To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

Inverter Control by μPD78F0714

120° Excitation Method Control by Zero-Cross Detection

μPD78F0714

Document No. U17297EJ1V0AN00 (1st edition)
Date Published May 2005 NS 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, 2005. 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

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC Electronics product in your application, please contact the NEC Electronics office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

[GLOBAL SUPPORT]
http://www.necel.com/en/support/support.html

NEC Electronics America, Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65030

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-244 58 45
- **Tyskland Filial**
  Taeby, Sweden
  Tel: 08-63 80 820
- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-558-3737

NEC Electronics Shanghai Ltd.
Shanghai, P.R. China
Tel: 021-5888-5400

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 6253-8311

J04.1

INTRODUCTION

Target Readers

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

- μPD78F0714

Purpose

The purpose of this application note is to help the user understand how a brushless DC motor is controlled via the 120° sensorless drive excitation method that uses PWM output and A/D converter input as a system example of the timer/counter function of the μPD78F0714.

Organization

This application note is divided into the following sections.

- Control method
- Hardware configuration
- Software configuration
- Program list

How to Use This Manual

It is assumed that the reader of this application note has 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**.

For details of instruction functions

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

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	This manual

Documents related to development softwere 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 hardwere 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 CONTROL METHOD

1.1 Outline of Brushless DC Motor Control

A brushless DC (BLDC) motor consists of a stator, coil, and rotor. The rotor, which includes a permanent magnet, is rotated by the action of the magnetic field generated by the coil of the stator.

The magnetic field is generated by exciting the coil wound around the stator in a specific sequence. By controlling the intensity and cycle of the magnetic field with a microcontroller, the torque response and the number of revolutions of the motor can be controlled.

This section explains how to control a BLDC motor without a sensor by using the μPD78F0714.

Figure 1-3 shows an example of the circuit of a three-phase brushless DC motor. The internal PWM output function of the microcontroller is used to control the current that flows through the motor, by using a transistor array consisting of six transistors.

The magnetic field is generated by controlling the excitation pattern of the six transistors as shown in Table 1-1.

Table 1-1. Excitation Pattern

Excitation Pattern	Upper Arm			Lower Arm			Excitation Direction
	U	V	W	$\overline{U}$	$\overline{V}$	$\overline{W}$	
<1>	Active	Inactive	Inactive	Inactive	Active	Inactive	U $\rightarrow \overline{V}$
<2>	Active	Inactive	Inactive	Inactive	Inactive	Active	U $\rightarrow \overline{W}$
<3>	Inactive	Active	Inactive	Inactive	Inactive	Active	V $\rightarrow \overline{W}$
<4>	Inactive	Active	Inactive	Active	Inactive	Inactive	V $\rightarrow \overline{U}$
<5>	Inactive	Inactive	Active	Active	Inactive	Inactive	W $\rightarrow \overline{U}$
<6>	Inactive	Inactive	Active	Inactive	Active	Inactive	W $\rightarrow \overline{V}$

Figure 1-1. Three-Phase DC Motor Voltage Waveform

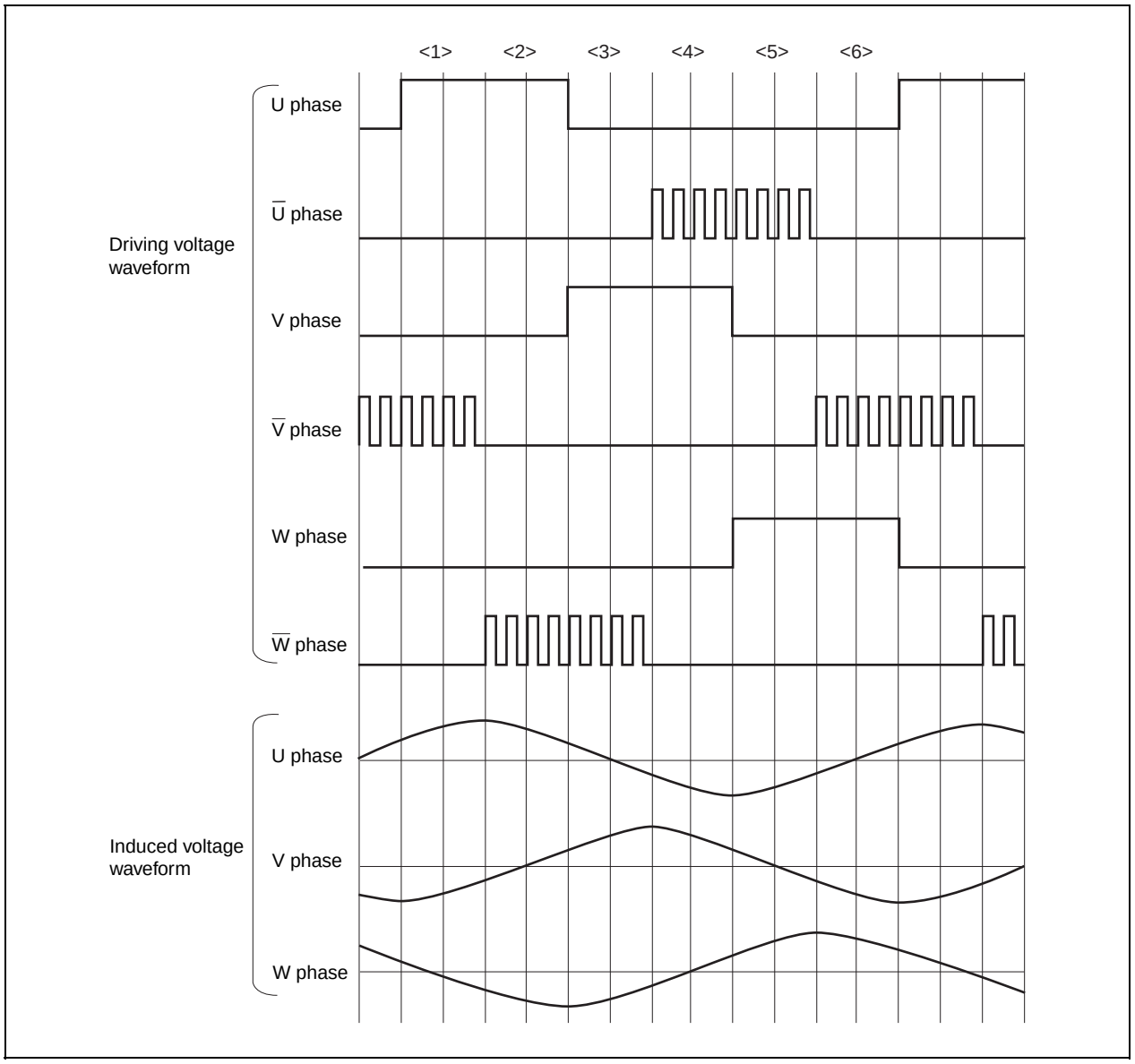
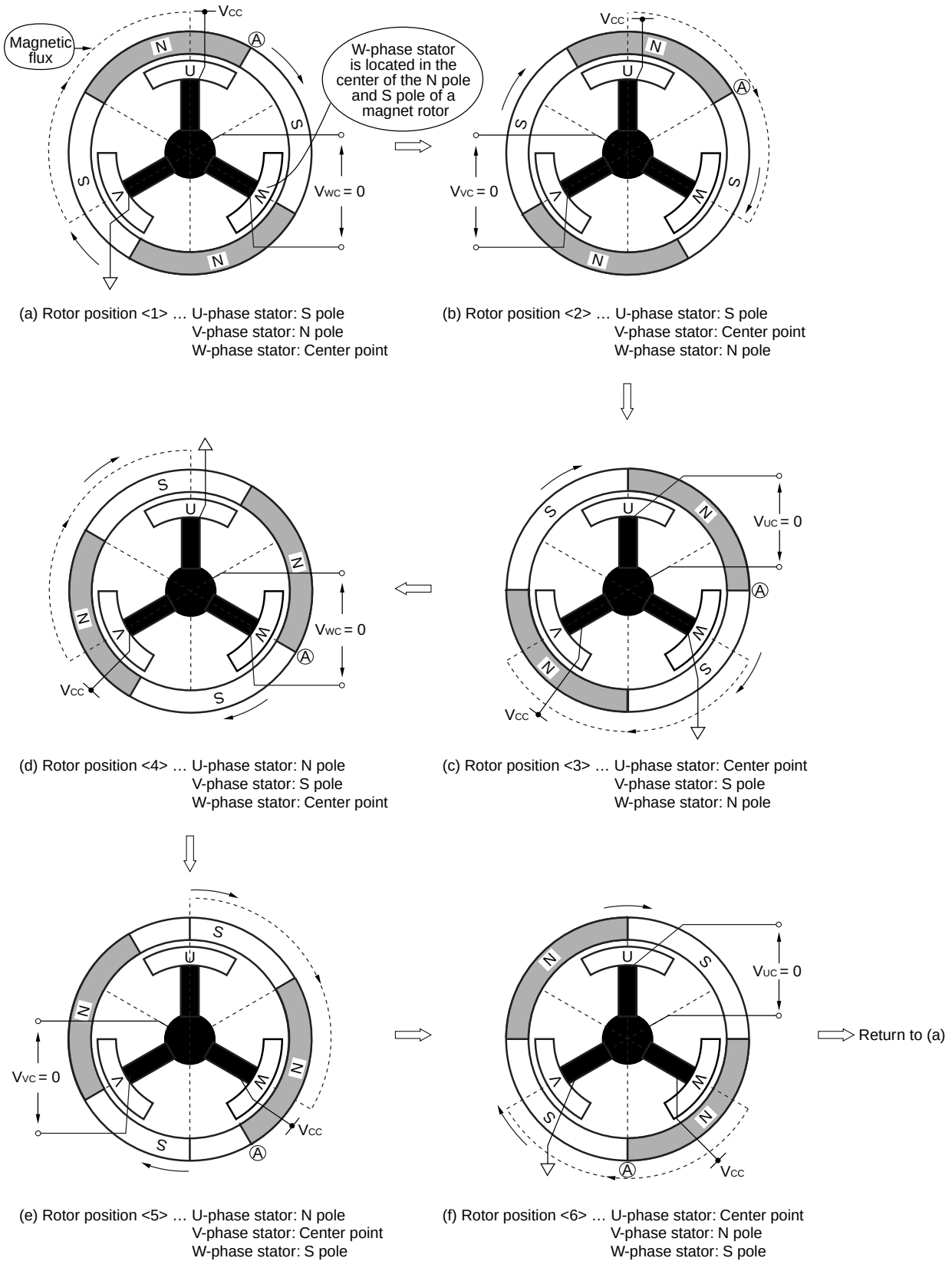


Figure 1-2. Rotor Position Detection Principle



(a) Rotor position <1> … U-phase stator: S pole
V-phase stator: N pole
W-phase stator: Center point

(b) Rotor position <2> … U-phase stator: S pole
V-phase stator: Center point
W-phase stator: N pole

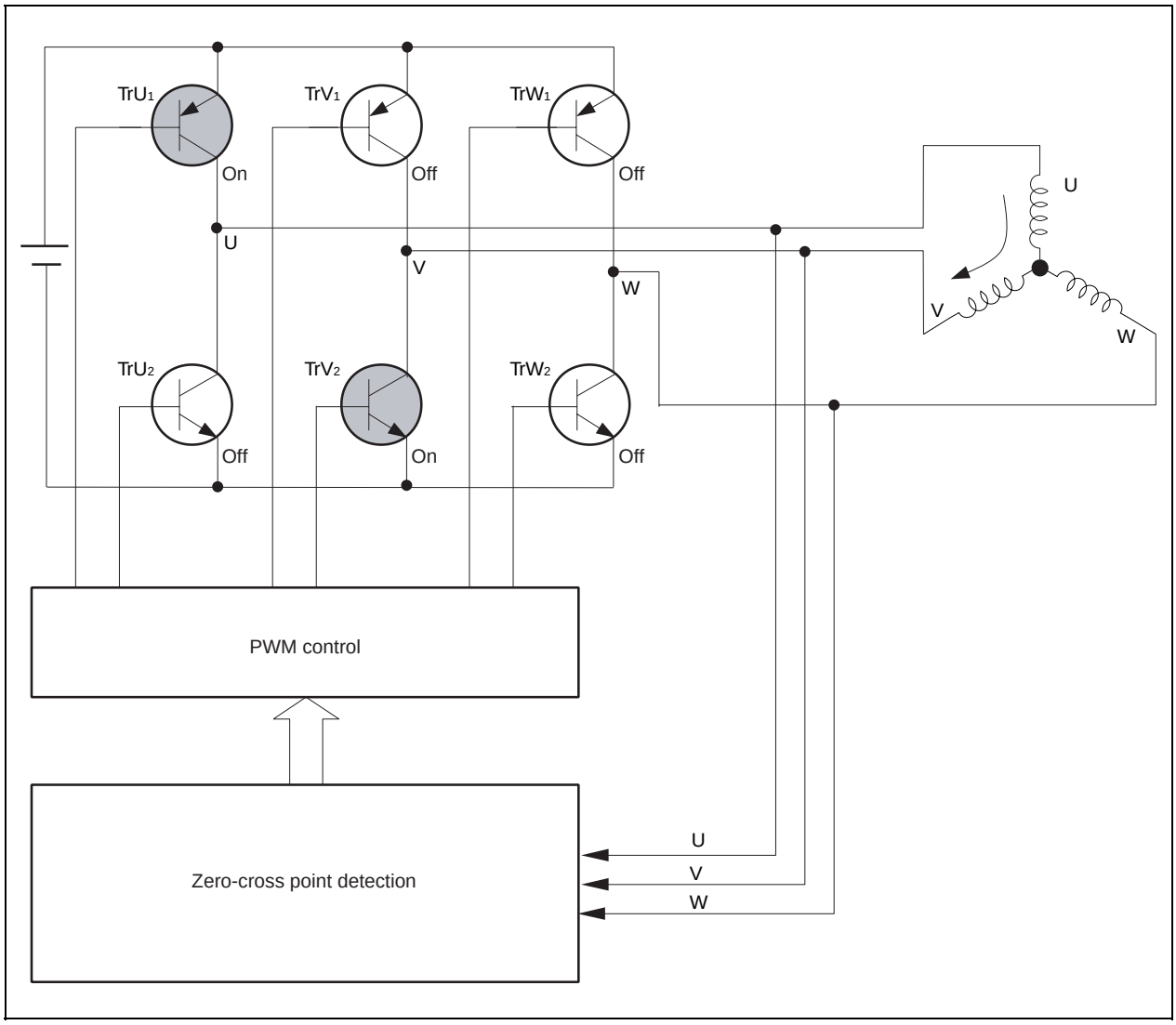
(c) Rotor position <3> … U-phase stator: Center point
V-phase stator: S pole
W-phase stator: N pole

(d) Rotor position <4> … U-phase stator: N pole
V-phase stator: S pole
W-phase stator: Center point

(e) Rotor position <5> … U-phase stator: N pole
V-phase stator: Center point
W-phase stator: S pole

(f) Rotor position <6> … U-phase stator: Center point
V-phase stator: N pole
W-phase stator: S pole

Figure 1-3. Configuration of Three-Phase Brushless DC Motor



The 120° control method for a BLDC motor without a sensor is described below.

To control a BLDC motor, the rotor position must be known.
To control a BLDC motor without a sensor, the rotor position is estimated using induced voltage.

The induced voltage is in phase with the driving voltage waveform and its waveform is close to a sine wave, as shown in Figure 1-1. Figure 1-2 illustrates how the polarity of the stator of the motor is switched and how the magnet rotor revolves.

As shown in Figures 1-1 and 1-2, a three-phase DC motor rotates its rotor by switching the three driving current patterns on the three coil phases.

During period <1> in Figure 1-1, for example, transistor TrU_1 in Figure 1-3 is turned on by the U-phase driving pin, and TrV_2 is turned on by the V-phase driving pin, causing the current to flow from the U-phase driving pin toward the V-phase driving pin. At this time, the W-phase coil seems to be disconnected from the driver circuit and induced voltage is generated.

This induced voltage is used to detect the rotor position.

To control the number of revolutions of the motor, the voltage applied to the motor is controlled to change the current flowing through the coil. To change the voltage, a waveform that is controlled by PWM is applied to the transistor.

The voltage is changed by applying a waveform (PWM waveform) in proportion to the voltage to be applied, to the transistors on the lower arm side (TrU_2, TrV_2, and TrW_2) while the transistors on the upper arm side (TrU_1, TrV_1, and TrW_1) are on.

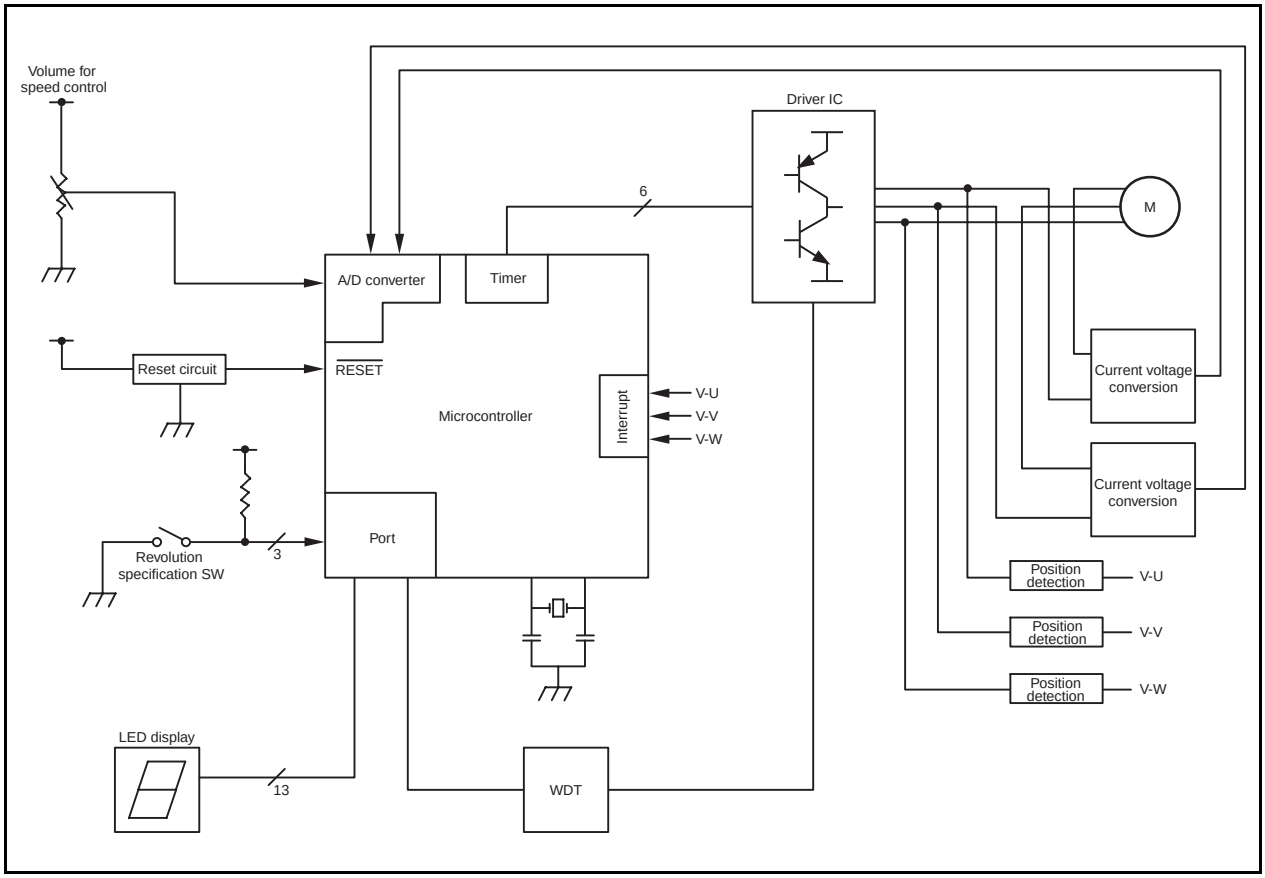
CHAPTER 2 HARDWARE CONFIGURATION

This chapter describes the hardware configuration.

2.1 Configuration

The reference system's main functions are described below. In this reference system, when the revolution specification switch is pressed after power application, the motor starts revolving in the direction specified.

Figure 2-1. Overall System Configuration



(1) Volume for speed control

Volume for increasing and decreasing the number of revolutions of the motor

(2) Revolution specification SW

CW, CCW, and STOP switches

(3) LED display

LED displaying the number of revolutions, operation time, etc.

(4) WDT

Watchdog timer

(5) Driver IC

Driver for driving motor

(6) Current voltage converter

Converting the motor driving current to voltage, used for detecting overcurrent

(7) Position detector

Rotor position estimation signal output from the induced voltage

2.2 Circuit Diagram

Figure 2-2 shows a diagram of the sample reference system circuit.

This sample reference system circuit diagram includes the μPD78F0714, a reset circuit, oscillator, a pin handling microcontroller peripheral block, operation mode switch block, LED output block, watchdog timer circuit block, drive circuit block, motor controller, and motor revolution indicator.

(1) Microcontroller and microcontroller peripheral block

The μPD78F0714 includes a reset circuit, an oscillator that uses a resonator, and a block for handling the MODE pin and unused pins.

(2) Operation mode switch block

This includes switches that set the operation mode as CW or CCW operation.

(3) LED output block

This block includes 16 LEDs, which are used to indicate the revolution speed (rpm), errors, etc.

(4) Watchdog timer circuit block

This block uses the μPD74HC123A to output stop signals when pulse output from the μPD78F0714 stops for one ms or longer.

(5) Drive circuit block

The 6-phase outputs from the inverter timer are converted to U-, V-, and W-phase output for the motor driver. This drive circuit is not shown in detail in this example, since it varies depending on the motor's specifications.

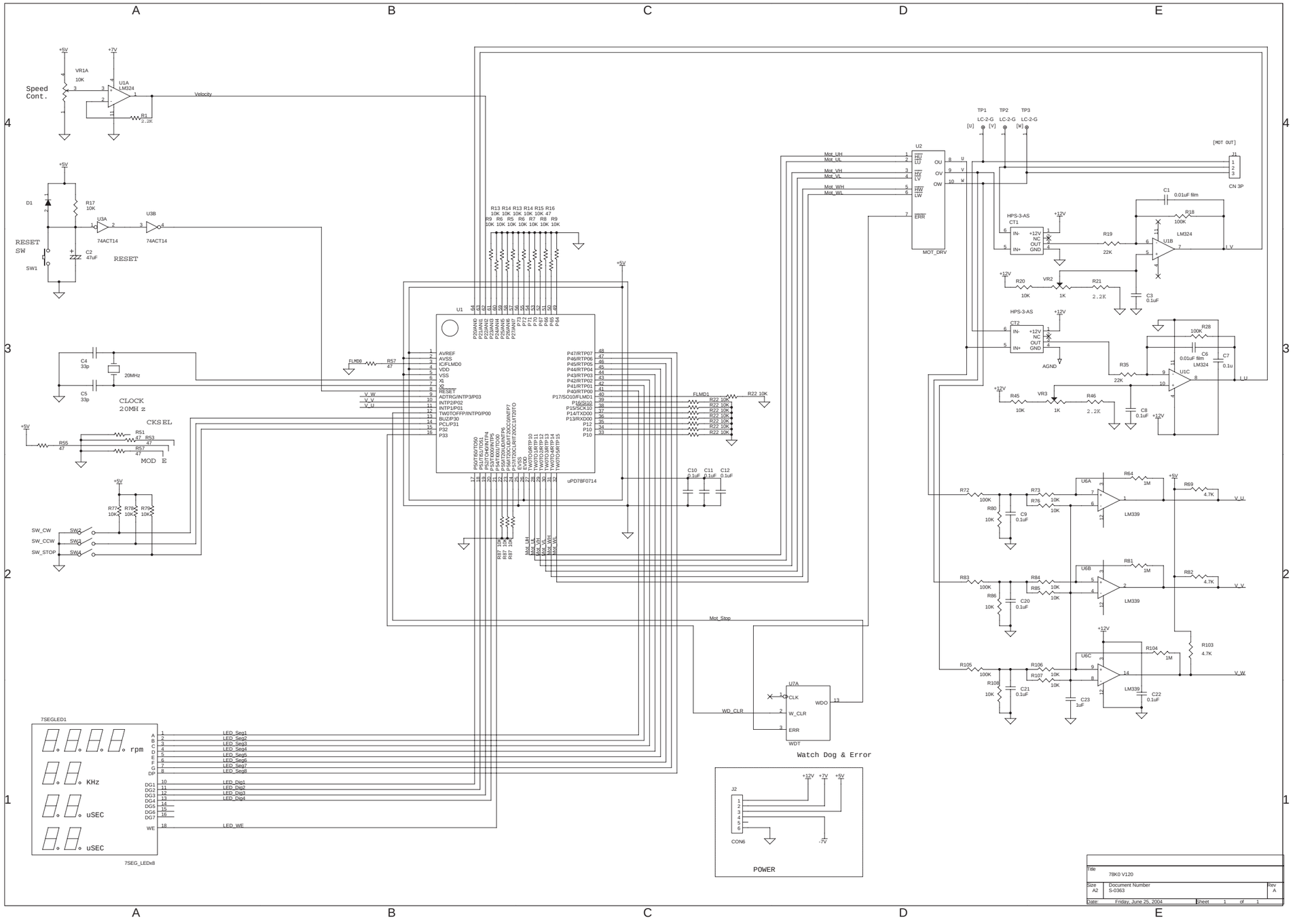
(6) Motor controller

This block includes the HPS-3-AS, LM324, and other devices that are used to measure the motor's U and V drive currents via A/D conversion.

(7) Motor rotation indicator

This block includes a volume adjuster and the LM324 for setting the motor's revolution speed (rpm).

[MEMO]


Speed Cont.
VR1A
10K
U1A
LM324
Velocity
R1
2.2K
+5V
+7V
D1
R17
10K
U3A
74ACT14
U3B
74ACT14
RESET
SW
SW1
C2
47uF
RESET
C4
33p
20MHz
C5
33p
CLOCK
20MH z
CKSEL
R51
47
R53
47
R55
47
R57
47
MOD E
R77
10K
R78
10K
R79
10K
SW_CW
SW_CCW
SW_STOP
SW2
SW3
SW4
R13 R14 R13 R14 R15 R16
10K 10K 10K 10K 10K 47
R9 R5 R5 R6 R7 R8 R9
10K 10K 10K 10K 10K 10K 10K
U1
AVREF
AVSS
IC/FLMD0
VDD
VSS
X1
X2
RESET
ADTRG/INTP3/P03
INTP2/P02
INTP1/P01
TW0TOFFP/INTP0/P00
BUZ/P30
PCL/P31
P32
P33
P47/RTP07
P46/RTP06
P45/RTP05
P44/RTP04
P43/RTP03
P42/RTP02
P41/RTP01
P40/RTP00
P17/SO10/FLMD1
P16/SI10
P15/SCK10
P14/TXD00
P13/RXD00
P12
P10
P10
FLMD0
R57
47
FLMD1
R22 10K
V_W
V_V
V_U
uPD78F0714
C10
0.1uF
C11
0.1uF
C12
0.1uF
Mot_Stop
U7A
CLK
W_CLR
ERR
WDT
WDO
WD_CLR
Watch Dog & Error
7SEGLED1
rpm
KHz
uSEC
uSEC
LED_Seg1
LED_Seg2
LED_Seg3
LED_Seg4
LED_Seg5
LED_Seg6
LED_Seg7
LED_Seg8
LED_Dig1
LED_Dig2
LED_Dig3
LED_Dig4
LED_WE
7SEG_LEDx8
J2
CON6
+12V
+7V
+5V
-7V
POWER
TP1
TP2
TP3
LC-2-G
[U]
[V]
[W]
[MOT OUT]
J1
CN 3P
U2
Mot_UH
Mot_UL
Mot_VH
Mot_VL
Mot_WH
Mot_WL
OU
OV
OW
ERR
MOT_DRV
C1
0.01uF film
HPS-3-AS
CT1
R18
100K
R19
22K
U1B
LM324
I_V
R20
10K
VR2
1K
R21
2.2K
C3
0.1uF
CT2
R28
100K
C6
0.01uF film
C7
0.1u
R35
22K
U1C
LM324
I_U
AGND
R45
10K
VR3
1K
R46
2.2K
C8
0.1uF
U6A
R64
1M
R69
4.7K
V_U
R72
100K
R73
10K
R76
10K
R80
10K
C9
0.1uF
LM339
U6B
R81
1M
R82
4.7K
V_V
R83
100K
R84
10K
R85
10K
R86
10K
C20
0.1uF
U6C
R104
1M
R103
4.7K
V_W
R105
100K
R106
10K
R107
10K
R108
10K
C21
0.1uF
C23
1uF
C22
0.1uF
Title
78K0 V120
Size
A2
Document Number
S-0363
Rev
A
Date:
Friday, June 25, 2004
Sheet
1
of
1

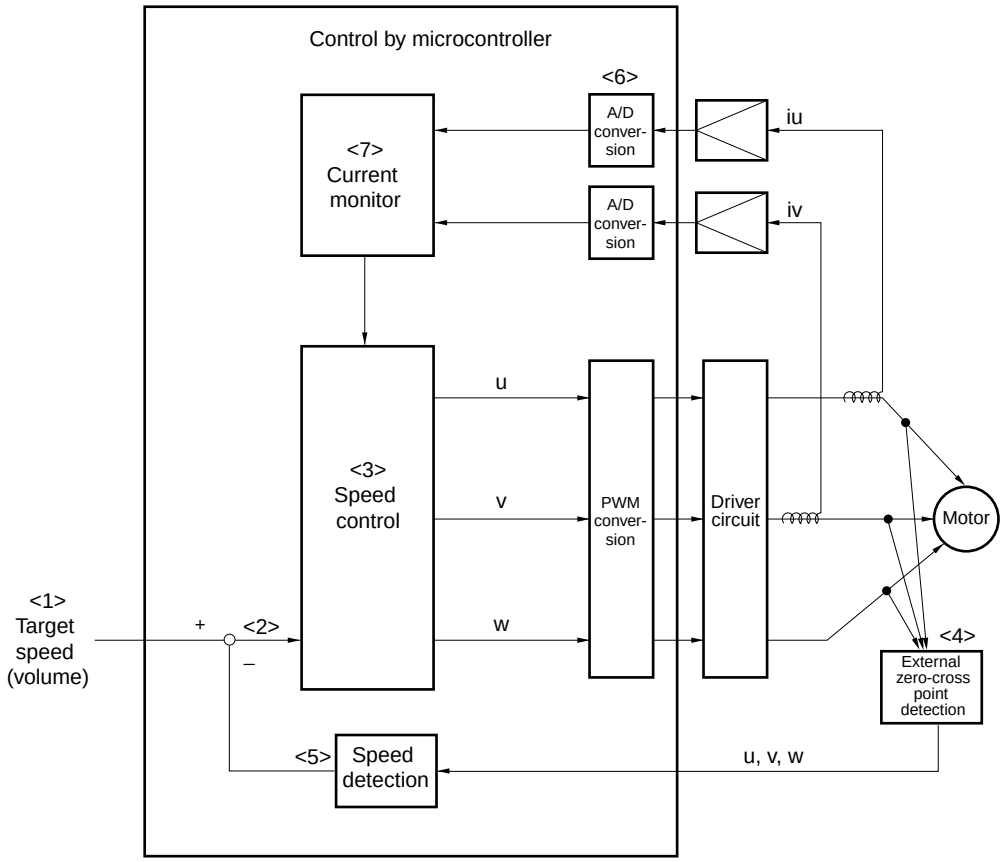
[MEMO]

CHAPTER 3 SOFTWARE CONFIGURATION

3.1 Control Block

Figure 3-1 shows a diagram of the software control block of the reference system.

Figure 3-1. Diagram of Software Control Block of Reference System



<1> The volume is loaded by the A/D converter as a target speed.

<2> The speed to be calculated is determined by the difference from the present speed.

<3> Values of phases U, V, and W are set.

<4> The zero-cross point is detected by an induced voltage (by hardware).

<5> The speed is calculated by the interval of zero-cross point in each phase.

<6> An overcurrent is detected by the A/D converter.

<7> The speed is controlled by a current value.

3.2 Peripheral I/O

The following types of peripheral I/O functions are used in this reference system.

Table 3-1. List of Peripheral I/O Functions

Function	Peripheral I/O Function Name (μPD78F0714)
Inverter timer	Timer W0 (TMW0)
10 ms timer	Timer 00 (TM00)
Motor control timer	Timer 00 (TM00)
Deray control timer	Timer 51 (TM51)
U-phase current	ANI0
V-phase current	ANI1
Setting speed (volume)	ANI2
U-phase zero-cross input	INTP1
V-phase zero-cross input	INTP2
W-phase zero-cross input	INTP3
CW key input	P30
CCW key input	P31
STOP key input	P32
WDT reset output	P33
LED output	P40-P47, P50-P57

(1) Description of peripheral I/O functions

(a) Inverter timer

Inverter timer is used to output PWM waveforms.

In this application circuit example, the settings are as shown below.

- 10 kHz symmetrical triangular waveform mode
- Inverter timer output: Low active
- When TW0TOFFP pin input is at high level, PWM output is stopped.

(b) Motor control timer

Motor control timer is used to issue interrupts at a 2 ms interval.

(c) 10 ms timer

10 ms timer is used to issue interrupts at a 10 ms interval.

(d) Delay control timer

Delay control timer is used to issue interrupts for 30 degrees delay switching.

(e) Current value input

ANI0: U-phase current value (–5 to +5 A)

ANI1: V-phase current value (–5 to +5 A)

(f) Speed specification volume value input

ANI2 is used to input a value from 0 to 1,023.

3.3 Software Processing Structure

The software processing structure is shown below.

Figure 3-2. Main Processing Structure

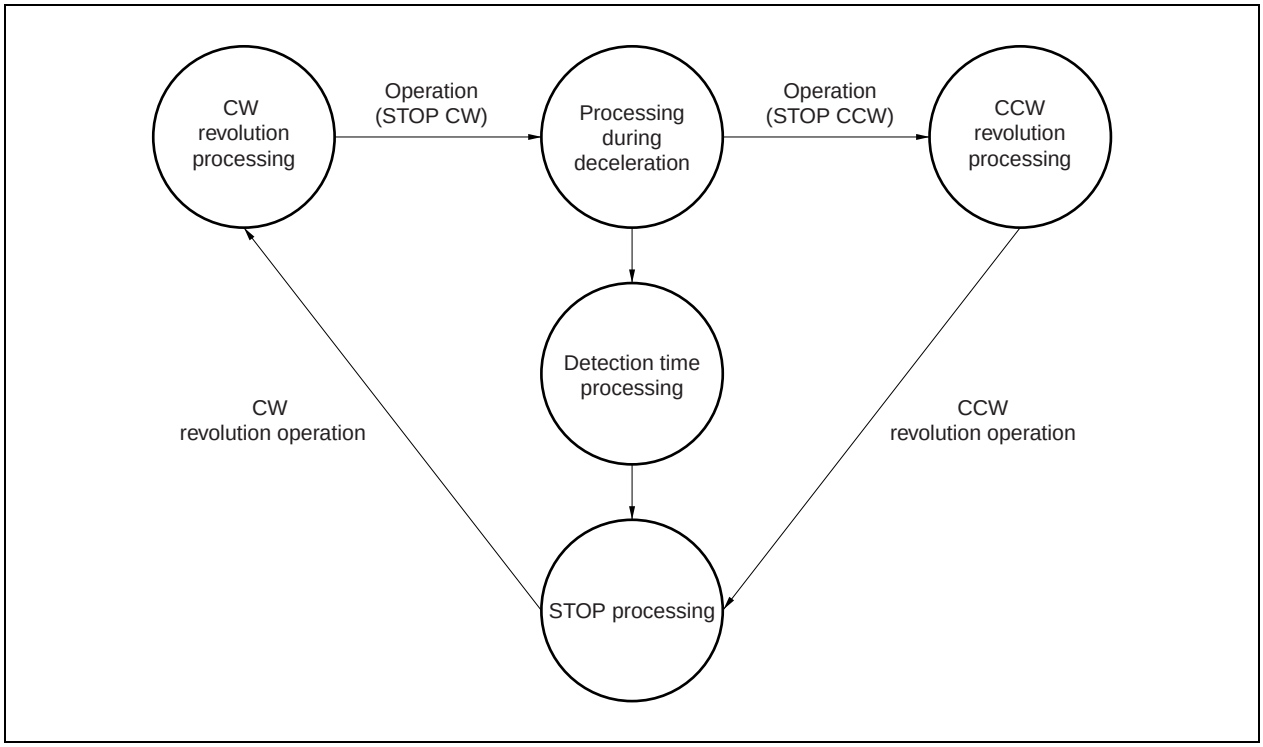
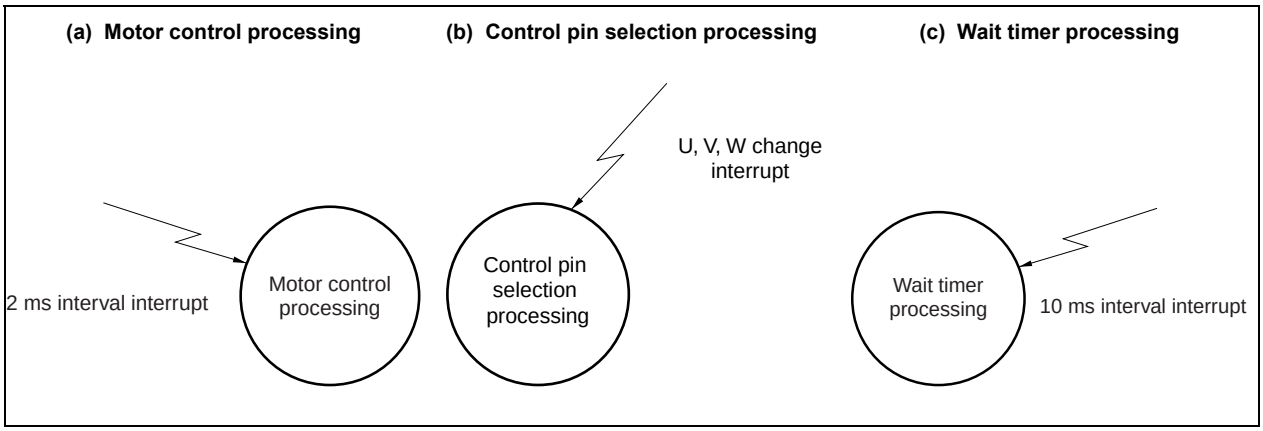


Figure 3-3. Interrupt Processing Structure



The status of the operation mode switch is monitored by the main processing, and processing is transferred to CW, CCW, and stop status. The motor is controlled in the specified status by using a 2 ms interval interrupt.

There are the following three motor control statuses.

- Stop status
  The motor is not controlled.
- Initial operation status
  Estimated revolution control is performed up to the speed at which the zero-cross point of electromotive force can be detected.
- Speed control status
  Feedback revolution control is performed so that the indication speed is attained.

3.4 Flowchart

3.4.1 Main processing

Figure 3-4 shows the flowchart of the main processing.

Figure 3-4. Main Processing

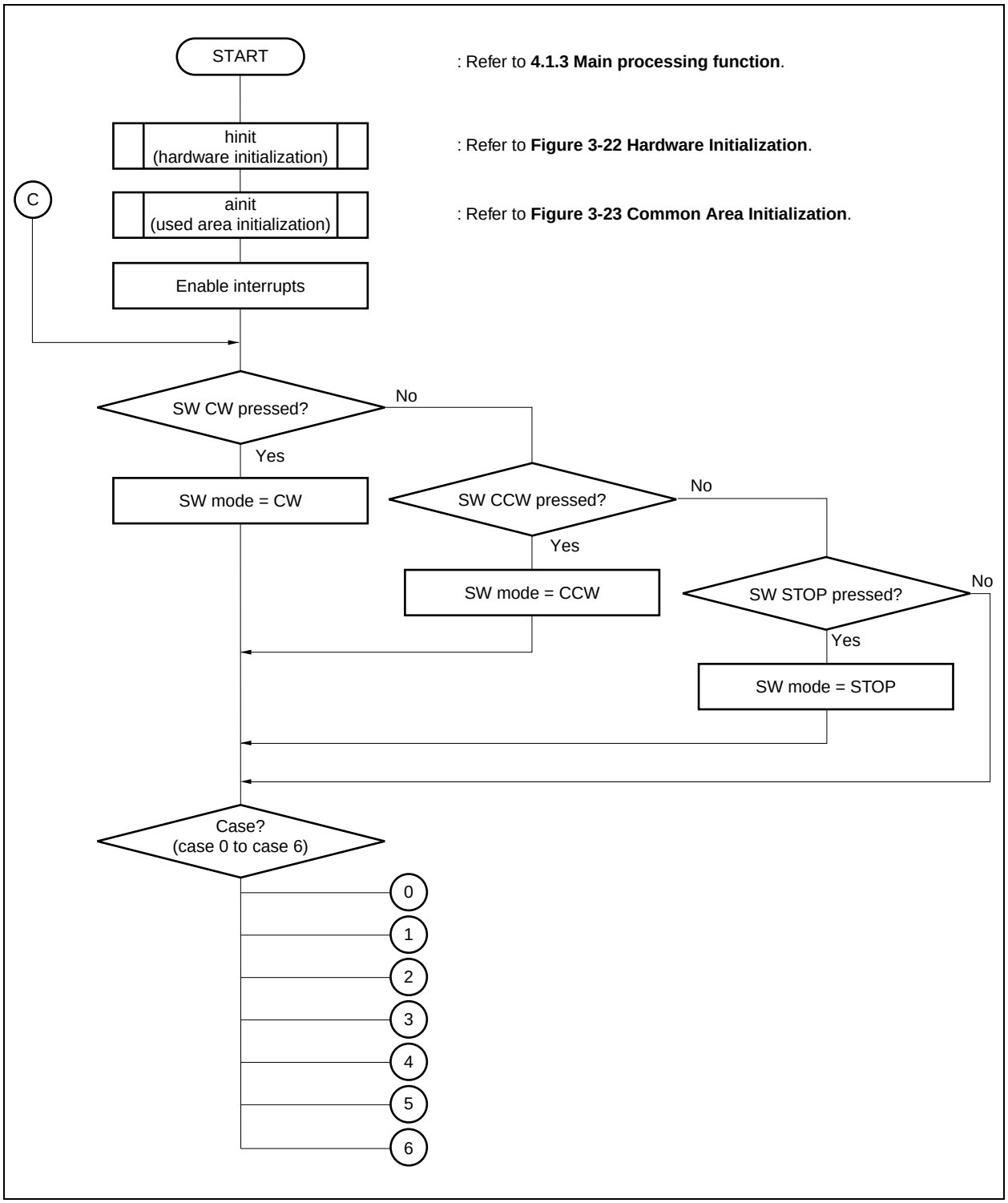


Figure 3-5. Case 0 (Processing During Stoppage)

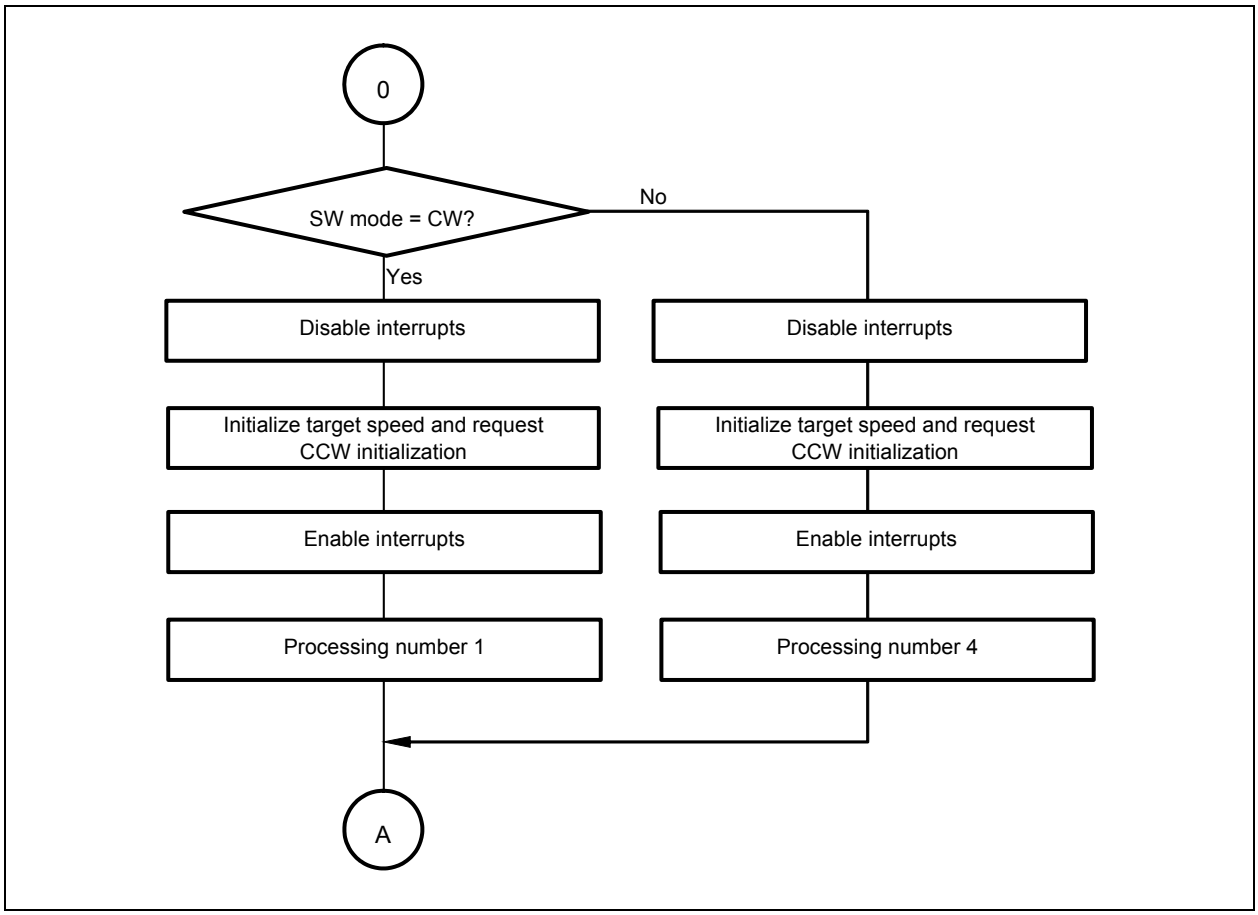


Figure 3-6. Case 1 (CW Acceleration Processing)

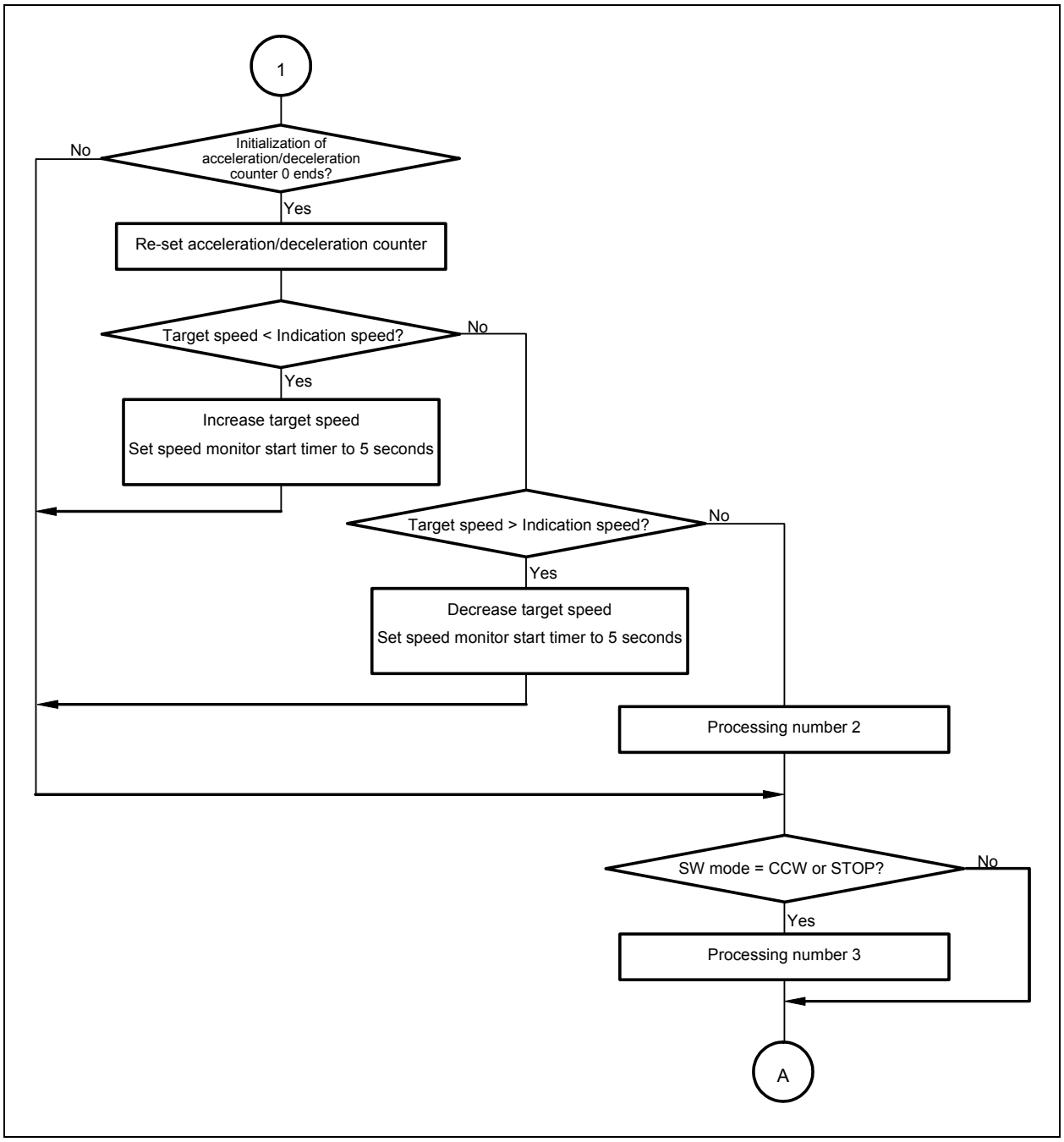


Figure 3-7. Case 2 (CW Constant-Speed Processing)

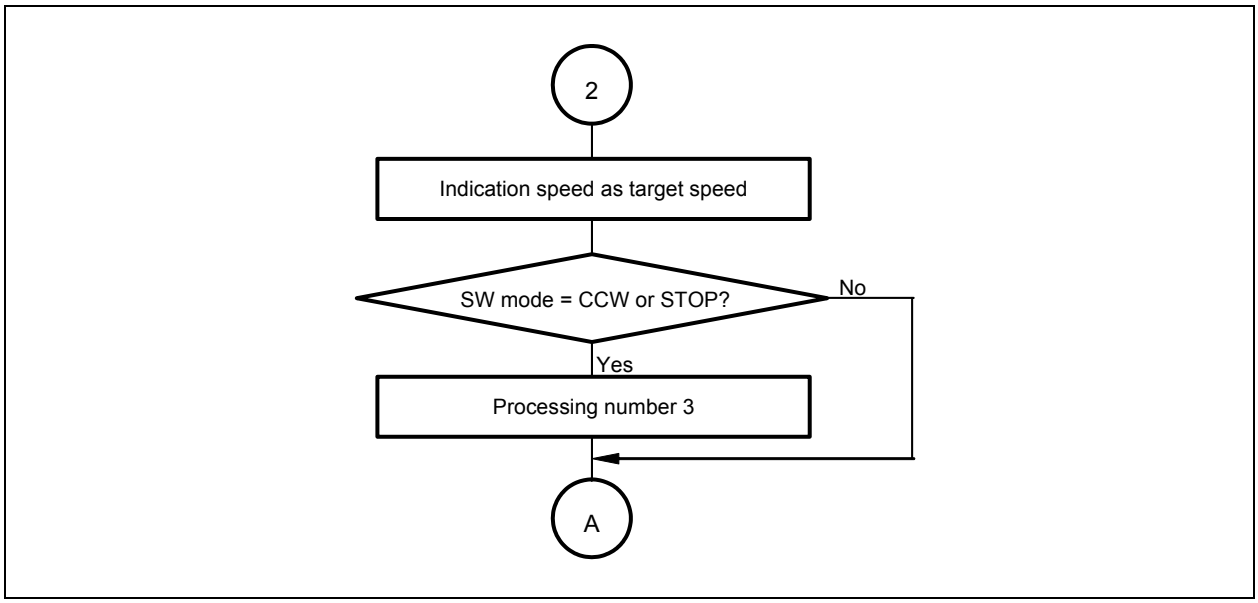


Figure 3-8. Case 3 (CW Stop Processing)

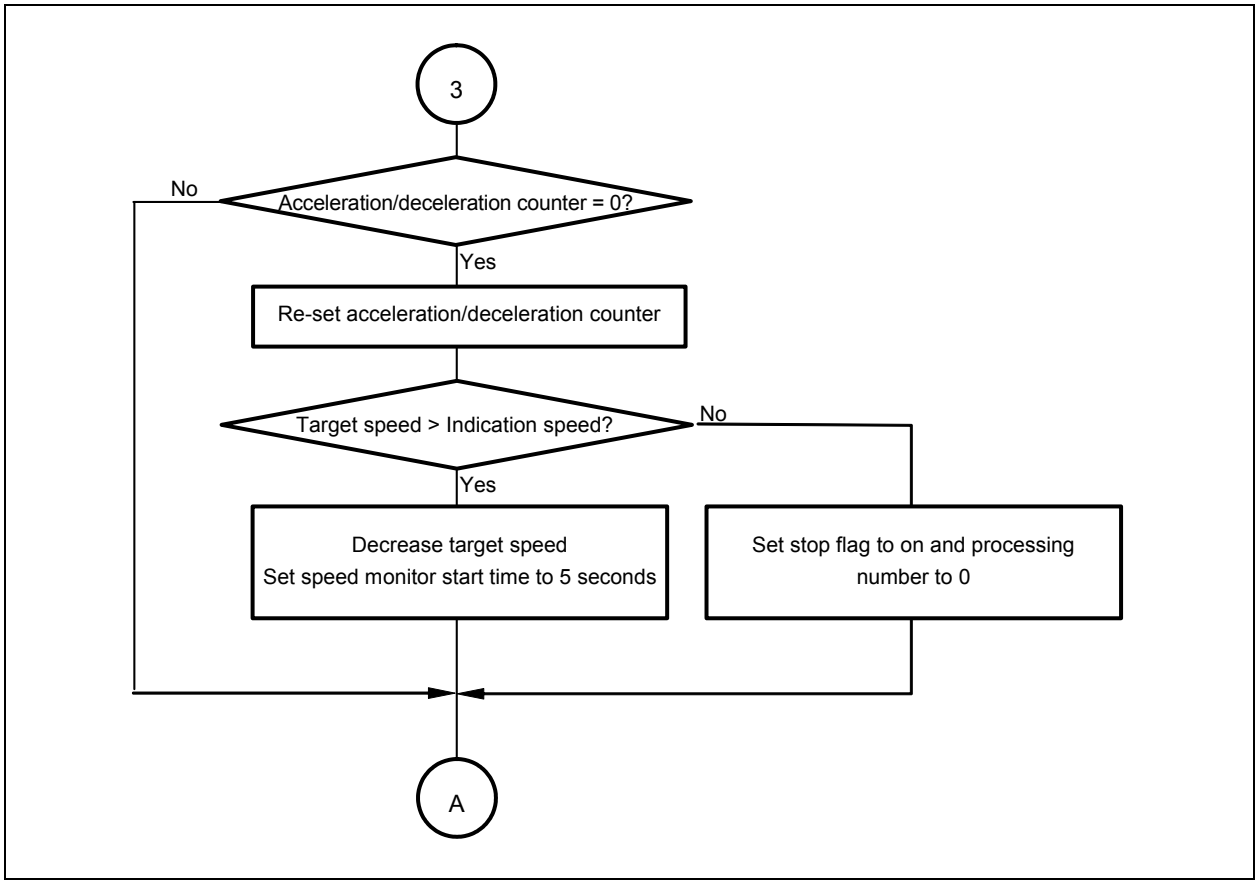


Figure 3-9. Case 4 (CCW Acceleration Processing)

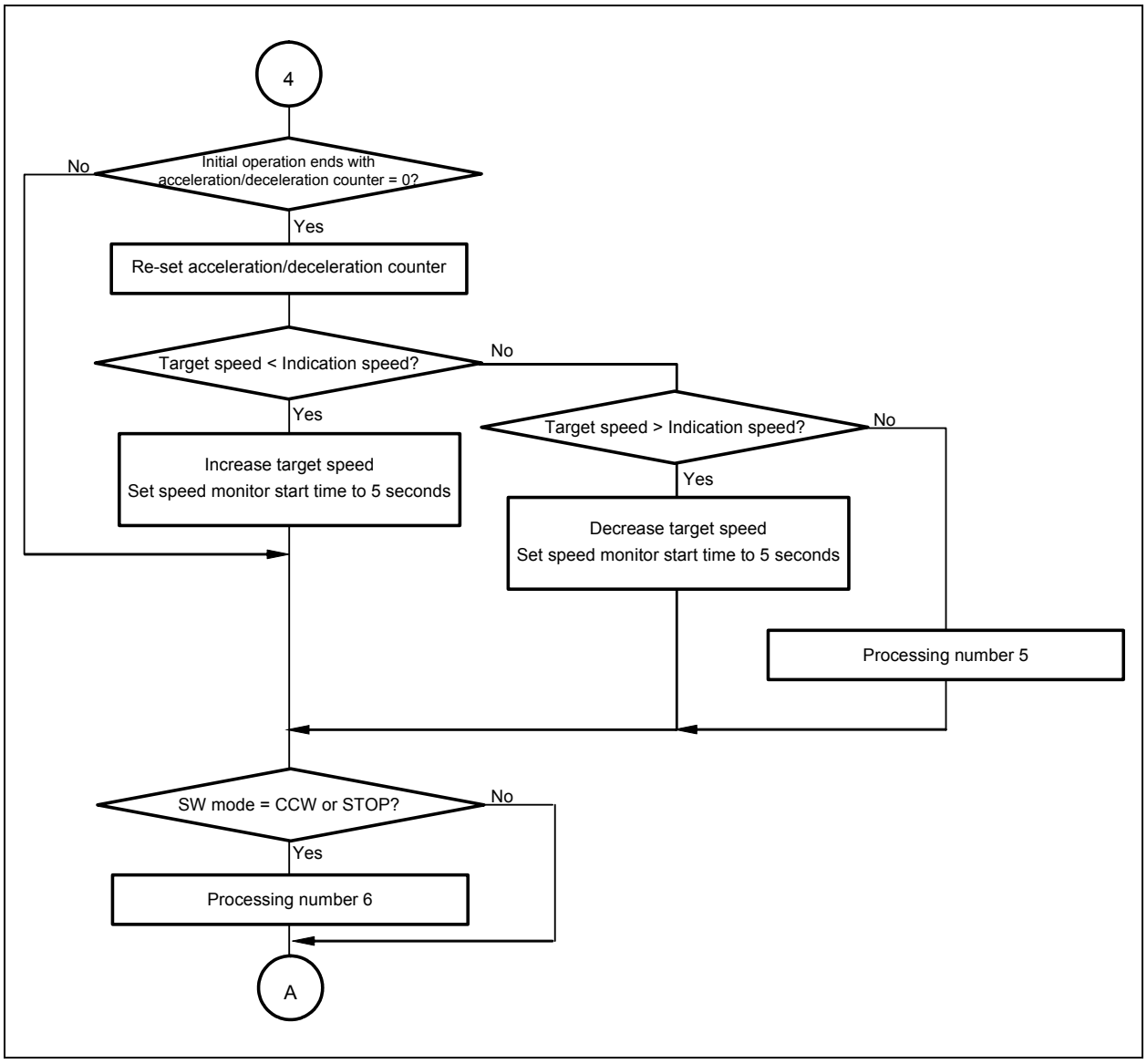


Figure 3-10. Case 5 (CCW Constant-Speed Processing)

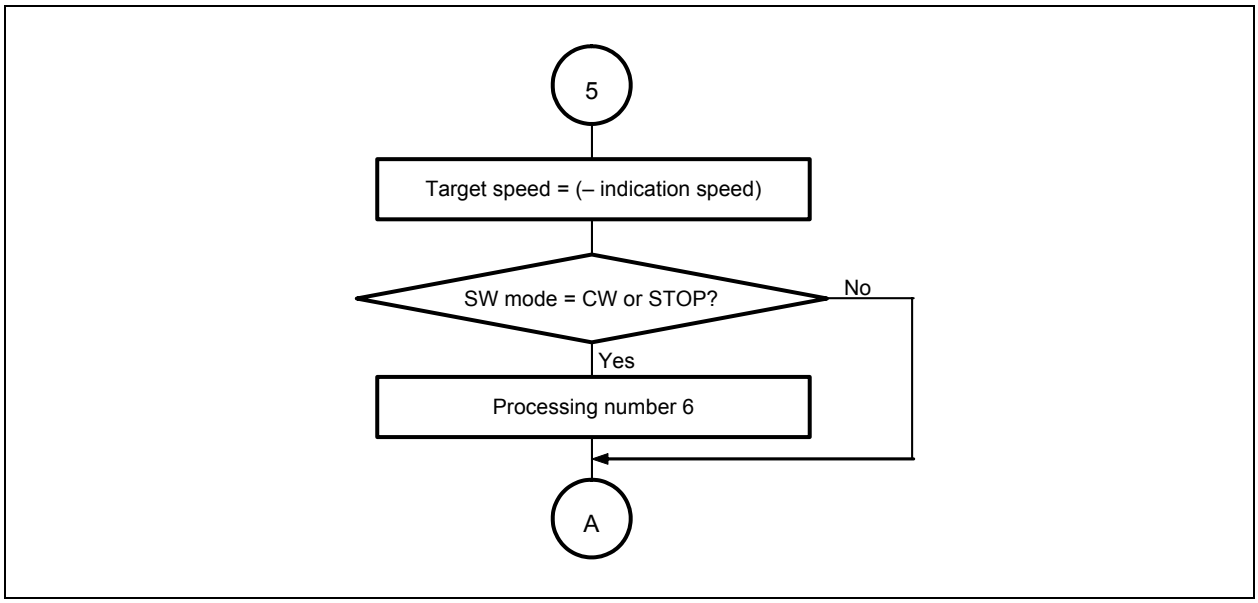


Figure 3-11. Case 6 (CCW Stop Processing)

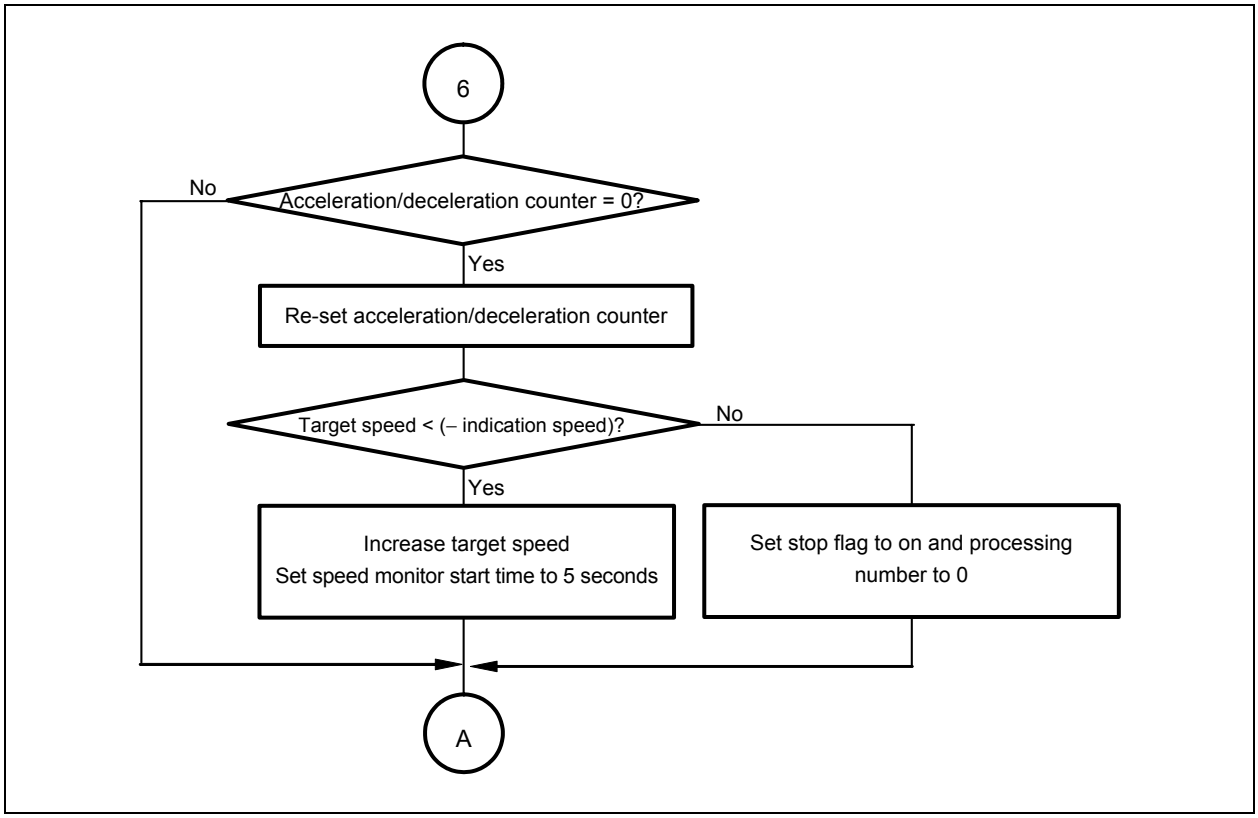


Figure 3-12. Detect Wait (1/2)

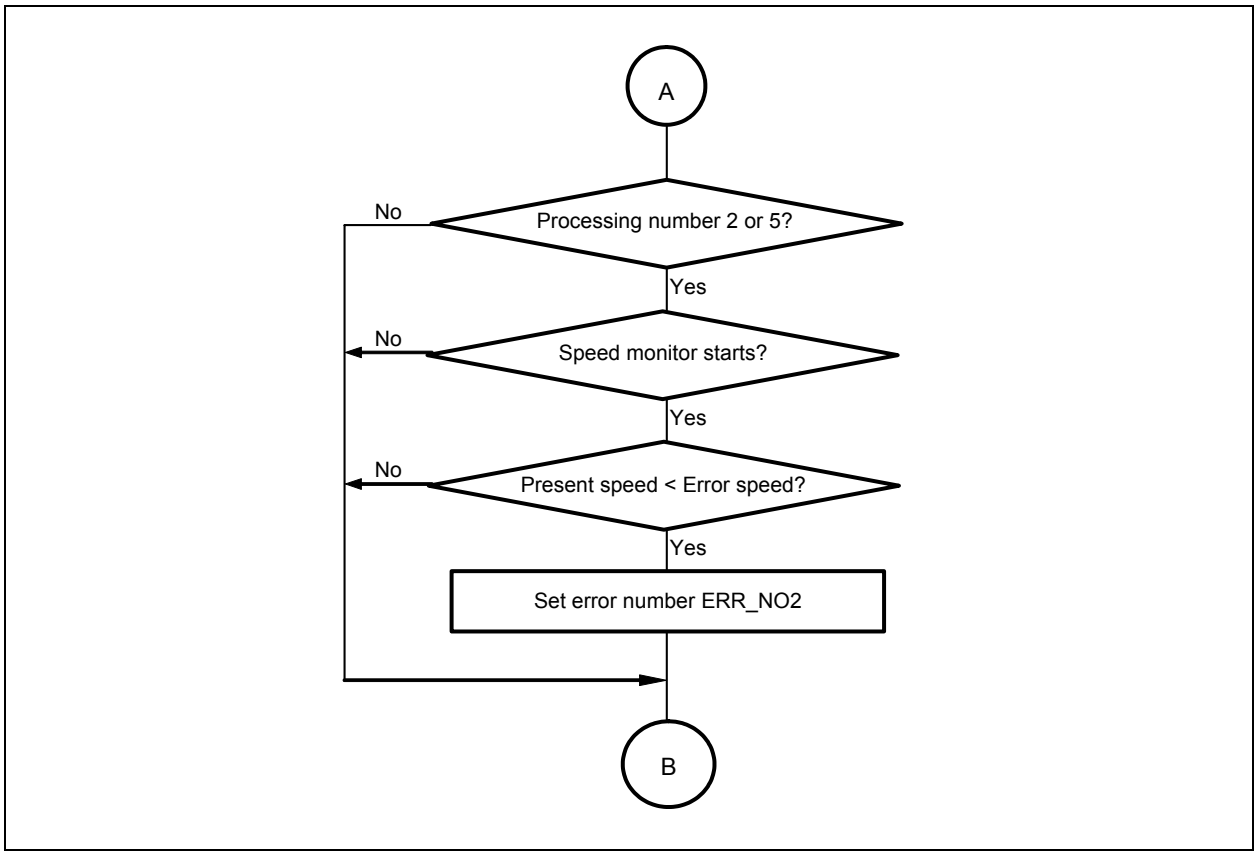
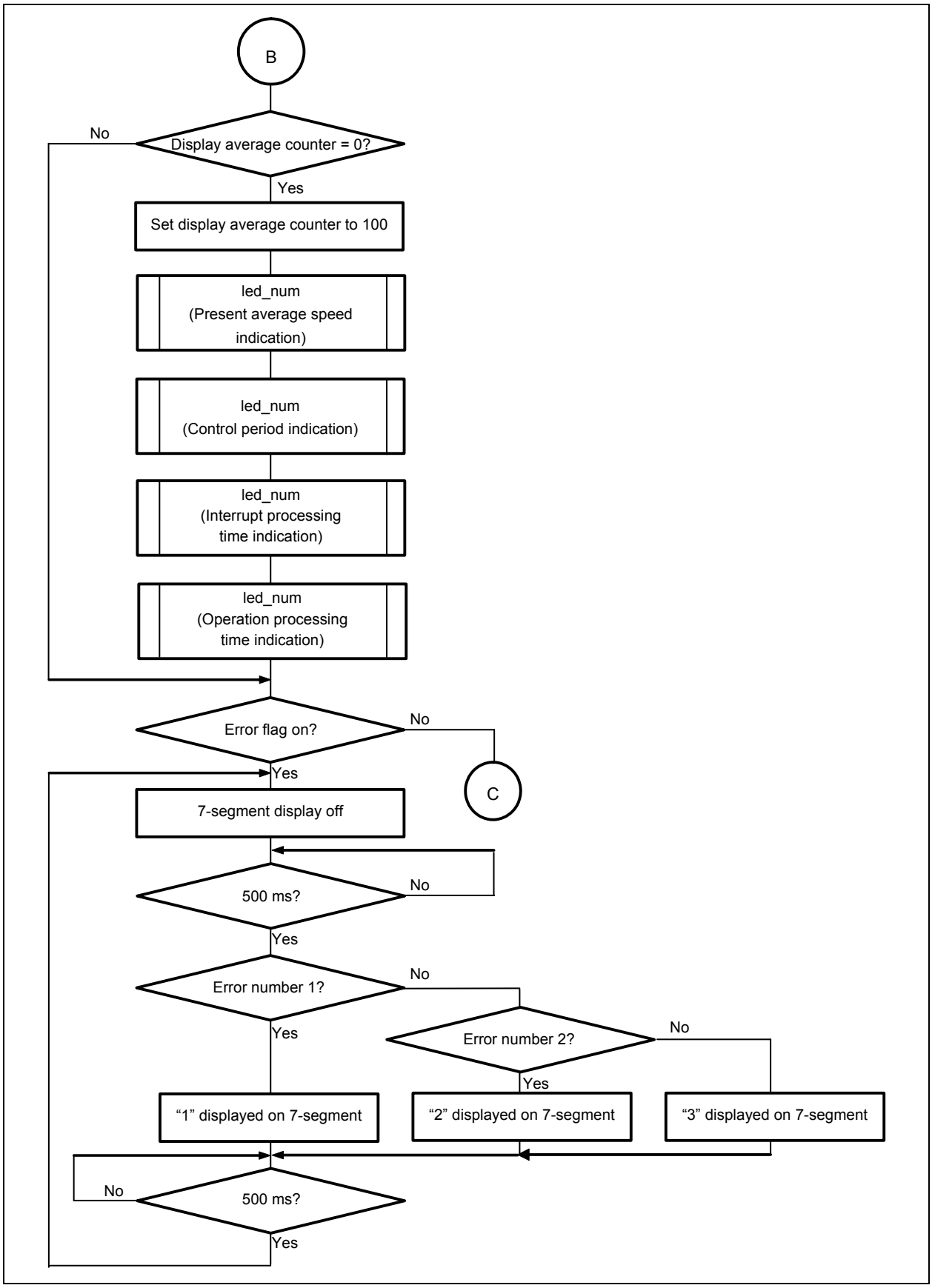


Figure 3-12. Detect Wait (2/2)



3.4.2 Motor control processing

Figure 3-13. Control Interrupt Processing (1/4)

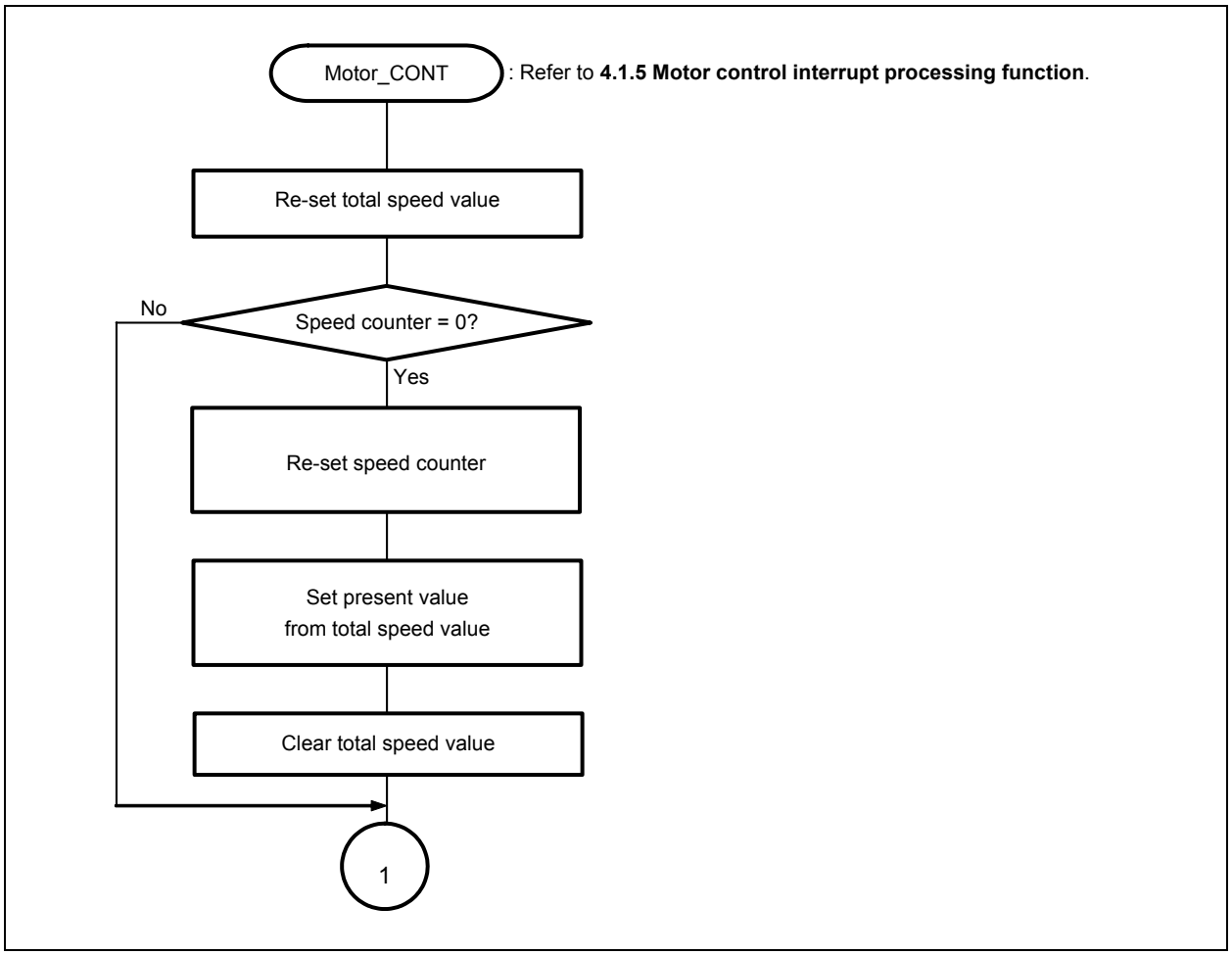


Figure 3-13. Control Interrupt Processing (2/4)

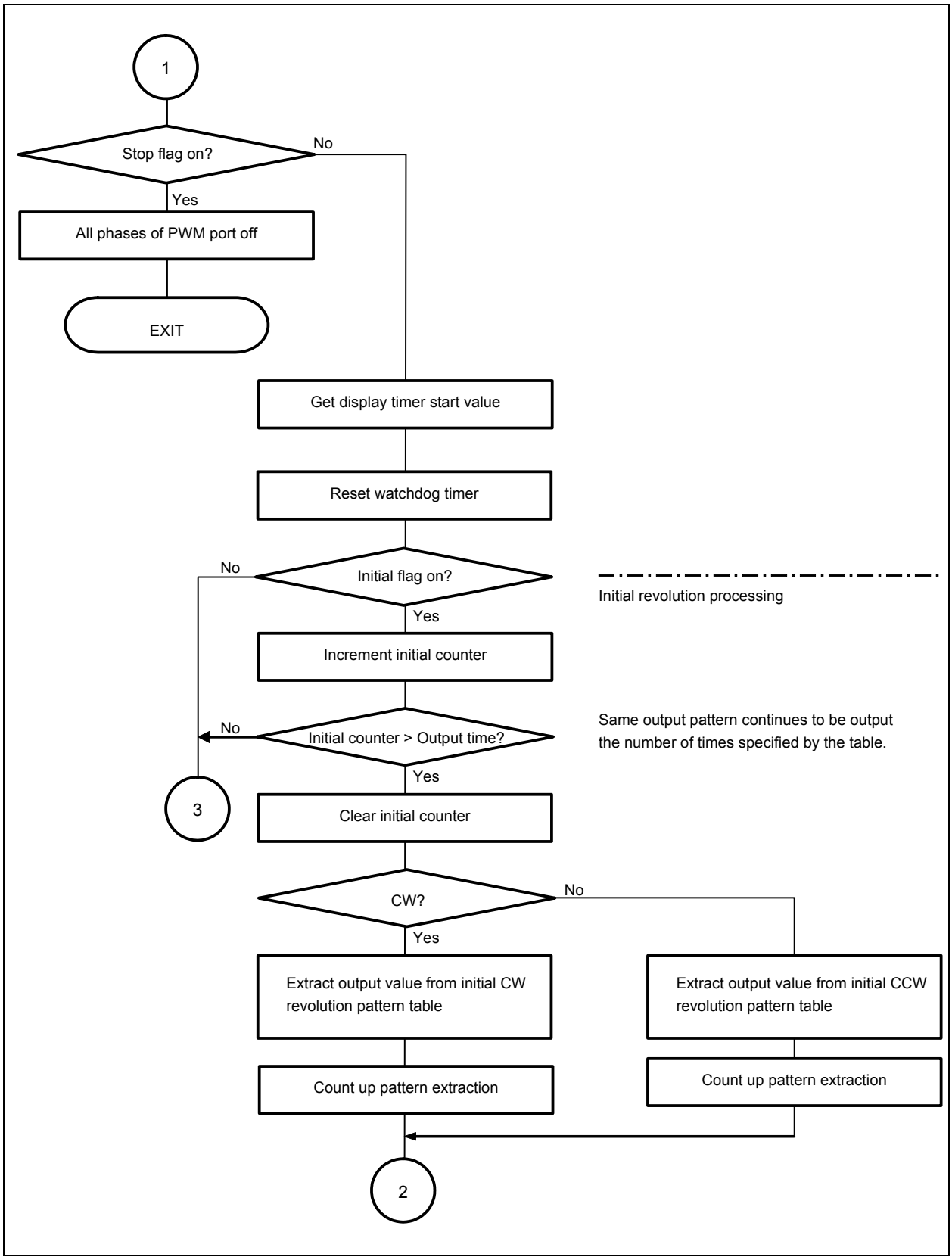


Figure 3-13. Control Interrupt Processing (3/4)

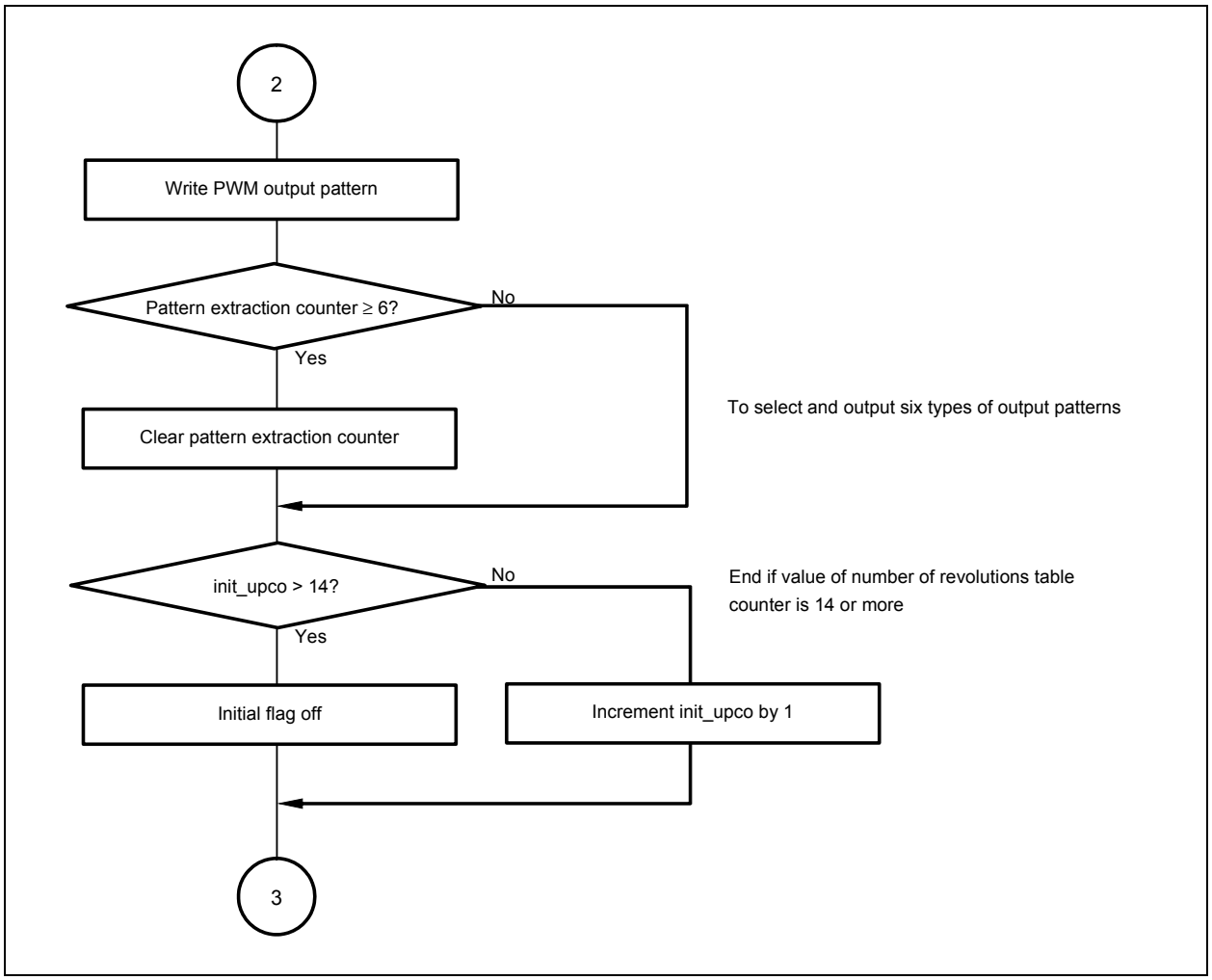
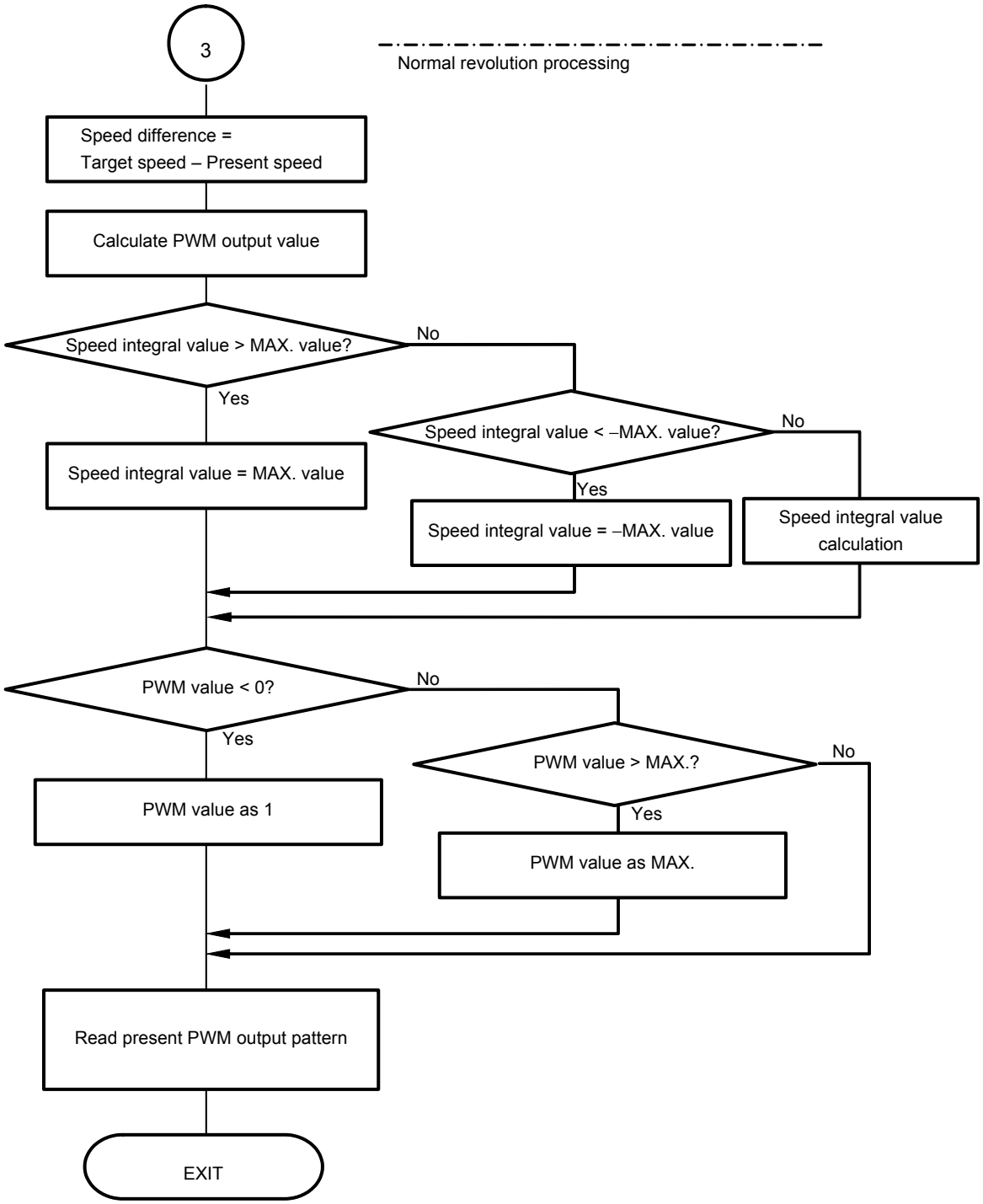


Figure 3-13. Control Interrupt Processing (4/4)



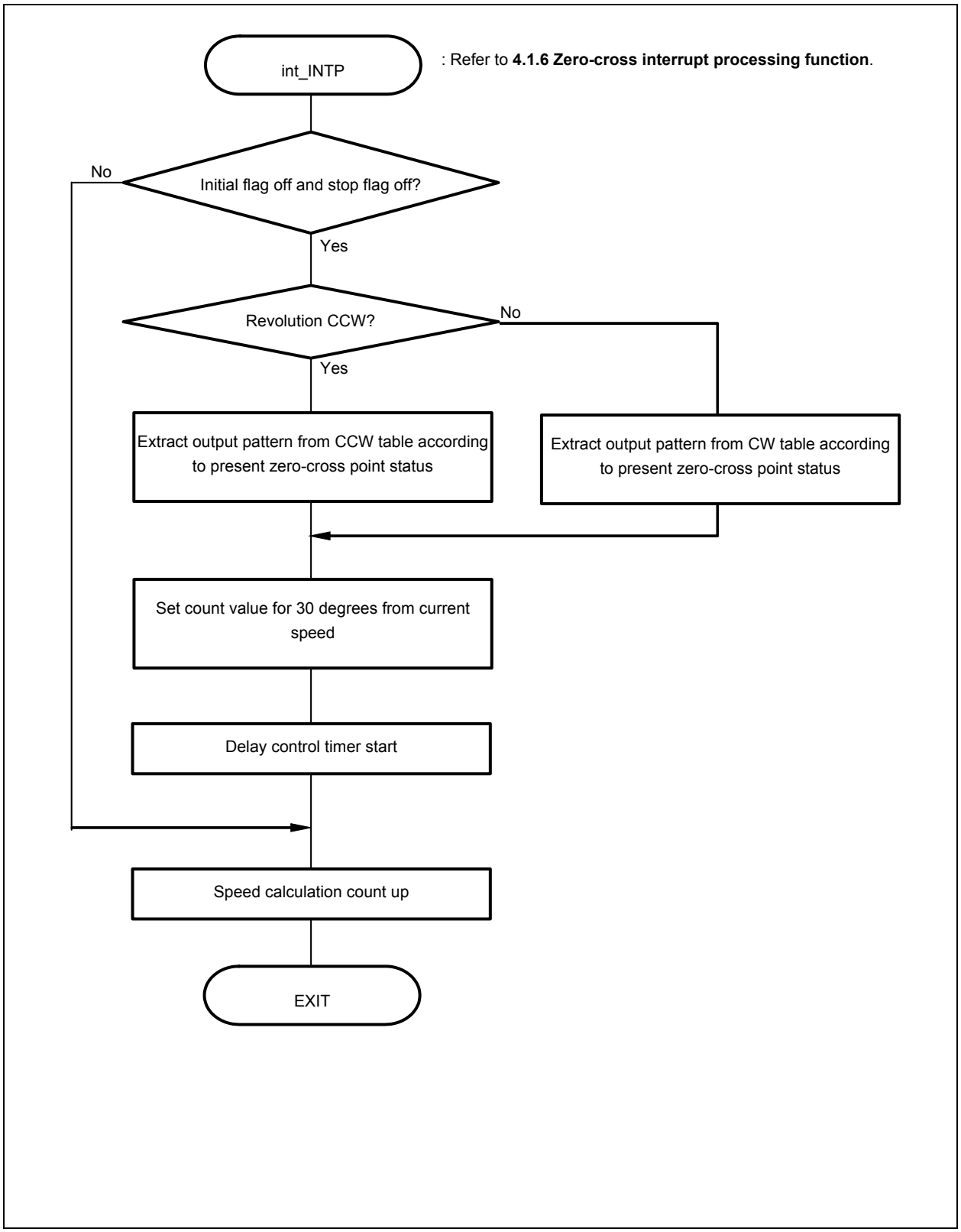
Remarks 1. PWM = (Present speed × ksp) + (Speed difference × ksp/10) + Speed integral value + Speed offset value

2. Speed integral value = Speed integral value + (Speed difference × ksi)

3. ksp: Speed proportion gain
ksi: Speed integral gain

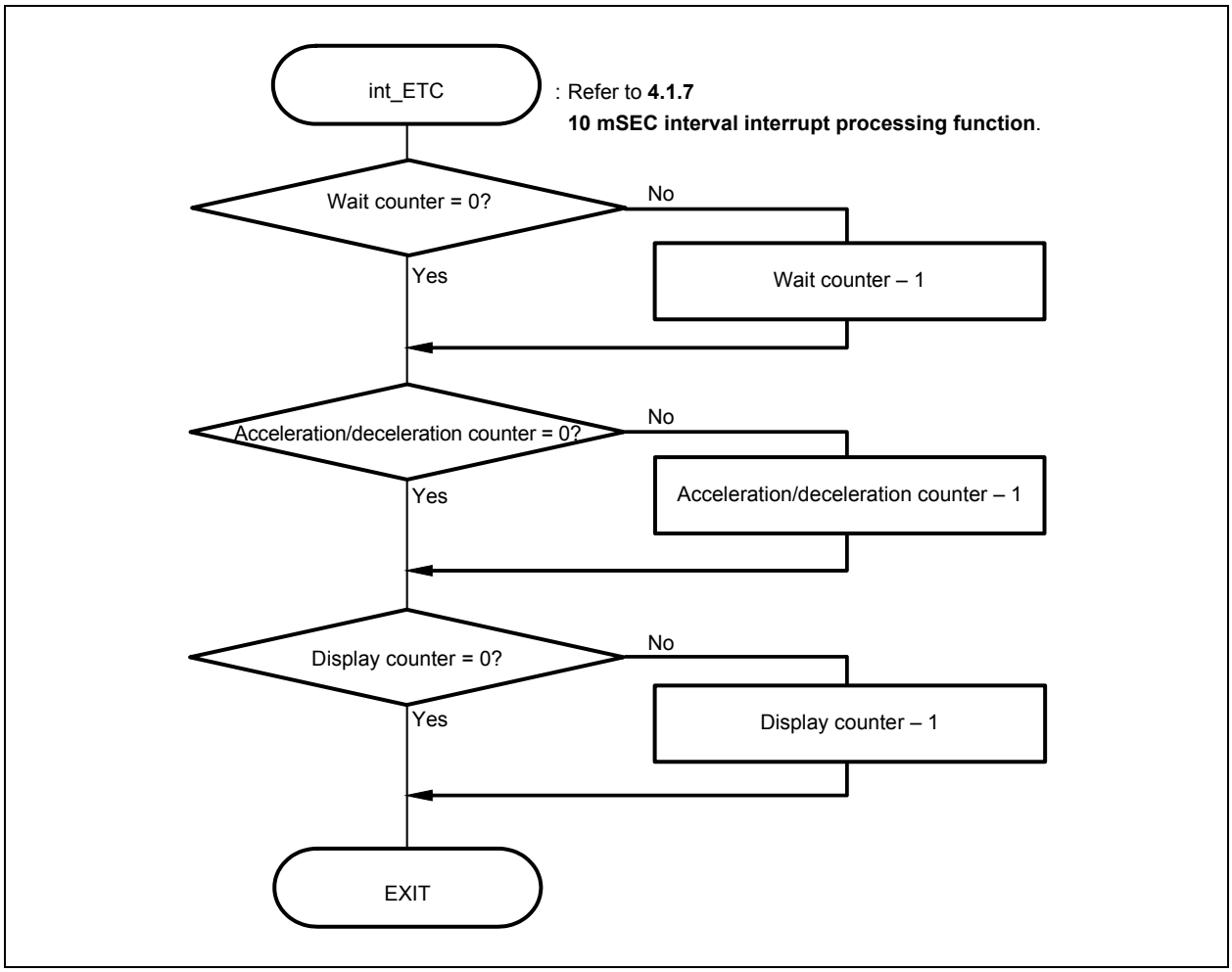
3.4.3 U, V, W zero-cross point interrupt processing

Figure 3-14. U, V, W Zero-Cross Point Interrupt Processing



3.4.4 10 mSEC interval interrupt processing

Figure 3-15. 10 mSEC Interval Interrupt Processing



3.4.5 A/D converter interrupt processing

Figure 3-16. A/D Converter Interrupt Processing

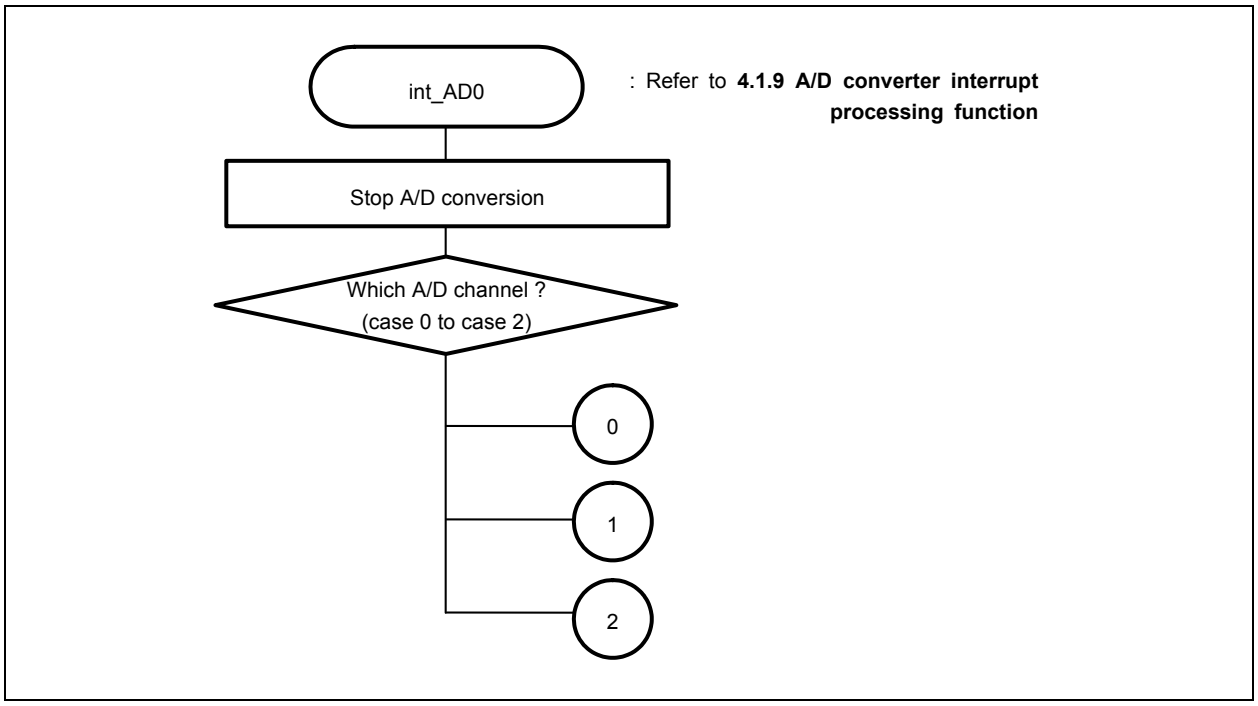


Figure 3-17. case 0 (A/D Converter Channel 1 Interrupt Processing)

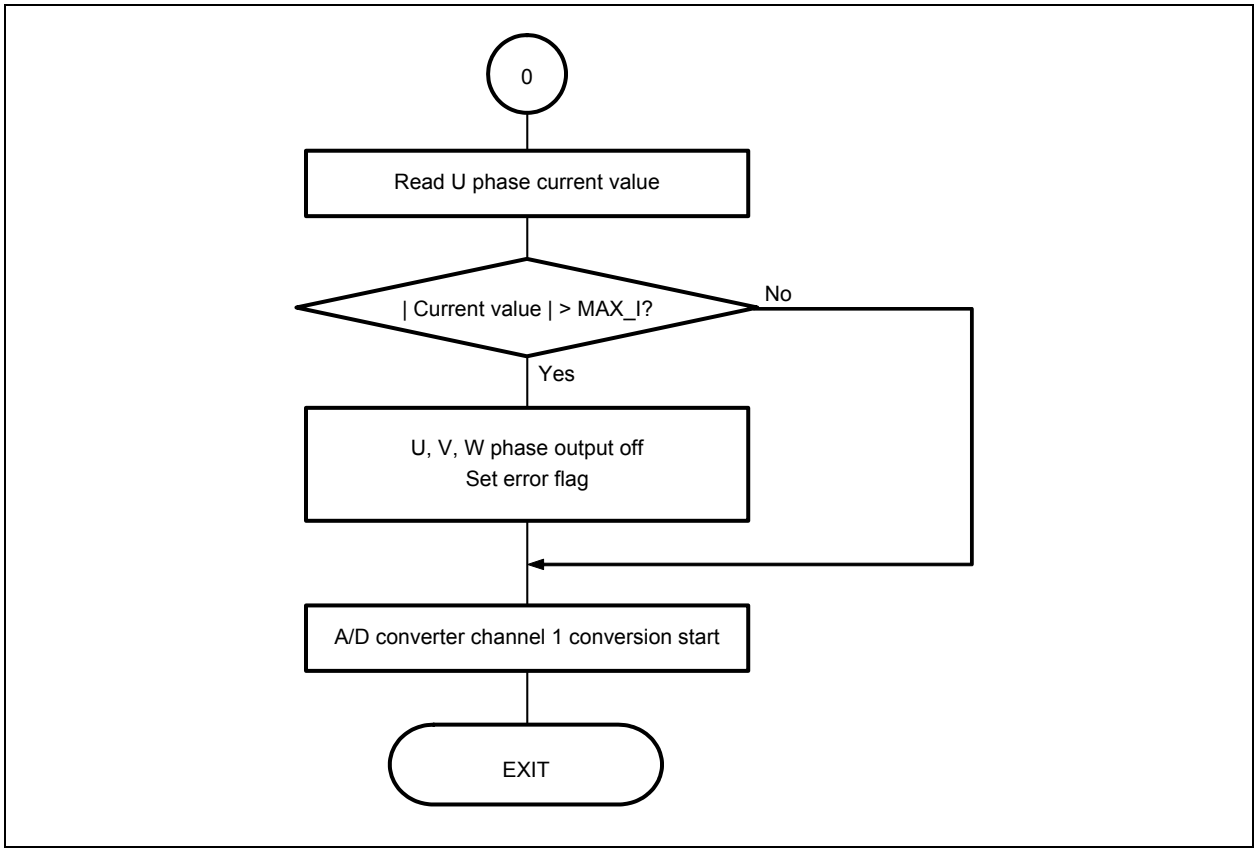


Figure 3-18. case 1 (A/D Converter Channel 2 Interrupt Processing)

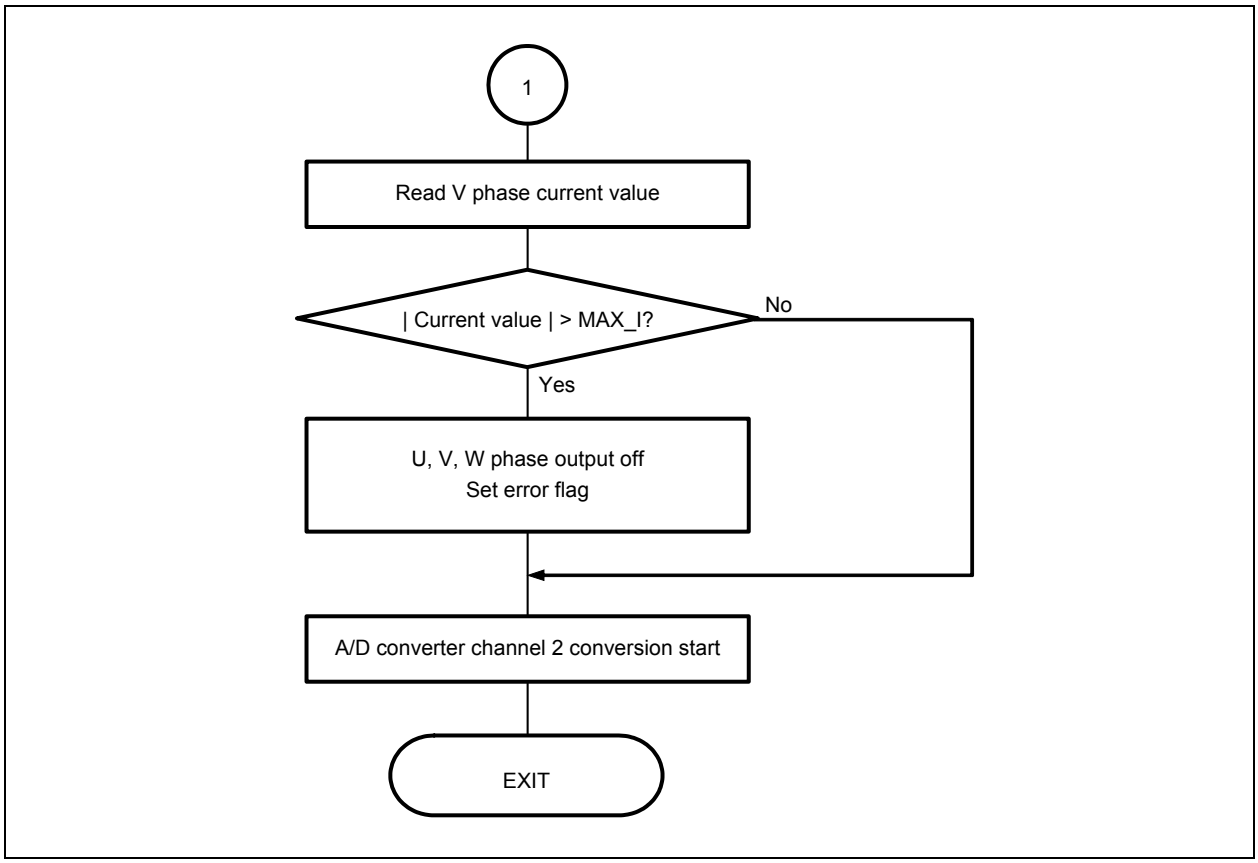
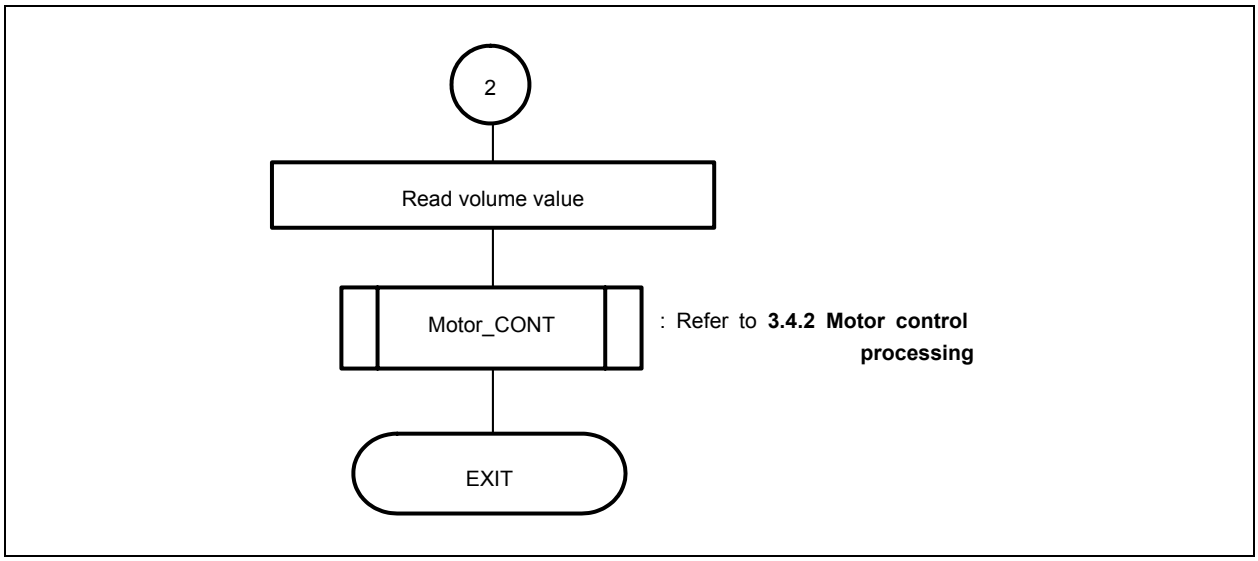
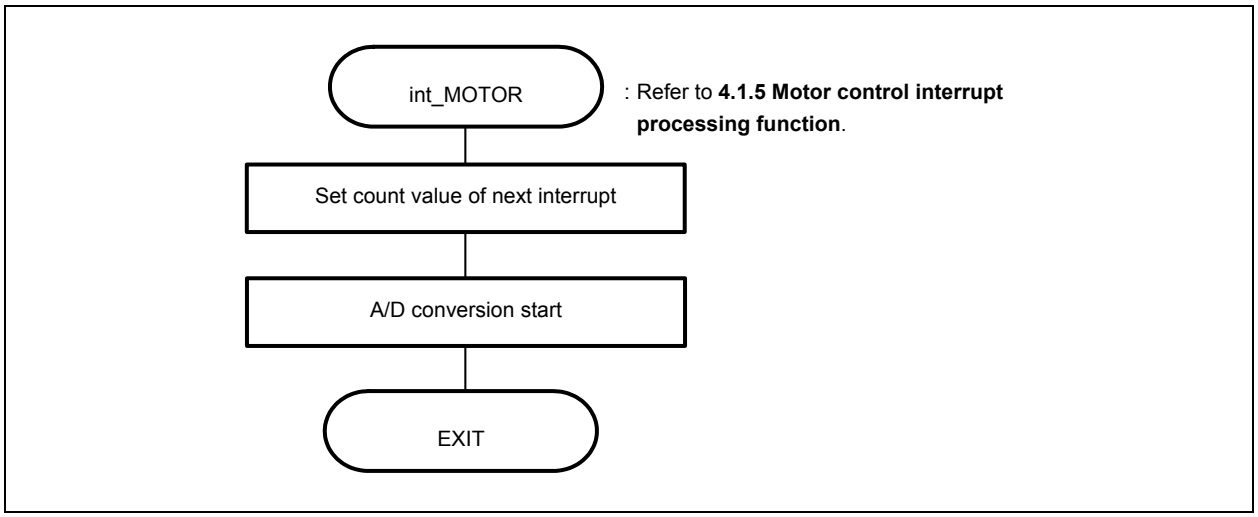


Figure 3-19. case 2 (To Motor Control Processing)



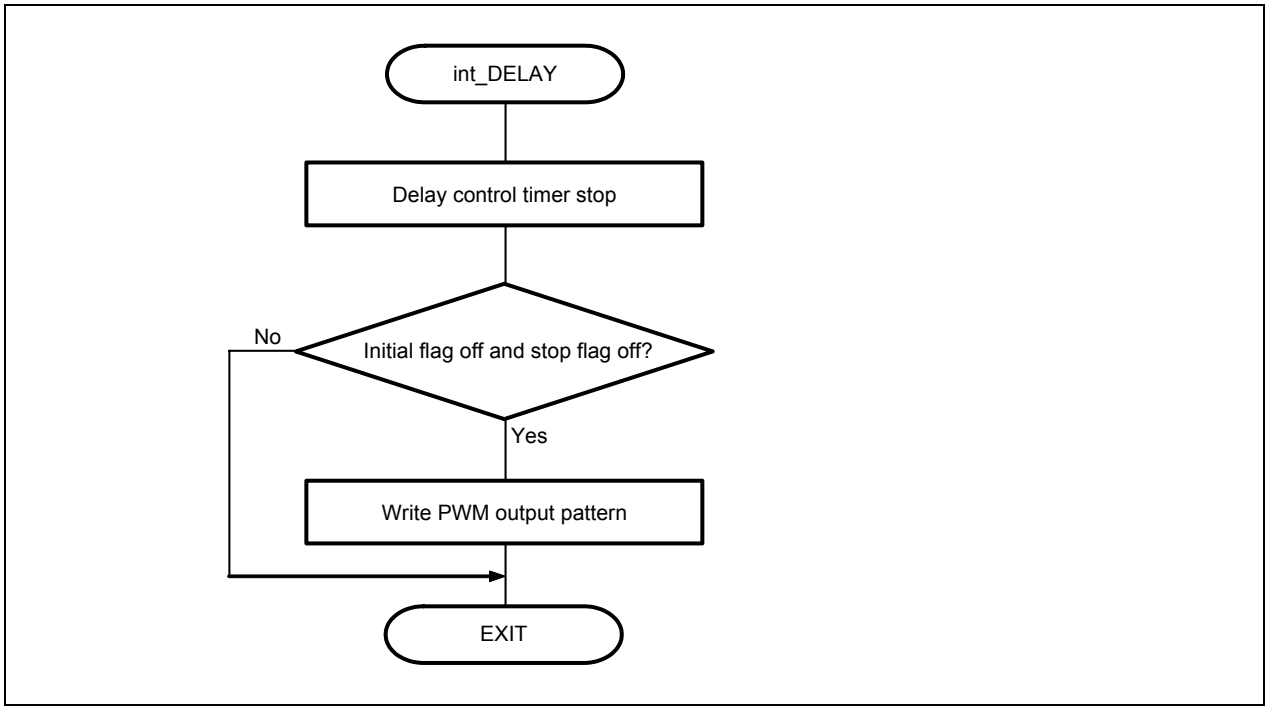
3.4.6 Motor control timer interrupt processing

Figure 3-20. Motor control timer Interrupt Processing



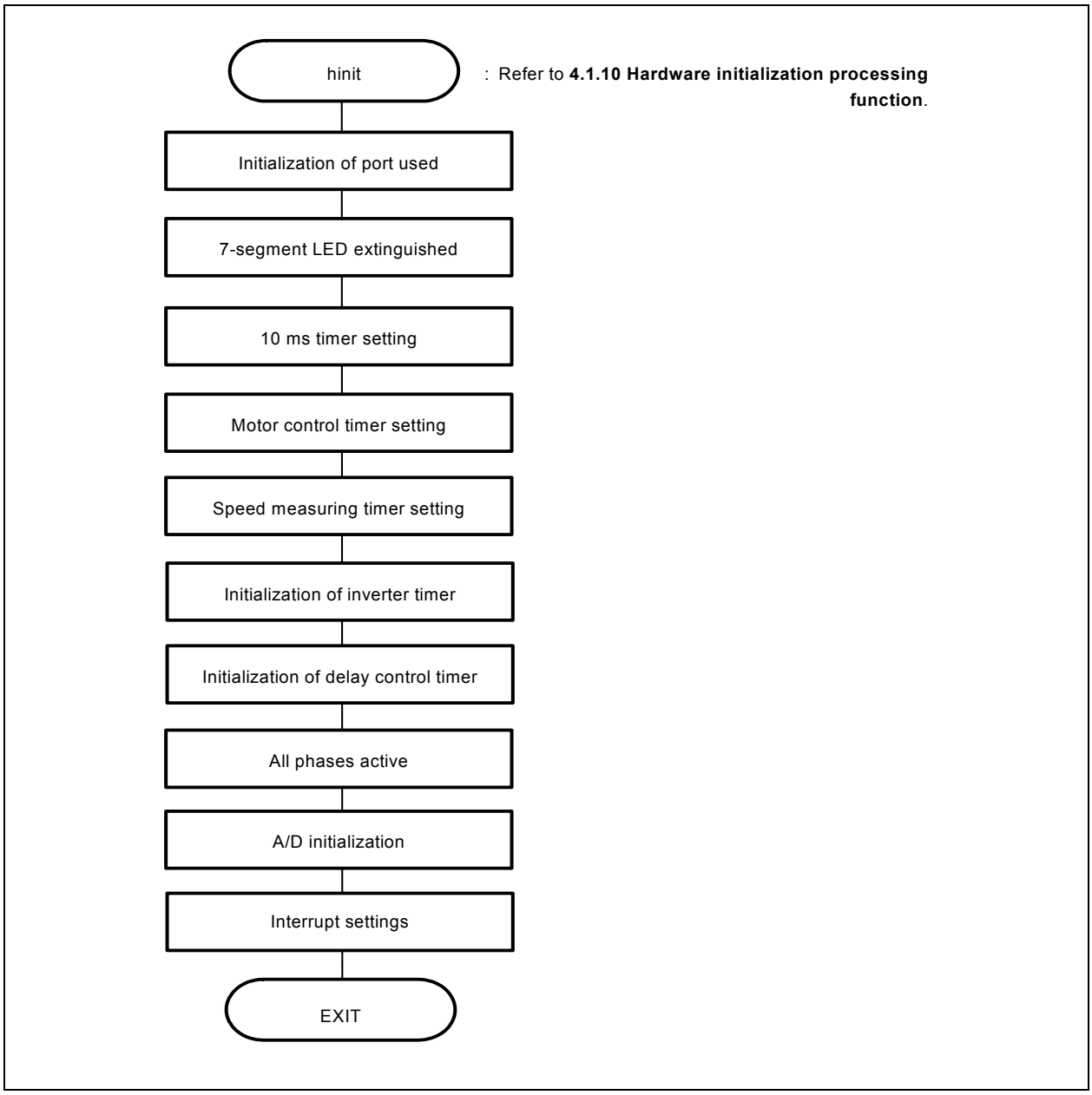
3.4.7 Delay control timer interrupt processing

Figure 3-21. Delay control timer Interrupt Processing



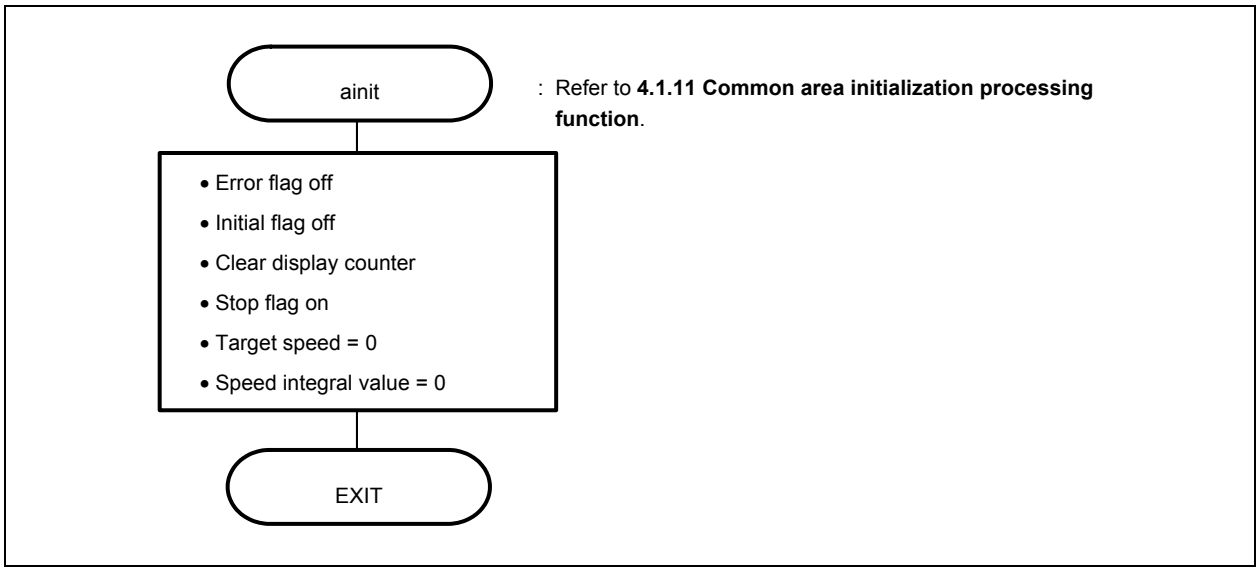
3.4.8 Hardware initialization

Figure 3-22. Hardware Initialization



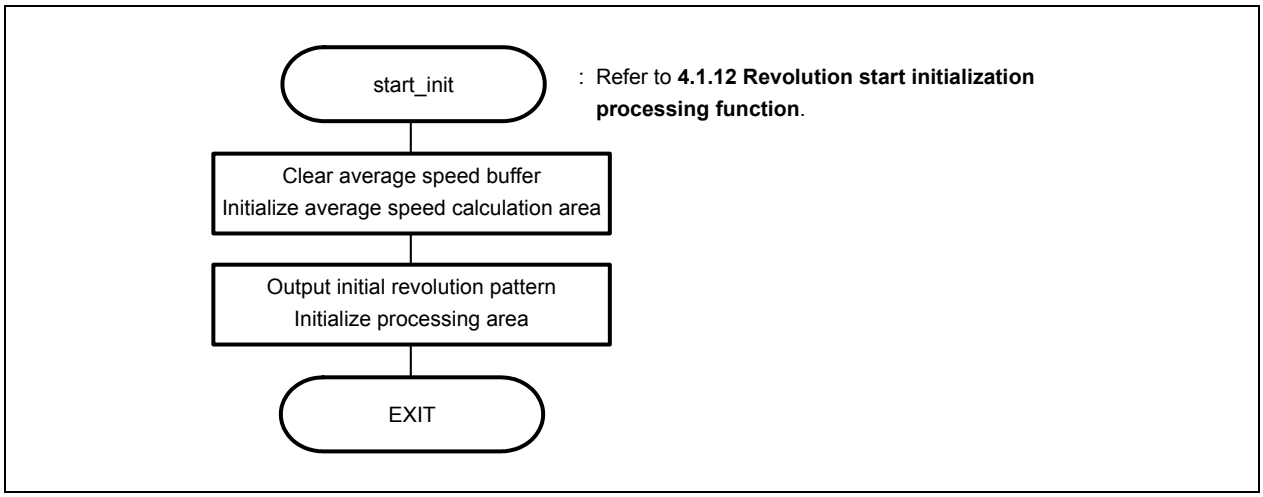
3.4.9 Common area initialization

Figure 3-23. Common Area Initialization



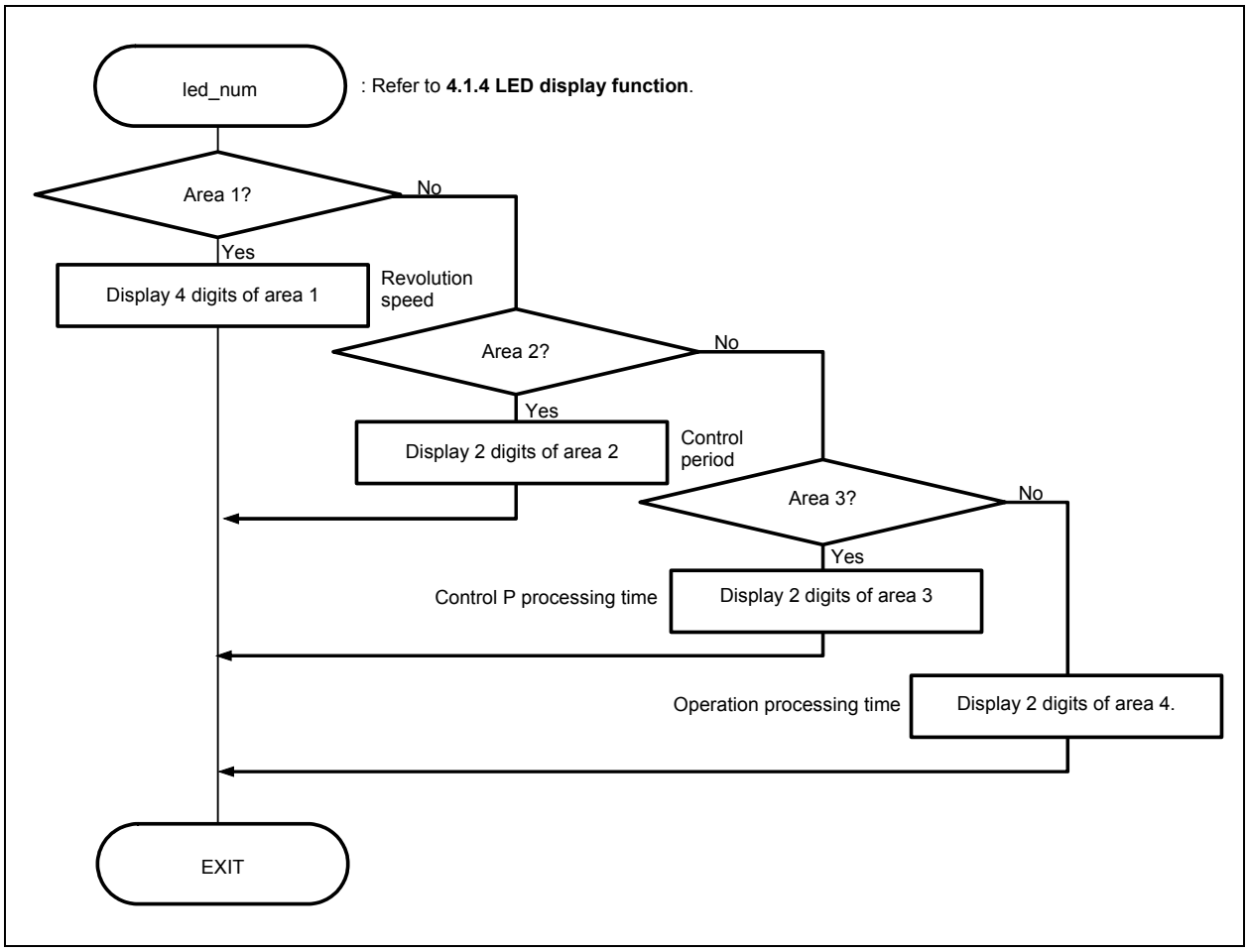
3.4.10 Revolution start initialization

Figure 3-24. Revolution Start Initialization



3.4.11 LED display

Figure 3-25. LED Display



3.5 Common Areas

The following table shows the major common areas used by the reference system.

Table 3-2. Common Area List

Symbol	Type	Usage	Set Value
error_flag	unsigned char	Error flag	0: No error ERR_NO1: Overcurrent ERR_NO3: Speed difference error
init_flag	unsigned char	Indicates initial revolution	ON: Initial revolution in progress OFF: Stop or normal revolution in progress
cont_time	unsigned short	Interrupt processing time	1 μs units
cont_time1	unsigned short	Operation time	1 μs units
disp_co	unsigned short	Average speed counter for display	
volume	unsigned short	Speed volume value	
timer_count	unsigned short	Time wait counter	10 ms units
accel_count	unsigned short	Acceleration/deceleration operation time counter	10 ms units
stop_flag	unsigned short	Stop flag	ON: Stopped OFF: Revolving
sum_speed	unsigned int	Total value area for average speed calculation	0, 1, …
speed_co	unsigned int	Counter for average speed calculation	0, 1, …
now_speed	signed int	Present speed	rpm
object_speed	signed int	Target speed	rpm
d_speed	signed int	Display speed	rpm
iua	signed short	U-phase current	
iva	signed short	V-phase current	
o_iqai	signed int	Speed integral value	
base_position	signed int	Speed estimation reference point	
sa_time	unsigned int	For speed measurement	
init_co	unsigned short	Output selection monitor counter during initial revolution	0, 1, …
init_pat	unsigned char	Initial revolution output pattern number	0 to 5
init_upco	unsigned short	Initial revolution output table number	0 to 10
int_co	unsigned int	U, V, W interrupt counter	0, 1, …
pwm_value	signed int	PWM output value	Output bit pattern

3.6 Tables

(1) LED output pattern

Contains display pattern data 0 to 9.

```
unsigned short led_pat[10] = { 0xfc. 0x60, ~ };
```

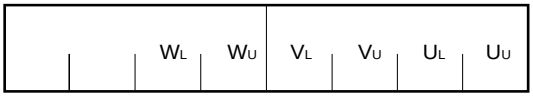
(2) Initial CW output pattern

Contains an output pattern for CW initial operation.

```
unsigned short cw_data[6][2] = { { 0x09, 0x00 }, { 0x21, 0x00 }, ~ };
```
Note

Note The underlined values differ depending on the target microcontroller.

Figure 3-26. Bit Assignment



(3) Initial CCW output pattern

Contains an output pattern for CCW initial operation.

```
unsigned short ccw_data[6][2] = { { 0x18, 0x00 }, { 0x12, 0x00 }, ~ };
```
Note

Note The underlined values differ depending on the target microcontroller.

(4) Initial revolution pattern output time

The initial pattern is output with the revolution speed increased each time the interrupt of this table occurs.

```
unsigned short up_data[ ] = { 255, 242, ~ };
```

(5) Normal CW revolution output pattern

Contains an output pattern in accordance with the status of the zero-cross point during normal CW revolution.

```
unsigned char run_cw_data[8][2] = { { 0x00, 0x00 }, { 0x21, 0x00 }, ~ };
```
Note

Note The underlined values differ depending on the target microcontroller.

(6) Normal CCW revolution output pattern

Contains an output pattern in accordance with the status of the zero-cross point during normal CCW revolution.

```
unsigned char run_ccw_data[8][2] = { { 0x00, 0x00 }, { 0x09, 0x00 }, ~ };
```
Note

Note The underlined values differ depending on the target microcontroller.

3.7 Constant Definitions

The following table shows the major constants used by the reference system.

Symbol	Usage	Value
KSP	Speed proportion constant	1100L
KSI	Speed integral constant	50L
P	Number of motor poles	2
KSPGETA	Speed proportion constant jack-up constant	10
KSIGETA	Speed integral constant jack-up constant	14
SGETA	sin jack-up constant	14
PWM_TS	PWM cycle	100 μs
PWM_DATA	PWM set value	PWM_TS/0.05
SPEED_MAX	Maximum speed	3000 rpm
SPEED_MINI	Minimum speed	800 rpm
SPEED_INIT	Initial revolution speed	700 rpm
SA_SPEED_MAX	Maximum speed difference	800 rpm
IQMAX	Maximum speed integral value	200000
MAX_I	Maximum current value	800
TS	Motor control period	2000 μs
ACCEL_TIME	Acceleration/deceleration time constant (unit: 10 ms)	1
ACCEL_DATA	Number of acceleration/deceleration incremental revolutions	40 rpm
WATCH_START	Speed monitor start time (unit: 10 ms)	500
ACCEL_VAL_1ST	Initial acceleration/deceleration time constant	50
ACCEL_VAL	Acceleration/deceleration time constant	3
ACCEL_SPD	Acceleration/deceleration constant	50
PWM_INIT	PWM initial value	PWM_DATA×3/4

CHAPTER 4 PROGRAM LIST

4.1 Program List (μPD78F0714)

4.1.1 Symbol definition

```
/************************************************************************/
/*      Common area                                                     */
/************************************************************************/
unsigned char   ram_start ;
unsigned char   error_flag ;          /* Error flag */
unsigned char   init_flag ;           /* Initial flag */
unsigned short  cont_time ;           /* Interrupt control time uSEC */
unsigned short  cont_time1 ;          /* Vector operation time uSEC */
unsigned short  disp_co ;             /* Interrupt control time display timer */
unsigned short  volume ;              /* Volume value */
unsigned short  timer_count ;         /* Time wait counter */
unsigned short  accel_count ;         /* Acceleration/deceleration operation time */
                                      /* counter */
unsigned char   stop_flag ;           /* Stop flag */
signed int      sum_speed ;
signed int      speed_co ;
signed int      now_speed ;           /* Present speed rms */
signed int      object_speed ;        /* Target speed rms */
unsigned int    d_speed ;             /* Display speed rms */
unsigned char   ram_end ;

const unsigned short led_pat[10] = { 0xfc, 0x60, 0xda, 0xf2, 0x66, 0xb6, 0xbe, 0xe0,
                                     0xfe, 0xe6 } ;
/************************************************************************/
/*      Common flags                                                    */
/************************************************************************/
extern unsigned char    ram_start ;
extern unsigned char    error_flag ;          /* Error flag */
extern unsigned char    init_flag ;           /* Initial flag */
extern unsigned short   cont_time ;           /* Interrupt control time uSEC */
extern unsigned short   cont_time1 ;          /* Vector operation time uSEC */
extern unsigned short   disp_co ;             /* Interrupt control time display timer */
extern unsigned short   volume ;              /* Volume value */
extern unsigned short   timer_count ;         /* Time wait counter */
extern unsigned short   accel_count ;         /* Acceleration/deceleration operation */
                                              /* time counter */
extern unsigned char    stop_flag ;           /* Stop flag */
extern signed int       sum_speed ;
extern signed int       speed_co ;
extern signed int       now_speed ;           /* Present speed rms */
extern signed int       object_speed ;        /* Target speed rms */
```

```
extern unsigned int     d_speed ;            /* Display speed rms */
extern unsigned char    ram_end ;

extern const unsigned short led_pat[] ;
/*************************************************************************/
/*     Motor common definition                                           */
/*************************************************************************/
extern signed short     iua ;              /* U-phase current */
extern signed short     iva ;              /* V-phase current */
extern signed int       o_iqai ;           /* Speed integral value area */
extern signed int       base_position ;    /* Speed estimation value reference point */
extern unsigned int     sa_time ;          /* Speed measurement value */
extern signed int       o_speed ;          /* Target revolution speed zero-cross */
                                           /* pulse/100 mSEC */

extern unsigned short   init_co ;          /* Initial interrupt counter */
extern unsigned char    init_pat ;         /* Initial pattern counter */
extern unsigned short   init_upco ;        /* Initial speed-up counter */
extern unsigned int     int_co ;           /* UVW interrupt counter */
extern signed int       pwm_value ;        /* PWM output value */
extern unsigned short out_pat;             /* Output pattern */
extern unsigned short CR01_int;            /* Conversion completion time temporary */
                                           /* variable */

extern const unsigned char cw_data[][2] ;
extern const unsigned char ccw_data[][2] ;
extern const unsigned char up_data[] ;
extern const unsigned char run_cw_data[][2] ;
extern const unsigned char run_ccw_data[][2] ;
```

4.1.2 Constant definition

```
/*************************************************************************/
/*      I/O                                                              */
/*************************************************************************/
#define BASE_IO   0xc200000
#define LED11     0x03
#define LED12     0x02
#define LED13     0x01
#define LED14     0x00
#define LED21     0x05
#define LED22     0x04
#define LED31     0x07
#define LED32     0x06
#define LED41     0x09
#define LED42     0x08

#define DIPSW     0x0f
```

```
#define SW        0x0e
#define DA1       0x0a
#define DA2       0x0b
#define DA3       0x0c
#define WRESET    0x0d
#define MODE      0x10
/*****************************************************************************/
/*      Constant                                                             */
/*****************************************************************************/
#define ON        1
#define OFF       0
#define CW        1                     /* CW operation mode */
#define CCW       2                     /* CCW operation mode */
#define STOP      0                     /* Operation stop mode */
#define ERR_NO1   1                     /* Overcurrent error */
#define ERR_NO2   2                     /* Speed difference error */
/*****************************************************************************/
/*      Motor constant                                                       */
/*****************************************************************************/
/* Motor constant */
#define KSP           1100L                 /* Speed proportion constant */
#define KSI           50L                   /* Speed integral constant */
#define P             2                     /* Number of poles */

#define KSPGETA       10                    /* KP jack-up constant */
#define KSIGETA       14                    /* KSI jack-up constant */
#define SGETA         14                    /* sin jack-up constant */

#define PWM_TS        100                   /* PWM cycle (us) 10kHz */
#define CPU_CLOCK_US  0.8                   // 1/TW0C-CLOCK*1000*1000
#define PWM_DATA      (PWM_TS/CPU_CLOCK_US/2) /* PWM set value */
#define SPEED_MAX     3000                  /* Maximum speed 3000 rpm */
#define SPEED_MINI    800                   /* Minimum speed 800 rpm */
#define SPEED_INIT    700                   /* Initial revolution speed rpm */
#define SA_SPEED_MAX  800                   /* Maximum speed difference rpm */
#define IQAMAX        200000                /* Maximum speed integral value */
#define MAX_I         800                   /* Maximum current value */
#define TS            2000                  /* Motor control time interval uSEC */
#define ACCEL_TIME    1                     /* Acceleration/deceleration time */
                                            /* constant 10 mSEC */
#define ACCEL_DATA    40                    /* Number of acceleration/deceleration */
                                            /* incremental revolutions rpm */
#define WATCH_START   500                   /* Speed monitor start time 10 mSEC */
#define ACCEL_VAL_1ST 50                    /* Initial acceleration/deceleration */
                                            /* time constant */
#define ACCEL_VAL     3                     /* Acceleration/deceleration time */
                                            /* constant */
#define ACCEL_SPD     50                    /* Acceleration/deceleration constant */
```

```
#define PWM_INIT      (int)(PWM_DATA*3/4)     /* PWM initial value */

#define LOW           0
#define HIGH          1
#define ENABLE        0

#define UP_TO_CPU(x)  ((x)*5U)                // us -> TM00 value  0.2us*5 = 1us
                                              // by prescaler
#define CPU_TO_US(x)  ((x+4U)/5U)             // TM00 value -> us

#define ADM_DEF       0x05                    // STOP operation, Select mode,
                                              // conversion time 3.6us, Comparator=ON
#define ADM_START     (ADM_DEF|0x80)          // AD conversion start
#define ADM_STOP      ADM_DEF                 // AD conversion stop

/*****************************************************************************/
/*      Function constant                                                    */
/*****************************************************************************/
void           OUT_data( unsigned short reg, unsigned short data );
unsigned short IN_data( int reg );
void           led_num( int no, long data ) ;
void           hinit( void );
void           ainit( void );
void           start_init( void );
/*****************************************************************************/
/*      Motor-related common area                                            */
/*****************************************************************************/
signed short   iua ;           /* U-phase current */
signed short   iva ;           /* V-phase current */
signed int     o_iqai ;        /* Speed integral value area */
signed int     base_position ; /* Speed estimation value reference point */
unsigned int   sa_time ;       /* Speed measurement value */
signed int     o_speed ;       /* Target revolution speed zero-cross pulse */
                               /* /100 mSEC */

unsigned short init_co ;       /* Initial interrupt counter */
unsigned char  init_pat ;      /* Initial pattern counter */
unsigned short init_upco ;     /* Initial speed-up counter */
unsigned int   int_co ;        /* UVW interrupt counter */
signed int     pwm_value ;     /* PWM output value */
unsigned short out_pat ;       /* Output pattern */
unsigned short CR01_int ;      /* Temporary variable to calculate conversion */
                               /* completion time */

const unsigned char cw_data[6][2] = { {0x09,0x00},{0x21,0x00},{0x24,0x00},
                                      {0x06,0x00},{0x12,0x00},{0x18,0x00} };
const unsigned char ccw_data[6][2] = { {0x18,0x00},{0x12,0x00},{0x06,0x00},
```

```
                                        {0x24,0x00},{0x21,0x00},{0x09,0x00} };
const unsigned char up_data[] = { 255, 242, 229, 217, 206, 195, 185, 176, 166, 158,
                                  158, 158, 158, 158, 158, 158, 158, 158, 158, 158 } ;
const unsigned char run_cw_data[8][2] = { {0x00,0x00},{0x21,0x00},{0x06,0x00},
                                          {0x24,0x00},{0x18,0x00},{0x09,0x00},
                                          {0x12,0x00},{0x00,0x00} };
const unsigned char run_ccw_data[8][2] = { {0x00,0x00},{0x09,0x00},{0x24,0x00},
                                           {0x21,0x00}, {0x12,0x00},{0x18,0x00},
                                           {0x06,0x00},{0x00,0x00} };
```

4.1.3 Main processing function

```
#pragma     sfr
#pragma     EI
#pragma     DI

#include    <stdlib.h>
#include    "Common.h"
#include    "Motor.h"

/*************************************************************************/
/*     3-phase motor control program                                     */
/*************************************************************************/
void   main()
{
unsigned char  proc_no ;                        /* Present processing number */
signed int     speed ;                          /* Indication speed rms */
signed int     accel_spd ;
int            sw, sw_mode ;
/* */
   hinit() ;                                    /* Hardware initialization */
   ainit() ;                                    /* Initialization of area used */
   proc_no = 0 ;
   EI();
   while( 1 ) {
      accel_spd = ( SPEED_MAX - SPEED_MINI ) / 100;
      speed  = ( ( SPEED_MAX - SPEED_MINI ) * (long) volume  / 1024 )
               + SPEED_MINI ;                   /* Indication speed calculation by volume */
      sw =  ~IN_data( SW ) & 0x07 ;             /* Read operation button */
      if  ( sw == 1 ) {
         sw_mode = CW ;
      } else if  ( sw == 2 ) {
         sw_mode = CCW ;
      } else if  ( sw == 4 ) {
         sw_mode = STOP ;
      }
      o_speed = ( int ) (abs(object_speed)/(100/P));
                                                /* rmp=> zero-cross pulse / 100 mSEC */
```

```
      switch( proc_no ) {
/* STOP processing */
          case 0 :
             if  ( sw_mode == CW ) {
                DI() ;
                object_speed  = SPEED_MINI ;   /* Set target speed to minimum value */
                stop_flag = OFF ;
                timer_count = WATCH_START ;    /* Set speed monitor start time to 5 SEC */
                accel_count = ACCEL_VAL_1ST ;  /* Set acceleration/deceleration counter */
                init_flag = 2 ;                /* CCW initial request */
                start_init() ;                 /* Initialize revolution start */
                EI() ;
                proc_no = 1 ;                  /* Set next processing number */
             } else if  ( sw_mode == CCW ) {
                DI() ;
                stop_flag = OFF ;              /* Stop flag off */
                object_speed = -SPEED_MINI ;   /* Set target speed to minimum value */
                timer_count = WATCH_START ;    /* Set speed monitor start time to */
                                               /* 5 SEC */
                accel_count = ACCEL_VAL_1ST ;  /* Set acceleration/deceleration counter */
                init_flag = 3 ;                /* CCW initial request */
                start_init() ;                 /* Initialize revolution start */
                EI() ;
                proc_no = 4 ;                  /* Set CCW processing number */
             }
             break ;
/* CW processing, acceleration */
          case 1 :
             if  ( accel_count == 0 ) {
                accel_count = ACCEL_VAL ;      /* Set acceleration/deceleration counter */
                if  ( object_speed  < speed ) {
                   object_speed += accel_spd ;
                   if  ( object_speed > speed ) object_speed = speed;
                   timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
                } else if ( object_speed  > speed ) {
                   object_speed -= accel_spd ;
                   if  ( object_speed < speed ) object_speed = speed;
                   timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
                } else {
                   proc_no = 2 ;               /* Constant-speed processing */
                }
             }
             if  ( (sw_mode == CCW) || (sw_mode == STOP) ) {
                proc_no = 3 ;                  /* Deceleration, set processing number */
             }
             break ;
/* CW processing, constant-speed */
          case 2 :
```

```
            object_speed = speed ;
            if  ( (sw_mode == CCW) || (sw_mode == STOP) ) {
               proc_no = 3 ;                  /* Deceleration, set processing number */
            }
            break ;
/* CW stop processing */
         case 3 :
            if  ( accel_count == 0 ) {
               accel_count = ACCEL_VAL ;      /* Set acceleration/deceleration counter */
               if  ( object_speed  > SPEED_MINI ) {
                  object_speed -= accel_spd ;
                  if  ( object_speed < SPEED_MINI ) object_speed = SPEED_MINI;
                  timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
               } else {
                  stop_flag = ON ;            /* Stop flag on */
                  proc_no = 0 ;               /* Set stop processing number */
               }
            }
            break ;
/* CCW processing, acceleration */
         case 4 :
            if  ( accel_count == 0 ) {
               accel_count = ACCEL_VAL ;      /* Set acceleration/deceleration counter */
               if  ( object_speed  < -speed ) {
                  object_speed += accel_spd ;
                  if  ( object_speed > -speed ) object_speed = -speed;
                  timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
               } else if ( object_speed  > -speed ) {
                  object_speed -= accel_spd ;
                  if  ( object_speed < -speed ) object_speed = -speed;
                  timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
               } else {
                  proc_no = 5 ;               /* Constant-speed processing */
               }
            }
            if  ( (sw_mode == CW) || (sw_mode == STOP) ) {
               proc_no = 6 ;                  /* Deceleration, set processing number */
            }
            break ;
/* CCW processing, constant-speed */
         case 5 :
            object_speed = -speed ;
            if  ( (sw_mode == CW) || (sw_mode == STOP) ) {
               proc_no = 6 ;                  /* Deceleration, set processing number */
            }
            break ;
/* CCW stop processing */
         case 6 :
```

```
        if  ( accel_count == 0 ) {
            accel_count = ACCEL_VAL ;       /* Set acceleration/deceleration counter */
            if  ( object_speed  < -SPEED_MINI ) {
                object_speed += accel_spd ;
                if  ( object_speed > -SPEED_MINI ) object_speed = -SPEED_MINI;
                timer_count = WATCH_START ; /* Set speed monitor start time to 5 SEC */
            } else {
                stop_flag = ON ;            /* Stop flag on */
                proc_no = 0 ;               /* Set stop processing number */
            }
        }
        break ;
    }

    if  ( ( proc_no == 2 ) || ( proc_no == 5 ) ) {
        if  (  timer_count == 0 ) {
            if  ( abs( object_speed - now_speed ) > SA_SPEED_MAX ) {
                error_flag = ERR_NO2 ;          /* Set error No. */
            }
        }
    }

    if  ( disp_co == 0 ) {
        led_num(1, d_speed / P );               /* Number of revolutions */
        d_speed = 0 ;
        disp_co = 100 ;
        if  ( abs(now_speed) == 0 ) {
            disp_co = 0;
        }
        led_num(2, 1000/PWM_TS );               /* Carrier frequency */
        led_num(3, cont_time / 10 );            /* Overall processing time */
        led_num(4, cont_time1 / 10 );           /* Operation processing time */
    }
    if  ( error_flag ) {
        while( 1 ) {
            OUT_data( LED41, ~0x00 ) ;          /* LED display off */
            OUT_data( LED42, ~0x00 ) ;
            timer_count = 50 ;
            while( timer_count ) ;
            if  ( error_flag == ERR_NO1 ) {
                OUT_data( LED41, ~0x9e ) ;      /* E1 display */
                OUT_data( LED42, ~0x60 ) ;
            } else if ( error_flag == ERR_NO2 ) {
                OUT_data( LED41, ~0x9e ) ;      /* E2 display */
                OUT_data( LED42, ~0xda ) ;
            } else {
                OUT_data( LED41, ~0x9e ) ;      /* E3 display */
                OUT_data( LED42, ~0xf2 ) ;
```

```
            }
            timer_count = 50 ;
            while( timer_count ) ;
         }
      }
   }
}
```

4.1.4 LED display function

```
/*****************************************************************************/
/*     LED value display subroutine                                          */
/*              no  : Display area number (1 to 4)                           */
/*              data: Display data (0 to 99)                                 */
/*****************************************************************************/
void led_num( int no, long data )
{
   if  ( no == 1 ) {
      data = data % 10000;
      OUT_data( LED11, ~led_pat[data/1000]&0xff ) ;
      OUT_data( LED12, ~led_pat[(data%1000)/100]&0xff ) ;
      OUT_data( LED13, ~led_pat[(data%100)/10]&0xff ) ;
      OUT_data( LED14, ~led_pat[data%10]&0xff ) ;
   } else if  ( no == 2 ) {
      OUT_data( LED21, ~led_pat[(data%100)/10]&0xff ) ;
      OUT_data( LED22, ~led_pat[data%10]&0xff ) ;
   } else if  ( no == 3 ) {
      OUT_data( LED31, ~led_pat[(data%100)/10]&0xff ) ;
      OUT_data( LED32, ~led_pat[data%10]&0xff ) ;
   } else {
      OUT_data( LED41, ~led_pat[(data%100)/10]&0xff ) ;
      OUT_data( LED42, ~led_pat[data%10]&0xff ) ;
   }
}
/*****************************************************************************/
/*     External I/O output subroutine                                        */
/*              reg : Output register number                                 */
/*              data: Output data                                            */
/*****************************************************************************/
void   OUT_data( unsigned short reg, unsigned short data )
{
      if  ( reg == WRESET ) {
            P3.3 = 0;
            data = 1;                        /* Dummy step */
            P3.3 = 1;
      } else {
            P5 = (unsigned char) reg ;
            P4 = (unsigned char) ( data & 0xff) ;
```

```
            PM4 = 0x00 ;
            P54 = LOW ;

              P54 = HIGH ;
      }
}
/*****************************************************************************/
/*     External I/O input subroutine                                         */
/*          reg: Input register number                                       */
/*****************************************************************************/
unsigned short  IN_data( int reg )
{
unsigned char *po;
/*  */
      if    ( reg == SW ) {
              return P3;
      } else {
              return 0;
      }
}
```

4.1.5 Motor control interrupt processing function

```
#pragma     sfr
#pragma     EI
#pragma     DI
#pragma     INTERRUPT INTTM00 int_ETC rb1
#pragma     INTERRUPT INTTM01 int_MOTOR rb1
#pragma     INTERRUPT INTAD int_AD0 rb2
#pragma     INTERRUPT INTP1 int_INTP rb1
#pragma     INTERRUPT INTP2 int_INTP rb1
#pragma     INTERRUPT INTP3 int_INTP rb1
#pragma     INTERRUPT INTTM51 int_DELAY rb3

#include    <stdlib.h>
#include     "Common.h"
#include     "Motor.h"
/*****************************************************************************/
/*      Motor control timer interrupt processing                             */
/*****************************************************************************/
_interrupt
void    int_MOTOR(void)
{
   CR01_int = CR01;
   CR01 = US_TO_CPU(TS)+CR01 ;
   ADS = 0x00 ;                                 // ANI0 sel
   ADM = ADM_START ;
```

```
}
/*****************************************************************************/
/*      Motor control processing                                             */
/*****************************************************************************/
void    Motor_CONT(void)
{
signed int       sa_speed, o_iqap, o_iqa ;
signed int       s_time, cow ;
unsigned char    wk ;
/* */
/*****************************************************************************/
/*      Calculation processing of speed                                      */
/*****************************************************************************/
    sum_speed += int_co ;
    int_co = 0;
    if  ( --speed_co == 0 ) {
        speed_co = 100000 / TS ;              /* Set 100 mSEC counter value */
        now_speed = sum_speed ;               /* Zero-cross pulse / 100 mSEC */
        sum_speed = 0 ;
    }

    if  ( ( stop_flag == OFF ) && ( error_flag == 0 ) ) {
        s_time = TM00 ;
        OUT_data( WRESET, 0 ) ;               /* Reset watchdog timer */
/*****************************************************************************/
/*     Initial revolution processing                                         */
/*****************************************************************************/
        if  ( init_flag ) {
            cow = init_upco ;
            if  ( cow > 4 ) cow = 4;
            if  ( ++init_co > ( (long)up_data[ cow ] * 34000L / ( SPEED_INIT * TS ) ) ) {
                init_co = 0 ;
                if  ( init_flag == 2 ) {
                    wk  = cw_data[ init_pat++  ][0] ;
                } else {
                    wk  = ccw_data[ init_pat++ ][0] ;
                }
                        RTPM01 = ~wk ;

                if  ( init_pat >= 6 ) {
                    init_pat = 0 ;
                    if  ( init_upco > 14 ) {
                        init_flag = 0 ;
                    } else {
                        init_upco++ ;
                    }
                }
            }
```

```
      } else {
/*************************************************************************/
/*    Normal revolution processing                                       */
/*************************************************************************/

      se_speed = o_speed - now_speed ;

      o_iqap  = (int) ( ( (now_speed + sa_speed) * KSP ) >> KSPGETA ) ;
      o_iqa   = (int) (o_iqap + ( o_iqai >> KSIGETA ) ) ;

      if  ( o_iqai > IQAMAX ) {
         o_iqai = IQAMAX ;
      } else if ( o_iqai < -IQAMAX ) {
         o_iqai = -IQAMAX ;
      } else {
         o_iqai += ( KSI * sa_speed ) ;
      }

      pwm_value = o_iqa ;
      if  ( pwm_value <= 0 ) {
         pwm_value = 1 ;
      } else if  ( pwm_value >= PWM_DATA  ) {
         pwm_value = ( PWM_DATA  ) - 1 ;
      }
            TW0BFCM0 =  PWM_DATA - pwm_value ;
            TW0BFCM1 =  PWM_DATA - pwm_value ;
            TW0BFCM2 =  PWM_DATA - pwm_value ;

            cont_time1 = CPU_TO_US( TM00 - s_time );       /* Convert to uSEC */
      }
   } else {
            CE0 = OFF ;                            /* PWM output OFF */
            RTPM01 = 0xFF ;                        /* latch data inversion output */
      now_speed = 0;
      cont_time1 = 0;
   }
}
```

4.1.6 Zero-cross interrupt processing function

```
/*************************************************************************/
/*     U zero-cross point interrupt                                      */
/*     V zero-cross point interrupt                                      */
/*     W zero-cross point interrupt                                      */
/*************************************************************************/
_interrupt void     int_INTP(void)
{
```

```
    int co;
/* */
    if  ( ( ( init_flag == 0 ) && ( stop_flag == OFF) ) ) {
        if  ( object_speed < 0 ) {
              out_pat  = run_ccw_data[ ( P0 >> 1 ) & 0x07 ][0] ;
        } else {
              out_pat  = run_cw_data [ ( P0 >> 1 ) & 0x07 ][0] ;
        }

        co = 781 / abs(now_speed);                    /* Count value for 30 degrees */
        if  ( co == 0 ) co = 1;
        CR51 = co;
        TMC51 = 0x80;                                 // Timer start
    }
    int_co++ ;
}
```

4.1.7 10 mSEC interval interrupt processing function

```
/*************************************************************************/
/*     Other timer interrupt processing (10 mSEC interval)               */
/*************************************************************************/
_interrupt void     int_ETC(void)
{
        EI() ;
        CR00 = US_TO_CPU (10000) + CR00 ;

/* Wait timer processing */
    if  ( timer_count != 0 ) {
        timer_count -= 1 ;
    }
/* Acceleration/deceleration timer processing */
    if  ( accel_count != 0 ) {
        accel_count -= 1 ;
    }
/* */
    if  ( disp_co != 0 ) {
        d_speed += now_speed ;
        disp_co -= 1 ;
    }
}
```

4.1.8 Delay control interrupt processing function

```
/*************************************************************************/
/*      30 degrees delay timer interrupt proccessing                     */
/*************************************************************************/
__interrupt void    int_DELAY(void)
```

```
{
        TMC51 = 0x00;                                   // Timer stop
        if  ( ( ( init_flag == 0 ) && ( stop_flag == OFF) ) ) {
                RTPM01 = ~out_pat ;
        }
}
```

4.1.9 A/D converter interrupt processing function

```
/*************************************************************************/
/*      A/D converter interrupt processing for U-phase current and speed volume */
/*      A/D converter interrupt processing for volume                    */
/*************************************************************************/

_interrupt void     int_AD0(void)
{
        ADM = ADM_STOP ;
        IF1H.4 = 0 ;
        EI() ;
        switch ( ADS & 0x07 ) {
        case 0x00 :
                // ANI0 comversion end
                iua = ((( ADCR >> 6 ) & 0x3ff ) - 0x200) ;
                if  ( abs(iua) > MAX_I ) {
                                CE0 = OFF ;             /* PWM output off */
                                RTPM01 = 0xFF ;         /* Latch data inversion output */
                                error_flag = ERR_NO1 ;  /* Set error No. */
                }
                ADS = 0x01 ;                            /* ANI1 sel */
                ADM = ADM_START ;
                break ;
        case 0x01 :
                // ANI1 comversion end
                iva = (( ADCR & 0x3ff ) - 0x200) ;
                if  ( abs(iva) > MAX_I ) {
                                CE0 = OFF ;             /* PWM output off */
                                RTPM01 = 0xFF ;         /* Latch data inversion output */
                                error_flag = ERR_NO1 ;  /* Set error No. */
                }
                ADS = 0x02 ;                            /* ANI2 sel */
                ADM = ADM_START ;
                break ;
        default :
                // ANI2 comversion end
                volume = 1023 - (( ADCR >> 6 ) & 0x3ff ) ;     /* Set volume value */

                EI() ;
```

```
            Motor_CONT () ;
            cont_time = CPU_TO_US (TM00-CR01_int) ;        /* Convert to uSEC */
            break ;
        }
}
```

4.1.10 Hardware initialization processing function

```
/************************************************************************/
/*     Hardware (peripheral I/O) initialization                         */
/************************************************************************/
void   hinit( void )
{
    IMS = 0xC8 ;                                /* Memory area */

    /* WDT stop */
    WDTM = 0x77 ;

    /* Main system clock and high speed mode setting */
    while ( OSTC != 0x1f ) ;                    /* oscillation stabilization wating */
    MCM = 0x03 ;                                /* X1 input clock sel */

    // Initialization of port mode registers for FPGA access
    PM5 = 0xE0 ;
    PM4 = 0x00 ;

    // LED OFF
    OUT_data( LED11, 0xff ) ;
    OUT_data( LED12, 0xff ) ;
    OUT_data( LED13, 0xff ) ;
    OUT_data( LED14, 0xff ) ;
    OUT_data( LED21, 0xff ) ;
    OUT_data( LED22, 0xff ) ;
    OUT_data( LED31, 0xff ) ;
    OUT_data( LED32, 0xff ) ;
    OUT_data( LED41, 0xff ) ;
    OUT_data( LED42, 0xff ) ;

    // 16-bit timer TM00 setting
    TMC00 = 0x00 ;                      // Set after disabling operation
    CRC00 = 0x00 ;                      // Set CR00 and CR01 as cpmpair registers
    TOC00 = 0x00 ;                      // Disable output
    PRM00 = 0x01 ;                      // fx/2^2 (5 MHz = 0.2 us)
    CR00 = US_TO_CPU (10000) + TM00 ;   // 10 mSEC
    CR01 = US_TO_CPU ( TS ) + TM00 ;    // TS uSEC
    TMC00 = 0x04 ;                      // free-running and no overflow
```

```
    // 8-bit timer TM51 setting
    TCL51 = 0x05 ;
    CR51  = 0xff ;
    TMC51 = 0x80 ;

    // TMW0 setting
    TW0C = 0x20 ;                       // Stop timer, fx/16 = 1.25 MHz,
                                        // Occurs interrupt each underflow
    TW0M = 0x04 ;                       //
    TW0OC = 0x00 ;                      // TW0TO0 to TW0TO5 output are permited

/* Real-time output port setting */
    RTBH01 = 0xFF ;                     // Output data latch
    RTBL01 = 0xFF ;                     // Output data latch
    DCCTL01 = 0xB0 ;                    // 7: PWM modulated RTP output
                                        // 6: RTP10,12,14 PWM modulated output
                                        // 5: RTP11,13,15 PWM modulated output
                                        // 4: inversion enable
    // A/D setting
    ADM = ADM_DEF ;                     // Stop operation, select mode,
                                        // conversion time 3.6us, Comparator=ON
    ADS = 0x00 ;
    PFM = 0x00 ;
    PFT = 0x00 ;

    // External interrupt setting
    EGP = 0x0e ;                        // rising edge
    EGN = 0x0e ;                        // falling edge

    // Interrupt mask setting
    TMMK00 = ENABLE ;
    TMMK01 = ENABLE ;
    PMK1 = ENABLE ;
    PMK2 = ENABLE ;
    PMK3 = ENABLE ;
    ADMK = ENABLE ;
    TMMK51 = ENABLE ;

    // Priority setting
    ADPR = 0 ;
    PPR1 = 0 ;
    PPR2 = 0 ;
    PPR3 = 0 ;
    TMPR51 = 0 ;
```

```
    // Port 3 setting
    PM3 = 0xF7 ;
}
```

4.1.11 Common area initialization processing function

```
/*************************************************************************/
/*     Common area initialization                                        */
/*************************************************************************/
void   ainit( void )
{
/* Initialization of flags */
    error_flag = 0 ;                    /* Clear error flag */
    init_flag = OFF ;                   /* Initial flag off */
    disp_co = 100 ;
    d_speed = 0 ;
/* Motor control area initialization */
    stop_flag   = ON ;                  /* Stop flag on */
    object_speed = 0 ;                  /* Target speed 0 */
    o_iqai = 0 ;                        /* Speed integral value 0 */
}
```

4.1.12 Revolution start initialization processing function

```
/*************************************************************************/
/*     Revolution start initialization                                   */
/*************************************************************************/
void   start_init( void )
{
    int  i;
/*  */
    sum_speed = 0 ;
    speed_co = 100000 / TS ;

    init_co = 0 ;
    init_pat = 0 ;
    init_upco = 0 ;

    // Real-time output port setting
    RTPM01    = 0xFF ;                  // Latch data inversion output
    RTPC01    = 0xA0 ;                  // 7: enable operation
                                        // 5: 6-bit x 1-channel

    // PWM period setting for initial revolution
    TW0BFCM3  = PWM_DATA ;
    TW0BFCM0  = PWM_INIT ;
    TW0BFCM1  = PWM_INIT ;
```

```
    TW0BFCM2  = PWM_INIT ;
    pwm_value = PWM_DATA - PWM_INIT ;
    CE0 = ON ;                                  // PWM output ON
}
```