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

Application Note

Motor Control by μPD78F0714

Sensorless (A/D Conversion of BEMF) 120° Excitation Method

μPD78F0714

Document No. U18912EJ1V0AN00 (1st edition)
Date Published May 2008 N

[MEMO]

NOTES FOR CMOS DEVICES

① VOLTAGE APPLICATION WAVEFORM AT INPUT PIN

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (MAX) and V_{IH} (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (MAX) and V_{IH} (MIN).

② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

④ STATUS BEFORE INITIALIZATION

Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

- **The information in this document is current as of September, 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "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.
  "Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

INTRODUCTION

Target Readers

This application note is intended for users who understand the functions of the μPD78F0714, and who design application systems that use these functions. The applicable product is shown below.

- μPD78F0714

Purpose

The purpose of this application note is to help the user understand how a motor is controlled via the sensorless (A/D conversion of BEMF) 120° excitation method that uses PWM output and a comparator as a system example of the timer/counter function of the μPD78F0714.

Organization

This application note is divided into the following sections.

- General information
- BLDCM control principle
- System overview
- Control program

How to Read This Manual

It is assumed that the readers of this application note have general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

For details of hardware functions (especially register functions, setting methods, etc.) and electrical specifications

→ See the **μPD78F0714 User's Manual (U16928E)**.

For details of instruction functions

→ See the **78K/0 Series Instructions User's Manual (U12326E)**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right	
Active low representation:	$\overline{xxx}$ (overscore over pin or signal name)	
Memory map address:	Higher addresses on the top and lower addresses on the bottom	
Note:	Footnote for item marked with **Note** in the text	
Caution:	Information requiring particular attention	
Remark:	Supplementary information	
Numeric representation:	Binary ... xxxx or xxxxB	
	Decimal ... xxxx	
	Hexadecimal ... xxxxH	
Prefix indicating the power of 2 (address space, memory capacity):	K (kilo):	$2^{10} = 1,024$
	M (mega):	$2^{20} = 1,024^2$
	G (giga):	$2^{30} = 1,024^3$
Data type:	Word:	32 bits
	Halfword:	16 bits
	Byte:	8 bits

Related documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to the device

Document Name	Document No.
μPD78F0714 User's Manual	U16928E
78K/0 Series Instructions User's Manual	U12326E
Inverter Control by μPD78F0714 120° Excitation Method Control by Zero-Cross Detection Application Note	U17297E
Single-Phase Induction Motor Control by μPD78F0714 Two-Phase Sine Wave Inverter Drive via V/f Control Application Note	U17481E
Motor Control by μPD78F0714 Hall IC 120° Excitation Method Application Note	U18774E
Motor Control by μPD78F0714 Sensorless (BEMF) 120° Excitation Method Application Note	U18051E
Motor Control by μPD78F0714 Hall IC 180° Excitation Method Application Note	U18913E
Motor Control by μPD78F0714 Sensorless (A/D Conversion of BEMF) 120° Excitation Method Application Note	This manual

Documents related to development software tools (user's manuals)

Document Name		Document No.
RA78K0 Ver. 3.80 Assembler Package	Operation	U17199E
	Language	U17198E
	Structured Assembly Language	U17197E
CC78K0 Ver. 3.70 C Compiler	Operation	U17201E
	Language	U17200E
ID78K0-QB Ver. 2.94 Integrated Debugger	Operation	U18330E
PM plus Ver.5.20		U16934E

Documents related to development hardware tools (user's manuals)

Document Name	Document No.
QB-780714 In-circuit Emulator	U17081E
QB-78K0MINI On-Chip Debug Emulator	U17029E
QB-MINI2 On-Chip Debug Emulator with Programming Function	U18371E

Documents related to flash memory writing

Document Name	Document No.
PG-FP4 Flash Memory Programmer User's Manual	U15260E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document for designing.

Other related documents

Document Name	Document No.
SEMICONDUCTOR SELECTION GUIDE - Products and Packages -	X13769X
Semiconductor Device Mount Manual	**Note**
Quality Grades on NEC Semiconductor Devices	C11531E
NEC Semiconductor Device Reliability/Quality Control System	C10983E
Guide to Prevent Damage for Semiconductor Devices by Electrostatic Discharge (ESD)	C11892E

Note See the “Semiconductor Device Mount Manual” website (http://www.necel.com/pkg/en/mount/index.html)

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document for designing.

CONTENTS

CHAPTER 1 GENERAL INFORMATION

This system uses 120° excitation to drive a three-phase star-connected brushless DC motor (hereinafter referred to as “BLDCM”).

- This system uses an NEC Electronics motor starter kit (μPD78F0714)[Note] and uses 120° excitation to drive, without a sensor, a BLDCM.

  The control gain is adjusted in accordance with the motor (no load when started) in the operating environment specified below. Operation when the motor or control program has been changed is not guaranteed.

  Note For the motor starter kit (μPD78F0714), contact an NEC Electronics sales representative.

1.1 Operating Environment

This system (sample program) is created on the assumption that it will be used in the following environment.

- Motor starter kit (μPD78F0714) board set
- Low-voltage inverter set
  BLDCM PITTMAN (N2311A011)
  - Reference voltage [V]: 12
  - No-load rotation speed [r/min]: 7197
  - Continuous torque [Nm]: 0.11
  - Maximum torque [Nm]: 0.23
  - Drive coil: 3 phases (Y connection)
  - Magnetic pole rotor: 4 poles (2 sets of poles)
  - Stator: 6 throttles
  - Position sensor: Hall IC
- PM plus environment platform V5.20
- CC78K0 compiler W3.70
- RA78K0 assembler W3.80
- DF0714.78K device file V1.10

1.2 Related Manuals

For the development environment and board, see the following manuals.

- Low-Voltage Motor Starter Kit Manual
- PM plus Ver. 5.20 User’s Manual
- Each User’s Manual of CC78K0 Ver. 3.70 C Compiler
- Each User’s Manual of RA78K0 Ver. 3.80 Assembler Package

CHAPTER 2 BLDCM CONTROL PRINCIPLE

A BLDCM rotates when its rotor, on which permanent magnets are mounted, rotates by the action of magnetic fields that are generated by the stator coils.

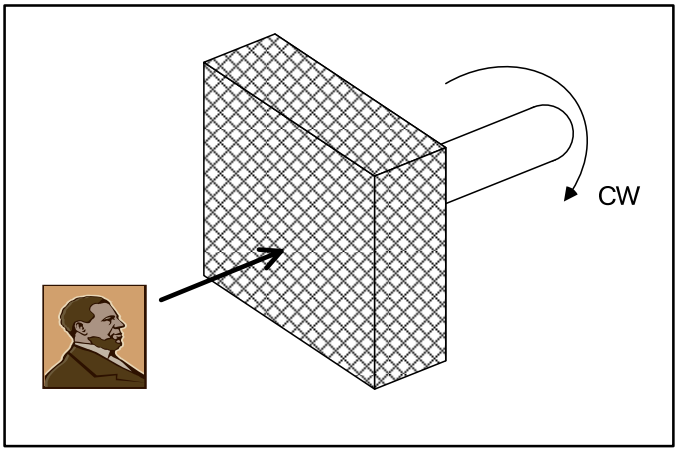
2.1 Rotation Direction Definition

First, the rotation direction of the motor is defined.

The rotation direction of the motor is either CW (clockwise) or CCW (counterclockwise).

CW or CCW is determined based on the direction in which the object to be rotated by the motor is rotated. As shown below, the rotation direction is based on when the surface on which the motor axis is located faces towards the object.

Figure 2-1. Motor Rotation Direction



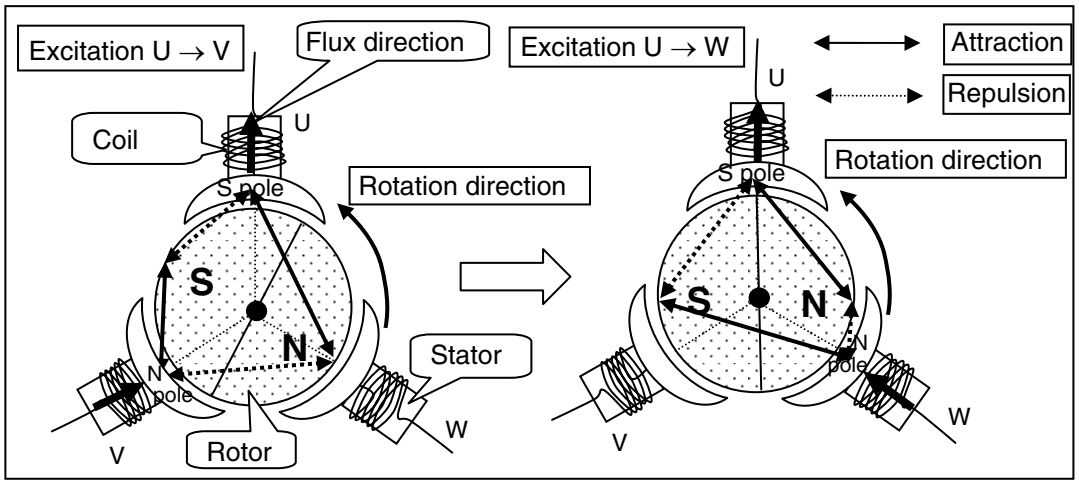
2.2 Rotation Principle

This section describes the BLDCM rotation principle.

The BLDCM shown in the next figure is a 3-phase bipolar 3-slot Y-connection inner-rotor type SPM (Surface Permanent Magnet: a structure having permanent magnets placed onto the surface).

Rotation is caused by the magnetic torque generated by attraction and repulsion of the stator poles and the poles on the permanent magnets of the rotor.

Figure 2-2. BLDCM Rotation Principle



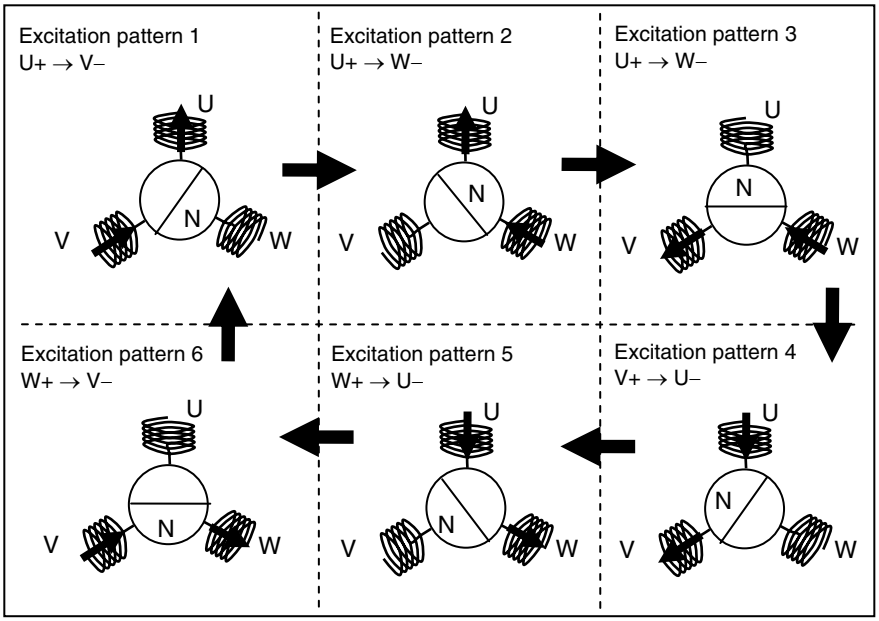
The polarity of the stator depends on the direction of the coiled winding.

When the magnetic polarity of the rotor is reversed, the rotation direction is reversed.

2.3 Excitation Patterns

The following figure shows excitation patterns and the relation between the current flux direction (stator poles) generated by the coil and magnetic polarity of the rotor.

Figure 2-3. Excitation Patterns and Coil Flux Directions



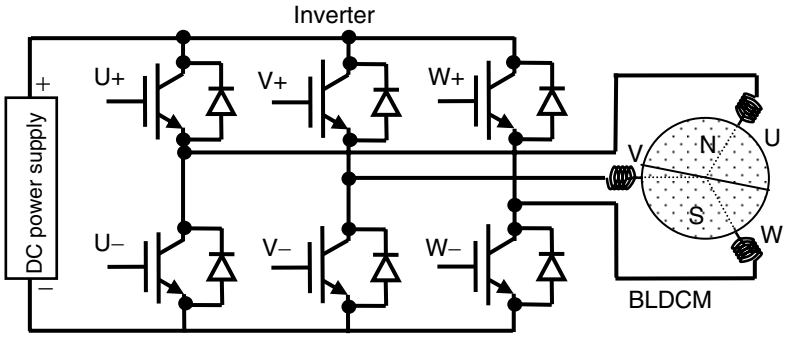
The polarity of the coils depends on the direction of the coiled winding.

2.4 Inverter

A BLDCM, which does not have brushes or a rectifier, uses an inverter to change the direction of the current with respect to the coils.

The following figure shows the connections of the 3-phase Y-connection BLDCM and inverter.

Figure 2-4. Inverter and BLDCM

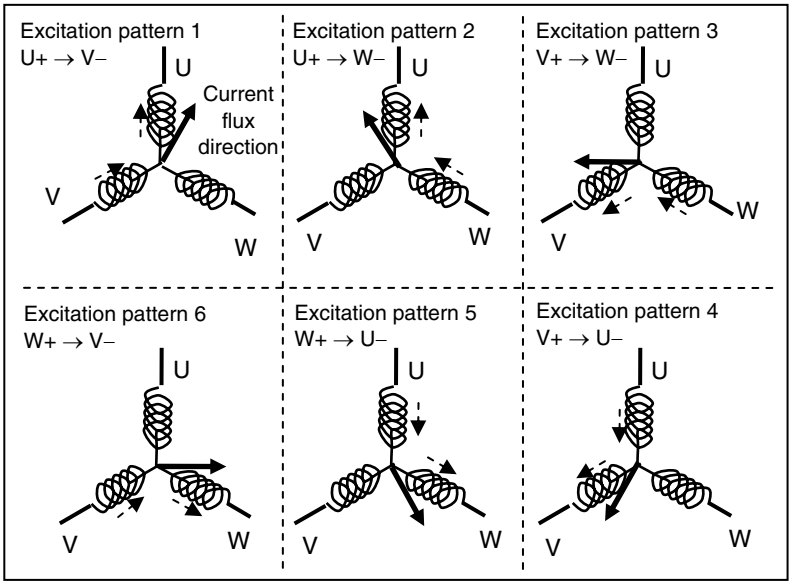


The six switching elements control the excitation time and excitation direction.

2.5 Excitation Patterns

The following figure shows excitation patterns and the current flux direction generated by the stator coil.

Figure 2-5. Excitation Patterns and Current Flux Directions

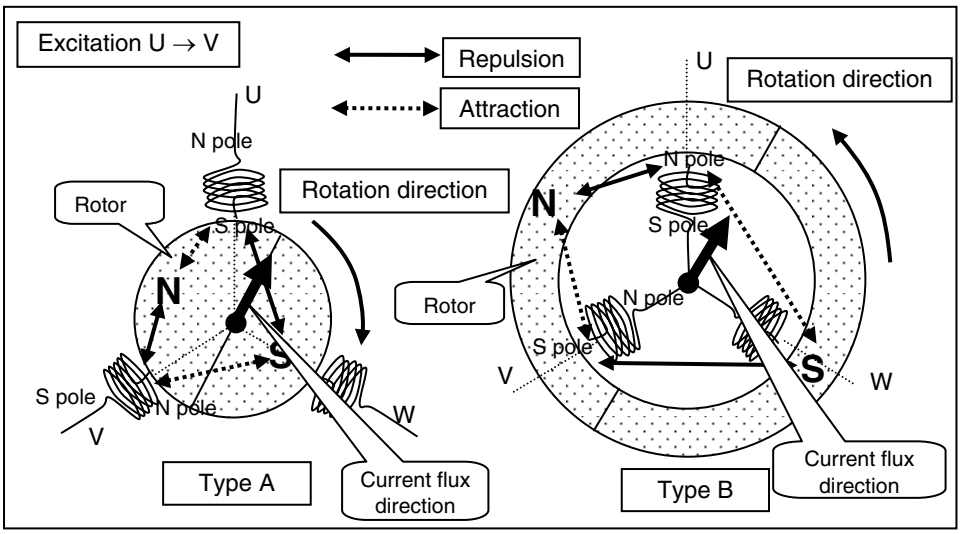


The current flux direction depends on the direction of the coiled winding.

The BLDC motor is caused to rotate by attraction and repulsion of the coil poles generated by the rotating magnetic field and the magnetic poles of the permanent magnets of the rotor. (The rotation direction differs, depending on the positions of the magnetic poles.)

The following figure shows an image of attraction and repulsion of the poles of the rotating magnetic field and the permanent magnets, shown above as excitation pattern 1.

Figure 2-6. Rotation Image



The rotation direction differs, depending on the position of the rotor (permanent magnets).

Usually, one cycle (six excitation patterns) is defined as 360 degrees of electrical angle and one rotation of the motor axis is defined as 360 degrees of mechanical angle (all angles in this document are electrical angles unless otherwise specified).

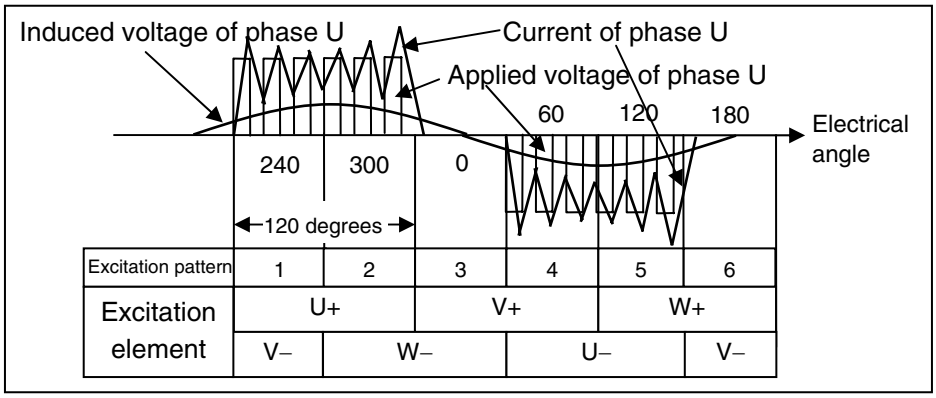
Mechanical angle: Electrical angle/Number of pole pairs
Number of pole pairs: Number of poles of rotor/2

2.6 120° Excitation Method

The following figure shows the applied voltage and induced voltage of phase U when the switching elements of the inverter are switched every 60 degrees by the excitation patterns.

The excitation period of each phase is 120 degrees (120° excitation method) and the BLDC motor is driven by switching as phase U → phase V → phase W, the phases of the upper arm (+) side and lower arm (–) side. (The order in which the phases are switched is reversed during reverse rotation.)

Figure 2-7. 120° Excitation Pattern and Current



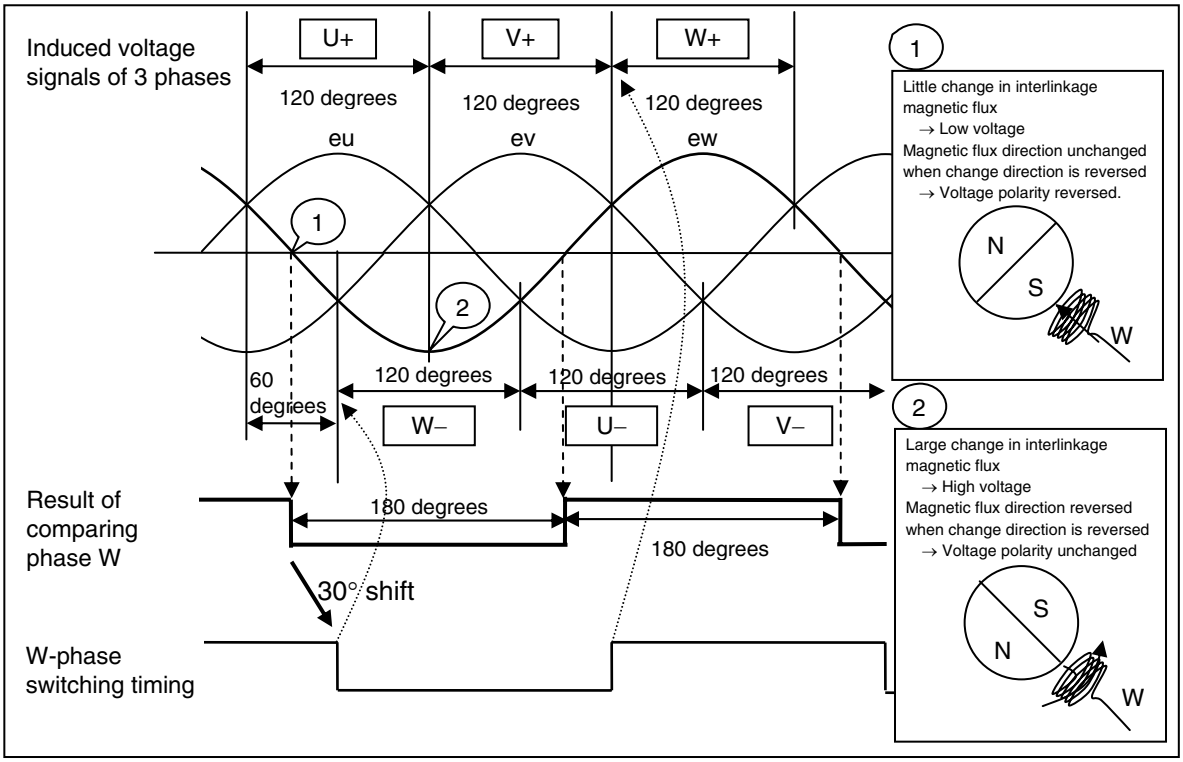
2.7 Position Detection

If the pins of the three phases of a BLDCM are opened and the rotor is rotated from an external source, an induced voltage (eu, ev, or ew) in a sine waveform is generated in each phase. The induced voltage is in proportion to a change per unit time of an interlinkage magnetic flux generated in the stator coil as the rotor magnet rotates, and therefore, the generated voltage indicates the rotation position of the rotor.

With the 120° excitation method, current is allowed to flow through two of the three phases of coils and the coils that are excited are switched every 60 degrees. Therefore, induced voltage (back EMF) can be detected from the open phase that is not excited. For example, in an interval in which excitation occurs from phase U to phase V, a point at which the induced voltage of phase W that is open crosses zero can be detected.

The following figure illustrates the operating principle of estimating the phase positions.

Figure 2-8. Principle of Estimating Phase Positions



A signal shifted by 30 degrees from the result of comparison of phase W matches the timing of switching the excitation pattern of phase W.

2.8 Starting Method

The position detection method using an induced voltage cannot be used when the motor is stopped, in which state no induced voltage is generated, or rotates at a low speed in which state the induced voltage is low. Therefore, the motor is started synchronously so that the excitation pattern is switched independently of the position until the number of revolutions, at which an induced voltage can be detected, is reached and, when the rotation speed of the motor increases, the driving method is changed to one that uses comparison result signals.

2.9 Excitation Pattern Switching

The excitation pattern is switched at a time shifted by 30 degrees from the points the comparison result of each phase has been changed.

2.10 Speed Detection

The rotation speed of the motor can be detected by using the following methods.

- Calculating the rotation speed of the motor by calculating the excitation pattern switching interval (time)
- Calculating the rotation speed of the motor, based on the interval of the zero-cross points reasoned by analogy from the gradient of the induced voltage
- Predicting the rotation speed of the motor from the gradient of the induced voltage

2.11 Speed Control

The rotation speed of the motor is controlled according to the voltage to be applied to the motor.

A PWM (pulse width modulation) voltage control method is used. The method adjusts the conduction rate (average voltage) by performing a chopper operation at a high frequency for a rectangular waveform of 120° excitation, the chopper operation occurring for a conduction period of any of the switching elements.

The PWM conduction rate is changed through PID control.

The interval of controlling the speed is every 150 ms in the initial setting. This can be changed by using a setting change function (PID_INTERVAL parameter of the motor_pset command).

2.11.1 PID control

A PID control is performed by using a deviation between a specified speed and the detected rotation speed to change the conduction rate of the PWM voltage control method (hereinafter referred to as “duty factor”).

A PID control consists of a proportional (P) action that produces an output in proportion to a deviation, an integral (I) action that produces an output in proportion to the integral of the deviation, and a derivative (D) action that produces an output in proportion to the derivative of the deviation.

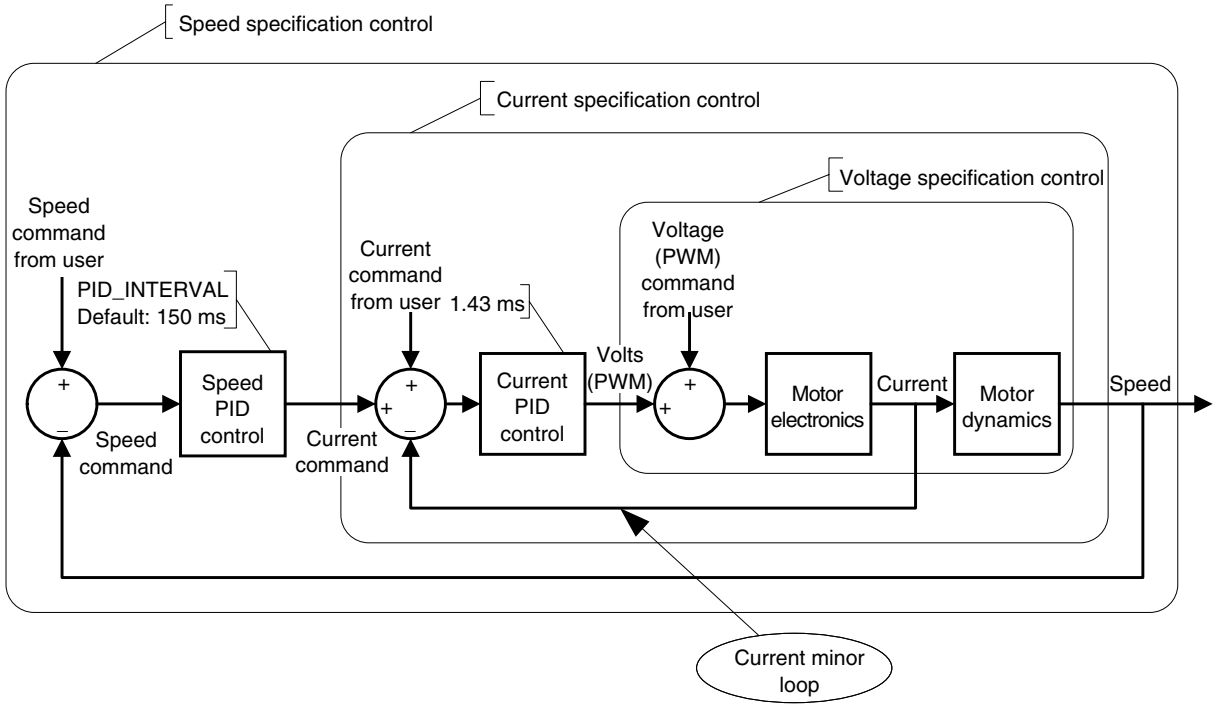
2.12 Current Minor Loop Control

A control for fine adjustment, which is performed within a closed loop by speed control and at an interval shorter than that for controlling the speed, is called a minor loop control. A minor loop control performed for a current is called a current minor loop.

The PWM duty factor is changed through a PID control performed by using the deviation between a reference current and a detected current. PID parameters are separately prepared similarly as for a speed control.

Introducing current minor loop control, enables support of load fluctuation at short control intervals, based on the current value when correction has been performed by speed control.

Figure 2-9. Control Loop



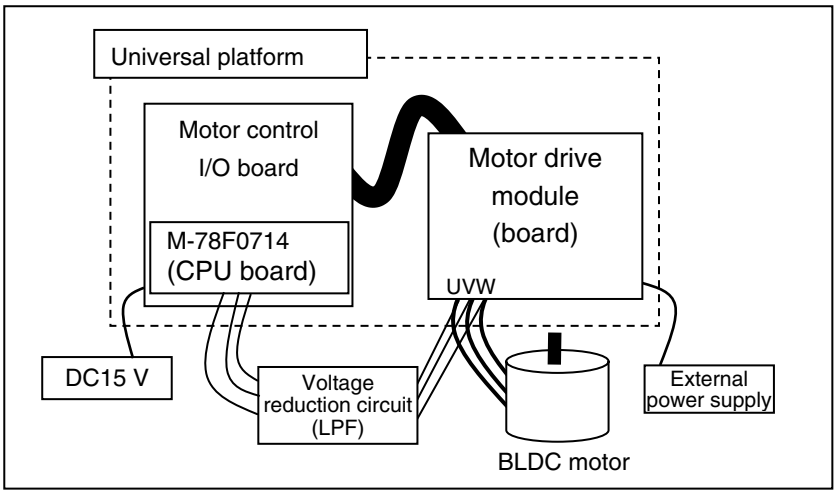
CHAPTER 3 SYSTEM OVERVIEW

This chapter presents an overview of the system.

3.1 Configuration

The following figure shows the configuration of this system.

Figure 3-1. System Configuration

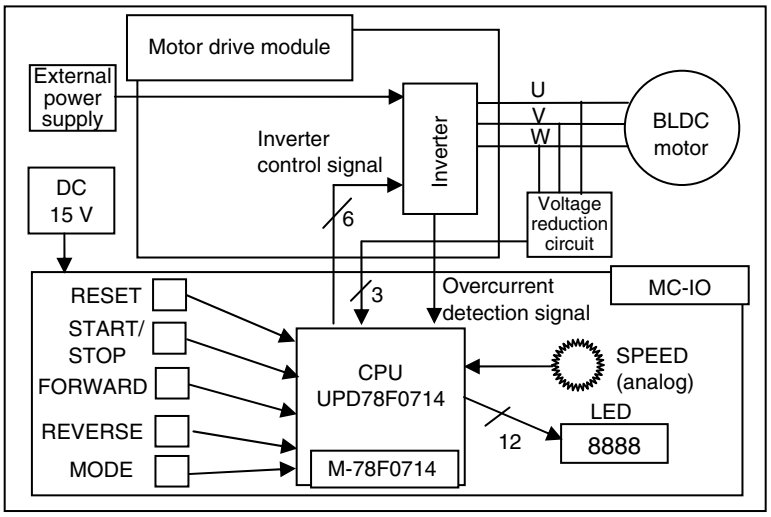


The system consists of a motor drive module that drives the motor, a motor control I/O board equipped with a switch that controls the motor, and M-78F0714 having a CPU.

A BLDCM has three phases and four poles (two pairs of poles).

The block configuration of the low-voltage motor starter kit is shown below.

Figure 3-2. Block Configuration



The motor is controlled by the switch on the motor control I/O board.

3.2 Interface

Table 3-1 lists the user interface functions.

Table 3-1. User Interface

Function Name	Parts Number	Function
RESET	SW1	Reset
START/STOP	SW2	Start/stop
FORWARD	SW3	Rotation direction (Right rotation, clockwise, CW)
REVERSE	SW4	Rotation direction (Left rotation, counterclockwise, CCW)
SPEED	R52	Switching speed specification Switching speed display
8-segment LED	DISP1 DISP2 DISP3 DISP4	Displaying speed (rpm)[Note]

Note The specified speed is always displayed while the motor is stopped.
A dot (“.”) is displayed at the lower right while the specified speed is fixed.
The rotation speed is displayed while the motor is rotating.
The specified speed is displayed while the MODE switch is depressed when the motor rotates.
Reset continues while RESET is depressed and is released when it is released.
“SELF” is displayed for 2 seconds immediately after reset release.

Table 3-2 lists errors.

Table 3-2. Errors

Error	LED Indication	Situation
Overcurrent	0.C.	Inverter current is abnormal.
Software overcurrent	S.0.C.	Motor current is abnormal.
System failure	FAIL	Motor is not rotating.

Table 3-3 lists the interfaces of the μPD78F0714 pin.

Table 3-3. Interface of Pin

Pin Number	Pin Name	Function
8	$\overline{\text{RESET}}$	RESET (SW1)[Note 1]
27 to 32	TW0TO0/RTP10 to TW0TO5/RTP15	3-phase PWM inverter selection
49 to 52	P64 to P67	8-segment LED selection (LD_LED0 to LD_LED3)
56	P73	START/STOP (SW2)
55	P72	FORWARD (SW3)
54	P71	REVERSE (SW4)
53	P70	MODE (SW5)
41 to 48	P40/RTP00 to P47/RTP07	Output data to 8-segment LED
12	TW0TOFFP/INTP0/P00	Overcurrent detection (+5 V → 0 V)
60	P24/ANI4	Speed change (R52)
59	P25/ANI5	ISHUNT current
21	P54/TI001/TO00	Motor drive module control
57, 58, 61	P27/ANI7, P26/ANI6, P23/ANI3	Coil voltage measurement[Note 2]

Notes 1. Shorts out the M-78F0714 board and 1-2 of 2JP7.

2. Inputs the pin voltage of each phase boosted up to 5 V.

3.3 Functions

Table 3-4 lists the functions and operation overview of this system.

Table 3-4. System Functions and Operation Overview (1/2)

Function	Overview
Start (power supply)	"SELF" is displayed with LEDs for 2 seconds. Specified speed (rpm) of SPEED volume is displayed with LEDs.
RESET switch	System is restarted regardless of the status of motor control.
START/STOP switch	While motor control is stopped: Motor control is started. "0" is displayed with LEDs. Motor starts rotating clockwise. Motor revolution speed (rpm) is displayed with LEDs.
	While motor is controlled: Motor control is stopped. Motor revolution speed (rpm) is displayed with LEDs until it reaches 0. Specified speed (rpm) of SPEED volume is displayed with LEDs.
	START and STOP do not toggle even if this switch is depressed.
FORWARD switch	While motor control is stopped: Does not function. Does not change.
	While motor is controlled: Rotation direction is changed. If rotation direction is CCW, it is changed to CW after being stopped once. If rotation direction is CW, it is not changed.
REVERSE switch	While motor control is stopped: Does not function. Does not change.
	While motor is controlled: Rotation direction is changed. If rotation direction is CCW, it is not changed. If rotation direction is CW, it is changed to CCW after being stopped once.
MODE switch	While motor control is stopped: Speed specification is switched. Change of specified speed by SPEED volume is enabled or disabled. When disabled, a dot (".") is displayed to the lower right of the LED value.
	While motor is controlled: Changes specified speed. Specified speed (rpm) of SPEED volume is displayed with LEDs while this switch is depressed.
SPEED volume	While motor control is stopped: Changes specified speed. Speed (rpm) displayed with LEDs changes.
	While motor is controlled: Changes specified speed. Speed specified to motor is displayed with LEDs.
Hardware overcurrent	Occurs if permissible current of motor drive module is exceeded. Motor control is stopped. "0.C." is displayed with LEDs. Functions other than RESET switch are disabled.
Software overcurrent	Occurs if permissible current of motor is exceeded. Motor control is stopped. "S.0.C." is displayed with LEDs. Functions other than RESET switch are disabled.

Caution The operation is not guaranteed if two or more switches are pressed at the same time. Perform reversal by using the FORWARD or REVERSE switch at no more than 300 rpm.

Table 3-4. System Functions and Operation Overview (2/2)

Function	Overview
System abnormality	Occurs when motor control is abnormal. • Motor does not start rotating within two seconds after START switch is pressed. • Induced voltage is abnormal (not switched after synchronous start). • Motor has stopped rotating for at least one second due to a sudden increase in load. • Power supply from motor drive module to motor is stopped. Motor control is stopped. “FAIL” is displayed with LEDs. Functions other than RESET switch are disabled.

Caution The operation is not guaranteed if two or more switches are pressed at the same time. Perform reversal by using the FORWARD or REVERSE switch at no more than 300 rpm.

3.4 Peripheral I/Os

This system uses the following peripheral I/Os.

Table 3-5. Peripheral I/Os

Function	Peripheral I/O Function Name (μPD78F0714)
Inverter timer	For PWM output 10-bit inverter control timer (TW0UDC, etc.) Carrier (modulation) frequency is 10 kHz (symmetrical triangular wave). Carrier (modulation) synchronization interrupt is generated at intervals of 100 μs (duty factor is calculated by main PID control action and updated by carrier synchronization interrupt).
Real-time output	For changing excitation pattern Real-time output port (RTBH01, RTBL01, etc.) 16-bit timer capture/compare register 01 (CR01) (Excitation pattern is changed by carrier synchronization interrupt.)
Timer for acquiring ISHUNT current value	For timing adjustment 8-bit timer/event counter 51 (TM51, CR51) Interrupts are generated at 1.43 ms intervals and ISHUNT current measurement function is performed.
Wait processing	For timing adjustment 8-bit timer/event counter 50 (TM50, CR50) Interrupt request flag is set in 1 ms intervals.
Acquiring specified speed	Converts voltage on variable resistor value into specified speed. A/D converter (ANI4) (Updated by main switch read processing.)
Acquiring ISHUNT current value	Acquires voltage value from pin (ANI5) by converting voltage of motor operating current.
Reading pin voltage	Converts voltage of phases U, V, and W. Processed by carrier synchronization (PWM output midpoint) interrupt. A/D converters (ANI3, ANI6, ANI7)
H/W overcurrent interrupt	Interrupt function (INTP0) Occurrence of overcurrent in motor drive module (low-active)
Fail safe	Watchdog timer

3.5 Interrupt

Table 3-6 shows the interrupts used in this system.

Table 3-6. Interrupts Used

Name	Function	Condition of Occurrence
INTP0	Detection of overcurrent occurrence	External pin
INTTW0UD	Occurrence of carrier synchronization interrupt	Underflow of inverter timer counter
INTTW0CM3	Occurrence of carrier synchronization interrupt	Underflow of inverter timer counter
INTTM51	Acquisition of ISHUNT current	Overflow of 8-bit timer counter
RESET	Occurrence of reset	$\overline{\text{RESET}}$ pin
WDT	Occurrence of internal reset	Watchdog timer overflow due to program runaway

Remark INTTM50 and INTAD are processed with polling of the interrupt request flag.

CHAPTER 4 CONTROL PROGRAM

This system uses a basic control program to actually control the speed of a motor.

4.1 Compiler Options

The two types of inverters that can be controlled by using this control program can be switched by using options when compiling[Note].

Table 4-1. Compiler Options

Option Name	Function
LOW	When using PITTMAN motor
None[Note]	When using another motor

Note LOW must be set, because a PITTMAN motor is used in this control program.

4.2 Excitation Pattern

The excitation pattern is selected by confirming the position of the rotor according to the induced voltage of each phase.

The excitation pattern is switched after a rotation time equivalent to 30 degrees has elapsed from the zero-cross point of the induced voltage.

At least two (200 μs) carrier synchronization interrupts are required at the rotation time equivalent to 60 degrees, because excitation switching processing is performed by carrier synchronization interrupts. (For a 3-phase bipolar pair, a theoretical value up to 25,000 rpm can be controlled.)

During reverse rotation, the excitation direction (excitation pattern) corresponding to the position of the rotor is reversed.

The assumed connections of this control program are described below.

4.2.1 Low-voltage inverter set

Table 4-2. BLDCM Pin Specifications

Color	Function	Remark
BROWN	MOTOR ϕA	Phase U
RED	MOTOR ϕB	Phase V
ORANGE	MOTOR ϕC	Phase W

4.3 Excitation Pattern Switching

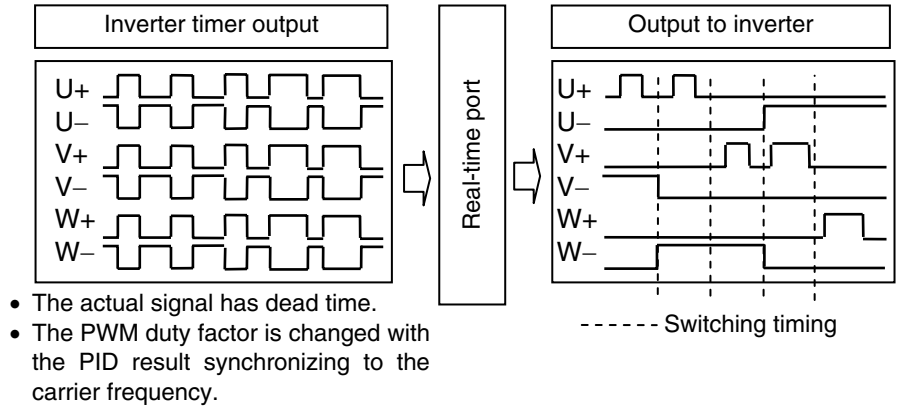
See **4.6 Starting Method** for excitation pattern switching when starting the motor.

Normally, the excitation pattern is switched to the next one every time a time equivalent to 30 degrees has elapsed (the time required for a 30-degree rotation is calculated by the number of times carrier synchronization interrupts have occurred).

The real-time output port function is used to switch the excitation pattern.

PWM control for a non-complementary operation of the entire upper-arm area is performed by setting the upper arm (+) side and lower arm (–) side of the 10-bit inverter control timer output to “through” or “off”, and “on” or “off”, respectively.

Figure 4-1. Excitation Pattern Switching Timing



- The actual signal has dead time.
- The PWM duty factor is changed with the PID result synchronizing to the carrier frequency.

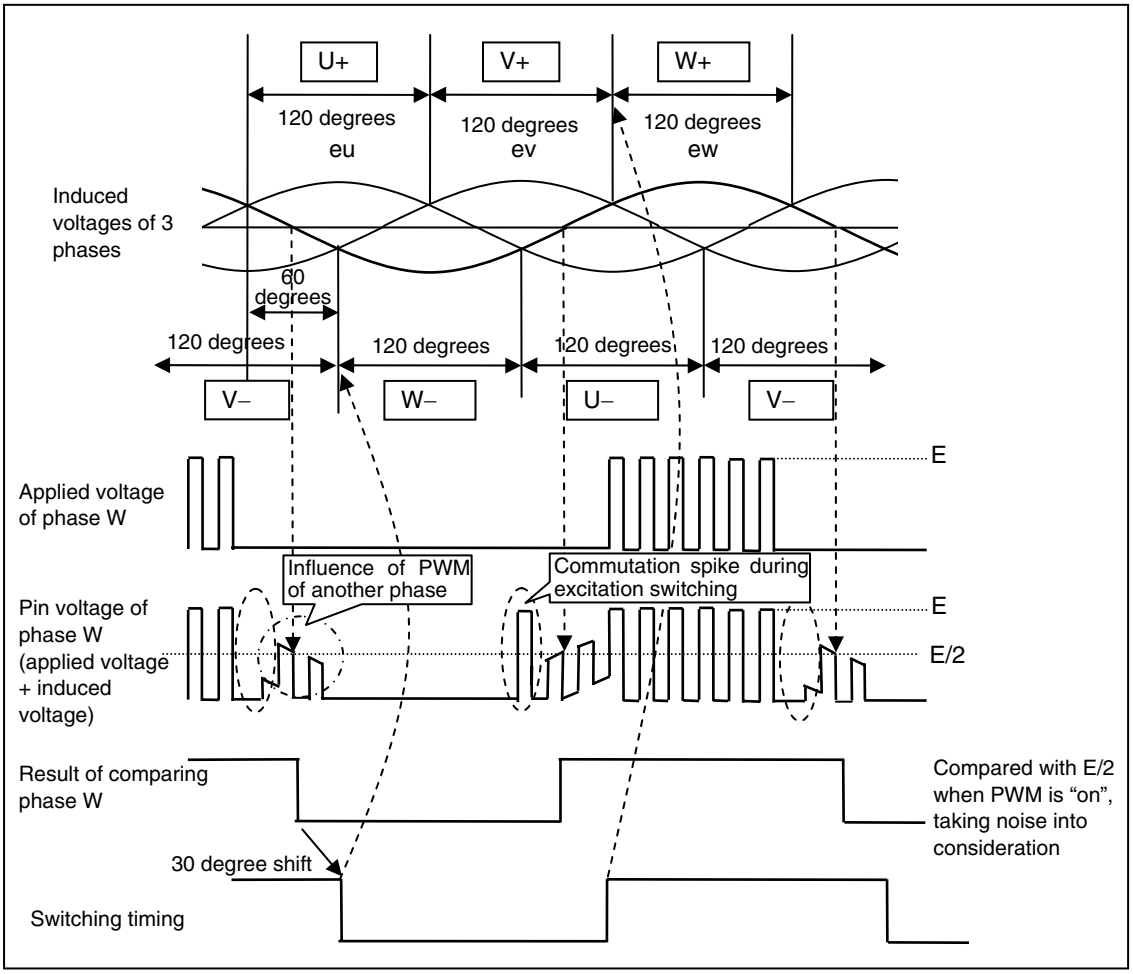
4.4 120° Excitation Method

When the motor is started synchronously, the excitation pattern is switched by using the 120° excitation method, regardless of the position of the rotor, and from a speed for which the position of the rotor can be predicted from the induced voltage, the 120° excitation method is performed according to the predicted position.

4.5 Position Detection

The procedure for generating a comparison signal from the pin voltage of phase W is shown below.

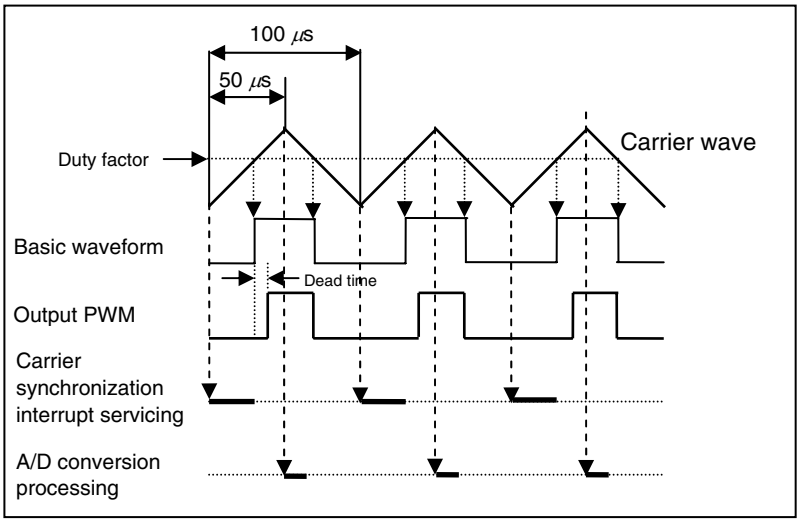
Figure 4-2. Switching Position Detection Method



If the induced voltage of phase W is 0, the neutral point where applied voltage E is equally divided for the phase U winding and phase V winding is E/2, and the pin voltage of phase W which is not excited is also E/2.

The A/D conversion timing of the pin voltage is shown below.

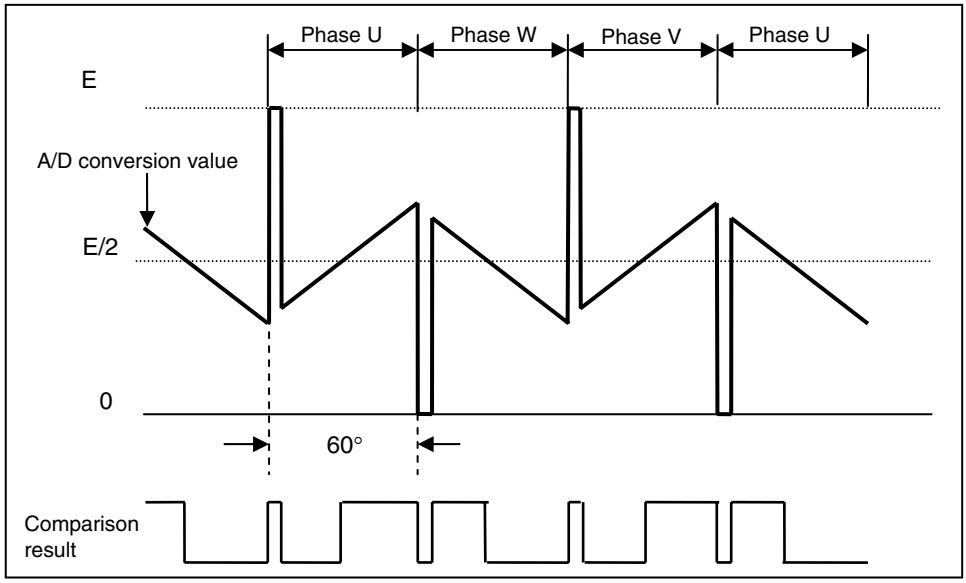
Figure 4-3. A/D Conversion Timing



The voltage of a non-excited pin is A/D converted by servicing of interrupts generated at the peaks of the carrier wave and the converted value is compared with E/2. Dead time is used to adjust the timings of A/D conversion and output PWM. The A/D conversion time is assumed to be 3.6 μs.

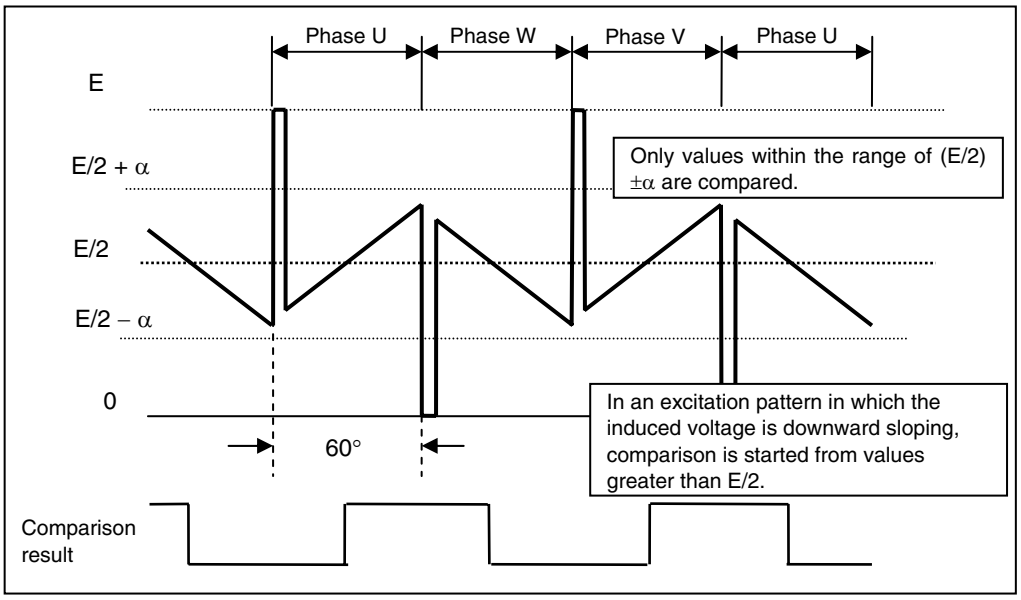
The result of comparing the voltage of the non-excited pin and E/2 includes unnecessary signal components due to the influence of noise generated by commutation spikes.

Figure 4-4. A/D Conversion of Non-Excited Pin and Comparison Result



Noise generated by commutation spikes can be removed from the comparison result by comparing only values close to E/2, adjusting the comparison start position by taking the gradient of the induced voltage from the excitation pattern into consideration, or other methods.

Figure 4-5. Noise Removal Example



In this system, noise was removed by comparing only the values within the range of (E/2) $\pm\alpha$.

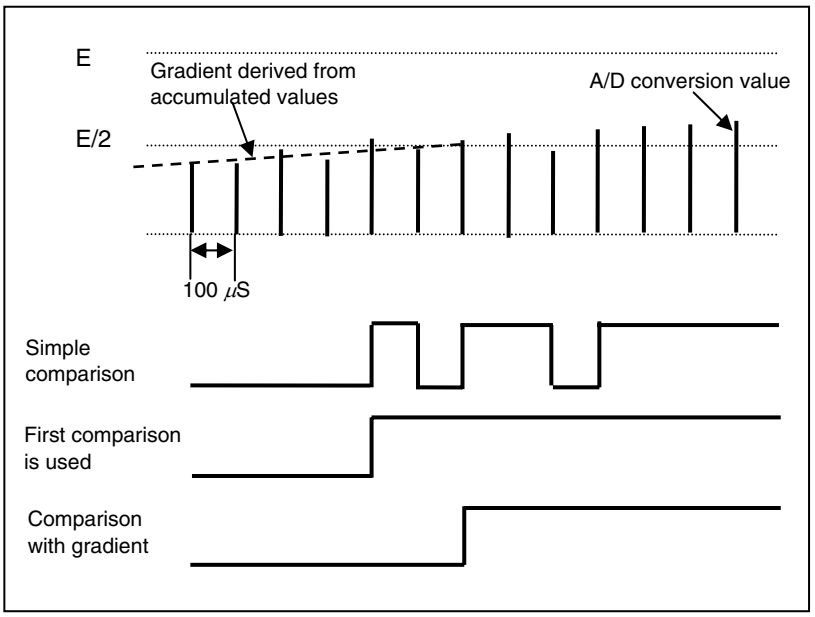
In this control program, constants are defined as follows.

E/2 – α:	ADDATA_300	Conversion value 300 (about 30%)
E/2 + α:	ADDATA_600	Conversion value 600 (about 60%)
E/2:	ADDATA_E_2	Conversion value 464 (about 46%)

If the induced voltage is low due to low rotation, noise is generated from A/D conversion errors.

Noise can be prevented by not performing comparison until the phase switches to the next non-exited phase once the comparison result has changed, comparing with the gradient derived by calculating the value of the accumulated A/D conversion values, or other methods.

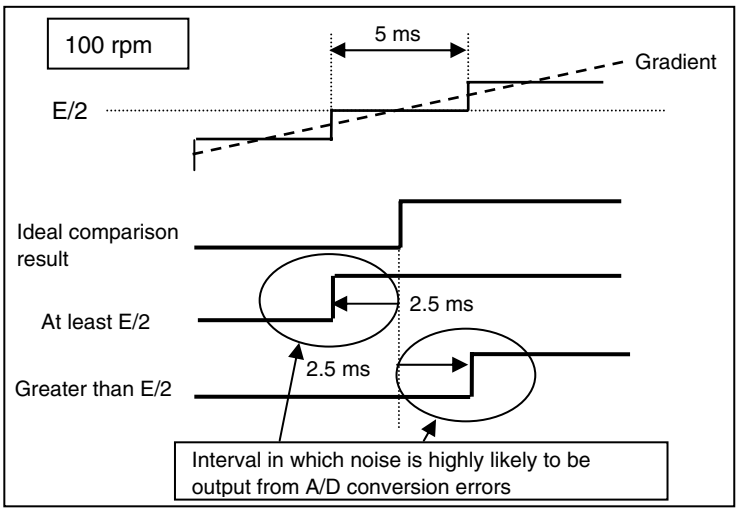
Figure 4-6. Handling of Noise During Low Rotation



Noise is prevented by not performing comparison until the phase switches to the next phase once the comparison result has changed, because the CPU of this system does not have the processing capability to accumulate the A/D conversion values and calculate the gradient at a high speed.

In this system, the voltage applied to the motor (15 V) is A/D converted (1 bit is about 0.015 V) in 10 bits (1,024). If the induced voltage at 100 rpm from the back EMF constant of the motor is 0.16 V, the number of carrier synchronization interrupts occurring at 100 rpm and at a rotation time equivalent to 60 degrees is 500, the number of 0.08 or 0.015 values is about 5 (10 within the range of ±0.08) because the voltage of an induced voltage of 0.16 V during a non-excited period is ±0.08 V, and the A/D conversion value includes no error, the A/D conversion value is changed every 50 times. (At least 100 units of data are required to calculate the gradient.)

Figure 4-7. Error due to Comparison Condition

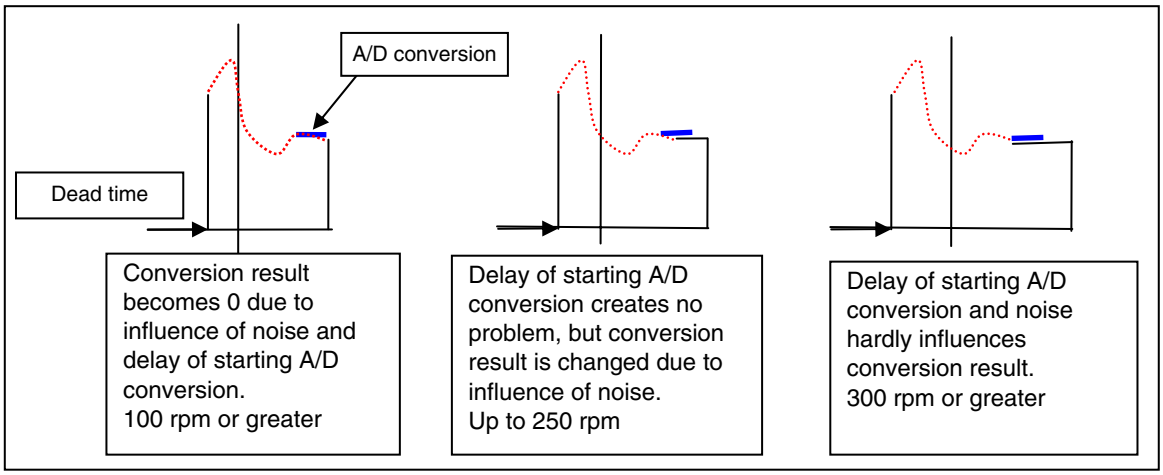


In this system, comparison is performed with “Greater than E/2”.

The A/D conversion result is influenced by a misalignment of up to 25 clocks (1.25 μs) that occurs when receiving an interrupt request with the CPU of this system.

The following states could be confirmed from the result of driving the BLDC motor by using Hall ICs and performing A/D conversion.

Figure 4-8. Number of Revolutions and A/D Conversion Result

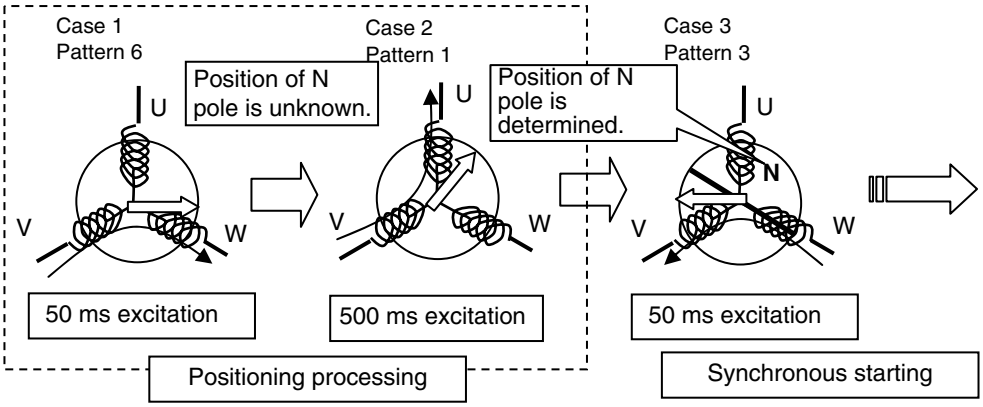


The lower-limit speed of this system was set to 300 rpm according to the above results.

4.6 Starting Method

The position of the rotor is forcibly determined by exciting two specific phases and the excitation pattern is sequentially switched (see **Figure 2-3 Excitation Patterns and Coil Flux Directions**), to increase the period (rotation speed). After the specified time has elapsed, the control is switched to control by an excitation pattern using the BEMF signal.

Figure 4-9. Switching Excitation Pattern on Starting



Case 1: Preparatory operation to prevent the rotor from not rotating (N pole does not move) if the N pole of the rotor is at the opposite side to the current flux direction in Case 2

Case 2: Forcibly moves the N pole of the rotor.

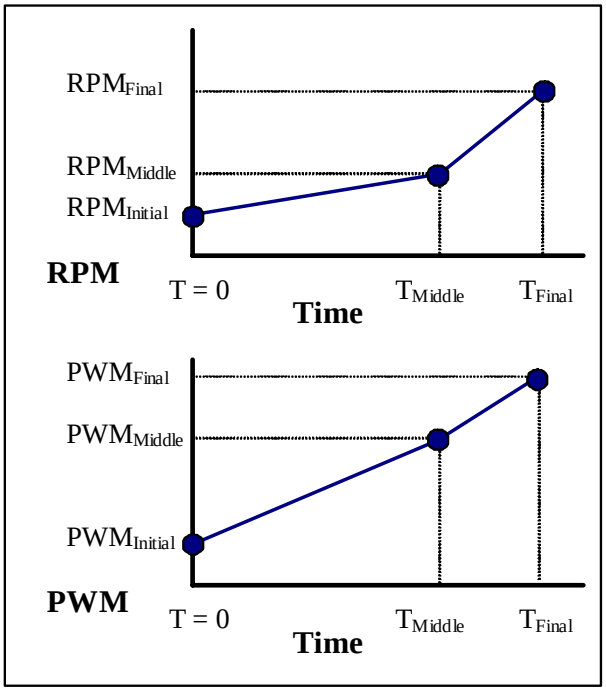
Case 3: Starts from an excitation pattern in which the direction of the current flux is at 60 to 120 degrees from the pole position of the rotor.

To achieve highly flexible starting, the starting characteristics shown in the following graphs can be executed. The RPM value and PWM value can each be specified immediately after starting (T = 0), after a given time has elapsed (T_{Middle}), or for the time that elapses when shifting to control by the BEMF signal (T_{Final}).

Table 4-3. Parameters that Can Be Specified

Time Elapsed	No. of Revolutions	PWM Duty Factor
T = 0	$RPM_{Initial}$	$PWM_{Initial}$
T_{Middle}	RPM_{Middle}	PWM_{Middle}
T_{Final}	RPM_{Final}	PWM_{Final}

Figure 4-10. Starting Characteristics Graphs



4.7 Speed Detection

The intervals of switching the excitation pattern are calculated from the number of carrier synchronization interrupts (100 μs cycles) that have occurred (furthermore, half of the number of interrupts having occurred is used as the value of a 30-degree shift), because the processing capability of the CPU of this system is not sufficient to control the motor while calculating the gradient of the induced voltage.

The speed is detected after rotation has been started according to the comparison result of A/D conversion.

The speed is calculated based on the time while the excitation pattern is switched two times (six times in one rotation), because the error is too large if the speed is calculated according to the value measured every time the excitation pattern is switched (twelve times in one rotation).

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

N: Number of revolutions per minute (*rpm*)

s: Resolution (100 μs)

n: Number of interrupts occurred

6/2: Number of times measured at an electrical angle of 360 degrees

2: Number of pole pairs

4.8 Speed Control

The rotation speed of the motor is controlled by adjusting the PWM duty factor to the 150 ms period.

The variable of the duty factor to be adjusted is derived by performing a PID control operation on the difference between the specified speed and motor rotation speed.

The value is generated as a reference current for a current minor loop, instead of directly operating PWM for current minor loop control.

4.8.1 PID operation

The speed is adjusted by feeding back the difference between the rotation speed and the specified speed and performing a PID control operation on the PWM duty factor (average voltage). The duty factor manipulated variable of PWM uses the following speed type PID algorithm suitable for the sampling method (discrete value).

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

MV_n: Current manipulated variable
MV_{n-1}: Previous manipulated variable
$\triangle MV_n$: Difference between current and previous manipulated variables
e_n: Current deviation (difference between specified speed and actual speed)
e_{n-1}: Previous deviation
e_{n-2}: Deviation before previous deviation
Kp: Feedback gain (proportional element)
Ki: Feedback gain (integral element)
Kd: Feedback gain (derivative element)

The optimum values of the feedback gains change depending on the motor characteristics and on the presence or absence of a load.

With this system, a value obtained by cut-and-try when checking the operation is set as the default value.

4.9 Current Minor Loop Control

The number of revolutions of the motor is controlled by adjusting the PWM duty factor in a 1.43 ms cycle.

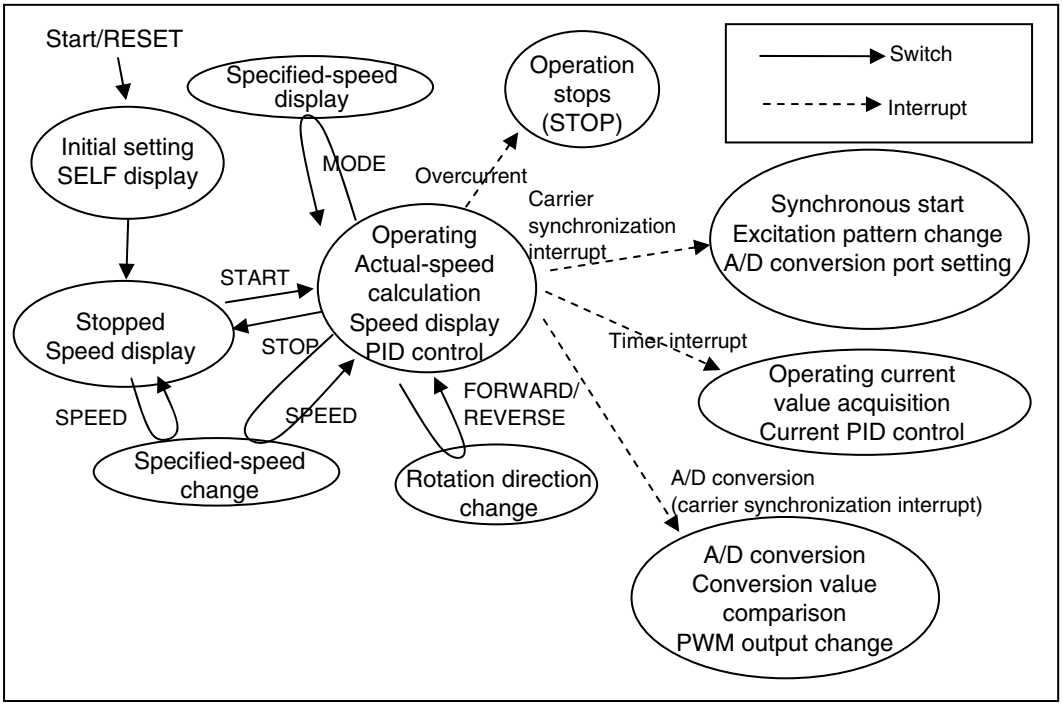
The variable of the duty factor to be adjusted is derived by performing a PID control operation on the difference between the reference current and operating current of the motor.

Similarly as for speed control, the PWM duty factor will be manipulated by performing a PID control operation on the difference in the measured operating current of the motor, for the reference current generated by speed control.

4.10 Module Configuration

Figure 4-11 shows status transition of the system.

Figure 4-11. Status Transition of System



A carrier synchronization interrupt is generated and an A/D conversion occurs alternately in 50 μs intervals.

4.11 Function List

The control program consists of many functions. The following tables list these functions and their features. For details of processing, see the flowcharts.

4.11.1 Functions usable by users

Table 4-4. Functions

Function Name	Function	Purpose
motor_init()	Initial settings	Performs initialization required for motor control, such as register setting for each function and interrupt setting.
motor_start()	Start instruction	Instructs start of motor rotation.
motor_stop()	Stop instruction	Instructs stopping of motor rotation.
motor_rotation()	Rotation direction instruction	Instructs the direction of motor rotation.
motor_pid()	Speed control PID operation	Instructs PID operation for speed control.
motor_pset()	Parameter setting	Sets the parameters required for motor control.

4.11.2 Motor library internal functions

Table 4-5. Motor Library Internal Functions (1/2)

Function Name	Function	Purpose
system_restart()	Restart processing	Restarts after stopping reverse rotation.
system_stop()	Stop processing	Stops the system.
init_openloop()	Start sequence setting processing for initial start	Calculates the parameters for the start sequence during initial start.
filters()	Digital filter processing	Performs digital filter processing of an acquired current value.
current_pwm()	PID operation processing for current minor	Performs PID operation for a current minor.
init_PORT()	Port setting processing	Sets the port to be used for motor control.
init_OSC()	Clock setting processing	Performs clock setting for CPU operating speed.
init_TW0()	Inverter setting processing	Sets the inverter timer.
start_TW0()	Inverter start processing	Starts the inverter timer.
stop_TW0()	Inverter stop processing	Stops the inverter timer.
set_TW0()	PWM setting processing	Sets PWM.
init_TM00()	TM00 setting processing	Sets the 16-bit timer.
start_TM00()	TM00 start processing	Starts the 16-bit timer.
stop_TM00()	TM00 stop processing	Stops the 16-bit timer.
init_RTPM01()	Real-time output port setting processing	Sets the real-time output port.
start_RTPM01()	Real-time output port start processing	Starts the real-time output port.
stop_RTPM01()	Real-time output port stop processing	Stops the real-time output port.
set_RTPM01()	Excitation pattern switching processing	Sets the real-time output port output value and performs output switching processing.
init_AD()	A/D setting processing	Sets A/D conversion.

Table 4-5. Motor Library Internal Functions (2/2)

Function Name	Function	Purpose
start_AD()	A/D start processing	Starts A/D conversion.
init_WDTM()	Watchdog timer setting processing	Sets the watchdog timer.
init_TM50()	TM50 setting processing	Sets (1 ms) the 8-bit timer.
wait()	Wait	Waits for a specified time.
init_TM51()	TM51 setting processing	Sets (1.43 ms) the 8-bit timer.
start_TM51()	TM51 start processing	Starts the 8-bit timer.
INTP0_on()	INTP0 enable processing	Enables overcurrent interrupts.
INTTW0UD_on()	INTTW0UD enable processing	Enables carrier synchronization (valley) interrupts.
INTTW0UD_off()	INTTW0UD disable processing	Disables carrier synchronization (valley) interrupts.
INTTW0CM3_on()	INTTW0CM3 enable processing	Enables carrier synchronization (crest) interrupts.
INTTW0CM3_off()	INTTW0CM3 disable processing	Disables carrier synchronization (crest) interrupts.
INTTM51_on()	INTTM51 enable processing	Enables timer interrupts for a current minor loop.
int_speed()	Interrupt servicing for speed information acquisition	Sets a flag urging updating of speed information.
int_faulta()	Overcurrent interrupt servicing	Stops motor processing.
int_carrier()	Carrier interrupt (valley) servicing	Switches the excitation pattern, updates the duty factor, and diagnoses the motor operating state.
int_tw0cm3()	Carrier interrupt (crest) servicing	A/D converts the voltage of a non-excited phase, updates speed information, and performs PWM change processing.
int_TM51()	Timer interrupt servicing for current minor loop	Performs operation processing for a current minor loop.

4.11.3 Reference program functions

Table 4-6. Reference Program Functions

Function Name	Function	Purpose
print_error()	Error display	Displays the error status with the LEDs.
get_sw()	Switch read instruction	Instructs reading of switch information on the MC-IO board and returns operation information corresponding to the switch read.
speed_print()	Speed display	Displays the speed with the LEDs.
led_print()	Data generation for display	Generates and passes data to the LEDs.
led_set()	LED display	Displays data with the LEDs.
wait()	Wait	Waits for a specified time.
startup_disp()	LED display during start	LED display during start
vol2speed()	Speed instruction	Acquires the speed.
get_vol()	Volume read instruction	Instructs reading of volume information on the MC-IO board.
init_PORT()	Port setting	Performs port setting on the MC-IO board.
init_TM50()	TM50 setting processing	Sets the 8-bit timer.
start_TM50()	TM50 start processing	Starts the 8-bit timer.
wait_TM50()	TM50 wait processing	Waits for a specified time.
clear_WDTM()	WDT clear processing	Clears the timer value of the watchdog timer.

4.11.4 Flowcharts

The flowchart of each function is shown below.

- Motor library

Figure 4-12. Motor Initial Setting Processing (motor_init Function)

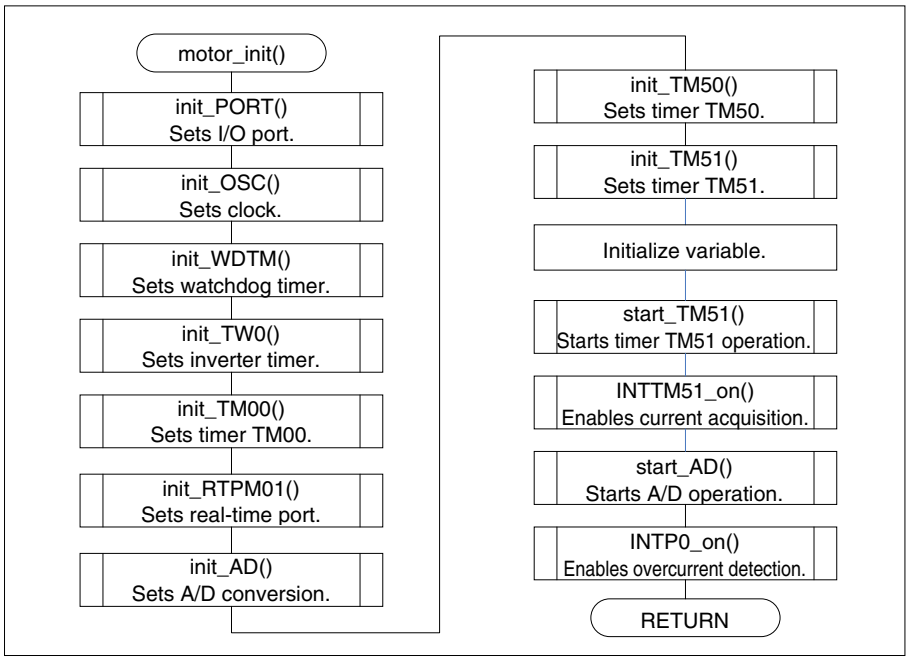


Figure 4-13. Motor Start Processing (motor_start Function)

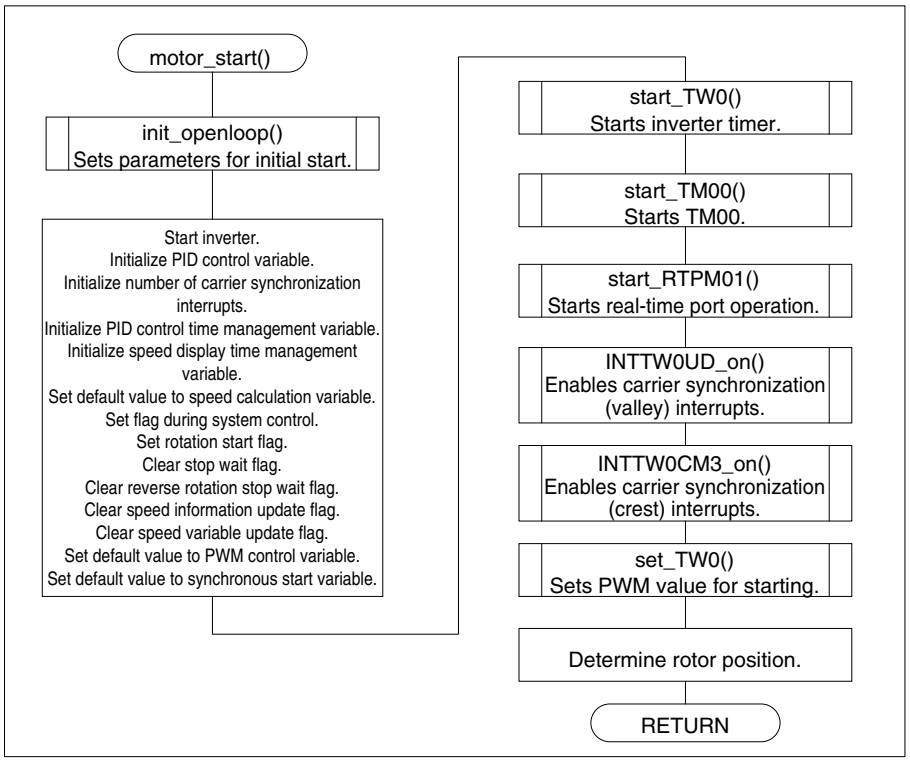


Figure 4-14. Motor Stop Processing (motor_stop Function)

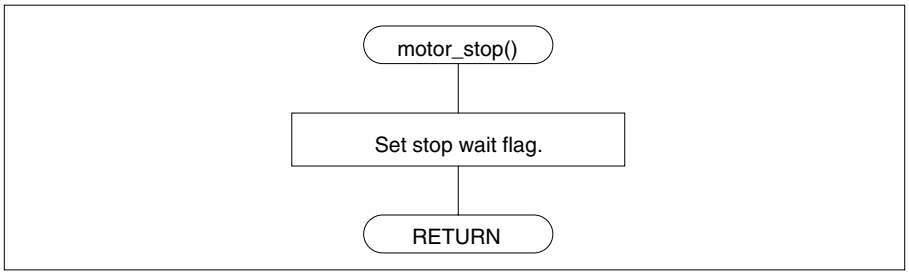


Figure 4-15. Motor Rotation Direction Change Processing (motor_rotation Function)

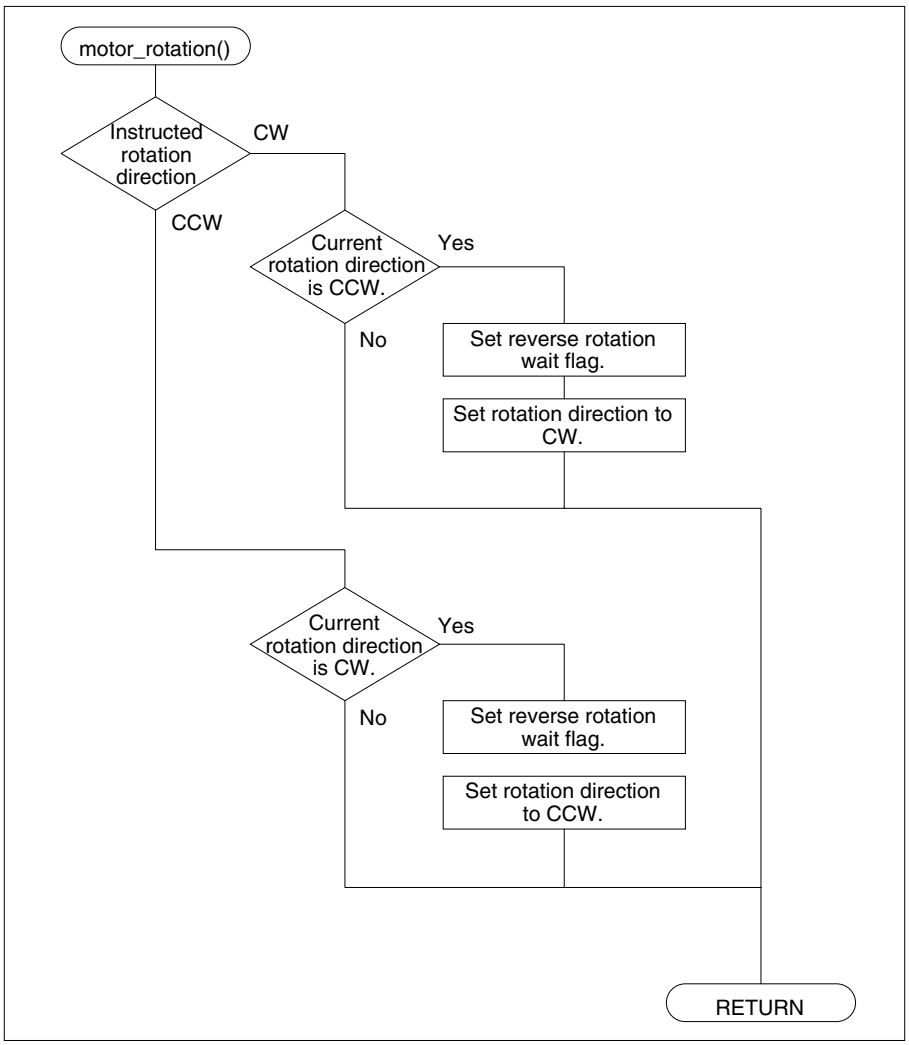


Figure 4-16. Speed PID Control Processing (motor_pid Function)

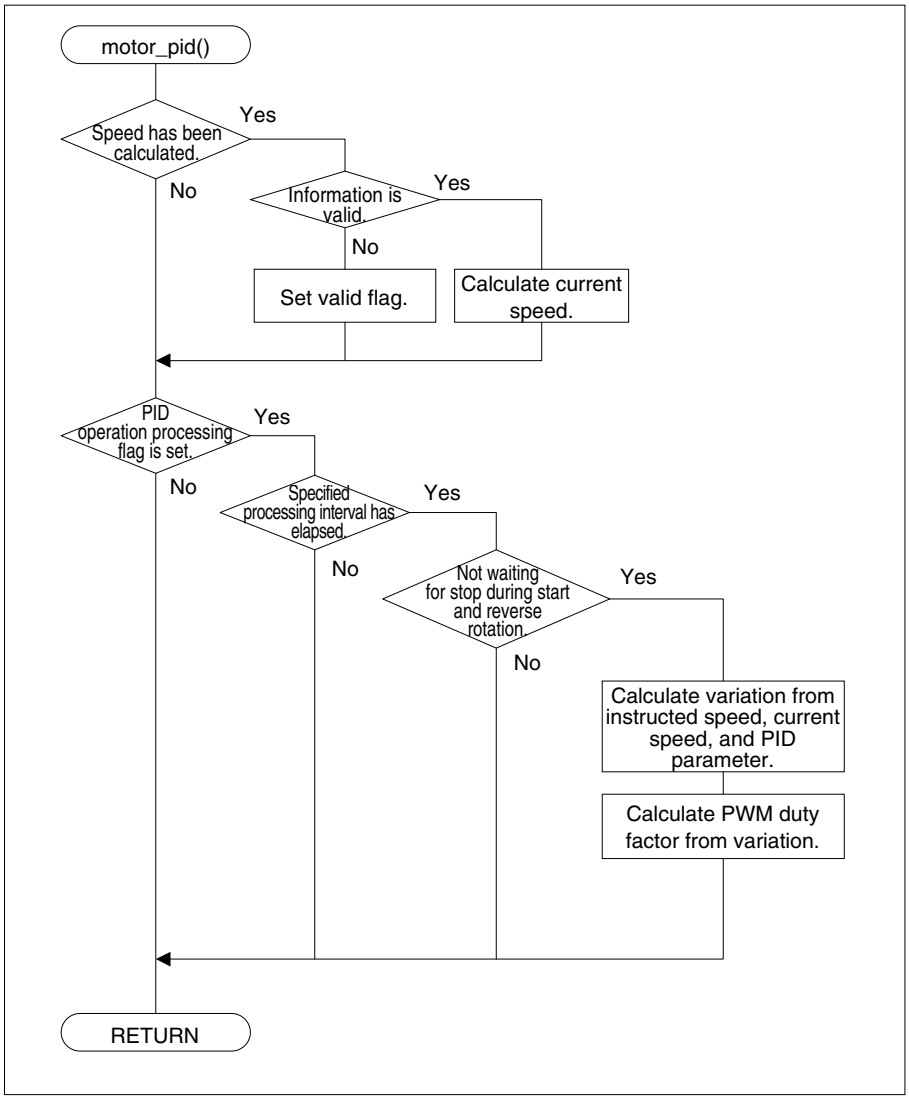


Figure 4-17. Motor Control Parameter Change Processing (motor_pset Function)

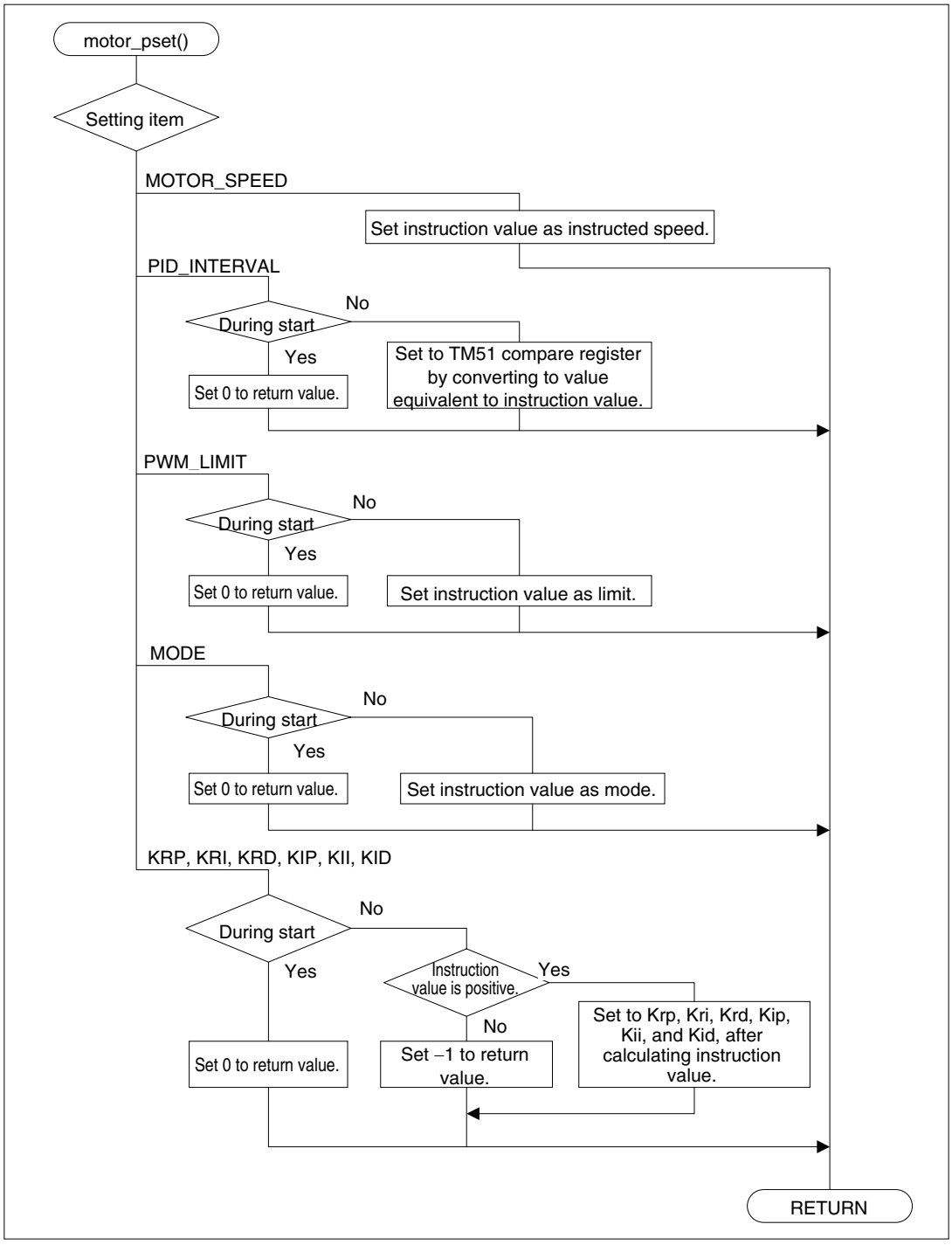


Figure 4-18. Processing of Restarting from Stop of Reverse Rotation (system_restart Function)

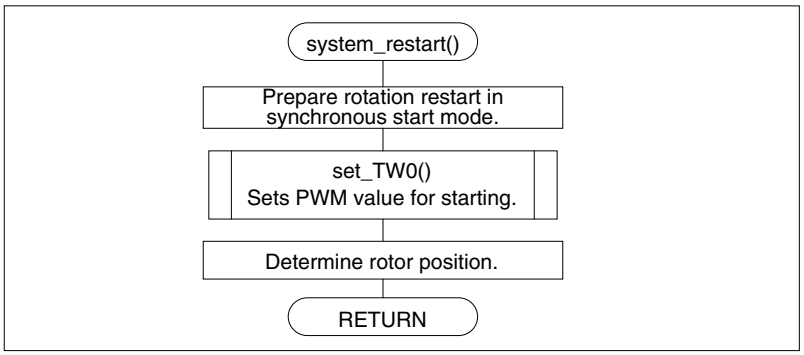


Figure 4-19. System Stop Processing (system_stop Function)

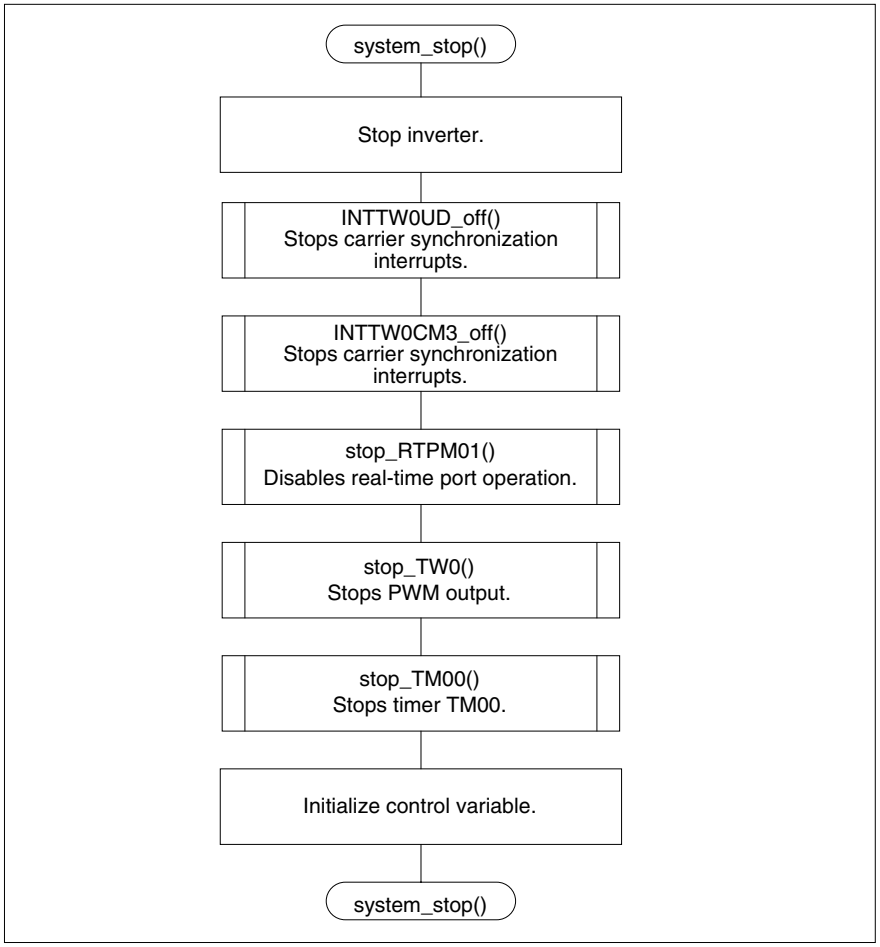


Figure 4-20. Processing of Restarting from Stop of Reverse Rotation (init_openloop Function)

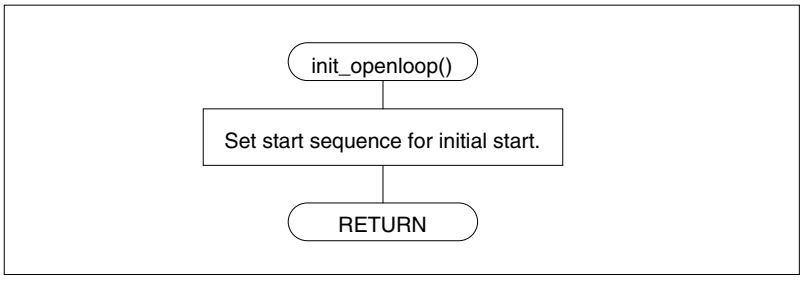


Figure 4-21. Port Setting Processing (init_PORT Function)

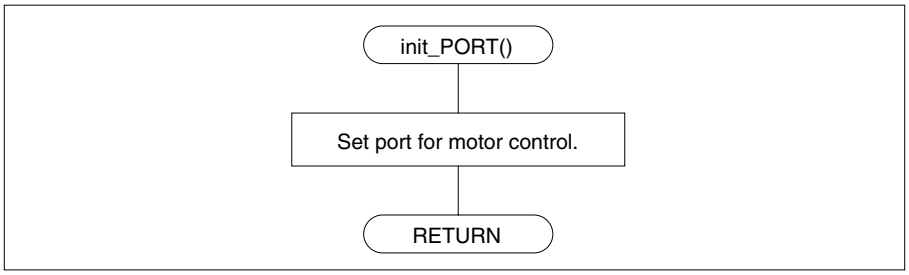


Figure 4-22. Clock Switching Processing (init_OSC Function)

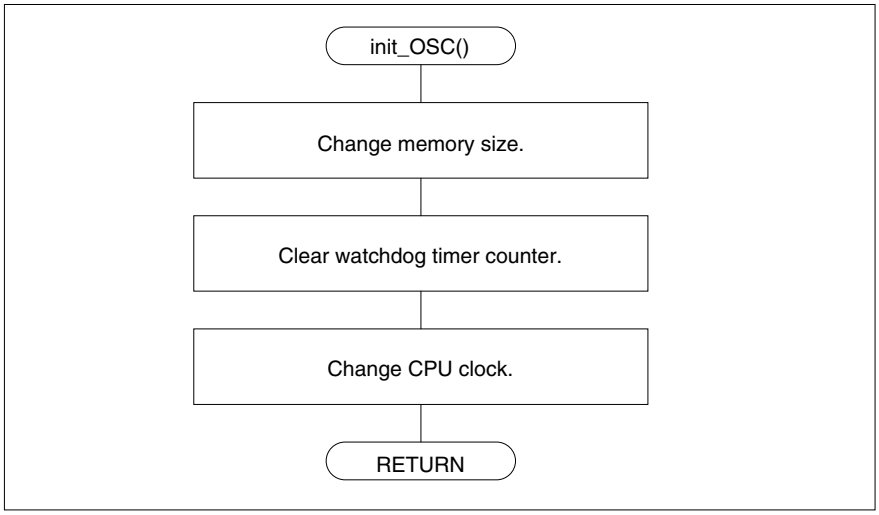


Figure 4-23. Inverter Timer Initial Setting Processing (init_TW0 Function)

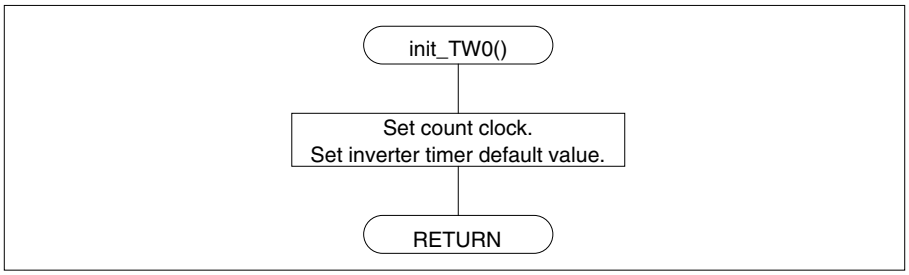


Figure 4-24. Inverter Timer Operation Start Processing (start_TW0 Function)

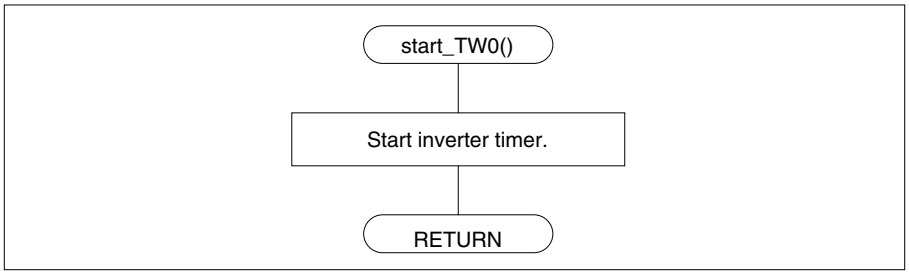


Figure 4-25. PWM Duty Factor Setting Processing (set_TW0 Function)

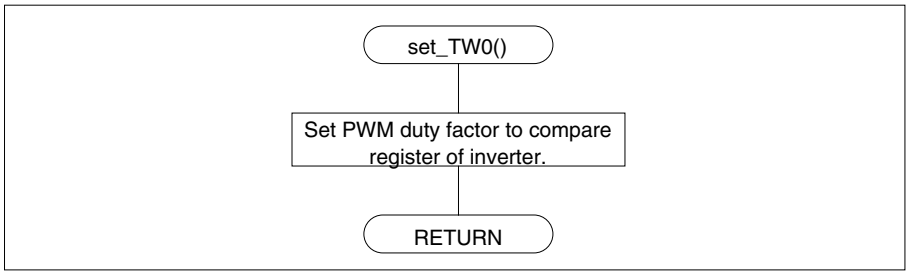


Figure 4-26. Inverter Timer Operation Stop Processing (stop_TW0 Function)

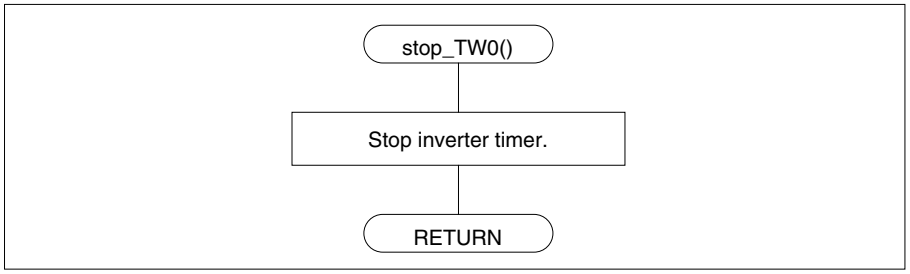


Figure 4-27. Timer Initial Setting Processing for Excitation Pattern Switching (init_TM00 Function)

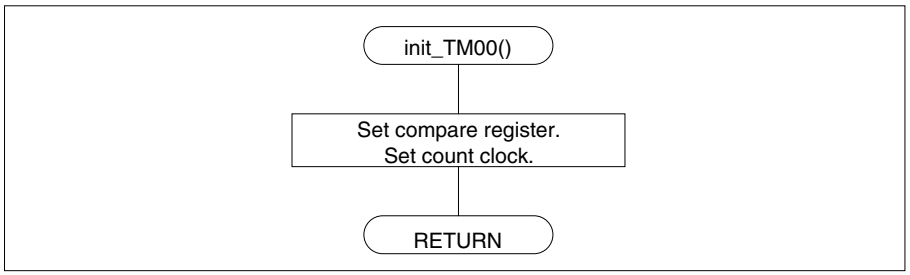


Figure 4-28. Timer Start Processing for Excitation Pattern Switching (start_TM00 Function)

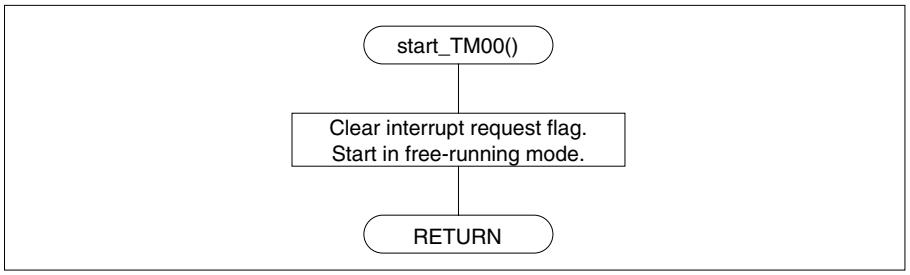


Figure 4-29. Timer Stop Processing for Excitation Pattern Switching (stop_TM00 Function)

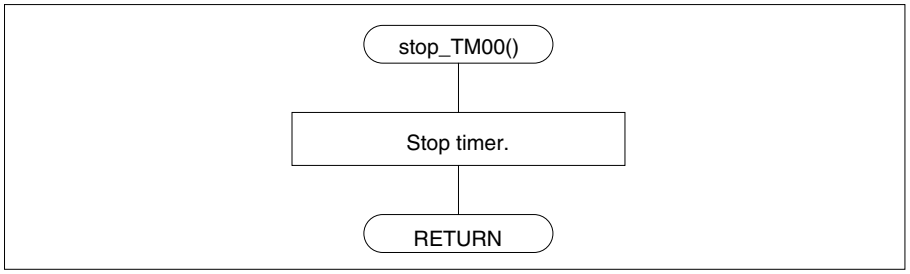


Figure 4-30. Port Initial Setting Processing for Excitation Pattern Switching (init_RTPM01 Function)

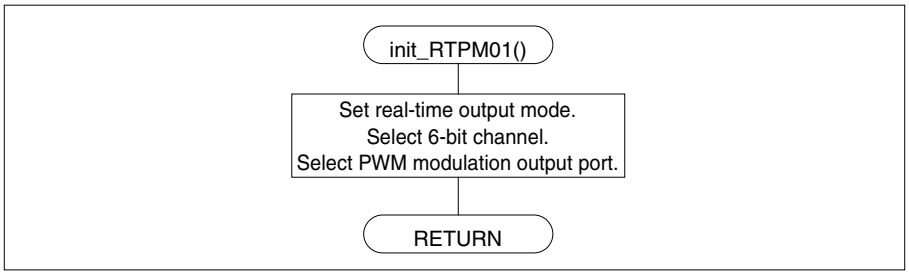


Figure 4-31. Excitation Pattern Setting (set_RTPM01 Function)

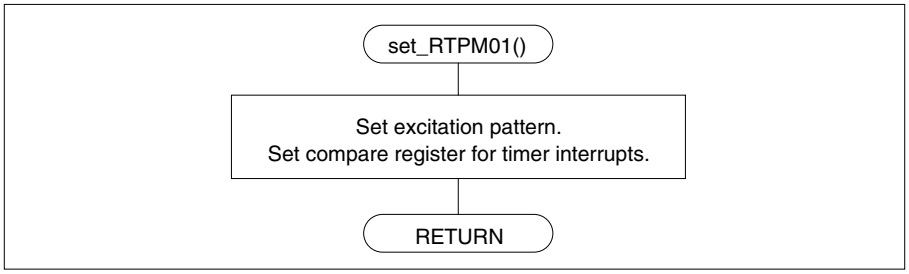


Figure 4-32. Port Output Enable Processing for Excitation Pattern Switching (start_RTPM01 Function)

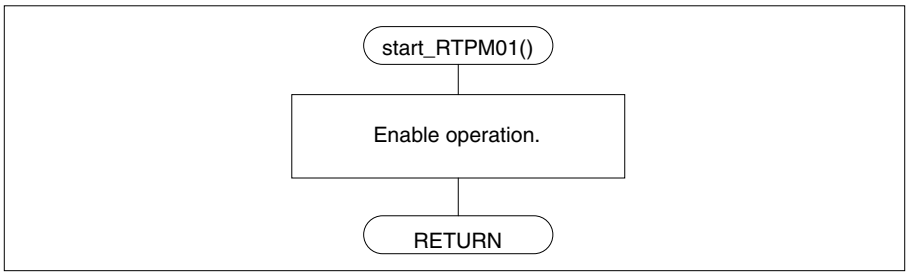


Figure 4-33. Port Output Disable Processing for Excitation Pattern Switching (stop_RTPM01 Function)

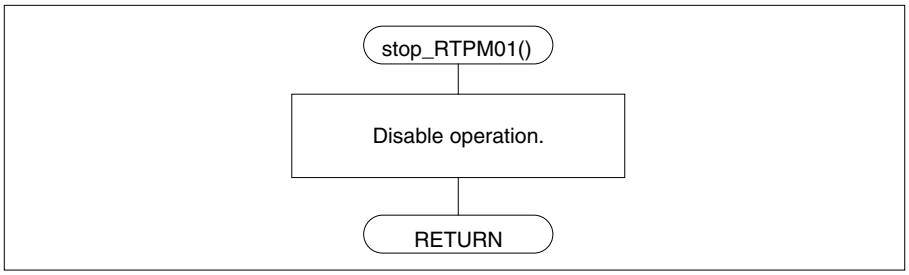


Figure 4-34. A/D Initial Setting Processing (init_AD Function)

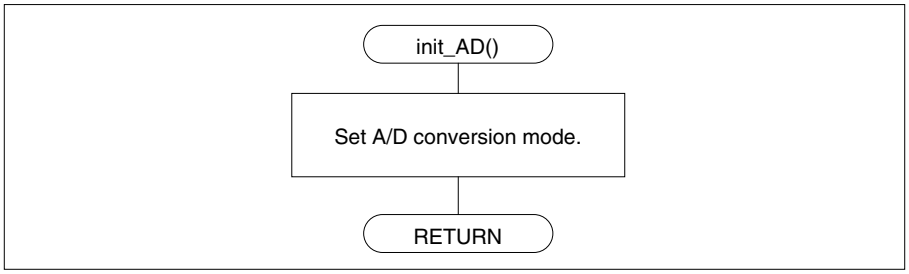


Figure 4-35. A/D Conversion Operation Start Processing (start_AD Function)

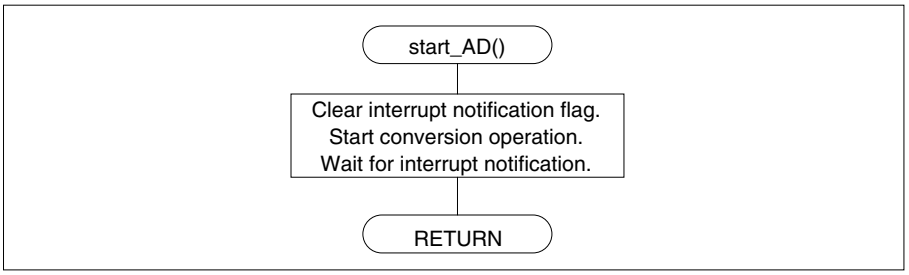


Figure 4-36. Watchdog Timer Initial Setting Processing (init_WDTM Function)

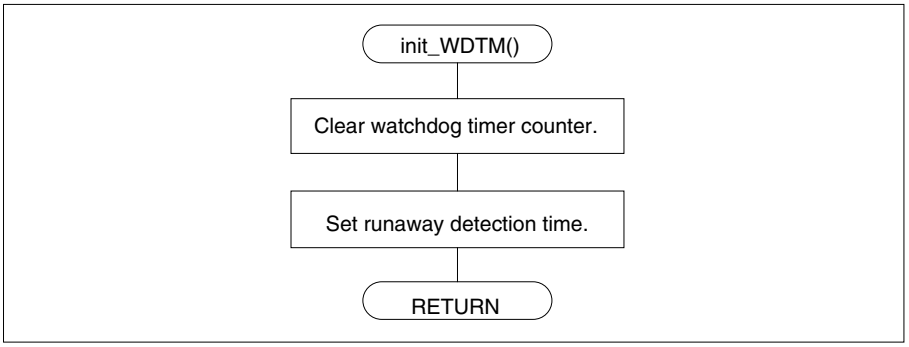


Figure 4-37. 8-bit Timer (TM51) Initial Setting Processing (init_TM51 Function)

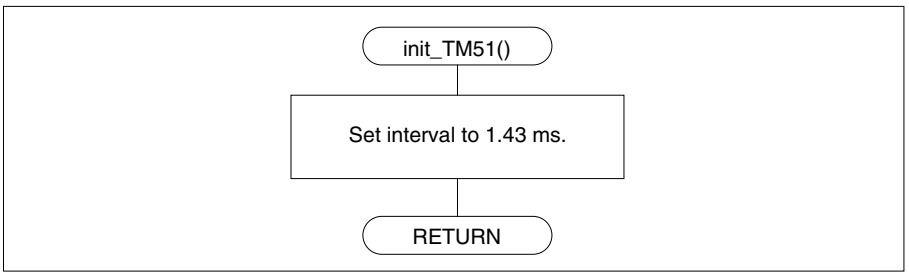


Figure 4-38. 8-bit Timer (TM51) Start Processing (start_TM51 Function)

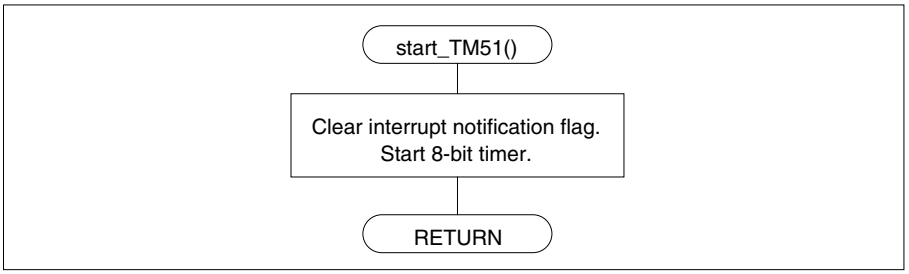


Figure 4-39. 8-bit Timer (TM50) Initial Setting Processing (init_TM50 Function)

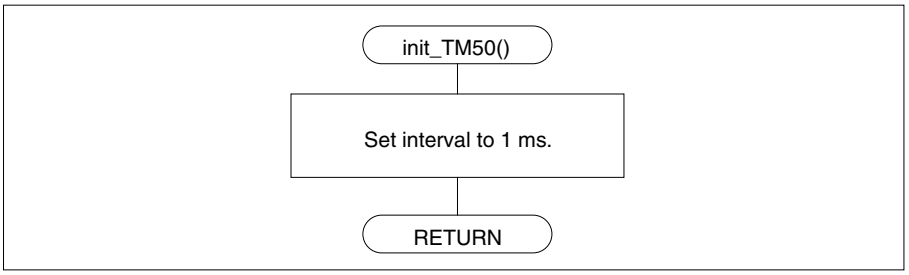


Figure 4-40. Overcurrent Interrupt Enable Processing (INTP0_on Function)

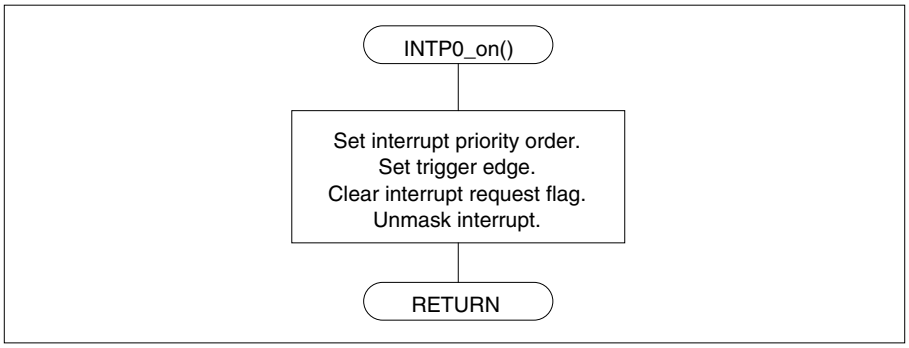


Figure 4-41. Carrier Synchronization Interrupt (Valley Processing) Enable Processing (INTTW0UD_on Function)

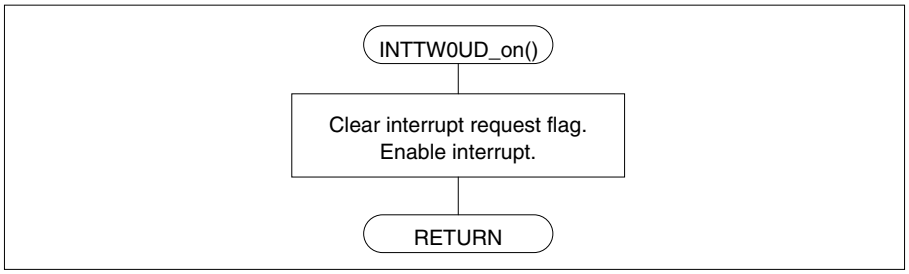


Figure 4-42. Carrier Synchronization Interrupt (Valley Processing) Disable Processing (INTTW0UD_off Function)

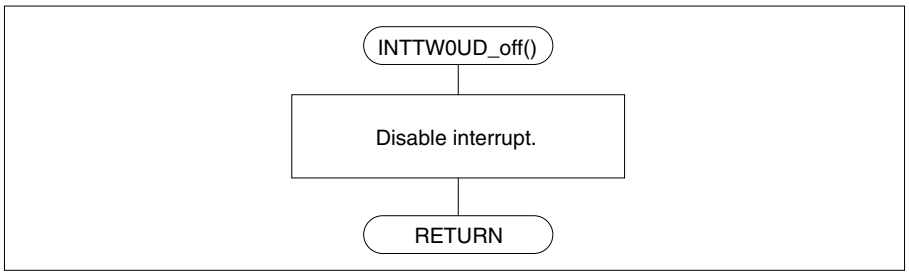


Figure 4-43. Carrier Synchronization Interrupt (Crest Processing) Enable Processing (INTTW0CM3_on Function)

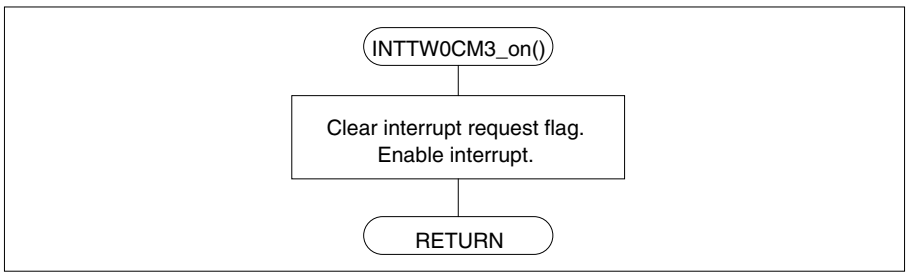


Figure 4-44. Carrier Synchronization Interrupt (Crest Processing) Disable Processing (INTTW0CM3_off Function)

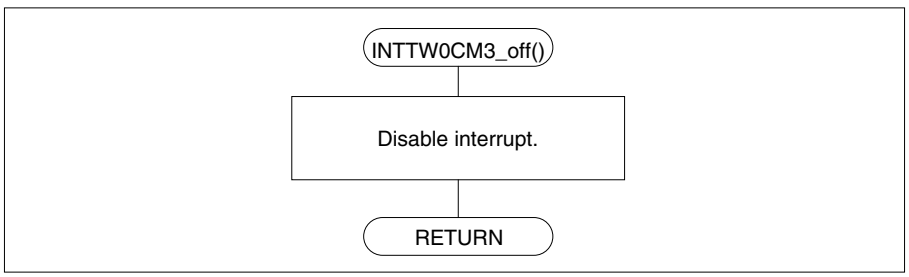


Figure 4-45. Current Minor Interrupt Enable Processing (INTTM51_on Function)

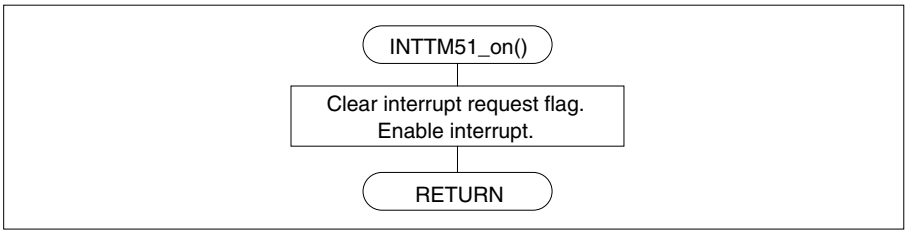


Figure 4-46. Processing for Waiting Specified Time to Elapse (wait Function)

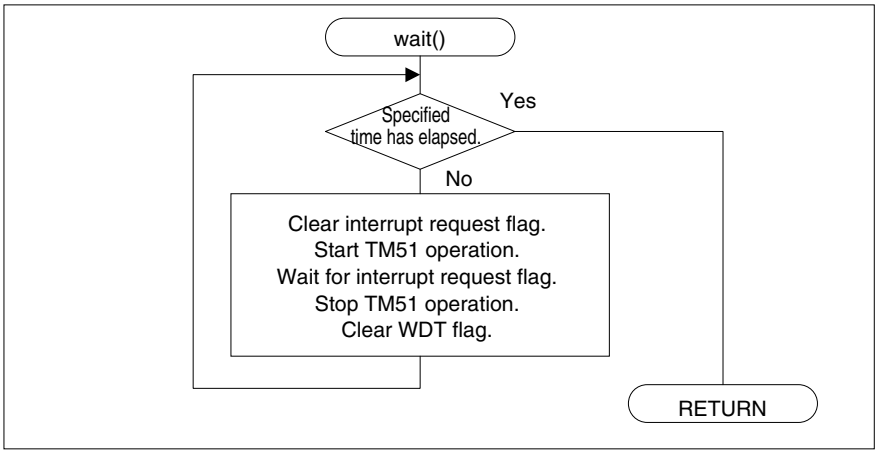


Figure 4-47. Overcurrent Interrupt Servicing (int_faulta Function)

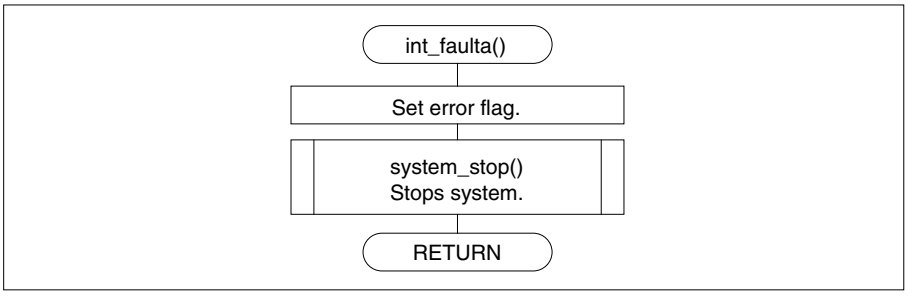


Figure 4-48. Current Minor Interrupt Servicing (int_TM51 Function)

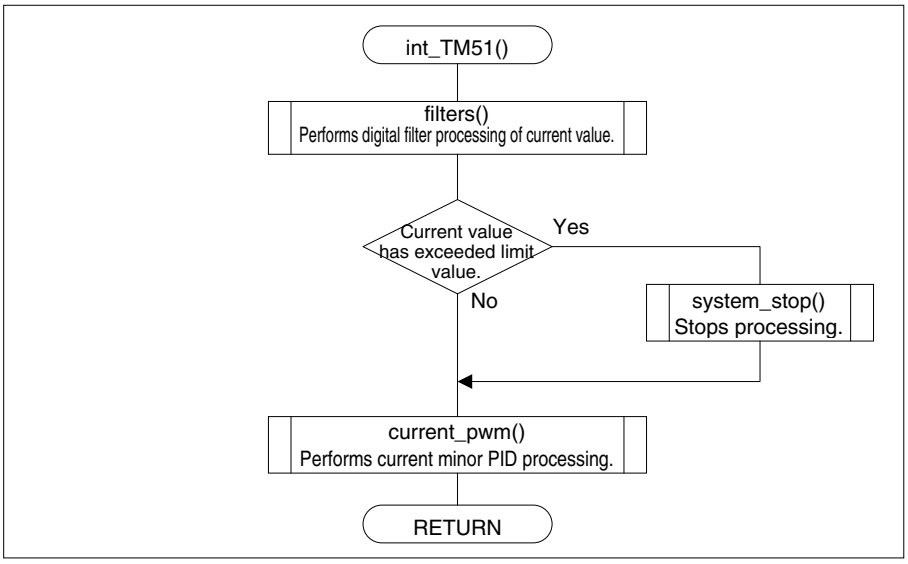


Figure 4-49. Carrier Synchronization Interrupt (Valley Processing) (1/4) (int_carrier Function)

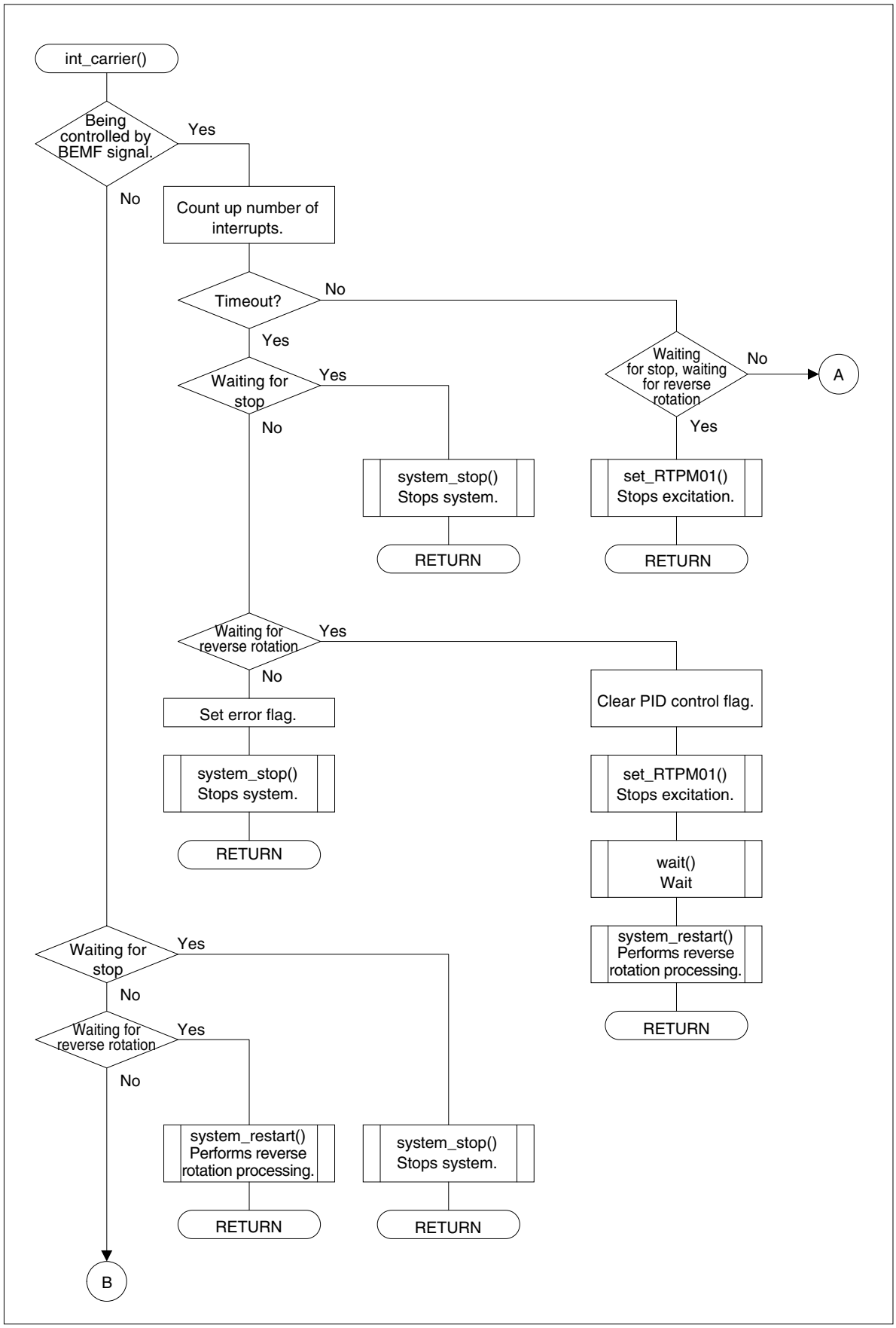


Figure 4-50. Carrier Synchronization Interrupt (Valley Processing) (2/4) (int_carrier Function)

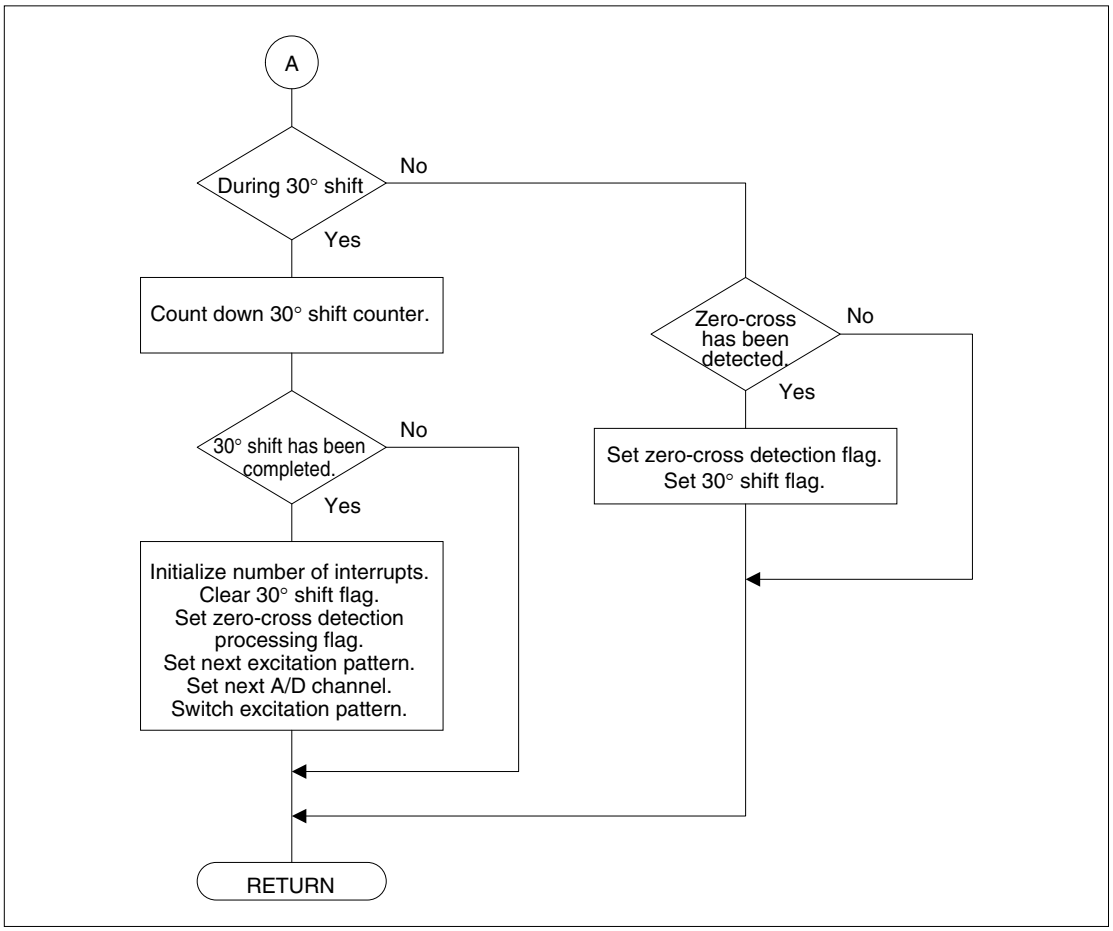


Figure 4-51. Carrier Synchronization Interrupt (Valley Processing) (3/4) (int_carrier Function)

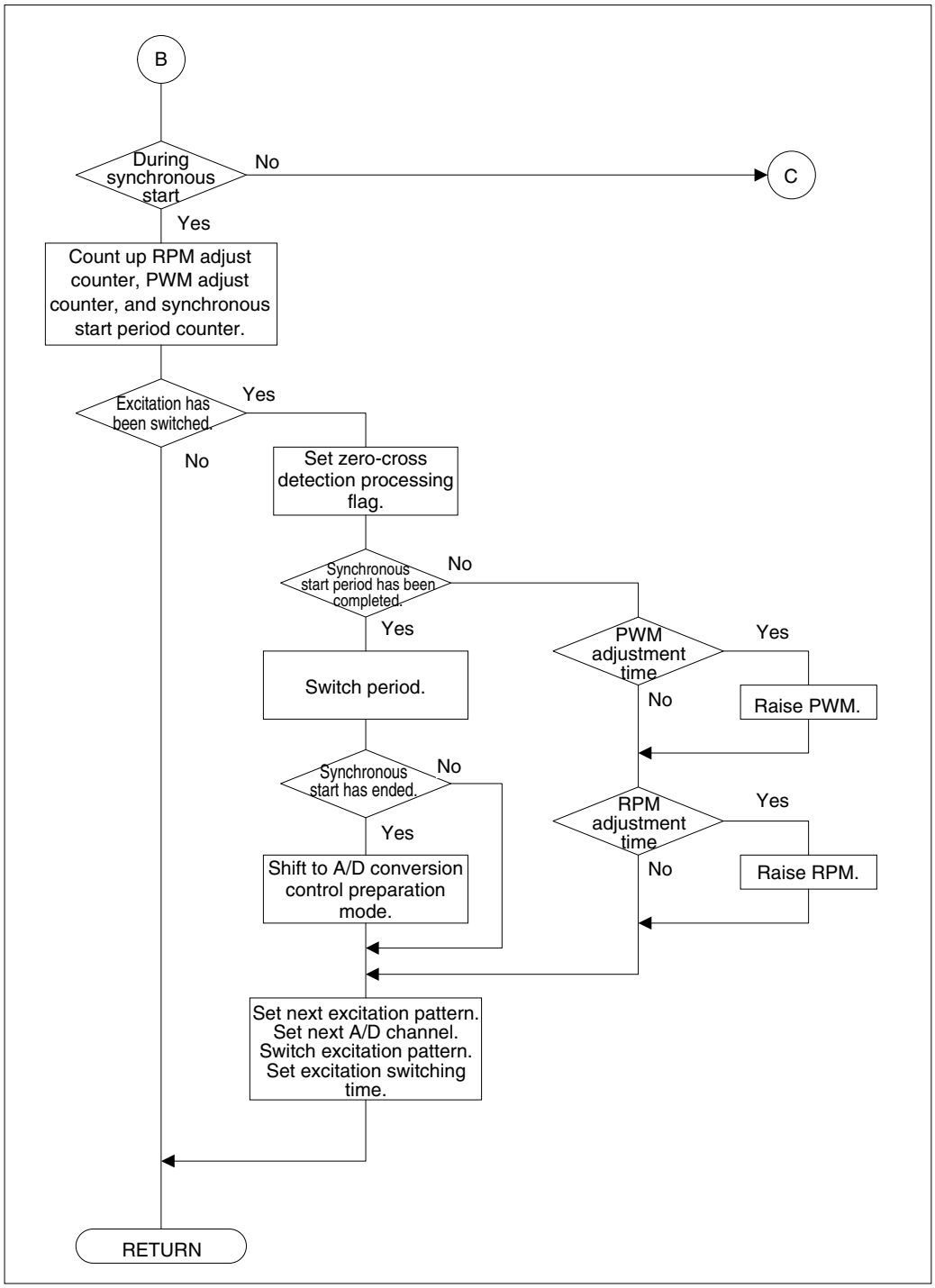


Figure 4-52. Carrier Synchronization Interrupt (Valley Processing) (4/4) (int_carrier Function)

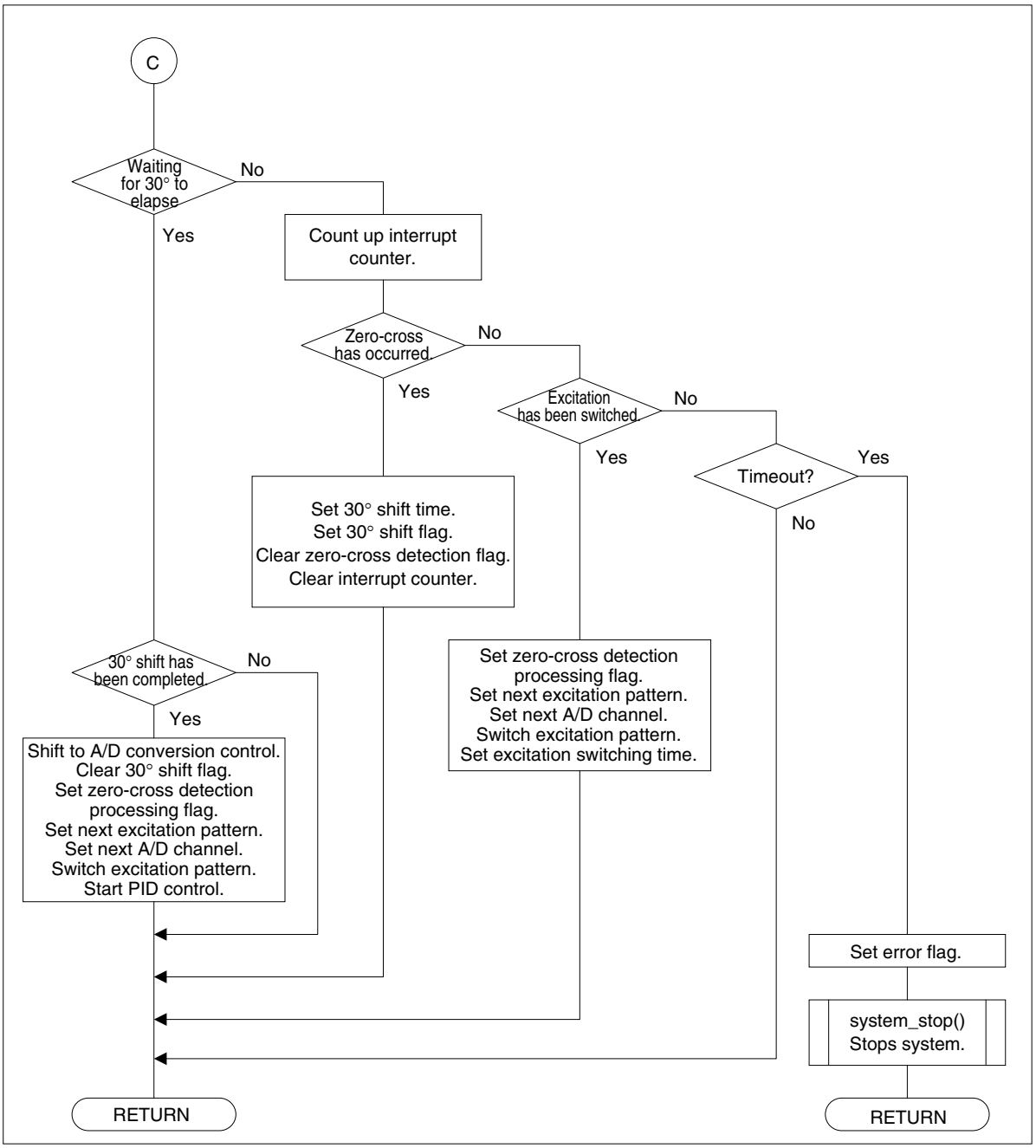


Figure 4-53. Carrier Synchronization Interrupt (Crest Processing) (int_tw0cm3 Function)

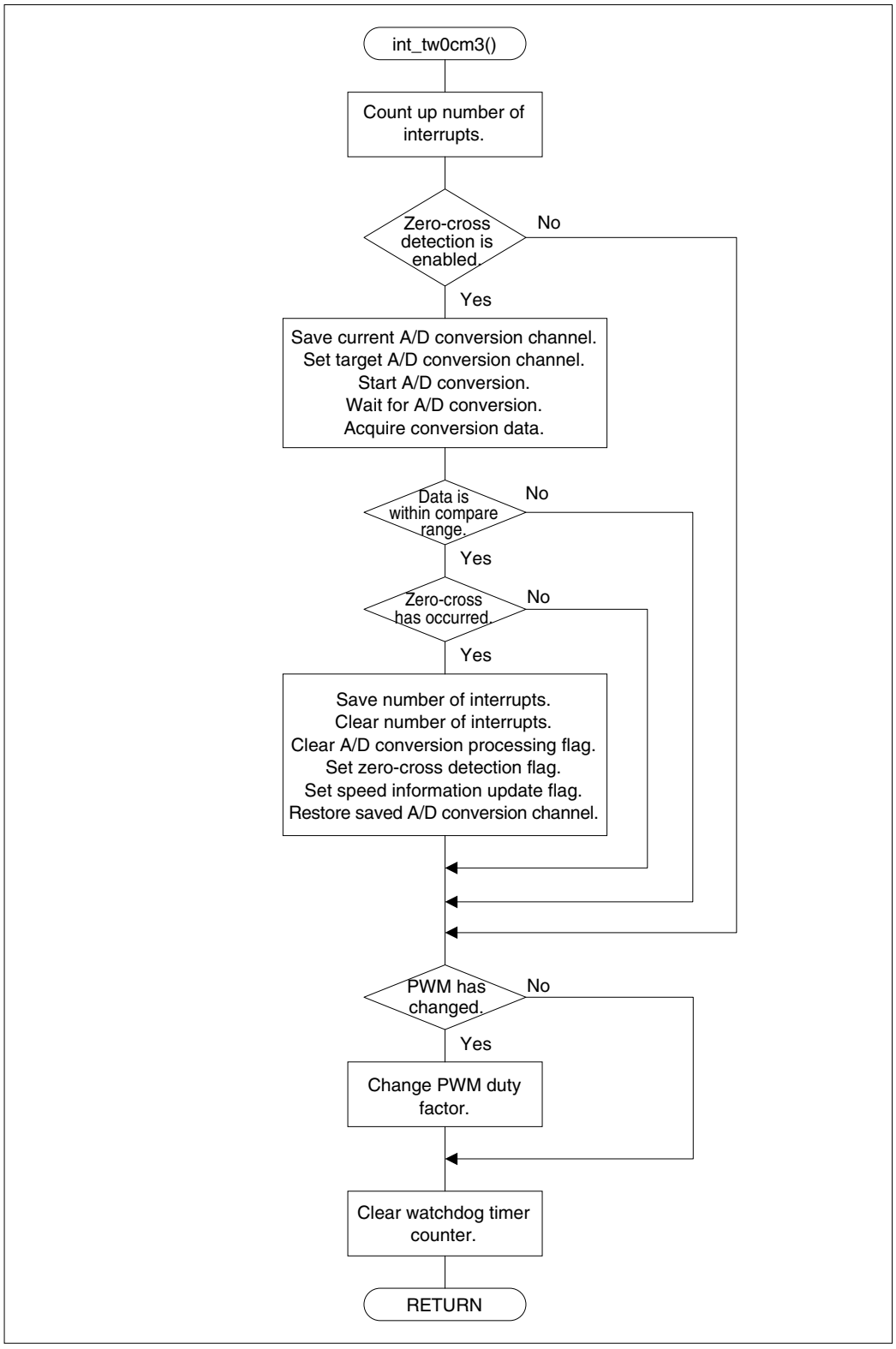


Figure 4-54. Digital Filter Processing (filters Function)

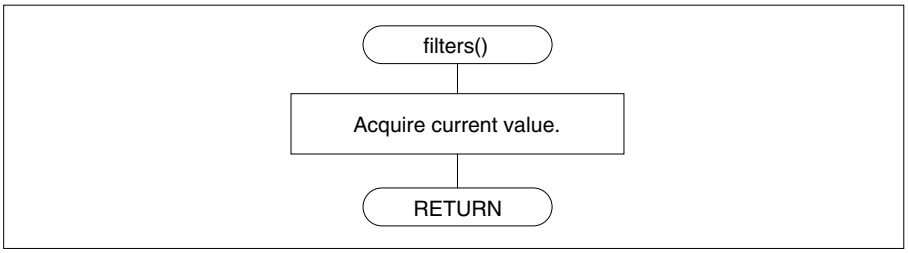


Figure 4-55. Current Minor PID Processing (current_pwm Function)

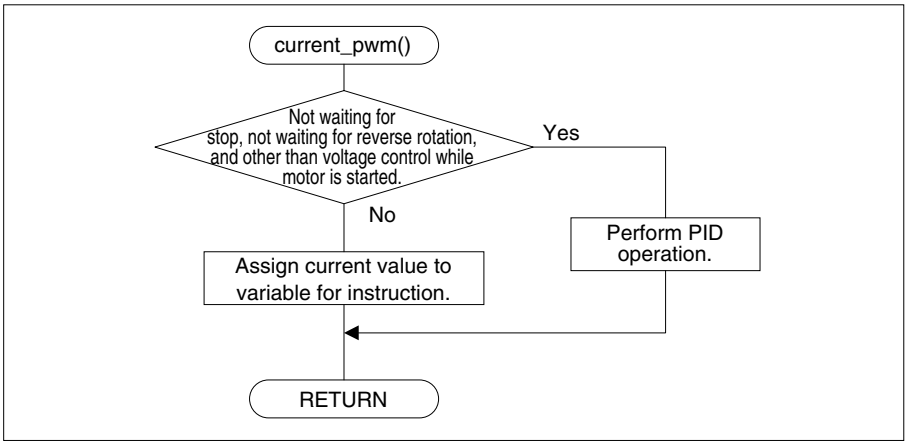
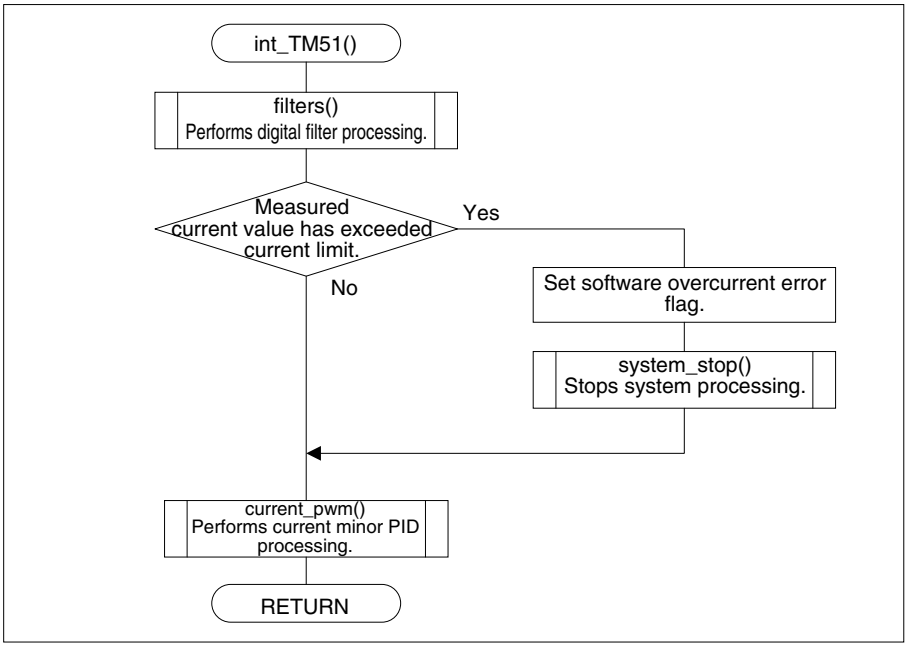


Figure 4-56. Interrupt Servicing for Current Acquisition (int_TM51 Function)



4.12 Motor Library Constant List

4.12.1 Constants changeable by users

- Low-voltage inverter settings

Table 4-7. Motor Library Constants Changeable by Users (L/V)

Name	Meaning	Setting Value	Remark
PWM_F_MIN	Minimum value of duty factor	5	
PWM_F_MAX	Maximum value of duty factor	PWM_BASE_REF	
STARTUP_METHOD_REF	Initial starting method	PWM_START	Settings for initial start process for sensorless control
T_KNEE_REF	Initial start switching time	0.75	
T_END_REF	Initial start end time	1.50	
RPM0_REF	Initial number of revolutions during initial start	60	
RPM1_REF	Number of revolutions during initial start switching	100	
RPM2_REF	Number of revolutions at end of initial start	200	
I_REF0_REF	Initial current value during initial start	140	
I_REF1_REF	Current value during initial start switching	160	
I_REF2_REF	Current value at end of initial start	160	
PWM0_REF	Initial PWM value during initial start	100	
PWM1_REF	PWM value during initial start switching	100	
PWM2_REF	PWM value at end of initial start	100	
PWM_F_START	PWM value at start	10	
ADGAIN_REF	Amplified variable of A/D conversion value	1.0	
ADOFFSET_REF	Offset variable of A/D conversion value	0	
MAXCURRENT_REF	Maximum current value	1024	
MINCURRENT_REF	Minimum current value	10	
MAXSPEED_REF	Maximum number of revolutions	5000	
MINSPEED_REF	Minimum number of revolutions	300	
I_RATE_MAX_REF	Rise rate for each current unit time	900	
RPM_RATE_MAX_REF	Rise rate for each unit time of number of revolutions	2000	
KRP_REF_REF	Kp value of number of revolutions	0.05	
KRI_REF_REF	Ki value of number of revolutions	0.02	
KRD_REF_REF	Kd value of number of revolutions	0.01	
KIP_REF_REF	Kp value of current	0.10	
KII_REF_REF	Ki value of current	0.04	
KID_REF_REF	Kd value of current	0.02	
CLIMIT	Current value for overcurrent detection	700	

4.12.2 Constants referenceable by users

Table 4-8. Motor Library Constants Referenceable by Users

Name	Meaning	Setting Value	Remark
FLG_ON	Flag value	1	Indicates whether flag is valid.
FLG_OFF	Flag value	0	
CW	Rotation direction	0	Indicates rotation direction of motor.
CCW	Rotation direction	1	
ERROR_NONE	Error flag bit	0x00	No error
ERROR_HALL	Error flag bit	0x01	Hall IC error
ERROR_OC	Error flag bit	0x02	Error due to overcurrent
ERROR_MOTOR	Error flag bit	0x04	Error of motor failure
ERROR_S_OC	Error flag bit	0x08	Error due to overcurrent
MOTOR_SPEED	Parameter set code	0x01	Sets number of revolutions of motor.
PID_INTERVAL	Parameter set code	0x10	Sets PID interval.
MODE	Parameter set code	0x12	Sets mode.
KRP	Parameter set code	0x20	Sets Kp of number of revolutions.
KRI	Parameter set code	0x21	Sets Ki of number of revolutions.
KRD	Parameter set code	0x22	Sets Kd of number of revolutions.
KIP	Parameter set code	0x23	Sets Kp of current.
KII	Parameter set code	0x24	Sets Ki of current.
KID	Parameter set code	0x25	Sets Kd of current.
WDTE_CLR	Watchdog timer value	0xac	WDT clear value
CURRENT_START	Initial start mode	0x01	
PWM_START	Initial start mode	0x02	
NOTABLE_START	Initial start mode	0x03	
SPEED_CMD	Operation mode	0x01	
I_CMD	Operation mode	0x02	
V_CMD	Operation mode	0x03	
AD_ISHUNT	Current detection channel	5	
UNIT_RPM	Constant for calculating number of revolutions	2343750	

4.12.3 Internal constants

Table 4-9. Motor Library Internal Constants

Name	Meaning	Setting Value	Remark
PWM_BASE_REF	PWM base	1000	Used for PWM output
PWM_F_REF	PWM default value	0	
PWM_DTM_REF	Dead time	0	
IN	For port setting	1	Used to specify port function
OUT	For port setting	0	
CLEAR	For register bit setting	0	Used to access bits of registers
SET	For register bit setting	1	
INV_OFF	For inverter control	1/0	Low-voltage inverter
INV_ON	For inverter control	0/1	
INVERTER_SW	Inverter control port	P54	Inverter control port
INVERTER_SW_MODE	Inverter control register	PM54	Inverter control port
INTP0	Port control register	PM00	Overcurrent detection port
IMS_DATA	Memory size switching	0xc8	
WDTM_SET	WDT setting value	0x6f	
P_OFF	Excitation pattern	0x3f	Excitation of real-time port
P_STOP	Excitation pattern	0x15	
P_T1	Excitation pattern	0x36	
P_T2	Excitation pattern	0x1e	
P_T3	Excitation pattern	0x1b	
P_T4	Excitation pattern	0x39	
P_T5	Excitation pattern	0x2d	
P_T6	Excitation pattern	0x27	

4.13 Reference Program Constant List

4.13.1 Internal constants

Table 4-10. Reference Program Internal Constants (1/2)

Name	Meaning	Setting Value	Remark
AD_VOL	A/D channel	4	Volume on MCIO board
IN	For port setting	1	Used to specify port function
OUT	For port setting	0	
CLEAR	For register bit setting	0	Used to access bits of registers
SET	For register bit setting	1	
KEY_WAIT	Switch observation time	10	Used to eliminate chattering
SW	Switch	(P7&0xf)	Switch connection port
SW2	Port control register	PM73	Port for START/STOP
SW3	Port control register	PM72	Port for FORWARD
SW4	Port control register	PM71	Port for REVERSE
SW5	Port control register	PM70	Port for MODE
LD_LED0	Port control register	PM64	Port for LED selection
LD_LED1	Port control register	PM65	
LD_LED2	Port control register	PM66	
LD_LED3	Port control register	PM67	
LD_DATA	Port control register	PM4	Port that outputs data to LED
START_SW	Start	0x7	Switch status
STOP_SW	Stop	0x7	
FORWARD_SW	CW	0xb	
REVERSE_SW	CCW	0xd	
MODE_SW	Mode	0xe	
START_TR	For status setting	0x01	Starts control.
STOP_TR	For status setting	0x02	Stops control.
FORWARD_TR	For status setting	0x04	Changes rotation to CW.
REVERSE_TR	For status setting	0x08	Changes rotation to CCW.
MODE_TR	For status setting	0x10	State where MODE switch is pressed
LED_0	LED display data	0xc0	Displays “0”.
LED_1		0cf9	Displays “1”.
LED_2		0xa4	Displays “2”.
LED_3		0xb0	Displays “3”.
LED_4		0x99	Displays “4”.
LED_5		0x92	Displays “5”.
LED_6		0x82	Displays “6”.
LED_7		0xf8	Displays “7”.
LED_8		0x80	Displays “8”.
LED_9		0x98	Displays “9”.
LED_O		0xc0	Displays “0” instead of “O”.
LED_I		0xcf	Displays “I”.
LED_C		0xc6	Displays “C”.
LED_H		0x89	Displays “H”.

Table 4-10. Reference Program Internal Constants (2/2)

Name	Meaning	Setting Value	Remark
LED_A	LED display data	0x88	Displays “A”.
LED_L		0xc7	Displays “L”.
LED_		0xff	Displays “ ”.
LED_S		0x92	Displays “S”.
LED_E		0x86	Displays “E”.
LED_F		0x8e	Displays “F”.
LED_P		0x8c	Displays “P”.
LED_dot		0x7f	Displays “.”.

4.14 Motor Library Variable List

4.14.1 Externally disclosed variables

Table 4-11. External Disclosed Motor Library Variables (1/2)

Variable Name	Type	Meaning	Remark
sys_flag	char	Control flag	Control status
err_flag	char	Error flag	Error status
stop_wait	char	Stop command flag	Indicates whether stop command is issued.
cw_ccw_flag	char	Rotation direction command flag	Rotation direction status
cw_ccw_wait	char	Reverse stop command flag	Indicates whether command to stop by reversing is issued.
maxed_flags	unsigned char	Upper-limit flag	Flag set when upper limit of unit time has been exceeded
print_cnt	unsigned int	Speed display timing	Number of carrier synchronization interrupts
pwm_ff	int	Current PWM command value	Current PWM duty factor value
pwm_ff_o	int	PWM command value	PWM duty factor command value
m_speed	int	Actual speed	Revolution speed of motor (rpm)
krp_ref	float	Kp value of number of revolutions	Speed control
kri_ref	float	Ki value of number of revolutions	
krd_ref	float	Kd value of number of revolutions	
en	float	Difference from current value	
en_1	float	Difference from previous value	
en_2	float	Difference from value before previous	
kip_ref	float	Kp value of current	Current minor loop control
kii_ref	float	Ki value of current	
kid_ref	float	Kd value of current	
ec	float	Difference from current value	
ec_1	float	Difference from previous value	
ec_2	float	Difference from value before previous	
startup_method	unsigned char	Synchronous start method	
I_ref	float	Current value of current command	
I_ref_o	float	Current command value	
I_measured	float	Operating current value	
speed_ref	int	Current speed command value	
speed_ref_o	int	Speed command value	

Table 4-11. External Disclosed Motor Library Variables (2/2)

Variable Name	Type	Meaning	Remark
maxspeed	int	Maximum speed	
minspeed	int	Minimum speed	
dspeed_ref_max	int	Controlled variable of speed within unit time	
dI_ref_max	float	Controlled variable of current within unit time	
RPM_rate_max	float	Variation of speed within unit time	
I_rate_max	float	Variation of current within unit time	
t_knee	float	Initial start switching time	
t_end	float	Initial start end time	
RPM0	float	Initial number of revolutions during initial start	
RPM1	float	Number of revolutions during initial start switching	
RPM2	float	Number of revolutions at end of initial start	
I_ref0	float	Initial current value during initial start	
I_ref1	float	Current value during initial start switching	
I_ref2	float	Current value at end of initial start	
PWM0	float	Initial PWM value during initial start	
PWM1	float	PWM value during initial start switching	
PWM2	float	PWM value at end of initial start	
adgain	float	Amplified variable of A/D conversion value	
adoffset	float	Offset variable of A/D conversion value	
maxcurrent	float	Maximum current value	
mincurrent	float	Minimum current value	

4.14.2 Internal variables

Table 4-12. Motor Library Internal Variables

Variable Name	Type	Meaning	Remark
old_hdata	char	Hall IC history	Value of previous Hall IC
speed_flag	char	Speed information flag	Indicates whether data required for calculating speed is present.
int_cnt	unsigned int	Interrupt counter	Number of carrier synchronization interrupts
pid_cnt	unsigned int	PID control timing	Number of carrier synchronization interrupts
pwm_base	int	PWM base clock value	PWM carrier width
old_ff	int	PWM command value history	Value of previous pwm_ff
up_flag	char	Actual-speed calculation status flag	For calculating rotation speed
capture_flag	char	Timer value acquisition flag	
clk_flag	char	Timer value calculation enable flag	
old_clk	unsigned int	Previous clock value	
new_clk	unsigned int	Newest clock value	
mvn_f	float	Previous amount operated	
startdelay	char	Current detection start delay value	
start_flag	char	Rotation start flag	
cy_time	int	PID control interval	
cmd_mode	char	Control mode	
dpwm_ff_max	int	Controlled variable of PWM within unit time	

4.15 Reference Program Variable List

4.15.1 Internal variables

Table 4-13. Reference Program Variables

ad_flag	char	Speed change flag	Limits specified-speed change function.
led_data[ ]	char	LED output data	Value displayed with LEDs

4.16 Motor Library Source File

Memory size ROM: 1F3Dh
RAM: 320h

```
/*


   BLDCM 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target      :  uPD78F0714 motor starter kit
   date        : 2007/05/10
   filename  : motor.h

   NEC Micro Systems,Ltd
*/

#ifdef LOW
/* ============================================== */
/* For Pittman motor */
#define PWM_F_MIN               5                   /* Minimum value              */
#define PWM_F_MAX               PWM_BASE_REF        /* Maximum value              */

/* startup parameters */
#define STARTUP_METHOD_REF PWM_START          /* which method to use: PWM_START, CURRENT_START, or NOTABLE_START */
#define T_KNEE_REF              2.0                 /* start of second startup segment */
#define T_END_REF               4.0                 /* end of startup second segment */
#define RPM0_REF                100                 /* initial startup RPM */
#define RPM1_REF                200                 /* midpoint startup RPM */
#define RPM2_REF                300                 /* final startup RPM */
#define I_REF0_REF              140                 /* initial startup current command */
#define I_REF1_REF              160                 /* midpoint  startup current command */
#define I_REF2_REF              160                 /* final  startup current command */
#define PWM0_REF                180                 /* initial startup voltage setting */
#define PWM1_REF                185                 /* midpoint startup voltage setting */
#define PWM2_REF                190                 /* final startup voltage setting */

/* original startup NOTABLE_START */
#define SYNC_DEF_REF            500
#define PWM_F_START             10                  /* Default value at start          */

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

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

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

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

/* ============================================== */
#else
/* ============================================== */
/* For Oriental motor */
#define PWM_F_MIN               5                   /* Minimum value              */
#define PWM_F_MAX               PWM_BASE_REF        /* Maximum value              */

/* startup parameters */
#define STARTUP_METHOD_REF PWM_START /* which method to use: PWM_START, CURRENT_START, or NOTABLE_START */
#define T_KNEE_REF              0.75                /* start of second startup segment */
#define T_END_REF               1.5                 /* end of startup second segment */
#define RPM0_REF                60                  /* initial startup RPM */
#define RPM1_REF                100                 /* midpoint startup RPM */
#define RPM2_REF                200                 /* final startup RPM */
#define I_REF0_REF              140                 /* initial startup current command */
#define I_REF1_REF              160                 /* midpoint  startup current command */
#define I_REF2_REF              160                 /* final  startup current command */
#define PWM0_REF                100                 /* initial startup voltage setting */
#define PWM1_REF                100                 /* midpoint startup voltage setting */
#define PWM2_REF                100                 /* final startup voltage setting */

/* original startup NOTABLE_START */
#define SYNC_DEF_REF            500
#define PWM_F_START             70                  /* Default value at start          */

/* a/d gain parameters (for GUI mostly) */
#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 */

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

/* Feedback Gains: current measured in A/D units, voltage in PWM%*10 */
/*  RPM feedback to I command  #define */
#define KRP_DEF_REF             0.02            /* dI_ints/dRPM error */
#define KRI_DEF_REF             0.01            /* dI_ints/RPM error  */
#define KRD_DEF_REF             0               /* dI_ints/d(dRPM error) */

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

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

#define CLIMIT                  1024            /* 1024 */

#define AD_CH1                  3               /* Phase U */
#define AD_CH2                  6               /* Phase V */
#define AD_CH3                  7               /* Phase W */
#define AD_ISHUNT               5               /* ISHUNT pin */
#define AD_VOL                  4               /* VOL pin    */

/* +++++++++++++++ The following cannot be changed. +++++++++++++++ */

#define CW                      0               /* Clockwise          */
#define CCW                     1               /* Counterclockwise  */

#define ERROR_NONE              0x00
#define ERROR_HALL              0x01            /* Hall IC failure                    */
#define ERROR_OC                0x02            /* Overcurrent                        */
#define ERROR_MOTOR             0x04            /* Motor failure                      */
#define ERROR_S_OC              0x08            /* Overcurrent (software detection) */

#define MOTOR_SPEED             0x01            /* Number of revolutions instructed */
#define PID_INTERVAL            0x10            /* PID control interval */
#define PWM_LIMIT               0x11            /* PWM variation limiter */
#define MODE                    0x12
#define KRP                     0x20            /* Kp value */
#define KRI                     0x21            /* Ki value */
#define KRD                     0x22            /* Kd value */
#define KIP                     0x23            /* Kp value */
#define KII                     0x24            /* Ki value */
#define KID                     0x25            /* Kd value */

#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 constant:  x[rpm] = ( 60[s] * 78.125[Krps] )/6
                                        ------          ---
                                          ↑               ↑ Counter value acquisition
                                          ↑               ↑ Timing
                                          ↑               ↑ HallIC(6)
                                          ↑ TM00 count clock
                                          ↑ 78.125[KHz]
 */
#define UNIT_RPM                781250   /* 2count */
#define CARRIRE_DEF_CONST       50000
#define SPEED_CAL_CONST         100000

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

extern char                     sys_flag;           /* System operation status          */
extern int                      speed_ref;          /* Number of revolutions instructed */
extern int                      speed_ref_o;
extern int                      m_speed;            /* Current number of revolutions    */
extern char                     stop_wait;          /* Stop wait flag                   */
extern char                     cw_ccw_wait;        /* Reversal wait flag               */
extern char                     cw_ccw_flag;        /* Rotation direction status        */
extern char                     rr_flag;            /* Error flag                       */
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                    dl_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 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target    :  uPD78F0714 motor starter kit
   date      : 2007/06/26
   filename: motor.c

   NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma di
#pragma ei

#pragma INTERRUPT INTP0          int_faulta     rb1   /* Overcurrent occurrence                */
#pragma INTERRUPT INTTW0UD       int_carrier    rb2   /* Carrier synchronization interrupt     */
#pragma INTERRUPT INTTW0CM3      int_tw0cm3     rb3   /* A/D conversion                        */
#pragma INTERRUPT INTTM51        int_TM51       rb1   /* ISHUNT measurement                    */


/* ---------------------------------------------- */
#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           /* For LowVoltage */
#define INV_ON                  0

#define IN                      1           /* Input */
#define OUT                     0           /* Output */

#define INVERTER_SW             P54         /* Inverter operation control port  */
#define INVERTER_SW_MODE        PM54        /* Inverter operation control port  */
#define INTP0                   PM00        /* Overcurrent detection port  */

#define IMS_DATA                0xc8

#define WDTM_SET                0x6f

#define P_OFF                   0x3f        /* Real-time output port off    */
#define P_STOP                  0x15
#define P_T1                    0x36        /* Excitation pattern 1 U → V */
#define P_T2                    0x1e        /* Excitation pattern 2 U → W */
#define P_T3                    0x1b        /* Excitation pattern 3 V → W */
#define P_T4                    0x39        /* Excitation pattern 4 V → U */
#define P_T5                    0x2d        /* Excitation pattern 5 W → U */
```

```
#define P_T6                    0x27    /* Excitation pattern 6 W → V */

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

#define CR01_ADD                3       /* Switching wait time counter value      */
                                        /* Dependent on TM00 setting              */

/* ------------------------------------------------- */
#define PWM_BASE_REF            1000    /* Half of PWM period                     */
#define PWM_F_REF               0       /* Default PWM waveform value             */
#define PWM_DTM_REF             200     /* Dead time                              */

#define INTERVAL_BASE           10      /* 1ms = 100[us] * INTERVAL_BASE          */
#define TIME_OUT                10000   /* 1s  = 100[us] * TIME_OUT               */
                                        /* Dependent on inverter timer setting    */

/* ------------------------------------------------- */
const unsigned char  sync_tbl[2][6] = {     /* Synchronous start excitation pattern */
   {P_T1, P_T6, P_T5, P_T4, P_T3, P_T2},    /* for LowVoltage                       */
   {P_T1, P_T2, P_T3, P_T4, P_T5, P_T6}
};

/* ------------------------------------------------- */
char                sys_flag;               /* System operation status          */
int                 m_speed;                /* Current number of revolutions    */
static char         cw_ccw_flag;            /* Rotation direction status        */
char                err_flag = FLG_OFF;     /* Error flag                       */
unsigned char       maxed_flags;            /* bits to specify if on limits: bits 7-6-5 are RPM-Current-Volts respectively */
static char         up_flag = FLG_OFF;      /* Speed update check flag          */

char                stop_wait;              /* Stop wait flag                   */
char                cw_ccw_wait;            /* Reversal wait flag               */

static char         cmd_mode = SPEED_CMD;

static float        mvn_f;                  /* Manipulated variable             */
static float        en, en_1, en_2;         /* Deviation                        */
static float        ec, ec_1, ec_2;         /* Deviation                        */

int                 maxspeed = MAXSPEED_REF;
int                 minspeed = MINSPEED_REF;

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

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

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

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

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

/* ------------------------------------------------- */
/* startup lookup tables                */
#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;                /* Excitation pattern                                 */
static char         sync_flag;              /* Identification flag during synchronous start       */
static unsigned int sync_cnt;               /* Excitation switching management (x100 us)          */
static unsigned int sync_def;               /* Excitation switching time (x100 us)                */

static unsigned int cy_time = 150 * INTERVAL_BASE;
static unsigned int int_cnt;                /* Timeout counter                */
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;              /* Time until 30-degree delay      */
static unsigned int     clk_cnt;                /* Speed information               */
static unsigned int     new_clk, old_clk;

int                     m_speed;
int                     speed_ref = 0;          /* Specified speed                 */
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; /* Half of PWM period: Fixed to 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);

/* ================================================== */
/* ----------------------------------------------------
   Motor control function section disclosed to users
  ---------------------------------------------------- */

/* --------------------------------
   Initial settings of functions related to motor control
  -------------------------------- */
void
motor_init(void) {
   init_PORT();         /* Port setting                */
   init_OSC();          /* Oscillator setting          */
   init_WDTM();         /* WDTM setting                */
   init_TW0();          /* Inverter setting            */
   init_TM00();         /* Timer TM00 setting          */
   init_RTPM01();       /* Real-time port setting      */
   init_AD();           /* A/D setting                 */
   init_TM50();         /* Timer TM50 setting          */
   init_TM51();         /* Timer TM51 setting          */

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

   start_TM51();
   INTTM51_on();

   start_AD();          /* Starts A/D operation */
   INTP0_on();          /* External interrupt INTP0 setting for overcurrent detection */

   EI();
}

/* --------------------------------
   Motor start processing
  -------------------------------- */
void
motor_start(void) {
   init_openloop();
```

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

    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();                  /* Unmasks carrier synchronization interrupt */
    INTTW0CM3_on();                 /* Unmasks A/D trigger interrupt             */

    set_TW0(pwm_base - pwm_ff);     /* Changes 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]);
}

/* ---------------------------------
    Motor stop processing
   --------------------------------- */
void
motor_stop(void) {
    stop_wait = FLG_ON;
}

/* ---------------------------------
    Motor rotation direction change processing
   --------------------------------- */
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 operation processing
   --------------------------------- */
void
motor_pid(void) {
    unsigned int tmp, new_tmp, old_tmp;
    static float dl_ref;

    if(speed_flag == FLG_ON){           /* Speed is already measured */
        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-degree shift time */
                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-degree shift time */
            };

            up_flag = FLG_ON;
```

```
        }else{
            start_flag = FLG_OFF;          /* First speed information is not used */
        };
    };

    if(pid_flag == SET) {
        if(pid_cnt >= cy_time){            /* cy_time * 1 ms has elapsed          */
            pid_cnt = 0;
            if(up_flag == FLG_ON) {        /* Speed is updated                    */
                up_flag = FLG_OFF;

                if((sys_flag  != FLG_OFF) &&          /* Being started                          */
                   (stop_wait != FLG_ON) &&           /* Not waiting for stop                   */
                   (cw_ccw_wait != FLG_ON)){          /* Not waiting for stop of reverse rotation */

                    maxed_flags = 0;                  /* reset all flags here */
                    switch(cmd_mode) {
                    case V_CMD:                       /* voltage control: pwm set by GUI */
                        /* no loop on speed */
                        speed_ref = m_speed;
                        /* no loop on current */
                        I_ref = I_measured;
                        /* adjust setpoint (acceleration limits) */
                        if ((pwm_ff_o - pwm_ff) > dpwm_ff_max) {
                            pwm_ff += dpwm_ff_max;
                            maxed_flags |= 0x20;          /* on limit, set bit 5 */
                        } else if ((pwm_ff_o - pwm_ff) < -dpwm_ff_max) {
                            pwm_ff -= dpwm_ff_max;
                            maxed_flags |= 0x20;          /* on limit, set bit 5 */
                        } else {
                            pwm_ff = pwm_ff_o;
                        };
                        break;

                    case I_CMD:                       /* current control */
                        /* current control mode: current control set by GUI */
                        /* no loop on speed */
                        speed_ref = m_speed;
                        /* adjust setpoint (acceleration limits) */
                        if ((I_ref_o - I_ref) > dI_ref_max) {
                            I_ref += dI_ref_max;
                            maxed_flags |= 0x40; /* on limit, set bit 6 */
                        } else if ((I_ref_o - I_ref) < -dI_ref_max) {
                            I_ref -= dI_ref_max;
                            maxed_flags |= 0x40; /* on limit, set bit 6 */
                        } else {
                            I_ref = I_ref_o;
                        };
                        break;

                    case SPEED_CMD:                   /* control speed */
                        /* adjust setpoint (acceleration limits) */
                        if ((speed_ref_o - speed_ref) > dspeed_ref_max) {
                            speed_ref += dspeed_ref_max;
                            maxed_flags |= (unsigned char)(0x80); /* on limit, set bit 7 */
                        } else if ((speed_ref_o - speed_ref) < -dspeed_ref_max) {
                            speed_ref -= dspeed_ref_max;
                            maxed_flags |= (unsigned char)(0x80); /* on limit, set bit 7 */
                        } else {
                            speed_ref = speed_ref_o;
                        };

                        en_2 = en_1;   /* last last error */
                        en_1 = en;     /* last error */
                        en   = (float)(speed_ref - m_speed); /* up-to-date error */
                        /* make current command for inner loop */

                        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;          /* on limit, set bit 6 */
                        } else {
                            if (dI_ref < -dI_ref_max) {
                                dI_ref = -dI_ref_max;
                                maxed_flags |= 0x40;      /* on limit, set bit 6 */
                            };
                        };
                        I_ref += dI_ref;
                        break;
                    };
                };
            };
        };
    };
}

/*  ---------------------------------
    Motor parameter setting
  --------------------------------- */
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);
}

/* ============================================================ */
/* ------------------------------------------------------------
   Motor control function section not disclosed to users
   ------------------------------------------------------------ */
/* --------------------------------
   Restart from reverse rotation stop
   -------------------------------- */
```

```
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);                 /* Changes PWM         */

    set_RTPM01(sync_tbl[cw_ccw_flag][3]);
    wait(2);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);       /* Determines rotor position */
    wait(20);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);

    WDTE = WDTE_CLR;
}

/* ---------------------------------
    System stop
   --------------------------------- */
static void
system_stop(void) {
    pwm_ff = 0;
    INVERTER_SW  = INV_OFF;         /* INVERTER disable       */
    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;                    /* Speed 0  */
}

/* ---------------------------------
   Initial open loop settings
   --------------------------------- */
static void
init_openloop(void) {
    unsigned char jj;
    float           t_def, RPM_def;
    int             ii;

    ii      = 0;
    t_def = 0;

   /*
     loop through ii's
     calculate how frequently table is used.
    */
    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); /* timer interval */
        ii++;
    };

   /* lookup table is i_sync_skip times too small. */
    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) { /* first segment */
            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 {                    /* 2nd segment */
            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); /* next time */

        if (jj >= (TABLESIZE-1)){
            t_def = (float)(t_end + 1.0); /* for memory protection-- shouldn't be necessary */
        };

        ii++;
    };

    i_sync_max = ii - 1; /* table size */
}


/* ---------------------------------
    Filter
  --------------------------------- */
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 bit A/D reading */
    };
    x_filt = reg>>6;
    I_measured = (float)x_filt;
    WDTE = WDTE_CLR;

}

/*  ---------------------------------
    Current minor loop PID
  --------------------------------- */
static void
current_pwm(void) {
#define STARTLAG 250

    unsigned int     pwm_tmp;

    if (startup_method == CURRENT_START) {
        startdelay = STARTLAG;          /*  don't do this for current startup scheme */
    } else {
        /* running on sensors */
        startdelay++;
        if (startdelay > STARTLAG) {
            startdelay = STARTLAG;
        };
    };

    if((sys_flag  != FLG_OFF) &&        /* being started */
       (stop_wait != FLG_ON) &&         /* not waiting for motor to stop */
       (cw_ccw_wait != FLG_ON) &&   /* not waiting for motor to stop reversing */
       (cmd_mode != V_CMD) &&           /* not controlling PWM directly */
       (startdelay == STARTLAG)
       ) {

       /* PID control for Current to PWM */
        ec_2 = ec_1;  /* last last current error */
        ec_1 = ec;      /* last current error;     */
        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));

       /* no limit on dpwm here! */
        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 { /* not controlling current for various reasons */
        I_ref = I_measured;
        speed_ref = m_speed;
        ec = 0;
        ec_2 = 0;
        ec_1 = 0;
    };
}

/* ---------------------------------
    Port settings
  --------------------------------- */
static void
init_PORT(void) {
    INTP0 = IN;                                 /* Interrupt of overcurrent notification   */

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

}

/* ---------------------------------
    Clock switching
```

```
   ---------------------------------- */
static void
init_OSC(void) {
   IMS = IMS_DATA;          /* Switches memory size */
   WDTE = WDTE_CLR;
   while(OSTC != 0x1f);     /* Waits for oscillation stabilization */
   PCC   = 0x00;            /* Sets division ratio */
   MCM.0  = 1;              /* X1 input clock */
   VSWC.1 = 1;
}

/* ----------------------------------
   Inverter timer
   ---------------------------------- */
static void
init_TW0(void) {
   TCL02        =  0;      /* Count clock: 20 MHz                    */
   TCL01        =  0;
   TCL00        =  0;
   IDEV02       =  0;      /* Generates interrupt every time         */
   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;                    /* Dead time */
   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;                    /* Starts timer */
}

static void
stop_TW0(void) {
   TW0C.7 = CLEAR;                  /* Stops timer */
}

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

/* ----------------------------------
   16-bit timer

   CR01    Real-time output port trigger
   ---------------------------------- */
static void
init_TM00(void) {
   CRC00  = 0x00;           /* CR01 compare register     */
   PRM001 = SET;
   PRM000 = CLEAR;     /* Count clock: 78.125 kHz */
}

static void
start_TM00(void) {
   TMIF00 = CLEAR;
   TMC00  = 0x04;           /* Event counter mode      */
}

static void
stop_TM00(void) {
   TMC00 = CLEAR;       /* Stops timer               */
}

/* ----------------------------------
   Real-time port
   ---------------------------------- */
static void
init_RTPM01(void) {
   RTPM01  = 0x3f;                  /* Real-time output mode   */
   RTPC01  = 0x20;                  /* 6 bits x 1 channel      */
   DCCTL01 = 0xc0;                  /* PWM modulation output */
   RTBL01  = 0x3f;                  /* Output buffer           */
}

static void
start_RTPM01(void) {
   RTPC01.7 = SET;                  /* Enables operation */
}

static void
stop_RTPM01(void) {
   RTPC01.7 = CLEAR;                /* Disables operation */
}

static void
set_RTPM01(unsigned char data) {
   RTBL01 = data;                   /* Sets excitation pattern          */
   CR01   = TM00 + CR01_ADD;
   TMIF01 = CLEAR;                  /* Clears interrupt notification flag */
}
```

```
/* ----------------------------------
   AD
  ---------------------------------- */
static void
init_AD(void) {
   ADS   = AD_ISHUNT;              /* SHUNT */
   ADM   = 0x04;                   /* 3.6us */
}

static void
start_AD(void) {
   ADIF = CLEAR;                   /* Clears interrupt notification flag  */
   ADCS = SET;                     /* Enables conversion operation      */
   while(ADIF != SET);             /* Waits for interrupt notification      */
}

/* ----------------------------------
   Watchdog timer
  ---------------------------------- */
static void
init_WDTM(void) {
   WDTE = WDTE_CLR;
   WDTM = WDTM_SET;
}

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

static void
start_TM51(void) {
   TMIF51 = CLEAR;                 /* Clears interrupt notification flag */
   TCE51  = SET;                   /* Enables count operation            */
}

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

/* ----------------------------------
   Wait
   TM50 is used
  ---------------------------------- */
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;
   };
}

/* ======================================================== */
/*
   Interrupt enable/disable processing
 */
/* ----------------------------------
   Overcurrent interrupt enable
  ---------------------------------- */
static void
INTP0_on(void) {
   PPR0 = 0;                       /* Priority        */
   EGN0 = 1;                       /* Falling edge  */
   PIF0 = CLEAR;                   /* Clears flag     */
   PMK0 = CLEAR;                   /* Unmasks       */
}

/* ----------------------------------
   Carrier synchronization interrupt enable
  ---------------------------------- */
static void
INTTW0UD_on(void) {
   UDIFW0 = CLEAR;                 /* Clears request flag  */
   UDMKW0 = CLEAR;                 /* Enables interrupt     */
}

/* ----------------------------------
   Carrier synchronization interrupt disable
  ---------------------------------- */
static void
INTTW0UD_off(void)
{
   UDMKW0 = SET;                   /* Disables interrupt   */
   UDIFW0 = CLEAR;                 /* Clears request flag */
}
```

```
/* ----------------------------------
  A/D trigger interrupt enable
  ---------------------------------- */
static void
INTTW0CM3_on(void) {
    CM3IFW0 = CLEAR;
    CM3MKW0 = CLEAR;
}

/* ----------------------------------
  A/D trigger interrupt disable
  ---------------------------------- */
static void
INTTW0CM3_off(void) {
    CM3MKW0 = SET;
    CM3IFW0 = CLEAR;
}

/* ----------------------------------
    Current minor interrupt enable
  ---------------------------------- */
static void
INTTM51_on(void) {
    TMIF51 = CLEAR;
    TMMK51 = CLEAR;
}


/* ================================================== */
/*
    Interrupt servicing
*/
/* ----------------------------------
    Hardware overcurrent detection
  ---------------------------------- */
__interrupt void
int_faulta(void) {
    err_flag = ERROR_OC;
    system_stop();                          /* System stop */
}

/* ----------------------------------
    Carrier synchronization interrupt (valley side)
  ---------------------------------- */
__interrupt void
int_carrier(void) {
    static char             cnt_flag;
    static unsigned int  cnt;
    unsigned char        j_sync;

  /* convenient macros */
#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;


   /* function start */
```

```
    print_cnt++;                                    /* Speed display timing   */
    if(pid_flag == SET) {
       pid_cnt++;                                   /* PID control timing    */
    };

    if(sync_flag == FLG_OFF){                       /* Motor is being rotated by BEMF signal */
       if(++int_cnt > TIME_OUT){                    /* Motor does not rotate */
          if(stop_wait == FLG_ON){                  /* Waits for stop */
             system_stop();
             return;
          }else if(cw_ccw_wait == FLG_ON){          /* Waits for stop of reverse rotation */
             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){                      /* During 30-degree shift */
          if(--cnt == 0){                           /* 30-degree shift has been completed */
             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){                    /* Detects zero-cross */
             ad_flag = FLG_OFF;
             cnt_flag = FLG_ON;
             cnt = (speed_cnt * 7) / 8;
          };
       };
       return;
    };

    if(stop_wait != FLG_OFF){                       /* Waits for stop */
       system_stop();
       return;
    };
    if(cw_ccw_wait != FLG_OFF){                     /* Waits for stop of reverse rotation */
       system_restart();
       return;
    };

    if(sync_flag == FLG_START){                     /* During synchronous start       */
       if(--sync_cnt == 0){                         /* Switches excitation         */
          speed_flag = FLG_ON;
          ad_read_flag = FLG_ON;
          if(i_sync == i_sync_max){
             /* Goes through open loop and to PID control pre-processing */
             sync_flag = FLG_WAIT;                  /* Prepares for end of synchronous start */
             m_speed   = (int)RPM2;                 /* Number of revolutions during control switching */
             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;          /* Sets excitation switching time */
       };
    }else{
       if(cnt_flag == FLG_ON){          /* Waits for 30 degrees to elapse */
          if(--cnt == 0){               /* Switches */
             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){        /* Zero-cross occurs */
             speed_cnt = sync_def>>1;
             cnt = speed_cnt;
```

```
            cnt_flag = FLG_ON;          /* Shifts by 30 degrees */
            ad_flag = FLG_OFF;
            int_cnt = 0;
          }else if(--sync_cnt == 0){    /* Switches at synchronous start */
            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){     /* Motor does not rotate */
              err_flag = ERROR_MOTOR;
              system_stop();
              return;
            };
          };
        };
    };
    WDTE = WDTE_CLR;

    return;
}


/* --------------------------------
  Non-excited phase voltage A/D conversion
  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;/* Bypasses channel */

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

    ADS = AD_ISHUNT;
    ADIF = CLEAR;
    while(ADIF != SET);
    ad_data_shunt = ADCR;               /* Saves value */

    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){           /* Downward slope      */
                        old_clk         = new_clk;
                        new_clk         = TM00;
                        clk_cnt         = ad_int_cnt;
                        ad_int_cnt      = 0;
                        ad_read_flag    = FLG_OFF;      /* Aborts A/D conversion processing */
                        ad_flag         = FLG_ON;       /* Zero-cross occurs */
                        speed_flag      = FLG_ON;       /* Updates speed information   */
                    };
                    break;

                case 1:
                case 3:
                default:
                    if(ad_data > ADDATA_E_2){           /* Upward slope      */
                        ad_read_flag    = FLG_OFF;      /* Aborts A/D conversion     */
                        ad_flag         = FLG_ON;       /* Zero-cross occurs */
                    };
                };
            }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;/* Returns channel */
ADIF = CLEAR;

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

WDTE = WDTE_CLR;                    /* Clears watchdog timer   */

return;
}

/* ---------------------------------
   Interrupt for TM51 overflow
  --------------------------------- */
__interrupt void
int_TM51(void) {

   filters();

  /* check over current limit on SOFTWARE */
#if 1
   if(I_measured > CLIMIT) {
      err_flag = ERROR_S_OC;
      system_stop();
   };
#endif

   current_pwm();
}
```

4.17 Reference Program Source File

```
/*

   BLDCM 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target    : uPD78F0714 motor starter kit
   date      : 2007/05/11
   filename: main_mcio.c

   NEC Micro Systems,Ltd
*/

#pragma sfr

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

/*
  Main function
*/
void
main(void) {
   unsigned char sw;
   unsigned char tmp_sw = 0;

   /* Initial system settings */
   motor_init();
   init_PORT();
   init_TM50();

   /* Motor parameter settings */
   motor_pset(MODE, V_CMD);
   motor_pset(PID_INTERVAL, 150);   /* PID control interval: 150 ms */

   motor_pset(KRP,  50);                /* RPM-Kp setting.  Sets value multiplied by 1000          */
   motor_pset(KRI,   20);               /* RPM-Ki setting.  Sets value multiplied by 1000          */
   motor_pset(KRD,  10);                /* RPM-Kd setting.  Sets value multiplied by 1000          */

   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){    /* Function being stopped */
                  ad_flag = FLG_OFF;     /* Restores function   */
               }else{                    /* Function being operated */
                  ad_flag = FLG_ON;      /* Stops function   */
               };
               break;
            default: ;
            };
         };
      };
      tmp_sw = sw;
   };
   return;
}
```

```
/*


   BLDCM 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target    :  uPD78F0714 motor starter kit
   date      : 2007/05/10
   filename: 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       /* Input */
#define  OUT            0       /* Output */

#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        /* Identifies pressed switches  */
#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 display data  */
#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 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target    :  uPD78F0714 motor starter kit
   date      : 2007/05/11
   filename: 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] = {        /* For displaying numerical values */
 LED_0, LED_1, LED_2, LED_3, LED_4,
 LED_5, LED_6, LED_7, LED_8, LED_9
};


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

/*
   Error display
*/
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();
}

/*
   Reading switches in MC-IO
*/
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;                          /* Reads switch                                  */
    if(data != 0xf){
        for(i=0;i<KEY_WAIT;){           /* Eliminates chattering.  Must be adjusted.     */
            tmp = SW;                   /* Re-reads switch                                */
            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){    /* No switch is pressed */
         start_stop_flag = FLG_OFF; /* Prevents toggling between START and STOP */
      };

      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
}

/*
  Displaying argument values with LEDs
*/
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){                        /* Most significant digit is not 0 */
         flg = FLG_ON;                     /* Numerical value is displayed on most significant digit */
         led_set(0, led_data[tmp]);
      }else{
         led_set(0, LED_);
      };
      data %= 1000;
      tmp = (unsigned int)(data / 100);
      if((tmp != 0) || (flg == FLG_ON)){     /* Value is not 0 or number has already been displayed */
         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)){     /* Value is not 0 or number has already been displayed */
         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);
      };
}

/*
   Displaying data on 8-segment LED
   no:   Location of LED to be displayed
   data: Data to be output
*/
static void
led_set(char no, unsigned char data)
{
      unsigned char p;

      P6 = 0x00;
      P4 = data;

      switch(no){                    /* Location to be displayed */
      case 0:  p = 0x80; break;  /* Left edge                    */
      case 1:  p = 0x40; break;
      case 2:  p = 0x20; break;
      default: p = 0x10; break;  /* Right edge                   */
```

```
    }
    P6 = p;
}

/*
    Reading voltage of speed specification variable resistor
    Converting into specified speed
*/
#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);
    }
}

/*
  Retrieving volume resistance value via A/D
*/
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);
}

/*
    Time adjustment    ms
*/
static void
wait(int cnt) {
    int i;

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

/*
  Startup display
*/
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);
}

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

    LD_LED0 = OUT;          /* LED selection: output */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;          /* LED display data: output */
}

/*
    8-bit timer 50
*/
```

```
void
init_TM50(void)
{
    TCL50 = 0x06;
    CR50  = 78;                 /* 1 ms */
}

static void
start_TM50(void)
{
    TMIF50 = CLEAR;         /* Clears interrupt notification flag    */
    TCE50  = SET;           /* Starts timer                          */
}

static void
wait_TM50(void)
{
    while(TMIF50 != SET); /* Waits for interrupt notification      */
    TCE50  = CLEAR;         /* Stops timer                           */
}

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

APPENDIX A PROGRAM EXAMPLE

A program example when controlling this system by connecting the separately provided motor operation panel (GUI program) and the RS-232C pin located on the motor control I/O board is included.

A.1 GUI Reference Program Function List

Table A-1. GUI Reference Program Functions

Function Name	Function	Purpose
uart_set()	UART setting	Sets UART function.
get_uart()	Command read instruction	Instructs reading of communication data in UART communication and returns operation information corresponding to read data.
uart_read()	Reading UART data	Instructs reading of UART receive data.
uart_wait()	Reading (provided with receive wait function) of UART data	Function used for successive reception of commands configured of multiple bytes in UART communication
uart_send()	UART data transmission	Instructs transmission in UART communication.
int_uart()	UART reception	Interrupt function performing reception in UART communication
set_error()	Storing error status	Stores error status in UART communication data.
led_set()	LED display	Displays data with LEDs.
startup_disp()	LED display at start	LED display at start
init_PORT()	Port setting	Performs port setting on the MC-IO board.

A.2 GUI Reference Program Constant List

A.2.1 Internal constants

Table A-2. GUI Reference Program Internal Constants (1/2)

Name	Meaning	Setting Value	Remark
IN	For port setting	1	Used to specify port function
OUT	For port setting	0	
CLEAR	For register bit setting	0	Used to access bits of registers
SET	For register bit setting	1	
LED_0	LED display data	0xc0	Displays “0”.
LED_1		0cf9	Displays “1”.
LED_2		0xa4	Displays “2”.
LED_3		0xb0	Displays “3”.
LED_4		0x99	Displays “4”.
LED_5		0x92	Displays “5”.
LED_6		0x82	Displays “6”.
LED_7		0xf8	Displays “7”.
LED_8		0x80	Displays “8”.
LED_9		0x98	Displays “9”.
LED_O		0xc0	Displays “0” instead of “O”.
LED_I		0xcf	Displays “I”.
LED_C		0xc6	Displays “C”.
LED_H		0x89	Displays “H”.
LED_A		0x88	Displays “A”.
LED_L		0xc7	Displays “L”.
LED_		0xff	Displays “ ”.
LED_S		0x92	Displays “S”.
LED_E		0x86	Displays “E”.
LED_F		0x8e	Displays “F”.
LED_P		0x8c	Displays “P”.
LED_dot		0x7f	Displays “.”.
LD_LED0	Port control register	PM64	Port for LED selection
LD_LED1		PM65	
LD_LED2		PM66	
LD_LED3		PM67	
LD_DATA		PM4	Port that outputs data to LED
START_TR	For status setting	0x01	Starts control.
STOP_TR		0x02	Stops control.
FORWARD_TR		0x04	Changes rotation to CW.
REVERSE_TR		0x08	Changes rotation to CCW.
MODE_TR		0x10	State where MODE switch is pressed
RTS	Communication ports	P11	
CTS		P10	
RX_BUFF_SIZE	Communication buffer size	6	
MD_ERROR_HALL	Communication information	0xF0	

Table A-2. GUI Reference Program Internal Constants (2/2)

Name	Meaning	Setting Value	Remark
MD_ERROR_OC	Communication information	0xF1	
MD_ERROR_MOTOR		0xF2	
MD_CMD_START	Communication commands	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 Reference Program Variable List

A.3.1 Internal variables

Table A-3. GUI Reference Program Internal Variables

ad_flag	char	Speed change flag	Limits specified-speed change function.
led_data[ ]	char	LED output data	Value displayed with LEDs

A.4 GUI Reference Program Source Program

```
/*


   BLDCM 120-degree excitation method without position sensor (BEMF_AD conversion)
   Module conversion supported
   Current control version

   target   :  uPD78F0714 motor starter kit
               Compiler option GUI: GUI is used
   date     : 2007/04/22
   filename: main_gui.c

   NEC Micro Systems,Ltd
*/

#pragma sfr

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

/*
  Main function
*/
void
main(void) {
   unsigned char sw;
   unsigned char tmp_sw = 0;

   /* Initial system settings */
   motor_init();
   init_PORT();
   uart_set();
   startup_disp();

   /* Motor parameter settings */
   motor_pset(PID_INTERVAL, 150);   /* PID control interval: 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 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target   : uPD78F0714 motor starter kit
              Compiler option GUI: GUI is used

   date     : 2007/04/03
   filename: sub_gui.h

   NEC Micro Systems,Ltd
*/

#define  IN                 1           /* Input */
#define  OUT                0           /* Output */

#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 failure          */
#define MD_ERROR_OC                 0xF1    /* Overcurrent              */
#define MD_ERROR_MOTOR              0xF2    /* Motor failure            */

#define RTS                         P11
#define CTS                         P10
#define RX_BUFF_SIZE                6       /* UART00 receive buffer size */

#define MD_CMD_START                0x20    /* START                    */
#define MD_CMD_STOP                 0x21    /* STOP                     */
#define MD_CMD_RESET                0x2f    /* RESET                    */
#define MD_CMD_GETID                0x10    /*ID request                */
#define MD_CMD_GETVER               0x11    /* Version request          */
#define MD_CMD_SETSSPEED            0x30    /* Specified-speed change   */
#define MD_CMD_GETSSPEED            0x31    /* Specified-speed read     */
#define MD_CMD_SETPIDPARAM          0x40    /* PID change               */
#define MD_CMD_GETPIDPARAM          0x41    /* PID read                 */
#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    /* Actual-speed read   */

#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    /* Firmware identification ID   */
#define VER_MAJOR                   0x01    /*Version information */
#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 display data  */
#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 120-degree excitation method without position sensor (BEMF_AD conversion)

   Module conversion supported
   Current control version

   target    :  uPD78F0714 motor starter kit
   date      : 2007/04/03
   filename: sub_gui.c

   NEC Micro Systems,Ltd
*/

#pragma sfr


#pragma INTERRUPT INTSR00          int_uart  rb1 /* For UART00 command reception */


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


unsigned char uart00_data[RX_BUFF_SIZE];        /* UART00 receive buffer   */
unsigned char read_p, write_p;                  /* Buffer pointer          */
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 settings
*/
void
uart_set(void) {
   PM10        = IN;
   PM11        = OUT;
   PM13        = IN;
   PM14        = OUT;
   RTS         = 1;
   P14         = 1;
   BRGC00      = 0x56;                 /* 115200 */
   PS001       = CLEAR;
   PS000       = CLEAR;
   CL00        = SET;                  /* 8bit */
   SL00        = CLEAR;
   POWER00     = SET;
   TXE00       = SET;
   RXE00       = SET;
   STIF00      = SET;
   SRIF00      = CLEAR;
   SRMK00      = CLEAR;                /* Enables receive interrupt */
   RTS         = 0;
   read_p      = 0;
   write_p     = 0;
}

/*
   Acquiring operation instructions via UART communication
 */
unsigned char
get_uart(void) {
   unsigned char data;
   int           ss, ref;
   unsigned char hi, lo;
   float         shunt_volt;
   float         shunt_command;
   float         tmp_float;
   int           tmp_int;

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

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

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

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

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

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

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

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

case MD_CMD_SETSSPEED:          /* Set Setting Speed */
   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);   /* speed control mode */
   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;         /* open loop on speed */
   motor_pset(MODE, I_CMD);     /* current control mode */
            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;         /* open loop on speed */
   motor_pset(MODE, V_CMD);     /* voltage control mode */
            break;

case MD_CMD_GETSSPEED:          /* Get Setting Speed */
   uart_send(data);
   ss = speed_ref;
   if(cw_ccw_flag == CCW){
      ss = ~ss + 1;
   }
```

```
    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:         /* Get shunt a/d (current) */
    shunt_volt = I_measured*10.0;    /* 10 bits */
    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:          /* Get shunt a/d (current) */
    uart_send(data);
    shunt_command = I_ref*10.0;
    ss = (int)shunt_command;         /* 10 bits (TEMPORARY)    */
    lo = (unsigned char)(ss&0xff);
    hi = (unsigned char)(ss>>8)&0xff;
    uart_send(lo);
    uart_send(hi);
    break;

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

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

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

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

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

case MD_CMD_GETSPEED:           /* Reads rotation speed */
    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{  /* Waits for stop of reverse rotation */
                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:          /* open-loop startup parameters defined by gui*/
        GET16to(ref);
        startup_method = (unsigned char)(ref);
        GET16to(ref);
        t_knee = (float)ref/1000.0;
        GET16to(ref);
        t_end = (float)ref/1000.0;
        GET16to(RPM0);
        GET16to(RPM1);
        GET16to(RPM2);
        GET16to(I_ref0);
        GET16to(I_ref1);
        GET16to(I_ref2);
        GET16to(PWM0);
        GET16to(PWM1);
        GET16to(PWM2);
        uart_send(data);
        break;

    case MD_CMD_GETSTART:          /* send startup parameters to gui*/
        uart_send(data);
        uart_send(startup_method);
        uart_send(0x00);           /* char sent as int for convenience in gui */
        SETUART(t_knee*1000);
        SETUART(t_end*1000);
        SETUART(RPM0);
        SETUART(RPM1);
        SETUART(RPM2);
        SETUART(I_ref0);
        SETUART(I_ref1);
        SETUART(I_ref2);
        SETUART(PWM0);
        SETUART(PWM1);
        SETUART(PWM2);
        break;

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

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

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

    default:
        ;
    };

    return(data);
}

/*
    Returning UART00 receive buffer data
*/
static unsigned char
uart_read(void) {
    unsigned char data = 0;

    if(read_p != write_p){         /* Receive data is present        */
        data = uart00_data[read_p++];  /* Extracts data        */
        read_p %= 6;               /*Updates read pointer   */
    }
    return(data);
}

/*
    Waiting for UART00 reception
*/
static unsigned char
uart_wait(void) {
    unsigned char data;

    while(read_p == write_p);      /* Waits until data is input to buffer   */
```

```
    data = uart00_data[read_p++];       /* Extracts data                */
    read_p %= 6;                        /*Updates read pointer          */
    return(data);
}

/*
    Transmission from UART00
*/
static void
uart_send(unsigned char data) {
    while(STIF00 != SET);               /* Waits for transmission completion */
    STIF00 = CLEAR;
    TXS00 = data;                       /* Transmits                        */
}

/*
    For UART00 command reception
*/
__interrupt void
int_uart(void) {
    unsigned char tmp;

    tmp = ASIS00;                       /* Reads error status               */
    uart00_data[write_p++] = RXB00;     /* Stores status in receive buffer  */
    write_p %= RX_BUFF_SIZE;            /* Updates write pointer            */
}


/* ---------------------------------------------------------- */
/*
   Error display
*/
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);
    };
}

/*
    Displaying data on 8-segment LED
    no:   Location of LED to be displayed
    data: Data to be output
*/
static void
led_set(unsigned char no, unsigned char data) {
    unsigned char p;

    P6 = 0x00;
    P4 = data;

    switch(no){                     /* Location to be displayed */
    case 0:  p = 0x80; break;  /* Left edge        */
    case 1:  p = 0x40; break;
    case 2:  p = 0x20; break;
    default: p = 0x10; break;  /* Right edge       */
    };
    P6 = p;
}

/*
  Startup display
*/
void
startup_disp(void) {
    led_set(0, LED_P);
    led_set(1, LED_C);
    led_set(2, LED_);
    led_set(3, LED_);
}

/*
    Watchdog timer
*/
void
```

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

/* ------------------------------------------------------------ */
/*
    Port settings
*/
void
init_PORT(void) {
    LD_LED0 = OUT;          /* LED selection: output */
    LD_LED1 = OUT;
    LD_LED2 = OUT;
    LD_LED3 = OUT;

    LD_DATA = OUT;          /* LED display data: output */
}
```

For further information,
please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
http://www.cn.necel.com/

Shanghai Branch
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai, P.R.China P.C:200120
Tel:021-5888-5400
http://www.cn.necel.com/

Shenzhen Branch
Unit 01, 39/F, Excellence Times Square Building,
No. 4068 Yi Tian Road, Futian District, Shenzhen,
P.R.China P.C:518048
Tel:0755-8282-9800
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
http://www.tw.necel.com/

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
http://www.kr.necel.com/

G0706