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 实现电机控制

无传感器 (BEMF 型 A/D 转换) 120° 励磁方式

μPD78F0714

文档编号. U18912CA1V0AN00 (第一版)
发行日期 2008 年 12 月 N

[备注]

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 2008 年 12 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。

引言

目标读者

本手册适用于那些希望理解μPD78F0714, 并设计开发相关应用系统和程序的用户工程师 下面显示的应用产品

- μPD78F0714

目的

本手册用于帮助用户了解怎样通过无传感器(BEMF 型 A/D 转换) 120° 励磁方式来控制电机，本文所描述的范例使用了μPD78F0714 的定时/计数功能，在系统中使用了 PWM 输出和比较器。

组织

应用笔记分为下面几个部分

- 概要信息
- BLDCM 控制原理
- 系统概述
- 程序控制

怎样阅读本手册

在阅读本手册前，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。

如何了解硬件的详细内容(尤其是寄存器功能,设置方法等) 和电气特性

→请参阅 ***μ*PD78F0714 用户手册 (U16928E)**.

如何获悉系列指令的详细信息

→请参阅 **78K/0 系列指令用户手册(U12326E)**.

规定

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：	$\overline{\times\times\times}$ (在引脚和信号名称上加划一条线)
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 ... ×××× 或 ××××B
	十进制 ... ××××
	十六进制 ... ××××H

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 **(千兆):**	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
通过零点检测 120° 励磁方式控制 μPD78F0714 实现变频控制应用笔记	U17297E
通过 V/f 控制使用μPD78F0714 两相正弦波变频控制实现单相感应电机应用笔记	U17481E
μPD78F0714 霍尔器件 120° 励磁方式实现电机控制应用笔记	U18774E
μPD78F0714 无传感器 (BEMF) 120° 励磁方式实现电机控制应用笔记	U18051E
μPD78F0714 霍尔器件 180° 励磁方式实现电机控制应用笔记	U18913E
μPD78F0714 无传感器 (BEMF 型 A/D 转换) 120° 励磁方式实现电机控制应用笔记	本手册

开发软件工具相关文档 (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＋ V5.20		U16934E

开发硬件工具相关文档 (用户手册)

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

Flash存储器编程的相关文档

文档名称	文档编号
PG-FP4 Flash 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。读者开发设计时，应该使用每个文档的最新版本。

其他相关文档

文档名称	文档编号
半导体选型指南 – 产品与封装-	X13769X
半导体设备焊装手册	**Note**
NEC 半导体设备的质量等级	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。读者开发设计时，应该使用每个文档的最新版本。

目录

第 1 章 概要信息

该系统使用 120° 励磁来激励一个三相星形连接无刷直流电动机（下文叫做“BLDCM”）。

- 该系统使用日电电子电动机入门套件（μPD78F0714）[注] 并使用 120° 励磁来激励一个 BLDCM，而没有传感器。

  控制增益根据以下指定的工作环境中的电动机（启动时无负载）而调整。电动机或控制程序被更改的操作不被保证。

  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（示例程序）是基于在以下环境中被使用的假设下产生的 。

- 电动机入门套件（μPD78F0714）板集合
- 低压逆变器集合
  BLDCM PITTMAN （N2311A011）
  - 参考电压[V]： 12
  - 无负载转速[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）
  - 磁极转子： 4 磁极 （2 个磁极集合）
  - 定子： 6 节流
  - 位置传感器： Hall IC
- 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

1.2 相关手册

关于开发环境和板，参见以下手册。

- 低压电动机入门套件手册
- PM plus 版本 5.20 用户手册
- CC78K0 版本 3.70 C 编译器的每个用户手册
- RA78K0 版本 3.80 汇编器包的每个用户手册

第 2 章 BLDCM 控制原理

当安装永磁体的转子通过定子绕组产生的磁场的作用旋转时，BLDCM 旋转。

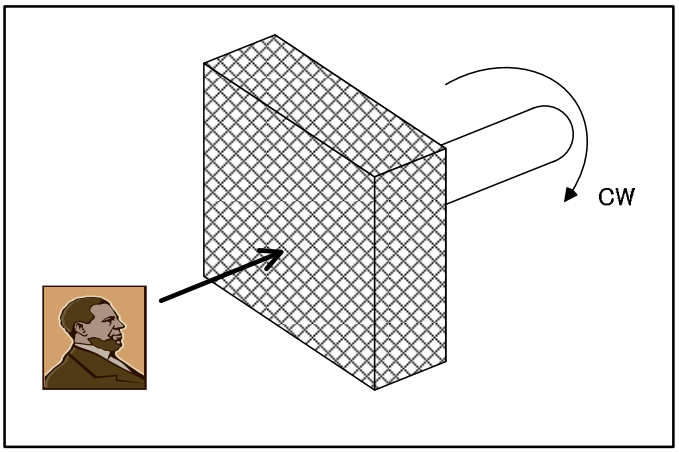
2.1 旋转方向定义

首先，电动机的旋转方向被定义。

电动机的旋转方向为 CW（顺时针）或 CCW（逆时针）。

CW 或 CCW 基于电动机要旋转的对象的旋转方向来确定。如下所示，旋转方向是基于电动机轴所在的表面面向对象时的。

图 2-1. 电动机旋转方向



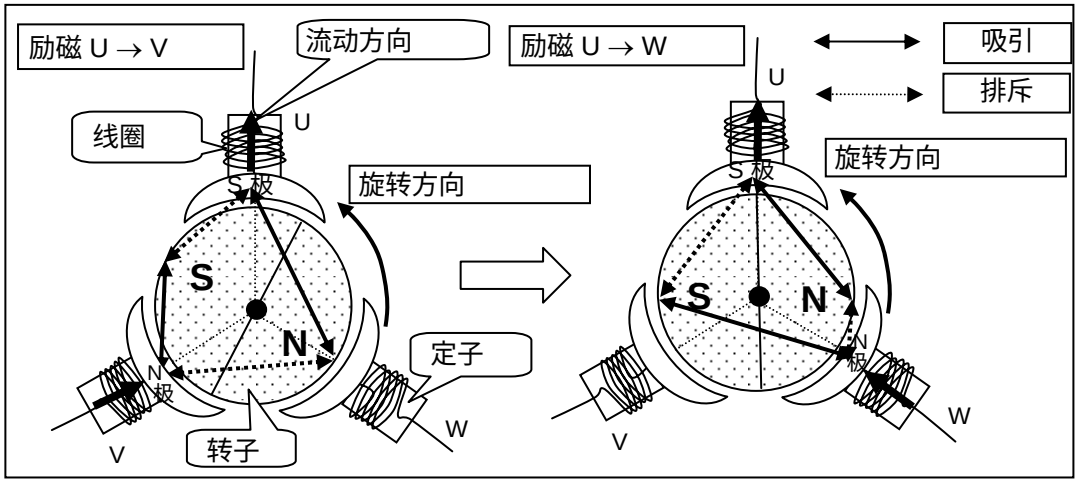
2.2 旋转原理

此部分介绍 BLDCM 旋转原理。

下图显示的 BLDCM 是一个 3 相双极 3 槽 Y 连接内转子类型 SPM（表面永磁体：永磁体放置到表面上的结构）。

通过定子磁极和转子的永磁体的磁极的吸引和排斥所产生的磁性转矩，旋转被产生。

图 2-2. BLDCM 旋转原理



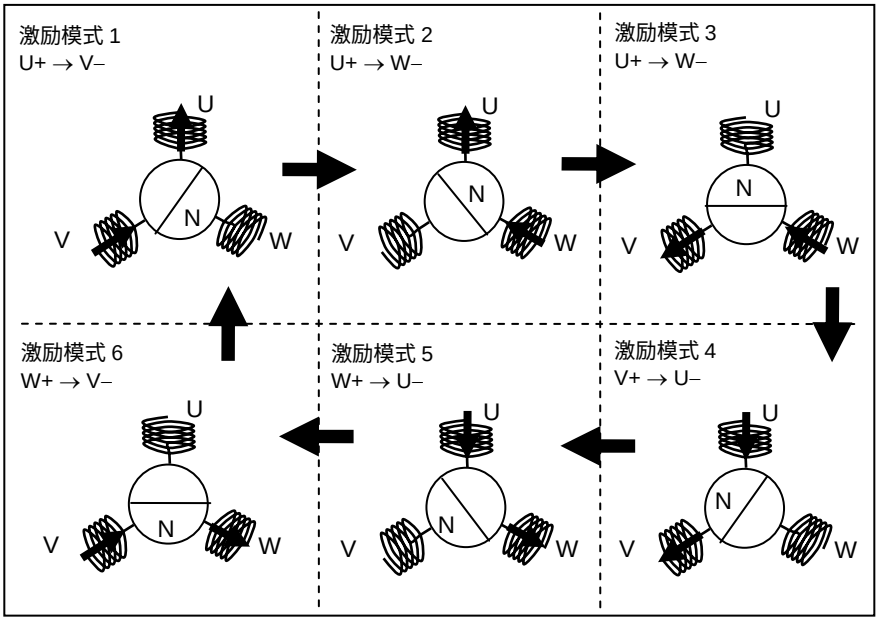
定子的极性依赖于线圈缠的方向。

当转子的磁极反转时，旋转方向反转。

2.3 激励模式

下图表示激励模式和线圈产生的电流流动方向（定子极性）和转子的磁性之间的关系。

图 2-3. 激励模式和线圈流动方向

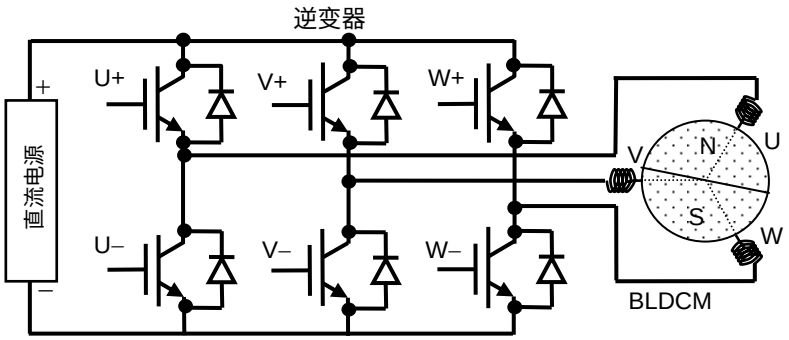


线圈的极性依赖于线圈缠的方向。

2.4 逆变器

无刷或整流器的 BLDCM 使用逆变器来改变关于线圈的电流方向。
下图表示 3 相 Y 连接 BLDCM 和逆变器的连接。

图 2-4. 逆变器和 BLDCM

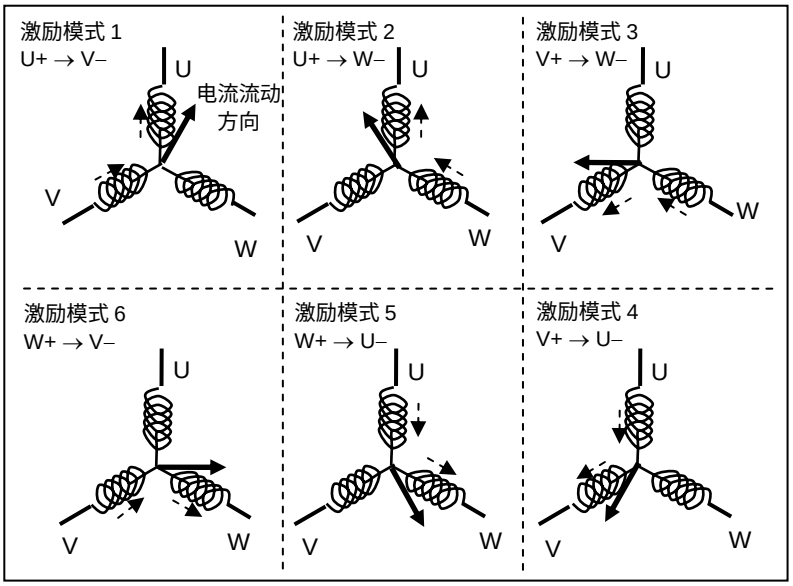


六个开关元件控制激励时间和激励方向。

2.5 激励模式

下图表示激励模式和定子线圈产生的电流流动方向。

图 2-5. 激励模式和电流流动方向

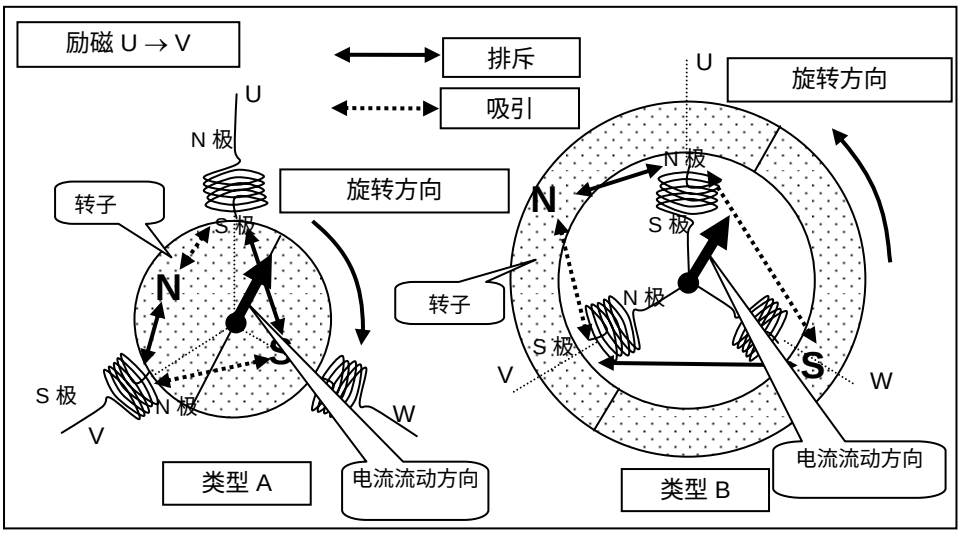


电流流动方向依赖于线圈缠的方向。

通过旋转磁场和转子的永磁体的磁极产生的线圈极性的吸引和排斥，BLDC 电动机旋转。（旋转方向根据磁极的位置而不同。）

下图表示旋转磁场和永磁体的极性的吸引和排斥，如上面的激励模式 1 所示。

图 2-6. 旋转图



旋转方向根据转子（永磁体）的位置而改变。

通常，一个周期（六个激励模式）定义为电角的 360 度，电动机轴的一个旋转定义为机械角的 360 度（本文档中的所有角都是电角，除非指定）。

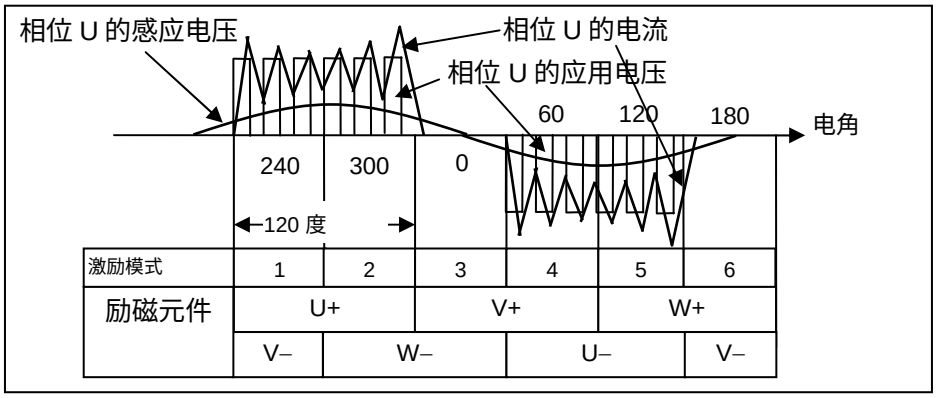
机械角： 电角／极性对的个数
极性对的个数： 转子的极性的个数/2

2.6　120° 励磁方法

下图表示当逆变器的开关元件由激励模式每 60 度切换时的应用电压和相位 U 的感应电压。

每个相位的励磁周期是 120 度（120° 励磁方法）并且 BLDC 电动机按照相位 U→ 相位 V → 相位 W 来切换上臂（+）端和下臂（–）端来激励。（反向旋转期间，相位切换的顺序被反转。）

图 2-7. 120° 激励模式和电流



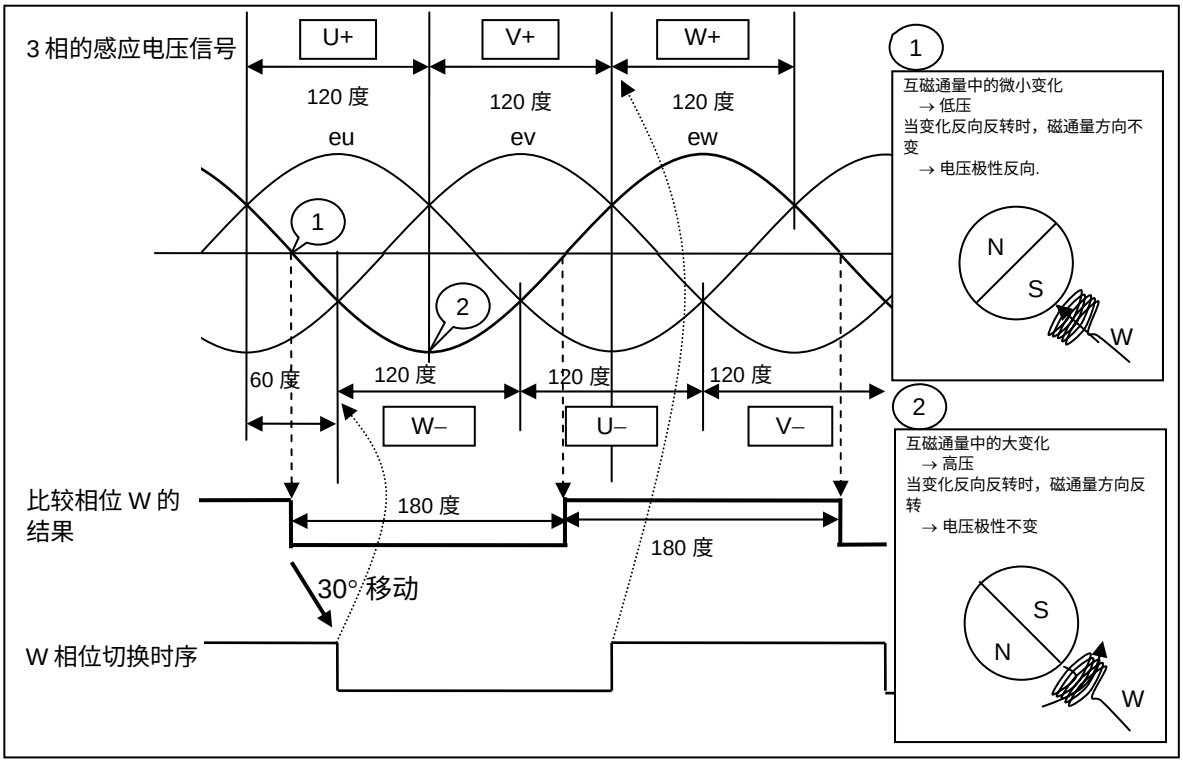
2.7 位置检测

如果 BLDCM 的三相的引脚开路并且转子从外部源旋转，正弦波中的感应电压（eu，ev 或 ew）在每个相位中被产生。感应电压与转子磁体旋转时单位时间内定子线圈中产生的互磁通量的变化成比例，因此产生的电压表明转子的旋转位置。

对于 120° 励磁方法，电流允许流过线圈的三个相位中的两个，并且激励的线圈每 60 度被切换。因此，感应电压（后 EMF）可以从没有激励的开路相位来检测。例如，从相位 U 到相位 V 激励发生的间隔内，开路的相位 W 的感应电压跨越零的点可以被检测。

下图表示估计相位位置的操作原理。

图 2-8. 估计相位位置的原理



从相位 W 的比较结果移动 30 度的信号与相位 W 的激励模式切换时序匹配。

2.8 启动方法

使用感应电压的位置检测方法在电动机停止时不能使用，此时没有感应电压产生，或者在电动机低速旋转时不能使用，此时感应电压较低。因此，电动机同步启动以使激励模式独立于位置来切换直到达到感应电压可以被检测的转数，并且当电动机转速增加时，激励方法更改为使用比较结果信号。

2.9 激励模式切换

从每个相位改变的比较结果移动 30 度时，激励模式被切换。

2.10 速度检测

电动机的转速可以使用以下方法检测。

- 通过计算激励模式切换间隔（时间）来计算电动机的转速
- 基于感应电压斜率引起的过零点的间隔来计算电动机的转速
- 从感应电压的斜率预测电动机的转速

2.11 速度控制

电动机的转速根据应用到电动机的电压来控制。

PWM（脉冲宽度调制）电压控制方法被使用。该方法通过对 120°励磁的方波进行高频率的斩波器操作来调整传导率（平均电压），斩波器操作以任意开关元件的传导周期发生。

PWM 传导率通过 PID 控制改变。

在初始设置下，控制速度的间隔是 150ms。这可以通过使用设置更改功能（motor_pset 命令的 PID_INTERVAL 参数）来改变。

2.12 PID 控制

通过使用指定的速度和检测的转速之间的偏差，PID 控制被执行用来更改 PWM 电压控制方法的传导率（下文叫做“占空比”）。

PID 控制由产生与偏差成比例输出的成比例（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导数成比例输出的导数（D）动作组成。

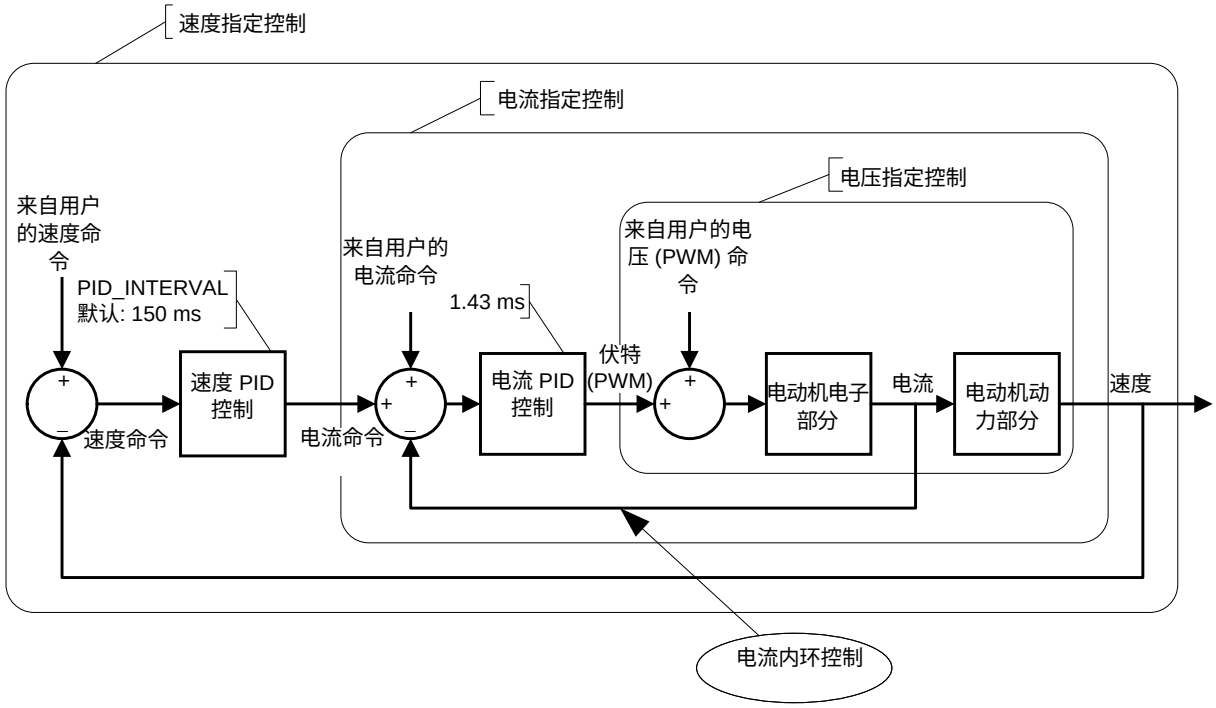
2.13 电流内环控制

用于精细调整，通过速度控制和比控制速度更短的间隔在闭环内执行的控制叫做内环控制。对电流进行的内环控制叫做电流内环控制。

PWM 占空比通过使用参考电流和检测电流之间的偏差执行的 PID 控制来更改。类似速度控制，PID 参数单独准备。

基于速度控制的修正被执行时的电流值，引入电流内环控制可以支持短控制间隔内的负载起伏。

图 2-9. 控制环


速度指定控制
电流指定控制
电压指定控制
来自用户的速度命令
PID_INTERVAL
默认: 150 ms
速度 PID 控制
速度命令
来自用户的电流命令
1.43 ms
电流命令
电流 PID 控制
伏特 (PWM)
来自用户的电压 (PWM) 命令
电动机电子部分
电流
电动机动力部分
速度
电流内环控制

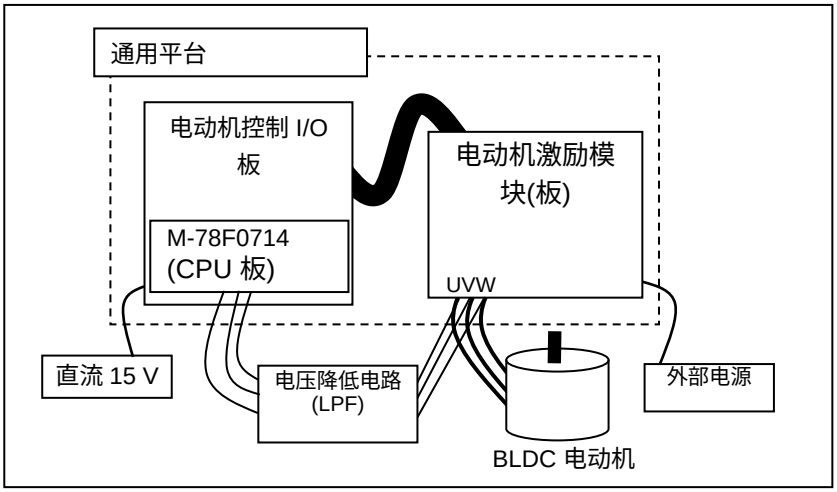
第 3 章 系统概述

本章介绍系统概述。

3.1 配置

下图表示该系统的配置。

图 3-1. 系统配置

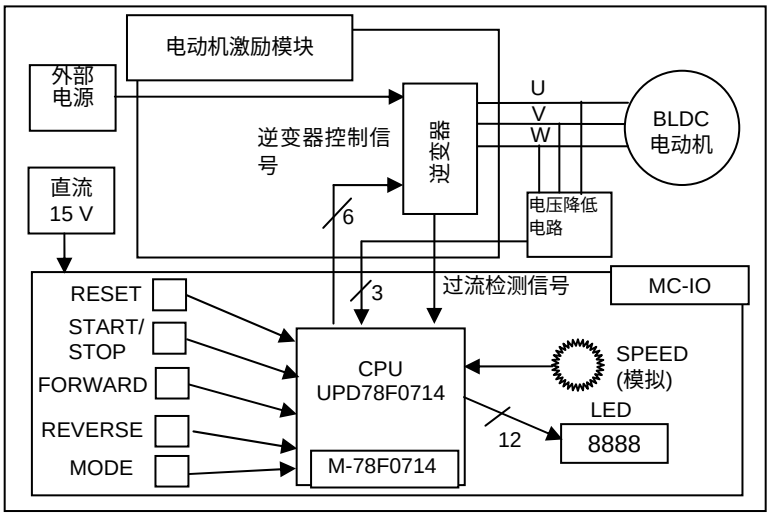


该系统由激励电动机的电动机激励模块、装备控制电动机的开关的电动机控制 I/O 板和带有 CPU 的 M-78F0714 组成。

BLDCM 具有三个相位和四个磁极（两对磁极）。

低压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）
SPEED	R52	切换速度指定 切换速度显示
8 段 LED	DISP1 DISP2 DISP3 DISP4	显示速度（rpm）[注]

注　电动机停止时，指定的速度总是显示。
指定的速度固定时，一个点（“.”）显示在右下角。
电动机旋转时，转速被显示。
电动机旋转时，MODE 开关压下期间，指定的速度被显示。
当 RESET 压下时，复位继续；当 RESET 释放时，复位释放。
复位释放后，“SELF”立即显示 2 秒钟。

表 3-2 列出了错误。

表 3-2. 错误

错误	LED 显示	情况
过流	0.C.	逆变器电流异常。
软件过流	S.0.C.	电动机电流异常。
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. 引脚的接口

引脚编号	引脚名称	功能
8	$\overline{\text{RESET}}$	RESET （SW1）注1
27 到 32	TW0TO0/RTP10 到 TW0TO5/RTP15	3 相 PWM 逆变器选择
49 到 52	P64 到 P67	8 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
21	P54/TI001/TO00	电动机激励模块控制
57，58，61	P27/ANI7，P26/ANI6，P23/ANI3	线圈电压测量注2

注　**1.** 短路 M-78F0714 板和 2JP7 的 1-2。

2. 输入每个相位的引脚电压抬高到 5 V。

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 调控器指定的速度（rpm）显示在 LED 上。
RESET 开关	无论电动机控制的状态，系统都重启。
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 调控器指定的速度（rpm）显示在 LED 上。
	即使该开关被压下，START 和 STOP 也不会切换。
FORWARD 开关	电动机控制停止期间：不起作用。
	不变。
	电动机被控制期间： 旋转方向被改变。
	如果旋转方向为 CCW，它一旦被停止后被更改为 CW。 如果旋转方向为 CW，不变。
REVERSE 开关	电动机控制停止期间： 不起作用。
	不变。
	电动机被控制期间： 旋转方向被改变。
	如果旋转方向为 CCW，不变。 如果旋转方向为 CW，它一旦被停止后被更改为 CCW。
MODE 开关	电动机控制停止期间： 速度指定被切换。
	通过 SPEED 调控器指定的速度的更改有效或禁止。 当禁止时，一个点（“.”）显示在 LED 值的右下角。
	电动机被控制期间： 更改指定的速度。
	该开关被压下期间，SPEED 调控器指定的速度（rpm）显示在 LED 上。
SPEED 调控器	电动机控制停止期间： 更改指定的速度。
	LED 上显示的速度 （rpm）改变。
	电动机被控制期间： 更改指定的速度。
	对电动机指定的速度显示在 LED 上。
硬件过流	如果超过电动机激励模块允许的电流，该功能发生。 电动机控制被停止。 “0.C.”显示在 LED 上。 RESET 开关以外的功能被禁止。
软件过流	如果超过电动机允许的电流，该功能发生。 电动机控制被停止。 “S.0.C.”显示在 LED 上。 RESET 开关以外的功能被禁止。

注意事项　如果两个或过多开关被同时压下，操作不被保证。
在不超过 300 rpm 时使用 FORWARD 或 REVERSE 开关执行反向。

表 3-4. 系统功能和操作概要（2／2）

功能	概要
系统异常	当电动机控制异常时，该功能发生。 • 在 START 开关被压下 2 秒钟内，电动机不开始旋转。 • 感应电压异常（同步启动后不切换）。 • 由于突然增加的负载，电动机停止旋转至少 1 秒钟。 • 电动机激励模块提供给电动机的电源被停止。 电动机控制被停止。 “FAIL”显示在 LED 上。 RESET 开关以外的功能被禁止。

注意事项 如果两个或过多开关被同时压下，操作不被保证。
在不超过 300 rpm 时使用 FORWARD 或 REVERSE 开关执行反向。

3.4 周边 I/O

该系统使用以下周边 I/O。

表 3-5. 周边 I/O

功能	周边 I/O 功能名称（μPD78F0714）
逆变器定时器	用于 PWM 输出 10 位逆变器控制定时器（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 100 μs 的间隔产生（占空比通过主 PID 控制动作计算，通过载波同步中断更新）。
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） （激励模式通过载波同步中断更改。）
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 ISHUNT 电流测量功能被执行。
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 1 ms 间隔被设置。
获取指定的速度	转换变量寄存器上的电压为指定的速度。 A/D 转换器 （ANI4） （由主开关读取处理来更新。）
获取 ISHUNT 的电流值	通过转换电动机工作电流来从引脚（ANI5）获取电压值。
读取引脚电压	转换相位 U、V 和 W 的电压。 通过载波同步（PWM 输出中点）中断处理。 A/D 转换器 （ANI3，ANI6，ANI7）
H/W 过流中断	中断功能（INTP0） 电动机激励模块中过流的发生（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	过流发生的检测	外部引脚
INTTW0UD	载波同步中断的发生	逆变器定时器计数器的下溢
INTTW0CM3	载波同步中断的发生	逆变器定时器计数器的下溢
INTTM51	ISHUNT 电流值的获取	8 位定时器计数器的溢出
RESET	复位发生	$\overline{\text{RESET}}$ 引脚
WDT	内部复位发生	由于程序失控，看门狗定时器溢出

备注 INTTM50 和 INTAD 按照中断请求标志的轮询来处理。

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

使用该控制程序来控制的两种逆变器可以通过使用编译时的选项来切换[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 LOW 必须被设置，因为在该控制程序中，PITTMAN 电动机被使用。

4.2 激励模式

激励模式根据每个相位的感应电压确定的转子的位置来选择。

在等效于从感应电压的过零点经过 30 度的旋转时间后，激励模式被切换。

在等效于 60 度的旋转时间内，至少需要两个（200 μs）载波同步中断，因为励磁切换处理通过载波同步中断来执行。（对于 3 相双极对，最高 25,000 rpm 的理论值可以被控制。）

方向旋转期间，对应于转子位置的励磁方向（激励模式）被反转。

该控制程序的假设连接如下所述。

4.2.1 低压逆变器设置

表 4-2. BLDCM 引脚规范

颜色	功能	备注
褐色	MOTOR ϕA	相位 U
红色	MOTOR ϕB	相位 V
橙色	MOTOR ϕC	相位 W

4.3 激励模式切换

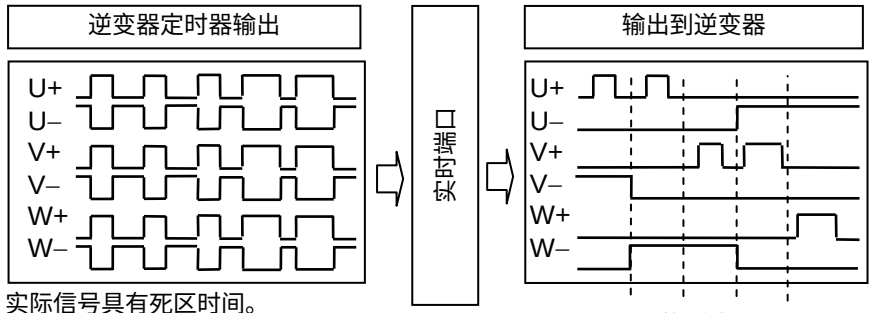
关于启动电动机时的激励模式切换，参见 **4.6 启动方法**。

通常，每隔等效于经过 30 度（30 度旋转需要的时间通过载波同步中断发生的次数来计算）的时间，激励模式被切换为下一个。

实时输出端口功能用来切换激励模式。

对于整个上臂区域的非互补操作，通过分别设置 10 位逆变器控制定时器的上臂（+）端和下臂（–）端的输出为“through”或“off”和“on”或“off”，PWM 控制被执行。

图 4-1. 激励模式切换时序



- 实际信号具有死区时间。
- PWM 占空比按照同步到载波频率的 PID 结果来更改。

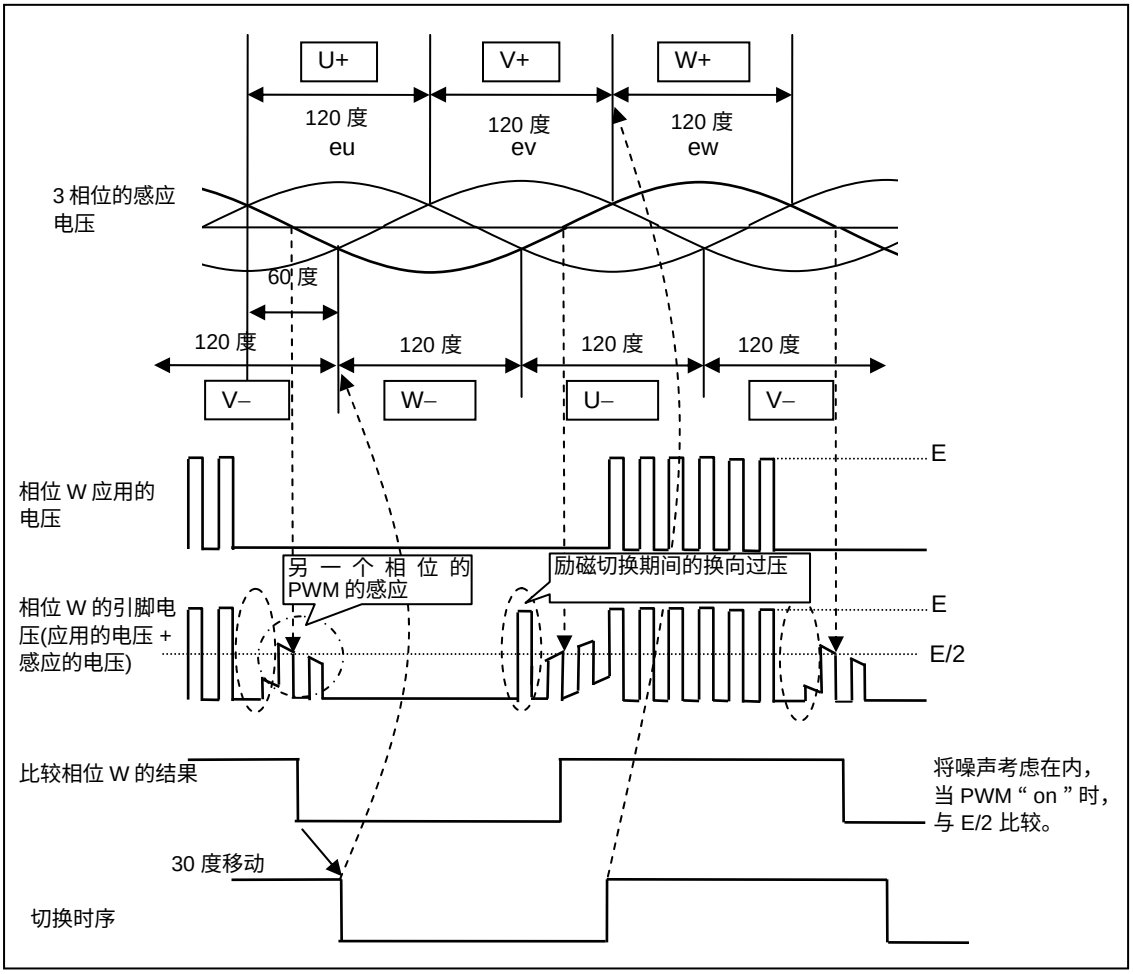
4.4 120° 励磁方法

当电动机同步启动时，激励模式使用 120°励磁方法来切换，而不考虑转子的位置，开始的速度是根据感应电压转子位置可以被预测的速度来确定的，120°励磁方法根据预测的位置来执行。

4.5 位置检测

根据相位 W 的引脚电压产生比较信号的过程如下所示。

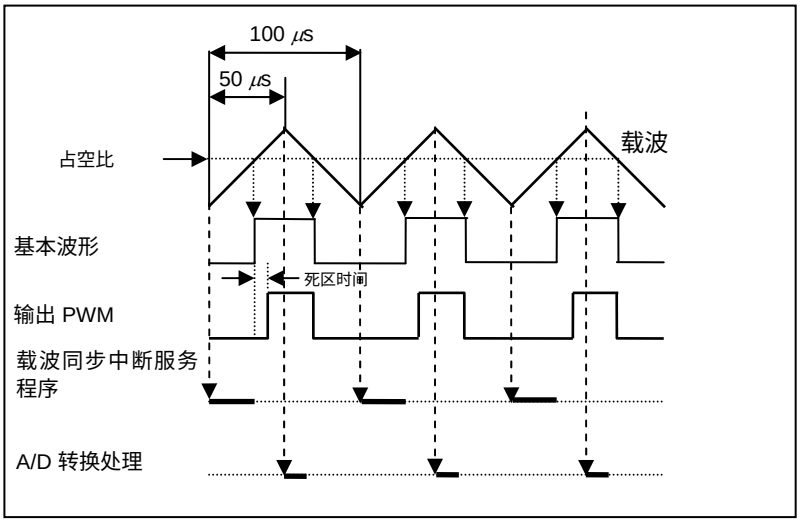
图 4-2. 切换位置检测方法



如果相位 W 的感应电压为 0，应用电压 E 为相位 U 绕组和相位 V 绕组平分的中点等于 E/2，并且未激励的相位 W 的引脚电压也是 E/2。

引脚电压的 A/D 转换时序如下所示。

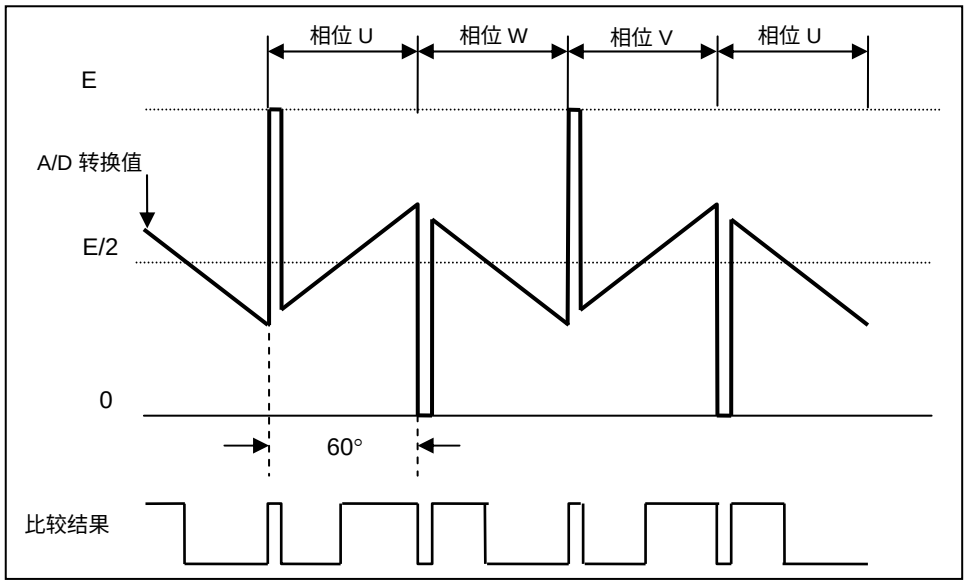
图 4-3. A/D 转换时序



通过在载波的峰值处产生的中断的服务程序，未激励的引脚被 A/D 转换，并且转换的值与 E/2 比较。死区时间用来调整 A/D 转换的时序并输出 PWM。A/D 转换时间假设为 3.6 μs。

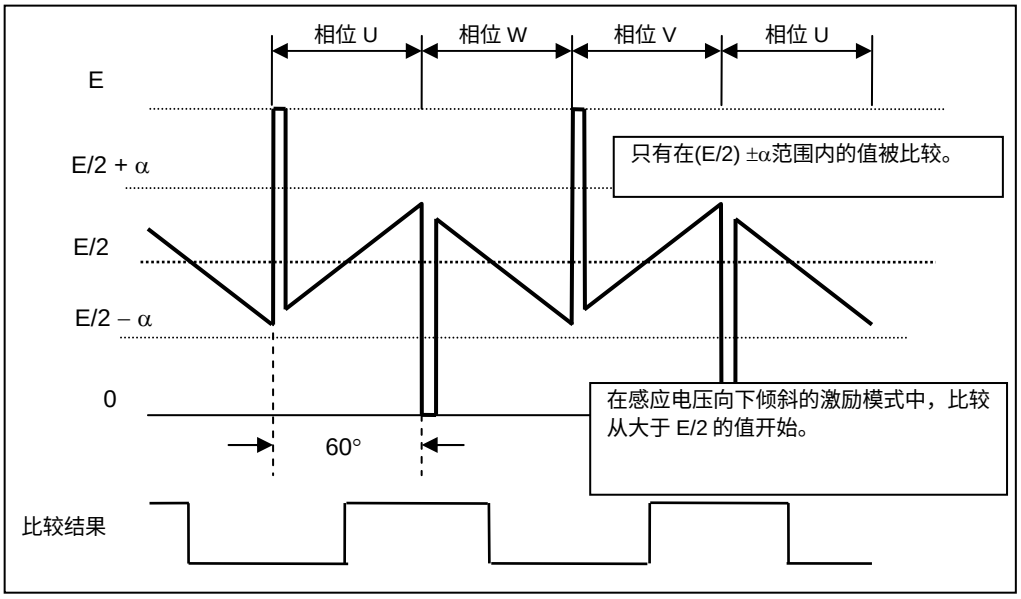
由于换向过压产生的噪声的影响，未激励引脚的电压和 E/2 的比较结果包含不需要的信号成分。

图 4-4. 未激励引脚的 A/D 转换和比较结果



通过只比较接近 E/2 的值、将从激励模式感应的电压的斜率考虑在内来调整比较开始位置或其它方法，换向过压产生的噪声可以从比较结果中移除。

图 4-5. 噪声移除举例



在该系统中，噪声通过只比较(E/2) ±α范围内的值来移除。

在该控制程序中，常量定义如下。

E/2 – α：ADDATA_300　转换值 300 （大约 30%）

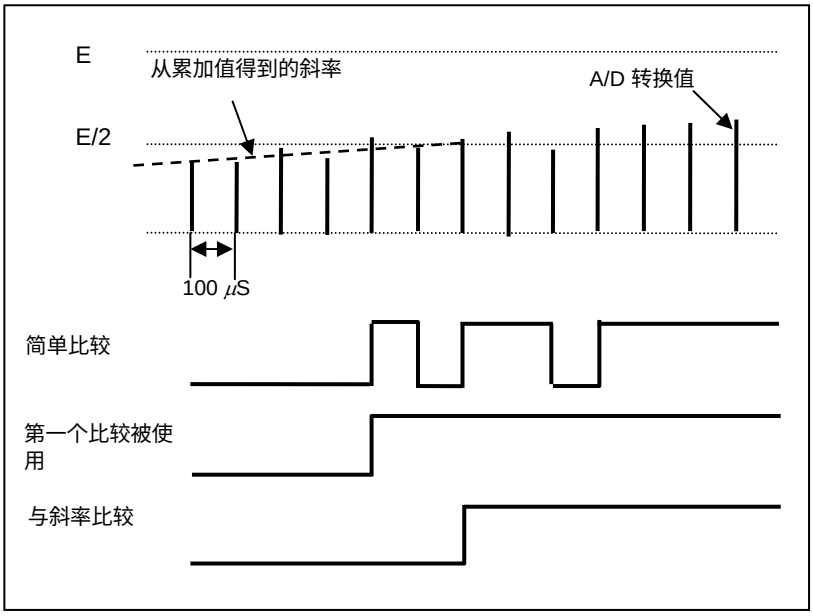
E/2 + α：ADDATA_600　转换值 600 （大约 60%）

E/2：　ADDATA_E_2　转换值 464 （大约 46%）

如果由于低转速感应电压过低，噪声从 A/D 转换误差中产生。

通过在比较结果更改后相位切换到下一个非激活相位之前不执行比较，比较从计算累加 A/D 转换值的数值得到的斜率或者其它方法，噪声可以预防。

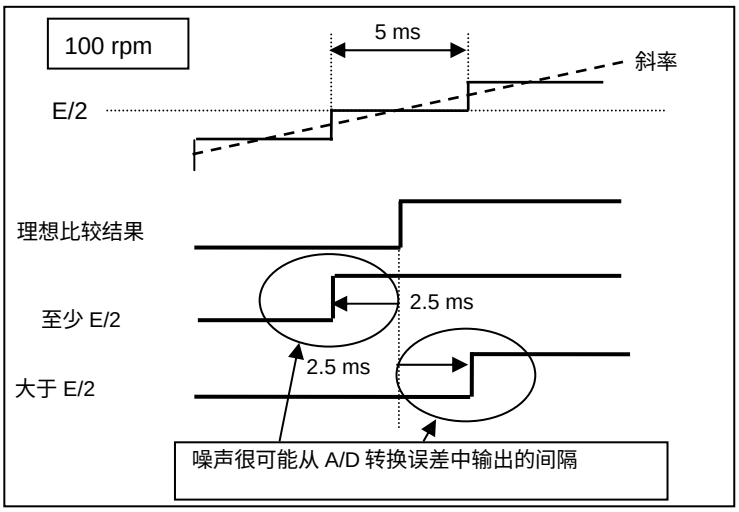
图 4-6. 低速旋转期间的噪声的处理



通过在比较结果更改后相位切换到下一个非激活相位之前不执行比较，噪声可以预防，因为该系统的 CPU 没有在高速时累加 A/D 转换值和计算斜率的处理能力。

在该系统中，应用到电动机的电压（15 V）以 10（1,024）位被 A/D 转换（1 位大约为 0.015 V）。如果在 100 rpm 时从电动机的后 EMF 常量感应的电压为 0.16 V，载波同步中断在 100 rpm 和等效于 60 度的旋转时间内发生的次数是 500，0.08 或 0.015 值的个数是 5（10 在±0.08 范围内），因为在非激活周期期间 0.16 V 的感应电压的电压为 ±0.08 V，并且 A/D 转换值没有包含误差，A/D 转换值每 50 次被更改。（计算斜率至少需要 100 个数据。）

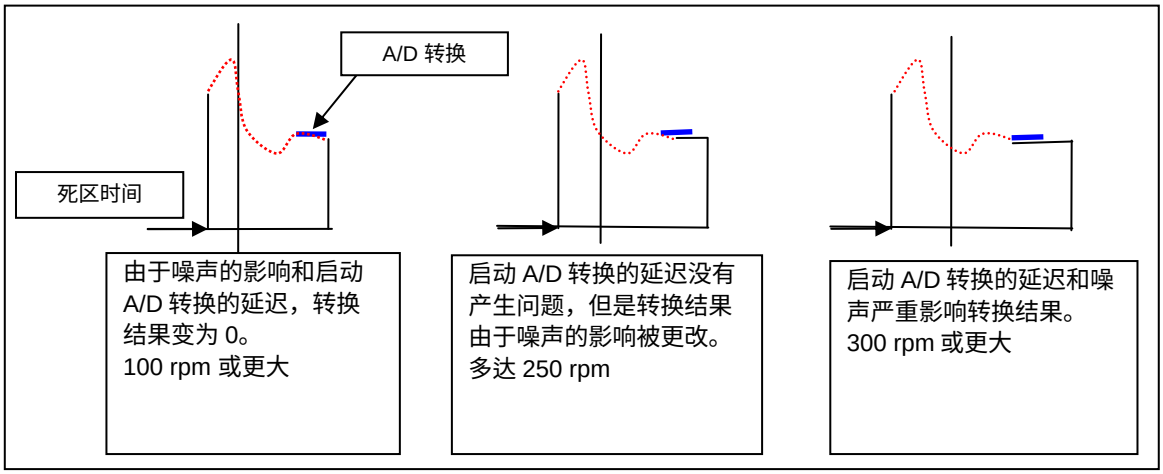
图 4-7. 比较条件引起的误差



在该系统中，比较以“大于 E/2”被执行。

A/D 转换结果受该系统的 CPU 接收中断请求时发生的多达 25 个时钟周期（1.25 μs）的未对准影响。以下状态可以从使用 Hall IC 和执行 A/D 转换驱动 BLDC 电动机的结果来证实。

图 4-8. 旋转个数和 A/D 转换结果

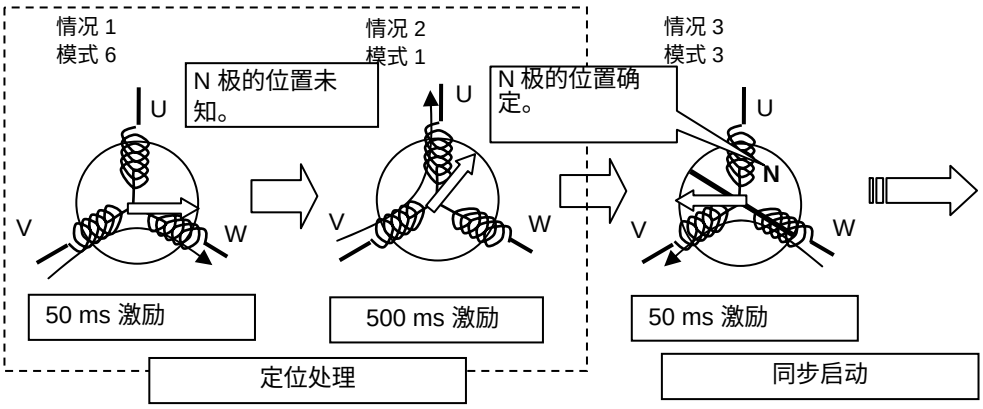


该系统的下限根据以上结果被数值为 300 rpm。

4.6 启动方法

转子的位置由两个指定的相位强制决定，并且激励模式按顺序切换（参见 **图 2-3 激励模式和线圈流动方向**）来增加周期（旋转速度）。指定的时间过去后，控制切换为由激励模式使用 BEMF 信号的控制。

图 0-9. 启动时切换激励模式



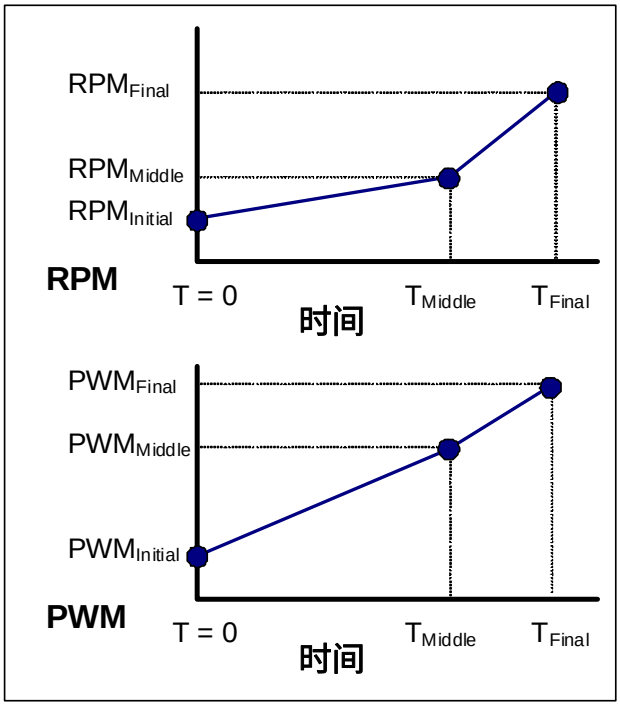
情况 1：如果转子的 N 极在情况 2 中电流流动方向的相反方向，预防转子不转（N 极不移动）的准备操作。
情况 2：强制移动转子的 N 极。
情况 3：从电流流动方向距离转子的磁极位置 60 到 120 度的激励模式启动。

要实现高度灵活的启动，下图表示的启动特性可以被执行。RPM 值和 PWM 值可以在启动（T = 0）后，在给定时间过去（T_{Middle}）后或者移动到 BEMF 信号的控制过去一定时间（T_{Final}）后立即指定。

表 4-3. 可以被指定的参数

经过的时间	转数	PWM 占空比
T = 0	$RPM_{Initial}$	$PWM_{Initial}$
T_{Middle}	RPM_{Middle}	PWM_{Middle}
T_{Final}	RPM_{Final}	PWM_{Final}

图 4-10. 启动特性图



4.7　速度检测

切换激励模式的间隔从发生的载波同步中断（100 μs 周期）的次数来计算（而且，发生的中断次数的一半用作 30 度移动的值），因为该系统的 CPU 处理能力不足以在计算感应电压的斜率的同时控制电动机。

旋转开始后，根据 A/D 转换的比较结果，速度被检测。

速度基于激励模式被切换两次（一次旋转内六次）的时间来计算，因为如果根据六次每次被切换（一次旋转内切换十二次）测量的值来计算，误差过大。

$$N = \frac{60}{s \times n \times 6/2 \times 2} = \frac{100{,}000}{n}$$

N： 每分钟旋转次数（*rpm*）

s： 分辨率 （100 μs）

n： 中断发生的次数

6/2：360 度电角内测量的次数

2： 磁极对的个数

4.8 速度控制

通过调整 PWM 占空比为 150 ms 周期，电动机的转速被控制。
通过基于指定速度和电动机转速之间的偏差执行的 PID 控制操作，要被调整的占空比的变量被得到。
该值作为电流内环的参考电流被产生，而不是对电流内环控制直接操作 PWM。

4.8.1 PID 操作

通过反馈转速和指定速度之间的偏差并在 PWM 占空比（平均电压）上执行 PID 控制操作，速度被调整。PWM 的占空比操作变量使用适合采样方法（离散值）的以下速度类型 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}：前面操作的变量
ΔMV_n：当前和前面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}：　前面的偏差
e_{n-2}：　前面偏差之前的偏差
Kp：　反馈增益（成比例部分）
Ki：　反馈增益（积分部分）
Kd：　反馈增益（导数部分）

反馈增益的最优值根据电动机特性和负载的有无而改变。
对于该系统，检查操作时通过试凑法得到的值被设置为默认值。

4.9 电流内环控制

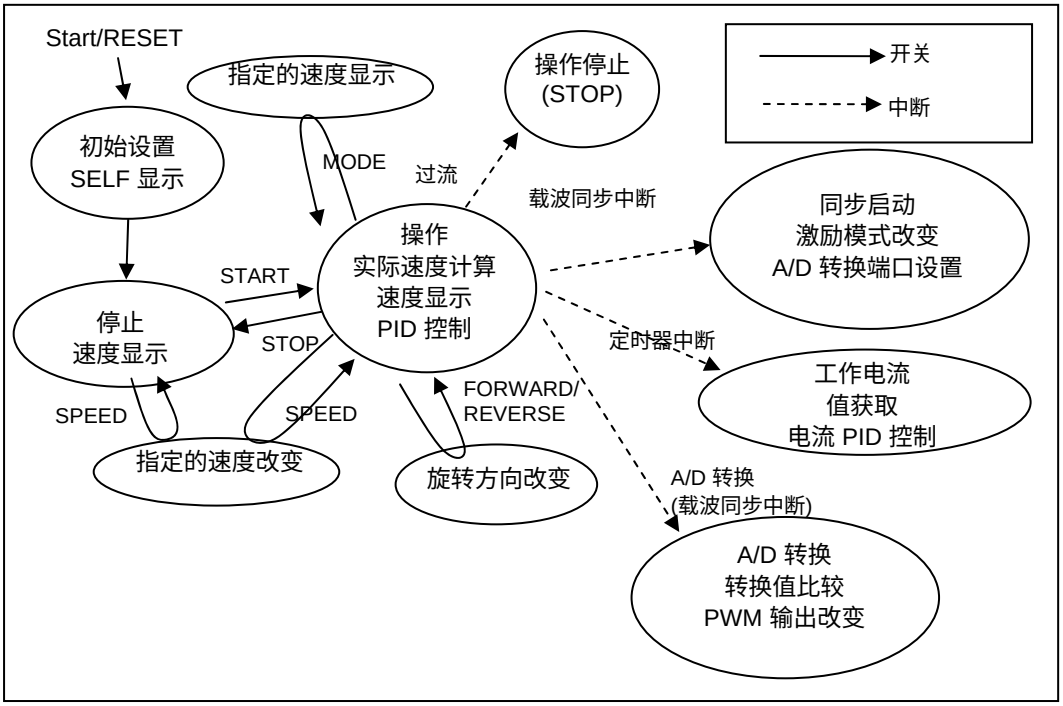
电动机的旋转次数通过在 1.43 ms 周期内调整 PWM 占空比来控制。
通过基于指定速度和电动机转速之间的偏差执行的 PID 控制操作，要被调整的占空比的变量被得到。

与速度控制类似，对于速度控制产生的参考电流，PWM 占空比将被基于电动机测量的工作电流的偏差执行的 PID 控制操作来修改。

4.10　模块配置

图 4-11 表示系统的状态转换。

图 0-11. 系统的状态转换



载波同步中断被初始并且 A/D 转换以 50 μs 的间隔交替发生。

4.11　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。关于处理的细节，参见流程图。

4.11.1 用户可以使用的函数

表 0-4. 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	指示电动机旋转开始。
motor_stop()	停止指令	指示电动机旋转停止。
motor_rotation()	旋转方向指令	指示电动机旋转的方向。
motor_pid()	速度控制 PID 操作	指示用于速度控制的 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 0-5. 电动机库内部函数（1／2）

函数名	功能	用途
system_restart()	重启处理	停止方向旋转后重启。
system_stop()	停止处理	停止系统。
init_openloop()	初次启动的启动顺序设置处理	计算初次启动期间启动顺序的参数。
filters()	数字滤波处理	执行电流获取值的数字滤波处理。
current_pwm()	对电流内环的 PID 操作处理	对电流内环执行 PID 操作。
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()	逆变器设置处理	设置逆变器定时器。
start_TW0()	逆变器启动处理	启动逆变器定时器。
stop_TW0()	逆变器停止处理	停止逆变器定时器。
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()	实时输出端口停止处理	停止实时输出端口。
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。

表 0-5. 电动机库内部函数（2／2）

函数名	功能	用途
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_TM50()	TM50 设置处理	设置 （1 ms）8 位定时器。
wait()	等待	等待指定的时间。
init_TM51()	TM51 设置处理	设置 （1.43 ms）8 位定时器。
start_TM51()	TM51 启动处理	启动 8 位定时器。
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（波谷）中断。
INTTW0UD_off()	INTTW0UD 禁止处理	禁止载波同步（波谷）中断。
INTTW0CM3_on()	INTTW0CM3 允许处理	允许载波同步（波峰）中断。
INTTW0CM3_off()	INTTW0CM3 禁止处理	禁止载波同步（波峰）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_faulta()	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_tw0cm3()	载波中断（波峰）服务程序	A/D 转换非激活相位的电压，更新速度信息并执行 PWM 更改处理。
int_TM51()	电流内环的定时器中断服务程序	对电流内环执行操作处理。

4.11.3　参考程序函数

表 0-6. 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	指示 MC-IO 板上开关信息的读取并返回对应开关读取的操作信息。
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	清除看门狗定时器的定时器值。

4.11.4 流程图

每个函数的流程图如下所示。

- 电动机库

图 4-12. 电动机初始设置处理（motor_init 函数）

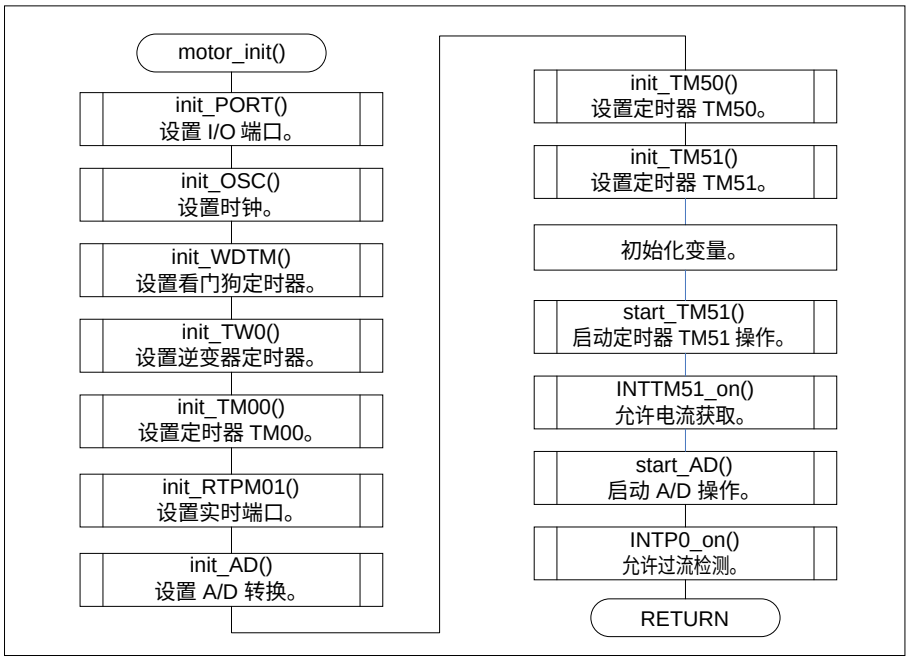


图 4-13. 电动机启动处理（motor_start 函数）

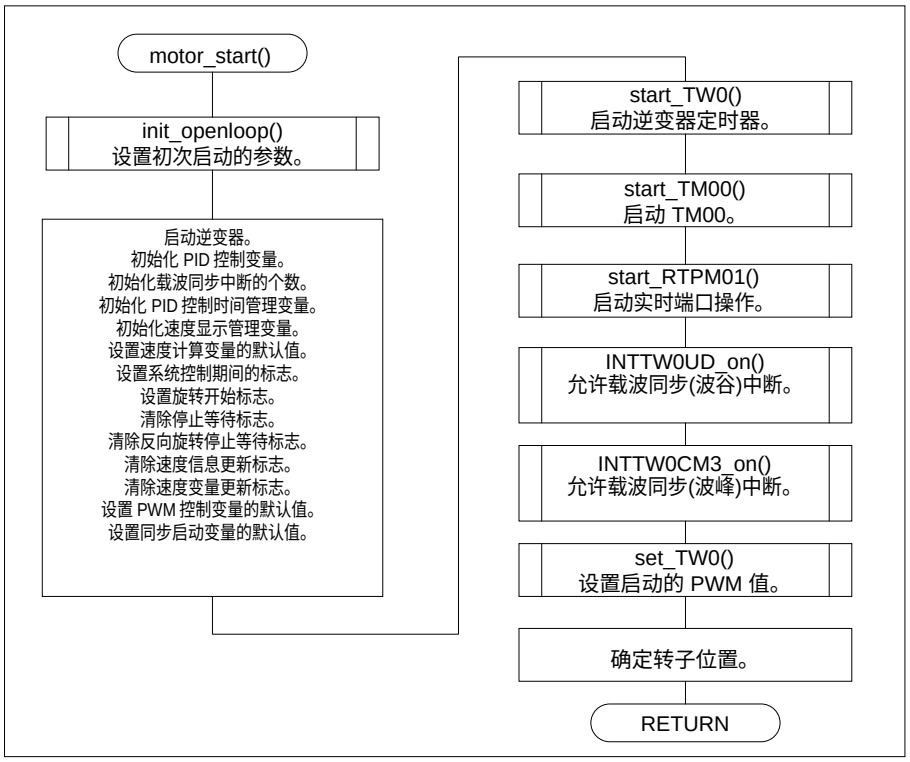


图 4-14. 电动机停止处理（motor_stop 函数）

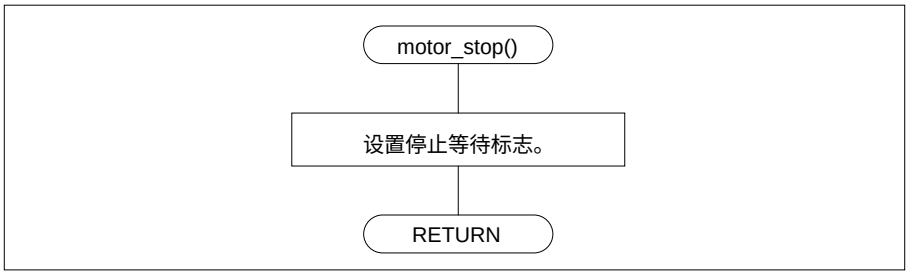


图 4-15. 电动机旋转方向更改处理（motor_rotation 函数）

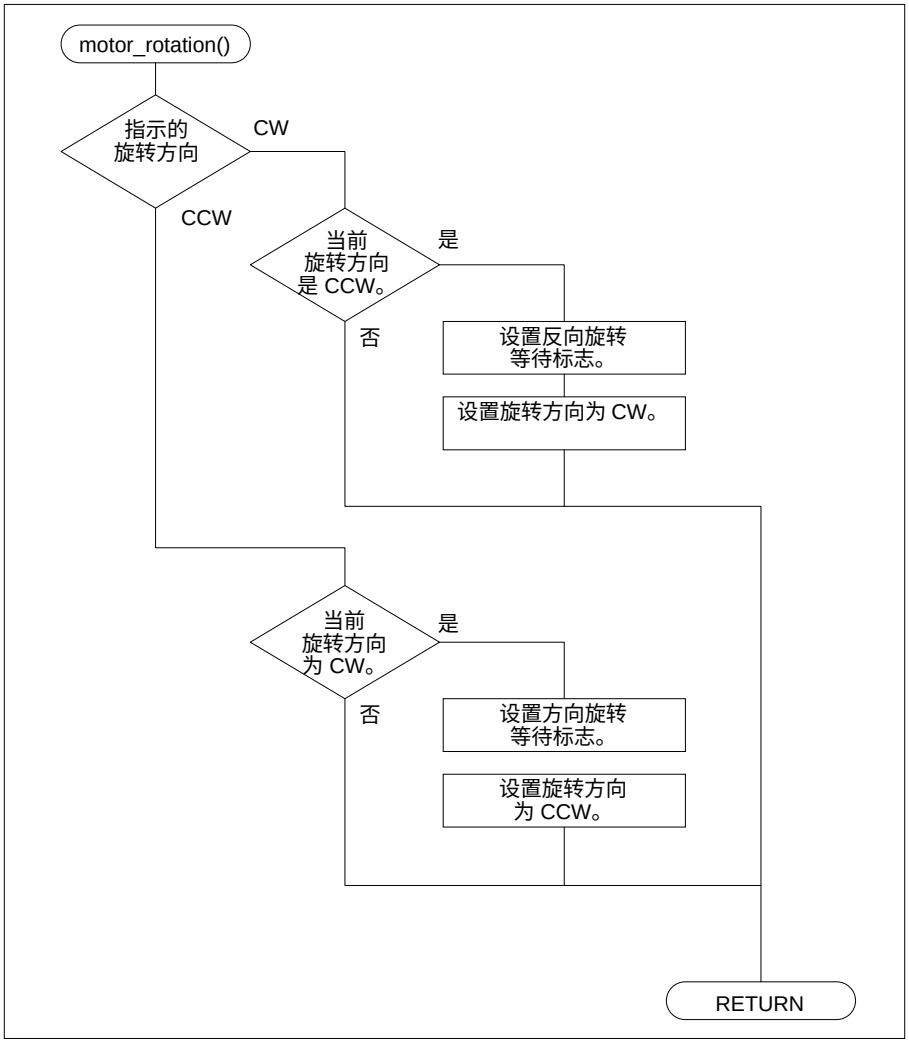


图 4-16. 速度 PID 控制处理（motor_pid 函数）

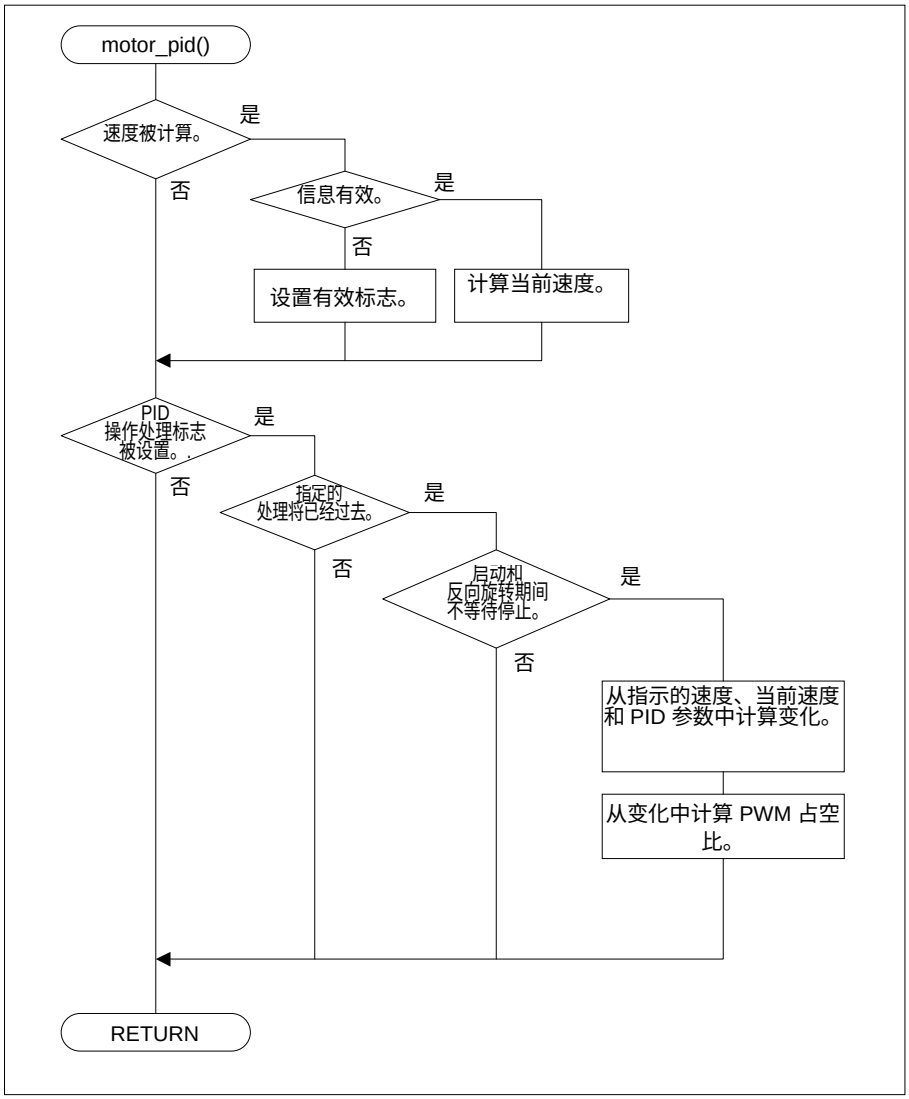


图 4-17. 电动机控制参数更改处理（motor_pset 函数）

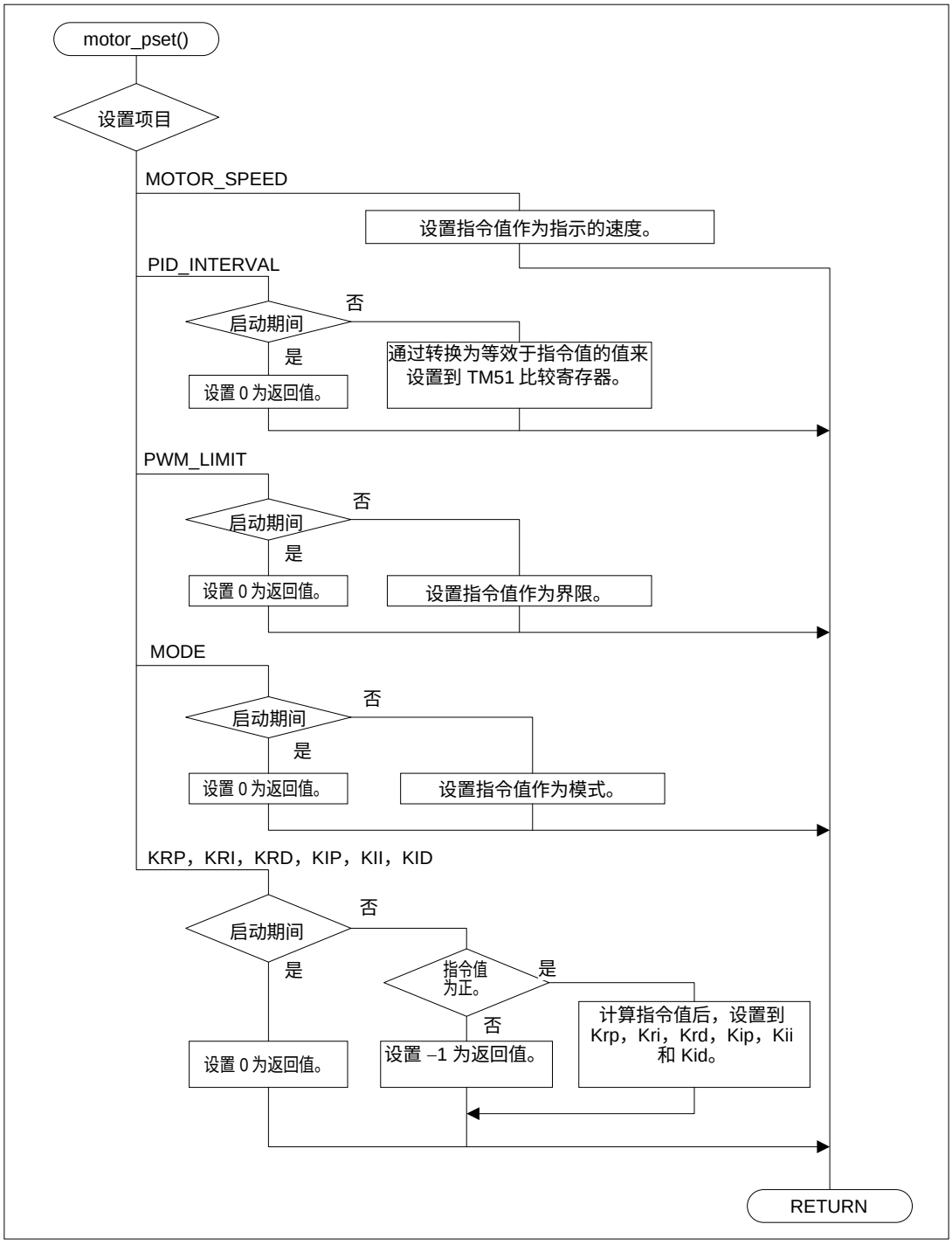


图 4-18. 从反向旋转的停止重启的处理（system_restart 函数）

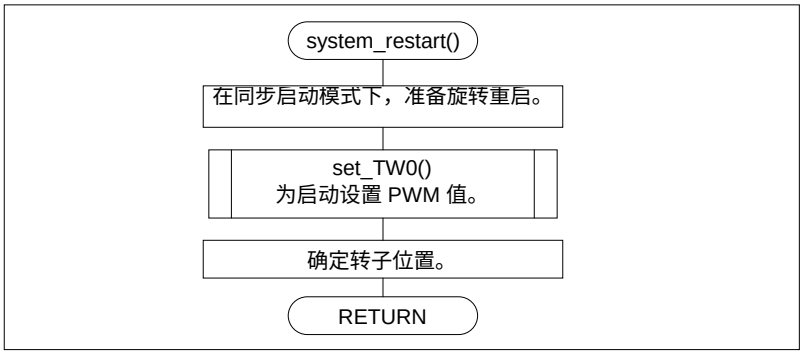


图 4-19. 系统停止处理（system_stop 函数）

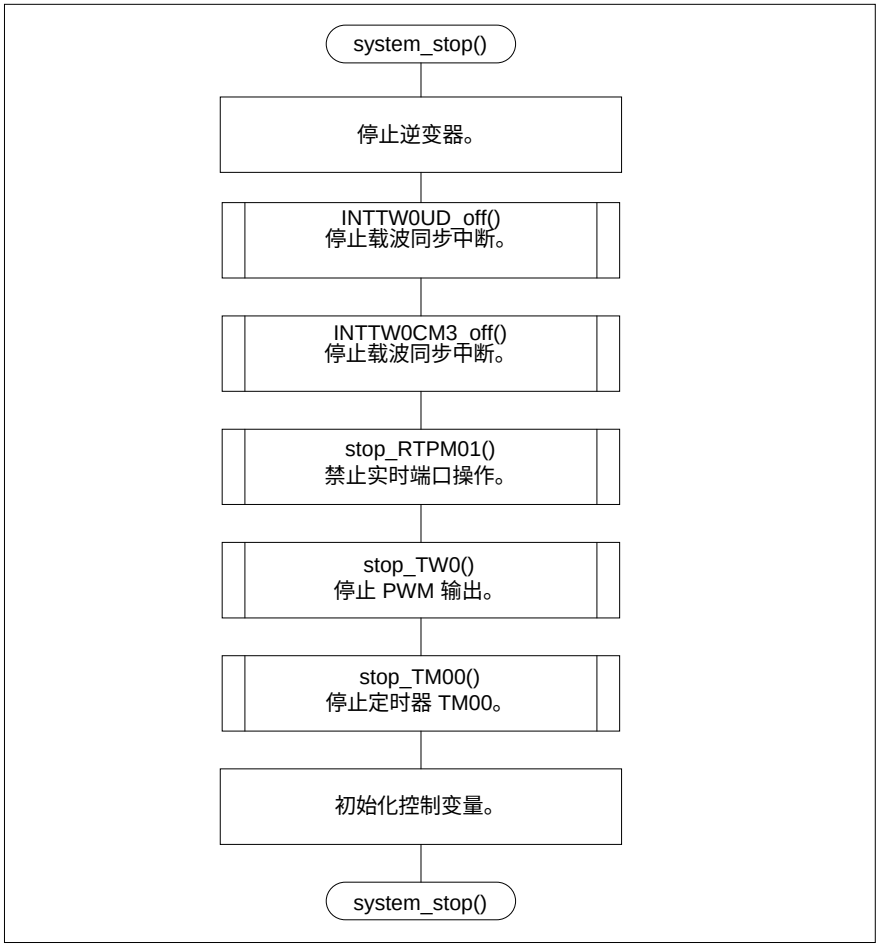


图 4-20. 从方向旋转的停止重启的处理（init_openloop 函数）

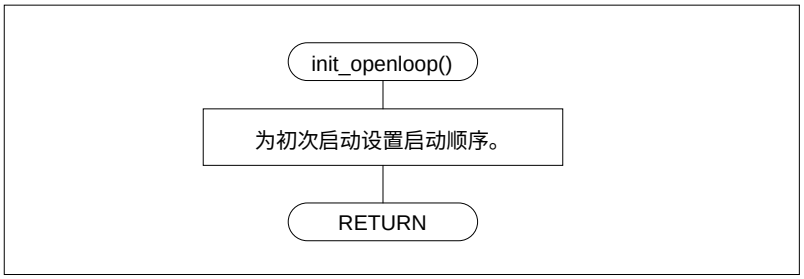


图 4-21. 端口设置处理（init_PORT 函数）

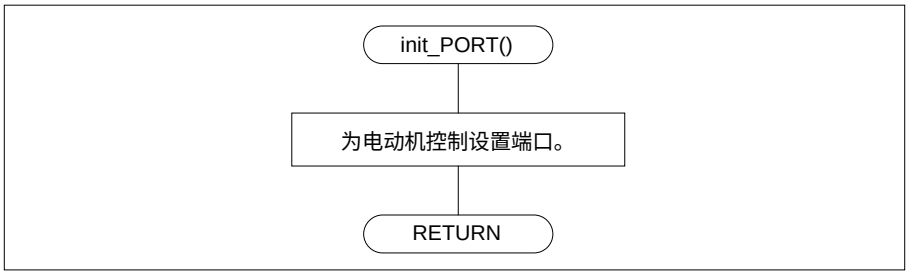


图 4-22. 时钟切换处理（init_OSC 函数）

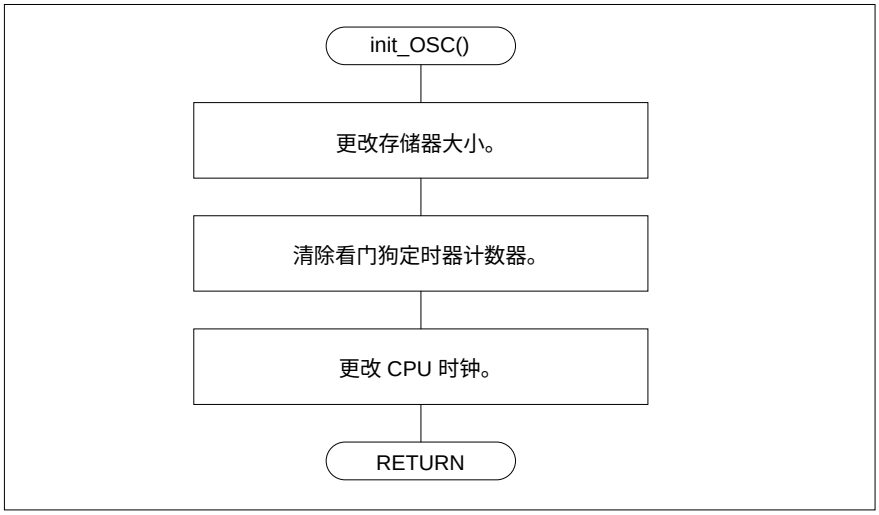


图 4-23. 逆变器定时器初始设置处理（init_TW0 函数）

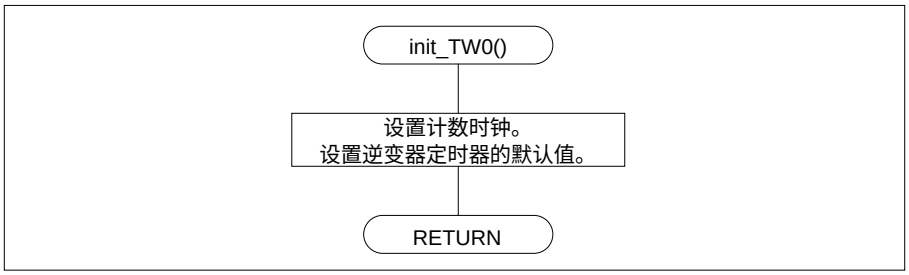


图 4-24. 逆变器定时器操作开始处理（start_TW0 函数）

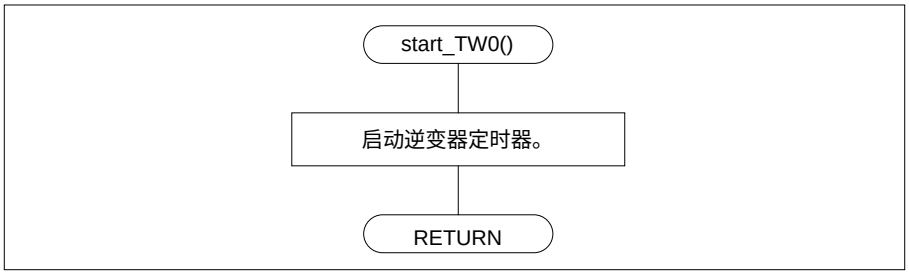


图 4-25. PWM 占空比设置处理（set_TW0 函数）

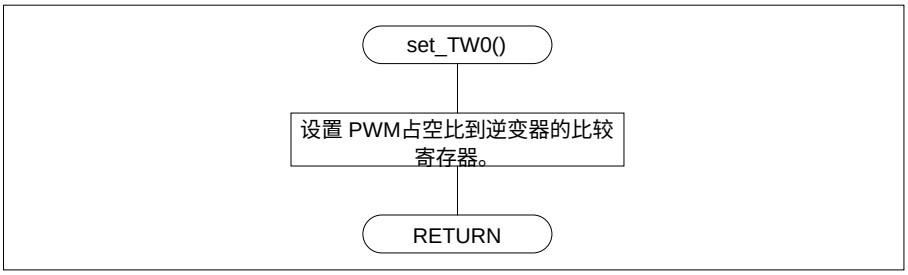


图 4-26. 逆变器定时器操作停止处理（stop_TW0 函数）

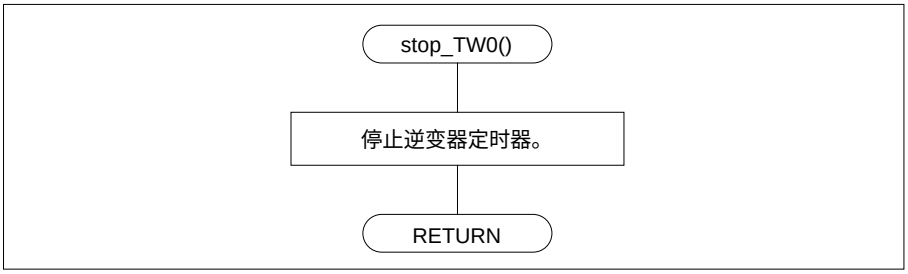


图 4-27. 激励模式切换的定时器初始设置处理（init_TM00 函数）

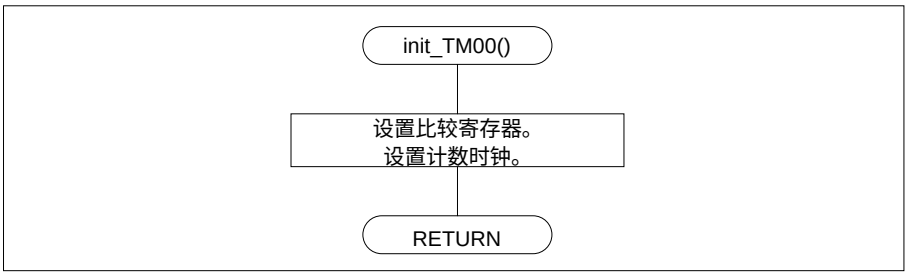


图 4-28. 激励模式切换的定时器启动处理（start_TM00 函数）

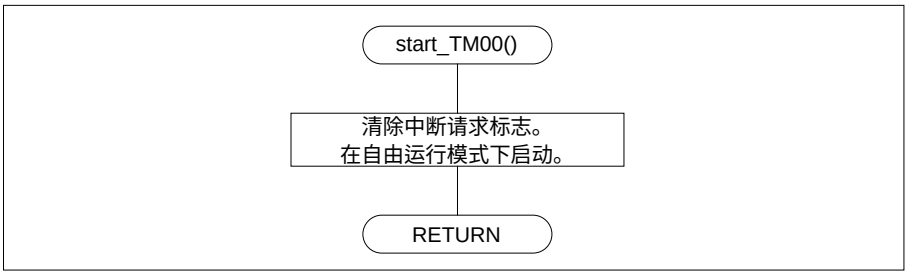


图 4-29. 激励模式切换的定时器停止处理（stop_TM00 函数）

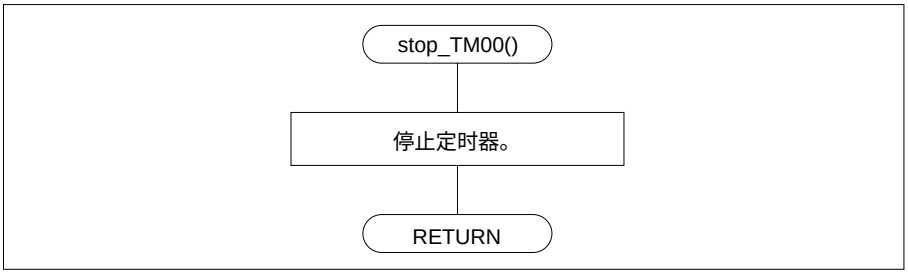


图 4-30. 激励模式切换的端口初始设置处理（init_RTPM01 函数）

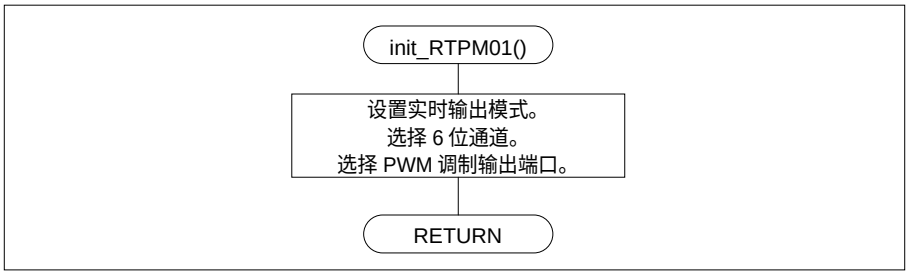


图 4-31. 激励模式设置（set_RTPM01 函数）

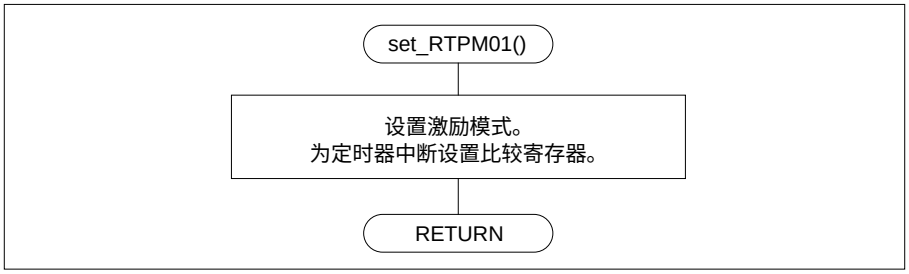


图 4-32. 激励模式切换的端口输出允许处理（start_RTPM01 函数）

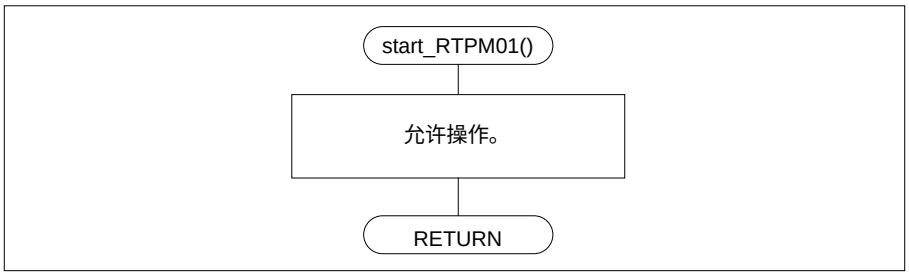


图 4-33. 激励模式切换的端口输出禁止处理（stop_RTPM01 函数）

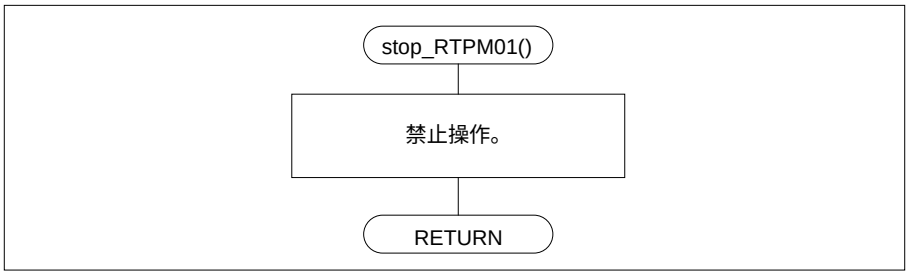


图 4-34. A/D 初始设置处理（init_AD 函数）

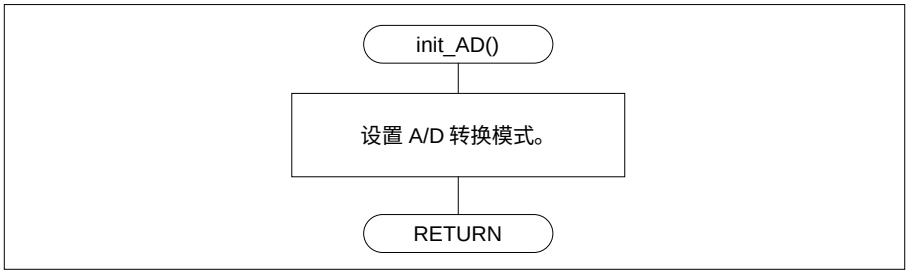


图 4-35. A/D 转换操作启动处理（start_AD 函数）

图 4-36. 看门狗定时器初始设置处理（init_WDTM 函数）

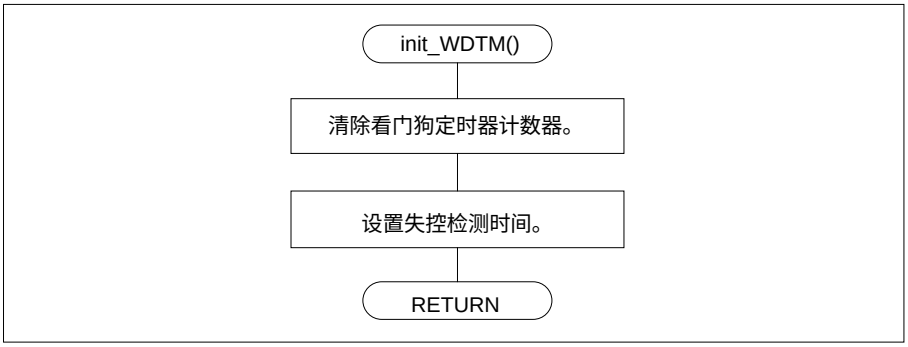


图 4-37. 8 位定时器（TM51）初始设置处理（init_TM51 函数）

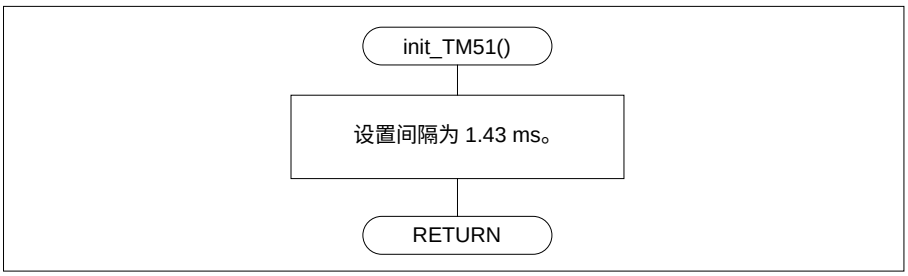


图 4-38. 8 位定时器（TM51）启动处理（start_TM51 函数）

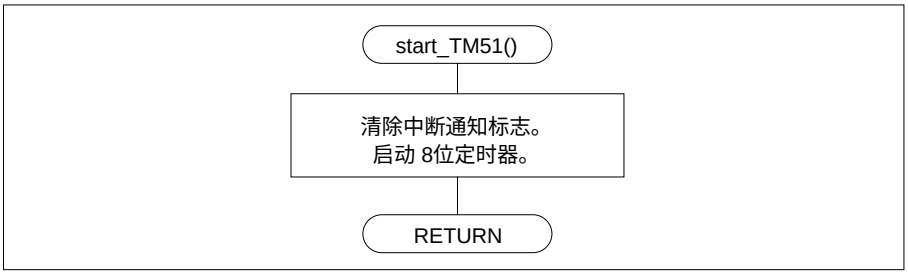


图 4-39. 8 位定时器（TM50）初始设置处理（init_TM50 函数）

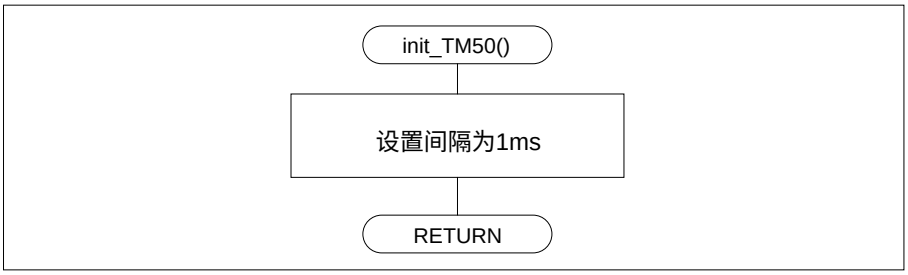


图 4-40. 过流中断允许处理（INTP0_on 函数）

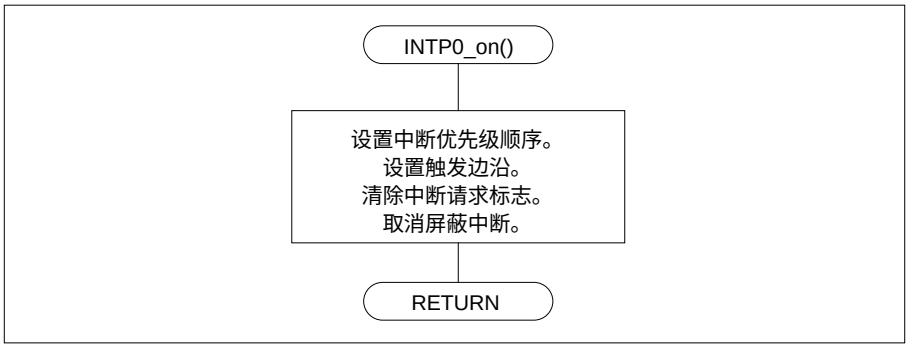


图 4-41. 载波同步中断（波谷处理）允许处理（INTTW0UD_on 函数）

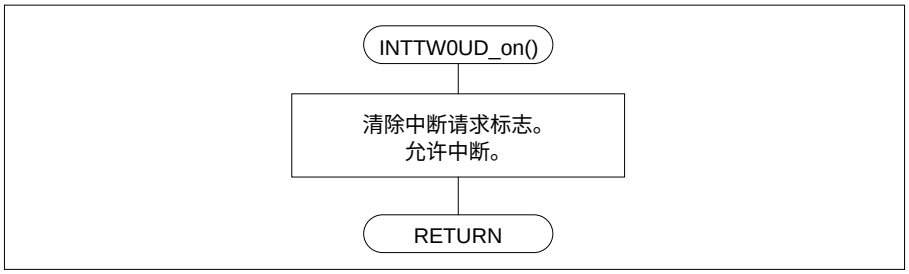


图 4-42. 载波同步中断（波谷处理）禁止处理 （INTTW0UD_off 函数）

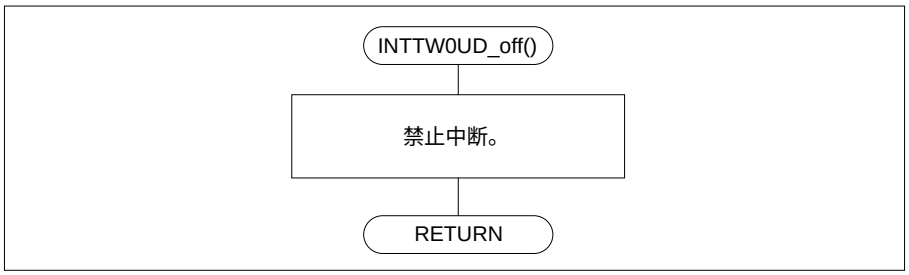


图 4-43. 载波同步中断（波峰处理）允许处理（INTTW0CM3_on 函数）

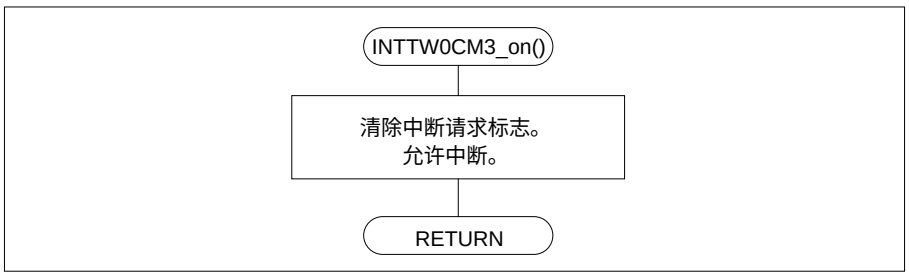


图 4-44. 载波同步中断（波峰处理）禁止处理 （INTTW0CM3_off 函数）

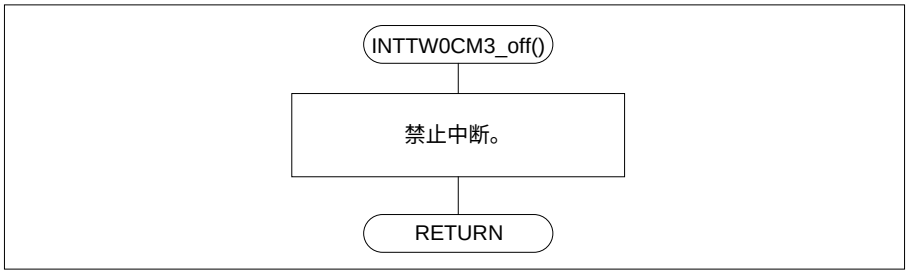


图 4-45. 电流内环中断允许处理（INTTM51_on 函数）

图 4-46. 等待指定的时间过去的处理（wait 函数）

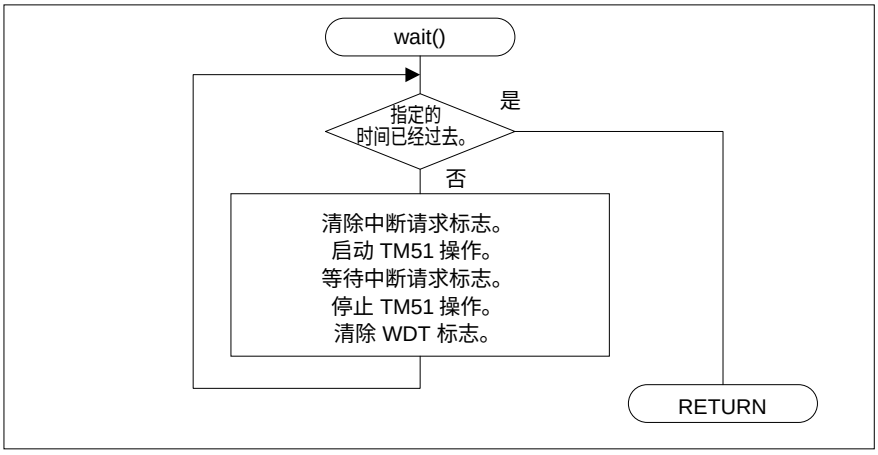


图 4-47. 过流中断服务程序（int_faulta 函数）

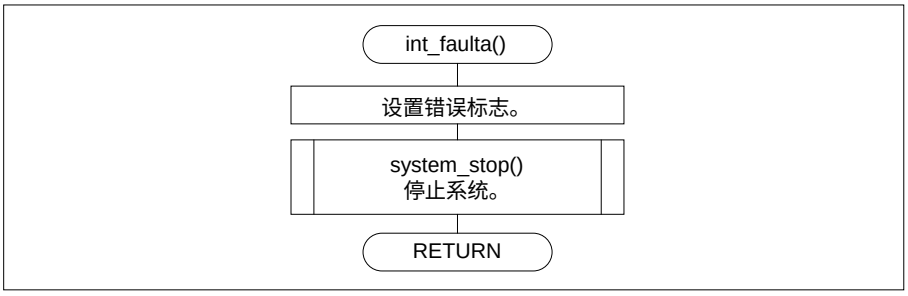


图 4-48. 电流内环中断服务程序（int_TM51 函数）

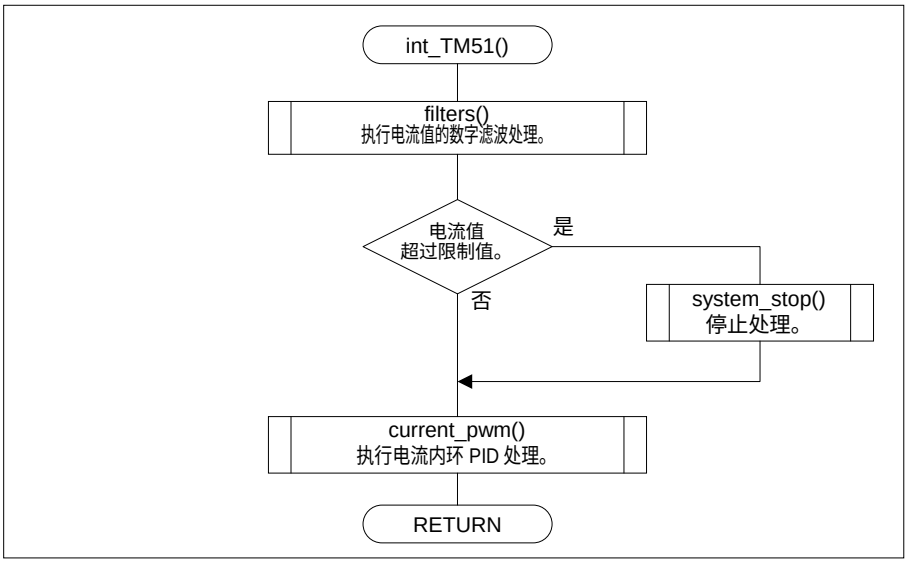


图 4-49. 载波同步中断（波谷处理）（1／4）（int_carrier 函数）

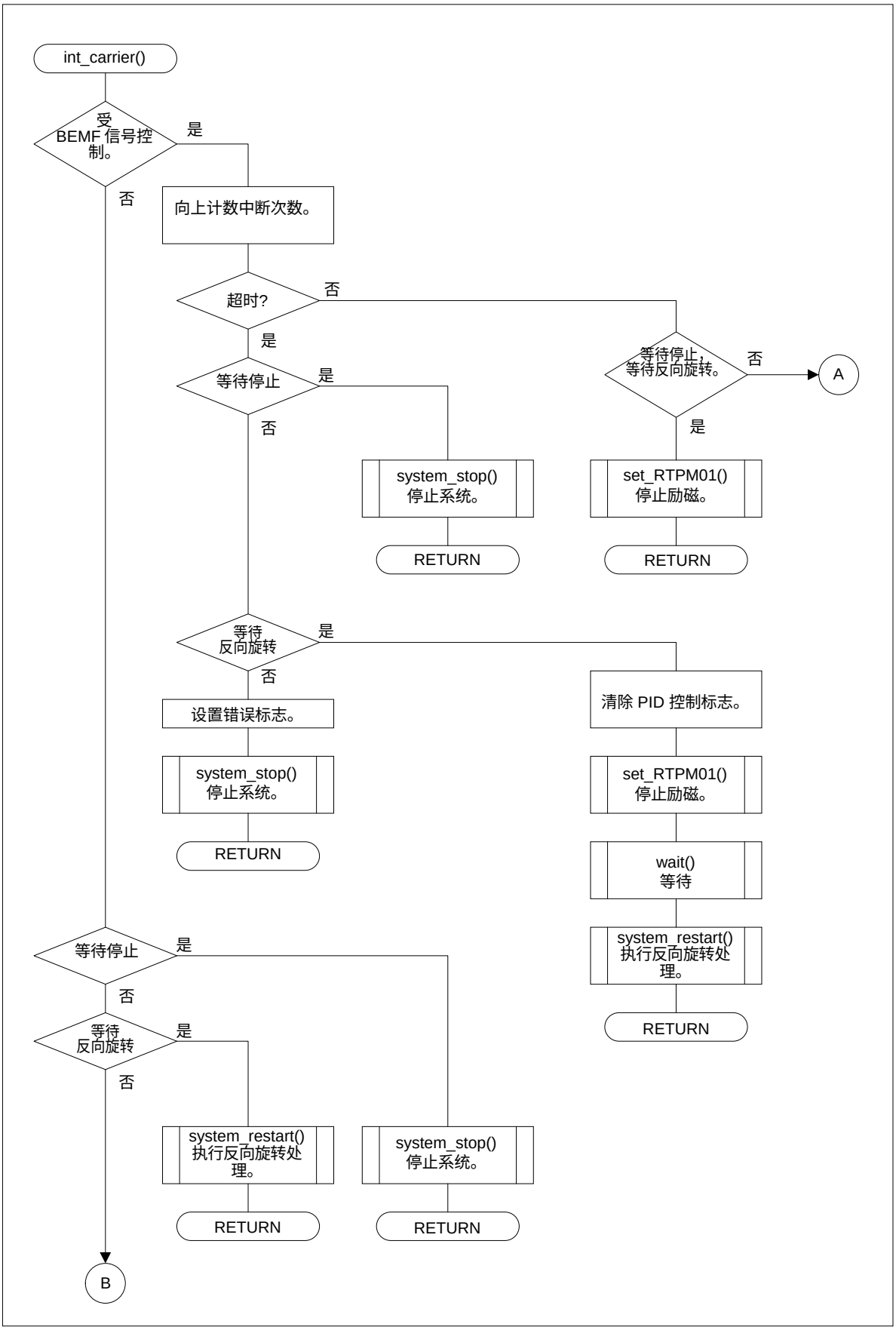


图 4-50. 载波同步中断（波谷处理）（2／4）（int_carrier 函数）

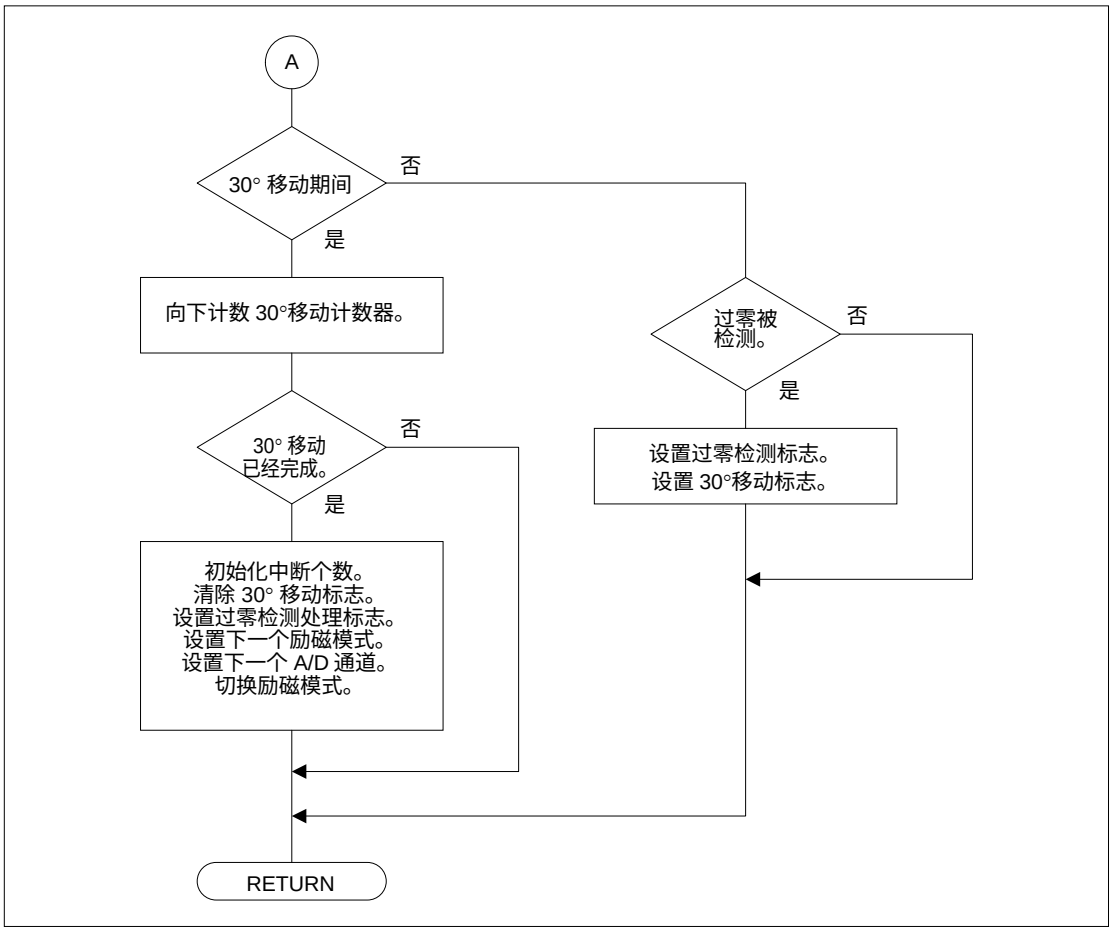


图 4-51. 载波同步中断（波谷处理）（3／4）（int_carrier 函数）

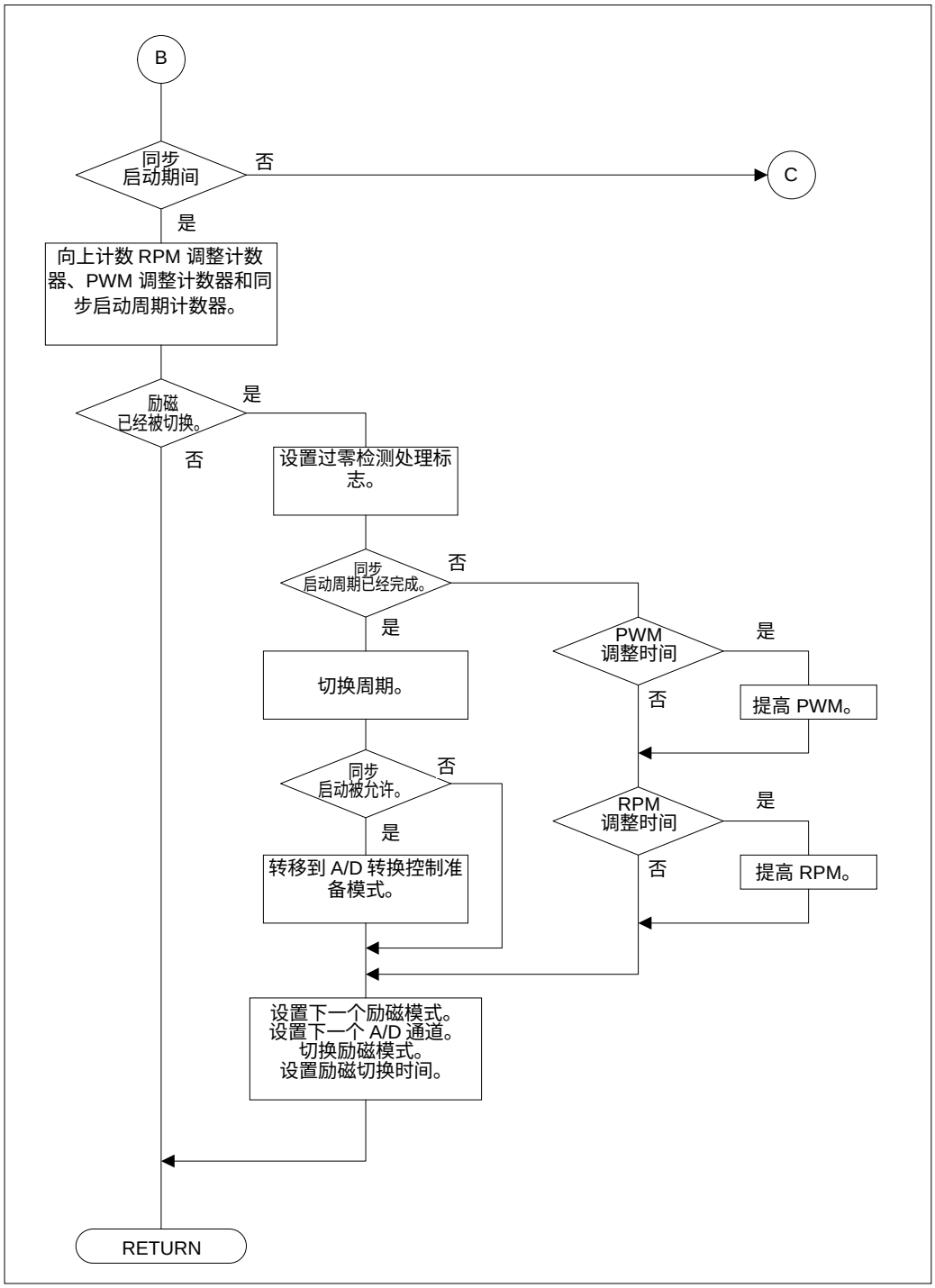


图 4-52. 载波同步中断（波谷处理）（4／4）（int_carrier 函数）

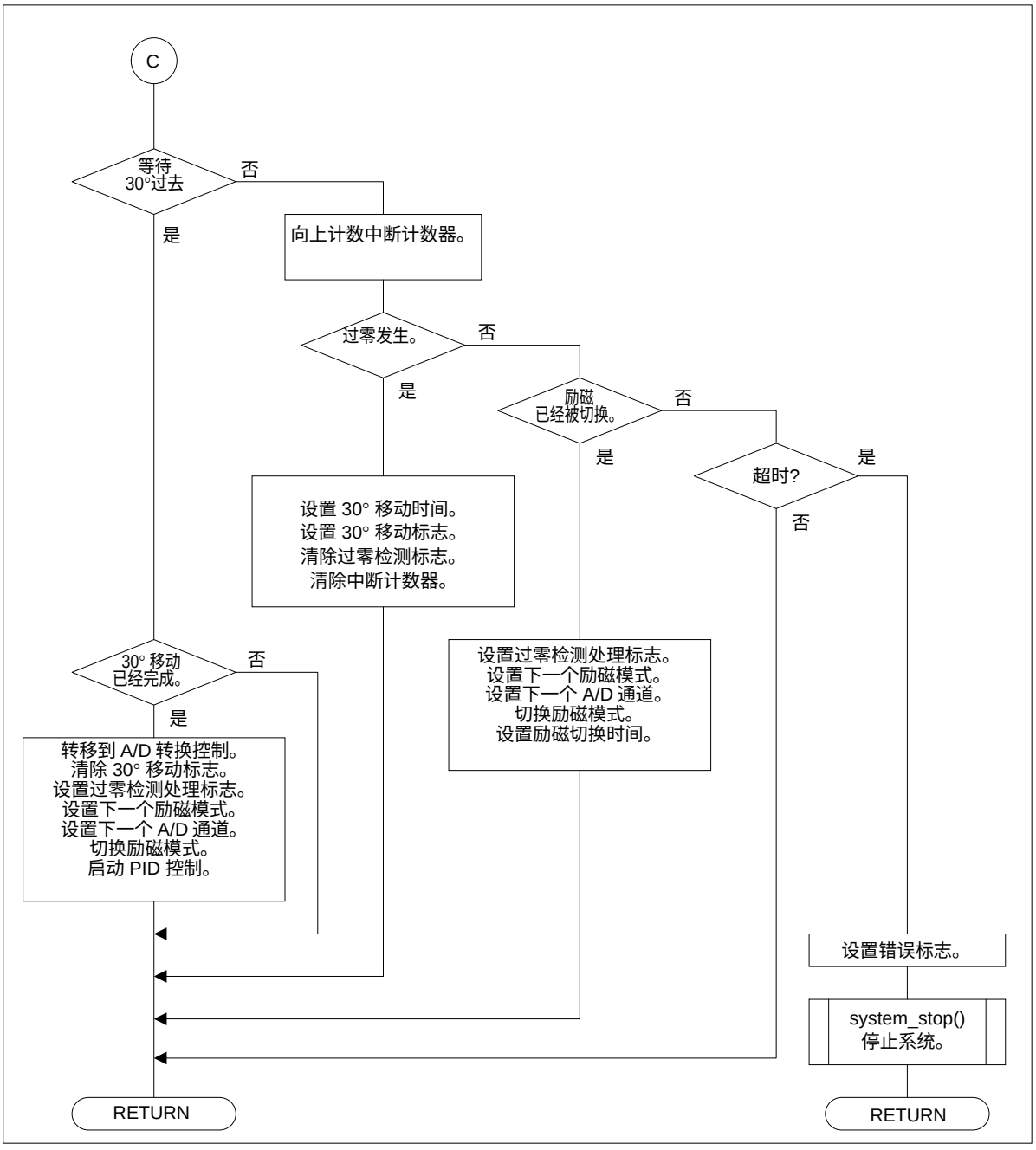


图 4-53. 载波同步中断（波峰处理）（int_tw0cm3 函数）

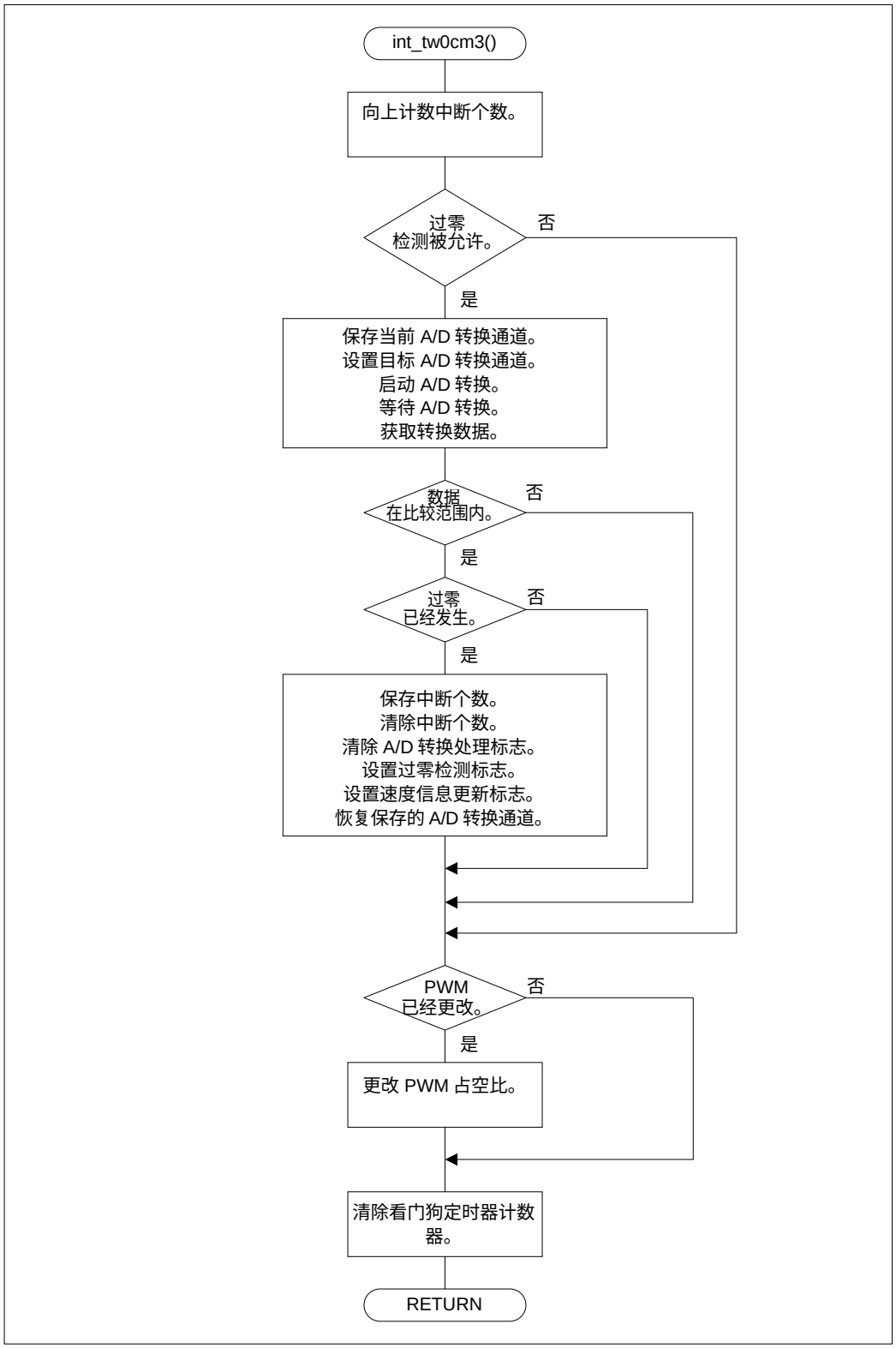


图 4-54. 数字滤波处理（filters 函数）

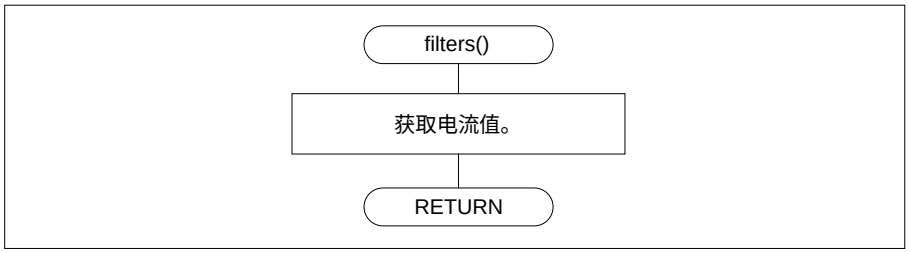


图 4-55. 电流内环 PID 处理（current_pwm 函数）

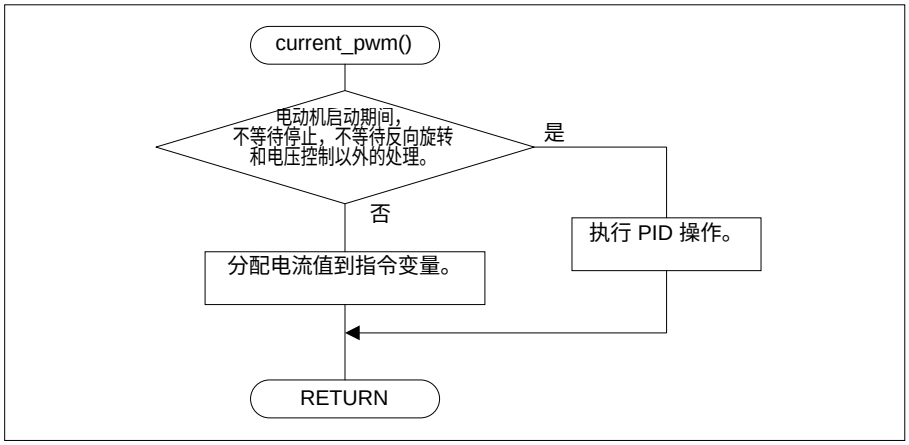
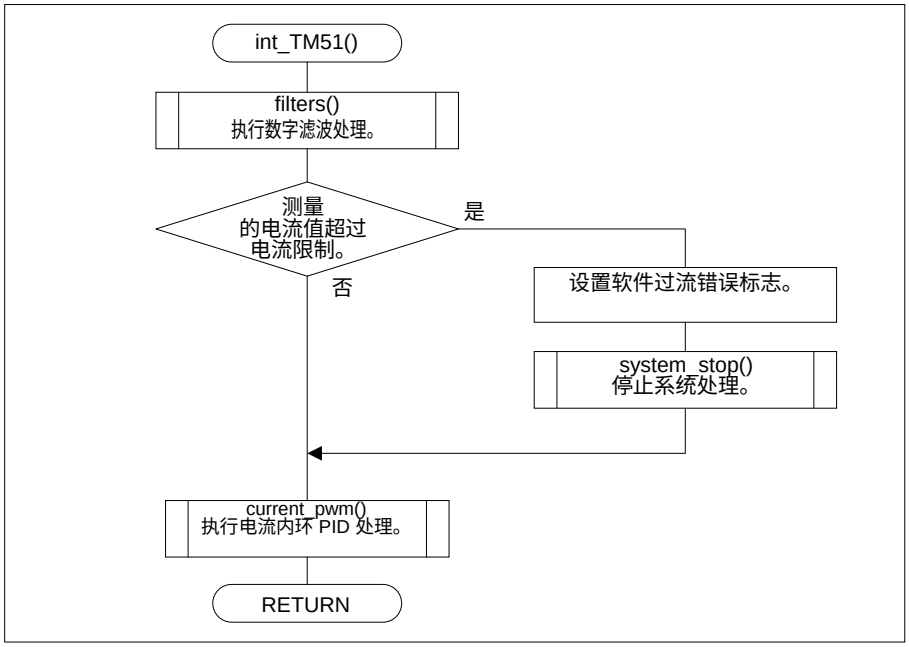


图 4-56. 电流获取的中断服务程序（int_TM51 函数）



4.12　电动机库常量列表

4.12.1　用户可更改的常量

- 低压逆变器设置

表 4-7. 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
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速	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	

4.12.2 用户可引用的常量

表 4-8. 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	Hall IC 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	

4.12.3 内部常量

表 4-9. 电动机库内部常量

名称	含义	设置值	备注
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	用于逆变器控制	1/0	低压逆变器
INV_ON	用于逆变器控制	0/1	
INVERTER_SW	逆变器控制端口	P54	逆变器控制端口
INVERTER_SW_MODE	逆变器控制端口	PM54	逆变器控制端口
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	励磁模式	0x3f	实时端口的激励
P_STOP	励磁模式	0x15	
P_T1	励磁模式	0x36	
P_T2	励磁模式	0x1e	
P_T3	励磁模式	0x1b	
P_T4	励磁模式	0x39	
P_T5	励磁模式	0x2d	
P_T6	励磁模式	0x27	

4.13　参考程序常量列表

4.13.1 内部常量

表 4-10. 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	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	输出数据到 LED 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”代替“O”。
LED_I		0xcf	显示 “I”。
LED_C		0xc6	显示 “C”。
LED_H		0x89	显示 “H”。

表 4-10. 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 “.”。

4.14 电动机库变量列表

4.14.1 外部公开变量

表 4-11. 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	表明停止命令是否被发布。
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	表明反向停止的命令是否被发布。
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	载波同步中断的个数
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	
en	float	与当前值的偏差	
en_1	float	与前面值的偏差	
en_2	float	与前面的前面值的偏差	
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	
ec	float	与当前值的偏差	
ec_1	float	与前面值的偏差	
ec_2	float	与前面的前面值的偏差	
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	

表 4-11. 外部公开的电动机库变量（2／2）

变量名	类型	含义	备注
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	
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	

4.14.2 内部变量

表 4-12. 电动机库内部变量

变量名	类型	含义	备注
old_hdata	char	Hall IC 历史	前面 Hall IC 的值
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	

4.15 参考程序变量列表

4.15.1 内部变量

表 4-13. 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。

4.16　电动机库源文件

存储器大小　ROM： 1F3Dh
　　RAM： 320h

```
/*


   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

   支持模块转换
   电流控制版本

   目标　　：  uPD78F0714 电动机入门套件
   日期　　：  2007/05/10
   文件名　：  motor.h

   NEC Micro Systems,Ltd
*/

#ifdef LOW
/* ================================================ */
/* For Pittman motor */
#define PWM_F_MIN               5               /* 最小值                       */
#define PWM_F_MAX               PWM_BASE_REF    /* 最大值                       */

/* 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                180             /* 初始启动电压设置 */
#define PWM1_REF                185             /* 中点启动电压设置 */
#define PWM2_REF                190             /* 最后启动电压设置 */

/* 最初启动 NOTABLE_START */
#define SYNC_DEF_REF            500
#define PWM_F_START             10              /* 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 error */
#define KRI_DEF_REF             0.02            /* dI_ints/RPM error  */
#define KRD_DEF_REF             0.01            /* dI_ints/d(dRPM error) */

/*  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
/* ================================================ */
/* For Oriental motor */
#define PWM_F_MIN               5               /* 最小值                   */
#define PWM_F_MAX               PWM_BASE_REF    /* 最大值                   */

/* 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                60              /* 初始启动 RPM */
#define RPM1_REF                100             /* 中点启动 RPM */
#define RPM2_REF                200             /* 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                100             /* 初始启动电压设置 */
#define PWM1_REF                100             /* 中点启动电压设置 */
#define PWM2_REF                100             /* 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            200             /* */
#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             0.02            /* dI_ints/dRPM error */
#define KRI_DEF_REF             0.01            /* dI_ints/RPM error  */
#define KRD_DEF_REF             0               /* dI_ints/d(dRPM error) */

/* 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_CH1                  3               /* 相位 U */
#define AD_CH2                  6               /* 相位 V */
#define AD_CH3                  7               /* 相位 W */
#define AD_ISHUNT               5               /* ISHUNT 引脚 */
#define AD_VOL                  4               /* VOL 引脚   */

/* +++++++++++++++ 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_HALL              0x01            /* Hall IC 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 常量： x[rpm] = ( 60[s] * 78.125[Krps] )/6
                                        ------          ---
                                          ↑               ↑计数器值获取
                                          ↑               ↑时序
                                          ↑               ↑HallIC(6)
                                          ↑TM00 计数时钟
                                          ↑78.125[KHz]
 */
#define UNIT_RPM                781250   /* 2 计数 */
#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;            /* 当前转数                */
extern char                     stop_wait;          /* 停止等待标志                    */
extern char                     cw_ccw_wait;        /* 反向等待标志            */
extern char                     cw_ccw_flag;        /* 旋转方向状态            */
extern char                     rr_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);

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

```
/*

 BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

  支持模块转换
  电流控制版本

  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 di
#pragma ei

#pragma INTERRUPT INTP0          int_faulta     rb1   /* 过流发生                  */
#pragma INTERRUPT INTTW0UD       int_carrier    rb2   /* 载波同步中断                */
#pragma INTERRUPT INTTW0CM3      int_tw0cm3     rb3   /* A/D 转换                  */
#pragma INTERRUPT INTTM51        int_TM51       rb1   /* ISHUNT 测量               */


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

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

/* ---------------------------------------- */
#define CLEAR                   0
#define SET                     1

#define FLG_OFF                 0
#define FLG_ON                  1

#define INV_OFF                 1           /* 用于低压 */
#define INV_ON                  0

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

#define INVERTER_SW             P54         /* 逆变器操作控制端口  */
#define INVERTER_SW_MODE        PM54        /* 逆变器操作控制端口  */
#define INTP0                   PM00        /* 过流检测端口  */

#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 U → V */
#define P_T2                    0x1e        /* 激励模式 2 U → W */
#define P_T3                    0x1b        /* 激励模式 3 V → W */
#define P_T4                    0x39        /* 激励模式 4 V → U */
#define P_T5                    0x2d        /* 激励模式 5 W → U */
```

```
#define P_T6                        0x27        /* 激励模式 6 W → V */

/* ------------------------------------------------ */
#define FLG_START                   1
#define FLG_WAIT                    2

#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 INTERVAL_BASE               10          /* 1ms = 100[us] * INTERVAL_BASE     */
#define TIME_OUT                    10000       /* 1s  = 100[us] * TIME_OUT          */
                                                /* 依赖于逆变器定时器设置          */

/* ------------------------------------------------ */
const unsigned char  sync_tbl[2][6] = {         /* 同步启动励磁模式                  */
   {P_T1, P_T6, P_T5, P_T4, P_T3, P_T2},        /* 用于低压                      */
   {P_T1, P_T2, P_T3, P_T4, P_T5, P_T6}
};

/* ------------------------------------------------ */
char                sys_flag;                   /* 系统操作状态                  */
int                 m_speed;                    /* 当前转数                      */
static char         cw_ccw_flag;                /* 旋转方向状态                  */
char                err_flag = FLG_OFF;         /* 错误标志                      */
unsigned char       maxed_flags;                /* 如果达到界限，指定的位： 位 7-6-5 分别是 RPM-电流-电压 */
static char         up_flag = FLG_OFF;          /* 速度更新检查标志              */

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

static char         cmd_mode = SPEED_CMD;

static float        mvn_f;                      /* 修改的变量                    */
static float        en, en_1, en_2;             /* 偏移                          */
static float        ec, ec_1, ec_2;             /* 偏移                          */

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;

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

/* 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;

/* ------------------------------------------------ */
/* 启动查找表                */
#define TABLESIZE 60

static unsigned int pwm_ff_table[TABLESIZE];
static float        I_ref_table[TABLESIZE];
static int          sync_def_table[TABLESIZE];
static int          i_sync;
static int          i_sync_max, i_sync_skip;
static unsigned char startdelay;
unsigned char       startup_method = STARTUP_METHOD_REF;

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;

/* ------------------------------------------------ */
static char         start_flag;
static char         sync_sw;                    /* 激励模式                      */
static char         sync_flag;                  /* 同步启动期间的鉴定标志        */
static unsigned int sync_cnt;                   /* 励磁切换管理（x100 us）       */
static unsigned int sync_def;                   /* 励磁切换时间（x100 us）       */

static unsigned int cy_time = 150 * INTERVAL_BASE;
static unsigned int int_cnt;                    /* 超时计数器                    */
static unsigned int pid_cnt;
static unsigned char pid_flag;
unsigned int        print_cnt;
```

```
static char                 speed_flag   = FLG_OFF;
static unsigned int         speed_cnt;                  /* 30 度延迟之前的时间      */
static unsigned int         clk_cnt;                    /* 速度信息                                  */
static unsigned int         new_clk, old_clk;

int                         m_speed;
int                         speed_ref = 0;              /* 指定的速度               */
int                         speed_ref_o;
float                       I_ref;
float                       I_ref_o;
float                       I_measured;
unsigned int                ad_data_shunt;
unsigned int                ad_data_vol;
int                         pwm_ff;
int                         pwm_ff_o;

const  unsigned int         pwm_base = PWM_BASE_REF; /* PWM 周期的一半： 固定为 1000  */

static char                 ad_read_flag = FLG_ON;
static char                 ad_flag          = FLG_OFF;
static unsigned char        ad_port;
static unsigned int         ad_int_cnt;

/* ------------------------------------------------ */
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_TM50(void);
static void init_TM51(void);
static void init_WDTM(void);
static void init_openloop(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 INTTM51_on(void);
static void INTTW0UD_on(void);
static void INTTW0CM3_on(void);

static void INTTW0UD_off(void);
static void INTTW0CM3_off(void);

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

static void wait(unsigned int);

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

static void filters(void);
static void current_pwm(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_PORT();          /* 端口设置         */
   init_OSC();           /* 振荡器设置       */
   init_WDTM();          /* WDTM 设置        */
   init_TW0();           /* 逆变器设置       */
   init_TM00();          /* 定时器 TM00 设置             */
   init_RTPM01();        /* 实时输出端口设置             */
   init_AD();            /* A/D 设置                     */
   init_TM50();          /* 定时器 TM50 设置             */
   init_TM51();          /* 定时器 TM51 设置             */

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

   start_TM51();
   INTTM51_on();

   start_AD();           /* 启动 A/D 操作 */
   INTP0_on();           /* 为过流检测设置外部中断 INTP0  */

   EI();
}

/* --------------------------------
   电动机启动处理
  -------------------------------- */
void
motor_start(void) {
   init_openloop();
```

```
    INVERTER_SW     = INV_ON;       /* INVERTER 允许     */

    m_speed         = 0;
    en_1            = 0.0;
    en              = 0.0;
    ec_1            = 0.0;
    ec              = 0.0;
    mvn_f           = 0.0;
    int_cnt         = 0;
    pid_cnt         = 0;
    pid_flag        = CLEAR;
    print_cnt       = 0;
    ad_int_cnt      = 0;
    sys_flag        = FLG_ON;
    start_flag      = FLG_ON;
    stop_wait       = FLG_OFF;
    cw_ccw_wait     = FLG_OFF;
    speed_flag      = FLG_OFF;
    err_flag        = ERROR_NONE;
    startdelay      = 0;

    sync_flag       = FLG_START;
    sync_sw         = 1;
    sync_def        = (unsigned int)sync_def_table[0];
    sync_cnt        = sync_def;
    i_sync          = 0;
    pwm_ff          = pwm_ff_table[0];

    start_TW0();
    start_TM00();
    start_RTPM01();
    INTTW0UD_on();                  /* 取消屏蔽载波同步中断 */
    INTTW0CM3_on();                 /* 取消屏蔽 A/D 触发中断          */

    set_TW0(pwm_base - pwm_ff);     /* 更改 PWM                       */

    set_RTPM01(sync_tbl[cw_ccw_flag][3]);
    wait(2);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(20);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
}

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

/* ---------------------------------
    电动机旋转方向更改处理
  --------------------------------- */
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 操作处理
  --------------------------------- */
void
motor_pid(void) {
    unsigned int tmp, new_tmp, old_tmp;
    static float dl_ref;

    if(speed_flag == FLG_ON){       /* 速度已经被测量 */
        speed_flag = FLG_OFF;
        if(start_flag == FLG_OFF){
            CM3MKW0 = SET;
            tmp = clk_cnt;
            new_tmp = new_clk;
            old_tmp = old_clk;
            CM3MKW0 = CLEAR;

            if(sync_flag == FLG_OFF){
                speed_cnt = (tmp>>2);       /* 30 度移动时间 */
                if(new_tmp < old_tmp){
                    tmp = (0xffff - old_tmp) + new_tmp + 1;
                }else{
                    tmp = new_tmp - old_tmp;
                };
                m_speed = (int)(UNIT_RPM/(long)tmp);
            }else{
                m_speed = (int)(SPEED_CAL_CONST/(long)tmp);
                speed_cnt = (tmp>>2);       /* 30 度移动时间 */
            };

            up_flag = FLG_ON;
```

```
        }else{
            start_flag = FLG_OFF;              /* 第一个速度信息不使用 */
        };
    };

    if(pid_flag == SET) {
        if(pid_cnt >= cy_time){                /* cy_time * 1 ms 已经过去                  */
            pid_cnt = 0;
            if(up_flag == FLG_ON) {            /* 速度被更新                               */
                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)){ /* 不等待反向旋转的停止                       */

                    maxed_flags = 0;                        /* 在这里复位所有标志 */
                    switch(cmd_mode) {
                    case V_CMD:                             /* 电压控制： pwm 由 GUI 设置 */
                        /* 速度上没有闭环 */
                        speed_ref = m_speed;
                        /* 电流上没有闭环 */
                        I_ref = I_measured;
                        /* 调整设定值（加速度限制） */
                        if ((pwm_ff_o - pwm_ff) > dpwm_ff_max) {
                            pwm_ff += dpwm_ff_max;
                            maxed_flags |= 0x20;            /* 达到限制，设置第 5 位 */
                        } else if ((pwm_ff_o - pwm_ff) < -dpwm_ff_max) {
                            pwm_ff -= dpwm_ff_max;
                            maxed_flags |= 0x20;            /* 达到限制，设置第 5 位 */
                        } else {
                            pwm_ff = pwm_ff_o;
                        };
                        break;

                    case I_CMD:                 /* 电流控制 */
                        /* 电流控制模式： GUI 设置的电流控制 */
                        /* 速度上没有闭环 */
                        speed_ref = m_speed;
                        /* 调整设定值（加速度限制） */
                        if ((I_ref_o - I_ref) > dI_ref_max) {
                            I_ref += dI_ref_max;
                            maxed_flags |= 0x40; /* 达到限制，设置第 6 位 */
                        } else if ((I_ref_o - I_ref) < -dI_ref_max) {
                            I_ref -= dI_ref_max;
                            maxed_flags |= 0x40; /* 达到限制，设置第 6 位 */
                        } else {
                            I_ref = I_ref_o;
                        };
                        break;

                    case SPEED_CMD:             /* 控制速度 */
                        /* 调整设定值（加速度限制） */
                        if ((speed_ref_o - speed_ref) > dspeed_ref_max) {
                            speed_ref += dspeed_ref_max;
                            maxed_flags |= (unsigned char)(0x80); /* 达到限制，设置第 7 位 */
                        } else if ((speed_ref_o - speed_ref) < -dspeed_ref_max) {
                            speed_ref -= dspeed_ref_max;
                            maxed_flags |= (unsigned char)(0x80); /* 达到限制，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 */
                        /* 对内环执行电流命令 */

                        dI_ref = (krp_ref*(en - en_1)
                                + kri_ref*en
                                + krd_ref*((en - en_1) - (en_1 - en_2)));
                        WDTE = WDTE_CLR;

                        if (dI_ref > dI_ref_max) {
                            dI_ref = dI_ref_max;
                            maxed_flags |= 0x40;            /* 达到限制，设置第 6 位 */
                        } else {
                            if (dI_ref < -dI_ref_max) {
                                dI_ref = -dI_ref_max;
                                maxed_flags |= 0x40;        /* 达到限制，设置第 6 位 */
                            };
                        };
                        I_ref += dI_ref;
                        break;
                    };
                };
            };
        };
    };
}

/*  --------------------------------
    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 = (unsigned 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
   --------------------------------- */
```

```
static void
system_restart(void) {
    m_speed             = 0;
    en_1                = 0;
    en                  = 0;
    ec_1                = 0.0;
    ec                  = 0.0;
    mvn_f               = 0.0;
    pid_cnt             = 0;
    pid_flag            = CLEAR;
    print_cnt           = 0;
    int_cnt             = 0;
    ad_int_cnt          = 0;
    start_flag          = FLG_ON;
    speed_flag          = FLG_OFF;
    cw_ccw_wait         = FLG_OFF;

    sync_flag           = FLG_START;
    sync_sw             = 1;
    sync_def            = (unsigned int)sync_def_table[0];
    sync_cnt            = sync_def;
    i_sync              = 0;
    pwm_ff              = pwm_ff_table[0];

    I_ref               = I_measured;
    speed_ref           = (int)RPM2;
    startdelay          = 0;

    set_TW0(pwm_base - pwm_ff);                 /* 更改 PWM         */

    set_RTPM01(sync_tbl[cw_ccw_flag][3]);
    wait(2);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);       /* 确定转子位置 */
    wait(20);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);

    WDTE = WDTE_CLR;
}

/* ----------------------------------
   系统停止
  ---------------------------------- */
static void
system_stop(void) {
    pwm_ff = 0;
    INVERTER_SW     = INV_OFF;          /* INVERTER 禁止       */
    INTTW0UD_off();
    INTTW0CM3_off();
    stop_RTPM01();
    stop_TW0();
    stop_TM00();
    sys_flag            = FLG_OFF;
    stop_wait           = FLG_OFF;
    cw_ccw_wait         = FLG_OFF;
    cw_ccw_flag         = CW;
    m_speed             = 0;                /* 速度 0  */
}

/* ----------------------------------
   初始开环设置
  ---------------------------------- */
static void
init_openloop(void) {
    unsigned char jj;
    float               t_def, RPM_def;
    int                 ii;

    ii      = 0;
    t_def = 0;

   /*
     Ii 的循环
     计算频率表如何被使用。
    */
    while (t_def < t_end) {
        WDTE = WDTE_CLR;
        if (t_def < t_knee) {
            RPM_def = RPM0 + ((RPM1 - RPM0)/ t_knee) * t_def;
        } else {
            RPM_def = RPM1 + ((RPM2 - RPM1)/(t_end-t_knee)) * (t_def - t_knee);
        };
        t_def += (float)(5.0/RPM_def); /* 定时器间隔 */
        ii++;
    };

   /* 查找表为 i_sync_skip 次，太小。 */
    i_sync_skip = (int)(ii/TABLESIZE)+1;

    ii = 0;
    t_def = 0;

    while (t_def < t_end) {
        WDTE = WDTE_CLR;

        jj = (unsigned char)(ii / i_sync_skip);
        if (t_def < t_knee) { /* 第一段 */
            RPM_def                 = RPM0 + ((RPM1-RPM0)/(t_knee)) * t_def;
            sync_def_table[jj]      = (int)(CARRIRE_DEF_CONST/RPM_def);
            pwm_ff_table[jj]        = (unsigned int)(PWM0 + ((PWM1-PWM0)/t_knee) * t_def);
            I_ref_table[jj]         = I_ref0 + ((I_ref1-I_ref0)/t_knee) * t_def;
        } else {                    /* 第二段 */
            RPM_def                 = RPM1 + ((RPM2-RPM1)/(t_end-t_knee)) * (t_def - t_knee);
            sync_def_table[jj]      = (int)(CARRIRE_DEF_CONST/RPM_def);
```

```
            pwm_ff_table[jj]       = (unsigned int)(PWM1 + ((PWM2-PWM1)/(t_end-t_knee)) * (t_def - t_knee));
            I_ref_table[jj]        = I_ref1 + ((I_ref2-I_ref1)/(t_end-t_knee)) * (t_def - t_knee);
        };
        t_def += (float)((double)sync_def_table[jj] * 100e-6); /* 下一次 */

        if (jj >= (TABLESIZE-1)){
            t_def = (float)(t_end + 1.0); /* 用于存储器保护—不必需 */
        };

        ii++;
    };

    i_sync_max = ii - 1; /* 表大小 */
}


/* ----------------------------------
   滤波器
 ---------------------------------- */
static void
filters(void) {
    unsigned int x_filt, reg;
    static float u_filt;

    if(sys_flag == FLG_OFF) {
        reg = ADCR;
    }else{
        reg = ad_data_shunt;  /* 10 位 A/D 读取 */
    };
    x_filt = reg>>6;
    I_measured = (float)x_filt;
    WDTE = WDTE_CLR;

}

/*  ----------------------------------
    电流内环 PID
 ---------------------------------- */
static void
current_pwm(void) {
#define STARTLAG 250

    unsigned int     pwm_tmp;

    if (startup_method == CURRENT_START) {
        startdelay = STARTLAG;              /*  由于电流启动计划，不要这样做 */
    } else {
        /* 在传感器上运行 */
        startdelay++;
        if (startdelay > STARTLAG) {
            startdelay = STARTLAG;
        };
    };

    if((sys_flag  != FLG_OFF) &&            /* 被启动 */
       (stop_wait != FLG_ON) &&             /* 不等待电动机停止 */
       (cw_ccw_wait != FLG_ON) &&           /* 不等待电动机停止反向 */
       (cmd_mode != V_CMD) &&               /* 不直接控制 PWM */
       (startdelay == STARTLAG)
       ) {

       /* 对到 PWM 的电流进行 PID 控制 */
        ec_2 = ec_1;  /* 上上个电流误差*/
        ec_1 = ec;      /* 上个电流误差;     */
        ec = I_ref - I_measured;

        mvn_f = kip_ref*(ec - ec_1);
        mvn_f += kii_ref*ec;
        mvn_f += kid_ref*((ec - ec_1) - (ec_1 - ec_2));

       /* 这里对 dpwm 没有限制! */
        pwm_tmp  = pwm_ff + (unsigned int)mvn_f;
        if(pwm_tmp > PWM_F_MAX){ /* limit */
            pwm_tmp = PWM_F_MAX;
        } else if(pwm_tmp < PWM_F_MIN){
            pwm_tmp = PWM_F_MIN;
        };

        pwm_ff = pwm_tmp;

    } else { /* 由于各种原因，不控制电流 */
        I_ref = I_measured;
        speed_ref = m_speed;
        ec = 0;
        ec_2 = 0;
        ec_1 = 0;
    };
}

/* ----------------------------------
   端口设置
 ---------------------------------- */
static void
init_PORT(void) {
    INTP0 = IN;                                 /* 过流通知的中断   */

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

}

/* ----------------------------------
   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;              /* 设置分频率 */
   MCM.0  = 1;                /* X1 输入时钟 */
   VSWC.1 = 1;
}

/* ----------------------------------
   逆变器定时器
   ---------------------------------- */
static void
init_TW0(void) {
   TCL02          =  0;      /* 计数时钟： 20 MHz                    */
   TCL01          =  0;
   TCL00          =  0;
   IDEV02         =  0;      /* 每次产生中断       */
   IDEV01         =  0;
   IDEV00         =  0;
   TW0M           =  0;
   TW0TRGS        =  0;
   TW0OC          =  0;
   TW0CM3         = PWM_BASE_REF;
   TW0CM0         = PWM_BASE_REF - PWM_F_REF;
   TW0CM1         = PWM_BASE_REF - PWM_F_REF;
   TW0CM2         = PWM_BASE_REF - PWM_F_REF;
   TW0DTIME       = PWM_DTM_REF;                          /* 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 */
}

static void
set_TW0(unsigned int duty) {
   TW0BFCM0 = duty;
   TW0BFCM1 = duty;
   TW0BFCM2 = duty;
}

/* ----------------------------------
   16 位定时器

   CR01   实时输出端口触发
   ---------------------------------- */
static void
init_TM00(void) {
   CRC00  = 0x00;             /* 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;             /* 事件计数器模式      */
}

static void
stop_TM00(void) {
   TMC00 = CLEAR;       /* 停止计数器                */
}

/* ----------------------------------
   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                */
}

static void
start_RTPM01(void) {
   RTPC01.7 = SET;                        /* 允许操作 */
}

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

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

   TMIF01 = CLEAR;                        /* 清除中断通知标志 */

}
```

```
/* ----------------------------------
   AD
  ---------------------------------- */
static void
init_AD(void) {
   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;                       /* 允许计数操作           */
}

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

/* ----------------------------------
   等待
   TM50 被使用
  ---------------------------------- */
static void
wait(unsigned int cnt) {
   unsigned int i;

   for(i=0;i<cnt;i++) {
      TMIF50 = CLEAR;
      TCE50 = SET;
      while(TMIF50 != SET);
      TCE50 = CLEAR;
      WDTE = WDTE_CLR;
   };
}

/* ================================================== */
/*
   中断允许/禁止处理
 */
/* ----------------------------------
   过流中断允许
  ---------------------------------- */
static void
INTP0_on(void) {
   PPR0 = 0;                           /* 优先级         */
   EGN0 = 1;                           /* 下降沿         */
   PIF0 = CLEAR;                       /* 清除标志       */
   PMK0 = CLEAR;                       /* 取消屏蔽       */
}

/* ----------------------------------
   载波同步中断允许
  ---------------------------------- */
static void
INTTW0UD_on(void) {
   UDIFW0 = CLEAR;                     /* 清除请求标志          */
   UDMKW0 = CLEAR;                     /* 允许中断              */
}

/* ----------------------------------
   载波同步中断禁止
  ---------------------------------- */
static void
INTTW0UD_off(void)
{
   UDMKW0 = SET;                       /* 禁止中断              */
   UDIFW0 = CLEAR;                     /* 清除请求标志          */
}
```

```
/* ----------------------------------
  A/D 触发中断允许
  ---------------------------------- */
static void
INTTW0CM3_on(void) {
    CM3IFW0 = CLEAR;
    CM3MKW0 = CLEAR;
}

/* ----------------------------------
  A/D 触发中断禁止
  ---------------------------------- */
static void
INTTW0CM3_off(void) {
    CM3MKW0 = SET;
    CM3IFW0 = CLEAR;
}

/* ----------------------------------
    电流内环中断允许
  ---------------------------------- */
static void
INTTM51_on(void) {
    TMIF51 = CLEAR;
    TMMK51 = CLEAR;
}


/* ================================================== */
/*
    中断复位程序
*/
/* ----------------------------------
    硬件过流检测
  ---------------------------------- */
__interrupt void
int_faulta(void) {
    err_flag = ERROR_OC;
    system_stop();              /* 系统停止 */
}

/* ----------------------------------
    载波同步中断（波谷端）
  ---------------------------------- */
__interrupt void
int_carrier(void) {
    static char              cnt_flag;
    static unsigned int  cnt;
    unsigned char         j_sync;

  /* 便利的宏 */
#define  CHG_PAT_LOW()    \
  if(cw_ccw_flag == CW){     \
      switch(++sync_sw){      \
      case 1:  ad_port = AD_CH3; RTBL01 = P_T1; break; \
      case 2:  ad_port = AD_CH1; RTBL01 = P_T6; break; \
      case 3:  ad_port = AD_CH2; RTBL01 = P_T5; break; \
      case 4:  ad_port = AD_CH3; RTBL01 = P_T4; break; \
      case 5:  ad_port = AD_CH1; RTBL01 = P_T3; break; \
      default: sync_sw = 0;                                    \
               ad_port = AD_CH2; RTBL01 = P_T2;           \
      };                                \
   }else{                               \
      switch(++sync_sw){ \
      case 1:  ad_port = AD_CH3; RTBL01 = P_T1; break; \
      case 2:  ad_port = AD_CH2; RTBL01 = P_T2; break; \
      case 3:  ad_port = AD_CH1; RTBL01 = P_T3; break; \
      case 4:  ad_port = AD_CH3; RTBL01 = P_T4; break; \
      case 5:  ad_port = AD_CH2; RTBL01 = P_T5; break; \
      default: sync_sw = 0;                                    \
               ad_port = AD_CH1; RTBL01 = P_T6;           \
      }; \
   };     \
   CR01   = TM00 + CR01_ADD; \
   TMIF01 = CLEAR;

#define  CHG_PAT_HIGH()    \
  if(cw_ccw_flag == CW){    \
      switch(++sync_sw){     \
      case 1:  ad_port = AD_CH3; RTBL01 = P_T1; break; \
      case 2:  ad_port = AD_CH2; RTBL01 = P_T2; break; \
      case 3:  ad_port = AD_CH1; RTBL01 = P_T3; break; \
      case 4:  ad_port = AD_CH3; RTBL01 = P_T4; break; \
      case 5:  ad_port = AD_CH2; RTBL01 = P_T5; break; \
      default: sync_sw = 0;                                    \
               ad_port = AD_CH1; RTBL01 = P_T6;           \
      };                                \
   }else{                               \
      switch(++sync_sw){ \
      case 1:  ad_port = AD_CH3; RTBL01 = P_T1; break; \
      case 2:  ad_port = AD_CH1; RTBL01 = P_T6; break; \
      case 3:  ad_port = AD_CH2; RTBL01 = P_T5; break; \
      case 4:  ad_port = AD_CH3; RTBL01 = P_T4; break; \
      case 5:  ad_port = AD_CH1; RTBL01 = P_T3; break; \
      default: sync_sw = 0;                                    \
               ad_port = AD_CH2; RTBL01 = P_T2;           \
      }; \
   };     \
   CR01   = TM00 + CR01_ ADD; \
   TMIF01 = CLEAR;



  /* 函数开始 */
```

```
    print_cnt++;                                        /* 速度显示时序  */
    if(pid_flag == SET) {
       pid_cnt++;                                       /* PID 控制时序    */
    };

    if(sync_flag == FLG_OFF){               /* 电动机由 BEMF 信号旋转 */
       if(++int_cnt > TIME_OUT){            /* 电动机不旋转 */
          if(stop_wait == FLG_ON){          /* 等待停止 */
             system_stop();
             return;
          }else if(cw_ccw_wait == FLG_ON){              /* 等待反向旋转的停止 */
             pid_flag = CLEAR;
             set_RTPM01(P_STOP);
             wait(10);
             system_restart();
             return;
          }else {
             err_flag = ERROR_MOTOR;
             system_stop();
             return;
          };
       };
       if((stop_wait != FLG_OFF) || (cw_ccw_wait != FLG_OFF)){
          set_RTPM01(P_OFF);
          return;
       };

       if(cnt_flag == FLG_ON){                          /* 30 度移动期间 */
          if(--cnt == 0){                               /* 30 度移动已经完成 */
             int_cnt = 0;
             cnt_flag = FLG_OFF;
             ad_read_flag = FLG_ON;
#ifdef LOW
             CHG_PAT_LOW();
#else
             CHG_PAT_HIGH();
#endif
          }
       }else{
          if(ad_flag == FLG_ON){                        /* 检测过零 */
             ad_flag = FLG_OFF;
             cnt_flag = FLG_ON;
             cnt = (speed_cnt * 7) / 8;
          };
       };
       return;
    };

    if(stop_wait != FLG_OFF){               /* 等待停止 */
       system_stop();
       return;
    };
    if(cw_ccw_wait != FLG_OFF){                         /* 等待反向旋转的停止 */
       system_restart();
       return;
    };

    if(sync_flag == FLG_START){             /* 同步启动期间       */
       if(--sync_cnt == 0){                             /* 切换励磁         */
          speed_flag = FLG_ON;
          ad_read_flag = FLG_ON;
          if(i_sync == i_sync_max){
                /* 经过开环跳转到 PID 控制预处理 */
             sync_flag = FLG_WAIT;          /* 准备同步启动的结束 */
             m_speed   = (int)RPM2;         /* 控制切换期间的转数 */
             cnt_flag  = FLG_OFF;
          }else{
             j_sync   = (unsigned char)(i_sync / i_sync_skip);
             sync_def = (unsigned int)sync_def_table[j_sync];
             if (startup_method == CURRENT_START) {
                I_ref = I_ref_table[j_sync];
             } else {
                pwm_ff = pwm_ff_table[j_sync];
             };
             i_sync++;
          };

#ifdef LOW
          CHG_PAT_LOW();
#else
          CHG_PAT_HIGH();
#endif
          sync_cnt = sync_def;                  /* 设置励磁切换时间 */
       };
    }else{
       if(cnt_flag == FLG_ON){                  /* 等待 30 度过去 */
          if(--cnt == 0){                       /* 切换 */
             sync_flag = FLG_OFF;
             cnt_flag = FLG_OFF;
             ad_read_flag = FLG_ON;
#ifdef LOW
             CHG_PAT_LOW();
#else
             CHG_PAT_HIGH();
#endif
             pid_flag = SET;
          };
       }else{
          int_cnt++;
          if(ad_flag == FLG_ON){                /* 过零发生 */
             speed_cnt = sync_def>>1;
             cnt = speed_cnt;
```

```
            cnt_flag = FLG_ON;              /* 移动 30 度 */
            ad_flag = FLG_OFF;
            int_cnt = 0;
          }else if(--sync_cnt == 0){        /* 同步启动时切换 */
            ad_read_flag = FLG_ON;
#ifdef LOW
            CHG_PAT_LOW();
#else
            CHG_PAT_HIGH();
#endif
            sync_cnt = sync_def;
          }else{
            if(int_cnt > TIME_OUT){         /* 电动机不旋转 */
              err_flag = ERROR_MOTOR;
              system_stop();
              return;
            };
          };
        };
    };
    WDTE = WDTE_CLR;

    return;
}


/* ---------------------------------
  未激活相位电压 A/D 转换
  AD_CH1
  AD_CH2
  AD_CH3
   --------------------------------- */
__interrupt void
int_tw0cm3(void)
{
    unsigned int    set_f;
    unsigned int    ad_data;
    unsigned char adch_backup;

    ad_int_cnt++;

    adch_backup = ADS;    /* 旁路通道 */

    if(ad_read_flag == FLG_ON){
        ADS = ad_port;
        ADIF = CLEAR;
        while(ADIF != SET);
        ad_data = ADCR;      /* 保存值 */
    };

    ADS = AD_ISHUNT;
    ADIF = CLEAR;
    while(ADIF != SET);
    ad_data_shunt = ADCR;                   /* 保存值 */

    ADS = AD_VOL;
    ADIF = CLEAR;

    if(ad_read_flag == FLG_ON){
        if((ADDATA_300 < ad_data) && (ad_data < ADDATA_600)){ /* 300 - 600 */
            if(cw_ccw_flag == CW){
                switch(sync_sw){
                case 0:
                case 2:
                case 4:
                    if(ad_data < ADDATA_E_2){               /* 向下倾斜      */
                        old_clk         = new_clk;
                        new_clk         = TM00;
                        clk_cnt         = ad_int_cnt;
                        ad_int_cnt      = 0;
                        ad_read_flag    = FLG_OFF;          /* 中止 A/D 转换处理 */
                        ad_flag         = FLG_ON;           /* 过零发生 */
                        speed_flag      = FLG_ON;           /* 更新速度信息   */
                    };
                    break;

                case 1:
                case 3:
                default:
                    if(ad_data > ADDATA_E_2){               /* 向上倾斜      */
                        ad_read_flag    = FLG_OFF;          /* 中止 A/D 转换    */
                        ad_flag         = FLG_ON;           /* 过零发生 */
                    };
                };
            }else{
                switch(sync_sw){
                case 1:
                case 3:
                case 5:
                    if(ad_data < ADDATA_E_2){
                        old_clk         = new_clk;
                        new_clk         = TM00;
                        clk_cnt         = ad_int_cnt;
                        ad_int_cnt      = 0;
                        ad_read_flag    = FLG_OFF;
                        ad_flag         = FLG_ON;
                        speed_flag      = FLG_ON;
                    };
                    break;

                case 0:
                case 2:
                default:
```

```
                if(ad_data > ADDATA_E_2){
                    ad_read_flag = FLG_OFF;
                    ad_flag      = FLG_ON;
                };
            };
        };
    };
};

    while(ADIF != SET);
    ad_data_vol = ADCR;

    ADS = adch_backup;    /* 返回通道 */
    ADIF = CLEAR;

    if((stop_wait == FLG_OFF) && (cw_ccw_wait == FLG_OFF)){
        /* 更改 PWM  */
        set_f = pwm_base - pwm_ff;
        TW0BFCM0 = set_f;
        TW0BFCM1 = set_f;
        TW0BFCM2 = set_f;
    };

    WDTE = WDTE_CLR;                        /* 清除看门狗定时器   */

    return;
}

/* ---------------------------------
   TM51 溢出的中断
   --------------------------------- */
__interrupt void
int_TM51(void) {

    filters();

   /* 检查 SOFTWARE 上的过流限制 */
#if 1
    if(I_measured > CLIMIT) {
        err_flag = ERROR_S_OC;
        system_stop();
    };
#endif

    current_pwm();

     }
```

4.17　参考程序源文件

```
/*

   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

   支持模块转换
   电流控制版本

   目标　　：uPD78F0714 电动机入门套件
   日期　　：2007/05/11
   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;
   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(MODE, V_CMD);
   motor_pset(PID_INTERVAL, 150);       /* PID 控制间隔： 150 ms */

   motor_pset(KRP,    50);              /* RPM-Kp 设置。设置由 1000 倍后修改的值                    */
   motor_pset(KRI,    20);              /* RPM-Ki 设置。 设置由 1000 倍后修改的值                    */
   motor_pset(KRD,    10);              /* RPM-Kd 设置。设置由 1000 倍后修改的值                     */

   startup_disp();

   while(1) {
      clear_WDTM();
      vol2speed();
      speed_print(200);
      sw = get_sw();
      if(sys_flag == FLG_ON) {
         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;
            case MODE_TR:
               if(ad_flag == FLG_ON){       /* 函数被停止 */
                  ad_flag = FLG_OFF;        /* 恢复函数   */
               }else{                       /* 函数被操作 */
                  ad_flag = FLG_ON;         /* 停止函数   */
               };
               break;
            default: ;
            };
         };
      };
      tmp_sw = sw;
   };
   return;
}
```

```
/*

   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换)

   支持模块转换
   电流控制版本

   目标　　：  uPD78F0714 电动机入门套件
   日期　　：2007/05/10
   文件名　：sub_mcio.h

   NEC Micro Systems,Ltd
*/

#define  MIN_SPEED        200
#define  MAX_SPEED        3200
```

```
#define  FLG_OFF          0
#define  FLG_ON           1
```

```
#define  CLEAR              0
#define  SET                1

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

#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  REAL               0
#define  REF                1
#define  DOT_REF            2

#define  DOT_OFF            0
#define  DOT_ON             1

#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            /* HALL         */
#define  LED_A              0x88
#define  LED_L              0xc7
#define  LED_               0xff            /* " "          */
#define  LED_S              0x92            /* SELF         */
#define  LED_E              0x86
#define  LED_F              0x8e
#define  LED_P              0x8c            /* PC           */
#define  LED_dot            0x7f            /* .            */

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

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

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

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

```
/*

   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

   支持模块转换
   电流控制版本

   目标      ： uPD78F0714 电动机入门套件
   日期      ： 2007/05/11
   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"
```

```
#if 0
#define  SPEED_MODE   1
#endif

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

unsigned char               ad_flag;

/* ============================================================ */
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
};


/* ============================================================ */

/*
   错误显示
*/
void
print_error(char eno) {
   switch(eno){
   case ERROR_HALL:
      led_set(0, LED_H);
      led_set(1, LED_A);
      led_set(2, LED_L);
      led_set(3, LED_L);
      break;

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

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

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

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

   STOP();
}

/*
   读取 MC-IO 中的开关
*/
unsigned char
get_sw(void) {
   int i;
   unsigned char data, 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; /* 防止 START 和 STOP 拴牢 */
  };

  return(data);
}

/*
  Speed display
*/
void
speed_print(unsigned int ms)
{
#ifdef SPEED_MODE
  if((MODE_SW == SW) || (sys_flag == FLG_OFF)){
    led_print((unsigned int)speed_ref_o, ad_flag);
  }else{
    if(print_cnt > (ms*13)){
      led_print((unsigned int)m_speed, FLG_OFF);
      print_cnt = 0;
    };
  };
#else
  if((MODE_SW == SW) || (sys_flag == FLG_OFF)){
    led_print((unsigned int)pwm_ff_o, ad_flag);
  }else{
    if(print_cnt > (ms*13)){
      led_print((unsigned int)m_speed, FLG_OFF);
      print_cnt = 0;
    };
  };
#endif
}

/*
 用 LED 显示参数值
*/
static void
led_print(unsigned int data, unsigned char dot_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(dot_flag == FLG_OFF){
    led_set(3, led_data[data]);
  }else{
    led_set(3, (unsigned char)(led_data[data])&LED_dot);
  };
}

/*
  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(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;
}

/*
    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);
    }
}

/*
  通过 A/D 找回调控器电阻值
*/
static unsigned int
get_vol(char s_bit) {
    unsigned int  data;
    unsigned char ads_backup;

    if(sys_flag == FLG_OFF){
        DI();
        ads_backup = ADS;
        ADS = AD_VOL;
        ADIF = CLEAR;
        while(ADIF != SET);
        data = ADCR;
        EI();
        ADS = ads_backup;
        ADIF = CLEAR;
    } else {
        DI();
        data = ad_data_vol;
        EI();
    };

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

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

    for(i=0;i<cnt;i++) {
        start_TM50();
        wait_TM50();
        clear_WDTM();
    };
    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);
}

/* ------------------------------------------------------------ */
/*
    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;                 /* 1 ms */
}

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;             /* 停止定时器                           */
}

void
clear_WDTM(void)
{
    WDTE = WDTE_CLR;

}
```

附录A　程序示例

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包含在内。

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 UART 通信中通信数据的读取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()	LED 启动时的显示	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”代替“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	
LD_LED2		PM66	
LD_LED3		PM67	
LD_DATA		PM4	Port that outputs data to LED
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_HALL	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

```
/*


   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）
   支持模块转换
   电流控制版本

   目标      ： uPD78F0714 电动机入门套件
               编译器选项 GUI： GUI 被使用
   日期      ： 2007/04/22
   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;
   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_ON) {
         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;
}
```

```
/*
   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

   支持模块转换
   电流控制版本

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

   日期      ：2007/04/03
   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 FLG_OFF              0
#define FLG_ON               1

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


#define MD_ERROR_HALL 0xF0          /* Hall IC 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        /* 固件鉴定 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_C              0xc6
#define  LED_I              0xcf        /* I            */
#define  LED_H              0x89        /* HALL         */
#define  LED_A              0x88
#define  LED_L              0xc7
#define  LED_               0xff        /* " "          */
#define  LED_S              0x92        /* SELF         */
#define  LED_E              0x86
#define  LED_F              0x8e
#define  LED_P              0x8c        /* PC           */
#define  LED_dot            0x7f        /* .            */

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

```
extern unsigned char get_uart(void);

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

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

```
/*

   BLDCM 无位置传感器的 120 度励磁方法（BEMF_AD 转换）

   支持模块转换
   电流控制版本

   目标      ： uPD78F0714 电动机入门套件
   日期      ： 2007/04/03
   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"


unsigned char uart00_data[RX_BUFF_SIZE];        /* UART00 接收缓存   */
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;

   /* 便利宏定义 */
#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:   /* 开始 */
   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 */
   lo = uart_wait();
   hi = uart_wait();
   ss = ((int)hi<<8) + lo;
   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:
   lo = uart_wait();
   hi = uart_wait();
   ss = ((int)hi<<8) + lo;
   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:
   lo = uart_wait();
   hi = uart_wait();
   ss = ((int)hi<<8) + lo;
   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;
   }
```

```
    lo = (unsigned char)((unsigned int)(ss)&0xff);
    hi = (unsigned char)(((unsigned int)(ss)>>8)&0xff);
    uart_send(lo);
    uart_send(hi);
    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);
    ss = (int)shunt_volt;
    lo = (unsigned char)(ss&0xff);
    hi = (unsigned char)(ss>>8)&0xff;
    uart_send(lo);
    uart_send(hi);
    break;

case MD_CMD_GET_I_CMD:              /* 获取分流 a/d （电流） */
    uart_send(data);
    shunt_command = I_ref*10.0;
    ss = (int)shunt_command;        /* 10 位 （TEMPORARY） */
    lo = (unsigned char)(ss&0xff);
    hi = (unsigned char)(ss>>8)&0xff;
    uart_send(lo);
    uart_send(hi);
    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:               /* 读取旋转速度 */
    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;
                }
            }
```

```
            uart_send((unsigned char)((unsigned int)(ss)&0xff));
            uart_send((unsigned char)(((unsigned int)(ss)>>8)&0xff));
          }
        }
        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);
}

/*
    Returning UART00 receive buffer data
*/
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_OC:
        led_set(0, LED_0 & LED_dot);
        led_set(1, LED_C & LED_dot);
        led_set(2, LED_);
        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:
        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

[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
成都市二环路南三段 15 号天华大厦 7 楼 703 室
电话：(+86)28-8512-5224
传真：(+86)28-8512-5334

[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

[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