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

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.

APPLICATION NOTE

H8/300H Tiny Series

Stepper Motor Using 1-2 Phase Excitation

Introduction

The H8/3687 offers various built-in functions. Of these, P63 to P60 and the timer Z compare match function can be used to control a two-phase stepper motor.

The stepper motor is controlled using 1-2 phase excitation.

Target Device

H8/300H Tiny Series H8/3687 CPU

Contents

1. Specifications

- The H8/3687 offers various built-in functions. Of these, P63 to P60 and the timer Z compare match function are used to control a two-phase stepper motor.
- The stepper motor is controlled using 1-2 phase excitation.
- This task involves repeating the operations for rotating the stepper motor forwards, stopping it, rotating the stepper motor in the reverse direction, and then again stopping it.
- The task realizes slew-up and slew-down processing by using software.
- Figure 1 shows the connection diagram for controlling a two-phase stepper motor.

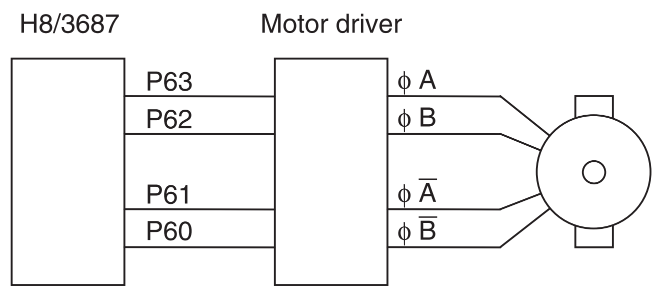


Figure 1 Connections for Controlling a Two-Phase Stepper Motor

2. Description of Functions

2.1 This sample task uses a permanent magnet stepper motor (Japan Servo KP6P8-701). Table 1 lists the standard specifications of the KP6P8-701.

Table 1 Standard Specifications of KP6P8-701

Item	Value
Model	KP6P8-701
Phases	2
Stepping angle [deg./step]	7.5
Voltage [V]	12
Current [A/PHASE]	0.33
Resistance of windings [Ω/PHASE]	36
Inductance [mH/PHASE]	28
Maximum static torque [mN·m]	78.4
Detent torque [mN·m]	1.3
Rotor inertia [g·cm^2]	23.7

2.2 The following describes the H8/3687 functions used for stepper motor control. Figure 2 is a block diagram of the functions used for this sample task.

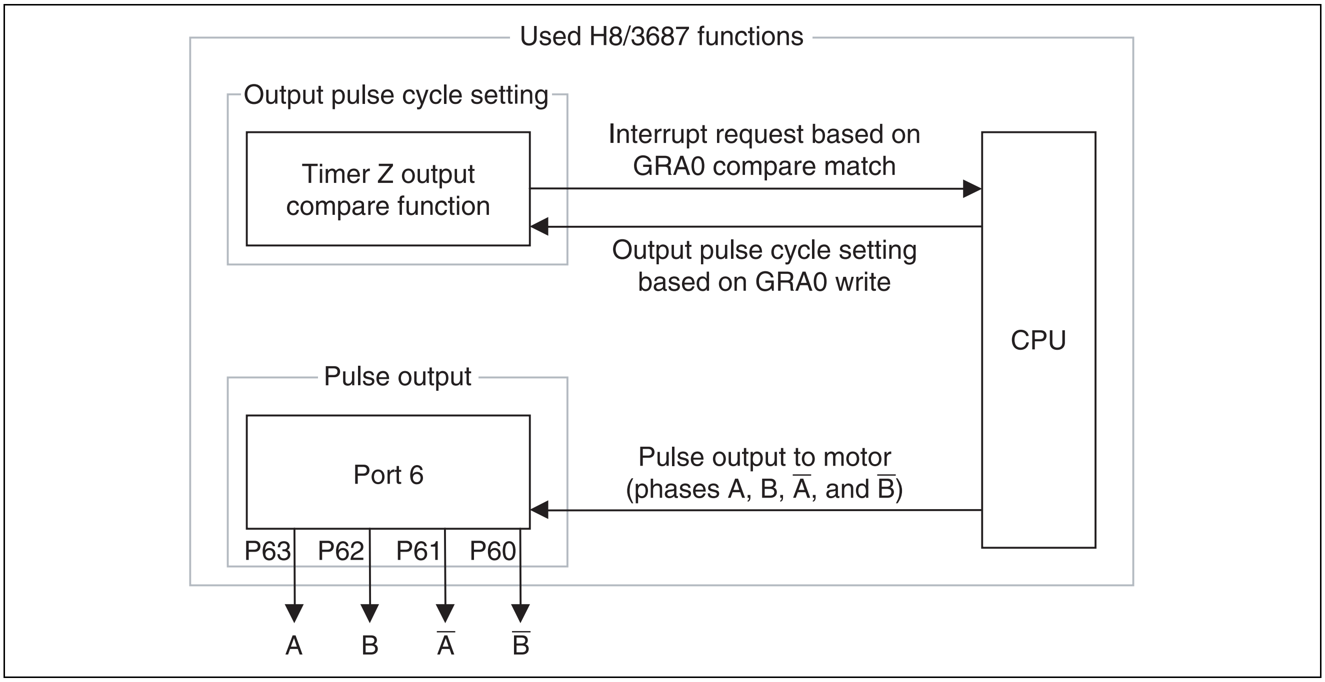


Figure 2 H8/3687 Functions Used

2.3 Timer Z is a 16-bit multi-function timer that incorporates an output compare function and input capture function. This sample task uses the output compare function of timer Z. Figure 3 is a block diagram of timer Z. This block diagram of timer Z is explained below.

— System clock (ϕ)
  16-MHz reference clock for controlling the CPU and peripheral functions
— Prescaler S (PSS)
  13-bit counter to which ϕ is input. This counter is incremented for each cycle.
— Timer control register 0 (TCR0)
  This register is used to select a clock to be input to TCNT0 and a method for clearing TCNT0. In this sample task, the counter inputs a clock of ϕ/4, counts up on the rising edge of the clock, and then clears TCNT0 when GRA0 compare match/input capture is performed.
— Timer I/O control register A0 (TIORA0)
  This register controls GRA0. This sample task sets GRA0 as an output compare register and disables output to the FTIOA0 pin.
— Timer status register 0 (TSR0)
  This register indicates the state of timer Z. In this sample task, input capture/compare match flag A (IMFA) is set to 1 when GRA0 compare match is performed.
— Timer interrupt enable register (TIER0)
  This register enables or disables each interrupt request. This sample task enables an interrupt request based on the IMFA or IMFB flag of TSR0, and disables any other interrupt requests.
— Timer counter 0 (TCNT0)
  This counter is a 16-bit read/write up-counter. This counter is incremented according to an input internal/external clock. This sample task uses an input clock of ϕ/4, and increments the counter on the rising edge of the clock.
— General register A0 (GRA0)
  This register is a 16-bit read/write register. The contents of GRA0 are compared with TCNT0 at all times. When a match is found, IMFA of TSR0 is set to 1. If IMIEA of TIER0 is set to 1 at this time, an interrupt request is issued to the CPU.
— Timer start register (TSTR)
  This register is used to select whether to start or stop TCNT0 and TCNT1. In this sample task, TCNT0 is set to start count, while TCNT1 is set to stop count.
— Timer mode register (TMDR)
  This register is used to specify whether to operate the TCNT0 and TCNT1 timers in sync or independently. This sample task uses TCNT0 independently of TCNT1.

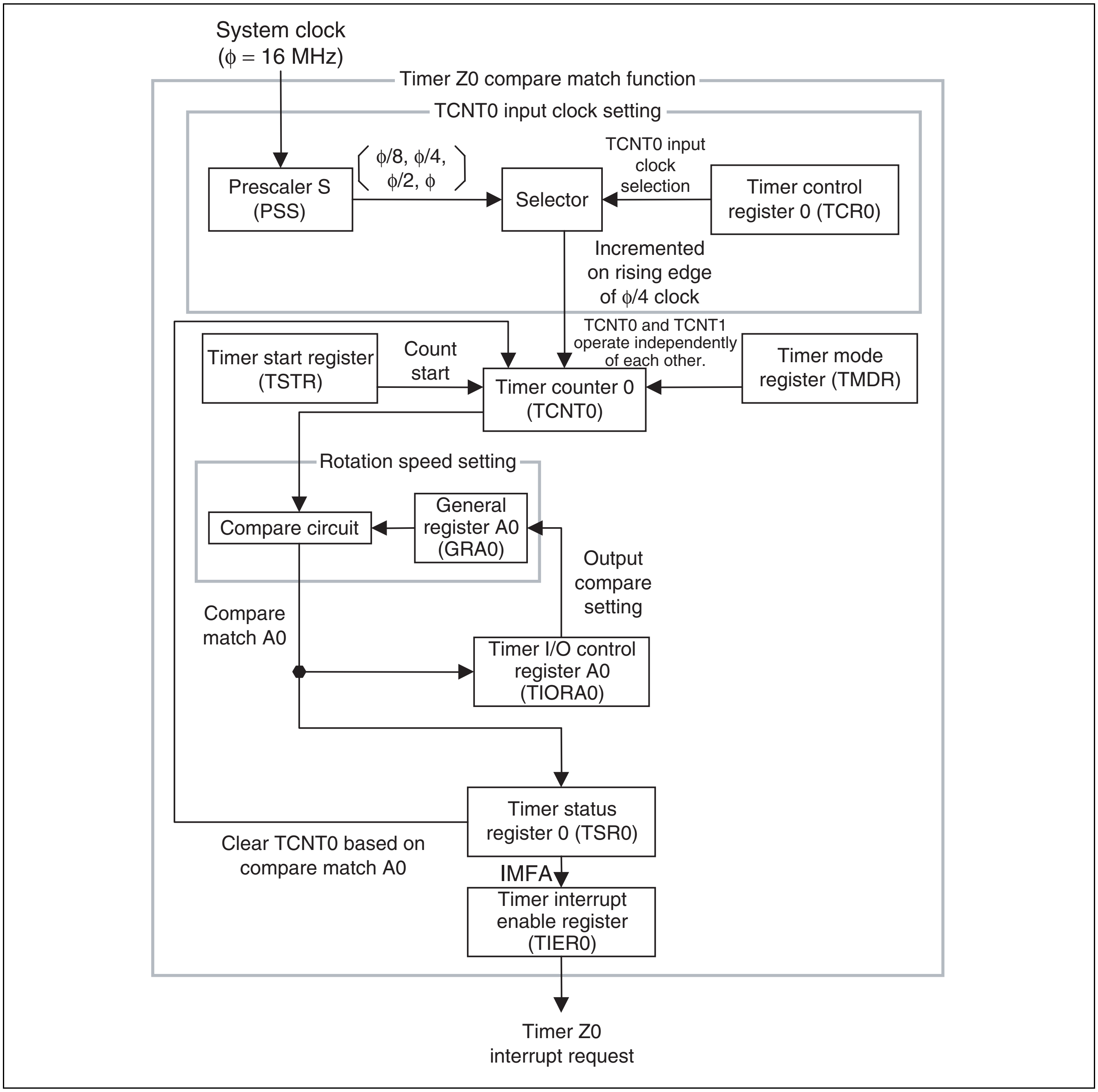


Figure 3 Block Diagram of Timer Z0

2.4 Port 6 is an 8-bit I/O port. This sample task uses P63 to P60 of port 6. Figure 4 is a block diagram of port 6. The functions of port 6 are explained below.

— Port data register 6 (PDR6)
  P63 to P60 are used for excitation phase driving of the stepper motor.
— Port control register 6 (PCR6)
  P63 to P60 are set as output pins.

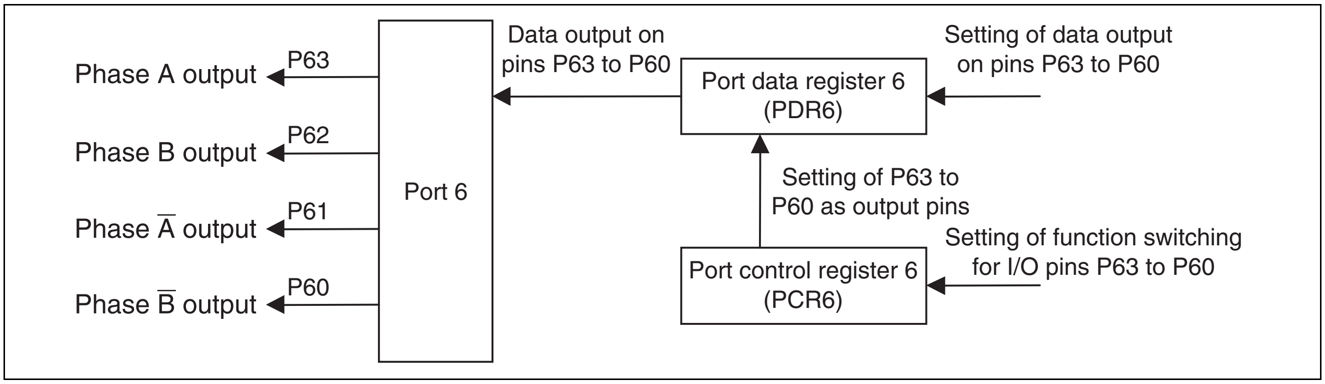


Figure 4 Block Diagram of Port 6 Functions

2.5 Table 2 lists the function assignments for this sample task.

Table 2 Function Assignments

Name	Assigned function
System clock PSS TCNT0	Reference clock for operating the stepper motor
TCR0	Sets TCNT0 operation.
TIORA0	Sets the output compare register.
TSR0	Indicates the state of timer Z.
TIER0	Enables or disables each interrupt request.
TSTR	Starts the counting of TCNT0.
TMDR	Sets TCNT0 and TCNT1 to operate independently of each other.
GRA0	Sets the duration of one step of the stepper motor.
PDR6 PCR6	Drives the excitation phase of the stepper motor.

3. Description of Operation

3.1 Example of stepper motor operation

Figure 5 shows an example of operating the two-phase stepper motor with a stepping angle of 7.5 [deg./step] by using 1-2 phase excitation. The operation is outlined below.

- As shown in Figure 5, a high pulse causes the corresponding phase to be excited.
- First, phase A is excited. At this time, the rotor is positioned to phase A.
- Next, phases A and B are excited simultaneously. At this time, the rotor is positioned halfway between phases A and B. Then, the rotor is rotated by excitation in the following order: phase B → phases B and $\overline{A}$ → phase $\overline{A}$ → phases $\overline{A}$ and $\overline{B}$ → phase $\overline{B}$ → phases $\overline{B}$ and A.
- For reverse rotation, the stepper motor is rotated by excitation in the following order: phases B and A → phase $\overline{B}$ → phases $\overline{A}$ and $\overline{B}$ → phase $\overline{A}$ → phase B and $\overline{A}$ → phase B → phase A and B→ phase A.
- For stop operation, the stepper motor is stopped by keeping the last phase of a forward rotation or reverse rotation excited for a certain period of time.

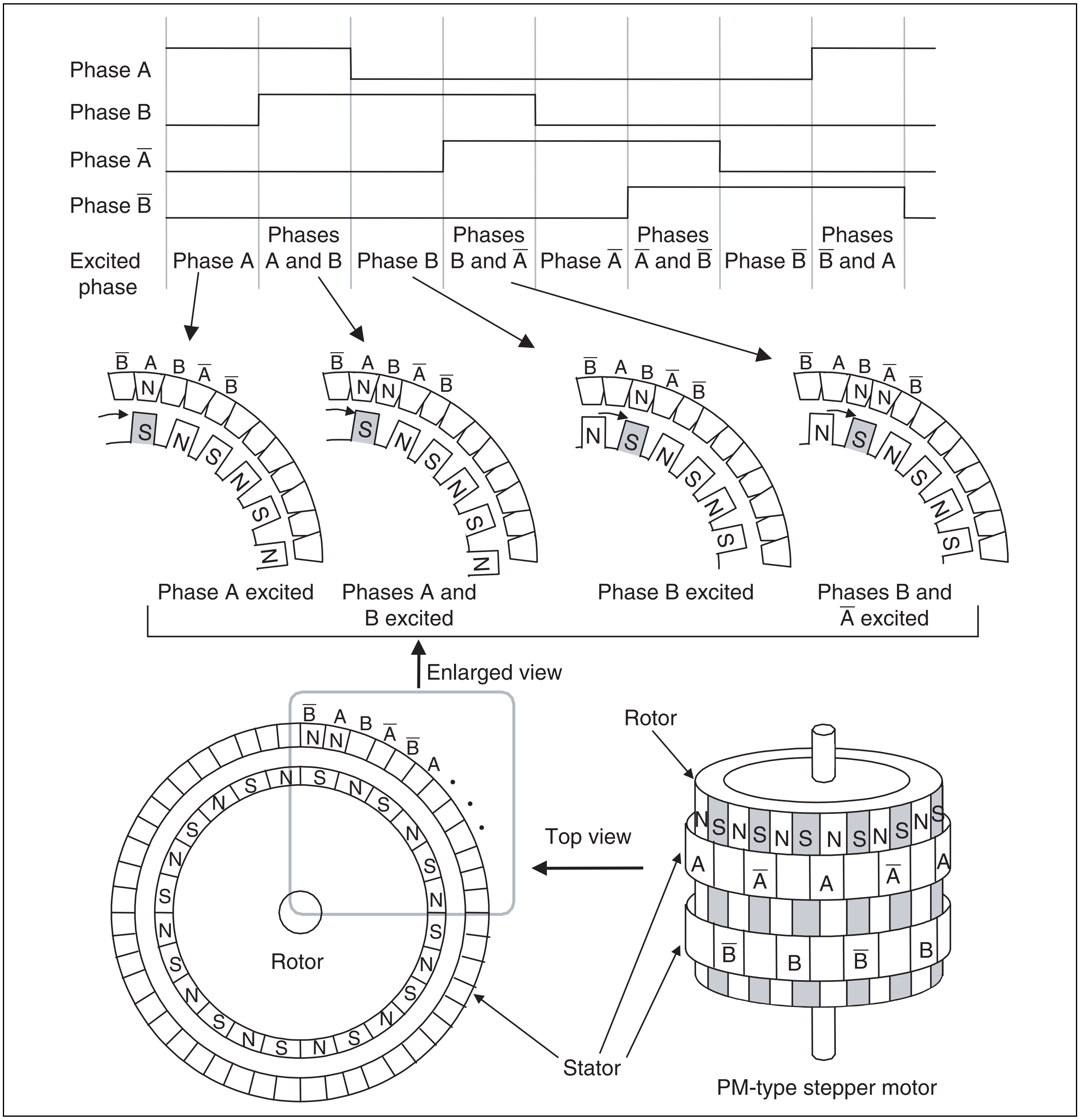


Figure 5 Example of Stepper Motor Operation

3.2 Slew-up and slew-down operation

Acceleration/deceleration-controlled pulses are output. Slew-up/slew-down operation maintains the synchronization of the motor. If a series of short-cycle pulses is suddenly output to operate the motor, the motor may not be able to handle the load and will not rotate. Slew-up and slew-down operation is used to avoid this problem.

The following explains the principle of operation.

— The pulse cycles are gradually shortened to output the specified number of pulses (slew-up operation).
— The specified number of pulses are output at a regular pulse cycle (constant-speed operation).
— The pulse cycle is gradually extended to output the specified number of pulses (slew-down operation).

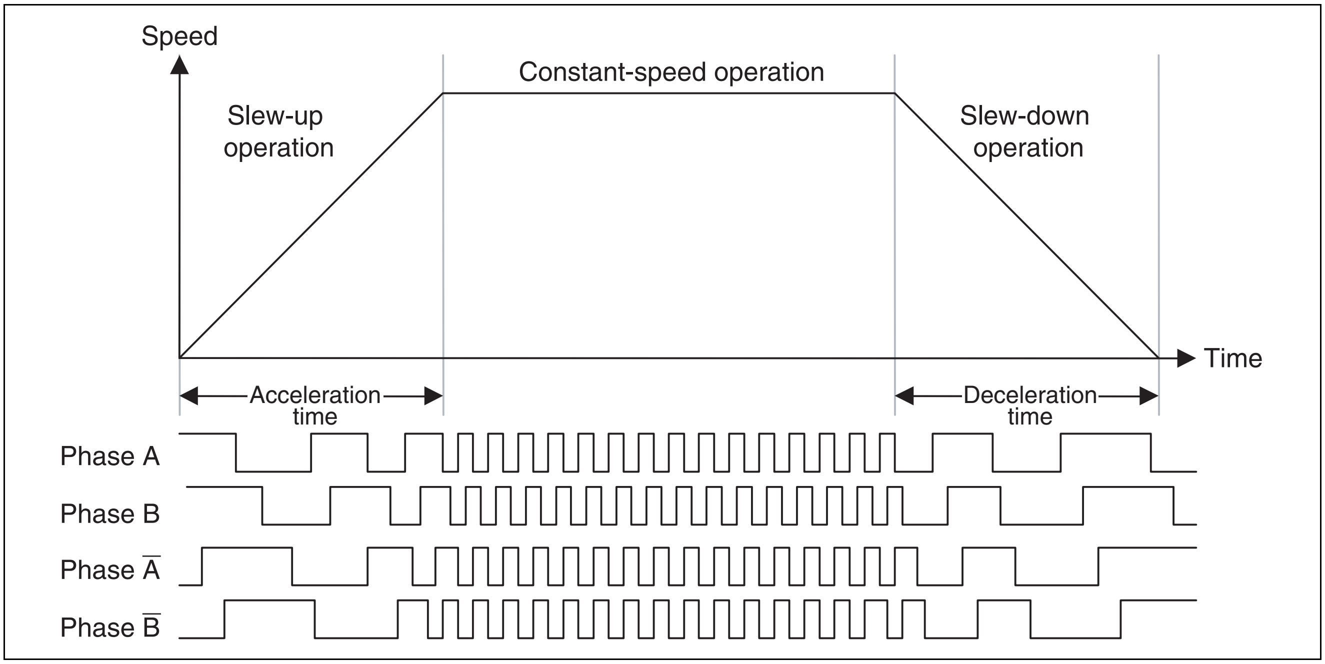


Figure 6 Example of Slew-up and Slew-down Operation

3.3 Figure 7 is a flowchart illustrating stepper motor control.

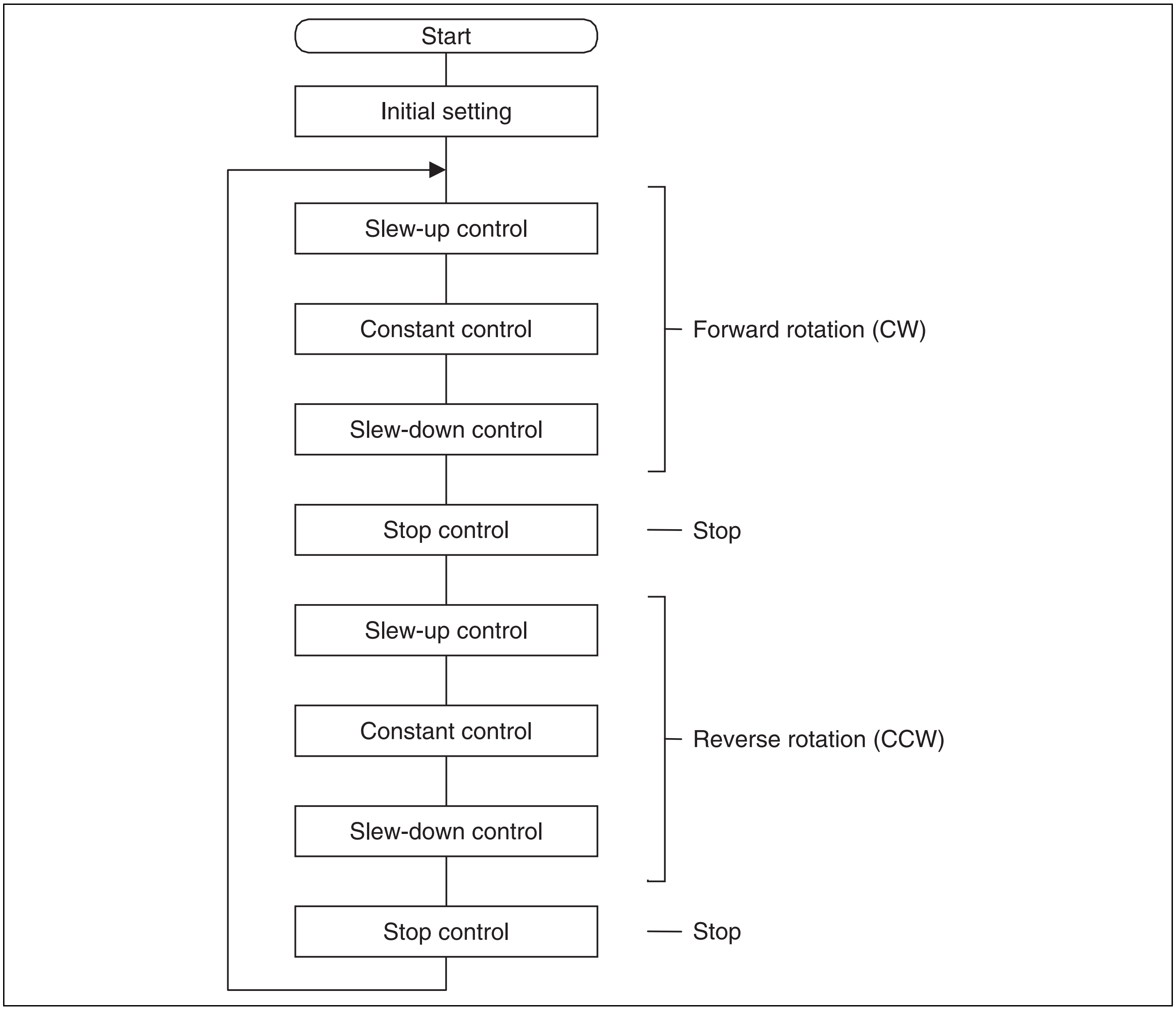


Figure 7 Flowchart of Stepper Motor Control

3.4 Expression for calculating timer Z interrupt time

— The expression for calculating the timer Z interrupt time based on the setting of the output compare register (GRA0) is as follows:

$$\text{Timer Z interrupt time} = \frac{\text{GRA0} + 1}{(\text{System clock } \phi/4)}$$

$$= \frac{\text{GRA0} + 1}{16\text{MHz}/4}$$

$$= \frac{\text{GRA0} + 1}{4} \quad [\mu\text{s}]$$

3.5 Figure 8 illustrates the principle of slew-up control during forward rotation

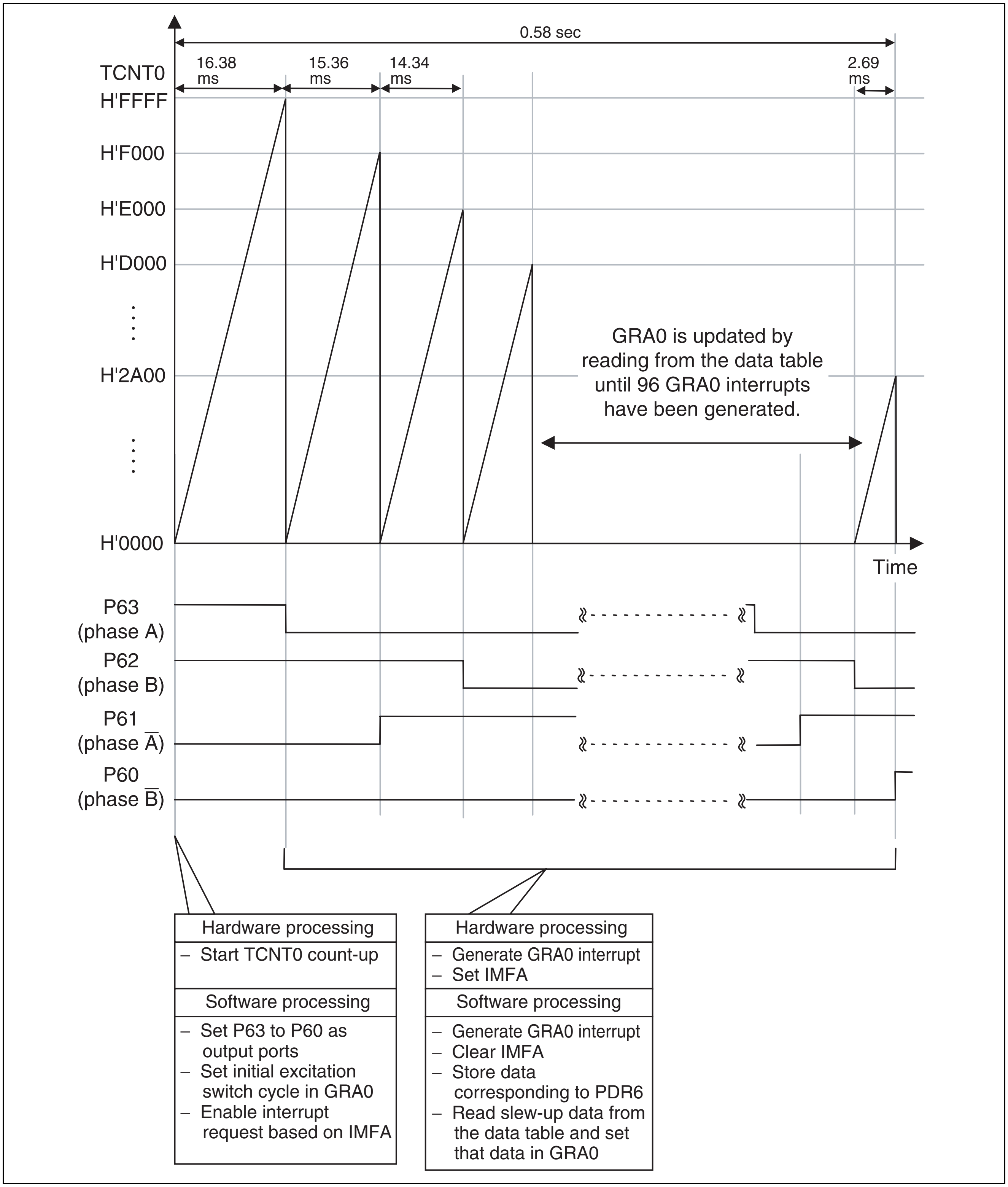


Figure 8 Principle of Slew-up Control during Forward Rotation

3.6 Figure 9 illustrates the principle of constant control during forward rotation

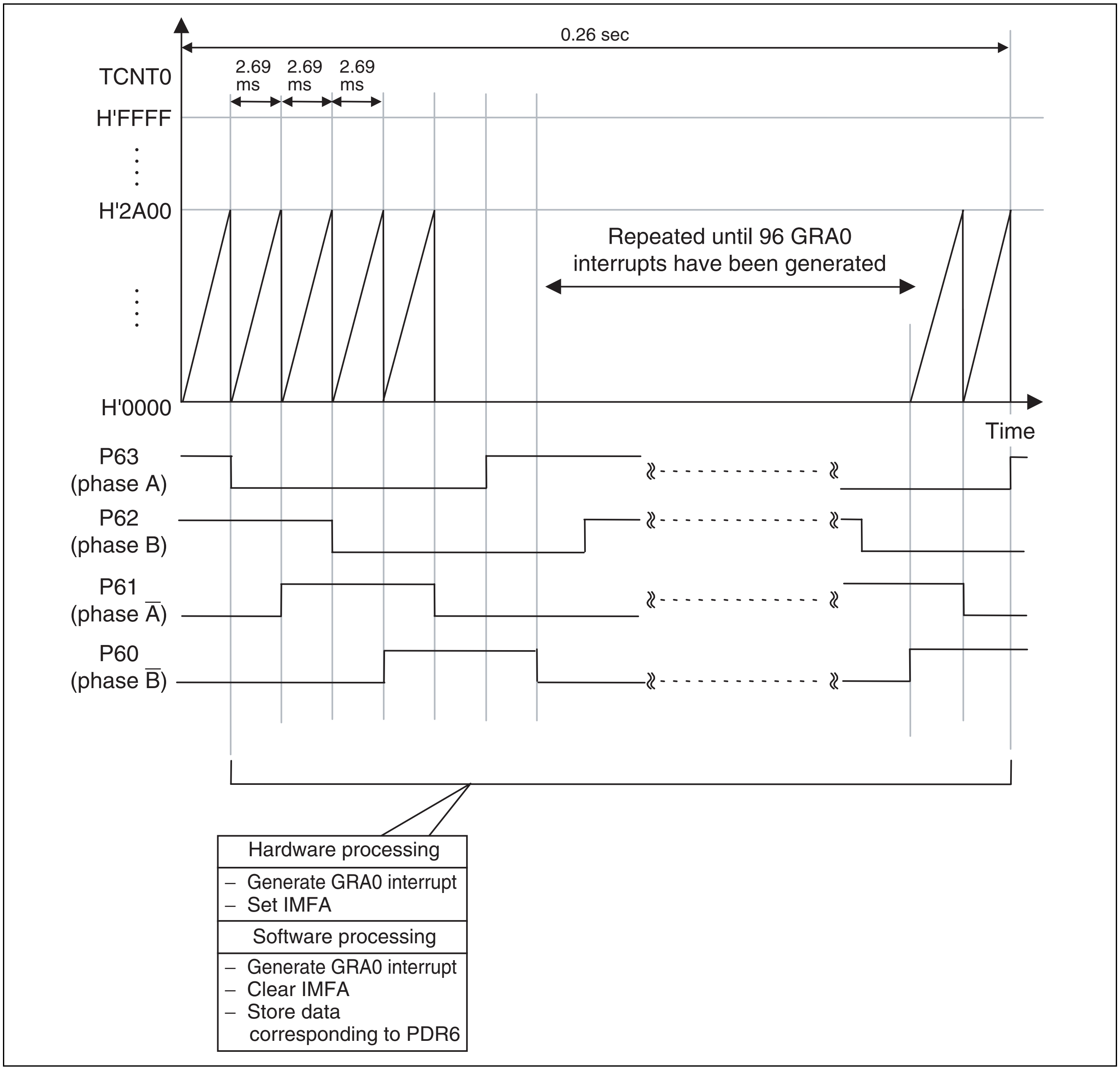


Figure 9 Principle of Constant Control during Forward Rotation

3.7 Figure 10 illustrates the principle of slew-down control during forward rotation.

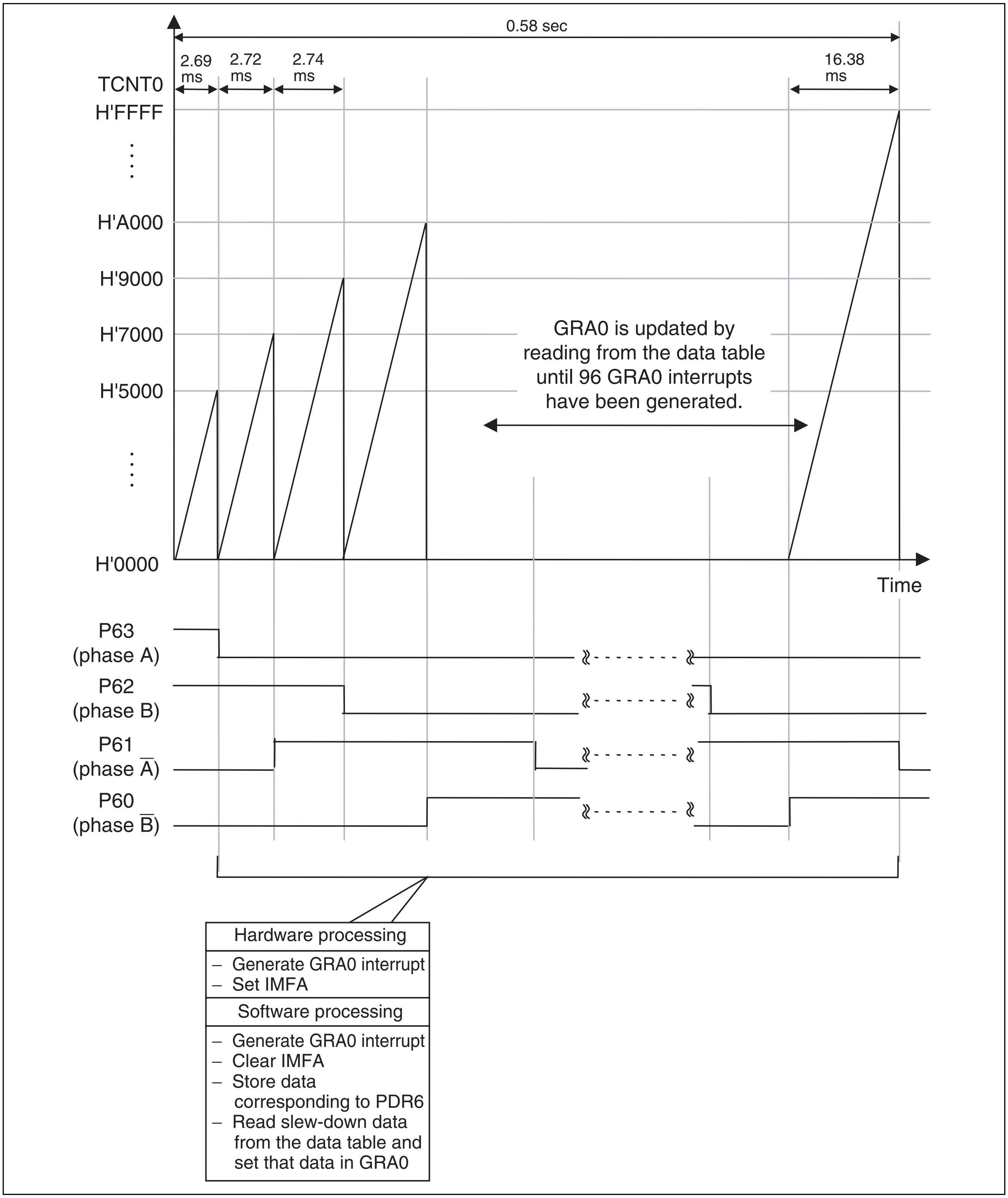


Figure 10 Principle of Slew-down Control during Forward Rotation

3.8 Figure 11 illustrates the principle of stop control.

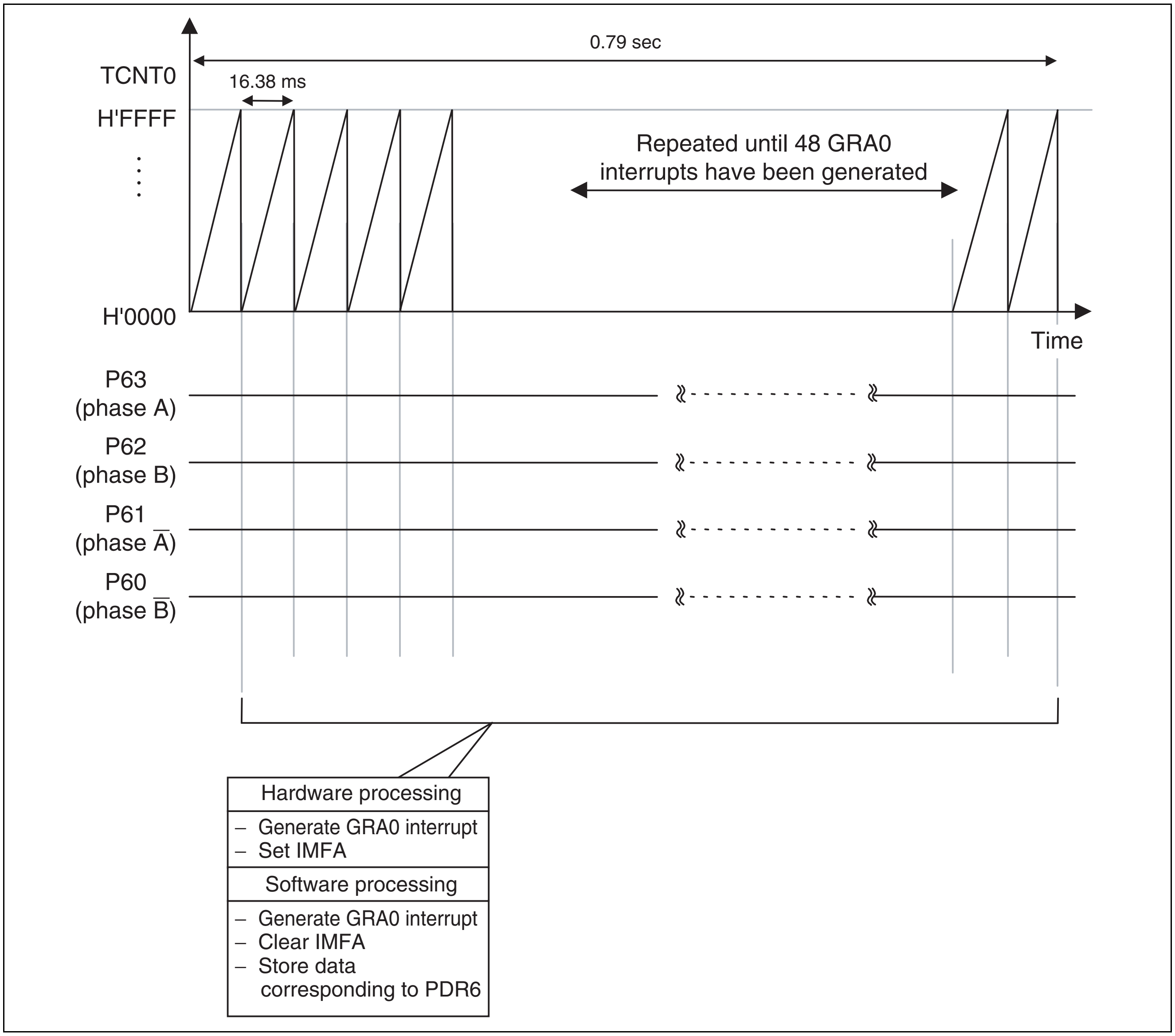


Figure 11 Principle of Stop Control

3.9 Figure 12 illustrates the principle of slew-up control during reverse rotation.

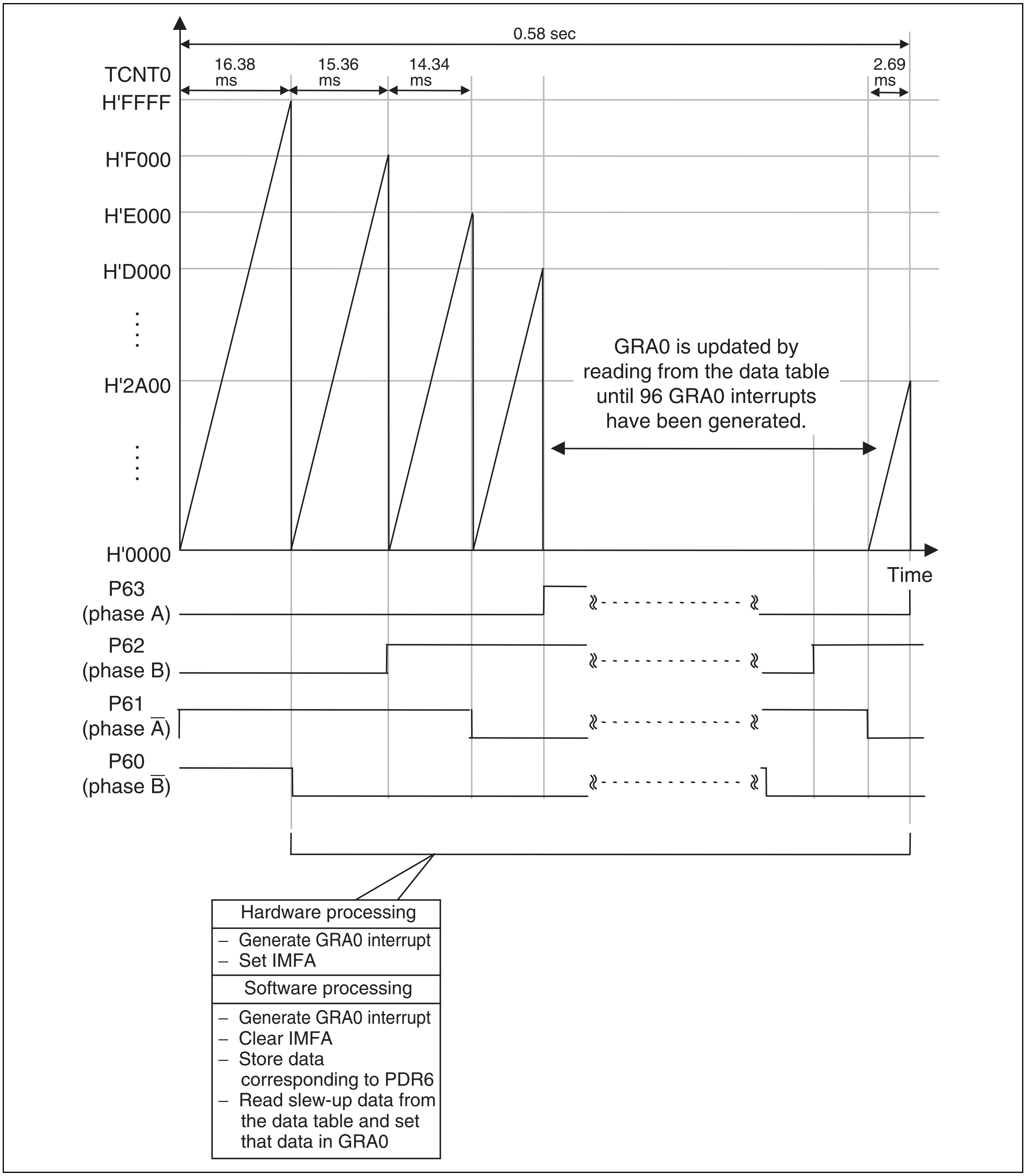


Figure 12 Principle of Slew-up Control during Reverse Rotation

3.10 Figure 13 illustrates the principle of constant control during reverse rotation.

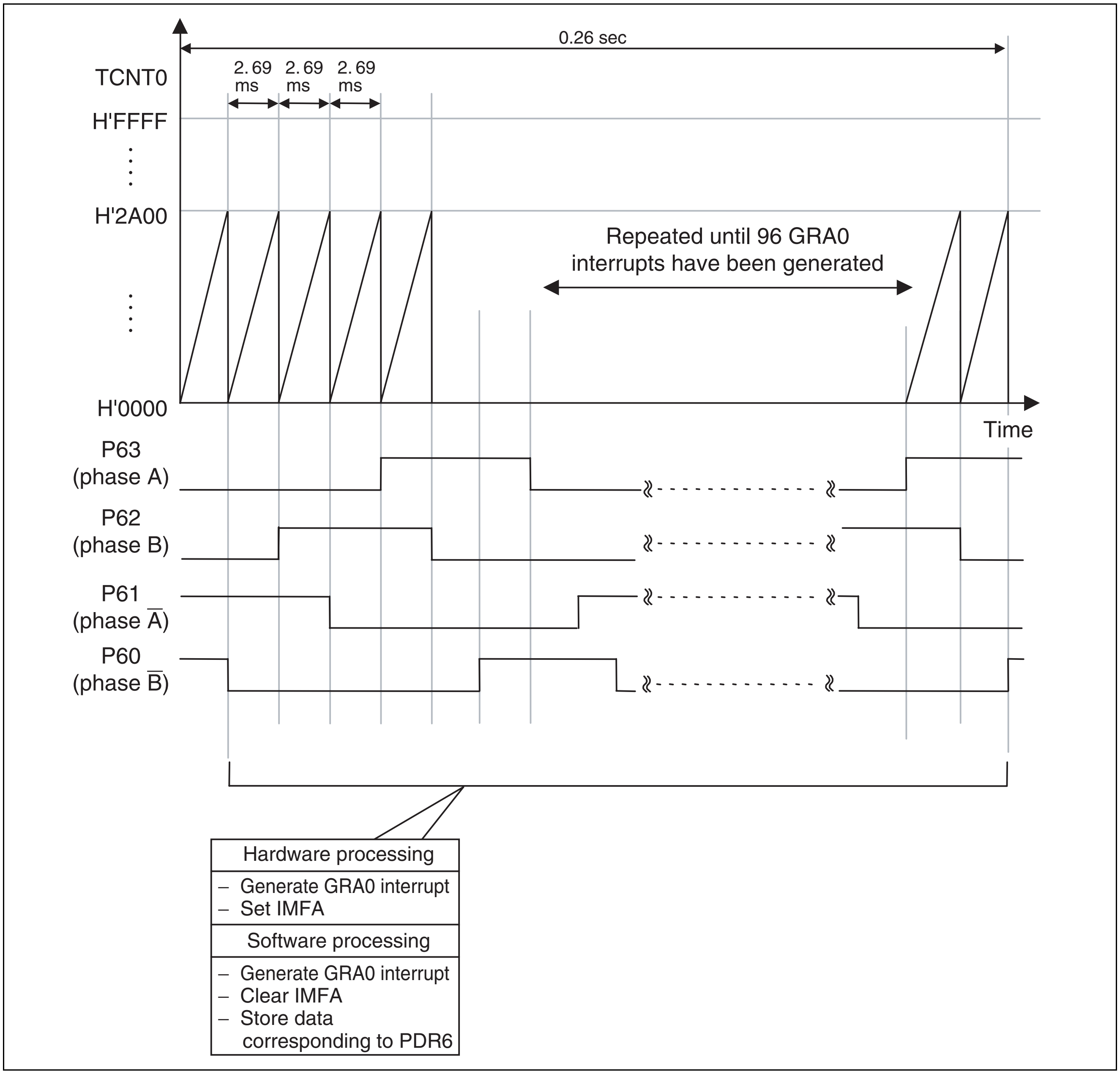


Figure 13 Principle of Constant Control during Reverse Rotation

3.11 Figure 14 illustrates the principle of slew-down control during reverse rotation.

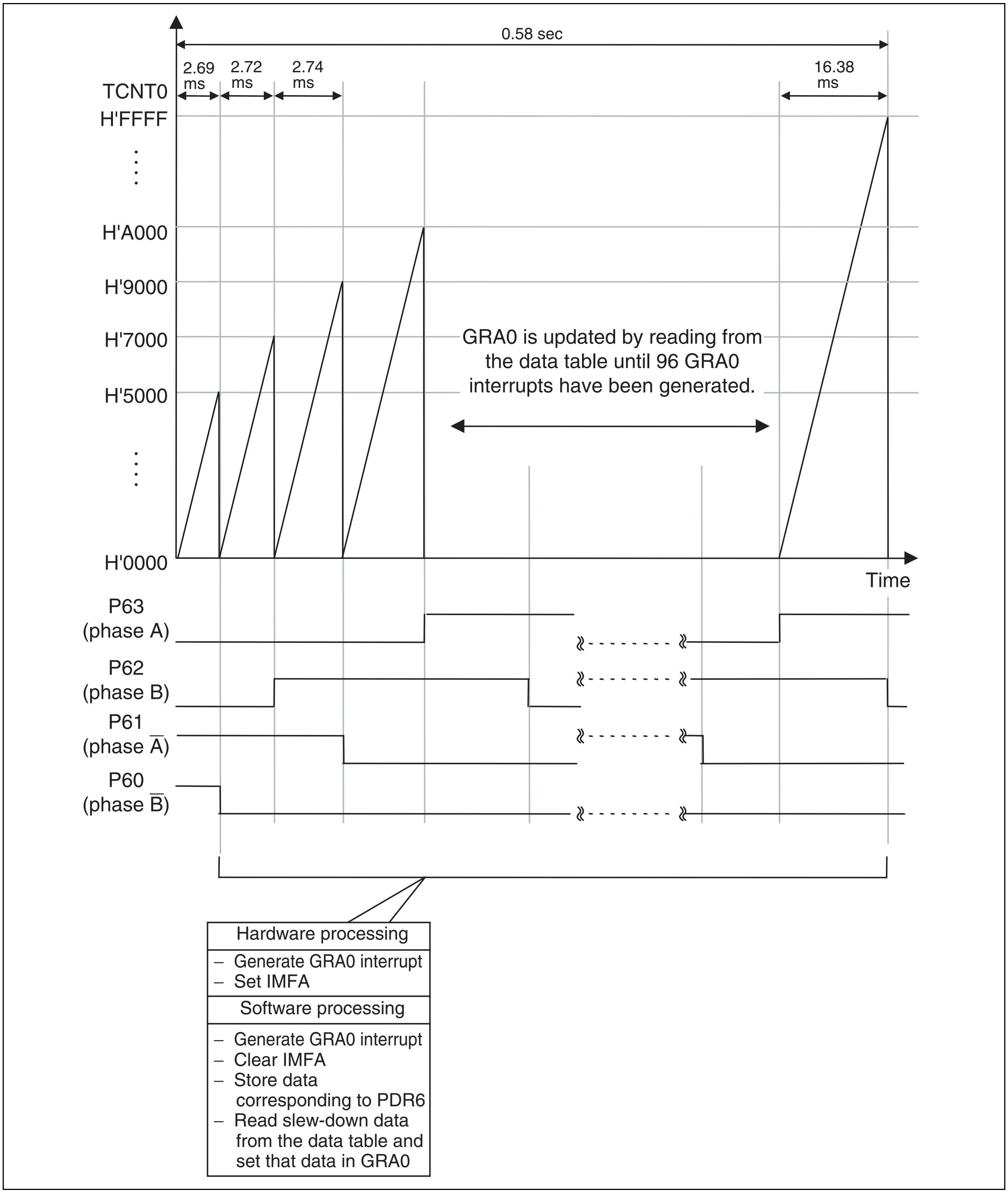


Figure 14 Principle of Slew-down Control during Reverse Rotation

4. Description of Software

4.1 Modules

Table 3 lists the modules used for this sample task.

Table 3 Modules

Module name	Label name	Function
Main routine	main	Initializes the global variables, I/O ports, and timer Z, and enables interrupts.
Timer Z interrupt processing	tzint	Main routine for the stepper motor
Slew-up control during forward rotation	fslueup	Applies slew-up control during forward rotation.
Slew-down control during forward rotation	fsluedwn	Applies slew-down control during forward rotation.
Constant control during forward rotation	fconst	Applies constant control during forward rotation.
Rotation stop	frstop	Stops forward/reverse rotation.
Slew-up control during reverse rotation	rslueup	Applies slew-up control during reverse rotation.
Slew-down control during reverse rotation	rsluedwn	Applies slew-down control during reverse rotation.
Constant control during reverse rotation	rconst	Applies constant control during reverse rotation.

4.2 Arguments

No arguments are used for this sample task.

4.3 Internal registers

The internal registers used for this sample task are described below.

— TCR0 Timer control register 0 Address: H'F700

Bit	Bit name	Setting	Function
7	CCLR2	CCLR2 = 0	Counter clear 2 to 0
6	CCLR1	CCLR1 = 0	CCLR2 = 0, CCLR1 = 0, CCLR0 = 1: Clears TCNT0 when GRA0
5	CCLR0	CCLR0 = 1	compare match/input capture is performed.
4	CKEG1	CKEG1 = 0	Clock edge 1 to 0
3	CKEG0	CKEG0 = 0	CKEG1 = 0, CKEG0 = 0: Increments the counter on the rising edge of the clock.
2	TPSC2	TPSC2 = 0	Timer prescaler 2 to 0
1	TPSC1	TPSC1 = 1	TPSC2 = 0, TPSC1 = 1, TPSC0 = 0: Increments the counter with $\phi/4$.
0	TPSC0	TPSC0 = 0	

— TIORA0 Timer I/O control register A0 Address: H'F701

Bit	Bit name	Setting	Function
2	IOA2	IOA2 = 0	I/O control A2 to 0
1	IOA1	IOA1 = 0	IOA2 = 0, IOA1 = 0, IOA0 = 0: Sets GRA0 as an output
0	IOA0	IOA0 = 0	compare register and disables pin output based on a compare match.

— TSR0 Timer status register 0 Address: H'F703

Bit	Bit name	Setting	Function
0	IMFA	0	Input capture/compare match flag A IMFA = 0: Mismatch between GRA0 and TCNT0 values IMFA = 1: Match between GRA0 and TCNT0 values

— TIER0 Timer interrupt enable register 0 Address: H'F704

Bit	Bit name	Setting	Function
0	IMIEA	1	Input capture/compare match interrupt enable A When IOA2 of TIORA0 = 0 (output compare setting): IMIEA = 0: Disables an interrupt based on the IMFA flag of TSR0. IMIEA = 1: Enables an interrupt based on the IMFA flag of TSR0.

— TCNT0 Timer counter 0 Address: H'F706
Function: 16-bit counter that is incremented on the rising edge of $\phi/4$
Setting: H'0000

— GRA0 General register A0 Address: H'F708
Function: A compare match occurs when a match is found between the setting of GRA0 and the count value of TCNT0.
Setting: H'FFFF

— TSTR Timer start register Address: H'F720

Bit	Bit name	Setting	Function
0	STR0	0	Channel 0 counter start STR0 = 0: Causes TCNT0 to stop count operation. STR0 = 1: Causes TCNT0 to start count operation.

— TMDR   Timer mode register   Address: H'F721

Bit	Bit name	Setting	Function
0	SYNC	0	Timer synchronization SYNC = 0: Operates TCNT0 independently of TCNT1. SYNC = 1: Operates TCNT0 and TCNT1 in sync with each other.

— TFCR   Timer function control register   Address: H'F723

Bit	Bit name	Setting	Function
1	CMD1	CMD1 = 0	Combination mode 1 to 0
0	CMD0	CMD0 = 0	CMD1 = 0, CMD0 = 0: Operates Channels 0 and 1 normally.

— TOER   Timer output master enable register   Address: H'F724

Bit	Bit name	Setting	Function
0	EA0	1	Master enable A0 EA0 = 0: Enables output on the FTIOA0 pin. EA0 = 1: Disables output on the FTIOA0 pin.

— TOCR   Timer output control register   Address: H'F725

Bit	Bit name	Setting	Function
0	TOA0	0	Output level select A0 TOA0 = 0: Causes FTIOA0 to output 0 initially. TOA0 = 1: Causes FTIOA0 to output 1 initially.

— PDR6   Port data register 6   Address: H'FFD9
Function: Uses P63 to P60 for excitation phase driving of the stepper motor.
Setting: H'08

— PCR6   Port control register 6   Address: H'FFE9
Function: Sets P63 to P60 as output pins when PCR6 = H'0F.
Setting: H'0F

4.4 RAM

Table 4 lists the RAM used for this sample task.

Table 4 RAM

Label name	Function	Memory consumption	Module
tzcnt	Elements of array pattbl[] representing stepper motor excitation data	1 byte	Main routine Timer Z interrupt processing Forward rotation slew-up control Forward rotation slew-down control Forward rotation constant control Rotation stop Reverse rotation slew-up control Reverse rotation slew-down control Reverse rotation constant control
sluecnt	Elements of array uptbl[] used for slew-up and slew-down operation	1 byte	Main routine Timer Z interrupt processing Forward rotation slew-up control Forward rotation slew-down control Reverse rotation slew-up control Reverse rotation slew-down control
nextmode	Sets the operating mode of the stepper motor.	1 byte	Main routine Timer Z interrupt processing
modecnt	Sets the number of interrupts in the operating mode of the stepper motor.	2 bytes	Main routine Timer Z interrupt processing
pattbl[8]	Excitation pattern data table for the stepper motor	8 bytes	Main routine Forward rotation slew-up control Forward rotation slew-down control Forward rotation constant control Rotation stop Reverse rotation slew-up control Reverse rotation slew-down control Reverse rotation constant control
uptbl[96]	Interrupt time data table for slew-up and slew-down operation	192 bytes	Main routine Forward rotation slew-up control Forward rotation slew-down control Reverse rotation slew-up control Reverse rotation slew-down control

4.5 Data table variables

— Data table for switching the excitation pattern of the stepper motor

pattbl[8]={

0x08,	Excites phase A (P63).
0x0C,	Excites phases A (P63) and B (P62).
0x04,	Excites phase B (62).
0x06,	Excites phases B (62) and $\overline{\mathrm{A}}$ (P61).
0x02,	Excites phase $\overline{\mathrm{A}}$ (P61).
0x03,	Excites phases $\overline{\mathrm{A}}$ (P61) and $\overline{\mathrm{B}}$ (P60).
0x01,	Excites phase $\overline{\mathrm{B}}$ (P60).
0x09,	Excites phases $\overline{\mathrm{B}}$ (P60) and A (P63).

}

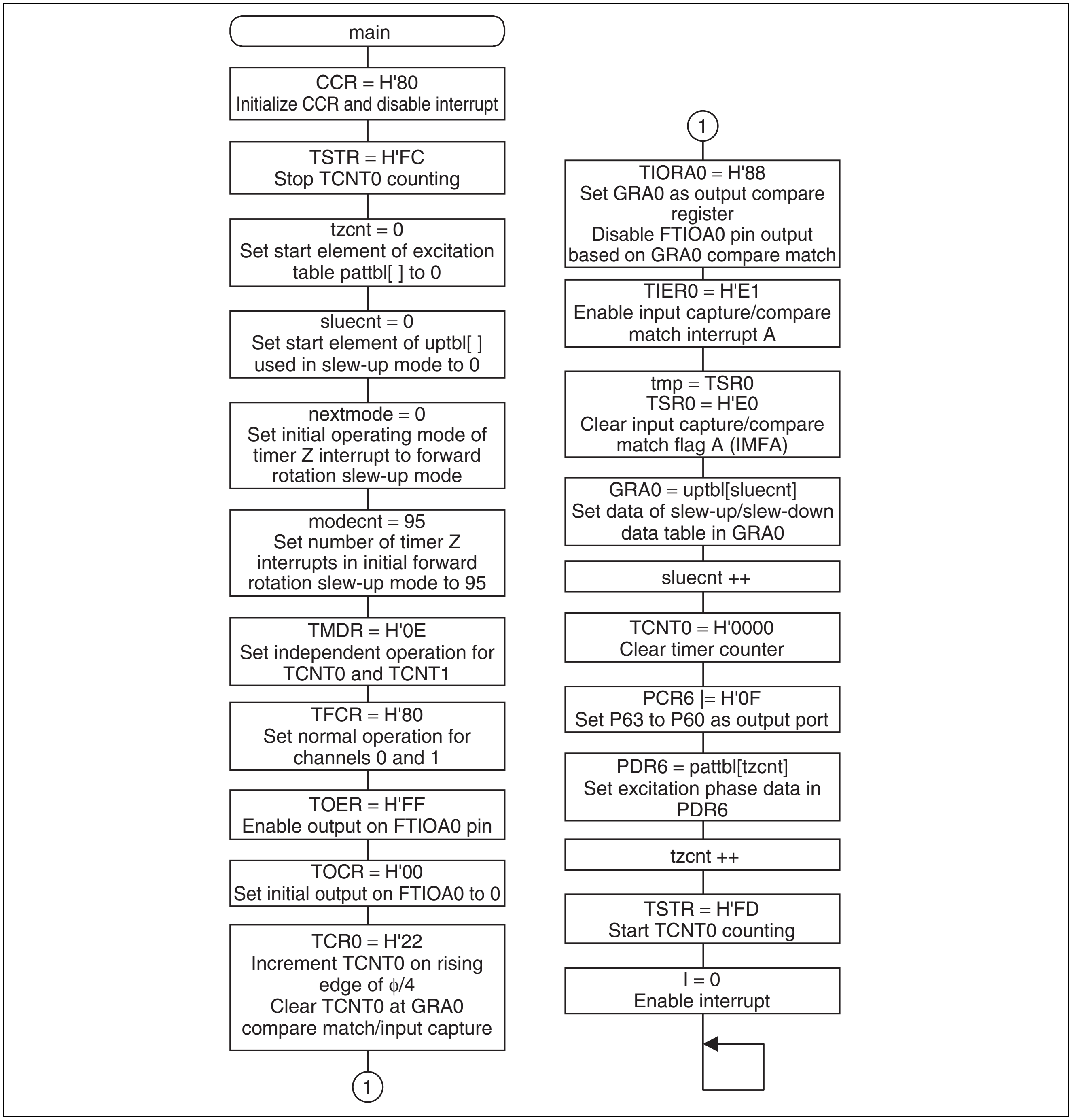
— Data table for slew-up and slew-down setting

```
uptbl[96]={
    0xFFFF,0xF000,0xE000,0xD500,0xCE40,0xC738,0xC030,0xB928,0xB220,0xAB18,
    0xA410,0x9DD0,0x9790,0x9150,0x8CA0,0x87F0,0x8340,0x8020,0x7D00,0x7AA8,
    0x7850,0x75F8,0x74BD,0x7148,0x6FB8,0x6E28,0x6C98,0x6B08,0x6978,0x67E8,
    0x66BC,0x6590,0x6464,0x6338,0x620C,0x6144,0x607C,0x5FB4,0x5EEC,0x5DCA,
    0x5CA8,0x5B86,0x5A64,0x5942,0x5820,0x56FE,0x55DC,0x54BA,0x5398,0x5276,
    0x5154,0x5032,0x4F10,0x4DEE,0x4CCC,0x4BAA,0x4A88,0x4966,0x4844,0x4722,
    0x4600,0x44DE,0x43BC,0x429A,0x4178,0x4056,0x3F34,0x3E12,0x3CF0,0x3BC4,
    0x3A34,0x3890,0x373C,0x35E8,0x3494,0x33D6,0x3318,0x325A,0x319C,0x30DE,
    0x3064,0x2FEA,0x2F70,0x2EF6,0x2E7C,0x2E02,0x2D9C,0x2D36,0x2CD0,0x2C6A,
    0x2C04,0x2B9E,0x2B38,0x2AD2,0x2A6C,0x2A00,
}
```

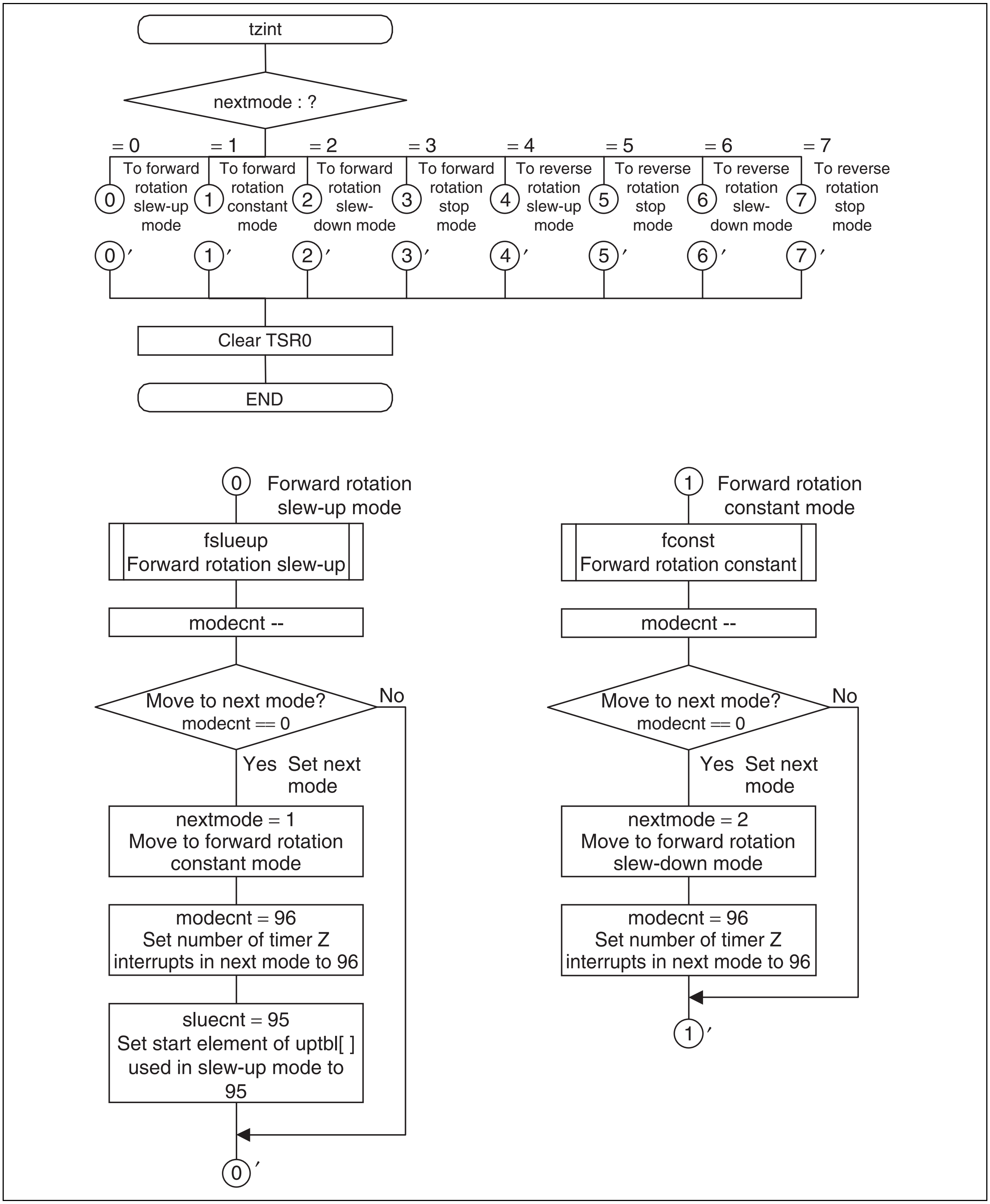
Data in uptbl[] is sequentially written to GRA0 by a GRA0 interrupt generated during slew-up and slew-down until the stepper motor makes one complete revolution (96 steps).

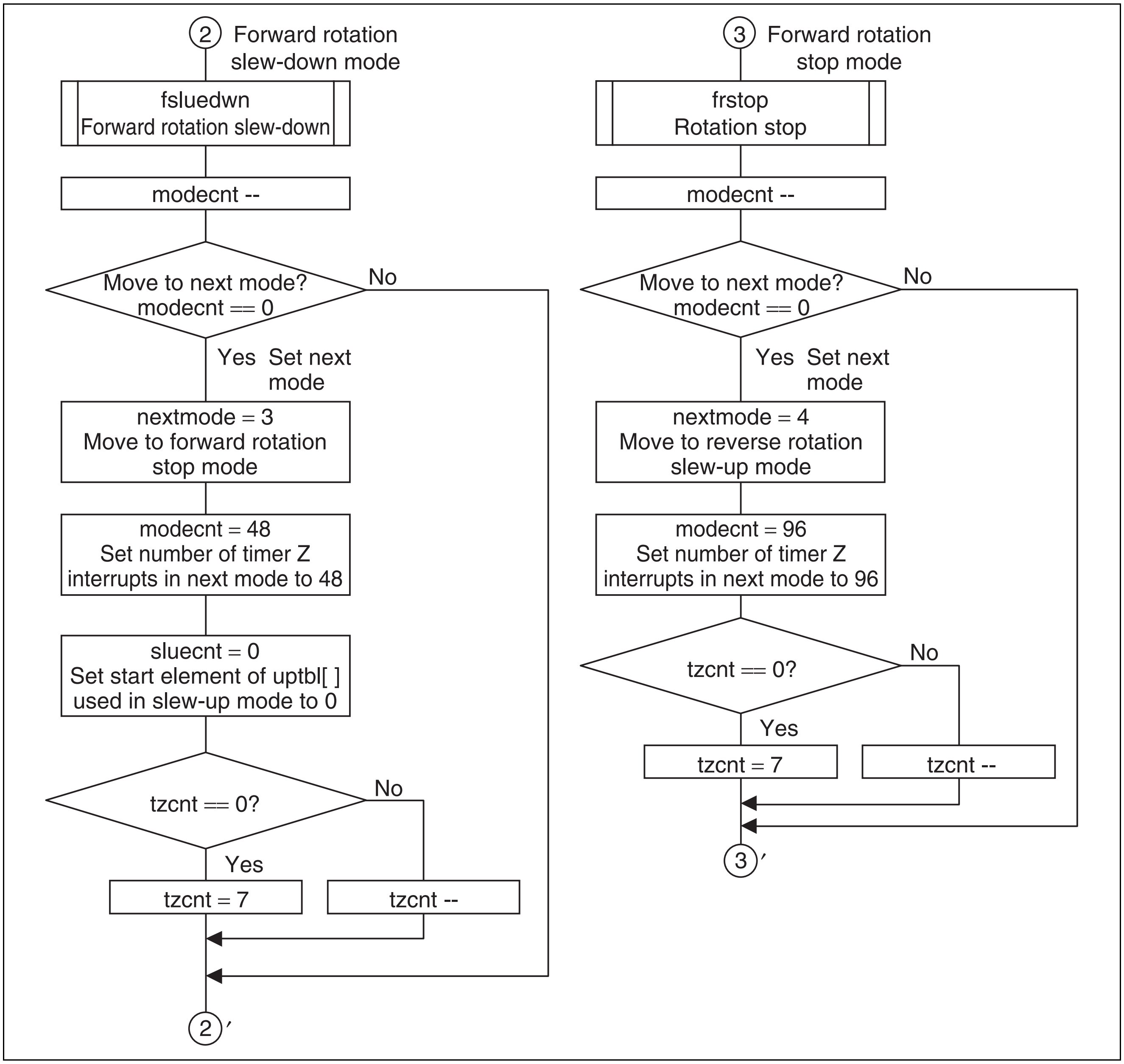
5. Flowchart

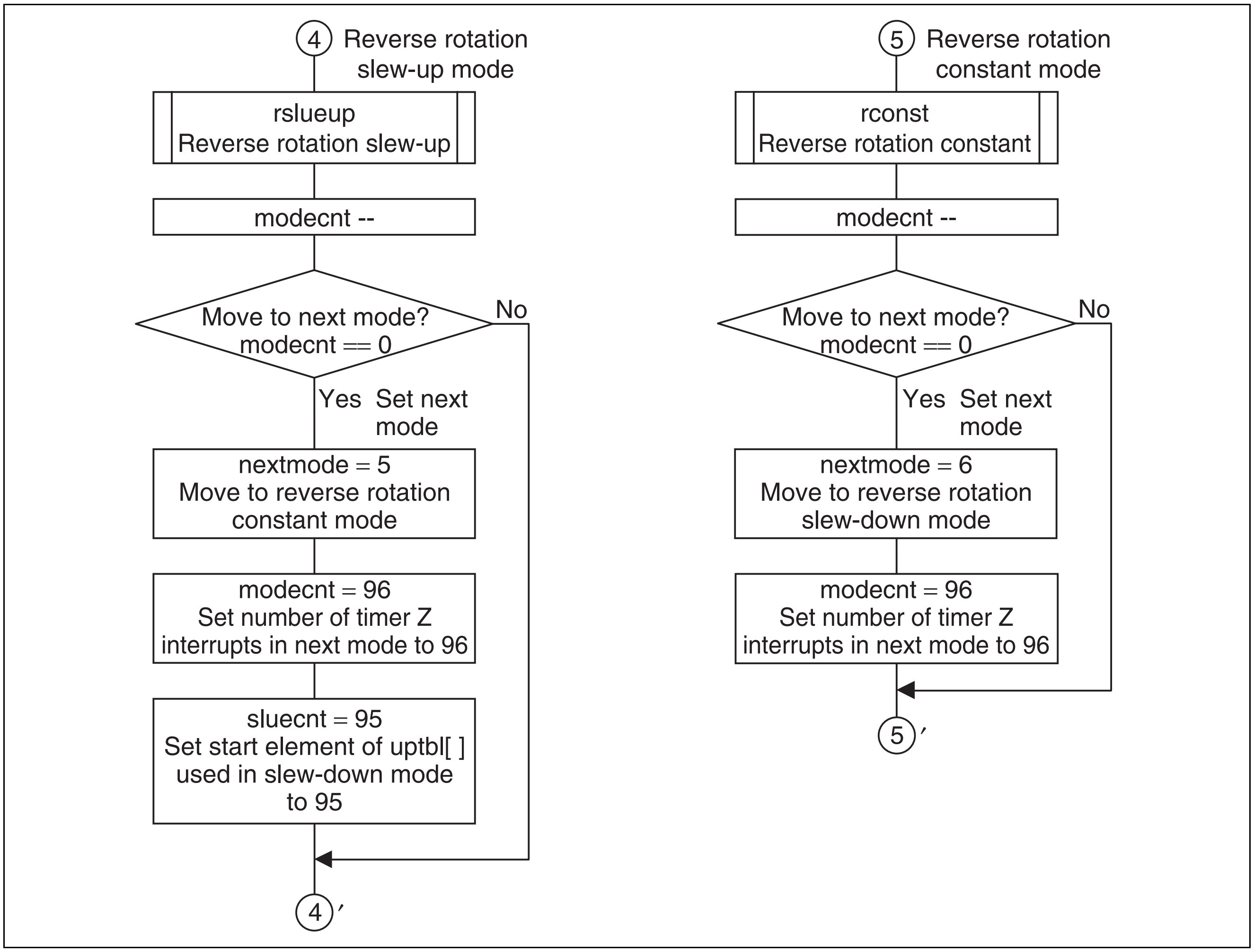
5.1 Main routine

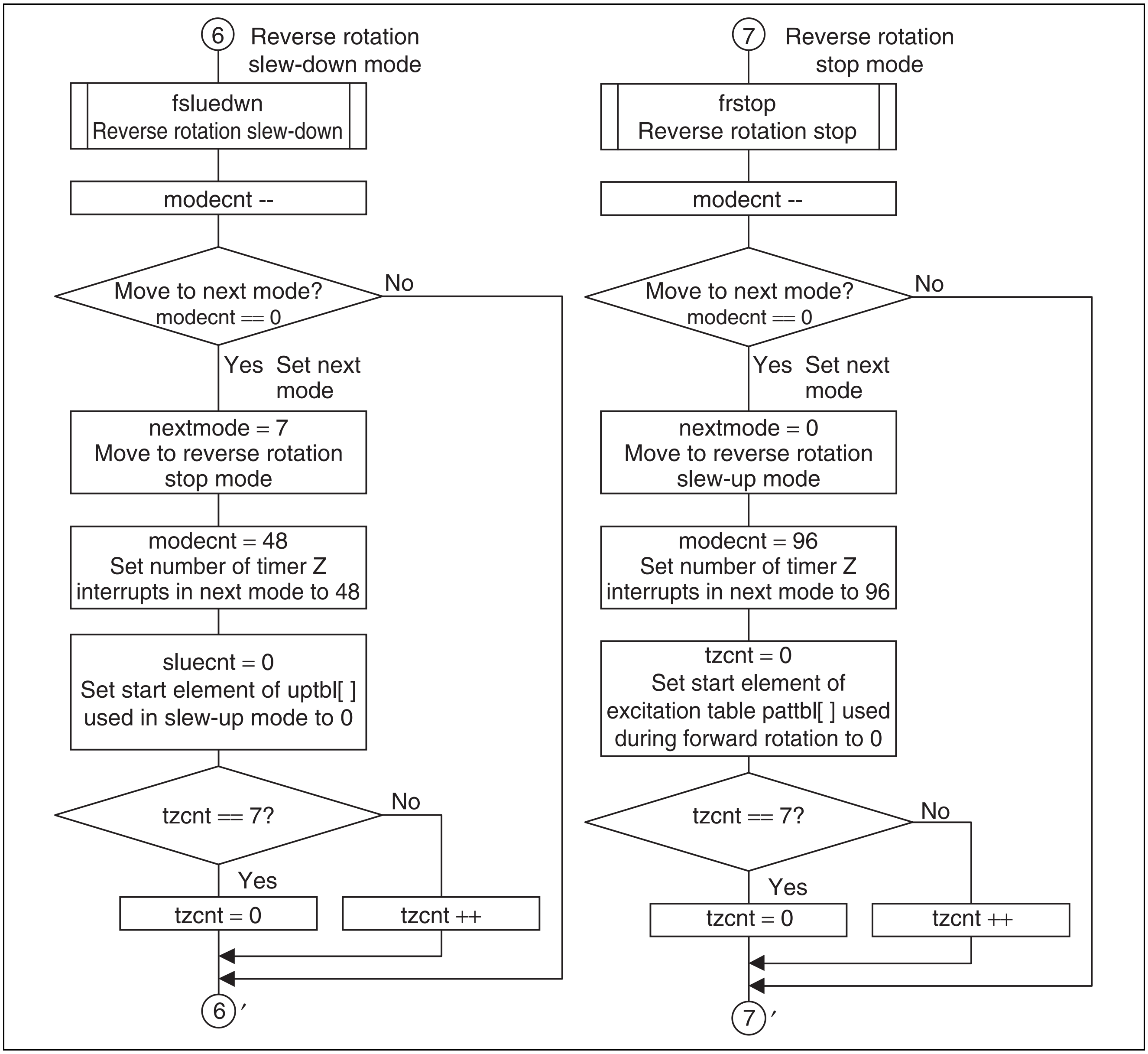


5.2 Timer Z interrupt

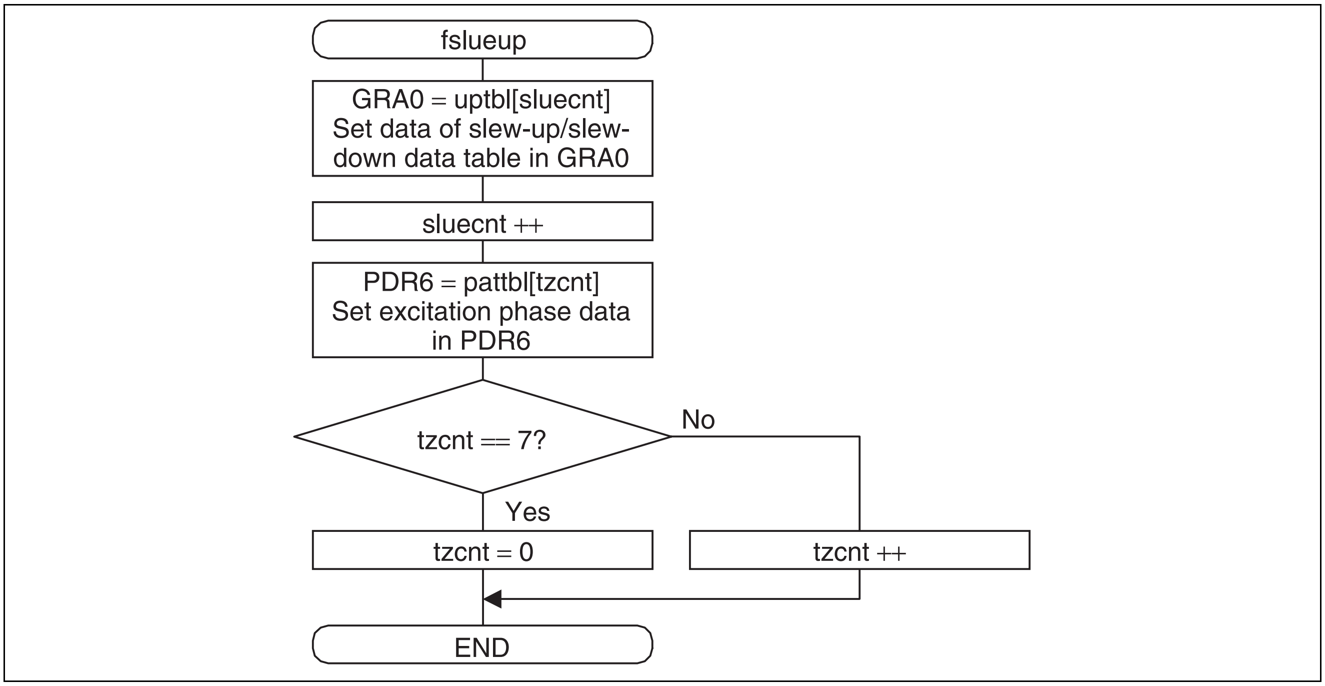



2 Forward rotation slew-down mode
fsluedwn
Forward rotation slew-down
modecnt --
Move to next mode?
modecnt == 0
No
Yes Set next mode
nextmode = 3
Move to forward rotation stop mode
modecnt = 48
Set number of timer Z interrupts in next mode to 48
sluecnt = 0
Set start element of uptbl[ ] used in slew-up mode to 0
tzcnt == 0?
No
Yes
tzcnt = 7
tzcnt --
2′
3 Forward rotation stop mode
frstop
Rotation stop
modecnt --
Move to next mode?
modecnt == 0
No
Yes Set next mode
nextmode = 4
Move to reverse rotation slew-up mode
modecnt = 96
Set number of timer Z interrupts in next mode to 96
tzcnt == 0?
No
Yes
tzcnt = 7
tzcnt --
3′

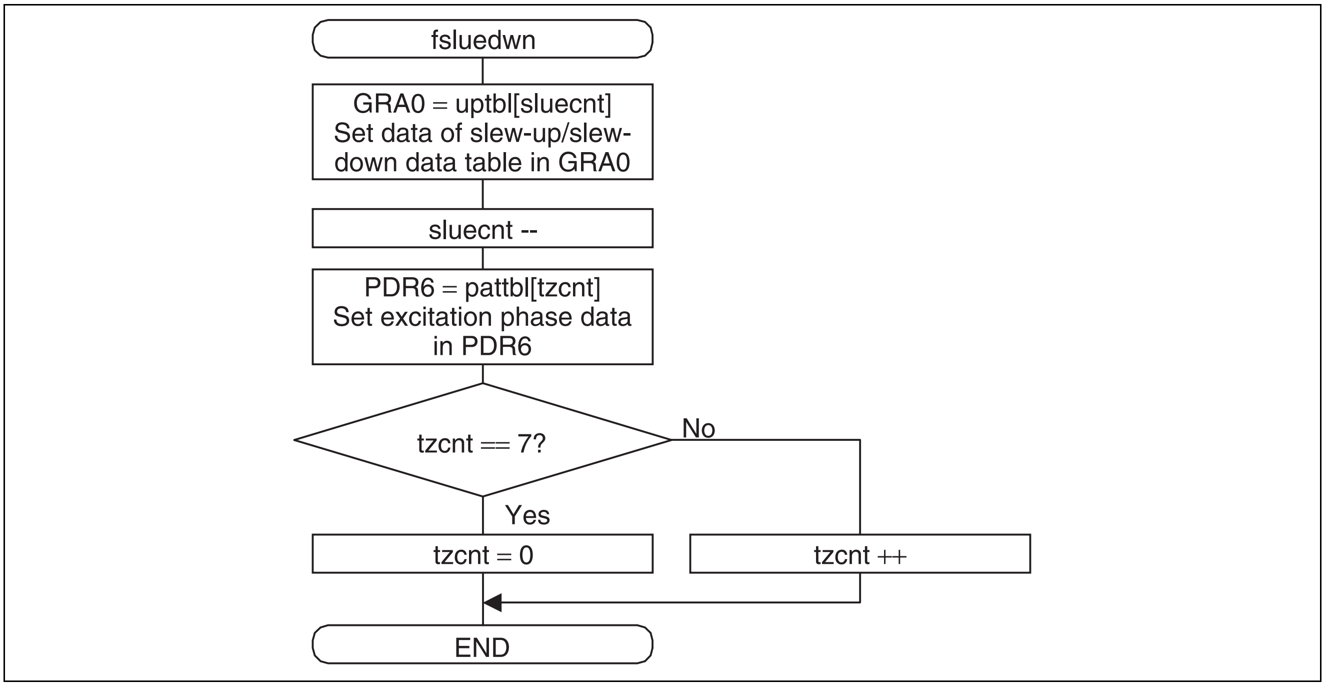

4 Reverse rotation slew-up mode
rslueup
Reverse rotation slew-up
modecnt --
Move to next mode?
modecnt == 0
No
Yes Set next mode
nextmode = 5
Move to reverse rotation constant mode
modecnt = 96
Set number of timer Z interrupts in next mode to 96
sluecnt = 95
Set start element of uptbl[ ] used in slew-down mode to 95
4′
5 Reverse rotation constant mode
rconst
Reverse rotation constant
modecnt --
Move to next mode?
modecnt == 0
No
Yes Set next mode
nextmode = 6
Move to reverse rotation slew-down mode
modecnt = 96
Set number of timer Z interrupts in next mode to 96
5′


6
Reverse rotation slew-down mode
fsluedwn
Reverse rotation slew-down
modecnt --
Move to next mode?
modecnt == 0
No
Yes
Set next mode
nextmode = 7
Move to reverse rotation stop mode
modecnt = 48
Set number of timer Z interrupts in next mode to 48
sluecnt = 0
Set start element of uptbl[ ] used in slew-up mode to 0
tzcnt == 7?
No
Yes
tzcnt = 0
tzcnt ++
6′
7
Reverse rotation stop mode
frstop
Reverse rotation stop
modecnt --
Move to next mode?
modecnt == 0
No
Yes
Set next mode
nextmode = 0
Move to reverse rotation slew-up mode
modecnt = 96
Set number of timer Z interrupts in next mode to 96
tzcnt = 0
Set start element of excitation table pattbl[ ] used during forward rotation to 0
tzcnt == 7?
No
Yes
tzcnt = 0
tzcnt ++
7′

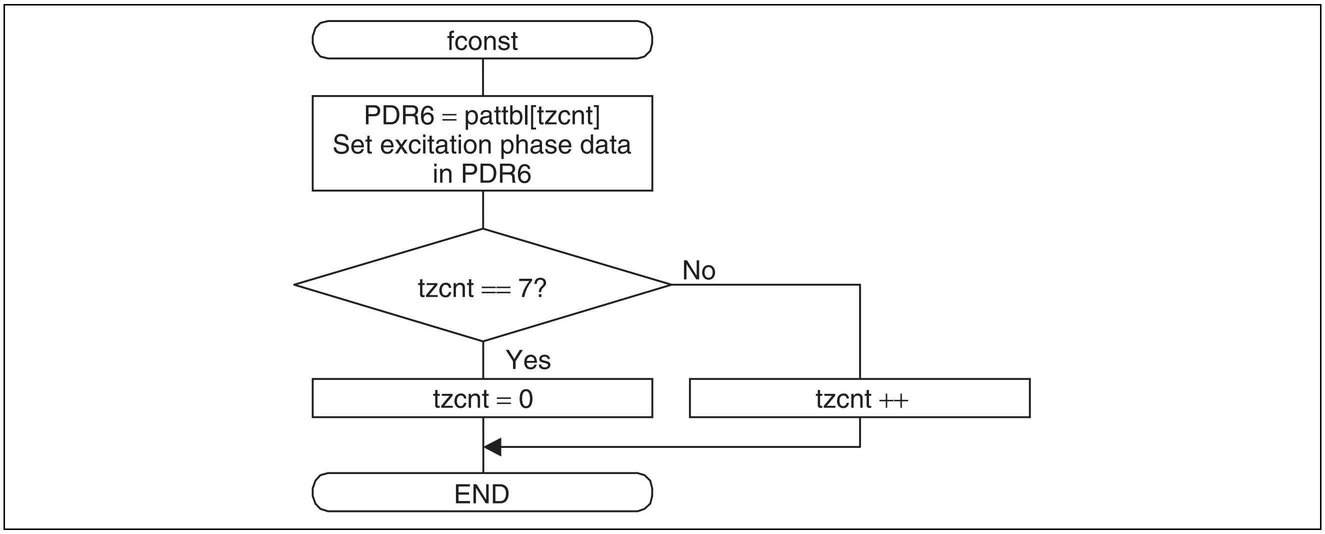
5.3 Slew-up control during forward rotation



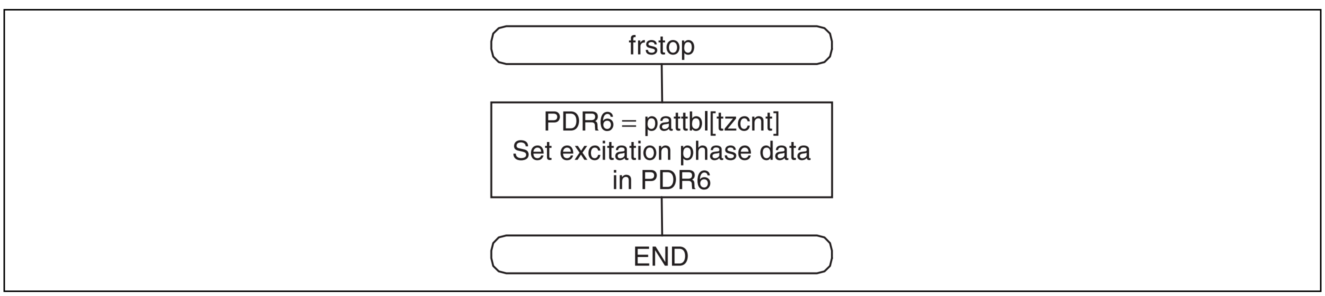
5.4 Slew-down control during forward rotation



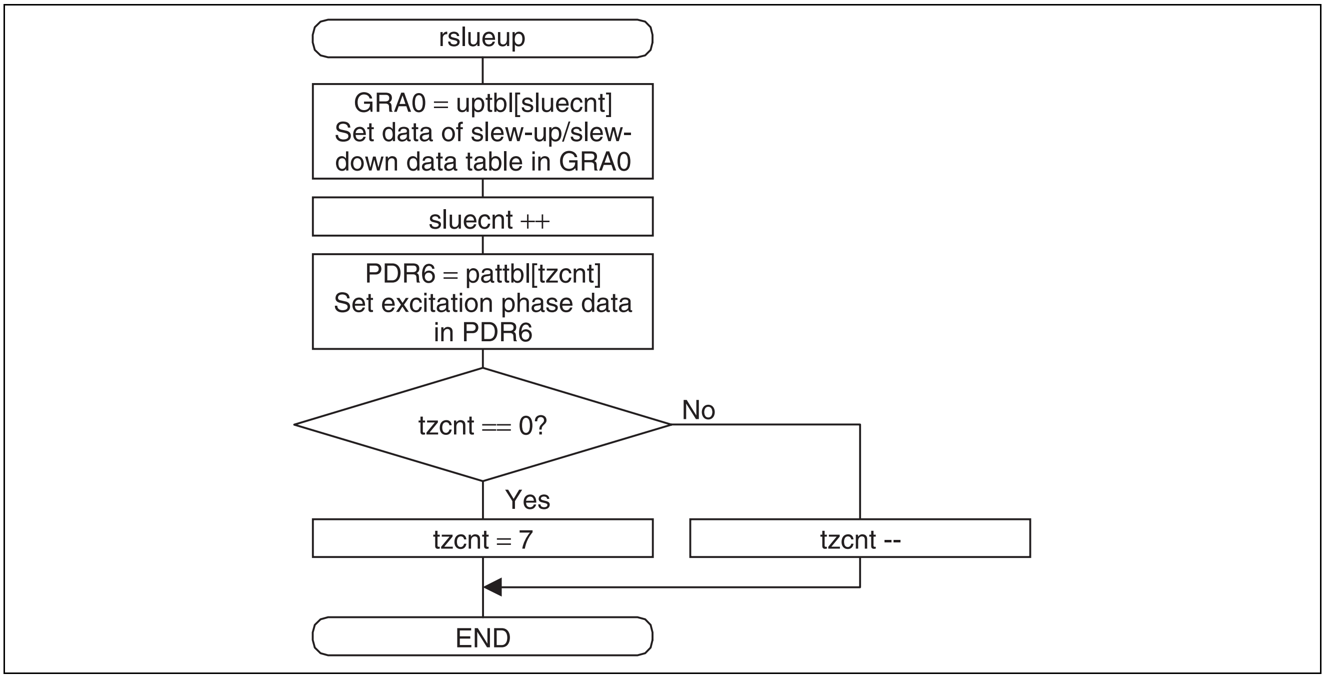
5.5 Constant control during forward rotation



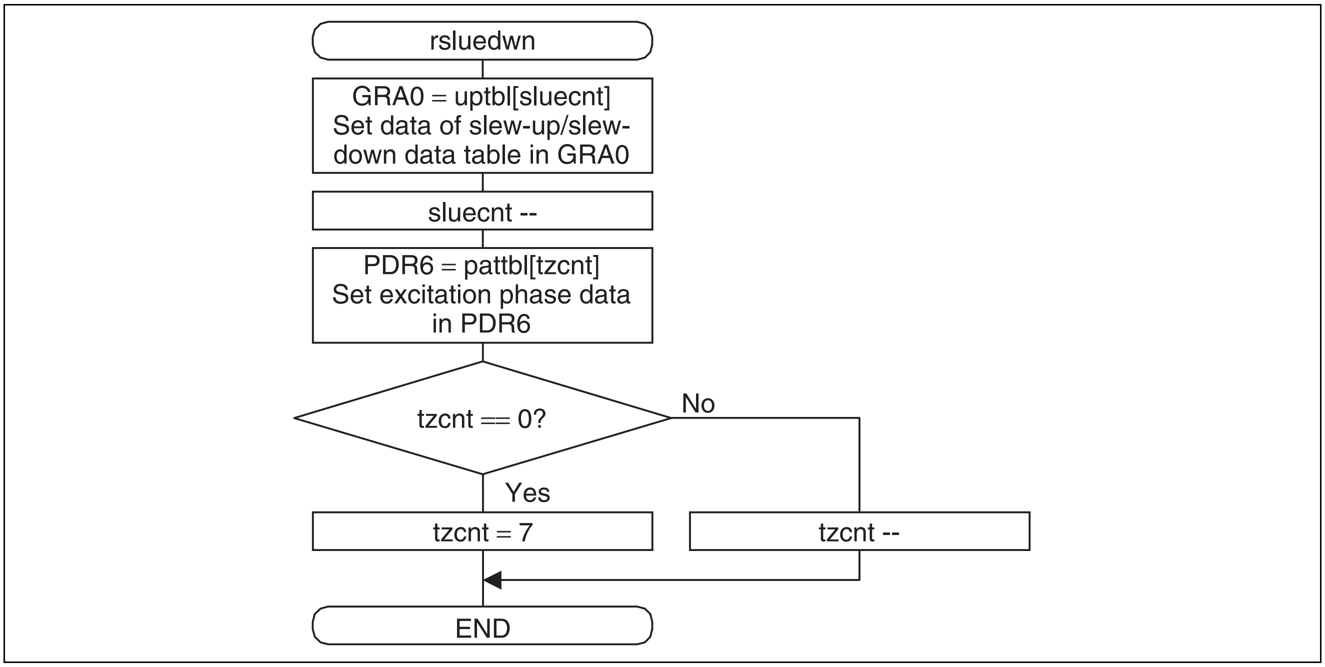
5.6 Stop control



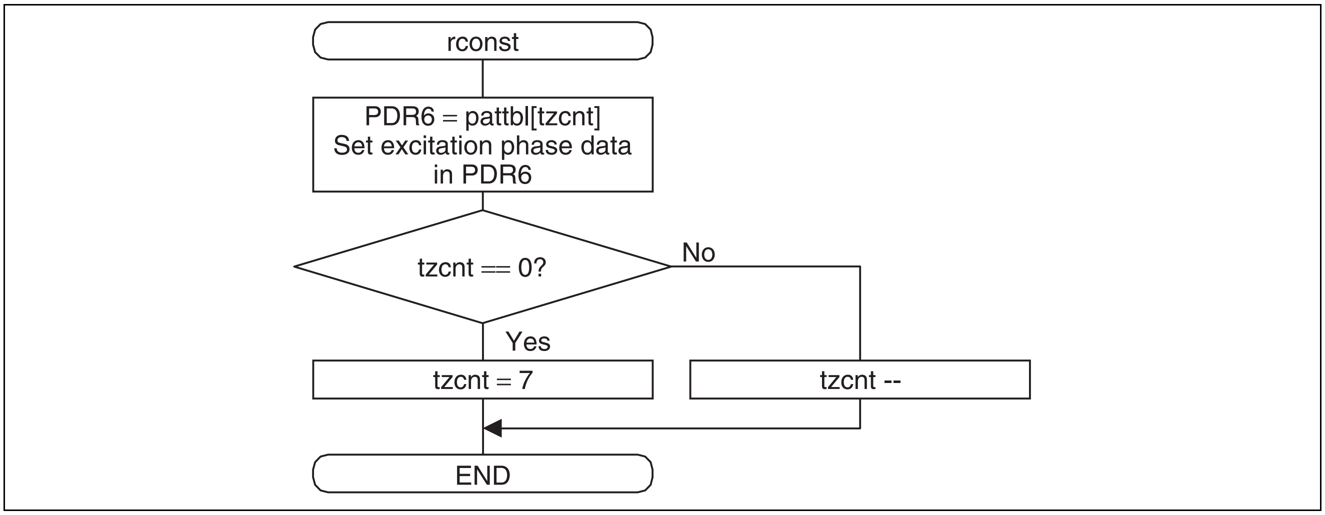
5.7 Slew-up control during reverse rotation



5.8 Slew-down control during reverse rotation



5.9 Constant control during reverse rotation



5.10 Link address specifications

Section name	Address
CV1	H'0000
CV2	H'0034
P	H'0100
C	H'0600
DOUTDT	H'0610
B	H'FB80

6. Program Listing

```
/***********************************************************/
/*                                                         */
/*  H8/300HN Series -H8/3687-                              */
/*  Application Note   */
/*                                                         */
/*  '1-2 Phase Excitation Control for a Stepping Motor     */
/*                                                         */
/*  Function                                               */
/*  : Timer Z Output Compare                               */
/*                                                         */
/*  External Clock : 16MHz                                 */
/*  Internal Clock : 16MHz                                 */
/*  Sub Clock      : 32.768kHz                             */
/*                                                         */
/***********************************************************/

#include   <machine.h>

/***********************************************************/
/*  Symbol Definition                                      */
/***********************************************************/
struct BIT {
    unsigned char  b7:1;        /* bit7 */
    unsigned cha   b6:1;        /* bit6 */
    unsigned char  b5:1;        /* bit5 */
    unsigned char  b4:1;        /* bit4 */
    unsigned char  b3:1;        /* bit3 */
    unsigned char  b2:1;        /* bit2 */
    unsigned char  b1:1;        /* bit1 */
    unsigned char  b0:1;        /* bit0 */
};

#define TCR0       *(volatile unsigned char *)0xF700        /* Timer control register_0          */
#define TIORA0     *(volatile unsigned char *)0xF701        /* Timer I/O Control Register A_0    */
#define TSR0       *(volatile unsigned char *)0xF703        /* Timer status register_0           */
#define TSR0_BIT   (*(struct BIT *)0xF703)                  /* Timer status register_0           */
#define IMFB       TSR0_BIT.b1                              /* Input Capture/Compare Match FlagB*/
#define IMFA       TSR0_BIT.b0                              /* Input Capture/Compare Match FlagA*/
#define TIER0      *(volatile unsigned char *)0xF704        /* Timer interrupt enable register0 */
#define TIER0_BIT  (*(struct BIT *)0xF704)                  /* Timer interrupt enable register0 */
#define IMIEA      TIER0_BIT.b0                             /* Input Capture/Compare Match       */
                                                            /*               Interrupt Enable A */
#define TCNT0      *(volatile unsigned short *)0xF706       /* Timer counter_0                   */
#define GRA0       *(volatile unsigned short *)0xF708       /* General register A_0              */
#define TSTR       *(volatile unsigned char *)0xF720        /* Timer start register              */
#define TMDR       *(volatile unsigned char *)0xF721        /* Timer mode register               */
#define TFCR       *(volatile unsigned char *)0xF723        /* Timer function control register  */
#define TOER       *(volatile unsigned char *)0xF724        /* Timer output master enable        */
                                                            /*                         register */
#define TOCR       *(volatile unsigned char *)0xF725        /* Timer output control register    */
#define PDR6       *(volatile unsigned char *)0xFFD9        /* Port Data Register 6              */
#define PCR6       *(volatile unsigned char *)0xFFE9        /* Port Control Register 6           */
```

```
#pragma interrupt  (tz0int)
/************************************************************/
/*  Function define                                         */
/************************************************************/
void main ( void );
void tz0int ( void );
void fslueup ( void );
void fsluedwn ( void );
void fconst ( void );
void frstop ( void );
void rslueup ( void );
void rsluedwn ( void );
void rconst ( void );

/************************************************************/
/*  RAM define                                              */
/************************************************************/
unsigned char   tzcnt,sluecnt,nextmode;
unsigned short  modecnt;

/************************************************************/
/*  Data table                                              */
/************************************************************/
#pragma section OUTDT
unsigned char   pattbl[8] = {                       /* Stepping Motor Output Pattern Table */
    0x08,0x0C,0x04,0x06,0x02,0x03,0x01,0x09,
};

unsigned short  uptbl[96] = {                       /* Stepping Motor Output Pattern Table */
    0xFFFF,0xF000,0xE000,0xD500,0xCE40,0xC738,0xC030,0xB928,0xB220,0xAB18,
    0xA410,0x9DD0,0x9790,0x9150,0x8CA0,0x87F0,0x8340,0x8020,0x7D00,0x7AA8,
    0x7850,0x75F8,0x74BD,0x7148,0x6FB8,0x6E28,0x6C98,0x6B08,0x6978,0x67E8,
    0x66BC,0x6590,0x6464,0x6338,0x620C,0x6144,0x607C,0x5FB4,0x5EEC,0x5DCA,
    0x5CA8,0x5B86,0x5A64,0x5942,0x5820,0x56FE,0x55DC,0x54BA,0x5398,0x5276,
    0x5154,0x5032,0x4F10,0x4DEE,0x4CCC,0x4BAA,0x4A88,0x4966,0x4844,0x4722,
    0x4600,0x44DE,0x43BC,0x429A,0x4178,0x4056,0x3F34,0x3E12,0x3CF0,0x3BC4,
    0x3A34,0x3890,0x373C,0x35E8,0x3494,0x33D6,0x3318,0x325A,0x319C,0x30DE,
    0x3064,0x2FEA,0x2F70,0x2EF6,0x2E7C,0x2E02,0x2D9C,0x2D36,0x2CD0,0x2C6A,
    0x2C04,0x2B9E,0x2B38,0x2AD2,0x2A6C,0x2A00,
};

/************************************************************/
/*  Vector Address                                          */
/************************************************************/
#pragma section     V1                              /* VECTOR SECTION SET          */
void (*const VEC_TBL1[])(void) = {
    main                                            /* 00 Reset                    */
};

#pragma section     V2                              /* VECTOR SECTION SET          */
void (*const VEC_TBL2[])(void) = {
    tz0int                                          /* 34 Timer Z0 Interrupt       */
};
```

```
#pragma entry main(sp=0xFF80)
#pragma section                                             /* P                                 */
/************************************************************/
/*  Main Program                                            */
/************************************************************/
void main ( void )
{
    unsigned char tmp;

    set_ccr(0x80);                                          /* Initialize CCR/Interrupt Disable */

    TSTR = 0xFC;                                            /* TCNT0 count stop                 */
    tzcnt = 0;                                              /* Output Pattern table counter set */
    sluecnt = 0;                                            /* Slue Up/Down table counter set   */
    nextmode = 0;
    modecnt = 95;                                           /* Motor Slue mode countset "95"    */

    TMDR = 0x0E;                                            /* TCNT0,TCNT1 Single Mode          */
    TFCR = 0x80;                                            /* Chanel 0,1 is Normal Mode        */
    TOER = 0xFF;                                            /* FTIOA0 Output Disable            */
    TOCR = 0x00;                                            /* FTIOA0 initial outputs is 0      */
    TCR0 = 0x22;                                            /* Rising edge, phi/4 Clock count   */
    TIORA0 = 0x88;                                          /* FTIOA0 Toggle Output             */
    TIER0 = 0xE1;                                           /* IMFA Interrupt Enable            */

    tmp = TSR0;
    TSR0 = 0xE0;                                            /* Interrupt Flag Clear             */
    GRA0 = uptbl[sluecnt];                                  /* Set GRA0                         */
    sluecnt++;

    TCNT0 = 0x0000;                                         /* Set TCNT0                        */
    PCR6 |= 0x0F;                                           /* Port8 Output                     */
    PDR6 = pattbl[tzcnt];                                   /* PDR6 Set Output Pattern          */
    tzcnt++;
    TSTR = 0xFD;                                            /* TCNT0 count start                */

    set_imask_ccr(0);                                       /* Interrupt Enable                 */

    while(1);
}

/************************************************************/
/*  Timer Z0 Interrupt                                      */
/************************************************************/
void tz0int ( void )
{
    unsigned char tmp;

    switch(nextmode){
        case 0:
            fslueup();                                      /* Forward Slue Up                  */
            modecnt--;
            if(modecnt == 0){                               /* Next mode?                       */
```

```
            nextmode = 1;                       /* nextmode = 1 Constant Speed      */
            modecnt = 96;                       /* Next mode countset "96"          */
            sluecnt = 95;                       /* Slue Up/Down table counter set   */
        }
        break;

    case 1:
        fconst();                               /* Constant Speed                   */
        modecnt--;
        if(modecnt == 0){                       /* Nextmode?                        */
            nextmode = 2;                       /* nextmode = 2 Forward Slue Down   */
            modecnt = 96;                       /* Nextmode countset "96"           */
        }
        break;

    case 2:
        fsluedwn();                             /* Forward Slue Down                */
        modecnt--;
        if(modecnt == 0){                       /* Next mode?                       */
            nextmode = 3;                       /* nextmode = 3 Slue Stop           */
            modecnt = 48                        /* Next mode countset "48"          */
            sluecnt = 0;                        /* Slue Up/Down table counter set   */
            if(tzcnt==0)
                tzcnt = 7;
            else
                tzcnt--;
        }
        break;

    case 3:
        frstop();                               /* Slue Stop                        */
        modecnt--;
        if(modecnt == 0){                       /* Next mode?                       */
            nextmode = 4;                       /* nextmode = 4 Reverse Slue Up     */
            modecnt = 96;                       /* Next mode countset "96"          */
            if(tzcnt==0)
                tzcnt = 7;
            else
                tzcnt--;
        }
        break;

    case 4:
        rslueup();                              /* Reverse Slue Up                  */
        modecnt--;
        if(modecnt == 0){                       /* Next mode?                       */
            nextmode = 5;                       /* nextmode = 5 Constant Speed      */
            modecnt = 96;                       /* Next mode countset "96"          */
            sluecnt = 95;                       /* Slue Up/Down table counter set   */
        }
        break;

    case 5:
        rconst();                               /* Constant Speed                   */
```

```
            modecnt--;
            if(modecnt == 0){                               /* Next mode?                       */
                nextmode = 6;                               /* nextmode = 6 Reverse Slue Down   */
                modecnt = 96;                               /* Next mode countset "96"          */
            }
            break;

        case 6:
            rsluedwn();                                     /* Reverse Slue Down                */
            modecnt--;
            if(modecnt == 0){                               /* Next mode?                       */
                nextmode = 7;                               /* nextmode = 7 Slue Stop           */
                modecnt = 48;                               /* Next mode countset "48"          */
                sluecnt = 0;                                /* Slue Up/Down table counter set   */
                if(tzcnt==7)
                    tzcnt = 0;
                else
                    tzcnt++;
            }
            break;

        case 7:
            frstop();                                       /* Slue Stop                        */
            modecnt--;
            if(modecnt == 0){                               /* Next mode?                       */
                nextmode = 0;                               /* nextmode = 0 Forward Slue Up     */
                modecnt = 96;                               /* Next mode countset "96"          */
                if(tzcnt==7)
                    tzcnt = 0;
                else
                    tzcnt++;
            }
            break;
    }

    tmp = TSR0;
    TSR0 = 0xE0;                                            /* Interrupt Flag Clear             */
}

/**************************************************************/
/*  Forward Slue Up    */
/**************************************************************/
void fslueup ( void )
{
    GRA0 = uptbl[sluecnt];                                  /* GRA Set Slue Up/Down table       */
    sluecnt++;

    PDR6 = pattbl[tzcnt];                                   /* PDR6 Set Output Pattern          */
    if(tzcnt==7)
        tzcnt = 0;
    else
        tzcnt++;
}
```

```
/************************************************************/
/*  Forward Slue Down                                       */
/************************************************************/
void fsluedwn ( void )
{
    GRA0 = uptbl[sluecnt];                               /* GRA Set Slue Up/Down table       */
    sluecnt--;

    PDR6 = pattbl[tzcnt];                                /* PDR6 Set Output Pattern          */
    if(tzcnt==7)
        tzcnt = 0;
    else
        tzcnt++;
}

/************************************************************/
/*  Forward Constant Speed                                  */
/************************************************************/
void fconst ( void )
{
    PDR6 = pattbl[tzcnt];                                /* PDR6 Set Output Pattern          */
    if(tzcnt==7)
        tzcnt = 0;
    else
        tzcnt++;
}

/************************************************************/
/*  Slue/Reverse Stop                                       */
/************************************************************/
void frstop ( void )
{
    PDR6 = pattbl[tzcnt];                                /* PDR6 Set Output Pattern          */
}

/************************************************************/
/*  Reverse Slue Up                                         */
/************************************************************/
void rslueup ( void )
{
    GRA0 = uptbl[sluecnt];                               /* GRA Set Slue Up/Down table       */
    sluecnt++;

    PDR6 = pattbl[tzcnt];                                /* PDR6 Set Output Pattern          */
    if(tzcnt==0)
        tzcnt = 7;
    else
        tzcnt--;
}

/************************************************************/
/*  Reverse Slue Down                                       */
/************************************************************/
void rsluedwn ( void )
```

```
{
    GRA0 = uptbl[sluecnt];                              /* GRA Set Slue Up/Down table       */
    sluecnt--;

    PDR6 = pattbl[tzcnt];                               /* PDR6 Set Output Pattern          */
    if(tzcnt==0)
        tzcnt = 7;
    else
        tzcnt--;
}

/************************************************************/
/*  Reverse Constant Speed                                  */
/************************************************************/
void rconst ( void )
{
    PDR6 = pattbl[tzcnt];                               /* PDR6 Set Output Pattern          */
    if(tzcnt==0)
        tzcnt = 7;
    else
        tzcnt--;
}
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Dec.20.03	—	First edition issued