角度细节：超前角固定 */
        switch(霍尔_new){
        case 3: /* 0 - 59 */
            p_u = p;
            p_v = p_u + 240;
            break;

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

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

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

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

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

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

    WDTE = WDTE_CLR;
}

__interrupt void
int_carrier(void)
{
    unsigned int     tmp_pwm;

    int_cnt++;
    print_cnt++;
    if(int_cnt > INT_CNT_MAX){                          /* 溢出                    */
        if(stop_wait != FLG_OFF){                       /* 等待停止                */
            system_stop();                               /* 停止系统                */
            return;
        }else if(cw_ccw_wait != FLG_OFF){               /* 等待反向旋转的停止 */
            system_restart();                            /* 重新启动                */
            return;
```

```
        }else if(pwm_ff > START_PWM_FF_MAX){    /* 电动机未旋转              */
            system_stop();                      /* 停止系统                  */
            err_flag = (ERROR_霍尔 | ERROR_MOTOR);
            return;
        }else{
            pwm_ff++;                           /* 增加电压                  */
            int_cnt = 0;
            start_cnt = 0;
            MDB0 = (unsigned int)pwm_ff;
        }
    }
    pid_cnt++;
    start_cnt++;
    霍尔_new = read_霍尔_IC();
    if(霍尔_new == next_type[cw_ccw_flag][霍尔_old]){  /* 霍尔传感器 值为下一个值 */
        霍尔_old  = 霍尔_new;
        cnt_data1 = int_cnt;
        int_cnt   = 0;
        speed_flag = FLG_ON;
    }
    if(sys_flag == FLG_120){                    /* 120 度励磁方法 */
        pid_cnt = 0;                            /* 不执行 PID 控制 */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            set_RTPM01(P_OFF);                  /* 停止激励           */
            sys_flag = FLG_WAIT;
        }else if((m_speed != 0) && (start_cnt > CHANGE_TIME)){
            set_RTPM01(P_OFF);                  /* 停止激励           */
            sys_flag = FLG_SW;
        }else{
            set_RTPM01(tr_sw[cw_ccw_flag][霍尔_new]);  /* 设置激励模式     */
            if(pwm_ff != old_ff) {
                set_TW0((unsigned int)pwm_ff,
                        (unsigned int)pwm_ff,
                        (unsigned int)pwm_ff,
                        (unsigned int)pwm_base);    /* 更改占空比             */
                old_ff = pwm_ff;
            }
        }
    }else if(sys_flag == FLG_180){              /* 180 度励磁方法 */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            start_RTPM01();
            set_RTPM01(P_OFF);                  /* 停止激励           */
            sys_flag = FLG_WAIT;
        }else{
            set_pwm();
            set_TW0(pwm_ff_u,
                    pwm_ff_v,
                    pwm_ff_w,
                    (unsigned int)(pwm_base));
        }
    }else if(sys_flag == FLG_SW){               /* 从 120 度切换为 180 度后立即执行 */
        if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
            sys_flag = FLG_WAIT;
        }else{
            end_RTPM01();                       /* 结束实时端口输出 */
            tmp_pwm = (unsigned int)(pwm_base >> 1);
            set_TW0(tmp_pwm, tmp_pwm, tmp_pwm, pwm_base);
            sys_flag = FLG_180;
            pwm_ff = (pwm_ff/3)*2;              /* 调整应用电压为 2/3 */
            MDB0 = (unsigned int)pwm_ff;
        }
    };

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

    return;
}


/* ---------------------------------
    TM51 溢出的中断
  --------------------------------- */
__interrupt void
int_TM51(void) {
    unsigned int x_filt, reg;

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

4.18 参考程序源文件

```
/*


   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标　：　uPD78F0714 电动机入门套件


   日期　：　2007/05/21
   文件名　：　main_mcio.c

   NEC Micro Systems,Ltd
*/

#pragma sfr

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

/*
   主函数
*/
void
main(void) {
   unsigned char  sw = 0;
   unsigned char  tmp_sw = 0;

   /* 初始系统设置 */
   motor_init();
   init_PORT();
   init_TM50();

   /* 电动机参数设置 */
   motor_pset(PID_INTERVAL, 150);                  /* PID 控制间隔：  150 ms */

   motor_pset(KRP,  50);                            /* RPM-Kp 设置。设置乘以 1000 的值          */
   motor_pset(KRI,  20);                            /* RPM-Ki 设置。设置乘以 1000 的值          */
   motor_pset(KRD,  10);                            /* RPM-Kd 设置。设置乘以 1000 的值          */

   startup_disp();
   while(1){
      clear_WDTM();
      vol2speed();
      speed_print(200);
      sw = get_sw();
      if(sys_flag != FLG_OFF){
         if((cw_ccw_wait == FLG_OFF) &&
            (stop_wait   == FLG_OFF)) {
            if(sw != tmp_sw) {
               switch(sw){
               case STOP_TR:    motor_stop();       break;
               case FORWARD_TR: motor_rotation(CW);  break;
               case REVERSE_TR: motor_rotation(CCW); break;
               default: ;
               };
            };
         };
         motor_pid();
      } else {                                          /* 被停止    */
         if(sw != tmp_sw) {
            switch(sw){
            case START_TR: motor_start(); break;
            case MODE_TR:
               if(ad_flag == FLG_ON){                  /* 被停止的函数  */
                  ad_flag = FLG_OFF;                   /* 恢复函数    */
               }else{                                  /* 被操作的函数  */
                  ad_flag = FLG_ON;                    /* 停止函数    */
               };
               break;
            default: ;
            };
         };
      };
      tmp_sw = sw;
   };
   return;
}
```

```
/*


   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标　：　uPD78F0714 电动机入门套件


   日期　：　2007/05/22
   文件名　：　sub_mcio.h

   NEC Micro Systems,Ltd
*/

#define  CLEAR      0
#define  SET        1

#define  IN         1    /* 输入 */
#define  OUT        0    /* 输出 */

#define  FLG_ON     1    /* 有效 */
#define  FLG_OFF    0    /* 无效 */
```

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

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

#define  LD_DATA     PM4

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

#define  START_TR    0x01                    /* 识别按下的开关 */
#define  STOP_TR     0x02
#define  FORWARD_TR  0x04
#define  REVERSE_TR  0x08
#define  MODE_TR     0x10

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

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

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

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

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

```
/*

   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标　：　uPD78F0714 电动机入门套件


   日期　：　2007/05/22
   文件名　：　sub_mcio.c

   NEC Micro Systems,Ltd
*/

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

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


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

/*
   静态变量
*/
static const unsigned char led_data[10] = {        /* 用于显示数字值   */
   LED_0, LED_1, LED_2, LED_3, LED_4,
   LED_5, LED_6, LED_7, LED_8, LED_9
};

static void          led_set(unsigned char, unsigned char);
```

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

unsigned char       ad_flag;

/* ============================================================ */
/*
错误显示
*/
static void
print_error(char eno) {
    switch(eno){
    case ERROR_霍尔:
        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();
}


/*
   读取 MC-IO 中的开关
*/
unsigned char
get_sw(void)
{
    unsigned char data;

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

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

    data = SW;                                          /* 读取开关                    */
    if(data != 0xf){
        for(i=0;i<KEY_WAIT;){                           /* 消除抖动。必须被调整。              */
            tmp = SW;                                   /* 重新读取开关                  */
            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){                                    /* 无开关被按下 */
        start_stop_flag = FLG_OFF;                      /* 放置 START 和 STOP 之间的拴牢 */
    };

    return(data);
}


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


/*
    LED 在 7 段 LED 上显示数值
*/
static void
led_print(int data, char flag)
{
    unsigned int tmp;
    int flg = FLG_OFF;

    tmp = (unsigned int)(data / 1000);
    if(tmp != 0){                                       /* 最高位不是 0 */
        flg = FLG_ON;                                   /* 数字值显示在最高位上 */
        led_set(0, led_data[tmp]);
    }else{
        led_set(0, LED_);
    };
    data %= 1000;
    tmp = (unsigned int)(data / 100);
    if((tmp != 0) || (flg == FLG_ON)){                  /* 值不是 0 或者数字已经被显示 */
        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)){                  /* 值不是 0 或者数字已经被显示 */
        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);
    };
}

/*
    在 7 段 LED 上显示数据
    no：　要显示的 LED 的位置
    data：　要输出的数据
*/
static void
led_set(unsigned char no, unsigned char data)
{
    unsigned char p;

    P6 = 0x00;
    P4 = data;

    switch(no){                                         /* 要显示的位置 */
    case 0:  p = 0x80; break;                           /* 左边缘           */
    case 1:  p = 0x40; break;
    case 2:  p = 0x20; break;
    default: p = 0x10; break;                           /* 右边缘           */
    };
    P6 = p;
}

/*
    时间调整　ms
*/
static void
wait(int cnt)
{
    int i;

    for(i=0;i<cnt;i++){
        start_TM50();                                   /* 启动定时器                   */
        wait_TM50();                                    /* 等待中断请求产生 */
        clear_WDTM();                                   /* 清除看门狗定时器             */
    };
    return;
}

/*
  启动显示
*/
void

startup_disp(void) {
```

```
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(2000);
}


/*
    读取速度指定变量电阻的电压
    转换为指定的速度
*/
#define SHIFT_BIT   2
void
vol2speed(void)
{
    unsigned int  data;
    unsigned long tmp;

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

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

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

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


/* ------------------------------------------------------- */
/*
    端口设置
*/
void
init_PORT(void) {
    SW2 = IN;                                       /* START/STOP           */
    SW3 = IN;                                       /* FORWARD              */
    SW4 = IN;                                       /* REVERSE              */
    SW5 = IN;                                       /* MODE                 */

    LD_LED0 = OUT;                                  /* LED 选择： 输出  */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;                                  /* LED 显示数据： 输出 */
}

/*
    8 位定时器 50
*/
void
init_TM50(void)
{
    TCL50 = 0x06;
    CR50  = 78;                                     /* 1ms */
}

static void
start_TM50(void)
{
    TMIF50 = CLEAR;                                 /* 清除中断通知标志 */
    TCE50  = SET;                                   /* 启动定时器                   */
}

static void
wait_TM50(void)
{
    while(TMIF50 != SET);                           /* 等待中断通知   */
    TCE50  = CLEAR;                                 /* 通知定时器                   */
}

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

附录A　程序示例

包含当通过连接单独提供的电动机操作面板（GUI 程序）和位于电动机控制 I/O 板上的 RS-232C 引脚来控制该系统时的程序示例。

A.1　GUI 参考程序函数列表

表 A-1. GUI 参考程序函数

函数名	功能	用途
uart_set()	UART 设置	设置 UART 功能。
get_uart()	命令读取指令	指示读取 UART 通信中通信数据并返回对应于读取数据的操作信息。
uart_read()	读取 UART 数据	指示 UART 接收的数据的读取。
uart_wait()	UART 数据的读取（与接收等待函数一并提供）	用于 UART 通信中多字节配置的命令的连续接收的函数
uart_send()	UART 数据发送	指示 UART 通信中的发送。
int_uart()	UART 接收	UART 通信中执行接收的中断函数
set_error()	存储错误状态	存储 UART 通信数据中的错误状态。
led_set()	LED 显示	用 LED 显示数据。
startup_disp()	启动时的 LED 显示	启动时的 LED 显示
init_PORT()	端口设置	执行 MC-IO 板上的端口设置。

A.2 GUI 参考程序常量列表

A.2.1 内部常量

表 A-2. GUI 参考程序内部常量（1 / 2）

名称	含义	设置值	备注
IN	用于端口设置	1	用来指定端口功能
OUT	用于端口设置	0	
CLEAR	用于寄存器位设置	0	用来访问寄存器的位
SET	用于寄存器位设置	1	
LED_0	LED 显示数据	0xc0	显示“0”。
LED_1		0cf9	显示“1”。
LED_2		0xa4	显示“2”。
LED_3		0xb0	显示“3”。
LED_4		0x99	显示“4”。
LED_5		0x92	显示“5”。
LED_6		0x82	显示“6”。
LED_7		0xf8	显示“7”。
LED_8		0x80	显示“8”。
LED_9		0x98	显示“9”。
LED_O		0xc0	显示“0”而不是“O”。
LED_I		0xcf	显示“I”。
LED_C		0xc6	显示“C”。
LED_H		0x89	显示“H”。
LED_A		0x88	显示“A”。
LED_L		0xc7	显示“L”。
LED_		0xff	显示“ ”。
LED_S		0x92	显示“S”。
LED_E		0x86	显示“E”。
LED_F		0x8e	显示“F”。
LED_P		0x8c	显示“P”。
LED_dot		0x7f	显示“.”。
LD_LED0	端口控制寄存器	PM64	用于 LED 选择的端口
LD_LED1		PM65	向 LED 输出数据的端口
LD_LED2		PM66	
LD_LED3		PM67	
LD_DATA		PM4	
START_TR	用于状态设置	0x01	开始控制。
STOP_TR		0x02	停止控制。
FORWARD_TR		0x04	更改旋转为 CW。
REVERSE_TR		0x08	更改旋转为 CCW。
MODE_TR		0x10	MODE 开关被按下的状态
RTS	通信端口	P11	
CTS		P10	
RX_BUFF_SIZE	通信缓存大小	6	
MD_ERROR_霍尔	通信信息	0xF0	

表 A-2. GUI 参考程序内部常量（2 / 2）

名称	含义	设置值	备注
MD_ERROR_OC	通信信息	0xF1	
MD_ERROR_MOTOR		0xF2	
MD_CMD_START	通信信息	0x20	
MD_CMD_STOP		0x21	
MD_CMD_RESET		0x2f	
MD_CMD_GETID		0x10	
MD_CMD_GETVER		0x11	
MD_CMD_SETSSPEED		0x30	
MD_CMD_GETSSPEED		0x31	
MD_CMD_SETPIDPARAM		0x40	
MD_CMD_GETPIDPARAM		0x41	
MD_CMD_GETPIDI		0x42	
MD_CMD_GETPIDV		0x43	
MD_CMD_SETPIDI		0x44	
MD_CMD_SETPIDV		0x45	
MD_CMD_GETSPEED		0x50	
MD_CMD_GET_I_SHNT		0x60	
MD_CMD_GET_PWM		0x61	
MD_CMD_GET_I_CMD		0x62	
MD_CMD_SETICMD		0x70	
MD_CMD_SETVCMD		0x71	
MD_CMD_SETSTART		0x72	
MD_CMD_GETSTART		0x73	
MD_CMD_SETLIMIT		0x74	
MD_CMD_GETLIMIT		0x75	
MY_ID		0x02	
VER_MAJOR		0x01	
VER_MINOR		0x00	
VER_SEQ		0x01	

A.3 GUI 参考程序变量列表

A.3.1 内部变量

表 A-3. GUI 参考程序内部变量

ad_flag	char	速度更改标志	限制指定的速度更改函数。
led_data[ ]	char	LED 输出数据	用 LED 显示的值

A.4 GUI 参考程序源程序

```
/*

   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标   ： uPD78F0714 电动机入门套件
          编译器选项 GUI：使用 GUI
   日期   ： 2007/05/22
   文件名 ： main_gui.c

   NEC Micro Systems,Ltd
*/

#pragma sfr

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

/*
   主函数
*/
void
main(void) {
   unsigned char  sw = 0;
   unsigned char  tmp_sw = 0;

   /* 初始系统设置 */
   motor_init();
   init_PORT();
   uart_set();
   startup_disp();

   /* 电动机参数设置 */
   motor_pset(PID_INTERVAL, 150);  /* PID 控制间隔： 150 ms */

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

```
/*

   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标   ： uPD78F0714 电动机入门套件
          编译器选项 GUI： 使用 GUI

   日期   ： 2007/05/22
   文件名 ： sub_gui.h

   NEC Micro Systems,Ltd
*/
```

```
#define IN              1          /* 输入 */
#define OUT             0          /* 输出 */
```

```
#define CLEAR          0
#define SET            1

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

#define FLG_ON         1                       /* 有效  */
#define FLG_OFF        0                       /* 无效  */

#define MD_ERROR_HALL      0xF0            /* 霍尔传感器 故障       */
#define MD_ERROR_OC        0xF1             /* 过流            */
#define MD_ERROR_MOTOR     0xF2             /* 电动机故障        */

#define RTS            P11
#define CTS            P10
#define RX_BUFF_SIZE       6                   /* UART00 接收缓存大小    */

#define MD_CMD_START       0x20            /* START            */
#define MD_CMD_STOP        0x21            /* STOP            */
#define MD_CMD_RESET       0x2f            /* RESET            */
#define MD_CMD_GETID       0x10            /* ID 请求            */
#define MD_CMD_GETVER      0x11            /* 版本请求           */
#define MD_CMD_SETSSPEED   0x30            /* 指定的速度改变        */
#define MD_CMD_GETSSPEED   0x31            /* 指定的速度读取        */
#define MD_CMD_SETPIDPARAM  0x40           /* PID 改变           */
#define MD_CMD_GETPIDPARAM  0x41           /* PID 读取           */
#define MD_CMD_GETPIDI     0x42
#define MD_CMD_GETPIDV     0x43
#define MD_CMD_SETPIDI     0x44
#define MD_CMD_SETPIDV     0x45
#define MD_CMD_GETSPEED    0x50            /* 实际速度读取          */

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

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

#define MY_ID              0x02                /* 固件标识 ID            */
#define VER_MAJOR          0x01                /* 版本信息              */
#define VER_MINOR          0x00
#define VER_SEQ            0x01

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

#define LD_DATA            PM4

#define LED_0              0xc0                /* LED 显示数据                  */
#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                /* 霍尔                    */
#define LED_A              0x88
#define LED_L              0xc7
#define LED_               0xff
#define LED_S              0x92                /* SELF                    */
#define LED_E              0x86
#define LED_F              0x8e
#define LED_P              0x8c                /* PC                    */
#define LED_dot            0x7f


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

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

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


/*


   PMSM 带位置传感器（霍尔传感器）的 180 度励磁方法

   改进的速度估计精度版本（定时器值捕获方法）
   支持模块转换

   目标  ： uPD78F0714 电动机入门套件
         编译器选项 GUI： 使用 GUI
*/
```

```
    日期  ：  2007/05/22
    文件名 ：  sub_gui.c

    NEC Micro Systems,Ltd
*/

#pragma sfr


#pragma INTERRUPT INTSR00       int_uart  rb1    /* 用于 UART00 命令接收        */


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


/*
    静态变量
*/
static const unsigned char led_data[10] = {         /* 用于显示数字值                */
    LED_0, LED_1, LED_2, LED_3, LED_4,
    LED_5, LED_6, LED_7, LED_8, LED_9
};

static unsigned char uart00_data[RX_BUFF_SIZE];     /* UART00 接收缓存               */
static unsigned char read_p, write_p;               /* 缓存指针                      */
static char          uart_err_flag;

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

#define WDTE_RESET      0x00

/* ============================================================== */
/* -------------------------------------------------------------- */
/*
    UART00 设置
*/
void
uart_set(void) {
    PM10    = IN;
    PM11    = OUT;
    PM13    = IN;
    PM14    = OUT;
    RTS     = 1;
    P14     = 1;
    BRGC00  = 0x56;                                 /* 115200 */
    PS001   = CLEAR;
    PS000   = CLEAR;
    CL00    = SET;                                  /* 8 位   */
    SL00    = CLEAR;
    POWER00 = SET;
    TXE00   = SET;
    RXE00   = SET;
    STIF00  = SET;
    SRIF00  = CLEAR;
    SRMK00  = CLEAR;                                /* 允许接收中断 */
    RTS     = 0;
    read_p  = 0;
    write_p = 0;
}

/*
    通过 UART 通信获取操作指令
 */
unsigned char
get_uart(void) {
    unsigned char data;
    int           ss, ref;
    unsigned char hi, lo;
    float         shunt_volt;
    float         shunt_command;
    float         tmp_float;
    int           tmp_int;

    /* 方便的宏 */
#define GET16to(thevar)             \
        lo = uart_wait();           \
        hi = uart_wait();           \
        thevar = (((int)(hi))<<8) + lo;

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

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

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

    case MD_CMD_GETVER:                             /* 获取版本 */
        uart_send(data);
        uart_send(VER_MAJOR);
        uart_send(VER_MINOR);
        uart_send(VER_SEQ);
        break;

    case MD_CMD_START:                              /* 启动 */
```

```
        uart_send(data);
        data = START_TR;
        break;

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

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

    case MD_CMD_SETSSPEED:                      /* 设置设定的速度 */
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){              /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CW;
            }
        }
        speed_ref_o = ss;
        motor_pset(MODE, SPEED_CMD);            /* 速度控制模式 */
        break;

    case MD_CMD_SETICMD:
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){                          /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CW;
            }
        }
        I_ref_o = (float)ss;
        speed_ref = m_speed;                    /* 速度上的开环  */
        motor_pset(MODE, I_CMD);                /* 电流控制模式 */
                break;

    case MD_CMD_SETVCMD:
#if 1
        lo = uart_wait();
        hi = uart_wait();
        ss = ((int)hi<<8) + lo;
#else
        GET16to(ss);
#endif
        uart_send(data);
        if(hi > 0x80){              /* CCW */
            ss = ~ss + 1;
            data = REVERSE_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CCW;
            }
        }else{
            data = FORWARD_TR;
            if(sys_flag == FLG_OFF){
                cw_ccw_flag = CW;
            }
        }
        pwm_ff_o = ss;
        I_ref = I_measured;
        speed_ref = m_speed;                    /* 速度上的开环 */
        motor_pset(MODE, V_CMD);                /* 电压控制模式 */
                break;

    case MD_CMD_GETSSPEED:                      /* 获取设定的速度 */
        uart_send(data);
        ss = speed_ref;
        if(cw_ccw_flag == CCW){
            ss = ~ss + 1;
        }
#if 1
        lo = (unsigned char)((unsigned int)(ss)&0xff);
        hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
        uart_send(lo);
        uart_send(hi);
#else
        SETUART(ss);
#endif
```

```
        break;

    case MD_CMD_GET_I_SHNT:                         /* 获取分流 a/d (电流) */
        shunt_volt = I_measured*10.0;               /* 10 位 */
        uart_send(data);
#if 1
        ss = (int)shunt_volt;
        lo = (unsigned char)(ss&0xff);
        hi = (unsigned char)(ss>>8)&0xff;
        uart_send(lo);
        uart_send(hi);
#else
        SETUART(shunt_volt);
#endif
        break;

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

    case MD_CMD_GET_PWM:                            /* 获取 PWM 值 */
        uart_send(data);
        ss = pwm_ff;
        if (ss > 1000) ss=1000;                     /* 标志空间的溢出预防 */
        if (ss < 0) ss=0;                           /* 标志空间的溢出预防 */
        lo = (unsigned char)((unsigned int)(ss)&0xff);
        hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
        /*
                    注：这是一个 10 位变量，所以高位被用来
                        发送 maxed_flags 变量。
        */
        hi |= maxed_flags;                          /* 使用第 7,6,5 位作为标志  */
        uart_send(lo);
        uart_send(hi);
        break;

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

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

    case MD_CMD_GETPIDI:                            /* 电流控制的 PID  */
        uart_send(data);
        uart_send((char)(((int)(kip_ref*1000))&0xff));
        uart_send((char)((((int)(kip_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kii_ref*1000))&0xff));
        uart_send((char)((((int)(kii_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kid_ref*1000))&0xff));
        uart_send((char)((((int)(kid_ref*1000))>>8)&0xff));
        break;

    case MD_CMD_GETPIDV:                            /* 电流控制的 PID  */
        uart_send(data);
        uart_send((char)(((int)(krp_ref*1000))&0xff));
        uart_send((char)((((int)(krp_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(kri_ref*1000))&0xff));
        uart_send((char)((((int)(kri_ref*1000))>>8)&0xff));
        uart_send((char)(((int)(krd_ref*1000))&0xff));
        uart_send((char)((((int)(krd_ref*1000))>>8)&0xff));
        break;

    case MD_CMD_GETSPEED:                           /* 读取转速 */
        if(uart_err_flag != ERROR_NONE){
            uart_send(uart_err_flag);
            err_flag = ERROR_NONE;
            uart_err_flag = ERROR_NONE;
        }else{
            uart_send(data);
            if(m_speed == 0){
                uart_send(0);
                uart_send(0);
            }else{
                ss = m_speed;
                if(cw_ccw_wait == FLG_OFF){
                    if(cw_ccw_flag == CCW){
                        ss = ~ss + 1;
                    }
                }else{                                  /* 等待反向旋转的停止 */
                    if(cw_ccw_flag == CW){
                        ss = ~ss + 1;
                    }
                }
```

```
#if 1
            uart_send((unsigned char)((unsigned int)(ss)&0xff));
            uart_send((unsigned char)(((unsigned int)(ss)>>8)&0xff));
#else
            SETUART(ss);
#endif
          }
        }
        break;

    case MD_CMD_SETSTART:                       /* gui 定义的开环启动参数 */
        GET16to(ref);
        startup_method = (unsigned char)(ref);
        GET16to(ref);
        t_knee = (float)ref/1000.0;
        GET16to(ref);
        t_end = (float)ref/1000.0;
        GET16to(RPM0);
        GET16to(RPM1);
        GET16to(RPM2);
        GET16to(I_ref0);
        GET16to(I_ref1);
        GET16to(I_ref2);
        GET16to(PWM0);
        GET16to(PWM1);
        GET16to(PWM2);
        uart_send(data);
        break;

    case MD_CMD_GETSTART:                       /* 发送启动参数到 gui */
        uart_send(data);
        uart_send(startup_method);
        uart_send(0x00);                        /* 在 gui 中，为方便起见 char 作为 int 发送 */
        SETUART(t_knee*1000);
        SETUART(t_end*1000);
        SETUART(RPM0);
        SETUART(RPM1);
        SETUART(RPM2);
        SETUART(I_ref0);
        SETUART(I_ref1);
        SETUART(I_ref2);
        SETUART(PWM0);
        SETUART(PWM1);
        SETUART(PWM2);
        break;

    case MD_CMD_SETLIMIT:                       /* A/D 增益和速率限制 */
        GET16to(ref);
        adgain = (float)ref/100.0;
        GET16to(adoffset);
        GET16to(ref);
        maxcurrent = (float)ref;
        GET16to(ref);
        mincurrent = (float)ref;
        GET16to(ref);
        I_rate_max = (float)ref/10.;
        GET16to(maxspeed);
        GET16to(minspeed);
        GET16to(ref);
        RPM_rate_max = (float)ref;

        dI_ref_max = I_rate_max*0.1;            /* 0.1 秒钟的 dt，这是每个周期的最大变化 */
        dspeed_ref_max = (int)(RPM_rate_max*0.1); /* 0.1 秒钟的 dt，这是每个周期的最大变化 */
        uart_send(data);
        break;

    case MD_CMD_GETLIMIT:                       /* A/D 增益和速率限制 */
        uart_send(data);
        SETUART(adgain*100);
        SETUART(adoffset);
        SETUART(maxcurrent);
        SETUART(mincurrent);
        SETUART(I_rate_max*10.);
        SETUART(maxspeed);
        SETUART(minspeed);
        SETUART(RPM_rate_max);
        break;

    default:
        ;
    };

    return(data);
}

/*
    返回 UART00 接收缓存数据
*/
static unsigned char
uart_read(void) {
    unsigned char data = 0;

    if(read_p != write_p){                      /* 接收数据存在 */
        data = uart00_data[read_p++];           /* 提取数据     */
        read_p %= 6;                            /* 更新读取指针 */
    }
    return(data);
}

/*
    等待 UART00 接收
*/
static unsigned char
uart_wait(void) {
    unsigned char data;

    while(read_p == write_p);                   /* 等待直到数据输入到缓存 */
```

```
    data = uart00_data[read_p++];                    /* 提取数据                */
    read_p %= 6;                                     /* 更新读取指针            */
    return(data);
}

/*
    从 UART00 发送
*/
static void
uart_send(unsigned char data) {
    while(STIF00 != SET);                            /* 等待发送完成   */
    STIF00 = CLEAR;
    TXS00 = data;                                    /* 发送                    */
}

/*
    用于 UART00 命令接收
*/
__interrupt void
int_uart(void) {
    unsigned char tmp;

    tmp = ASIS00;                                    /* 读取错误状态            */
    uart00_data[write_p++] = RXB00;                  /* 在接收缓存中存储状态    */
    write_p %= RX_BUFF_SIZE;                         /* 更新写指针              */
}


/* ---------------------------------------------------------- */
/*
错误显示
*/
static void
set_error(char eno) {

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

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

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

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

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

/*
    在 8 段 LED 上显示数据
    no：   要显示的 LED 的位置
    data：要输出的数据
*/
static void
led_set(unsigned char no, unsigned char data) {
    unsigned char p;

    P6 = 0x00;
    P4 = data;

    switch(no){                                      /* 要显示的位置 */
    case 0:  p = 0x80; break;                        /* 左边缘               */
    case 1:  p = 0x40; break;
    case 2:  p = 0x20; break;
    default: p = 0x10; break;                        /* 右边缘               */
    };
    P6 = p;
}

/*
  启动显示
*/
void
startup_disp(void) {
    led_set(0, LED_P);
    led_set(1, LED_C);
    led_set(2, LED_);
    led_set(3, LED_);
```

```
}

/*
看门狗定时器
*/
void
clear_WDTM(void) {
    WDTE = WDTE_CLR;
}

/* ------------------------------------------------------ */
/*
    端口设置
*/
void
init_PORT(void) {
    LD_LED0 = OUT;                                  /* LED 选择： 输出 */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;                                   /* LED 显示数据： 输出 */
}
```

详细信息请联系：

中国区

MCU 技术支持热线：
电话：+86-400-700-0606 (普通话)
服务时间：9:00-12:00，13:00-17:00 （不含法定节假日）

网址：
http://www.cn.necel.com/ （中文）
http://www.necel.com/ （英文）

[北京]

日电电子（中国）有限公司
中国北京市海淀区知春路 27 号
量子芯座 7，8，9，15 层
电话：（+86）10-8235-1155
传真：（+86）10-8235-7679

[上海]

日电电子（中国）有限公司上海分公司
中国上海市浦东新区银城中路 200 号
中银大厦 2409-2412 和 2509-2510 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

上海恩益禧电子国际贸易有限公司
中国上海市浦东新区银城中路 200 号
中银大厦 2511-2512 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

[长春]

日电电子（中国）有限公司长春分公司
吉林省长春市朝阳区
西安大路 727 号中银大厦 A 座 1609 室
电话：(+86)431-8859-7533 / 8859-8533
传真：(+86)431-8680-2944

[深圳]

日电电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼
3901，3902，3909 室
电话：（+86）755-8282-9800
传真：（+86）755-8282-9899

[香港]

香港日电电子有限公司
香港九龙旺角太子道西 193 号新世纪广场
第 2 座 16 楼 1601-1613 室
电话：（+852）2886-9318
传真：（+852）2886-9022
2886-9044

[成都]

日电电子（中国）有限公司成都分公司
四川省成都市二环路南三段 15 号
天华大厦 608 室
电话：(+86)28-8512-5224
传真：(+86)28-8512-5334

[大连]

日电电子（中国）有限公司长春分公司
大连市中山路 88 号天安国际大厦 2701 室
电话：(+86)411-8230-8815 / 8230-8825
传真：(+86)411-8230-8835