NEC

Application Note

V853™

32-/16-bit Single-Chip Microcontrollers Software for IAR Compiler

---

μPD703003
μPD70F3003
μPD703003A
μPD70F3003A
μPD703025A
μPD70F3025A

Document No. U14467EE1V0AN00
Date Published December 1999

© NEC Corporation 1999

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

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 once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build 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 bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

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

V853, V850 Family, and EEPROM are trademarks of NEC Corporation.
Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.
UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/OPEN Company Limited.

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

License not needed: µPD70F3003, 70F3003A, 70F3025A

The customer must judge the need for license: µPD703003, 703003A, 703004A, 703025A

The information in this document is subject to change without notice.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document. NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or others.

While NEC Corporation has been making continuous effort to enhance the reliability of its semiconductor devices, the possibility of defects cannot be eliminated entirely. To minimize risks of damage or injury to persons or property arising from a defect in an NEC semiconductor device, customers must incorporate sufficient safety measures in its design, such as redundancy, fire-containment, and anti-failure features.

NEC devices are classified into the following three quality grades:

"Standard", "Special", and "Specific". The Specific quality grade applies only to devices developed based on a customer designated “quality assurance program“ for a specific application. The recommended applications of a device depend on its quality grade, as indicated below. Customers must check the quality grade of each device 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: Aircrafts, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems or medical equipment for life support, etc.

The quality grade of NEC devices is "Standard" unless otherwise specified in NEC's Data Sheets or Data Books. If customers intend to use NEC devices for applications other than those specified for Standard quality grade, they should contact an NEC sales representative in advance.

Anti-radioactive design is not implemented in this product.

M7 96.5

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC 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.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Vélizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J99.1

Major Revisions in This Edition

Page	Description
Throughout	Addition of µPD703004A, 703025A, and 70F3025A, and deletion of µPD703005A
pp.21 to 23	Modification of **Figure 1-3 (a) For µPD703003 and 70F3003**, and addition of **Figure 1-3 (b) For µPD703003A, 703004A, and 70F3003A**, and **Figure 1-3(c) For µPD703025A, 70F3025A**.
p.98	Addition of **Note** in **3.4.1 Asynchronous serial interface (UART0, UART1)**
p.135	Modification of **Figure 3-55 Block Diagram of A/D Converter**
p.182	Modification of **Figure 3-84 Clock Control Register (CKC)**

The mark ⋆ shows major revised points.

PREFACE

Target reader This manual is written for users who wish to understand the functions of the V853 (μPD703003, 70F3003, 703003A, 70F3003A, 703004A, 703025A, 70F3025A) in order to design application systems in which it is used.

Purpose This manual is intended to help the user understand by providing examples of programs in which the V853 actually is used.

Organization The contents of this application note are organized as follows.

- Overview of the V853
- Functions of the V853

How to read this manual It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, and the C language.

For V853 hardware functions
→ Refer to **V853 User's Manual Hardware**.

For V853 command functions
→ Refer to **V850 Family™ User's Manual Architecture**.

For V853 electrical specifications
→ Refer to the separate data sheet.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active row representation :	$\overline{xxx}$ (overscore over pin or signal name) or /xxx ("/" before signal name)
Memory map address:	High order at high stage and low order at low stage
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information

Numerical representation: Binary ... xxxx or xxxB
Decimal ... xxxx
Hexadecimal ... xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)
K (kilo): $2^{10} = 1024$
M (mega): $2^{20} = 1024^2$
G (giga): $2^{30} = 1024^3$

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

Documents related to V853

Document title	Document number
V850 Family User's Manual Architecture	U10243E
V850 Family Instruction Application Table	U10229J Note
µPD703003 Data Sheet	U12261E
µPD703003A, 703004A, 703025A Data Sheet	U13188E
µPD70F3003 Data Sheet	U12036E
µPD70F3003A, 70F3025A Data Sheet	U13189E
V853 User's Manual Hardware	U10913E
V853 Application Note Hardware	U12619E
V853 Application Note Software	This manual

Note: Japanese version

Documents related to development tools (User's manuals)

Document title	Document number
IE-703002-MC (In-circuit Emulator)	U11595E
IE-703003-MC-EM1 (In-circuit Emulator Option Board)	U11596E

Table of Contents

List of Figures

List of Tables

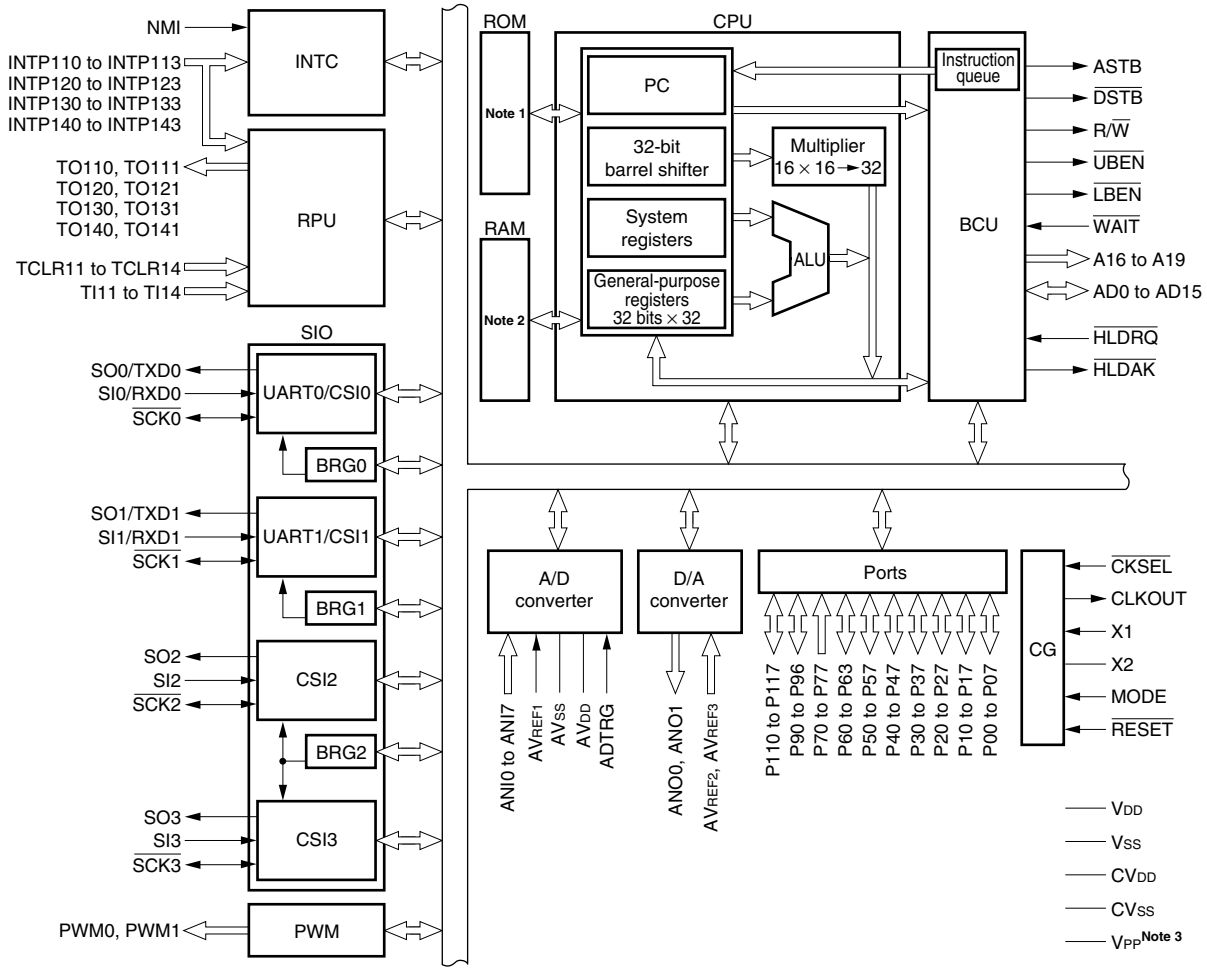
CHAPTER 1 OVERVIEW OF V853

This chapter provides an overview of the V853.

1.1 Configuration

Figure 1-1 shows the internal blocks of the V853.

Figure 1-1: Internal Block Diagram



★ **Notes:1.** μPD703003, 703003A: 128 Kbytes (mask ROM)
μPD70F3003, 70F3003A: 128 Kbytes (flash memory)
μPD703004A: 96 Kbytes (mask ROM)
μPD703025A: 256 Kbytes (mask ROM)
μPD70F3025A: 256 Kbytes (flash memory)

★ **2.** μPD703003, 703003A, 703004A, 70F3003, 70F3003A: 4 Kbytes
μPD703025A, 70F3025A: 8 Kbytes

★ **3.** μPD70F3003, 70F3003A, 70F3025A

1.2 Functions

Table 1-1 shows a list of the functions of the V853.

Table 1-1: List of Functions

Item		Function
CPU performance		38 MIPS (at 33-MHz operation)
★ Internal memory	Internal ROM	Flash memory: 256 Kbytes (μPD70F3025A) Mask ROM: 256 Kbytes (μPD703025A) Flash memory: 128 Kbytes (μPD70F3003, 70F3003A) Mask ROM: 128 Kbytes (μPD703003, 703003A) Mask ROM: 96 Kbytes (μPD703004A)
★	Internal RAM	8 Kbytes (μPD703025A, 70F3025A) 4 Kbytes (μPD703003, 703003A, 703004A, 70F3003, 70F3003A)
Interrupts		External interrupts: 17 (including NMI) Internal interrupts: 31 sources Exceptions: 1 source Programmable priorities: 8 levels
Timer/counter		16-bit timer/event counter: 4 ch 16-bit interval timer: 1 ch
Serial interface		Asynchronous serial interface (UART) Clock synchronous serial interface (CSI) UART/CSI: 2 ch CSI: 2 ch Dedicated baud rate generator: 3 ch
PWM		2 ch (8/9/10/12-bit resolution can be selected)
A/D converter		8 ch (10-bit resolution), successive approximation method
D/A converter		2 ch (8-bit resolution), R-2R method
I/O ports		75 (8 inputs, 67 I/Os, both also used as control pins)
Power saving		HALT/IDLE/STOP modes

1.3 Register Set

Table 1-2 shows a summary of the V853 program register set. The bit width of all program registers is 32 bits.

Table 1-2: Summary of Register Set

Register name	Use	Details of use
r0	Zero register	Always holds 0
r1	Assembler reserved register	Used as a working register for 32-bit immediate data creation
r2	Interrupt stack pointer	Used as the interrupt handler stack pointer
r3	Stack pointer	Used when generating a stack frame at the time of a function call
r4	Global pointer	Used when accessing global variables in a data area
r5	Text pointer	Used as a register to point to the head of a text area
r6 to r29	General-purpose registers	Registers for address/data variables
r30	Element pointer	Used as a base pointer when accessing memory
r31	Link pointer	Used when the compiler makes a function call
PC	Program counter	Holds the address of the instruction during the program is executing

r0 through r31 are provided as general-purpose registers. However, because the methods of using r0 and r30 are determined implicitly (as zero register and element pointer, respectively), be careful when using them for other purposes.

In addition, r1 through r5 and r31 are used implicitly by the V850 family assembler and C compiler. If these registers are used for purposes other than those shown in Table 1-2, handle them appropriately so that the register contents are not lost.

Bits 31 through 24 and bit 0 of the program counter (PC) are fixed at 0. Thus, the values that can be represented by the PC are 00000000H through 00FFFFFEH, which cannot be used to access odd-numbered locations.

Example 1 Reading data at an odd-numbered address

Let r10 = 00000111H

```
ld.w [r10], r11
```

The address that is read by the instruction above is 00000110H, not 00000111H (the lowest bit is dropped).

Example 2 Example of PC wrap-around

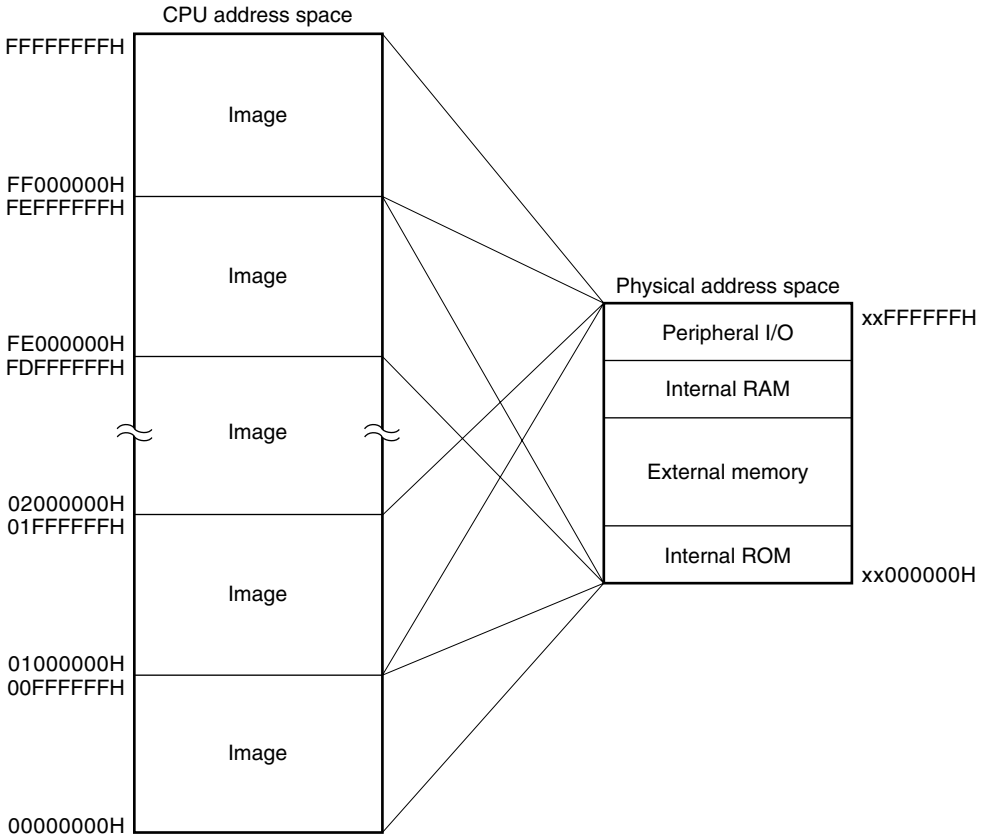
PC: 00FFFFFEH + 4 → PC: 00000002H (the carry to bit 24 is ignored)

1.4 Address Space

The V853 CPU has a 32-bit architecture that supports a linear address space of up to 4 Gbytes as a data area. It supports a linear address space of up to 16 Mbytes as a program area.

The physical address space of the V853 is 16 Mbytes. In a 4-Gbyte CPU address space, 256 16-Mbyte spaces appear as images.

Figure 1-2: Address Space Images



1.5 Memory Map

Figure 1-3 shows the V853 memory map. The memory map differs depending on the mode. Each mode is set using the MODE pin and memory expansion mode register (MM) at the time of a reset.

For the µPD703003 and 70F3003, a MODE pin value of 0 sets ROM-less mode and a value of 1 sets single-chip mode.

★ For the µPD703003A, 70F3003A, 703004A, 703025A, and 70F3025A, single-chip mode is fixed regardless of the status of the MODE pin.

The external expansion mode of single-chip mode is specified by setting the memory expansion mode register (MM). Refer to **V853 User's Manual Hardware** regarding the memory expansion mode register.

Figure 1-3: Memory Map (1/3)

★

(a) For µPD703003 and 70F3003

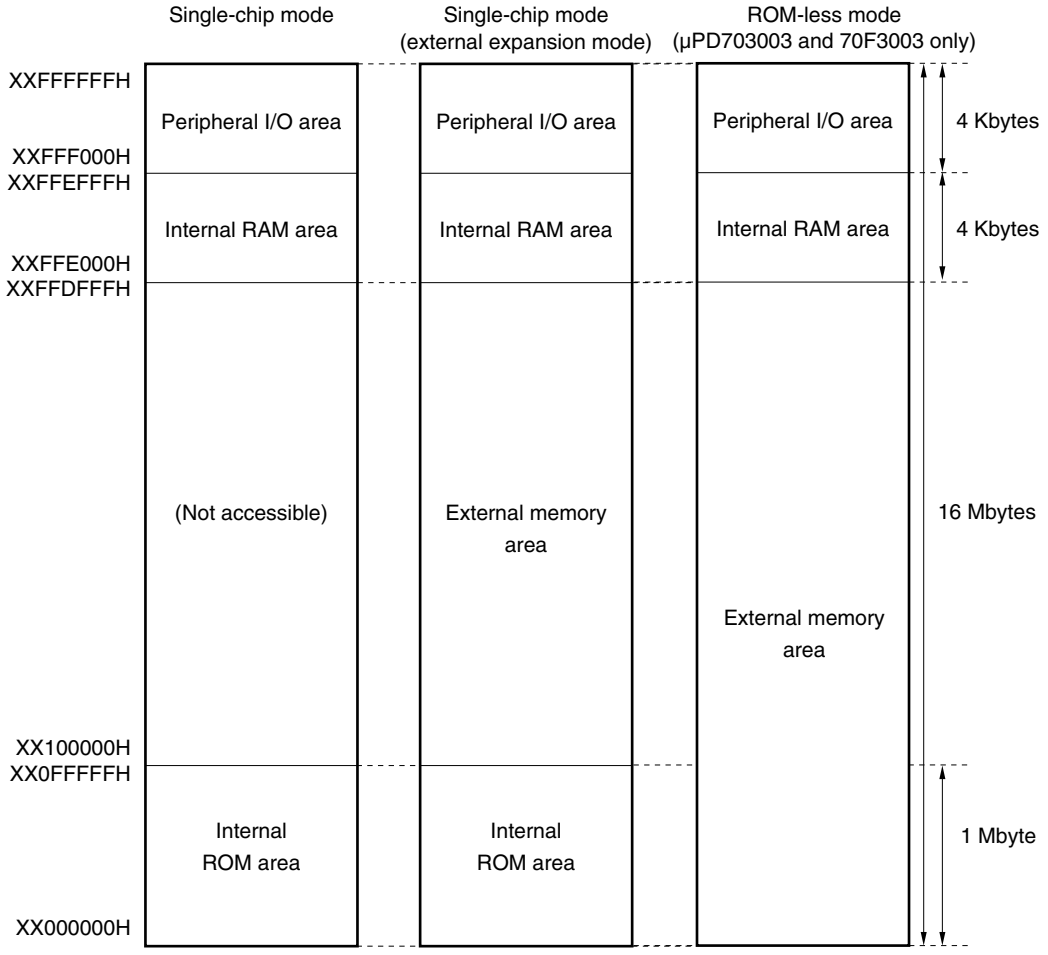


Figure 1-3: Memory Map (2/3)

★ *(b) For µPD703003A, 703004A, and 70F3003A*

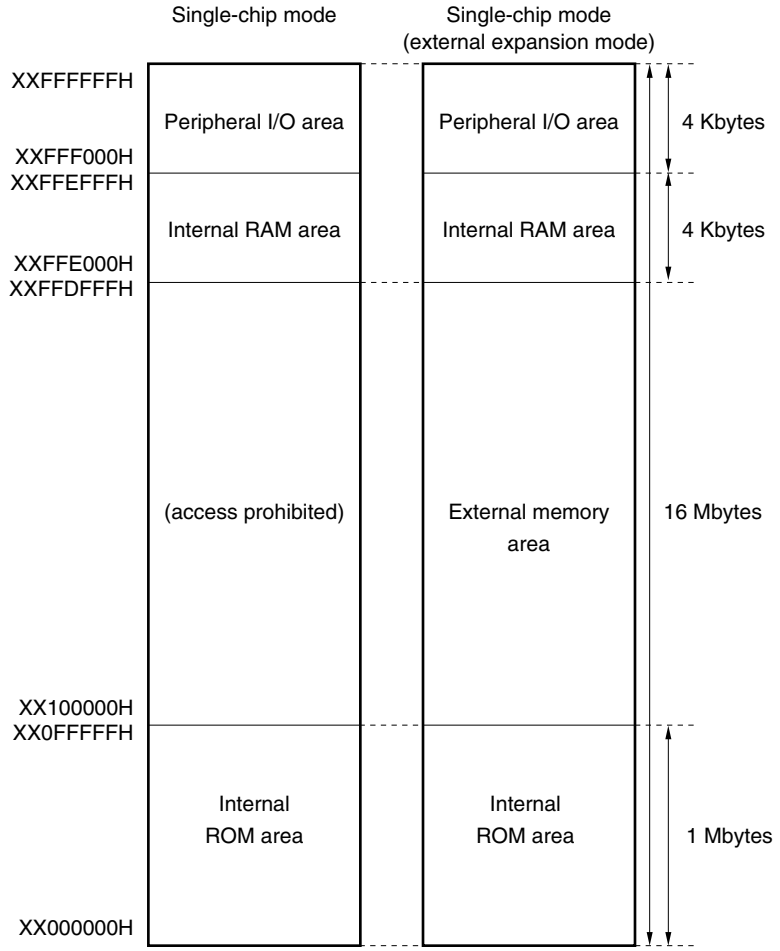
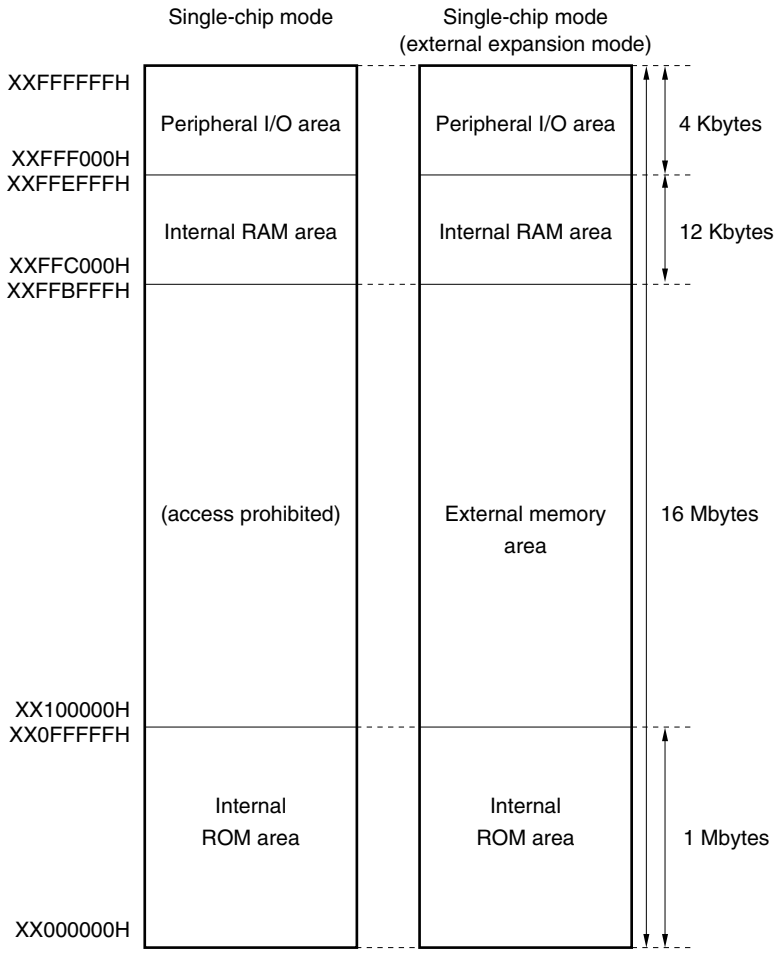


Figure 1-3: Memory Map (3/3)

★ ***(c) For µPD703025A and 70F3025A***



[MEMO]

CHAPTER 2 OVERVIEW OF IAR V850 C COMPILER

This chapter provides an overview of the IAR V850 C compiler package. It has the following features:

- Conformance to the ISO/ANSI standard for a free-standing environment
- Conformance to Embedded C++ which is defined by the Embedded C++ Technical Committee
- IEEE-compatible floating-point arithmetic

2.1 How General-Purpose Registers are used by IAR V850 C compiler

Which registers are used by the IAR V850 C compiler depends upon the number of locked registers. Normally the following registers are available to the compiler:

R1, R5 – R29

Table 2-1: Registers used by IAR V850 C compiler

Number of locked registers	Free registers (used by the compiler)	Locked registers (can be used by user or OS)
0	R1, R5 - R29	R4
2	R1, R5 - R17, R20 - R29	R4, R18 - R19
6	R1, R5 - R16, R23 - R29	R4, R17 - R22
10	R1, R5 - R14, R25 - R29	R4, R15 - R24

If enabled, the compiler can hold 255 (0xff) and 65535 (0xffff) in registers R18 and R19. If this is enabled the compiler can generate more effective code. In order to use this feature at least two registers must be locked.

2.2 Notes on Source Code Specification

Notes on specifying source code to be compiled by the IAR V850 C compiler are shown below. Refer to **IAR V850 C/EC++ COMPILER Programming Guide** for details of each item.

2.2.1 Specifying assembler instructions

The ***asm*** statement generates in-line assembly code and can be used anywhere a statement can be used within a function and anywhere a declaration can be used outside of a function.

There are two spellings, **__asm** and **__ASM:**

```
__asm (“any_line_of_assembly_language);
or
__ASM (“any_line_of_assembly_language);
```

2.2.2 Specifying inline expansion

When optimizing for speed with level 7-9, the compiler heuristically decides, which functions will be inlined. Inlining can be disabled with compiler switch –no_inlining.

2.2.3 Specifying interrupt enabling/disabling and interrupt handling

The IAR V850 C compiler provides inline functions for enabling and disabling interrupts, because ***asm*** statements are not portable:

```
void __DI (void);   or    __disable_interrupt(void);
void __EI (void);    or   __enable_interrupt (void);
```

Example →

```
(Interrupt enabled state)
        :
__DI();
        :
(Interrupt disabled state)
        :
__EI();
        :
(Interrupt enabled state)
        :
```

Interrupt functions may be declared to be an interrupt function by putting the **__interrupt** keyword before the function definition. The compiler will generate code for this function that will save all the registers the function uses, including the registers that are normally destroyable across function calls.

Example →

```
__interrupt void handle_clock_interrupt (void)
{
    clock_ticks = clock_ticks + 1;
}
```

It is possible to inhibit the code generation for save and restore of special registers by adding the __flat keyword to a function definition:

Example →

```
__flat __interrupt void handle_clock_interrupt (void)
{
    clock_ticks = clock_ticks + 1;
}
```

Note: You should not use this keyword if nested interrupts are enabled for the interrupt function.

```
#pragma type_attribut=__interrupt
#pragma object_attribute= __flat
```

Putting this **#pragma** lines before a function definition is equivalent to declaring the function with the **__interrupt** and **__flat** keywords.

Example → #pragma type_attribut=__interrupt
#pragma object_attribute= __flat
void handle_clock_interrupt (void)
{
clock_ticks = clock_ticks + 1;
}

#pragma vector

The purpose of **#pragma vector=***integer_constant* is to establish interrupt vectors. The compiler arranges for the following interrupt function an interrupt vector to be placed in memory at the address specified by *integer_constant* using a **.org** directive to the assembler.

Example → #pragma vector=0x1c0;
__flat __interrupt void handle_clock_interrupt (void)
{
clock_ticks = clock_ticks + 1;
}

2.3 Calling Functions

This section describes the calling of functions in the IAR V850 C compiler. For a call to a procedure, subroutine or function a **jarl routine,lp** instruction is used, which saves the return address in **r31 (lp)**. The return uses a **jmp [lp]** function.

The following registers are used for small (32-bit or less) arguments:
R1, R5-R8, [R9], R10-R14, and, depending on the register locking, R15-R19.

For large (64 bit) objects that are passed in registers (currently only doubles) the following register pairs are available:

(R1, R5), (R6, R7), [(R8, R9)], (R10, R11), (R12, R13), and, depending on the register locking, (R14, R15) to (R18, R19). The least significant bits are passed in the lower register for the pairs (R1, R5), (R6, R7) and (R8, R9) and in the higher register for the other register pairs.

Note that the registers in brackets [] are not available when the V850 CPU variant (-v0) has been selected. They are also not available in the large code model for the V850E.

For prototyped functions, scalar objects are passed in registers. The mechanism used to select the registers is to find the first available in the list. Should no registers be available, or if the parameter is of non-scalar type (e.g. a struct) the object is passed on the stack. The return value from a function is either R1 or the register pair (R1, R5) depending on the size of the object.

For example,

int test_function (int first, struct a_struct_name second, double third, int fourth)

In this case *first* will be passed in R1, *second* will be passed on the stack since it is a non-scalar type. *third* will be passed in the pair (R6, R7), and finally *fourth* will use R5.

2.4 How to Call Programs Between C Language and Assembly Language

This section describes program calls between C language programs and assembly language programs in the IAR V850 C compiler.

2.4.1 Calling an assembly language program from a C language program

Notes on calling an assembly language program from a C language program when the programs were created with IAR V850 C compiler are shown below.

(1) Identifiers

When an external name is specified, the IAR V850 C compiler outputs it with no extra symbols to the assembler. Therefore, when calling an assembly language program from a C language program, use the same name as on the Assembler language program side.

(2) Stack frames

The IAR V850 C compiler outputs code that assumes that the stack pointer (**sp**) always points to the lowest stack frame address. Therefore, if the assembly language program changes the contents of an area at a lower address than the address to which **sp** points, subsequent operations in the C programming language program are not guaranteed. In addition, if **sp** is used in the assembly language program, be sure the values of register variable registers before and after calling it are held.

(3) Arguments to assembly language program

Refer to section 2.3 for explanations of how the IAR V850 C compiler passes arguments when calling a function.

(4) Return value from assembly language program

The IAR V850 C compiler stores all return values except structures or unions in **r1** and **r5**, depending of the type of return value. Refer to **V850 C/EC++ Compiler Programming Guide** for details.

(5) Return address

The IAR V850 C compiler outputs code that stores the return address in the **lp** (link pointer) on a function call. Therefore, execute **jmp [lp]** in returning control to the C language program from the assembly program.

2.4.2 Calling a C language program from an assembly language program

Notes on calling a C language program from an assembly language program when the programs were created in IAR V850 C compiler are shown below.

(1) Stack frames

The IAR V850 C compiler outputs code that assumes that the stack pointer (**sp**) always points to the lowest stack frame address. Therefore, be sure that an assembly language program that calls a C language program sets **sp** so that it points to the highest address of an unused area.

(2) Work registers

The IAR V850 C compiler holds the value of register variable registers before and after calling a C program language program but it does not hold register values. However, if using the compile time options —**lock_regs (2), —lock_regs (6) or —lock_regs (10)** some registers can be reserved for the user.

(3) Arguments to C language program

Refer to section 2.3 for explanations of how the IAR V850 C compiler passes arguments when calling a function.

(4) Return value from C language program

The IAR V850 C compiler stores all return values except structures or unions in **r1** and **r5**, depending of the type of return value. Refer to **V850 C/EC++ Compiler Programming Guide** for details.

(5) Return address

The IAR V50 C compiler outputs code that assumes that the return address is stored in the **lp** (link pointer) on a function call. Therefore, be sure that the assembly language program that calls the C language function sets the return address in **lp** using the **jarl** instruction.

2.5 Startup Module

Code generated by the IAR V850 C Compiler supposes that appropriate values are set in the stack pointer (**sp**), global pointer (**gp**) and element pointer (**ep**). These settings normally are done by a startup module, which is automatically included in the executable program by the linker.

The sources for the startup code is included in the compiler package and could be changed corresponding to your needs.

[MEMO]

CHAPTER 3 FUNCTIONS OF V853

This chapter specifies sample programs based on overviews of V853 internal peripheral I/O functions and concrete sample hardware configurations.

Each function is specified in the following order.

<1> Internal peripheral I/O configuration diagram (omitted if not diagrammable)
<2> Internal peripheral I/O operations and contents of each control register
<3> Sample hardware configuration diagram and function overview
<4> Sample control register settings for function being specified
<5> Program flow chart
<6> Program list

3.1 Interrupt and Exception Handling Functions

The V853 has the following interrupt sources.

- Non-maskable interrupts: 1 source
- Maskable interrupts: 32 sources

For sources of maskable interrupts, it is possible to set eight levels of programmable priorities and handle multiple interrupts according to their priorities. In addition, an interrupt control register is allocated to an individual source of maskable interrupts and this can be used to set an interrupt mask or priority levels.

The V853 can activate software exception handling using a TRAP instruction. The interrupt codes 00H-1FH can be specified as causes of interrupts.

An exception trap also is activated if an illegal instruction code is fetched.

3.1.1 Interrupt servicing procedure

V853 interrupts have a specific interrupt handler address set for each source of interrupt. When an interrupt request is accepted, processing transfers to the interrupt handler.

The space allocated to handler processing is 16 bytes for resets, software exceptions, and maskable interrupts, 48 bytes for non-maskable interrupts, and 32 bytes for exception traps.

Since the CPU accesses instructions using one clock, it is possible to speed up interrupt servicing by locating the interrupt service body in internal ROM.

3.1.2 Non-maskable interrupts

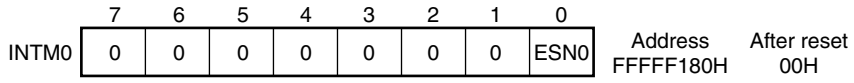
Non-maskable interrupts are accepted unconditionally, even in an interrupt disabled (DI) state. In addition, they are not subject to interrupt priorities and are the highest priority interrupt requests of all interrupts.

For a non-maskable interrupt, an interrupt occurs when the effective edge specified by bit 0 (ESN0) of external interrupt mode register 0 (INTM0) is detected by the NMI pin.

The NP flag of the PSW is set to 1 during non-maskable interrupt service (see Figure 3-2). If a non-maskable interrupt is requested during non-maskable interrupt service, it is pended. The pended non-maskable interrupt request is accepted after the currently executing interrupt service ends (after RETI instruction execution) or after an LDSR instruction sets the NP flag of the PSW to 0.

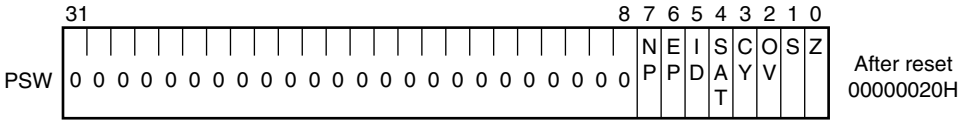
External interrupt mode register 0 (INTM0) can specify that the effective edge of the non-maskable interrupt either rising edge or falling edge (see Figure 3-1).

Figure 3-1: External Interrupt Mode Register 0 (INTM0)



Bit position	Bit name	Description
0	ESN0	Edge Select NMI Specifies effective edge of NMI pin. 0: Falling edge 1: Rising edge

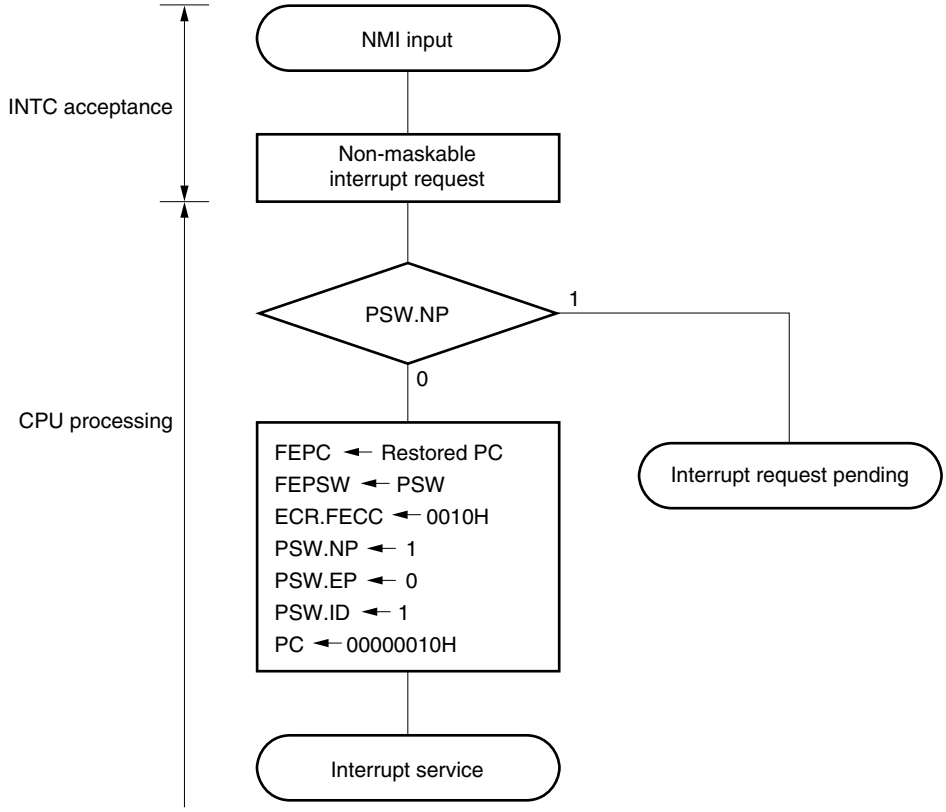
Figure 3-2: NP Flag



Bit position	Bit name	Description
7	NP	NMI Pending Indicates whether NMI interrupt service is in progress. 0: NMI interrupt service not in progress. 1: NMI interrupt service in progress.

Figure 3-3 shows the flow of operation of non-maskable interrupt request acceptance.

Figure 3-3: Non-maskable Interrupt Request Processing Flow



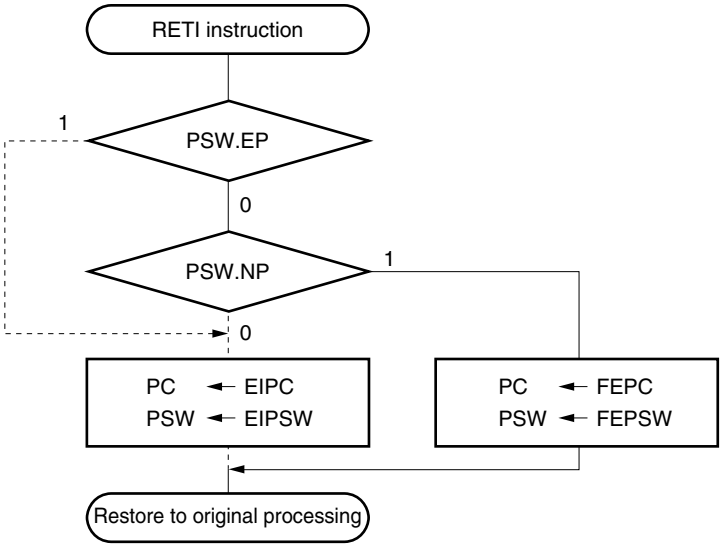
The operation of non-maskable interrupt request acceptance processing is shown below.

<1> Save restored PC to FEPC.
<2> Save current PSW to FEPSW.
<3> Set exception code (0010H) in upper half-word (FECC) of ECR.
<4> Set NP flag and ID flag of PSW to 1 and clear EP flag to 0.
<5> Set handler address for non-maskable interrupt (00000010H) in PC and transfer control to handler.

If a non-maskable interrupt is requested while a non-maskable interrupt is being kept pending, the request is ignored. That is, only one non-maskable interrupt can be kept pending.

Figure 3-4 shows the flow of non-maskable interrupt restore processing.

Figure 3-4: Flow of Restore from Non-maskable Interrupt



Caution: If the EP flag or NP flag of the PSW was changed by an LDSR instruction during non-maskable interrupt servicing, use an LDSR instruction to set PSW.EP to 0 and PSW.NP to 1 preceding the RETI instruction when restoring using the RETI instruction in order to restore the PC and PSW normally.

Remark: The CPU process flow is indicated by the solid line.

The operation of non-maskable interrupt restore processing is shown below.

<1>Since the EP flag of the PSW is 0 and the NP flag is 1, fetch the restored PC and saved PSW from FEPC and FEPSW, respectively.
<2>Pass control by transferring the fetched restored PC and PSW to the PC and PSW.

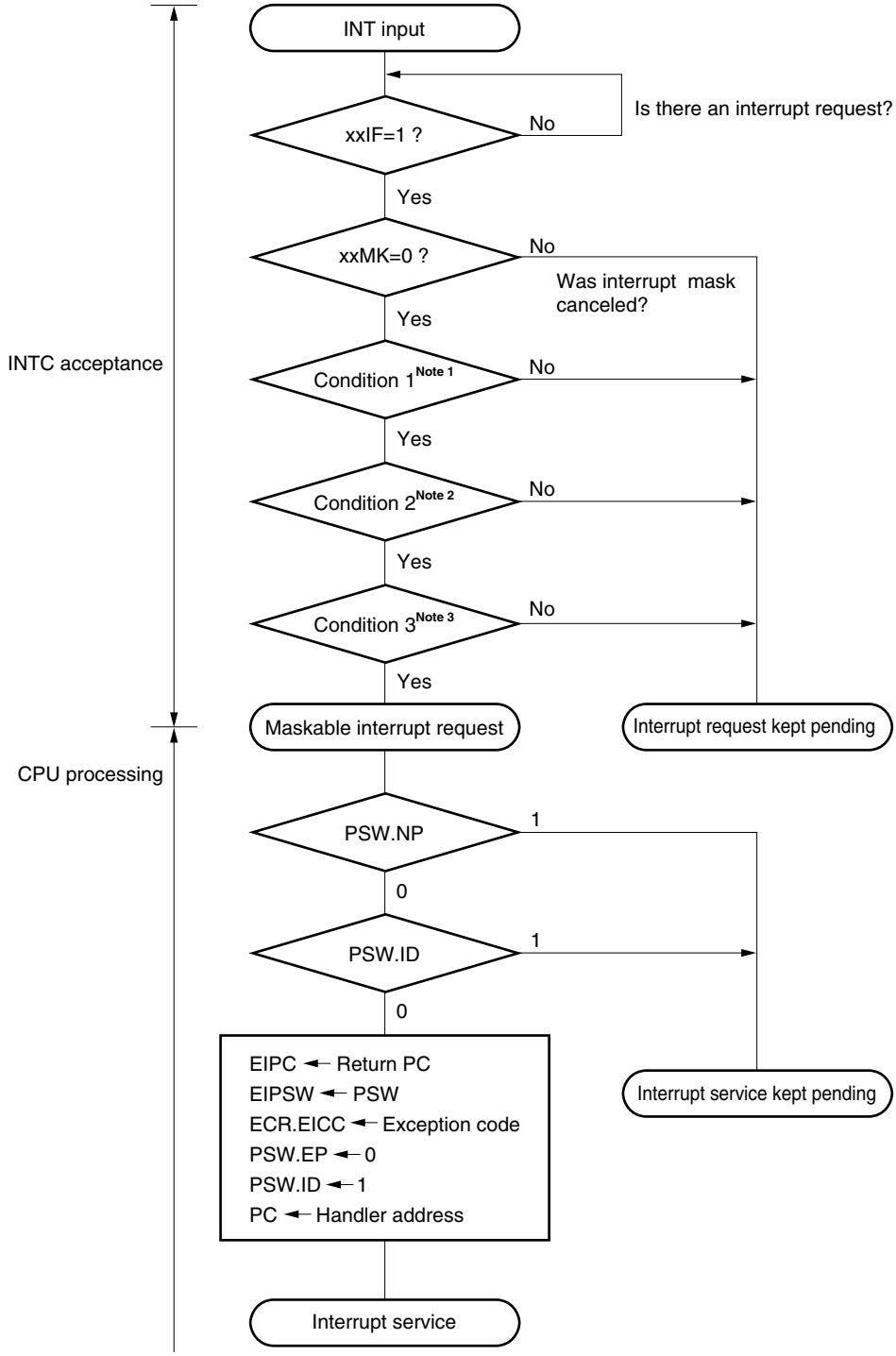
3.1.3 Maskable Interrupts

V853 maskable interrupt requests include 32 interrupt sources. For each interrupt source, there is an interrupt control register. The interrupt control register is used to set the interrupt mask and interrupt priority.

When a maskable interrupt request is accepted, the status becomes interrupt disabled.

Figure 3-5 shows the flow of maskable interrupt request acceptance operation.

Figure 3-5: Flow of Maskable Interrupt Request Processing



Notes:1. Condition 1: Is priority higher than currently servicing interrupt?

2. Condition 2: Is priority higher than another interrupt request?

3. Condition 3: Is default priority highest among interrupt requests with the same priority?

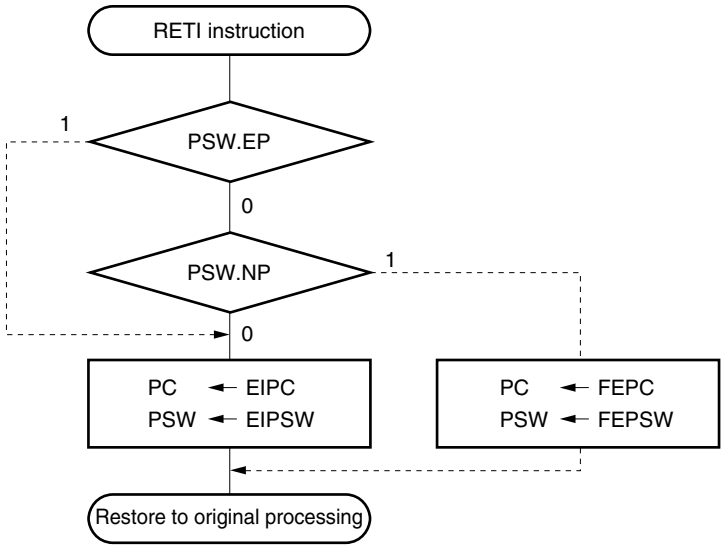
The operation of maskable interrupt acceptance processing is shown below.

<1> Save restored PC to EIPC.
<2> Save current PSW to EIPSW.
<3> Write exception code in lower half-word (EICC) of ECR.
<4> Set ID flag of PSW to 1 and clear EP flag to 0.
<5> Pass control by setting handler address for each interrupt in PC.

For maskable interrupts, it is possible to set a priority for each interrupt source by setting an interrupt control register.

Figure 3-6 shows the flow of maskable interrupt restore processing.

Figure 3-6: Flow of Restore from Maskable Interrupt



Caution: If the EP flag or NP flag of the PSW was changed by an LDSR instruction during maskable interrupt servicing, use an LDSR instruction to set PSW.EP to 0 and PSW.NP to 0 preceding the RETI instruction when restoring using the RETI instruction in order to restore the PC and PSW normally.

Remark: The CPU process flow is indicated by the solid line.

The operation of maskable interrupt restore processing is shown below.

<1>Since the EP flag of the PSW is 0 and the NP flag is 0, fetch the restored PC and saved PSW from EIPC and EIPSW, respectively.
<2>Pass control by transferring the fetched restored PC and PSW to the PC and PSW, respectively.

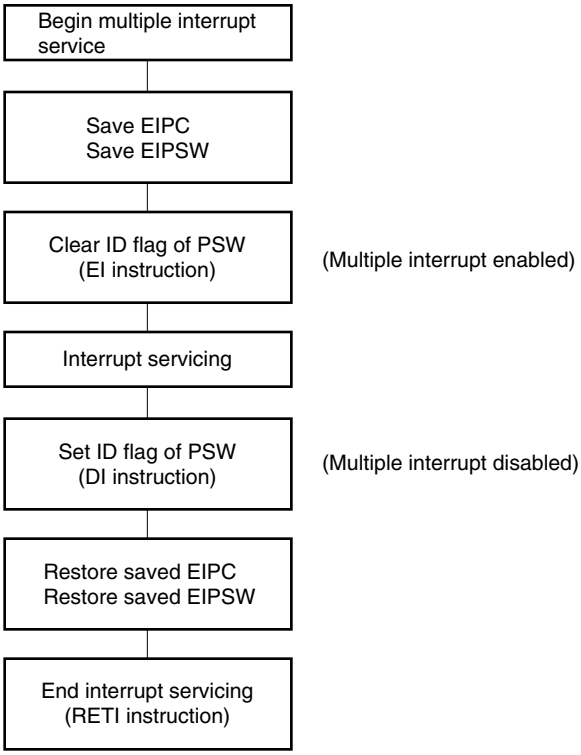
3.1.4 Multiple interrupts

Although the status becomes interrupt disabled (DI) when a maskable interrupt is accepted, if there is a multiple interrupt, the status is made interrupt enabled (EI) within the interrupt servicing.

When a multiple interrupt is enabled, be sure to save the EIPC and EIPSW in which the restored PC and interrupt acceptance time PSW are saved in memory or in a register. In addition, before returning from interrupt service (RETI instruction), return the saved EIPC and EIPSW while in a multiple interrupt disabled state and then return from interrupt service.

Figure 3-7 shows the process flow for enabling multiple interrupts.

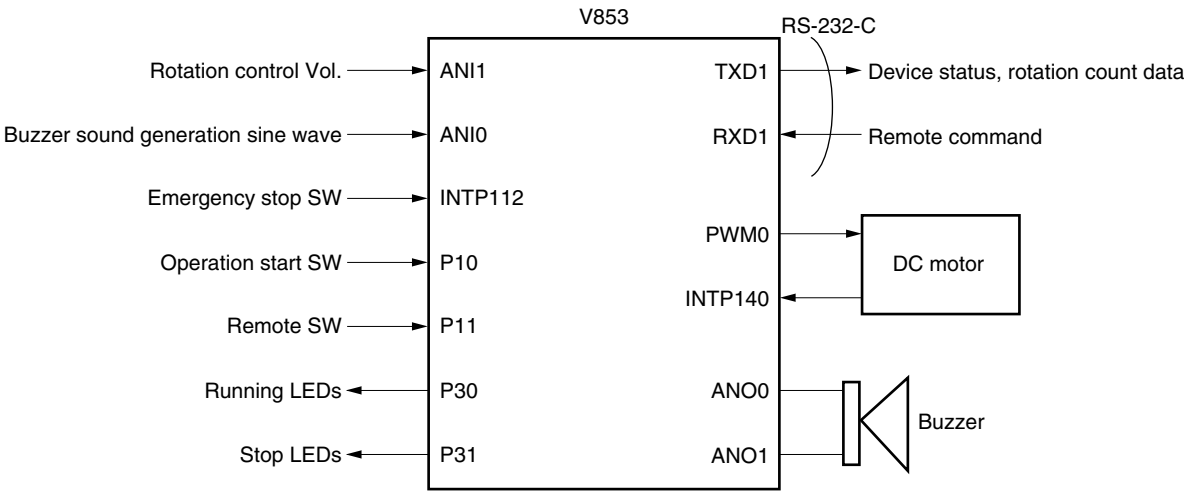
Figure 3-7: Flow of Multiple Interrupt Service



3.1.5 Sample program for setting interrupts

This program sets interrupts based on the hardware configuration below.

Figure 3-8: Sample Hardware Configuration



(1) Function overview

- From the value input A/D by the rotation control Vol., use the PWM function to cause the DC motor to rotate.
- Count the output pulses from the DC motor and find the number of rotations of the motor. Serially output the number of rotations calculated at fixed intervals.
- Input operation starts and remote switching from ports. Indicate the operating condition of the device by lighting the LEDs connected to a port.
- The emergency stop SW uses an interrupt input.
- Sound the buzzer at the start of operation or at the time of an emergency stop. The buzzer sounds by D/A output of an A/D input sine wave.
- Set up remote mode to perform operation starts and stops serially.

The five maskable interrupts are as follows.

<1> INTP112 (emergency stop SW)
<2> INTP140 (DC motor pulse counter)
<3> INTSR1 (serial receive interrupt)
<4> INTCM4 (interval timer interrupt)
<5> INTAD (A/D conversion end)

Figure 3-9 shows the interrupt control register for each interrupt source.

Figure 3-9: Interrupt Control Registers (xxICn)

	7	6	5	4	3	2	1	0	Address	After reset
xxICn	xxIFn	xxMKn	0	0	0	xxPRn2	xxPRn1	xxPRn0	FFFFF100H to FFFFF13EH	47H

Bit position	Bit name	Description
7	xxIFn	Interrupt Request Flag This is the interrupt request flag. 0: No interrupt request 1: Interrupt request present The xxIFn flag is reset automatically by the hardware when an interrupt request is accepted.
6	xxMKn	Mask Flag This is the interrupt mask flag. 0: Enable interrupt service 1: Disable interrupt service (pending)
2-0	xxPRn2 to xxPRn0	Priority Specifies 8 levels of priorities for each interrupt.

xxPRn2	xxPRn1	xxPRn0	Interrupt priority specification bit
0	0	0	Specifies level 0 (highest)
0	0	1	Specifies level 1
0	1	0	Specifies level 2
0	1	1	Specifies level 3
1	0	0	Specifies level 4
1	0	1	Specifies level 5
1	1	0	Specifies level 6
1	1	1	Specifies level 7 (lowest)

Remark: xx: Peripheral unit identifier (OV, P11 to P14, CM, CS, SE, SR, ST, AD)
n: Peripheral unit number (none or 0 to 4, 11 to 14)

Figure 3-10 shows interrupt control register settings.

Figure 3-10: Interrupt Control Register Settings

	7	6	5	4	3	2	1	0	Address	Value
P11IC2	0	0	0	0	0	1	1	1	FFFFF10CH	07H
P14IC0	0	0	0	0	0	1	1	1	FFFFF120H	07H
CMIC4	0	0	0	0	0	1	1	1	FFFFF128H	07H
SRIC1	0	0	0	0	0	1	1	1	FFFFF13AH	07H
ADIC	0	0	0	0	0	1	1	1	FFFFF13EH	07H

Setting the xxMKn bit (bit 6) to 0 sets interrupt enabled status.

Since the priorities of each interrupt request are set to the same rank (lowest), the order is as shown below due to default priorities. (Refer to **APPENDIX A LIST OF INTERRUPT CONTROL REGISTERS** regarding default priorities.) Set priorities using bits 2 to 0 of each control register.

Priority High ◄——————————————► Low

P11IC2 > P14IC0 > CMIC4 > SRIC1 > ADIC

The sample hardware uses the external interrupt inputs INTP112 and INTP140. To cause an interrupt using a signal input from a pin in this way, specify the effective edge. Also specify the effective edge for NMI.

The external interrupt mode registers set the effective edge of each external interrupt input.

Figure 3-11: External Interrupt Mode Registers 1 to 4 (INTM1 to INTM4)

	7	6	5	4	3	2	1	0	Address	After reset
INTM1	ES031	ES030	ES021	ES020	ES011	ES010	ES001	ES000	FFFFF182H	00H
Control pin	INTP113/ADTRG		INTP112		INTP111		INTP110			
INTM2	ES131	ES130	ES121	ES120	ES111	ES110	ES101	ES100	FFFFF184H	00H
Control pin	INTP123		INTP122		INTP121		INTP120			
INTM3	ES231	ES230	ES221	ES220	ES211	ES210	ES201	ES200	FFFFF186H	00H
Control pin	INTP133		INTP132		INTP131		INTP130			
INTM4	ES331	ES330	ES321	ES320	ES311	ES310	ES301	ES300	FFFFF188H	00H
Control pin	INTP143		INTP142		INTP141		INTP140			

Bit position	Bit name	Description
7, 5, 3, 1 6, 4, 2, 0	ES(m–1)n1 ES(m–1)n0 (m = 4 to 1, n = 3 to 0)	Edge Select Specifies effective edge of INTP1mn pin and ADTRG pin.

ES(m–1)n1	ES(m–1)n0	Operation
0	0	Falling edge
0	1	Rising edge
1	0	RFU (reserved)
1	1	Rising and falling edges

INTP113 is also used as an A/D converter external trigger input (ADTRG). Therefore, if bits TRG0 to TRG2 of A/D converter mode register 1 (ADM1) are set to external trigger mode, the effective edge of INTP113 becomes the effective edge of ADTRG.

Figure 3-12 shows sample settings of external interrupt mode registers when the effective edge of sample hardware external interrupts and NMI interrupts is rising. (INTP112 is set using bits 5 and 4 of INTM1 and INTP140 is set using bits 1 and 0 of INTM4.)

Figure 3-12: Sample External Interrupt Mode Register Settings

	7	6	5	4	3	2	1	0	Address	Value
INTM0	0	0	0	0	0	0	0	1	FFFFF180H	01H
INTM1	0	0	0	1	0	0	0	0	FFFFF182H	10H
INTM2	0	0	0	0	0	0	0	0	FFFFF184H	00H
INTM3	0	0	0	0	0	0	0	0	FFFFF186H	00H
INTM4	0	0	0	0	0	0	0	1	FFFFF188H	01H

(2) Sample program

```
/* ————————————————————————
 * V853 interrupt initialization sample program
 *
 * Initialization program for the following hardware configuration
 *
 * Interrupt functions used
 * A/D conversion end       : INTAD
 * Serial receive interrupt : INTSR1
 * Interval timer interrupt : INTCM4
 * SW interrupt             : INTP112
 * Motor rotation encoder   : INTP140
 *
 * ————————————————————————
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/08 adapted to IAR compiler by NEC Duesseldorf AH
 * File Name: intinit.c
 * ————————————————————————
*/

#include <io_v850_d3003.h>

/* ————————————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: DC motor emergency stop : Level 7 */
   P14IC0 = 0x07; /* INTP140: DC motor pulse counter  : Level 7 */
   CMIC4 = 0x07;  /* INTSR1 : Serial receive interrupt: Level 7 */
   SRIC1 = 0x07;  /* INTCM4 : Interval timer interrupt: Level 7 */
   ADIC = 0x07;   /* INTAD  : A/D conversion end      : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI    : Rising edge */
   INTM1 = 0x10; /* INTP112: Rising edge */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x01; /* INTP140: Rising edge */
}

void main()
{
}
```

3.2 Port Functions

The V853 has the following ports for port functions.

- Input only ports: 8
- I/O ports: 67

I/O ports can be set for input or output in one-bit units.

3.2.1 Port mode and control mode

The V853 has on-chip hardware that uses single pins as both control pins and port pins. The user selects whether to use each pin as a port pin or a control pin.

Table 3-1 shows the relationship between control pins and port pins.

Table 3-1: Relationship between Control Pins and Port Pins (1/2)

Port		Control mode	Function in control mode
Port 0	P00	TO110	Real-time pulse unit (RPU) output
	P01	TO111	
	P02	TCLR11	Real-time pulse unit (RPU) input
	P03	TI11	
	P04 to P06	INTP110 to INTP112	External interrupt input
	P07	INTP113/ADTRG	External interrupt input, A/D converter external trigger input
Port 1	P10	TO120	Real-time pulse unit (RPU) output
	P11	TO121	
	P12	TCLR12	Real-time pulse unit (RPU) input
	P13	TI12	
	P14	INTP120	External interrupt input
	P15	INTP121/SO2	External interrupt input
	P16	INTP122/SI2	Serial interface (CSI2) I/O
	P17	INTP123/$\overline{\text{SCK2}}$	
Port 2	P20	PWM0	PWM output
	P21	PWM1	
	P22	TXD0/SO0	Serial interface (UART/CSI) I/O
	P23	$\overline{\text{RXD0}}$/SI0	
	P24	SCK0	
	P25	TXD1/SO1	
	P26	RXD1/SI1	
	P27	$\overline{\text{SCK1}}$	

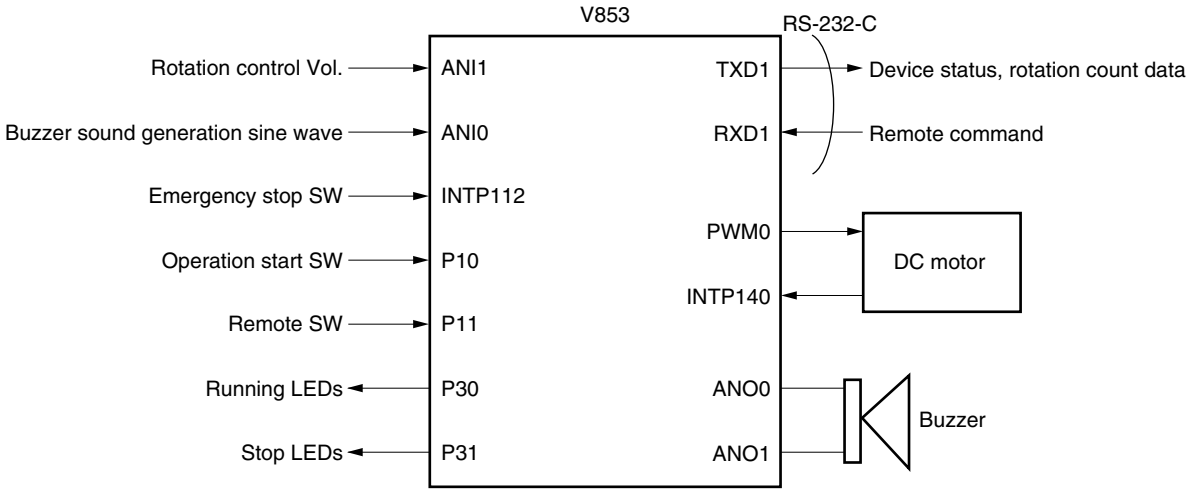
Table 3-1: Relationship between Control Pins and Port Pins (2/2)

Port		Control mode	Function in control mode
Port 3	P30	TO130	Real-time pulse unit (RPU) output
	P31	TO131	
	P32	TCLR13	Real-time pulse unit (RPU) input
	P33	TI13	
	P34	INTP130	External interrupt input
	P35	INTP131/SO3	External interrupt input
	P36	INTP132/SI3	Serial interface (CSI3) I/O
	P37	INTP133/$\overline{\text{SCK3}}$	
Port 4	P40 to P47	AD0 to AD7	Address/data bus when memory is expanded
Port 5	P50 to P57	AD8 to AD15	Address/data bus when memory is expanded
Port 6	P60 to P63	AD16 to AD19	Address bus when memory is expanded
Port 7	P70 to P77	ANI0 to ANI7	Analog input to A/D converter
Port 9	P90	$\overline{\text{LBEN}}$	Control signal output when memory is expanded
	P91	$\overline{\text{UBEN}}$	
	P92	R/$\overline{\text{W}}$	
	P93	$\overline{\text{DSTB}}$	
	P94	ASTB	
	P95	$\overline{\text{HLDAK}}$	Bus hold acknowledge signal output
	P96	$\overline{\text{HLDRQ}}$	Bus hold request signal input
Port 11	P110	TO140	Real-time pulse unit (RPU) output
	P111	TO141	
	P112	TCLR14	Real-time pulse unit (RPU) input
	P113	TI14	
	P114 to P117	INTP140 to INTP143	External interrupt request input

3.2.2 Sample program for setting ports

An example of selecting port or control functions based on the following hardware configuration is shown below.

Figure 3-13: Sample Hardware Configuration



The pins used in control mode in the sample hardware are shown below. On-chip hardware and control pins used are shown (items in parentheses indicate alternate functions as a port pin).

<1> A/D converter: ANI0 (P70)
ANI1 (P71)
<2> Serial interface: TXD1 (P25)
RXD1 (P26)
<3> D/A converter: ANO0 (no alternate function)
ANO1 (no alternate function)
<4> PWM unit: PWM0 (P20)
<5> External interrupt: INTP112 (P06)

Caution: Although ANI0 and ANI1 are also used as port 7, input port and analog input pins cannot be switched.

For the pins above, the port n mode control register (PMCn) is used to set the relevant pin to control mode (n = 0, 1, 2, 3, 11). Refer to **V853 User's Manual Hardware** for details of port n mode control registers (PMCn).

Figure 3-14: Setting Port n Mode Control Registers (PMCn)

	7	6	5	4	3	2	1	0	Address	After reset
PMCn	PMCn7	PMCn6	PMCn5	PMCn4	PMCn3	PMCn2	PMCn1	PMCn0	FFFFF040H to FFFFF056H	00H

Bit position	Bit name	Description
7-0	PMCn7 to PMCn0 (n=0 to 3, 11)	Port Mode Control Sets the operating mode of pins Pn0 to Pn7. 0: I/O port mode 1: Control mode

	7	6	5	4	3	2	1	0	Address	Value
PMC0	0	1	0	0	0	0	0	0	FFFFF040H	40H
PMC1	0	0	0	0	0	0	0	0	FFFFF042H	00H
PMC2	0	1	1	0	0	0	0	1	FFFFF044H	61H
PMC3	0	0	0	0	0	0	0	0	FFFFF046H	00H
PMC11	0	0	0	1	0	0	0	0	FFFFF056H	10H

Select INTP112 → Set bit 6 of PMC0 register to 1
Select INTP140 → Set bit 4 of PMC11 register to 1
Select TXD1 → Set bit 5 of PMC2 register to 1
Select RXD1 → Set bit 6 of PMC2 register to 1
Select PWM0 → Set bit 0 of PMC2 register to 1

The control mode of ports 4 through 6 and 9 when the V853 is in memory expansion mode is address/data bus (ports 4 to 6) and memory control signal (port 9). Therefore, select these port functions and control functions using the memory expansion mode register (MM).

It is possible to select port functions and on-chip hardware control functions using the register settings discussed so far.

Next, specify input/output for pins selected to function as the port pin.

The port pins to be set are P10, P11, P30, and P31.

<1> P10, P11 Specify as input port
<2> P30, P31 Specify as output port

Port input/output mode is set by using the Port n Mode Register (PMn) (n = 0 to 6, 9, 11)

Figure 3-15: Port n Mode Register (PMn) Settings

	7	6	5	4	3	2	1	0	Address	After reset
PMn	PMn7	PMn6	PMn5	PMn4	PMn3	PMn2	PMn1	PMn0	FFFFF020H to FFFFF036H	FFH

Bit position	Bit name	Description
7 to 0	PMn7 to PMn0 (n = 0 to 5, 11)	Port Mode Sets input or output mode for pins Pn0 to Pn7. 0: Output mode (output buffer on) 1: Input mode (output buffer off)

	7	6	5	4	3	2	1	0	Address	After reset
PM6	—	—	—	—	PM63	PM62	PM61	PM60	FFFFF02CH	xFH

Bit position	Bit name	Description
3 to 0	PM63 to PM60	Port Mode Sets input or output mode for pins P63 to P70. 0: Output mode (output buffer on) 1: Input mode (output buffer off)

	7	6	5	4	3	2	1	0	Address	After reset
PM9	—	PM96	PM95	PM94	PM93	PM92	PM91	PM90	FFFFF032H	x1111111B

Bit position	Bit name	Description
6 to 0	PM96 to PM90	Port Mode Sets input or output mode for pins P96 to P90. 0: Output mode (output buffer on) 1: Input mode (output buffer off)

	7	6	5	4	3	2	1	0	Address	Value
PM0	1	1	1	1	1	1	1	1	FFFFF020H	FFH
PM1	1	1	1	1	1	1	1	1	FFFFF022H	FFH
PM2	1	1	1	1	1	1	1	1	FFFFF024H	FFH
PM3	1	1	1	1	1	1	0	0	FFFFF026H	FCH
PM4	1	1	1	1	1	1	1	1	FFFFF028H	FFH
PM5	1	1	1	1	1	1	1	1	FFFFF02AH	FFH
PM6	1	1	1	1	1	1	1	1	FFFFF02CH	FFH
PM9	1	1	1	1	1	1	1	1	FFFFF032H	FFH
PM11	1	1	1	1	1	1	1	1	FFFFF036H	FFH

Specify input for port P10 → Set bit 0 of register PM1 to 1
Specify input for port P11 → Set bit 1 of register PM1 to 1
Specify output for port P30 → Set bit 0 of register PM3 to 0
Specify output for port P31 → Set bit 1 of register PM3 to 0
Unused ports and ports used in control mode → Specify as input

As port function control registers in addition to these, there is a port control mode register (PCM) and a pull-up resistor option register (PUO).

Figure 3-16: Port Control Mode Register (PCM)

	7	6	5	4	3	2	1	0	Address	After reset
PCM	0	0	0	0	PCM3	0	PCM1	0	FFFFF05CH	00H

Bit position	Bit name	Description
3	PCM3	Port Control Mode Specifies the operating mode of pins P35 to P37. 0: INTP133 to INTP131 input mode 1: CSI3 I/O mode
1	PCM1	Port Control Mode Specifies the operating mode of pins P15 to P17. 0: INTP123 to INTP121 input mode 1: CSI2 I/O mode

Caution: This register setting becomes invalid if port mode is specified for PCM1 or PCM3 by the register.

Since the sample hardware does not use the CSI function, the PCM becomes 00H after reset.

If using CSI channel 2 or channel 3, also set the PCM together with PMC1 or PMC3 on initialization. If the values of the PCM is unchanged from reset time, the CSI pin signal operates as an external interrupt signal.

Figure 3-17: Pull-up Resistor Option Register (PUO)

	7	6	5	4	3	2	1	0	Address	After reset
PUO	0	0	0	0	PUO3	0	0	0	FFFFF05EH	00H
PM3	PM37	PM36	PM35	PM34	PM33	PM32	PM31	PM30	FFFFF026H	FFH

Bit position	Bit name	Description
3	PUO3	Pull-up Option Specifies on-chip pull-up resistor of pin P3n (n = 5 to 7).

PUO3	PM3n	Use/do not use on-chip pull-up resistor
0	0	Do not use on-chip pull-up resistor.
0	1	
1	0	
1	1	Use on-chip pull-up resistor.

Since pins P35 through P37 are not connected in the sample hardware, they are set to input mode. Therefore, the value of PUO becomes 00H after reset.

(1) Sample program

```
/* ————————————————————————
 * V853 port initialization sample program
 *
 * Initialization program for the following hardware configuration
 *
 * Functions used
 * Asynchronous serial    : TXD1,RXD1
 * SW interrupt           : INTP112
 * Analog input           : ANI0,ANI1
 * Headphone output       : ANO0,ANO1
 * DC motor control       : PWM0
 * Motor rotation encoder : INTP140
 * SW input               : P10,P11
 * LEDs output            : P30,P31
 * Uses 1-Mbyte external memory space
 *
 * ————————————————————————
 * History:
 * 1997/3/17   Ver 0.00.00
 * 1999/7/8    adapted to IAR compiler by NEC Duesseldorf AH
 * File Name:  portinit.c
 * ————————————————————————
*/

#include <io_v850_d3003.h>

/* ——————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * —————————————————— */
void PortInit(void)
{
   /* Port 0 (P0) initialization
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x40; /* Use INTP112 P06 in control mode */
                /* P00 to P05, P07 in port mode */

   /* Port 1 (P1) initialization
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Port 2 (P2) initialization
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x61; /* Use RXD1, TXD1, PWM0 P20, P22 and P23 in control mode */
                /* P21, P24 to P27 in port mode */
```

```
   /* Port 3 (P3) initialization
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */

   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Port 4 (P4) initialization
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Port 5 (P5) initialization
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Port 6 (P6) initialization
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Port 9 (P9) initialization
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Port 11 (P11) initialization
   */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x10; /* Use INTP140 P114 in control mode */
                 /* P110 to P113, P115 to P117 in port mode */
}

void main()
{
}
```

3.3 Timer/Counter (Real-time Pulse Unit) Function

The V853 has the following two types of timer.

- Timer 1 (16-bit timer/event counter)
- Timer 4 (16-bit interval timer)

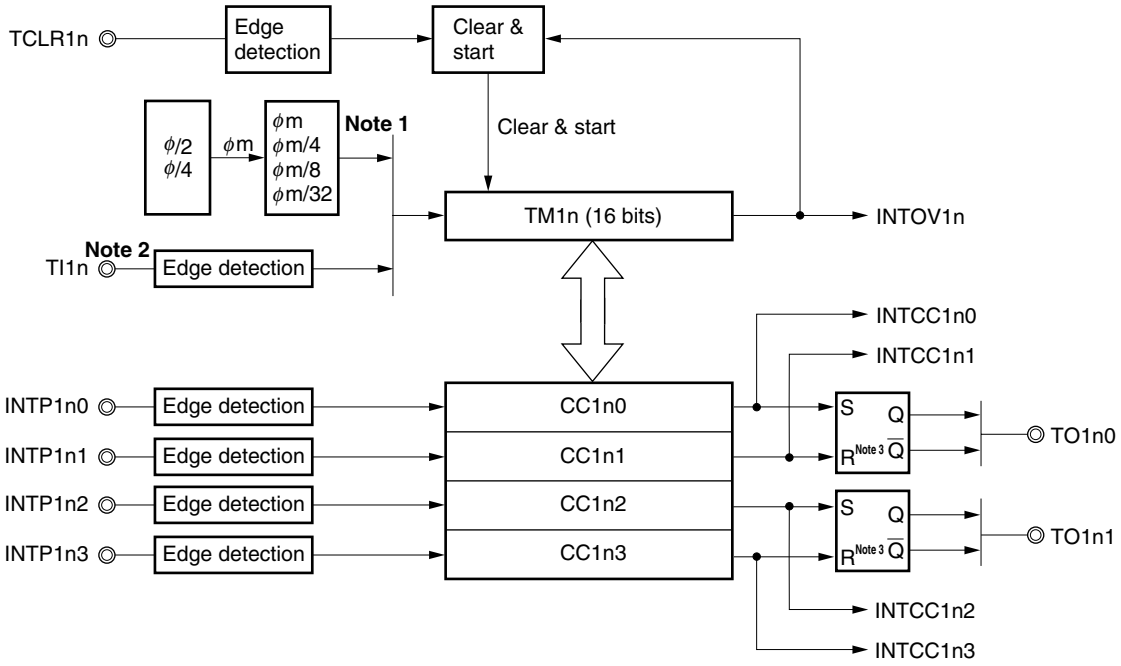
There are four timers for timer 1 and they have four capture/compare common registers each, for a total of 16. For the count clock source, system clock divisions can be selected or an external pulse can be input.

There is only one timer for timer 4 and it has one compare register. The count clock source is generated by dividing the system clock.

3.3.1 Configuration of timer 1

Timer 1 is a 16-bit timer/event counter. Figure 3-18 shows the configuration of timer 1.

Figure 3-18: Configuration of Timer 1



Notes:1. Internal count clock
2. External count clock
3. Reset priority

Remarks:1. n = 1 to 4
2. →: Internal system clock

Timer 1 operates as a 16-bit free-running timer or an external signal event counter.

An internal clock or external clock (TI1n pin input) can be selected for the count clock using the ETIn bit of the timer control register (TMC1n) (n = 1 to 4).

Figure 3-19: Timer Control Register 1n (TMC1n)

	7	6	5	4	3	2	1	0	Address	After reset
TMC1n	CE1n	0	0	ETI1n	PRS1n1	PRS1n0	PRM1n1	0	FFFFF242H to FFFFF2A2H	00H

Bit position	Bit name	Description
7	CE1n	Count Enable Controls timer operation. See Figure 3-20.
4	ETI1n	External TI1n Input Specifies switching between internal and external count clock. 0: Specifies ϕ system (internal). 1: Specifies TI1n (external).
3, 2	PRS1n1, PRS1n0	Prescaler Clock Select Selects internal count clock (ϕ m indicates the intermediate clock).
1	PRM1n1	Prescaler Clock Mode Selects intermediate clock (ϕ m) of count clock. (ϕ : Internal system clock) 0: ϕ/2 1: ϕ/4

PRS1n1	PRS1n0	Count clock
0	0	ϕm
0	1	ϕm/4
1	0	ϕm/8
1	1	ϕm/32

Caution: Do not change the count clock during timer operation.

Remark: n = 1 to 4

When an internal count clock is specified, the clock can be selected in the seven ways shown in Table 3-2 by setting the PRS1n1 bit, PRS1n0 bit, and PRM1n1 bit.

Table 3-2: Count Clock Selection

PRS1n1	PRS1n0	PRM1n1	Count clock
0	0	0	ϕ/2
0	0	1	ϕ/4
0	1	0	ϕ/8
0	1	1	ϕ/16
1	0	0	
1	0	1	ϕ/32
1	1	0	ϕ/64
1	1	1	ϕ/128

Remark: n = 1 to 4, ϕ: Internal system clock

When an external count clock is specified, specify the effective edge of the signal input to the TI1n pin. Specify the effective edge of the TI1n pin in the TES1n1 bit and TES1n0 bit of the timer unit mode register (TUM1n) (n = 1 to 4).

Table 3-3: Specification of TES Bit of TUM1n

TES1n1	TES1n0	Effective edge
0	0	Rising edge
0	1	Falling edge
1	0	RFU (reserved)
1	1	Rising and falling edges

Remark: n = 1 to 4

The timer 1 count operation start processing is shown below.

The count start trigger for timer 1 differs according to the settings of the CE1n bit of TMC1n and ECLR1n bit of TUM1n (n = 1 to 4).

Figure 3-20: CE1n Bit of Timer Control Register 1n (TMC1n)

	7	6	5	4	3	2	1	0	Address	After reset
TMC1n	CE1n	0	0	ETI1n	PRS1n1	PRS1n0	PRM1n1	0	FFFFF242H to FFFFF2A2H	00H

Bit position	Bit name	Description
7	CE1n (n = 1 to 4)	Count Enable Controls operation of timer. 0: Timer stops in “0000H” state and does not operate. 1: Timer performs count operation. However, if ECLR1n of TUM1n register is 1, timer does not start counting up until there is TCLR1n input. If ECLR1n = 0, the operation of writing “1” to the CE1n bit is the start trigger for starting a timer count due to CE1n = 1. Therefore, setting ECLR1n = 0 after setting CE1n in ECLR1n = 1 status does not start the timer.

Caution: Do not change the count clock during timer operation.

Table 3-4: Specification of ECLR1n Bit of TUM1n

ECLR1n	Operation due to changing CE1n bit	Operation due to TCLR1n input
0	CE1n 1: Start count CE1n 0: Stop count & clear counter	No effect on operation
1	CE1n 1: No effect on operation CE1n 0: Stop count & clear counter	Clear counter & start count (if CE1n=1)

Remark: n = 1 to 4

Figure 3-21: Timer Unit Mode Register 1n (TUM1n)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	After reset
TUM1n	0	0	OSTn	ECLR1n	TES1n1	TES1n0	CES1n1	CES1n0	CMS1n3	CMS1n2	CMS1n1	CMS1n0	IMS1n3	IMS1n2	IMS1n1	IMS1n0	FFFFF240H to FFFFF2A0H	0000H

Bit position	Bit name	Description
13	OSTn	Overflow Stop Specifies operation after timer overflow. This flag is in effect only for TM1n. 0: After timer overflow, timer continues counting up. 1: After timer overflow, timer is in stop state holding 0000H. At this time, the CE1n bit of the TMC1n register remains “1”. Counting up restarts according to the following operation. If ECLR1n = 0: Operation to write “1” to bit CE1n If ECLR1n = 1: Trigger input to timer clear pin (TCLR1n)
12	ECLR1n	External Input Timer Clear Enables clearing of timer according to TM1n external clear input (TCLR1n). 0: Do not clear according to external input. 1: Clear TM1n according to external input. After clearing, start counting up.
11, 10	TES1n1, TES1n0	TI1n Edge Select Specifies effective edge of external clock input (TI1n).
9, 8	CES1n1, CES1n0	TCLR1n Edge Select Specifies effective edge of external clear input (TCLR1n).
7 to 4	CMS1nm (m = 0 to 3)	Capture/Compare Mode Select Selects operating mode of capture/compare register (CC1nm). See Figure 3-23.
3 to 0	IMS1nm (m = 0 to 3)	Interrupt Mode Select Selects INTP1nm or INTCC1nm as interrupt source. See Figure 3-23.

CES1n1	CES1n0	Effective edge
0	0	Falling edge
0	1	Rising edge
1	0	RPU (reserved)
1	1	Rising and falling edges

Remark: n = 1 to 4

If timer 1 overflows as a result of counting, the processing after the overflow differs depending on the setting of the OSTn bit of TUM1n.

Table 3-5: Specifying OSTn Bit of TUM1n

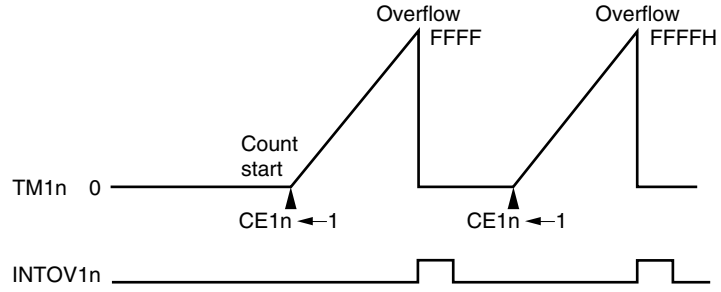
OSTn	Operation on counter overflow
0	Set OVF1n bit of TOVS register to 1 Generate overflow interrupt Continue count up operation (count value: FFFF ∅ 0)
1	Set OVF1n bit of TOVS register to 1 Generate overflow interrupt Stop count up operation (count value: held at 0)

Remark: n = 1 to 4

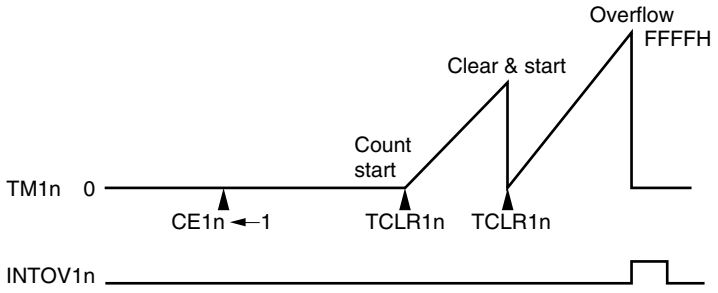
Figure 3-22 shows count start and overflow operations.

Figure 3-22: Count Start and Overflow Operations (1/2)

(a) Operation after overflow (when ECLR1n = 0, OSTn = 1)



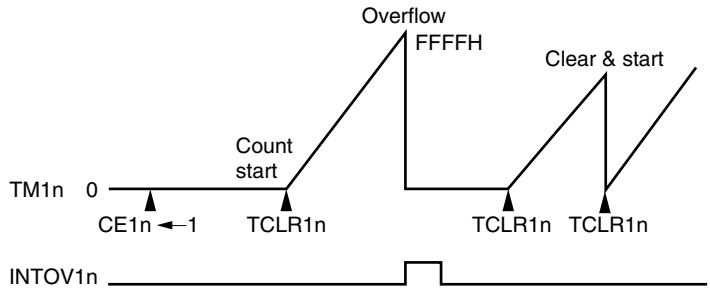
(b) Timer clear/start operation due to TCLR1n signal input (when ECLR1n = 1, OSTn = 0)



Remark: n = 1 to 4

Figure 3-22: Count Start and Overflow Operations (2/2)

(c) Clear/start and overflow operations due to TCLR1n signal input (when ECLR1n=1, OSTn=1)



Remark: n = 1 to 4

3.3.2 Capture operation (timer 1)

If a timer 1 capture/compare register is set as a capture register, the real-time pulse unit performs the following operations.

<1> Make the effective edge of the external interrupt corresponding to the capture register the capture trigger and latch the timer 1 count value in the capture register. This capture operation is performed asynchronous to the timer count clock. The latched value is held in the capture register until the next capture trigger.

<2> An interrupt occurs whenever the effective edge of the external interrupt that is the capture trigger is detected.

Caution: If the timing of a latch to a capture register and a write to the capture register by the program are in contention, the write by the program is given priority and the capture operation (latch to register) is ignored.

Specify capture/compare register selection and interrupt source selection using the CMS1nm bit and IMS1nm bit of TUM1n (n = 1 to 4, m = 0 to 3).

Figure 3-23: Timer Unit Mode Register 1n (TUM1n)

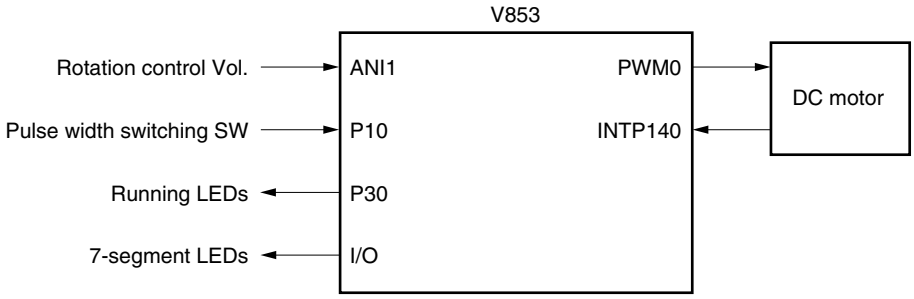
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	After reset
TUM1n	0	0	OSTn	ECLR 1n	TES 1n1	TES 1n0	CES 1n1	CES 1n0	CMS 1n3	CMS 1n2	CMS 1n1	CMS 1n0	IMS 1n3	IMS 1n2	IMS 1n1	IMS 1n0	FFFFF240H to FFFFF2A0H	0000H

Bit position	Bit name	Description
13	OSTn	Overflow Stop Specifies operation after timer overflow. See Figure 3-21.
12	ECLR1n	External Input Timer Clear Enables clearing of timer due to TM1n external clear input (TCLR1n). See Figure 3-21.
11, 10	TES1n1,TES1n0	TI1n Edge Select Specifies effective edge of external clock input (TI1n).
9, 8	CES1n1,CES1n0	TCLR1n Edge Select Specifies effective edge of external clear input (TCLR1n). See Figure 3-21.
7 to 4	CMS1nm (m = 0 to 3)	Capture/Compare Mode Select Selects operating mode of capture/compare register (CC1nm). 0: Operate as capture register. However, capture operation when capture register is specified is performed only when CE1n bit of TMC1n is 1. 1: Operate as compare register.
3 to 0	IMS1nm (m = 0 to 3)	Interrupt Mode Select Selects INTP1nm or INTCC1nm as interrupt source. 0: Make signal that matches compare register (INTCC1nm) interrupt signal. 1: Make external input signal (INTP1nm) interrupt signal.

Remark: n = 1 to 4

3.3.3 Capture operation sample program

Figure 3-24: Sample Hardware Configuration



(1) Function overview

- Use the PWM function to make the DC motor rotate using the A/D input value from the rotation Vol.
- Measure the pulse width of the pulse counter signal from the DC motor and display it in the 7-segment LEDs.
- Depending on the pulse width display switching SW, display in the LEDs the pulse widths for which the pulse is in an active/inactive state.

In Figure 3-24 the pulse counter signal from the DC motor is input to INTP140. This external interrupt signal is taken as a capture trigger and the pulse width of the pulse counter is measured. Set register TUM14 and register TMC14 using the following conditions.

<1> Since INTP140 is a capture trigger, set register CC140 to a capture register.
<2> Make the interrupt source an external input signal (INTP140).
<3> The operation after the timer overflows is to continue to count up.
<4> Do not clear due to external input.
<5> Make the count clock ϕ/128 and select an internal clock.

Make the effective edge of the capture trigger both edges (set using external interrupt mode register 4 (INTM4)).

Figure 3-25: TUM14, TMC14, and INTM4 Settings

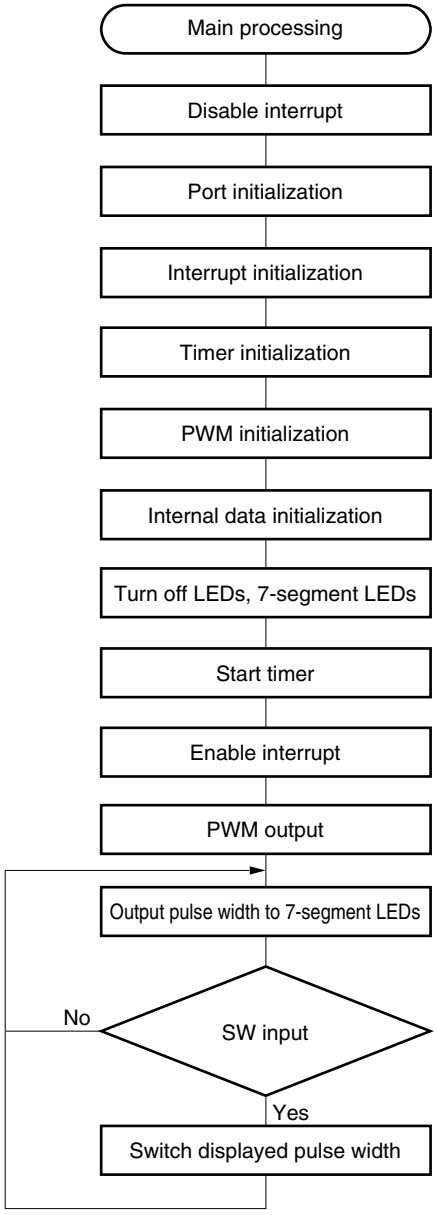
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Value
TUM14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FFFFF2A0H	0000H

	7	6	5	4	3	2	1	0	Address	Value
TMC14	0	0	0	0	1	1	1	0	FFFFF2A2H	0EH

	7	6	5	4	3	2	1	0	Address	Value
INTM4	0	0	0	0	0	0	1	1	FFFFF188H	03H

Figure 3-26 shows the flow of pulse width measurement processing that operates on the sample hardware.

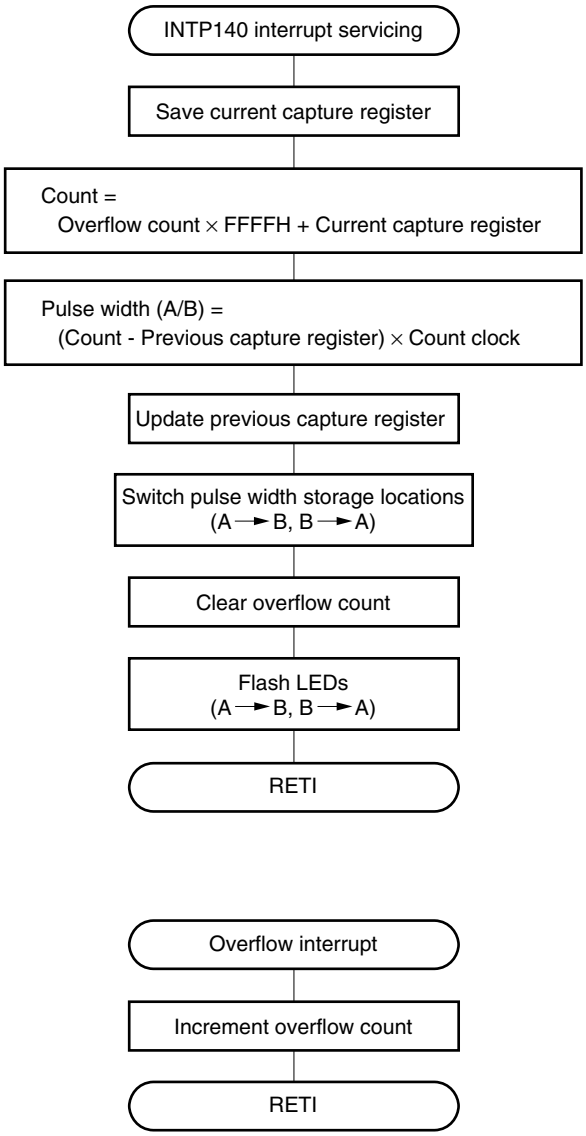
Figure 3-26: Flow of Pulse Width Measurement Processing (1/2)



Functions:

- Measure pulse width
- Display pulse width in 7-segment LEDs
- Switch pulse width display (active/inactive) using SW input
- Turn on LEDs at pulse active level and turn off when inactive
- Take count overflow into account in pulse width measurement

Figure 3-26: Flow of Pulse Width Measurement Processing (2/2)



If a capture trigger was specified only at the rising edge of INTP140, the frequency of the input pulse can be measured.

Figure 3-27: INTM4 Settings

	7	6	5	4	3	2	1	0	Address	Value
INTM4	0	0	0	0	0	0	0	1	FFFFF188H	01H

Figure 3-28 shows the flow of pulse frequency measurement processing.

Figure 3-28: Flow of Pulse Frequency Measurement Processing (1/2)

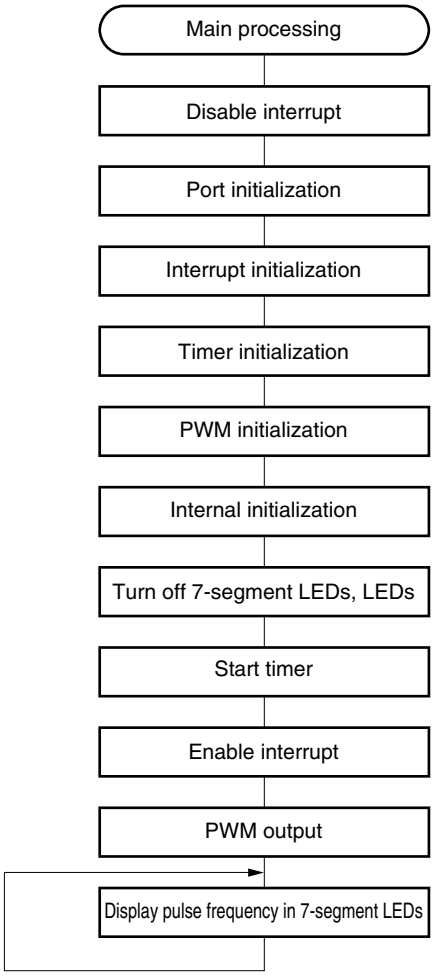
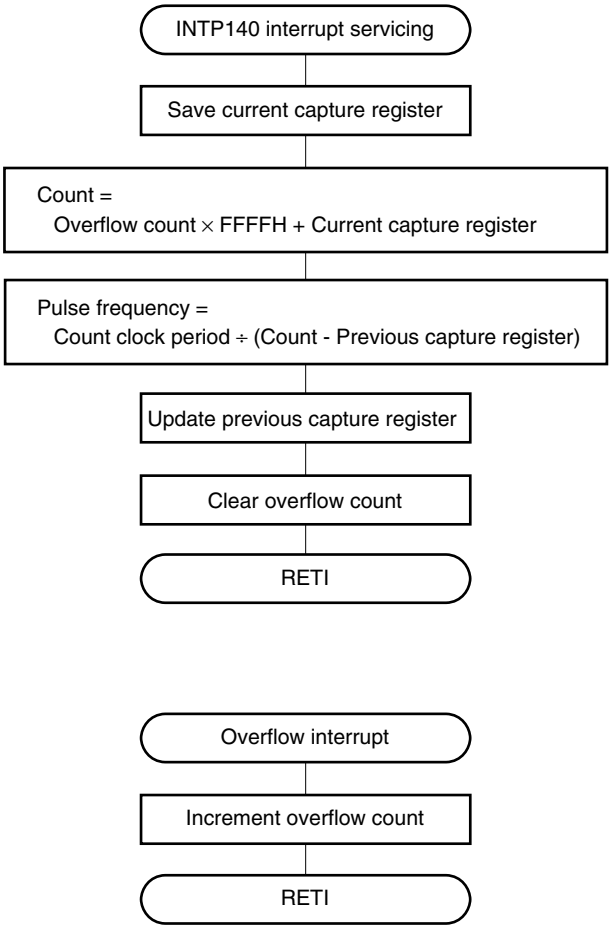


Figure 3-28: Flow of Pulse Frequency Measurement Processing (2/2)



(2) Sample program

```
/* ————————————————————————-
 * V853 timer/counter initialization program
 *                  (capture function)
 *
 * Initialization program for following hardware configuration
 *     ANI1 : Rotation control Vol. input
 *     PMW0 : DC motor control
 *     P10  : Run SW (On: run Off: stop)
 *     P30  : Running LEDs
 *
 * ————————————————————————-
 * History:
 *     1997/3/17 Ver 0.00.00
 *     1999/7/8 adapted for IAR compiler by NEC Duesseldorf AH
 * File Name: timcapt.c
 * ————————————————————————-
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

#include "7segLed.h"

static int SelFlg = 0;    /* Flag to determine pulse width storage location
*/
static long PulsWidth[2]; /* Pulse width storage variable (0: Active 1: Inac-
tive) */
static int OvflowCnt;     /* Overflow counter */


/* ———————————————
 * NMI service
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ———————————————
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
 ;    /* Do nothing */
}

/* ———————————————
 * Pulse width measurement
 * Interrupt source: INTP140
 * Arguments: None
 * Return value: None
 * ———————————————
*/
```

```
#pragma vector=INTP140_vector
__flat __interrupt void int_intp140(void)
{
   static int oldCC140 = 0; /* Previous capture register value */
   int countVal;            /* Work area to store timer counts */

   /* Pulse width calculation & storage */
   countVal = OvflowCnt * 0x10000 + CC140;
   PulsWidth[SelFlg] = (countVal - oldCC140); /* Pulse width [ms] */

   /* Stores capture register value */
   oldCC140 = CC140;
   SelFlg = (SelFlg == 0) ? 1 : 0;

   /* Clear overflow counter */
   OvflowCnt = 0;

   /* Processing to turn LEDs on/off */
   if (SelFlg != 0) {
      P3_bit.no0 = 1;
   } else {
      P3_bit.no0 = 0;
   }
}

/* ————————————————
 * A/D converter conversion end
 * Interrupt source: INTAD
 * Arguments: None
 * Return value: None
 * ————————————————
*/
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   /* Set A/D conversion result in PWM buffer register */
   PWM0 = ADCR1;

     /* Start A/D conversion */
     ADM0_bit.no7= 1;  /* Set Bit CE */
}

/* ————————————————
 * Count timer counter overflows
 * Interrupt source: INTOV14
 * Arguments: None
 * Return value: None
 * ————————————————
*/
```

```
#pragma vector=INTOV14_vector
__flat __interrupt void int_ov14(void)
{
   /* Count number of overflows of timer 14 */
   ++OvflowCnt;
}

/* ————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
void PortInit(void)
{
   /* Port 0 (P0) initialization
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x00; /* P00 to P07 in port mode */

   /* Port 1 (P1) initialization
   */
   PM1  = 0xFF; /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Port 2 (P2) initialization
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x01; /* Use PWM0 P20 in control mode */
                /* P21 to P27 in port mode */

   /* Port 3 (P3) initialization
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00; /* Port 1 and port 3 all in port mode */

   /* Port 4 (P4) initialization
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Port 5 (P5) initialization
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Port 6 (P6) initialization
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Port 9 (P9) initialization
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */
```

```
   /* Port 11 (P11) initialization
   */
   PM11  = 0xFF; /* All ports in input mode */
   PMC11 = 0x10; /* Use INTP140 P114 in control mode */
                 /* P110 to P113, P115 to P117 in port mode */
}

/* ————————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void IntInit(void)
{
   /* Interrupt register settings (set interrupt level) */
   OVIC14 = 0x07; /* INTOV14: Timer 14 overflow : Level 7 */
   P14IC0 = 0x07; /* INTP140: DC motor pulse counter: Level 7 */
   ADIC   = 0x07; /* INTAD  : A/D conversion end : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x00; /* Not used */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0X03; /* INTP140: Rising and falling edges */
}

/* ————————————————————
 * Timer initialization processing
 * Arguments : None
 * Return value: None
 * ————————————————————
*/
void TimerInit(void)
{
   /* Timer 14 settings (TUM14) */
   TUM14 = 0x0000; /* OST4 = 0: Continue count up even after timer overflow
*/
                   /* ECLR14 = 0: Do not enable external clear input */
                   /* TES141 to TES140 = 0: Falling edge */
                   /* CMS143 to CMS140 = 0: Make capture register */
                   /* IMS143 to IMS140 = 0: Interrupt source INTCC */
   /* Timer 14 settings (TMC14) */
   TMC14 = 0x0e;   /* ETI14 = 0: Timer count clock is internal clock */
                   /* PRM141 = 1: Intermediate clock is 1/4 system clock */
                   /* PRS141 = 1: Timer count clock is f m/32 */
                   /* PRS140 = 1: (Timer count clock = f/128) */
                   /* CE14 = 0: Stop timer */
}
```

```
/* ————————————————————
 * PWM unit initialization processing
 * Arguments : None
 * Return value: None
 * ————————————————————
*/
void PwmInit(void)
{
   /* PWM settings (PWMC): Only set PWM0 */
   PWMC = 0x02; /* PWME0 = 0: PWM in stopped state */
                /* ALV0 = 0: Active level is low */
                /* PRM01 = 1: Compare register is 10 bits */
                /* PRM00 = 0; */
   PWPR = 0x00; /* Prescaler f/1 */
   PWM0 = 0; /* Clear buffer register */
}

/* ————————————————————
 * A/D converter initialization processing
 * Arguments : None
 * Return value: None
 * ————————————————————
*/
void ADCInit(void)
{
   /* ADM0 register settings */
   ADM0 = 0x11; /* (BS) 1 buffer mode */
                /* (MS) select mode */
                /* (ANIS2 to ANIS0) Input analog signal from ANI1 */
   /* ADM1 register settings */
   ADM1 = 0x04; /* (TRG2 to TRG0) A/D trigger mode */
                /* (FR2 to FR0) Conversion clocks = 96 */
}

/* ————————————————————
 * 7-segment LED display processing
 * Arguments: disp_data Display data
              dot_pos Decimal point display position (1-LEDMAX)
              mode Decimal/hexadecimal (0: Decimal, 1: Hexadecimal)
 * Return value: None
 * ————————————————————
*/
```

```
void Disp7SegLED(unsigned long disp_data, int dot_pos, int mode)
{
   register unsigned char *segaddr = SEG7ADDR; /* 7-segment LED set address
*/
   register unsigned char cnt; /* Work counter */
   register int base; /* Decimal or hexadecimal radix */

   /* Set radix */
   if (mode == 0) {
      base = 10; /* Decimal display */
   } else {
      base = 16; /* Hexadecimal display */
   }
   /* Set all columns of 7-segment LEDs */
   for ( cnt = 1 ; cnt <= LEDMAX ; cnt ++ ) {
      /* Display of 7-segment LEDs */
      switch ( disp_data % base ) {
         case 0: *segaddr = SEGLED_0; break;
         case 1: *segaddr = SEGLED_1; break;
         case 2: *segaddr = SEGLED_2; break;
         case 3: *segaddr = SEGLED_3; break;
         case 4: *segaddr = SEGLED_4; break;
         case 5: *segaddr = SEGLED_5; break;
         case 6: *segaddr = SEGLED_6; break;
         case 7: *segaddr = SEGLED_7; break;
         case 8: *segaddr = SEGLED_8; break;
         case 9: *segaddr = SEGLED_9; break;
         case 10: *segaddr = SEGLED_A; break;
         case 11: *segaddr = SEGLED_B; break;
         case 12: *segaddr = SEGLED_C; break;
         case 13: *segaddr = SEGLED_D; break;
         case 14: *segaddr = SEGLED_E; break;
         case 15: *segaddr = SEGLED_F; break;
      }
      /* Create data of next column */
      disp_data /= base;
      /* If displaying decimal point */
      if ( cnt == dot_pos ) {
         /* Display of 7-segment LEDs */
         *segaddr += SEGLED_DOT;
      }
      /* Create set address of next column */
      segaddr += 2;
   }
}
```

```
/* ————————————————
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
void main(void)
{
   int pulsFlg = 0; /* Display pulse selection flag */
   int DispCnt = 0; /* Display counter */

   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* Timer initialization processing */
   TimerInit();

   /* PWM initialization processing */
   PwmInit();

   /* A/D converter initialization processing */
   ADCInit();

   /* Clear LEDs */
   P3 = 0;

   /* Internal data initialization */
   SelFlg = 0;
   PulsWidth[0] = 0;
   PulsWidth[1] = 0;
   OvflowCnt = 0;

   /* Start A/D conversion */
   ADM0_bit.no7= 1;  /* Set Bit CE */

   /* Start timer */
   TMC14_bit.no7= 1;  /* Set Bit CE14 */

   /* Enable interrupt */
   __EI();

   /* Start PWM output */
   PWMC_bit.no3= 1; /* Set Bit PWME0 */
```

```
   while (1) {
      /* Get display pulse selection flag */
      pulsFlg = P1_bit.no0;
      /* To confirm display, display once per 100,000 times */
      DispCnt++;
      if ( DispCnt >= 100000 ) {
         /* Display pulse width */
         Disp7SegLED(PulsWidth[pulsFlg], 0, 1); /* Hexadecimal display with-
out */
         DispCnt = 0;                           /* decimal point */
      }
   }
}
```

```
/* ————————————————————————-
* TU-V853 7-segment LED register definition
* ————————————————————————-
* History:
* 1997/3/17 Ver 0.00.00
* File Name: 7segled.h
* ————————————————————————-
*/

/* 7-segment LED data */
#define LEDMAX 5 /* Number of columns used */

/* Address of first column */
#define SEG7ADDR (unsigned char *) 0x00460000L

/* Segment data */
#define SEGLED_DOT 0x80 /* Point */
#define SEGLED_0 0x3f /* 0 */
#define SEGLED_1 0x06 /* 1 */
#define SEGLED_2 0x5b /* 2 */
#define SEGLED_3 0x4f /* 3 */
#define SEGLED_4 0x66 /* 4 */
#define SEGLED_5 0x6d /* 5 */
#define SEGLED_6 0x7d /* 6 */
#define SEGLED_7 0x07 /* 7 */
#define SEGLED_8 0x7f /* 8 */
#define SEGLED_9 0x6f /* 9 */
#define SEGLED_A 0x77 /* A */
#define SEGLED_B 0x7c /* b */
#define SEGLED_C 0x39 /* C */
#define SEGLED_D 0x5e /* d */
#define SEGLED_E 0x79 /* E */
#define SEGLED_F 0x71 /* F */
```

3.3.4 Compare operation (timer 1)

When a timer 1 capture/compare register is set as a compare register, the real-time pulse unit performs the following operations.

<1> For each timer count clock, compare the value of the compare register to the timer. If the values match, generate a match signal (INTCC1nm). An interrupt occurs if the interrupt source is made a match signal (INTCC1nm) in setting the TUM1n register.

<2> The compare register, which provides a timer output pin set/reset function, sets or resets timer output synchronized with a match signal (INTCC1nm).

Remarks: **1.** If the external input (INTP1n0 to INTP1n4) is selected as the interrupt source, set/reset output of pins TO1n0 and TO1n1 due to the match signal of a compare register (INTCC1nm) can be processed in parallel with acceptance of an interrupt request due to an external interrupt signal (INTP1n0 to INTP1n4).

2. n = 1 to 4, m = 0 to 3

Specify capture/compare register selection and interrupt source selection by using the CMS1nm bit and IMS1nm bit of TUM1n (n = 1 to 4, m = 0 to 3).

Figure 3-29: Timer Unit Mode Register 1n (TUM1n)

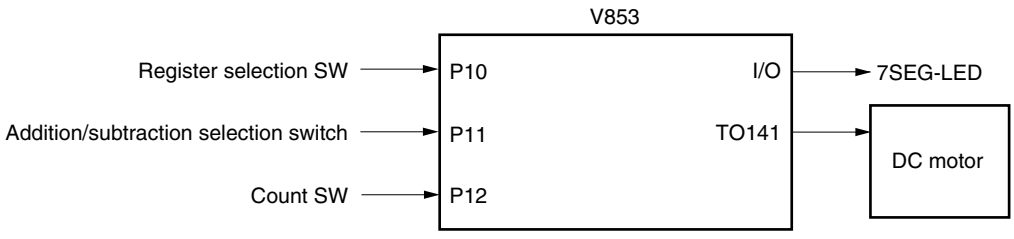
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	After reset
TUM1n	0	0	OSTn	ECLR 1n	TES 1n1	TES 1n0	CES 1n1	CES 1n0	CMS 1n3	CMS 1n2	CMS 1n1	CMS 1n0	IMS 1n3	IMS 1n2	IMS 1n1	IMS 1n0	FFFFF240H to FFFFF2A0H	0000H

Bit position	Bit name	Description
13	OSTn	Overflow Stop Specifies operation after timer overflow. See Figure 3-21.
12	ECLR1n	External Input Timer Clear Enables clearing of timer due to TM1n external clear input (TCLR1n). See Figure 3-21.
11, 10	TES1n1,TES1n0	TI1n Edge Select Specifies effective edge of external clock input (TI1n).
9, 8	CES1n1,CES1n0	TCLR1n Edge Select Specifies effective edge of external clear input (TCLR1n). See Figure 3-21.
7 to 4	CMS1n3 to CMS1n0	Capture/Compare Mode Select Selects operating mode of capture/compare register (CC1n0 to CC1n3). 0: Operate as capture register. However, capture operation when capture register is specified is only performed if CE1n bit of TMC1n is 1. 1: Operate as compare register.
3 to 0	IMS1n3 to IMS1n0	Interrupt Mode Select Selects INTP1nm or INTCC1nm as interrupt source (m = 0 to 3). 0: Make compare register match signal (INTCC1nm) the interrupt signal. 1: Make external input signal (INTP1nm) the interrupt signal.

Remark: n = 1 to 4

3.3.5 Compare operation sample program

Figure 3-30: Sample Hardware Configuration



(1) Function overview

- Control the DC motor using timer output instead of using the PWM function.
- The value of the compare register of the timer is changed by the count SW. Display the value of the compare register in the 7-segment LEDs.
- Whether to add or subtract a register value can be selected using the compare register whose value is updated using the register selection SW and input from the addition/subtraction selection SW and count SW.

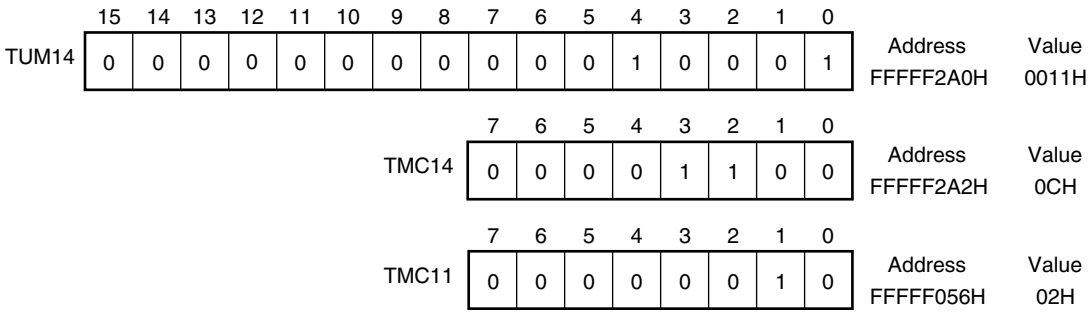
In the capture operation sample program, the DC motor outputs a control signal from PWM0. Here, the DC motor is controlled by using the compare register to set or reset the TO141 pin instead of using PWM0.

Set the TUM14 register and TMC14 register to the conditions shown below.

<1> In order to use the TO141 pin, set CC142 and CC143 to compare registers.
<2> Make the active level of the TO141 pin high level.
<3> The operation after the timer overflows is to continue to count up.
<4> Do not enable external clear input.
<5> Make the count clock ϕ /64 and select an internal clock.

Since the TO141 pin also is used as pin P111 of port 11, set pin P111 to control mode using register PMC11.

Figure 3-31: TUM14, TMC14, and PMC11 Settings



Set the signal output from the TO141 pin using the timer output control register (TOC14).
Since compare registers (CC142, CC143) are used in the sample program, set a value in TOC14.

Figure 3-32: Timer Output Control Register 14 (TOC14) Settings

	7	6	5	4	3	2	1	0	Address	After reset
TOC14	1	1	0	0	0	0	0	0	FFFFF2A4H	00H

Bit position	Bit name	Description
7, 5	ENTO141, ENTO140	Enable TO pin Enables corresponding timer output (TO140, TO141). 0: Timer output is disabled. Inverted phase level (inactive level) of bit ALV140 or ALV141 is output from corresponding TO140 or TO141 pin. Even if there is a match signal from the corresponding compare register, the level of pin TO140 or TO141 does not change. 1: Timer output function is enabled. Timer output changes if there is match signal from corresponding compare register. The inverted phase level (inactive level) of bit ALV140 or ALV141 is output until the first match signal after enabling timer output.
6, 4	ALV141, ALV140	Active Level TO pin Specifies active level of timer output. 0: Active level is low level. 1: Active level is high level.

Caution: TO140 and TO141 outputs do not change on an external interrupt signal (INTP140 to INTP143). When using TO140 and TO141 signals, specify the capture/compare register as a compare register (CMS bit = 1).

Remark: Flip-flops of TO140 and TO141 outputs give reset priority.

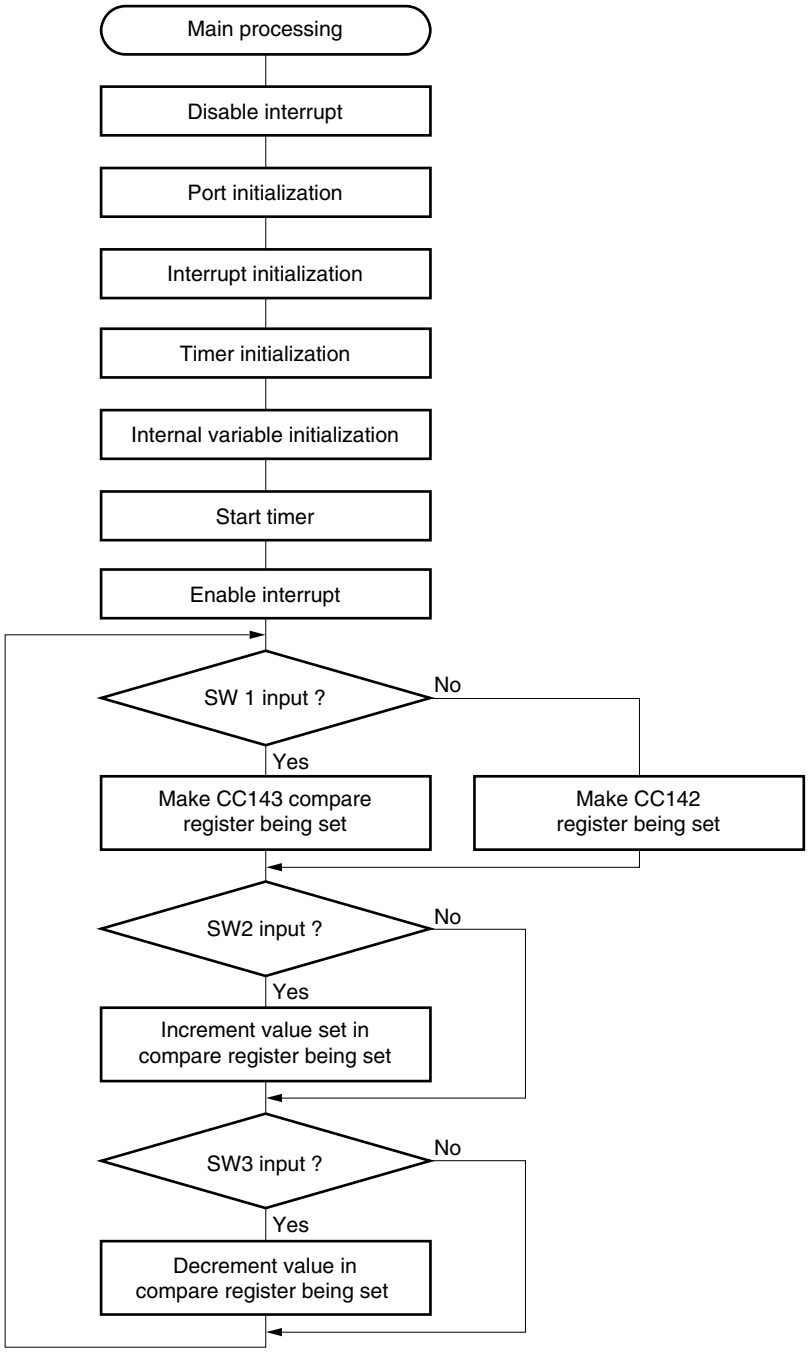
A function overview of PWM output processing in which timer output is used is shown below.

<1> Change timing of setting/resetting timer output according to SW.

<2> Select compare register to change (set/reset) and change value using increment SW or decrement SW.

<3> Display value of compare register in 7-segment LEDs.

Figure 3-33: PWM Control Flow



(2) Sample program

```
/* ————————————————————————-
 * V853 timer/counter initialization program
 *                   (compare function)
 * Initialization program for following hardware configuration
 *    P10: Register selection SW
 *    P11: Addition/subtraction selection SW
 *    P12: Count SW
 *    TO141: DC motor output
 *
 * ————————————————————————-
 * History:
 *    1997/3/17 Ver 0.00.00
 *    1999/7/8  adapted to IAR compiler by NEC Duesseldorf AH
 * File Name:   timcomp.c
 * ————————————————————————-
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

#include "7segLed.h"

static int SelFlg = 0;            /* Flag determining pulse width storage
location */
static int CntUpFlg = 0;          /* Counter start flag */
static int IncFlg = 0;            /* Count Up (!0)/Down(0) flag */
static unsigned int PulsWidth[2]; /* Pulse width storage variable */

/* ————————————————————————-
 * NMI service
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/

#pragma vector=NMI_vector

__flat __interrupt void int_nmi(void)
{
   ; /* Do nothing */
}

/* ————————————————————————-
 * Interval timer interrupt servicing
 * Interrupt source: INTCM4
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
```

```
#pragma vector=INTCM4_vector
__flat __interrupt void int_cm4(void)
{
   /* Is count up SW On? */
   if (CntUpFlg != 0) {
      /* Is increment SW On? */
      if (IncFlg != 0) {
         /* Increment pulse width */
         ++PulsWidth[SelFlg];
      } else {
         /* Decrement pulse width */
         —PulsWidth[SelFlg];
      }
      if (SelFlg ==1) {
         PulsWidth[0] = 0x10000 - PulsWidth[1];
      } else {
         PulsWidth[1] = 0x10000 - PulsWidth[0];
      }
   }
}

/* ———————————————————————-
 * Compare register match signal interrupt
 * Interrupt source: INTCC142
 * Arguments: None
 * Return value: None
 * ———————————————————————-
*/

#pragma vector=INTCC142_vector
__interrupt void int_cc142(void)
{
   /* Enable multiple interrupt */
   __EI();
   /* Set pulse width */
   CC142 = PulsWidth[0];
   /* Disable multiple interrupt */
   __DI();
}

/* ———————————————————————-
 * Compare register match signal interrupt
 * Interrupt source: INTCC143
 * Arguments: None
 * Return value: None
 * ———————————————————————-
*/
#pragma vector=INTCC143_vector
__interrupt void int_cc143(void)
{
   /* Enable multiple interrupt */
   __EI();
   /* Set pulse width */
   CC143 = PulsWidth[1];
   /* Disable multiple interrupt */
   __DI();
}
```

```
/* ————————————————————————-
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
void PortInit(void)
{
   /* Initialize port 0 (P0)
   */
   PM0 = 0xFF; /* All ports in input mode */
   PMC0 = 0x00; /* P00 to P07 in port mode */

   /* Initialize port 1 (P1)
   */
   PM1 = 0xFF; /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize Port 2 (P2)
   */
   PM2 = 0xFF; /* All ports in input mode */
   PMC2 = 0x00; /* P20 to P27 in port mode */

   /* Initialize port 3 (P3)
   */
   PM3 = 0x00; /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00; /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */
   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x03; /* Use TO140 and TO141 PMC110, PMC111 in control mode */
                 /* P112 to P117 in port mode */
}
```

```
/* ————————————————————————-
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
void IntInit(void)
{
   /* Interrupt register settings (set interrupt levels) */
   P14IC2 = 0x07; /* INTCC142: CC142 match signal: Level 7 */
   P14IC3 = 0x07; /* INTCC143: CC143 match signal: Level 7 */
   CMIC4  = 0x06; /* INTCM4 : CM4 match signal : Level 6 */

   /* Set interrupt effective edges */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x00; /* Not used */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}

/* ————————————————————————-
 * Timer initialization processing
 * Arguments : None
 * Return value: None
 * ————————————————————————-
*/
void TimerInit(void)
{
   /* Set timer 14 (TUM14) */
   TUM14 = 0x00c0; /* OST4 = 0: Continue count up after timer overflow */
                   /* ECLR14 = 0: Do not enable external clear input */
                   /* TES141 and TES140 = 0: Falling edge */
                   /* CMS143 and CMS142 = 1: Make compare register */
                   /* IMS143 to IMS140 = 0 : Interrupt source INTCC */

   /* Set timer 14 (TMC14) */
   TMC14 = 0x06; /* ETI14 = 0: Timer count clock is internal clock */
                 /* PRM141 = 1: Intermediate clock is 1/4 system clock */
                 /* PRS141 = 0: Timer count clock is f m/4 */
                 /* PRS140 = 1: (Timer count clock = f/16) */
                 /* CE14 = 0: Stop timer */

   /* Set timer 14 output control (TOC14) */
   TOC14 = 0xc0; /* ENTO141 = 1: Timer output function enabled */
                 /* ALV141 = 1: Active level is high level */

   /* Set compare registers to initial values */
   CC142 = 0x4000;
   CC143 = 0xC000;

   /* Set timer 4 */
   TMC4 = 0x05;  /* Count clock = f/128 */
   CM4 = 0x9896; /* 0.2-second approx. interval timer */
}
```

```
/* ————————————————————————-
 * 7-segment LED display processing
 * Arguments: disp_data Display data
   dot_pos Decimal point display position (1-LEDMAX)
   mode Decimal/hexadecimal specification (0: Decimal, 1: Hex)
 * Return value: None
 * ————————————————————————-
*/
void Disp7SegLED(unsigned long disp_data, int dot_pos, int mode)
{
   register unsigned char *segaddr = SEG7ADDR; /* 7-segment LED set address
*/
   register unsigned char cnt; /* Work counter */
   register int base; /* Decimal or hexadecimal radix */

   /* Set radix */
   if (mode == 0) {
      base = 10; /* Decimal display */
   } else {
      base = 16; /* Hexadecimal display */
   }
   /* Set all columns of 7-segment LEDs */
   for ( cnt = 1; cnt <= LEDMAX ; cnt ++ ) {
      /* Display 7-segment LEDs */
      switch ( disp_data % base ) {
         case 0:  *segaddr = SEGLED_0; break;
         case 1:  *segaddr = SEGLED_1; break;
         case 2:  *segaddr = SEGLED_2; break;
         case 3:  *segaddr = SEGLED_3; break;
         case 4:  *segaddr = SEGLED_4; break;
         case 5:  *segaddr = SEGLED_5; break;
         case 6:  *segaddr = SEGLED_6; break;
         case 7:  *segaddr = SEGLED_7; break;
         case 8:  *segaddr = SEGLED_8; break;
         case 9:  *segaddr = SEGLED_9; break;
         case 10: *segaddr = SEGLED_A; break;
         case 11: *segaddr = SEGLED_B; break;
         case 12: *segaddr = SEGLED_C; break;
         case 13: *segaddr = SEGLED_D; break;
         case 14: *segaddr = SEGLED_E; break;
         case 15: *segaddr = SEGLED_F; break;
      }
      /* Create data of next column */
      disp_data /= base;
      /* If displaying decimal point */
      if ( cnt == dot_pos ) {
         /* Display 7-segment LEDs */
         *segaddr += SEGLED_DOT;
      }
      /* Create set address of next column */
      segaddr += 2;
   }
}
```

```
/* ————————————————————————-
 * Main processing
 * Arguments : None
 * Return value: None
 * ————————————————————————-
*/
void main(void)
{
   unsigned long width;

   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* Timer initialization processing */
   TimerInit();

   /* Internal data initialization */
   SelFlg = 0;
   PulsWidth[0] = 0x4000;
   PulsWidth[1] = 0xC000;

   /* Start timers */

   TMC14_bit.no7= 1;    /* Set Bit CE14, start timer 14 */

   /* Enable interrupt */
   __EI();

   while ( 1 ) {
      /* Input pulse width setting switching SW */
      SelFlg = P1_bit.no0;
      /* Input count setting switching SW (On: Increment, Off: Decrement) */
      IncFlg = P1_bit.no1;
      /* Input count SW (On: Count pulse width value up/down) */
      CntUpFlg = P1_bit.no2;

      width = PulsWidth[SelFlg];
      /* Display set pulse width */
      Disp7SegLED(width, 0, 1);
   }
}
```

```
/* ----------------------------------
* TU-V853 7-segment LED register definition
* ----------------------------------
* History:
* 1997/3/17 Ver 0.00.00
* File Name: 7segled.h
* ----------------------------------
*/

/* 7-segment LED data */
#define LEDMAX 5 /* Number of columns used */

/* Address of first column */
#define SEG7ADDR (unsigned char *) 0x00460000L

/* Segment data */
#define SEGLED_DOT 0x80 /* Point */
#define SEGLED_0 0x3f /* 0 */
#define SEGLED_1 0x06 /* 1 */
#define SEGLED_2 0x5b /* 2 */
#define SEGLED_3 0x4f /* 3 */
#define SEGLED_4 0x66 /* 4 */
#define SEGLED_5 0x6d /* 5 */
#define SEGLED_6 0x7d /* 6 */
#define SEGLED_7 0x07 /* 7 */
#define SEGLED_8 0x7f /* 8 */
#define SEGLED_9 0x6f /* 9 */
#define SEGLED_A 0x77 /* A */
#define SEGLED_B 0x7c /* b */
#define SEGLED_C 0x39 /* C */
#define SEGLED_D 0x5e /* d */
#define SEGLED_E 0x79 /* E */
#define SEGLED_F 0x71 /* F */
```

3.3.6 Configuration of timer 4

Timer 4 operates as a 16-bit interval timer.

The only count clock is an internal clock. Operation settings are specified by using timer control register 4 (TMC4).

Figure 3-34: Timer Control Register 4 (TMC4)

	7	6	5	4	3	2	1	0	Address	After reset
TMC4	CE4	0	0	0	0	PRS40	PRM41	PRM40	FFFFF342H	00H

Bit position	Bit name	Description
7	CE4	Count Enable Controls timer operation. 0: Timer stops in "0000H" status and does not operate. 1: Timer performs count operation.
2	PRS40	Prescaler Clock Select Selects internal count clock (ϕ m is the intermediate clock) 0: ϕ m 1: ϕ m/32
1, 0	PRM41, PRM40	Prescaler Clock Mode Selects count clock intermediate clock (ϕ m) (where ϕ: internal system clock).

PRM41	PRM40	ϕ m
0	0	ϕ/2
0	1	ϕ/4
1	0	ϕ/16
1	1	ϕ/32

Caution: Do not change the count clock during timer operation.

Select count clocks shown in Table 3-6 by setting bits PRS40, PRM40, and PRM41 of TMC4.

Table 3-6: Count Clock Selection

PRS40	PRM41	PRM40	Count clock
0	0	0	ϕ/2
0	0	1	ϕ/4
0	1	0	ϕ/16
0	1	1	ϕ/32
1	0	0	ϕ/64
1	0	1	ϕ/128
1	1	0	ϕ/512
1	1	1	ϕ/1024

Remark: ϕ**:** Internal system clock

3.3.7 Operation of interval timer (timer 4)

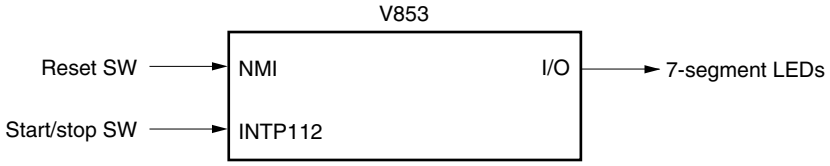
For each internal count clock specified by TMC4, compare the compare register (CM4) to timer 4 (TM4). If CM4 matches TM4, a match interrupt occurs and TM4 is cleared.

If TM4 overflows as a result of counting the count clock, bit OVF4 of TOVS is set (1).

There is no overflow interrupt in TM4.

3.3.8 Interval timer operation sample program

Figure 3-35: Sample Hardware Configuration

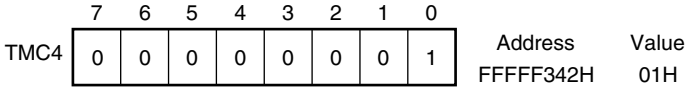


(1) Function overview

- Make a 100-ms unit stopwatch by generating a 1-ms interval timer using timer 4 (TM4).
- Display the value of the 100-ms counter in the 7-segment LEDs and operate the start/stop SW and reset SW by using interrupts.

Set TMC4 and CM4 as follows to generate a 1-ms interval timer by using TM4.

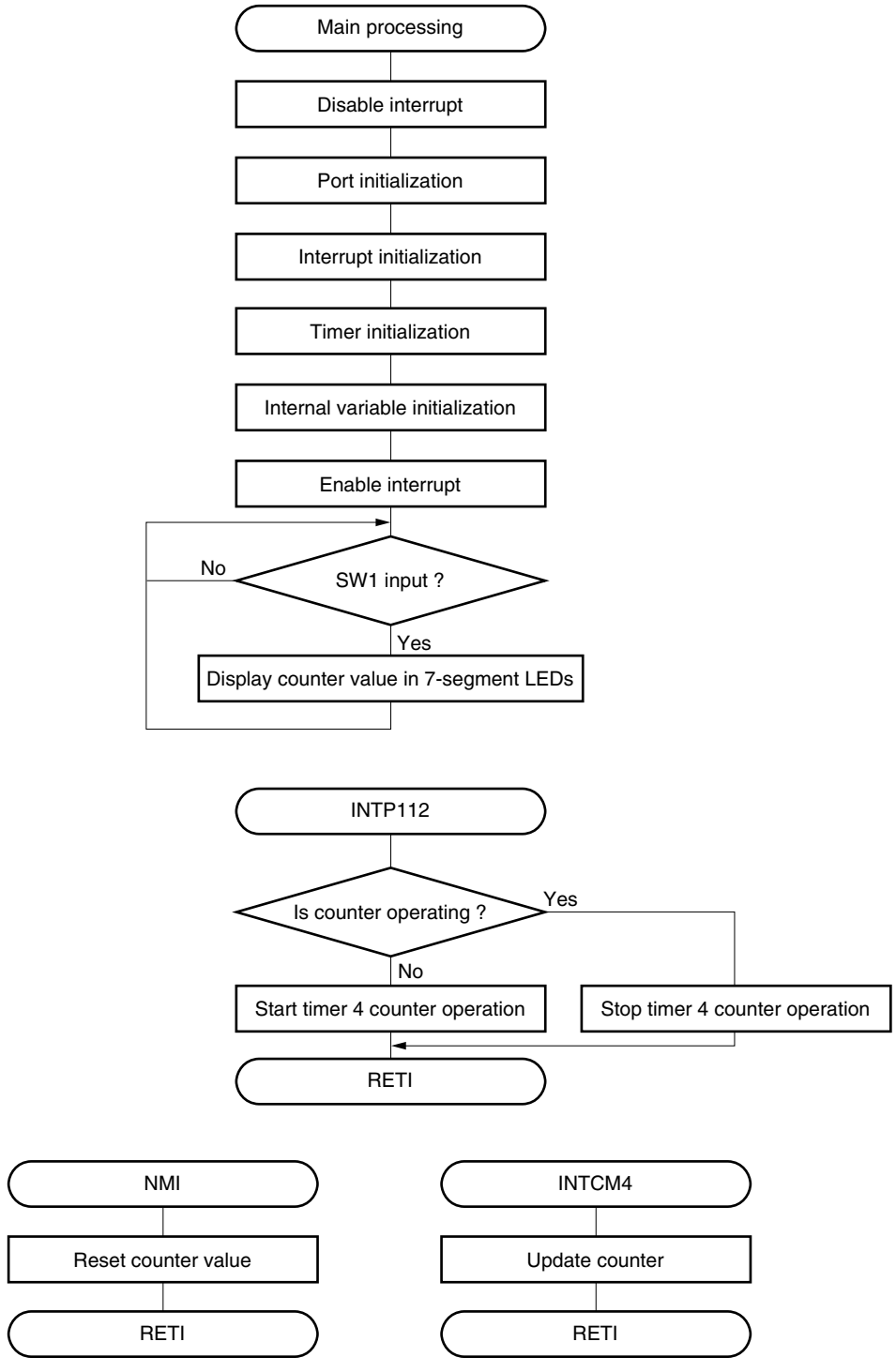
Figure 3-36: Setting TMC4



Set CM4 to the value generated by INTCM4's interrupt when the count clock of ϕ/4 counts up to 1 ms. The value that is set differs according to the operation clock.

CM4 = 8250 – 1 (Operation clock = 33 MHz)
= 6520 – 1 (Operation clock = 25 MHz)

Figure 3-37: Flow of Stopwatch Processing



(2) Sample program

```
/* ————————————————————————————————————————-
 * V853 program
 *    Display measured time in 7-segment LEDs of training-board (create stop-
watch)
 *
 * Interrupts: NMI    : Stop/reset
 *             INTCM4 : Measure elapsed time
 *             INTP112: Switch between counter operation/stopping
 *
 * ————————————————————————————————————————-
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/8  adapted to IAR compiler by NEC Duesseldorf AH
 * File Name: stpwatch.c
 * ————————————————————
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

#include “stpwatch.h” /* Header file */

/* ——————————————-
 * Timer data reset processing
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ——————————————-
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   tdata.timecnt = 0L; /* Clear time data */
   tdata.onems = 0L;   /* Clear counter every 1 ms */
}

/* ——————————————-
 * Set elapsed time
 * Interrupt source: INTCM4
 * Arguments: None
 * Return value: None
 * ——————————————-
*/

#pragma vector=INTCM4_vector
__flat __interrupt void int_cm4(void)
{
   tdata.onems ++;           /* Increment number of counter interrupts */
   if ( tdata.onems == 100 ){/* When 100 ms is reached */
      tdata.timecnt ++;      /* Reflect in time data */
      tdata.onems = 0L;      /* Clear number of counter interrupts */
   }
}
```

```
/* ————————————————-
 * Switching counter operation and stopping
 * Interrupt source: INTP112
 * Arguments: None
 * Return value: None
 * ————————————————-
*/
#pragma vector=INTP112_vector
__flat __interrupt void int_p112(void)
{
   /* Switch counter operation mode */
   if (tdata.exec == 1) { /* Counter is operating */
      /* Processing to stop operation */
      TMC4_bit.no7 = 0;   /* Clear Bit CE4 */
      /* Stop counter */
      tdata.exec = 0;
   }
   else { /* Counter is stopped */
     /* Processing to start operation */
     TMC4_bit.no7 = 1;    /* Set Bit CE4 */
     tdata.exec = 1;
   }
}

/* ————————————————-
 * 7-segment LED display processing
 * Arguments: unsigned long disp_data Display data
              int      dot_pos        Decimal point display position (1-
LEDMAX)
 * Return value: None
 * ————————————————-
*/
void disptimer(
   unsigned long disp_data , /* Display data */
   int dot_pos               /* Decimal point display position (1-LEDMAX) */
)
{
   unsigned char *segaddr = SEG7ADDR; /* 7-segment LED set address */
   unsigned char cnt;                 /* Work counter */
   unsigned char wkdata;              /* For data creation */

   /* Set all columns of 7-segment LEDs */
   for (cnt = 1 ; cnt <= LEDMAX ; cnt ++ ) {
      /* Create 7-segment LED data */
      switch ( disp_data % 10 ) {
         case 0: wkdata = SEGLED_0; break;
         case 1: wkdata = SEGLED_1; break;
         case 2: wkdata = SEGLED_2; break;
         case 3: wkdata = SEGLED_3; break;
         case 4: wkdata = SEGLED_4; break;
         case 5: wkdata = SEGLED_5; break;
         case 6: wkdata = SEGLED_6; break;
         case 7: wkdata = SEGLED_7; break;
         case 8: wkdata = SEGLED_8; break;
         case 9: wkdata = SEGLED_9; break;
      }
```

```
      /* If displaying decimal point */
      if ( cnt == dot_pos ) {
         wkdata |= SEGLED_DOT;
      }
      /* Display 7-segment LEDs */
      *segaddr = wkdata;
      /* Create data of next column */
      disp_data /= 10;
      /* Create set address of next column */
      segaddr += 2;
   }
}

/* ------------------
* Port initialization processing
* Arguments: None
* Return value: None
* ------------------
*/
void PortInit(void)
{
   /* Port 0 (P0) initialization */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x40; /* Use INTP112 P06 in control mode */
                /* P00 to P05, P07 in port mode */

   /* Port 1 (P1) initialization */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Port 2 (P2) initialization */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x00; /* P20 to P27 in port mode */

   /* Port 3 (P3) initialization */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Port 4 (P4) initialization */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Port 5 (P5) initialization */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Port 6 (P6) initialization */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/
```

```
   /* Port 9 (P9) initialization */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Port 11 (P11) initialization */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}

/* ————————————————-
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————-
*/
void IntInit(void)
{
   /* Interrupt register settings (set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: Emergency stop switch */
   CMIC4 = 0x07; /* INTCM4: Set interval timer */

   /* Set interrupt effective edge */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 - 0x10; /* INTP112: Rising edge */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}

/* ————————————————-
 * Interval timer initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————-
*/
void TimerInit(void)
{
   /* Set interrupt every 1 ms (setting at 33 MHz) */
   /* PRS40, PRM40, PRM41, CE4 are all bits of Register TMC 4 */
   PRS40 = 0; /* Count clock setting f /4 */
   PRM40 = 1; /* Count clock setting f /4 */
   PRM41 = 0; /* Count clock setting f /4 */
   CM4 = 8250 - 1; /* Comparison counter */
   CE4 = 0;
}
```

```
/* ————————————————-
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————-
*/
void main(void)
{
   unsigned long tmpcnt; /* Work variable */

   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* Timer initialization processing */
   TimerInit();

   /* Internal variables initialization */
   tdata.exec = 0;              /* Initialize execute flag (stop) */
   tdata.onems = 0L;             /* Clear number of counter interrupts */
   tdata.timecnt = 0L;          /* Initialize time data */
   tdata.timeold = 0xffffffffL; /* Initialize previous time data */
                                /* (to display first data) */
   /* Enable interrupt */
   __EI();

   /* Monitor time data */
   while ( 1 ) {
      /* If counter value changed */
      tmpcnt = tdata.timecnt;
      if ( tmpcnt != tdata.timeold ) {
         disptimer( tmpcnt , 2 ); /* LED display processing */
         tdata.timeold = tmpcnt;  /* Hold display counter */
      }
   }
}
```

```
/* ------------------
* STPWTCH.C header file
* ------------------
* History:
* 1997/3/17 Ver 0.00.00
* File Name: stpwatch.h
* ------------------
*/

/* 7-segment LED data */
#define LEDMAX 5 /* Number of columns used */

/* Address of first column */
#define SEG7ADDR (unsigned char *)0x00460000L

/* Segment data */
#define SEGLED_DOT 0x80 /* Point */
#define SEGLED_0 0x3f /* 0 */
#define SEGLED_1 0x06 /* 1 */
#define SEGLED_2 0x5b /* 2 */
#define SEGLED_3 0x4f /* 3 */
#define SEGLED_4 0x66 /* 4 */
#define SEGLED_5 0x6d /* 5 */
#define SEGLED_6 0x7d /* 6 */
#define SEGLED_7 0x07 /* 7 */
#define SEGLED_8 0x7f /* 8 */
#define SEGLED_9 0x6f /* 9 */

/* Time measurement structure */
struct timedata
{
  unsigned char exec; /* Counter execute/stop flag */
  unsigned long onems; /* 1-ms counter (count every 1 ms) */
  unsigned long timecnt; /* Time counter (100 ms units) */
  unsigned long timeold; /* Displayed time counter */
};
struct timedata tdata;
```

3.4 Serial Interface Functions

The V853 has six transmit/receive channels of two types as serial interface functions. Up to four channels can be used simultaneously. The two kinds of interfaces are shown below.

- Asynchronous serial interface (UART0, UART1): 2 channels
- Clock synchronous serial interface (CSI0 to CSI3): 4 channels

Pins are shared between the two asynchronous serial interface channels and two of the clock synchronous serial interface channels. (Pins are shared between UART0 and CSI0, and between UART1 and CSI1.)

UART and CSI options are specified using the asynchronous serial interface mode registers (ASIM00 and ASIM10).

3.4.1 Asynchronous serial interface (UART0, UART1)

Table 3-7 shows the features of the asynchronous serial interface.

Table 3-7: Features of Asynchronous Serial Interface

Item	Features
Transfer speed	150 bps to 76800 bps (Using a baud rate generator, at ϕ = 33 MHz) 110 bps to 307200 bps (Using a baud rate generator, at ϕ = 20 MHz) Maximum 1031 kbps (Using ϕ/2, at ϕ = 33 MHz)
Full duplex communication	On-chip receive buffer (RXBn)
2-pin configuration	TXDn: Transmit data output pin RXDn: Receive data input pin
Receive error detection function	• Parity error • Framing error • Overrun error
3 kinds of interrupt source	• Receive error interrupt (INTSERn) • Receive completion interrupt (INTSRn) • Transmit completion interrupt (INTSTn)
Character length (specified by ASIM register)	7 or 8 bits 9 bits (when extended bit is attached)
Parity function	Odd, even, 0, none
Transmit stop bits	1 or 2 bits
On-chip baud rate generator	—

Remark: n = 0, 1
ϕ: Internal system clock

There are no control pins (such as CTS pins) in the asynchronous serial interface. To confirm that the intended recipient is receive enabled, perform status notification using a general purpose port.

The asynchronous serial interface mode registers (ASIM00 and ASIM10) that make each asynchronous serial interface setting are shown below.

Figure 3-38: Asynchronous Serial Interface Mode Registers 00 and 10 (ASIM00, ASIM10) (1/2)

	7	6	5	4	3	2	1	0	Address	After reset
ASIM00	TXE0	RXE0	PS01	PS00	CL0	SL0	0	SCLS0	FFFFF0C0H	00H
ASIM10	TXE1	RXE1	PS11	PS10	CL1	SL1	0	SCLS1	FFFFF0D0H	00H

Bit position	Bit name	Description
7, 6	TXEn, RXEn	Transmit/Receive Enable Specifies transmit and receive enable status or disable status. TXEn / RXEn / Operation: 0 / 0 / Disable transmit (Select CSIn) 0 / 1 / Enable receive 1 / 0 / Enable transmit 1 / 1 / Enable transmit and receive When receive disabled, the receive shift register does not perform start bit detection. Shift in processing and receive buffer transmission processing are not performed and the contents of the receive buffer are maintained. In receive enabled status, receive shift operation starts in synchronization with start bit detection and the contents of the receive shift register are transferred to the receive buffer when one frame has been received. In addition, a receive completion interrupt (INTSRn) is generated synchronized with the transfer to the receive buffer. The TXDn pin, which becomes high impedance when transmit is disabled, outputs high level when transmit is enabled and not transmitting.
5, 4	PSn1, PSn0	Parity Select Specifies the parity bit. PSn1 / PSN0 / Operation: 0 / 0 / No parity, extended bit operation 0 / 1 / Specifies 0 parity Transmitting end ∅ Transmit making parity bit 0 Receiving end ∅ Do not generate parity error when receiving 1 / 0 / Specifies odd parity 1 / 1 / Specifies even parity
3	CLn	Character Length Specifies character length of 1 frame. 0: 7 bits 1: 8 bits
2	SLn	Stop Bit Length Specifies stop bit. 0: 1 bit 1: 2 bits

Caution: Operation is not guaranteed if these registers are changed while UARTn is transmitting or receiving.

Remark: n = 0, 1

Figure 3-38: Asynchronous Serial Interface Mode Registers 00 and 10 (ASIM00, ASIM10) (2/2)

	7	6	5	4	3	2	1	0	Address	After reset
ASIM00	TXE0	RXE0	PS01	PS00	CL0	SL0	0	SCLS0	FFFFF0C0H	00H
ASIM10	TXE1	RXE1	PS11	PS10	CL1	SL1	0	SCLS1	FFFFF0D0H	00H

Bit position	Bit name	Description
0	SCLSn	Serial Clock Source Specifies the serial clock. 0: Specify using BRGCn, BPRMn 1: ϕ/2 When SCLSn = 1 Select ϕ/2 (system clock) as serial clock source. Because sixteen-fold sampling rate is used in asynchronous mode, the baud rate is represented by the following formula. Baud rate = $\frac{\phi/2}{16}$ bps Based on the formula above, the baud rate values when typical clocks are used are shown below. ϕ | 33MHz | 25MHz | 20MHz | 16MHz | 12.5MHz | 10MHz | 8MHz | 5MHz Baud rate | 1031K | 781K | 625K | 500K | 390K | 312K | 250K | 156K When SCLSn = 0 Select baud rate generator output as serial clock source. See **3.4.5 Baud rate generator (BRG0 to BRG2)** regarding the baud rate generator.

Caution: Operation is not guaranteed if these registers are changed while UARTn is transmitting or receiving.

Remark: n = 0, 1

ϕ: Internal system clock

Figure 3-39: Asynchronous Serial Interface Mode Registers 01 and 11 (ASIM01, ASIM11)

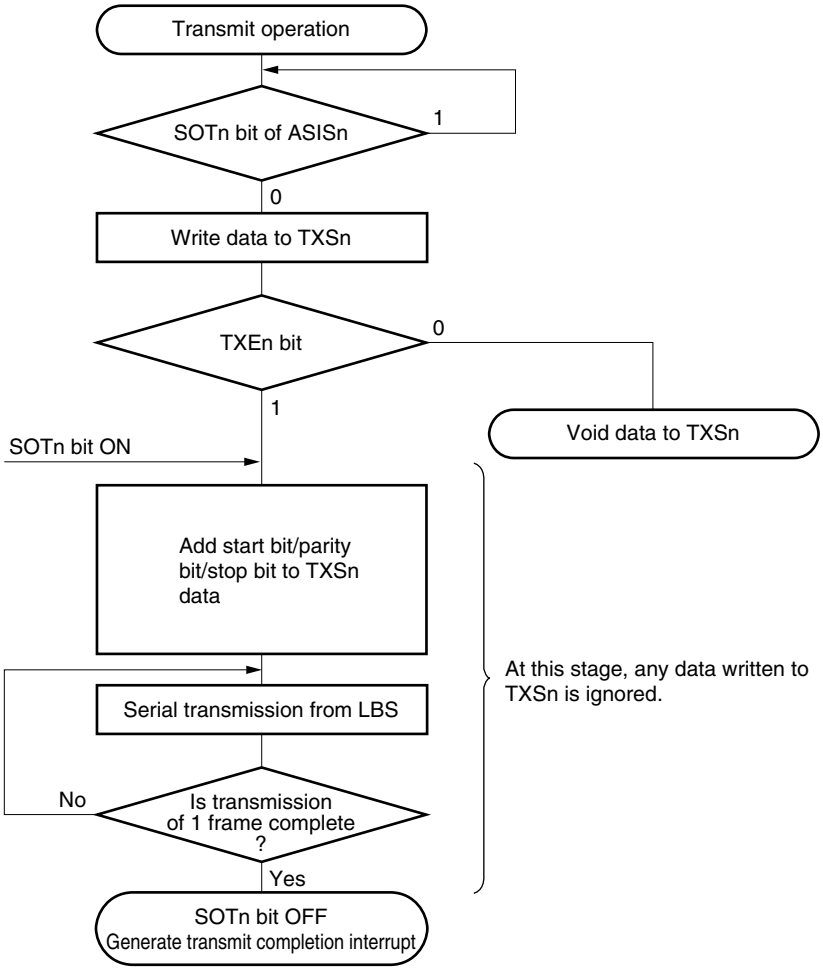
	7	6	5	4	3	2	1	0	Address	After reset
ASIM01	0	0	0	0	0	0	0	EBS0	FFFFF0C2H	00H
ASIM11	0	0	0	0	0	0	0	EBS1	FFFFF0D2H	00H

Bit position	Bit name	Description
0	EBSn	Extended Bit Select Specifies extended bit operation for transmit and receive data when no-parity operation is specified (PSn1, PSn0 = 00). 0: Disable extended bit operation 1: Enable extended bit operation When extended bit is specified, 1 data bit is added to the high end of 8-bit transmit and receive data to make communication using 9-bit data possible. Extended bit operation is in effect only when no-parity operation is specified by the ASIMn0 register. If 0 parity or odd/even parity operation is specified, the specification of the EBSn bit is void and an extended bit is not added.

Remark: n = 0, 1

Figure 3-40 shows an asynchronous serial interface transmit operation.

Figure 3-40: Asynchronous Serial Interface Transmit Operation



Remark: n = 0, 1

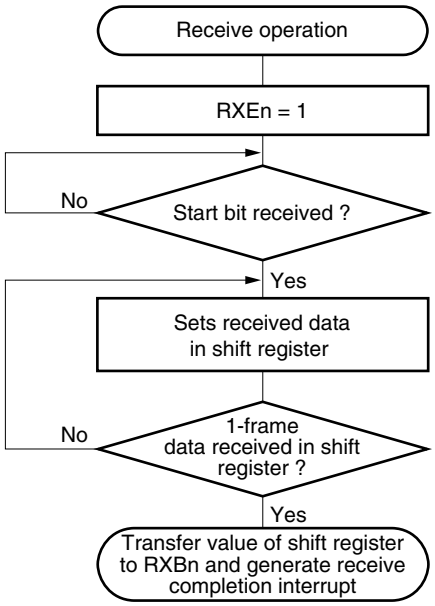
A transmit completion interrupt occurs when one frame of data has been sent from the transmit shift register.

If data is written to the register while data is being sent from the transmit shift register, the data is ignored.

To use transmit enabled status (TXEn = 1), set the CTXEn and CRXEn bits of the clock synchronous serial interface mode register (CSIMn) which shares a pin with an asynchronous serial interface to 0 (n = 0, 1).

Figure 3-41 shows an asynchronous serial interface receive operation.

Figure 3-41: Asynchronous Serial Interface Receive Operation



Remark: n = 0, 1

A receive completion interrupt is generated when one frame of data is set in the receive shift register and transferred to the receive buffer. Therefore, if there is an overrun error, the overrunning data overwrites the receive buffer and the previous data is erased.

To use receive enabled status (RXEn = 1), set the CTXEn and CRXEn bits of the clock synchronous serial interface mode register (CSIMn) which is used as an asynchronous serial interface and a pin to 0 (n = 0, 1).

If a parity error, framing error, or overflow error is detected after completion of a receive operation, a receive error interrupt request is generated. These errors are confirmed when status registers (ASISn) are read (n = 0, 1).

Figure 3-42: Asynchronous Serial Interface Status Registers 0 and 1 (ASIS0, ASIS1)

	7	6	5	4	3	2	1	0	Address	After reset
ASIS0	SOT0	0	0	0	0	PE0	FE0	OVE0	FFFFF0C4H	00H
ASIS1	SOT1	0	0	0	0	PE1	FE1	OVE1	FFFFF0D4H	00H

Bit position	Bit name	Description
7	SOTn	Status of Transmission This is a status flag that shows the transmit operation status. Set (1): Transmission start timing (writing to TXSn or TXSnL registers) Clear (0): Transmission end timing (generating INTSTn interrupt) Used as a means of distinguishing whether or not it is possible to write to the transmit shift register when serial data transmission is to begin.
2	PEn	Parity Error This is a status flag that indicates a parity error. Set (1): Transmit parity and receive parity do not match Clear (0): Data read processing from receive buffer
1	FEn	Framing Error This is a status flag that indicates a framing error. Set (1): Stop bit was not detected Clear (0): Data read processing from receive buffer
0	OVEn	Overrun Error This is a status flag that indicates an overrun error. Set (1): UARTn completed next receive processing before received data was fetched from receive buffer Clear (0): Received data read processing from receive buffer Because receive shift register contents are transferred to the receive buffer each time 1 frame is received, when an overrun error occurs the next received data overwrites the receive buffer and the previously received data is destroyed.

Remark: n = 0, 1

Status flags indicating receive errors always show the status of the error that occurs the most recently. Thus, even if the same error occurs multiple times before reading received data, they maintain only the status of the error that occurs last.

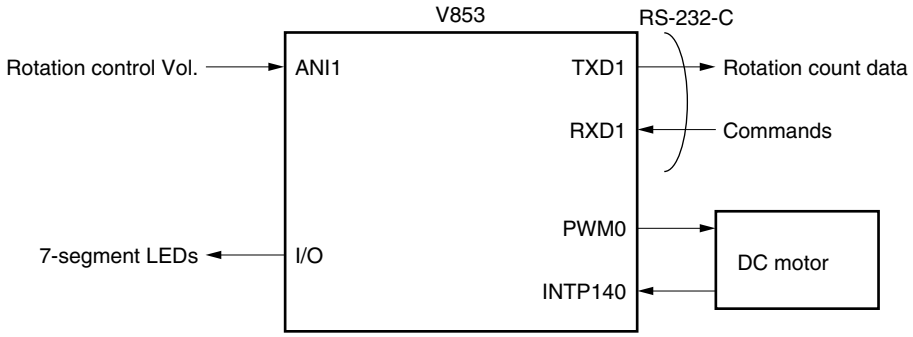
The status registers (ASISn) are reset when the receive buffer (RXBn, RXBnL) is read**Note** or the next data of 1 frame is received (n = 0, 1).

★ **Note:** If RXBn or RXBnL isn't read, an overrun error occurs at the reception of the next data, and the reception error status continues.

3.4.2 Asynchronous serial interface sample program

The sample program below sets the asynchronous serial interface based on the following hardware configuration.

Figure 3-43: Sample Hardware Configuration



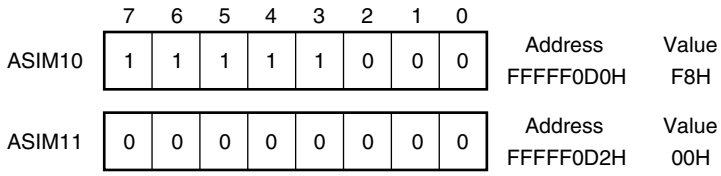
(1) Function overview

- Cause the DC motor to turn using the PWM function from the value A/D input using the rotation control Vol.
- Count the output pulses from the DC motor and measure the number of rotations of the motor. Output from the motor is 12 pulses per rotation.
- Display the measured number of rotations in the 7-segment LEDs.
- The number of rotations of the motor can be obtained from outside using asynchronous serial communication (UART). It also is possible to know the operating condition of the device.

Create a DC motor rotations monitor using the asynchronous serial interface and the sample hardware.
Figure 3-44 shows the settings of the asynchronous serial interface mode registers (ASIM10 and ASIM11) when the asynchronous serial communication protocol is as follows.

- Protocol
  Transfer speed: 9600 bps
  Transfer data length: 8 bits
  Stop bit: 1 bit
  Parity: Even parity
- Use baud rate generator for serial clock

Figure 3-44: ASIM10 and ASIM11 Settings



The serial clock uses what is generated by the baud rate generator. See **3.4.5 Baud rate generator (BRG0 to BRG2)** regarding the baud rate generator.

In order to generate 9600 bps, set baud rate generator compare register 1 (BRGC1) and baud rate generator prescaler mode register 1 (BPRM1) to the following values.

BPRM1 = 80H (count operation enabled, count clock = ϕ /2)

BRGC1 = 54 (operation clock = 33 MHz)
= 41 (operation clock = 25 MHz)

Format of commands transmitted and received via UART

- Number of motor rotations request command (external device to V853) “R”
- Number of motor rotations status (V853 to external device) “= number-of-rotations”

Remark: Number-of-rotations is in decimal notation using 5 bytes of ASCII characters (00000 to 65535).

Figure 3-45: DC Motor Rotations Monitor (1/2)

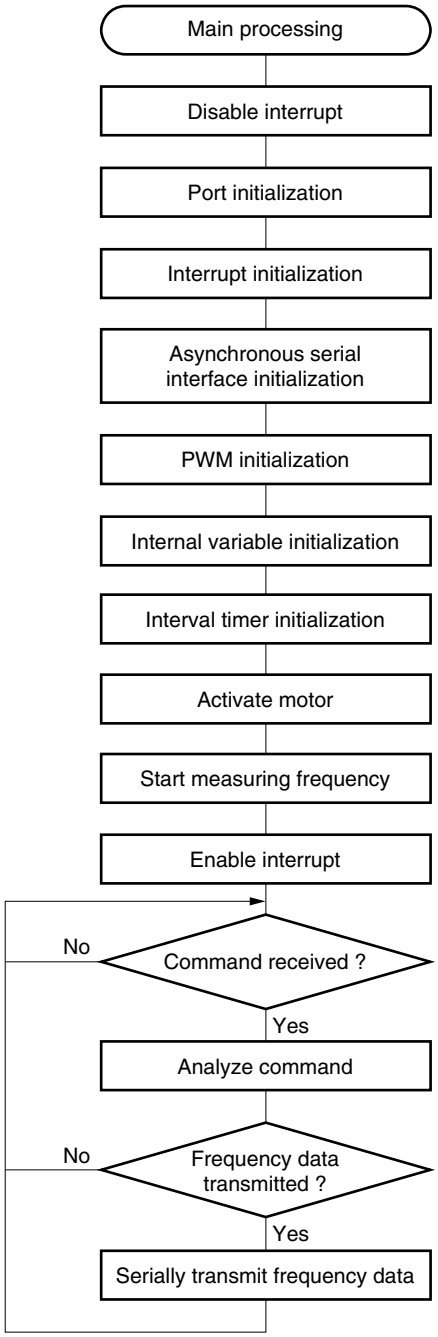
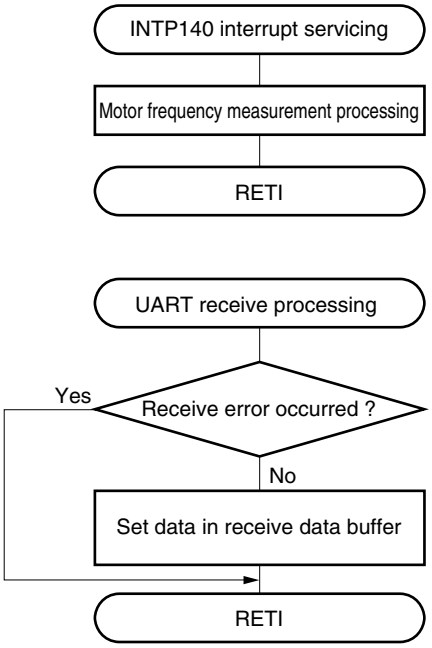


Figure 3-45: DC Motor Rotations Monitor (2/2)



(2) Sample program

```
/* ————————————————————————————————————-
 * V853 program
 * DC motor rotation frequency monitor
 *
 * Interrupts: INTSER1: Receive error
 *             INTSR1 : Receive completion
 *             INTST1 : Transmit completion
 *             INTP140: DC motor rotation frequency monitor processing
 *             INTCM4 : Set elapsed time
 *             INTAD  : A/D conversion completion
 *
 * ————————————————————————————————————-
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/8  adapted to IAR compiler by NEC Duesseldorf AH
 * File Name dcmotor.c
 * ————————————————————————
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

#include "dcmotor.h" /* Header file */

/* Operation flag */
static short led_disp = 1; /* 7-segment LED display flag */

/* ——————————————————
 * NMI servicing
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ——————————————————
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   ; /* Do nothing */
}

/* ————————————————————————————————————-
 * Asynchronous serial interface: Receive error interrupt service
 * Interrupt source: INTSER1
 * Arguments: None
 * Return value: None
 * ————————————————————————————————————-
*/
```

```
#pragma vector=INTSER1_vector
__flat __interrupt void Int_ser1(void)
{
   unsigned short tmp; /* Work variable */

   /* Set error flag */
   /*  uartdt.err_flg = ASIS1 & 0x87; */
   uartdt.err_flg = ASIS1 & 0x87;
   /* Clear error flag by reading */
   tmp = RXB1;
/*   ASIS1 &= 0xF8; */
}

/* ————————————————————————————-
 * Asynchronous serial interface: Receive completion interrupt service
 * Interrupt source: INTSR1
 * Arguments: None
 * Return value: None
 * ————————————————————————————-
*/
#pragma vector=INTSR1_vector
__flat __interrupt void int_sr1(void)
{
   /* Place and maintain received data in receive buffer */
   uartdt.rd_buf[uartdt.rd_pos++] = RXB1L;
   if (uartdt.rd_pos >= RDBUFSIZE ) {
      uartdt.rd_pos = 0;
   }
}

/* ————————————————————————————-
 * Asynchronous serial interface: Transmit interrupt service
 * Interrupt source: INTST1
 * Arguments: None
 * Return value: None
 * ————————————————————————————-
*/
#pragma vector=INTST1_vector
__flat __interrupt void int_st1(void)
{
   if (uartdt.wt_cnt != uartdt.wt_pos ) { /* There is transmit data */
      /* Transmit data */
      TXS1L = uartdt.wt_buf[uartdt.wt_cnt++];
   }
   else {
      uartdt.wt_flg = 1; /* Transmit completion */
      TXE1 = 0;          /* Transmit disable */
      RXE1 = 1;          /* Receive enable */
   }
}
```

```
/* ————————————————————
 * DC motor rotation frequency monitor processing
 * Interrupt source: INTP140
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTP140_vector
__flat __interrupt void int_p140(void)
{
   pwmdt.int_cnt ++;
}

/* ————————————————————
 * Setting elapsed time
 * Interrupt source: INTCM4
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTCM4_vector
__flat __interrupt void int_cm4 (void)
{
   /* Update PWM motor rotation count */
   pwmdt.tim_cnt++; /* Increment number of counter interrupts */
   if ( pwmdt.tim_cnt == 125 ) { /* If 1 second is reached */
      /* Creation of motor rotations per minute */
      /* (*60): per minute, (/12): 12 pulses per rotation */
      pwmdt.pwm_cnt = pwmdt.int_cnt * 60 / 12;
                           /* Disk and motor gear ratio 1:18 */
                           /* pwmdt.pwm_cnt /= 18; (to find disk rotations
*/
      /* Clear data */
      pwmdt.tim_cnt = 0; /* Clear number of interval timer interrupts */
      pwmdt.int_cnt = 0; /* Clear number of PWM interrupts */
      led_disp = 1; /* 7-segment display */
   }
   /* A/D conversion processing */
   if ( (ADM0_bit.no6) == 0 ) { /* Start conversion when converter is stopped
*/
      ADM0_bit.no7= 1;          /* Start conversion */
   }
}

/* ————————————————————
 * A/D conversion completion interrupt
 * Interrupt source: INTAD
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   PWM0 = ADCR1; /* Reflect data in PWM */
}
```

```
/* ————————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PortInit(void)
{
   /* Initialize port 0 (P0) */
   PM0 = 0xFF; /* All ports in input mode */
   PMC0 = 0x00; /* P00 to P07 in port mode */

   /* Initialize port 1 (P1) */
   PM1 = 0xFF; /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize port 2 (P2) */
   PM2 = 0xFF; /* All ports in input mode */
   PMC2 = 0x61; /* Use RXD1, TXD1, PWM0 P20, P24, P25 in control mode */
               /* 1P21 to P23, P26, P27 in port mode */

   /* Initialize port 3 (P3) */
   PM3 = 0x00; /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00; /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4) */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5) */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6) */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9) */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                (reference MM register) */

   /* Initialize port 11 (P11) */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x10; /* Use INTP140 P114 in control mode */
                /* P110 to P113, P115 to P117 in port mode */
}
```

```
/* ————————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   P14IC0 = 0x07; /* INTP140 : DC motor pulse counter : Level 4 */
   SRIC1 = 0x07;   /* INTSR1  : Serial interrupt       : Level 1 */
   STIC1 = 0x07;   /* INTST1  : Serial interrupt       : Level 2 */
   SEIC1 = 0x07;   /* INTSER1 : Serial interrupt       : Level 3 */
   CMIC4 = 0x07;   /* INTCM4  : Set interval timer     : Level 6 */
   ADIC = 0x07;    /* INTAD   : A/D conversion end     : Level 5 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x00; /* Not used */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x01; /* INTP140: Rising edge */
}

/* ————————————————————
 * PWM initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PWMInit(void)
{
   /* Initialize internal data */
   pwmdt.tim_cnt = 0; /* Clear number of interval timer interrupts */
   pwmdt.pwm_cnt = 0; /* Clear number of motor rotations per minute */
   pwmdt.int_cnt = 0; /* Clear number of PWM interrupts */

   /* PWM settings (PWMC): Only set PWM0 */
   PWMC = 0x02; /* PWME0 = 0: PWM in operation stopped state */
                /* ALV0 = 0: Active level is low */
                /* PRM01 = 1: Compare register is 10 bits */
                /* PRM00 = 0; */
   PWPR = 0x04; /* Prescaler f/16 */
   PWM0 = 0;    /* Clear buffer register */
}

/* ————————————————————
 * A/D initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
```

```
void ADInit(void)
{
   ADM0 = 0x11; /* Operation not enabled: 1 buffer mode: Select mode: ANI1 */
   ADM1 = 0x03; /* A/D trigger mode */
}

/* ————————————————————————-
 * Asynchronous serial interface initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/

void UartInit(void)
{
   /* Clear data */
   uartdt.err_flg = 0; /* Error flag */
   uartdt.rd_cnt = 0;  /* Received data processed location */
   uartdt.rd_pos = 0;  /* Received data write location */
   uartdt.rd_flg = 1;  /* Receive enabled flag */
   uartdt.wt_cnt = 0;  /* Transmit data processed location */
   uartdt.wt_pos = 0;  /* Transmit data write location */
   uartdt.wt_flg = 1;  /* Transmission end flag */

   ASIM00 = 0;    /* UART0: Not used */
   ASIM10 = 0xC8; /* UART1: Transmit/receive enabled, no parity, 8 bits, 1
stop bit */
   ASIM01 = 0;    /* UART0: Disable adding extended bits */
   ASIM11 = 0;    /* UART1: Disable adding extended bits */
   BPRM1 = 0x80;  /* UART1: Set 9600 bps at 33-MHz operation */
   BRGC1 = 41;    /* UART1: Set 9600 bps at 33-MHz operation */
}

/* ————————————————————
 * Interval timer initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void TimerInit(void)
{
   /* Set interrupt every 8 ms (setting at 33 MHz) */
   TMC4 = 0x03; /* Put counter in stopped state */
                /* Count clock setting f/32 */
   CM4 = 8250 - 1; /* Comparison counter */
}
```

```
/* ————————————————————
 * Motor rotations count write, UART write processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void sendpwmcnt(void)
{
   unsigned short tmp1; /* Work variable */
   unsigned short tmp2; /* Work variable */
   char tmp3; /* Work variable */
   short i; /* Work variable */

   RXE1 = 0; /* Receive disabled */
   TXE1 = 1; /* Transmit enabled */
   uartdt.wt_flg = 0; /* Transmit completion OFF */
   uartdt.wt_cnt = 0; /* Clear buffer pointer */
   uartdt.wt_pos = 0; /* Clear buffer pointer */

   /* Assign transmit data */
   uartdt.wt_buf[uartdt.wt_pos++] = '=';
   for ( i = 10000 , tmp1 = pwmdt.pwm_cnt ; i >= 1 ; i /= 10 ) {
      tmp2 = tmp1 / i;
      tmp1 -= tmp2 * 1;
      tmp3 = '0' + tmp2;
      uartdt.wt_buf[uartdt.wt_pos++] = tmp3;
   }
   uartdt.wt_buf[uartdt.wt_pos++] = 0x0d;
   uartdt.wt_buf[uartdt.wt_pos++] = 0x0a;
   /* Start transmit */
   TXS1L = uartdt.wt_buf[uartdt.wt_cnt++];
}

/* ————————————————————
  7-segment LED display processing
  Arguments: unsigned long disp_data display data
  Return value: None
  ————————————————————
*/
```

```
void disptimer(
   unsigned short disp_data /* Display data */
)
{
   unsigned char *segaddr = SEG7ADDR; /* 7-segment LED set address */
   unsigned char cnt; /* Work counter */

   /* Set all columns of 7-segment LEDs */
   for ( cnt = 1 ; cnt <= LEDMAX ; cnt ++ ) {
      /* Display 7-segment LEDs */
      switch ( disp_data % 10 ) {
         case 0: *segaddr = SEGLED_0; break;
         case 1: *segaddr = SEGLED_1; break;
         case 2: *segaddr = SEGLED_2; break;
         case 3: *segaddr = SEGLED_3; break;
         case 4: *segaddr = SEGLED_4; break;
         case 5: *segaddr = SEGLED_5; break;
         case 6: *segaddr = SEGLED_6; break;
         case 7: *segaddr = SEGLED_7; break;
         case 8: *segaddr = SEGLED_8; break;
         case 9: *segaddr = SEGLED_9; break;
      }
      /* Generate data of next column */
      disp_data /= 10;
      /* Generate set address of next column */
      segaddr += 2;
   }
}

/* ————————————————
 * Operation start processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
void StartProc(void)
{
   /* Control flag initialization */
   led_disp = 1; /* 7-segment LED display flag */

   /* Start timer */
   TMC4_bit.no7 = 1;

   /* Start A/D conversion */
   ADM0_bit.no7 = 1;    /* Set bit CE */

   /* Start PWM output */

   PWMC_bit.no3= 1;

   /* Enable interrupt */
   __EI();
}
```

```
/* ————————————————
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
void main(void)
{
   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* Asynchronous serial interface initialization processing */
   UartInit();

   /* PWM initialization processing */
   PWMInit();

   /* A/D initialization processing */
   ADInit();

   /* Interval timer initialization processing */
   TimerInit();

   /* Start operation */
   StartProc();

   /* Monitor time data */
   while ( 1 ) {
      /* 7-segment LED display */
      if ( led_disp == 1 ) {
         disptimer(pwmdt.pwm_cnt);
         led_disp = 0;
      }
      /* Received data and transmit completion flag is ON */
      if ( ( uartdt.rd_cnt != uartdt.rd_pos ) && ( uartdt.wt_flg == 1 ) ) {
         if ( uartdt.rd_buf[uartdt.rd_cnt] == 'R' ) {
            /* Transmit data after confirming command */
            sendpwmcnt();
         }
         uartdt.rd_cnt ++;
         if ( uartdt.rd_cnt >= RDBUFSIZE ) {
            uartdt.rd_cnt = 0;
         }
      }
   }
}
```

```
/* ————————————————
* DCMOTOR.C header file
* ————————————————
* History:
* 1997/3/17 Ver 0.00.00
* File Name: dcmotor.h
* ————————————————
*/
/* Time measurement structure */
struct pwmdata
{
unsigned short tim_cnt; /* Timer counter */
unsigned short int_cnt; /* PWM interrupt counter */
unsigned short pwm_cnt; /* Number of motor rotations per minute */
};
struct pwmdata pwmdt;
/* UART data structure */
struct uartdata
{
#define RDBUFSIZE 64 /* Receive buffer size */
#define WTBUFSIZE 16 /* Transmit buffer size */

unsigned char err_flg; /* Error flag */
unsigned char rd_cnt; /* Receive data processed location */
unsigned char rd_pos; /* Receive data write location */
unsigned char rd_buf[RDBUFSIZE]; /* Receive data buffer */
unsigned char rd_flg; /* Receive enabled flag */
unsigned char wt_cnt; /* Transmit data processed location */
unsigned char wt_pos; /* Transmit data write location */
unsigned char wt_buf[WTBUFSIZE]; /* Transmit data buffer */
unsigned char wt_flg; /* Transmit completion flag */
};
struct uartdata uartdt;
/* 7-segment LED data */
#define LEDMAX 5 /* Number of columns used */
/* Address of first column */
#define SEG7ADDR (unsigned char *)0x00460000L
/* Segment data */
#define SEGLED_DOT 0x80 /* Point */
#define SEGLED_0 0x3f /* 0 */
#define SEGLED_1 0x06 /* 1 */
#define SEGLED_2 0x5b /* 2 */
#define SEGLED_3 0x4f /* 3 */
#define SEGLED_4 0x66 /* 4 */
#define SEGLED_5 0x6d /* 5 */
#define SEGLED_6 0x7d /* 6 */
#define SEGLED_7 0x07 /* 7 */
#define SEGLED_8 0x7f /* 8 */
#define SEGLED_9 0x6f /* 9 */
```

3.4.3 Clock synchronous serial interface (CSI0 to CSI3)

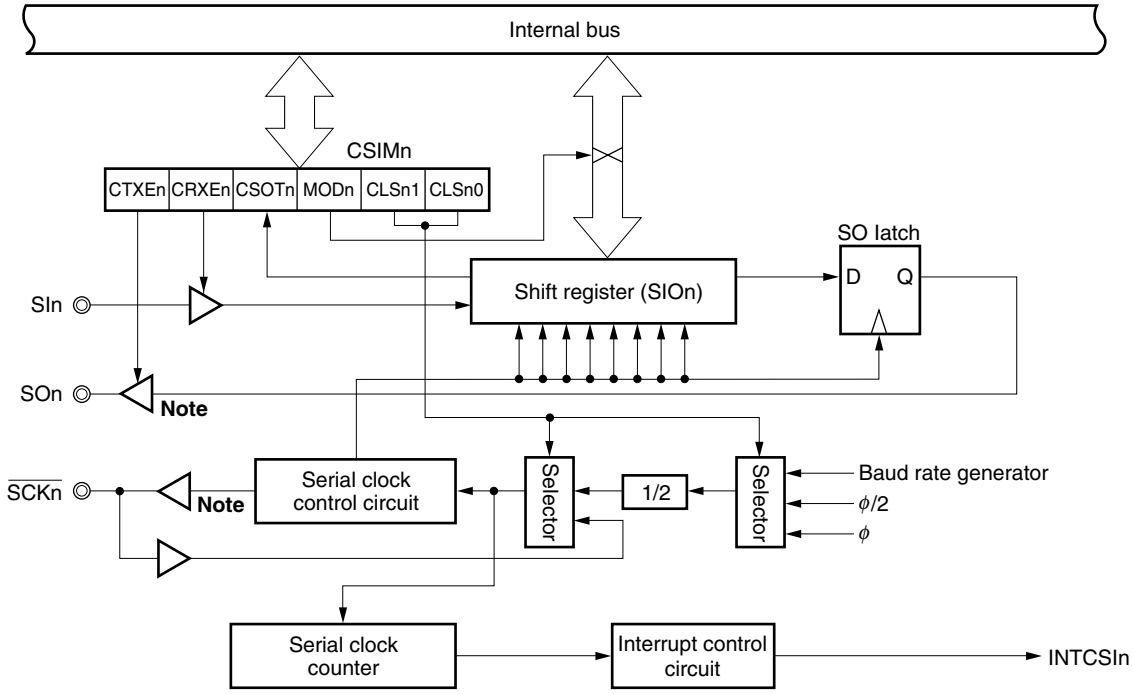
The clock synchronous serial interface performs data transmission/reception by using three lines: one clock line and two data lines. Features of the clock synchronous serial interface are shown below.

- Transfer speed: 8.25 Mbps (at ϕ = 33-MHz operation)
- Character length: 8 bits
- Half-duplex communication
- MSB/LSB can be switched as first bit of data transmission
- External serial clock input or internal serial clock output optional
- Interrupt source: Transmit/receive completion interrupt (INTCSIn)

Remark: n = 0 to 3
ϕ: Internal system clock

Figure 3-46 shows a block diagram of the clock synchronous serial interface.

Figure 3-46: Block Diagram of Clock Synchronous Serial Interface



Note: SO0 to SO2, $\overline{SCK0}$ to $\overline{SCK2}$: CMOS output
SO3, $\overline{SCK3}$: N-ch open-drain output

Remark: n = 0 to 3

Figure 3-47 shows the clock synchronous serial mode registers (CSIM0 to CSIM3) that make each clock synchronous serial interface setting.

Figure 3-47: Clock Synchronous Serial Interface Mode Registers 0 to 3 (CSIM0 to CSIM3)

	7	6	5	4	3	2	1	0	Address	After reset
CSIMn	CTXEn	CRXEn	CSOTn	0	0	MODn	CLSn1	CLSn0	FFFFF088H to FFFFF0B8H	00H

Bit position	Bit name	Description
7	CTXEn	CSI Transmit Enable Specifies transmit enabled/disabled status. 0: Transmit disabled status 1: Transmit enabled status When CTXEn = 0, SOn and SIn pin output buffers are high impedance.
6	CRXEn	CSI Receive Enable Specifies receive enabled/disabled status. 0: Receive disabled status 1: Receive enabled status If transmit enabled (CTXEn = 1) and receive disabled, "0" is input to shift register after serial clock is input. If made receive disabled (CRXEn = 0) while receiving, contents of SIOn register are undefined.
5	CSOTn	CSI Status of Transmission Indicates transmission operation in progress. Set (1): Transmit start timing (write to SIOn register) Clear (0): Transmit end timing (INTCSIn generation) Used as means of distinguishing whether or not writing to serial I/O shift register n (SIOn) is possible when starting serial data transmission in transmit enabled status (CTXEn = 1).
2	MODn	Mode Specifies operating mode. 0: MSB first 1: LSB first
1, 0	CLSn1, CLSn0	Clock Source Specifies serial clock.

CLSn1	CLSn0	Serial clock specification		$\overline{\text{SCK}}$n pin
0	0	External clock		Input
0	1	Internal clock	Specify in BPRMn register Note 1	Output
1	0		ϕ/4 Note 2	Output
1	1		ϕ/2 Note 2	Output

Notes:1. Refer to **V853 User's Manual Hardware** regarding BPRMn register settings.

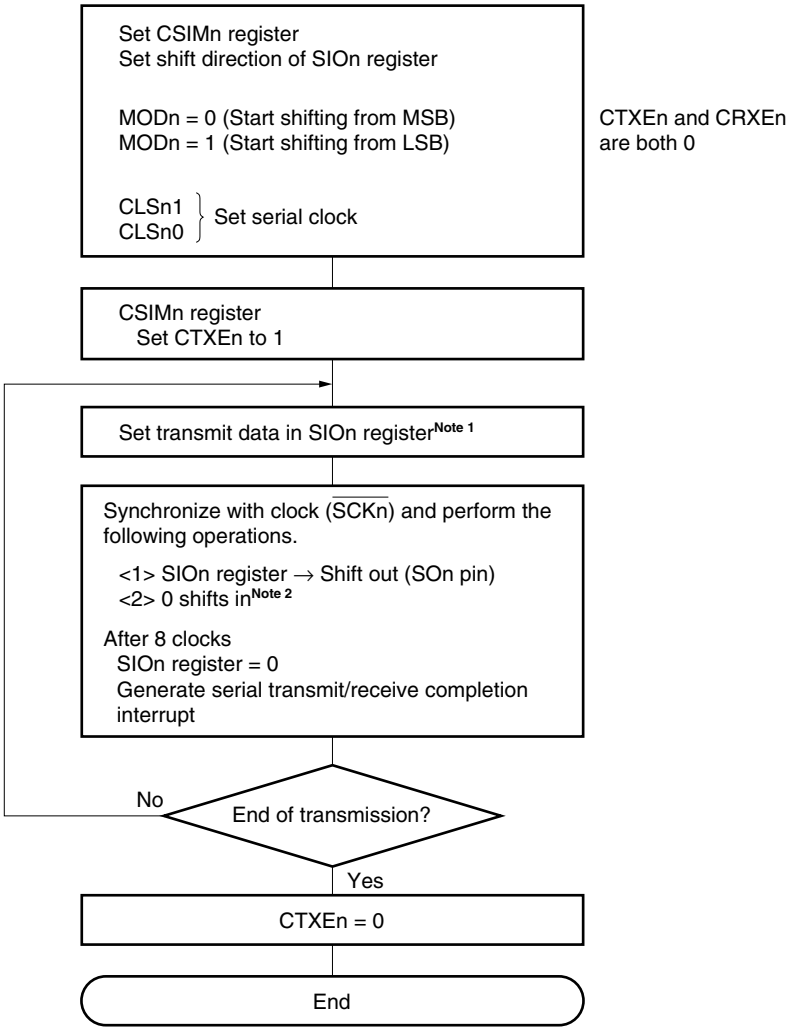
2. ϕ/4 and ϕ/2 are division signals (ϕ : Internal system clock).

Remark: n = 0 to 3

CSI2 and CSI3 share BRG2 of the three baud rate generators. Therefore, if the clock source in CSI2 and CSI3 is made the baud rate generator, communication cannot be performed using different transmission speeds.

Figure 3-48 shows clock synchronous serial interface transmit operation.

Figure 3-48: Clock Synchronous Serial Interface Transmit Operation

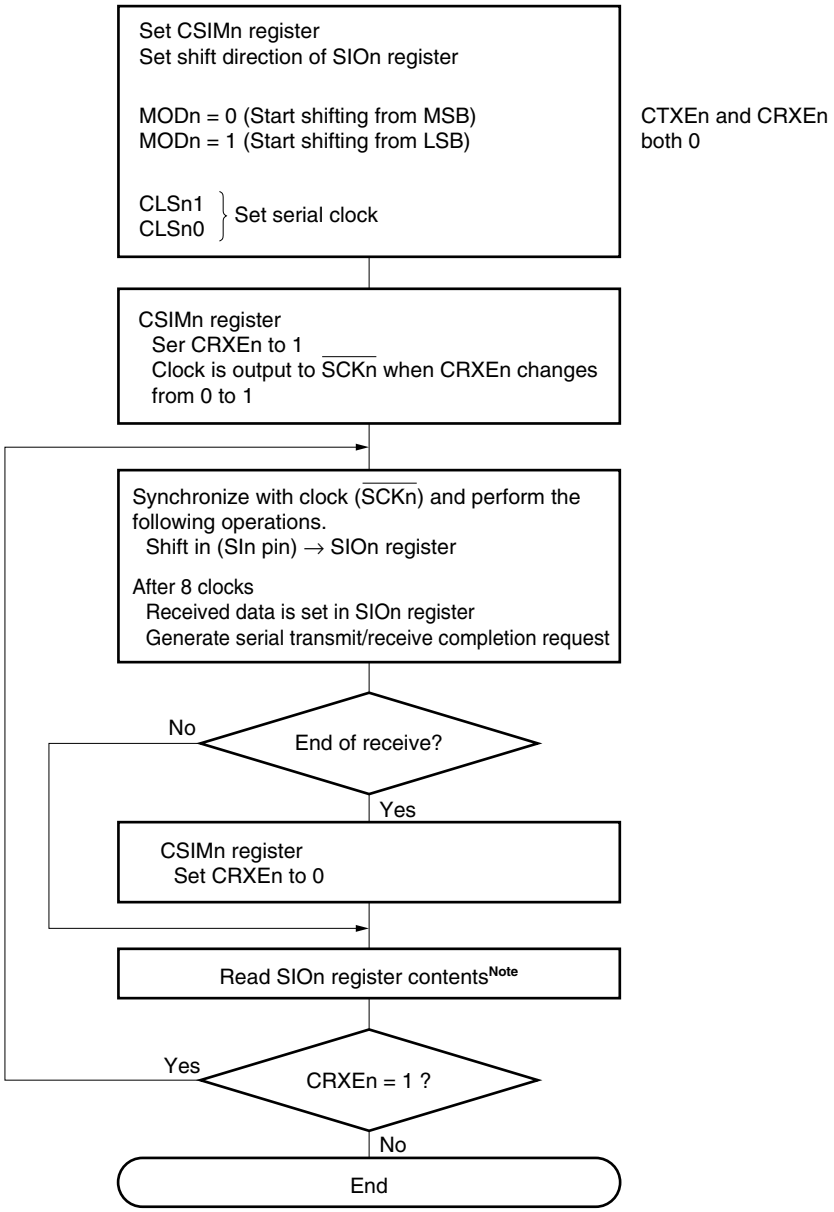


Notes: **1.** If CTXEn = 1, the clock is output to $\overline{\text{SCKn}}$ when data is written to the SIOn register.
2. If CRXEn = 0, input to the shift register is 0.

Remark: n = 0 to 3

Figure 3-49 shows clock synchronous serial interface receive operation.

Figure 3-49: Clock Synchronous Serial Interface Receive Operation

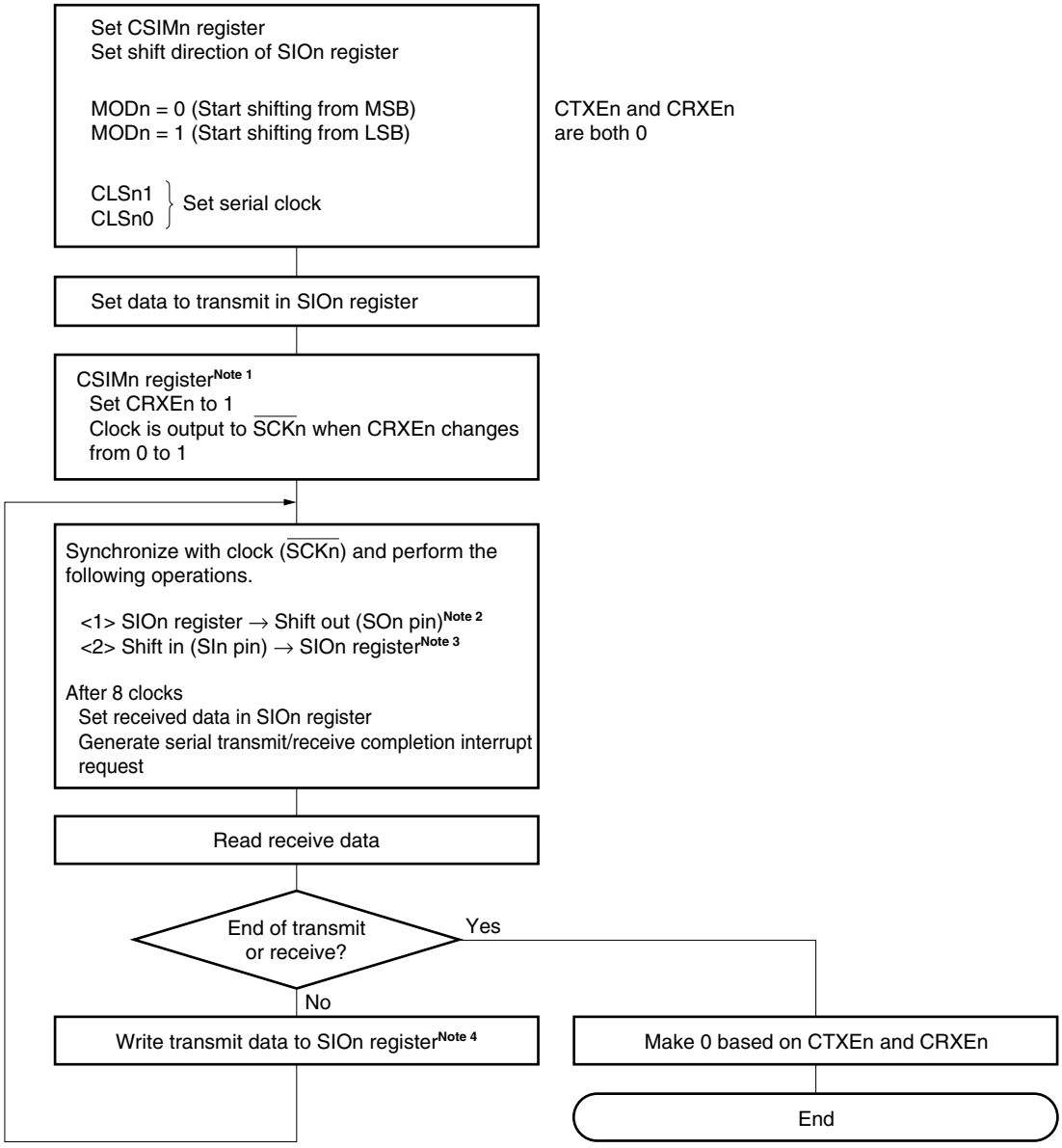


Note: If CRXEn = 1, the clock is output to $\overline{\text{SCKn}}$ when data in the SIOn register is read.

Remark: n = 0 to 3

Figure 3-50 shows clock synchronous serial interface transmit and receive operation.

Figure 3-50: Clock Synchronous Serial Interface Transmit and Receive Operation



Notes: **1.** Set enable flags in the order CTXEn, CRXEn.
If CTXEn is 0 when CRXEn changes from 0 to 1, transmission is not performed.

2. The operation <1> synchronizes with falling clock

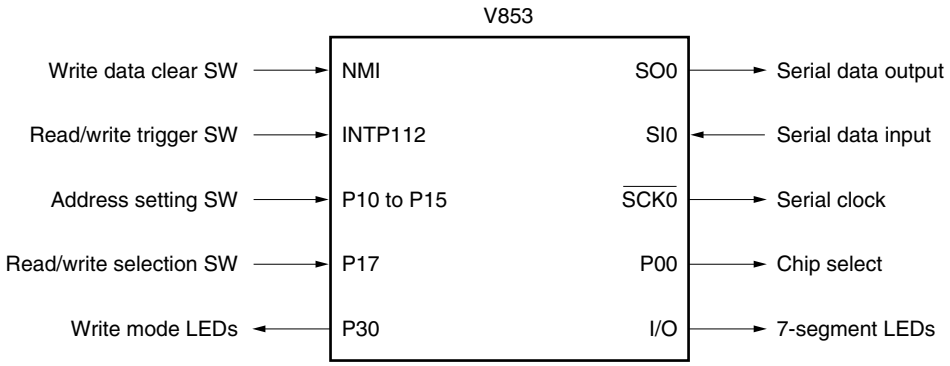
3. The operation <2> synchronizes with rising clock

4. If CRXEn = 1 and CTXEn = 1, the clock is output to $\overline{SCKn}$ when data is written to the SIOn register.

Remark: n = 0 to 3

3.4.4 Clock synchronous serial interface sample program

Figure 3-51: Sample Hardware Configuration



(1) Function overview

- Read/write data for EEPROM™ connected to clock synchronous serial interface.
- Obtain address of EEPROM for which performing read/write from port. The addresses that can be set are 00H to 3FH.
- Data communication with EEPROM is performed by using read/write trigger SW input.
- The initial value of write data is set to 1 and incremented by 1 each time it is written.
- Display data read/written in 7-segment LEDs.

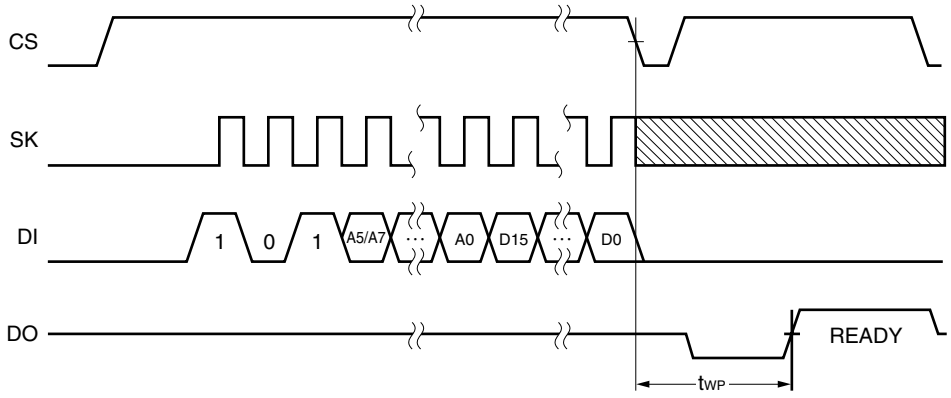
Next, a program is created to read/write data to a serial type EEPROM connected to the clock synchronous serial interface using the sample hardware.
The functions of the EEPROM connected to the sample hardware are shown below.

- Read: Read data
- Write: Write data
- Delete: Delete data
- Delete/write enable: Enable delete/write operation
- Delete/write disable: Disable delete/write operation

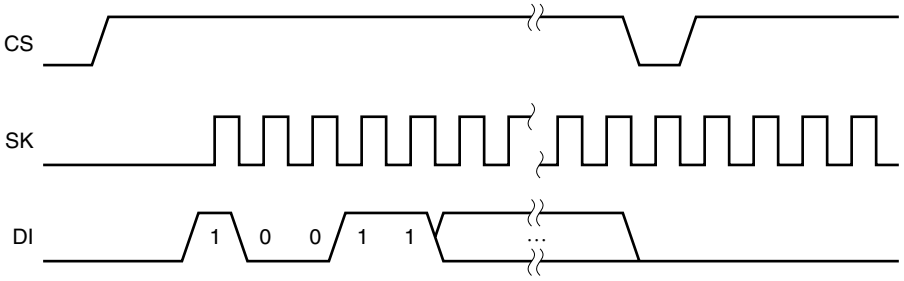
Figure 3-52 shows the timing of EEPROM operations.

Figure 3-52: EEPROM Operation Timing (1/2)

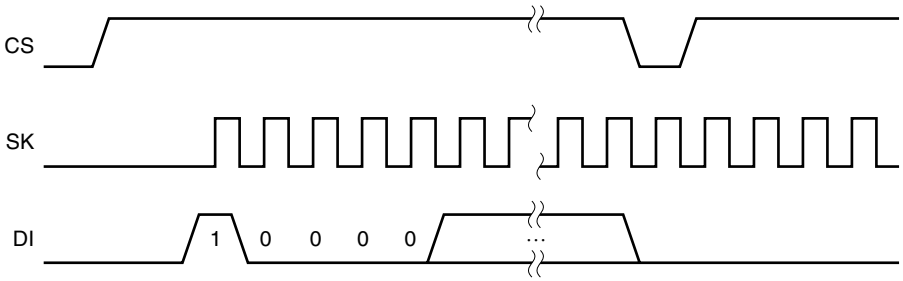
(a) Write



(b) Delete/write enable



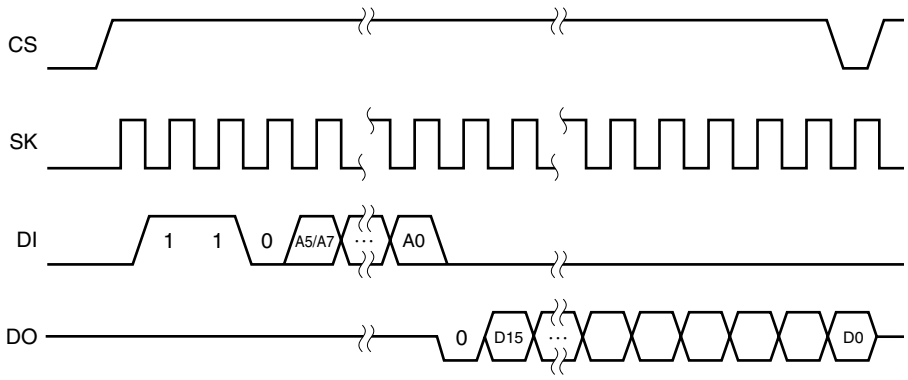
(c) Delete/write disable



Remarks: CS: Chip select signal
SK: Serial clock
DI: Serial data input
DO: Serial data output

Figure 3-52: EEPROM Operation Timing (2/2)

(d) Data read



Remarks: CS: Chip select signal
SK: Serial clock
DI: Serial data input
DO: Serial data output

Data transmission is performed starting from the MSB in operations of the connected EEPROM. In addition, the serial clock must have a period of at least 1 µs. Therefore, the transfer speed is made 76800 bps for good measure. The serial clock uses an internal clock generated by using the baud rate generator.
Set clock synchronous serial interface mode register 0 (CSIM0) according to these conditions.

Figure 3-53: Clock Synchronous Serial Interface Mode Register 0 (CSIM0) Setting

	7	6	5	4	3	2	1	0	Address	Value
CSIM0	0	0	0	0	0	0	0	1	FFFFF088H	01H

For the serial clock, use one generated by the baud rate generator. See **3.4.5 Baud rate generator (BRG0 to BRG2)** regarding the baud rate generator.
In order to generate 76800 bps, set the following values in baud rate generator compare register 1 (BRGC1) and baud rate generator prescaler mode register 1 (BPRM1).

BPRM1 = 80H (count operation enabled, count clock = ϕ/2)
BRGC1 = 107 (operation clock = 33 MHz)
= 81 (operation clock = 25 MHz)

As for enabling receive and transmit in the clock synchronous serial interface mode register, enable transmit and receive to suit the EEPROM operation when a mode setting or data is read/ written to the EEPROM.

- Data read → Enable both transmit and receive
- Data write → Enable transmit
- Data delete → Enable transmit
- Delete/write enable → Enable transmit
- Delete/write disable → Enable transmit

The CS (chip select) signal of the EEPROM is connected to bit 0 of port 0. Control the program using the operation timing shown in Figure 3-52.

Figure 3-54: Flow of Sample Program to Read/Write to EEPROM (1/2)

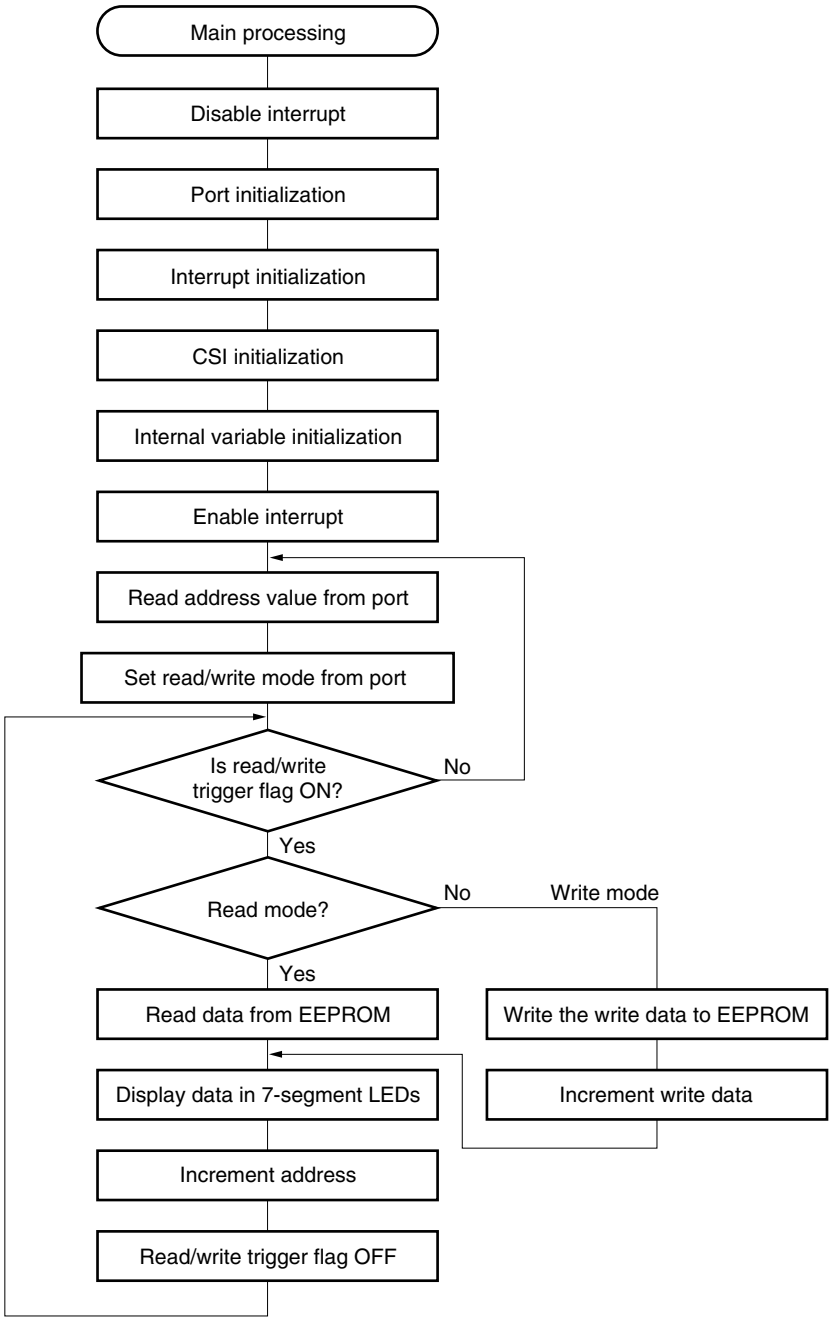
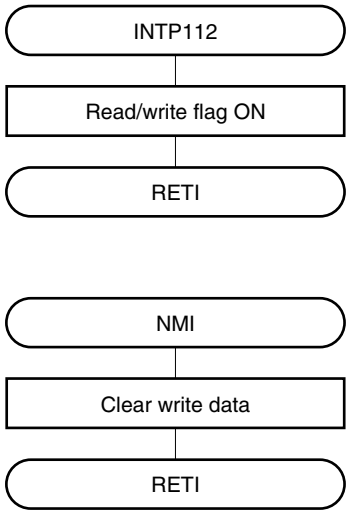


Figure 3-54: Flow of Sample Program to Read/Write to EEPROM (2/2)



(2) Sample program

```
/* ————————————————————————————
 * V853 clock synchronous serial interface sample program
 *
 * Initialization program for the following hardware configuration
 * SO0       : Serial data output
 * SI0       : Serial data input
 * SCK0      : Serial clock
 * PO0       : Chip select
 * NMI       : Write data clear SW
 * INTP112   : Read/write trigger SW
 * P10 to P15: Read/write address setting SW
 * P17       : Read/write selection SW
 * P30       : Write mode LED
 *
 * ————————————————————————————
 * History:
 *    1997/3/17  Ver 0.00.00
 *    1999/7/8   adapted to IAR compiler by NEC Duesseldorf AH
 * File Name:    eeprom.c
 * ————————————————————————————-
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

#include "7segLed.h"

/* ————————————————————————————
 * Internal variable definition
 * ————————————————————————————
*/

/* Variable to store value of received data */
static unsigned short readData = 0;

/* Variable for saving shift register data */
static short tempData = 0;

/* Data for writing */
static short writeData = 1;

/* Read/write address */
static unsigned char addrData;

/* Operation flag */
static int StartFlag = 0; /* CSI transmit/receive start flag
                             (!0: Transmitting or receiving) */
static int TrigFlag = 0; /* Trigger flag */

/* ————————————————————
 * NMI processing (write data clear)
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
```

```
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   writeData = 1;
}

/* ———————————————————
 * Clock synchronous serial communication end
 * Interrupt source: INTCSI
 * Arguments: None
 * Return value: None
 * ———————————————————
*/
#pragma vector=INTCSI0_vector
__flat __interrupt void int_csi0(void)
{
   /* Set received data to save */
   tempData = SIO0;

   /* CSI communication end flag OFF */
   StartFlag = 0;
}

/* ———————————————————
 * CSI receive start interrupt servicing
 * Interrupt source: INTP112
 * Arguments: None
 * Return value: None
 * ———————————————————
*/
#pragma vector=INTP112_vector
__flat __interrupt void int_p112(void)
{
   if (TrigFlag == 0) {
     /* Set trigger flag */
     TrigFlag = 1;
   }
}

/* ———————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ———————————————————
*/
void PortInit(void)
{
   /* Initialize port 0 (P0)
   */
   PM0 = 0xFE; /* P00 in output mode (for chip select) */

   /* P01 to P07 in input mode */
   PMC0 = 0x40; /* Use INTP112 P06 in control mode */
                /* P00 to P05, P07 in port mode */
```

```
   /* Initialize Port 1 (P1)
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize port 2 (P2)
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x1C; /* Use SO0, SI0, SCK0 P22, P23, P24 in control mode */
                /* P20, P21, P25 to P27 in port mode */

   /* Initialize port 3 (P3)
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}

/* ————————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
```

```
void IntInit(void)
{
   /* Interrupt register settings (set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: CSI transmit/receive start: Level 7 */
   CSIC0 = 0x07; /* INTCSI0: CSI transmit/receive end : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x10; /* INTP112: Rising edge */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}

/* ————————————————————
 * CSI initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void CSIInit(void)
{
   /* Asynchronous serial mode settings */
   ASIM00 = 0x00; /* Transmit/receive disabled CSI mode */
   ASIM10 = 0x00; /* Transmit/receive disabled CSI mode */

   /* Clock synchronous serial mode settings */
   CSIM0 = 0x01; /* Receive/transmit disabled, MSB first */
                 /* Internal clock (using baud rate generator) */
   /* Baud rate setting (76.8 kbps) */
   BPRM0 = 0x80;
   BRGC0 = 81; /* 25MHz */
}

/* ————————————————————
 * 7-segment LED display processing
 * Arguments: disp_data  Display data
              dot_pos  Decimal point display position (1-LEDMAX)
              mode  Decimal/hexadecimal specification (0: Decimal, 1: Hexa-
decimal)
 * Return value: None
 * ————————————————————
*/
void Disp7SegLED(unsigned long disp_data, int dot_pos, int mode)
{
   register unsigned char *segaddr = SEG7ADDR; /* 7-segment LED set address
*/
   register unsigned char cnt; /* Work counter */
   register int base; /* Decimal/hexadecimal radix */
```

```
   /* Set radix */
   if (mode == 0) {
      base = 10; /* Decimal display */
   } else {
      base = 16; /* Hexadecimal display */
   }

   /* Set all columns of 7-segment LEDs */
   for ( cnt = 1 ; cnt <= LEDMAX ; cnt ++ ) {
      /* 7-segment LED display */
      switch ( disp_data % base ) {
         case  0: *segaddr = SEGLED_0; break;
         case  1: *segaddr = SEGLED_1; break;
         case  2: *segaddr = SEGLED_2; break;
         case  3: *segaddr = SEGLED_3; break;
         case  4: *segaddr = SEGLED_4; break;
         case  5: *segaddr = SEGLED_5; break;
         case  6: *segaddr = SEGLED_6; break;
         case  7: *segaddr = SEGLED_7; break;
         case  8: *segaddr = SEGLED_8; break;
         case  9: *segaddr = SEGLED_9; break;
         case 10: *segaddr = SEGLED_A; break;
         case 11: *segaddr = SEGLED_B; break;
         case 12: *segaddr = SEGLED_C; break;
         case 13: *segaddr = SEGLED_D; break;
         case 14: *segaddr = SEGLED_E; break;
         case 15: *segaddr = SEGLED_F; break;
      }
      /* Generates data of next column */
      disp_data /= base;

      /* If displaying decimal point */
      if ( cnt == dot_pos ) {
         /* 7-segment LED display */
         *segaddr += SEGLED_DOT;
      }
      /* Create set address of next column */
      segaddr += 2;
   }
}

/* ————————————————————
 * Write enable processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
```

```
void WriteEnable(void)
{
   /* Chip select ON */
   P0_bit.no0 = 1;
   /* Transmit write enable data */
   StartFlag = 1;
   SIO0 = 0x98; /* (10011000b) */
   /* Monitor transmit end */
   while (1) {
      if (StartFlag == 0) {
        break;
      }
   }
   /* Since at least 11 clocks are needed, write data 0 */
   StartFlag = 1;
   SIO0 = 0x00;
   while (1) { /* Monitor transmit end */
      if (StartFlag == 0) {
         break;
      }
   }
   /* Chip select OFF */
   P0_bit.no0 = 0;
}

/* ————————————————————
 * Write disable processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void WriteDisable(void)
{
   /* Chip select ON */
   P0_bit.no0 = 1;
   /* Transmit write disable data */
   StartFlag = 1;
   SIO0 = 0x80; /* 10000000b */
   while (1) { /* Monitor transmit end */
      if (StartFlag == 0) {
         break;
      }
   }
   /* Since at least 11 clocks are needed, write data 0 */
   StartFlag = 1;
   SIO0 = 0x00;
   while (1) { /* Monitor transmit end */
      if (StartFlag == 0) {
         break;
      }
   }
   /* Chip select OFF */
   P0_bit.no0 = 0;
}
```

```
/* ————————————————————
 * Write processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void WriteProc(void)
{
   unsigned char sendChar[4];
   unsigned char wk;
   unsigned int i;

   /* CSI transmit flag ON */
   CTXE0 = 1;
   CRXE0 = 0;

   /* Write enable */
   WriteEnable();

   /* Transmit data setup */
      /* 101b + address 6 bits */
   wk = (addrData >> 1) & 0x1f;
   sendChar[0] = 0xA0 | wk;
   sendChar[1] = (addrData << 7) & 0x80; /* Remaining 7 clocks are discarded
*/
     /* Data 16 bits */
   sendChar[2] = (unsigned char)((writeData >> 8) & 0x00ff);
   sendChar[3] = (unsigned char)(writeData & 0x00ff);

   /* Chip select ON */
   P0_bit.no0 = 1;
   /* Data write */
   for (i=0; i<4; i++) {
      StartFlag = 1;
      SIO0 = sendChar[i];
      /* Monitor transmit end */
      while (1) {
         if (StartFlag == 0) {
            break;
         }
      }
   }
   /* Chip select OFF */
   P0_bit.no0 = 0;
   /* Write disable */
   WriteDisable();
}

/* ————————————————————
 * Read processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
```

```
void ReadProc(void)
{
   unsigned char sendChar[4];
   unsigned char recvChar[4];
   unsigned char wk;
   unsigned short wkShort;
   unsigned int i;

   /* CSI transmit flag ON */
   CTXE0 = 1;
   CRXE0 = 1;

   /* Transmit data setup */
   /* 110b + address 6 bits */
   /* Set other data to 0 */
   wk = (addrData >> 1) & 0x1f;
   sendChar[0] = 0xC0 | wk;
   sendChar[1] = (addrData << 7) & 0x80;
   sendChar[2] = 0x00;
   sendChar[3] = 0x00;

   /* Chip select ON */
   P0_bit.no0 = 1;

   /* Data write & read */
   for (i=0; i<4; i++) {
      StartFlag = 1;
      SIO0 = sendChar[i];
      /* Monitor transmit end */
      while (1) {
         if (StartFlag == 0) {
            break;
         }
      }
      recvChar[i] = SIO0;
   }
   /* Chip select OFF */
   P0_bit.no0 = 0;
   /* Get received data */
   /* Data begins from command 9 clocks + 1 clock */
   wkShort = (unsigned short)recvChar[1];
   readData = (wkShort << 10) & 0xfc00;
   wkShort = (unsigned short)recvChar[2];
   readData |= (wkShort << 2) & 0x03fc;
   wkShort = (unsigned short)recvChar[3];
   readData |= (wkShort << 6) & 0x0003;
}

/* ————————————————
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
```

```
void main(void)
{
   unsigned char WriteMode; /* Work variable */

   /* Interrupt disable */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* CSI initialization processing */
   CSIInit();

   /* Clear LEDs */
   P3 = 0;

   /* Internal data initialization */
   writeData = 1; /* Initialize write data */
   TrigFlag = 0; /* Clear interrupt flag */
   StartFlag = 0; /* Clear start flag */

   /* Interrupt enable */
   __EI();

   while ( 1 ) {
      /* Get address information */
      addrData = P1;
      addrData &= 0x03f;
      /* Get read/write mode */
      WriteMode = P1_bit.no7;
      if (WriteMode) {
         /* Write LED lit Write mode */
         P3_bit.no0 = 1;
      } else {
      /* Write LED out Read mode */
      P3_bit.no0 = 0;
      }
      /* Trigger SW ON? */
      if (TrigFlag==1) {
         if (WriteMode) {
            WriteProc();
            Disp7SegLED(writeData, 0, 1);
            ++writeData;
         } else {
         ReadProc();
         Disp7SegLED(readData, 0, 1);
         }
         TrigFlag = 0;
      }
   }
}
```

```
/* ————————————————————————-
* TU-V853 7-segment LED register definition
* ————————————————————————-
* History:
* 1997/3/17 Ver 0.00.00
* File Name: 7segled.h
* ————————————————————————-
*/

/* 7-segment LED data */
#define LEDMAX 5 /* Number of columns used */

/* Address of first column */
#define SEG7ADDR (unsigned char *) 0x00460000L

/* Segment data */
#define SEGLED_DOT 0x80 /* Point */
#define SEGLED_0 0x3f /* 0 */
#define SEGLED_1 0x06 /* 1 */
#define SEGLED_2 0x5b /* 2 */
#define SEGLED_3 0x4f /* 3 */
#define SEGLED_4 0x66 /* 4 */
#define SEGLED_5 0x6d /* 5 */
#define SEGLED_6 0x7d /* 6 */
#define SEGLED_7 0x07 /* 7 */
#define SEGLED_8 0x7f /* 8 */
#define SEGLED_9 0x6f /* 9 */
#define SEGLED_A 0x77 /* A */
#define SEGLED_B 0x7c /* b */
#define SEGLED_C 0x39 /* C */
#define SEGLED_D 0x5e /* d */
#define SEGLED_E 0x79 /* E */
#define SEGLED_F 0x71 /* F */
```

3.4.5 Baud rate generator (BRG0 to BRG2)

The serial clock of the serial interface can be selected from the following two for each channel.

<1> Baud rate generator output
<2> Internal system clock (ϕ)

Since the serial clock of a channel is common to transmitting and receiving, the transmit and receive baud rates are the same.

The baud rate generated using the baud rate generator is determined by the value of the baud rate generator prescaler mode register (BPRM0 to BPRM2) and the value of the baud rate generator compare register (BRGC0 to BRGC2).

Formulas for calculating baud rate are shown below.

(1) Formula to calculate baud rate in UART0 and UART1

$$\text{Baud rate} = \frac{\phi}{2 \times j \times 2^k \times 16 \times 2} \text{ [bps]}$$

ϕ = Internal system clock frequency (Hz)
j = Timer count value (1 - j - 256[Note]): Set using BRGCn
k = Prescaler setting (k = 0, 1, 2, 3, 4): Set using BPRMn

Note: Make the setting j = 256 by writing 0 to the BRGCn register.

(2) Formula to calculate baud rate in CSI0 to CSI3

$$\text{Baud rate} = \frac{\phi}{2 \times j \times 2^k \times 2} \text{ [bps]}$$

ϕ = Internal system clock frequency (Hz)
j = Timer count value (1 - j - 256[Note]): Set using BRGCn
k = Prescaler setting (k = 0, 1, 2, 3, 4): Set using BPRMn

Note: Make the setting j = 256 by writing 0 to the BRGCn register.

Table 3-8 shows the settings of BPRMn and BRGCn when typical clocks are used.

Table 3-8: Baud Rate Generators 0 to 2 Setting Data

Baud rate [bps]		ϕ = 33 MHz			ϕ = 25 MHz			ϕ = 16 MHz			ϕ = 12.5 MHz		
UART0,UART1	CSI0 to CSI3	BPR	BRG	Error	BPR	BRG	Error	BPR	BRG	Error	BPR	BRG	Error
110	1760	–	–	–	4	222	0.02 %	4	142	0.03 %	3	222	0.02 %
150	2400	4	215	0.07 %	4	163	0.15 %	3	208	0.16 %	3	163	0.15 %
300	4800	3	215	0.07 %	3	163	0.15 %	2	208	0.16 %	2	163	0.15 %
600	9600	2	215	0.07 %	2	163	0.15 %	1	208	0.16 %	1	163	0.15 %
1200	19200	1	215	0.07 %	1	163	0.15 %	0	208	0.16 %	0	163	0.15 %
2400	38400	0	215	0.07 %	0	163	0.15 %	0	104	0.16 %	0	81	0.47 %
4800	76800	0	107	0.39 %	0	81	0.47 %	0	52	0.16 %	0	41	0.76 %
9600	153600	0	54	0.54 %	0	41	0.76 %	0	26	0.16 %	0	20	1.73 %
10400	166400	0	50	0.84 %	0	38	1.16 %	0	24	0.16 %	0	19	1.16 %
19200	307200	0	27	0.54 %	0	20	1.73 %	0	13	0.16 %	0	10	1.73 %
38400	614400	0	13	3.29 %	0	10	1.73 %	0	7	6.99 %Note	0	5	1.73%
76800	1228800	0	7	4.09 %	0	5	1.73 %	–	–	–	0	3	15.2 %Note
153600	2457600	0	3	11.90 %Note	0	2	27.2 %Note	–	–	–	–	–	–

Baud rate [bps]		ϕ = 20 MHz			ϕ = 14.746 MHz			ϕ = 12.288 MHz			ϕ = 9.830 MHz		
UART0,UART1	CSI0 to CSI3	BPR	BRG	Error	BPR	BRG	Error	BPR	BRG	Error	BPR	BRG	Error
110	1760	4	178	0.25 %	4	131	0.07 %	3	218	0.08 %	3	175	0.26 %
150	2400	4	130	0.16 %	3	192	0.0 %	3	160	0.0 %	3	128	0.0 %
300	4800	3	130	0.16 %	2	192	0.0 %	2	160	0.0 %	2	128	0.0 %
600	9600	2	130	0.16 %	1	192	0.0 %	1	160	0.0 %	1	128	0.0 %
1200	19200	1	130	0.16 %	0	192	0.0 %	0	160	0.0 %	0	128	0.0 %
2400	38400	0	130	0.16 %	0	96	0.0 %	0	80	0.0 %	0	64	0.0 %
4800	76800	0	65	0.16 %	0	48	0.0 %	0	40	0.0 %	0	32	0.0 %
9600	153600	0	33	1.36 %	0	24	0.0 %	0	20	0.0 %	0	16	0.0 %
10400	166400	0	30	0.16 %	0	22	0.7 %	0	18	2.6 %	0	15	1.5 %
19200	307200	0	16	1.73 %	0	12	0.0 %	0	10	0.0 %	0	8	0.0 %
38400	614400	0	8	1.73 %	0	6	0.0 %	0	5	0.0 %	0	4	0.0 %
76800	1228800	0	4	1.73 %	0	3	0.0 %	0	3	16.7 %Note	0	2	0.0 %
153600	2457600	0	2	1.73 %	0	2	25.0 %Note	–	–	–	0	1	0.0 %
307200	4915200	0	1	1.73 %	–	–	–	–	–	–	–	–	–

Note: Error is too great to use.

Remark: BPR: Prescaler setting (set using BPRMn register)
BRG: Timer count value (set using BRGCn register)
ϕ: Internal system clock

(3) Baud rate error

The baud generator error is represented as follows.

$$\text{Error [\%]} = \left(\frac{\text{Actual baud rate (baud rate with error)}}{\text{Desired baud rate (normal baud rate)}} \right) - 1 \quad \times 100$$

Examples: $(\frac{9520}{9600} - 1) \times 100 = -0.833$ [%]

$(\frac{5000}{4800} - 1) \times 100 = +4.167$ [%]

The permissible error range of the baud rate depends on the number of bits per frame. Using 16 bits, a baud rate error of $\pm$5% and sample timing of $\pm$ 4.5% are basic permissible limits. However, the permissible limit in actual use is a baud rate error of $\pm$2.3%, assuming both transmit end and receive end error is included.

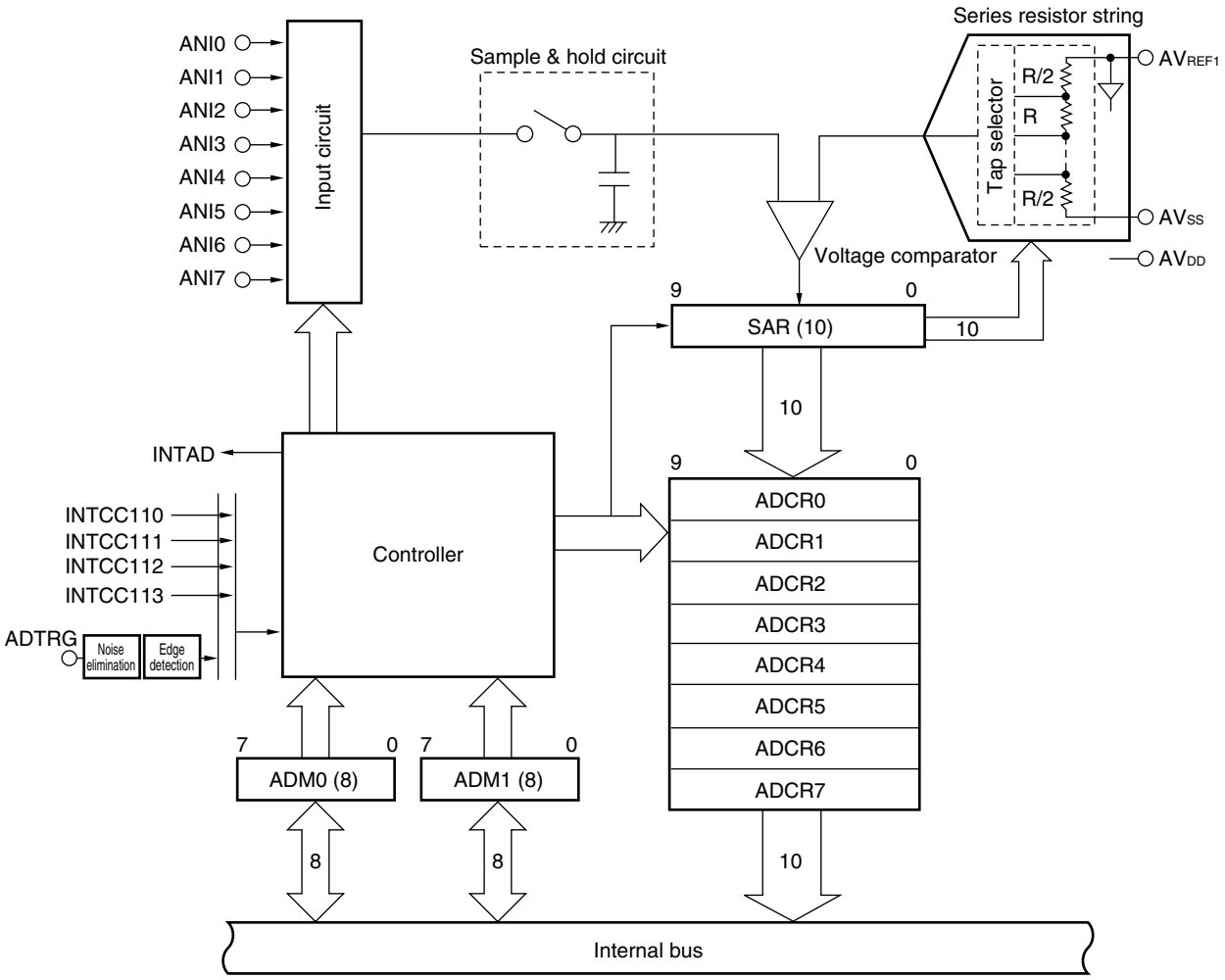
3.5 A/D Converter Function

The A/D converter function of the V853 has the following features.

- Analog input: 8 channels
- On-chip 10 bit A/D converter
- On-chip conversion result registers (ADCR0 to ADCR7): 10 bits x 8
- Selection from 3 kinds of A/D conversion trigger: A/D trigger mode
  : Timer trigger mode
  : External trigger mode
- Adopts a successive approximation method

Figure 3-55 shows a block diagram of the A/D converter.

★ ***Figure 3-55: Block Diagram of A/D Converter***



The operation of the A/D converter is determined by the combination of three trigger modes and two operating modes. The following table shows the A/D conversion start timing for the three trigger modes.

Table 3-9: A/D Conversion Start Timing

Trigger mode	Timing of A/D conversion start
A/D trigger mode	Set CE bit of ADM0 register to 1
Timer trigger mode	TM11 compare register (CC110 to CC113) match interrupt
External trigger mode	Effective edge to ADTRG pin

The analog inputs that can select the trigger modes shown in Table 3-9 are the following.

- A/D trigger mode: ANI0 to ANI7 (all analog input pins)
- Timer trigger mode: ANI0 to ANI3
- External trigger mode: ANI0 to ANI3

Table 3-10 shows features of the A/D conversion operating modes.

Table 3-10: Features of A/D Conversion Operating Modes

Operating mode	Features of operation
Select mode (1-buffer mode)	Optimal when reading A/D conversion result each time
Select mode (4-buffer mode)	Optimal when finding average of A/D conversion results
Scan mode	Optimal when always monitoring multiple analog inputs

Figure 3-56: One-Buffer Mode

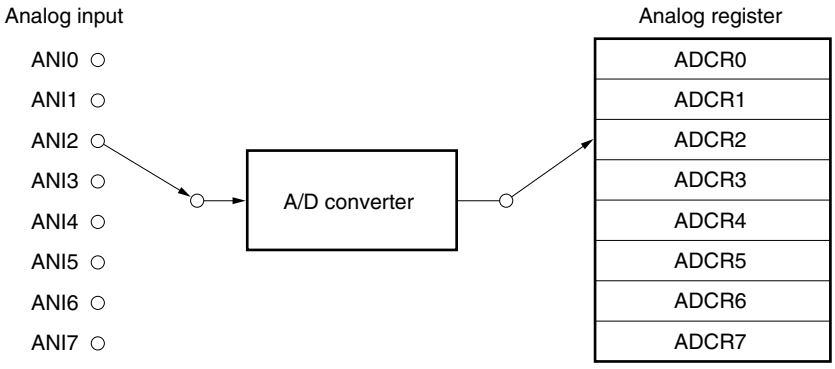


Figure 3-57: Four-Buffer Mode

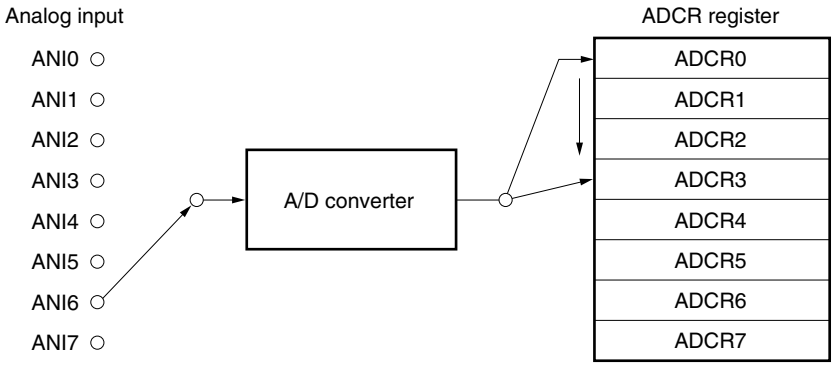
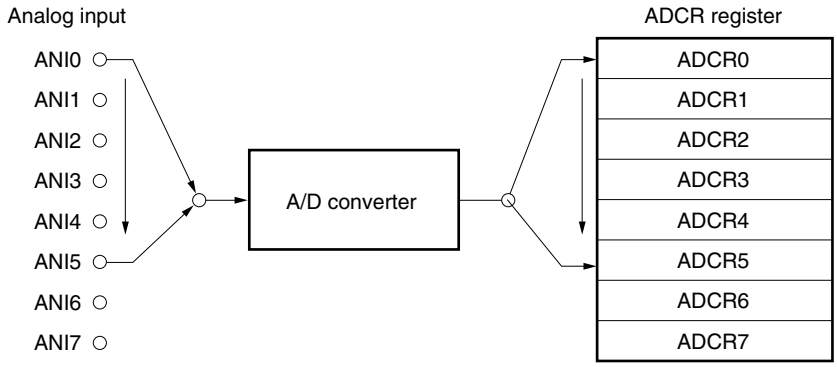


Figure 3-58: Scan Mode



The operation of A/D conversion is selected using A/D converter mode registers 0 and 1 (ADM0, ADM1).

Figure 3-59: A/D Converter Mode Register 0 (ADM0)

	7	6	5	4	3	2	1	0	Address	After reset
ADM0	CE	CS	BS	MS	0	ANIS2	ANIS1	ANIS0	FFFFF380H	00H

Bit position	Bit name	Description
7	CE	Convert Enable Specifies A/D conversion operation enabled/disabled. 0: Disabled 1: Enabled
6	CS	Convert Status Shows status of A/D converter. This bit is read-only. 0: Stopped 1: Operating
5	BS	Buffer Select Specifies buffer mode in select mode. 0: 1-buffer mode 1: 4-buffer mode
4	MS	Mode Select Specifies operating mode of A/D converter. 0: Scan mode 1: Select mode
2 to 0	ANIS2 to ANIS0	Analog Input Select Specifies A/D conversion analog input pin

ANIS2	ANIS1	ANIS0	Select mode	Scan mode	
				A/D trigger mode	Timer trigger mode[Note]
0	0	0	ANI0	ANI0	1 time
0	0	1	ANI1	ANI0, ANI1	2 times
0	1	0	ANI2	ANI0 to ANI2	3 times
0	1	1	ANI3	ANI0 to ANI3	4 times
1	0	0	ANI4	ANI0 to ANI4	ANI0 to ANI4
1	0	1	ANI5	ANI0 to ANI5	ANI0 to ANI5
1	1	0	ANI6	ANI0 to ANI6	ANI0 to ANI6
1	1	1	ANI7	ANI0 to ANI7	ANI0 to ANI7

Note When using timer trigger mode in scan mode, since the scan order of pins ANI0 to ANI3 follows the order of compare register match signal generation, a specific analog input pin is not specified but the number of trigger inputs is specified.

Caution: If the CE bit is 1 in timer trigger mode or external trigger mode, the status is awaiting trigger signal. To clear the CE bit, write "0" or reset it. Writing 1 to the CE bit in A/D trigger mode is a conversion trigger. After the operation, if the mode is changed to the timer trigger mode or external trigger mode not clearing the CE bit, trigger input wait status enters immediately after the mode is changed.

Figure 3-60: A/D Converter Mode Register 1 (ADM1)

	7	6	5	4	3	2	1	0	Address	After reset
ADM1	0	TRG2	TRG1	TRG0	0	FR2	FR1	FR0	FFFFF382H	07H

Bit position	Bit name	Description
6 to 4	TRG2 to TRG0	Trigger Mode Specifies the trigger mode.

TRG2	TRG1	TRG0	Trigger mode
0	0	x	A/D trigger mode
0	1	0	Timer trigger mode (1-trigger mode)
0	1	1	Timer trigger mode (4-trigger mode)
1	1	0	External trigger mode
Others			Setting prohibited

Remarks x: Arbitrary

Specify the effective edge of the external input signal in external trigger mode using bits 7 and 6 (ES031, ES030) of the external interrupt mode register (INTM1). (See **Figure 3-11 External Interrupt Mode Registers 1 to 4 (INTM1 to INTM4)**.)

Bit position	Bit name	Description
2 to 0	FR2 to FR0	Frequency Specifies the conversion operating time. These are control bits for making the A/D conversion time largely not change even if the oscillation frequency is changed.

FR2	FR1	FR0	Conversion clocks	Conversion operating time (μs)		
				ϕ = 33 MHz	ϕ = 25 MHz	ϕ = 16 MHz
0	0	0	36	–	–	2.25
0	0	1	48	–	1.92	3.00
0	1	0	60	1.82	2.40	3.75
0	1	1	72	2.18	2.88	4.50
1	0	0	96	2.91	3.84	6.00
1	0	1	120	3.64	4.80	7.50
1	1	0	144	4.36	5.76	9.00
1	1	1	192	5.82	7.68	12.00

Remarks 1. The conversion operating time is a target value.

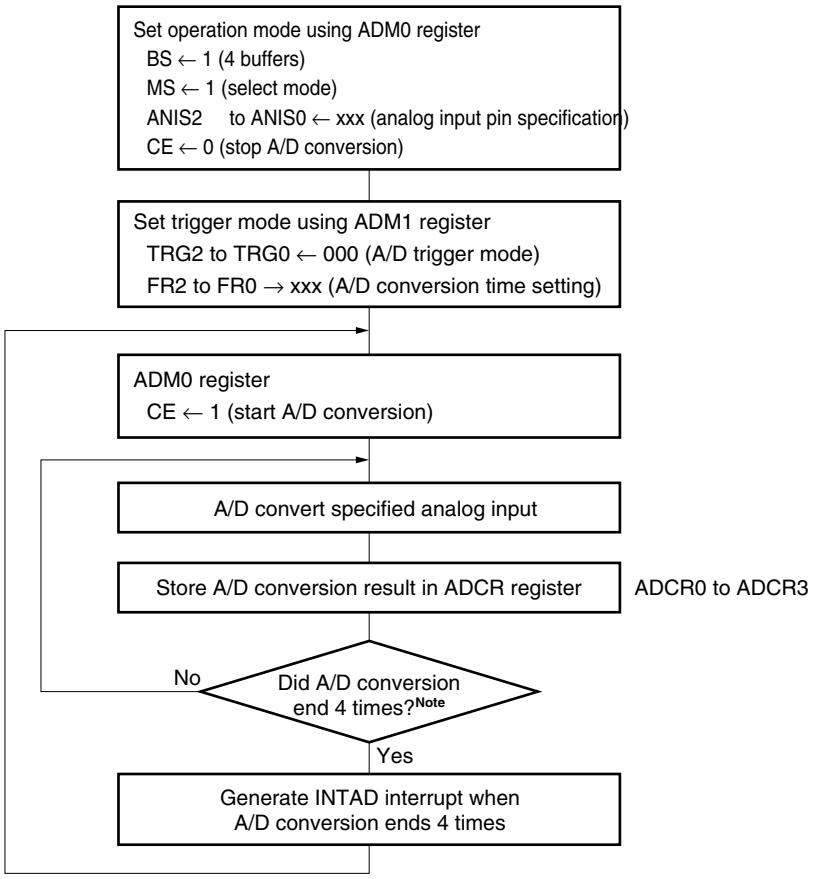
2. ϕ = internal system clock frequency

A timer trigger mode of one-trigger mode makes a TM11 compare register (CC110) match interrupt an A/D conversion trigger. Four-trigger mode differs by starting A/D conversion on a match interrupt from any of the TM11 compare registers (CC110 to CC113).

3.5.1 Operation in A/D trigger mode

Figure 3-61 shows the flow of A/D conversion operation in A/D trigger mode.

Figure 3-61: A/D Trigger Mode + Select Mode Operation

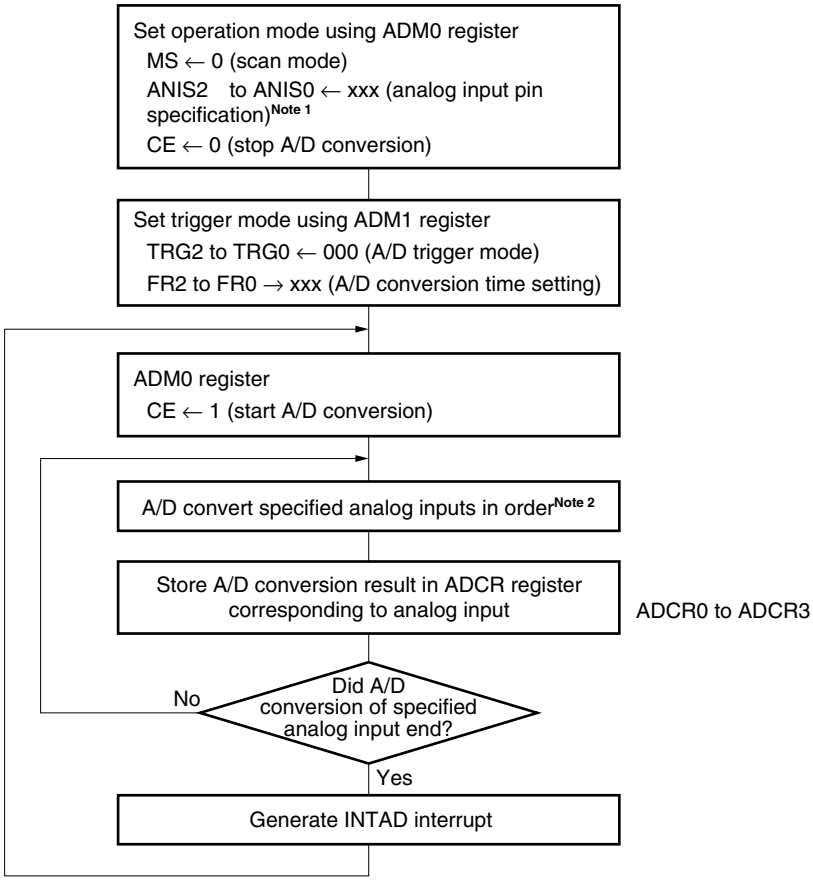


Note: In one-buffer mode, an INTAD interrupt occurs at A/D conversion end.

In one-buffer mode of select mode, ADCR registers that store A/D conversion results correspond one-to-one to the converted analog inputs (for example, stores input = ANI3 pin in ADCR3).

In four-buffer mode, A/D conversion results are stored in ADCR0 to ADCR3. Conversion results are stored in order starting from ADCR0.

Figure 3-62: A/D Trigger Mode + Scan Mode Operation

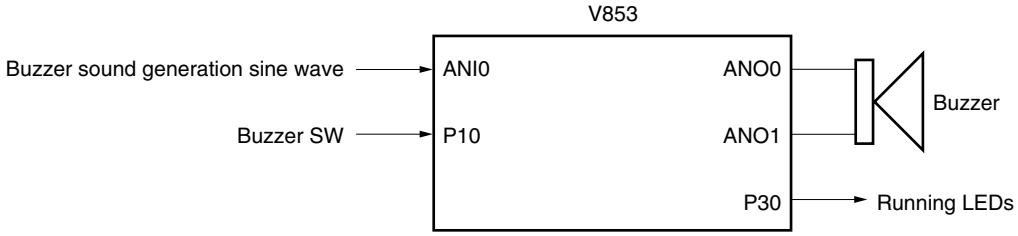


Notes:1. Multiple analog input pins can be specified for conversion.

2. A/D convert from ANI0 through the specified range.

3.5.2 A/D trigger mode sample program

Figure 3-63: Sample Hardware Configuration



(1) Function overview

- Input the buzzer sound generation sine wave from the ANI0 pin, write the A/D converted value to the D/A conversion register, and output to pins ANO0 and ANO1.
- Sound is output when the buzzer SW is ON.

Set ADM0 and ADM1 as follows.

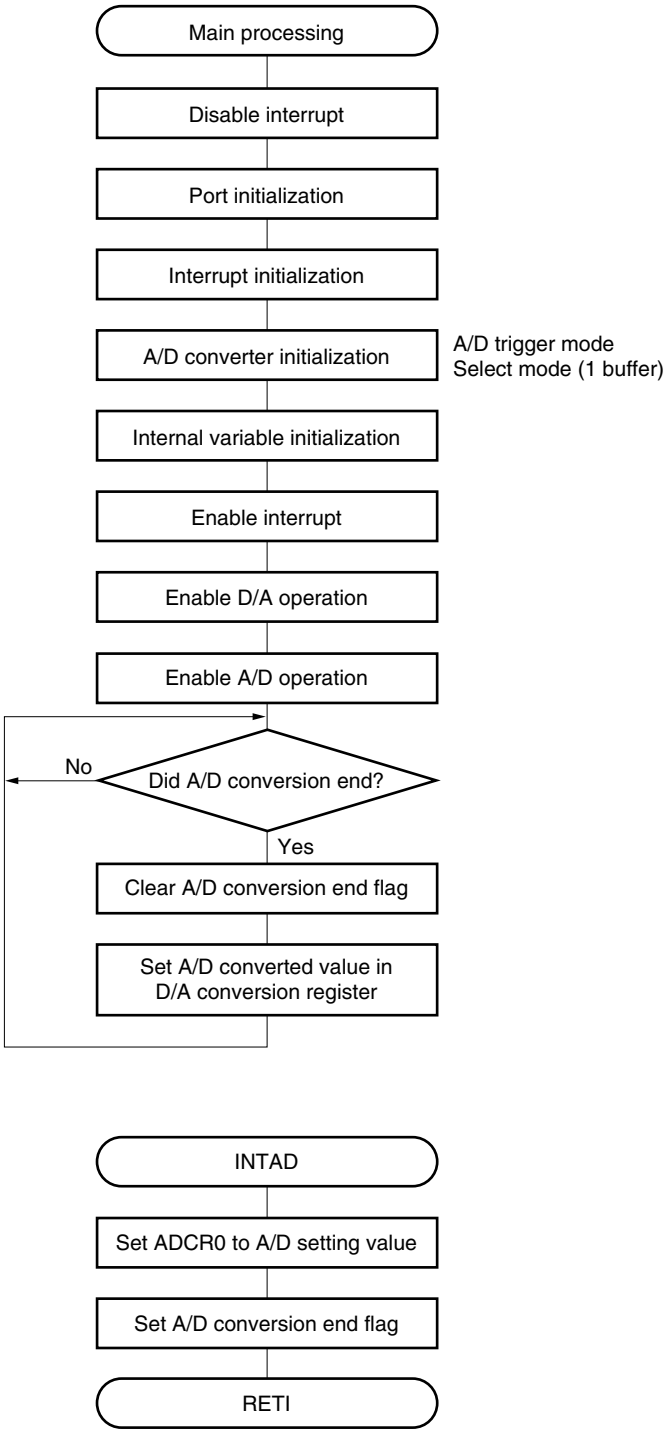
<1> Set the A/D converter trigger mode to A/D trigger mode.
<2> Set the operating mode to select mode (one-buffer mode).
<3> Specify ANI0 for the analog input pin.
<4> Specify 192 for the number of conversion clocks.

Figure 3-64: ADM0 and ADM1 Settings

	7	6	5	4	3	2	1	0	Address	After reset
ADM0	0	0	0	1	0	0	0	0	FFFFF380H	00H
ADM1	0	0	0	0	0	1	1	1	FFFFF382H	07H

The flow of buzzer sound modulation processing that operates on the sample hardware is shown below.

Figure 3-65: Flow of Buzzer Sound Modulation Processing



(2) Sample program

```
/* ————————————————————————-
 * V853 A/D converter sample program
 * (A/D trigger mode)
 *
 * Initialization program for the following hardware configuration
 * ANI0 : Buzzer sound generation sine wave
 * ANO0/1: Buzzer output
 * P10 : Run SW (ON: Run, OFF: Stop)
 * P30 : Running LEDs
 *
 * ————————————————————————-
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/8  adapted to IAR compiler by NEC Duesseldorf AH
 * File Name: buzzer.c
 * ————————————————————————-
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

static long BuzzerDat; /* D/A conversion output value */
static int AdFlag;     /* A/D conversion completion flag */

/* ————————————————————————-
NMI servicing
Interrupt source: NMI
Arguments: None
Return value: None
 ————————————————————————-
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   ; /* Do nothing */
}

/* ————————————————————————-
A/D converter conversion end
Interrupt source: INTAD
Arguments: None
Return value: None
 ————————————————————————-
*/
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   /* Assign conversion results */
   BuzzerDat = ADCR0H; /* ADCR0 is 10-bit data. */
                       /* Since high-order 8 bits are required, */
                       /* ADCR0H is used. */
   AdFlag = 1; /* Conversion end flag ON */
}
```

```
/* ————————————————————————-
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
void PortInit(void)
{
   /* Initialize port 0 (P0))
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x00; /* P00 to P07 in port mode */

   /* Initialize port 1 (P1)
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize port 2 (P2)
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x00; /* P20 to P27 in port mode */

   /* Initialize port 3 (P3)
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF;  /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}
```

```
/* ————————————————————————-
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   ADIC = 0x07; /* INTAD: A/D conversion end: Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x00; /* Not used */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}

/* ————————————————————————-
 * A/D converter initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/
void ADCInit(void)
{
   /* ADM0 register settings */
   ADM0 = 0x10; /* (BS) 1-buffer mode */
                /* (MS) Select mode */
                /* (ANIS2 to ANIS0) Input analog signal from ANI0 */

   /* ADM1 register settings */
   ADM1 = 0x07; /* (TRG2 to TRG0) A/D trigger mode */
                /* (FR2 to FR0) Conversion clocks = 192 */
}

/* ————————————————————————-
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————————————-
*/

   /* Interrupt initialization processing */
   IntInit();

   /* A/D converter initialization processing */
   ADCInit();

   /* Initialize internal data */
   BuzzerDat = 0;
   AdFlag = 0;

   /* Start D/A conversion */
   DACS0 = 0; /* Initialize output registers */
   DACS1 = 0;

   DAM = 3; /* Enable output operation */

   /* Start A/D conversion */
```

```
    /* Interrupt initialization processing */
    IntInit();

    /* A/D converter initialization processing */
    ADCInit();

    /* Initialize internal data */
    BuzzerDat = 0;
    AdFlag = 0;

    /* Start D/A conversion */
    DACS0 = 0; /* Initialize output registers */
    DACS1 = 0;

    DAM = 3; /* Enable output operation */

    /* Start A/D conversion */
    ADM0_bit.no7= 1;   /* Set bit CE */

    /* Enable interrupt */
    __EI();

    /* Clear LEDs */
    P3 = 0;

    while ( 1 ) {
       /* Execution switch ON */
       if ( P1_bit.no0 != 0 ) {
          /* A/D conversion completion */
          if ( AdFlag != 0 ) {
             /* Set A/D converted value in D/A register */
             DACS0 = BuzzerDat;
             DACS1 = BuzzerDat;
             AdFlag = 0;
             ADM0_bit.no7= 1;  /* Start AD-Conversion */
          }
          if ( StartFlag == 0 ) {
             /* LEDs ON */
             P3_bit.no0 = 1;
             StartFlag = 1;
          }
       }
       /* Execution switch OFF */
       else {
          if ( StartFlag == 1 ) {
             /* Clear LEDs */
             P3_bit.no0 = 0;
             StartFlag = 0;
          }
       }
    }
}
```

3.5.3 Operation in timer trigger mode

Figure 3-66 shows the flow of A/D conversion operation in timer trigger mode.

Figure 3-66: Timer Trigger Mode + Select Mode Settings and Operation

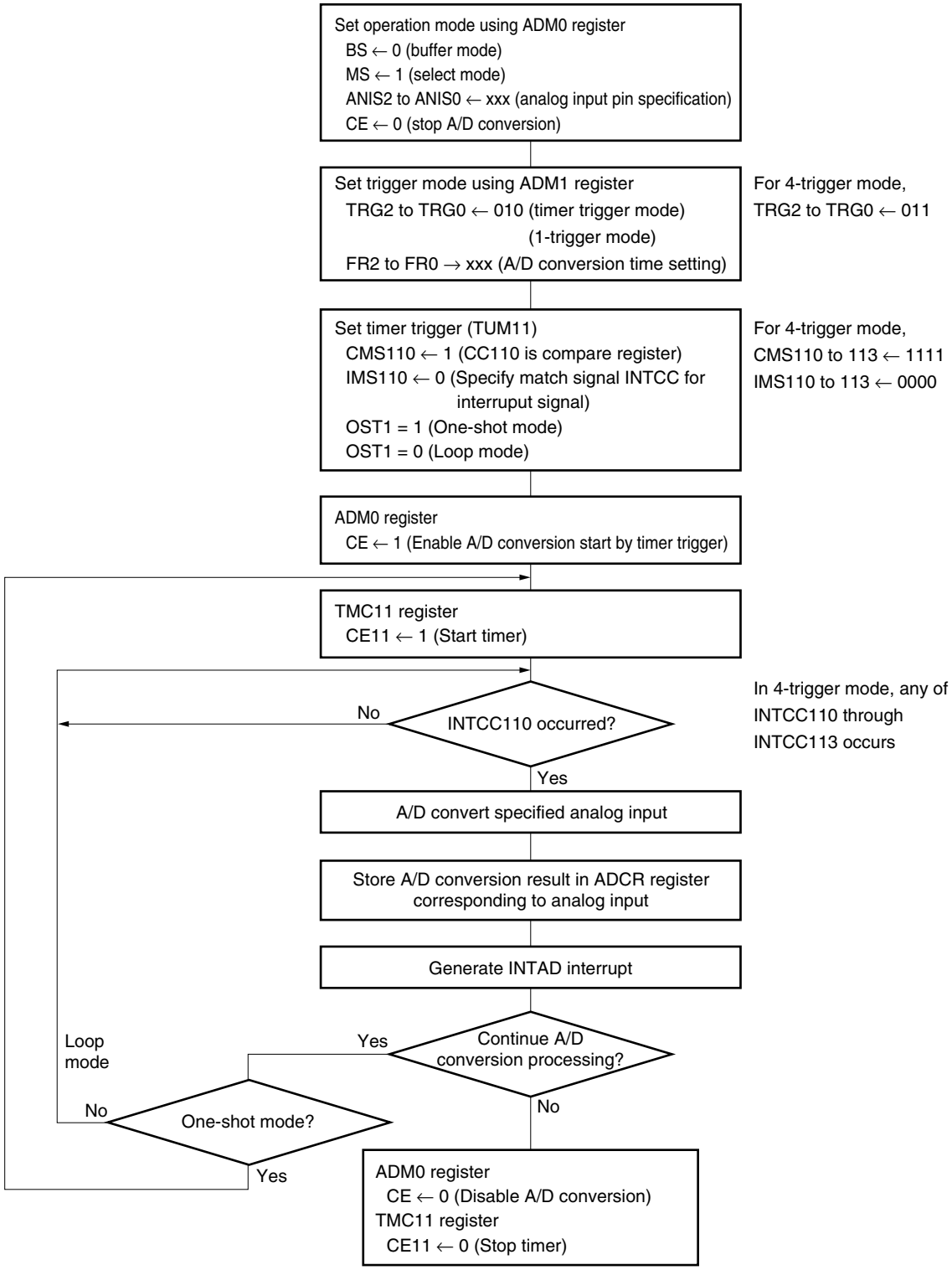
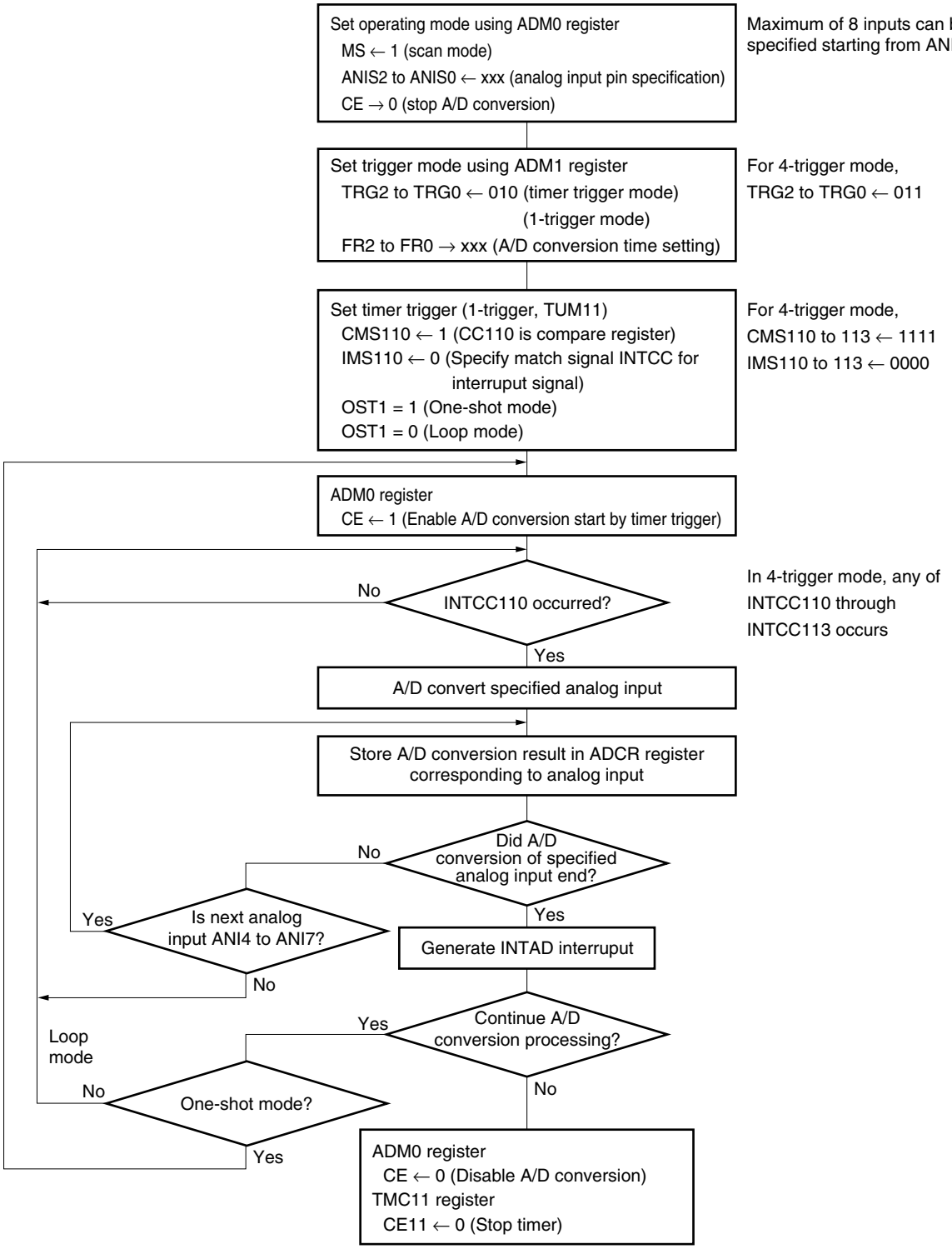
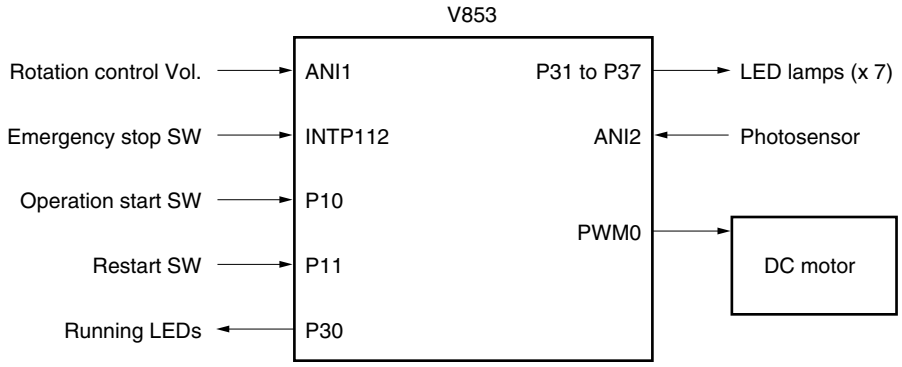


Figure 3-67: Timer Trigger Mode + Scan Mode Settings and Operation



3.5.4 Timer trigger mode sample program

Figure 3-68: Sample Hardware Configuration



(1) Function overview

- Input three kinds of analog signal from pins ANI0 to ANI2 and output A/D converted values as follows.
  <1> Rotation control Vol. (ANI1): DC motor
  <2> Photosensor (ANI2): LED lamp
- If LED lamp display is bright, many LEDs are lit.
- The motor is started when the operation start SW is ON and stopped when it is OFF. To stop rotation at once, stop the motor using the emergency stop SW. Restart it using the restart SW.

Set ADM0 and ADM1 as follows.
<1> Make the A/D converter trigger mode A/D trigger mode.
<2> Make the operating mode scan mode (one-trigger mode).
<3> Specify ANI0 to ANI3 for the analog input pins.
<4> Specify 192 for the number of conversion clocks.

Figure 3-69: ADM0 and ADM1 Settings

	7	6	5	4	3	2	1	0		
ADM0	0	0	0	0	0	0	1	1	Address FFFFF380H	Value 03H
ADM1	0	0	1	0	0	1	1	1	Address FFFFF382H	Value 27H

Since the trigger mode is timer trigger mode, set TM11 as follows.

<1> Timer 11 continues counting up even after an overflow.
<2> The counter is not cleared by external input.
<3> Specify CC110 as compare register to generate INTCC10, which is specified as A/D conversion start trigger.
<4> Since INTP112 is used, specify CC112 as a compare register (because it is not captured by effective edge of interrupt). Also select INTP112 as an interrupt source.
<5> Use an internal count clock.

Set timer unit mode register 11 (TUM11) and timer control register 11 (TMC11) using the above conditions.

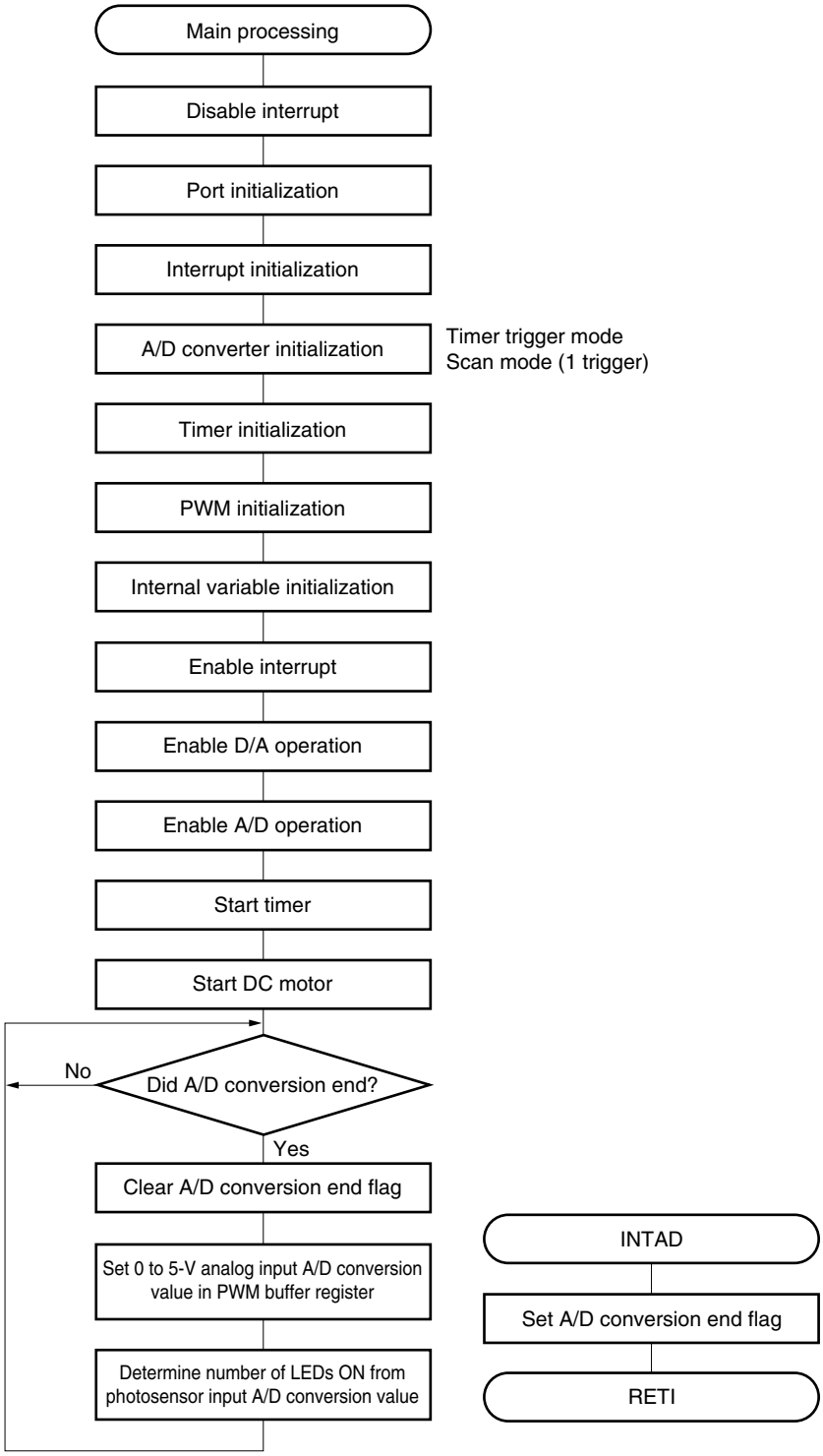
Figure 3-70: TUM11 and TMC11 Settings

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
TUM11	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	Address FFFFF240H	Value 0054H

	7	6	5	4	3	2	1	0		
TMC11	0	0	0	0	1	0	0	0	Address FFFFF242H	Value 08H

The following figure shows the flow of buzzer sound modulation processing that operates on the sample hardware.

Figure 3-71: Flow of Buzzer Sound Modulation Processing



(2) Sample program

```
/* ————————————————————
 * V853 A/D converter sample program
 * (timer trigger & scan mode)
 *
 * Initialization program for the following hardware configuration
     ANI1 : DC motor rotation control input
     ANI2 : Photosensor input
     PWM0 : DC motor control
     INTP112 : Motor emergency stop SW
     P10 : Run SW (ON: Run, OFF: Stop)
     P11 : DC motor restart SW (ON: Run)
     P30 : Running LEDs
     P31 to P37: Photosensor gauge LEDs
 * ————————————————————
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/8  adapted to IAR compiler by NEC Nuesseldorf AH
 * File Name: adtimtrg.c
 * ————————————————————
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

/* ————————————————————————————
 * Internal variable definition
 * ————————————————————————————
*/
  /* Variables to store values after A/D conversion */
static short ContVol = 0; /* Value of DC motor rotation control Vol. */
static short OptSensor = 0; /* Value of photosensor */

  /* Operation flags */
static int ADCFlg = 0; /* A/D conversion end flag (!0: Conversion end) */
static int EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop
state) */
static int StartFlg = 0; /* Operation start flag */

/* ————————————————————
 * NMI servicing
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
  ; /* Do nothing */
}
```

```
/* ————————————————————
 * Motor emergency stop processing
 * Interrupt source: INTP112
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTP112_vector
__flat __interrupt void int_p112(void)
{
   /* Process only while executing */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM operation */
   PWM0 = 0;
   PWMC_bit.no3= 0;   /* Disable PWM */
}

/* ————————————————————
 * A/D converter conversion end
 * Interrupt source: INTAD
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   /* DC motor rotation control input */
   ContVol = ADCR1;
   /* Photosensor input */
   OptSensor = ADCR2;
   /* Set A/D conversion end flag */
   ADCFlg = 1;
}

/* ————————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PortInit(void)
{
   /* Initialize port 0 (P0)
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x40; /* Use INTP112 P06 in control mode */
                /* P00 to P05, P07 in port mode */

   /* Initialize port 1 (P1)
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */
```

```
   /* Initialize port 2 (P2)
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x01; /* Use PWM0 P20 in control mode */
                /* P21 to P27 in port mode */

   /* Initialize port 3 (P3)
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF; /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}

/* ————————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: DC motor emergency stop: Level 7 */
   ADIC = 0x07;   /* INTAD : A/D conversion end : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x10; /* INTP112: Rising edge */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}
```

```
/* ————————————————————
 * A/D converter initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void ADCInit(void)
{
   /* ADM0 register settings */
   ADM0 = 0x02; /* (MS) Scan mode */
                /* (ANIS2 to ANIS0) Input analog signal from ANI0 to ANI3 */

   /* ADM1 register settings */
   ADM1 = 0x27; /* (TRG2 to TRG0) Timer trigger mode */
                /* (1-trigger mode) */
                /* (FR2 to FR0) Conversion clocks = 192 */
}

/* ————————————————————
 * Timer initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void TimerInit(void)
{
   /* Timer 11 settings (TUM11) */
   TUM11 = 0x0054; /* OST4   = 0: Continue to count up after timer overflow
*/
                   /* ECLR14 = 0: Do not enable external clear input */
                   /* CMS112 = 1: To use INTP112 signal */
                   /* CMS110 = 1: Make compare register */
                   /* IMS112 = 1: Interrupt source INTP112 */
   /* Timer 11 settings (TMC11) */
   TMC11 = 0x08; /* ETI11  = 0: Timer count clock is internal */
                 /* PRM111 = 0: Intermediate clock is 1/2 system clock */
                 /* PRS111 = 1: Timer count clock is f m/8 */
                 /* PRS110 = 0: (Timer count clock = f/16) */
                 /* CE11   = 0: Stop timer */
}

/* ————————————————————
 * PWM unit initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PwmInit(void)
{
   /* PWM settings (PWMC): Set PWM0 only */
   PWMC = 0x02; /* PWME0 = 0: PWM in operation stopped state */
                /* ALV0 = 0: Active level is low */
                /* PRM01 = 1: Compare register is 10 bits */
                /* PRM00 = 0; */
   PWPR = 0x00; /* Prescaler is f/1 */
   PWM0 = 0;    /* Clear buffer register */
}
```

```
/* ————————————————————
 * LED gauge display processing
 * Arguments: value Value to display (16 bits)
 * Return value: None
 * ————————————————————
*/
void DispLEDGage (unsigned short value)
{
   unsigned short count;
   unsigned char leddat;

   /* Make A/D conversion effective range for photosensor 0x2A0 to 0x39F */
   if (value < 0x02A0) {
      value = 0; /* 0 if too small */
   }
   else if (value < 0x03A0) {
      value -= 0x02A0; /* Convert to 0 to 0xff */
   }
   else {
      value = 0xff; /* 0xff if too large */
   }
   /* LED data generation */
   for ( count = 0 , leddat = 0 ; count < 8 ; count++ ) {
       if ( value > ( count * 0x20 ) ) {
          leddat |= 1 << count;
       }
   }
   /* LED display */
   P3 = leddat;
}

/* ————————————————————
 * Operation start processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void StartProc(void)
{
   /* Start D/A conversion */
   DACS0 = 0;
   DACS1 = 0;
   DAM |= 3;  /* DACE0 = 1; DACE1 = 1; */

   /* Start A/D conversion */
   ADM0_bit.no7= 1;   /* CE = 1; */

   /* Start timer */
   TMC11_bit.no7= 1;   /* CE11 = 1 */

   /* Start PWM output */
   PWMC_bit.no3= 1;  /* PWME0 = 1 */

   /* Enable interrupt */
   __EI();
}
```

```
/* ————————————————————
 * Operation stop processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void StopProc(void)
{
   /* Disable interrupt */
   __DI();

   /* Stop D/A conversion */

   DAM &= 0xfc;   /*  DACE0 = 0; DACE1 = 0; */

   /* Stop A/D conversion */

   ADM0_bit.no7= 0;

   /* Stop timer */
   TMC11_bit.no7= 0;

   /* PWM output initialization */
   PWM0 = 0;

   /* Stop PWM output */
   PWMC_bit.no3= 0;   /* Disable PWM, PWME0= 0 */
}

/* ————————————————————
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void main(void)
{
   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* A/D converter initialization processing */
   ADCInit();

   /* Timer initialization processing */
   TimerInit();

   /* PWM initialization processing */
   PwmInit();
```

```
   /* Initialize internal data */
   ContVol = 0;      /* Value of DC motor rotation control Vol. */
   OptSensor = 0;    /* Value of photosensor */
   ADCFlg = 0;       /* A/D conversion end flag (!0: Conversion end) */
   EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop status) */
   StartFlg = 0;     /* Clear start flag */

   while ( 1 ) {
      /* Check for start switch ON */
      if ((P1_bit.no0 != 0) && (StartFlg == 0)) {
         StartProc();
         StartFlg = 1;
      }
      /* Stop until start flag is ON */
      if (StartFlg == 0) {
         StopProc();
         continue;
      }
      /* Check for start switch OFF */
      if ((P1_bit.no0 == 0) && (StartFlg != 0)) {
         StartFlg = 0;
         StopProc();
         /* Also clear emergency stop flag */
         EmergencyFlg = 0;
      }
      /* During emergency stop */
      if ( EmergencyFlg != 0 ) {
         /* Do not process until emergency stop cancelled */
         if ( P1_bit.no1 == 0 ) {
            continue;
         }
         /* Clear emergency stop flag */
         EmergencyFlg = 0;
         /* Start PWM operation */
          PWMC_bit.no3= 1;

      }
      /* A/D conversion ended? */
      if (ADCFlg != 0) {
         /* Clear A/D conversion end flag */
         ADCFlg = 0;
         /* Set value of DC motor control Vol. in PWM register */
         PWM0 = ContVol;
         /* Display value of photosensor in gauge LEDs */
         DispLEDGage(OptSensor);
      }
   }
}
```

3.5.5 Operation in external trigger mode

Figure 3-72 shows the flow of A/D conversion operation in external trigger mode.

Figure 3-72: External Trigger Mode + Select Mode Settings and Operation

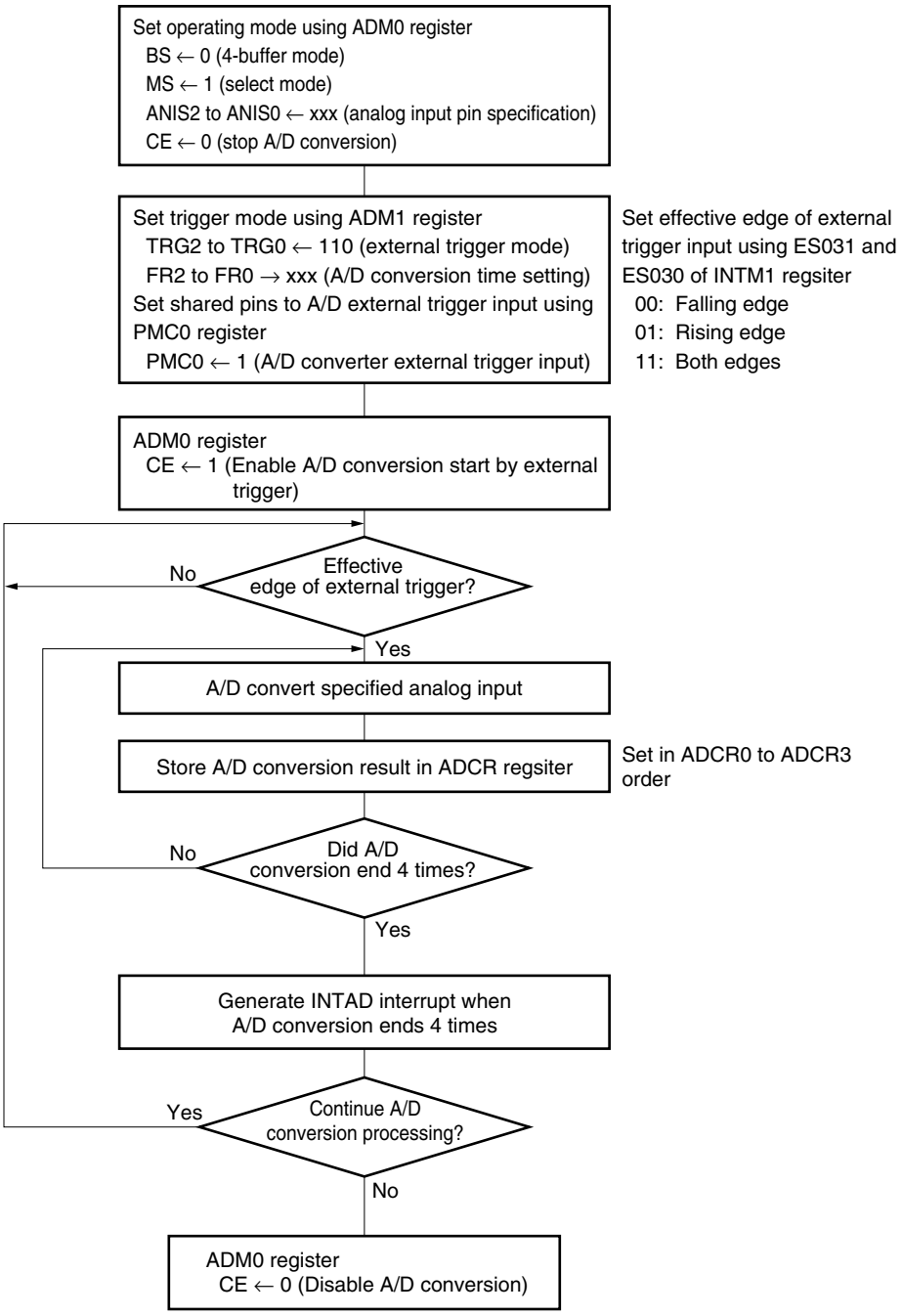
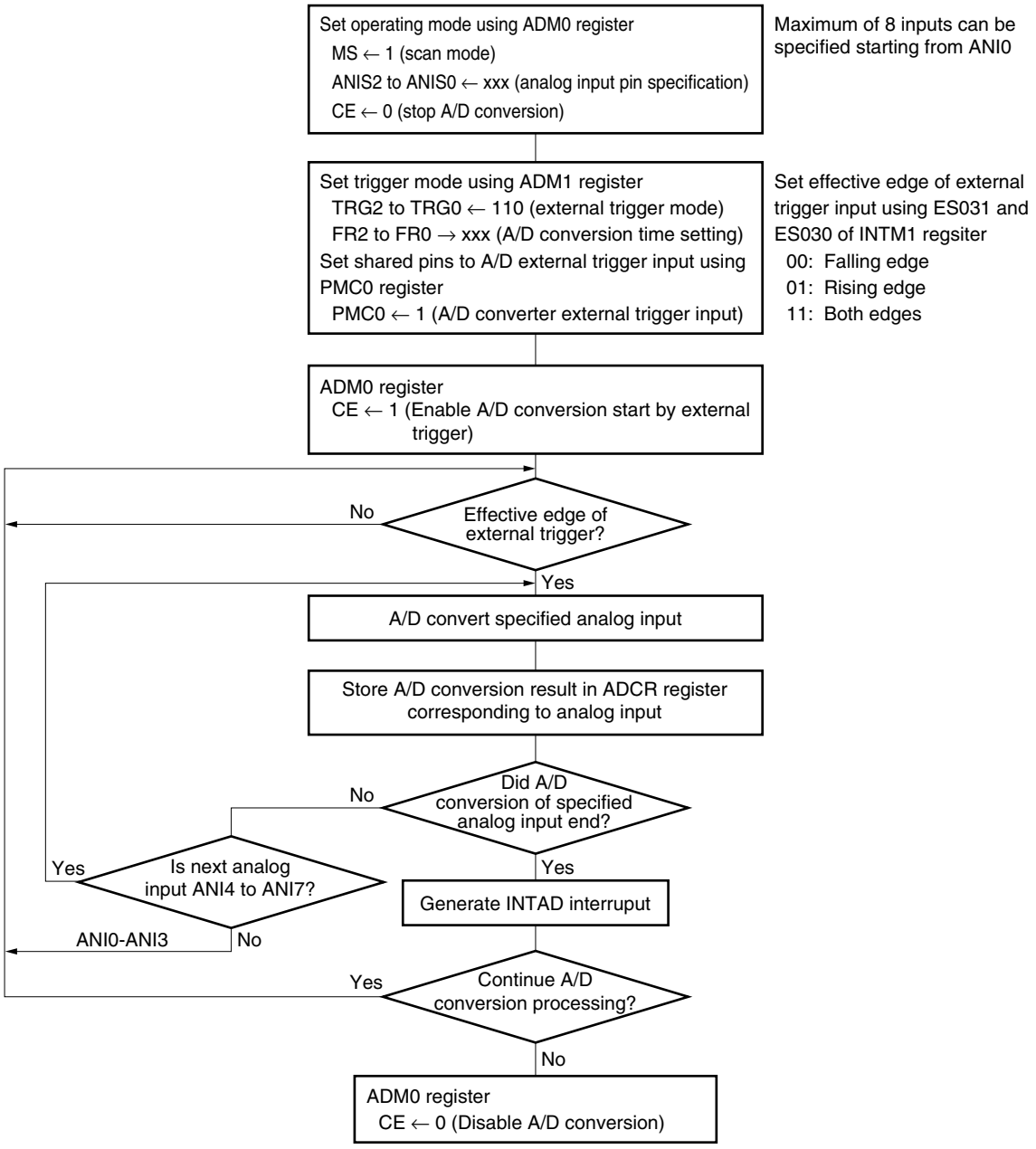
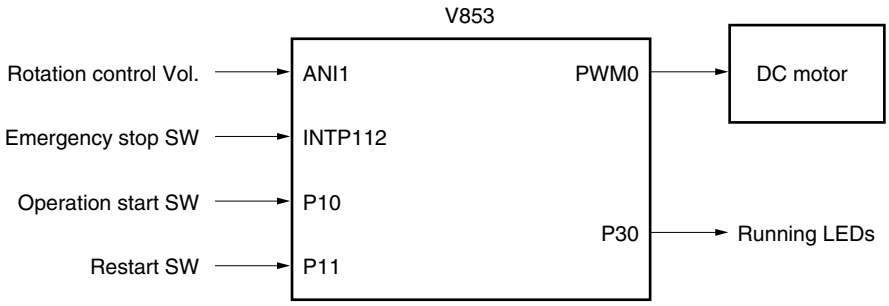


Figure 3-73: External Trigger Mode + Scan Mode Settings and Operation



3.5.6 External trigger mode sample program

Figure 3-74: Sample Hardware Configuration



(1) Function overview

- From the value A/D input from the rotation control Vol., use the PWM function to make the DC motor rotate.
- Start the motor by turning the operation start SW ON and stop it by turning it OFF. To make the rotation stop right away, stop the motor using the emergency stop SW. Restart it using the restart SW.

Set ADM0 and ADM1 as follows.

<1> Specify the external trigger mode for the trigger mode of the A/D converter.
<2> Specify the select mode (1-buffer mode) for the operating mode.
<3> Specify ANI1 for the analog input pin.
<4> Specify 192 for the number of conversion clocks.
<5> Use the ADTRG pin signal only as an A/D conversion trigger.

Figure 3-75: ADM0 and ADM1 Settings

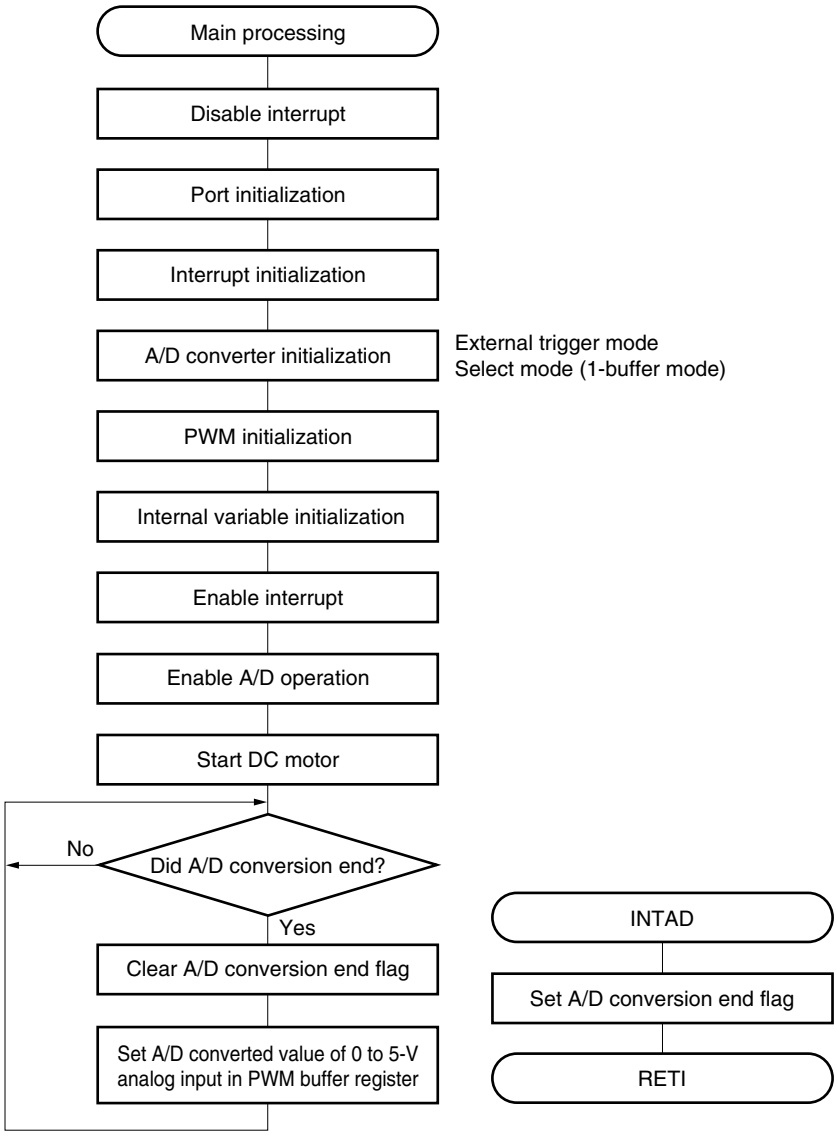
	7	6	5	4	3	2	1	0		
ADM0	0	0	0	1	0	0	0	1	Address FFFFF380H	Value 11H
ADM1	0	1	1	0	0	1	1	1	Address FFFFF382H	Value 67H

Because the trigger mode was made external trigger mode, an action is performed on INTP113 which is shared with the ADTRG pin. In the sample program, it is assumed that an adjusted waveform is put into INTP113.

- PMC0 register settings
  Since INTP113 is used as a trigger, set port 07 to control mode.
- INTM1 register settings
  Set INTP113 to rising edge.
- P11IC3 register settings
  Since INTP113 is not used, mask interrupts using an interrupt control register. The default value of P11IC3 is masked.

The following figure shows the flow of DC motor rotation control operating on the sample hardware.

Figure 3-76: Flow of DC Motor Rotation Control



(2) Sample program

```
/* ------------------------------
 * V853 A/D converter sample program
 * (external trigger mode)
 *
 * Initialization program for the following hardware configuration
 *   ANI1 : DC motor rotation control input
 *   PWM0 : DC motor control
 *   INTP112: Motor emergency stop SW
 *   P10 : Run SW (ON: Run, OFF: Stop)
 *   P11 : DC motor restart SW (ON: Run)
 *   P30 : Running LEDs
 * ------------------------------
 * History:
 * 1997/3/17 Ver 0.00.00
 * 1999/7/8 adapted to IAR compiler by NEC Duesseldorf AH
 * File Name: exttrig.c
 * ------------------------------
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

/* ------------------------------------
 * Internal variable definition
 * ------------------------------------
*/
  /* Variables to store values after A/D conversion */
static short ContVol = 0; /* Value of DC motor rotation control Vol. */
  /* Operation flags */
static int ADCFlg = 0;       /* A/D conversion end flag (!0: Conversion end)
*/
static int EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop
state) */
static int StartFlg = 0;     /* */

/* ----------------------
 * NMI servicing
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ----------------------
*/
#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   ; /* Do nothing */
}

/* ----------------------
 * A/D converter conversion end
 * Interrupt source: INTAD
 * Arguments: None
 * Return value: None
 * ----------------------
*/
```

```
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   /* Assign conversion result */
   ContVol = ADCR1;
   /* A/D conversion end flag ON */
   ADCFlg = 1;
}

/* ————————————————————-
 * DC motor emergency stop processing
 * Interrupt source: INTP112
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
#pragma vector=INTP112_vector
__flat __interrupt void int_p112(void)
{
   /* Process only during execution */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM output */
   PWM0 = 0;
   PWMC_bit.no3= 0;   /* Clear Bit PWME0 */
}

/* ————————————————————-
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void PortInit(void)
{
   /* Initialize port 0 (P0)
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0xC0; /* Use INTP112 P06 in control mode */
                /* Use A/D trigger P07 in control mode */
                /* P00 to P05 in port mode */

   /* Initialize port 1 (P1)
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize port 2 (P2)
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x01; /* Use PWM0 P20 in control mode */
                /* P21 to P27 in port mode */
```

```
   /* Initialize port 3 (P3)
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF; /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}

/* ————————————————————-
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: DC motor emergency stop: Level 7 */
   ADIC = 0x07;   /* INTAD : A/D conversion end : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x50; /* INTP113, INTP112: Rising edge */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}
```

```
/* ————————————————————-
 * A/D converter initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void ADCInit(void)
{
   /* ADM0 register settings */
   ADM0 = 0x11; /* (BS) 1 buffer mode */
                /* (MS) Select mode */
                /* (ANIS2 to ANIS0) Input analog signal from ANI1 */

   /* ADM1 register settings */
   ADM1 = 0x67; /* (TRG2 to TRG0) External trigger mode */
                /* (FR2 to FR0) Conversion clocks = 192 */
}

/* ————————————————————-
 * PWM unit initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void PwmInit(void)
{
   /* PWM settings (PWMC): Set PWM0 only */
   PWMC = 0x02; /* PWME0 = 0: PWM in operation stopped state */
                /* ALV0 = 0: Active level is low */
                /* PRM01 = 1: Compare register is 10 bits */
                /* PRM00 = 0; */
   PWPR = 0x00; /* Prescaler is f /1 */
   PWM0 = 0; /* Clear buffer register */
}

/* ————————————————————-
 * Operation start processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void StartProc(void)
{
   /* Start A/D conversion */

    ADM0_bit.no7= 1;    /* Set Bit CE */

   /* Start PWM output, PWME0=1 */
   PWMC_bit.no3= 1;   /* Set Bit PWME0 */

   /* Enable interrupt */
   __EI();
}
```

```
/* ————————————————————-
 * Operation stop processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void StopProc(void)
{
   /* Disable interrupt */
   __DI();

   /* Stop A/D conversion */
   ADM0_bit.no7= 0;   /* Clear Bit CE */

   /* PWM output initialization */
   PWM0 = 0;

   /* Stop PWM output */

   PWMC_bit.no3= 0;   /* Clear Bit PWME0 */
}

/* ————————————————————-
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————————-
*/
void main(void)
{
   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* A/D converter initialization processing */
   ADCInit();

   /* PWM initialization processing */
   PwmInit();

   /* Internal data initialization */
   ContVol = 0;      /* Value of DC motor rotation control Vol. */
   ADCFlg = 0;       /* A/D conversion end flag (!0: Conversion end) */
   EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop status) */
   StartFlg = 0;     /* Clear start flag */
```

```
   while ( 1 ) {
      /* Check for start switch ON */
      if ((P1_bit.no0 !=0) && (StartFlg == 0)) {
         StartProc();
         StartFlg = 1;
      }
      /* Do not process until start flag is ON */
      if (StartFlg == 0) {
         continue;
      }
      /* Check for start switch OFF */
      if ((P1_bit.no0 == 0) && (StartFlg != 0)) {
         StopProc();
         StartFlg = 0;
         /* Also clear emergency stop flag */
         EmergencyFlg = 0;
      }
      /* During emergency stop */
      if ( EmergencyFlg != 0 ) {
         /* Do not process until emergency stop is cancelled */
         if ( P1_bit.no1 == 0 ) {
            continue;
         }
         /* Clear emergency stop flag */
         EmergencyFlg = 0;
         /* Start PWM operation */
        PWMC = PWMC | 8;
      }
      /* Did A/D conversion end? */
      if (ADCFlg != 0) {
         /* Clear A/D conversion end flag */
         ADCFlg = 0;
         /* Set value of DC motor control Vol. in PWM register */
         PWM0 = ContVol;
      }
   }
}
```

3.5.7 Notes on A/D conversion operation

Note the following when using the A/D converter.

(1) ADM0 register

If the CE bit of the ADM0 register is cleared during a conversion operation, the conversion operation stops. The result is not stored in the ADCR register.

(2) Timer trigger mode

When set to timer trigger mode, a compare register match interrupt that is an A/D conversion trigger also is in effect for the CPU. Therefore, if you do not want the CPU to detect compare match interrupts, set (to 1) the interrupt mask bit of the interrupt control register (P11MKn bit of P11ICn register, n = 0 to 3).

(3) External trigger mode

Like in (2) above, when set to external trigger mode, the detection of the ADTRG pin effective edge that is an A/D conversion trigger is a capture trigger or external interrupt signal for the CPU. If capture operation is not required, make the capture/compare register setting compare register. In addition, if you do not want external interrupts to be detected, set (to 1) the interrupt mask bit of the interrupt control register (P11MKn bit of P11ICn register).

(4) Interval between A/D conversion start triggers

When the A/D conversion trigger mode is set to external trigger mode or timer trigger mode, make the interval between A/D conversion start triggers longer than the A/D conversion operating time set by using bits FR2 to FR0 of the ADM1 register.

(5) Power save modes

Operation in power save modes is shown below.

- HALT mode
  Maintains all values in the ADM register and ADCR register.
- IDLE mode, STOP mode
  Since clock service stops, A/D conversion operation stops. Values in the ADM register and ADCR register are maintained, but the value in the ADCR register is undefined if IDLE mode or STOP mode is set during A/D conversion.

3.6 D/A Converter Function

The features of the V853 D/A converter function are shown below.

- 8-bit resolution D/A converter: 2 channels
- R-2R method

3.6.1 D/A converter operation

When the value to be D/A converted is written to the DACS0 or DACS1 register, an analog voltage is output via the ANOn pin. The output analog voltage is determined using the following formula.

$$ANOn = \frac{AV_{REF2} - AV_{REF3}}{256} \times DACSn + AV_{REF3}\ [V]\ (n = 1, 0)$$

D/A converter operation is set by using the DAM register.

Figure 3-77: D/A Converter Mode Register (DAM)

	7	6	5	4	3	2	1	0	Address	After reset
DAW	0	0	0	0	0	0	DACE1	DACE0	FFFFF3D0H	03H

Bit position	Bit name	Description
1, 0	DACEn (n = 1, 0)	D/A Convert Controls conversion operation of channel n (n = 1, 0).

DACEn	Channel n	ANOn pin
0	Disable operation	High impedance
1	Enable operation	Analog voltage output

The D/A conversion operation in power save modes is shown below.

- HALT mode
  Maintain DACS register and DAM register values.
- IDLE mode, STOP mode
  If the DACE bit is 1, output from the ANOn pin is continued. However, since clock supply to the D/A converter stops, new conversion tasks are not performed. DACS register and DAM register values are maintained. To reduce power consumption, set the DACS register to 0 before shifting to IDLE/STOP mode.

3.7 PWM Unit

Features of the V853 PWM unit are shown below.

- PWMn: 2 channels
- Selectable PWMn output pulse active level
- Selection of operation clock from ϕ, ϕ/2, ϕ/4, ϕ/8, and ϕ/16 (ϕ: Internal system clock)
- PWMn output resolution: Selection from 8, 9, 10, and 12 bits

Remark: n = 0, 1

PWM unit operation is set by using the PWM control register (PWMC). The PWMn operation clock is selected by using the PWM prescaler register (PWPR). Each register is shown below.

Figure 3-78: PWM Control Register (PWMC)

	7	6	5	4	3	2	1	0	Address	After reset
PWMC	PWME1	ALV1	PRM11	PRM10	PWME0	ALV0	PRM01	PRM00	FFFFF360H	00H

Bit position	Bit name	Description
7, 3	PWMEn	PWM Enable Controls PWMn operation. 0: Disable operation 1: Enable operation When PWMEn is set to "1" after being "0", the counter (TMPn) is reset and counting starts from 000H (in the 12-bit case). On the first overflow, the PWMn signal becomes active. If the operation clock (prescaler value) is made the same as the bit length of PWM0 and PWM1, the timing of the two PWMn signals becoming active can be coordinated. The counter cannot be reset by writing "1" to PWMEn while it is already "1". Write "1" again once it is set to "0".
6, 2	ALVn	Active Level Specify the active level of PWMn. 0: Active low 1: Active high
5, 4, 1, 0	PRMn1, PRMn0	Prescaler Mode Specifies the bit length of counters (TMPn) and compare registers (CMPn).

PRMn1	PRMn0	Bit length
0	0	8 bits
0	1	9 bits
1	0	10 bits
1	1	12 bits

Remark: n = 1, 0

Figure 3-79: PWM Prescaler Register (PWPR)

	7	6	5	4	3	2	1	0	Address	After reset
PWPR	0	PWP12	PWP11	PWP10	0	PWP02	PWP01	PWP00	FFFFF362H	00H

Bit position	Bit name	Description
6 to 4, 2 to 0	PWPn2, PWPn1, PWPn0	PWM Prescaler Clock Mode Selects the PWMn operation clock.

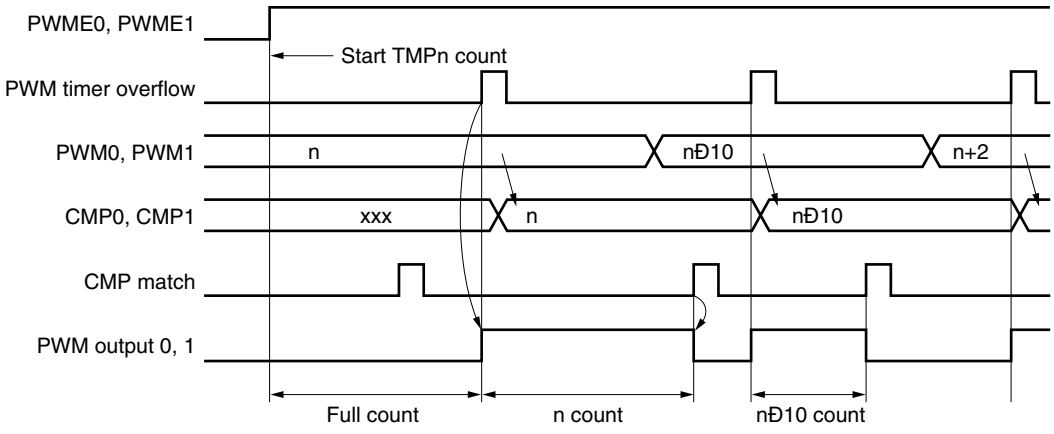
PWPn2	PWPn1	PWPn0	Operation clock
0	0	0	ϕ
0	0	1	ϕ/2
0	1	0	ϕ/4
0	1	1	ϕ/8
1	0	0	ϕ/16
Other			RFU (reserved)

Remark: n = 1, 0

3.7.1 PWM operation

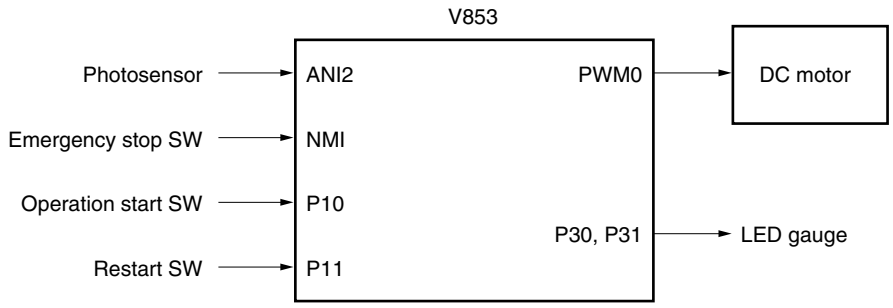
To output a PWM pulse, select an operation clock by using the PWPR register and set the number of counter clocks during the period in which PWMn output is at the active level in the PWM register. After this, setting the PWMEn bit of the PWMC register resets the internal counter (TMPn) and PWMn output becomes active level on the first counter overflow. At this time, data is transferred from the PWM register to an internal compare register (CMPn) and if CMPn matches TMPn after that, PWMn output becomes inactive (n = 1, 0).

Figure 3-80: PWM Basic Operation Timing



3.7.2 PWM function sample program

Figure 3-81: Sample Hardware Configuration



(1) Function overview

- Make the DC motor rotate using the PWM function and the value A/D input by the photosensor. Display the photosensor value in the LEDs.
- Start the DC motor using the operation start SW and stop the DC motor when emergency stop SW input is detected.

Set the PWM unit as follows.

- Select active high for PWM active level.
- Select 10 bits as the register bit length to match the A/D converter.
- Select ϕ/2 as the operation clock.

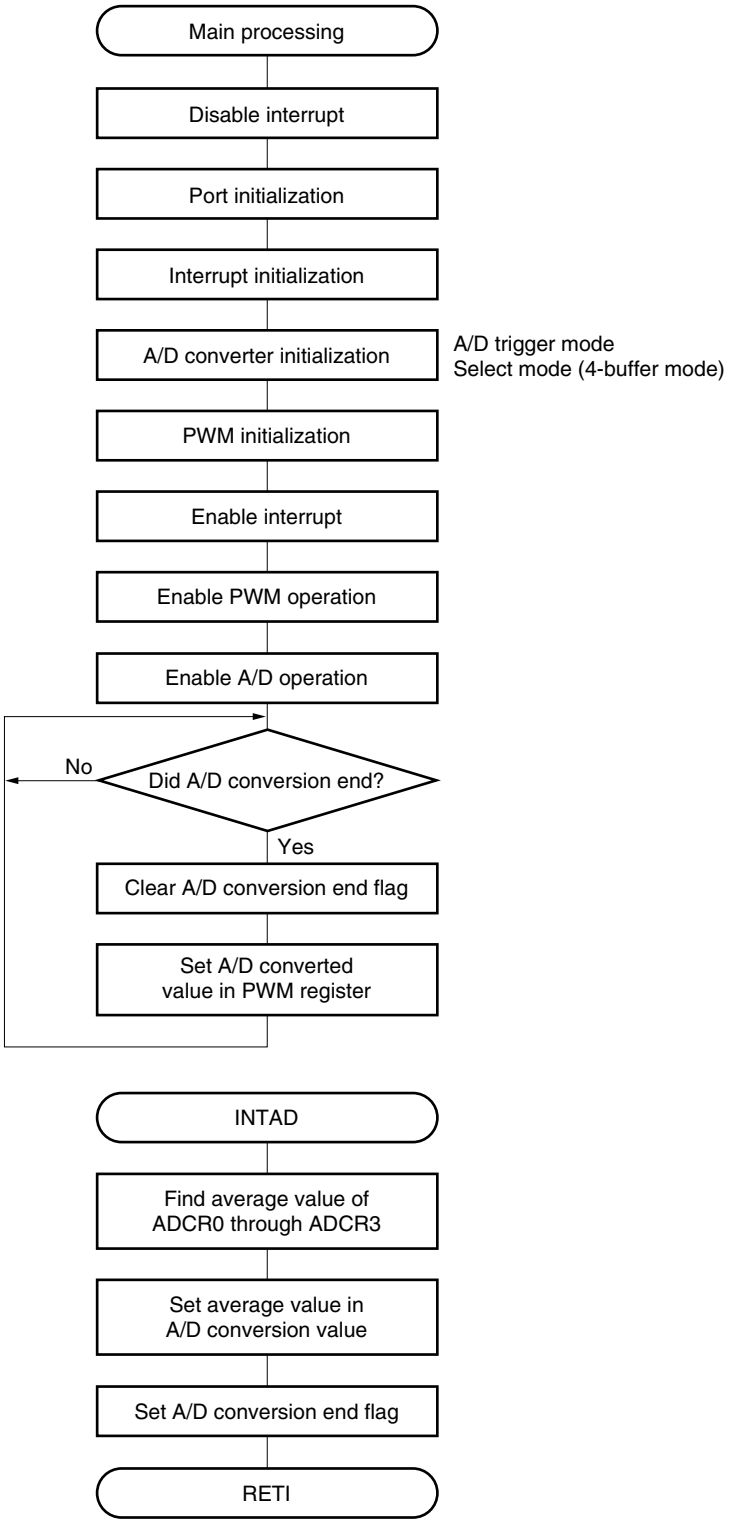
Set the PWM control register (PWMC) and PWM prescaler register (PWPR) using the above conditions.

Figure 3-82: PWMC and PWPR Settings

	7	6	5	4	3	2	1	0		
PWMC	0	0	0	0	0	1	1	0	Address FFFFF360H	Value 06H
PWPR	0	0	0	0	0	0	0	1	Address FFFFF362H	Value 01H

The flow of the sample hardware controlling the DC motor according to the photosensor is shown below.

Figure 3-83: Flow of DC Motor Rotation Control using Photosensor



(2) Sample program

```
/* ————————————————————-
 * V853 PWM unit sample program
 *
 * Initialization program for the following hardware configuration
 *   ANI2: Photosensor input
 *   PWM0: DC motor control
 *   NMI : Motor emergency stop SW
 *   P10 : Run SW (On: Run, Off: Stop)
 *   P11 : DC motor restart SW (On: Run)
 *   P30 : Running LEDs
 * ————————————————————-
 * History:
 * 1997/3/17 Ver 0.00.0
 * 1999/7/8  adapted to IAR compiler by NEC Duesseldorf AH
 * File Name: smplpwm.c
 * ————————————————————-
*/

#include <io_v850_d3003.h>
#include <intrv850.h>

/* ————————————————————-
 * Internal variable definition
 * ————————————————————-
*/
  /* Variable to store value after A/D conversion */
static short OptSensor = 0; /* Photosensor value */
  /* Operation flag */
static int ADCFlg = 0;       /* A/D conversion end flag (!0: Conversion end)
*/
static int EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop sta-
tus) */
static int StartFlg = 0;     /* Execution start flag */

/* ————————————————————
 * NMI servicing
 * DC motor emergency stop processing
 * Interrupt source: NMI
 * Arguments: None
 * Return value: None
 * ————————————————————
*/

#pragma vector=NMI_vector
__flat __interrupt void int_nmi(void)
{
   /* Process only while executing */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM operation */
   PWM0 = 0;
   PWMC_bit.no3= 0;   /* Clear bit PWME0 */
   }
```

```
/* ————————————————————
 * A/D converter conversion end
 * Interrupt source: INTAD
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
#pragma vector=INTAD_vector
__flat __interrupt void int_ad(void)
{
   long wk;
   /* Calculate average value of conversion results */
   wk = ADCR0;
   wk += ADCR1;
   wk += ADCR2;
   wk += ADCR3;
   wk /= 4;
   OptSensor = wk;

   /* Make A/D conversion effective range of photosensor 0x02A0 to 0x39F */
   if (OptSensor < 0x02A0) {
      OptSensor = 0; /* 0 if too small */
   }
   else if (OptSensor < 0x03A0) {
      OptSensor -= 0x02A0; /* Convert to 0 to 0xff */
   }
   else {
      OptSensor = 0xff; /* 0xff if too large */
   }
   /* Set A/D conversion end flag */
   ADCFlg = 1;
}

/* ————————————————————
 * Port initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PortInit(void)
{
   /* Initialize port 0 (P0)
   */
   PM0 = 0xFF;  /* All ports in input mode */
   PMC0 = 0x00; /* P00 to P07 in port mode */

   /* Initialize port 1 (P1)
   */
   PM1 = 0xFF;  /* All ports in input mode */
   PMC1 = 0x00; /* P10 to P17 in port mode */

   /* Initialize port 2 (P2)
   */
   PM2 = 0xFF;  /* All ports in input mode */
   PMC2 = 0x01; /* Use PWM0 P20 in control mode */
                /* P21 to P27 in port mode */
```

```
   /* Initialize port 3 (P3)
   */
   PM3 = 0x00;  /* All ports in output mode */
   PMC3 = 0x00; /* P30 to P37 in port mode */
   PCM = 0x00;  /* Port 1 and port 3 all in port mode */

   /* Initialize port 4 (P4)
   */
   PM4 = 0xFF; /* Use P40 to P47 as address/data bus (reference MM register)
*/

   /* Initialize port 5 (P5)
   */
   PM5 = 0xFF; /* Use P50 to P57 as address/data bus (reference MM register)
*/

   /* Initialize port 6 (P6)
   */
   PM6 = 0xFF; /* Use P60 to P63 as address/data bus (reference MM register)
*/

   /* Initialize port 9 (P9)
   */
   PM9 = 0xFF; /* Use P90 to P96 as external memory expansion control signal
                  (reference MM register) */

   /* Initialize port 11 (P11)
   */
   PM11 = 0xFF; /* All ports in input mode */
   PMC11 = 0x00; /* P110 to P117 in port mode */
}

/* ————————————————
 * Interrupt initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————
*/
void IntInit(void)
{
   /* Interrupt register settings (Set interrupt levels) */
   P11IC2 = 0x07; /* INTP112: DC motor emergency stop: Level 7 */
   ADIC = 0x07; /* INTAD: A/D conversion end : Level 7 */

   /* Interrupt effective edge settings */
   INTM0 = 0x01; /* NMI: Rising edge */
   INTM1 = 0x00; /* Not used */
   INTM2 = 0x00; /* Not used */
   INTM3 = 0x00; /* Not used */
   INTM4 = 0x00; /* Not used */
}
```

```
/* ————————————————————
* A/D converter initialization processing
* Arguments: None
* Return value: None
* ————————————————————
*/
void ADCInit(void)
{
   /* ADM0 register settings */
   ADM0 = 0x32; /* (BS) 4-buffer mode */
                /* (MS) Select mode */
                /* (ANIS2 to ANIS0) Input analog signal from ANI2 */
                /* ADM1 register settings */
   ADM1 = 0x07; /* (TRG2 to TRG0) A/D trigger mode */
                /* (FR2 to FR0) Conversion clocks = 192 */
}

/* ————————————————————
 * PWM unit initialization processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void PWMInit(void)
{
   /* PWM settings (PWMC): Set PWM0 only */
   PWMC = 0x02; /* PWME0 = 0: PWM in operation stopped state */
                /* ALV0 = 0: Active level is low */
                /* PRM01 = 1: Compare register is 10 bits */
                /* PRM00 = 0; */
   PWPR = 0x00; /* Prescaler is f/1 */
   PWM0 = 0; /* Clear buffer register */
}

/* ————————————————————
 * LED gauge display processing
 * Argument: value Value to display (16 bits)
 * Return value: None
 * ————————————————————
*/
void DispLEDGage(unsigned short value)
{
   unsigned short count;
   unsigned char leddat;

   /* LED data generation */
   for ( count = 0 , leddat = 0 ; count < 8 ; count++ ) {
      if ( value > ( count * 0x20 ) ) {
         leddat |= 1 << count;
      }
   }

   /* LED display */
   P3 = leddat;
}
```

```
/* ————————————————————
 * Operation start processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void StartProc(void)
{
   /* Start A/D conversion */
   ADM0_bit.no7= 1;

   /* Start PWM output */

   PWMC_bit.no3= 1;

   /* Enable interrupt */
   __EI();
}

/* ————————————————————
 * Operation stop processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void StopProc(void)
{
   /* Disable interrupt */
   __DI();

   /* Stop A/D conversion */
   ADM0_bit.no7= 0;

   /* Stop PWM output */
   PWM0 = 0;
   PWMC_bit.no3= 0;
}

/* ————————————————————
 * Main processing
 * Arguments: None
 * Return value: None
 * ————————————————————
*/
void main(void)
{
   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();
```

```
   /* Interrupt initialization processing */
   IntInit();

   /* A/D converter initialization processing */
   ADCInit();

   /* PWM initialization processing */
   PWMInit();

   /* Internal data initialization */
   OptSensor = 0;    /* Value of photosensor */
   ADCFlg = 0;       /* A/D conversion end flag (!0: Conversion end) */
   EmergencyFlg = 0; /* Emergency stop flag (!0: Emergency stop status) */
   StartFlg = 0;     /* Clear start flag */

   while ( 1 ) {
      /* Check for start switch ON */
      if ((P1_bit.no0 != 0) && (StartFlg == 0)) {
         StartProc();
         StartFlg = 1;
      }
      /* Do not process until start flag is ON */
      if (StartFlg == 0) {
         StopProc();
         continue;
      }
      /* Check for start switch OFF */
      if ((P1_bit.no0 == 0) && (StartFlg != 0)) {
         StopProc();
         StartFlg = 0;
         /* Also clear emergency stop flag */
         EmergencyFlg = 0;
      }
      /* During emergency stop */
      if (EmergencyFlg != 0) {
         /* Do not process until emergency stop is cancelled */
         if (P1_bit.no1 == 0) {
            continue;
         }
         /* Clear emergency stop flag */
         EmergencyFlg = 0;
         /* Start PWM operation */
         PWMC_bit.no3= 1;
      }
      /* Did A/D conversion end? */
      if (ADCFlg != 0) {
         /* Clear A/D conversion end flag */
         ADCFlg = 0;
         /* Set value of photosensor in PWM register */
         PWM0 = OptSensor * 4;
         /* Display value of photosensor in gauge LEDs */
         DispLEDGage(OptSensor);
         /* Start A/D converter conversion */
          ADM0_bit.no7= 1;
      }
   }
}
```

3.8 Clock Generation Function

The clock generator (CG) generates and controls the internal system clock (ϕ) that is supplied to the CPU and internal peripheral hardware units.

3.8.1 Input clock selection

The V853 generates a system clock using a division rate of 1, 1/2, or 5 with respect to an external oscillator or the frequency of an external clock. The division rate of the system clock is set using the clock control register (CKC). So that the clock control register (CKC) setting is not easily rewritten by, for example, a runaway program in PLL mode, the register contents can be rewritten only by using a specific instruction sequence.

★

Figure 3-84: Clock Control Register (CKC)

	7	6	5	4	3	2	1	0	Address	After reset
CKC	0	0	0	0	0	0	CKDIV1	CKDIV0	FFFFF072H	**Note**

Note μPD703003, 70F3003 : 00H
μPD703003A, 70F3003A, 703004A, 703025A, 70F3025A: 00H (if MODE = 1)
: 03H (if MODE = 0)

Bit position	Bit name	Description
1, 0	CKDIV1, CKDIV0	Clock Divide Sets the internal system clock frequency division rate for the external clock (fxx) when in PLL mode.

Operation mode	Pin	CKC Register		Internal System
	$CV_{DD}/\overline{CKSEL}$	CKDIV1	CKDIV0	Clock (ϕ)
Direct mode	L	0	0	fxx/2
	L	0	1	Setting prohibited
	L	1	0	Setting prohibited
	L	1	1	Setting prohibited
PLL mode	V_{DD}	0	0	5 ∞ fxx
	V_{DD}	0	1	Setting prohibited
	V_{DD}	1	0	fxx
	V_{DD}	1	1	fxx/2

Example: Set a division rate of 1

```
   :
movea   0x02,r0,r12     ; Transfer the value to be set in CKC register to r12
stsr    5,r10           ; Load the value of PSW into r10
mov     r10,r11         ; Transfer the loaded PSW value to r11
or      0x80,r11        ; Turn bit 7 (NP bit) of r11 ON
ldsr    r11,5           ; Set the value of r11 in PSW
st.b    r0,PRCMD[r0]    ; Write to PRCMD
st.b    r12,CKC[r0]     ; Set a value in CKC register
ldsr    r10,5           ; Return the loaded PSW value
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
   :
```

3.8.2 Power save function

The V853 power save function consists of the following three modes.

- HALT mode
- IDLE mode
- Software STOP mode

(1) HALT mode

The state of the V853 when shifted to HALT mode is shown below.

Table 3-11: Operating States in HALT Mode

Function		Operating state	
Clock generator		Operating	
Internal system clock		Operating	
CPU		Stopped	
I/O port		Maintained	
Peripheral functions	Operating		
Internal data		Internal data such as CPU registers, status, data, and internal RAM contents all maintain their states before HALT mode was set.	
In external extended mode	AD0 to AD15	High impedance[Note]	
	A16 to A19	Maintained[Note]	If $\overline{\text{HLDAK}}$ = 0, high impedance
	$\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$		
	R/$\overline{\text{W}}$	High level output[Note]	
	$\overline{\text{DSTB}}$		
	ASTB		
	$\overline{\text{HLDAK}}$	Operating	
CLKOUT		Clock output (if clock output is not inhibited)	

Note: Even after HALT instruction execution, the instruction fetch operation continues until the internal instruction prefetch queue becomes full. After it becomes full, operation stops in the state shown in Table 3-11.

(2) IDLE mode

The following table shows the state of the V853 when shifted to IDLE mode.

Table 3-12: Operating States in IDLE Mode

Function		Operating state
Clock generator		Operating
Internal system clock		Stopped
CPU		Stopped
I/O port		Maintained
Peripheral functions		Stopped
Internal data		Internal data such as CPU registers, status, data, and internal RAM contents all maintain their states before IDLE mode was set.
In external extended mode	AD0 to AD15	High impedance
	A16 to A19	
	$\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$	
	R/$\overline{\text{W}}$	
	$\overline{\text{DSTB}}$	
	ASTB	
	$\overline{\text{HLDAK}}$	
CLKOUT		Low-level output

(3) Software STOP mode

The following table shows the state of the V853 when shifted to software STOP mode.

Table 3-13: Operating States in Software STOP Mode

Function		Operating state
Clock generator		Stopped
Internal system clock		Stopped
CPU		Stopped
I/O port[Note]		Maintained
Peripheral functions		Stopped
Internal data[Note]		Internal data such as CPU registers, status, data, and internal RAM contents all maintain their states before software STOP mode was set.
In external extended mode	AD0 to AD15	High impedance
	A16 to A19	
	$\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$	
	R/$\overline{\text{W}}$	
	$\overline{\text{DSTB}}$	
	ASTB	
	$\overline{\text{HLDAK}}$	
CLKOUT		Low-level output

Note: When the value of V_{DD} is within the operating range. Even if it is below the minimum operating voltage, the contents of internal RAM are maintained if the data maintenance voltage (V_{DDDR}) is maintained.

The HALT instruction is used to shift to HALT mode. For IDLE mode or software STOP mode, setting a value in the power save control register (PSC) shifts to the appropriate power save mode.
Like the clock control register (CKC), the power save control register (PSC) also can only have its contents rewritten using a specific sequence. An example of shifting to software STOP mode is shown below.

Example Shifting to software STOP mode

```
   :
movea   0xC2,r0,r12     ; Transfer the value to be set in PSC register to r12
stsr    5,r10           ; Load the value of PSW into r10
mov     r10,r11         ; Transfer the loaded PSW value to r11
or      0x80,r11        ; Turn bit 7 (NP bit) of r11 ON
ldsr    r11,5           ; Set value of r11 in PSW
st.b    r0,PRCMD[r0]    ; Write to PRCMD
st.b    r12,PSC[r0]     ; Set a value in PSC register
ldsr    r10,5           ; Return the loaded PSW value
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
nop                     ; Dummy instruction
   :
```

The power save control register (PSC) is shown below.

Figure 3-85: Power Save Control Register (PSC)

	7	6	5	4	3	2	1	0	Address	After reset
PSC	DCLK1	DCLK0	TBCS	CESEL	0	IDLE	STP	0	FFFFF070H	00H (in ROM-less mode) C0H (in single-chip mode)

Bit position	Bit name	Description
7, 6	DCLK1, DCLK0	Disable CLKOUT Specifies the operating mode of the CLKOUT pin. DCLK1 / DCLK0 / Mode: 0 / 0 / Normal output mode 0 / 1 / RFU (reserved) 1 / 0 / RFU (reserved) 1 / 1 / Clock output inhibit mode
5	TBCS	Time Base Count Select Selects the time base counter clock. 0: $fxx/2^8$ 1: $fxx/2^9$ For details, refer to **V853 User's Manual Hardware**.
4	CESEL	Crystal/External Select Specifies the function of the X1 and X2 pins. 0: Connect oscillator to pins X1 and X2 1: Connect external clock to pin X1 Setting CESEL = 1 cuts off the feedback loop of the oscillation circuit to prevent a current leak in STOP mode. In addition, the oscillation stabilization time is not counted by the time base counter (TBC) after STOP mode is released.
2	IDLE	IDLE Mode Specifies IDLE mode. IDLE state is entered by writing "1". This is automatically reset to "0" when IDLE mode is released.
1	STP	STOP Mode Specifies software STOP mode. STOP state is entered by writing "1". This is automatically reset to "0" when STOP mode is released.

How to release each power save mode is shown below.

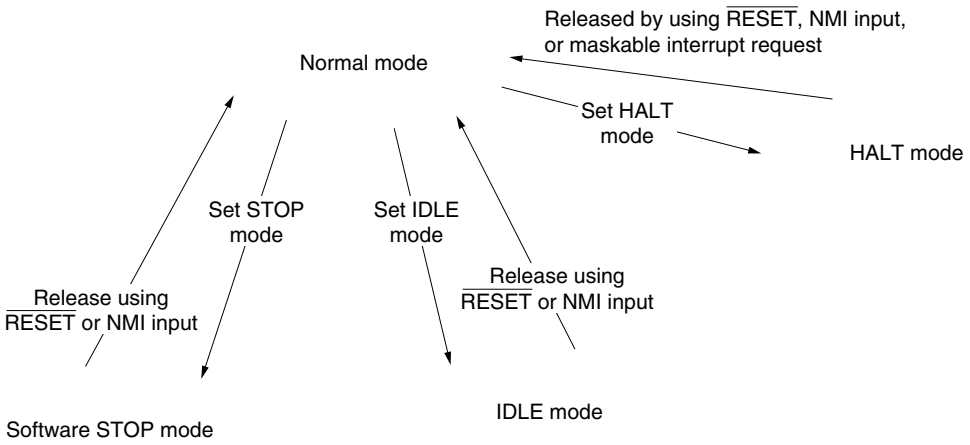
(1) How to release HALT mode
- Input to RESET pin
- NMI request
- Non-masked maskable interrupt request

(2) How to release IDLE mode
- Input to RESET pin
- NMI request

(3) How to release software STOP mode
- Input to RESET pin
- NMI request

A state transition diagram for each mode is shown below.

Figure 3-86: State Transitions for Each Mode



3.9 Reset Function

The initial value of each register after reset is shown below.

Table 3-14: Initial Value of Each Register After Reset (1/2)

Register		Initial value after reset
r0		00000000H
r1 to r31		Undefined
PC		00000000H
PSW		00000020H
EIPC		Undefined
EIPSW		Undefined
FEPC		Undefined
FEPSW		Undefined
ECR		00000000H
Internal RAM		Undefined
Ports	Input/output latch (P0 to P6, P9, P11)	Undefined
	Input latch (P7)	Undefined
	Mode register (PM0 to PM5, PM11)	FFH
	Mode register (PM6)	xFH
	Mode register (PM9)	x1111111B
	Mode control register (PMC0 to PMC3, PMC11)	00H
	Memory expansion mode register (MM)	07H/00H
	Port control mode register (PCM)	00H
	Pull-up resistor option register (PUO)	00H
Clock generator	Clock control register (CKC)	00H/03H
	System status register (SYS)	0000000xB
Real-time pulse unit	Timer unit mode register (TUM11 to TUM14)	0000H
	Timer control register (TMC11 to TMC14, TMC4)	00H
	Timer output control register 1 (TOC11 to TOC14)	00H
	Timer (TM11 to TM14, TM4)	0000H
	Capture/compare register (CC110 to CC113, CC120 to CC123, CC130 to CC133, CC140 to CC143)	Undefined
	Compare register 4 (CM4)	Undefined
	Timer overflow status register (TOVS)	00H
A/D converter	A/D converter mode register (ADM0)	00H
	A/D converter mode register (ADM1)	07H
	A/D conversion result register (ADCR0 to ADCR7, ADCR0H to ADCR7H)	Undefined
D/A converter	D/A converter conversion value setting register (DACS0, DACS1)	00H
	D/A converter mode register (DAM)	03H

Remark: x: Undefined

Table 3-14: Initial Value of Each Register After Reset (2/2)

Register		Initial value after reset
Serial interface	Asynchronous serial interface mode register (ASIM00, ASIM10)	00H
	Asynchronous serial interface mode register (ASIM01, ASIM11)	00H
	Asynchronous serial interface status register (ASIS0, ASIS1)	00H
	Receive buffer (RXB0, RXB0L, RXB1, RXB1L)	Undefined
	Transmit shift register (TXS0, TXS0L, TXS1, TXS1L)	Undefined
	Clock synchronous serial interface mode register 0 (CSIM0 to CSIM3)	00H
	Serial I/O shift register (SIO0 to SIO3)	Undefined
	Baud rate generator register (BRGC0 to BRGC2)	Undefined
	Baud rate generator prescaler mode register (BPRM0 to BPRM2)	00H
PWM	PWM control register (PWMC)	00H
	PWM prescaler register (PWPR)	00H
	PWM buffer register (PWM0, PWM0L, PWM1, PWM1L)	Undefined
Interrupt/exception handling function	Interrupt control register (XXICn)	47H
	In-service priority register (ISPR)	00H
	External interrupt mode register (INTM0 to INTM4)	00H
Memory management function	Data wait control register (DWC)	FFFFH
	Bus cycle control register (BCC)	AAAAH
Power save control	Command register (PRCMD)	Undefined
	Power save control register (PSC)	00H/C0H

[MEMO]

APPENDIX A. LIST OF INTERRUPT CONTROL REGISTERS

Type	Class	Interrupt/exception source				Default priority	Exception code	Handler address	Restored PC
		Name	Control register	Cause of occurrence	Generating unit				
Reset	Interrupt	RESET	–	Reset input	–	–	0000H	00000000H	Undefined
Non-maskable next PC		Interrupt	NMI	–	NMI input	–	–	0010H	00000010H
Software exception	Exception	TRAP0n(n = 0 to FH)	–	TRAP instruction	–	–	004nH	00000040H	
		TRAP1n (n = 0 to FH)	–	TRAP instruction	–	–	005nH	00000050H	
Exception trap	Exception	ILGOP	–	Illegal op code	–	–	0060H	00000060H	
Maskable	Interrupt	INTOV11	OVIC11	Timer 11 overflow	RPU	0	0080H	00000080H	
		INTOV12	OVIC12	Timer 12 overflow	RPU	1	0090H	00000090H	
		INTOV13	OVIC13	Timer 13 overflow	RPU	2	00A0H	000000A0H	
		INTOV14	OVIC14	Timer 14 overflow	RPU	3	00B0H	000000B0H	
		INTP110/INTCC110	P11IC0	INTP110/CC110 match	Pin/RPU	4	00C0H	000000C0H	
		INTP111/INTCC111	P11IC1	INTP111/CC111 match	Pin/RPU	5	00D0H	000000D0H	
		INTP112/INTCC112	P11IC2	INTP112/CC112 match	Pin/RPU	6	00E0H	000000E0H	
		INTP113/INTCC113	P11IC3	INTP113/CC113 match	Pin/RPU	7	00F0H	000000F0H	
		INTP120/INTCC120	P12IC0	INTP120/CC120 match	Pin/RPU	8	0100H	00000100H	
		INTP121/INTCC121	P12IC1	INTP121/CC121 match	Pin/RPU	9	0110H	00000110H	
		INTP122/INTCC122	P12IC2	INTP122/CC122 match	Pin/RPU	10	0120H	00000120H	
		INTP123/INTCC123	P12IC3	INTP123/CC123 match	Pin/RPU	11	0130H	00000130H	
		INTP130/INTCC130	P13IC0	INTP130/CC130 match	Pin/RPU	12	0140H	00000140H	
		INTP131/INTCC131	P13IC1	INTP131/CC131 match	Pin/RPU	13	0150H	00000150H	
		INTP132/INTCC132	P13IC2	INTP132/CC132 match	Pin/RPU	14	0160H	00000160H	
		INTP133/INTCC133	P13IC3	INTP133/CC133 match	Pin/RPU	15	0170H	00000170H	
		INTP140/INTCC140	P14IC0	INTP140/CC140 match	Pin/RPU	16	0180H	00000180H	
		INTP141/INTCC141	P14IC1	INTP141/CC141 match	Pin/RPU	17	0190H	00000190H	
		INTP142/INTCC142	P14IC2	INTP142/CC142 match	Pin/RPU	18	01A0H	000001A0H	
		INTP143/INTCC143	P14IC3	INTP143/CC143 match	Pin/RPU	19	01B0H	000001B0H	
		INTCM4	CMIC4	CM4 match signal	RPU	20	01C0H	000001C0H	
		INTCSI0	CSIC0	CSI0 transmit/receive completion	SIO	21	01D0H	000001D0H	
		INTCSI1	CSIC1	CSI1 transmit/receive completion	SIO	22	01E0H	000001E0H	
		INTCSI2	CSIC2	CSI2 transmit/receive completion	SIO	23	01F0H	000001F0H	
		INTCSI3	CSIC3	CSI3 transmit/receive completion	SIO	24	0200H	00000200H	
		INTSER0	SEIC0	UART0 receive error	SIO	25	0210H	00000210H	
		INTSR0	SRIC0	UART0 receive completion	SIO	26	0220H	00000220H	
		INTST0	STIC0	UART0 transmit completion	SIO	27	0230H	00000230H	
		INTSER1	SEIC1	UART1 receive error	SIO	28	0240H	00000240H	
		INTSR1	SRIC1	UART1 receive completion	SIO	29	0250H	00000250H	
		INTST1	STIC1	UART1 transmit completion	SIO	30	0260H	00000260H	
		INTAD	ADIC	A/D conversion end	ADC	31	0270H	00000270H	

Remarks 1. Default priority: The priority used for prioritizing when multiple maskable interrupt requests of the same priority level occur simultaneously. 0 is the highest priority.

Restored PC: The value of the PC saved in the EIPC or FEPC at the start of interrupt/exception service. If an interrupt is accepted during execution of a DIVH (division) instruction, the saved restored PC value is the PC value of the current instruction (DIVH).

2. At the time of an illegal op code exception, the execution address of the illegal instruction is given by restored PC – 4.

APPENDIX B. LIST OF PERIPHERAL I/O CONTROL REGISTERS

(1/4)

Address	Function register name	Symbol	R/W	Manipulatable bits			After reset
				1 bit	8 bits	16 bits	
FFFFF000H	Port 0	P0	R/W				Undefined
FFFFF002H	Port 1	P1					
FFFFF004H	Port 2	P2					
FFFFF006H	Port 3	P3					
FFFFF008H	Port 4	P4					
FFFFF00AH	Port 5	P5					
FFFFF00CH	Port 6	P6					
FFFFF00EH	Port 7	P7	R				
FFFFF012H	Port 9	P9	R/W				
FFFFF016H	Port 11	P11					
FFFFF020H	Port 0 mode register	PM0					FFH
FFFFF022H	Port 1 mode register	PM1					
FFFFF024H	Port 2 mode register	PM2					
FFFFF026H	Port 3 mode register	PM3					
FFFFF028H	Port 4 mode register	PM4					
FFFFF02AH	Port 5 mode register	PM5					
FFFFF02CH	Port 6 mode register	PM6					xFH
FFFFF032H	Port 9 mode register	PM9					x1111111B
FFFFF036H	Port 11 mode register	PM11					FFH
FFFFF040H	Port 0 mode control register	PMC0					00H
FFFFF042H	Port 1 mode control register	PMC1					
FFFFF044H	Port 2 mode control register	PMC2					
FFFFF046H	Port 3 mode control register	PMC3					
FFFFF04CH	Memory expansion mode register	MM					07H/00H
FFFFF056H	Port 11 mode control register	PMC11					00H
FFFFF05CH	Port control mode register	PCM					
FFFFF05EH	Pull-up resistor option register	PUO					
FFFFF060H	Data wait control register	DWC					FFFFH
FFFFF062H	Bus cycle control register	BCC					AAAAH
FFFFF070H	Power save control register	PSC					00H/C0H
FFFFF072H	Clock control register	CKC					**Note**
FFFFF078H	System status register	SYS					0000000xB
FFFFF084H	Baud rate generator compare register 0	BRGC0					Undefined
FFFFF086H	Baud rate generator prescaler mode register 0	BPRM0					00H
FFFFF088H	Clock synchronous serial interface mode register 0	CSIM0					
FFFFF08AH	Serial I/O shift register 0	SIO0					Undefined

Note μPD703003, 70F3003: 00H

★ μPD703003A, 70F3003A, 703004A, 703025A, 70F3025A: 00H/03H

(2/4)

Address	Function register name	Symbol	R/W	Manipulatable bits			After reset
				1 bit	8 bits	16 bits	
FFFFF094H	Baud rate generator compare register 1	BRGC1	R/W				Undefined
FFFFF096H	Baud rate generator prescaler mode register 1	BPRM1					00H
FFFFF098H	Clock synchronous serial interface mode register 1	CSIM1					
FFFFF09AH	Serial I/O shift register 1	SIO1					Undefined
FFFFF0A4H	Baud rate generator compare register 2	BRGC2					
FFFFF0A6H	Baud rate generator prescaler mode register 2	BPRM2					00H
FFFFF0A8H	Clock synchronous serial interface mode register 2	CSIM2					
FFFFF0AAH	Serial I/O shift register 2	SIO2					Undefined
FFFFF0B8H	Clock synchronous serial interface mode register 3	CSIM3					00H
FFFFF0BAH	Serial I/O shift register 3	SIO3					Undefined
FFFFF0C0H	Asynchronous serial interface mode register 00	ASIM00					00H
FFFFF0C2H	Asynchronous serial interface mode register 01	ASIM01					
FFFFF0C4H	Asynchronous serial interface status register 0	ASIS0	R				
FFFFF0C8H	Receive buffer 0 (9 bits)	RXB0					Undefined
FFFFF0CAH	Receive buffer 0L (Low-order 8 bits)	RXB0L					
FFFFF0CCH	Transmit shift register 0 (9 bits)	TXS0	W				
FFFFF0CEH	Transmit shift register 0L (Low-order 8 bits)	TXS0L					
FFFFF0D0H	Asynchronous serial interface mode register 10	ASIM10	R/W				00H
FFFFF0D2H	Asynchronous serial interface mode register 11	ASIM11					
FFFFF0D4H	Asynchronous serial interface status register 1	ASIS1	R				
FFFFF0D8H	Receive buffer 1 (9 bits)	RXB1					Undefined
FFFFF0DAH	Receive buffer 1L (Low-order 8 bits)	RXB1L					
FFFFF0DCH	Transmit shift register 1 (9 bits)	TXS1	W				
FFFFF0DEH	Transmit shift register 1L (Low-order 8 bits)	TXS1L					
FFFFF100H	Interrupt control register	OVIC11	R/W				47H
FFFFF102H	Interrupt control register	OVIC12					
FFFFF104H	Interrupt control register	OVIC13					
FFFFF106H	Interrupt control register	OVIC14					
FFFFF108H	Interrupt control register	P11IC0					
FFFFF10AH	Interrupt control register	P11IC1					
FFFFF10CH	Interrupt control register	P11IC2					
FFFFF10EH	Interrupt control register	P11IC3					
FFFFF110H	Interrupt control register	P12IC0					
FFFFF112H	Interrupt control register	P12IC1					
FFFFF114H	Interrupt control register	P12IC2					
FFFFF116H	Interrupt control register	P12IC3					
FFFFF118H	Interrupt control register	P13IC0					
FFFFF11AH	Interrupt control register	P13IC1					
FFFFF11CH	Interrupt control register	P13IC2					
FFFFF11EH	Interrupt control register	P13IC3					
FFFFF120H	Interrupt control register	P14IC0					
FFFFF122H	Interrupt control register	P14IC1					

(3/4)

Address	Function register name	Symbol	R/W	Manipulatable bits			After reset
				1 bit	8 bits	16 bits	
FFFFF124H	Interrupt control register	P14IC2	R/W				47H
FFFFF126H	Interrupt control register	P14IC3					
FFFFF128H	Interrupt control register	CMIC4					
FFFFF12AH	Interrupt control register	CSIC0					
FFFFF12CH	Interrupt control register	CSIC1					
FFFFF12EH	Interrupt control register	CSIC2					
FFFFF130H	Interrupt control register	CSIC3					
FFFFF132H	Interrupt control register	SEIC0					
FFFFF134H	Interrupt control register	SRIC0					
FFFFF136H	Interrupt control register	STIC0					
FFFFF138H	Interrupt control register	SEIC1					
FFFFF13AH	Interrupt control register	SRIC1					
FFFFF13CH	Interrupt control register	STIC1					
FFFFF13EH	Interrupt control register	ADIC					
FFFFF166H	In-service priority register	ISPR	R				00H
FFFFF170H	Command register	PRCMD	W				xxH
FFFFF180H	External interrupt mode register 0	INTM0	R/W				00H
FFFFF182H	External interrupt mode register 1	INTM1					
FFFFF184H	External interrupt mode register 2	INTM2					
FFFFF186H	External interrupt mode register 3	INTM3					
FFFFF188H	External interrupt mode register 4	INTM4					
FFFFF230H	Timer overflow status register	TOVS					
FFFFF240H	Timer unit mode register 11	TUM11					0000H
FFFFF242H	Timer control register 11	TMC11					00H
FFFFF244H	Timer output control register 11	TOC11					
FFFFF250H	Timer 11	TM11	R				0000H
FFFFF252H	Capture/compare register 110	CC110	R/W				Undefined
FFFFF254H	Capture/compare register 111	CC111					
FFFFF256H	Capture/compare register 112	CC112					
FFFFF258H	Capture/compare register 113	CC113					
FFFFF260H	Timer unit mode register 12	TUM12					0000H
FFFFF262H	Timer control register 12	TMC12					00H
FFFFF264H	Timer output control register 12	TOC12					
FFFFF270H	Timer 12	TM12	R				0000H
FFFFF272H	Capture/compare register 120	CC120	R/W				Undefined
FFFFF274H	Capture/compare register 121	CC121					
FFFFF276H	Capture/compare register 122	CC122					
FFFFF278H	Capture/compare register 123	CC123					
FFFFF280H	Timer unit mode register 13	TUM13					0000H
FFFFF282H	Timer control register 13	TMC13					00H
FFFFF284H	Timer output control register 13	TOC13					
FFFFF290H	Timer 13	TM13	R				0000H

(4/4)

Address	Function register name	Symbol	R/W	Manipulatable bits			After reset
				1 bit	8 bits	16 bits	
FFFFF292H	Capture/compare register 130	CC130	R/W				Undefined
FFFFF294H	Capture/compare register 131	CC131					
FFFFF296H	Capture/compare register 132	CC132					
FFFFF298H	Capture/compare register 133	CC133					
FFFFF2A0H	Timer unit mode register 14	TUM14					0000H
FFFFF2A2H	Timer control register 14	TMC14					00H
FFFFF2A4H	Timer output control register 14	TOC14					
FFFFF2B0H	Timer 14	TM14	R				0000H
FFFFF2B2H	Capture/compare register 140	CC140	R/W				Undefined
FFFFF2B4H	Capture/compare register 141	CC141					
FFFFF2B6H	Capture/compare register 142	CC142					
FFFFF2B8H	Capture/compare register 143	CC143					
FFFFF342H	Timer control register 4	TMC4					00H
FFFFF350H	Timer 4	TM4	R				0000H
FFFFF352H	Compare register 4	CM4	R/W				Undefined
FFFFF360H	PWM control register	PWMC					00H
FFFFF362H	PWM prescaler register	PWPR					
FFFFF364H	PWM buffer register 0 (12 bits)	PWM0					Undefined
FFFFF366H	PWM buffer register 0L (Low-order 8 bits)	PWM0L					
FFFFF368H	PWM buffer register 1 (12 bits)	PWM1					
FFFFF36AH	PWM buffer register 1L (Low-order 8 bits)	PWM1L					
FFFFF380H	A/D converter mode register 0	ADM0					00H
FFFFF382H	A/D converter mode register 1	ADM1					07H
FFFFF390H	A/D conversion result register 0	ADCR0	R				Undefined
FFFFF392H	A/D conversion result register 0H	ADCR0H					
FFFFF394H	A/D conversion result register 1	ADCR1					
FFFFF396H	A/D conversion result register 1H	ADCR1H					
FFFFF398H	A/D conversion result register 2	ADCR2					
FFFFF39AH	A/D conversion result register 2H	ADCR2H					
FFFFF39CH	A/D conversion result register 3	ADCR3					
FFFFF39EH	A/D conversion result register 3H	ADCR3H					
FFFFF3A0H	A/D conversion result register 4	ADCR4					
FFFFF3A2H	A/D conversion result register 4H	ADCR4H					
FFFFF3A4H	A/D conversion result register 5	ADCR5					
FFFFF3A6H	A/D conversion result register 5H	ADCR5H					
FFFFF3A8H	A/D conversion result register 6	ADCR6					
FFFFF3AAH	A/D conversion result register 6H	ADCR6H					
FFFFF3ACH	A/D conversion result register 7	ADCR7					
FFFFF3AEH	A/D conversion result register 7H	ADCR7H					
FFFFF3C0H	D/A converter conversion value setting register 0	DACS0	R/W				00H
FFFFF3C2H	D/A converter conversion value setting register 1	DACS1					
FFFFF3D0H	D/A converter mode register	DAM					03H

APPENDIX C. INDEX

NEC

Facsimile Message

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	☐	☐	☐	☐
Technical Accuracy	☐	☐	☐	☐
Organization	☐	☐	☐	☐

CS 99.1

[MEMO]