To our customers,

## Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

# Notice

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

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

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

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

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

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

RENESAS

# Application Note

# V853™

## 32-Bit Single-Chip Microcontrollers

## Software

---

## μPD703003A
## μPD703003A(A)
## μPD703004A
## μPD703025A
## μPD703025A(A)
## μPD70F3003A
## μPD70F3003A(A)
## μPD70F3025A

Document No. U12784EJ3V0AN00 (3rd edition)
Date Published April 2002 N CP(K)

**[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.

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: μPD70F3003A, 70F3003A(A), 70F3025A

The customer must judge the need for license: μPD703003A, 703003A(A), 703004A, 703025A, 703025A(A)

- **The information in this document is current as of December, 2001. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.
  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.
  (Note)
  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.
  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

# 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 do Brasil S.A.**
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

**NEC Electronics (Europe) GmbH**
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

• Branch The Netherlands
Eindhoven, The Netherlands
Tel: 040-244 58 45
Fax: 040-244 45 80

• Branch Sweden
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

• Filiale Italiana
Milano, Italy
Tel: 02-667541
Fax: 02-66754299

**NEC Electronics (France) S.A.**
Vélizy-Villacoublay, France
Tel: 01-3067-58-00
Fax: 01-3067-58-99

**NEC Electronics (France) S.A.**
**Representación en España**
Madrid, Spain
Tel: 091-504-27-87
Fax: 091-504-28-60

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

**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 Shanghai, Ltd.**
Shanghai, P.R. China
Tel: 021-6841-1138
Fax: 021-6841-1137

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

**NEC Electronics Singapore Pte. Ltd.**
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

J02.3-1

## Major Revisions in This Edition

| Page | Description |
|---|---|
| Throughout | • Deletion of following products from target devices<br>μPD703003, 70F3003<br>• Addition of following products to target devices<br>μPD703003A(A), 703025A(A), 70F3003A(A)<br>• Deletion of description on ROMless mode |
| p.18 | Modification of description in **1.3 Register Set** |
| p.20 | Deletion of description in **1.5 Memory Map** |
| p.20 | Addition of **Note** in **Figure 1-3 Memory Map** |
| p.22 | Modification of description in **Table 2-1 Register Set Used by CA850** |
| p.89 | Addition of **Note** in **Table 3-7 Features of Asynchronous Serial Interface** |
| p.109 | Addition of description to **3.4.3 Clocked serial interface (CSI0 to CSI3)** |
| p.135 | Addition of **Caution** in **Figure 3-59 A/D Converter Mode Register 0 (ADM0)** |
| p.178 | Deletion of description in **Figure 3-84 Clock Control Register (CKC)** |
| p.181 | Addition of **Caution** in **Example** under **3.8.2 (3) Software STOP mode** |
| p.182 | Deletion of description in **Figure 3-85 Power Save Control Register (PSC)** |
| p.184 | Addition of **3.8.3 Cautions on power save function** |
| p.185 | Deletion of description in **Table 3-14 Initial Value of Each Register After Reset (1/2)** |
| p.186 | Deletion of description and addition of **Caution** in **Table 3-14 Initial Value of Each Register After Reset (2/2)** |
| p.222 | Addition of **Caution** to **APPENDIX A LIST OF INTERRUPT CONTROL REGISTERS** |
| p.223 | Modification of description in **APPENDIX B LIST OF PERIPHERAL I/O CONTROL REGISTERS** |

**The mark ★ shows major revised points.**

# INTRODUCTION

**Target Reader**

This manual is intended for users who wish to understand the functions of the V853 and design application systems using the V853.
The target devices are as follows.

- Standard version: μPD703003A, 703004A, 703025A, 70F3003A, 70F3025A
- Special version: μPD703003A(A), 703025A(A), 70F3003A(A)

**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
- Overview of the CA850
- Functions of the V853
- Application programs that operate on the TU-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.

**Cautions 1. The examples in this manual are for products used in general electrical equipment with a "standard" quality grade. If examples in this manual are used for applications requiring a "special" quality grade, check the quality grade for the parts and circuits actually used before use.**

**2. If this manual is used as for special products, read as follows.**

**μPD703003A → μPD703003A(A)**
**μPD703025A → μPD703025A(A)**
**μPD70F3003A → μPD70F3003A(A)**

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 |
| representation | /xxx ("/" before signal name) |
| Memory map address: | Higher address on the top and lower address on the bottom |
| **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

Prefix indicating the power 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 Name | Document No. |
|---|---|
| V850 Series Architecture User's Manual | U10243E |
| $\mu$PD703003A, 703004A, 703025A, 703003A(A), 703025A(A) Data Sheet | U13188E |
| $\mu$PD70F3003A, 70F3025A, 70F3003A(A) Data Sheet | U13189E |
| V853 Hardware User's Manual | U10913E |
| V853 Hardware Application Note | U12619E |
| V853 Software Application Note | This manual |

**Documents related to development tools (User's manuals)**

| Document Name | | Document No. |
|---|---|---|
| IE-703002-MC (In-Circuit Emulator) | | U11595E |
| IE-703003-MC-EM1 (In-Circuit Emulator Option Board) | | U11596E |
| CA850 (Ver. 2.30 or Later) (C Compiler Package) | Operation | U14568E |
| | C Language | U14566E |
| | Project Manager | U14569E |
| | Assembly Language | U14567E |
| CA850 (Ver. 2.40 or Later) (C Compiler Package) | Operation | U15024E |
| | C Language | U15025E |
| | Project Manager | U15026E |
| | Assembly Language | U15027E |
| ID850 (Ver. 2.40 or Later) (Integrated Debugger) | Operation Windows™ Based | U15181E |
| SM850 (Ver. 2.00 or Later) (System Simulator) | Operation Windows Based | U15182E |
| | External Part User Open Interface Specifications | U14873E |
| RX850 (Ver. 3.13 or Later) (Real-Time OS) | Basics | U13430E |
| | Installation | U13410E |
| | Technical | U13431E |
| RX850 Pro (Ver. 3.13) (Real-Time OS) | Fundamental | U13773E |
| | Installation | U13774E |
| | Technical | U13772E |
| RD850 (Ver. 3.01) (Task Debugger) | | U13737E |
| RD850 Pro (Ver. 3.01) (Task Debugger) | | U13916E |
| AZ850 (Ver. 3.0) (System Performance Analyzer) | | U14410E |
| PG-FP3 (Flash Memory Programmer) | | U13502E |

**CONTENTS**

## LIST OF FIGURES (1/3)

## LIST OF FIGURES (2/3)

## LIST OF FIGURES (3/3)

## LIST OF TABLE

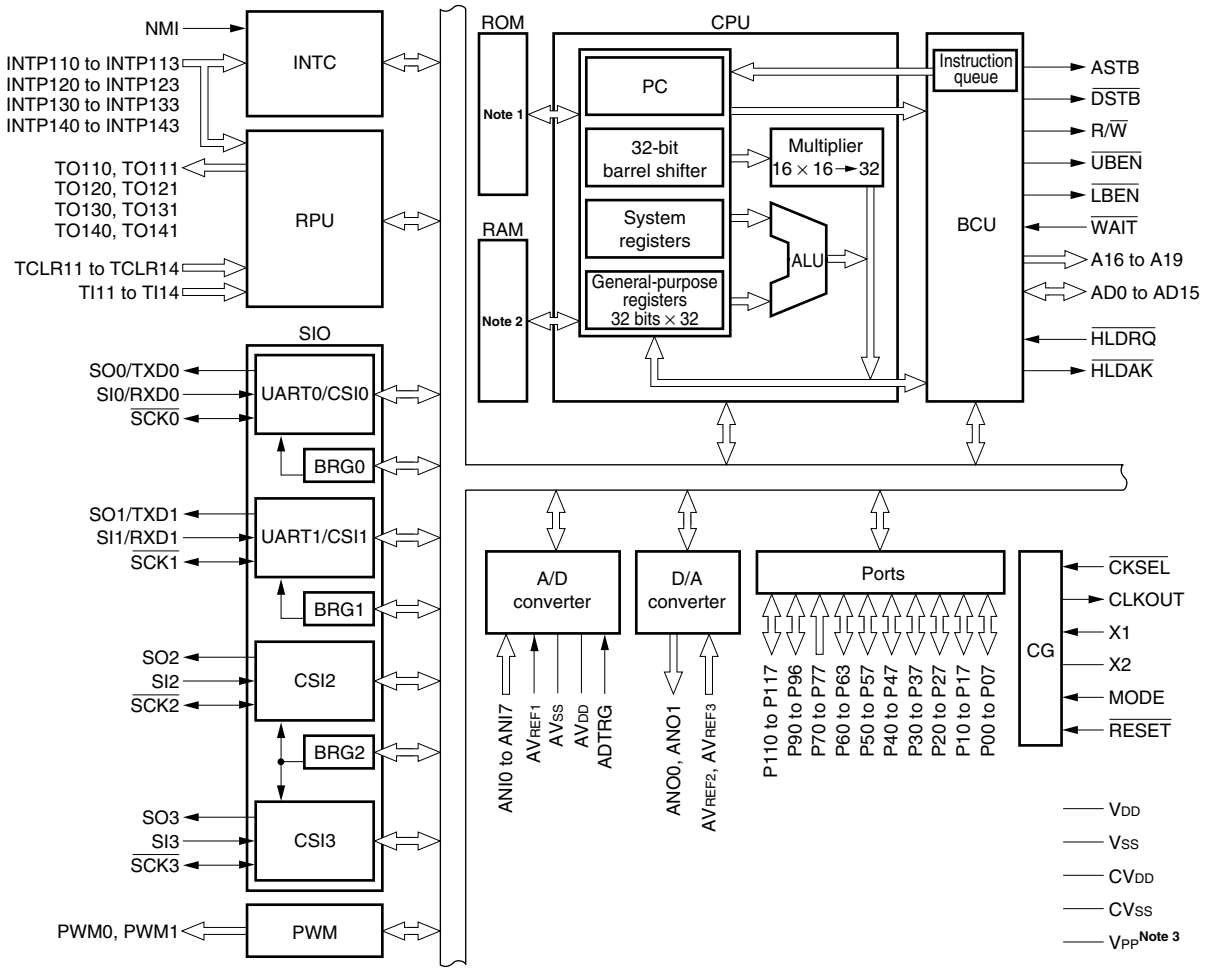
# CHAPTER 1 OVERVIEW OF V853

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.** μPD703003A: 128 KB (mask ROM)
μPD70F3003A: 128 KB (flash memory)
μPD703004A: 96 KB (mask ROM)
μPD703025A: 256 KB (mask ROM)
μPD70F3025A: 256 KB (flash memory)

**2.** μPD703003A, 703004A, 70F3003A: 4 KB
μPD703025A, 70F3025A: 8 KB

**3.** μPD70F3003A, 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 KB (μPD70F3025A)<br>Mask ROM: 256 KB (μPD703025A)<br>Flash memory: 128 KB (μPD70F3003A)<br>Mask ROM: 128 KB (μPD703003A)<br>Mask ROM: 96 KB (μPD703004A) |
| | Internal RAM | 8 KB (μPD703025A, 70F3025A)<br>4 KB (μPD703003A, 703004A, 70F3003A) |
| Interrupts | | External interrupts: 17 (including NMI)<br>Internal interrupts: 32 sources<br>Exceptions: 1 source<br>Programmable priorities: 8 levels |
| Timer/counter | | 16-bit timer/event counter: 4 ch<br>16-bit interval timer: 1 ch |
| Serial interface | | Asynchronous serial interface (UART)<br>Clocked serial interface (CSI)<br>UART/CSI: 2 ch<br>CSI: 2 ch<br>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/Software 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 | Address/data variable registers (when r2 is not used by the real-time OS to be used) | |
| 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**Note** |
| r6 to r29 | Address/data variable registers | |
| 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 |

**Note** Area where program code is allocated.

Thirty-two general-purpose registers, r0 to r31, are available. Any of these registers can be used as a data variable or address variable.

However, r0 and r30 are implicitly used by instructions, and care must be exercised when using these registers. Also, r1, r3 to r5 and r31 are implicitly used by the assembler and C compiler. Therefore, before using these registers, their contents must be saved so that they are not lost. The contents must be restored to the registers after the registers have been used. In some case, r2 is used by the real-time OS. Consequently, r2 can be used as a variable register only when r2 is not used by the real-time OS to be used.

Bits 31 to 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 to 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**

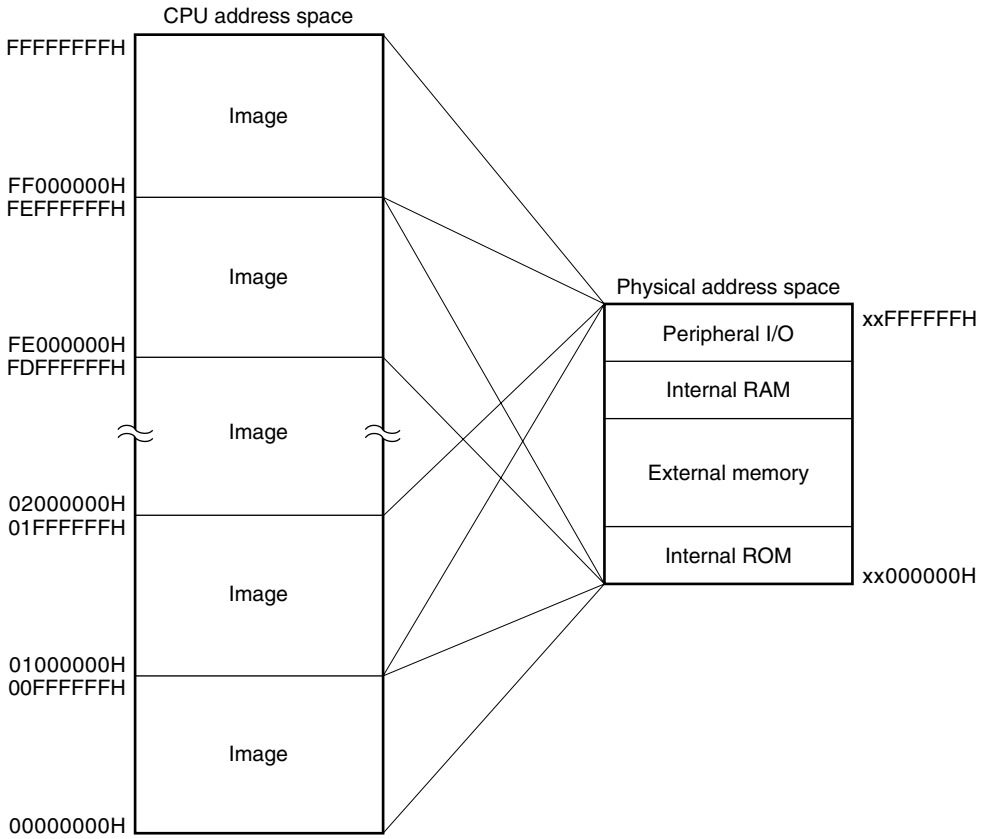
PC: 00FFFFFEH + 4 → PC: 00000002H (the carry to bit 24 is ignored)

## 1.4 Address Space

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

The physical address space of the V853 is 16 MB. In a 4 GB CPU address space, 256 × 16 MB spaces appear as images.

**Figure 1-2. Address Space Images**



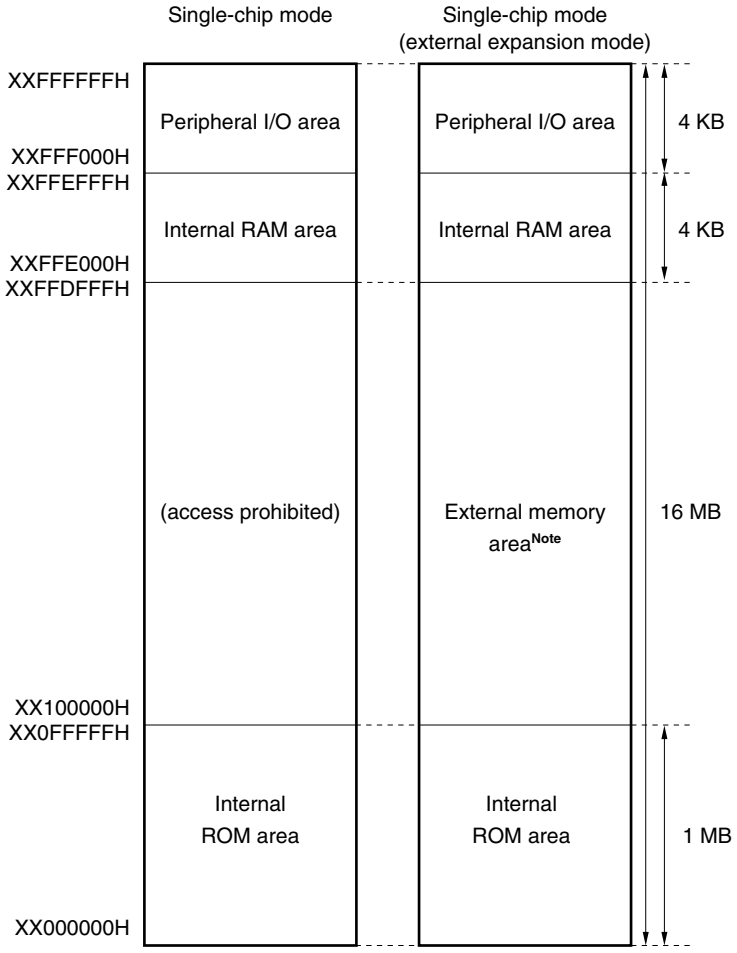
## ★ 1.5 Memory Map

Figure 1-3 shows the V853 memory map. The external expansion mode is set using the memory expansion mode register (MM) at the time of a reset.

Refer to **V853 Hardware User's Manual** regarding the MM register.

**Figure 1-3. Memory Map (1/2)**

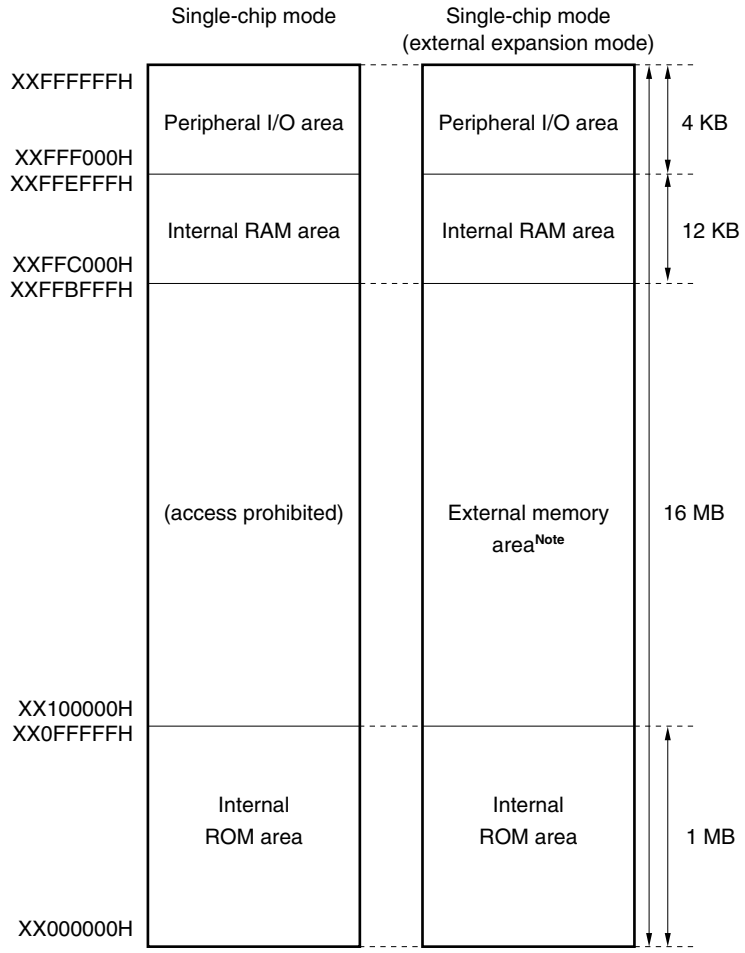
**(a) For μPD703003A, 703004A, and 70F3003A**




★ **Note** The program space for the V853 is 16 MB, but a space of up to 1 MB can be used for the physical external memory. Note that an external device cannot be used connected to the internal RAM area, peripheral I/O area, or internal ROM area; even if connected physically, the device will not be regarded as an access target.

**Figure 1-3. Memory Map (2/2)**

**(b) For μPD703025A and 70F3025A**




★ **Note** The program space for the V853 is 16 MB, but a space of up to 1 MB can be used for the physical external memory. Note that an external device cannot be used connected to the internal RAM area, peripheral I/O area, or internal ROM area; even if connected physically, the device will not be regarded as an access target.

# CHAPTER 2 OVERVIEW OF CA850

This chapter provides an overview of the V850 Series C compiler package (CA850). The CA850 has the following features.

- Language specifications that conform to ANSI standards
- Advanced optimization
- Features for embedded control (ROMization support)
- Improved descriptiveness due to extended language specifications

The CA850 provides device files that contain machine type dependent information for each target device product. The same compiler can be used to generate object code corresponding to each V850 device.

## 2.1 How General-Purpose Registers are Used in CA850

A list of the registers used by the CA850 is shown below.

**Table 2-1. Register Set Used by CA850**

| Register Name | Use | Details of Use |
|---|---|---|
| r0 | Zero register | Used in arithmetic operations as the value of 0. Also used in referencing data located in the const section (.const section) (a read-only section placed in a ROM area). |
| r1 | Assembler reserved register | Used at time of instruction expansion in assembler. |
| ★ r2 | Address/data variable registers (when r2 is not used by the real-time OS to be used) | |
| r3 (sp) | Stack pointer | Used as pointer to head of stack frame. |
| r4 (gp) | Global pointer | Used when referencing external variable. |
| r5 (tp) | Text pointer | Used as pointer to head of text section (.text section). |
| r6 to r9 | Argument registers | Used to pass arguments. |
| r10 to r19 | Work registers | Used as work registers during an operation.<br>(r10 also is used to pass the return value of a function.) |
| r20 to r29 | Register variable registers | Used as areas for register variables. |
| r30 (ep) | Element pointer | Used when referencing an external variable located in the internal RAM or external RAM section. |
| r31 (lp) | Link pointer | Used to pass the return address of a function. |

## 2.2 Notes on Source Code Specification

Notes on specifying source code to be compiled by the CA850 are shown below. Refer to **CA850 C Language User's Manual** for details of each item.

### 2.2.1 Accessing peripheral I/O registers

Specifying the pragma directive shown below makes it easy to access peripheral function registers using register names at the C language level.

*#pragma ioreg*

Peripheral function register names can be used after this pragma directive. Refer to **CA850 C Language User's Manual** for register names that can be specified.

### 2.2.2 Specifying bit access

V850 Series instructions support bit access. Methods of specifying bit access in the C language are shown below.

- *register-name.bit-number* → **Example**) r10.3 = 1;
- *peripheral-register-name.bit-name* → **Example**) TMC4.CE4 = 0;

### 2.2.3 Specifying assembler instructions

The CA850 can be used to specify assembler instructions within C language source in the formats shown below.

- *_ _asm (character-string-constant)*; → **Example**) _ _asm("nop");
- *#pragma asm* → **Example**)
  *assembly description*
  *#pragma endasm*

```
#pragma asm
    mov     r0,r10
    st.w    r10,$_i
#pragma endasm
```

### 2.2.4 Specifying inline expansion

If the -Ot (execution speed priority optimization) option is specified during a compile operation, the compiler performs inline expansion of the specified functions.

*#pragma inline function-name[,function-name...]*

Performing inline expansion increases the size of the program code, but it can prevent overhead on function calls.

### 2.2.5 Specifying interrupt disabling and interrupt level settings

The CA850 can disable interrupts using the two methods shown below.

- Set or clear ID bit of PSW using __EI( ) or __DI( ).

**Example** →

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

- *#pragma block_interrupt function-name*

  The function specified in the above pragma directive is interrupt disabled, including prolog processing and epilog processing.

**Example** →

```
void func1(void)
{
    __DI( );
      :                 Prolog processing and epilog processing are
    __EI( );            not interrupt disabled.
}

#pragma block_interrupt func2
void func2(void)
{
    a = 0;
      :                 Entire function including prolog processing and epilog
    x++;                processing is interrupt disabled.
}
```

### 2.2.6 Specifying interrupt handler processing

In the CA850, the method of specifying the interrupt handler processing that operates when an interrupt or exception occurs differs for "applications using a real-time OS" and "applications not using a real-time OS". The interrupt handler specification method in an "application not using a real-time OS" is shown here.

- *#pragma interrupt interrupt-request-name function-name*
  *_ _interrupt function-definition or function-declaration*

**Example)** → 
```
#pragma interrupt NMI nmi_int_rtn
_ _interrupt void nmi_int_rtn(void)
{
    (NMI interrupt servicing)
}
```

**Remarks** #pragma interrupt performs a registration to a handler address.
For the function qualified by _ _interrupt, saving/returning data to/from a register and a reti instruction are output by the compiler.

Functions that are specified as interrupt handlers are restricted as shown below.

- A handler function cannot be expanded inline. If a #pragma inline directive is specified, it is ignored.
- A handler function cannot be called by another function. However, if it is called by a function in another file, the compiler cannot check this.

## 2.3 Calling Functions

This section describes the calling of functions in the CA850. In the CA850, the *jarl* instruction is used when calling a function and the *jmp* instruction is used in returning from the called function.

Up to four words of arguments are stored in argument registers r6 to r9 and arguments in excess of four words are stored in a stack frame.

A stack frame is generated by the prolog code of the called function moving the stack pointer (sp) the necessary amount in the direction of lower addresses. The epilog code of the function erases a stack frame by moving sp the amount by which it was moved in prolog processing in the direction of higher addresses.

In the CA850, sp always points to the lowest stack frame.

## 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 CA850.

### 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 in CA850 are shown below.

**(1) Identifiers**

When an external name is specified, the CA850 outputs it with "_" (underscore) attached at the beginning in output to the assembler. Therefore, when calling an assembly language program from a C language program, begin the referenced identifier with "_" and reference it on the C language program side with the leading "_" removed.

**(2) Stack frames**

The CA850 outputs code that supposes 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 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**

The CA850 stores four words of arguments in argument registers and stores excess arguments in the stack frame of the function on the calling side.

**(4) Return value from assembly language program**

The CA850 generates code that supposes that a function call stores the return value from the function in r10. Therefore, be sure that an assembly language program called from a C language program stores the return value in r10.

**(5) Return address**

The CA850 outputs code that stores the return address in the link pointer (lp) on a function call. Therefore, execute "jmp [lp]" in returning control to the C language program from the assembly language 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 CA850 are shown below.

**(1) Stack frames**

The CA850 outputs code that supposes 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 CA850 holds the values of register variable registers before and after calling a C language program but it does not hold work register values.

**(3) Arguments to C language program**

When passing arguments to a C language program, store the first four words in argument registers and place any excess in the direction of higher addresses starting from the address to which the stack pointer (sp) is pointing.

**(4) Return value from C language program**

The CA850 stores the return value from a function in r10.

**(5) Return address**

The CA850 outputs code that supposes that the return address is stored in the link pointer (lp) on a function call. Therefore, be sure that the assembly language program that calls the C language program sets the return address in lp using the *jarl* instruction.

## 2.5 Sections Positioned by CA850

The CA850 positions each section as shown in Figure 2-1.

**Figure 2-1. Sample Image of Positioning of Each Section in Memory by CA850 (With Internal ROM)**

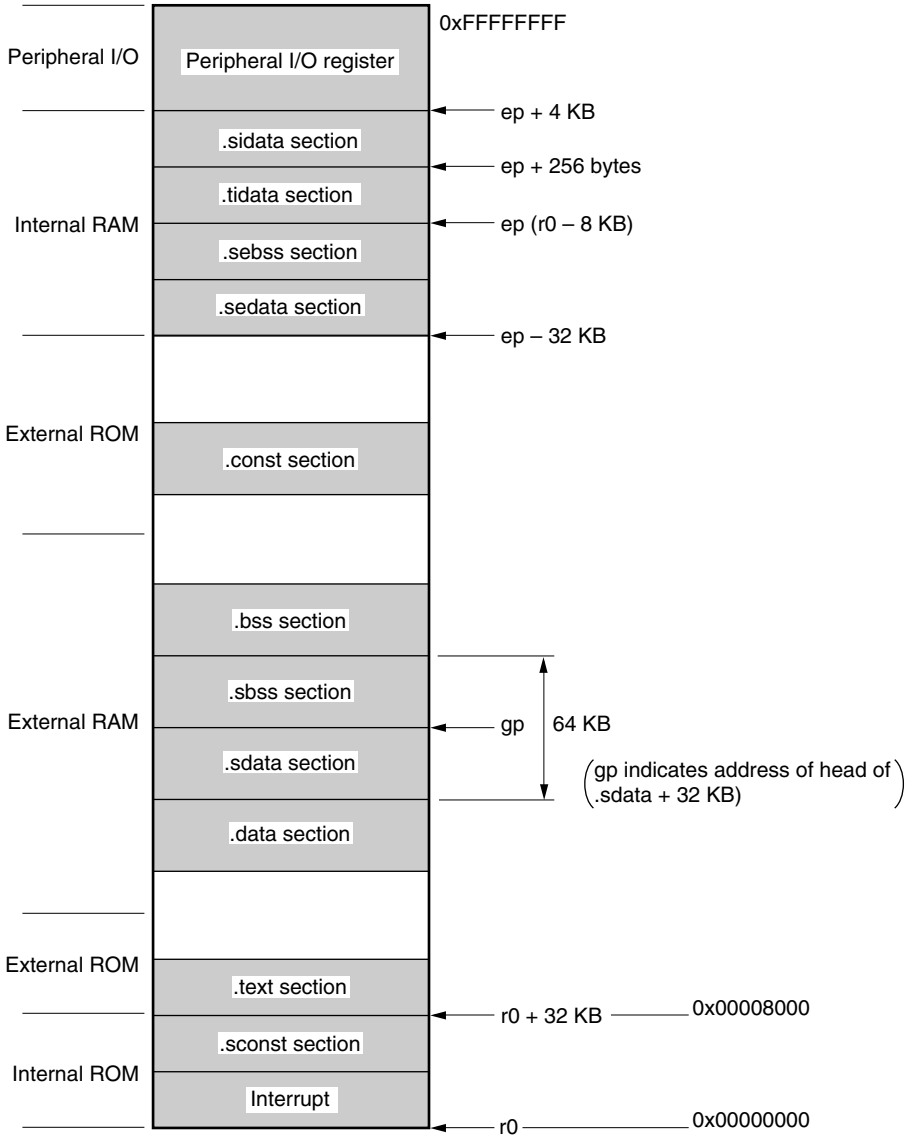


**Table 2-2. Types of Sections Positioned by CA850**

| Type | Characteristics | Specification String |
|---|---|---|
| .tidata section<br>(tiny internal data) | Section accessible using ep-relative sld or sst instruction.<br>Fast access is possible by placing in internal RAM. | tidata |
| .data section<br>.bss section<br>(data/bss) | Section accessible using gp-relative ld and st instructions (2 instructions).<br>Allocated in the .data section if initial values are set, and in the .bss section if they are not. | data |
| .sdata section<br>.sbss section<br>(sdata/sbss) | Sections accessible using gp-relative ld or st instruction (64 KB combined).<br>Allocated in the .sdata section if initial values are set, and in the .sbss section if they are not. | sdata |
| .sedata section<br>.sebss section<br>(small extended data/<br>small extended bss) | Sections accessible using ep-relative ld or st instruction.<br>Allocated in the .sedata section if initial values are set, and in the .sebss section if they are not. | sedata |
| .sidata section<br>(small internal data) | Section accessible using ep-relative ld or st instruction.<br>Fast access is possible by placing in internal RAM. | sidata |
| .sconst section<br>(small const data) | Section accessible using r0-relative ld or st instruction.<br>Place in internal ROM on a device that has internal ROM and in external memory if the device does not have internal ROM. Interrupt functions are given priority in allocation. For read-only data. | sconst |
| .const section<br>(const data) | Section accessible using r0-relative ld and st instructions (2 instructions).<br>Place in external memory for read-only data. | const |

## 2.6 Startup Module

Code generated by the CA850 supposes that appropriate values are set in the stack pointer (sp), text pointer (tp), global pointer (gp), and element pointer (ep). These settings normally are made using a startup module.

A startup module is provided in the compiler package. Customize a startup module for each target system based on this startup module.

# 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 to 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 acknowledged, 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

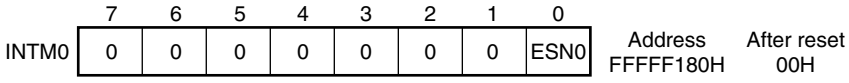
Non-maskable interrupts are acknowledged unconditionally, even in an interrupt disabled (DI) state. In addition, they are not subject to interrupt priorities and are the highest priority interrupt requests of all interrupts.

For a non-maskable interrupt, an interrupt occurs when the valid 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 servicing (see Figure 3-2). If a non-maskable interrupt is requested during non-maskable interrupt servicing, it is held pending. The pending non-maskable interrupt request is acknowledged after the currently executing interrupt servicing ends (after RETI instruction execution) or after an LDSR instruction sets the NP flag of the PSW to 0.

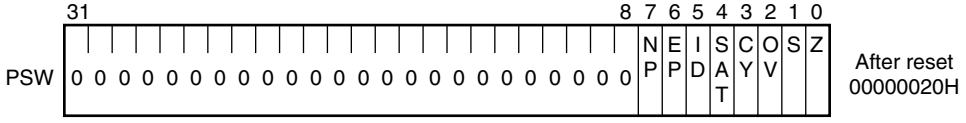
External interrupt mode register 0 (INTM0) can specify that the valid edge of the non-maskable interrupt is either the rising edge or falling edge (see Figure 3-1).

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




| Bit position | Bit name | Description |
|---|---|---|
| 0 | ESN0 | Edge Select NMI<br>Specifies valid edge of NMI pin.<br>0: Falling edge<br>1: Rising edge |

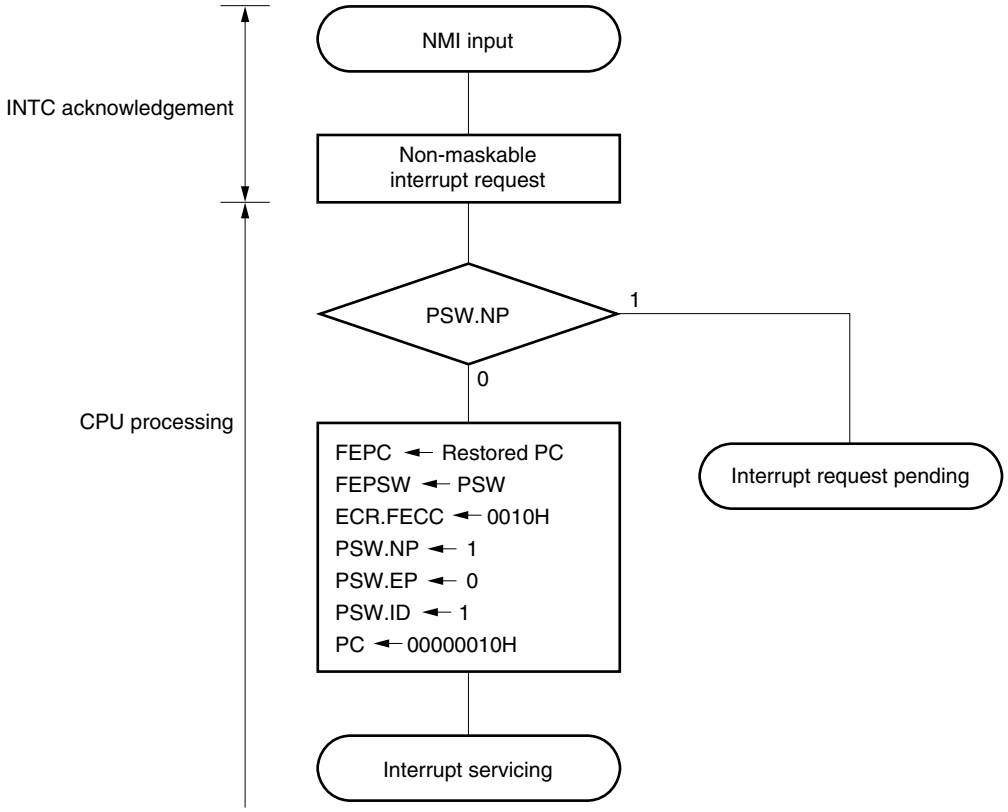
**Figure 3-2. NP Flag**




| Bit position | Bit name | Description |
|---|---|---|
| 7 | NP | NMI Pending<br>Indicates whether NMI interrupt servicing is in progress.<br>0: NMI interrupt servicing not in progress.<br>1: NMI interrupt servicing in progress. |

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

**Figure 3-3. Non-Maskable Interrupt Request Processing Flow**



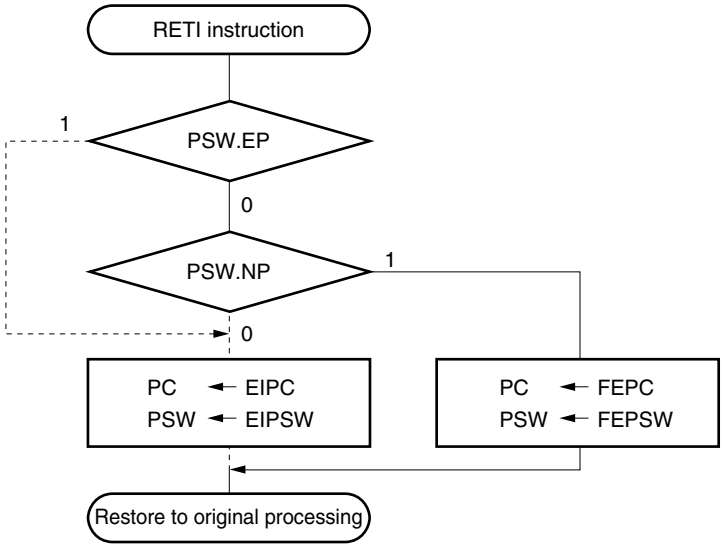

The operation of non-maskable interrupt request acknowledgement processing is shown below.

<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 held pending, the request is ignored. That is, only one non-maskable interrupt can be held 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.

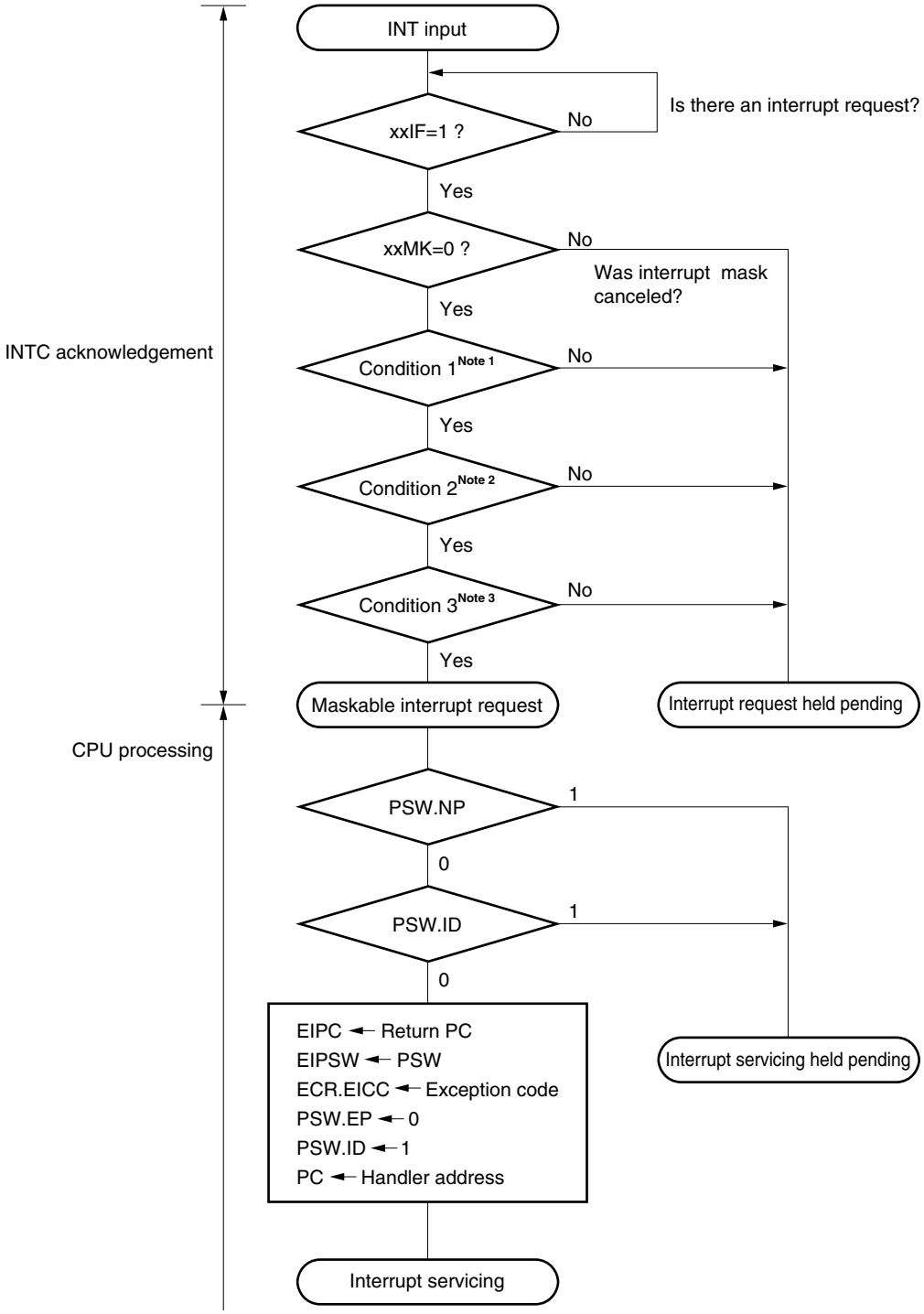
### 3.1.3 Maskable Interrupts

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 acknowledged, the status becomes interrupt disabled.

Figure 3-5 shows the flow of maskable interrupt request acknowledgement 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?

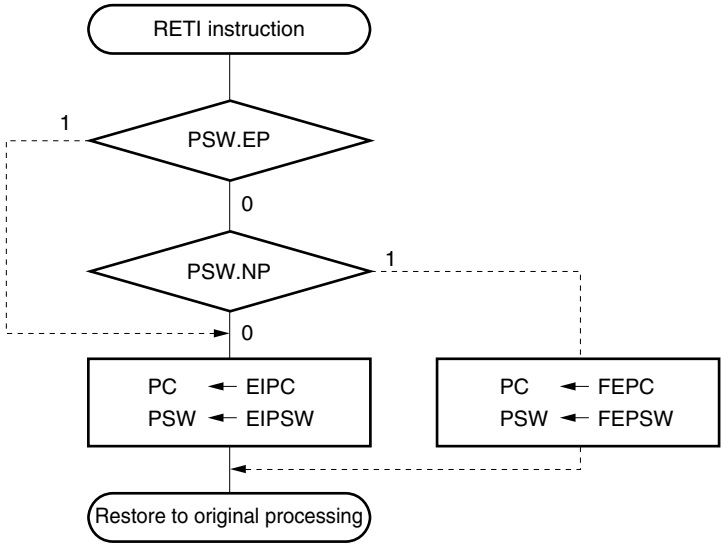
**3.** Condition 3: Is default priority highest among interrupt requests with the same priority?

The operation of maskable interrupt acknowledgement 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.

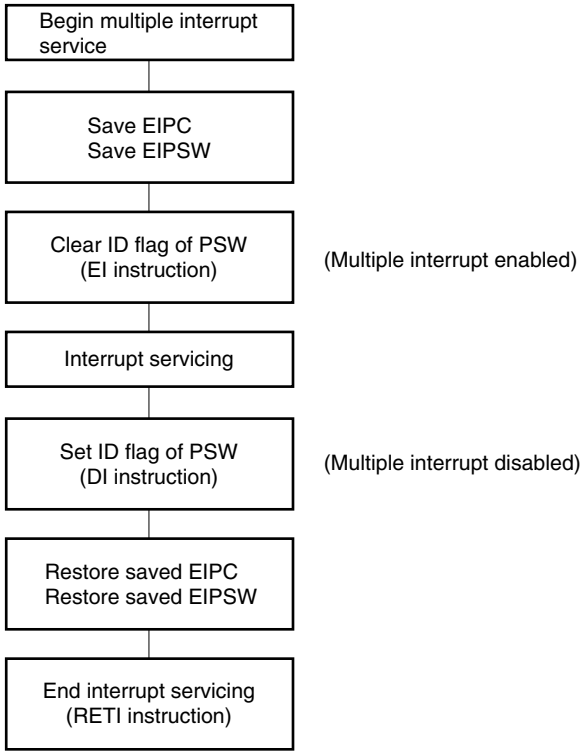
### 3.1.4 Multiple interrupts

Although the status becomes interrupt disabled (DI) when a maskable interrupt is acknowledged, 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 acknowledgement 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.

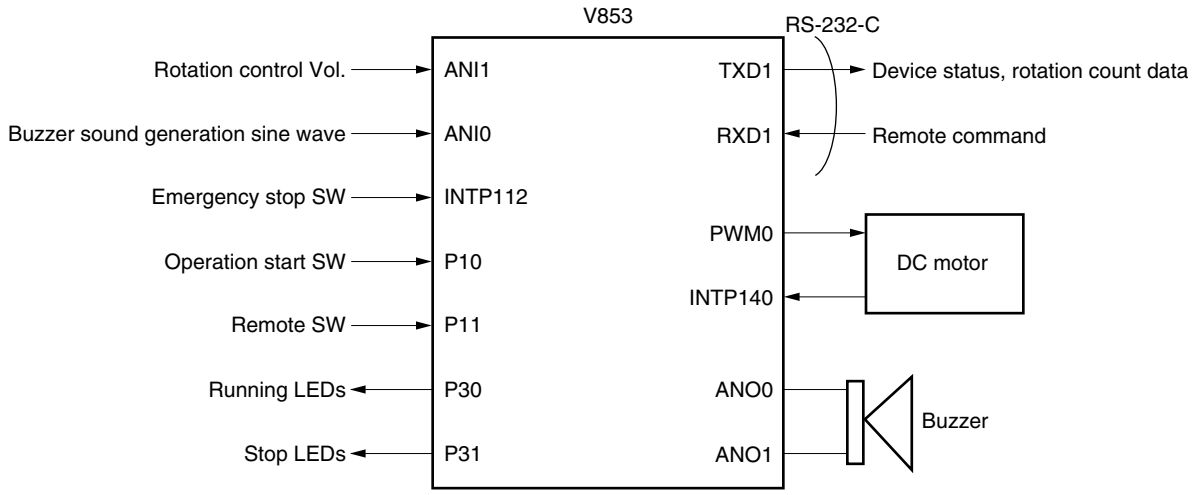
**Figure 3-7. Flow of Multiple Interrupt Service**



### 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<br>This is the interrupt request flag.<br>0: No interrupt request<br>1: Interrupt request present<br>The xxIFn flag is reset automatically by the hardware when an interrupt request is acknowledged. |
| 6 | xxMKn | Mask Flag<br>This is the interrupt mask flag.<br>0: Enable interrupt servicing<br>1: Disable interrupt servicing (pending) |
| 2-0 | xxPRn2 to xxPRn0 | Priority<br>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 valid edge. Also specify the valid edge for NMI.

The external interrupt mode registers set the valid 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<br>6, 4, 2, 0 | ES(m–1)n1<br>ES(m–1)n0<br>(m = 4 to 1,<br>n = 3 to 0) | Edge Select<br>Specifies valid 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 valid edge of ADTRG.

Figure 3-12 shows sample settings of external interrupt mode registers when the valid 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
 * File Name:    intinit.c
 * ------------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */

/* ------------------------------------------------
 * 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 valid 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 */
}
```

## 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 | RXD0/SI0 | |
| | P24 | $\overline{\text{SCK0}}$ | |
| | P25 | TXD1/SO1 | |
| | P26 | RXD1/SI1 | |
| | P27 | $\overline{\text{SCK1}}$ | |

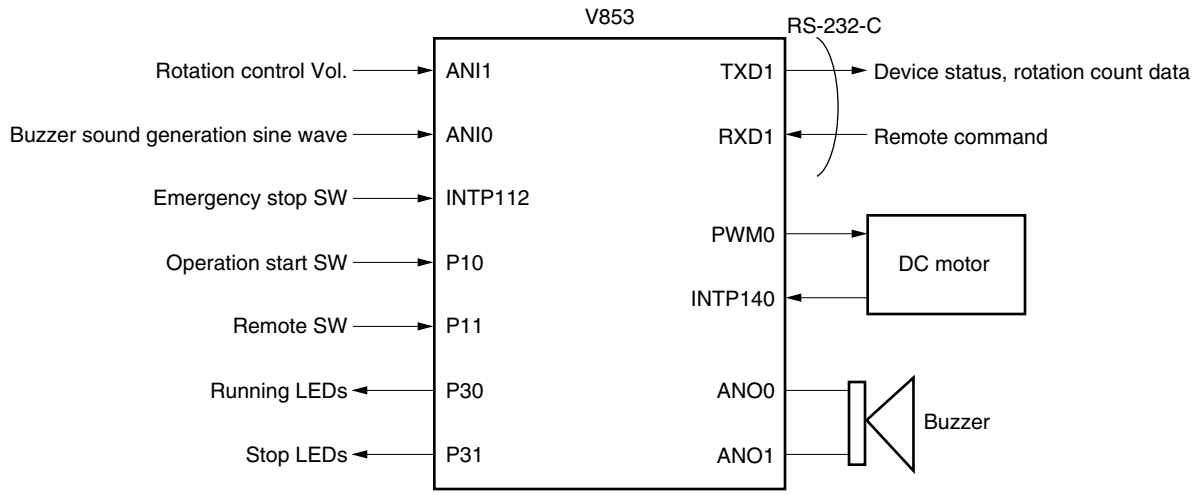
**Table 3-1. Relationship Between Control Pins and Port Pins (2/2)**

| 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 Hardware User's Manual** for details of port n mode control registers (PMCn).

**Figure 3-14. Port n Mode Control Registers (PMCn) Settings**

| | 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<br>Sets the operating mode of pins Pn0 to Pn7.<br>0: I/O port mode<br>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 to 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 I/O 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<br>(n = 0 to 5, 11) | Port Mode<br>Sets input or output mode for pins Pn0 to Pn7.<br>0: Output mode (output buffer on)<br>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<br>Sets input or output mode for pins P63 to P70.<br>0: Output mode (output buffer on)<br>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<br>Sets input or output mode for pins P96 to P90.<br>0: Output mode (output buffer on)<br>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<br>Specifies the operating mode of pins P35 to P37.<br>0: INTP133 to INTP131 input mode<br>1: CSI3 I/O mode |
| 1 | PCM1 | Port Control Mode<br>Specifies the operating mode of pins P15 to P17.<br>0: INTP123 to INTP121 input mode<br>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<br>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 to 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 MB external memory space
 *
 * ------------------------------------------------
 * History:
 *   1997/3/17   Ver 0.00.00
 * File Name:    portinit.c
 * ------------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */
/* ------------------------------------
 * 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 */
}
```

## 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)

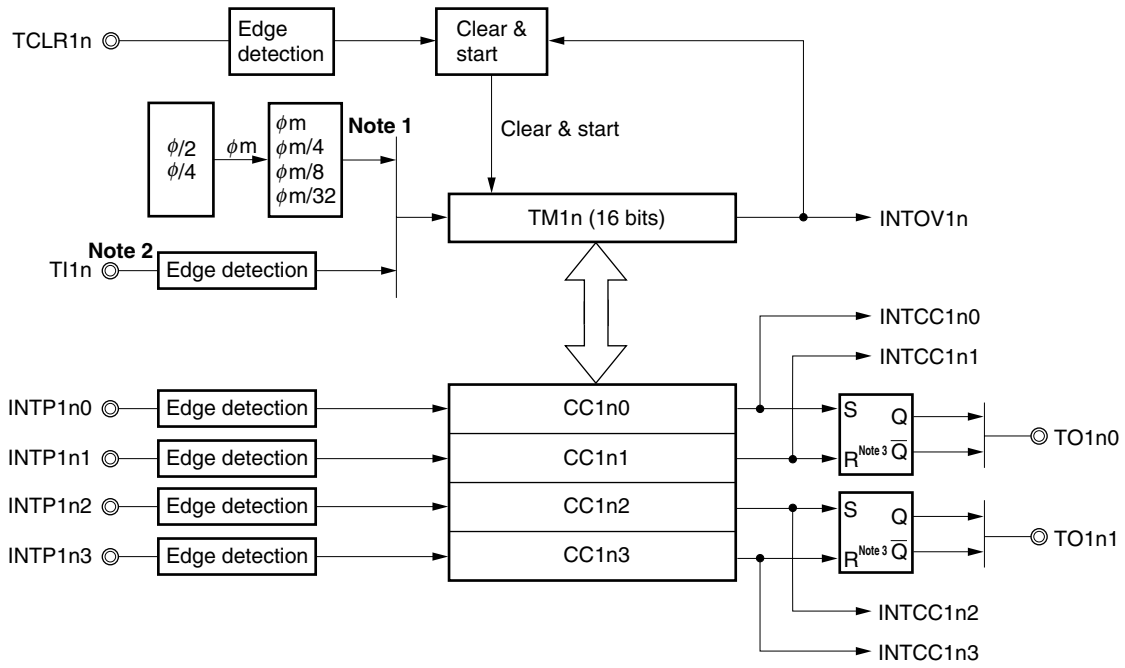
There are four timers for timer 1 and they have four capture/compare common registers each, for a total of 16. For the count clock source, system clock divisions can be selected or an external pulse can be input.

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.** $\phi$: 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<br>Controls timer operation.<br>See Figure 3-20. |
| 4 | ETI1n | External TI1n Input<br>Specifies switching between internal and external count clock.<br>0: Specifies $\phi$ system (internal).<br>1: Specifies TI1n (external). |
| 3, 2 | PRS1n1,<br>PRS1n0 | Prescaler Clock Select<br>Selects internal count clock ($\phi$ m indicates the intermediate clock). |
| 1 | PRM1n1 | Prescaler Clock Mode<br>Selects intermediate clock ($\phi$ m) of count clock.<br>($\phi$ : Internal system clock)<br>0: $\phi$/2<br>1: $\phi$/4 |

| PRS1n1 | PRS1n0 | Count clock |
|---|---|---|
| 0 | 0 | $\phi$m |
| 0 | 1 | $\phi$m/4 |
| 1 | 0 | $\phi$m/8 |
| 1 | 1 | $\phi$m/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 | $\phi$/2 |
| 0 | 0 | 1 | $\phi$/4 |
| 0 | 1 | 0 | $\phi$/8 |
| 0 | 1 | 1 | $\phi$/16 |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | $\phi$/32 |
| 1 | 1 | 0 | $\phi$/64 |
| 1 | 1 | 1 | $\phi$/128 |

**Remark** n = 1 to 4, $\phi$: Internal system clock

When an external count clock is specified, specify the valid edge of the signal input to the TI1n pin. Specify the valid 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 | Valid 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<br>(n = 1 to 4) | Count Enable<br>Controls operation of timer.<br>0: Timer stops in “0000H” state and does not operate.<br>1: Timer performs count operation. However, if ECLR1n of TUM1n register is 1, timer does not start counting up until there is TCLR1n input.<br>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<br>CE1n 0: Stop count & clear counter | No effect on operation |
| 1 | CE1n 1: No effect on operation<br>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 | 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<br>Specifies operation after timer overflow. This flag is in effect only for TM1n.<br>0: After timer overflow, timer continues counting up.<br>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.<br>If ECLR1n = 0: Operation to write “1” to bit CE1n<br>If ECLR1n = 1: Trigger input to timer clear pin (TCLR1n) |
| 12 | ECLR1n | External Input Timer Clear<br>Enables clearing of timer according to TM1n external clear input (TCLR1n).<br>0: Do not clear according to external input.<br>1: Clear TM1n according to external input. After clearing, start counting up. |
| 11, 10 | TES1n1, TES1n0 | TI1n Edge Select<br>Specifies valid edge of external clock input (TI1n). |
| 9, 8 | CES1n1, CES1n0 | TCLR1n Edge Select<br>Specifies valid edge of external clear input (TCLR1n).<br>CES1n1 / CES1n0 / Valid edge:<br>0 / 0 / Falling edge<br>0 / 1 / Rising edge<br>1 / 0 / RPU (reserved)<br>1 / 1 / Rising and falling edges |
| 7 to 4 | CMS1nm<br>(m = 0 to 3) | Capture/Compare Mode Select<br>Selects operating mode of capture/compare register (CC1nm).<br>See Figure 3-23. |
| 3 to 0 | IMS1nm<br>(m = 0 to 3) | Interrupt Mode Select<br>Selects INTP1nm or INTCC1nm as interrupt source.<br>See Figure 3-23. |

**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. Specification of OSTn Bit of TUM1n**

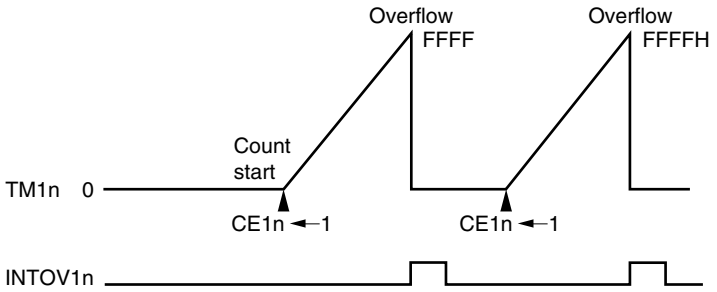
| OSTn | Operation on Counter Overflow |
|---|---|
| 0 | Set OVF1n bit of TOVS register to 1<br>Generate overflow interrupt<br>Continue count up operation (count value: FFFF → 0) |
| 1 | Set OVF1n bit of TOVS register to 1<br>Generate overflow interrupt<br>Stop count up operation (count value: held at 0) |

**Remark** n = 1 to 4

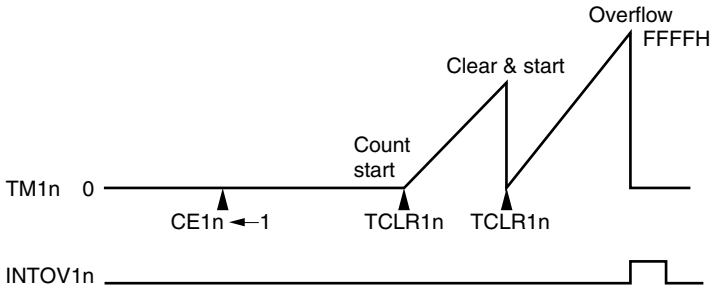
Figure 3-22 shows count start and overflow operations.

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

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



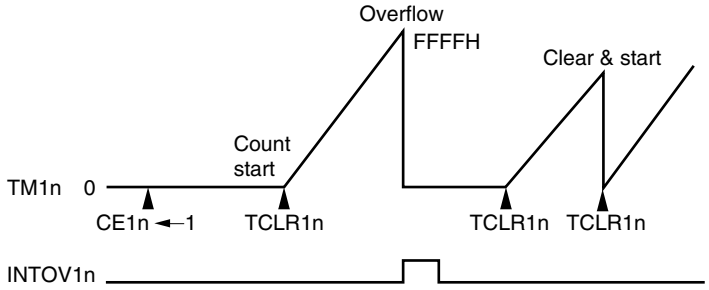

**(b) Timer clear/start operation due to TCLR1n signal input (when ECLR1n = 1, OSTn = 0)**




**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 valid 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 valid 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)**

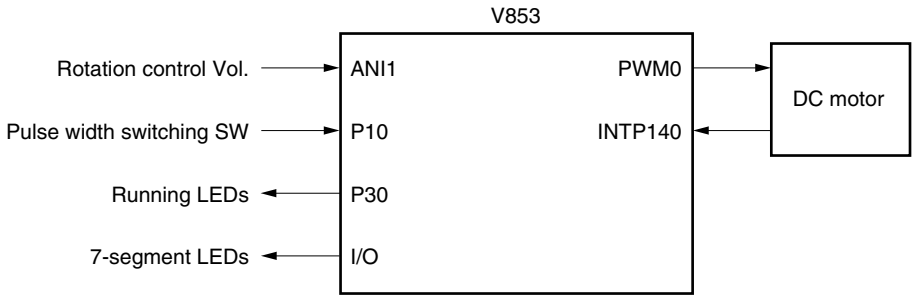
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | After reset |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TUM1n | 0 | 0 | OSTn | ECLR 1n | TES 1n1 | TES 1n0 | CES 1n1 | CES 1n0 | CMS 1n3 | CMS 1n2 | CMS 1n1 | CMS 1n0 | IMS 1n3 |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

| Bit position | Bit name | Description |
|---|---|---|
| 13 | OSTn | Overflow Stop<br>Specifies operation after timer overflow.<br>See Figure 3-21. |
| 12 | ECLR1n | External Input Timer Clear<br>Enables clearing of timer due to TM1n external clear input (TCLR1n).<br>See Figure 3-21. |
| 11, 10 | TES1n1, TES1n0 | TI1n Edge Select<br>Specifies valid edge of external clock input (TI1n). |
| 9, 8 | CES1n1, CES1n0 | TCLR1n Edge Select<br>Specifies valid edge of external clear input (TCLR1n).<br>See Figure 3-21. |
| 7 to 4 | CMS1nm<br>(m = 0 to 3) | Capture/Compare Mode Select<br>Selects operating mode of capture/compare register (CC1nm).<br>0: Operate as capture register. However, capture operation when capture register is specified is performed only when CE1n bit of TMC1n is 1.<br>1: Operate as compare register. |
| 3 to 0 | IMS1nm<br>(m = 0 to 3) | Interrupt Mode Select<br>Selects INTP1nm or INTCC1nm as interrupt source.<br>0: Make signal that matches compare register (INTCC1nm) interrupt signal.<br>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 $\phi$/128 and select an internal clock.

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

**Figure 3-25. TUM14, TMC14, and INTM4 Settings**

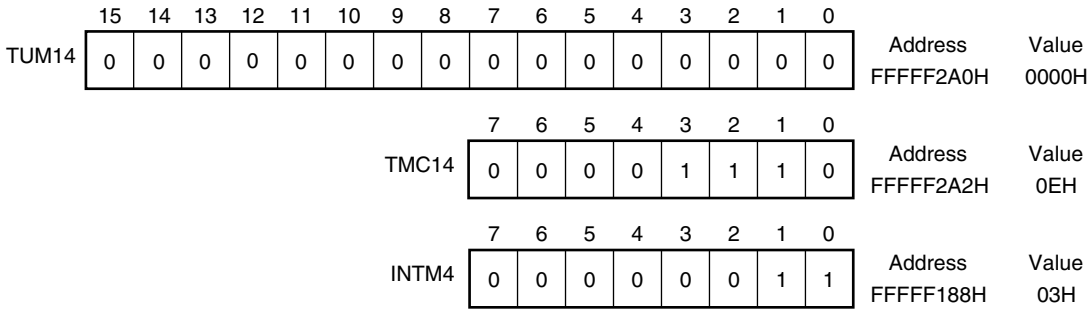
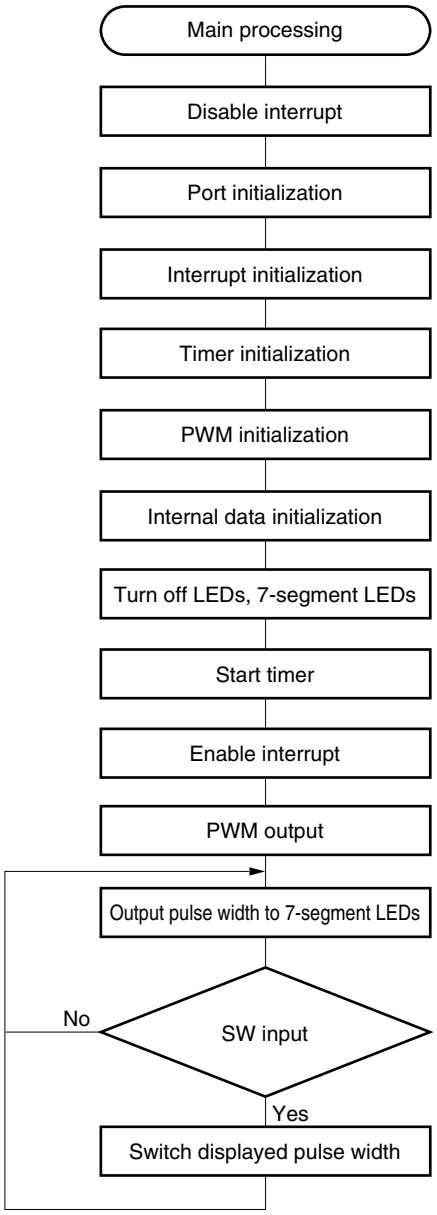


Figure 3-26 shows the flow of pulse width measurement processing that operates on the sample hardware.

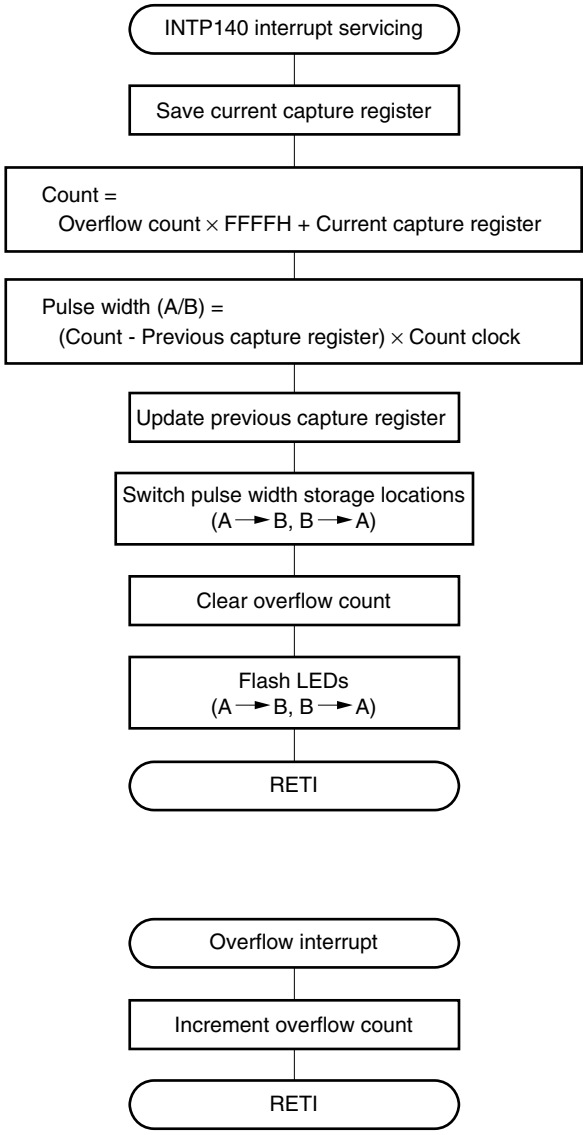
**Figure 3-26. Flow of Pulse Width Measurement Processing (1/2)**




**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)**

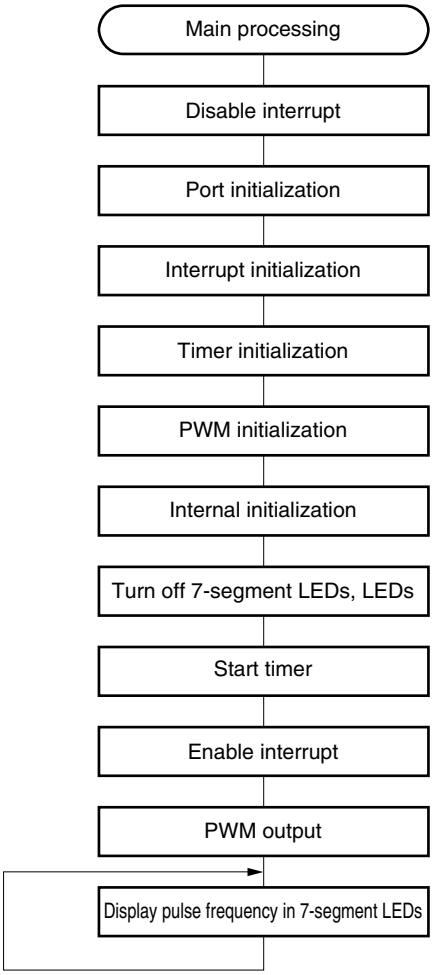
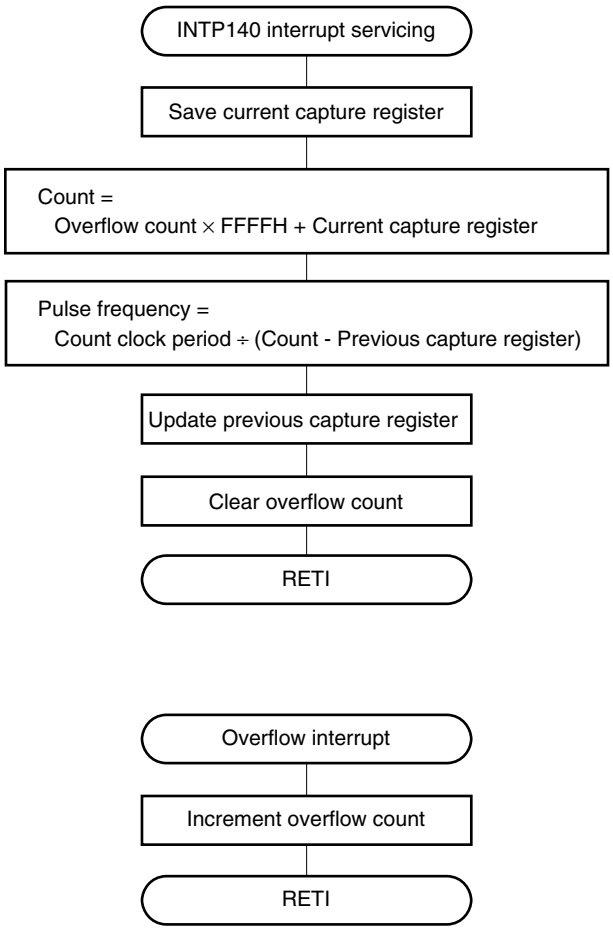


**Figure 3-28. Flow of Pulse Frequency Measurement Processing (2/2)**



### (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
 * File Name:    timcapt.c
 * ------------------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */

#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: Inactive) */
static int OvflowCnt;   /* Overflow counter */
static inc Clk;            /* Clock value */
/* ------------------------------------
 * NMI service
 * Interrupt source:  NMI
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   ;      /* Do nothing */
}

/* ------------------------------------
 * Pulse width measurement
 * Interrupt source:  INTP140
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTP140 int_intp140
__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) {
      P3.0 = 1;
   } else {
      P3.0 = 0;
   }

}

/* ------------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTAD int_ad
__interrupt
void int_ad(void)
{
   /* Set A/D conversion result in PWM buffer register */
   PWM0 = ADCR1;
   /* A/D converter conversion start */
   CE = 1;
}

/* ------------------------------------
 * Count timer counter overflows
 * Interrupt source:  INTOV14
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTOV14 int_ov14
__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 valid 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 φm/32 */
                   /* PRS140 = 1:  (Timer count clock = φ/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 φ/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;
   Clk = 1 / (5000000 * 5);  /* System clock  5 x 5 MHz = 25 MHz */
                             /* Operate using 1 system clock = 40 ns */

   /* A/D conversion start */
   CE = 1;   /* (CE) A/D conversion start */

   /* Start timer */
   CE14 = 1;

   /* Enable interrupt */
   __EI();

   /* Start PWM output */
   PWME0 = 1;

   while (1) {
      /* Get display pulse selection flag */
      pulsFlg = P1.0;
      /* To confirm display, display once per 100,000 times */
      DispCnt++;
      if ( DispCnt >= 100000 ) {
         /* Display pulse width */
         Disp7SegLED(PulsWidth[pulsFlg], 0, 1);  /* Hexadecimal display without
         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 acknowledgement 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)**

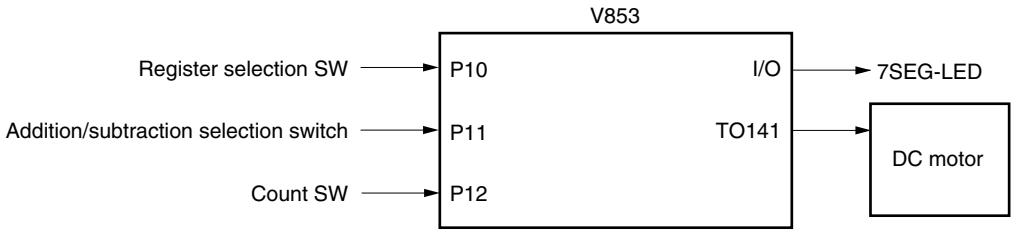
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | After reset |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TUM1n | 0 | 0 | OSTn | ECLR1n | TES1n1 | TES1n0 | CES1n1 | CES1n0 | CMS1n3 | CMS1n2 | CMS1n1 | CMS1n0 | IMS1n3 | IMS1n2 | IMS1n1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

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

| | 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 | 1 | 0 | 0 | 0 | 1 | FFFFF2A0H | 0011H |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| TMC14 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | FFFFF2A2H | 0CH |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| TMC11 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | FFFFF056H | 02H |

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<br>Enables corresponding timer output (TO140, TO141).<br>0: Timer output is disabled.<br>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.<br>1: Timer output function is enabled.<br>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<br>Specifies active level of timer output.<br>0: Active level is low level.<br>1: Active level is high level. |

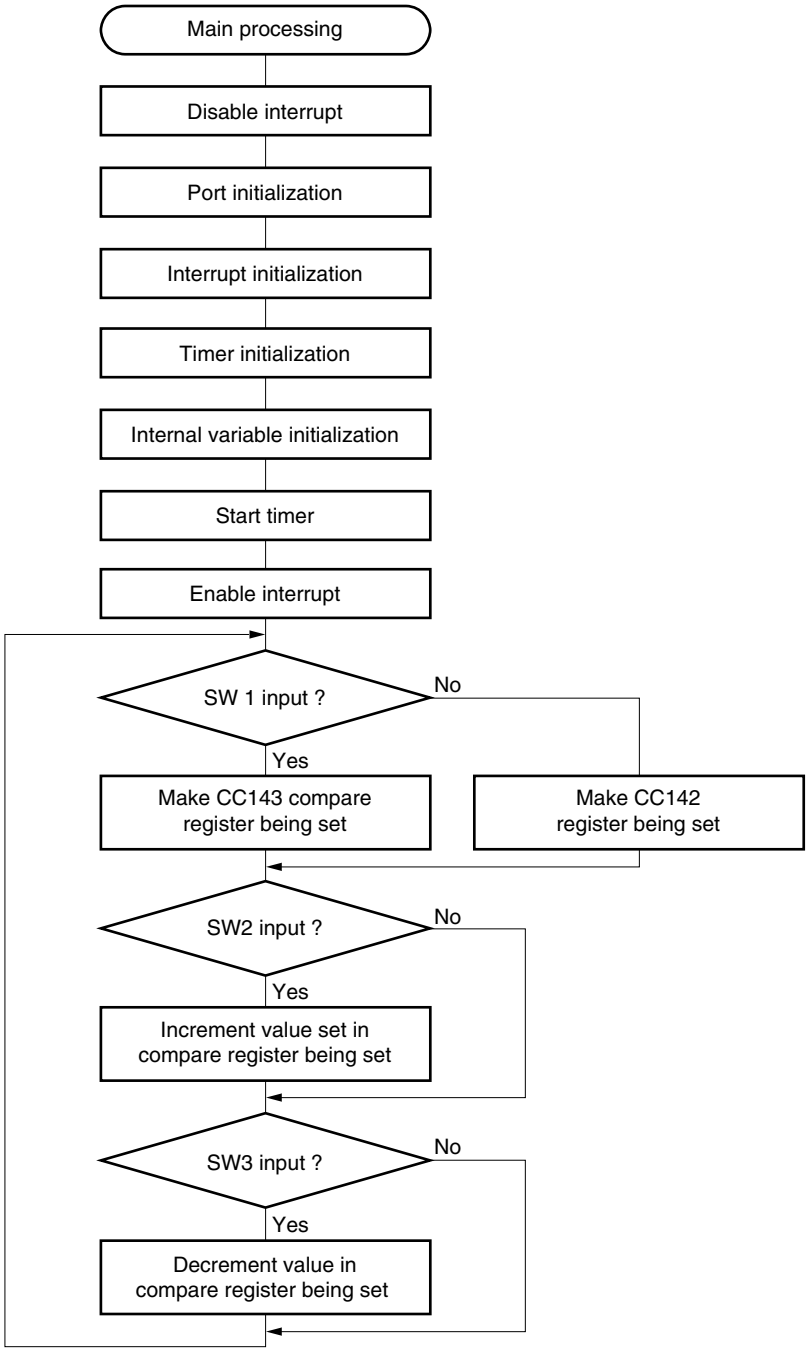
**Caution TO140 and TO141 outputs do not change on an external interrupt signal (INTP140 to INTP143). When using TO140 and TO141 signals, specify the capture/compare register as a compare register (CMS bit = 1).**

**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
 * File Name:     timcomp.c
 * ------------------------------------------------------
 */

#pragma ioreg      /* Make peripheral I/O register names usable */

#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 interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   ;      /* Do nothing */
}

/* ------------------------------------------------------
 * Interval timer interrupt servicing
 * Interrupt source:  INTCM4
 * Arguments:  None
 * Return value: None
 * ------------------------------------------------------
 */
#pragma interrupt INTCM4 int_cm4
__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 interrupt INTCC142 int_cc142
__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 interrupt INTCC143 int_cc143
__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 valid 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 φm/4 */
                   /* PRS140 = 1:  (Timer count clock = φ/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 = φ/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)
   {
      int    pulsFlg = 0;   /* Display pulse selection flag */
      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 */
      CE14 = 1;    /* Start timer 14 */
      CE4 = 1;     /* Start timer 4 */

      /* Enable interrupt */
      __EI();

      while ( 1 ) {
         /* Input pulse width setting switching SW */
         SelFlg = P1.0;
         /* Input count setting switching SW (On: Increment, Off: Decrement) */
         IncFlg = P1.1;
         /* Input count SW (On: Count pulse width value up/down) */
         CntUpFlg = P1.2;
```

```
      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<br>Controls timer operation.<br>0: Timer stops in “0000H” status and does not operate.<br>1: Timer performs count operation. |
| 2 | PRS40 | Prescaler Clock Select<br>Selects internal count clock ($\phi$ m is the intermediate clock)<br>0: $\phi$ m<br>1: $\phi$ m/32 |
| 1, 0 | PRM41, PRM40 | Prescaler Clock Mode<br>Selects count clock intermediate clock ($\phi$ m) (where $\phi$: internal system clock). |

| PRM41 | PRM40 | $\phi$ m |
|---|---|---|
| 0 | 0 | $\phi$/2 |
| 0 | 1 | $\phi$/4 |
| 1 | 0 | $\phi$/16 |
| 1 | 1 | $\phi$/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 | $\phi$/2 |
| 0 | 0 | 1 | $\phi$/4 |
| 0 | 1 | 0 | $\phi$/16 |
| 0 | 1 | 1 | $\phi$/32 |
| 1 | 0 | 0 | $\phi$/64 |
| 1 | 0 | 1 | $\phi$/128 |
| 1 | 1 | 0 | $\phi$/512 |
| 1 | 1 | 1 | $\phi$/1024 |

**Remark** $\phi$**:** Internal system clock

### 3.3.7 Operation of interval timer (timer 4)

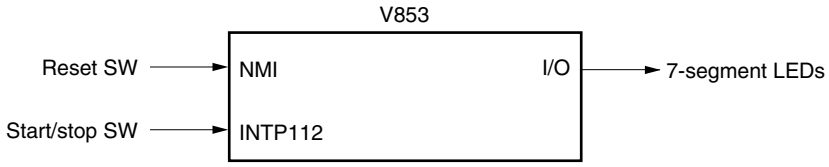
For each internal count clock specified by TMC4, compare the compare register (CM4) to timer 4 (TM4). If CM4 matches TM4, a match interrupt occurs and TM4 is cleared.

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.

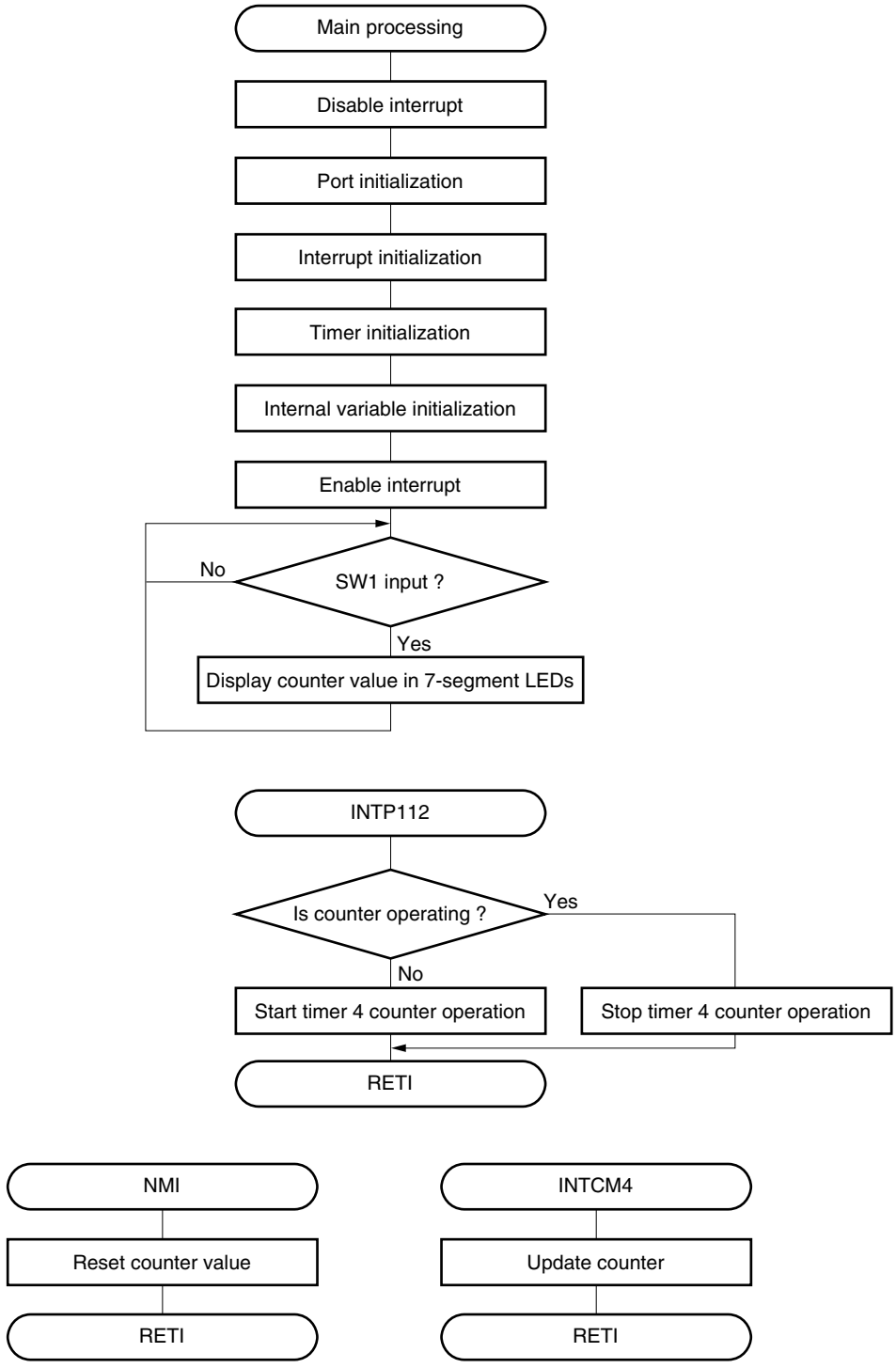
**Figure 3-36. Setting TMC4**

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| TMC4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | FFFFF342H | 01H |

Set CM4 to the value generated by INTCM4's interrupt when the count clock of $\phi$/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 stopwatch)
 *
 *    Interrupts:  NMI    :  Stop/reset
 *                 INTCM4 :  Measure elapsed time
 *                 INTP112:  Switch between counter operation/stopping
 *
 * ------------------------------------------------------------------------------
 * History:
 *    1997/3/17  Ver 0.00.00
 * File Name:    stpwatch.c
 * ---------------------------------------
 */

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

#pragma ioreg               /* Make peripheral I/O register names usable */

/* ------------------------------
 * Timer data reset processing
 * Interrupt source:  NMI
 * Arguments:  None
 * Return value:  None
 * ------------------------------
 */
#pragma interrupt NMI int_nmi
__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 interrupt INTCM4 int_cm4
__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 interrupt INTP112 int_p112
__interrupt
void int_p112(void)
{
   /* Switch counter operation mode */
   if ( tdata.exec == 1) {   /* Counter is operating */
      /* Processing to stop operation */
      CE4 = 0;               /* Stop counter */
      tdata.exec = 0;
   }
   else {                    /* Counter is stopped */
      /* Processing to start operation */
      CE4 = 1;               /* Start counter */
      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 valid 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 = 0;      /* Count clock setting φ /4 */
    PRM40 = 1;      /* Count clock setting φ /4 */
    PRM41 = 0;      /* Count clock setting φ /4 */
    CM4 = 8250 - 1; /* Comparison counter */
    CE4 = 0;        /* Put counter in stopped state */
}

/* ------------------------------
 * 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
- Clocked serial interface (CSI0 to CSI3): 4 channels

Pins are shared between the two asynchronous serial interface channels and two of the clocked 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[Note] (Using a baud rate generator, at $\phi$ = 33 MHz)<br>110 bps to 307200 bps[Note] (Using a baud rate generator, at $\phi$ = 20 MHz)<br>Maximum 1031 kbps[Note] (Using $\phi$/2, at $\phi$ = 33 MHz) |
| Full-duplex communication | On-chip receive buffer (RXBn) |
| 2-pin configuration | TXDn: Transmit data output pin<br>RXDn: Receive data input pin |
| Receive error detection function | • Parity error<br>• Framing error<br>• Overrun error |
| 3 kinds of interrupt source | • Receive error interrupt (INTSERn)<br>• Receive completion interrupt (INTSRn)<br>• Transmit completion interrupt (INTSTn) |
| Character length (specified by ASIM register) | 7 or 8 bits<br>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 | — |

★ **Note** Refer to **Table 3-8 Baud Rate Generators 0 to 2 Setting Data** for details of the baud rate error.

**Remark** n = 0, 1

$\phi$: 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<br>Specifies transmit and receive enable status or disable status.<br><br>TXEn / RXEn / Operation:<br>0 / 0 / Disable transmit (Select CSIn)<br>0 / 1 / Enable receive<br>1 / 0 / Enable transmit<br>1 / 1 / Enable transmit and receive<br><br>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.<br>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.<br>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<br>Specifies the parity bit.<br><br>PSn1 / PSN0 / Operation:<br>0 / 0 / No parity, extended bit operation<br>0 / 1 / Specifies 0 parity<br>Transmitting end → Transmit making parity bit 0<br>Receiving end → Do not generate parity error when receiving<br>1 / 0 / Specifies odd parity<br>1 / 1 / Specifies even parity |
| 3 | CLn | Character Length<br>Specifies character length of 1 frame.<br>0: 7 bits<br>1: 8 bits |
| 2 | SLn | Stop Bit Length<br>Specifies stop bit.<br>0: 1 bit<br>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<br>Specifies the serial clock.<br>0: Specify using BRGCn, BPRMn<br>1: $\phi$/2<br><br>When SCLSn = 1<br>Select $\phi$/2 (system clock) as serial clock source.<br>Because sixteen-fold sampling rate is used in asynchronous mode, the baud rate is represented by the following formula.<br><br>Baud rate = $\frac{\phi/2}{16}$ bps<br><br>Based on the formula above, the baud rate values when typical clocks are used are shown below.<br><br>$\phi$ \| 33MHz \| 25MHz \| 20MHz \| 16MHz \| 12.5MHz \| 10MHz \| 8MHz \| 5MHz<br>Baud rate \| 1031K \| 781K \| 625K \| 500K \| 390K \| 312K \| 250K \| 156K<br><br>When SCLSn = 0<br>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

$\phi$: Internal system clock

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

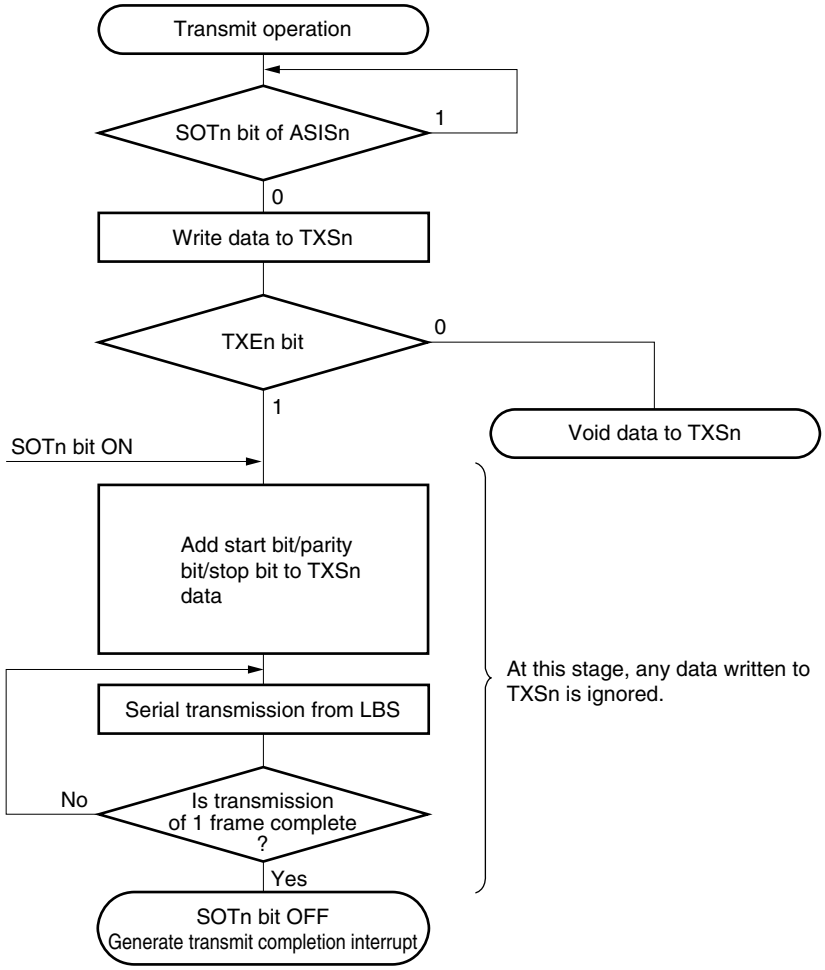
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Address | After reset |
|---|---|---|---|---|---|---|---|---|---|---|
| ASIM01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | EBS0 | FFFFF0C2H | 00H |
| ASIM11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | EBS1 | FFFFF0D2H | 00H |

| Bit position | Bit name | Description |
|---|---|---|
| 0 | EBSn | Extended Bit Select<br>Specifies extended bit operation for transmit and receive data when no-parity operation is specified (PSn1, PSn0 = 00).<br>0: Disable extended bit operation<br>1: Enable extended bit operation<br>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.<br>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

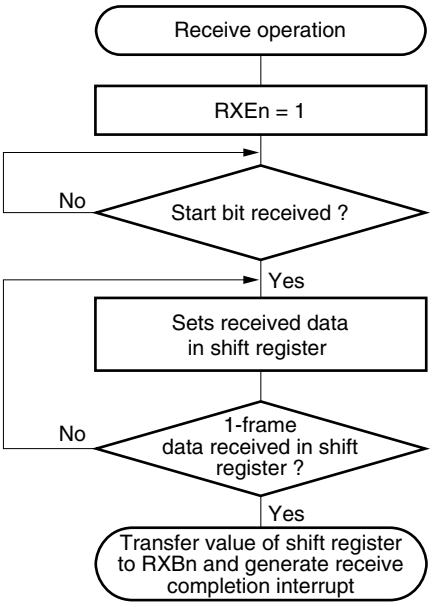
A transmit completion interrupt occurs when one frame of data has been sent from the transmit shift register.

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 clocked 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 clocked 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<br>This is a status flag that shows the transmit operation status.<br>Set (1): Transmission start timing (writing to TXSn or TXSnL registers)<br>Clear (0): Transmission end timing (generating INTSTn interrupt)<br>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<br>This is a status flag that indicates a parity error.<br>Set (1): Transmit parity and receive parity do not match<br>Clear (0): Data read processing from receive buffer |
| 1 | FEn | Framing Error<br>This is a status flag that indicates a framing error.<br>Set (1): Stop bit was not detected<br>Clear (0): Data read processing from receive buffer |
| 0 | OVEn | Overrun Error<br>This is a status flag that indicates an overrun error.<br>Set (1): UARTn completed next receive processing before received data was fetched from receive buffer<br>Clear (0): Received data read processing from receive buffer<br>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.

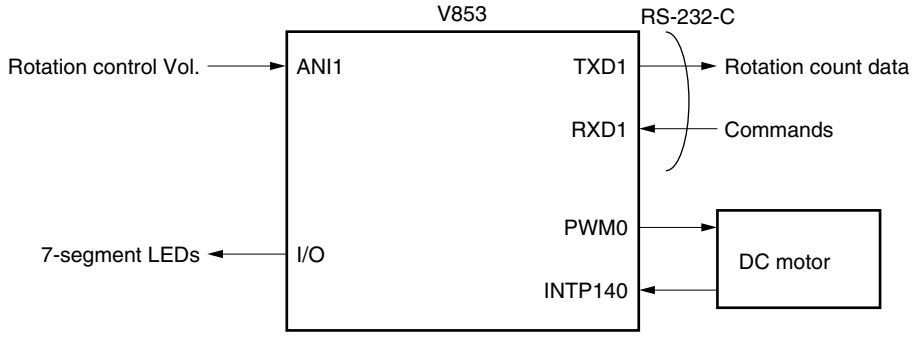
The status registers (ASISn) are reset when the receive buffer (RXBn, RXBnL) is read**Note** or the next data of 1 frame is received (n = 0, 1).

**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**



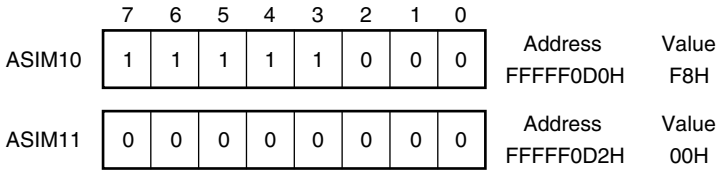

**(1) Function overview**

- 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 = $\phi$/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)**

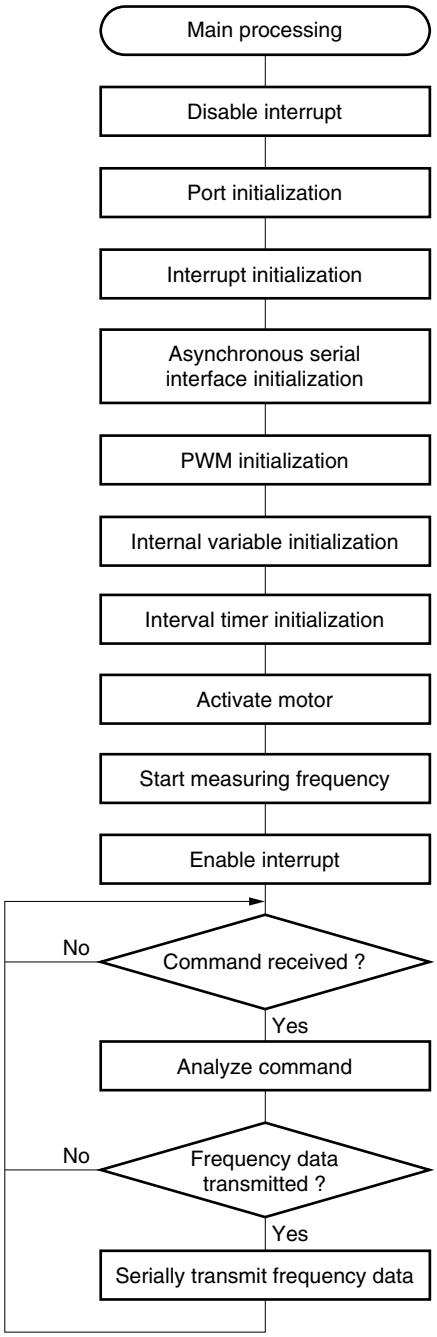
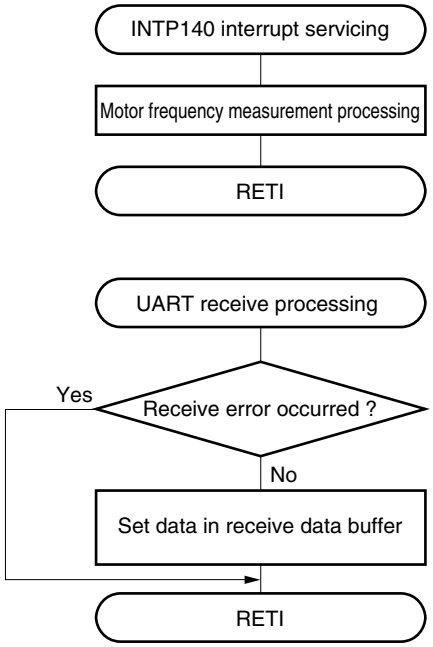


**Figure 3-45. DC Motor Rotations Monitor (2/2)**



### (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
 * File Name     dcmoter.c
 * ------------------------------------------------
 */

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

#pragma ioreg          /* Make peripheral I/O register names usable */

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

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

/* ------------------------------------------------------------------------------
 * Asynchronous serial interface:  Receive error interrupt servicing
 * Interrupt source:  INTSER1
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------------------------------
 */
#pragma interrupt INTSER1 int_ser1
__interrupt
void Int_ser1(void)
{
   unsigned   short   tmp;  /* Work variable */
   /* Set error flag */
   uartdt.err_flg = ASIS1 & 87;
   /* Clear error flag by reading */
   tmp = RXB1;
}

/* ------------------------------------------------------------------
```

```
 * Asynchronous serial interface:  Receive completion interrupt servicing
 * Interrupt source:  INTSR1
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------------------------------
 */
#pragma interrupt INTSR1 int_sr1
__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 servicing
 * Interrupt source:  INTST1
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------------------------------
 */
#pragma interrupt INTST1 int_st1
__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 interrupt INTP140 int_p140
__interrupt
void int_p140(void)
{
   pwmdt.int_cnt ++;
}

/* ------------------------------------
 * Setting elapsed time
 * Interrupt source:  INTCM4
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTCM4 int_cm4
__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 ( CS == 0 ) {      /* Start conversion when converter is stopped */
      CE = 1;            /* Start conversion */
   }
}

/* ------------------------------------
 * A/D conversion completion interrupt
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTAD int_ad
__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 */
                  /* P21 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 valid 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 ϕ/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 ϕ/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 */
   CE4 = 1;

   /* Start A/D conversion */
   CE = 1;

   /* Start PWM output */
   PWME0 = 1;

   /* Enable interrupt */
   __EI();
}

/* ------------------------------------
 * 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();

   /* 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 */
               sendpwmct();
            }
            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:    dcmoter.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 Clocked serial interface (CSI0 to CSI3)

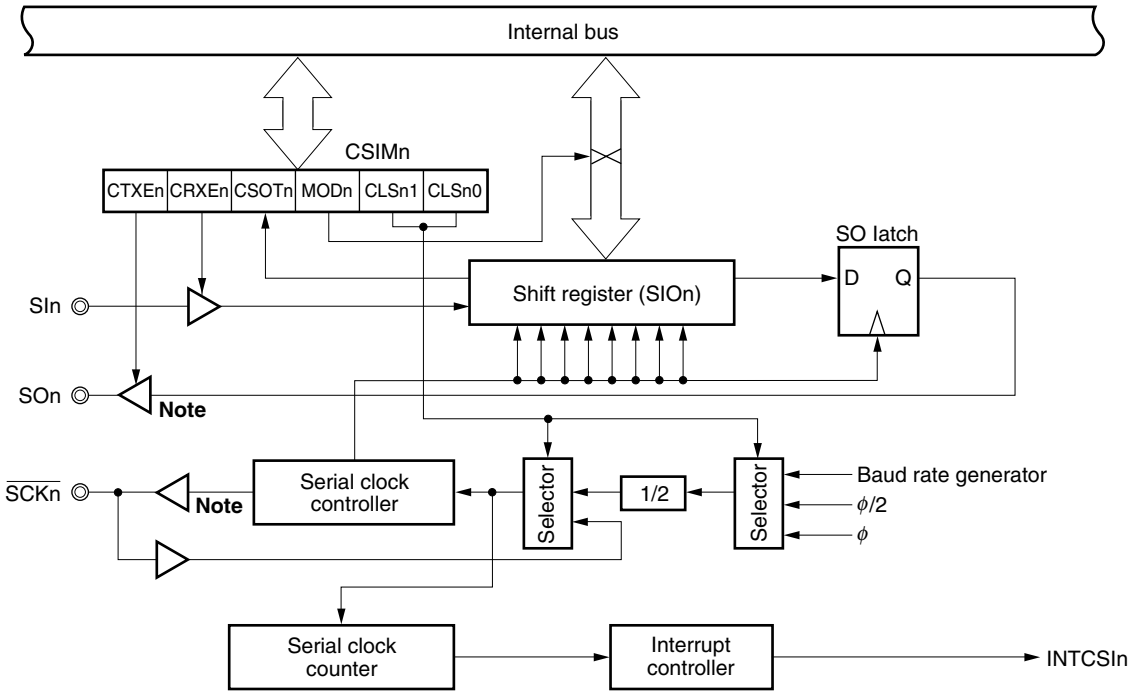
The clocked serial interface performs data transmission/reception by using three lines: one clock line and two data lines. Features of the clocked serial interface are shown below.

- Transfer speed: 8.25 Mbps max. (at $\phi$ = 33 MHz operation: CSI0 to CSI2)
★ 2 Mbps max. (at $\phi$ = 33 MHz operation: CSI3)
- 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
$\phi$: Internal system clock

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

**Figure 3-46. Block Diagram of Clocked 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 clocked serial mode registers (CSIM0 to CSIM3) that make each clocked serial interface setting.

**Figure 3-47. Clocked 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<br>Specifies transmit enabled/disabled status.<br>0: Transmit disabled status<br>1: Transmit enabled status<br>When CTXEn = 0, SOn and SIn pin output buffers are high impedance. |
| 6 | CRXEn | CSI Receive Enable<br>Specifies receive enabled/disabled status.<br>0: Receive disabled status<br>1: Receive enabled status<br>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<br>Indicates transmission operation in progress.<br>Set (1): Transmit start timing (write to SIOn register)<br>Clear (0): Transmit end timing (INTCSIn generation)<br>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<br>Specifies operating mode.<br>0: MSB first<br>1: LSB first |
| 1, 0 | CLSn1, CLSn0 | Clock Source<br>Specifies serial clock. |

| CLSn1 | CLSn0 | Serial clock specification | | $\overline{\text{SCKn}}$ pin |
|---|---|---|---|---|
| 0 | 0 | External clock | | Input |
| 0 | 1 | Internal clock | Specify in BPRMn register[Note 1] | Output |
| 1 | 0 | | $\phi/4$[Note 2] | Output |
| 1 | 1 | | $\phi/2$[Note 2] | Output |

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

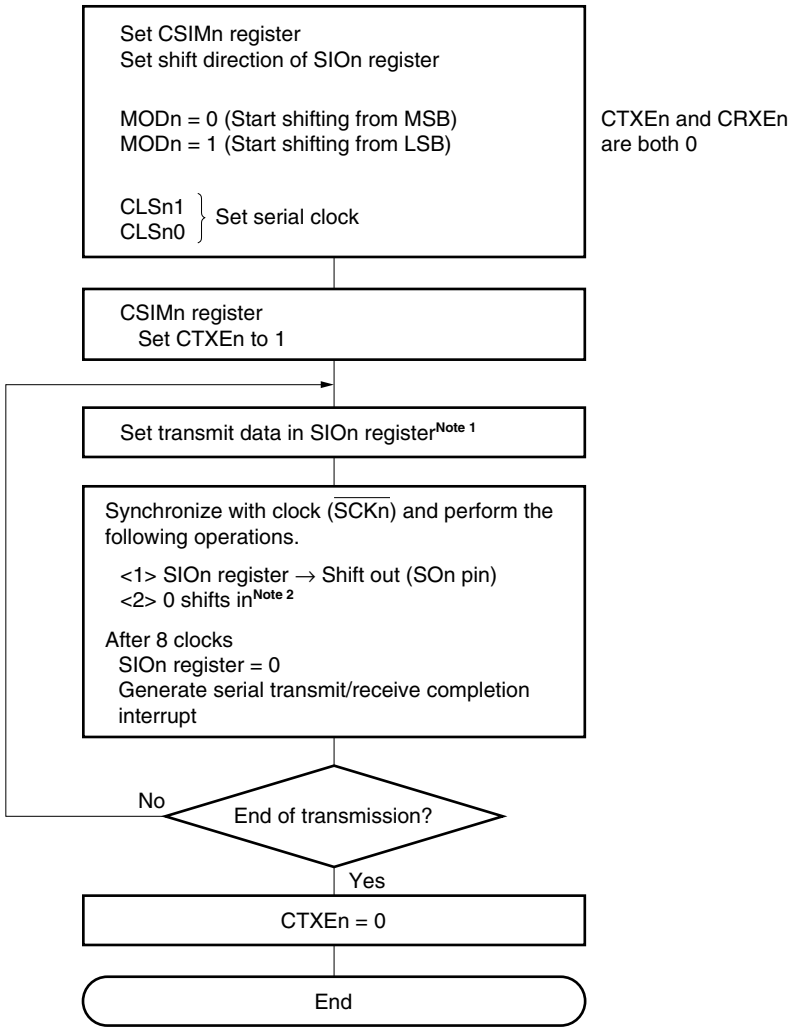
**2.** $\phi/4$ and $\phi/2$ are division signals ($\phi$ : Internal system clock).

**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 clocked serial interface transmit operation.

**Figure 3-48. Clocked Serial Interface Transmit Operation**



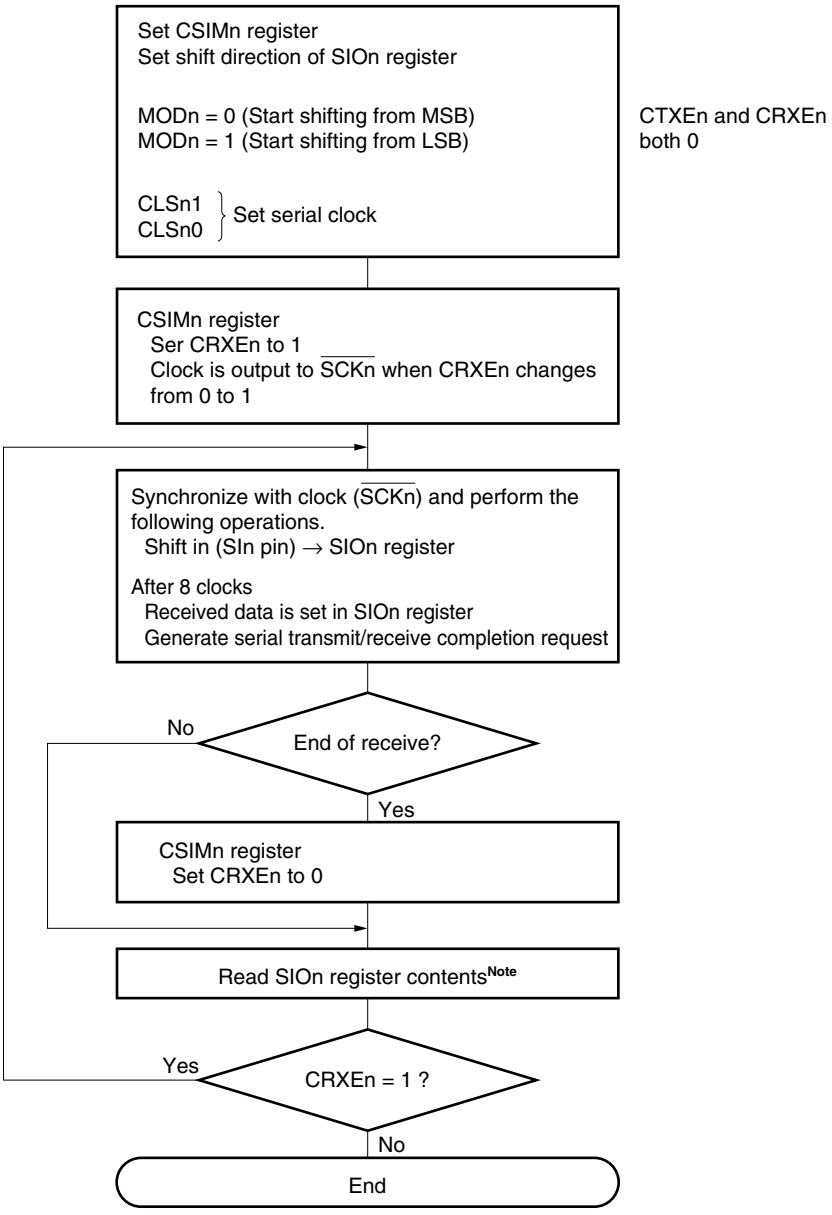

**Notes 1.** If CTXEn = 1, the clock is output to $\overline{\text{SCKn}}$ when data is written to the SIOn register.

**2.** If CRXEn = 0, input to the shift register is 0.

**Remark** n = 0 to 3

Figure 3-49 shows clocked serial interface receive operation.

**Figure 3-49. Clocked Serial Interface Receive Operation**

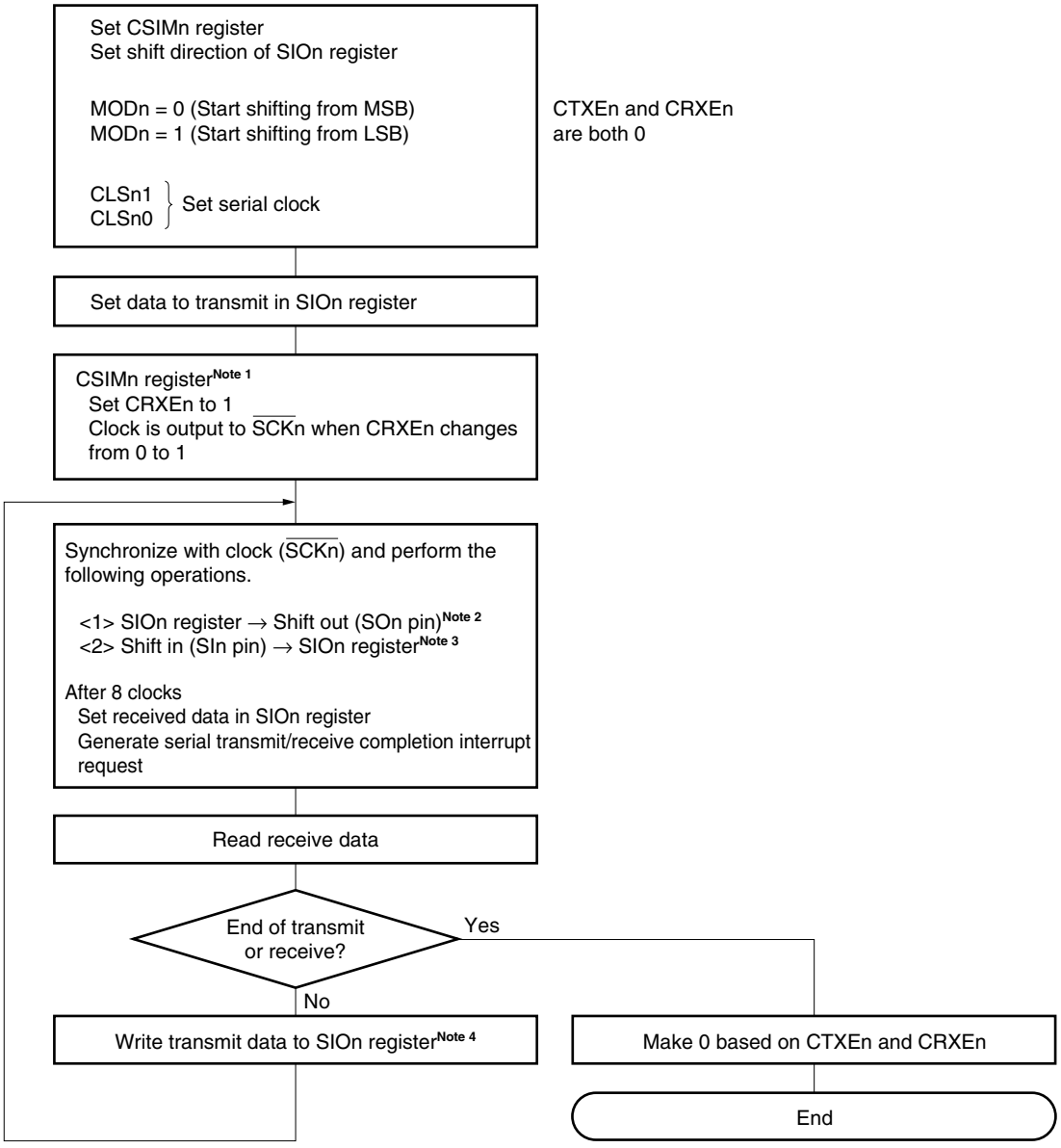



**Note** If CRXEn = 1, the clock is output to $\overline{\text{SCKn}}$ when data in the SIOn register is read.

**Remark** n = 0 to 3

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

**Figure 3-50. Clocked 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

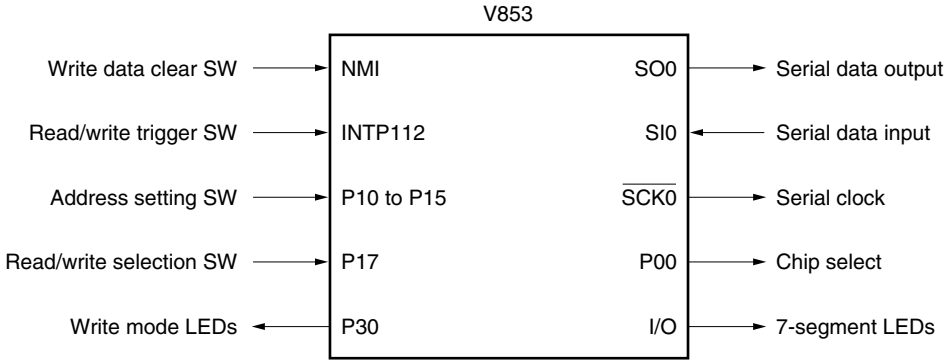
**3.** The operation <2> synchronizes with rising clock

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

**Remark** n = 0 to 3

### 3.4.4 Clocked serial interface sample program

**Figure 3-51. Sample Hardware Configuration**




**(1) Function overview**

- Read/write data for EEPROM™ connected to clocked 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.

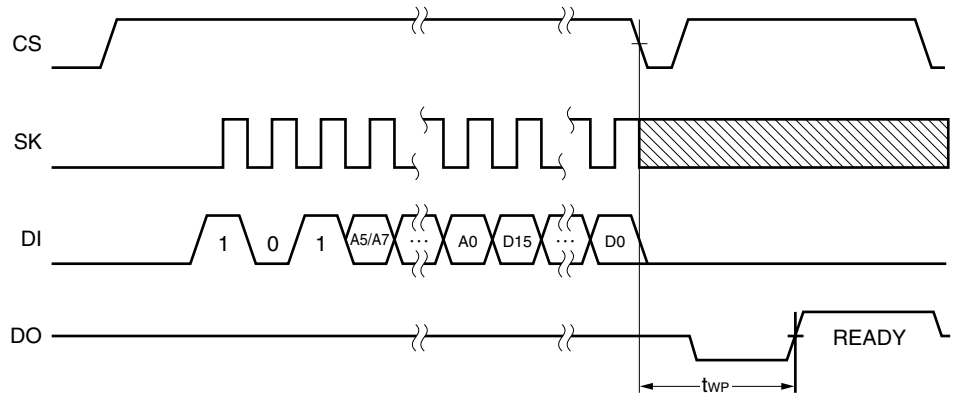
Next, a program is created to read/write data to a serial type EEPROM connected to the clocked serial interface using the sample hardware.
The functions of the EEPROM connected to the sample hardware are shown below.

- Read: Read data
- Write: Write data
- Erase: Erase data
- Erase/write enable: Enable erase/write operation
- Erase/write disable: Disable erase/write operation

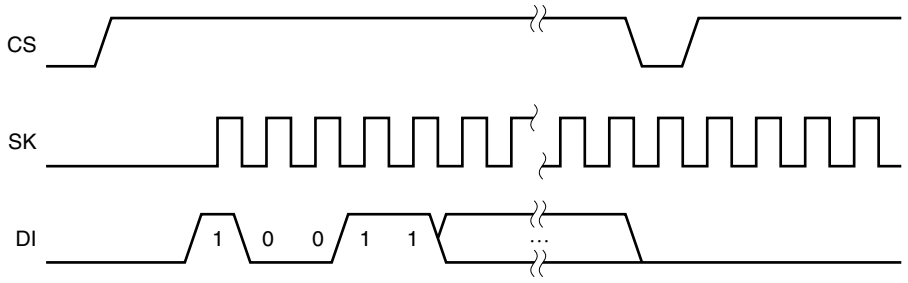
Figure 3-52 shows the timing of EEPROM operations.

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

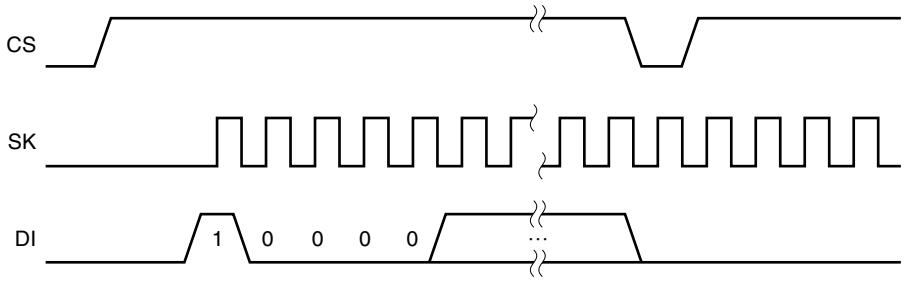
**(a) Write**




**(b) Erase/write enable**



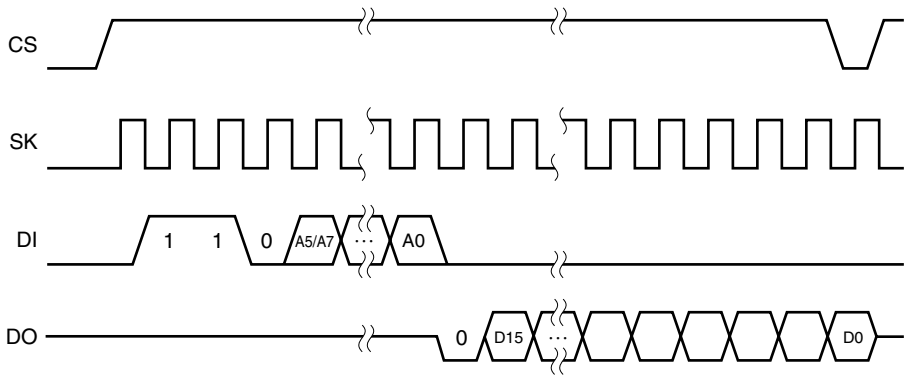

**(c) Erase/write disable**




**Remark** 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**




**Remark** 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 clocked serial interface mode register 0 (CSIM0) according to these conditions.

**Figure 3-53. Clocked 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 = $\phi$/2)
BRGC1 = 107 (operation clock = 33 MHz)
= 81 (operation clock = 25 MHz)

As for enabling receive and transmit in the clocked 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 erase→ Enable transmit
- Erase/write enable → Enable transmit
- Erase/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)**

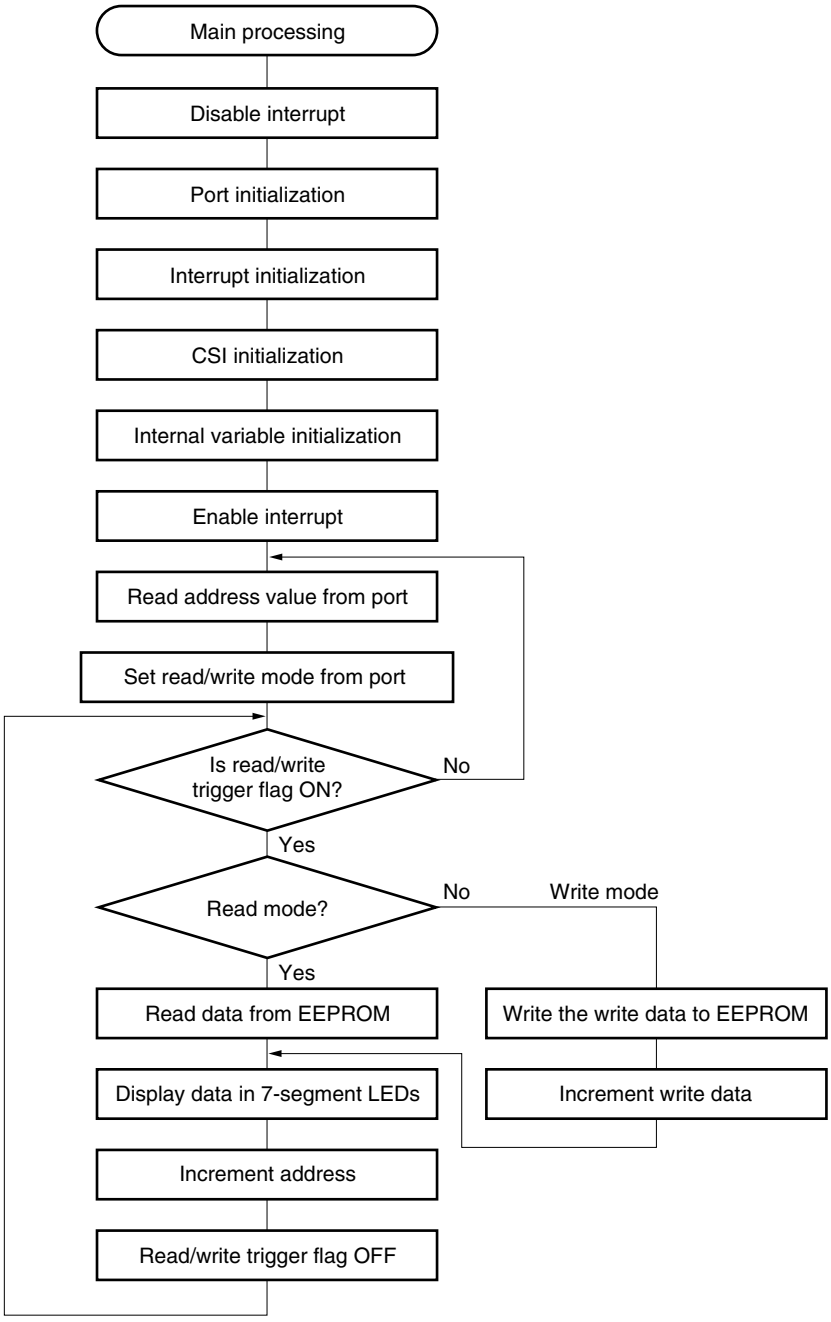
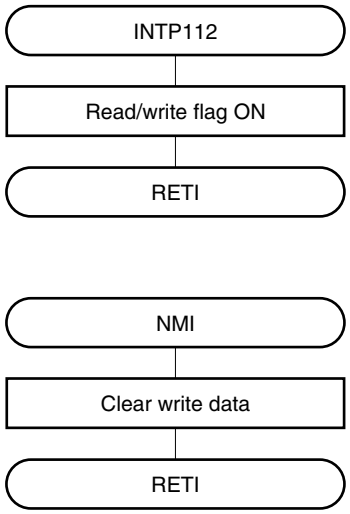


**Figure 3-54. Flow of Sample Program to Read/Write to EEPROM (2/2)**



### (2) Sample program

```
/* ----------------------------------------------------------------------
 * V853 clocked 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
 * File Name:     eeprom.c
 * -----------------------------------------------------
 */

#pragma ioreg     /* Make peripheral I/O register names usable */

#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 interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   writeData = 1;
}
```

```
/* ------------------------------------
 * Clocked serial communication end
 * Interrupt source:  INTCSI
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTCSI0 int_csi0
__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 interrupt INTP112 int_p112
__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 valid 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 */
```

```
   /* Clocked 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: 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/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 */
   P0.0 = 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 */
   P0.0 = 0;
}

/* -------------------------------------
 * Write disable processing
 * Arguments:  None
 * Return value:  None
 * -------------------------------------
 */
void WriteDisable(void)
{
   /* Chip select ON */
   P0.0 = 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 */
   P0.0 = 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 */
   P0.0 = 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 */
   P0.0 = 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 */
    P0.0 = 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 */
    P0.0 = 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 = P1.7;
      if (WriteMode) {
         /* Write LED lit → Write mode */
         P3.0 = 1;
      } else {
         /* Write LED out → Read mode */
         P3.0 = 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 ($\phi$)

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**

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

$\phi$ = Internal system clock frequency (Hz)
j = Timer count value ($1 \leq j \leq 256$[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**

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

$\phi$ = Internal system clock frequency (Hz)
j = Timer count value ($1 \leq j \leq 256$[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] | | $\phi$ = 33 MHz | | | $\phi$ = 25 MHz | | | $\phi$ = 16 MHz | | | $\phi$ = 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] | | $\phi$ = 20 MHz | | | $\phi$ = 14.746 MHz | | | $\phi$ = 12.288 MHz | | | $\phi$ = 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)
$\phi$: Internal system clock

**(3) Baud rate error**

The baud generator error is represented as follows.

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

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

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

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.

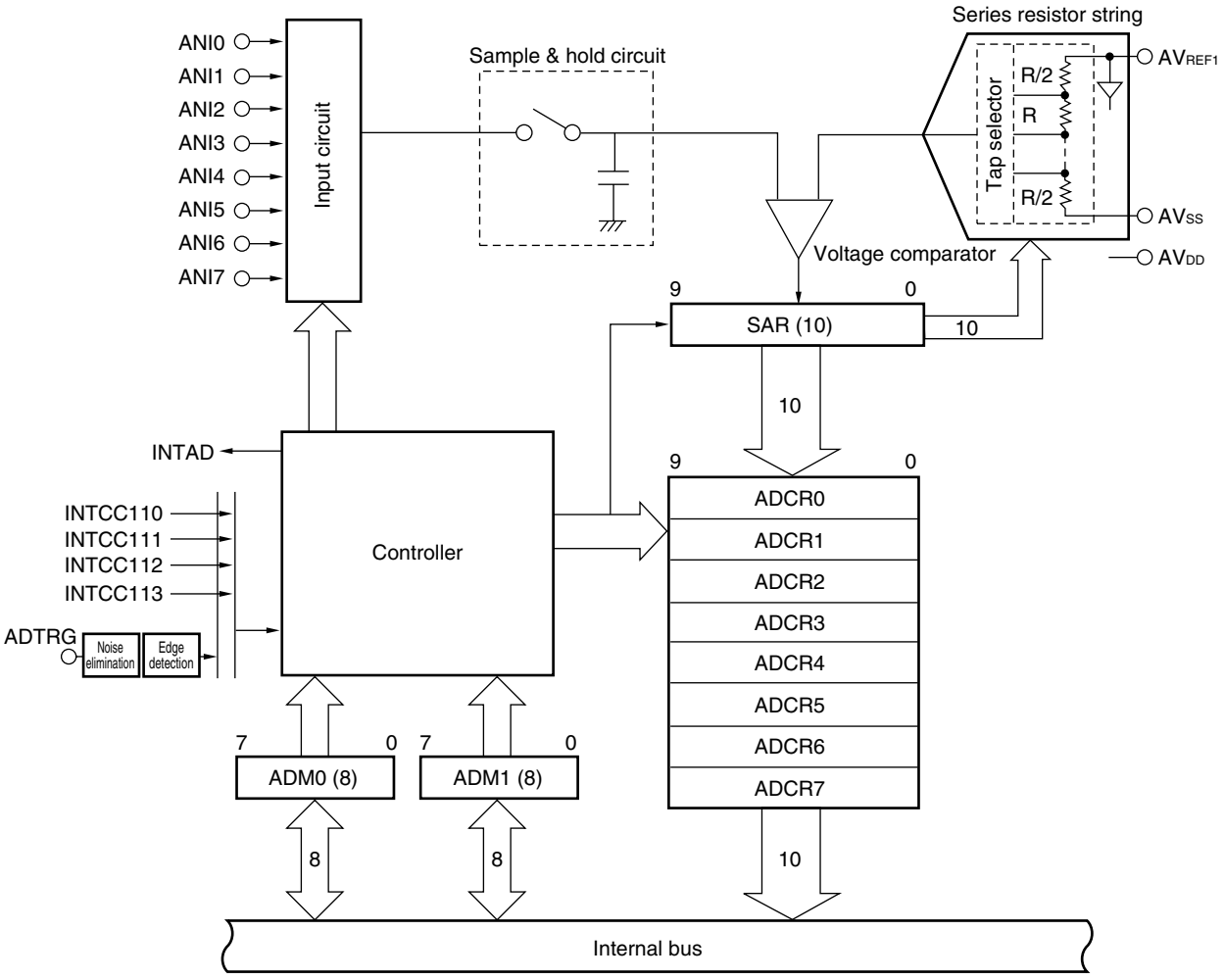
## 3.5 A/D Converter Function

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 | Valid 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**

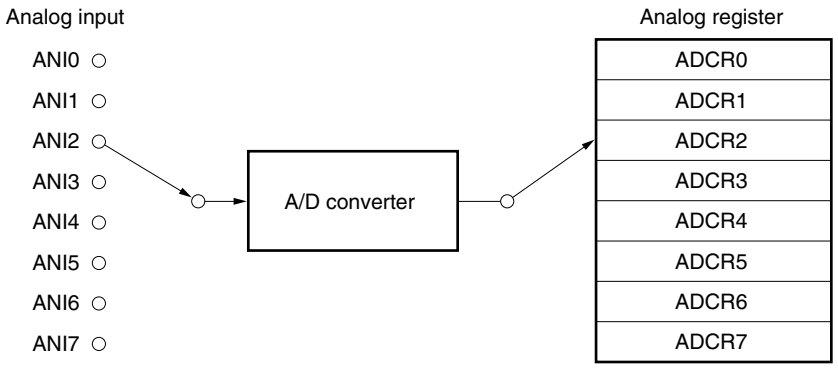


**Figure 3-57. Four-Buffer Mode**

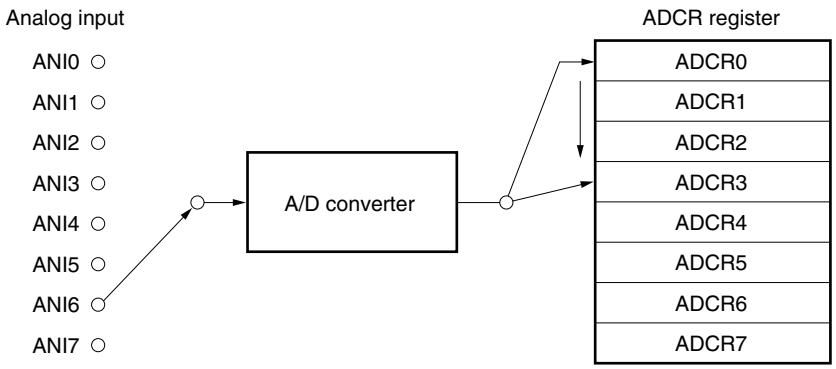
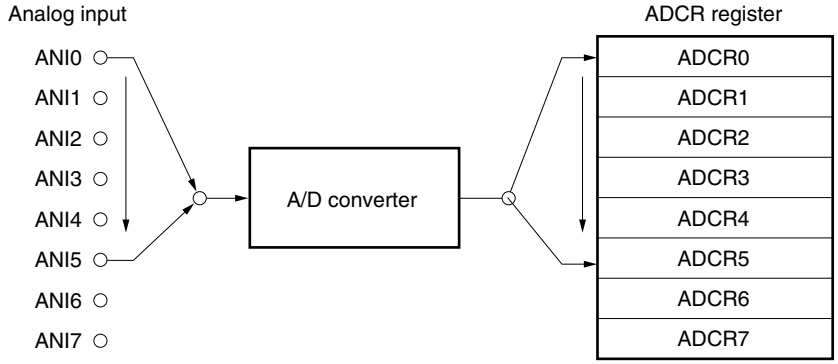



**Figure 3-58. Scan Mode**



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<br>Specifies A/D conversion operation enabled/disabled.<br>0: Disabled<br>1: Enabled |
| 6 | CS | Convert Status<br>Shows status of A/D converter. This bit is read-only.<br>0: Stopped<br>1: Operating |
| 5 | BS | Buffer Select<br>Specifies buffer mode in select mode.<br>0: 1-buffer mode<br>1: 4-buffer mode |
| 4 | MS | Mode Select<br>Specifies operating mode of A/D converter.<br>0: Scan mode<br>1: Select mode |
| 2 to 0 | ANIS2 to ANIS0 | Analog Input Select<br>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.

**Cautions 1. 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.**

★ **2. It takes 3 clocks from the start of A/D conversion to when the CS bit is set to 1. Use the A/D conversion completion interrupt (INTAD) to read the A/D conversion value.**

**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<br>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 valid 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)**.)

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 ($\mu$s) | | |
|---|---|---|---|---|---|---|
| | | | | $\phi$ = 33 MHz | $\phi$ = 25 MHz | $\phi$ = 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 |

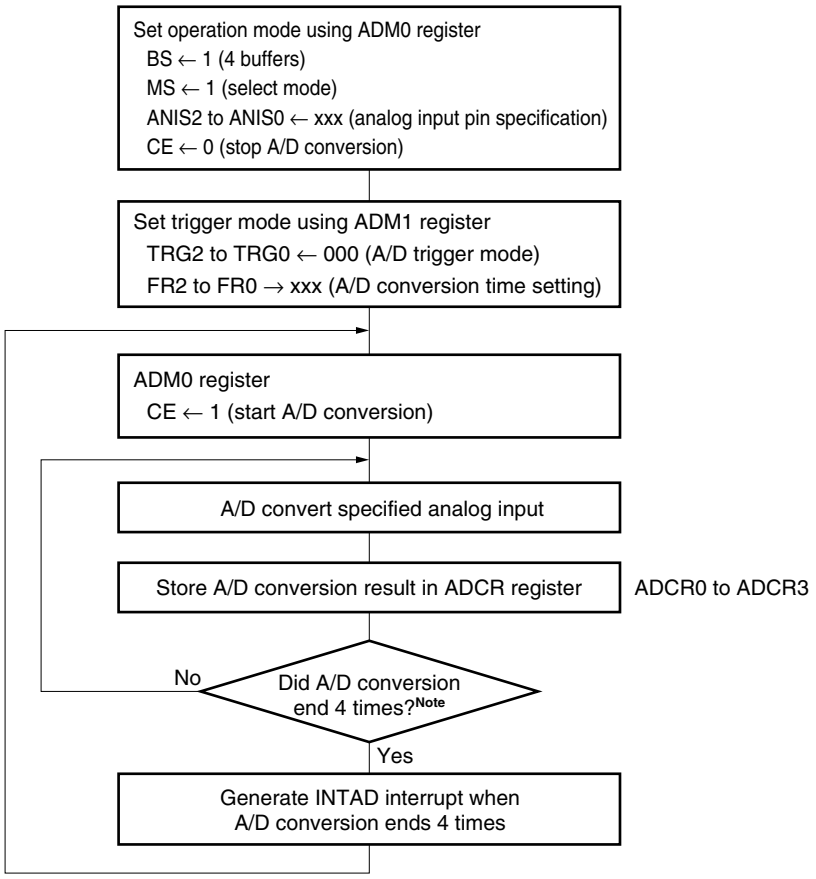
**Remarks 1.** The conversion operating time is a target value.
**2.** $\phi$ = internal system clock frequency

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**

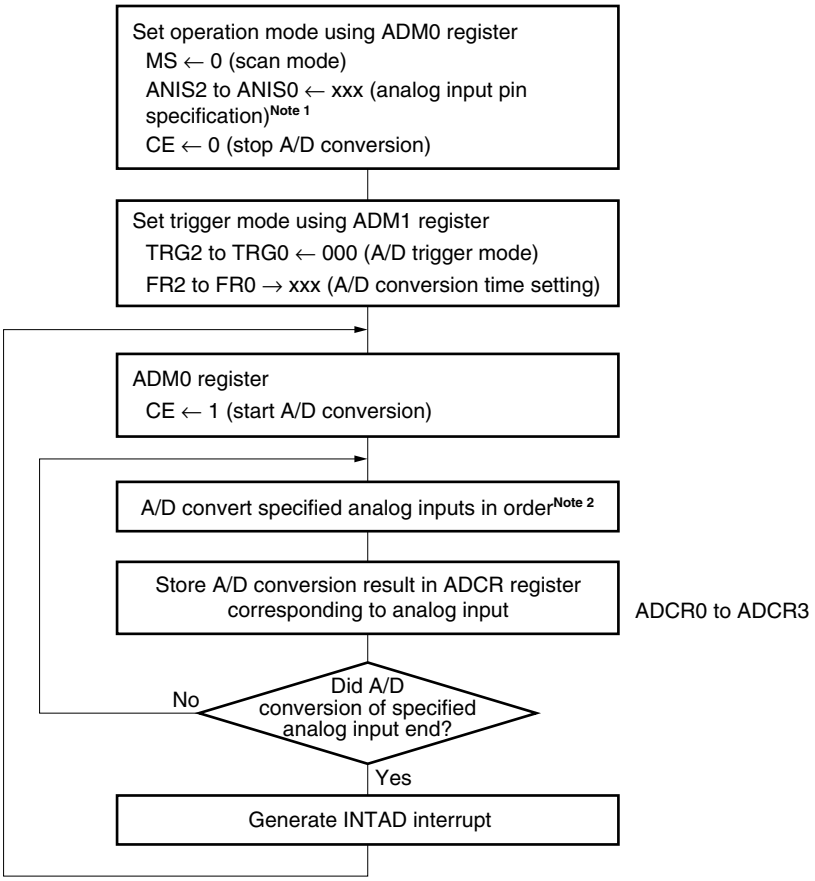



**Note** In one-buffer mode, an INTAD interrupt occurs at A/D conversion end.

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**

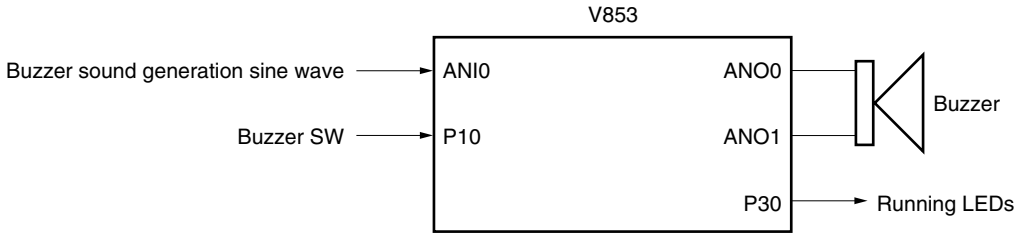



**Notes 1.** Multiple analog input pins can be specified for conversion.

**2.** A/D convert from ANI0 to 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.

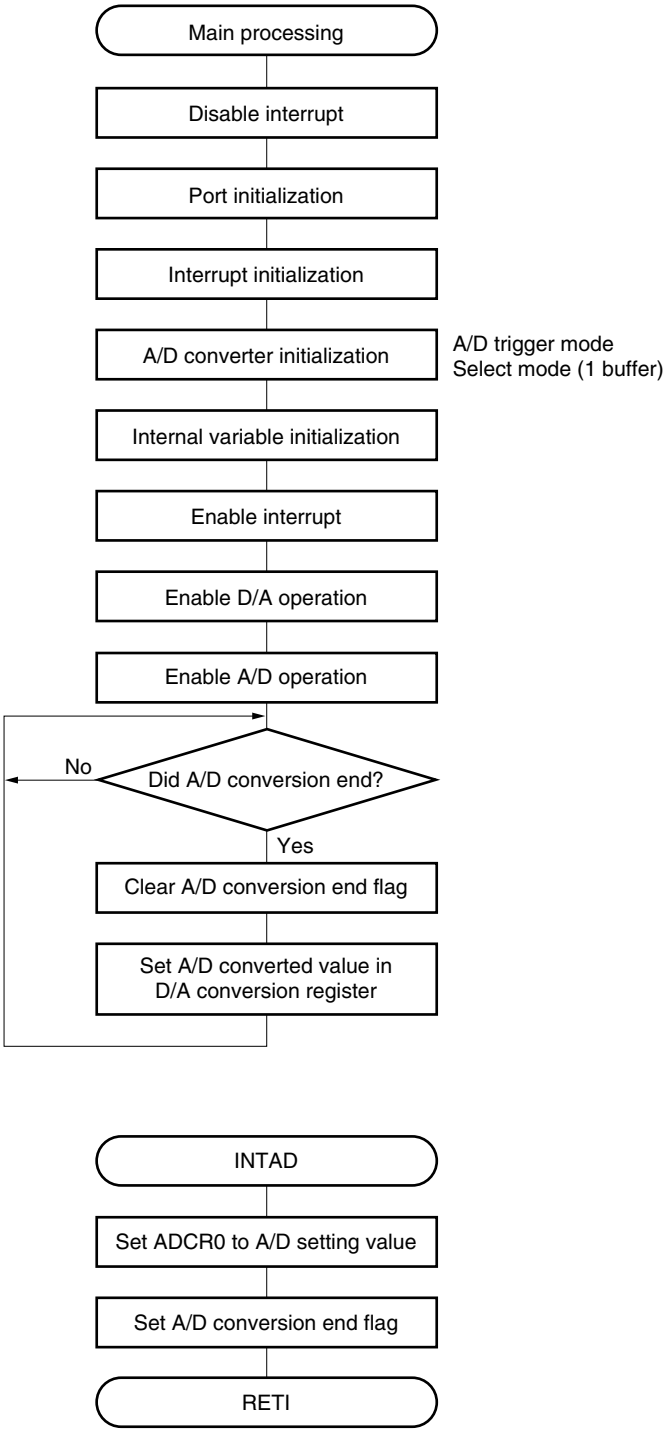
<1> Set the A/D converter trigger mode to A/D trigger mode.
<2> Set the operating mode to select mode (one-buffer mode).
<3> Specify ANI0 for the analog input pin.
<4> Specify 192 for the number of conversion clocks.

**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
 * File Name:    buzzer.c
 * ------------------------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */

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

/* ------------------------------------------------------------
 * A/D converter conversion end */
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------------
 */
#pragma interrupt INTAD int_ad
__interrupt
void int_ad(void)
{
   /* Assign conversion results */
   BuzzerDat = ADCR0H;  /* ADCR0 is 10-bit data. */
                        /* Since higher 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 valid 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
 * ------------------------------------------------------------
 */
void main(void)
{
    short  StartFlag = 0;  /* Execution flag: Non-zero = executing */

    /* Disable interrupt */
    __DI();

    /* Port initialization processing */
    PortInit();

    /* 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;
    DACE0 = 1;    /* Enable output operation */
    DACE1 = 1;

    /* Start A/D conversion */
    CE = 1;

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

```
    /* Clear LEDs */
    P3 = 0;

    while ( 1 ) {
       /* Execution switch ON */
       if ( P1.0 != 0 ) {
          /* A/D conversion completion */
          if ( AdFlag != 0 ) {
             /* Set A/D converted value in D/A register */
             DACS0 = BuzzerDat;
             DACS1 = BuzzerDat;
             AdFlag = 0;
             CE = 1;       /* Start A/D conversion */
          }
          if ( StartFlag == 0 ) {
             /* LEDs ON */
             P3.0 = 1;
             StartFlag = 1;
          }
       }
       /* Execution switch OFF */
       else {
          if ( StartFlag == 1 ) {
             /* Clear LEDs */
             P3.0 = 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**

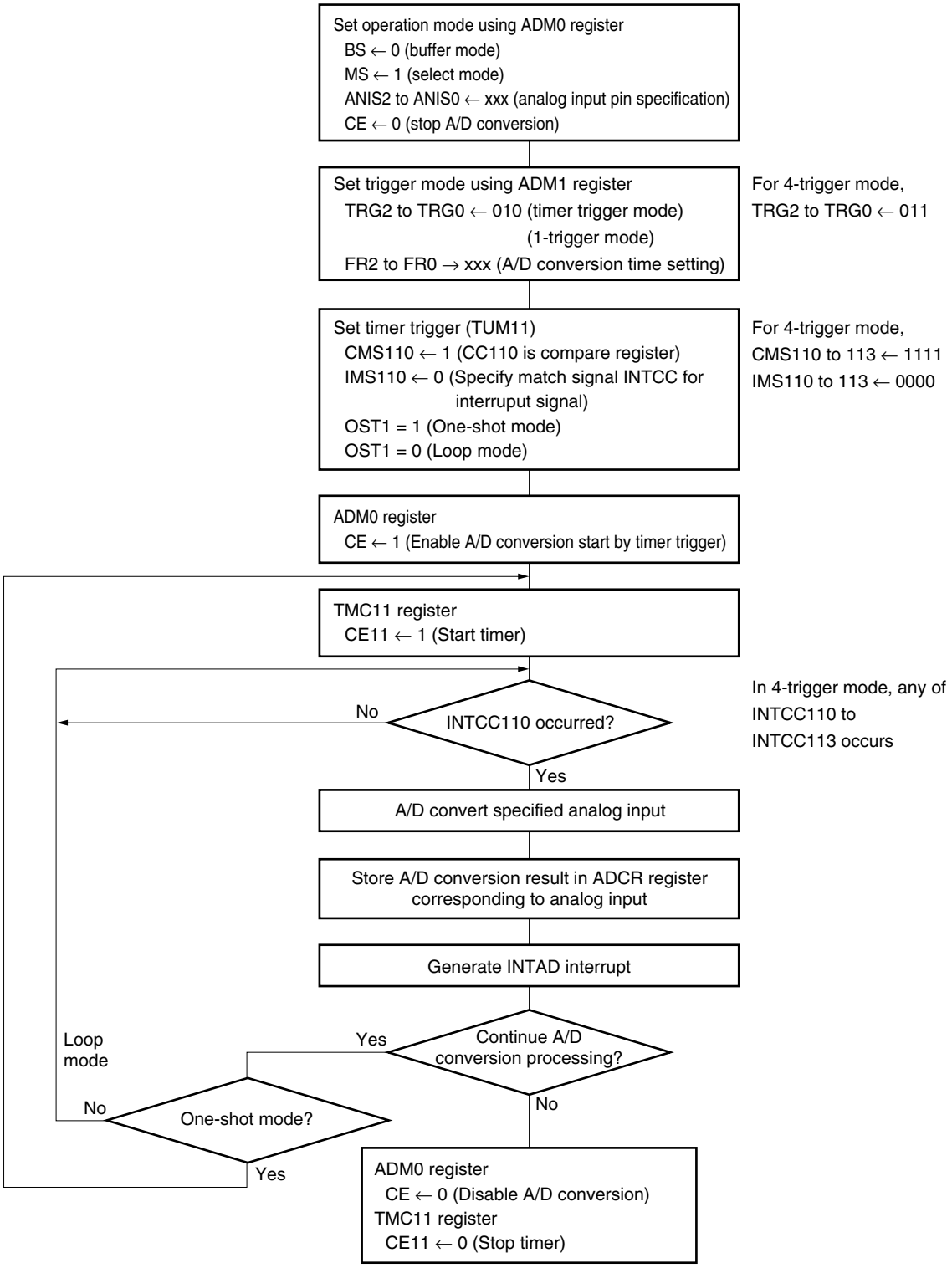
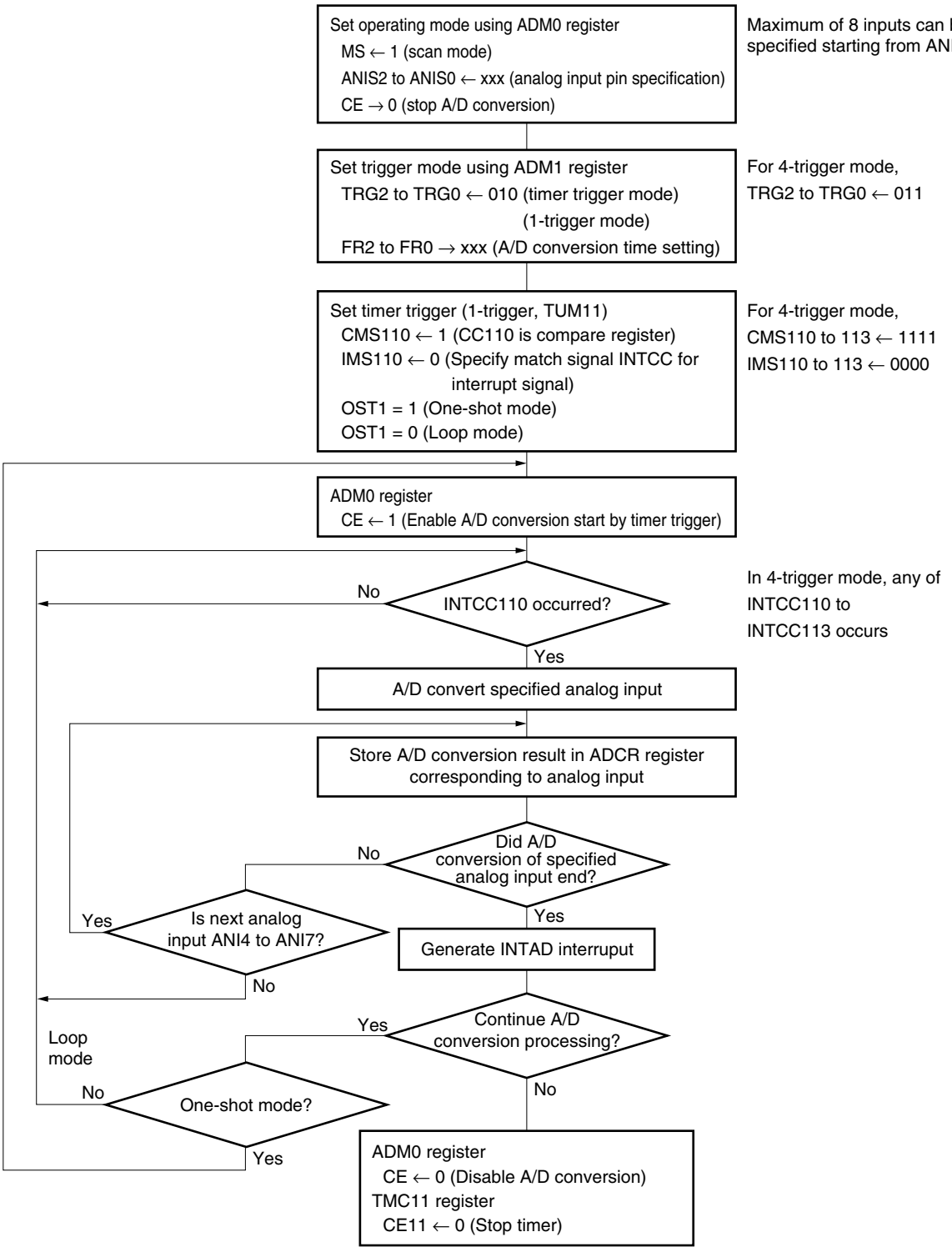
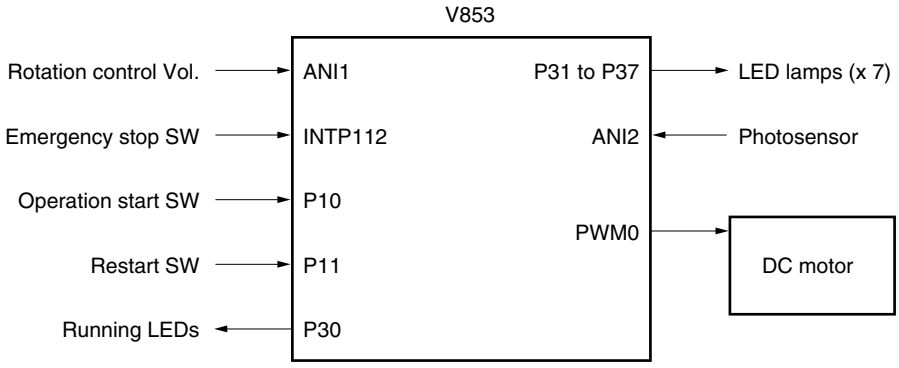


**Figure 3-67. Timer Trigger Mode + Scan Mode Settings and Operation**



### 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**

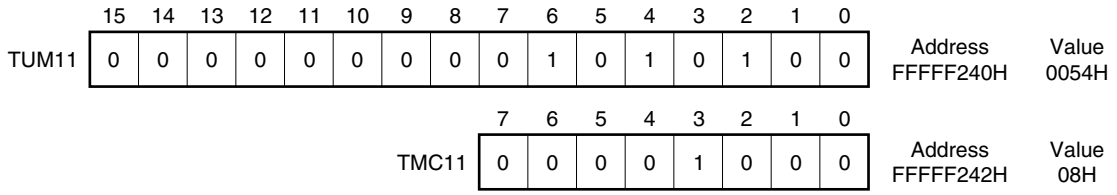
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ADM0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | Address FFFFF380H | Value 03H |
| ADM1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | Address FFFFF382H | Value 27H |

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 valid edge of interrupt). Also select INTP112 as an interrupt source.
<5> Use an internal count clock.

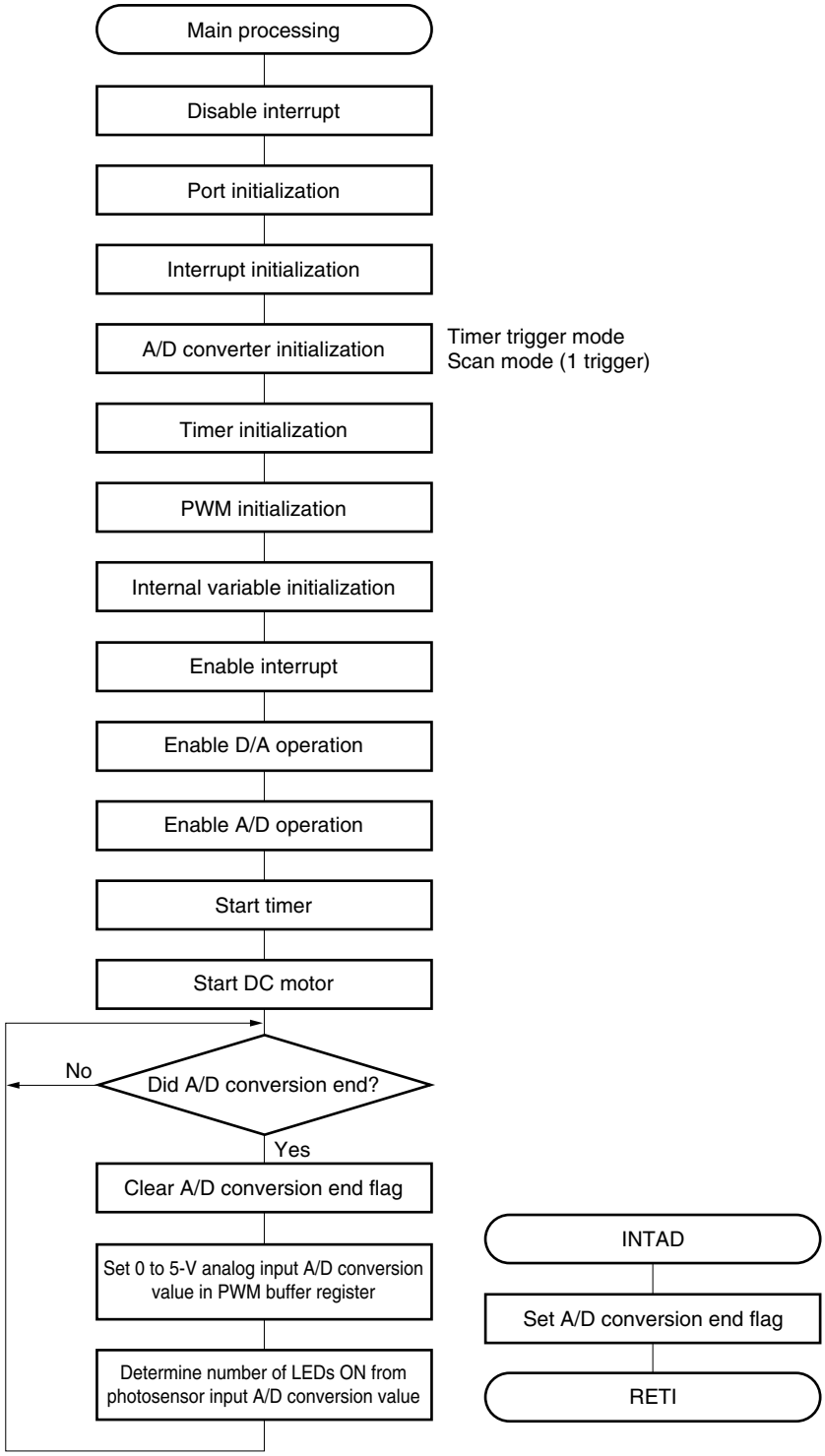
Set timer unit mode register 11 (TUM11) and timer control register 11 (TMC11) using the above conditions.

**Figure 3-70. TUM11 and TMC11 Settings**



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
 * File Name:    adtimtrg.c
 * ----------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */

/* ------------------------------------------------------------
 * 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 interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   ;      /* Do nothing */
}

/* ----------------------------------------------
 * Motor emergency stop processing
 * Interrupt source: INTP112
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------------
 */
#pragma interrupt INTP112 int_p112
```

```
__interrupt
void int_p112(void)
{
   /* Process only while executing */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM operation */
   PWM0 = 0;
   PWME0 = 0;
}

/* ------------------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
#pragma interrupt INTAD int_ad
__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 valid 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 ϕ m/8 */
                   /* PRS110 = 0: (Timer count clock = ϕ/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 ϕ/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 valid 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;
    DACE0 = 1;
    DACE1 = 1;

    /* Start A/D conversion */
    CE = 1;

    /* Start timer */
    CE11 = 1;

    /* Start PWM output */
    PWME0 = 1;

    /* Enable interrupt */
    __EI();
}

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

    /* Stop D/A conversion */
    DACE0 = 0;
    DACE1 = 0;

    /* Stop A/D conversion */
    CE = 0;

    /* Stop timer */
    CE11 = 0;

    /* PWM output initialization */
    PWM0 = 0;
```

```
    /* Stop PWM output */
    PWME0 = 0;
}

/* ------------------------------------------
 * Main processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
void main(void)
{
    int    pulsFlg = 0;   /* Display pulse option flag */
    char   bzwk;

    /* 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 ((P1.0 != 0) && (StartFlg == 0)) {
            StartProc();
            StartFlg = 1;
        }
        /* Stop until start flag is ON */
        if (StartFlg == 0) {
            StopProc();
            continue;
        }
        /* Check for start switch OFF */
        if ((P1.0 == 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 ( P1.1 == 0 ) {
                continue;
```

```
            }
            /* Clear emergency stop flag */
            EmergencyFlg = 0;
            /* Start PWM operation */
            PWME0 = 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**

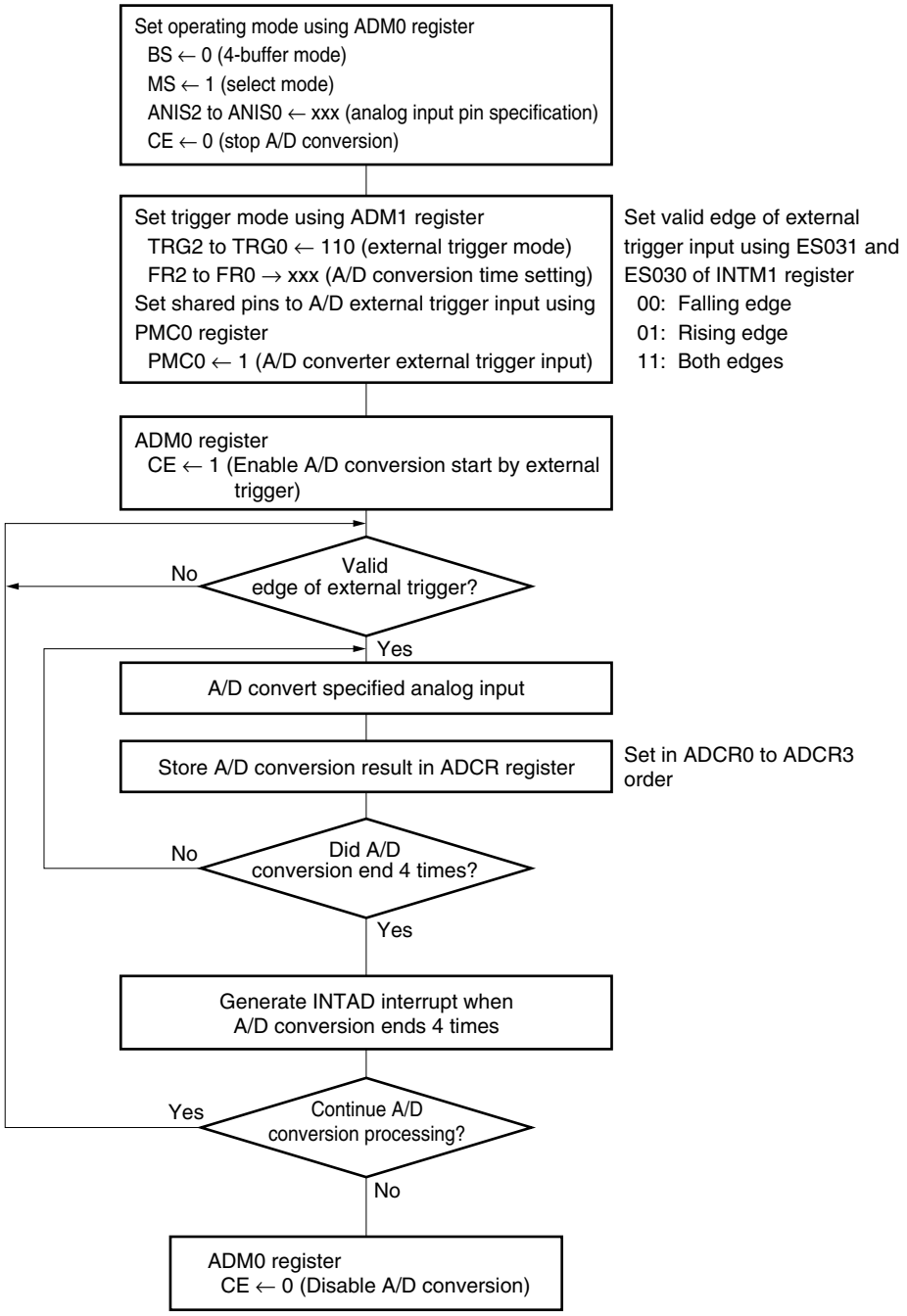
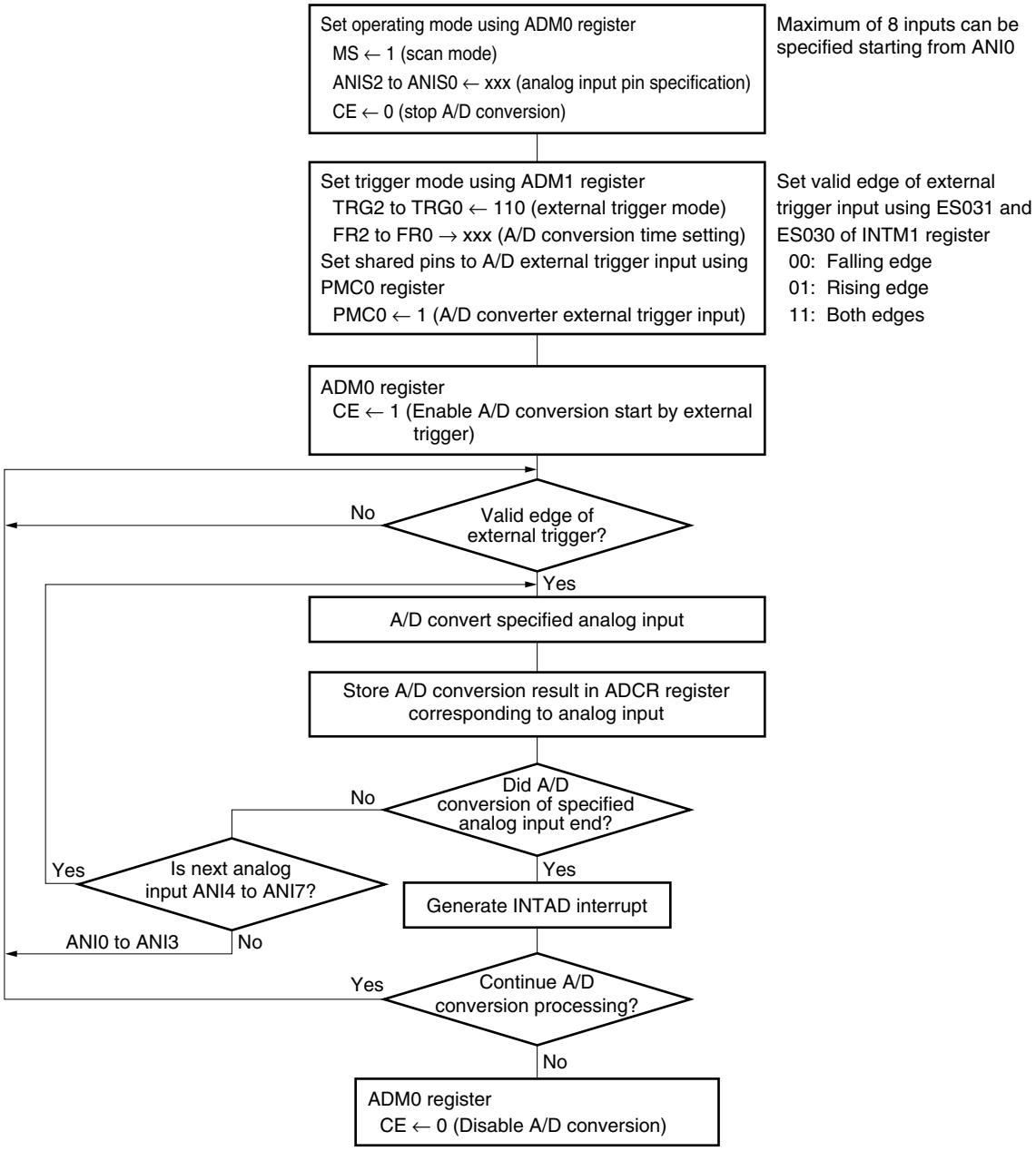
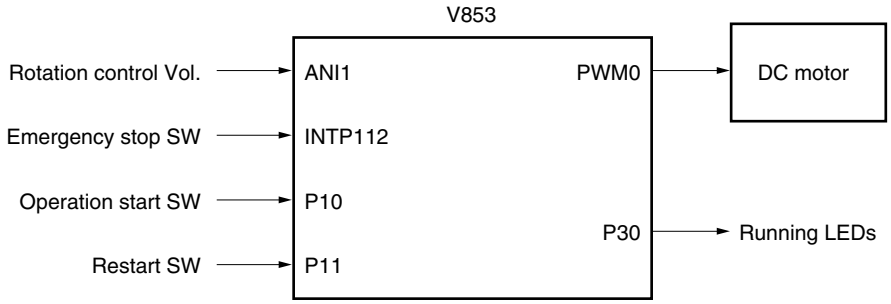


**Figure 3-73. External Trigger Mode + Scan Mode Settings and Operation**



### 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**

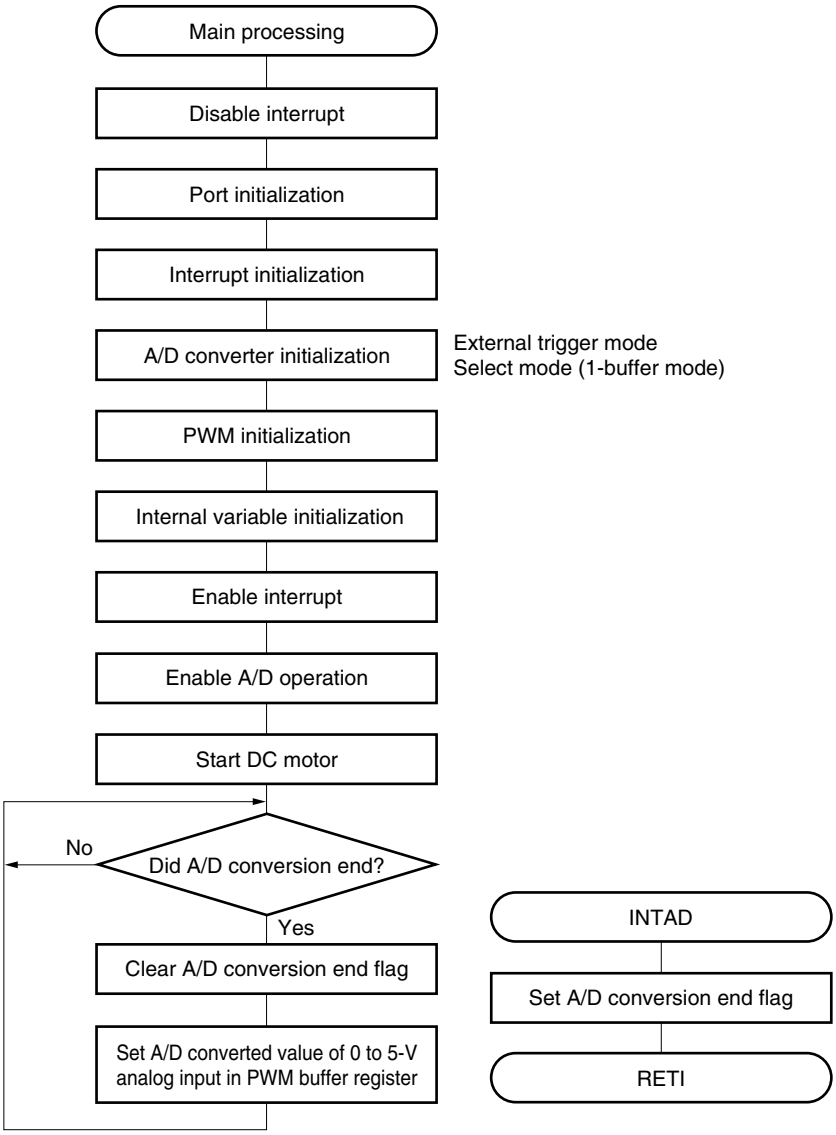
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ADM0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | Address FFFFF380H | Value 11H |
| ADM1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | Address FFFFF382H | Value 67H |

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
 * File Name:      exttrig.c
 * ------------------------------------------------
 */

#pragma ioreg      /* Make peripheral I/O register names usable */

/* ------------------------------------------------------------------
 * 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 interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   ;      /* Do nothing */
}

/* ----------------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------
 */
#pragma interrupt INTAD int_ad
__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 interrupt INTP112 int_p112
__interrupt
void int_p112(void)
{
    /* Process only during execution */
    if ( StartFlg != 0) {
        /* Set emergency stop flag */
        EmergencyFlg = 1;
    }

    /* Stop PWM output */
    PWM0 = 0;
    PWME0 = 0;
}

/* ----------------------------------------
 * 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 valid 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 A/DCInit(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 φ/1 */
   PWM0 = 0;      /* Clear buffer register */
}

/* ----------------------------------------
 * Operation start processing
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------
 */
void StartProc(void)
{
   /* Start A/D conversion */
   CE = 1;

   /* Start PWM output */
   PWME0 = 1;

   /* Enable interrupt */
   __EI();
}

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

   /* Stop A/D conversion */
   CE = 0;

   /* PWM output initialization */
   PWM0 = 0;

   /* Stop PWM output */
   PWME0 = 0;
}

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

```
void Main(void)
{
   int    pulsFlg = 0;   /* Display pulse option flag */

   /* 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 ((P1.0 !=0) && (StartFlg == 0)) {
         StartProc();
         StartFlg = 1;
      }
      /* Do not process until start flag is ON */
      if (StartFlg == 0) {
         continue;
      }
      /* Check for start switch OFF */
      if ((P1.0 == 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 ( P1.1 == 0 ) {
            continue;
         }
         /* Clear emergency stop flag */
         EmergencyFlg = 0;
         /* Start PWM operation */
         PWME0 = 1;
      }
      /* 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 valid 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<br>(n = 1, 0) | D/A Convert<br>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 $\phi$, $\phi$/2, $\phi$/4, $\phi$/8, and $\phi$/16 ($\phi$: 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<br>Controls PWMn operation.<br>0: Disable operation<br>1: Enable operation<br>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<br>Specify the active level of PWMn.<br>0: Active low<br>1: Active high |
| 5, 4, 1, 0 | PRMn1, PRMn0 | Prescaler Mode<br>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,<br>2 to 0 | PWPn2,<br>PWPn1,<br>PWPn0 | PWM Prescaler Clock Mode<br>Selects the PWMn operation clock. |

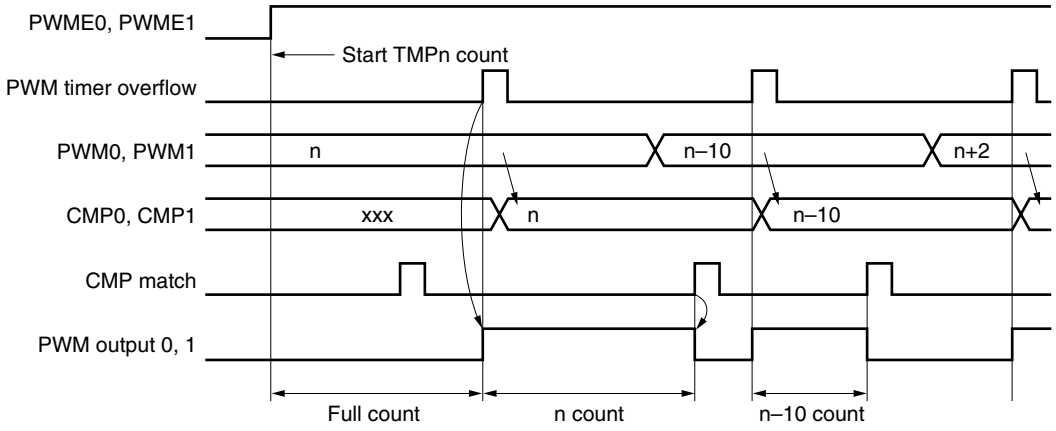
| PWPn2 | PWPn1 | PWPn0 | Operation clock |
|---|---|---|---|
| 0 | 0 | 0 | $\phi$ |
| 0 | 0 | 1 | $\phi$/2 |
| 0 | 1 | 0 | $\phi$/4 |
| 0 | 1 | 1 | $\phi$/8 |
| 1 | 0 | 0 | $\phi$/16 |
| Other | | | RFU (reserved) |

**Remark** n = 1, 0

### 3.7.1 PWM operation

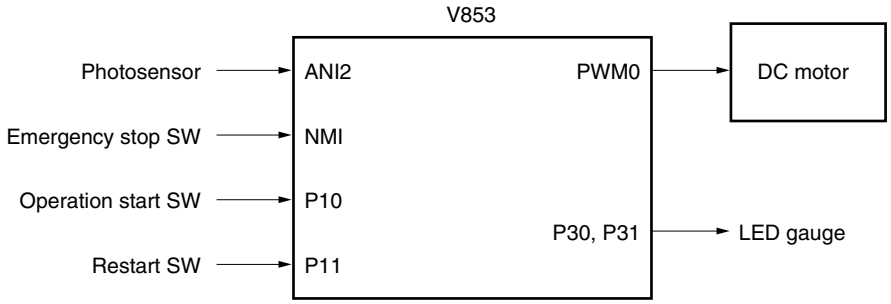
To output a PWM pulse, select an operation clock by using the PWPR register and set the number of counter clocks during the period in which PWMn output is at the active level in the PWM register. After this, setting the PWMEn bit of the PWMC register resets the internal counter (TMPn) and PWMn output 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 $\phi$/2 as the operation clock.

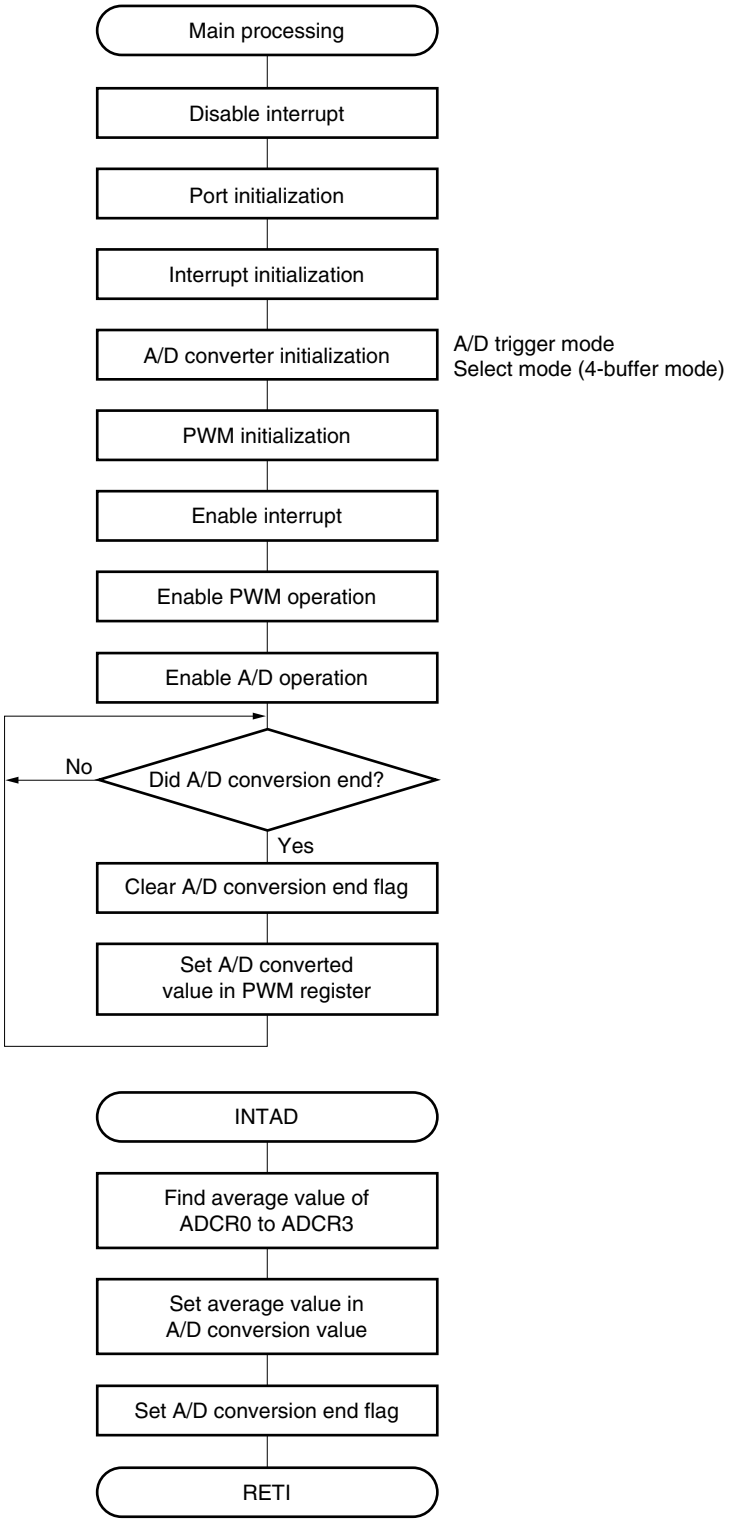
Set the PWM control register (PWMC) and PWM prescaler register (PWPR) using the above conditions.

**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
 * File Name:    smplpwm.c
 * ------------------------------------------
 */

#pragma ioreg    /* Make peripheral I/O register names usable */

/* ----------------------------------------------------------
 * 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 status) */
static int StartFlg = 0;      /* Execution start flag */

/* ------------------------------------
 * NMI servicing
 * DC motor emergency stop processing
 * Interrupt source:  NMI
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   /* Process only while executing */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM operation */
   PWM0 = 0;
   PWME0 = 0;
}

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

```
 * ------------------------------------
 */
#pragma interrupt INTAD int_ad
__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 valid 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 valid 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 ϕ/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 */
   CE = 1;

   /* Start PWM output */
   PWME0 = 1;

   /* Enable interrupt */
   __EI();
}

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

```
    __DI();

    /* Stop A/D conversion */
    CE = 0;

    /* Stop PWM output */
    PWM0 = 0;
    PWME0 = 0;
}

/* ------------------------------------
 * Main processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
void main(void)
{
    int    pulsFlg = 0;    /* Display pulse option flag */

    /* 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 ((P1.0 != 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 ((P1.0 == 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 (P1.1 == 0) {
                continue;
```

```
            }
            /* Clear emergency stop flag */
            EmergencyFlg = 0;
            /* Start PWM operation */
            PWME0 = 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 */
            CE = 1;
        }
    }
}
```

## 3.8 Clock Generation Function

The clock generator (CG) generates and controls the internal system clock ($\phi$) 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** 00H (if MODE = 1)
03H (if MODE = 1)

| Bit position | Bit name | Description |
|---|---|---|
| 1, 0 | CKDIV1, CKDIV0 | Clock Divide<br>Sets the internal system clock frequency division rate for the external clock (fxx) when in PLL mode. |

| Operation mode | Pin | CKC register | | Internal system clock ($\phi$) |
|---|---|---|---|---|
| | $CV_{DD}/\overline{CKSEL}$ | CKDIV1 | CKDIV0 | |
| 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
   :
```

★ **Caution See 3.8.3 Cautions on power save function when setting software STOP mode during instruction execution on the external ROM.**

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 | C0H |

| Bit position | Bit name | Description |
|---|---|---|
| 7, 6 | DCLK1, DCLK0 | Disable CLKOUT<br>Specifies the operating mode of the CLKOUT pin.<br><br>DCLK1 / DCLK0 / Mode<br>0 / 0 / Normal output mode<br>0 / 1 / RFU (reserved)<br>1 / 0 / RFU (reserved)<br>1 / 1 / Clock output inhibit mode |
| 5 | TBCS | Time Base Count Select<br>Selects the time base counter clock.<br>0: fxx/$2^8$<br>1: fxx/$2^9$<br>For details, refer to **V853 Hardware User's Manual**. |
| 4 | CESEL | Crystal/External Select<br>Specifies the function of the X1 and X2 pins.<br>0: Connect oscillator to pins X1 and X2<br>1: Connect external clock to pin X1<br>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<br>Specifies IDLE mode.<br>IDLE state is entered by writing "1".<br>This is automatically reset to "0" when IDLE mode is released. |
| 1 | STP | STOP Mode<br>Specifies software STOP mode.<br>STOP state is entered by writing "1".<br>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 $\overline{\text{RESET}}$ pin
- NMI request
- Non-masked maskable interrupt request

**(2) How to release IDLE mode**

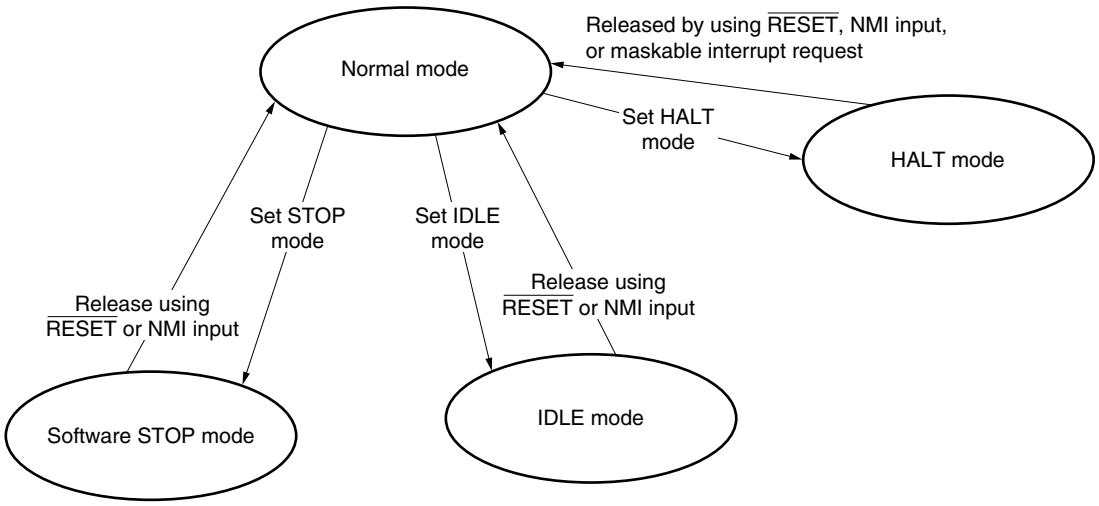
- Input to $\overline{\text{RESET}}$ pin
- NMI request

**(3) How to release software STOP mode**

- Input to $\overline{\text{RESET}}$ pin
- NMI request

A state transition diagram for each mode is shown below.

**Figure 3-86. State Transitions for Each Mode**



### ★ 3.8.3 Cautions on power save function

If the V853 is used under the following conditions[Note], a discrepancy occurs between the address indicated by the program counter (PC) and the address at which an instruction is actually read after the power save mode is released.

This may result in the CPU ignoring a 4-byte or 8-byte instruction from between 4 bytes and 16 bytes after an instruction is executed to write to the PSC register, which could in turn result in the execution of an erroneous instruction.

**Note** PC errors occur only when conditions (i) to (iii) below are all satisfied.

**[Conditions]**

(i) Setting of power save mode (IDLE mode or STOP mode) while an instruction is being executed on external ROM

(ii) Cancellation of power save mode as the result of an NMI interrupt request

(iii) Execution of the next instruction when an interrupt request is held pending following release of the power save mode

Conditions for interrupt request to be held pending:

When NP flag of PSW register is “1” (NMI servicing in progress/set by software)

Therefore, use the V853 under the following conditions.

**[Usage Conditions]**

(i) Do not use a power save mode (IDLE mode or STOP mode) during instruction execution on external ROM.

(ii) If it is necessary to use a power save mode during instruction execution on external ROM, implement the following software measures.

- Insert 6 NOP instructions 4 bytes after an instruction that writes to the PSC register.
- After the NOP instructions, insert a br$+2 instruction to cancel the PC discrepancy.

**[Workaround program example]**

```
LDST   rX,5              ;Sets rX value to PSW
ST.B   r0,PRCMD[r0]      ;Writes to PRCMD
ST.B   rD,PSC[r0]        ;Sets PSC register
LDST   rY,5              ;Returns PSW value
NOP                      ;6 or more NOP instructions
NOP
NOP
NOP
NOP
NOP
BR   $+2                 ;Cancels PC discrepancy
```

**Remark** It is assumed that rD (PSC setting value), rX (value written to PSW), and rY (value written back to PSW) have been set.

## 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 | I/O 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) | 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 |
| | | Clocked 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) | C0H |

★ **Caution “Undefined” means an undefined value due to power-on reset or data corruption when a falling edge of RESET matches a data write operation. The previous status of data is retained by a falling edge of RESET in cases other than the above.**

# CHAPTER 4 APPLICATION PROGRAMS THAT OPERATE ON TU-V853

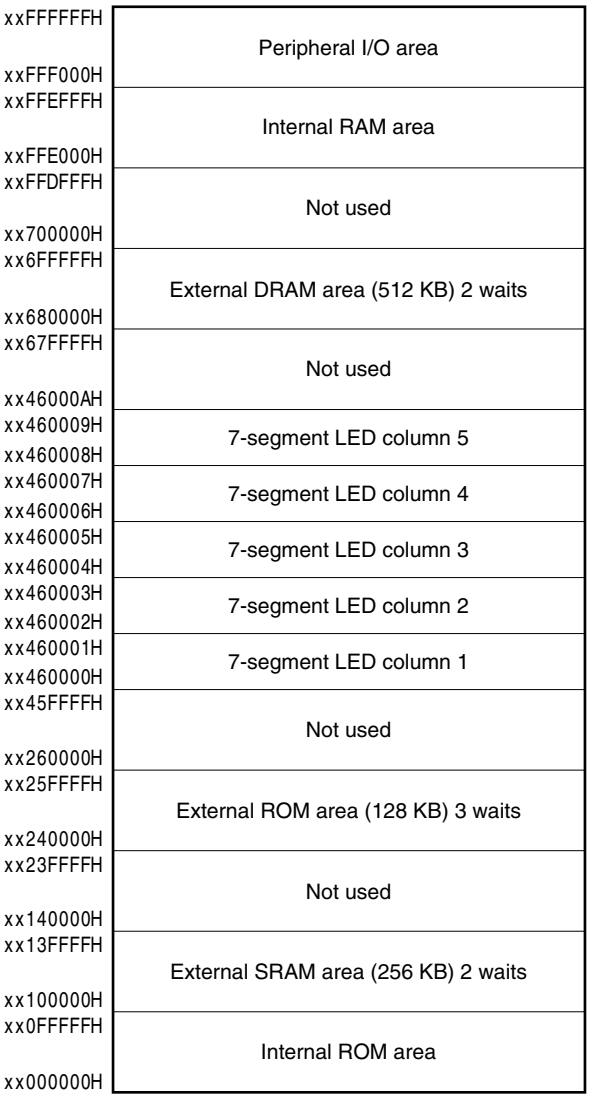
## 4.1 Overview of TU-V853

The TU-V853 seminar unit is a training unit developed for learning the V853. The training board (TB-V853) with V853 in-circuit emulator and other device components are housed in trunks for transport. Features of the unit are shown below.

- Connects the in-circuit emulator for V853
  - In-circuit emulator/CPU connection socket implemented
- Operating speed: Supports 33 MHz
- Serial interface
  - RS-232-C × 1 channel
  - Connection to V853 on-chip UART1
- Serial EEPROM implemented
  - Connection to V853 on-chip CSI0
- Analog input
  - Connects output of sine-wave oscillator
  - Connects 0 to 5 V variable output
  - Connects photosensor output
- Analog output
  - Audio output is possible by connecting stereo headphones
- 8 points general purpose switch input, 8 points LED lamp output
  - Uses V853 on-chip PIO
- Seven-segment LEDs × 5 columns
  - Display by selecting software control or hardware control using switch
  - When under hardware control, displays frequency of output of sine-wave oscillator
- DC motor with pulse generator output implemented
  - Pulse generator output connects to INTP140

### 4.1.1 Address map

**Figure 4-1. Address Map**



### 4.1.2 List of I/O ports used for internal I/O

**Table 4-1. List of I/O Ports Used for Internal I/O**

| Port Name | I/O | Usage |
|---|---|---|
| P10 | Input | General purpose toggle switch 0 |
| P11 | | General purpose toggle switch 1 |
| P12 | | General purpose toggle switch 2 |
| P13 | | General purpose toggle switch 3 |
| P14 | | General purpose toggle switch 4 |
| P15 | | General purpose toggle switch 5 |
| P16 | | General purpose toggle switch 6 |
| P17 | | General purpose toggle switch 7 |
| P30 | Output | General purpose LED lamp 0 (“1” turns on) |
| P31 | | General purpose LED lamp 1 (“1” turns on) |
| P32 | | General purpose LED lamp 2 (“1” turns on) |
| P33 | | General purpose LED lamp 3 (“1” turns on) |
| P34 | | General purpose LED lamp 4 (“1” turns on) |
| P35 | | General purpose LED lamp 5 (“1” turns on) |
| P36 | | General purpose LED lamp 6 (“1” turns on) |
| P37 | | General purpose LED lamp 7 (“1” turns on) |
| P10 | Input | CS pin of EEPROM |

### 4.1.3 External maskable interrupt sources

**Table 4-2. External Maskable Interrupt Sources**

| Interrupt Name | External Pin | Edge Specification |
|---|---|---|
| INTP112 | INT switch | Rising edge |
| INTP113 | Sine-wave interrupt | |
| INTP140 | Pulse generator output interrupt | |

### 4.1.4 Initial values of internal registers

**Table 4-3. Initial Values of Internal Registers**

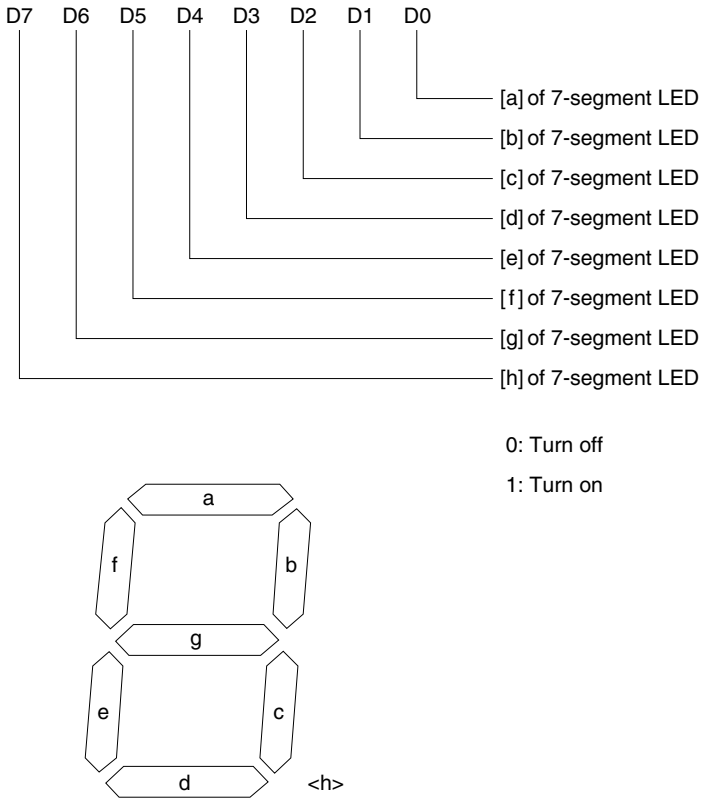
| Address | Register Name | Symbol | Initial Value |
|---|---|---|---|
| FFFFF020H | Port 0 mode register | PM0 | FFH |
| FFFFF022H | Port 1 mode register | PM1 | FFH |
| FFFFF024H | Port 2 mode register | PM2 | FFH |
| FFFFF026H | Port 3 mode register | PM3 | 00H |
| FFFFF028H | Port 4 mode register | PM4 | FFH |
| FFFFF02AH | Port 5 mode register | PM5 | FFH |
| FFFFF02CH | Port 6 mode register | PM6 | FFH |
| FFFFF032H | Port 9 mode register | PM9 | FFH |
| FFFFF036H | Port 11 mode register | PM11 | FFH |
| FFFFF040H | Port 0 mode control register | PMC0 | C0H |
| FFFFF042H | Port 1 mode control register | PMC1 | 00H |
| FFFFF044H | Port 2 mode control register | PMC2 | 7DH |
| FFFFF046H | Port 3 mode control register | PMC3 | 00H |
| FFFFF04CH | Memory expansion mode register | MM | AFH |
| FFFFF056H | Port 11 mode control register | PMC11 | 10H |
| FFFFF060H | Data wait control register | DWC | 002FH |
| FFFFF062H | Bus cycle control register | BCC | 0000H |
| FFFFF180H | External interrupt mode register 0 | INTM0 | 01H |
| FFFFF182H | External interrupt mode register 1 | INTM1 | 50H |
| FFFFF184H | External interrupt mode register 2 | INTM2 | 00H |
| FFFFF186H | External interrupt mode register 3 | INTM3 | 00H |
| FFFFF188H | External interrupt mode register 4 | INTM4 | 01H |

### 4.1.5 Seven-segment LED display registers

These registers turn on the seven-segment LEDs.

Address: xx460000H Column 1 (write/byte)
Address: xx460002H Column 2 (write/byte)
Address: xx460004H Column 3 (write/byte)
Address: xx460006H Column 4 (write/byte)
Address: xx460008H Column 5 (write/byte)

**Figure 4-2. Seven-Segment LED Display Register**



## 4.2 Application Program Creation Procedure

### 4.2.1 Compile and link procedure

Software can be developed efficiently by using a project manager in V853 software development. This section describes the procedure for creating an application by compiling and linking a coded source program using a project manager. For details and operating techniques, refer to **CA850 Project Manager User's Manual**.

**(1) Project file creation**

When the project manager starts up and "New" is selected from the "Project" menu, the "Project setup" dialog is opened. Input each item and click on the "OK" button.

**(2) Source file setup**

Clicking on the "OK" button in the "Project setup" dialog box opens the "Source file setup" dialog box. Click on the "ADD" button in the "Source file setup" dialog box to register source files to link. Click on the "OK" button when setup is complete.

**(3) Option setup**

Set the following four items using the "Options" menu.

- Project manager option settings
- Compiler option settings
- Assembler option settings
- Linker option settings

Be sure to set the editor to use in "Project manager option settings".

**(4) Make file creation**

Selecting "Create make file" from the "Project" menu automatically executes a make file.

**(5) Executable file creation**

Selecting "Build" from the "Execution" menu displays an error in the "PRJTMAKE" window. Double-clicking on the error opens the source in question. Repeating this task creates an executable file.

### 4.2.2 ROMization procedure

This section describes a ROMization procedure that uses the ROMization processor (romp) included in the compiler package.

**(1) Copy program creation**

Within the program to make into ROM, start the copy routine _rcopy() using the arguments needed at the time the program starts up.

**Example** Use of copy routine _rcopy

```
#define ALL_COPY
extern unsigned long _S_romp;
main()
{
   int  ret;
   ret = _rcopy(&_Sromp,ALL_COPY)
}
```

**(2) Object file creation**

In the VHS command line, create the object file by specifying the ca option for ROMization. When doing so, specify reference to the library for ROMization and link the code for reserving areas for the rompsec section. Also reference a link directive that takes into account the areas for the rompsec section.

**(3) Creation of object file having romp section**

Create an object file that has a rompsec section using romp. Sections that have the data attribute or sdata attribute (attribute of having an initial value and being placed in RAM) are moved to the rompsec section.

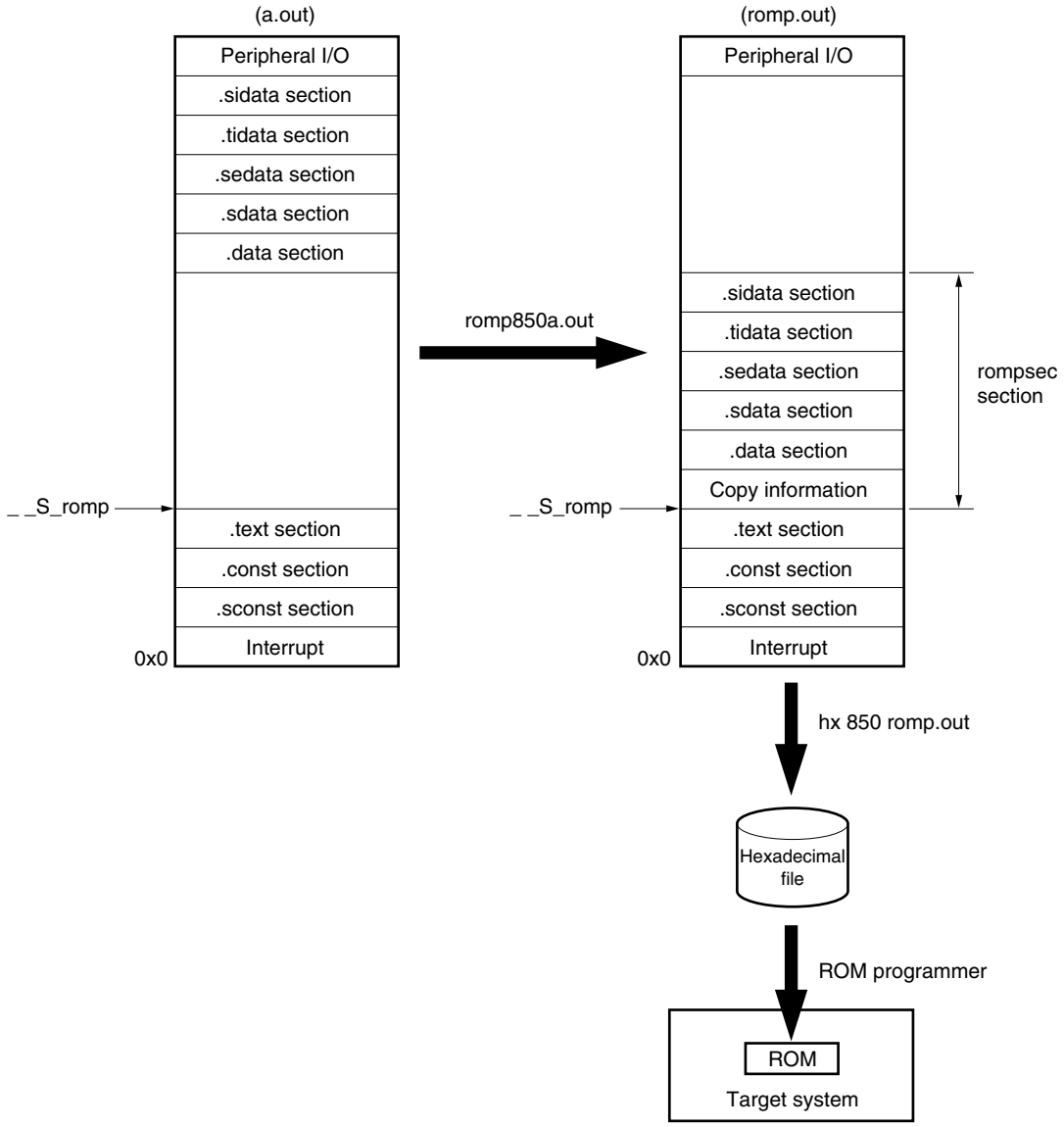
**(4) Hexadecimal data creation**

Create hexadecimal data using hx.

**(5) ROM creation**

Load the created hexadecimal data into the ROM of the target system.

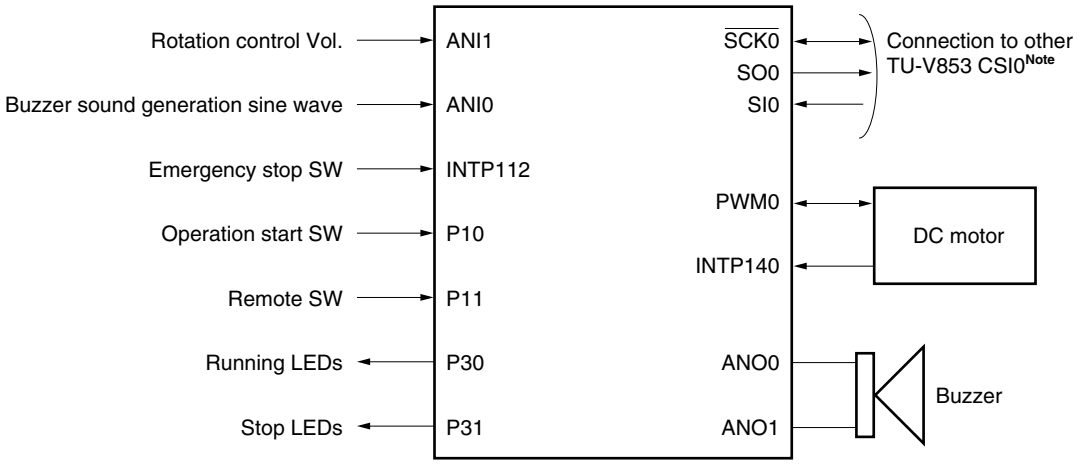
**Figure 4-3. Image of ROMization**



## 4.3 Application Program Examples

The sample hardware in Figure 4-4 will be used to create application programs.

**Figure 4-4. Sample Hardware Configuration**



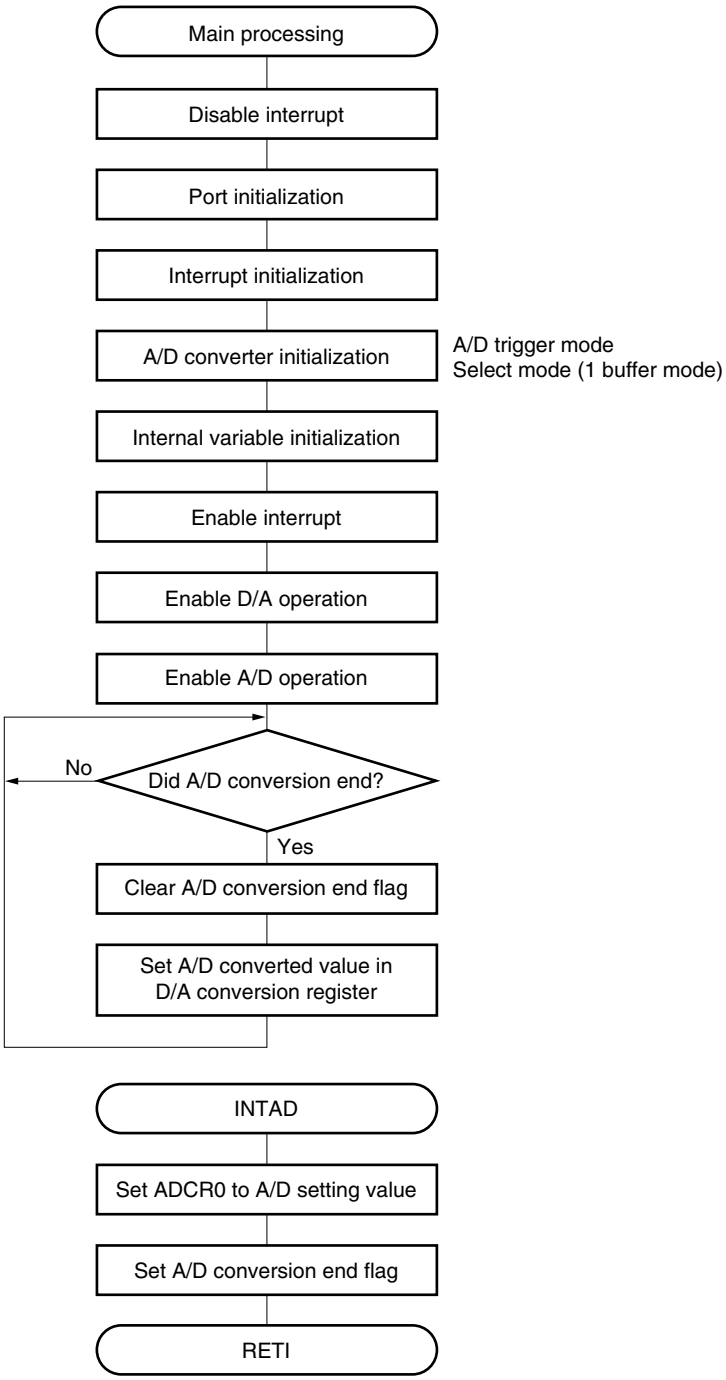

**Note** The TU-V853 CSI0 connects to an EEPROM but is remodeled for testing the “Communication program using CSI function and interrupt function on two V853” (see Section 4.3.3).

### 4.3.1 Buzzer generation program using analog input and digital output

#### (1) Function overview

- Input a sine wave for buzzer sound generation from the ANI0 pin and write the A/D converted value to the D/A conversion register and output it to the ANO0 and ANO1 pins.

**Figure 4-5. Flow of Buzzer Sound Modulation Processing**



### (2) Sample program

```
/* ------------------------------------------------
 * V853 Buzzer generation program using analog
 *                   input and digital output
 *
 *  Initialization program for the following hardware configuration
 *      ANI0  :  Buzzer sound generation sine wave input
 *      ANO0/1:  Buzzer output
 * ------------------------------------------------
 * History:
 *    1997/3/17  ver 0.00.00
 * File Name:    apl531.c
 * ------------------------------------------------
 */

#pragma ioreg     /* Make peripheral I/O register names usable */

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

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

/* ----------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ----------------------------------
 */
#pragma interrupt INTAD int_ad
__interrupt
void int_ad(void)
{
   /* Assign conversion result */
   BuzzerDat = ADCR0H;  /* ADCR0 is 10-bit data */
                        /* Since higher 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 valid 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
 * ----------------------------------
 */
void main(void)
{
    /* Disable interrupt */
    __DI();

    /* Port initialization processing */
    PortInit();

    /* Interrupt initialization processing */
    IntInit();

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

    /* Internal data initialization */
    BuzzerDat = 0;
    AdFlag = 0;

    /* Start D/A conversion */
    DACS0 = 0;
    DACS1 = 0;
    DACE0 = 1;
    DACE1 = 1;

    /* Start A/D conversion */
    CE = 1;

    /* Enable interrupt */
    __EI();

    while ( 1 ) {
        /* A/D conversion completion */
        if ( AdFlag != 0 ) {
            /* Set A/D converted value in D/A register */
```

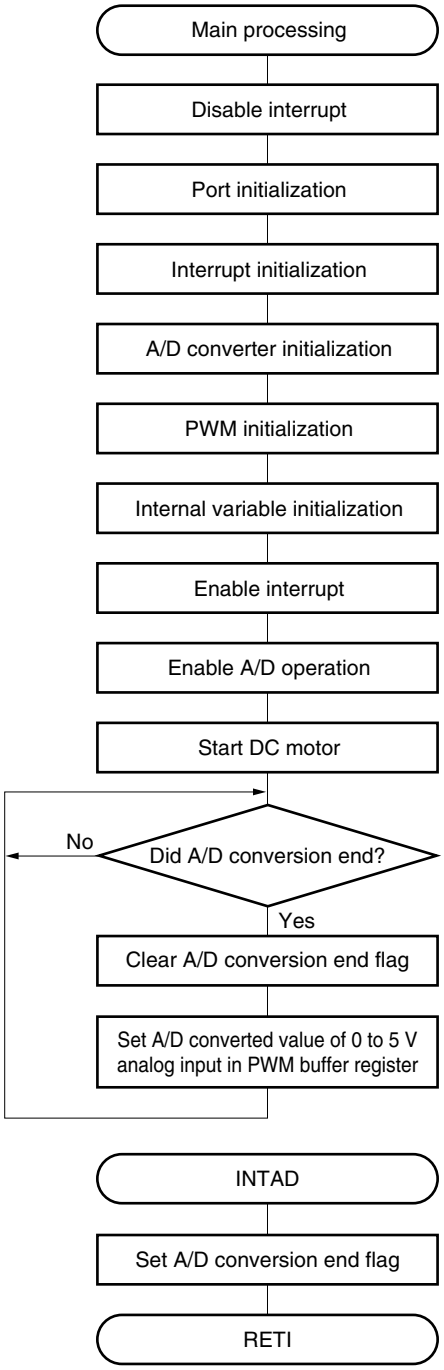
```
            DACS0 = BuzzerDat;
            DACS1 = BuzzerDat;
            AdFlag = 0;
            CE = 1;        /* Start A/D conversion */
        }
    }
}
```

### 4.3.2 Motor rotation control program using interrupt function and PWM function

#### (1) Function overview

- Make the DC motor rotate using the PWM function and the value A/D input by the rotation control Vol.
- Start the motor by setting the operation start SW to ON and stop it by setting the operation start SW to OFF. To stop the rotation at once, stop the motor using the emergency stop SW. Restart the motor using the restart SW.

**Figure 4-6. Flow of Motor Rotation Control Processing**



### (2) Sample program

```
/* ------------------------------------------------
 * V853  Motor rotation control program using
 *             interrupt function and PWM function
 *
 *   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
 * File Name:    apl532.c
 * ------------------------------------------------
 */

#pragma ioreg     /* Make peripheral I/O register names usable */

/* ----------------------------------------------------------------
 * Internal variable definition
 * ----------------------------------------------------------------
 */
   /* Variable to store value after A/D conversion */
static short ContVol = 0;      /* Value of DC motor rotation control Vol. */

   /* Operation flags */
static int EmergencyFlg = 0;   /* Emergency stop flag (!0: Emergency stop status) */
static int StartFlg = 0;       /* Execution start flag */
static int AdFlag = 0;         /* A/D completion flag */

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

/* ------------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTAD int_ad
__interrupt
void int_ad(void)
```

```
{
   /* Calculate average of conversion results */
   ContVol = ADCR0 + ADCR1 + ADCR2 + ADCR3;
   ContVol /= 4;
   AdFlag = 1;
}

/* ------------------------------------
 * DC motor emergency stop processing
 * Interrupt source:  INTP112
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
#pragma interrupt INTP112 int_p112
__interrupt
void int_p112(void)
{
   /* Process only while executing */
   if ( StartFlg != 0) {
      /* Set emergency stop flag */
      EmergencyFlg = 1;
   }
   /* Stop PWM operation */
   PWM0  = 0;
   PWME0 = 0;
}

/* ------------------------------------
 * 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)
```

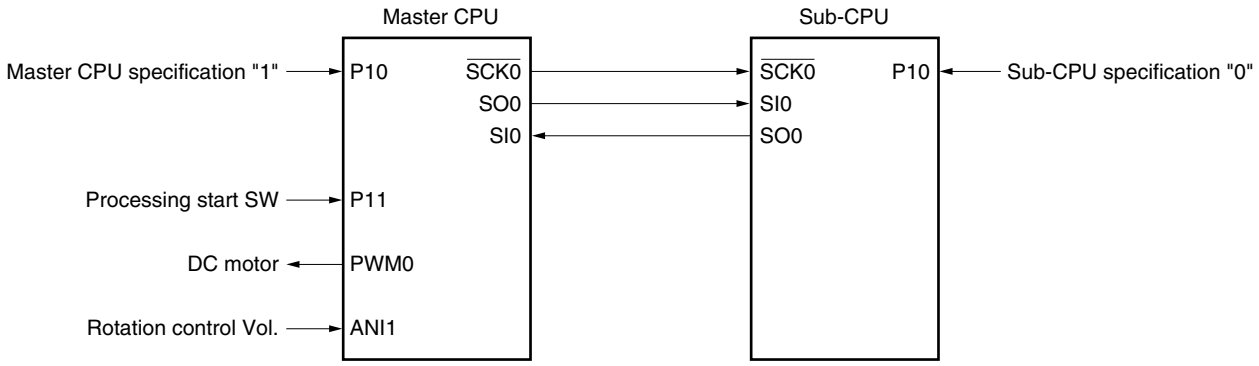
```
    */
    PM4  = 0xFF;   /* Use P40 to P47 as address/data bus (reference MM register) */

    /* 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 */

    /* Masked by default, but specify for clarity */
    P11IC3 = 0x47;         /* Mask interrupts because A/D external trigger is used */

    /* Interrupt valid 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 = 0x11;   /* Select mode: Input analog signal from ANI1 */
    /* ADM1 register settings */
    ADM1 = 0x07;   /* (FR2 to FR0) conversion clocks = 192 */
}

/* ------------------------------------
```

```
 * PWM unit initialization processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
void PwmInit(void)
{
   /* PWM settings (PWMC):  Set only 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 = 0x00;  /* Prescaler is φ/1 */
   PWM0 = 0;     /* Clear buffer register */
}

/* ------------------------------------
 * Operation start processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
void StartProc(void)
{
   /* Start A/D conversion */
   CE = 1;

   /* Start PWM output */
   PWME0 = 1;

   /* Enable interrupt */
   __EI();
}

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

   /* Stop A/D conversion */
   CE = 0;

   /* Initialize PWM output */
   PWM0 = 0;

   /* Stop PWM output */
   PWME0 = 0;
}

/* ------------------------------------
 * Main processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------
 */
void main(void)
```

```
{
   int   pulsFlg = 0;    /* Display pulse option flag */

   /* 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. */
   EmergencyFlg = 0;   /* Emergency stop flag (!0: Emergency stop status) */
   StartFlg = 0;       /* Clear start flag */

   while ( 1 ) {
      /* Check for start switch ON */
      if ((P1.0 != 0) && (StartFlg == 0)) {
         StartProc();
         StartFlg = 1;
      }
      /* Stop processing until start flag is ON */
      if (StartFlg == 0) {
         StopProc();
         continue;
      }
      /* Check for start switch OFF */
      if ((P{1.0 == 0) && (StartFlg != 0)) {
         StartFlg = 0;
         /* Also clear emergency stop flag */
         EmergencyFlg = 0;
      }
      /* During emergency stop */
      if ( EmergencyFlg != 0 ) {
         /* Do not process unless emergency stop is cancelled */
         if ( P1.1 == 0 ) {
            continue;
         }
         /* Clear emergency stop flag */
         EmergencyFlg = 0;
         /* Start PWM operation */
         PWME0 = 1;
      }
      /* A/D conversion completion */
      if ( AdFlag != 0 ) {
         /* Set DC motor control Vol. value in PWM register */
         PWM0 = ContVol;
         AdFlag = 0;
         CE = 1;
      }
   }
}
```

### 4.3.3 Communication program using CSI function and interrupt function on two V853 devices

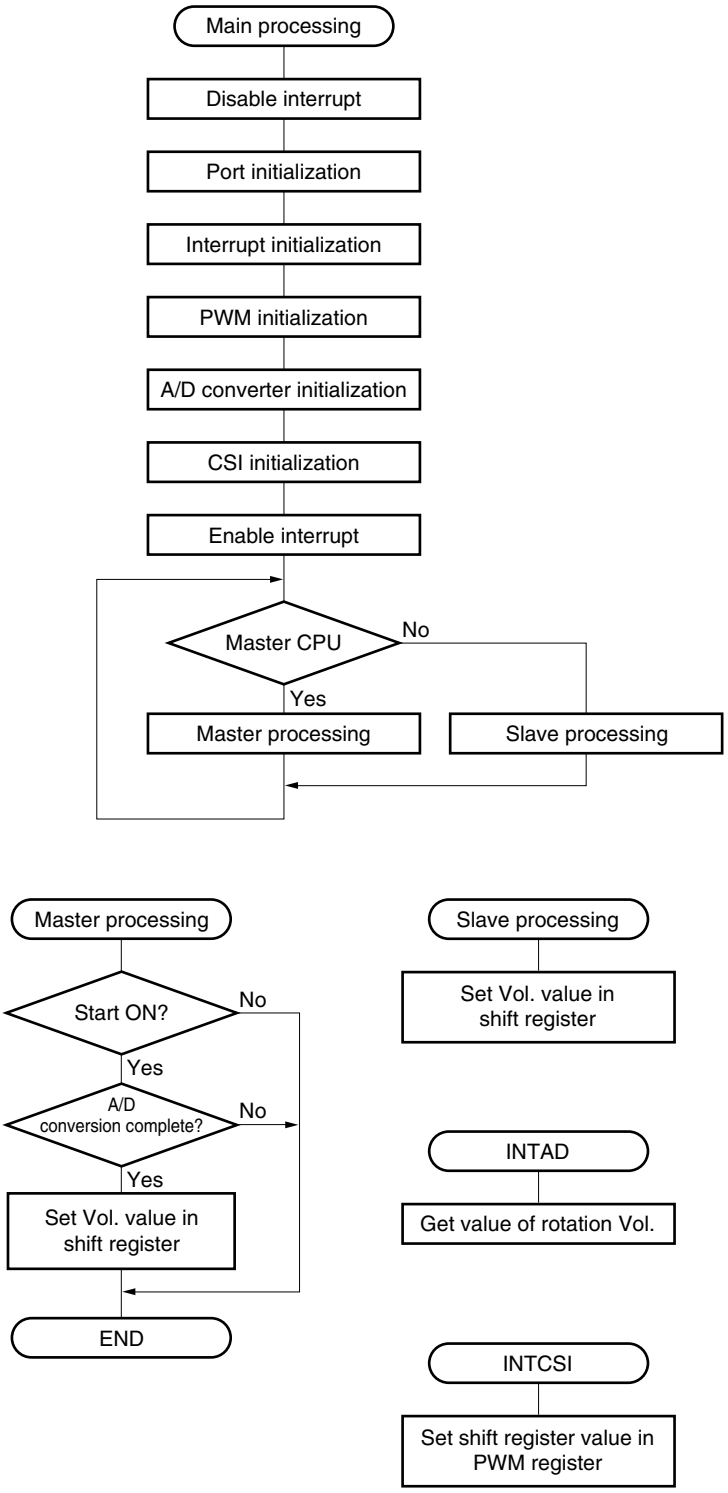
**Figure 4-7. Hardware Configuration**




**(1) Function overview**

- Transmit and receive data between two V853 devices using clocked serial interface.
- Divide the two V853 devices into master CPU and sub-CPU and start communication from the master CPU.
- Transmit the A/D input rotation control Vol. value.
- Make the DC motor rotate using the PWM function and received data.

**Figure 4-8. Flow of Data Transmit and Receive Processing**



### (2) Sample program

```
/* ----------------------------------------------------
 * V853  Communication program using CSI function and
 *             interrupt function on two V853 devices
 *
 *    Connection of two V853 devices
 *   <Master>      <Sub>
 *     SCK0 ------→ SCK0
 *     SO0  ------→ SI0
 *     SI0  ←------ SO0
 *
 *     P10:  Master/slave switching SW (Off: Master, On: Slave)
 *     P11:  Processing start SW (On: Start processing, Off: Stop processing)
 *
 * ----------------------------------------------------
 * History:
 *     1997/3/17  Ver 0.00.00
 *  File Name:    apl533.c
 * ----------------------------------------------------
 */

#pragma ioreg     /* Make peripheral I/O register names usable */

/* ------------------------------------------------------------------
 * Internal variable definition
 * ------------------------------------------------------------------
 */
   /* Variable to store value after A/D conversion  */
static short ContVol = 0;     /* Value of DC motor rotation control Vol. */

   /* Operation flags */
static int StartFlg = 0;      /* CSI transmit/receive start flag
                                 (!0: Transmitting or receiving) */
static short ADCFlg = 0;      /* A/D conversion end flag (!0: Conversion end) */
static short IntCsiFlg = 0;   /* Transmit/receive interrupt flag
                                 (!0: Transmit/receive completion) */


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

/* ------------------------------------------
 * Clocked serial communication end
 * Interrupt source:  INTCSI0
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
#pragma interrupt INTCSI0 int_csi0
```

```
__interrupt
void int_csi0(void)
{
   PWM0 = SIO0 * 4;    /* Write data to PWM register */
   IntCsiFlg = 1;      /* Transmit/receive interrupt end flag ON */
}

/* ------------------------------------------
 * A/D converter conversion end
 * Interrupt source:  INTAD
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
#pragma interrupt INTAD int_ad
__interrupt
void int_ad(void)
{
   long wk;

   /* Calculate average of conversion results */
   wk = ADCR0;
   wk += ADCR1;
   wk += ADCR2;
   wk += ADCR3;
   wk /= 4;            /* Average */
   /* Input DC motor rotation control Vol. */
   ContVol = wk / 4;   /* Convert 10 bits to 8 bits */
   /* A/D conversion end flag ON */
   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 = 0x1D;   /* Use SO0, SI0, SCK0  P22, P23, P24 in control mode */
                  /* Use PWM0  P20 in control mode */
                  /* 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) */
    ADIC = 0x07;       /* INTAD: A/D conversion end        : Level 7 */
    CSIC0 = 0x07;      /* INTCSI0: CSI transmit/receive end: Level 7 */

    /* Interrupt valid 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 = 0x31;   /* (BS) 4-buffer mode */
                   /* (MS) Select mode */
                   /* (ANIS2 to ANIS0) Input analog signal from ANI1 */

    /* ADM1 register settings */
```

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

/* ------------------------------------------
 * CSI initialization processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
void CSIInit(void)
{
   /* Asynchronous serial mode settings */
   ASIM00 = 0x00;     /* Disable transmit/receive → CSI mode */
   ASIM10 = 0x00;     /* Disable transmit/receive → CSI mode */

   /* Clocked serial mode settings */
   CSIM0 = 0x00;      /* Set to external clock */

   /* Baud rate settings (76.8 kbps) */
   BPRM0 = 0x80;
   BRGC0 = 81;        /* 25 MHz */
}

/* ------------------------------------------
 * PWM unit initialization processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
void PwmInit(void)
{
   /* PWM settings (PWMC):  Set only 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 = 0x00;   /* Prescaler is  ϕ/1 */
   PWM0 = 0;      /* Clear buffer register */
}

/* ------------------------------------------
 * Master processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------
 */
void MasterProc(void)
{
   If (StartFlg!=1) {
      StartFlg = 1;     /* Start execution as master */
      CSIM0 = 0xc1;     /* Receive/transmit able status, MSB first */
                        /* Internal clock (using baud rate generator) */
   }
   /* If start button was pressed and A/D conversion ended */
   if (P1.1 != 0 && ADCFlg != 0) {
      ADCFlg = 0;              /* Conversion flag OFF */
      while( CSOT0 ) {};       /* Confirm transmitting */
      SIO0 = ContVol;          /* Set Vol. value in shift register */
      IntCsiFlg = 0;           /* Interrupt end flag OFF */
```

```
      while (IntCsiFlg == 0) { };  /* Confirm transmitting */
      CE = 1;                   /* Enable A/D interrupt */
   }
}

/* ----------------------------------------
 * Slave processing
 * Arguments:  None
 * Return value:  None
 * ----------------------------------------
 */
void SraveProc(void)
{
   if (StartFlg!=2) {
      StartFlg = 2;     /* Start execution as slave */
      CSIM0 = 0xc0;     /* Receive/transmit able status, MSB first */
                        /* External clock */
   }
   while( CSOT0 ) {}    /* Confirm transmitting */
   SIO0 = ContVol;      /* Set Vol. value in shift register */
   if (ADCFlg != 0) {   /* If A/D conversion ended */
      ADCFlg = 0;       /* Conversion flag OFF */
      CE = 1;           /* Enable A/D interrupt */
   }
}

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

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* PWM initialization processing */
   PwmInit();

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

   /* CSI initialization processing */
   CSIInit();

   /* Initialize internal data */
   ContVol = 0;     /* Value of DC motor rotation control Vol. */
   StartFlg = 0;    /* Clear start flag */

   /* Enable interrupt */
   __EI();
   CE = 1;          /* Start A/D conversion */

   /* Start PWM output */
```

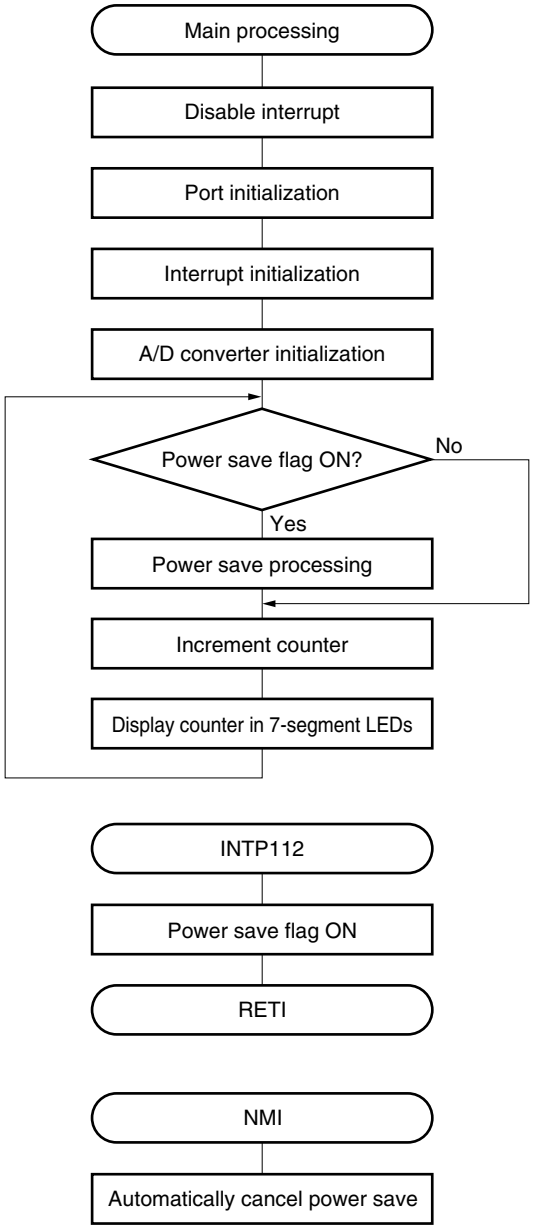
```
    PWME0 = 1;

    while ( 1 ) {
       /* Check for master/slave */
       if (P1.0 == 0) {
          MasterProc();  /* Master processing */
       } else {
          SraveProc();   /* Slave processing */
       }
    }
}
```

### 4.3.4 Program using power save functions

**(1) Function overview**

- Increment a counter and display the value in 7-segment LEDs.
- Enter power save (IDLE) mode on INTP112 interrupt.
- Cancel power save status (IDLE mode) on NMI interrupt.

**Figure 4-9. Flow of DC Motor Control Using Photosensor**



### (2) Sample program

```
/* ------------------------------------------------------
 * V853  Program using power save function
 *
 *  NMI    :  Cancel power save
 *  INTP112:  Power save ON
 *
 * ------------------------------------------------------
 * History:
 *    1997/3/17  Ver 0.00.00
 * File Name:    apl534.c
 * ------------------------------------------------------
 */

#pragma ioreg     /* Make peripheral I/O register names usable */

#include "7segLed.h"

static int IdleFlag;    /* Idle status flag */

/* ------------------------------------------------------
 * NMI servicing
 * Interrupt source:  NMI
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------
 */
#pragma interrupt NMI int_nmi
__interrupt
void int_nmi(void)
{
   /* Do nothing */
}

/* ------------------------------------------------------
 * Flag setting for power save mode
 * Interrupt source:  INTP112
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------
 */
#pragma interrupt INTP112 int_p112
__interrupt
void int_p112(void)
{
   /* Idle status enters on this interrupt. */
   IdleFlag = 1;    /* Idle status 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 = 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 = 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) */
    P11IC2 = 0x07;       /* INTP112: DC motor emergency stop: Level 7 */

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

```
}

/* ------------------------------------------------------
 * 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;                    /* Working 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 ++ ) {
   /* 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;
   }
   /* Generate data of next column */
   disp_data /= base;
   /* If displaying decimal point */
   if ( cnt == dot_pos ) {
      /* Display 7-segment LEDs */
      *segaddr += SEGLED_DOT;
   }
   /* Generate set address of next column */
   segaddr += 2;
   }
}

/* ------------------------------------------------------
 * Power save processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------
 */
void PowerSaveProc(void)
```

```
{
/*
   Points:
      Before an instruction to write to the PSC register, any data must be written
      to the command register.  In this case, r0 (value: 0) can be written.
      After being in IDLE mode, this program cancels it on an NMI or RESET interrupt.
      Since the NP flag of the PWS register is reset to 0 when IDLE mode is canceled,
      an interrupt occurs at the time of cancellation.
*/
   #pragma asm
      stsr   5,r10              —/* r10 = PSW register */
      mov    r10,r11            —/* r10 = Original PSW register value */
      or     0x80,r11           —/* r11 = NP flag ON */
      ldsr   r11,5              —/* Disable NMI interrupt */
      ld.b   PSC,r12            —/* r12 = PSC register */
      or     0x04,r12           —/* r12 = IDLE mode */
      st.b   r0,PRCMD[r0]       —/* Set command */
      st.b   r12,PSC[r0]        —/* Write PSC register */
      ldsr   r10,5              —/* Enable NMI interrupt */
      nop
      nop
      nop
      nop
      nop
   #pragma endasm
}

/* ------------------------------------------------------
 * Main processing
 * Arguments:  None
 * Return value:  None
 * ------------------------------------------------------
 */
void main(void)
{
   unsigned   long   count = 0;   /* Display counter */

   /* Disable interrupt */
   __DI();

   /* Port initialization processing */
   PortInit();

   /* Interrupt initialization processing */
   IntInit();

   /* Enable interrupt */
   __EI();

   /* Initialize internal data */
   IdleFlag = 0;        /* Clear idle status flag */

   while ( 1 ) {
      if ( IdleFlag != 0 ) {
         PowerSaveProc();      /* Put in power save state */
         IdleFlag = 0;         /* Clear idle status flag */
      }
      Disp7SegLED( count , 0, 1 );   /* 7-segment LEDs display */
      count++;
```

```
   }
}


/* ------------------------------------------------------
 * 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   */
```

# 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 | Interrupt | NMI | – | NMI input | – | – | 0010H | 00000010H | next PC |
| 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 | |

★ **Caution The INTP1nm (external interrupt) and INTCC1nm (compare register match interrupt) are controlled by the same control registers (when n = 1 to 4, m = 0 to 3). Set either INTP1nm or INTCC1nm interrupt request to be used, using bits 3 to 0 (IMS1nm) of timer unit mode registers 11 to 14 (TUM11 to TUM14).**

**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 acknowledged 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 | | √ | √ | | |
| FFFFF056H | Port 11 mode control register | PMC11 | | √ | √ | | |
| 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 | | √ | √ | | C0H |
| ★ FFFFF072H | Clock control register | CKC | | √ | √ | | 00H/03H |
| 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 | Clocked serial interface mode register 0 | CSIM0 | | √ | √ | | |
| FFFFF08AH | Serial I/O shift register 0 | SIO0 | | √ | √ | | Undefined |

(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 | Clocked 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 | Clocked serial interface mode register 2 | CSIM2 | | √ | √ | | |
| FFFFF0AAH | Serial I/O shift register 2 | SIO2 | | √ | √ | | Undefined |
| FFFFF0B8H | Clocked 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 (Lower 8 bits) | RXB0L | | √ | √ | | |
| FFFFF0CCH | Transmit shift register 0 (9 bits) | TXS0 | W | | | √ | |
| FFFFF0CEH | Transmit shift register 0L (Lower 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 (Lower 8 bits) | RXB1L | | √ | √ | | |
| FFFFF0DCH | Transmit shift register 1 (9 bits) | TXS1 | W | | | √ | |
| FFFFF0DEH | Transmit shift register 1L (Lower 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 (Lower 8 bits) | PWM0L | | | √ | | |
| FFFFF368H | PWM buffer register 1 (12 bits) | PWM1 | | | | √ | |
| FFFFF36AH | PWM buffer register 1L (Lower 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 TB-V853 CIRCUIT DIAGRAMS

Figures C-1 to C-6 show TB-V853 circuit diagrams. For details of the TB-V853, refer to **V853 Hardware Application Note**.

**(1) CPU and EEPROM peripheral circuit (Figure C-1)**

This mainly consists of the V853 CPU, oscillator, 74FCT16373, toggle switches × 8, and EEPROM.
The 74FCT16373 is used in 16-bit transparent latch address isolation.

**(2) Power and switches (Figure C-2)**

This mainly consists of the power supply connector, reset switch, and NMI switch.
The TL7705A (power monitor) is used in the reset generation function.

**(3) Device controller (Figure C-3)**

This mainly consists of the EMP7128-84 and MAX232.
The EMP7128-84 inputs signals related to the V853 CPU bus interface, creates memory and I/O device select signals and control signals, and outputs WAIT signals to the V853 CPU.
The EMP7128-84 is a field programmable gate array (FPGA).

**(4) Memory peripheral circuit (Figure C-4)**

This mainly consists of the EPROM, SRAM, DRAM, 74FCT16244, and 74AC157.
The 74FCT16244 is used in avoidance of data bus collisions due to EPROM output floating delays (16-bit buffer).
The 74AC157 (selector) is used in DRAM address generation.

**(5) LED controller (Figure C-5)**

This mainly consists of the seven-segment LEDs × 5, EMP7160-84, and 74FCT16244 × 3.
The EMP7160-84 (FPGA) controls the seven-segment LEDs. When the switch is on the hardware control side, it counts the output of the sine-wave oscillator and outputs the count value to the seven-segment LEDs per second.

**(6) I/O circuit (Figure C-6)**

This mainly consists of analog-related amplifier components, a photosensor, pin jack connector for audio output, and connector for a DC motor.
The MJN2096 is a headphone amplifier.

**[MEMO]**

**Figure C-1. CPU and EEPROM Peripheral Circuit**

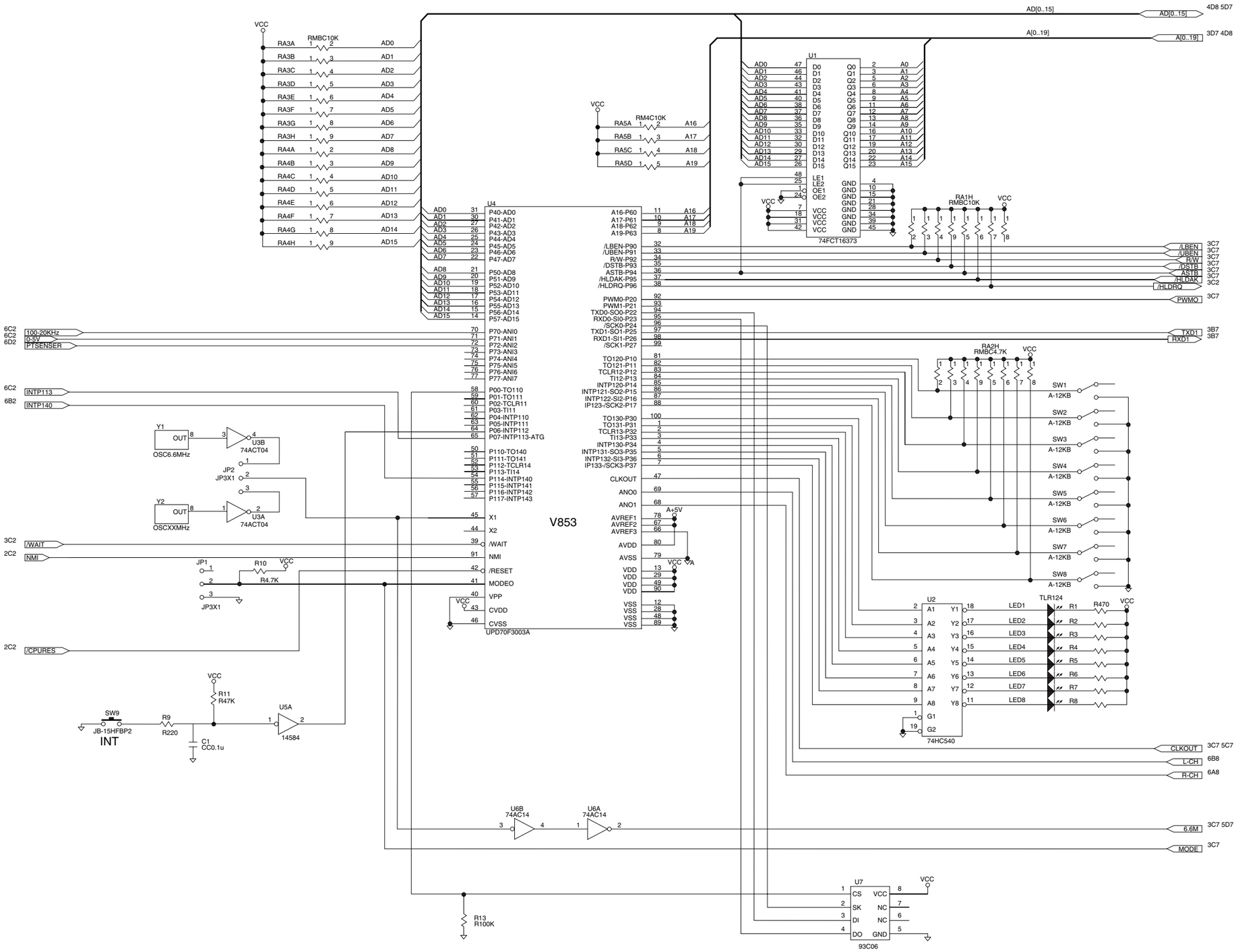


**Figure C-2. Power and Switches**

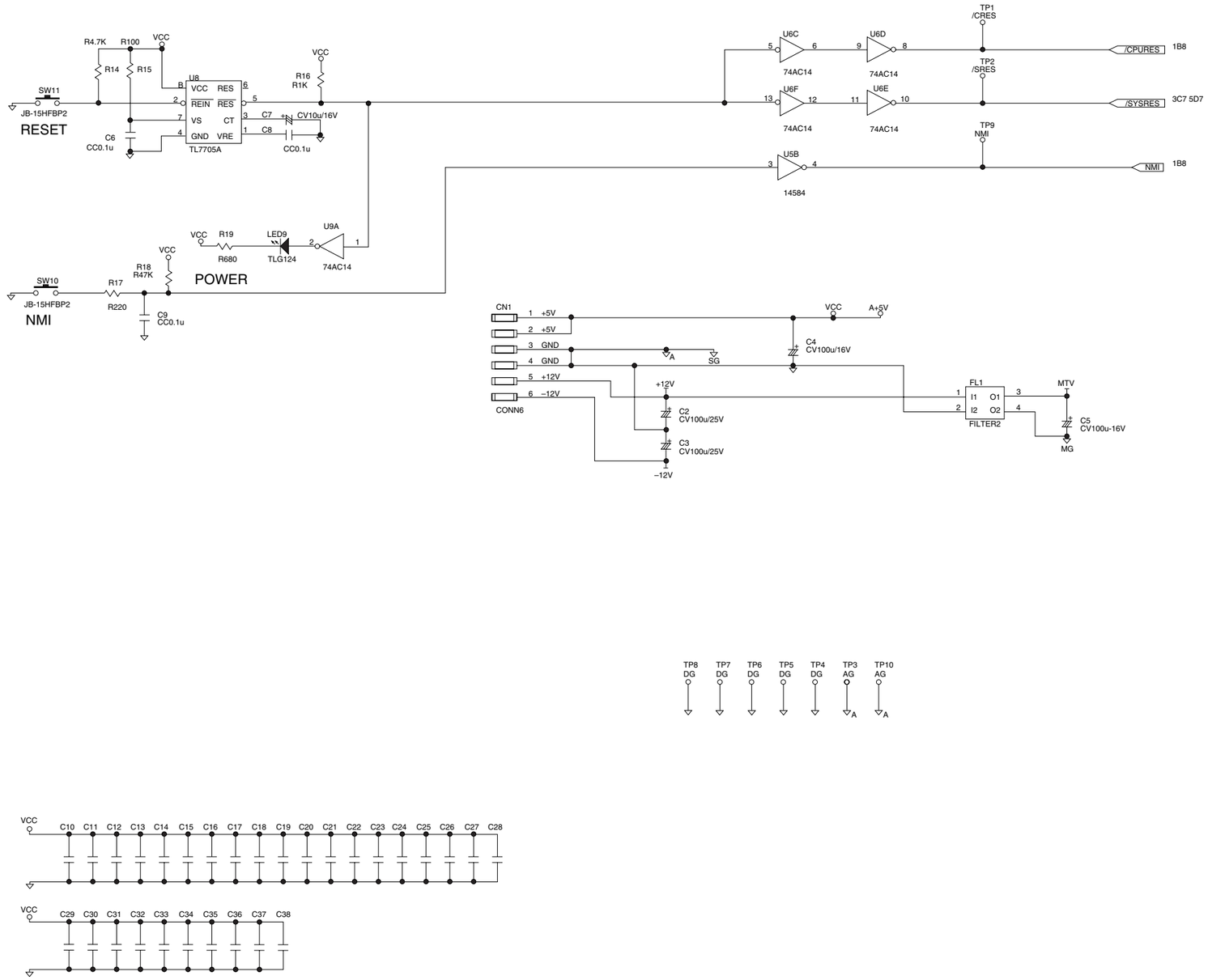


**Figure C-3. Device Controller**

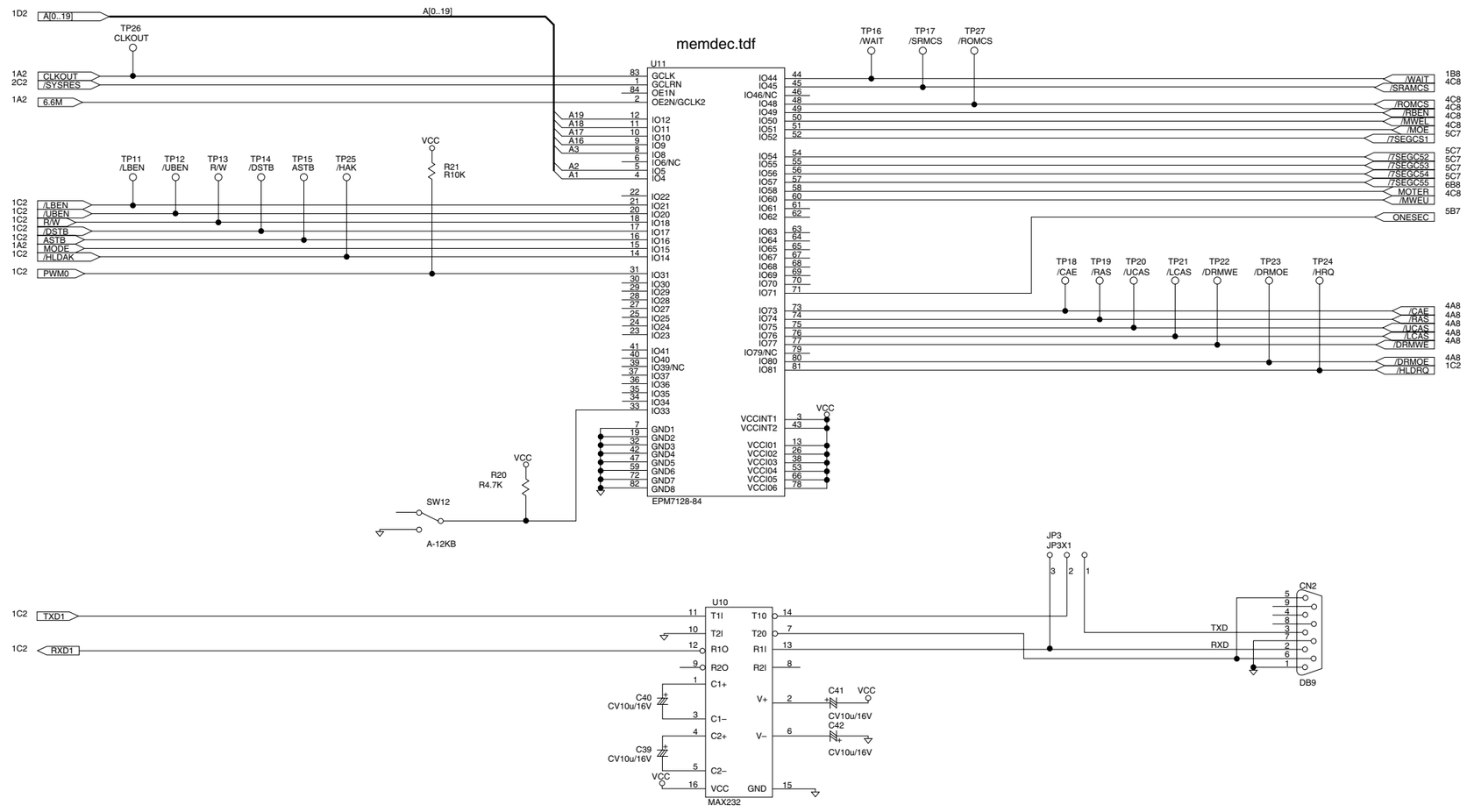


**Figure C-4. Memory Peripheral Circuit**

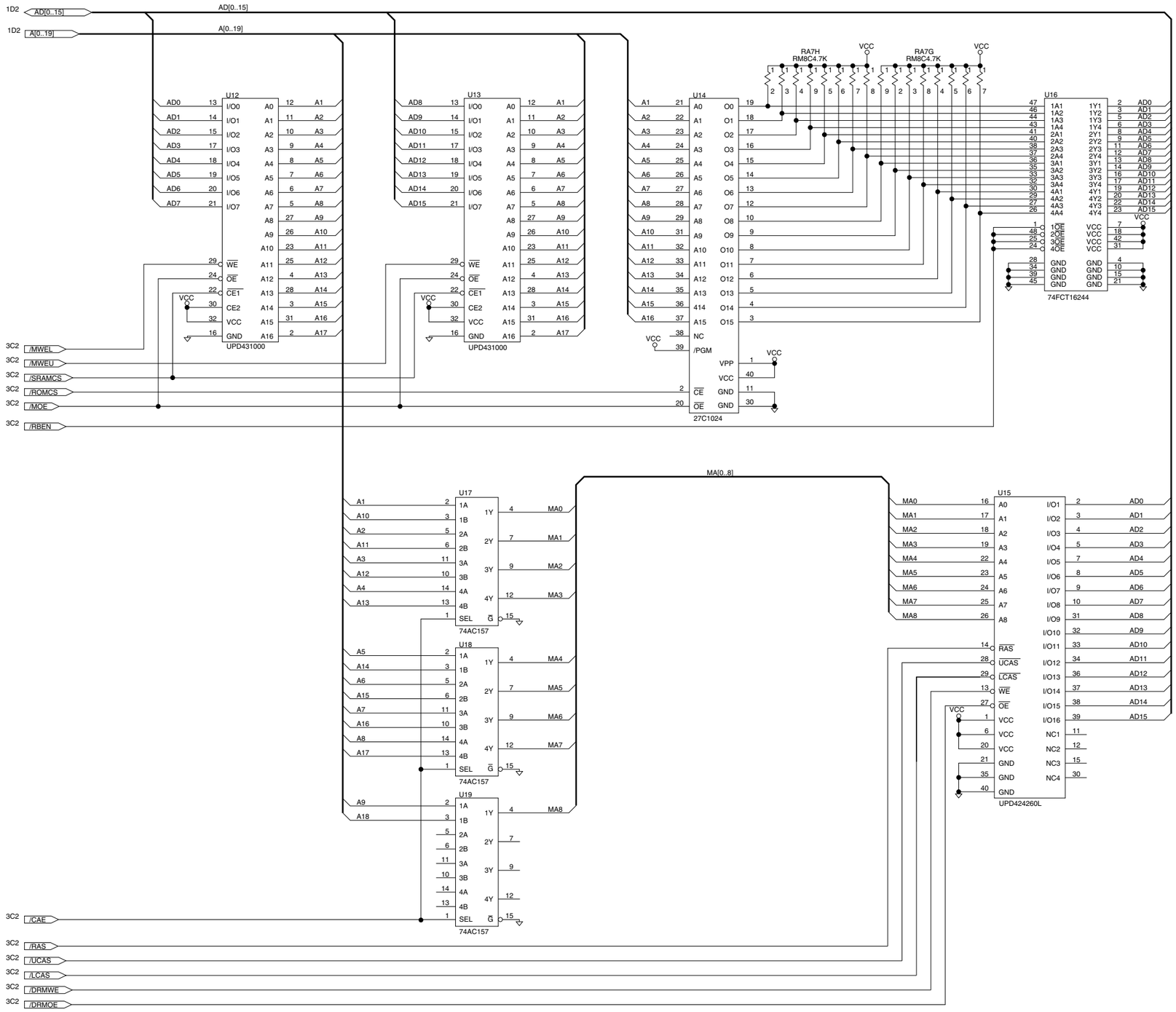


**Figure C-5. LED Controller**

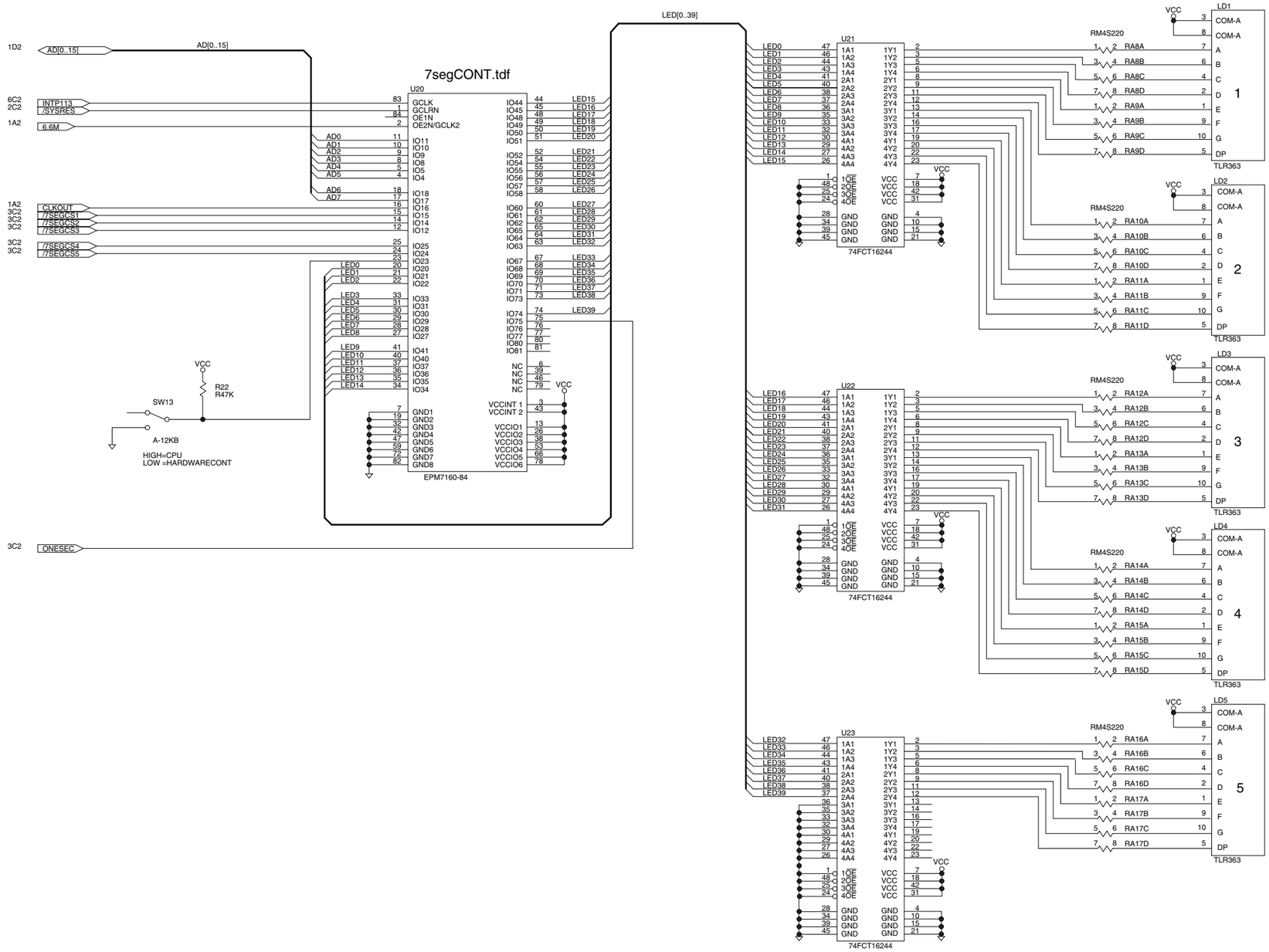
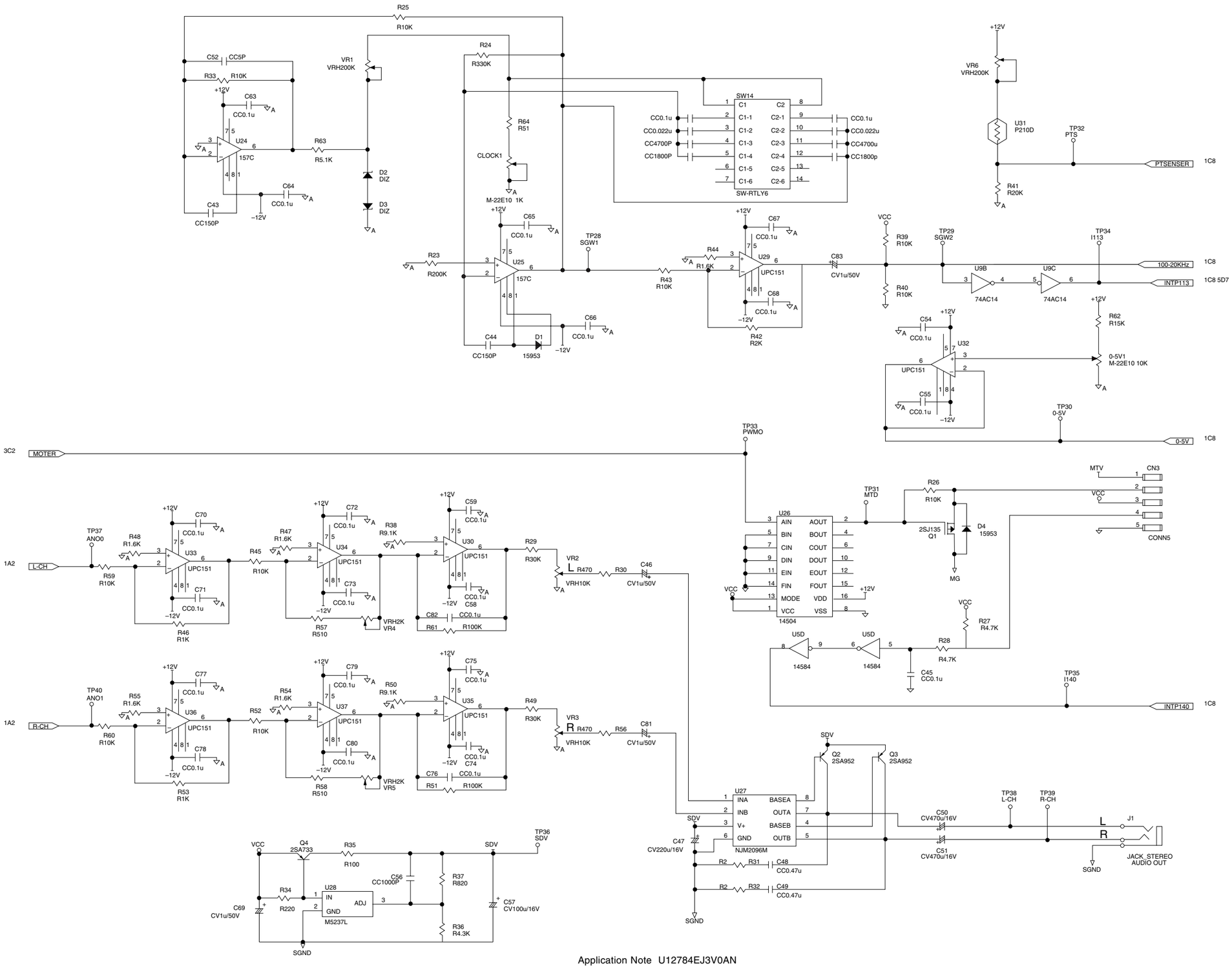


**Figure C-6. I/O Circuit**



# APPENDIX D INDEX

**[N]**



**[O]**



**[P]**



**[R]**



**[S]**



**[T]**

NEC

# Facsimile Message

From:

Name

Company

Tel. FAX

Address

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.

*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
Market Communication Dept.
Fax: +49-211-6503-274

**South America**
NEC do Brasil S.A.
Fax: +55-11-6462-6829

**Hong Kong, Philippines, Oceania**
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

**Korea**
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

**P.R. China**
NEC Electronics Shanghai, Ltd.
Fax: +86-21-6841-1137

**Taiwan**
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

**Asian Nations except Philippines**
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

**Japan**
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608


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 | ❑ | ❑ | ❑ | ❑ |