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.

RENESAS

Application Note

Motor Control by μPD78F0714

Sensorless (BEMF) 120° Excitation Method

μPD78F0714

Document No. U18051EJ2V0AN00 (2nd edition)
Date Published September 2007 N CP(K)

[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 May, 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 (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	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
SM+ System Simulator	Operation	U17246E
	User Open Interface	U17247E
ID78K0-QB Ver. 2.81 Integrated Debugger	Operation	U16996E
PM plus Ver.5.20		U16934E

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

Document Name	Document No.
QB-78K0KX1H In-circuit Emulator	U17081E

Documents related to flash memory writing

Document Name	Document No.
PG-FP3 Flash Memory Programmer User's Manual	U13502E
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 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 in the operating environment specified below.

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 V3.60
- RA78K0 assembler V3.70
- DF0714.78K device file V1.0

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. 3.60 C Compiler
- Each User's Manual of RA78K0 Ver. 3.70 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.

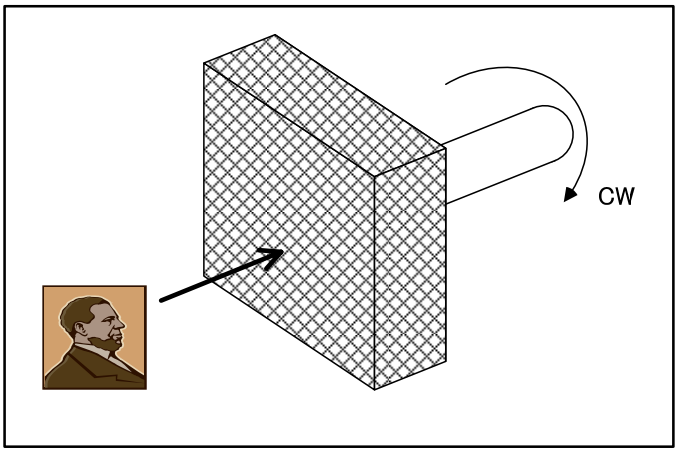
2.1 Rotation Direction Definition

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



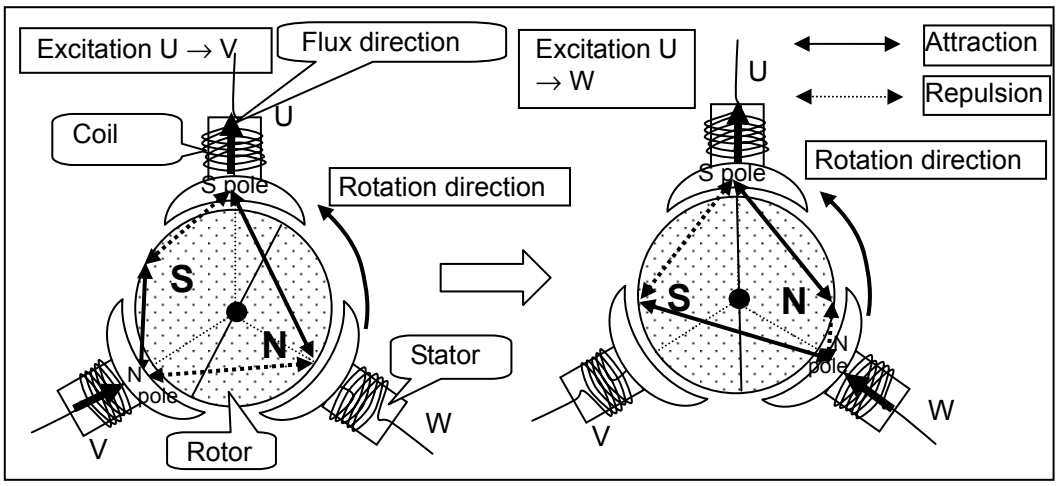
2.2 Rotation Principle

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

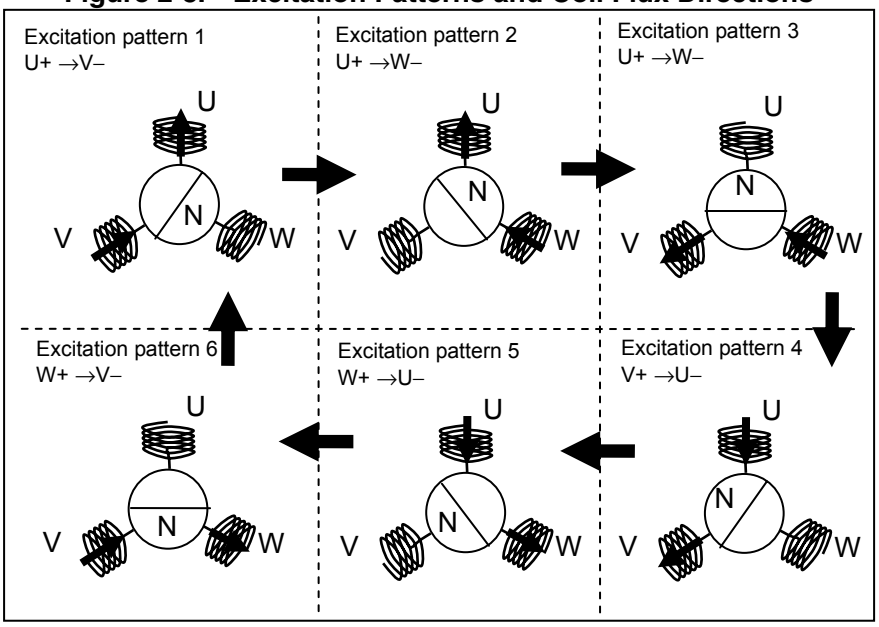


The polarity of the stator depends on the direction of the coiled winding.
When the magnetic polarity of the rotor is reversed, the rotation direction is reversed.

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



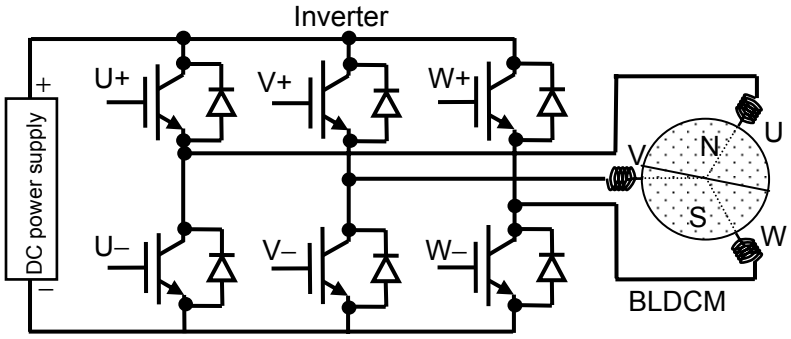
The polarity of the coils depends on the direction of the coiled winding.

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



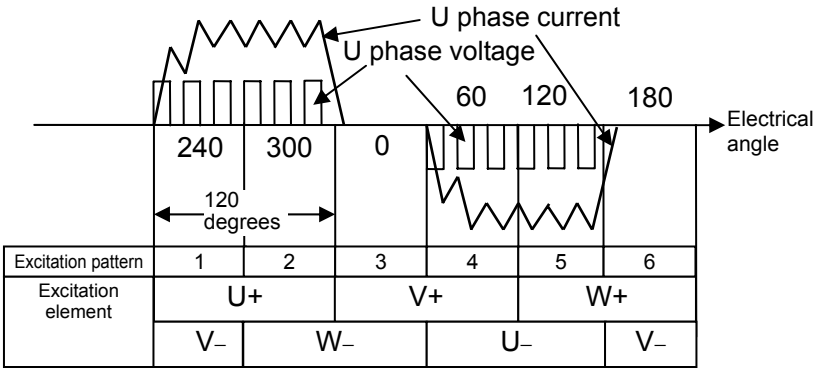
The six switching elements control the excitation time and excitation direction.

2.5 120° Excitation Method

The following figure shows the status (voltage, current) of phase U when a rectangular-waveform voltage is applied to each phase in the excitation patterns shown in 2.3. In phase U, a 120° excitation period and a 60° non-excitation period occur repeatedly.

This method, which performs excitation with the period of each phase being 120 degrees, is called the 120° excitation method.

Figure 2-5. 120° Excitation Pattern and Current



This system uses 120° excitation to drive a BLDCM.

2.6 Position Information

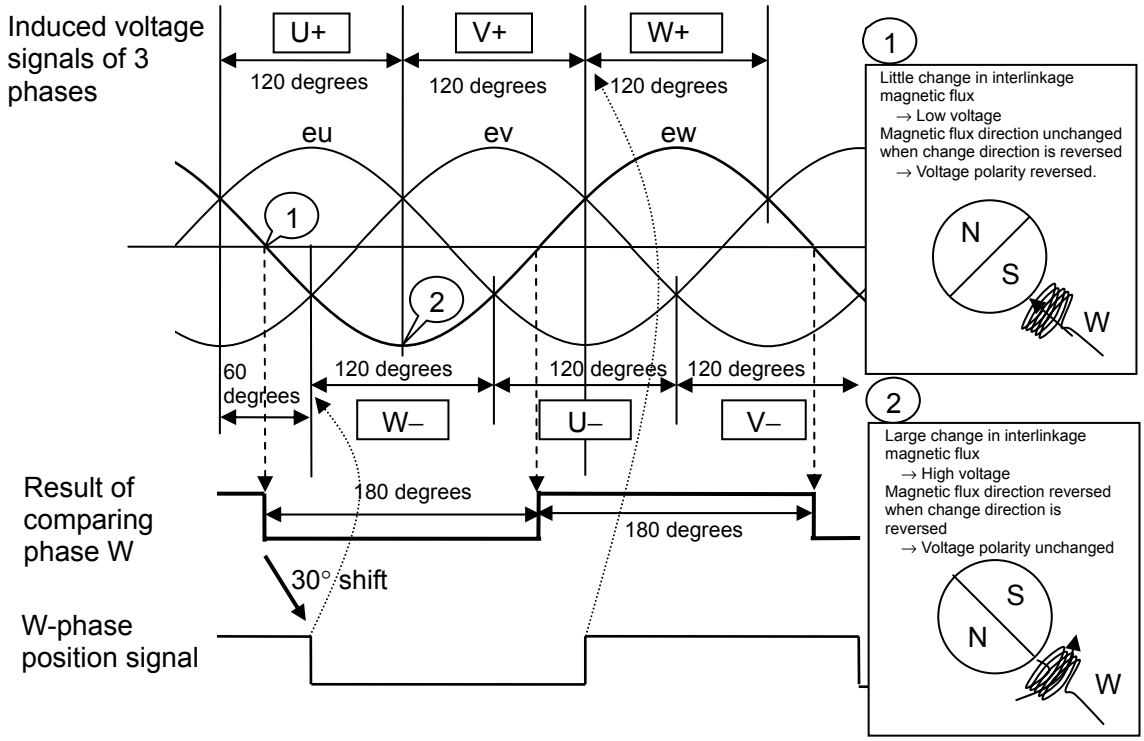
With the 120° excitation method, position information is required for switching the inverter.

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 (counter electromotive force) 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 counter electromotive force (Back EMF) of phase W that is open crosses zero can be detected.

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

Figure 2-6. 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.

When the rotor has four magnetic poles, the comparison result of phase W changes every 90 degrees and the excitation pattern is switched every 30 degrees.

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/2

2.7 Starting Method

The position detection method using an induced voltage cannot be used when the motor is stopped or rotates at a low speed. Therefore, the motor is started synchronously so that the excitation pattern is switched independently of the position and, when the rotation speed of the motor increases, the driving method is changed to one that uses comparison result signals.

2.8 Excitation Pattern Switching

After time equivalent to 30 degrees has elapsed since the comparison result of each phase has changed, excitation patterns that correspond to the comparison results are set.

2.9 Speed Detection

The rotation speed of the motor is derived from calculating the period of time over which the comparison result of each phase changes.

2.10 Voltage Control

The voltage to be applied to the motor coil is controlled by PWM (Pulse Width Modulation) which adjusts the conduction rate (average voltage) through a chopper operation of the conduction period of any of the switching elements at a high frequency, with a rectangular waveform of excitation at 120 degrees.

2.11 Speed Control

The speed is controlled by changing the voltage applied to the motor coil (PWM conduction rate) through PID control.

2.11.1 PID control

A PID control action 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 action derives the manipulated variable of the PWM duty factor from 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.

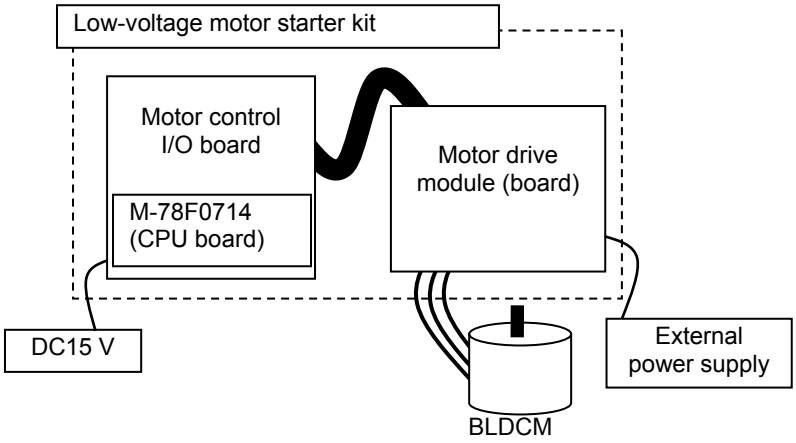
CHAPTER 3 SYSTEM OVERVIEW

This chapter presents an overview of the system.

3.1 Configuration

The following figure shows the configuration of this system.

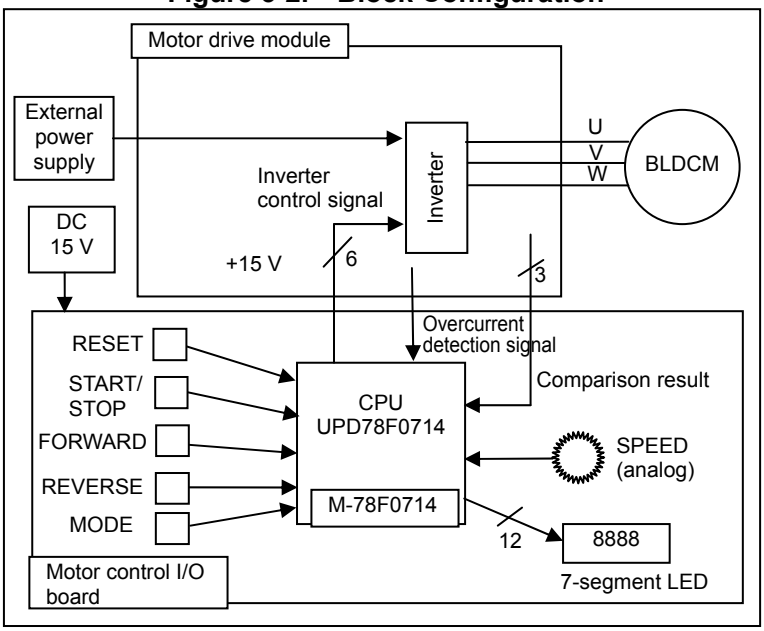
Figure 3-1. System Configuration



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)
MODE	SW5	Switching speed specification Switching speed display
SPEED	R52	Changing specified speed
7-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 1 second immediately after reset release.

Table 3-2 lists errors.

Table 3-2. Errors

Error	LED Indication	Situation
Overcurrent	0.C.	Inverter 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]
27 to 32	TW0TO0 to TW0TO5	3-phase PWM inverter selection
11	P01/INTP1	Comparison result signal (CMPU)
10	P02/INTP2	Comparison result signal (CMPV)
9	P03/INTP3	Comparison result signal (CMPW)
49 to 52	P64 to P67	7-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 to P47	Output data to 7-segment LED
12	TW0TOFFP/INTP0	Overcurrent detection (+5 V $\rightarrow$ 0 V)
60	ANI4	Speed change (R52)
20	P53/TI000/INTP5	Timer capture trigger
21	P54/TI00/TO00	Motor drive module control

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

3.3 Functions

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

Table 3-4. System Functions and Operation Overview

Function	Overview
Start (power supply)	• "SELF" is displayed with LEDs for 1 second. • 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 display with LEDs. • 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 in range of 200 to 3200. • No change if disabled with MODE switch.
	While motor is controlled: Changes specified speed. • Motor rotates at specified speed (average).
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.
No rotation	• If motor does not rotate after 1 second from START and if motor has stopped for 0.05 seconds or longer. • 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.

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 13 kHz (symmetrical triangular wave). • Carrier (modulation) synchronization interrupt is generated at intervals of 76.9 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.)
Wait processing	• For timing adjustment • 8-bit timer/event counter 50 (TM50, CR50)
Reading specified speed	• Converts voltage on variable resistor value into specified speed. • A/D converter (ANI4)
Capture interrupt	• For speed calculation • Interrupt function (TI000/INTP5)
H/W overcurrent interrupt	• Interrupt function (INTP0) • Occurrence of overcurrent in motor driver 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
INTP5	Detection of change in comparison result (speed calculation)	External pin
INTTW0UD	Occurrence of carrier synchronization interrupt	Underflow of inverter timer counter
RESET	Occurrence of reset	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.

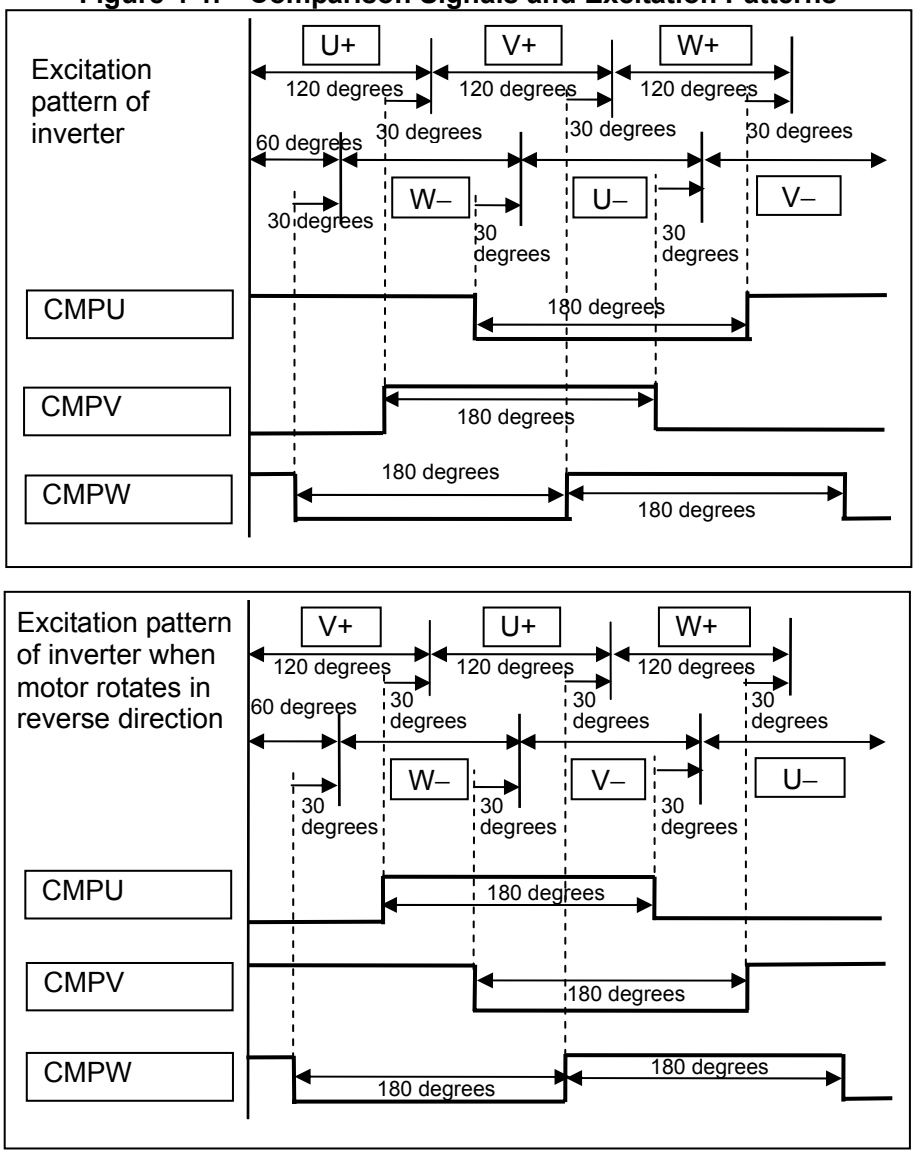
CHAPTER 4 CONTROL PROGRAM

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

4.1 Excitation Patterns

Figure 4-1 shows the timing of the comparison signals (CMPU, CMPV, and CMPW) of this system and switching the 120° excitation pattern.

Figure 4-1. Comparison Signals and Excitation Patterns



Pin information is described with reference to the BLDCM specifications.

4.1.1 Low-voltage inverter set

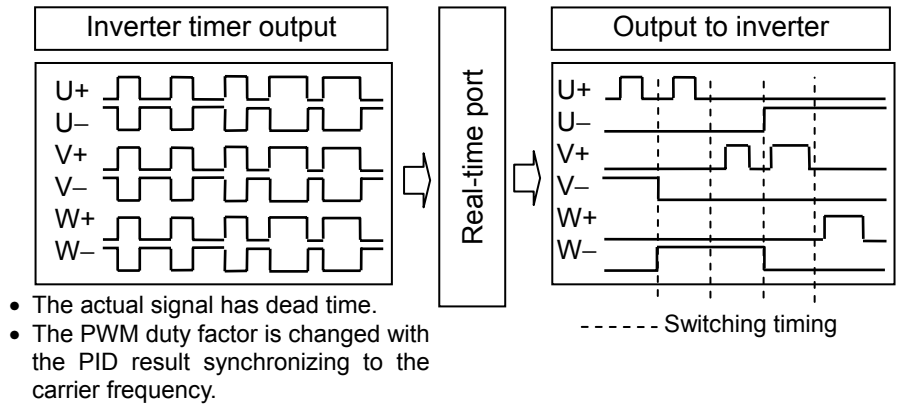
Table 4-1. BLDCM Pin Specifications

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

4.2 Excitation Pattern Switching

The excitation pattern must be switched in synchronization with inverter timer output (PWM waveform). This is therefore done by setting to the real-time port an excitation pattern that corresponds to an elapse of time equivalent to 30 degrees from the change of the comparison result signal, with the carrier synchronization interrupt (76.9 μs period) of the inverter timer.

Figure 4-2. Excitation Pattern Switching Timing

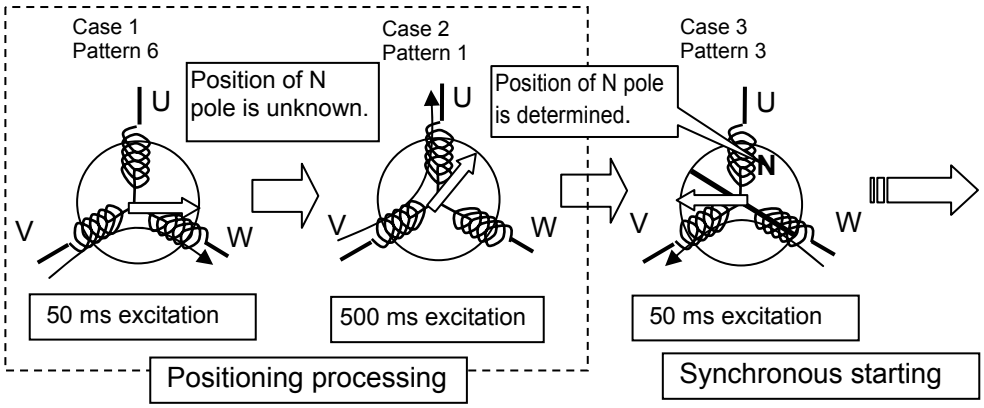


- The actual signal has dead time.
- The PWM duty factor is changed with the PID result synchronizing to the carrier frequency.

4.3 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 4-3. 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.

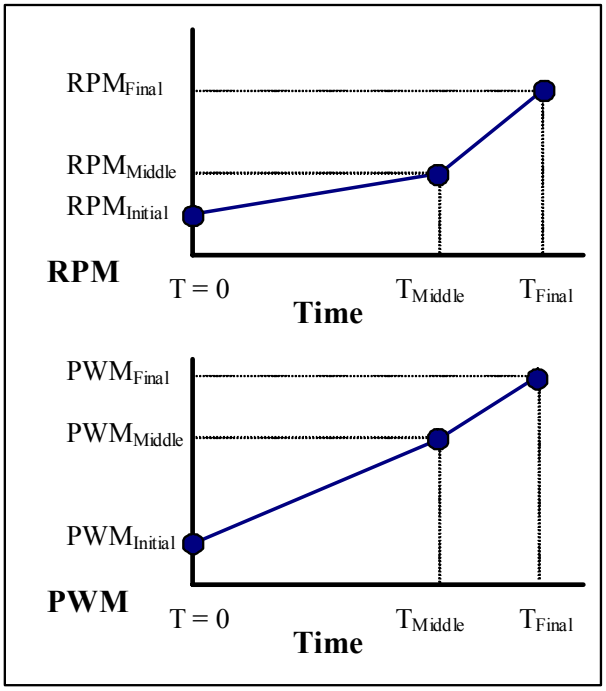
Case 3: Starts from an excitation pattern in which the direction of the current flux is at 60 to 120 degrees from the pole position of the rotor.

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 4-4. Starting Characteristics Graphs



4.4 Speed Detection

High-precision speed detection is performed with speed calculation that is not affected by delay which occurs when count values are saved in normal processing, by using the timer capture function to instantaneously save the timer count values at the point when the comparison result signal changes.

In the case of the BLDCM of this system (three phases, four poles), a comparison result signal changes four times with one rotation; however, only the change on the rising side is processed due to the specifications of the timer function to be used, so the speed is calculated from the timer value that has changed during half a rotation.

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

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

s: Resolution of timer (12.8 μs)

n: Value of timer

2: Number of poles facing each other

This system supports a revolution speed range of 200 to 3200 rpm.

4.5 Voltage Control

The voltage to be applied to the motor is controlled with the average voltage, which has been derived by performing a chopper operation (PWM) on the positive side of the inverter with a 13 kHz carrier frequency.

No complementary operation is performed on the negative side of PWM, so no braking torque occurs during deceleration (natural deceleration).

4.6 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.

4.6.1 PID control

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)

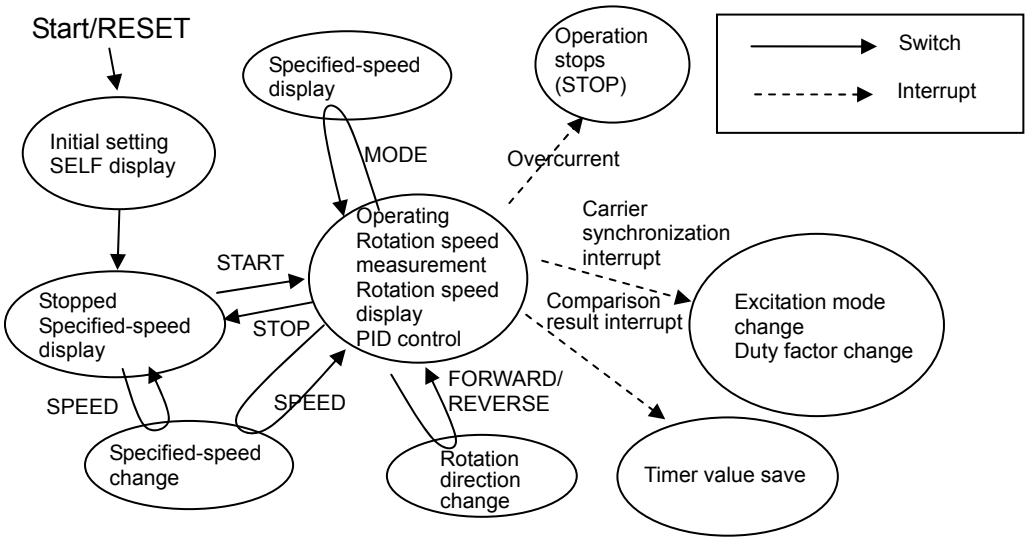
The optimum values of the feedback gains change depending on the motor characteristics and on the presence or absence of a load.

The cycle of the PID control action is set to the speed updating interval (150 ms) at the minimum revolution speed (200 rpm).

4.7 Module Configuration

Figure 4-5 shows status transition of the system.

Figure 4-5. Status Transition of System



The carrier synchronization interrupt is generated every 76.9 μs.

4.8 Control Program Functions and Flowcharts

4.8.1 Function list

The control program consists of many functions. Table 4-4 lists these functions and their features. For details of processing, see **4.8.2 Flowcharts**.

Table 4-4. Functions (1/2)

Function Name	Features
void main(void)	Main routine Starting, initialization, infinite loop waiting for switch input (start, stop, PID control)
void int_carrier(void)	Carrier synchronization interrupt Detects no rotation of motor (error processing). Acquires speed information. Changes duty factor of PWM. Changes excitation pattern.
void int_fault (void)	Overcurrent interrupt Stops control. Displays error.
void system_init(void)	Initialization Initializes internal peripherals. Displays “SELF” with LEDs.
void system_start(void)	Starts motor control. Sets value for starting as variable. Starts internal peripheral timer. Unmasks interrupt.
void system_restart(void)	Resumes motor control after stop of reverse rotation. Sets value for resumption as variable
void system_stop(void)	Stops motor control. Masks interrupt. Sets value for stopping as variable. Stops internal peripheral timer.
void init_openloop(void)	Parameter operation required for initial start control
void INTP0_on(void)	Enables overcurrent interrupt. Unmasks interrupt.
void print_error(char)	Error display Displays error on 7-segment LEDs. Stops program by STOP instruction.
void pwm_pid(void)	PID control of speed Calculates speed from number of carrier synchronization interrupts and timer value. Performs PID control for each specified time, based on difference from specified speed and calculates PWM duty factor.
void get_speed(void)	Reads specified speed. Calculates specified speed from A/D converted voltage of variable resistor. Sets specified speed of 200 to 3200 rpm to variable.
unsigned char get_sw(void)	Reads switch. Returns status of control switch.
void init_PORT(void)	Initializes I/O port.
void init_OSC(void)	Changes CPU clock.

Table 4-4. Functions (2/2)

Function Name	Features
void init_TW0(void)	Initializes 10-bit inverter timer.
void set_TW0(int, int, int, int)	Changes PWM duty factor.
void start_TW0(void)	Starts PWM output.
void stop_TW0(void)	Stops PWM output.
void init_TM00(void)	Initializes 16-bit timer (TM00).
void start_TM00(void)	Starts 16-bit timer (TM00).
void stop_TM00(void)	Stops 16-bit timer (TM00).
void init_RTPM01(void)	Initializes real-time port.
void set_RTPM01(unsigned char)	Sets real-time port. Changes excitation pattern (port that outputs PWM).
void start_RTPM01(void)	Real-time port function operation
void stop_RTPM01(void)	Stops real-time port function operation.
void init_AD(void)	Initializes A/D function. Sets A/D conversion of SPEED volume.
void start_AD(void)	Starts A/D conversion.
unsigned int get_AD(char)	Reads A/D conversion value.
void init_WDTM(void)	Initializes watchdog timer.
void clear_WDTM(void)	Clears watchdog timer counter.
void reset_WDTM(void)	Generates internal reset signal from watchdog timer.
void init_TM50(void)	Initializes 8-bit timer (TM50). Used as 1 ms timer.
void start_TM50(void)	Starts 8-bit timer (TM50).
void wait_TM50(void)	Waits for 8-bit timer (TM50).
void speed_print(int)	Displays speed with LEDs. Specified speed is displayed when motor control is stopped. Rotation speed of motor is displayed when motor is controlled and while MODE switch is depressed.
void INTTW0UD_on(void)	Enables carrier synchronization interrupt.
void INTTW0UD_off(void)	Disables carrier synchronization interrupt.
void INTP5_on(void)	Enables INTP5 interrupt.
void INTP5_off(void)	Disables INTP5 interrupt.
void led_print(int, char)	Displays value of 1 to 4 digits with LEDs.
void led_set(unsigned char, unsigned char)	Outputs value to specified LED.
void wait(int)	Waits for lapse of specified time (ms).
char read_BEMF(void)	Reads value of BEMF signal.
void int_speed(void)	INTP5 interrupt

4.8.2 Flowcharts

Figures 4-6 through 4-50 show a flowchart of each function.

<Main routine: main()>

Figure 4-6. Main Processing (1/2)

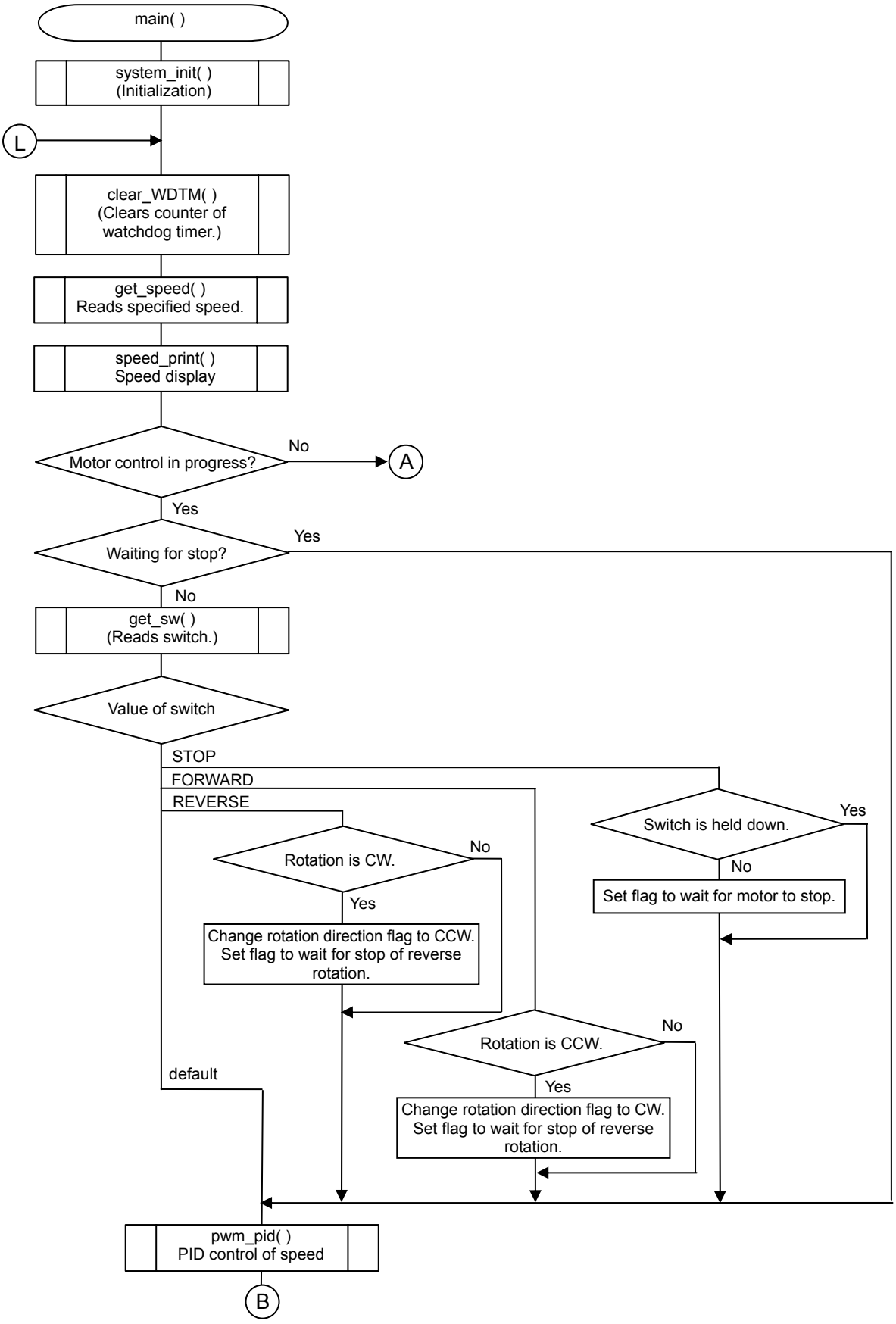
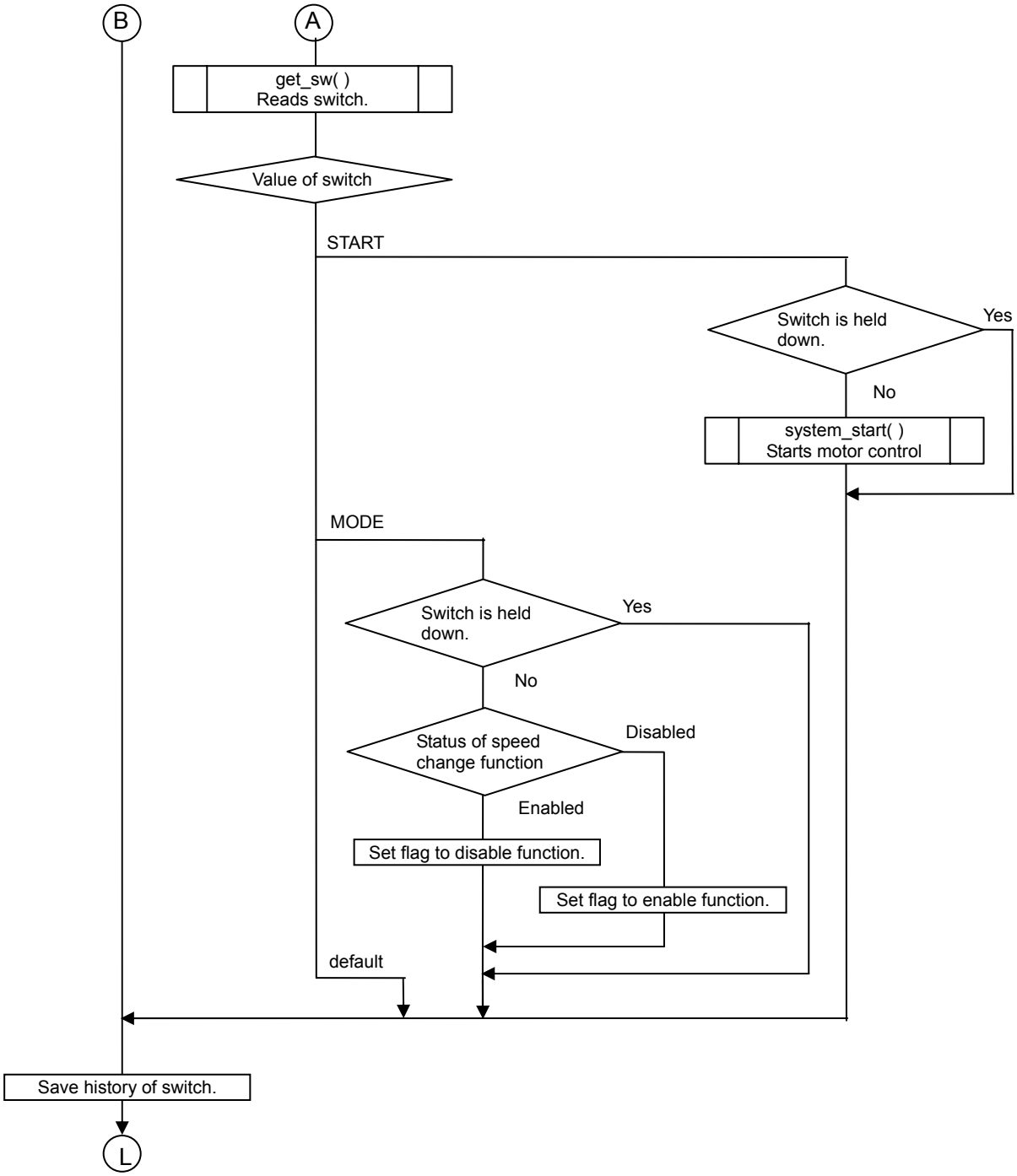


Figure 4-6. Main Processing (2/2)



<Carrier synchronization interrupt>

Figure 4-7. Carrier Synchronization Interrupt Processing (1/3)

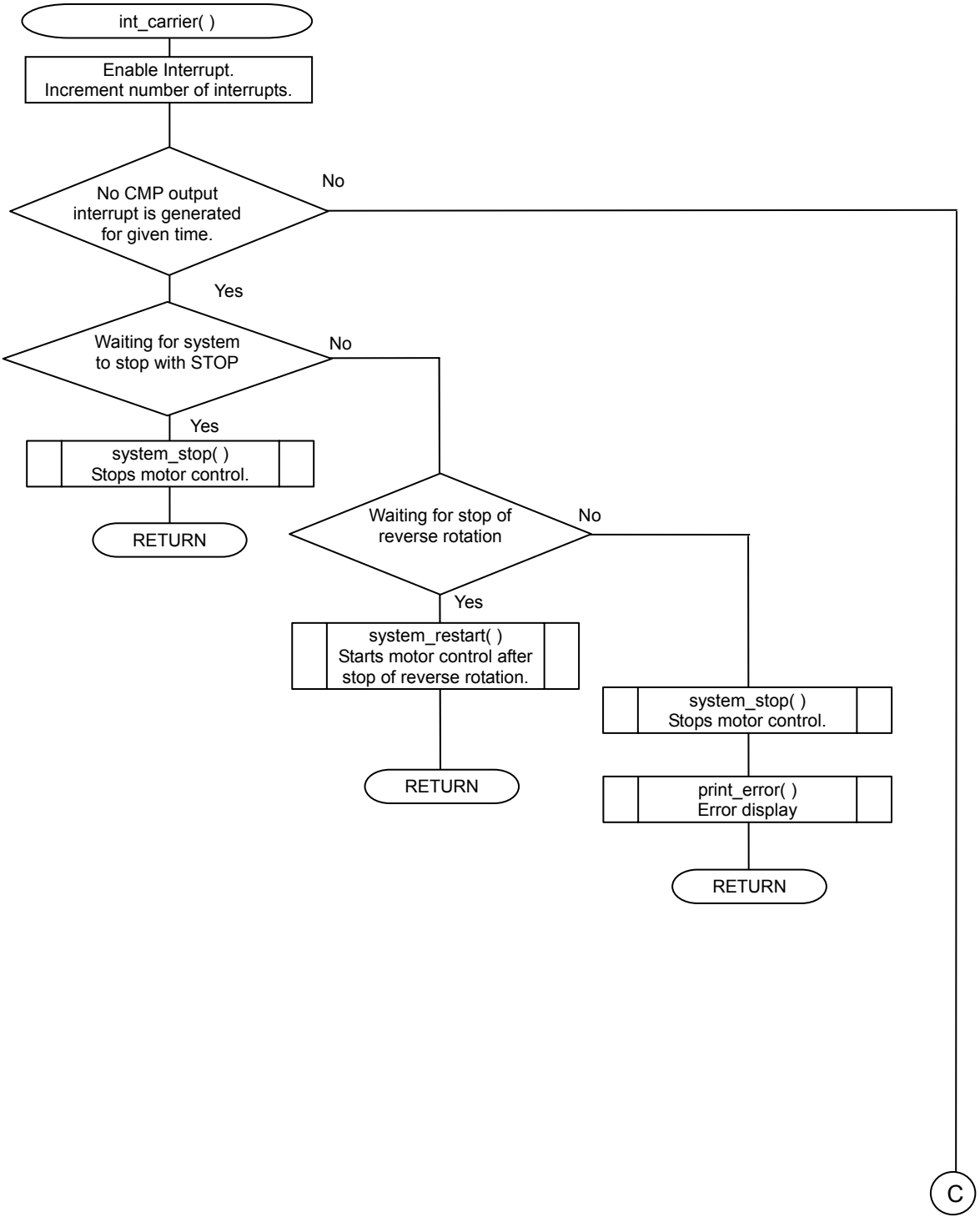


Figure 4-7. Carrier Synchronization Interrupt Processing (2/3)

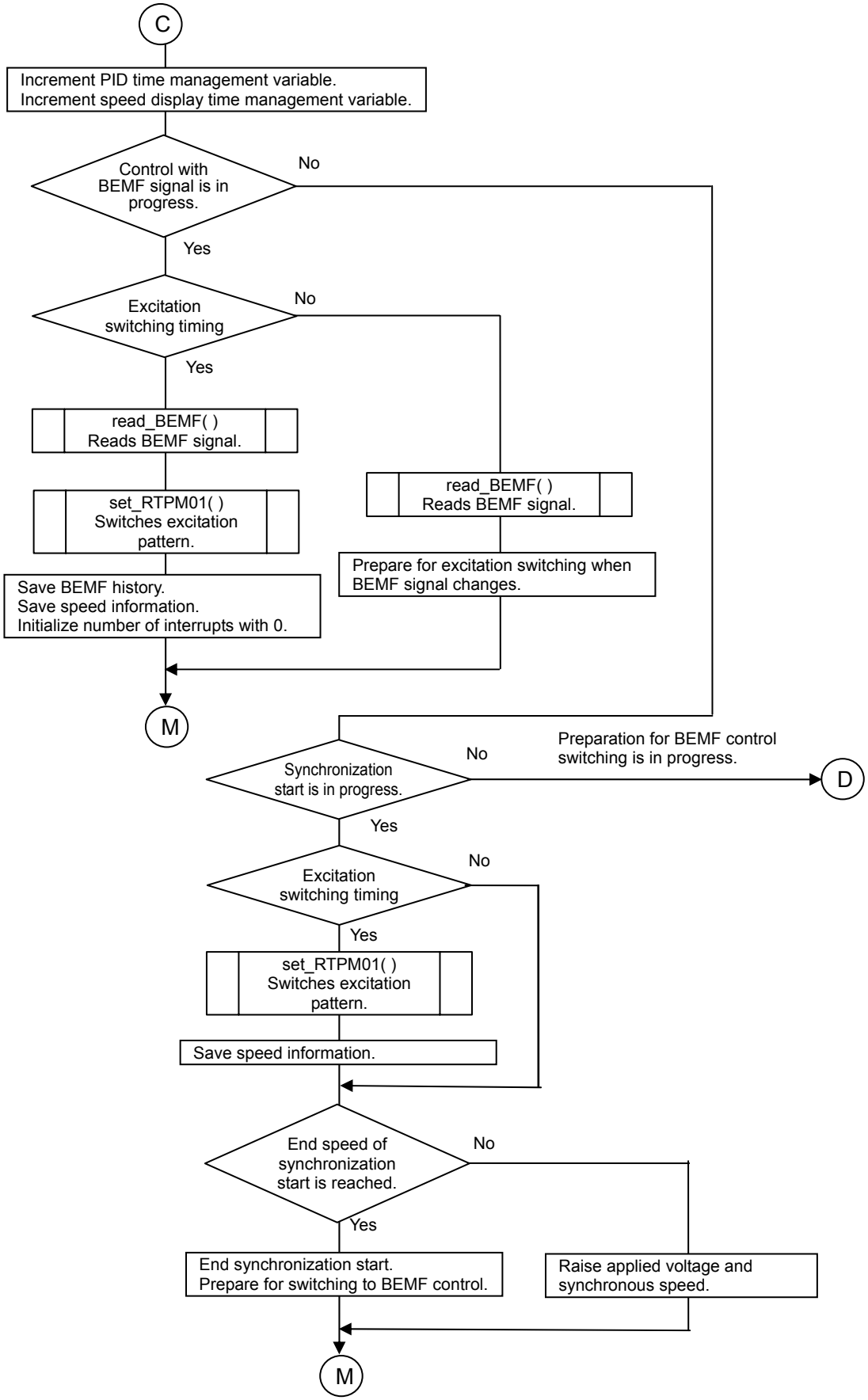
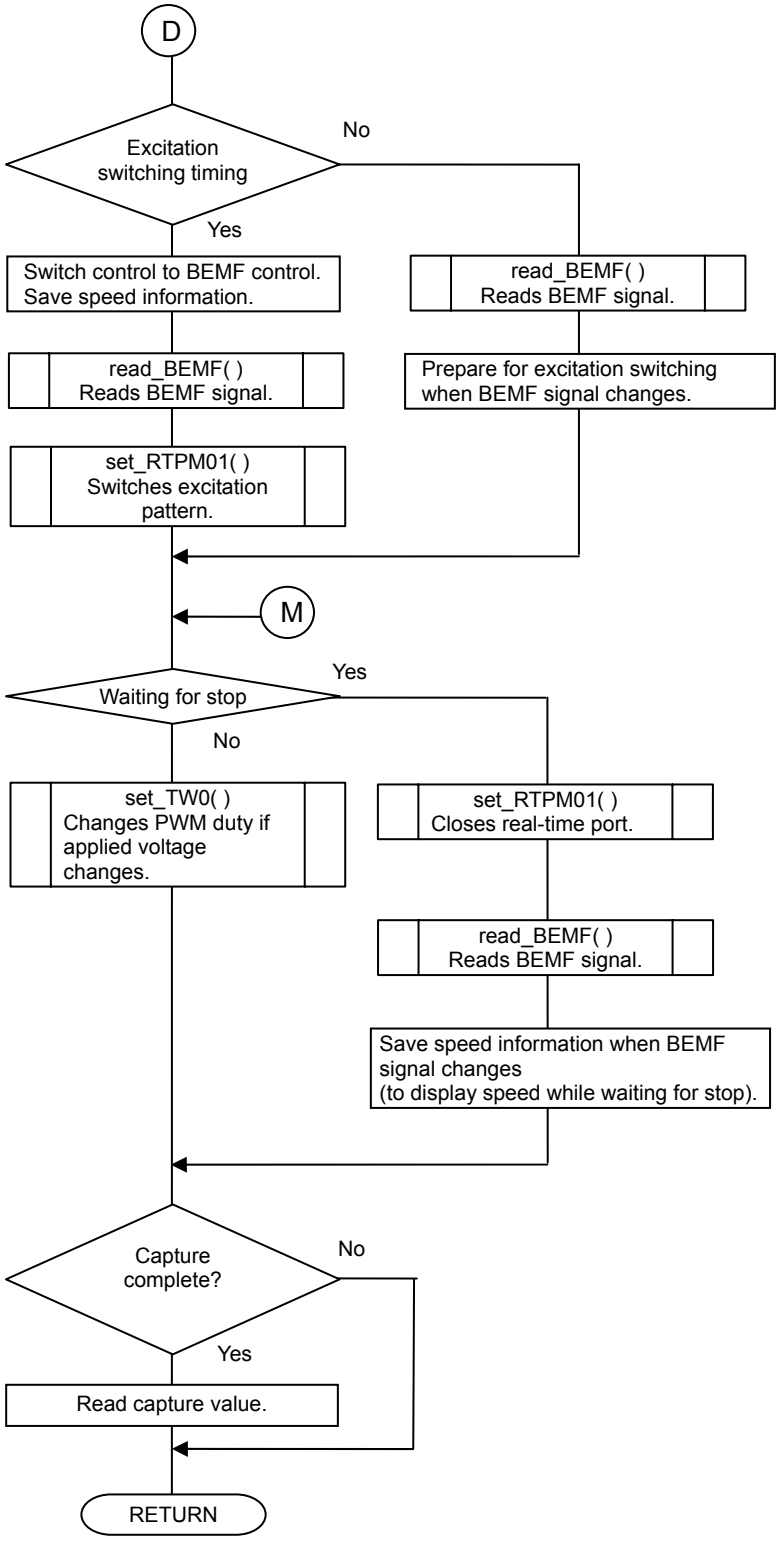
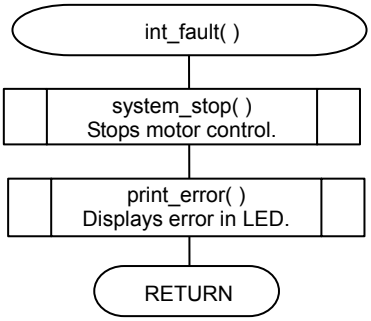


Figure 4-7. Carrier Synchronization Interrupt Processing (3/3)



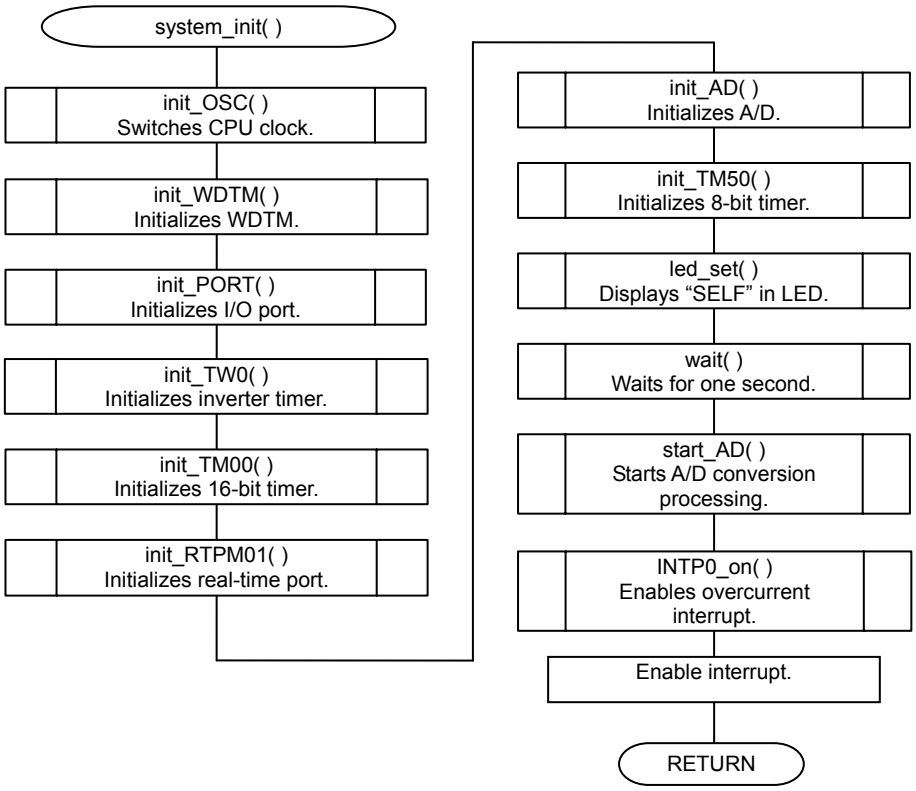
<Overcurrent interrupt>

Figure 4-8. Overcurrent Interrupt Processing



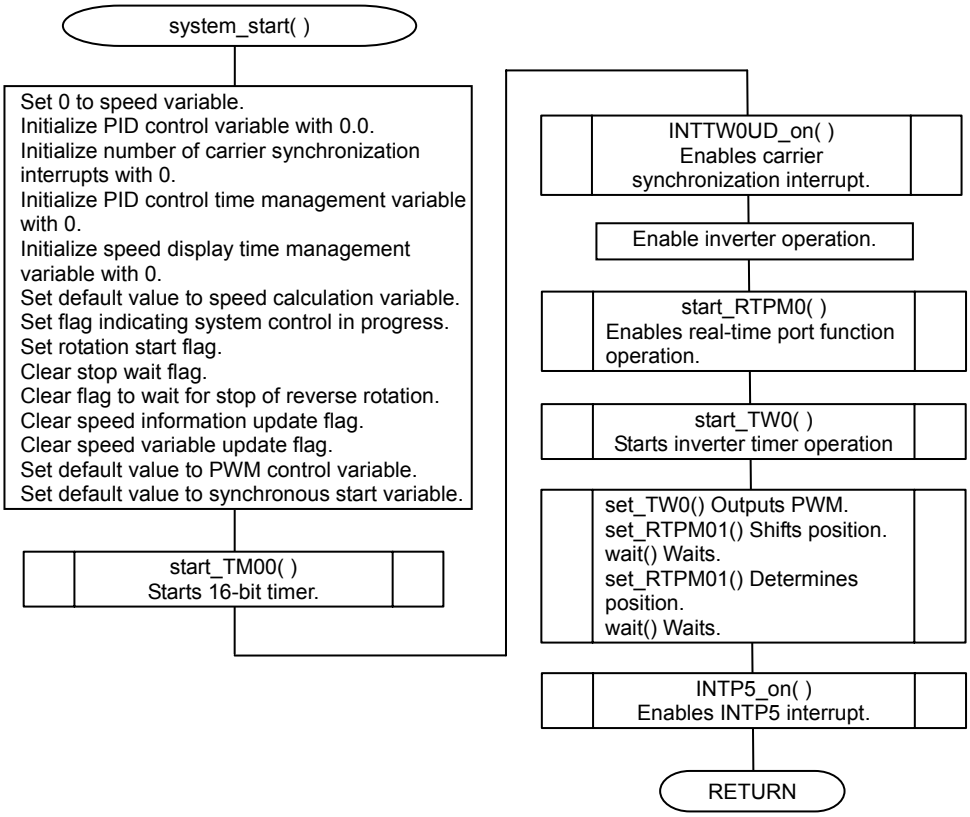
<Initialization>

Figure 4-9. Initialization Processing



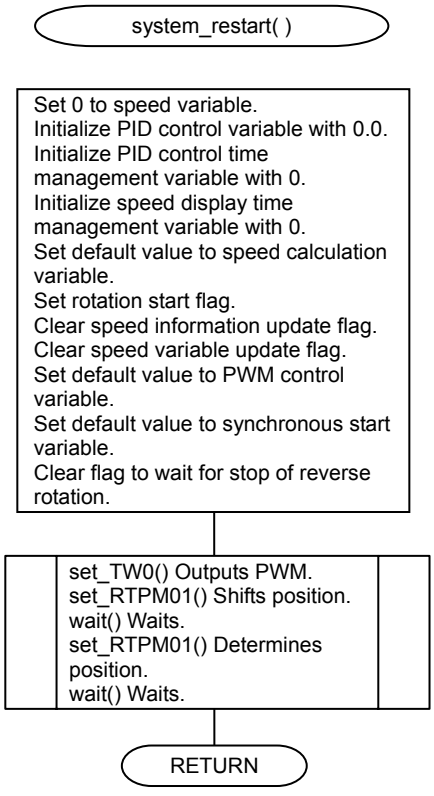
<Motor control start>

Figure 4-10. Motor Control Start Processing



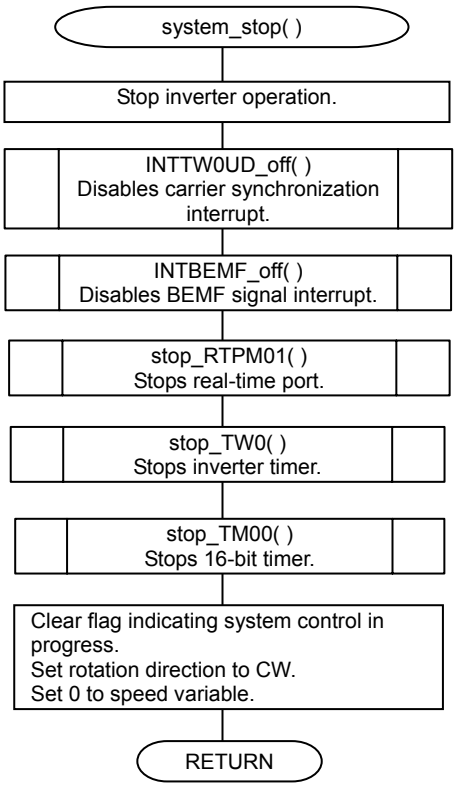
<Motor control resumption after stop of reverse rotation>

Figure 4-11. Motor Control Resumption Processing After Stop of Reverse Rotation



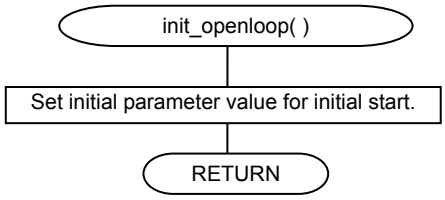
<Motor control stop>

Figure 4-12. Motor Control Stop Processing



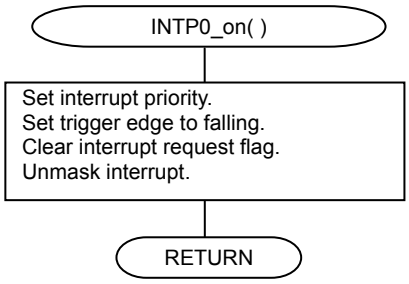
<Setting default value for initial start>

Figure 4-13. Processing of Setting Default Value for Initial Start



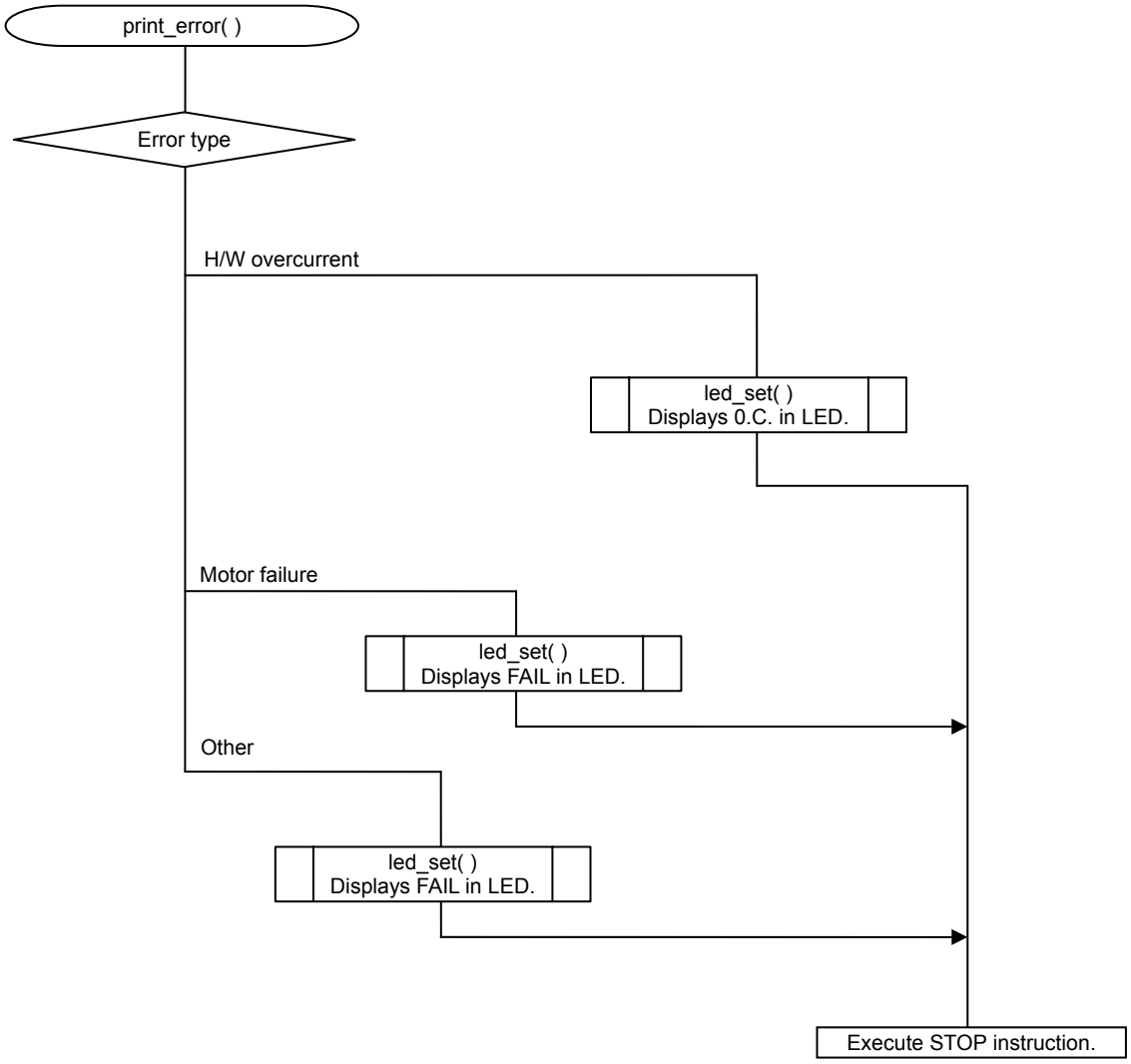
<Overcurrent interrupt enable>

Figure 4-14. Overcurrent Interrupt Enable Processing



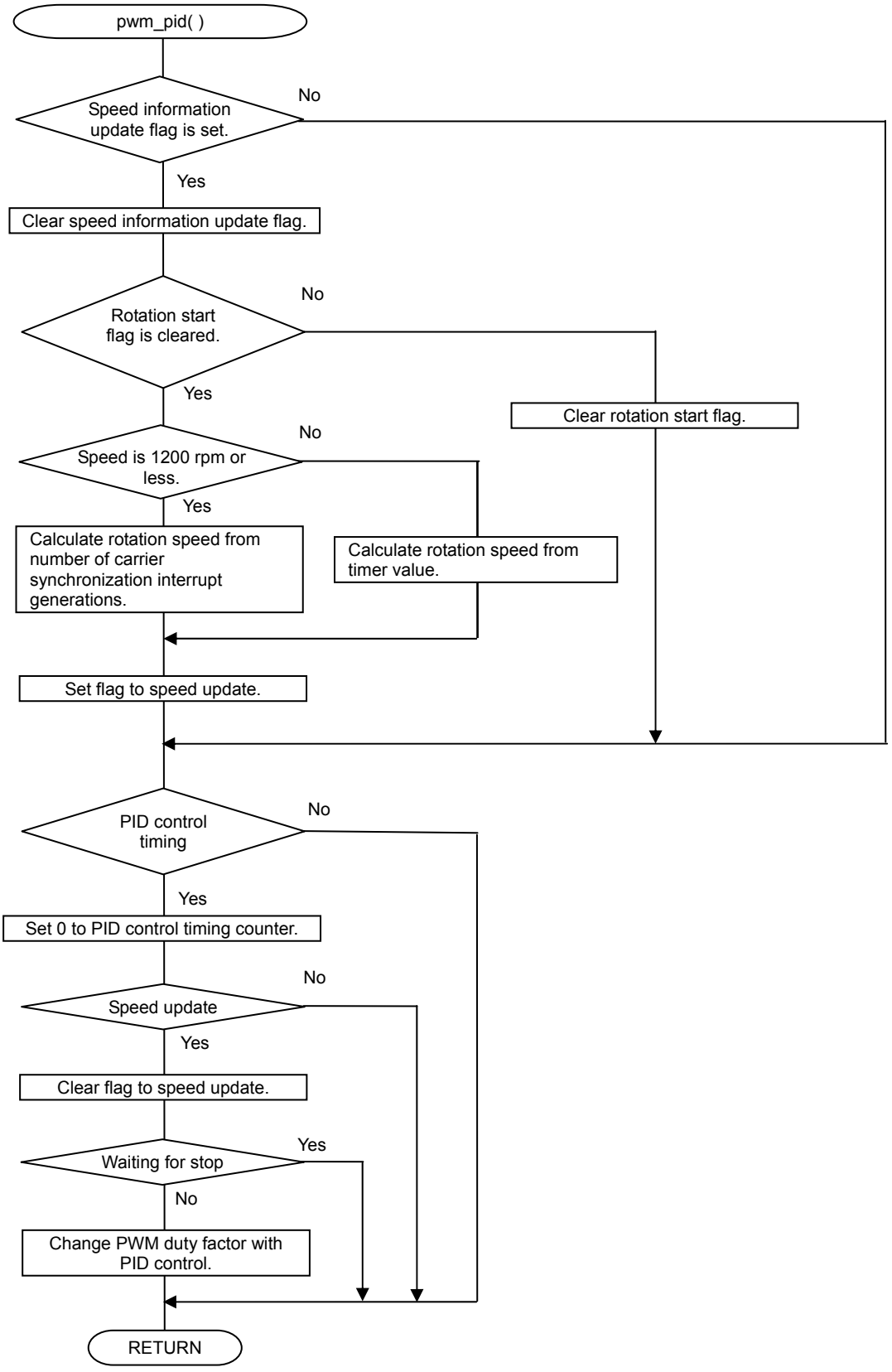
<Error display>

Figure 4-15. Error Display Processing



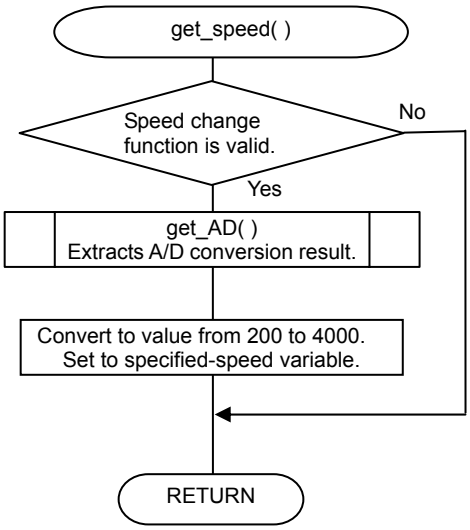
<PID control of speed>

Figure 4-16. Speed PID Control Processing



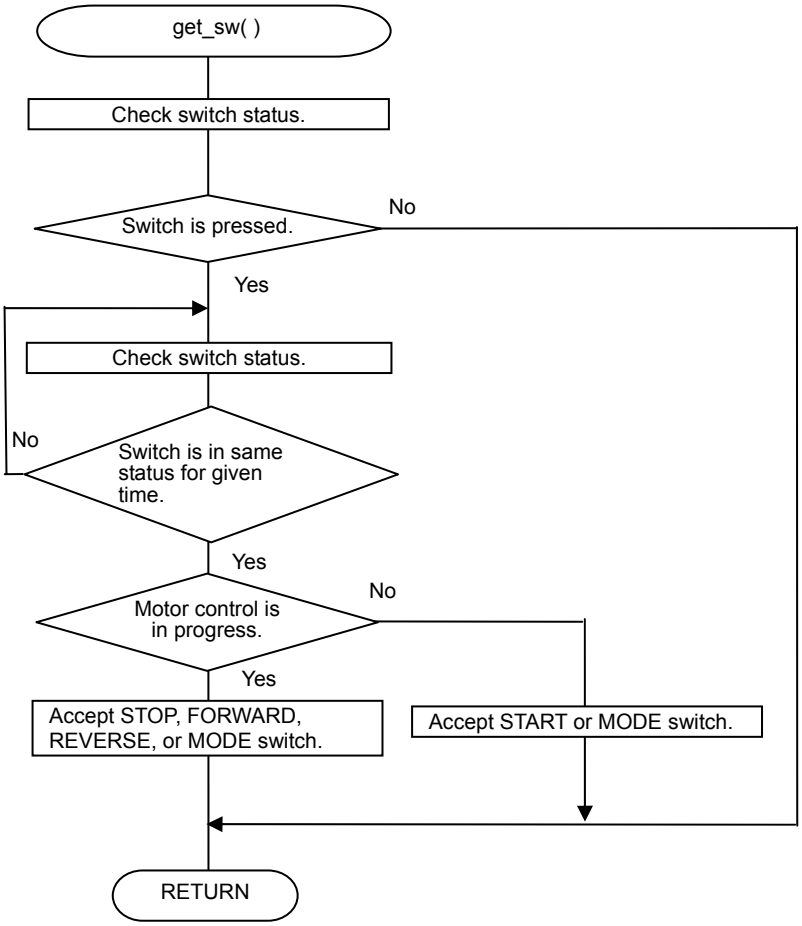
<Specified-speed read>

Figure 4-17. Specified-Speed Read Processing



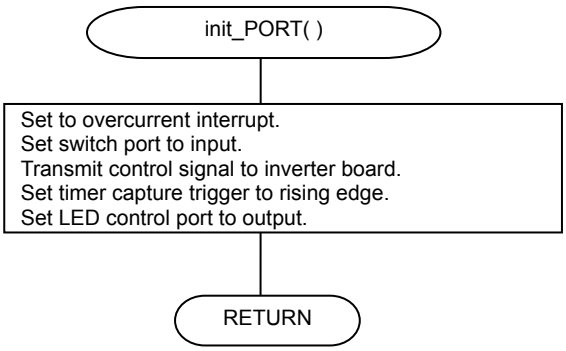
<Switch read>

Figure 4-18. Switch Read Processing



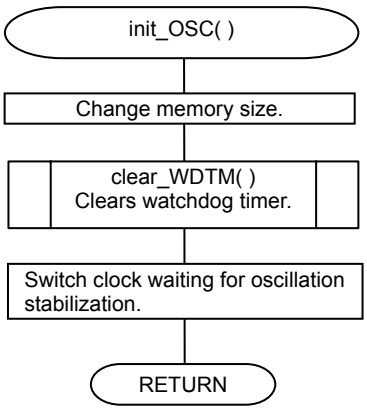
<I/O port initialization>

Figure 4-19. I/O Port Initialization Processing



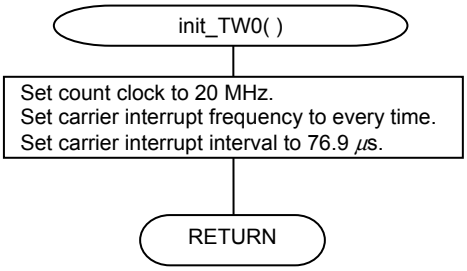
<CPU clock switching>

Figure 4-20. CPU Clock Switch Processing



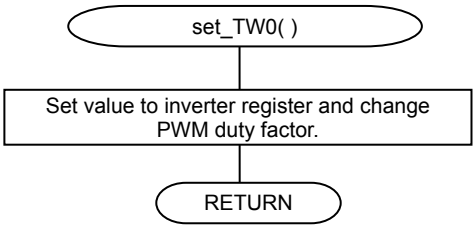
<10-bit inverter timer initialization>

Figure 4-21. 10-bit Inverter Timer Initialization Processing



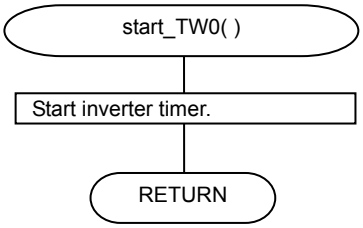
<PWM duty factor change>

Figure 4-22. PWM Duty Factor Change Processing



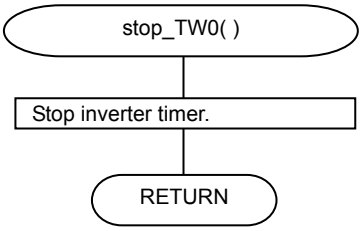
<PWM output start>

Figure 4-23. PWM Output Start Processing



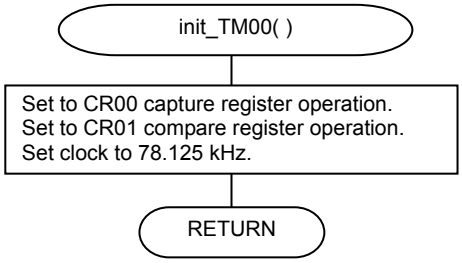
<PWM output stop>

Figure 4-24. PWM Output Stop Processing



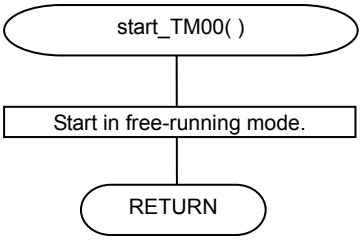
<16-bit timer (TM00) initialization>

Figure 4-25. 16-bit Timer (TM00) Initialization Processing



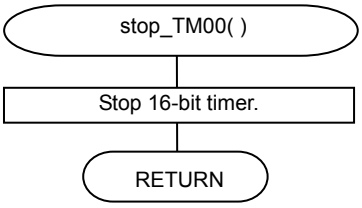
<16-bit timer (TM00) start>

Figure 4-26. 16-bit Timer (TM00) Start Processing



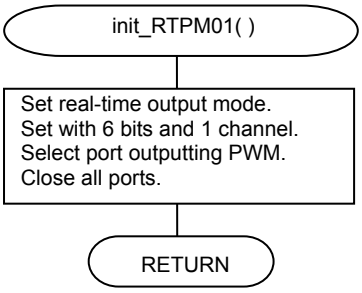
<16-bit timer (TM00) stop>

Figure 4-27. 16-bit Timer (TM00) Stop Processing



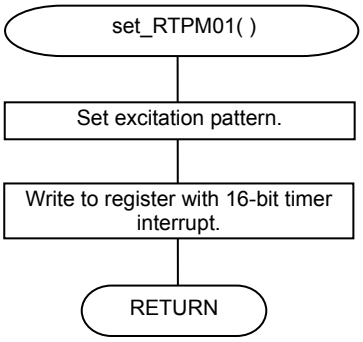
<Real-time port initialization>

Figure 4-28. Real-Time Port Initialization Processing



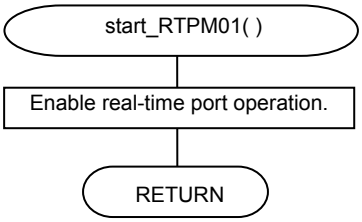
<Real-time port setting>

Figure 4-29. Real-Time Port Setting Processing



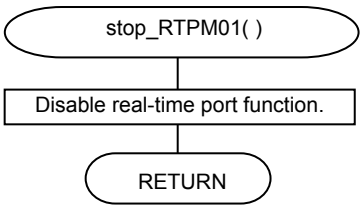
<Real-time port start>

Figure 4-30. Real-Time Port Function Operation Processing



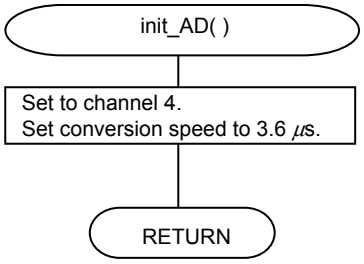
<Real-time port stop>

Figure 4-31. Real-Time Port Function Operation Stop Processing



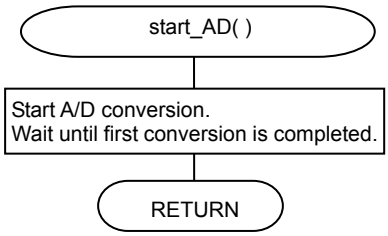
<A/D function initialization>

Figure 4-32. A/D Function Initialization Processing



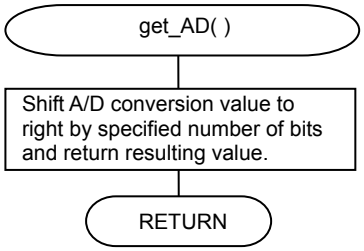
<A/D conversion start>

Figure 4-33. A/D Conversion Start Processing



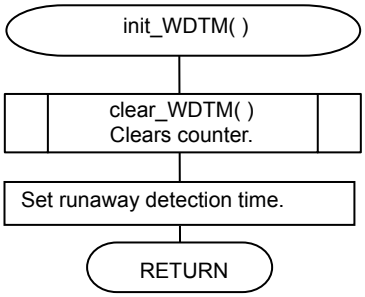
<A/D conversion value read>

Figure 4-34. A/D Conversion Value Read Processing



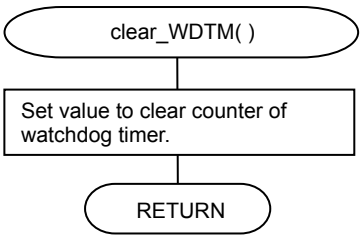
<Watchdog timer initialization>

Figure 4-35. Watchdog Timer Initialization Processing



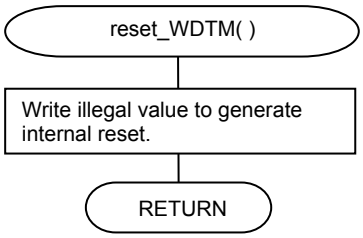
<Watchdog timer counter clear>

Figure 4-36. Watchdog Timer Counter Clear Processing



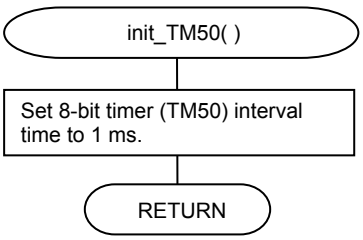
<Internal reset signal generation by watchdog timer>

Figure 4-37. Internal Reset Signal Generation Processing by Watchdog Timer



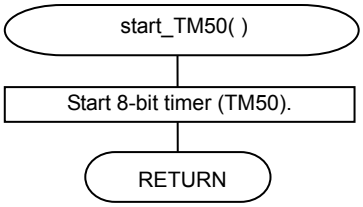
<8-bit timer (TM50) initialization>

Figure 4-38. 8-bit Timer (TM50) Initialization Processing



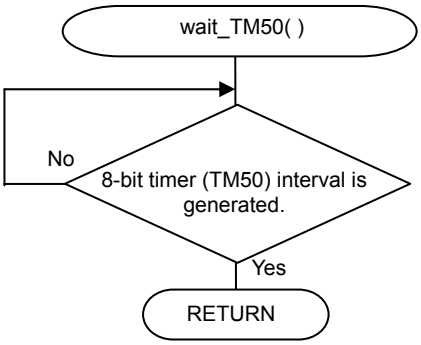
<8-bit timer (TM50) start>

Figure 4-39. 8-bit Timer (TM50) Start Processing



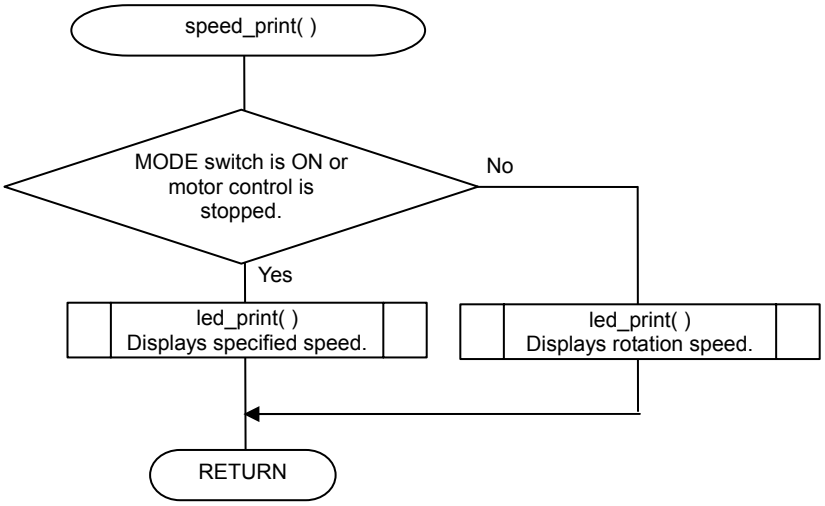
<8-bit timer (TM50) wait>

Figure 4-40. 8-bit Timer (TM50) Wait Processing



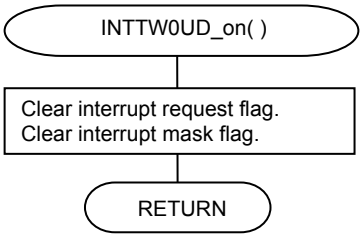
<Displaying speed on LEDs>

Figure 4-41. Processing of Displaying Speed on LEDs



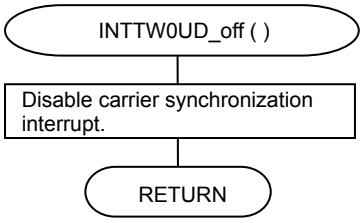
<Carrier synchronization interrupt enable>

Figure 4-42. Carrier Synchronization Interrupt Enable Processing



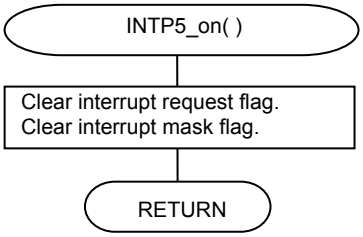
<Carrier synchronization interrupt disable>

Figure 4-43. Carrier Synchronization Interrupt Disable Processing



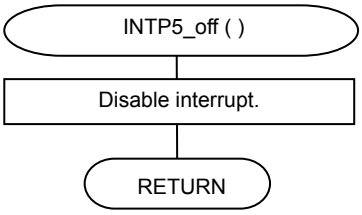
<INTP5 interrupt enable>

Figure 4-44. INTP5 Interrupt Enable Processing



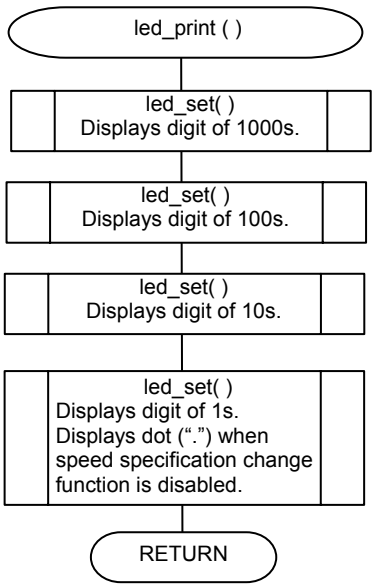
<INTP5 interrupt disable>

Figure 4-45. INTP5 Interrupt Disable Processing



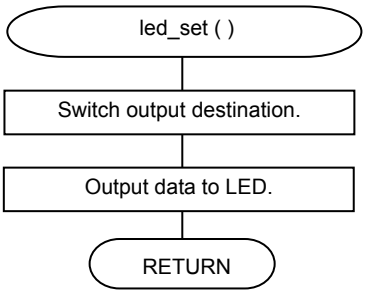
<Displaying 1- to 4-digit value on LEDs>

Figure 4-46. Processing of Displaying 1- to 4-Digit Value on LEDs



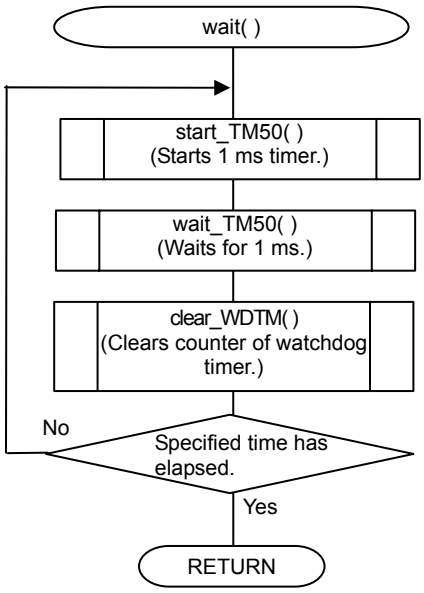
<Outputting value to specified LED>

Figure 4-47. Processing of Outputting Value to Specified LED



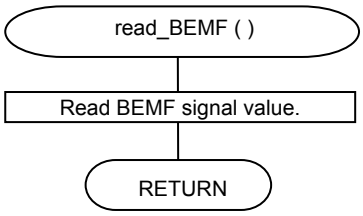
<Waiting for lapse of specified time (ms)>

Figure 4-48. Processing of Waiting for Lapse of Specified Time (ms)



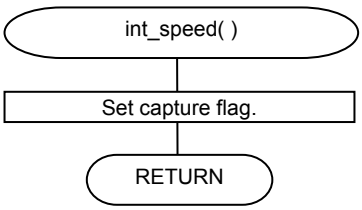
<Reading BEMF signal value>

Figure 4-49. Processing of Reading BEMF Signal



<INTP5 interrupt>

Figure 4-50. INTP5 Interrupt Processing



4.9 Variables and Constants of Control Program

4.9.1 Values defined by #define of main.h

Name	Meaning	Set Value	Remark
FLG_ON	Flag value	1	Indicates whether flag is valid.
FLG_OFF	Flag value	0	
FLG_START	Flag value	2	
FLG_WAIT	Flag value	4	
CW	Rotation direction	0	Indicates rotation direction of motor.
CCW	Rotation direction	1	
START_TR	For status setting	0x01	Starts control.
STOP_TR	For status setting	0x02	Stops control.
FORWARD_TR	For status setting	0x04	Changes rotation direction to CW.
REVERSE_TR	For status setting	0x08	Changes rotation direction to CCW.
MODE_TR	For status setting	0x10	Status in which MODE switch is pressed
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
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	

4.9.2 Values defined by #define of lib_eu.h

(1/2)

Name	Meaning	Set Value	Remark
KP_DEF	Feedback gain	0.075	Used for PID control
KI_DEF	Feedback gain	0.001	
KD_DEF	Feedback gain	0.02	
PWM_LIMIT_H	Upper limit of manipulated variable of PWM	50.0	
PWM_LIMIT_L	Lower limit of manipulated variable of PWM	−50.0	
PWM_BASE_REF	PWM base	769	Used for PWM output
PWM_F_REF	PWM default value	0	
PWM_F_START	PWM start value	44	
PWM_F_MIN	Minimum PWM value	10	
PWM_F_MAX	Maximum PWM value	769	
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 port control register	PM54	Inverter control port
INTP0	Port control register	PM00	Overcurrent detection 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

(2/2)

Name	Meaning	Set Value	Remark
LED_0	LED display data	0xc0	Displays “0”.
LED_1		0xf9	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 “ . ”.
IMS_DATA	Memory size	0xc8	Used to change memory size.
WDTM_OFF	Stop	0x77	For controlling watchdog timer
WDTM_SET	Set	0x67	
WDTE_CLR	Clear	0xac	
WDTE_RESET	Reset	0x00	
KEY_WAIT	Switch observation time	10	Used to eliminate chattering
SW	Switch	(P7&0xf)	Switch connection port
START_SW	Start	0x7	Switch status
STOP_SW	Stop	0x7	
FORWARD_SW	CW	0xb	
REVERSE_SW	CCW	0xd	
MODE_SW	Mode	0xe	
OFF_SW	Not pressed	0xf	

4.9.3 Variables

Variable Name	Type	Meaning	Remark
sys_flag	char	Control flag	Control 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.
speed_flag	char	Speed information flag	Indicates whether data required for calculating speed is present.
ad_flag	char	Speed change flag	Limits specified-speed change function.
int_cnt	unsigned int	Interrupt counter	Number of times of carrier synchronization interrupt
pid_cnt	unsigned int	PID control timing	Number of times of carrier synchronization interrupt
print_cnt	unsigned int	Speed display timing	Number of times of carrier synchronization interrupt
cnt_data cnt_data1 cnt_data2 cnt_data3	unsigned int	Data for speed calculation	Number of times of carrier synchronization interrupt
sync_flag	char	Synchronous start flag	Differentiates synchronization start and BEMF control.
cnt_flag	char	Time management	For 30-degree delay
sync_sw	char	Excitation pattern	Pointer
sync_tbl[][]	unsigned char	Excitation pattern	Table
sync_cnt	unsigned int	For 30-degree delay	Number of times of carrier synchronization interrupt
sync_def	unsigned int	Reference value for 30-degree delay	Number of times of carrier synchronization interrupt
int_cnt	int	Interrupt counter	Number of times of carrier synchronization interrupt
pwm_base	int	PWM base clock value	Carrier width of PWM
pwm_ff	int	PWM command value	Distribution rate of PWM
old_ff	int	History of PWM command value	Value of previous pwm_ff
m_speed	int	Actual speed	Revolution speed of motor (rpm)
led_data[ ]	unsigned char	LED output data	Value displayed with LEDs
up_flag	char	Actual speed calculation status flag	For rotation speed calculation
old_clk	unsigned int	Previous clock value	
new_clk	unsigned int	Newest clock value	
kp_ref	float	P gain	Speed control
ki_ref	float	I gain	
kd_ref	float	D gain	
mvn	float	Previous manipulated variable	
en	float	Difference from current value	
en_1	float	Difference from previous value	
en_2	float	Difference from value before previous	

4.10 Control Program Source File

Program size ROM– 1EDDh
RAM– 3C0h

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

    target    : uPD78F0714 motor starter kit

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

    NEC Micro Systems,Ltd
*/

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

#define CW              0       /* Clockwise                    */
#define CCW             1       /* Counterclockwise             */
#define START_TR        0x01    /* Identifies switch pressed    */
#define STOP_TR         0x02
#define FORWARD_TR      0x04
#define REVERSE_TR      0x08
#define MODE_TR         0x10

#define ERROR_HALL      0x01    /* Hall IC failure              */
#define ERROR_OC        0x02    /* Overcurrent                  */
#define ERROR_MOTOR     0x04    /* Motor failure                */
#define ERROR_S_OC      0x08    /* Overcurrent                  */

#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   */

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

extern void             system_init(void);
extern void             speed_print(int);
extern unsigned char    get_sw(void);
extern void             pwm_pid(int);
extern void             system_start(void);
extern void             system_stop(void);
extern void             system_restart(void);
extern char             read_BEMF(void);
extern void             set_RTPM01(unsigned char);
extern void             set_TW0(int, int, int, int);
extern void             print_error(char);
extern void             clear_WDTM(void);
extern void             clear_WDTM(void);
```

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

    target   : uPD78F0714 motor starter kit

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

    NEC Micro Systems,Ltd
*/

#include    "main.h"

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

    system_init();                              /* Initialization                                   */
    while(1){
        clear_WDTM();                           /* Clears watchdog timer                            */
        get_speed();                            /* Reads specified speed                            */
        speed_print(300);                       /* Displays speed (300 ms interval)                 */
        if(sys_flag != FLG_OFF){                /* Being started                                    */
            if((cw_ccw_wait == FLG_OFF) &&      /* Not waiting for motor to stop reverse rotation   */
               (stop_wait   == FLG_OFF)){       /* Not waiting for stop                             */
                sw = get_sw();                  /* Checks switch                                    */
                switch(sw){
                case STOP_TR:
                    if(sw != tmp_sw){           /* Handles toggle                                   */
                        stop_wait = FLG_ON;     /* Sets wait for stop                               */
                    }
                    break;

                case FORWARD_TR:
                    if(cw_ccw_flag == CCW){     /* Rotating CCW                                     */
                        cw_ccw_wait = FLG_ON;   /* Sets wait for motor to stop reverse rotation     */
                        cw_ccw_flag = CW;       /* Sets rotation direction                          */
                    }
                    break;

                case REVERSE_TR:
                    if(cw_ccw_flag == CW){      /* Rotating CW                                      */
                        cw_ccw_wait = FLG_ON;   /* Sets wait for motor to stop reverse rotation     */
                        cw_ccw_flag = CCW;      /* Sets rotation direction                          */
                    }
                    break;

                default:
                    ;
                }
            }
            pwm_pid(150);                       /* PID control: 150 ms period                       */
        }else{                                  /* Being stopped                                    */
            sw = get_sw();                      /* Checks switch                                    */
            switch(sw){
            case START_TR:                      /* Starts                                           */
                if(sw != tmp_sw){               /* Handles toggle                                   */
                    system_start();             /* Starts                                           */
                }
                break;

            case MODE_TR:                       /* Speed change function                            */
                if(sw != tmp_sw){               /* Handles toggle                                   */
                    if(ad_flag == FLG_ON){      /* Function being stopped                           */
                        ad_flag = FLG_OFF;      /* Resumes function                                 */
                    }else{
                        ad_flag = FLG_ON;       /* Stops function                                   */
                    }
                }
                break;

            default:
                ;
            }
        }
        tmp_sw = sw;
    }
}
```

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

    target    : uPD78F0714 motor starter kit

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

    NEC Micro Systems,Ltd
*/

#pragma sfr
#pragma ei

#pragma INTERRUPT INTP0         int_fault    rb1   /* Overcurrent occurrence    */
#pragma INTERRUPT INTTW0UD  int_carrier rb2   /* Carrier synchronization interrupt   */


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

/* ---------------------------------------------- */
#define INT_CNT_MAX      650                              /* Identification of non-rotation for 50 ms × 100 us      */
#define SYNC_END_CNT     325                              /* 200 rpm */
#define SYNC_UP_CNT      30
#define V_ADD_CNT        50

#define ISHUNT_AD        5          /* Number of AD channel to which ISHUNT pin is connected */

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

#define CLEAR            0
#define SET              1

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

const unsigned char tr_sw[2][8] = {                          /* Excitation pattern  */
    {P_OFF, P_T2, P_T4, P_T3, P_T6, P_T1, P_T5, P_OFF},  /* CW                  */
    {P_OFF, P_T1, P_T3, P_T2, P_T5, P_T6, P_T4, P_OFF}   /* CCW                 */
};

/* ======================================================== */
__interrupt void
int_fault(void)
{
    system_stop();                          /* Stops system */
    print_error(ERROR_OC);                  /* Processes error    */
}


/*
    Carrier synchronization interrupt
*/
__interrupt void
int_carrier(void)
{
    static char             r_data, o_data;
    static int              cnt;
    int                     tmp_pwm;

    int_cnt++;                                                  /* Number of interrupts                          */
    if(int_cnt > INT_CNT_MAX){                                  /* No excitation switching for 50 ms or more     */
        if(stop_wait != FLG_OFF){                               /* Waits for stop with STOP                      */
            system_stop();
        }else if(cw_ccw_wait != FLG_OFF){                       /* Waits for stop of reverse rotation            */
            system_restart();
        }else{
            system_stop();
            print_error(ERROR_MOTOR);
        };
        return;
    };
    pid_cnt++;                                                  /* PID control timing                  */
    print_cnt++;                                                /* Speed display timing                */
    if(sync_flag == FLG_OFF){                                   /* Being controlled by BEMF signal     */
        if(cnt_flag != FLG_OFF){                                /* Waits for excitation switching      */
            if(--cnt <= 0){                                     /* 30-degree delay                     */
                cnt_flag = FLG_OFF;
                o_data    = read_BEMF();
                set_RTPM01(tr_sw[cw_ccw_flag][o_data]);
                cnt_data3   = cnt_data2;
                cnt_data2   = cnt_data1;
                cnt_data1   = int_cnt;
                cnt_data    = cnt_data1 + cnt_data2 + cnt_data3;
                int_cnt    = 0;
                speed_flag = FLG_ON;
            }else{
                if(r_data != read_BEMF()){
                    cnt_flag = FLG_OFF;
                };
```

```
            };
        }else{
            r_data = read_BEMF();
            if(o_data != r_data){                          /* BEMF signal changes          */
                cnt         = (int)(cnt_data1>>1);
                cnt_flag = FLG_ON;
            };
        };
    }else if(sync_flag == FLG_START){                      /* Synchronization being started */
        int_cnt = 0;
        pid_cnt = 0;
        cnt_limit_tim++;
        cnt_limit_pwm++;
        cnt_limit_rpm++;

        if(--sync_cnt == 0){
            /* Excitation pattern switching */
            if(++sync_sw > 5){ sync_sw = 0;};
            set_RTPM01(sync_tbl[cw_ccw_flag][sync_sw]);
            speed_flag = FLG_ON;
            cnt_data3 = cnt_data2;
            cnt_data2 = cnt_data1;
            cnt_data1 = sync_def;
            cnt_data   = cnt_data1 + cnt_data2 + cnt_data3;

            if(cnt_limit_tim >= limit_tim[openloop_dim]){
                cnt_limit_tim = 0;
                openloop_dim++;
                if(openloop_dim >= 2){
                    /* Goes through open loop to PID control preprocessing */
                    sync_flag = FLG_WAIT;
                    o_data      = read_BEMF();
                    cnt_flag   = FLG_OFF;
                };
            }else{
                if(cnt_limit_pwm >= limit_pwm[openloop_dim]){
                    pwm_ff++;
                    cnt_limit_pwm = 0;
                };
                if(cnt_limit_rpm >= limit_rpm[openloop_dim]){
                    sync_def -= const_rpm[cnt_const_rpm];
                    cnt_const_rpm++;
                    cnt_limit_rpm = 0;
                };
            };
            sync_cnt = sync_def;
        };
    }else{                                                 /* Switching */
        pid_cnt = 0;
        if(cnt_flag != FLG_OFF){
            if(--cnt <= 0){
                sync_flag  = FLG_OFF;
                cnt_flag    = FLG_OFF;
                o_data      = read_BEMF();
                set_RTPM01(tr_sw[cw_ccw_flag][o_data]);
                int_cnt      = 0;
                speed_flag= FLG_ON;
                cnt_data3  = cnt_data2;
                cnt_data2  = cnt_data1;
                cnt_data1  = int_cnt;
                cnt_data   = cnt_data1 + cnt_data2 + cnt_data3;
            }else{
                if(r_data != read_BEMF()){
                    cnt_flag = FLG_OFF;
                };
            };
        }else{
            r_data = read_BEMF();
            if(o_data != r_data){ /* BEMF changes */
                cnt         = (int)(sync_def>>1);
                cnt_flag = FLG_ON;
            };
        };
    };
    if((stop_wait == FLG_OFF) && (cw_ccw_wait == FLG_OFF)){    /* Not waiting for stop          */
        if(pwm_ff != old_ff){                                  /* Duty changes */
            tmp_pwm = pwm_base - pwm_ff;
            set_TW0(tmp_pwm, tmp_pwm, tmp_pwm, pwm_base); /* Changes duty                    */
            old_ff = pwm_ff;
        };
    }else{                              /* Processing for continuing speed display even when waiting for stop  */
        set_RTPM01(P_OFF);                                     /* Stops excitation            */
        r_data = read_BEMF();
        if(r_data != o_data){                                  /*      */
            o_data      = r_data;
            cnt_data3   = cnt_data2;
            cnt_data2   = cnt_data1;
            cnt_data1   = int_cnt;
            cnt_data    = cnt_data1 + cnt_data2 + cnt_data3;
            if((10 < cnt_data1) && (cnt_data1 < 1200)){
                int_cnt     = 0;
                speed_flag = FLG_ON;
            };
        };
    };
    if(capture_flag == FLG_ON){
        if(CR00 != new_clk) {
            old_clk       = new_clk;
            new_clk       = CR00;
            clk_flag      = FLG_ON;
            capture_flag  = FLG_OFF;
        };
    };
    return;
```

```
}
```

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

    target    : uPD78F0714 motor starter kit

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

    NEC Micro Systems,Ltd
*/

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

#define PWM_BASE_REF            769             /* Half of PWM period                   */
#define PWM_F_REF               0               /* Default value of PWM waveform        */
#define PWM_F_MIN               10              /* Minimum value                        */
#define PWM_F_MAX               769             /* Maximum value                        */
#define PWM_DTM_REF             0               /* Dead time                            */

    /*
        m_speed constant: 400 [rpm] * (0.15 [s] / 12.8 [us]) / 2
                                       ----      ----      ---
                                    ↑         ↑         ↑ Number of pole pairs
                                    ↑         ↑ TM00 count clock: 78.125 [kHz] (1 / 78.125 [kHz])
                                    ↑ Period when 400 [rpm] (1 / (400 / 60))
    */
#define UNIT_RPM                2343750   /* 2count */

#define MIN_SPEED               200
#define MAX_SPEED               3200

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

#define CLEAR                   0
#define SET                     1

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

#define INVERTER_SW             P54             /* Inverter operation control port   */
#define INVERTER_SW_MODE        PM54            /* Inverter operation control port   */
#define INTP0                   PM00            /* Overcurrent detection port           */
#define SW2                     PM73
#define SW3                     PM72
#define SW4                     PM71
#define SW5                     PM70

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

#define LD_DATA                 PM4

#define LED_0                   0xc0            /* LED 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_I                   0xcf
#define LED_C                   0xc6
#define LED_H                   0x89            /* HALL                                 */
#define LED_A                   0x88
#define LED_L                   0xc7
#define LED_                    0xff
#define LED_S                   0x92            /* SELF                                 */
#define LED_E                   0x86
#define LED_F                   0x8e
#define LED_P                   0x8c            /* PC                                   */
#define LED_dot                 0x7f

#define IMS_DATA                0xc8

#define WDTM_OFF                0x77            /* For controlling watchdog timer       */
#define WDTM_SET                0x67
#define WDTE_CLR                0xac
#define WDTE_RESET              0x00

#define KEY_WAIT                10              /* Chattering elimination ms            */
#define SW                      (P7&0xf)        /* Switch port                          */

#define START_SW                0x7             /* Identifies switch pressed            */
#define STOP_SW                 0x7
#define FORWARD_SW              0xb
#define REVERSE_SW              0xd
#define MODE_SW                 0xe

static void         INTP0_on(void);
static void         led_set(unsigned char, unsigned char);
static void         led_print(int, char);
static void         wait(int);
```

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

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

    target    : uPD78F0714 motor starter kit

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

    NEC Micro Systems,Ltd
*/

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

#pragma INTERRUPT INTP5             int_speed      rb3

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

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

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

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

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

unsigned char       const_rpm[70];    /* Arrangement is determined assuming revolution speed of 100 [rpm] to 800 [rpm]. */

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

float               carrier_freq;

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

int                 pwm_ff;                     /* Active width: 0 to 1000                                          */
int                 old_ff;                     /* PWM history                                                      */
int                 pwm_base = PWM_BASE_REF;    /* Half of PWM period: Fixed to 1000                                */
int                 speed_ref = 200;            /* Specified speed                                                  */
int                 m_speed;                    /* Motor rotation speed                                             */
unsigned int        int_cnt;                    /* Number of times of carrier synchronization interrupt             */
unsigned int        pid_cnt;                    /* For checking PID control period                                  */
unsigned int        wait_cnt;                   /* For checking processing interval                                 */
unsigned int        print_cnt;                  /* For checking speed display interval                              */
unsigned int        cnt_data1, cnt_data2, cnt_data3;  /* History of number of times of carrier synchronization interrupt */
unsigned int        cnt_data;
char                cw_ccw_flag = CW;           /* Rotation direction                                               */
char                start_flag;                 /* Identification immediately after rotation start                  */
char                speed_flag;                 /* Presence/absence of speed information                            */
char                cw_ccw_wait;                /* Wait for stop of reverse rotation                                */
char                sys_flag;                   /* System operation status                                          */
char                stop_wait;                  /* Indicates whether system is waiting for motor to stop            */
char                ad_flag       = FLG_OFF;    /* Limitation of specified-speed change function                    */

char                clk_flag;
char                capture_flag;
```

```
unsigned int        new_clk, old_clk, bck_clk;      /* Timer value for speed measurement                    */

/*
    Static variable
*/
static const unsigned char    led_data[10] = {      /* For number display                 */
    LED_0, LED_1, LED_2, LED_3, LED_4,
    LED_5, LED_6, LED_7, LED_8, LED_9
};
static char         up_flag = FLG_OFF;              /* Speed update check flag            */
static float        kp_ref = KP_DEF;                /* Kp default value                   */
static float        ki_ref = KI_DEF;                /* Ki default value                   */
static float        kd_ref = KD_DEF;                /* Kd default value                   */
static float        mvn;                            /* Manipulated variable               */
static float        en, en_1, en_2;                 /* Deviation                          */

/*
    Initialization
*/
void
system_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TW0();
    init_TM00();
    init_RTPM01();
    init_AD();
    init_TM50();
    init_openloop();
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(1000);
    start_AD();                                     /* Starts AD conversion               */
    INTP0_on();                                     /* Unmasks overcurrent interrupt      */
    EI();                                           /* Enables interrupt                  */

}

/*
    System start
*/
void
system_start(void)
{
    m_speed         = 0;
    en_1            =
    en              =
    mvn             = 0.0;
    int_cnt         =
    pid_cnt         =
    print_cnt       = 0;
    cnt_data1       =
    cnt_data2       =
    cnt_data3       = 1000;
    cnt_data        = 3000;
    sys_flag        = FLG_ON;
    start_flag      = FLG_ON;
    stop_wait       = FLG_OFF;
    cw_ccw_wait     = FLG_OFF;
    speed_flag      = FLG_OFF;
    up_flag         = FLG_OFF;
    clk_flag        = FLG_OFF;
    capture_flag    = FLG_OFF;
    cnt_flag        = FLG_OFF;
    sync_flag       = FLG_START;
    sync_sw         = 1;
    sync_def        = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt        = sync_def;
    openloop_dim    = 0;
    cnt_limit_tim   = 0;
    cnt_limit_pwm   = 0;
    cnt_limit_rpm   = 0;
    cnt_const_rpm   = 0;
    pwm_ff          = (int)pwm_t0;
    old_ff          = 0;
    start_TM00();
    INTTW0UD_on();          /* Unmasks carrier synchronization interrupt          */
    INVERTER_SW   = INV_ON;     /* INVERTER enable            */
    start_RTPM01();
    start_TW0();
    set_TW0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
    INTP5_on();
}

/*
    Restart from stop of reverse rotation
*/
void
system_restart(void)
{
    m_speed         = 0;
    mvn             =
    en_1            =
    en              = 0.0;
```

```
    int_cnt         =
    pid_cnt         =
    print_cnt       = 0;
    cnt_data1       =
    cnt_data2       =
    cnt_data3       = 1000;
    cnt_data        = 3000;
    start_flag      = FLG_ON;
    speed_flag      = FLG_OFF;
    up_flag         = FLG_OFF;
    clk_flag        = FLG_OFF;
    capture_flag    = FLG_OFF;
    cnt_flag        = FLG_OFF;
    sync_flag       = FLG_START;
    sync_sw         = 1;
    sync_def        = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt        = sync_def;
    openloop_dim    = 0;
    cnt_limit_tim   = 0;
    cnt_limit_pwm   = 0;
    cnt_limit_rpm   = 0;
    cnt_const_rpm   = 0;
    pwm_ff          = (int)pwm_t0;
    old_ff          = 0;
    cw_ccw_wait     = FLG_OFF;
    set_TW0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
}

/*
    System stop
*/
void
system_stop(void)
{
    INVERTER_SW   = INV_OFF;            /* INVERTER disable          */
    INTTW0UD_off();
    INTP5_off();
    stop_RTPM01();
    stop_TW0();
    stop_TM00();
    sys_flag        = FLG_OFF;
    stop_wait       = FLG_OFF;
    cw_ccw_flag   = CW;
    m_speed         = 0;                        /* Speed 0               */
}

/*

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

    rpm_step = 10;

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

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

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

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

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

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

/*
    Enables overcurrent interrupt
*/
static void
INTP0_on(void)
{
    PR0L.1 = 0;                         /* Priority          */
    EGN.0   = 1;                        /* Falling edge      */
    IF0L.1 = CLEAR;                     /* Clears flag       */
    MK0L.1 = CLEAR;                     /* Unmasks           */
```

```
}

/*
    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();
}

/*
    PID control
*/
void
pwm_pid(int cy_time)
{
    unsigned long tmp;
    unsigned long old_tmp, new_tmp;
    int             pwm_tmp;

    if(speed_flag == FLG_ON){       /* Speed is already measured   */
        speed_flag = FLG_OFF;
        if(start_flag == FLG_OFF){
            up_flag = FLG_ON;       /* Speed is already updated */
            if(sync_flag == FLG_OFF){
                if(clk_flag == FLG_ON){
                    clk_flag = FLG_OFF;
                    DI();
                    old_tmp = (unsigned long)old_clk;
                    new_tmp = (unsigned long)new_clk;
                    EI();
                    if(old_tmp > new_tmp){
                        tmp = (65536 - old_tmp) + new_tmp;
                    }else{
                        tmp = new_tmp - old_tmp;
                    }
                    m_speed = (int)(UNIT_RPM/tmp);
                }else{
                    up_flag = FLG_OFF;
                };
            }else{
                m_speed = (int)(194400/(long)cnt_data);
            };
        }else{
            start_flag = FLG_OFF;    /* First speed information is not used */
        }
    }

    if(pid_cnt >= (unsigned int)(cy_time*13)){    /* 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        */
                up_flag = FLG_OFF;
                en_2 = en_1;
                en_1 = en;
                en    = (float)(speed_ref - m_speed);
                mvn  = mvn +
                        kp_ref*(en - en_1) +
                        ki_ref*en +
                        kd_ref*((en - en_1) - (en_1 - en_2));
                if(mvn > PWM_LIMIT_H){
                    mvn = PWM_LIMIT_H;
                }else if(mvn < (float)PWM_LIMIT_L){
                    mvn = (float)PWM_LIMIT_L;
                }
```

```
                pwm_tmp   = pwm_ff + (int)mvn;
                if(pwm_tmp > PWM_F_MAX){ /* Limiter                              */
                    pwm_ff = PWM_F_MAX;
                }else if(pwm_tmp < PWM_F_MIN){
                    pwm_ff = PWM_F_MIN;
                }else{
                    pwm_ff = pwm_tmp;
                }
            }
        }
    }
}

/*
    Reading voltage of speed specification variable resistor
    Converts into specified speed
*/

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

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

/*
    Reading value from switch (P7) and UART

    Reads switch value
    Returns stable value after checking for given time


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

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

    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);
}

/*
    Port settings
*/
static void
init_PORT(void)
{
    INTP0 = IN;                         /* Interrupt of overcurrent notification  */
    SW2 = IN;                           /* START/STOP                             */
    SW3 = IN;                           /* FORWARD                                */
```

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

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

    EGP5 = SET;                         /* Rising edge is valid */
    EGN5 = CLEAR;

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

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

/*
    Clock switching
*/
static void
init_OSC(void)
{

    IMS = IMS_DATA;                     /* Switches memory size */
    clear_WDTM();
    while(OSTC != 0x1f);                /* Waits for oscillation to stabilize */
    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;
    TW0TRGS   = 1;
}

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

static void
start_TW0(void)
{
    TW0C.7 = SET;                       /* Starts timer     */
}

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

/*
    16-bit timer

    CR01      Real-time output port trigger
*/
static void
init_TM00(void)
{
    ES000   = 0;
    ES001   = 0;                        /* Valid edge of TI000 pin: falling edge                */
    CRC000 = 1;                         /* CR00 capture register                                */
    CRC001 = 1;                         /* Capture with reverse phase of valid edge of TI000 pin */
    CRC002 = 0;                         /* CR01 compare register                                */
    PRM001 = SET;
    PRM000 = CLEAR;                     /* Count clock: 78.125 kHz                              */
}

static void
start_TM00(void)
{
    TMIF00 = CLEAR;
    TMC00   = 0x04;                     /* Free-running 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 × 1 channel           */
    DCCTL01 = 0xc0;                 /* PWM modulation output        */
    RTBL01   = 0x3f;                /* Output buffer                */
}

void
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                  /* Sets excitation pattern      */
    CR01    = TM00 + 1;
}

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

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

/*
    AD
*/
static void
init_AD(void)
{
    ADS    = 4;                     /* SPEED */
    ADM    = 0x1a;                  /* 3.6 us */
    ADCS2 = SET;                    /* Enables circuit operation */
}

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

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

/*
    Watchdog timer
*/
static void
init_WDTM(void)
{
    clear_WDTM();
    WDTM = WDTM_SET;
}

void
clear_WDTM(void)
{
    WDTE = WDTE_CLR;
}


void
reset_WDTM(void)
{
    WDTE = WDTE_RESET;
}


/*
    8-bit timer 50
*/
static 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                          */
}

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

/*
    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;
}

/*
    INTP5 interrupt enable
*/
static void
INTP5_on(void)
{
    PIF5 = CLEAR;
    PMK5 = CLEAR;
}

/*
    INTP5 interrupt disable
*/
static void
INTP5_off(void)
{
    PMK5 = SET;
}

/*
    Displaying value on 7-segment LED
*/
static void
led_print(int data, char 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(flag == FLG_OFF){
        led_set(3, led_data[data]);
    }else{
        led_set(3, (unsigned char)led_data[data]&LED_dot);
    }
}

/*
    Displaying data on 7-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;
}

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

    for(i=0;i<cnt;i++){
        start_TM50();                   /* Starts timer                             */
        wait_TM50();                    /* Waits for generation of interrupt request */
        clear_WDTM();                   /* Clears watchdog timer                    */
    }
    return;
}

/*
    Reading BEMF signal
*/
char
read_BEMF(void)
{
    return((char)((P0>>1)&0x7));
}

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

APPENDIX

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

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

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

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

    NEC Micro Systems,Ltd
*/

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

#define PWM_BASE_REF            769                 /* Half of PWM period                  */
#define PWM_F_REF               0                   /* Default value of PWM waveform       */
#define PWM_F_MIN               10                  /* Minimum value                       */
#define PWM_F_MAX               769                 /* Maximum value                       */
#define PWM_DTM_REF             0                   /* Dead time                           */

    /*
        m_speed constant: 400 [rpm] * (0.15 [s] / 12.8 [us]) / 2
                                      ----      ----       ---
                                      ↑         ↑          ↑ Number of pole pairs
                                      ↑         ↑ TM00 count clock: 78.125 [kHz] (1 / 78.125 [kHz])
                                      ↑ Period when 400 [rpm] (1 / (400 / 60))
    */
#define UNIT_RPM                2343750   /* 2count */

#define MIN_SPEED               200
#define MAX_SPEED               3200

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

#define CLEAR                   0
#define SET                     1

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

#define INVERTER_SW             P54                 /* Inverter operation control port */
#define INVERTER_SW_MODE        PM54                /* Inverter operation control port   */
#define INTP0                   PM00                /* Overcurrent detection port          */
#define SW2                     PM73
#define SW3                     PM72
#define SW4                     PM71
#define SW5                     PM70

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

#define LD_DATA                 PM4

#define LED_0                   0xc0                /* LED 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_I                   0xcf
#define LED_C                   0xc6
#define LED_H                   0x89                /* HALL                                */
#define LED_A                   0x88
#define LED_L                   0xc7
#define LED_                    0xff
#define LED_S                   0x92                /* SELF                                */
#define LED_E                   0x86
#define LED_F                   0x8e
#define LED_P                   0x8c                /* PC                                  */
#define LED_dot                 0x7f

#define IMS_DATA                0xc8

#define WDTM_OFF                0x77                /* For controlling watchdog timer      */
#define WDTM_SET                0x67
#define WDTE_CLR                0xac
#define WDTE_RESET              0x00

#define KEY_WAIT                10                  /* Chattering elimination ms           */
#define SW                      (P7&0xf)            /* Switch port                         */
```

```
#define START_SW                    0x7             /* Identifies switch pressed                    */
#define STOP_SW                     0x7
#define FORWARD_SW                  0xb
#define REVERSE_SW                  0xd
#define MODE_SW                     0xe

#ifdef GUI
#define MD_ERROR_HAL                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               /* Receive buffer size of UART00 */
#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_GETSPEED             0x50            /* Actual-speed read                */
#define MY_ID                       0x21            /* Firm identification ID           */
#define VER_MAJOR                   0x01            /* Version information              */
#define VER_MINOR                   0x00
#define VER_SEQ                     0x01
#endif

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

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

```
/*
    BLDCM 120-degree excitation method without position sensor (BEMF: CMP)

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

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

    NEC Micro Systems,Ltd
*/

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

#pragma INTERRUPT INTP5             int_speed      rb3

#ifdef GUI
#pragma INTERRUPT INTSR00           int_uart00   rb1    /* For receiving UART00 command       */
#endif

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

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

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

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

unsigned char       const_rpm[70];    /* Arrangement is determined assuming revolution speed of 100 [rpm] to 800 [rpm] */

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

float               carrier_freq;

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

int                 pwm_ff;                     /* Active width: 0 to 1000                                          */
int                 old_ff;                     /* PWM history                                                      */
int                 pwm_base = PWM_BASE_REF;    /* Half of PWM period: Fixed to 1000                                */
int                 speed_ref = 200;            /* Specified speed                                                  */
int                 m_speed;                    /* Motor rotation speed                                             */
unsigned int        int_cnt;                    /* Number of times of carrier synchronization interrupt             */
unsigned int        pid_cnt;                    /* For checking PID control period                                  */
unsigned int        wait_cnt;                   /* For checking processing interval                                 */
unsigned int        print_cnt;                  /* For checking speed display interval                              */
unsigned int        cnt_data1, cnt_data2, cnt_data3;  /* History of number of times of carrier synchronization interrupt */
unsigned int        cnt_data;
char                cw_ccw_flag = CW;           /* Rotation direction                                               */
char                start_flag;                 /* Identification immediately after rotation start                  */
char                speed_flag;                 /* Presence/absence of speed information                            */
char                cw_ccw_wait;                /* Wait for stop of reverse rotation                                */
char                sys_flag;                   /* System operation status                                          */
char                stop_wait;                  /* Indicates whether system is waiting for motor to stop            */
char                ad_flag       = FLG_OFF;    /* Limitation of specified-speed change function                    */

char                clk_flag;
char                capture_flag;
unsigned int        new_clk, old_clk, bck_clk;  /* Timer value for speed measurement                                */

/*
    Static variable
*/
static const unsigned char    led_data[10] = {  /* For number display                                               */
    LED_0, LED_1, LED_2, LED_3, LED_4,
    LED_5, LED_6, LED_7, LED_8, LED_9
};
static char         up_flag = FLG_OFF;          /* Speed update check flag                                          */
static float        kp_ref = KP_DEF;            /* Kp default value                                                 */
static float        ki_ref = KI_DEF;            /* Ki default value                                                 */
static float        kd_ref = KD_DEF;            /* Kd default value                                                 */
static float        mvn;                        /* Manipulated variable                                             */
static float        en, en_1, en_2;             /* Deviation                                                        */
#ifdef GUI
static unsigned char err_flag = FLG_OFF;        /* Error notification flag                                          */
static unsigned char uart00_data[RX_BUFF_SIZE]; /* UART00 receive buffer                                            */
static unsigned char read_p, write_p;           /* Buffer pointer                                                   */
static int          kp_tmp, ki_tmp, kd_tmp;     /* Kp, Ki, Kd communication data                                    */
#endif

/*
    Initialization
*/
```

```
void
system_init(void)
{
    init_OSC();
    init_WDTM();
    init_PORT();
    init_TW0();
    init_TM00();
    init_RTPM01();
    init_AD();
    init_TM50();
    init_openloop();
#ifdef GUI
    init_UART00();
    led_set(0, LED_P);
    led_set(1, LED_C);
    kp_tmp = (int)(kp_ref * 10000);
    ki_tmp = (int)(ki_ref * 10000);
    kd_tmp = (int)(kd_ref * 10000);
#else
    led_set(0, LED_S);
    led_set(1, LED_E);
    led_set(2, LED_L);
    led_set(3, LED_F);
    wait(1000);
#endif
    start_AD();                         /* Starts A/D conversion            */
    INTP0_on();                         /* Unmasks overcurrent interrupt    */
    EI();                               /* Enables interrupt                */

}

/*
    System start
*/
void
system_start(void)
{
    m_speed         = 0;
    en_1            =
    en              =
    mvn             = 0.0;
    int_cnt         =
    pid_cnt         =
    print_cnt       = 0;
    cnt_data1       =
    cnt_data2       =
    cnt_data3       = 1000;
    cnt_data        = 3000;
    sys_flag        = FLG_ON;
    start_flag      = FLG_ON;
    stop_wait       = FLG_OFF;
    cw_ccw_wait     = FLG_OFF;
    speed_flag      = FLG_OFF;
    up_flag         = FLG_OFF;
    clk_flag        = FLG_OFF;
    capture_flag    = FLG_OFF;
    cnt_flag        = FLG_OFF;
    sync_flag       = FLG_START;
    sync_sw         = 1;
    sync_def        = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt        = sync_def;
    openloop_dim    = 0;
    cnt_limit_tim   = 0;
    cnt_limit_pwm   = 0;
    cnt_limit_rpm   = 0;
    cnt_const_rpm   = 0;
    pwm_ff          = (int)pwm_t0;
    old_ff          = 0;
    start_TM00();
    INTTW0UD_on();          /* Unmasks carrier synchronization interrupt        */
    INVERTER_SW = INV_ON;   /* INVERTER enable          */
    start_RTPM01();
    start_TW0();
    set_TW0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
    INTP5_on();
}

/*
    Restart from stop of reverse rotation
*/
void
system_restart(void)
{
    m_speed     = 0;
    mvn         =
    en_1        =
    en          = 0.0;
    int_cnt     =
    pid_cnt     =
    print_cnt   = 0;
    cnt_data1   =
    cnt_data2   =
    cnt_data3   = 1000;
    cnt_data    = 3000;
    start_flag  = FLG_ON;
    speed_flag  = FLG_OFF;
    up_flag     = FLG_OFF;
```

```
    clk_flag            = FLG_OFF;
    capture_flag        = FLG_OFF;
    cnt_flag            = FLG_OFF;
    sync_flag           = FLG_START;
    sync_sw             = 1;
    sync_def            = (unsigned int)((5.0 / (float)rpm_t0) / (float)carrier_freq);
    sync_cnt            = sync_def;
    openloop_dim        = 0;
    cnt_limit_tim       = 0;
    cnt_limit_pwm       = 0;
    cnt_limit_rpm       = 0;
    cnt_const_rpm       = 0;
    pwm_ff              = (int)pwm_t0;
    old_ff              = 0;
    cw_ccw_wait         = FLG_OFF;
    set_TW0(pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base-pwm_ff,
            pwm_base);
    set_RTPM01(sync_tbl[cw_ccw_flag][5]);
    wait(50);
    set_RTPM01(sync_tbl[cw_ccw_flag][0]);
    wait(400);
}

/*
    System stop
*/
void
system_stop(void)
{
    INVERTER_SW   = INV_OFF;            /* INVERTER disable          */
    INTTW0UD_off();
    INTP5_off();
    stop_RTPM01();
    stop_TW0();
    stop_TM00();
    sys_flag            = FLG_OFF;
    stop_wait           = FLG_OFF;
    cw_ccw_flag         = CW;
    m_speed             = 0;                        /* Speed 0               */
}

/*

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

    rpm_step = 10;

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

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

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

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

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

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

/*
    Overcurrent interrupt enable
*/
static void
INTP0_on(void)
{
    PR0L.1 = 0;                         /* Priority          */
    EGN.0   = 1;                        /* Falling edge      */
    IF0L.1 = CLEAR;                     /* Clears flag       */
    MK0L.1 = CLEAR;                     /* Unmasks           */
}

/*
    Error display
*/
void
print_error(char eno)
{
    switch(eno){
    case ERROR_HALL:
```

```
#ifdef GUI
            err_flag = MD_ERROR_HALL;
#endif
            led_set(0, LED_H);
            led_set(1, LED_A);
            led_set(2, LED_L);
            led_set(3, LED_L);
            break;

        case ERROR_OC:
#ifdef GUI
            err_flag = MD_ERROR_OC;
#endif
            led_set(0, LED_);
            led_set(1, LED_O & LED_dot);
            led_set(2, LED_C & LED_dot);
            led_set(3, LED_);
            break;

        case ERROR_MOTOR:
#ifdef GUI
            err_flag = MD_ERROR_MOTOR;
#endif
            led_set(0, LED_F);
            led_set(1, LED_A);
            led_set(2, LED_I);
            led_set(3, LED_L);
            break;

        case ERROR_S_OC:
#ifdef GUI
            err_flag = MD_ERROR_OC;
#endif
            led_set(0, LED_S & LED_dot);
            led_set(1, LED_);
            led_set(2, LED_O & LED_dot);
            led_set(3, LED_C & LED_dot);
            break;

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

/*
    PID control
*/
void
pwm_pid(int cy_time)
{
    unsigned long tmp;
    unsigned long old_tmp, new_tmp;
    int             pwm_tmp;

    if(speed_flag == FLG_ON){           /* Speed is already measured   */
        speed_flag = FLG_OFF;
        if(start_flag == FLG_OFF){
            up_flag = FLG_ON;           /* Speed is already updated */
            if(sync_flag == FLG_OFF){
                if(clk_flag == FLG_ON){
                    clk_flag = FLG_OFF;
                    DI();
                    old_tmp = (unsigned long)old_clk;
                    new_tmp = (unsigned long)new_clk;
                    EI();
                    if(old_tmp > new_tmp){
                        tmp = (65536 - old_tmp) + new_tmp;
                    }else{
                        tmp = new_tmp - old_tmp;
                    }
                    m_speed = (int)(UNIT_RPM/tmp);
                }else{
                    up_flag = FLG_OFF;
                };
            }else{
                m_speed = (int)(194400/(long)cnt_data);
            };
        }else{
            start_flag = FLG_OFF;     /* First speed information is not used */
        }
    }

    if(pid_cnt >= (unsigned int)(cy_time*13)){     /* 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      */
                up_flag = FLG_OFF;
                en_2 = en_1;
                en_1 = en;
                en    = (float)(speed_ref - m_speed);
                mvn   = mvn +
                            kp_ref*(en - en_1) +
                            ki_ref*en +
```

```
                        kd_ref*((en - en_1) - (en_1 - en_2));
                if(mvn > PWM_LIMIT_H){
                    mvn = PWM_LIMIT_H;
                }else if(mvn < (float)PWM_LIMIT_L){
                    mvn = (float)PWM_LIMIT_L;
                }
                pwm_tmp   = pwm_ff + (int)mvn;
                if(pwm_tmp > PWM_F_MAX){ /* Limiter                              */
                    pwm_ff = PWM_F_MAX;
                }else if(pwm_tmp < PWM_F_MIN){
                    pwm_ff = PWM_F_MIN;
                }else{
                    pwm_ff = pwm_tmp;
                 }
            }
        }
    }
}

/*
    Reading voltage of speed specification variable resistor
    Converts into specified speed
*/

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

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

/*
    Reading value from switch (P7) and UART

    Reads switch value
    Returns stable value after checking for given time


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

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

    data = SW;                          /* 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 */
    }
#else
    int               ss;
    unsigned char hi, lo;
```

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

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

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

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

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

case MD_CMD_SETSSPEED:                  /* Set Setting Speed */
    lo = wait_UART00();
    hi = wait_UART00();
    set_UART00(data);
    ss = ((int)hi<<8) + lo;
    if(hi > 0x80){                      /* CCW */
        ss = ~ss + 1;
        data = REVERSE_TR;
        if(sys_flag == FLG_OFF){
            cw_ccw_flag = CCW;
        }
    }else{
        data = FORWARD_TR;
        if(sys_flag == FLG_OFF){
            cw_ccw_flag = CW;
        }
    }
    speed_ref = ss;
    break;

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

case MD_CMD_SETPIDPARAM:                /* Sets PID feedback gain */
    lo = wait_UART00();
    hi = wait_UART00();
    kp_tmp = (((int)(hi))<<8) + lo;
    lo = wait_UART00();
    hi = wait_UART00();
    ki_tmp = (((int)(hi))<<8) + lo;
    lo = wait_UART00();
    hi = wait_UART00();
    kd_tmp = (((int)(hi))<<8) + lo;
    set_UART00(data);
    kp_ref = ((float)kp_tmp)/10000;
    ki_ref = ((float)ki_tmp)/10000;
    kd_ref = ((float)kd_tmp)/10000;
    break;

case MD_CMD_GETPIDPARAM:                /* Reads PID feedback gain */
    set_UART00(data);
    kp_tmp = ((int)kp_ref)*10000;
    set_UART00((unsigned char)((unsigned int)(kp_tmp)&0xff));
    set_UART00((unsigned char)(((unsigned int)(kp_tmp)>>8)&0xff));
    ki_tmp = ((int)ki_ref)*10000;
    set_UART00((unsigned char)((unsigned int)(ki_tmp)&0xff));
    set_UART00((unsigned char)(((unsigned int)(ki_tmp)>>8)&0xff));
    kd_tmp = ((int)kd_ref)*10000;
    set_UART00((unsigned char)((unsigned int)(kd_tmp)&0xff));
    set_UART00((unsigned char)((unsigned int)((kd_tmp)>>8)&0xff));
    break;

case MD_CMD_GETSPEED:                   /* Reads rotation speed */
    if(err_flag != FLG_OFF){
        set_UART00(err_flag);
        err_flag = FLG_OFF;
    }else{
        set_UART00(data);
        if(m_speed == 0){
            set_UART00(0);
            set_UART00(0);
        }else{
            ss = m_speed;
            if(cw_ccw_wait == FLG_OFF){
                if(cw_ccw_flag == CCW){
```

```
                        ss = ~ss + 1;
                    }
                }else{   /* Waits for stop of reverse rotation */
                    if(cw_ccw_flag == CW){
                        ss = ~ss + 1;
                    }
                }
                set_UART00((unsigned char)((unsigned int)(ss)&0xff));
                set_UART00((unsigned char)(((unsigned int)(ss)>>8)&0xff));
            }
        }
        break;

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

/*
    Port settings
*/
static void
init_PORT(void)
{
    INTP0 = IN;                         /* Interrupt of overcurrent notification */
    SW2 = IN;                           /* START/STOP                            */
    SW3 = IN;                           /* FORWARD                               */
    SW4 = IN;                           /* REVERSE                               */
    SW5 = IN;                           /* MODE                                  */

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

    EGP5 = SET;                         /* Rising edge is valid                  */
    EGN5 = CLEAR;

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

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

/*
    Clock switching
*/
static void
init_OSC(void)
{

    IMS = IMS_DATA;                     /* Switches memory size                  */
    clear_WDTM();
    while(OSTC != 0x1f);                /* Waits for oscillation to stabilize    */
    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;
    TW0TRGS = 1;
}

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

static void
start_TW0(void)
{
    TW0C.7 = SET;                           /* Starts timer     */
}

static void
```

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

/*
    16-bit timer

    CR01    Real-time output port trigger
*/
static void
init_TM00(void)
{
    ES000   = 0;
    ES001   = 0;                        /* Valid edge of TI000 pin: falling edge                    */
    CRC000 = 1;                         /* CR00 capture register                                    */
    CRC001 = 1;                         /* Capture with reverse phase of valid edge of TI000 pin    */
    CRC002 = 0;                         /* CR01 compare register                                    */
    PRM001 = SET;
    PRM000 = CLEAR;                     /* Count clock: 78.125 kHz                                  */
}

static void
start_TM00(void)
{
    TMIF00 = CLEAR;
    TMC00   = 0x04;                     /* Free-running 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 × 1 channel       */
    DCCTL01 = 0xc0;                     /* PWM modulation output */
    RTBL01   = 0x3f;                    /* Output buffer           */
}

void
set_RTPM01(unsigned char data)
{
    RTBL01 = data;                      /* Sets excitation pattern    */
    CR01    = TM00 + 1;
}

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

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

/*
    AD
*/
static void
init_AD(void)
{
    ADS    = 4;                         /* SPEED */
    ADM    = 0x1a;                      /* 3.6 us */
    ADCS2 = SET;                        /* Enables circuit operation */
}

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

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

/*
    Watchdog timer
*/
static void
init_WDTM(void)
{
    clear_WDTM();
    WDTM = WDTM_SET;
}

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

```
}


void
reset_WDTM(void)
{
    WDTE = WDTE_RESET;
}


/*
    8-bit timer 50
*/
static 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                           */
}

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

/*
    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;
}

/*
    INTP5 interrupt enable
*/
static void
INTP5_on(void)
{
    PIF5 = CLEAR;
    PMK5 = CLEAR;
}

/*
    INTP5 interrupt disable
*/
static void
INTP5_off(void)
{
    PMK5 = SET;
}

/*
    Displaying value on 7-segment LED
*/
static void
led_print(int data, char 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(flag == FLG_OFF){
        led_set(3, led_data[data]);
    }else{
        led_set(3, (unsigned char)led_data[data]&LED_dot);
    }
}

/*
    Displaying data on 7-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;
}

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

    for(i=0;i<cnt;i++){
        start_TM50();                   /* Starts timer                             */
        wait_TM50();                    /* Waits for generation of interrupt request */
        clear_WDTM();                   /* Clears watchdog timer                    */
    }
    return;
}

/*
    Reading BEMF signal
*/
char
read_BEMF(void)
{
    return((char)((P0>>1)&0x7));
}

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

#ifdef GUI
/*
    UART00 settings
*/
static void
init_UART00(void)
{
    PM10        = IN;
    PM11        = OUT;
    PM13        = IN;
    PM14        = OUT;
    RTS         = 1;
    P14         = 1;
    BRGC00      = 0x56;                 /* 115200 */
    PS001       = CLEAR;
    PS000       = CLEAR;
    CL00        = SET;                  /* 8 bits */
    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;
}

/*
    Returning UART00 receive buffer data
*/
static unsigned char
get_UART00(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
wait_UART00(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
set_UART00(unsigned char data)
{
    while(STIF00 != SET);               /* Waits for transmission completion */
    STIF00 = CLEAR;
    TXS00 = data;                       /* Transmits                        */
}

/*
    For UART00 command reception
*/
__interrupt void
int_uart00(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            */
}
#endif
```

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