To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

RENESAS

Application Note

V850/SA1™

32-Bit Single-Chip Microcontrollers

μPD703014A
μPD703014AY
μPD703014B
μPD703014BY
μPD703015A
μPD703015AY
μPD703015B
μPD703015BY
μPD703017A
μPD703017AY
μPD70F3015B
μPD70F3015BY
μPD70F3017A
μPD70F3017AY

Document No. U13851EJ2V0AN00 (2nd edition)
Date Published July 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.

Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

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: μPD70F3015B, 70F3015BY, 70F3017A, 70F3017AY
The customer must judge
The need for license: μPD703014A, 703014AY, 703015B, 703014BY,
μPD703015A, 703015AY, 703015B, 703015BY, 703017A, 703017AY

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

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
  Fax: 091-504 28 60

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
  Fax: 01-30-67 58 99

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
  Fax: 02-66 75 42 99

- **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

- **United Kingdom Branch**
  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

J02.4

Major Revisions in This Edition

Page	Description
Throughout	Addition of μPD703014B, 703014BY, 703015B, 703015BY, 70F3015B, 70F3015BY
p. 14	Addition of **Table 1-1 List of V850/SA1 Products**
p. 15	Addition of description to Minimum instruction execution time in **1.2 Features**
p. 21	Deletion of description from **1.6.2 (2) Bus control unit (BCU)**
p. 26	Change of purpose of r2 in **1.8 CPU Register Set**
p. 27	Change of purpose and operation of r2 in **1.8.1 (1) General-purpose registers**
p. 42	Addition of flash memory programming mode control register (FLPMC) in **1.13 Peripheral I/O Registers**
p. 43	Addition of **Notes** in **1.13 Peripheral I/O Registers**
p. 41 in previous edition	Deletion of **2.1.1 Connection to BV_{DD} pin**

The mark ★ shows major revised points.

INTRODUCTION

Readers	This manual is intended for users who understand the V850/SA1 (μPD703014A, 703014AY, 703014B, 703014BY, 703015A, 703015AY, 703015B, 703015BY, 703017A, 703017AY, 70F3015B, 70F3015BY, 70F3017A, 70F3017AY) functions and who design application systems using these products.
Purpose	The purpose of these application notes is to help the user understand the composition of the V850/SA1 training board (TB-V850/SA1), an example of a system that uses the V850/SA1.
Organization	These application notes are broadly divided into the following sections. • General description of V850/SA1 • Bus interface-connected circuit examples • Examples of connected peripheral functions • Application examples
How to Read This Manual	In these application notes, it is assumed that the reader has general knowledge of electrical engineering, logic circuits, and microcontrollers. For details of V850/SA1 hardware functions → Refer to the **V850/SA1 Hardware User's Manual**. For details of V850/SA1 instruction functions → Refer to the **V850 Series™ Architecture User's Manual**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{\times\times\times}$ (overscore over pin or signal name)
Memory map address:	High order at high stage and low order at low stage
Note:	Footnote for item marked with **Note** in the text.
Caution:	Information requiring particular attention.
Remark:	Supplementary information
Numeric representation:	Binary… ×××× or ××××B
	Decimal… ××××
	Hexadecimal… ××××H

Suffix representing an exponent of 2 (in address space or memory capacity)

K (Kilo): $2^{10} = 1024$
M (Mega): $2^{20} = 1024^2$
G (Giga): $2^{30} = 1024^3$

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

Documents related to V850/SA1

Document Name	Document No.
V850 Series Architecture User's Manual	U10243E
μPD703014A, 703014AY, 703014B, 703014BY, 703015A, 703015AY, 703015B, 703015BY, 703017A, 703017AY Data Sheet	U14526E
μPD703017A, 703017AY Data Sheet	U14527E
V850/SA1 Hardware User's Manual	U12768E
V850/SA1 Application Note	This manual

Documents related to development tools (User's Manuals)

Document Name		Document No.
IE-703002-MC (In-Circuit Emulator)		U11595E
IE-703017-MC-EM1 (V850/SA1 Peripheral I/O Board)		U12898E
CA850 Ver. 2.40 or Later C Compiler Package	Operation	U15024E
	C Language	U15025E
	Project Manager	U15026E
	Assembly Language	U15027E
ID850 Ver. 2.40 Integrated Debugger	Operation (Windows™ Based)	U15181E
SM850 Ver. 2.40 System Simulator	Operation (Windows Based)	U15182E
SM850 Ver. 2.00 or Later System Simulator	External Part User Open Interface Specifications	U14873E
RX850 Ver. 3.13 or Later Real-time OS	Basic	U13430E
	Installation	U13410E
	Technical	U13431E
RX850 Pro Ver. 3.13 or Later Real-time OS	Basic	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		U15260E

CONTENTS

LIST OF FIGURES (1/2)

LIST OF FIGURES (2/2)

LIST OF TABLES

CHAPTER 1 GENERAL DESCRIPTION OF V850/SA1

The V850/SA1 is one of the low power series products in NEC's V850 Series of single-chip microcontrollers for real-time control.

1.1 General

The V850/SA1 is a 32-bit single-chip microcontroller that uses a V850 Series CPU core and includes ROM, RAM, and various peripheral functions such as a timer/counter, a serial interface, A/D converter, and DMA controller.

In addition to its real-time responsiveness and one-clock-pitch execution of instructions, the V850/SA1 includes a hardware multiplier for multiplication instructions, saturation instructions, and bit manipulation instructions, all of which are instructions suited to digital servo control applications. As a real-time control system, this device provides a high level of cost performance suitable for applications ranging from low-power camcorders and other AV equipment to portable telephone equipment such as cellular phones and PHS phone systems.

The following shows a list of the V850/SA1 products.

★ **Table 1-1. List of V850/SA1 Products**

Product Name	I^2C Function	ROM		RAM Size	Package
		Type	Size		
μPD703014A	Not available	Mask ROM	64 KB	4 KB	121-pin FBGA (12 × 12)
μPD703014AY	Available				
μPD703014B	Not available				100-pin LQFP (14 × 14)
μPD703014BY	Available				
μPD703015A	Not available		128 KB		121-pin FBGA (12 × 12)
μPD703015AY	Available				
μPD703015B	Not available				100-pin LQFP (14 × 14)
μPD703015BY	Available				
μPD703017A	Not available		256 KB	8 KB	100-pin LQFP (14 × 14)/ 121-pin FBGA (12 × 12)
μPD703017AY	Available				
μPD70F3015B	Not available	Flash memory	128 KB	4 KB	100-pin LQFP (14 × 14)
μPD70F3015BY	Available				
μPD70F3017A	Not available		256 KB	8 KB	100-pin LQFP (14 × 14)/ 121-pin FBGA (12 × 12)
μPD70F3017AY	Available				

1.2 Features

O Number of instructions:	74
★ O Minimum instruction execution time:	50 ns (when main clock is operating at f_{XX} = 20 MHz) 58.8 ns (when main system clock is operating at f_{XX} = 17 MHz) 30.5 μs (when subsystem clock is operating at f_{XT} = 32.768 kHz)
O General-purpose registers:	32 bits $\times$ 32 registers
O Instruction set:	Signed multiplication ($16 \times 16 \rightarrow 32$): 100 ns (operating at 20 MHz) (able to execute instructions in parallel continuously without creating any register hazards). Saturation operations (overflow and underflow detection functions are included) 32-bit shift instruction: 1 clock Bit manipulation instructions Load/store instructions with long/short format
O Memory space:	16 MB of linear address space (for programs and data) External expandability: Expandable to 4 MB Memory block allocation function: 2 MB per block Programmable wait function Idle state insertion function
O External bus interface:	16-bit data bus (address/data multiplex) Address bus: Separate output enabled Bus hold function External wait function
★ O On-chip memory:	μPD703014A, 703014AY, 703014B, 803014BY (ROM: 64 KB/RAM: 4 KB) μPD703015A, 703015AY, 703015B, 703015BY (ROM: 128 KB/RAM: 4 KB) μPD703017A, 703017AY (ROM: 256 KB, RAM: 8 KB) μPD70F3015B, 70F3015BY (Flash memory: 128 KB/RAM: 4 KB) μPD70F3017A, 70F3017AY (Flash memory: 256 KB/RAM: 8 KB)
O Interrupts and exceptions:	External interrupts: 8 (5[Note]) sources Internal interrupts: 24 sources Software exceptions: 32 sources Exception traps: 1 source Interrupt priority levels are variable (8 levels) **Note** Number of external interrupts that can cancel software STOP mode
O I/O lines:	Total: 85 (13 input ports and 72 I/O ports)
O Timer/counter:	16-bit timer: 2 channels (PWM output) 8-bit timer: 4 channels (PWM output, cascade connection enabled)
O Watch timer:	Subclock or main clock operation: 1 channel
O Watchdog timer:	1 channel

O Serial interface (SIO):	Asynchronous serial interface (UART)
	Clocked serial interface (CSI)
	I^2C bus interface (I^2C)
★	(μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY only)
	UART: 1 channel
	CSI: 1 channel
	UART/CSI: 1 channel
	I^2C/CSI: 1 channel
	UART dedicated baud rate generator: 2 channels
O A/D converter:	10-bit resolution: 12 channels
O DMA controller:	Internal RAM ←→ on-chip peripheral I/O: 3 channels
O Real-time output support:	8 bits × 1 ch or 4 bits × 2 ch
O Clock generator:	Main clock or subclock operation
	5-level CPU clock (including clock-through mode and subclock operations)
O Power saving functions:	HALT/IDLE/software STOP modes
O Package:	100-pin plastic LQFP (fine pitch) (14 × 14 mm)
	121-pin plastic FBGA (12 × 12 mm)
O CMOS structure:	Full static circuits

1.3 Application

General battery-driven equipment such as camcorders (including DVC), meters, etc.

★ 1.4 Ordering Information

Part Number	Package	Internal ROM
μPD703014AF1-×××-EA6	121-pin plastic FBGA (12×12)	64 KB (mask ROM)
μPD703014AYF1-×××-EA6	121-pin plastic FBGA (12×12)	64 KB (mask ROM)
μPD703014BGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	64 KB (mask ROM)
μPD703014BYGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	64 KB (mask ROM)
μPD703015AF1-×××-EA6	121-pin plastic FBGA (12×12)	128 KB (mask ROM)
μPD703015AYF1-×××-EA6	121-pin plastic FBGA (12×12)	128 KB (mask ROM)
μPD703015BGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	128 KB (mask ROM)
μPD703015BYGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	128 KB (mask ROM)
μPD703017AF1-×××-EA6	121-pin plastic FBGA (12×12)	256 KB (mask ROM)
μPD703017AGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	256 KB (mask ROM)
μPD703017AYF1-×××-EA6	121-pin plastic FBGA (12×12)	256 KB (mask ROM)
μPD703017AYGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14×14)	256 KB (mask ROM)
μPD70F3015BGC-8EU	100-pin plastic LQFP (fine pitch) (14×14)	128 KB (flash memory)
μPD70F3015BYGC-8EU	100-pin plastic LQFP (fine pitch) (14×14)	128 KB (flash memory)
μPD70F3017AF1-EA6	121-pin plastic FBGA (12×12)	256 KB (flash memory)
μPD70F3017AGC-8EU	100-pin plastic LQFP (fine pitch) (14×14)	256 KB (flash memory)
μPD70F3017AYF1-EA6	121-pin plastic FBGA (12×12)	256 KB (flash memory)
μPD70F3017AYGC-8EU	100-pin plastic LQFP (fine pitch) (14×14)	256 KB (flash memory)

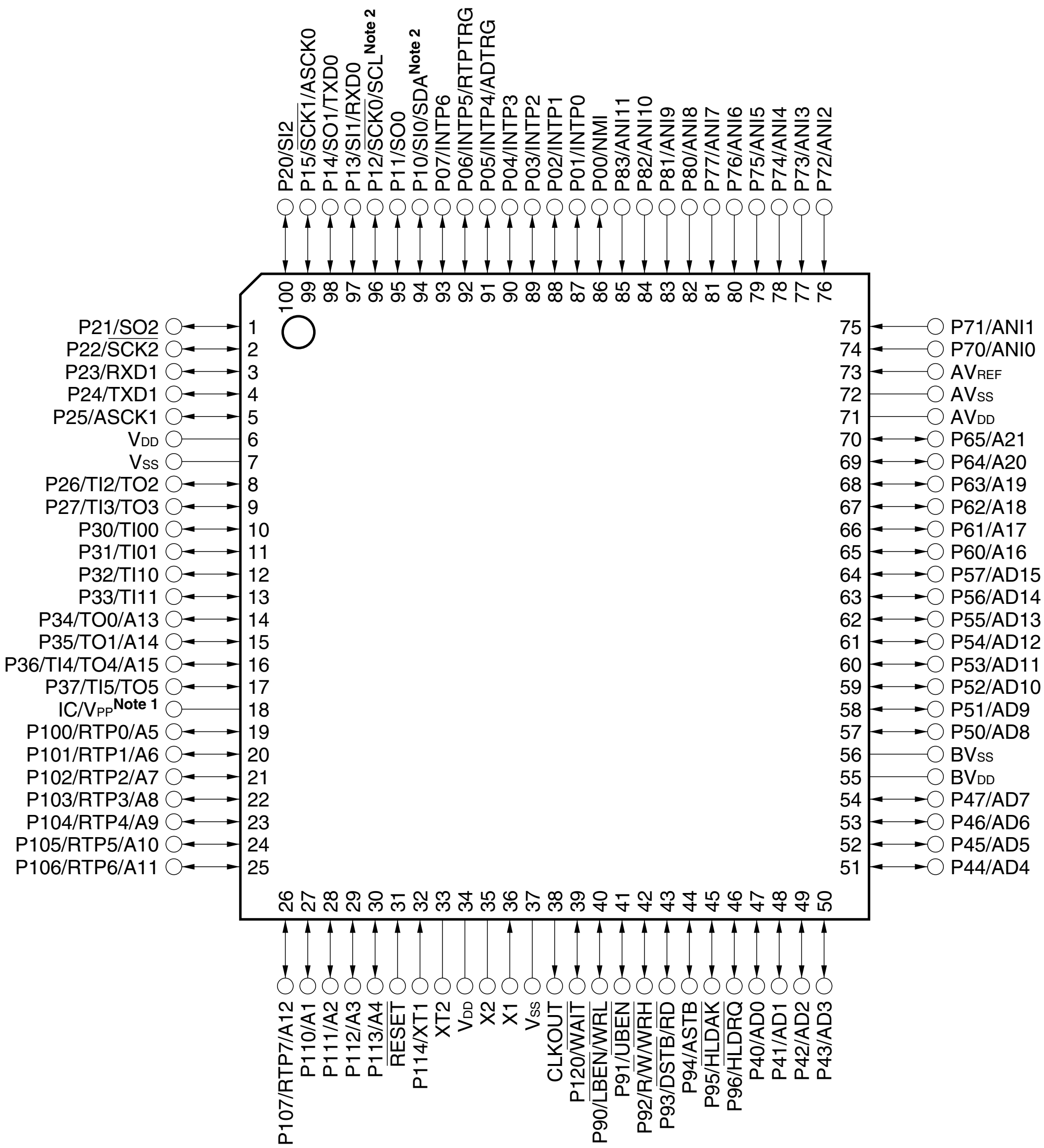
Remarks 1. “×××” indicates ROM code suffix.

2. The V850/SA1 does not include any ROMless versions.

1.5 Pin Configuration (Top View)

★ 100-pin plastic LQFP (fine pitch) (14 × 14 mm)

- μPD703014BGC-×××-8EU
- μPD703014BYGC-×××-8EU
- μPD703015BGC-×××-8EU
- μPD703015BYGC-×××-8EU
- μPD703017AGC-×××-8EU
- μPD703017AYGC-×××-8EU
- μPD70F3015BGC-×××-8EU
- μPD70F3015BYGC-×××-8EU
- μPD70F3017AGC-8EU
- μPD70F3017AYGC-8EU

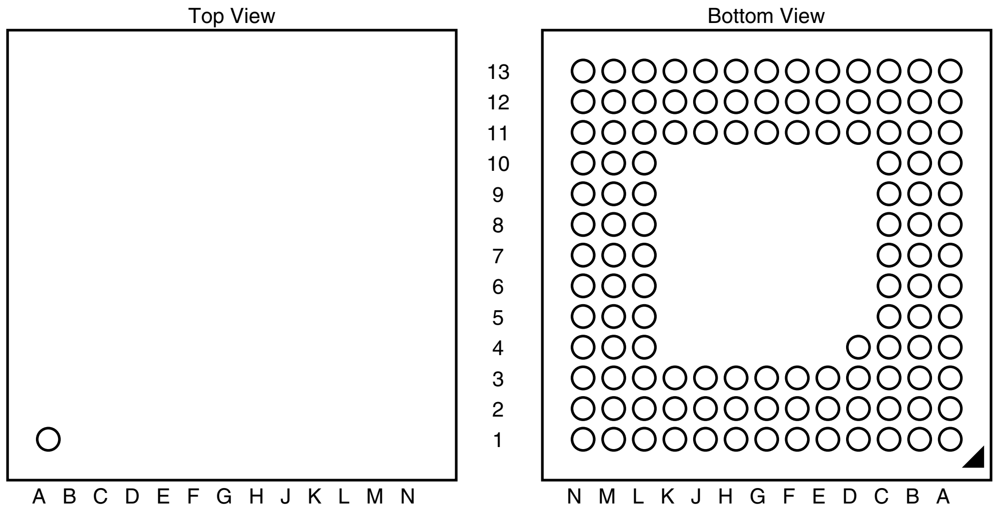


Notes 1. μPD703014B, 703014BY, 703015B, 703015BY, 703017A, 703017AY: IC (Connect directly to V_{SS})
μPD70F3015B, 70F3015BY, 70F3017A, 70F3017AY: V_{PP} (Connect directly to V_{SS} in normal operation mode)

2. The SCL and SDA pins are valid only for the μPD703014BY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY.

★ 121-pin plastic FBGA (12 × 12 mm)

- μPD703014AF1-×××-EA6
- μPD703014AYF1×××-EA6
- μPD703015AF1-×××-EA6
- μPD703015AYF1-×××-EA6
- μPD703017AF1-×××-EA6
- μPD703017AYF1-×××-EA6
- μPD70F3017AF1-EA6
- μPD70F3017AYF1-EA6



Pin No.	Name	Pin No.	Name	Pin No.	Name	Pin No.	Name	Pin No.	Name	Pin No.	Name
A1	P20	B8	P83	D2	V_{DD}	G11	P60	K13	BV_{DD}	M7	V_{SS}
A2	P15	B9	P80	D3	V_{SS}	G12	P56	L1	P104	M8	V_{SS}
A3	V_{SS}	B10	P75	D11	AV_{DD}	G13	P57	L2	P105	M9	P92
A4	P13	B11	AV_{SS}	D12	AV_{DD}	H1	P34	L3	$\overline{RESET}$	M10	P95
A5	P11	B12	AV_{SS}	D13	AV_{DD}	H2	P37	L4	V_{DD}	M11	P41
A6	P06	B13	P71	E1	P25	H3	P35	L5	V_{SS}	M12	P45
A7	P03	C1	P22	E2	V_{DD}	H11	P55	L6	X2	M13	P44
A8	P00	C2	P23	E3	P30	H12	P53	L7	P90	N1	P107
A9	P81	C3	V_{SS}	E11	AV_{DD}	H13	P54	L8	P120	N2	P110
A10	P76	C4	P24	E12	P64	J1	IC/V_{PP} Note	L9	P93	N3	P112
A11	P73	C5	P07	E13	P65	J2	IC/V_{PP} Note	L10	P96	N4	V_{DD}
A12	P72	C6	P04	F1	P26	J3	P100	L11	BV_{SS}	N5	XT1
A13	AV_{SS}	C7	P01	F2	P27	J11	P52	L12	BV_{SS}	N6	V_{SS}
B1	P21	C8	P82	F3	P33	J12	P50	L13	BV_{SS}	N7	V_{SS}
B2	P14	C9	P77	F11	P63	J13	P51	M1	P106	N8	CLKOUT
B3	V_{SS}	C10	P74	F12	P61	K1	P101	M2	P111	N9	P91
B4	P12	C11	AV_{SS}	F13	P62	K2	P102	M3	P113	N10	P94
B5	P10	C12	P70	G1	P31	K3	P103	M4	V_{DD}	N11	P40
B6	P05	C13	AV_{REF}	G2	P32	K11	P46	M5	XT2	N12	P42
B7	P02	D1	V_{DD}	G3	P36	K12	P47	M6	X1	N13	P43

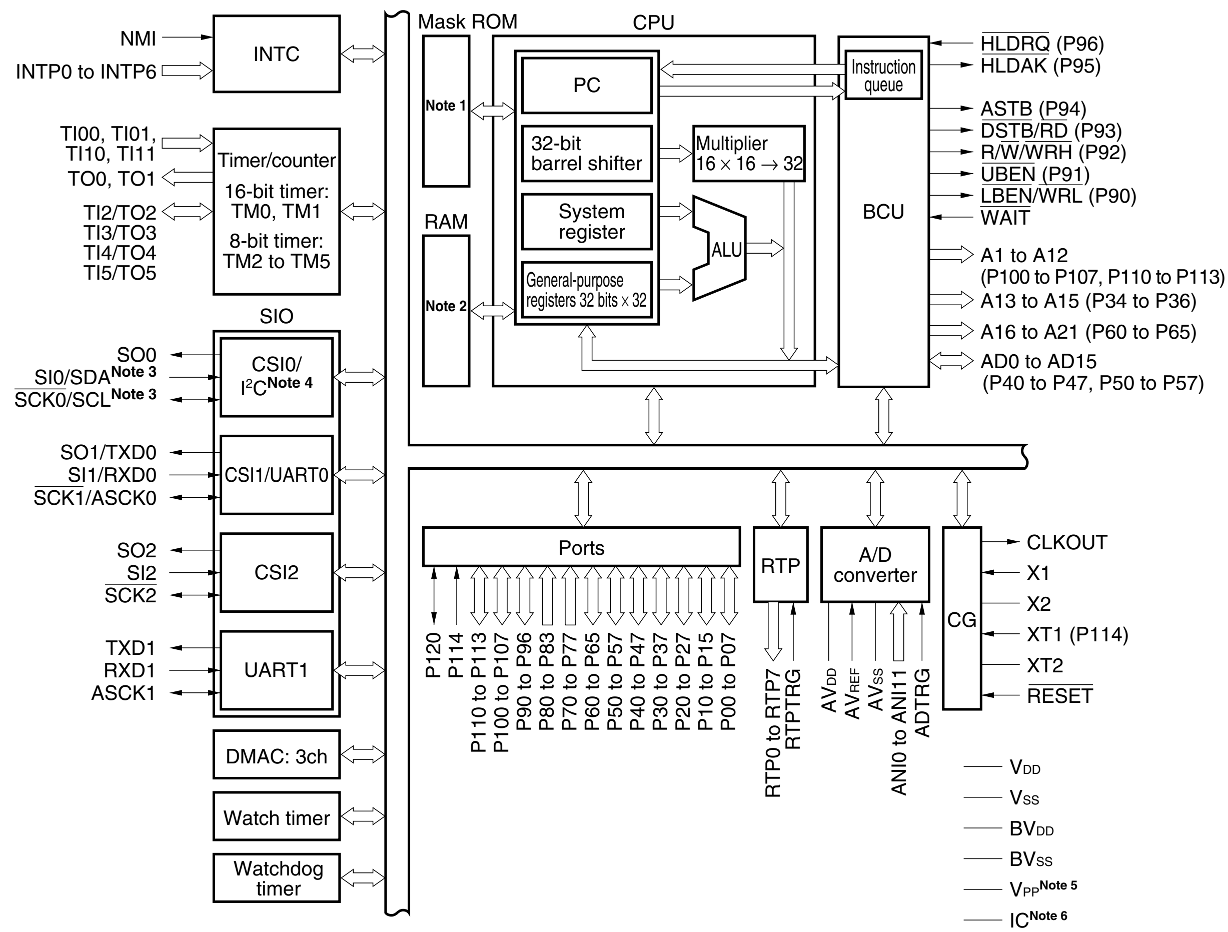
Note μPD703014A, 703014AY, 703015A, 703015AY, 703017A, 703017AY: IC (Connected directly to V_{SS})
μPD70F3017A, 70F3017AY: V_{PP} (Connect directly to V_{SS} in normal operation mode)

Remarks 1. Names of shared (alternate-function) pins are omitted. Shared pins are the same as in the 100-pin plastic LQFP version.
Note that the SCL and SDA pins are valid only for the µPD703014AY, 703015AY, 703017AY, and 70F3017AY.

2. Connect the D4 pin directly to V_{SS}.

1.6 Function Blocks

★ 1.6.1 Internal block diagram



Notes 1. μPD703014A, 703014AY, 703014B, 703014BY: 64 KB (mask ROM)
μPD703015A, 703015AY, 703015B, 703015BY: 128 KB (mask ROM)
μPD703017A, 703017AY: 256 KB (mask ROM)
μPD703015B, 70F3017BY: 128 KB (flash memory)
μPD70F3017A, 70F3017AY: 256 KB (flash memory)

2. μPD703014A, 703014AY, 703014B, 703014BY, 703015A, 703015AY, 703015B, 703015BY, 70F3015B, 70F3015BY: 4 KB
μPD703017A, 703017AY, 70F3017A, 70F3017AY: 8 KB

3. The SCL and SDA pins are valid only for the μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY.

4. The I^2C function is valid only for the μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY.

5. μPD70F3015B, 70F3015BY, 70F3017A, 70F3017AY

6. μPD703014A, 703014AY, 703014B, 703014BY, 703015A, 703015AY, 703015B, 703015BY, 703017A, 703017AY

1.6.2 On-chip units

(1) CPU

The CPU uses five-stage pipeline control to enable single-clock execution of address calculations, arithmetic logic operations, data transfers, and almost all other instruction processing.

Other dedicated on-chip hardware, such as the multiplier (16 bits × 16 bits → 32 bits) and the barrel shifter (32 bits) help accelerate processing of complex instructions.

★ **(2) Bus control unit (BCU)**

The BCU starts a required external bus cycle based on the physical address obtained by the CPU. When an instruction is fetched from external memory space and the CPU does not send a bus cycle start request, the BCU generates a prefetch address and prefetches the instruction code. The prefetched instruction code is stored in a prefetch queue.

★ **(3) ROM**

This consists of a 64/128/256 KB mask ROM mapped to the address space starting at 00000000H and a 128/256 KB flash memory. Both types of memory can be accessed by the CPU in one clock cycle when an instruction is fetched.

★ **(4) RAM**

This consists of a 4 KB RAM mapped to the address space starting at FFFFE000H or an 8 KB RAM mapped to the address space starting at FFFFD000H.

When accessing data, the CPU can access RAM in one clock cycle.

(5) Interrupt controller (INTC)

This controller handles hardware interrupt requests (NMI, INTP0 to INTP6) from on-chip peripheral hardware and external hardware. Eight levels of interrupt priorities can be specified for these interrupt requests, and multiplexed servicing control can be performed for interrupt sources.

(6) Clock generator (CG)

The clock generator includes two types of oscillators, one each for the main clock (f_{XX}) and the subclock (f_{XT}). It generates five types of clocks (f_{XX}, $f_{XX}/2$, $f_{XX}/4$, $f_{XX}/8$, and f_{XT}), one of which is used as the operating clock (f_{CPU}) for the CPU.

(7) Timer/counter

A 2-channel 16-bit timer/event counter and a 4-channel 8-bit timer/event counter are both on chip, which enables measurement of pulse intervals and frequency as well as programmable pulse output.

The 2-channel 8-bit timer/event counter can be connected via a cascade connection to enable use as a 16-bit timer.

(8) Watch timer

This timer counts the reference time period (0.5 seconds) for counting the clock (the 32.768 kHz subclock or the 16.777 MHz main clock). At the same time, the watch timer can be used as an interval timer for the main clock.

(9) Watchdog timer

A watchdog timer has been included to detect inadvertent program loops, system abnormalities, etc. It can also be used as an interval timer.

When used as a watchdog timer, it generates a non-maskable interrupt request (INTWDT) after an overflow occurs. When used as an interval timer, it generates a maskable interrupt request (INTWDTM) after an overflow occurs.

(10) Serial interface (SIO)

The V850/SA1 includes two kinds of serial interfaces: an asynchronous serial interface (UART0, UART1) and a clocked serial interface (CSI0 to CSI2). These plus the I^2C bus interface comprise four channels. One of these channels is switchable between UART and CSI and another is switchable between CSI and I^2C, while the remaining two channels are fixed (one as UART and one as CSI).

For UART0 and UART1, data is transferred via the TXD0, TXD0, and RXD1 pins.

For CSI0 to CSI2, data is transferred via the SO0 to SO2, SI0 to SI2, and $\overline{\text{SCK0}}$ to $\overline{\text{SCK2}}$ pins.

★ For I^2C, data is transferred via the SDA and SCL pins. I^2C is included in the μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY only.

For UART only, there is a 2-channel dedicated baud rate generator on chip.

(11) A/D converter

This high-speed, high-resolution 10-bit A/D converter includes 12 analog input pins. Conversion uses the successive conversion method.

(12) DMA controller

A 3-channel DMA controller is on chip. This controller transfers data between the internal RAM and peripheral I/O devices in response to interrupt requests sent by on-chip peripheral hardware.

(13) Real-time output port (RTP)

The RTP is a real-time output function that transfers preset 8-bit data to an output latch when an external trigger signal occurs or when there is a matching signal in a timer compare register. It can also be used for 2-channel, 4-bit data.

(14) Ports

As shown below, the following ports have general port functions and control pin functions.

Port	I/O	Port Function	Control Function
P0	8-bit I/O	General port	NMI, external interrupt, A/D converter trigger, RTP trigger
P1	6-bit I/O		Serial interface
P2	8-bit I/O		Serial interface, timer output
P3	8-bit I/O		Timer I/O, external address bus
P4	8-bit I/O		External address/data bus
P5	8-bit I/O		
P6	6-bit I/O		External address bus
P7	8-bit input		A/D converter analog input
P8	4-bit input		
P9	7-bit I/O		External bus interface control signal I/O
P10	8-bit I/O		Real-time output port, external address bus
P11	4-bit I/O, 1-bit input		External address bus, subclock input
P12	1-bit I/O		Wait control

1.7 Pin Functions

(1) Bus interface pins

Pin Name	I/O	PULL	Function
A1 to A15	O	No	Lower address bus used for external memory expansion
A16 to A21	O	No	Higher address bus used for external memory expansion
AD0 to AD15	I/O	No	16-bit multiplexed address/data bus used for external memory expansion
ASTB	O	No	External address strobe signal output
$\overline{\text{DSTB}}$	O	No	External data strobe signal output
$\overline{\text{HLDAK}}$	O	No	Bus hold acknowledge output
$\overline{\text{HLDRQ}}$	I	No	Bus hold request input
$\overline{\text{LBEN}}$	O	No	External data bus's lower byte enable signal output
$\overline{\text{RD}}$	O	No	Read strobe signal output
R/$\overline{\text{W}}$	O	No	External read/write status output
$\overline{\text{UBEN}}$	O	No	External data bus's higher byte enable signal output
$\overline{\text{WAIT}}$	I	No	Control signal input for inserting wait in bus cycle
$\overline{\text{WRH}}$	O	No	Higher byte write strobe signal output for external data bus
$\overline{\text{WRL}}$	O	No	Lower byte write strobe signal output for external data bus

Remark PULL: On-chip pull-up resistor

(2) Peripheral I/O pins

(1/2)

Pin Name	I/O	PULL	Function
TI00	I	Yes	External capture trigger input and external count clock input for TM0
TI01	I	Yes	External capture trigger input for TM0
TI10	I	Yes	External capture trigger input and external count clock input for TM1
TI11	I	Yes	External capture trigger input for TM1
TI2 to TI5	I	Yes	External count clock input for TM2 to TM5
TO0 to TO5	O	Yes	Pulse signal output for TM0 to TM5
INTP0 to INTP3	I/O	Yes	External interrupt request input (analog noise elimination)
INTP4 to INTP6			External interrupt request input (digital noise elimination)
SI0 to SI2	I	Yes	Serial receive data input (3-wire) for CSI0 to CSI2
SO0 to SO2	O	Yes	Serial transmit data output (3-wire) for CSI0 to CSI2
$\overline{\text{SCK0}}$ to $\overline{\text{SCK2}}$	I/O	Yes	Serial clock I/O (3-wire) for CSI0 to CSI2
SCL[Note]	I/O	Yes	Serial clock I/O for I^{2}C
SDA[Note]	I/O	Yes	Serial transmit data I/O for I^{2}C

★ **Note** μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY only.

Remark PULL: On-chip pull-up resistor

(2/2)

Pin Name	I/O	PULL	Function
ASCK0	I	Yes	Serial baud rate clock input for UART0
ASCK1	I	Yes	Serial baud rate clock input for UART1
TXD0	O	Yes	Serial transmit data output for UART0
TXD1	O	Yes	Serial transmit data output for UART1
RXD0	I	Yes	Serial receive data input for UART0
RXD1	I	Yes	Serial receive data input for UART1
ANI0 to ANI11	I	No	Analog input to A/D converter
ADTRG	I	Yes	External trigger input to A/D converter
NMI	I	Yes	Non-maskable interrupt request input (analog noise elimination)
RTP0 to RTP7	O	Yes	Real-time output port
RTPTRG	I	Yes	RTP external trigger input

Remark PULL: On-chip pull-up resistor

(3) Power supply, clock, and reset pins

Pin Name	I/O	PULL	Function
CLKOUT	O	–	On-chip system clock output
$\overline{\text{RESET}}$	I	–	System reset input
X1	I	No	Resonator connection for main clock
X2	–		
XT1	I	No	Resonator connection for subsystem clock
XT2	–		
IC[Note 1]	–	–	Internal connection
AV_{REF}	I	–	Reference voltage input for A/D converter
AV_{DD}	–	–	Positive power supply for A/D converter
AV_{SS}	–	–	Ground potential for A/D converter
BV_{DD}	–	–	Positive power supply for bus interface
BV_{SS}	–	–	Ground potential for bus interface
V_{DD}	–	–	Positive power supply pin
V_{SS}	–	–	Ground potential
V_{PP}[Note 2]	–	–	High-voltage application pins for program write/verify

★ **Notes 1.** μPD703014A, 703014AY, 703014B, 703014BY, 703015A, 703015AY, 703015B, 703015BY, 703017A, and 703017AY only.

2. μPD70F3015B, 70F3015BY, 70F3017A, and 70F3017AY only.

Remark PULL: On-chip pull-up resistor

(4) Port pins

Pin Name	I/O	PULL	Function
P00 to P07	I/O	Yes	Port 0 8-bit I/O port Input/output can be specified in 1-bit units
P10 to P15	I/O	Yes	Port 1 6-bit I/O port Input/output can be specified in 1-bit units
P20 to P27	I/O	Yes	Port 2 8-bit I/O port Input/output can be specified in 1-bit units
P30 to P37	I/O	Yes	Port 3 8-bit I/O port Input/output can be specified in 1-bit units
P40 to P47	I/O	No	Port 4 8-bit I/O port Input/output can be specified in 1-bit units
P50 to P57	I/O	No	Port 5 8-bit I/O port Input/output can be specified in 1-bit units
P60 to P65	I/O	No	Port 6 6-bit I/O port Input/output can be specified in 1-bit units
P70 to P77	I	No	Port 7 8-bit input port
P80 to P83	I	No	Port 8 4-bit input port
P90 to P96	I/O	No	Port 9 7-bit I/O port Input/output can be specified in 1-bit units
P100 to P107	I/O	Yes	Port 10 8-bit I/O port Input/output can be specified in 1-bit units
P110 to P113	I/O	Yes	Port 11 5-bit I/O port Input/output can be specified in 1-bit units P114 only fixed as input port.
P114	I	No	
P120	I/O	No	Port 12 1-bit I/O port

Remark PULL: On-chip pull-up resistor

1.8 CPU Register Set

The V850/SA1's CPU registers can be classified into two types: a general-purpose program register set and special-purpose system register set. All registers are 32 bits wide. For details, refer to the V850 Series Architecture User's Manual.

Program register set

31	0
r0	Zero register
r1	Reserved for address register
★ r2	
r3	Stack pointer (SP)
r4	Global pointer (GP)
r5	Text pointer (TP)
r6	
r7	
r8	
r9	
r10	
r11	
r12	
r13	
r14	
r15	
r16	
r17	
r18	
r19	
r20	
r21	
r22	
r23	
r24	
r25	
r26	
r27	
r28	
r29	
r30	Element pointer (EP)
r31	Link pointer (LP)

31	0
PC	Program counter

System register set

31	0
EIPC	Exception/interrupt PC
EIPSW	Exception/interrupt PSW

31	0
FEPC	Fatal error PC
FEPSW	Fatal error PSW

31	0
ECR	Exception cause register

31	0
PSW	Program status word

1.8.1 Program register set

Program registers include general-purpose registers and a program counter (PC).

★ **(1) General-purpose registers**

A total of 32 program registers (r0 to r31) are provided. All of these registers can be used as data variables or address variables.

However, r0 and r30 are used implicitly by instructions, so caution is required when using these registers. Also, r1, r3 to r15, and r31 are used implicitly by the assembler and C compiler, so be sure to save the contents of these registers before using them and restore the contents afterward so that their contents will not be destroyed. There may be cases when r2 is used by the real-time OS. If the real-time OS to be used does not use r2, r2 can be used as a register for variables.

Table 1-2. List of Program Registers

Name	Application	Operation
r0	Zero register	Always retains “0”
r1	Assembler reserve register	Used as a working register for 32-bit immediate generation
r2	Registers used for address/data variable (when the real-time OS to be used does not use r2)	
r3	Stack pointer	Used to generate stack frame for function calls
r4	Global pointer	Used to access global variables in data area
r5	Text pointer	Used as the register that indicates the start of the text area[Note]
r6 to r29	Registers used for address/data variable	
r30	Element pointer	Used as the base pointer when accessing memory
r31	Link pointer	Used when the compiler makes a function call
PC	Program counter	Retains instruction addresses during program execution

Note The text area is the area where program code is allocated.

(2) Program counter

This register holds the address of the instruction being executed. The lower 24 bits of this register are valid, and bits 31 to 24 are fixed to 0. If a carry occurs from bit 23 to 24, it is ignored.

Bit 0 is fixed to 0, and branching to an odd address is not possible.

After reset: 00000000H

	31 24	23 1	0
PC	Fixed to 0	Instruction address being executed	0

1.8.2 System register set

System registers are used to hold CPU status control information, interrupt information, etc.

Table 1-3. System Register Set

No.	System Register	Application	Operation
0	EIPC	Register for saving interrupt status data	When an exception or interrupt has occurred, the PC and PSW (program status word) are saved to these registers. Since there are only two of these registers, when multiple interrupts are enabled data must be saved to these registers by the program.
1	EIPSW		
2	FEPC	NMI status registers	When an NMI (non-maskable interrupt) has occurred, the PC and PSW are saved to these registers.
3	FEPSW		
4	ECR	Interrupt source register	When an exception, maskable interrupt, or NMI has occurred, the source is retained in this register. The higher 16 bits of this register, called “FECC”, are where the NMI's exception code is set. The lower 16 bits, called “EICC”, are where the exception or interrupt exception code is set.
5	PSW	Program status word	The program status word is a set of flags that indicates the program’s status (instruction execution results) and the CPU status.
6 to 31	Reserved		

To read/write these system registers, specify the system register number indicated by the system register load/store instruction (LDSR or STSR instruction).

(1) Interrupt Source Register (ECR)

After reset: 00000000H

	31 – 16	15 – 0
ECR	FECC	EICC

FECC	NMI exception code (see **V850/SA1 Hardware User’s Manual** for details of exception codes)
EICC	Exception code for exception/interrupt

(2) Program status word (PSW)

After reset: 00000020H

	31–8	7	6	5	4	3	2	1	0
PSW	RFU	NP	EP	ID	SAT	CY	OV	S	Z

RFU	Reserved field (fixed to 0).

NP	Non-maskable interrupt (NMI) processing status
0	NMI processing not in progress.
1	NMI processing in progress. This flag is set (1) when an NMI is acknowledged. This flag disables multiple interrupts.

EP	Exception processing status
0	NMI processing not in progress.
1	NMI processing in progress. This flag is set (1) when an exception is generated Interrupt requests can be acknowledged.

ID	Maskable interrupt processing specification
0	Enables maskable interrupt acknowledgement.
1	Disables maskable interrupt acknowledgement. This flag is set (1) when a maskable interrupt request is acknowledged.

SAT Note	Detection overflow in result of saturated operation instruction
0	Overflow does not occur. Even though the result of executing a saturated operation instruction does not overflow with this flag set (1), this flag will not be cleared (0). To clear this flag (0), write directly to the PSW.
1	Overflow occurs.

CY	Detection of carry or borrow in result of operation
0	Carry or borrow does not occur.
1	Carry or borrow occurs.

OV Note	Detection of overflow during operation
0	Overflow does not occur.
1	Overflow occurs.

S Note	Detection of positive/negative of operation result
0	Result is not negative (positive or zero).
1	Result is negative.

Note The contents of the OV and S bits during a saturation operation determine the operation results of the saturation instruction. The SAT bit is set (1) only when the OV bit is set (1) during a saturation operation.

Z	Detection of zero of operation result
0	Result is not zero
1	Result is zero

Status of operation result	Status of flag			Operation result after saturation instruction
	SAT	OV	S	
More than maximum positive value	1	1	0	7FFFFFFFH
More than maximum negative value	1	1	1	80000000H
Positive (not exceeding maximum value)	Value before operation retained	0	0	Same as the operation result
Negative (not exceeding maximum value)			1	

1.9 Operation Modes

The V850/SA1 provides the following operations modes.

(1) Normal operation mode (single-chip mode)

After the system has been released from the reset status, the pins related to the bus interface are set to port mode, execution branches to the reset entry address of the internal ROM, and the instruction processing written to internal ROM is started. The external expansion mode, in which an external device is connected to the external memory area, is enabled by setting the memory expansion mode register (MM) via an instruction.

(2) Flash memory programming mode

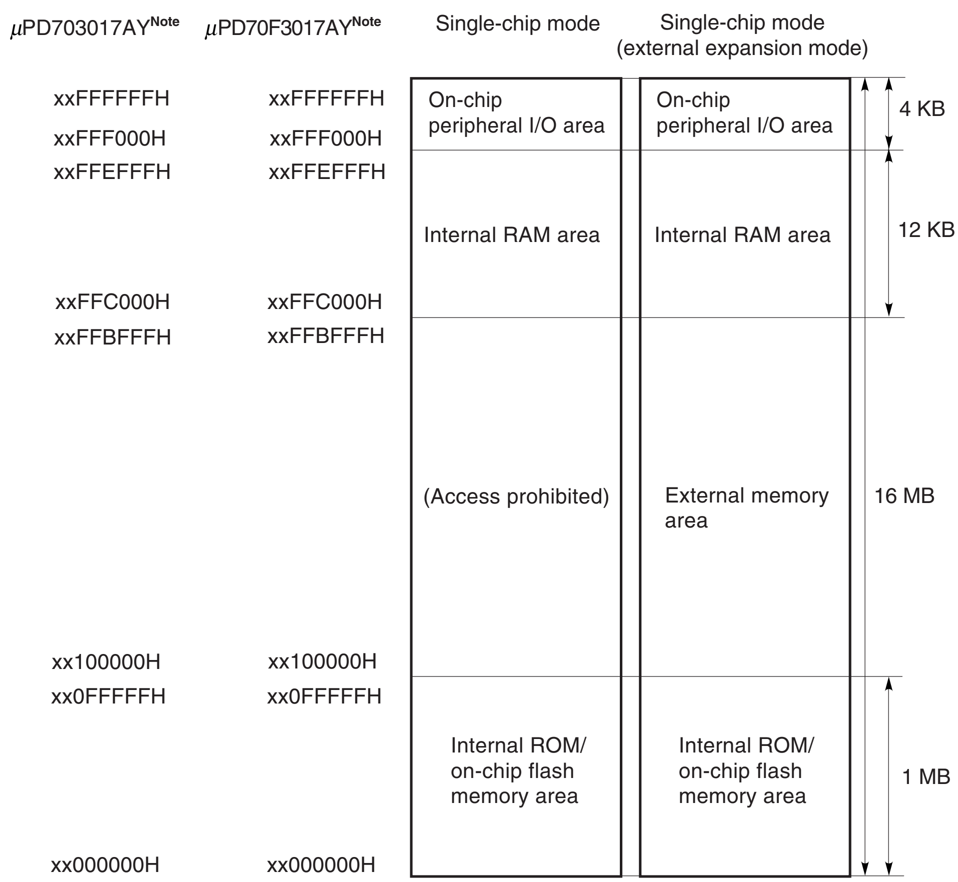
★ This mode is provided only in the μPD70F3015B, 70F3015BY, 70F3017A, and 70F3017AY. The internal flash memory is programmable or erasable only when the V_{PP} voltage is applied to V_{PP} pin.

V_{PP}	Operation Mode
0	Normal operation mode
7.8 V	Flash memory programming mode
V_{DD}	Setting prohibited

1.10 Address Space

The CPU address space is reserved as shown below.

Figure 1-1. Memory Map



Note Here, the μPD703017AY and μPD70F3017AY are given as typical examples.
For details, refer to the **V850/SA1 Hardware User's Manual**.

1.10.1 Interrupt/exception table

The V850/SA1 accelerates interrupt responsiveness by assigning handler addresses corresponding to interrupts/exceptions.

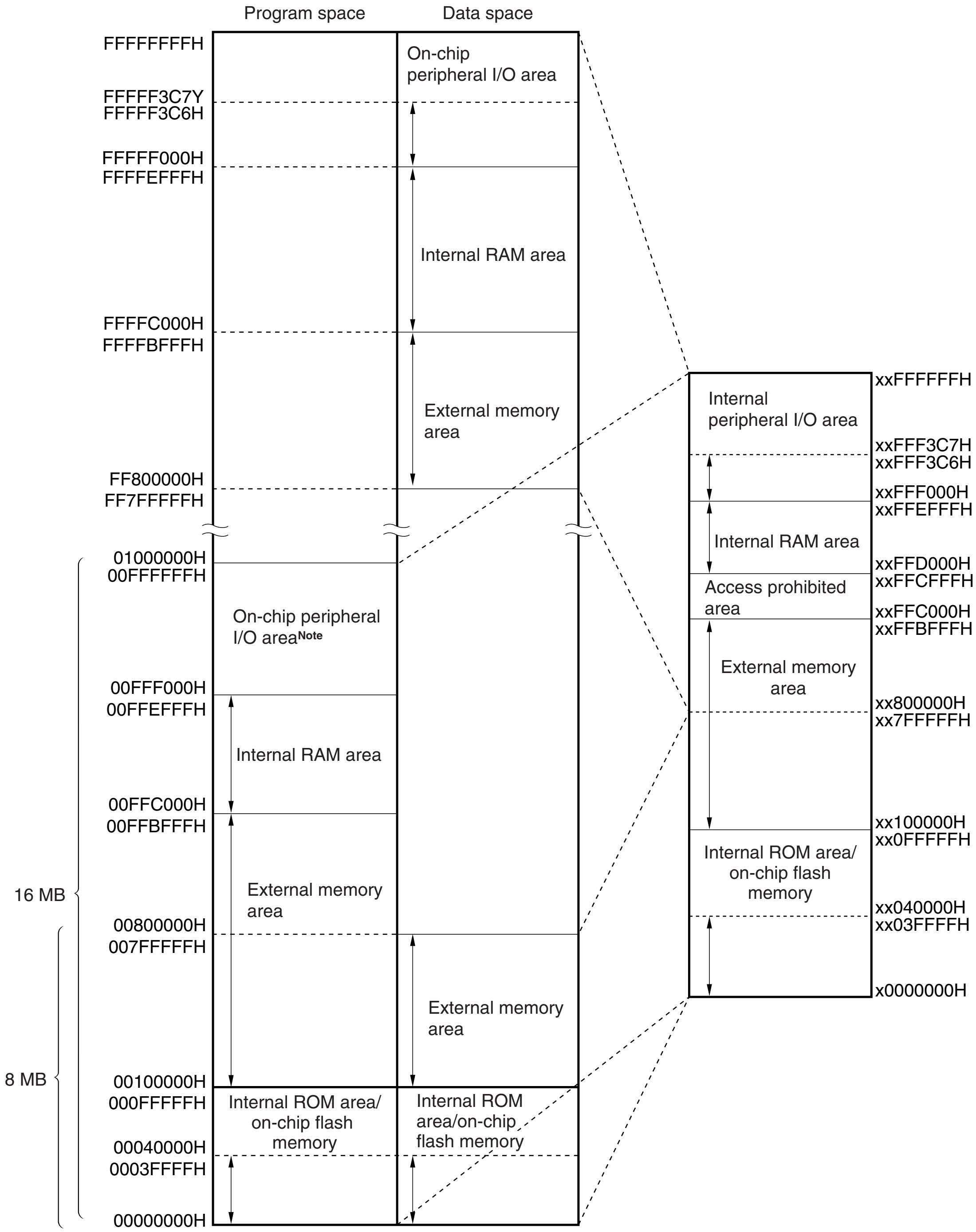
These handler addresses are collected in what is called an interrupt/exception table, which is located in the internal ROM/internal flash memory area. When an interrupt/exception request is acknowledged, processing jumps to the handler address, and the program written at that memory address is executed. The sources of interrupts/exceptions and the corresponding addresses are shown below.

Table 1-4. Interrupt/Exception Table

Start Address of Interrupt/Exception Table	Interrupt/Exception Source
00000000H	RESET
00000010H	NMI
00000020H	INTWDT
00000040H	TRAP0n (n = 0 to F)
00000050H	TRAP1n (n = 0 to F)
00000060H	ILGOP
00000080H	INTWDTM
00000090H	INTP0
000000A0H	INTP1
000000B0H	INTP2
000000C0H	INTP3
000000D0H	INTP4
000000E0H	INTP5
000000F0H	INTP6
00000100H	INTWTI
00000110H	INTTM00
00000120H	INTTM01
00000130H	INTTM10
00000140H	INTTM11
00000150H	INTTM2
00000160H	INTTM3
00000170H	INTTM4
00000180H	INTTM5
00000190H	INTIIC0[Note]/INTCSI0
000001A0H	INTSER0
000001B0H	INTSR0/INTCSI1
000001C0H	INTST0
000001D0H	INTCSI2
000001E0H	INTSER1
000001F0H	INTSR1
00000200H	INTST1
00000210H	INTAD
00000220H	INTDMA0
00000230H	INTDMA1
00000240H	INTDMA2
00000250H	INTWT

★ **Note** Valid only in the μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY.

Figure 1-2. Recommended Memory Map (for Device with On-Chip Flash Memory)



Note This area cannot be used as a program area.

Remarks 1. Vertical arrows (↕) indicate areas recommended for use.

2. The above shows the recommended memory map for the μPD703017A, 703017AY, 70F3017A, and 70F3017AY.

1.11 Pipeline

The V850/SA1 processes each instruction in five stages. One clock cycle is required for each stage, which means that five clock cycles are needed to process an entire instruction. However, since five instructions can be processed via a five-stage pipeline in parallel, almost all instructions can be processed in one clock cycle.

IF	ID	EX	MEM	WB

The five pipeline stages are described below.

(1) IF (Instruction Fetch)

At this stage, the instruction is fetched and the fetch pointer is incremented.

(2) ID (Instruction Decode)

At this stage, the fetched instruction is decoded, immediate data is generated, and register contents are read.

(3) EX (Execution using ALU, multiplier, and barrel shifter)

At this stage, the decoded instruction is executed.

(4) MEM (Memory access)

Memory is accessed at the target address.

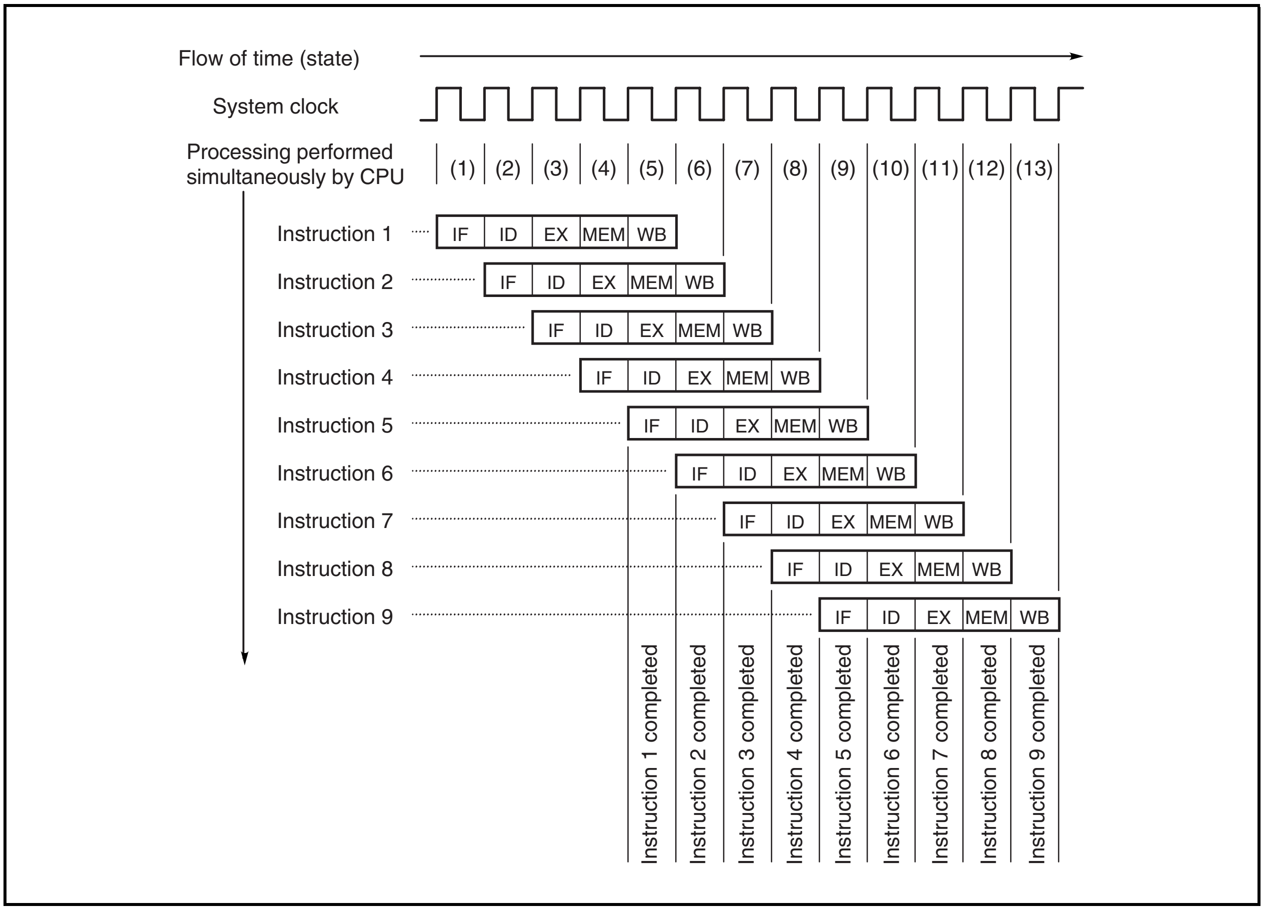
(5) WB (Writeback)

The results of instruction execution are written back to the target registers.

1.11.1 Pipeline operation

The V850/SA1's instruction execution stages consist of five stages from IF (instruction fetch) to WB (writeback). The execution time at each stage differs depending on the type of instruction and the memory area to be accessed. CPU processing during continuous execution of nine typical instructions is illustrated in Figure 1-3 below as an example of pipeline operation.

Figure 1-3. Pipeline Operation (During Continuous Execution of Nine Typical Instructions)

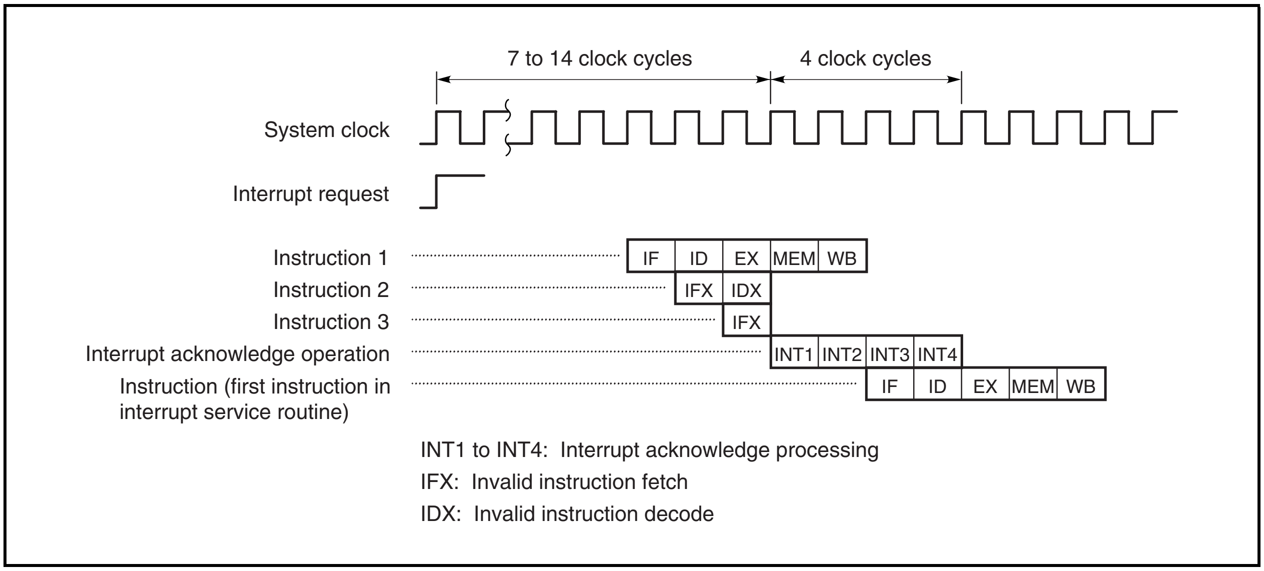


In the figure, (1) to (13) indicate the CPU's states. In each state, writeback for instruction n is performed simultaneously with memory access for instruction n+1, execution of instruction n+2, decoding of instruction n+3, and fetching of instruction n+4. For a typical instruction, the five stages from fetch to writeback require five clock cycles. However, since five instructions can be processed in parallel, a typical instruction can be processed in one clock cycle (on average).

1.11.2 Pipeline operation when interrupt occurs

Figure 1-4 shows an example of pipeline operation when an interrupt has occurred. The operation of acknowledging the interrupt requires four clock cycles. An instruction that has reached the EX stage continues to be executed even when an interrupt is acknowledged. An instruction that has not reached the EX stage is temporarily rendered invalid, and processing of that instruction resumes at the IF stage after the interrupt has been serviced.

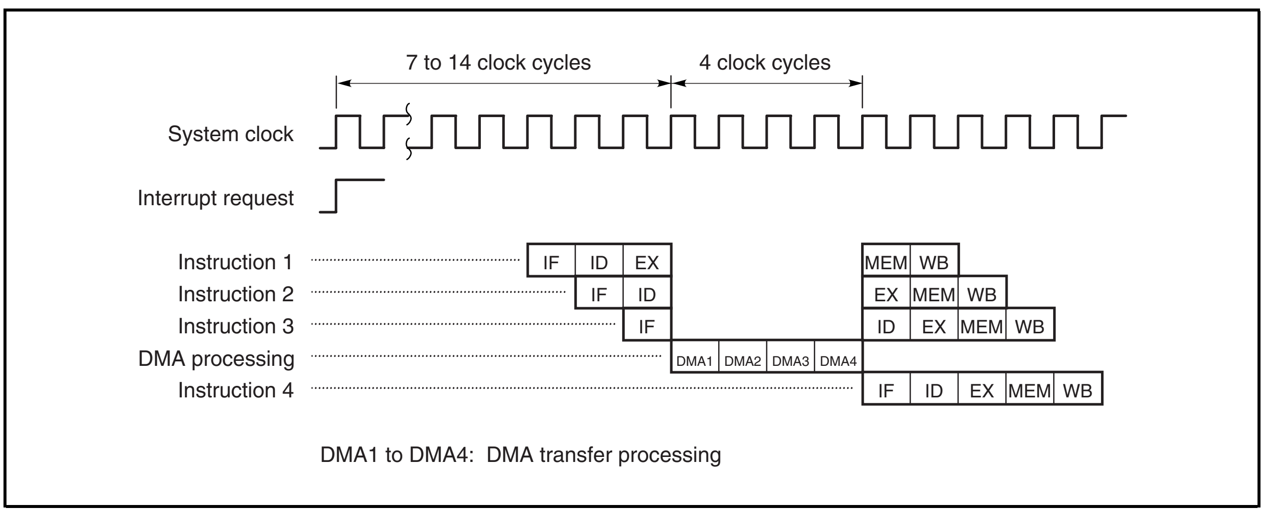
Figure 1-4. Pipeline Operation (When Interrupt Is Acknowledged)



1.11.3 Pipeline operation during DMA transfer

A DMA transfer requires four clock cycles. During a DMA transfer, pipeline operation stops at each stage and resumes from the same stage after DMA transfer has been completed.

Figure 1-5. Pipeline Operation (During DMA Transfer)



1.12 Power Saving Function

1.12.1 Operation modes of power saving function

The power saving function has the following modes.

These modes can be combined and switched to suit the target application, which enables effective implementation of low-power systems.

(1) HALT mode

When in this mode, the clock's oscillator continues to operate but the CPU's operating clock is stopped. A clock continues to be supplied for other on-chip peripheral functions to maintain operation of those functions. This enables the system's total power consumption to be reduced.

(2) IDLE mode

This mode stops the entire system by stopping the CPU's operating clock as well as the operating clock for on-chip peripheral functions while the clock oscillator is still operating. However, the subclock continues to operate and supplies a clock to the on-chip peripheral functions. When this mode is canceled, there is no need for the oscillator to wait for the oscillation stabilization time, so normal operation can be resumed quickly.

(3) Software STOP mode

This mode stops the entire system by stopping the main clock oscillator. The subclock continues to be supplied to keep on-chip peripheral functions operating. If a subclock is not used, low power consumption mode is set. STOP mode setting is prohibited if the CPU is operating via the subclock.

(4) Subclock operation

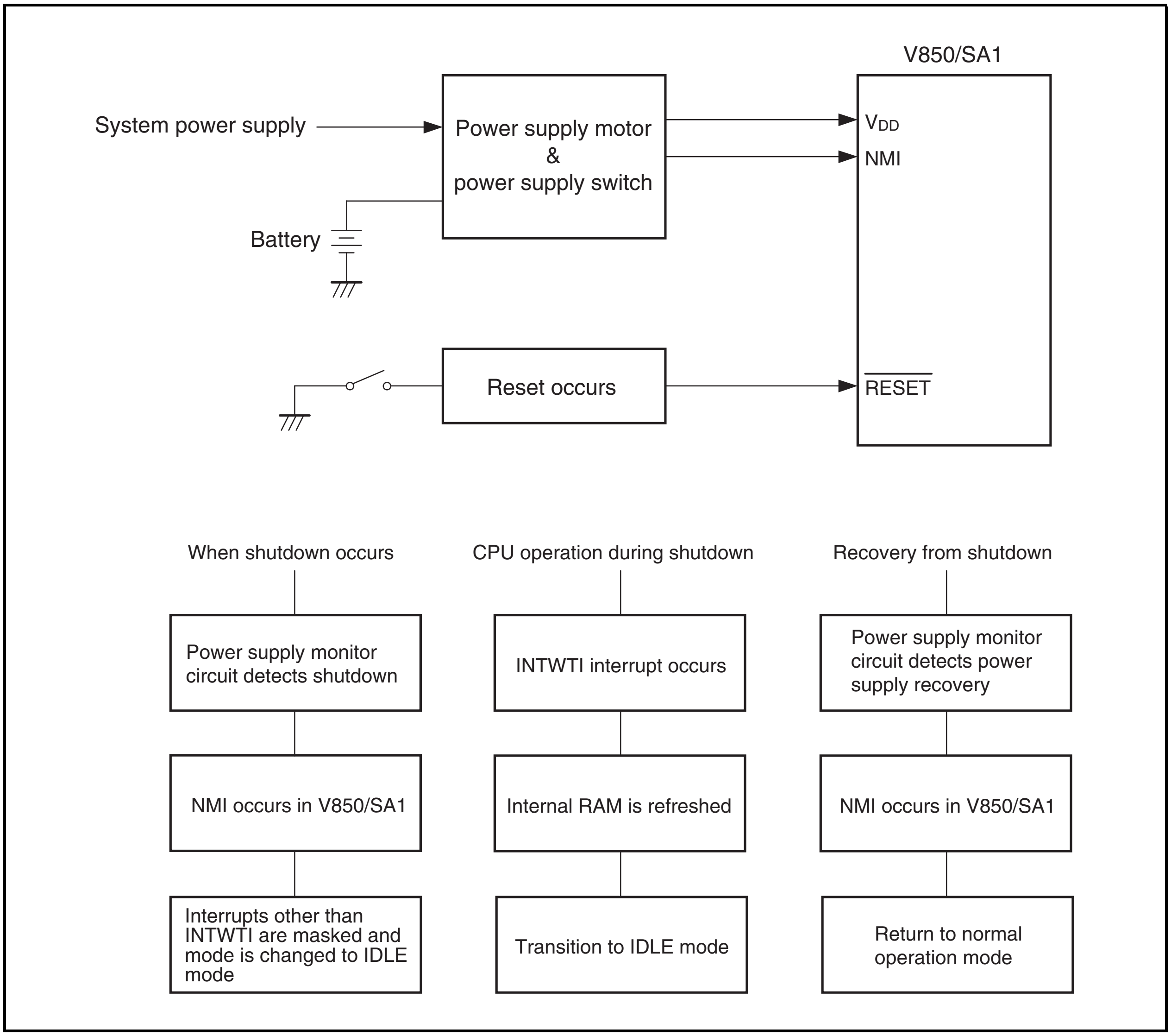
In this mode, the CPU clock is set to operate using the subclock and the PSC register's MCK bit is set (to 1) to set the entire system to low power consumption mode in which the system operates using only the subclock. When IDLE mode has been set, the CPU's operating clock and some peripheral functions (DMAC and BCU) are stopped, so that power consumption can be reduced.

Power Saving Mode	Transition to Mode	Cancellation of Mode
HALT mode	Execution of HALT instruction	$\overline{\text{RESET}}$, NMI, or maskable interrupt that is not masked
IDLE mode	Setting of PSC register	$\overline{\text{RESET}}$, NMI, maskable interrupt that is not masked and is from an operable internal peripheral I/O area, or external interrupt that is not masked.
Software STOP mode	Setting of PSC register	$\overline{\text{RESET}}$, NMI, maskable interrupt that is not masked and is from an operable internal peripheral I/O area, or external interrupt that is not masked
Subclock operation	Setting of PCC register	Setting of PCC register

1.12.2 System configuration example using power saving function

The following is a system configuration example in which watch functions are valid even during power saving mode. In the example shown in Figure 1-6, an NMI notifies the V850/SA1 when a power supply shutdown has been detected, and battery power V_{DD} is supplied while the main power supply is down. The V850/SA1's $\overline{\text{RESET}}$ pin is connected to a separate switch so that a power-on reset does not occur. Also, the time value is saved to internal RAM (for a circuit example, refer to **3.11 Shutdown Detection Circuit and Backup Power Supply**).

Figure 1-6. System Configuration Example Using Power Saving Function



1.13 Peripheral I/O Registers

(1/4)

Address	Function Register Name	Symbol	R/W	Operable Bits			After Reset
				1	8	16	
FFFFF000H	Port 0	P0	R/W	√	√		00H[Note]
FFFFF002H	Port 1	P1	R/W	√	√		00H[Note]
FFFFF004H	Port 2	P2	R/W	√	√		00H[Note]
FFFFF006H	Port 3	P3	R/W	√	√		00H[Note]
FFFFF008H	Port 4	P4	R/W	√	√		00H[Note]
FFFFF00AH	Port 5	P5	R/W	√	√		00H[Note]
FFFFF00CH	Port 6	P6	R/W	√	√		00H[Note]
FFFFF00EH	Port 7	P7	R	√	√		Undefined
FFFFF010H	Port 8	P8	R	√	√		Undefined
FFFFF012H	Port 9	P9	R/W	√	√		00H[Note]
FFFFF014H	Port 10	P10	R/W	√	√		00H[Note]
FFFFF016H	Port 11	P11	R/W	√	√		00H[Note]
FFFFF018H	Port 12	P12	R/W	√	√		00H[Note]
FFFFF020H	Port 0 mode register	PM0	R/W	√	√		FFH
FFFFF022H	Port 1 mode register	PM1	R/W	√	√		3FH
FFFFF024H	Port 2 mode register	PM2	R/W	√	√		FFH
FFFFF026H	Port 3 mode register	PM3	R/W	√	√		FFH
FFFFF028H	Port 4 mode register	PM4	R/W	√	√		FFH
FFFFF02AH	Port 5 mode register	PM5	R/W	√	√		FFH
FFFFF02CH	Port 6 mode register	PM6	R/W	√	√		3FH
FFFFF032H	Port 9 mode register	PM9	R/W	√	√		7FH
FFFFF034H	Port 10 mode register	PM10	R/W	√	√		FFH
FFFFF036H	Port 11 mode register	PM11	R/W	√	√		1FH
FFFFF038H	Port 12 mode register	PM12	R/W	√	√		01H
FFFFF04CH	Memory expansion mode register	MM	R/W	√	√		00H
FFFFF058H	Port 12 mode control register	PMC12	R/W	√	√		00H
FFFFF060H	Data wait control register	DWC	R/W			√	FFFFH
FFFFF062H	Bus cycle control register	BCC	R/W			√	AAAAH
FFFFF064H	System control register	SYC	R/W	√	√		00H
FFFFF068H	Memory address output mode register	MAM	W		√		00H
FFFFF070H	Power save control register	PSC	R/W	√	√		C0H
FFFFF074H	Processor clock control register	PCC	R/W	√	√		03H
FFFFF078H	System status register	SYS	R/W	√	√		00H
FFFFF080H	Pull-up resistor option register 0	PU0	R/W	√	√		00H
FFFFF082H	Pull-up resistor option register 1	PU1	R/W	√	√		00H
FFFFF084H	Pull-up resistor option register 2	PU2	R/W	√	√		00H
FFFFF086H	Pull-up resistor option register 3	PU3	R/W	√	√		00H
FFFFF094H	Pull-up resistor option register 10	PU10	R/W	√	√		00H
FFFFF096H	Pull-up resistor option register 11	PU11	R/W	√	√		00H
FFFFF0A2H	Port 1 function register	PF1	R/W	√	√		00H

Note A reset initializes the system to input mode, so the pin level is read during a read operation. Output latches are initialized to 00H.

(2/4)

Address	Function Register Name	Symbol	R/W	Operable Bits			After Reset
				1	8	16	
FFFFF0A4H	Port 2 function register	PF2	R/W	√	√		00H
FFFFF0B4H	Port 10 function register	PF10	R/W	√	√		00H
FFFFF0C0H	Rising edge enable register 0	EGP0	R/W	√	√		00H
FFFFF0C2H	Falling edge enable register 0	EGN0	R/W	√	√		00H
FFFFF100H	Interrupt control register	WDTIC	R/W	√	√		47H
FFFFF102H	Interrupt control register	PIC0	R/W	√	√		47H
FFFFF104H	Interrupt control register	PIC1	R/W	√	√		47H
FFFFF106H	Interrupt control register	PIC2	R/W	√	√		47H
FFFFF108H	Interrupt control register	PIC3	R/W	√	√		47H
FFFFF10AH	Interrupt control register	PIC4	R/W	√	√		47H
FFFFF10CH	Interrupt control register	PIC5	R/W	√	√		47H
FFFFF10EH	Interrupt control register	PIC6	R/W	√	√		47H
FFFFF110H	Interrupt control register	WTIIC	R/W	√	√		47H
FFFFF112H	Interrupt control register	TMIC00	R/W	√	√		47H
FFFFF114H	Interrupt control register	TMIC01	R/W	√	√		47H
FFFFF116H	Interrupt control register	TMIC10	R/W	√	√		47H
FFFFF118H	Interrupt control register	TMIC11	R/W	√	√		47H
FFFFF11AH	Interrupt control register	TMIC2	R/W	√	√		47H
FFFFF11CH	Interrupt control register	TMIC3	R/W	√	√		47H
FFFFF11EH	Interrupt control register	TMIC4	R/W	√	√		47H
FFFFF120H	Interrupt control register	TMIC5	R/W	√	√		47H
FFFFF122H	Interrupt control register	CSIC0	R/W	√	√		47H
FFFFF124H	Interrupt control register	SERIC0	R/W	√	√		47H
FFFFF126H	Interrupt control register	CSIC1	R/W	√	√		47H
FFFFF128H	Interrupt control register	STIC0	R/W	√	√		47H
FFFFF12AH	Interrupt control register	CSIC2	R/W	√	√		47H
FFFFF12CH	Interrupt control register	SERIC1	R/W	√	√		47H
FFFFF12EH	Interrupt control register	SRIC1	R/W	√	√		47H
FFFFF130H	Interrupt control register	STIC1	R/W	√	√		47H
FFFFF132H	Interrupt control register	ADIC	R/W	√	√		47H
FFFFF134H	Interrupt control register	DMAIC0	R/W	√	√		47H
FFFFF136H	Interrupt control register	DMAIC1	R/W	√	√		47H
FFFFF138H	Interrupt control register	DMAIC2	R/W	√	√		47H
FFFFF13AH	Interrupt control register	WTIC	R/W	√	√		47H
FFFFF166H	In-service priority register	ISPR	R	√	√		00H
FFFFF170H	Command register	PRCMD	W		√		Undefined
FFFFF180H	DMA peripheral I/O address register 0	DIOA0	R/W			√	Undefined
FFFFF182H	DMA internal RAM address register 0	DRA0	R/W			√	Undefined
FFFFF184H	DMA byte count register 0	DBC0	R/W		√		Undefined
FFFFF186H	DMA channel control register 0	DCHC0	R/W	√	√		00H
FFFFF190H	DMA peripheral I/O address register 1	DIOA1	R/W			√	Undefined
FFFFF192H	DMA internal RAM address register 1	DRA1	R/W			√	Undefined
FFFFF194H	DMA byte count register 1	DBC1	R/W		√		Undefined
FFFFF196H	DMA channel control register 1	DCHC1	R/W	√	√		00H
FFFFF1A0H	DMA peripheral I/O address register 2	DIOA2	R/W			√	Undefined
FFFFF1A2H	DMA internal RAM address register 2	DRA2	R/W			√	Undefined

(3/4)

Address	Function Register Name	Symbol	R/W	Operable Bits			After Reset
				1	8	16	
FFFFF1A4H	DMA byte count register 2	DBC2	R/W		√		Undefined
FFFFF1A6H	DMA channel control register 2	DCHC2	R/W	√	√		00H
★ FFFFF1F4H	Flash memory programming mode control register	FLPMC	R/W	√	√		**Note 1**
FFFFF200H	16-bit timer register 0	TM0	R			√	0000H
FFFFF202H	16-bit capture/compare register 00	CR00	**Note 2**			√	0000H
FFFFF204H	16-bit capture/compare register 01	CR01	**Note 2**			√	0000H
FFFFF206H	Prescaler mode register 0	PRM0	R/W		√		00H
FFFFF208H	16-bit timer mode control register 0	TMC0	R/W	√	√		00H
FFFFF20AH	Capture/compare control register 0	CRC0	R/W	√	√		00H
FFFFF20CH	Timer output control register 0	TOC0	R/W	√	√		00H
FFFFF20EH	Prescaler mode register 01	PRM01	R/W		√		00H
FFFFF210H	16-bit timer register 1	TM1	R			√	0000H
FFFFF212H	16-bit capture/compare register 10	CR10	**Note 2**			√	0000H
FFFFF214H	16-bit capture/compare register 11	CR11	**Note 2**			√	0000H
FFFFF216H	Prescaler mode register 1	PRM1	R/W		√		00H
FFFFF218H	16-bit timer mode control register 1	TMC1	R/W	√	√		00H
FFFFF21AH	Capture/compare control register 1	CRC1	R/W	√	√		00H
FFFFF21CH	Timer output control register 1	TOC1	R/W	√	√		00H
FFFFF21EH	Prescaler mode register 11	PRM11	R/W		√		00H
FFFFF240H	8-bit counter 2	TM2	R		√		00H
FFFFF242H	8-bit compare register 2	CR20	R/W		√		00H
FFFFF244H	Timer clock select register 2	TCL2	R/W		√		00H
★ FFFFF246H	8-bit timer mode control register 2	TMC2	R/W	√	√		04H Note 3
FFFFF24AH	16-bit counter 23	TM23	R			√	0000H
FFFFF24CH	16-bit compare register 23	CR23	R/W			√	0000H
FFFFF24EH	Timer clock select register 21	TCL21	R/W		√		00H
FFFFF250H	8-bit counter 3	TM3	R		√		00H
FFFFF252H	8-bit compare register 3	CR30	R/W		√		00H
FFFFF254H	Timer clock select register 3	TCL3	R/W		√		00H
★ FFFFF256H	8-bit timer mode control register 3	TMC3	R/W	√	√		04H Note 3
FFFFF25EH	Timer clock select register 31	TCL31	R/W		√		00H
FFFFF260H	8-bit counter 4	TM4	R		√		00H
FFFFF262H	8-bit compare register 4	CR40	R/W		√		00H
FFFFF264H	Timer clock select register 4	TCL4	R/W		√		00H
★ FFFFF266H	8-bit timer mode control register 4	TMC4	R/W	√	√		04H Note 3
FFFFF26AH	16-bit counter 45	TM45	R			√	0000H
FFFFF26CH	16-bit compare register 45	CR45	R/W			√	0000H
FFFFF26EH	Timer clock select register 41	TCL41	R/W		√		00H
FFFFF270H	8-bit counter 5	TM5	R		√		00H
FFFFF272H	8-bit compare register 5	CR50	R/W		√		00H

Notes 1. In single chip mode: 18H or 38H
In flash memory programming mode: 1CH or 3CH
2. In compare mode: R/W
In capture mode: R
3. The hardware status is initialized to 04H, however, 00H is read during a read operation.

(4/4)

Address	Function Register Name	Symbol	R/W	Operable Bits			After Reset
				1	8	16	
FFFFF274H	Timer clock select register 5	TCL5	R/W		√		00H
★ FFFFF276H	8-bit timer mode control register 5	TMC5	R/W	√	√		04H[Note 1]
FFFFF27EH	Timer clock select register 51	TCL51	R/W		√		00H
FFFFF2A0H	Serial I/O shift register 0	SIO0	R/W		√		00H
FFFFF2A2H	Serial operation mode register 0	CSIM0	R/W	√	√		00H
FFFFF2A4H	Serial clock select register 0	CSIS0	R/W	√	√		00H
FFFFF2B0H	Serial I/O shift register 1	SIO1	R/W		√		00H
FFFFF2B2H	Serial operation mode register 1	CSIM1	R/W	√	√		00H
FFFFF2B4H	Serial clock select register 1	CSIS1	R/W	√	√		00H
FFFFF2C0H	Serial I/O shift register 2	SIO2	R/W		√		00H
FFFFF2C2H	Serial operation mode register 2	CSIM2	R/W	√	√		00H
FFFFF2C4H	Serial clock select register 2	CSIS2	R/W	√	√		00H
FFFFF300H	Asynchronous serial interface mode register 0	ASIM0	R/W	√	√		00H
FFFFF302H	Asynchronous serial interface status register 0	ASIS0	R	√	√		00H
FFFFF304H	Baud rate generator control register 0	BRGC0	R/W		√		00H
FFFFF306H	Transmission shift register 0	TXS0	W		√		FFH
FFFFF308H	Reception buffer register 0	RXB0	R		√		FFH
FFFFF30EH	Baud rate generator mode control register 0	BRGMC0	R/W		√		00H
FFFFF310H	Asynchronous serial interface mode register 1	ASIM1	R/W	√	√		00H
FFFFF312H	Asynchronous serial interface status register 1	ASIS1	R	√	√		00H
FFFFF314H	Baud rate generator control register 1	BRGC1	R/W		√		00H
FFFFF316H	Transmission shift register 1	TXS1	W		√		FFH
FFFFF318H	Reception buffer register 1	RXB1	R		√		FFH
FFFFF31EH	Baud rate generator mode control register 1	BRGMC1	R/W		√		00H
FFFFF320H	Baud rate generator mode control register 01	BRGMC01	R/W		√		00H
FFFFF340H	IIC control register[Note 2]	IICC0	R/W	√	√		00H
FFFFF342H	IIC status register[Note 2]	IICS0	R	√	√		00H
FFFFF344H	IIC clock select register[Note 2]	IICCL0	R/W	√	√		00H
FFFFF346H	Slave address register[Note 2]	SVA0	R/W		√		00H
FFFFF348H	IIC shift register[Note 2]	IIC0	R/W		√		00H
FFFFF34AH	IIC function expansion register 0[Note 2]	IICX0	R/W	√	√		00H
FFFFF360H	Watch timer mode register	WTM	R/W	√	√		00H
FFFFF380H	Oscillation stabilization time select register	OSTS	R/W		√		04H
FFFFF382H	Watchdog timer clock select register	WDCS	R/W		√		00H
FFFFF384H	Watchdog timer mode register	WDTM	R/W	√	√		00H
FFFFF3A0H	Real-time output buffer register L	RTBL	R/W		√		00H
FFFFF3A2H	Real-time output buffer register H	RTBH	R/W		√		00H
FFFFF3A4H	Real-time output port mode register	RTPM	R/W	√	√		00H
FFFFF3A6H	Real-time output port control register	RTPC	R/W	√	√		00H
FFFFF3C0H	A/D converter mode register	ADM	R/W	√	√		00H
FFFFF3C2H	Analog input channel specification register	ADS	R/W	√	√		00H
FFFFF3C4H	A/D conversion result register	ADCR	R			√	0000H
FFFFF3C6H	A/D conversion result register (higher 8 bits)	ADCRH	R		√		00H

Notes 1. The hardware status is initialized to 04H, however, 00H is read during a read operation.

★ **2.** μPD703014AY, 703014BY, 703015AY, 703015BY, 703017AY, 70F3015BY, and 70F3017AY only.

CHAPTER 2 BUS INTERFACE CONNECTION CIRCUIT EXAMPLES

This chapter provides circuit examples that use pins related to the V850/SA1 bus interface to connect to external devices. Connection-related cautions points and pin functions are described below in **2.1 Bus Interface Connection Circuit Examples**. Actual circuit examples are shown in **2.2 PROM Connection Circuits**.

O The configuration diagrams that include timing generators and decoders emphasize theoretical descriptions. Generally, even blocks that use gate arrays or PLDs may use only components such as AND/OR/flip-flops. Accordingly, no model names are given for the timing generators and decoders. Be sure to consider delay time and DC characteristics when selecting components for the actual circuit design.

O In this chapter, it is assumed either that address signals separated from AD0 to AD15 on the V850/SA1 are used or that A1 to A15 on the V850/SA1 are used (refer to **2.1.1 Address bus** below).

O When the circuit example includes a connection to a 5 V device, a level converting device is used (except for gates) even when DC characteristics can be met. In an actual circuit, the level converting device can be omitted if DC characteristics are met by the V850/SA1 and the connected device.

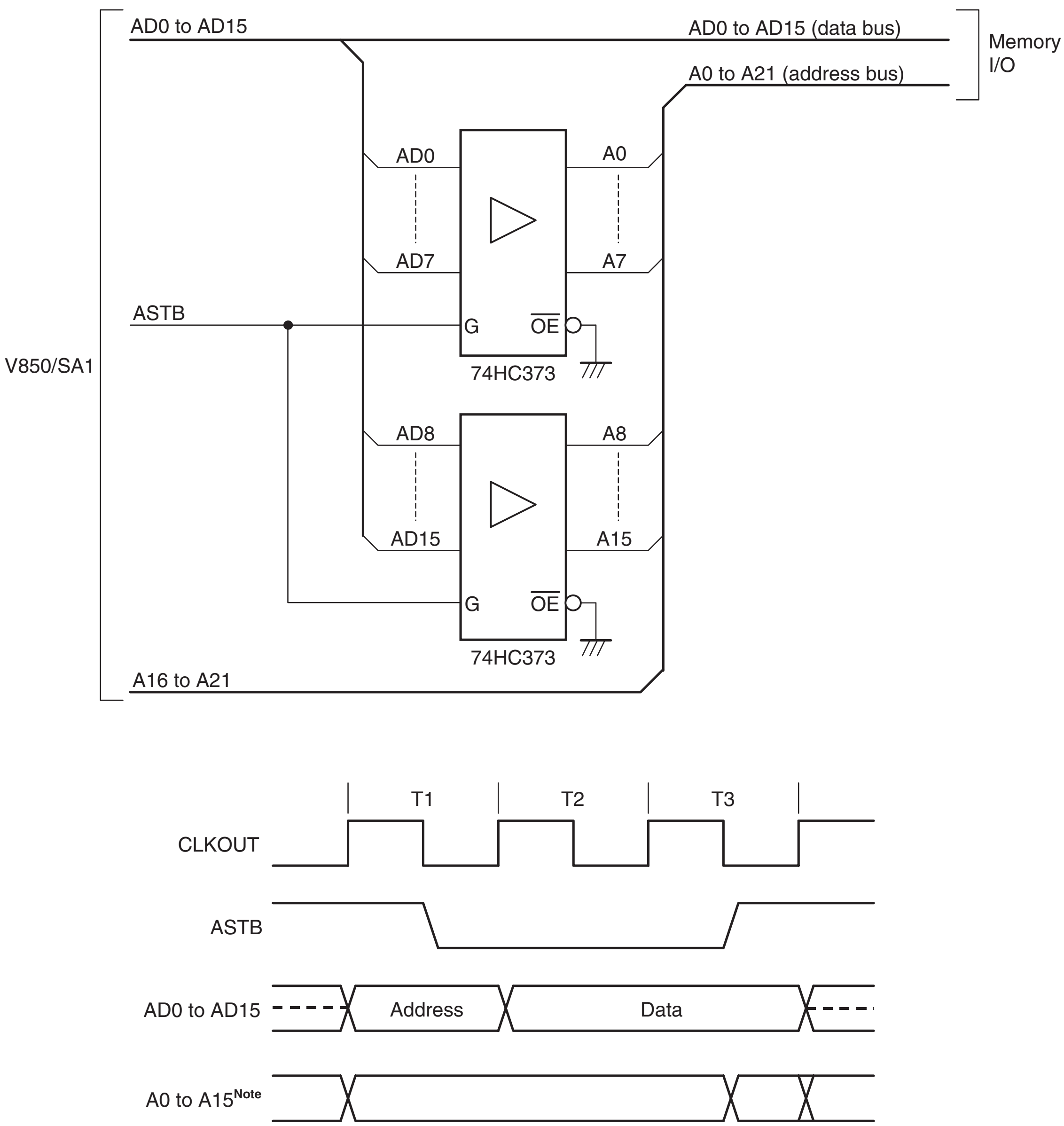
★ 2.1 Bus Interface Connection Circuit Examples

2.1.1 Address bus

The V850/SA1 is able to use address signals up to 22 bits in width. The lower 16 bits can use AD0 to AD15, which are multiplexed as the data bus, or they can use the separated A1 to A15 (the separated A0 signal cannot be used).

If using AD0 to AD15, address/data separation is required. The typical method for addresses is to use transparent latch devices and to generate address signals using an ASTB signal. The data bus uses AD0 to AD15 as they are.

Figure 2-1. Circuit Example with Separate Address Bus and Data Bus



Note A0 to A15 are signals separated by ASTB.

Remarks **1.** The timing chart assumes a zero-wait state.
2. Broken lines indicate high impedance.

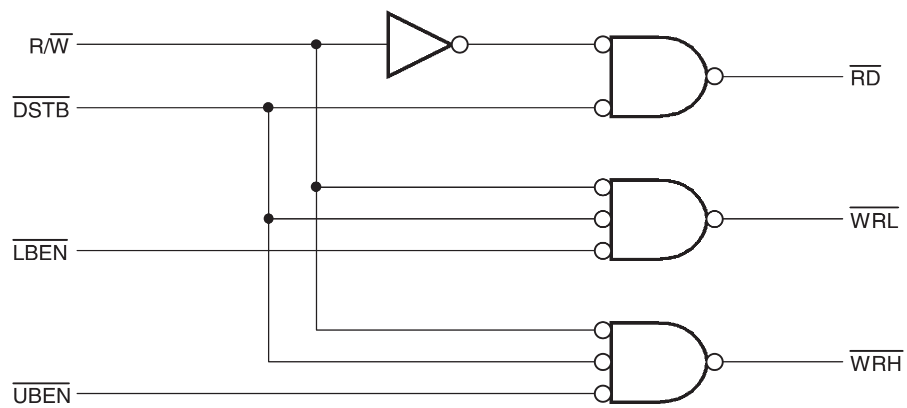
2.1.2 Operation modes of read/write control pins

The following two modes can be set for the read/write control pins.

- $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$ signal output mode
- $\overline{\text{RD}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, $\overline{\text{UBEN}}$ signal output mode

These modes are set via the system control register (SYC). The pin logic is shown below.

Figure 2-2. Logic of Read/Write Control Pins

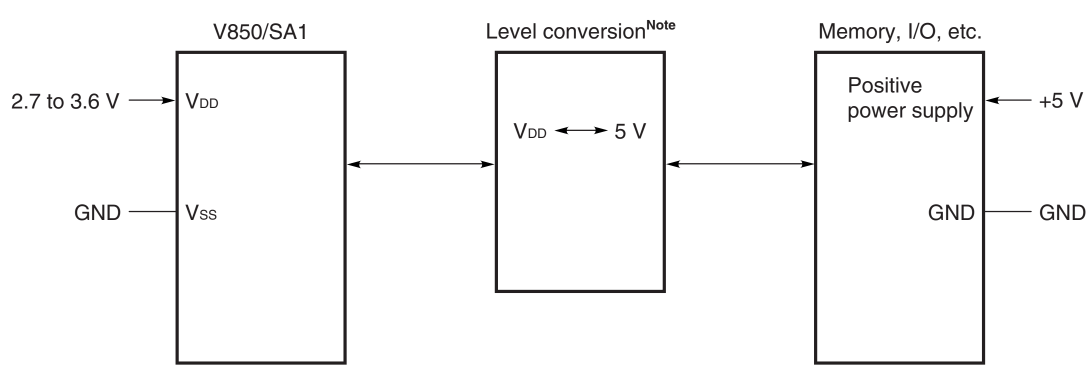


2.1.3 5 V device connection

When connecting the V850/SA1 to a 5 V system (device), a level converting device is used to maintain the V850/SA1's DC characteristics.

If the connected device already satisfies the V850/SA1's DC characteristics, there is no need to use a level converting device.

Figure 2-3. 5 V Device Connection



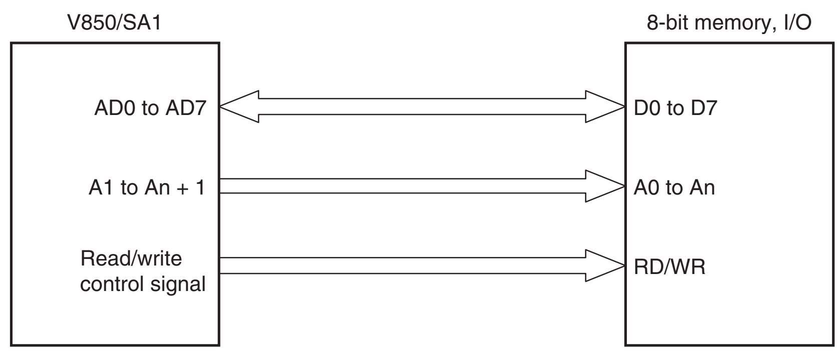
Note Various level converting devices are available, including the LCX Series, LVX Series, and FCT Series. These devices are sold by IC manufacturers.

2.1.4 Connection of 8-bit bus width unit and 16-bit bus width unit

Although the V850/SA1 data bus has a fixed width of 16 bits, its data space (i.e., the space accessed by LD, ST, and bit manipulation instructions) is byte access-enabled, so both 8-bit bus width and 16-bit bus width configurations are possible. The program space is entirely 16-bit accessible, so a 16-bit bus width configuration is required. In any case, the address bus is arranged starting from the A1 signal.

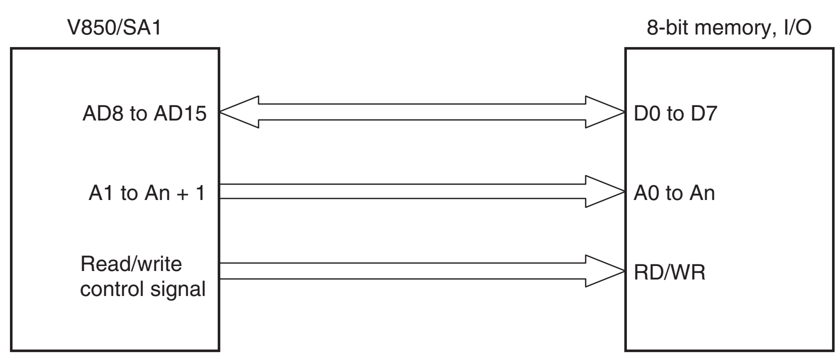
8-bit bus width unit and 16-bit bus width unit connections are shown below.

Figure 2-4. 8-Bit Bus Width Unit Arranged in Data Space (Connection to AD0 to AD7)



Caution This is valid only for byte access to even addresses.

Figure 2-5. 8-Bit Bus Width Unit Arranged in Data Space (Connection to AD8 to AD15)



Caution This is valid only for byte access to odd addresses.

Figure 2-6. Connection to 16-Bit Bus Width Unit

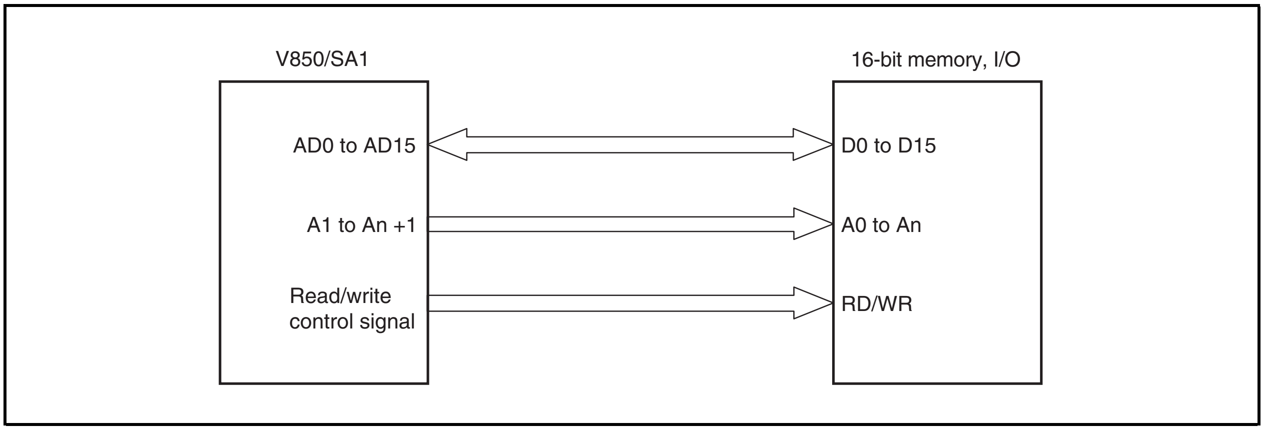
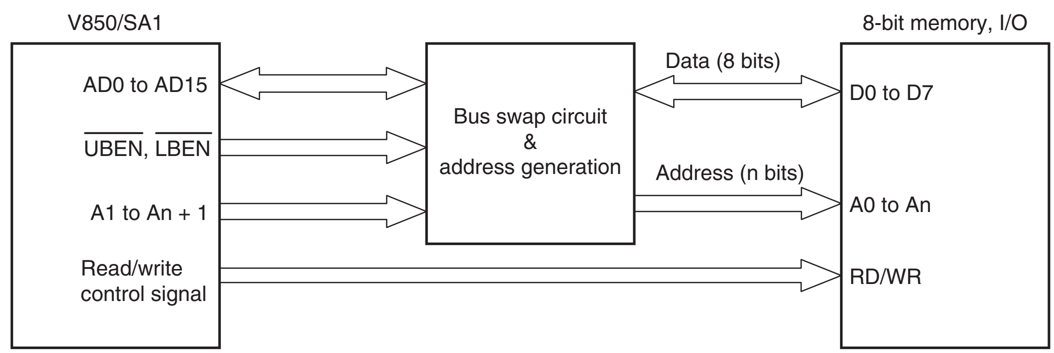


Figure 2-7. Connection to 8-Bit Bus Width Unit Arranged in Space Required for Program Space/16-Bit Access



Remark For a circuit example, refer to **2.5 Connection of 8-Bit PROM Using Bus Sizing Circuit**.

2.2 PROM Connection Circuits

The following circuit example uses one 27C1024 (PROM, 64 K × 16 bits). The V850/SA1 is connected to a 128 KB external ROM space.

[Circuit configuration]

- Main system clock: 16 MHz
- Connected devices: 27C1024 × 1
- Memory usage: Address range 00000H to 1FFFFH in external memory space
  (128 KB space starting at address 0)

[DWC and BCC settings]

- Wait setting: 0 waits
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$ signal output

[Circuit type]

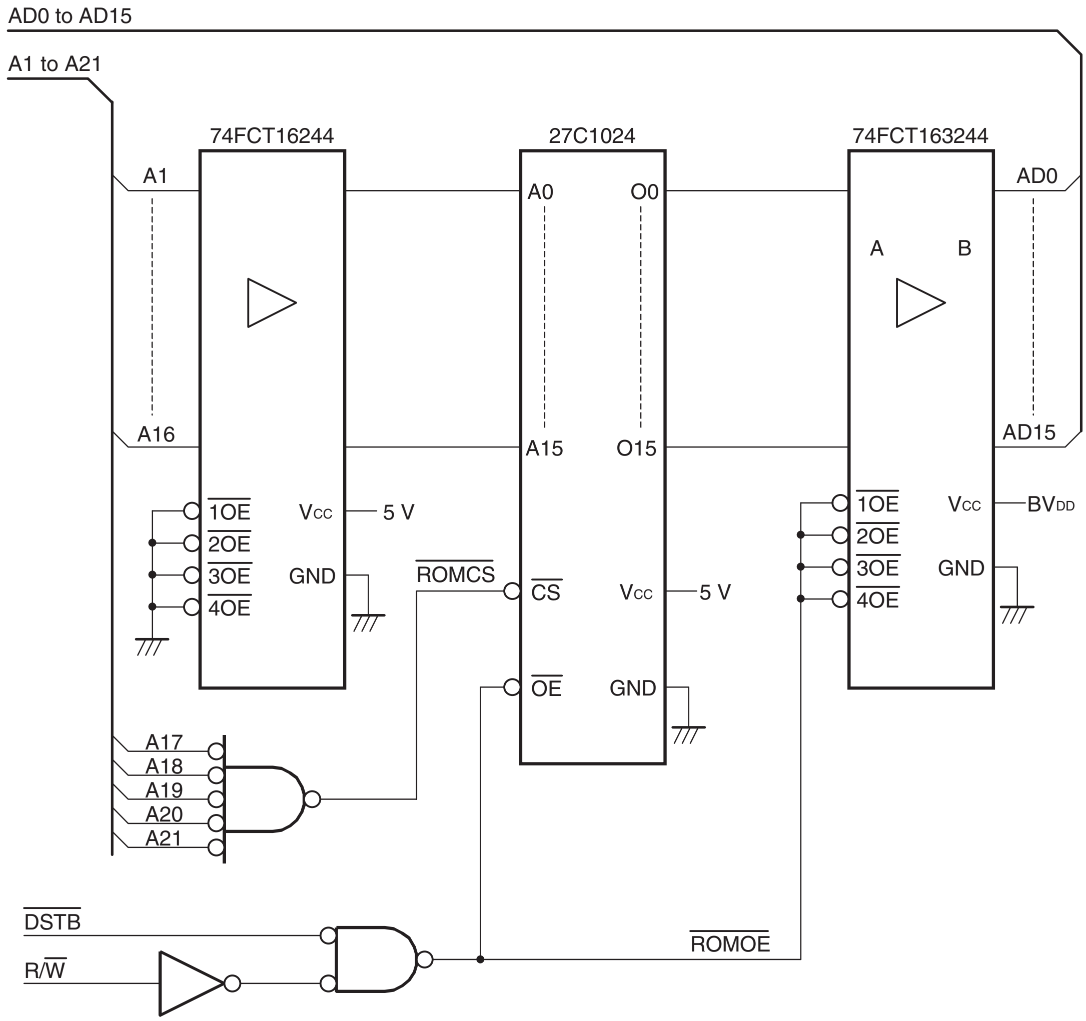
- Since the 27C1024 is a 5 V product, a level converting device is used to connect to the address bus and data bus.

  Address bus: 74FCT16244
  Data bus: 74FCT163244

  Caution For $\overline{\text{OE}}$ control of the decoder and data buffer used to generate $\overline{\text{CS}}$, use a product that maintains the DC characteristics of the V850/SA1 and the 27C1024.

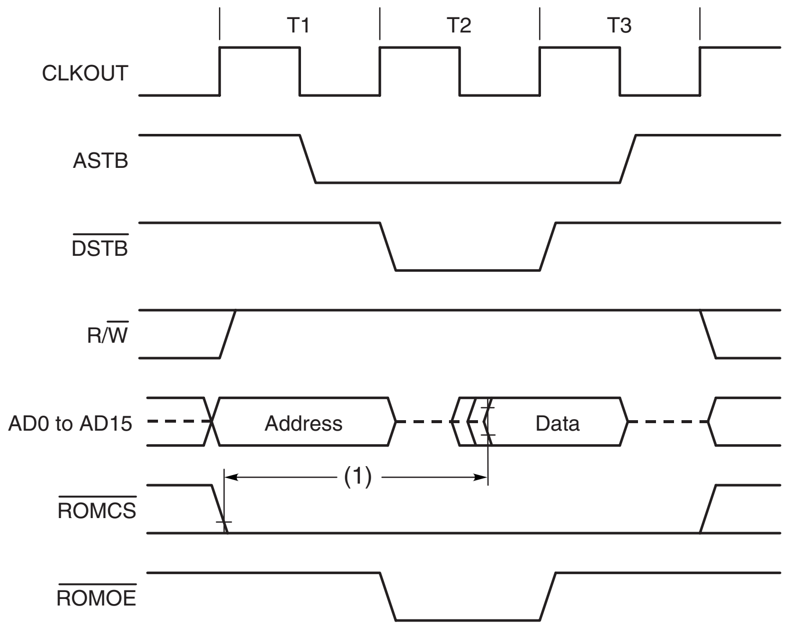
- ROM's $\overline{\text{CS}}$ signal ($\overline{\text{ROMCS}}$) is generated by decoding the higher 5 bits of the address.
- The 74FCT163244's $\overline{\text{OE}}$ signal ($\overline{\text{ROMOE}}$) is generated from the R/$\overline{\text{W}}$ and $\overline{\text{DSTB}}$ signals (the $\overline{\text{UBEN}}$ and $\overline{\text{LBEN}}$ signals are not used).
- The $\overline{\text{WAIT}}$ pin is not controlled (it is fixed to high level).

Figure 2-8. PROM Connection Circuit Example



Remark 27C1024: 64 K address space × 16 bits (1 Mb)

Figure 2-9. PROM Read Operation



(1) Data output delay time following active state transition of 27C1024's $\overline{\text{CS}}$ signal: 85 ns (Max.)

Remarks
1. $\overline{\text{ROMCS}}$ and $\overline{\text{ROMOE}}$ are signals generated in this circuit.
2. Broken lines indicate high impedance.

2.3 SRAM Connection Circuits

The following are examples of connections between the V850/SA1 and SRAM devices.

In **2.3.1 SRAM connection circuit example 1 (connection of 8-bit SRAMs)**, two 8-bit SRAM devices are connected to configure a 16-bit bus space.

In **2.3.2 SRAM connection circuit example 2 (connection of 16-bit SRAM)**, one 16-bit SRAM device is connected.

2.3.1 SRAM connection circuit example 1 (connection of 8-bit SRAMs)

In the following example, two μPD431000A (SRAM, 128 K $\times$ 8 bits) devices are used to configure a 256 KB external RAM space which is connected to the V850/SA1.

[Circuit configuration]

- V850/SA1's main system clock: 16 MHz
- Connected devices: μPD431000A $\times$ 2
- Memory usage: Address range 040000H to 07FFFFH in external memory space (256 KB space starting at address 40000H)

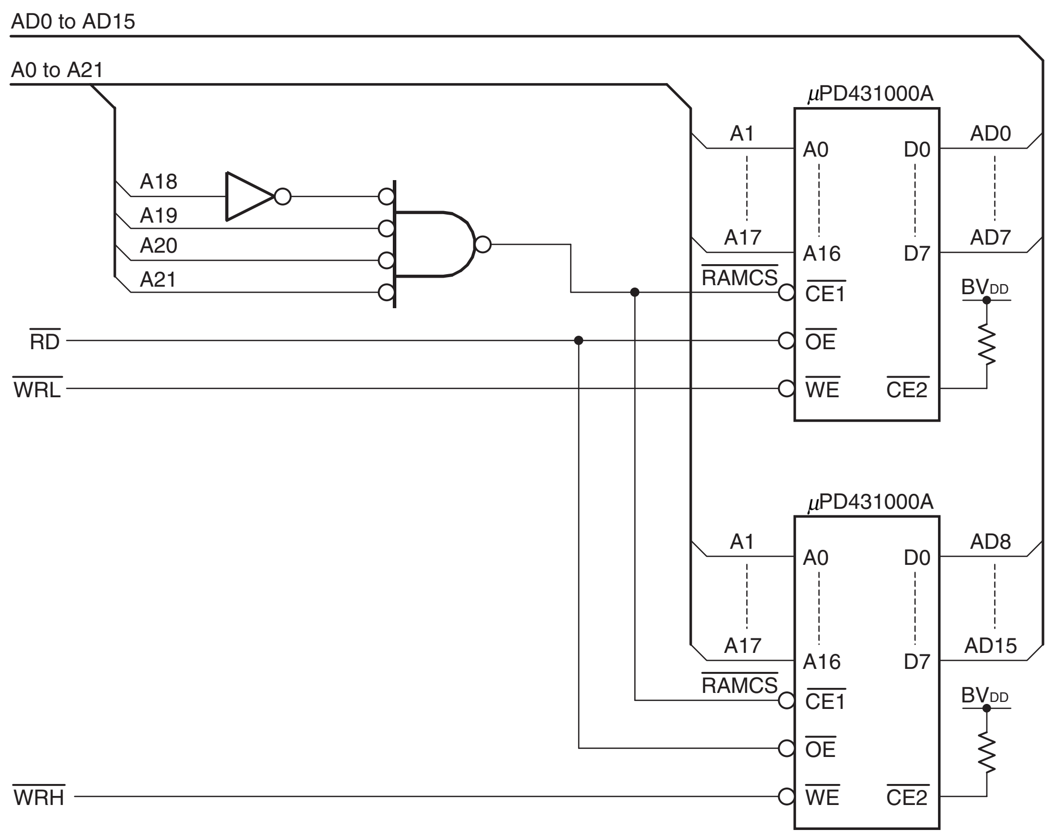
[DWC and BCC settings]

- Wait setting: 2 wait cycles
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{RD}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, $\overline{\text{UBEN}}$ signal output

[Circuit type]

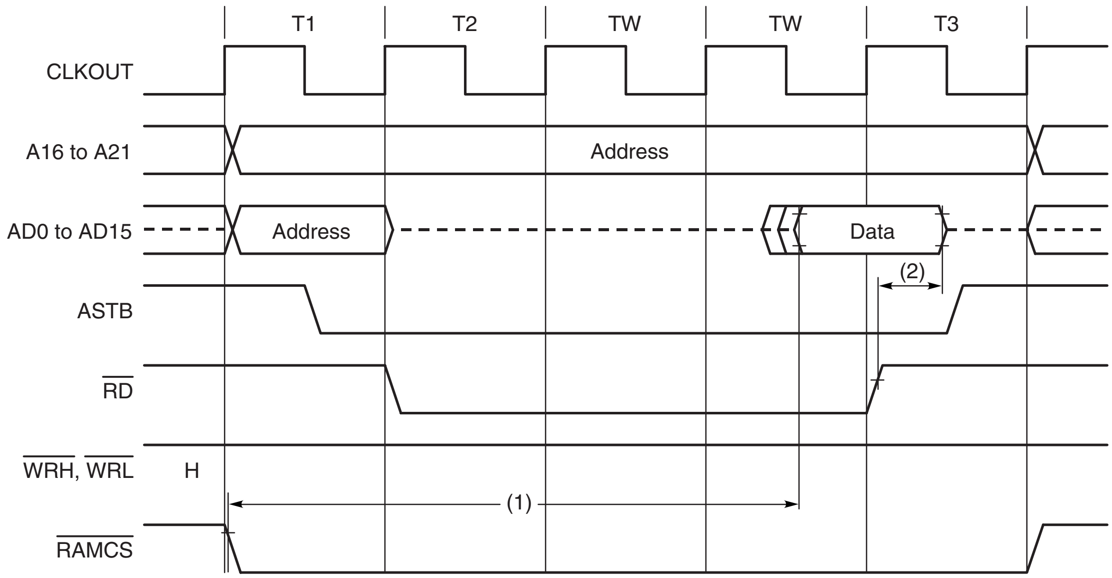
- SRAM's $\overline{\text{CS}}$ signal ($\overline{\text{RAMCS}}$) is generated by decoding the higher 4 bits of the address.
- SRAM's $\overline{\text{OE}}$ pin is connected to $\overline{\text{RD}}$.
- The $\overline{\text{WE}}$ pin of the SRAM connected to AD0 to AD7 is connected to $\overline{\text{WRL}}$.
  The $\overline{\text{WE}}$ pin of the SRAM connected to AD8 to AD15 is connected to $\overline{\text{WRH}}$.
- The $\overline{\text{WAIT}}$ pin is not controlled.

Figure 2-10. μPD431000A Connection Circuit Example



Remark μPD431000A: 128 K address space × 8 bits (1 Mb)

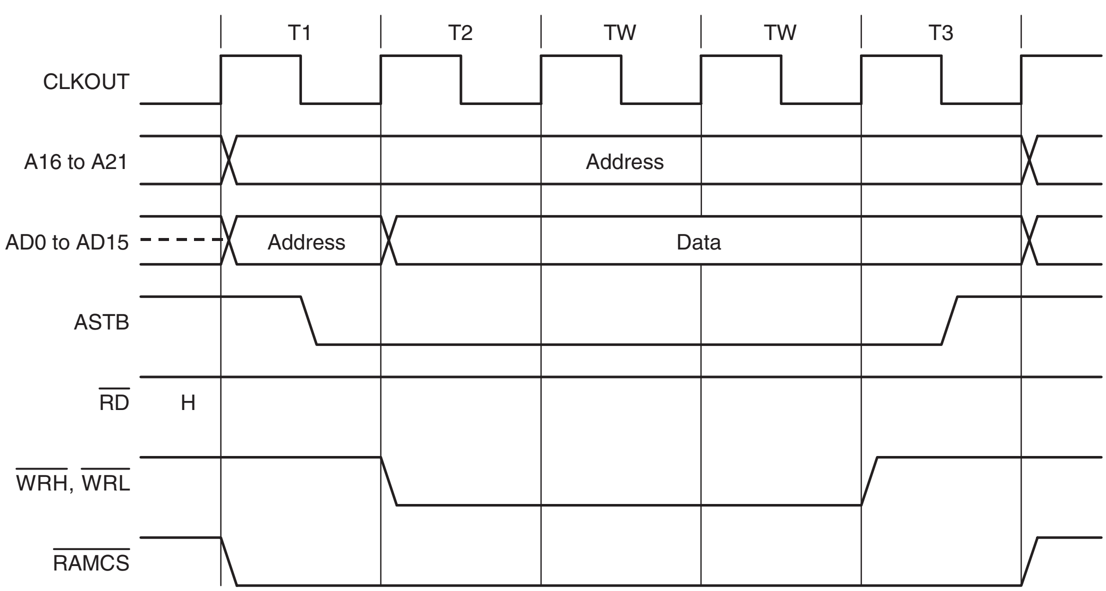
Figure 2-11. Read Operation (μPD431000A)



(1) Data output delay time following active state transition of $\overline{\text{CS}}$: 150 ns (Max.)
(2) Output float delay time following inactive state transition of $\overline{\text{OE}}$: 50 ns (Max.)

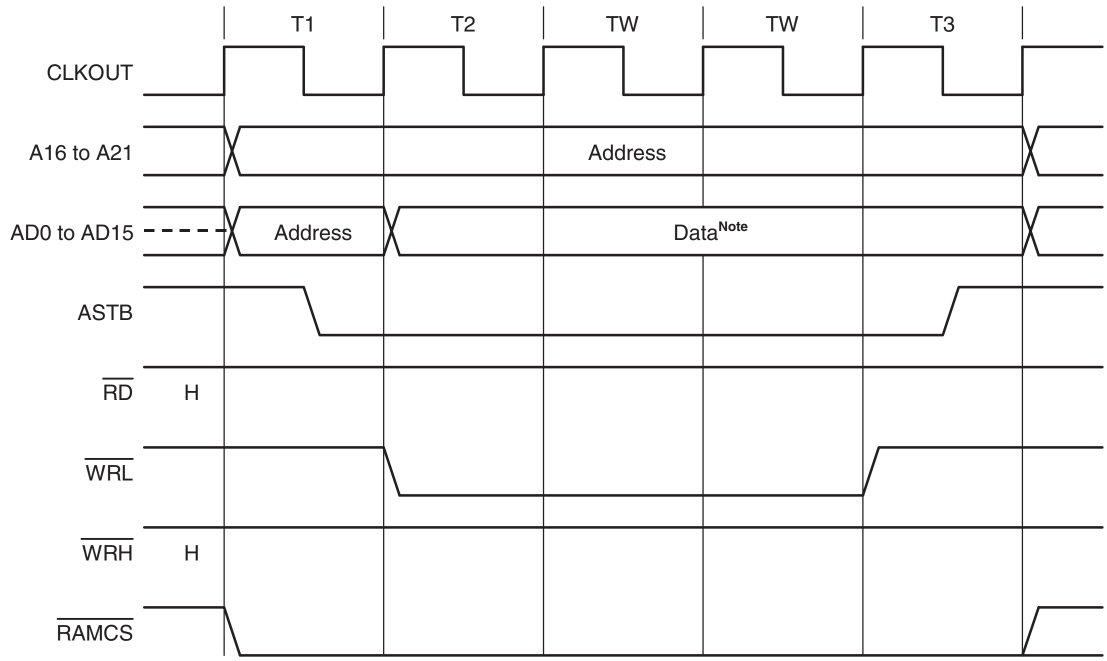
Remarks **1.** $\overline{\text{RAMCS}}$ is a signal generated in this circuit.
2. Broken lines indicate high impedance.

Figure 2-12. Write Operation (μPD431000A)



Remarks **1.** $\overline{\text{RAMCS}}$ is a signal generated in this circuit.
2. Broken lines indicate high impedance.

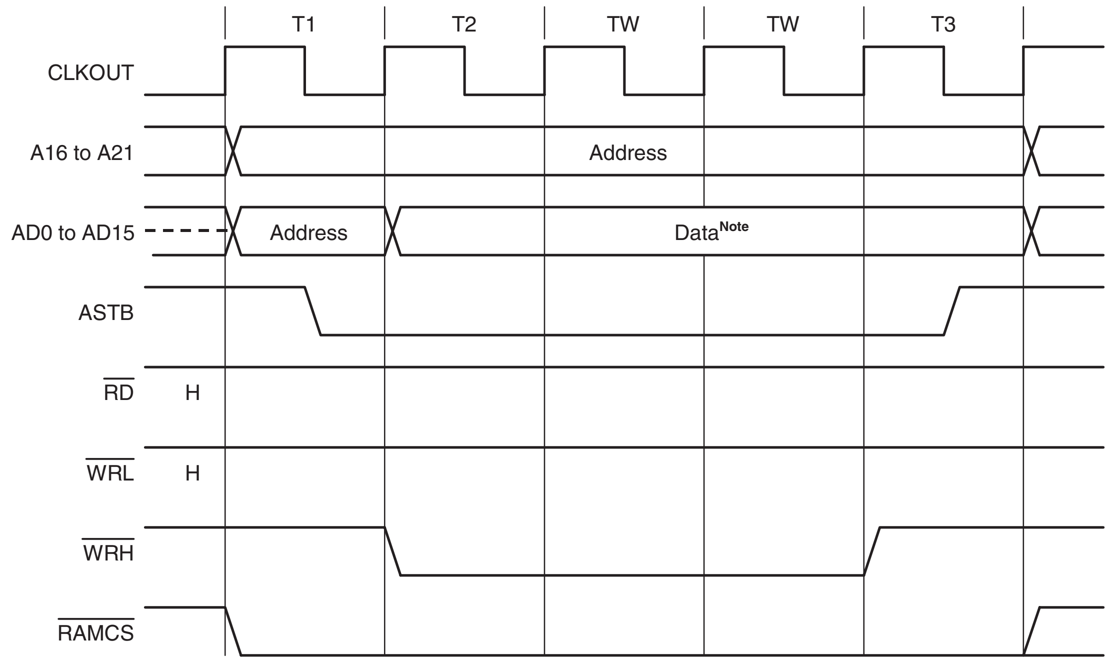
Figure 2-13. Even Address Byte Write Operation (μPD431000A)



Note AD8 to AD15 output invalid data.

Remarks **1.** RAMCS is a signal generated in this circuit.
2. Broken lines indicate high impedance.

Figure 2-14. Odd Address Byte Write Operation (μPD431000A)



Note AD0 to AD7 output invalid data.

Remarks **1.** $\overline{\text{RAMCS}}$ is a signal generated in this circuit.
2. Broken lines indicate high impedance.

2.3.2 SRAM connection circuit example 2 (connection of 16-bit SRAM)

In the following example, one μPD431016LE (SRAM, 64 K $\times$ 16 bits) device is used to configure a 128 KB external RAM space which is connected to the V850/SA1.

[Circuit configuration]

- V850/SA1's main system clock: 16 MHz
- Connected devices: μPD431016LE $\times$ 1
- Memory usage: Address range 040000H to 07FFFFH in external memory space (256 KB space starting at address 40000H)

Caution The address range from 60000H to 7FFFFH is the image of address range 40000H to 5FFFFH.

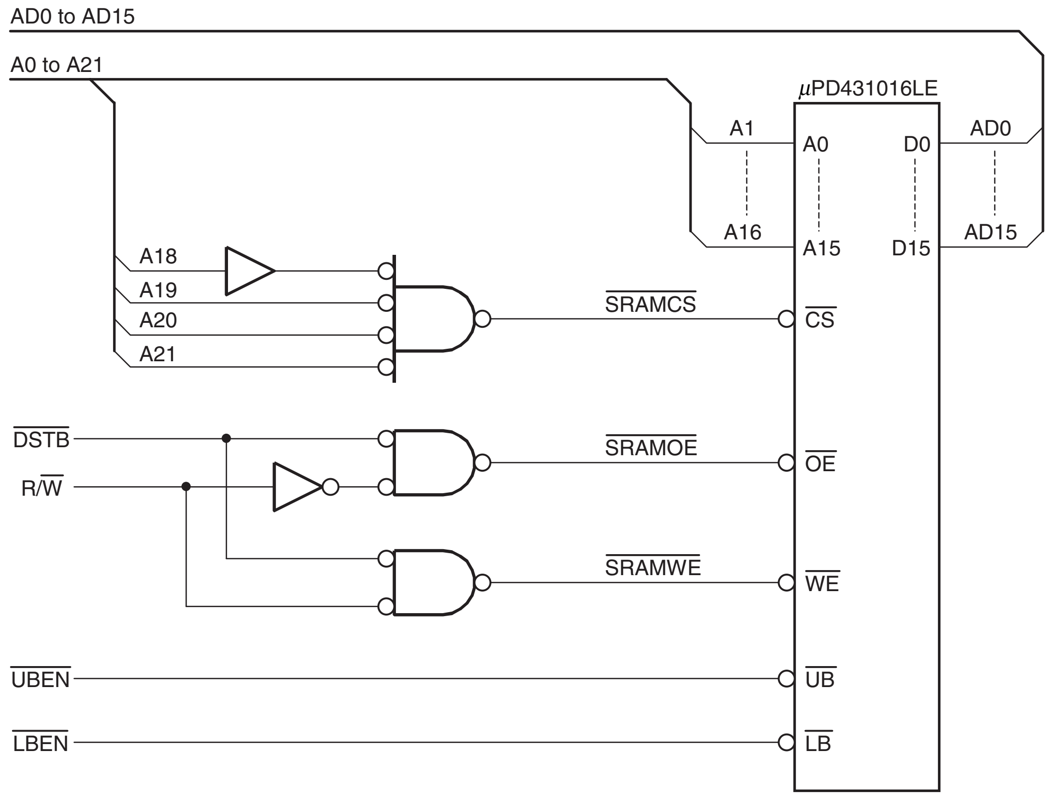
[DWC and BCC settings]

- Wait setting: 0 waits
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$ signal output

[Circuit type]

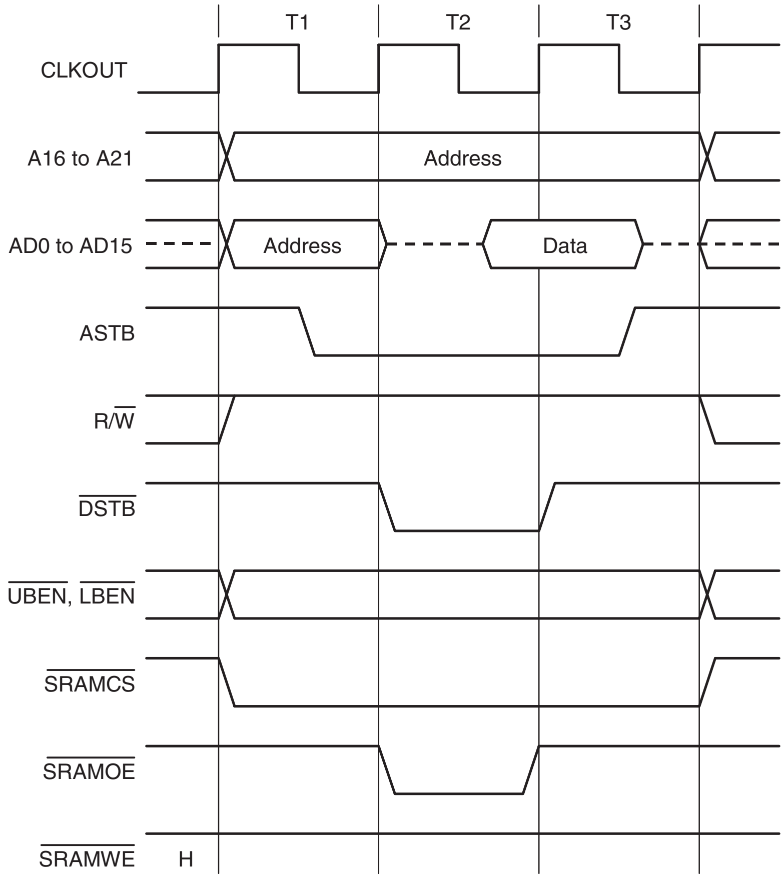
- SRAM's $\overline{\text{CS}}$ signal ($\overline{\text{SRAMCS}}$) is generated by decoding the higher 4 bits of the address.
- SRAM's $\overline{\text{OE}}$ and $\overline{\text{WE}}$ signals ($\overline{\text{SRAMOE}}$, $\overline{\text{SRAMWE}}$) are generated from R/$\overline{\text{W}}$ and $\overline{\text{DSTB}}$.
- SRAM's $\overline{\text{UB}}$ and $\overline{\text{LB}}$ pins are connected to $\overline{\text{UBEN}}$ and $\overline{\text{LBEN}}$ respectively.
- The $\overline{\text{WAIT}}$ pin is not controlled.

Figure 2-15. μPD431016LE Connection Circuit Example



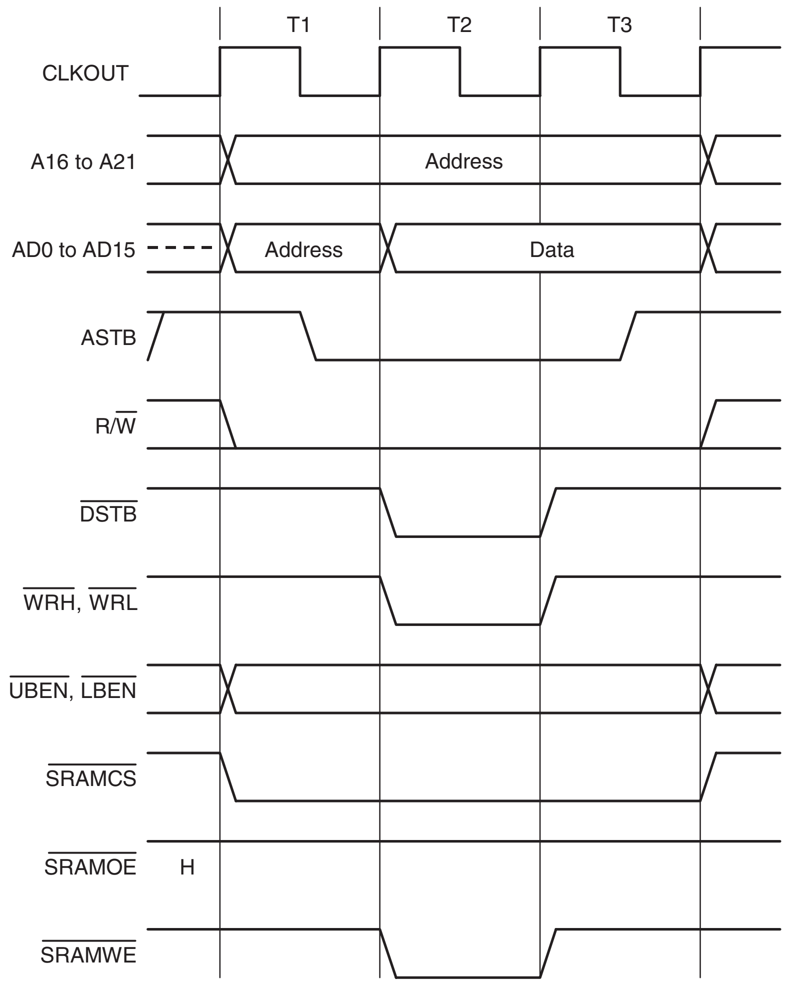
Remark μPD431016LE: 64 K address space × 16 bits (1 Mb)

Figure 2-16. Read Operation (μPD431016LE)



Remarks **1.** SRAMCS, SRAMOE, and SRAMWE are signals generated in this circuit.
2. Broken lines indicate high impedance.

Figure 2-17. Write Operation (μPD431016LE)



Remarks 1. $\overline{SRAMCS}$, $\overline{SRAMOE}$, and $\overline{SRAMWE}$ are signals generated in this circuit.

2. Broken lines indicate high impedance.

2.4 DRAM Connection Circuits

In the following circuit example, the V850/SA1 uses one μPD42S16160L (DRAM, 1 M × 16 bits) device and is connected to a 2 MB external RAM space.

Different refresh circuits are used in DRAM connection circuit example 1 (refer to 2.4.1 below) and DRAM connection circuit example 2 (refer to 2.4.2 below). Although the CBR method is used as the DRAM refresh method in both examples, in DRAM connection circuit example 1 the refresh period is the hold state period for the V850/SA1's bus. In DRAM connection circuit example 2, bus arbitration is performed by an external bus arbiter.

2.4.1 DRAM connection circuit 1 (refresh uses $\overline{\text{HLDRQ}}$ pin)

[Circuit configuration]

- V850/SA1's main system clock: 16 MHz
- Connected device: μPD42S16160L × 1
- Memory usage: Address range 200000H to 3FFFFFH in external memory space
- Supported cycles: Read cycle, early write cycle, and CBR refresh cycle

[DWC and BCC settings]

- Wait setting: 1 wait cycle
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$ signal output

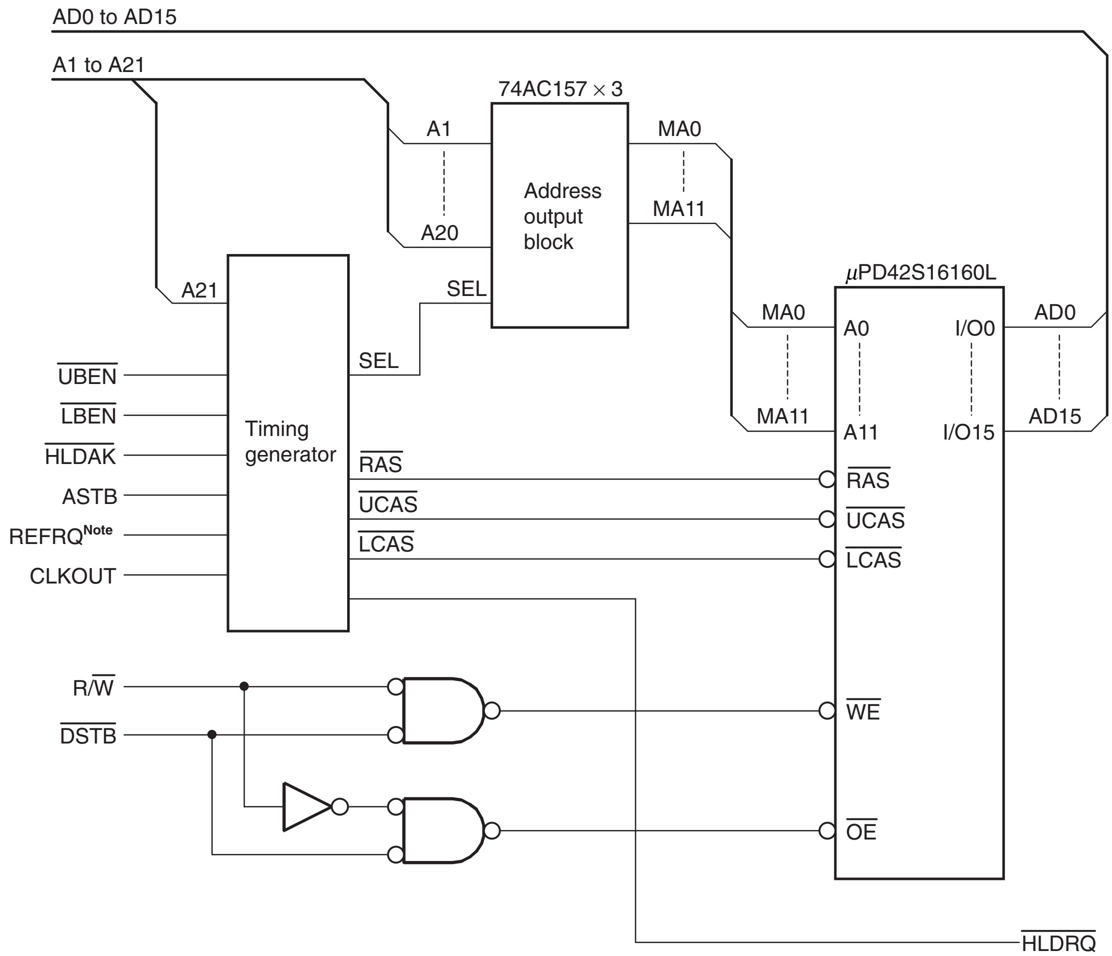
[Circuit type]

- The V850/SA1's address A21 is monitored at the rising edge of the T2 clock. If the address matches, a CLKOUT signal is generated via read cycle/write cycle timing.
- The $\overline{\text{WAIT}}$ pin is not controlled (it is fixed at high level).
- A refresh request is generated in 31.25 μs[Note] by the external counter and the refresh is executed via the following sequence.

Note Since the μPD42S16160L's refresh cycle timing is 4096 cycles per 128 μs, the refresh interval is (128000/4096 =) 31.25 (μs). Consequently, a refresh interval of 31.25 μs or less is required.
Since one cycle is 62.5 ns at 16 MHz, the refresh interval is (31.25 μs/62.5 ns) = 500. Therefore, one refresh can be performed every 500 CLKOUT cycles.

(1) A refresh request is generated by the external counter.
(2) The V850/SA1's $\overline{\text{HLDRQ}}$ pin becomes active.
(3) The V850/SA1's $\overline{\text{HLDAK}}$ pin becomes active and enters the hold state.
(4) One CBR refresh cycle is executed (timing is generated using CLKOUT signal).
(5) The V850/SA1's $\overline{\text{HLDRQ}}$ pin becomes inactive.
(6) The V850/SA1's $\overline{\text{HLDAK}}$ pin becomes inactive and its hold state is canceled.

Figure 2-18. μPD42S16160L Connection Circuit Example (Refresh Uses $\overline{\text{HLDRQ}}$ Pin)



Note The REFRQ signal occurs once using the high level of the clock synchronized with the CLKOUT signal as the refresh timing.

Remark μPD42S16160L: 1 M × 16 bits (16 Mb)

Figure 2-19. Timing Generator (DRAM Connection Circuit Example 1)

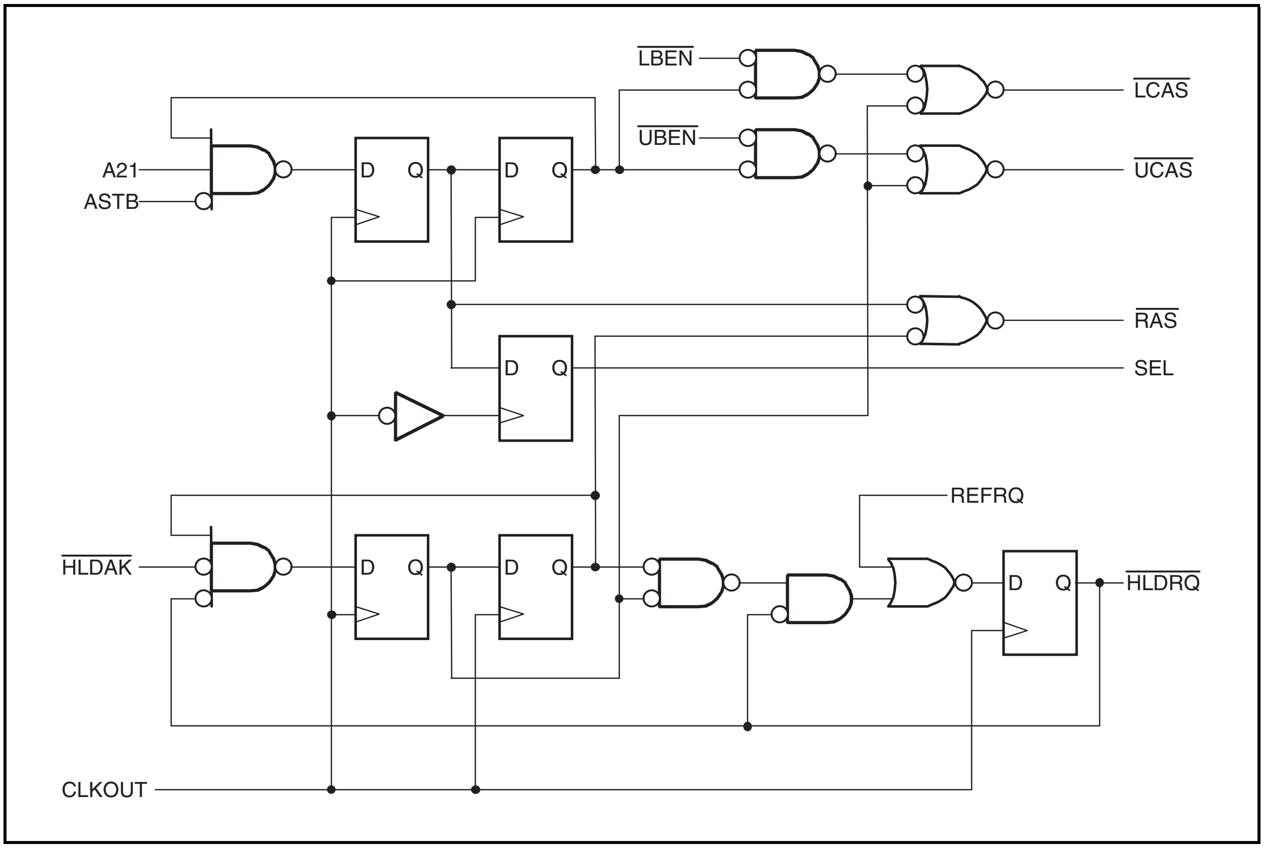


Figure 2-20. Address Output Block

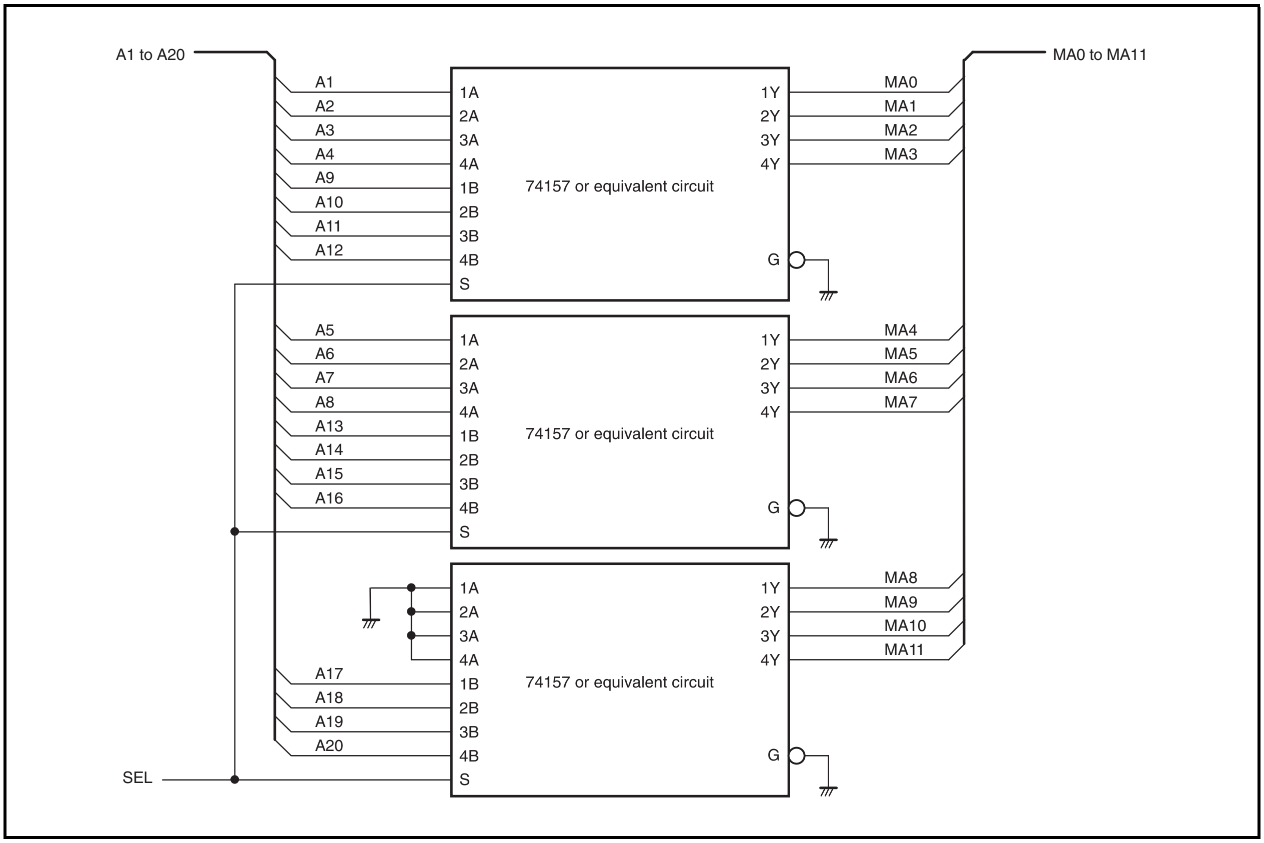
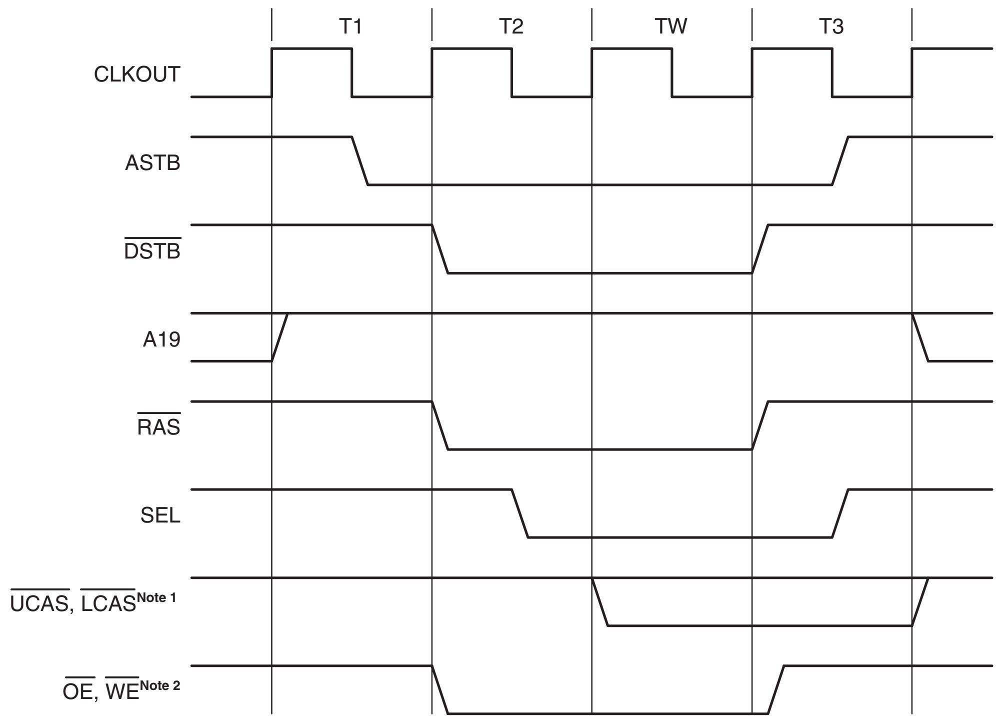


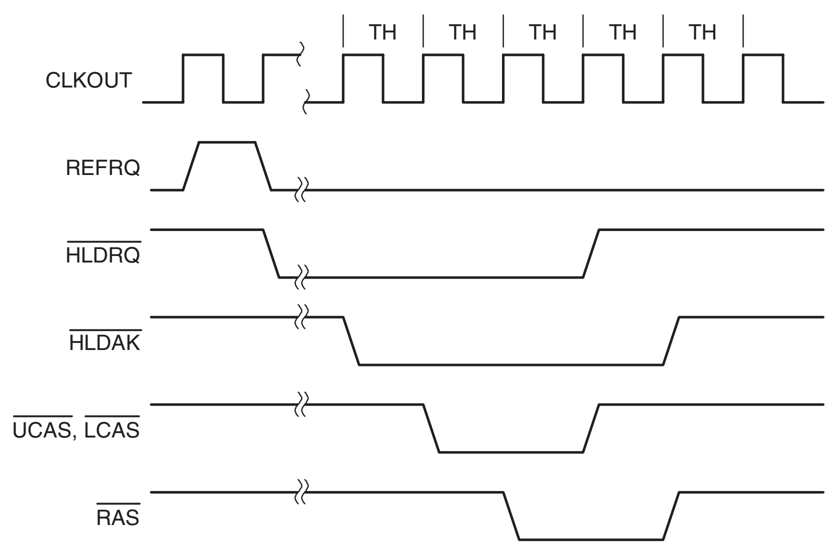
Figure 2-21. Read/Write Timing



Notes 1. When $\overline{\text{LBEN}}$ is active, $\overline{\text{LCAS}}$ becomes active.
When $\overline{\text{UBEN}}$ is active, $\overline{\text{UCAS}}$ becomes active.
When $\overline{\text{LBEN}}$ and $\overline{\text{UBEN}}$ are active, $\overline{\text{LCAS}}$ and $\overline{\text{UCAS}}$ become active.

2. $\overline{\text{OE}}$ becomes active during a read operation.
$\overline{\text{WE}}$ becomes active during a write operation.

Figure 2-22. Refresh Timing (DRAM Connection Circuit Example 1)



2.4.2 DRAM connection circuit 2 (refresh arbiter uses $\overline{\text{WAIT}}$ pin)

[Circuit configuration]

- V850/SA1's main system clock: 16 MHz
- Connected device: μPD42S16160L $\times$ 1
- Memory usage: Address range 200000H to 3FFFFFH in external memory space
- Supported cycles: Read cycle, early write cycle, and CBR refresh cycle

[DWC and BCC settings]

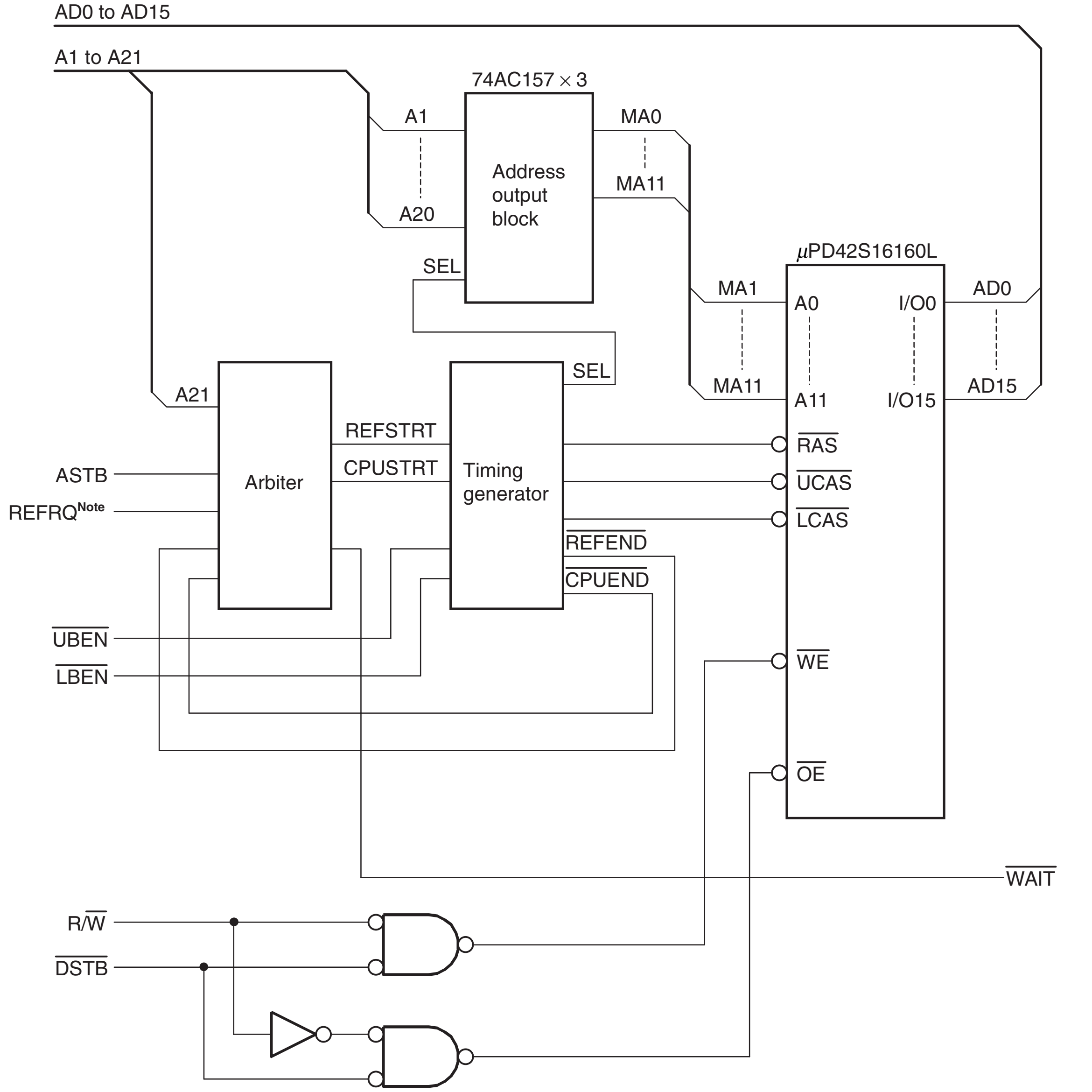
- Wait setting: 0 waits (controlled by $\overline{\text{WAIT}}$ pin)
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, $\overline{\text{UBEN}}$ signal output

[Circuit type]

- The V850/SA1's address A21 is monitored at the rising edge of the T2 clock. If the address matches, a CLKOUT signal is generated via read cycle/write cycle timing.
- An arbiter circuit is implemented to arbitrate the V850/SA1's access requests and refresh requests.
- If a V850/SA1 access request and a refresh request occur at the same time, the refresh cycle takes priority.
- If access from the V850/SA1 occurs during a refresh operation, a wait is inserted by hardware and a read cycle or write cycle is started after the refresh operation is completed.
- One[Note] refresh request is generated by the external counter at an interval of 31.25 μs.

Note Since the μPD42S16160L's refresh cycle timing is 4096 cycles per 128 μs, the refresh interval is (128000/4096 =) 31.25 (μs). Consequently, a refresh interval of 31.25 μs or less is required.
Since one cycle is 62.5 ns at 16 MHz, the refresh interval is (31.25 μs/62.5 ns) = 500. Therefore, one refresh can be performed every 500 CLKOUT cycles.

Figure 2-23. μPD42S16160L Connection Circuit Example (Refresh Arbiter Uses $\overline{\text{WAIT}}$ Pin)



Note The REFRQ signal occurs once using the high level of the clock synchronized with the CLKOUT signal as the refresh timing.

Remark μPD42S16160L: 1 M × 16 bits (16 Mb)

Figure 2-24. Arbiter

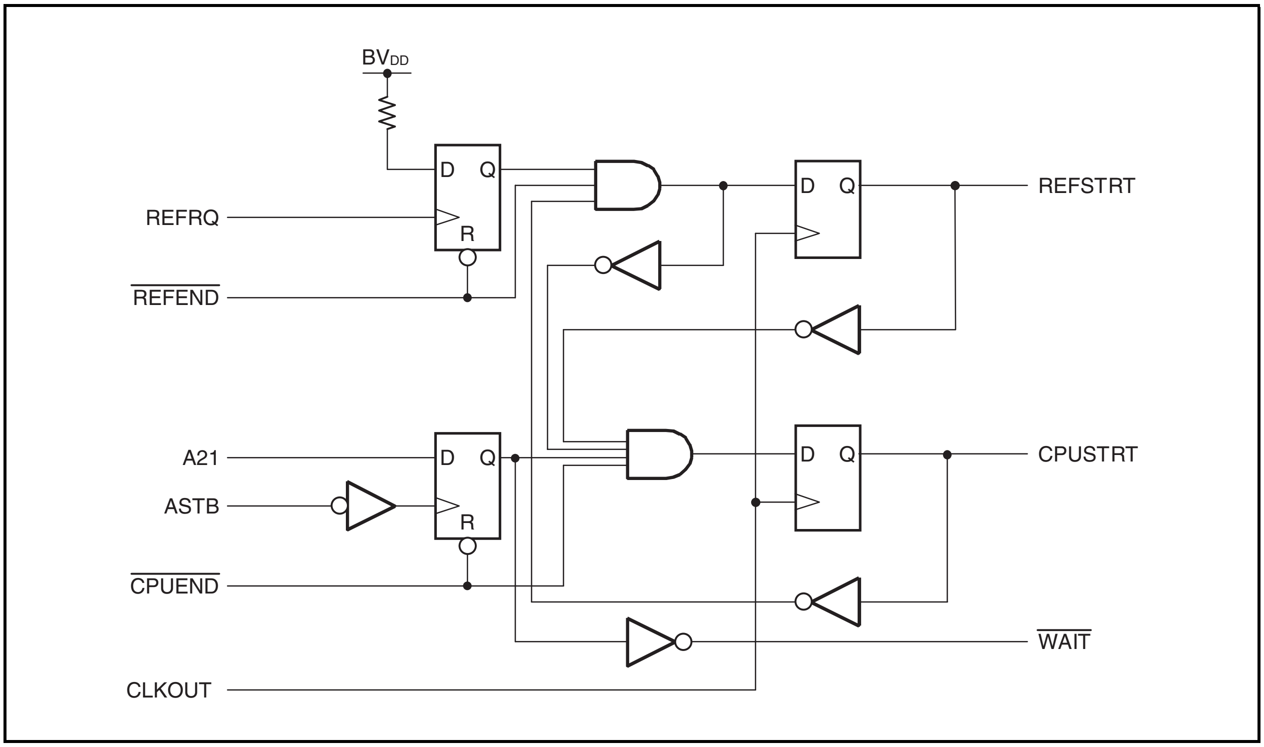


Figure 2-25. Timing Generator (DRAM Connection Circuit Example 2)

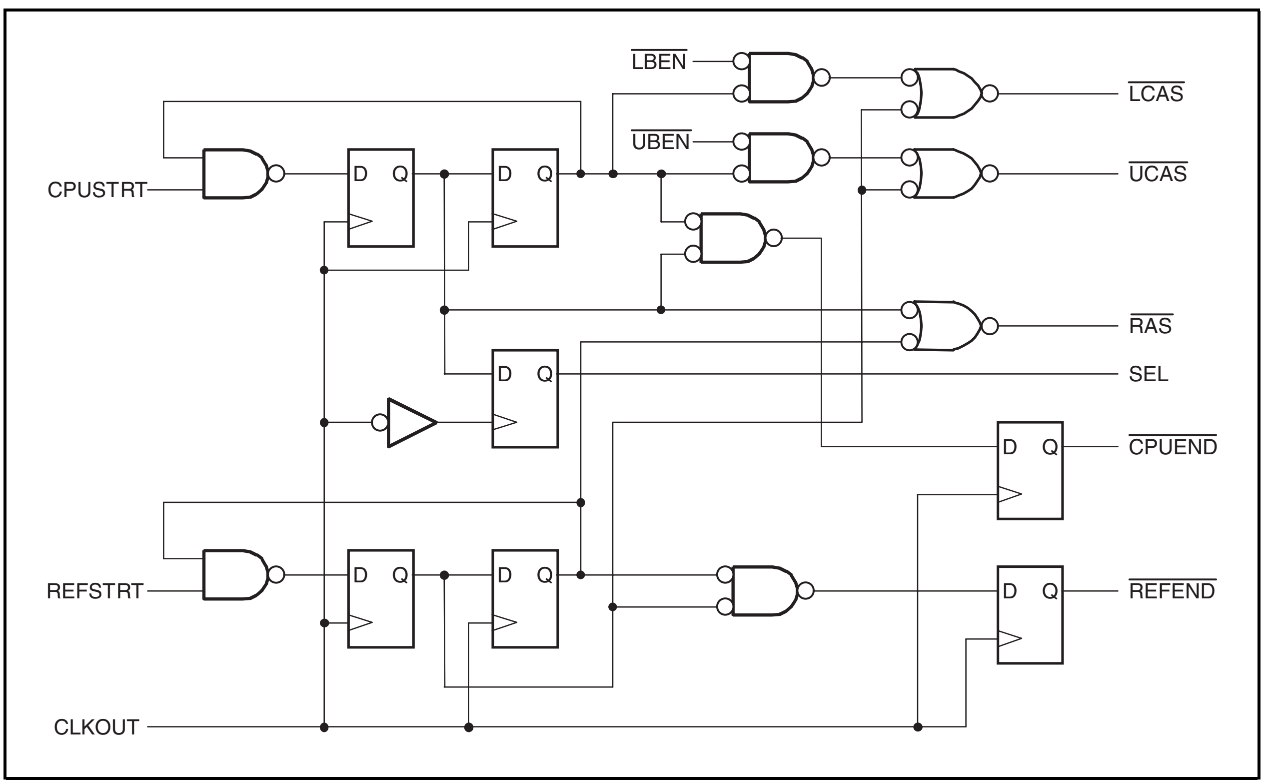
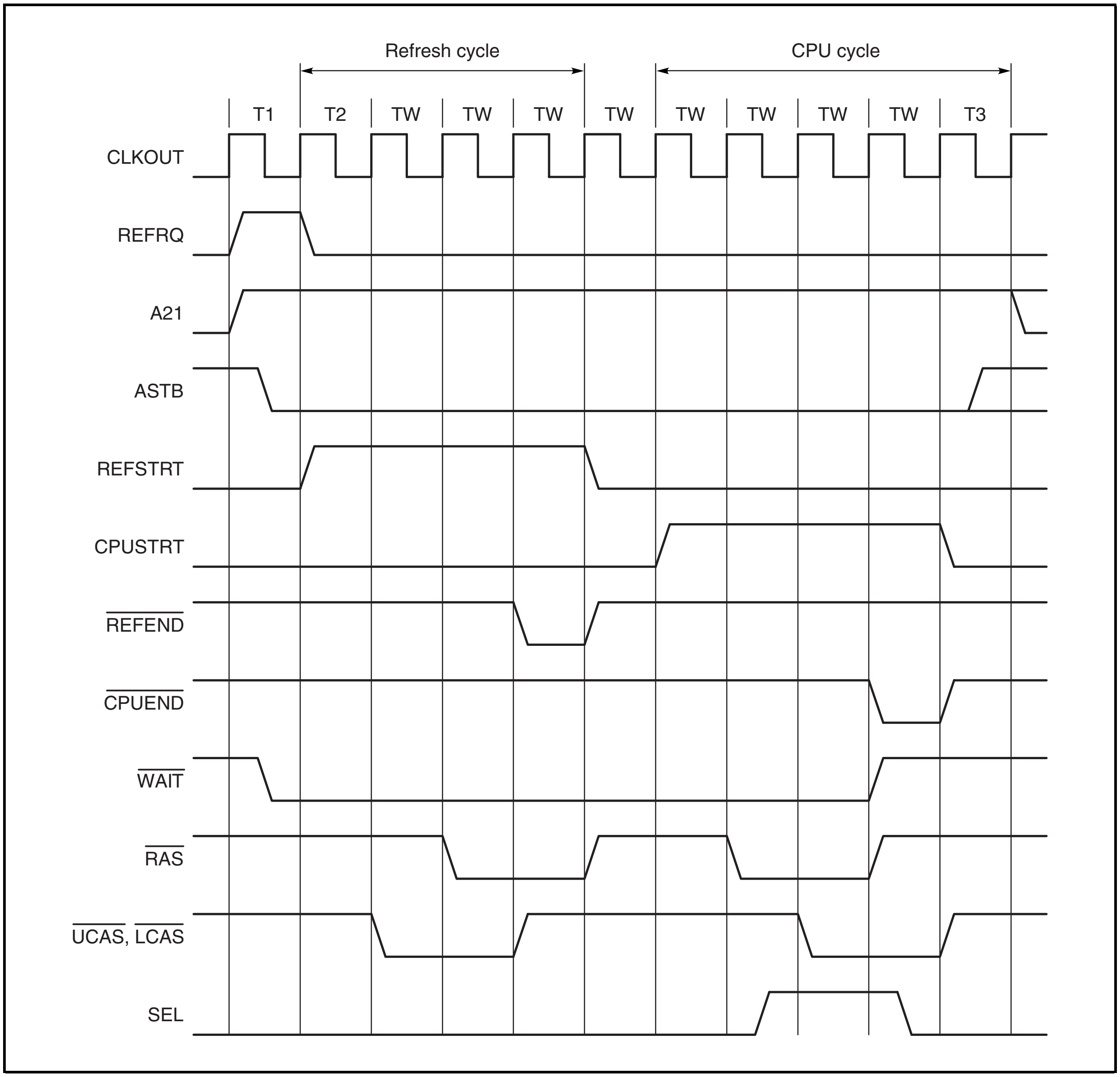


Figure 2-26. Operation of Arbiter
(When Refresh Request and V850/SA1 Access Request Occur Simultaneously)



2.5 Connection of 8-Bit PROM Using Bus Sizing Circuit

The following is an example of a V850/SA1 that is connected to an external ROM space.

In this circuit example, both 16-bit access and 8-bit access are supported. During a 16-bit read operation, read access of PROM is executed twice by hardware and the 16-bit data is arranged and transferred to the V850/SA1. Wait control is performed by hardware.

[Circuit configuration]

- V850/SA1's main system clock: 16 MHz
- Connected devices: HN27C4001G × 1
- Memory usage: Address range 000000H to 07FFFFH in external memory space

[DWC and BCC settings]

- Wait setting: 0 wait cycles (controlled by $\overline{\text{WAIT}}$ pin)
  During 8-bit read: 1 wait cycle (inserted by hardware)
  During 16-bit read: 3 wait cycles (inserted by hardware)
- Idle state: Not inserted
- Read/write control pin mode: $\overline{\text{RD}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, $\overline{\text{UBEN}}$ signal output

[Circuit design]

- ROM's $\overline{\text{CS}}$ signal ($\overline{\text{ROMCS}}$) is generated by decoding the higher 3 bits of the address.
- The A0 (signal generated when separated from AD0) and $\overline{\text{UBEN}}$ signals are decoded to determine the V850/SA1's access type.

A0	$\overline{\text{UBEN}}$	Access Type
Low level	Low level	16-bit access
Low level	High level	Even address byte access
High level	Low level	Odd address byte access

- One 8-bit buffer is used and the lower byte is latched during a 16-bit read operation.
- The CLKOUT signal is used to generate the ROM's A0 signal, and latch signal as well as the V850/SA1's $\overline{\text{WAIT}}$ signal.
- Level converting devices are used to connect ROM's I/O signals.
  ROM's A0 to A18, $\overline{\text{CS}}$, and $\overline{\text{OE}}$ pins: 74FCT244 × 3
  ROM's O0 to O7: 74FCT3373 (connects to V850/SA1's AD0 to AD7) × 1
  74FCT3373 (connects to V850/SA1's AD8 to AD15) × 1

Figure 2-27. Connection of 8-Bit PROM Using Bus Sizing Circuit

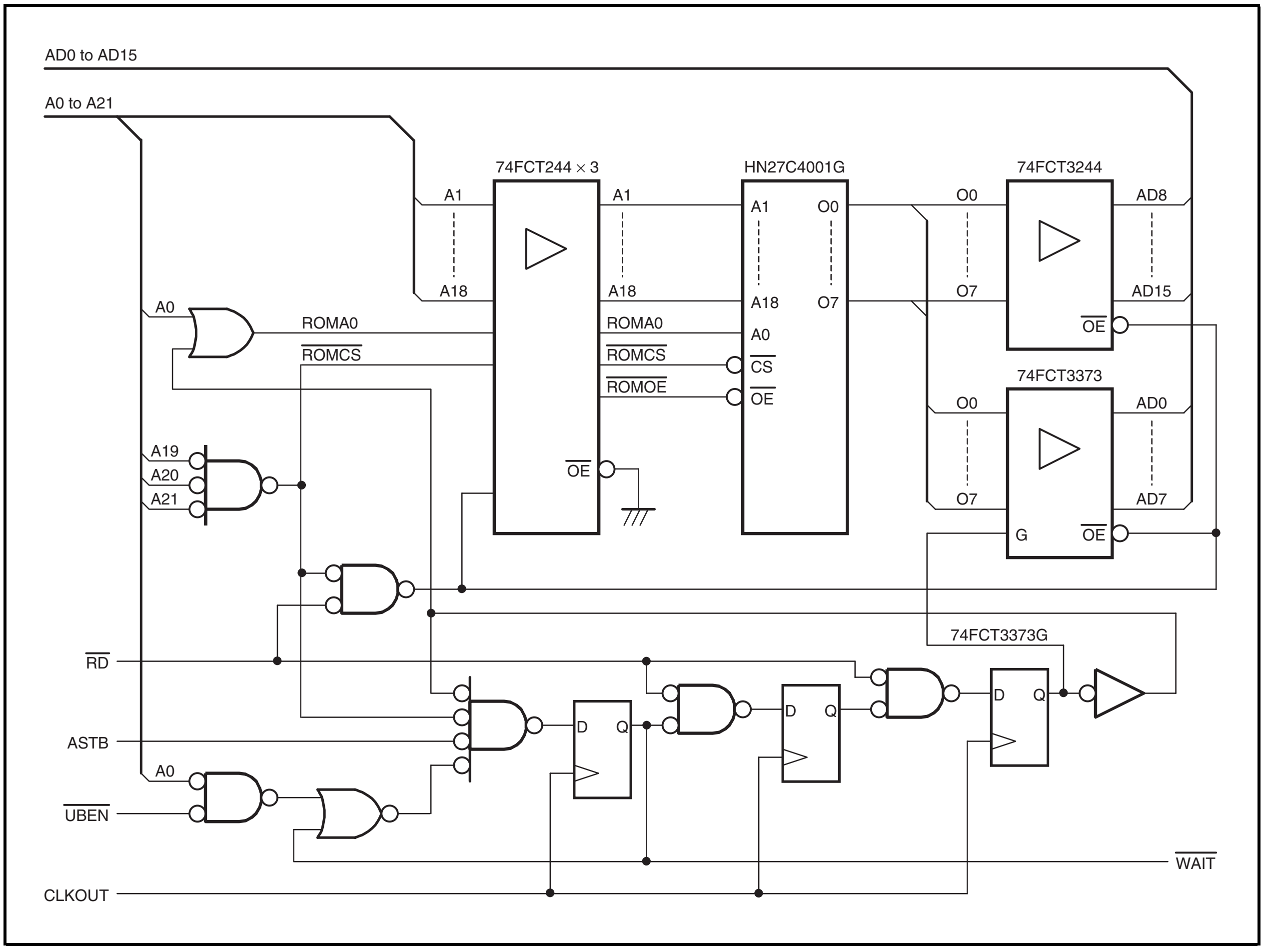


Figure 2-28. Byte Access

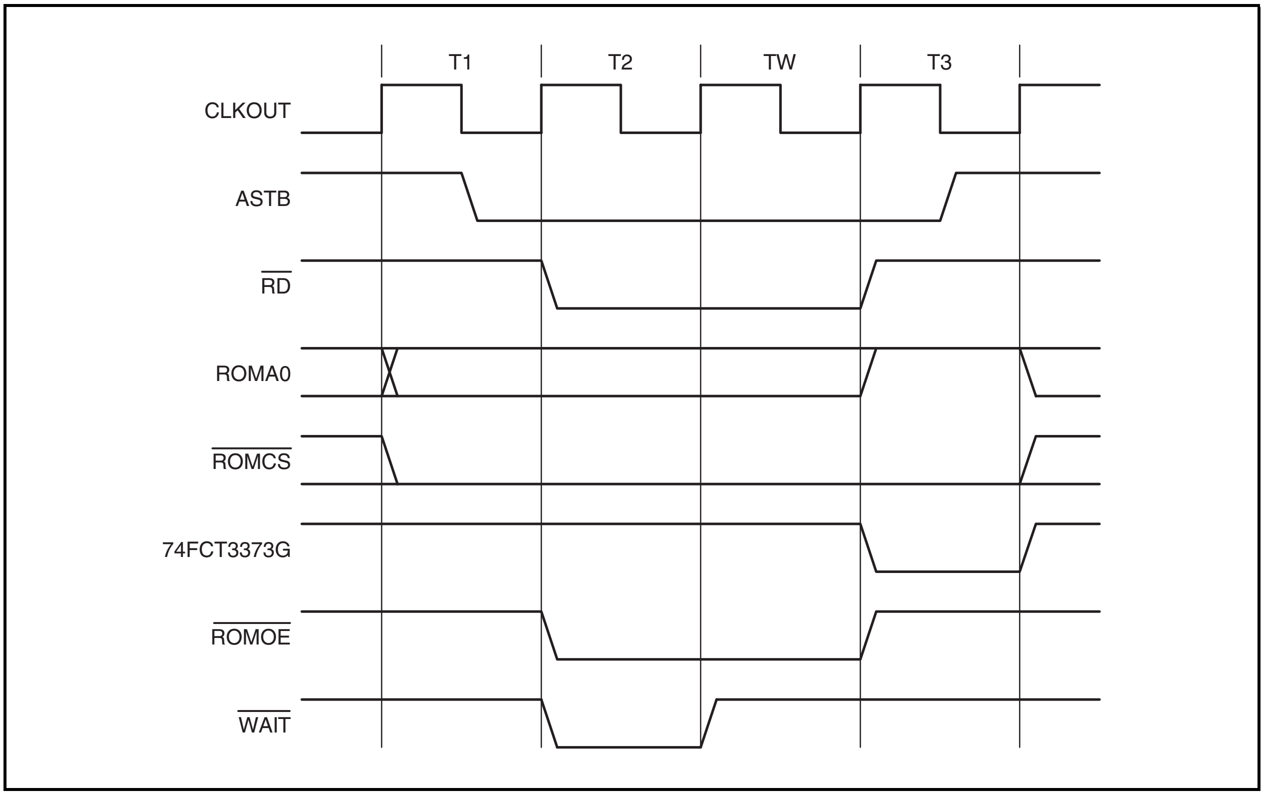
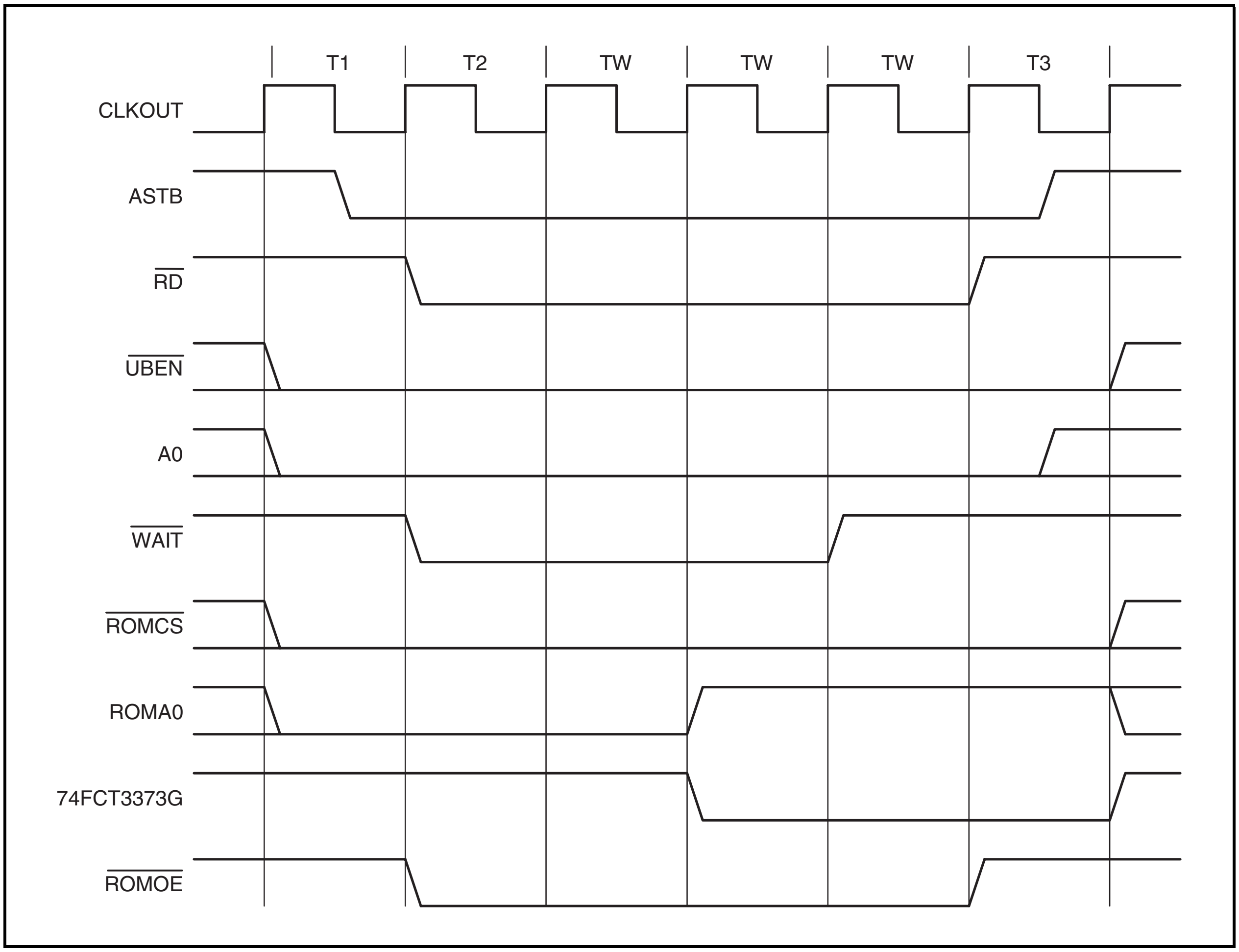


Figure 2-29. Word Access/Halfword Access



CHAPTER 3 PERIPHERAL FUNCTION CONNECTION EXAMPLES

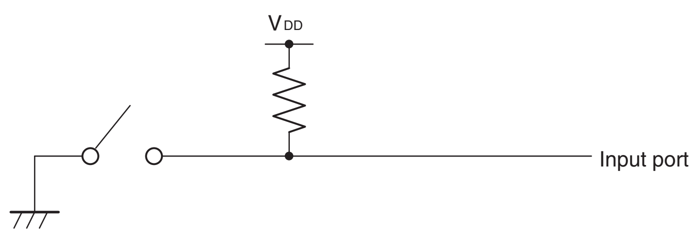
This chapter described connection examples using the V850/SA1's internal peripheral I/O functions. In these examples, the V850/SA1's V_{DD} and BV_{DD} power supply voltages are both 3.3 V.

In these circuit examples, a level converting device is used when connecting to 5 V devices, as was the case in chapter 2. In an actual circuit, the level converting device can be omitted if DC characteristics are met by the V850/SA1 and the connected device.

3.1 Pushbutton Switch Connection (Port Function)

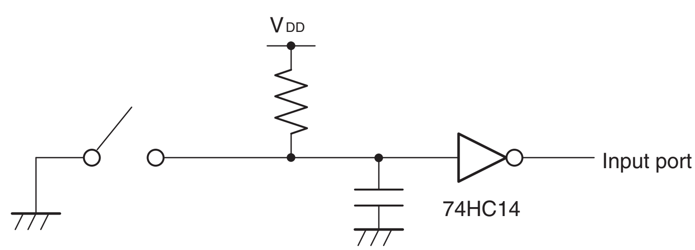
The following example shows a pushbutton switch connected to an input port.

Figure 3-1. When Using Software to Eliminate Chattering



Remarks 1. "0" is input when the switch is on, and "1" is input when it is off.
2. Use software to eliminate switch chattering.
3. The external pull-up resistor can be omitted if the port enables internal pull-up.

Figure 3-2. When Using Hardware to Eliminate Chattering



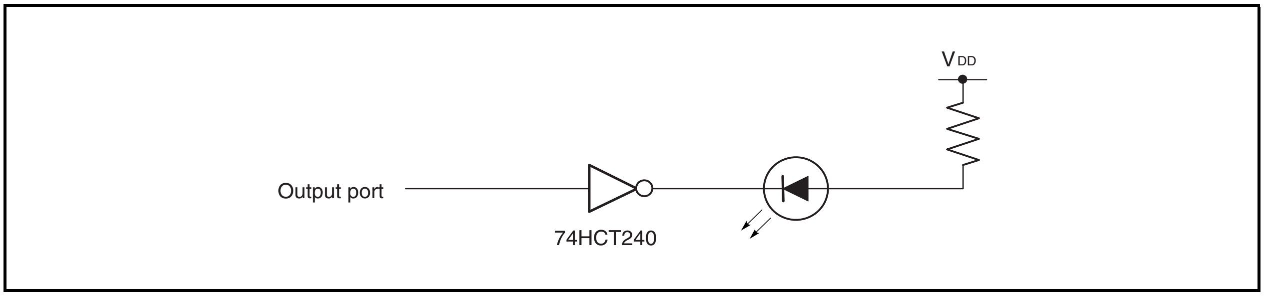
Remark "1" is input when the switch is on, and "0" is input when it is off.

3.2 LED Connection (Port Function)

An example of an LED connected to an output port is shown below.

- The LED is ON when the port output value is “1”.
- The LED is OFF when the port output value is “0”.

Figure 3-3. Example of LED Connection to Output Port

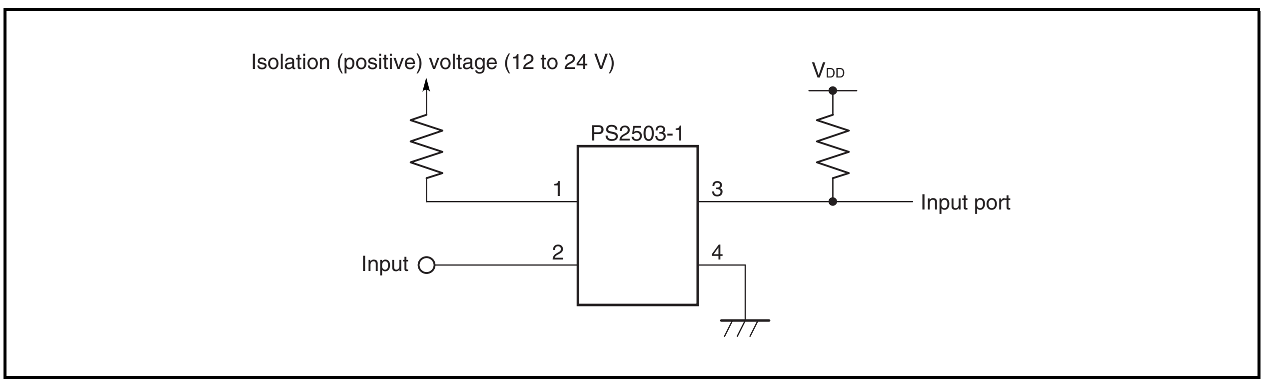


3.3 Digital Input for Isolation (Port Function)

In the following example, a photocoupler is used to isolate an input port.

- When port input is at low level, “0” is input.
- When port input is at high level, “1” is input.

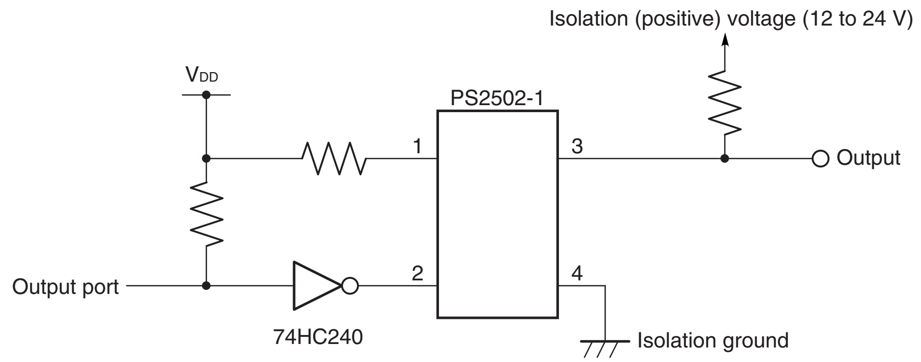
Figure 3-4. Isolation of Input Port (Example)



3.4 Digital Output for Isolation (Port Function)

In the following example, a photocoupler is used to isolate an output port.

Figure 3-5. Isolation of Output Port (Example)

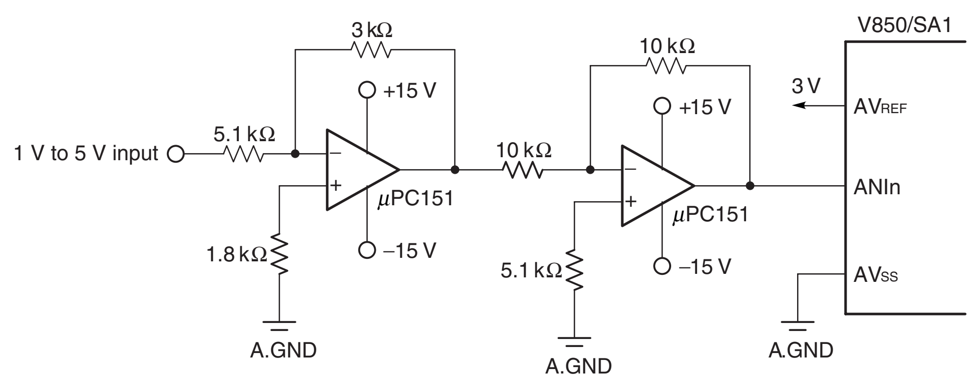


3.5 1 V to 5 V Type Analog Input (A/D Converter)

A connection example for an analog voltage 1 V to 5 V input is shown below.

- When the input voltage is 1 V, 0.59 V are input.
- When the input voltage is 5 V, 2.94 V are input.

Figure 3-6. Analog Voltage 1 V to 5 V Input Connection Example



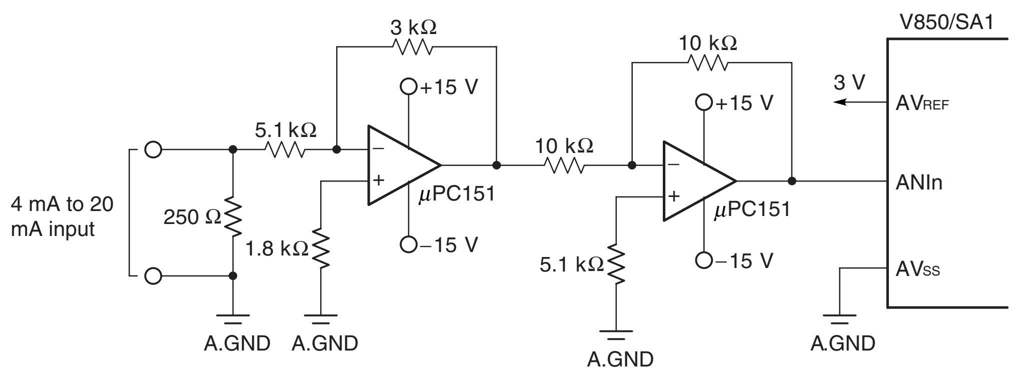
Remark n = 0 to 11

3.6 4 mA to 20 mA Type Analog Input (A/D Converter)

A connection example for a 4 mA to 20 mA analog input is shown below.

- When the input value is 4 mA, 0.59 V are input.
- When the input value is 20 mA, 2.94 V are input.

Figure 3-7. 4 mA to 20 mA Analog Input Connection Example



Remark n = 0 to 11

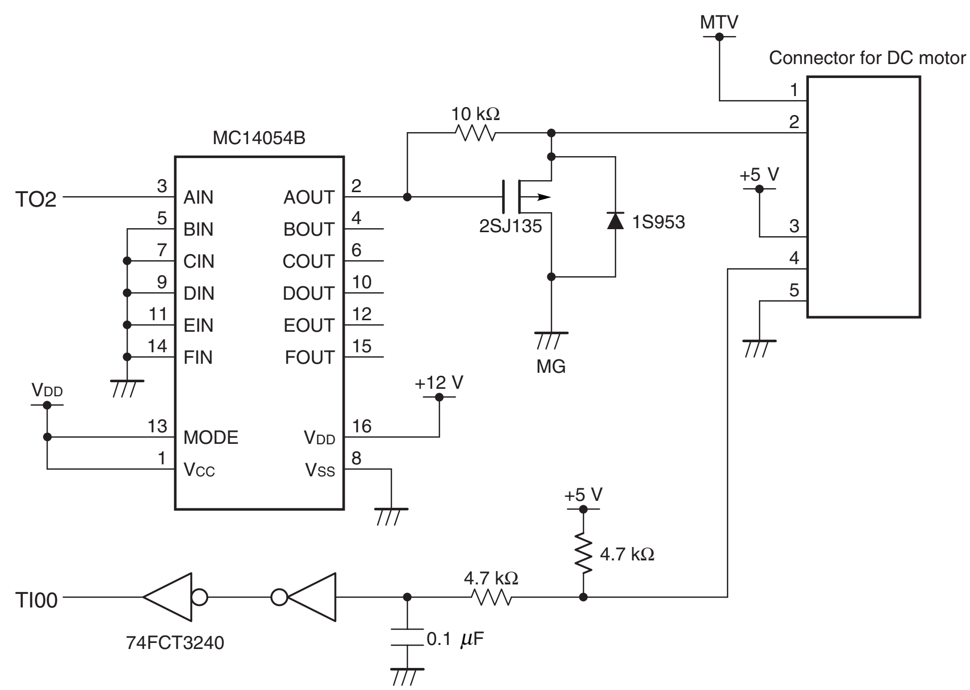
3.7 DC Motor Connection (Timer/Counter Function)

In the following example, the TM2 output (TO2) is used to control a DC motor.

Set TM2 to PWM output and change TO2's output pulse to control the rotational speed of the DC motor.

The rotary encoder pulse output from the DC motor is connected to T100 to enable monitoring of the DC motor's rotational speed.

Figure 3-8. Example of Using TM2 Output (TO2) to Control DC Motor



Remark MTV and MG are isolated from other power supplies.
The DC motor connects to DME33BM37G.

Table 3-1. Connector for DC Motor

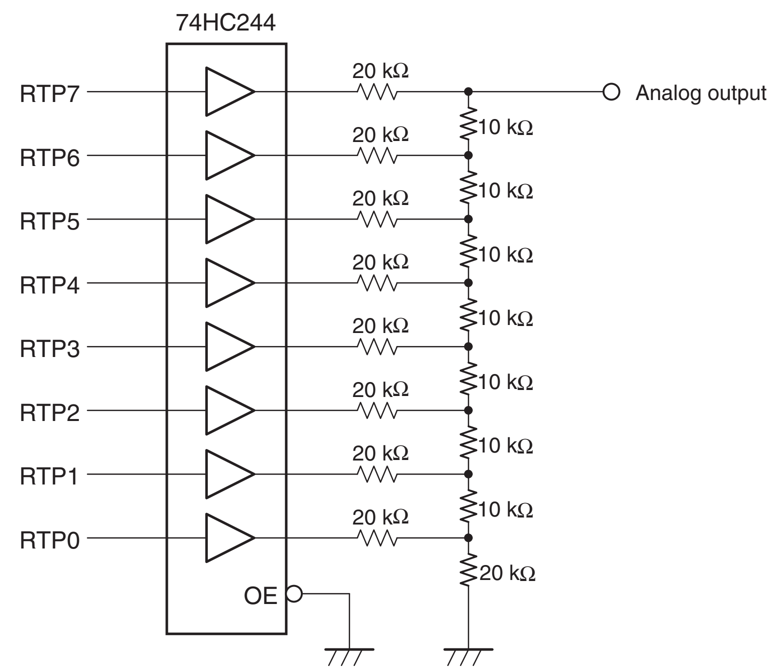
No.	Function
1	Power supply for motor (+12 V)
2	Ground for motor
3	Power supply for pulse generator
4	Pulse generator output
5	Ground for pulse generator

3.8 Analog Output (RTP) Using R-2R Circuit

Output via RTP0 to RTP7 is used in the following example which includes an 8-bit D/A converter based on an R-2R circuit.

RTP7 corresponds to the MSB and RTP0 to the LSB, and output is full-scale output using FFH.

Figure 3-9. Example of 8-Bit D/A Converter Using R-2R Circuit



3.9 RS-232C Interface Connection (UART)

In the following example, UART1 is connected as an asynchronous RS-232C interface.
The transmit/receive clock is divided from the internal clock.

Figure 3-10. Example of UART1 Connected as Asynchronous RS-232C Interface

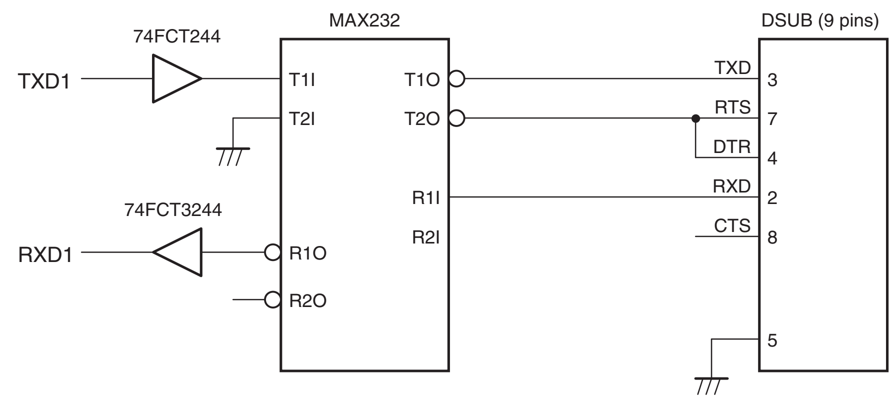


Table 3-2. RS-232C Interface Connector

No.	Signal Name	Description
1	NC	Not connected
2	RXD	Receive data
3	TXD	Transmit data
4	DTR	Terminal ready
5	SG	Signal ground
6	NC	Not connected
7	RTS	Request to send
8	CTS	Clear to send
9	NC	Not connected

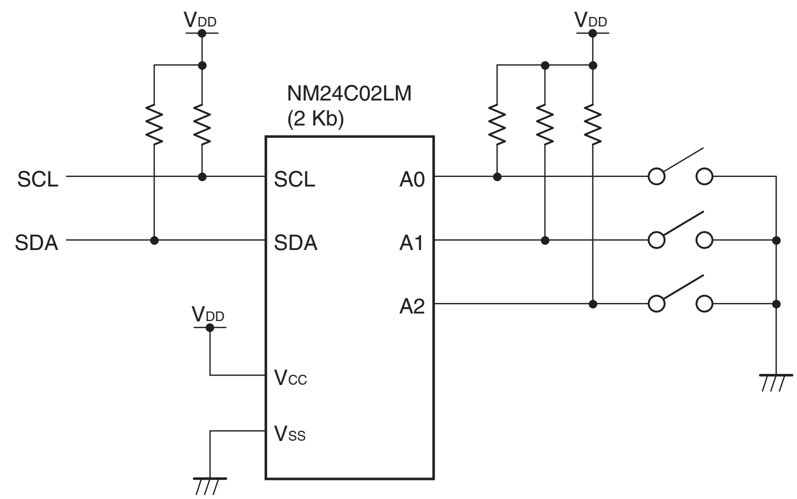
Remark RTS and DTR are both fixed to the active level.
CTS is not connected.

3.10 Serial EEPROM™ Connection (I²C)

The following example shows an I²C bus interface connected to a serial EEPROM (NM24C02LM).

The NM24C02LM's device address (A0 to A2) is set via switches. SCL and SDA are connected directly to the V850/SA1's SCL and SDA respectively.

Figure 3-11. Example of I²C Bus Interface Connected to Serial EEPROM (NM24C02LM)



3.11 Shutdown Detection Circuit and Backup Power Supply

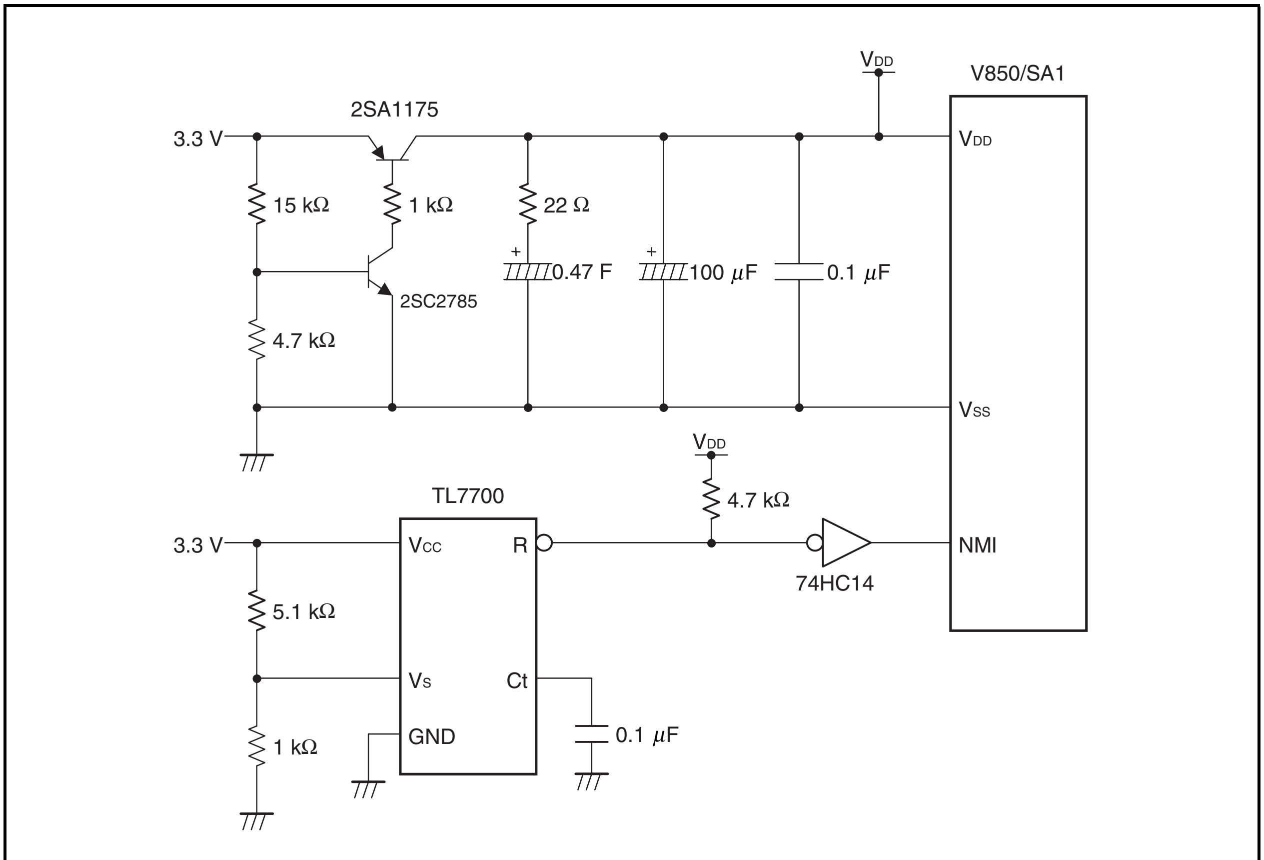
A power supply voltage monitor circuit and power backup are shown below.

In this circuit example, the NMI pin goes to high level when the power supply voltage drops below 3 V and returns to low level when the power supply voltage rises to 3 V again.

A non-maskable interrupt is designed to be triggered by either a rising edge or falling edge. When a non-maskable interrupt has occurred, if there is a program to confirm the state of the NMI pin, it provides greater safety for the system.

When a power shutdown occurs, V_{DD} is supplied from a capacitor (0.47 F).

Figure 3-12. Example of Power Supply Monitor Circuit and Power Backup



CHAPTER 4 APPLICATION EXAMPLES

This chapter describes features of the TB-V850/SA1 CPU training board that has been developed as a tool for evaluating and studying the V850/SA1.

4.1 Features of TB-V850/SA1

4.1.1 General description of TB-V850/SA1

(1) CPU clock

Main clock: 16 MHz
Subclock: 32.768 kHz

(2) CPU power supply

Power is supplied separately via the system power supply (+5 V) and CPU power supply (+3.3 V) (V_{DD}, BV_{DD}).

(3) Bus interface connections

Memory:	EPROM (128 KB)
	SRAM (128 KB)
	DRAM (2 MB)
I/O:	LCD (16 characters × two lines)

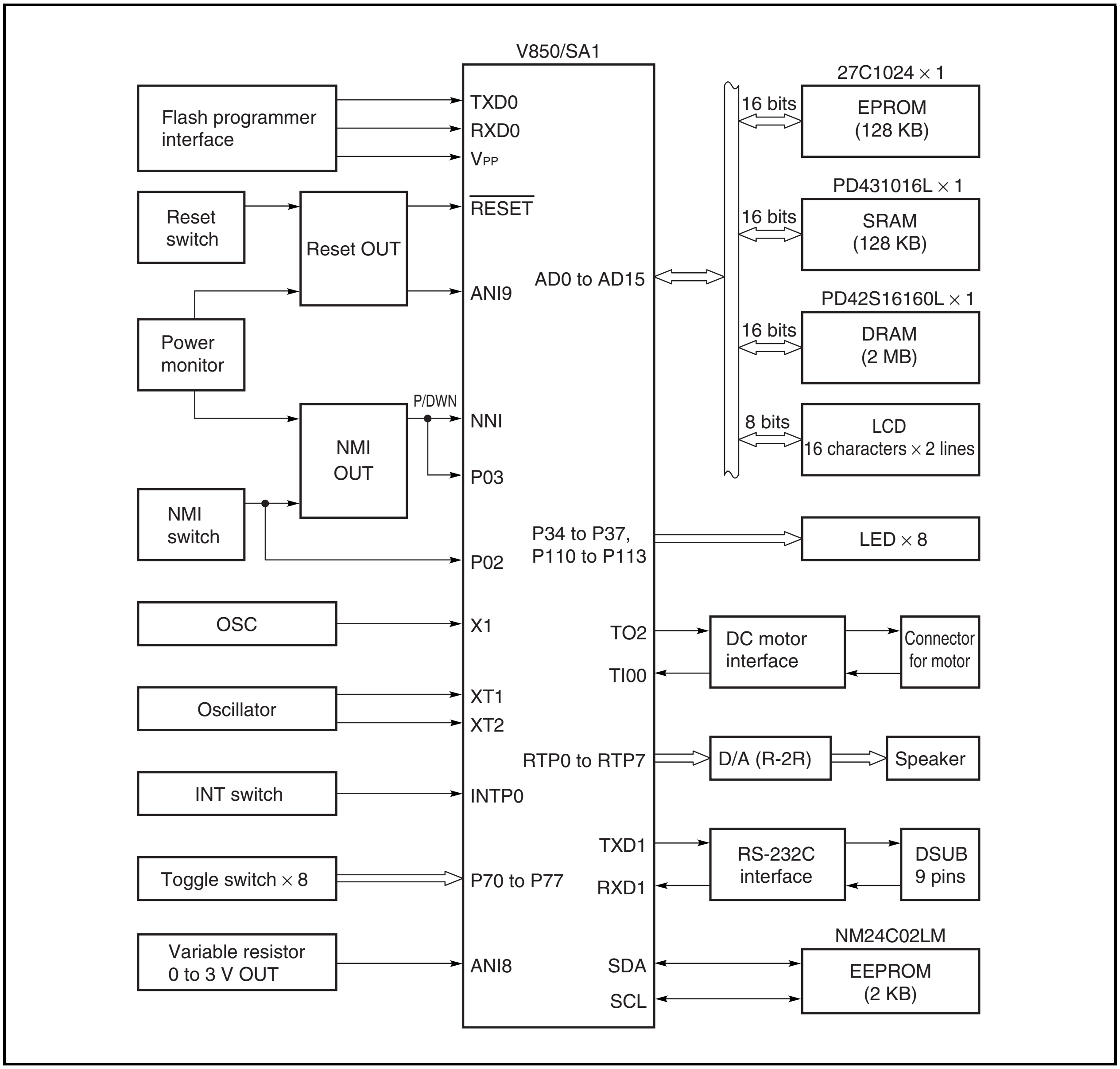
(4) Connection of peripheral functions

INTC:	NMI switch
	INT switch
	Encoder pulse from DC motor
Timer counter:	DC motor × 1
UART:	RS-232C interface
	Dedicated flash programmer interface
I^2C bus:	EEPROM (NM24C02LM)
A/D converter:	Voltage input is variable from 0 to 3 V using variable resistor
RTP:	Speaker output via R-2R circuit
Port functions:	Output port (LED × 8)
	Input port (toggle switch × 8)

4.1.2 Configuration of TB-V850/SA1

The composition of the TB-V850/SA1 is shown below.

Figure 4-1. Configuration of TB-V850/SA1



4.1.3 Bus interface connections

The V850/SA1's external bus interface connects with a PROM, SRAM, DRAM, and LCD using a 4 MB space.

On the training board, addresses AD0 to AD15 are separated. $\overline{\text{DSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{LBEN}}$, and $\overline{\text{UBEN}}$ are used as control signals.

Programmable waits are used to set wait cycles for PROM and SRAM.

Hardware waits are inserted under the control of the $\overline{\text{WAIT}}$ pin for DRAM and the LCD.

EPROM: A 128 KB area starting from address xx000000H is the ROM area.
One 27C1024 1 Mb (64 K × 16 bits) device is mounted. A one-cycle programmable wait is required when the 16 MHz system clock is used.

SRAM: A 128 KB area starting from address xx020000H is the SRAM area.
One μPD431016L 1 Mb (64 K × 16 bits) device is mounted. This area can be accessed with zero waits.

DRAM: A 2 MB area starting from address xx200000H is the DRAM area.
One μPD42S16160L 16 Mb (1 M × 16 bits) device is mounted. This area can be accessed with two wait cycles inserted by hardware. As for the refresh cycle, the hardware in this unit drives a CBR refresh cycle. When the V850/SA1's DRAM is accessed during a refresh operation, wait cycles are inserted by the board's arbiter circuit until the refresh cycle is completed. If a refresh request and access to the V850/SA1 occur at the same time, the refresh cycle takes priority.

LCD: One L1682 (16 characters × 2 lines) LCD device is mounted.
Command status access uses byte access at address xx140000H, and data write uses byte access at address xx140002H. When the LCD is accessed, 7 wait cycles are inserted by hardware. Programmable waits are not required.

4.1.4 Peripheral function connections

The V850/SA1's internal peripheral functions control various switches, LEDs, and the DC motor unit.

Table 4-1. Peripheral Function Connections

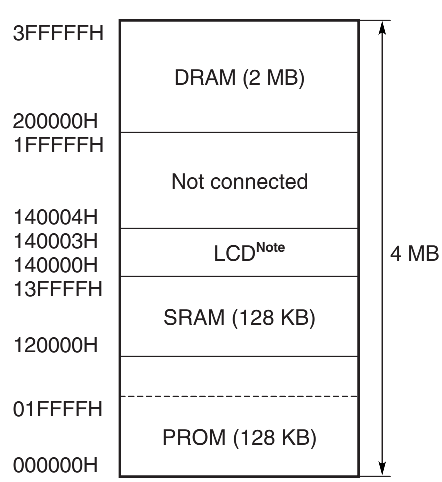
CPU Unit Name	Name of Connected Pin	Name of Connected Unit
Port function	P70 to P77	General-purpose toggle switches (× 8)
	P34 to P37, P110 to P113	General-purpose LEDs (× 8)
ICU	NMI	NMI switch
	INTP0	INT switch
UART	TXD0, RXD0	Flash programmer interface
	TXD1, RXD1	RS-232C interface
I^2C	SDA, SCL	NM24C02LM[Note] (EEPROM, 2 KB)
Timer/counter	TO2	DC motor speed control
	TI00	DC motor rotary encoder pulse
RTP	RTP0 to RTP7	Speaker
A/D	ANI8	Variable resistor (0 to 3 V)
	ANI9	System power supply (+5 V) status monitor

Note The NM24C02LM's slave address is fixed to 0 by hardware.

4.1.5 Allocation of external memory

The allocation of external memory connected to the V850/SA1's bus interface (4 MB space) is shown below. For details of accessing the V850/SA1's various memory spaces, refer to **4.3.1 Memory map**.

Figure 4-2. Bus Interface (4 MB Space)



Note For the LCD, only byte access of even-numbered addresses is valid.

4.2 TB-V850/SA1 Specifications

Table 4-2. List of TB-V850/SA1 Specifications

Item	Description
V850/SA1	Main clock = 16 MHz Connect quartz resonator to pin X1 (leave pin X2 unconnected). Subclock: 32.768 kHz Connect quartz resonator to XT1 and XT2 pins.
EPROM	Capacity: 128 KB ROM usage: One 27C1024 or equivalent device Wait cycles: 1-wait operation (when 16 MHz)
SRAM	Capacity: 256 KB SRAM usage: One μPD431016L Wait cycles: 0-wait operation (when 16 MHz)
DRAM	Capacity: 512 KB DRAM usage: One μPD42S1616L Wait cycles: 2-wait operation (controlled by hardware) Up to six wait cycles may be inserted by hardware to avoid refresh cycle conflicts.
LCD	No. of characters: 16 characters $\times$ 2 lines Model: L1682 Wait cycles: 7 wait cycles (controlled by hardware)
Serial EEPROM	One NM24C02LM Uses V850/SA1's on-chip I^2C
Switches	Toggle switches: 8 Pushbutton switches: 3 NMI switch: 1 INT switch: 1 Reset switch: 1
LED	General-purpose lamps (red): 8 Power lamp (green): 1
DC motor	Model: DME33S37G18 Connect pulse generator output (12 pulses per rotation) to TI00 pin.
Analog output	Volume-adjusted voltage level is input to ANI8. Voltage level: 0 to 3 V (variable)
Speaker	Connect RTP output via R-2R circuit for D/A conversion.
RS-232C interface	Uses V850/SA1's on-chip UART1
Flash programmer interface	Uses V850/SA1's on-chip UART0

4.3 Internal Register Settings

This section describes internal registers whose settings are determined by the TB-V850/SA1's hardware configuration. The values set to other registers are determined by the application program. Refer to **4.4 Application Program Examples**.

4.3.1 Memory map

The memory map shown in Figure 4-3 below is used in the TB-V850/SA1 for effective utilization of the V850/SA1's wait insertion function.

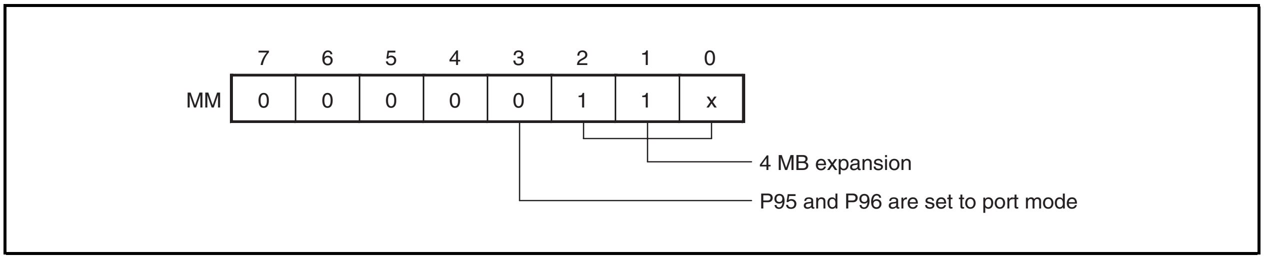
Figure 4-3. Memory Map

xxFFFFFFH xxFFC000H	Internal RAM & on-chip I/O	
xxFFBFFFH xxD40004H	Not used	
xxD40003H xxD40000H	LCD[Note] Programmable waits: Zero waits (insertion of waits is controlled by hardware) Inserted idle states: None	Block 13
xxD3FFFFH xxC00000H	Not used	
xxBFFFFFH xxA00000H	DRAM Programmable waits: Zero waits (insertion of waits is controlled by hardware) Inserted idle states: None	Blocks 10 and 11
xx9FFFFFH xx420000H	Not used	
xx41FFFFH xx400000H	PROM Programmable waits: One wait Inserted idle states: None	Block 4
xx3FFFFFH xx140000H	Not used	
xx13FFFFH xx120000H	SRAM Programmable waits: Zero waits Inserted idle states: None	Block 1
xx11FFFFH xx100000H	Not used	
xx0FFFFFH xx000000H	Internal ROM	

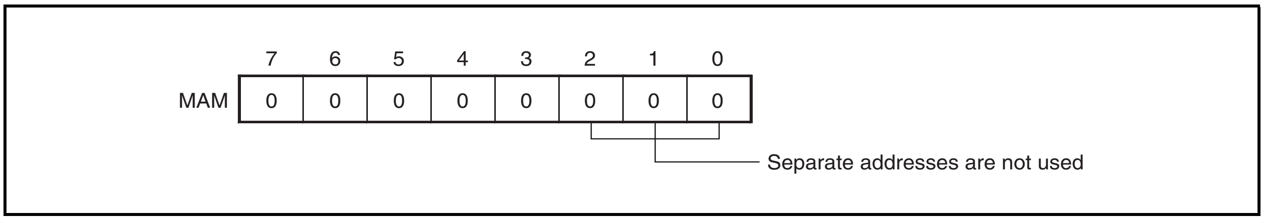
Note Even addresses can be accessed in byte units in the LCD area.

4.3.2 Bus interface-related register settings

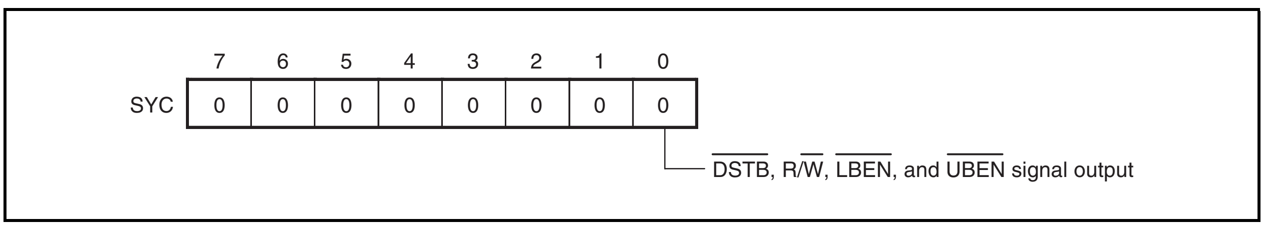
(1) Memory expansion mode register (MM) settings



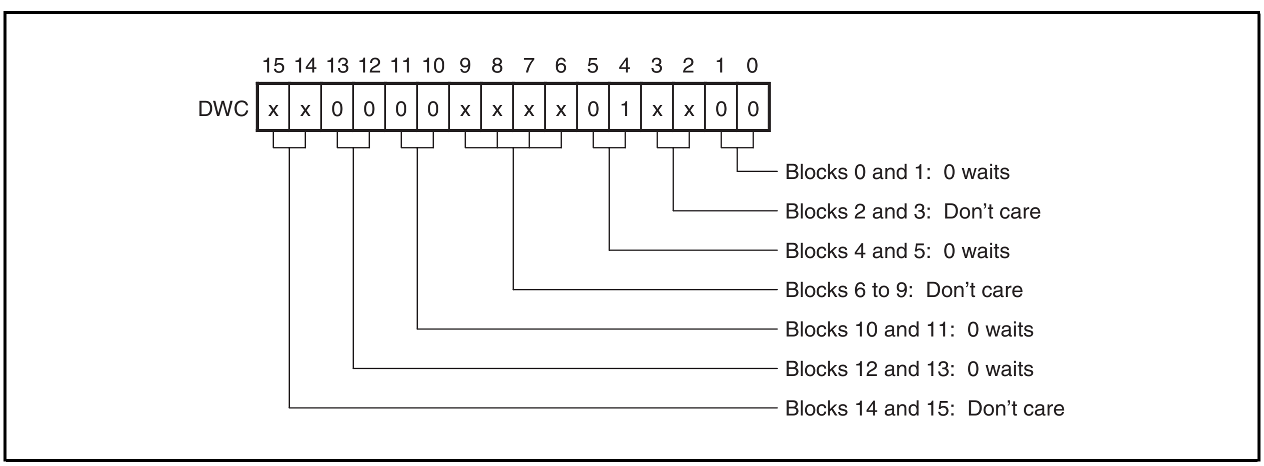
(2) Memory address output mode register (MAM) settings



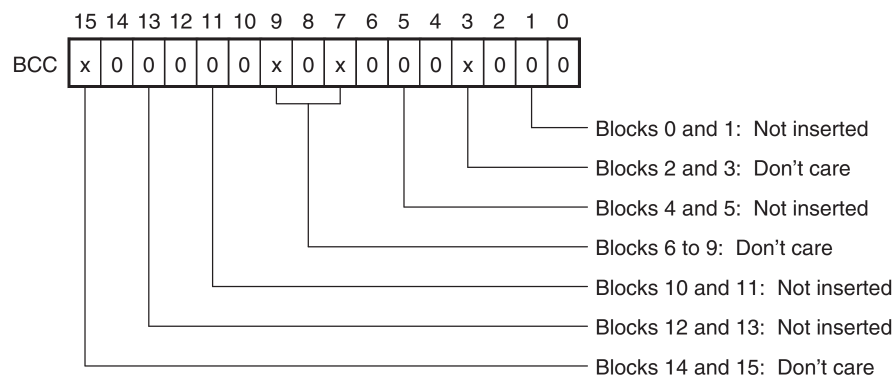
(3) System control register (SYC) settings



(4) Data wait control register (DWC) settings



(5) Bus cycle control register (BCC) settings



4.3.3 Port functions

The TB-V850/SA1's port function settings are described below.

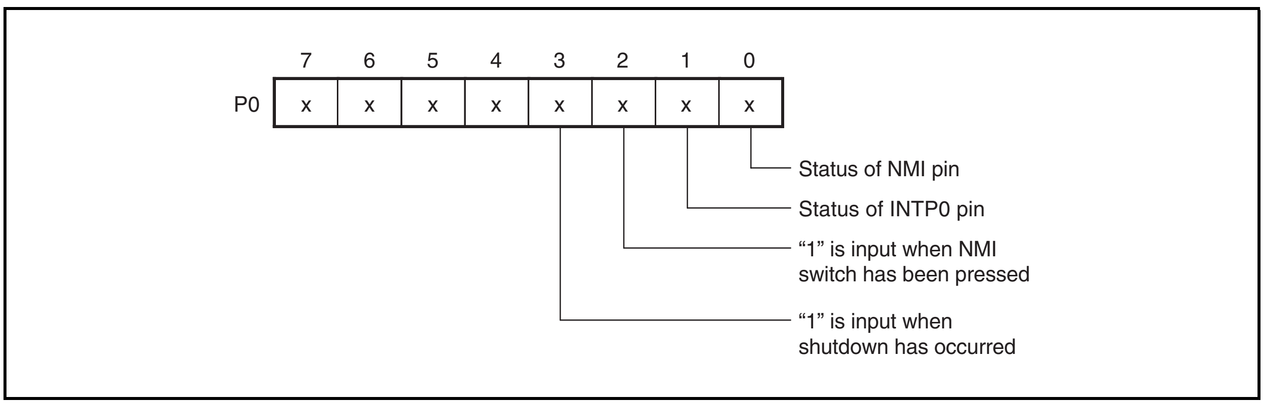
The V850/SA1's usage of each port and the corresponding register settings are also shown below.

(1) Port 0

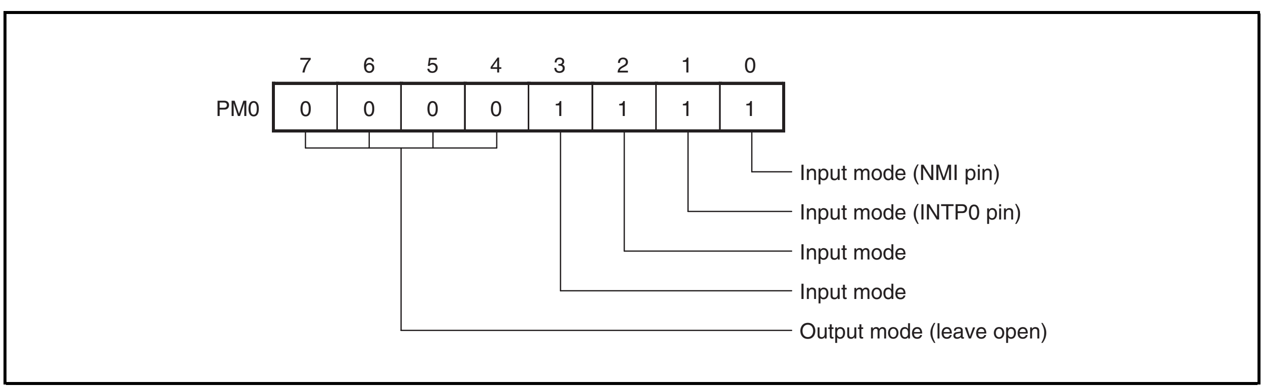
(a) Usage of port 0

Port Name	Pin(s) Used	Remark
P00	NMI	Interrupt occurs at both rising edge and falling edge
P01	INTP0	Interrupt occurs at rising edge
P02	Input port	Status monitor for NMI switch
P03	Input port	Status monitor for power supply monitor unit
P04 to P07	Not used (leave open)	Set to output mode

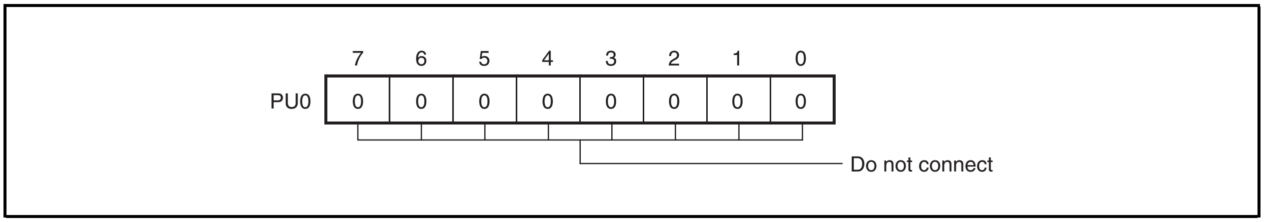
(b) Status of port 0 (P0) during read operation



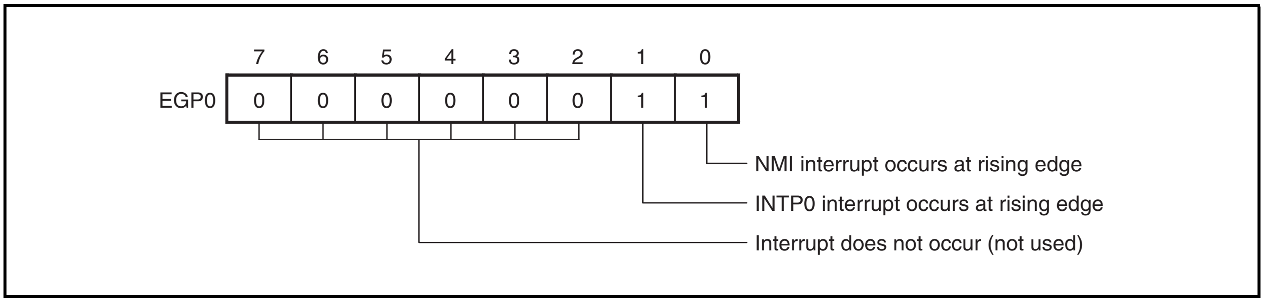
(c) Port 0 mode register (PM0) settings



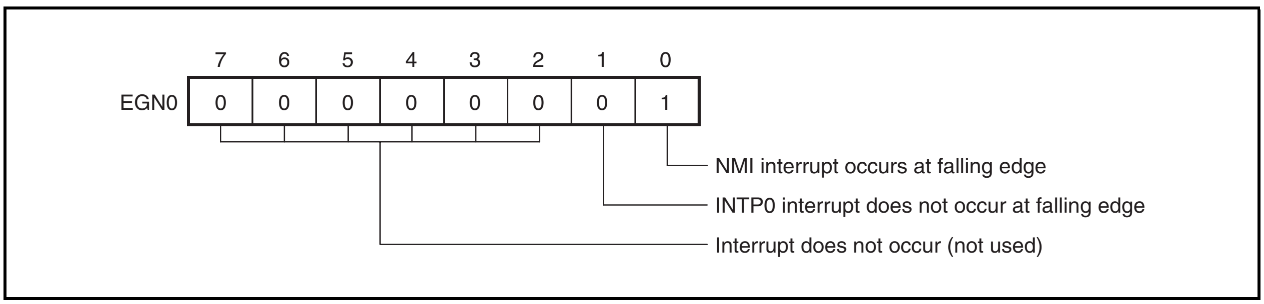
(d) Pull-up resistor option register (PU0) settings



(e) Rising edge enable register (EGP0) settings



(f) Falling edge enable register (EGN0) settings

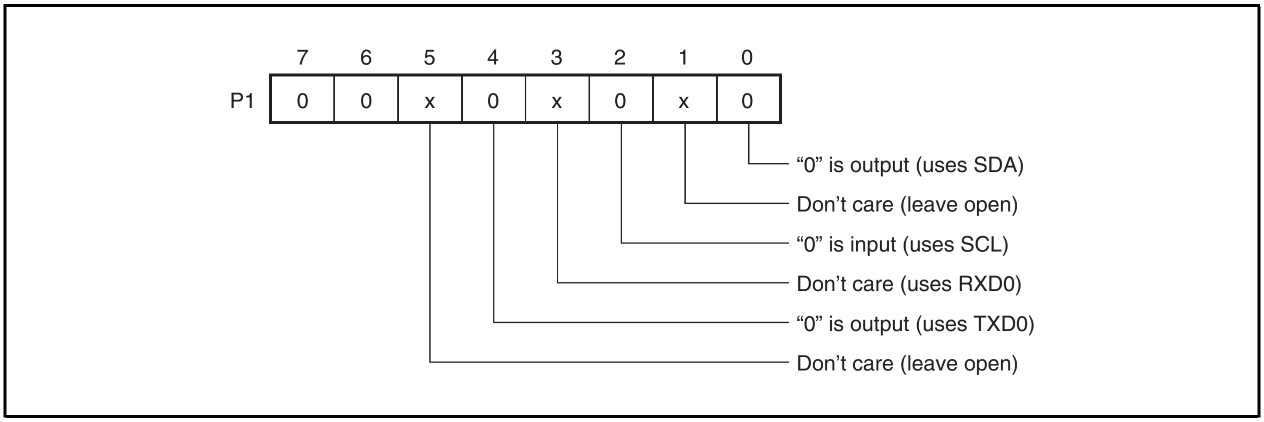


(2) Port 1

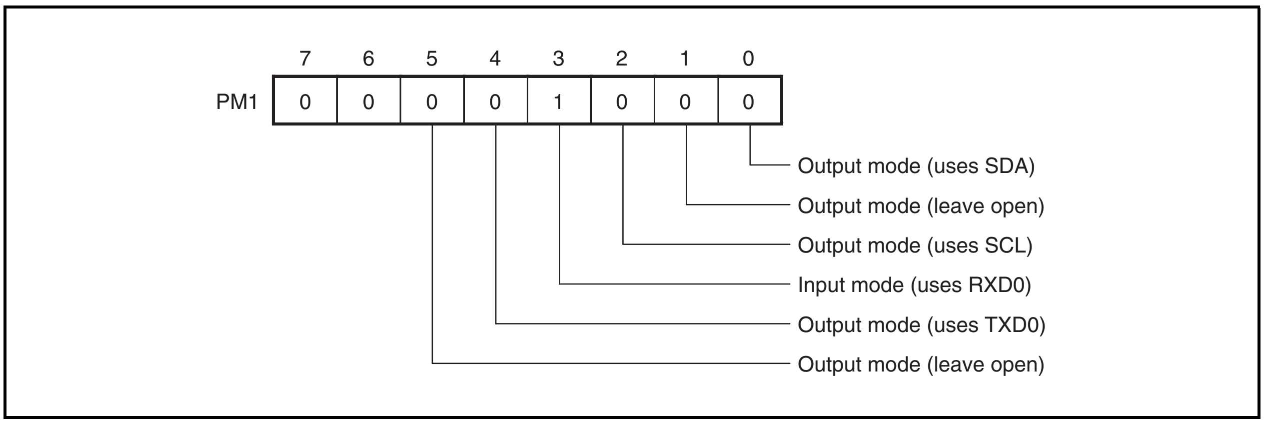
(a) Usage of port 1

Port Name	Pin(s) Used	Remark
P10	SDA	P10 is set to 0 for output mode
P11	Not used (leave open)	Set to output mode
P12	SCL	P12 is set to 0 for output mode
P13	RXD0	Set to input mode
P14	TXD0	P14 is set to 0 for output mode
P15	Not used (leave open)	Set to output mode

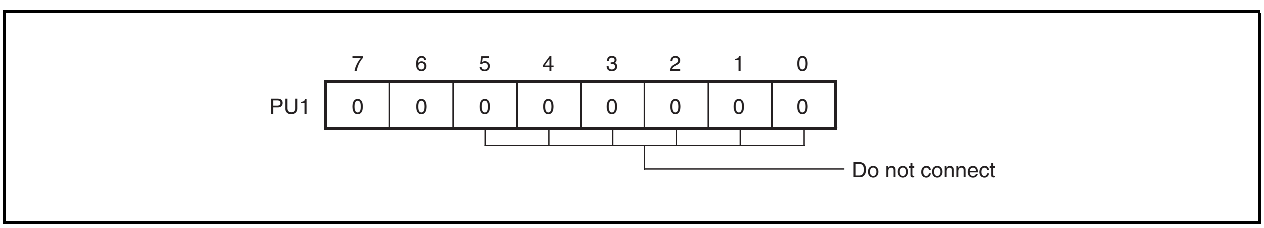
(b) Port 1 (P1) settings



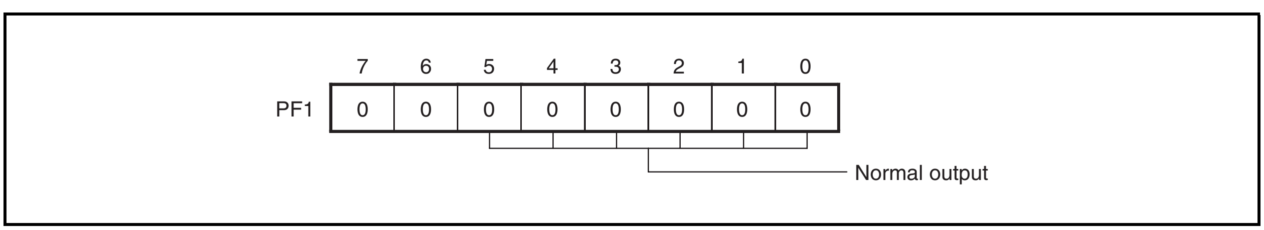
(c) Port 1 mode register (PM1) settings



(d) Pull-up resistor option register 1 (PU1) settings



(e) Port 1 function register (PF1) settings

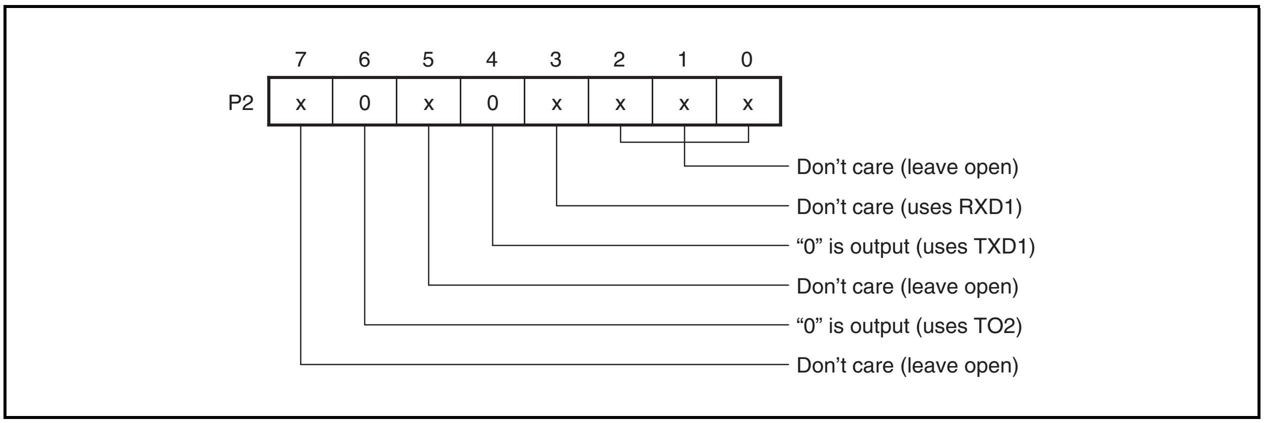


(3) Port 2

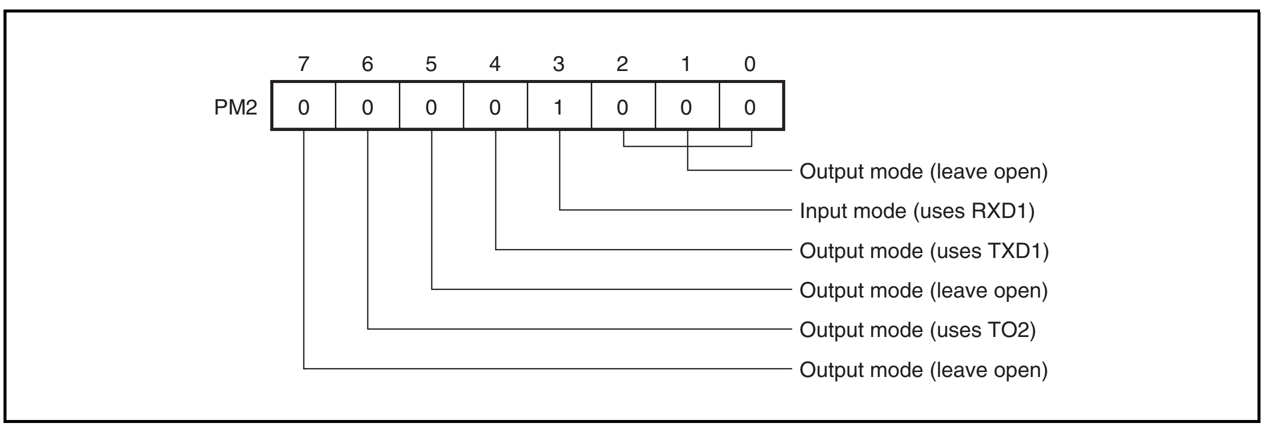
(a) Usage of port 2

Port Name	Pin(s) Used	Remark
P20 to P22, P25, P27	Not used (leave open)	Set to output mode
P23	RXD1	Set to input mode
P24	TXD1	P24 is set to 0 for output mode
P2	TO2	P26 is set to 0 for output mode

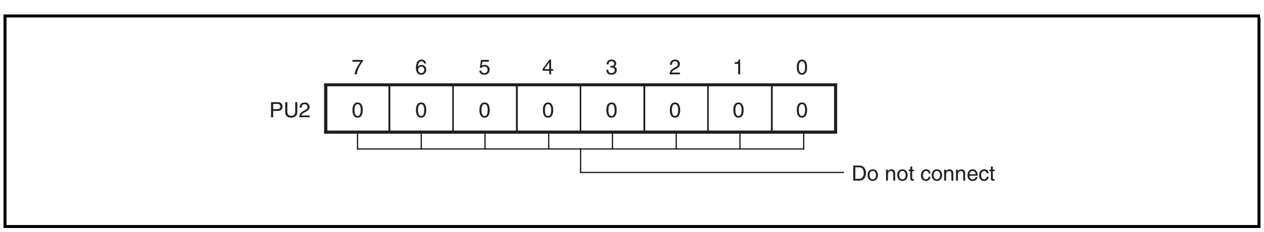
(b) Port 2 (P2) settings



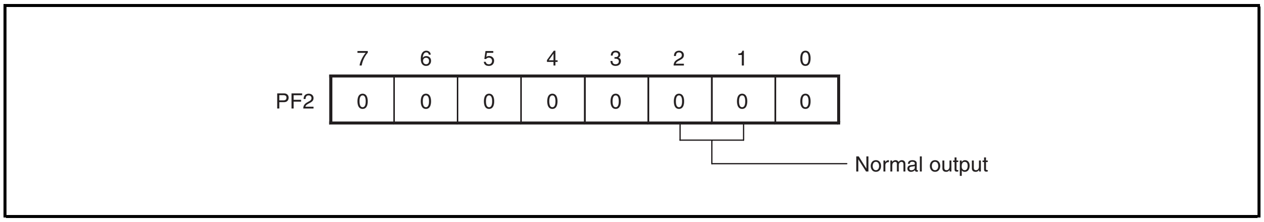
(c) Port 2 mode register (PM2) settings



(d) Pull-up resistor option register 2 (PU2) settings



(e) Port 2 function register (PF2) settings

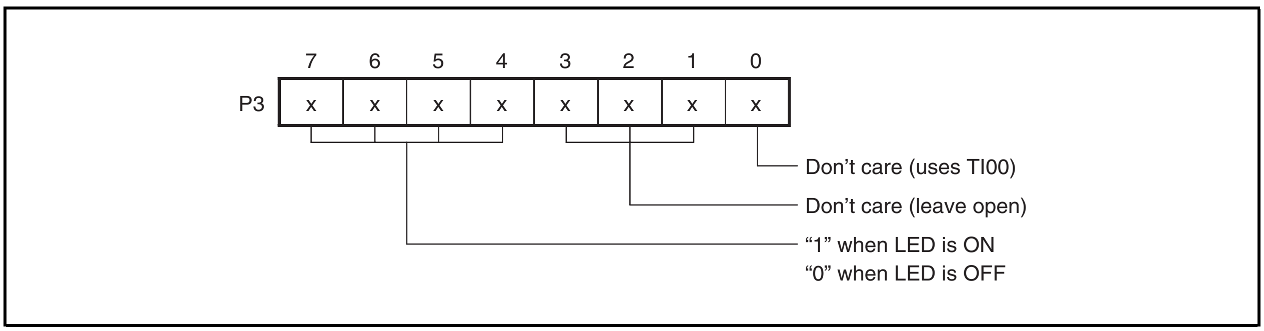


(4) Port 3

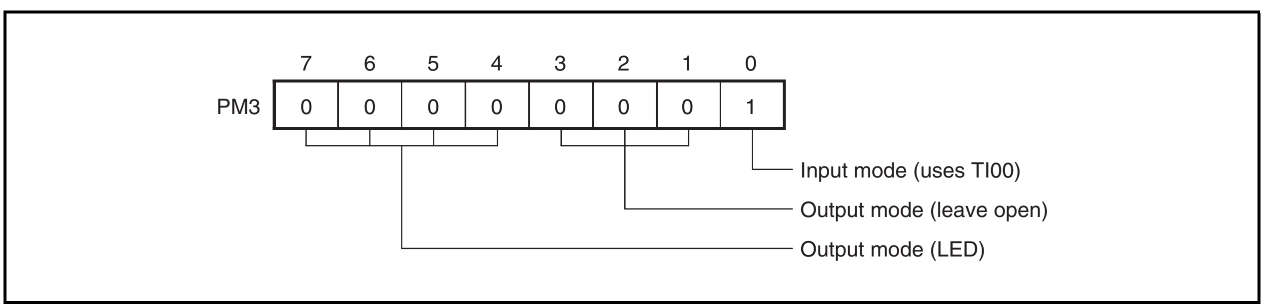
(a) Usage of port 3

Port Name	Pin(s) Used	Remark
P30	TI00	Set to input mode
P31 to P33	Not used (leave open)	Set to output mode
P34 to P37	Output port	LED output port

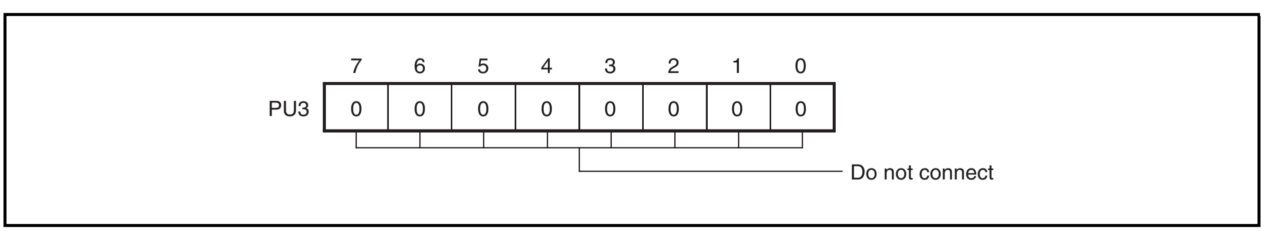
(b) Port 3 (P3) settings



(c) Port 3 mode register (PM3) settings



(d) Pull-up resistor option register 3 (PU3) settings



(5) Port 4 and port 5

(a) Usage of port 4 and port 5

When external expansion mode has been specified via the memory expansion mode register (MM), the settings for port 4 (P4), port 5 (P5), port 4 mode register (PM4), and port 5 mode register (PM5) all have no significance.

Port Name	Pin(s) Used	Remark
P40 to P47, P50 to P57	AD0 to AD15	External expansion mode is specified via MM register

(6) Port 6

(a) Usage of port 6

When 4 MB expansion mode has been specified via the memory expansion mode register (MM), the settings for port 6 (P6) and port 6 mode register (PM6) have no significance.

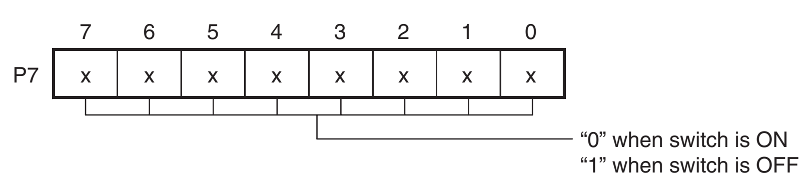
Port Name	Pin(s) Used	Remark
P60 to P65	A16 to A21	4 MB expansion mode is specified via MM register

(7) Port 7

(a) Usage of port 7

Port Name	Pin(s) Used	Remark
P70 to P77	Input port	Switch input port

(b) Status when port 7 (P7) is read

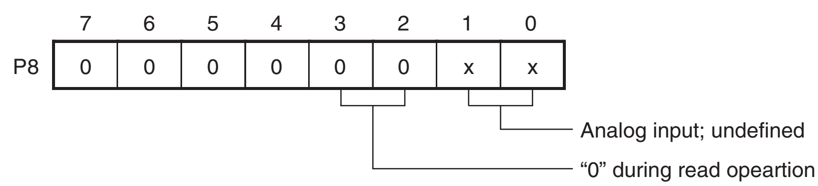


(8) Port 8

(a) Usage of port 8

Port Name	Pin(s) Used	Remark
P80	ANI8	Analog input using variable resistor
P81	ANI9	System power supply monitor
P82, P83	Not used (Connect to AV_{SS})	

(b) Status of port 8 (P8) during read operation



(9) Port 9

(a) Usage of port 9

The settings for port 9 (P9) and the port 9 mode register (PM9) have no significance.

Port Name	Pin(s) Used	Remark
P90	$\overline{\text{LBEN}}$	Specified via MM register and SYC register
P91	$\overline{\text{UBEN}}$	
P92	R/$\overline{\text{W}}$	
P93	$\overline{\text{DSTB}}$	
P94	ASTB	
P95	$\overline{\text{HLDAK}}$	
P96	$\overline{\text{HLDRQ}}$	

(10) Port 10

(a) Usage of port 10

Port Name	Pin(s) Used	Remark
P100 to P107	RTP0 to RTP7	Set to 0 for output mode

(b) Port 10 (P10) setting

	7	6	5	4	3	2	1	0
P10	0	0	0	0	0	0	0	0

“0” is output (uses RTP0 to RTP7)

(c) Port 10 mode register (PM10) settings

	7	6	5	4	3	2	1	0
PM10	0	0	0	0	0	0	0	0

Output mode (uses RTP0 to RTP7)

(d) Pull-up resistor option register 10 (PU10) settings

	7	6	5	4	3	2	1	0
PU10	0	0	0	0	0	0	0	0

Do not connect

(e) Port 10 function register (PF10) settings

	7	6	5	4	3	2	1	0
PF10	0	0	0	0	0	0	0	0

Normal output

(11) Port 11

(a) Usage of port 11

Port Name	Pin(s) Used	Remark
P110 to P113	Output port	LED output port
P114	XT1	Fixed to input mode

(b) Port 11 (P11) settings

	7	6	5	4	3	2	1	0
P11	0	0	0	x	x	x	x	x

Bits 3 to 0: "1" when LED is ON, "0" when LED is OFF

Bit 4: Don't care (uses XT1)

(c) Port 11 mode register (PM11) settings

	7	6	5	4	3	2	1	0
PM11	0	0	0	1	0	0	0	0

Bits 3 to 0: Output mode (LED)

(d) Pull-up resistor option register 11 (PU11) settings

	7	6	5	4	3	2	1	0
PU11	0	0	0	0	0	0	0	0

Bits 3 to 0: Do not connect

(12) Port 12

(a) Usage of port 12

Port Name	Pin(s) Used	Remark
P120	$\overline{\text{WAIT}}$	Specifies input port

(b) Port 12 (P12) settings

	7	6	5	4	3	2	1	0
P12	0	0	0	0	0	0	0	x

Bit 0: Don’t care (uses $\overline{\text{WAIT}}$)

(c) Port 12 mode register (PM12) settings

	7	6	5	4	3	2	1	0
PM12	0	0	0	0	0	0	0	1

Bit 0: Input mode (uses $\overline{\text{WAIT}}$)

(d) Port 12 mode control register (PMC12) settings

	7	6	5	4	3	2	1	0
PMC12	0	0	0	0	0	0	0	1

Bit 0: $\overline{\text{WAIT}}$ pin

4.4 Application Program Examples

Examples of programs that are run in the TB-V850/SA1 are shown below. These program examples are coded in assembly language and C language (code in 4.4.10 and 4.4.11 are in assembly language only).

In the assembly language programs, it is assumed that the program is linked to the assembler's initialization program.

In the C language programs, it is assumed that the C startup module has been linked with C reset processing.

The clocks supplied to the CPU are the main clock (16 MHz) and subclock (32.768 kHz).

(1) Assembler's initialization program

```
###### Reset interrupt processing.
.text
        .section        "RESET"
        jr      init

.text
############# Initializes the I/O register ##############
init:
        mov     0x06,   r11
        st.b    r11,    MM[r0]              -- Setting in memory expansion mode register
        mov     0x00,   r11
        st.b    r11,    MAM[r0]             -- Setting in memory address output mode register
        mov     0x00,   r11
        st.b    r11,    SYC[r0]             -- Setting in system control register
        movea   0x10,   r0, r11
        st.h    r11,    DWC[r0]             -- Setting in data wait control register
        mov     0x00,   r11
        st.h    r11,    BCC[r0]             -- Setting in bus cycle control register
        mov     0x0f,   r11
        st.b    r11,    PM0[r0]             -- Setting in port 0 mode register
        mov     0x00,   r11
        st.b    r11,    PU0[r0]             -- Setting in pull-up resistor option register 0
        mov     0x03,   r11
        st.b    r11,    EGP0[r0]            -- Setting in rising edge enable register
        mov     0x01,   r11
        st.b    r11,    EGN0[r0]            -- Setting in falling edge enable register
        mov     0x08,   r11
        st.b    r11,    PM1[r0]             -- Setting in port 1 mode register
        mov     0x00,   r11
        st.b    r11,    P1[r0]              -- Setting in port 1 register
        mov     0x00,   r11
        st.b    r11,    PU1[r0]             -- Setting in pull-up resistor option register 1
        mov     0x00,   r11
        st.b    r11,    PF1[r0]             -- Setting in port 1 function register
        mov     0x08,   r11
        st.b    r11,    PM2[r0]             -- Setting in port 2 mode register
        mov     0x00,   r11
        st.b    r11,    P2[r0]              -- Setting in port 2 register
        mov     0x00,   r11
        st.b    r11,    PU2[r0]             -- Setting in pull-up resistor option register 2
        mov     0x00,   r11
        st.b    r11,    PF2[r0]             -- Setting in port 2 function register
        mov     0x01,   r11
        st.b    r11,    PM3[r0]             -- Setting in port 3 mode register
        mov     0x00,   r11
        st.b    r11,    P3[r0]              -- Setting in port 3 register
        mov     0x00,   r11
        st.b    r11,    PU3[r0]             -- Setting in pull-up resistor option register 3
        mov     0x00,   r11
        st.b    r11,    PM10[r0]            -- Setting in port 10 mode register
        mov     0x00,   r11
        st.b    r11,    P10[r0]             -- Setting in port 10 register
```

```
mov     0x00,   r11
st.b    r11,    PU10[r0]        -- Setting in pull-up resistor option register 10
mov     0x00,   r11
st.b    r11,    PF10[r0]        -- Setting in port 10 function register
mov     0x10,   r11
st.b    r11,    PM11[r0]        -- Setting in port 11 mode register
mov     0x00,   r11
st.b    r11,    P11[r0]         -- Setting in port 11 register
mov     0x00,   r11
st.b    r11,    PU11[r0]        -- Setting in pull-up resistor option register 11
mov     0x01,   r11
st.b    r11,    PM12[r0]        -- Setting in port 12 mode register
mov     0x01,   r11
st.b    r11,    PMC12[r0]       -- Setting in port 12 mode control register

stsr    5,      r12
mov     r12,    r13
or      0x20,   r13
mov     0x00,   r11
ldsr    r13,    5
st.b    r0,     PRCMD[r0]       -- Write to PRCMD
## Setting in processor control register
st.b    r11,    PCC[r0]         -- CPU clock selected for fxx
ldsr    r12,    5
nop                             -- Dummy instructions (5 instructions)
nop
nop
nop
nop
jr      start
```

(2) C startup module

```
#   Copyright (C) NEC Corporation 1994,1995,1996
#   All rights reserved by NEC Corporation. This program must be used solely
#   for the purpose for which it was furnished by NEC Corporation. No part
#   of this program may be reproduced or disclosed to others, in any form,
#   without the prior written permission of NEC Corporation.

#       @(#)crtN850.s   1.13 96/05/31 17:10:34

#=============================================================================
# NAME
#    crtN850.s -  start up module for ca850
#
# DESCRIPTIONS:
#      This assembly program is a sample of start-up module for ca850.
#    If you modified this program, you must assemble this file, and
#    locate a given directory.
#
#    Unless -G is specified, sections are located as the following.
#
#                          |              :              |
#                          |              :              |
#                   tp->-  +-----------------------------+ __start   __tp_TEXT
#                          |           start up          |
#                          |-----------------------------|
#           text section   |                             |
#                          |         user program        |
#                          |                             |
#                          |-----------------------------|
#                          |           library           |
#                      -   +-----------------------------+ __argc
#                          |              0              |
#                          |-----------------------------| __argv
#           data section   |            #.L16            |
#                          |-----------------------------| .L16
#                          |       0x0,0x0,0x0,0x0       |
#                      -   +-----------------------------+
#                          |                             |
#          sdata section   |                             |
#                          |                             |
#                   gp->-  +-----------------------------+ __ssbss
#                          |                             |
#           sbss section   |                             |
#                          |                             |
#                          +-----------------------------+ __stack   __esbss   __sbss
#                          |          stack area         |
#            bss section   |                             |
#                          |         0x10000 bytes       |
#                   sp->-  +-----------------------------+ __stack + STACKSIZE   __ebss
#                          |              :              |
#                          |              :              |
#                      -   +-----------------------------+
#         sedata section   |                             |
#                          |                             |
#                      -   +-----------------------------+ __ssebss
#          sebss section   |                             |
#                      -   +-----------------------------+ __esebss
#                          |              :              |
#                   ep->-  +-----------------------------+ __ep_DATA
#         tidata section   |                             |
#                      -   +-----------------------------+
#         sidata section   |                             |
#                      -   +-----------------------------+
#                          |              :              |
#                          |              :              |
#
#=============================================================================
```

```
#-----------------------------------------------------------------------------
#       register mode
#-----------------------------------------------------------------------------
        .option reg_mode 5 5

#-----------------------------------------------------------------------------
#       special symbols
#-----------------------------------------------------------------------------
        .extern __tp_TEXT, 4
        .extern __gp_DATA, 4
        .extern __ep_DATA, 4
        .extern __ssbss, 4
        .extern __esbss, 4
        .extern __sbss, 4
        .extern __ebss, 4
        .extern __ssebss, 4
        .extern __esebss, 4


#-----------------------------------------------------------------------------
#       C program main function
#-----------------------------------------------------------------------------
        .extern _main


#-----------------------------------------------------------------------------
#       for argv
#-----------------------------------------------------------------------------
        .data
        .size   __argc, 4
        .align  4
__argc:
        .word   0
        .size   __argv, 4
__argv:
        .word   #.L16
.L16:
        .byte   0
        .byte   0
        .byte   0
        .byte   0


#-----------------------------------------------------------------------------
#       dummy data declaration for creating sbss section
#-----------------------------------------------------------------------------
        .sbss
        .lcomm  __sbss_dummy, 0, 0


#-----------------------------------------------------------------------------
#       dummy data declaration for creating sebss section
#-----------------------------------------------------------------------------
        .sebss
        .lcomm  __sebss_dummy, 0, 0


#-----------------------------------------------------------------------------
#       system stack
#-----------------------------------------------------------------------------
        .set    STACKSIZE, 0x4000
        .bss
        .lcomm  __stack, STACKSIZE, 4
```

```
#---------------------------------------------------------------------------
#       start up
#               pointers:  tp - text pointer
#                          gp - global pointer
#                          sp - stack pointer
#                          ep - element pointer
#               mask reg:  r20 - 0xff
#                          r21 - 0xffff
#       exit status is set to r10
#---------------------------------------------------------------------------
        .text
        .align  4

        .globl  __start
        .globl  _exit
        .globl  __exit
__start:
        mov     #__tp_TEXT, tp                  -- set tp register
        mov     #__gp_DATA, gp                  -- set gp register offset
        add     tp, gp                          -- set gp register
        mov     #__stack+STACKSIZE, sp          -- set sp register
        mov     #__ep_DATA, ep                  -- set ep register
#
        .option nowarning
        movea   0xff,   r20                     -- set mask register
        movea   0xffff, r0, r21
        andi    0xffff, r21, r21                -- set mask register
        .option warning
#
        mov     0x06,   r11
        st.b    r11,    MM[r0]                  -- Setting in memory expansion mode register
        mov     0x00,   r11
        st.b    r11,    MAM[r0]                 -- Setting in memory address output mode register
        mov     0x00,   r11
        st.b    r11,    SYC[r0]                 -- Setting in system control register
        movea   0x10,   r0, r11
        st.h    r11,    DWC[r0]                 -- Setting in data wait control register
        mov     0x00,   r11
        st.h    r11,    BCC[r0]                 -- Setting in bus cycle control register
        mov     0x0f,   r11
        st.b    r11,    PM0[r0]                 -- Setting in port 0 mode register
        mov     0x00,   r11
        st.b    r11,    PU0[r0]                 -- Setting in pull-up resistor option register 0
        mov     0x03,   r11
        st.b    r11,    EGP0[r0]                -- Setting in rising edge enable register
        mov     0x01,   r11
        st.b    r11,    EGN0[r0]                -- Setting in falling edge enable register
        mov     0x08,   r11
        st.b    r11,    PM1[r0]                 -- Setting in port 1 mode register
        mov     0x00,   r11
        st.b    r11,    P1[r0]                  -- Setting in port 1 register
        mov     0x00,   r11
        st.b    r11,    PU1[r0]                 -- Setting in pull-up resistor option register 1
        mov     0x00,   r11
        st.b    r11,    PF1[r0]                 -- Setting in port 1 function register
        mov     0x08,   r11
        st.b    r11,    PM2[r0]                 -- Setting in port 2 mode register
        mov     0x00,   r11
        st.b    r11,    P2[r0]                  -- Setting in port 2 register
        mov     0x00,   r11
        st.b    r11,    PU2[r0]                 -- Setting in pull-up resistor option register 2
        mov     0x00,   r11
        st.b    r11,    PF2[r0]                 -- Setting in port 2 function register
        mov     0x01,   r11
        st.b    r11,    PM3[r0]                 -- Setting in port 3 mode register
        mov     0x00,   r11
        st.b    r11,    P3[r0]                  -- Setting in port 3 register
        mov     0x00,   r11
```

```
        st.b    r11,    PU3[r0]              -- Setting in pull-up resistor option register 3
        mov     0x00,   r11
        st.b    r11,    PM10[r0]             -- Setting in port 10 mode register
        mov     0x00,   r11
        st.b    r11,    P10[r0]              -- Setting in port 10 register
        mov     0x00,   r11
        st.b    r11,    PU10[r0]             -- Setting in pull-up resistor option register 10
        mov     0x00,   r11
        st.b    r11,    PF10[r0]             -- Setting in port 10 function register
        mov     0x10,   r11
        st.b    r11,    PM11[r0]             -- Setting in port 11 mode register
        mov     0x00,   r11
        st.b    r11,    P11[r0]              -- Setting in port 11 register
        mov     0x00,   r11
        st.b    r11,    PU11[r0]             -- Setting in pull-up resistor option register 11
        mov     0x01,   r11
        st.b    r11,    PM12[r0]             -- Setting in port 12 mode register
        mov     0x01,   r11
        st.b    r11,    PMC12[r0]            -- Setting in port 12 mode control register

        stsr    5,      r12
        mov     r12,    r13
        or      0x20,   r13
        mov     0x00,   r11
        ldsr    r13,    5
        st.b    r0,     PRCMD[r0]            -- Write to PRCMD
        ## Setting in processor control register
        st.b    r11,    PCC[r0]              -- CPU clock selected for fxx
        ldsr    r12,    5
        nop                                  -- Dummy instructions (5 instructions)
        nop
        nop
        nop
        nop
#
        mov     #__ssbss, r13                -- clear sbss section
        mov     #__esbss, r12
        cmp     r12, r13
        jnl     .L11
.L12:
        st.w    r0, [r13]
        add     4, r13
        cmp     r12, r13
        jl      .L12
.L11:
#
        mov     #__sbss, r13                 -- clear bss section
        mov     #__ebss, r12
        cmp     r12, r13
        jnl     .L14
.L15:
        st.w    r0, [r13]
        add     4, r13
        cmp     r12, r13
        jl      .L15
.L14:
#
        mov     #__ssebss, r13               -- clear sebss section
        mov     #__esebss, r12
        cmp     r12, r13
        jnl     .L17
.L18:
        st.w    r0, [r13]
        add     4, r13
        cmp     r12, r13
        jl      .L18
.L17:
#
```

```
        ld.w    $__argc, r6                         -- set argc
        movea   $__argv, gp, r7                     -- set argv
        jarl    _main, lp                           -- call main function
__exit:
        halt                                        -- end of program
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
__startend:
#                                                                #
#-------------------- end of start up module -------------------#
#
```

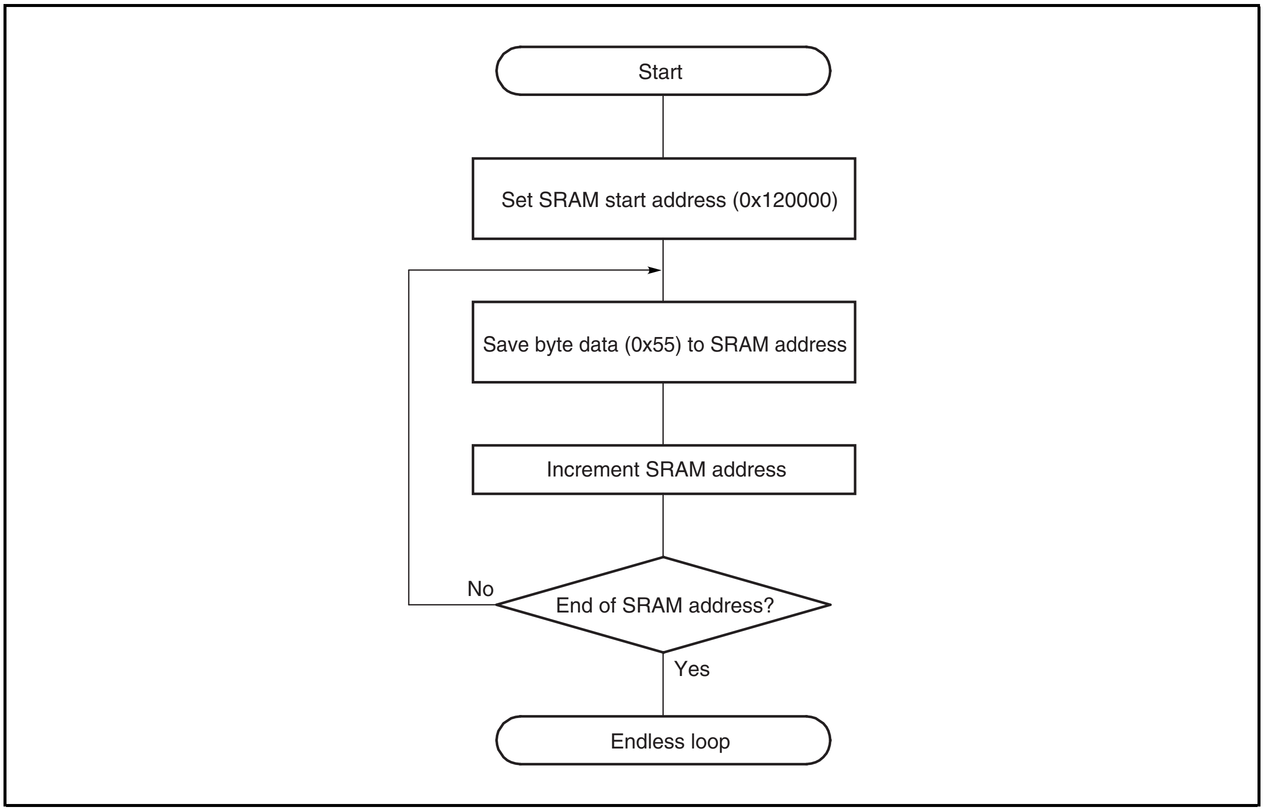
(3) C-language reset processing

```
        .option reg_mode 5 5
        .extern __start

.text
        .section        "RESET"
        di
        jr      __start
```

4.4.1 External memory fill program

Fixed data (0x55) is written in byte units for the external SRAM area (0x120000 to 0x13FFFF).

(1) Flow chart



(2) Assembly language program

```
.globl  start

.text
.set    sram_addr,        0x120000              -- SRAM address
.set    sram_addr_end,    0x140000              -- End of SRAM address

        .align 4
start:
        movea   sram_addr & 0x00ffff, r0, r10    -- SRAM address
        movhi   sram_addr >> 16, r10, r10        -- SRAM address
        movea   0x55,            r0, r11         -- 55 := fill data
write_loop:
        st.b    r11,             0[r10]          -- Write data
        add     1,               r10             -- Increment write address
        cmp     sram_addr_end,   r10             -- End of SRAM address?
        bnz     write_loop
fv_loop:
        br      fv_loop
```

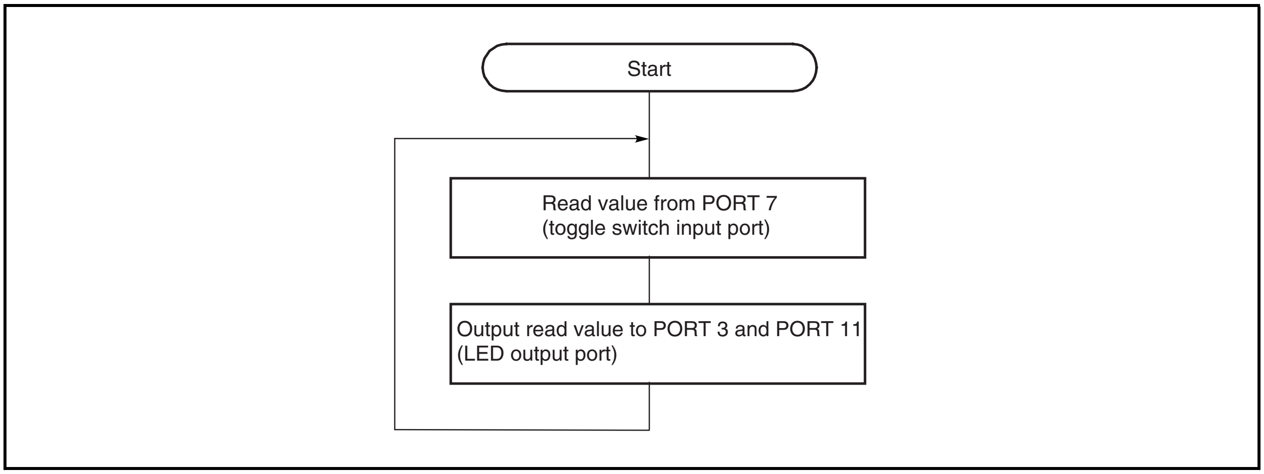
(3) C language program

```
#pragma ioreg                                   /* Enable use of peripheral I/O register name */

/* -----------------------------------------------------------
 *   MAIN processing
 *-----------------------------------------------------------
 */
void main(void)
{
   char *p ;                                    /* SRAM address pointer */

   p = (char *)0x120000 ;                       /* SRAM address start pointer */

   for ( ; p < (char *)0x140000 ; p++ ){        /* Loop until end of SRAM address */
      *p = 0x55 ;                               /* Write to memory at 55 */
   }
   /* Endless loop */
   for ( ; ; ) {
   }
}
```

4.4.2 Switch input and LED lamp activation

This program reads the status of Port 7 (toggle switch) and writes the read data to ports 3 and 11 (LED).

(1) Flow chart



(2) Assembly language program

```
.globl  start

.text
        .align 4
start:
        ld.b    P7[r0],   r10             -- Read switch input port
        mov     r10,      r11             -- r11 := r10
        and     0x0f,     r11             -- Output lower 4 bits of
        shl     4,        r11             -- port 7 to higher 4 bits of port 3
        st.b    r11,      P3[r0]          -- LED ON
        and     0xf0,     r10             -- Output higher 4 bits of
        shr     4,        r10             -- port 7 to lower 4 bits of port 11
        st.b    r10,      P11[r0]         -- LED ON
        br      start
```

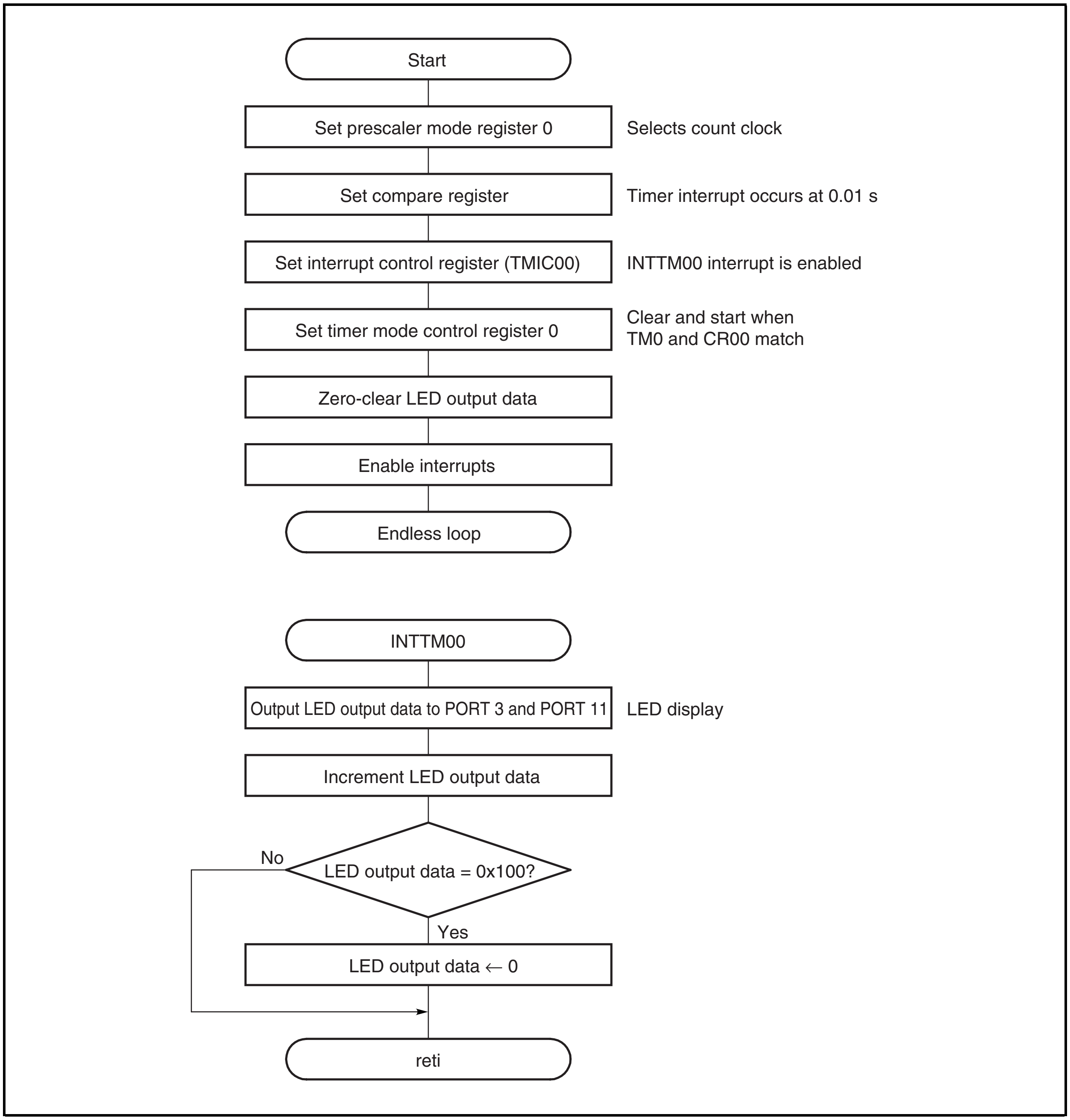
(3) C language program

```
#pragma ioreg                            /* Enable use of peripheral I/O register name */

/* ---------------------------------------------------------
 *   MAIN processing
 *---------------------------------------------------------
 */
void main(void)
{
   char isw ;                            /* Switch input port */
   char p3out ;                          /* Port 3's output value */
   char p11out ;                         /* Port 11's output value */

   /* Endless loop */
   for ( ; ; ) {
      isw = P7 ;                         /* Read switch input port */
      /* Lower 4 bits of ISW are shifted 4 bits to left and output to port 3 */
      P3 = ( isw & 0x0f ) << 4 ;
      /* Higher 4 bits of ISW are shifted 4 bits to right and output to port 11 */
      P11 = ( isw & 0xf0 ) >> 4 ;
   }
}
```

4.4.3 Activation of LED lamp by timer interrupt

The LED lamp is incremented and activated (lit) when a timer interrupt occurs. The timer interrupt interval is set as 10 ms.

(1) Flow chart



(2) Assembly language program

```
.globl  start

## Set interrupt handler
.section "INTTM00"                          -- INTTM00 interrupt handler
        jr      _int_tm00                   -- Go to interrupt processing

.text
        .align 4
start:
        mov     0x01,     r10
        ## Count clock = fxx/16 (1 MHz: fxx = 16 MHz)
        st.b    r10,      PRM0[r0]          -- Set prescaler mode register 0
        movea   0x2710,   r0, r10           -- 1 MHz*0x2710 = 0.01 sec
        st.h    r10,      CR00[r0]          -- Set compare register
        mov     0x07,     r10
        st.b    r10,      TMIC00[r0]        -- INTTM00 interrupt is enabled
        mov     0x0c,     r10
        ## Clear and start when TM0 and CR00 match
        st.b    r10,      TMC0[r0]          -- Set timer mode control register 0
        mov     r0,       r10               -- Initialize display data
        ei
fv_loop:
        br      fv_loop                     -- Endless loop

_int_tm00:
        mov     r10,      r11               -- r11 = r10
        mov     r10,      r12               -- r12 = r10
        and     0x0f,     r11               -- Output lower 4 bits of
        shl     4,        r11               -- display data to higher 4 bits of port 3
        st.b    r11,      P3[r0]            -- LED ON
        and     0xf0,     r12               -- Output higher 4 bits of
        shr     4,        r12               -- display data to lower 4 bits of port 11
        st.b    r12,      P11[r0]           -- LED ON
        add     1,        r10               -- Increment r10 (once)
        cmp     0x100,    r10               -- After r10 reaches 0x100,
        bnz     _int_tm00_not0              --
        mov     r0,       r10               -- return 0 to r10
_int_tm00_not0:
        reti                                -- Recovery from interrupt routine
```

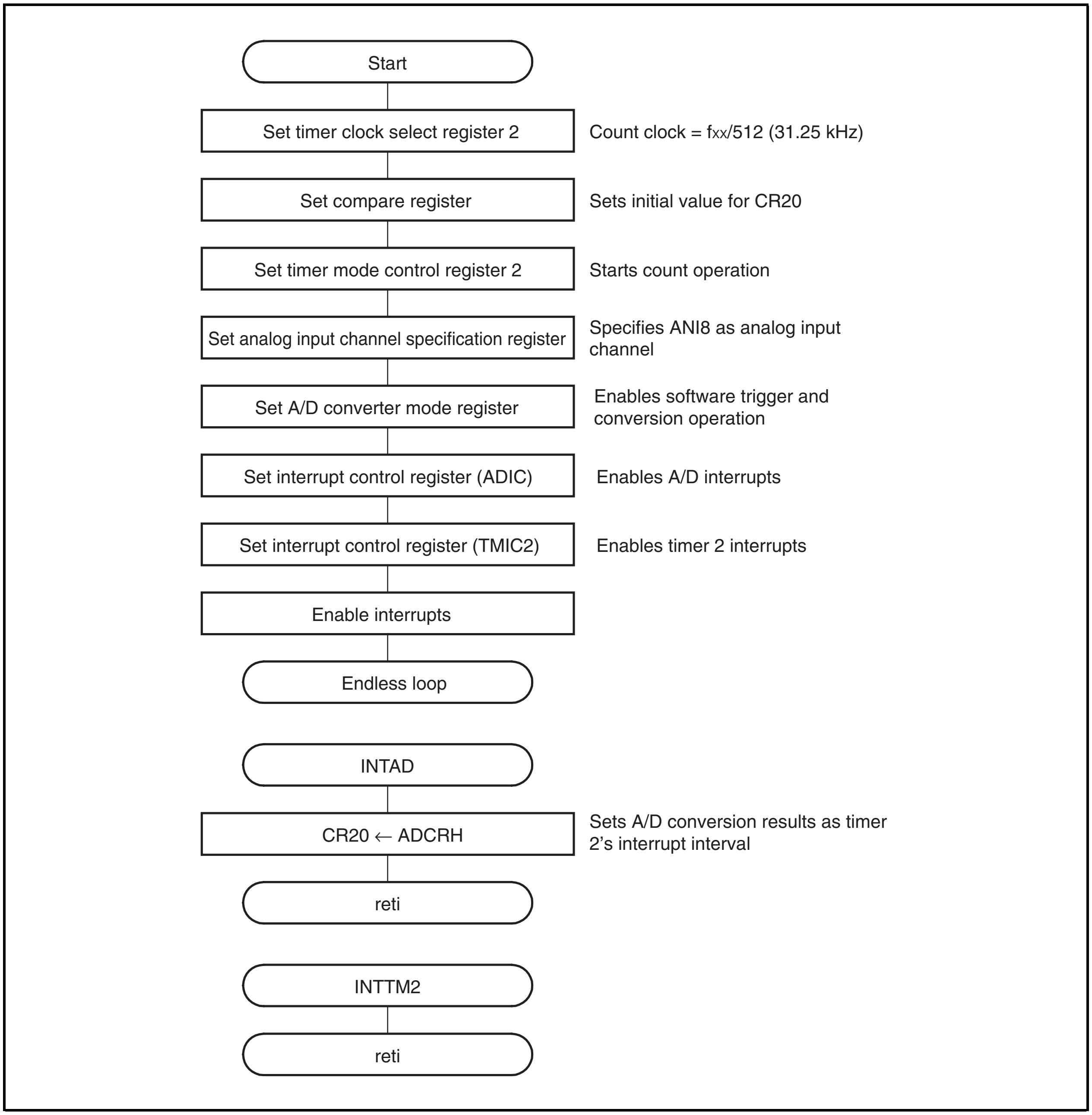
(3) C language program

```
#pragma ioreg                                       /* Enable use of peripheral I/O register name */

/* ---------------------------------------------------------
 *   Global variable
 *---------------------------------------------------------
 */
int dsp_data ;                                      /* LED display data */

/* ---------------------------------------------------------
 *   MAIN processing
 *---------------------------------------------------------
 */
void main(void)
{
    PRM0 = 0x01 ;                                   /* Count clock := fxx/16 (1 MHz: fxx = 16 MHz) */
    /* Compare register settings */
    CR00 = 0x2710 ;                                 /* 1 MHz*0x2710 = 0.01 sec */
    TMIC00 = 0x07 ;                                 /* Enables INTM00 interrupt */
    /* Timer mode control register settings */
    TMC0 = 0x0c ;                                   /* Clear and start when TM0 and CR00 match */
    dsp_data = 0 ;                                  /* Initialize display data */
    __EI();                                         /* Enable interrupts */
    /* Endless loop */
    for ( ; ; ) {
    }
}

/* ---------------------------------------------------------
 *   Timer 4 interrupt function
 *---------------------------------------------------------
 */
#pragma interrupt INTTM00 _int_tm00
void _int_tm00(void)
{
    /* Lower 4 bits of data are shifted 4 bits to left and output to port 3 */
    P3 = ( dsp_data & 0x0f ) << 4 ;
    /* Higher 4 bits of data are shifted 4 bits to right and output to port 11 */
    P11 = ( dsp_data & 0xf0 ) >> 4 ;

    /* Increment if dsp_data reaches 0xfe or below, otherwise return to zero */
    if ( dsp_data < 0xfe ) {
        dsp_data++ ;
    }else{
        dsp_data = 0 ;
    }
}
```

4.4.4 Analog input sampling and DC motor speed control

The A/D function reads variable resistance values. The read data is set to timer 2's compare register. Timer 2 is operated as a PWM output, so it is used to control the motor's speed.

(1) Flow chart



(2) Assembly language program

```
.globl  start

## Set interrupt handler
.section "INTAD"                                    -- INTAD interrupt handler
        jr      _int_ad                             -- Go to interrupt processing
.section "INTTM2"                                   -- INTTM2 interrupt handler
        jr      _int_tm2                            -- Go to interrupt processing

.text
        .align 4
start:
        mov     0x07,           r10
        ## Count clock = fxx/512 (31.25 kHz)
        st.b    r10,            TCL2[r0]            -- Sets timer clock select register 2
        movea   0xff,           r0, r10
        st.b    r10,            CR20[r0]            -- Sets compare register (initial value)
        movea   0xC3,           r0, r10
        ## Start count operation, enable timer output, PWM mode
        st.b    r10,            TMC2[r0]            -- Sets timer mode control register 2
        mov     0x08,           r10
        st.b    r10,            ADS[r0]             -- Specifies ANI8 as analog input channel
        movea   0x88,           r0, r10
        st.b    r10,            ADM[r0]             -- Enables software trigger and conversion operation
        mov     0x07,           r10
        st.b    r10,            ADIC[r0]            -- Enables A/D interrupts
        mov     0x07,           r10
        st.b    r10,            TMIC2[r0]           -- Enables timer 2 interrupts
        ei
fv_loop:
        br      fv_loop                             -- Endless loop

_int_ad:                                            -- Interrupt processing upon completion of A/D conversion
        ld.b    ADCRH[r0],      r10                 -- Reads analog input value
        st.b    r10,            CR20[r0]            -- Sets timer 2's interrupt interval
        reti                                        -- Recovers from interrupt routine

_int_tm2:                                           -- Timer 2 interrupt processing
        reti                                        -- Recovers from interrupt routine
```

(3) C language program

```
#pragma ioreg                          /* Enable use of peripheral I/O register name */

/* ---------------------------------------------------------
 *   MAIN processing
 *----------------------------------------------------------
 */
void main(void)
{
    /* Set timer clock select register */
    TCL2 = 0x07 ;                      /* Count clock = fxx/512 (31.25 kHz) */
    CR20 = 0xff ;                      /* Sets compare register (initial value) */
    /* Sets timer clock control register */
    TMC2 = 0xc3 ;                      /* Start count operation, enable timer output, PWM mode */
    ADS  = 0x08 ;                      /* Specifies ANI8 as analog input channel */
    ADM  = 0x88 ;                      /* Enables software trigger and conversion operation */
    ADIC = 0x07 ;                      /* Enables A/D interrupts */
    TMIC2 = 0x07 ;                     /* Enables timer 2 interrupts */
    __EI();                            /* Enables interrupts */
    /* Endless loop */
    for ( ; ; ) {
    }
}

/* ---------------------------------------------------------
 *   Interrupt processing upon completion of A/D conversion
 *----------------------------------------------------------
 */
#pragma interrupt INTAD _int_ad
void _int_ad(void)
{
    /* Reads analog input value and sets it as timer 2's interrupt interval */
    CR20 = ADCRH ;
}
/* ---------------------------------------------------------
 *   Timer 2 interrupt processing
 *----------------------------------------------------------
 */
#pragma interrupt INTTM2 _int_tm2
void _int_tm2(void)
{
}
```

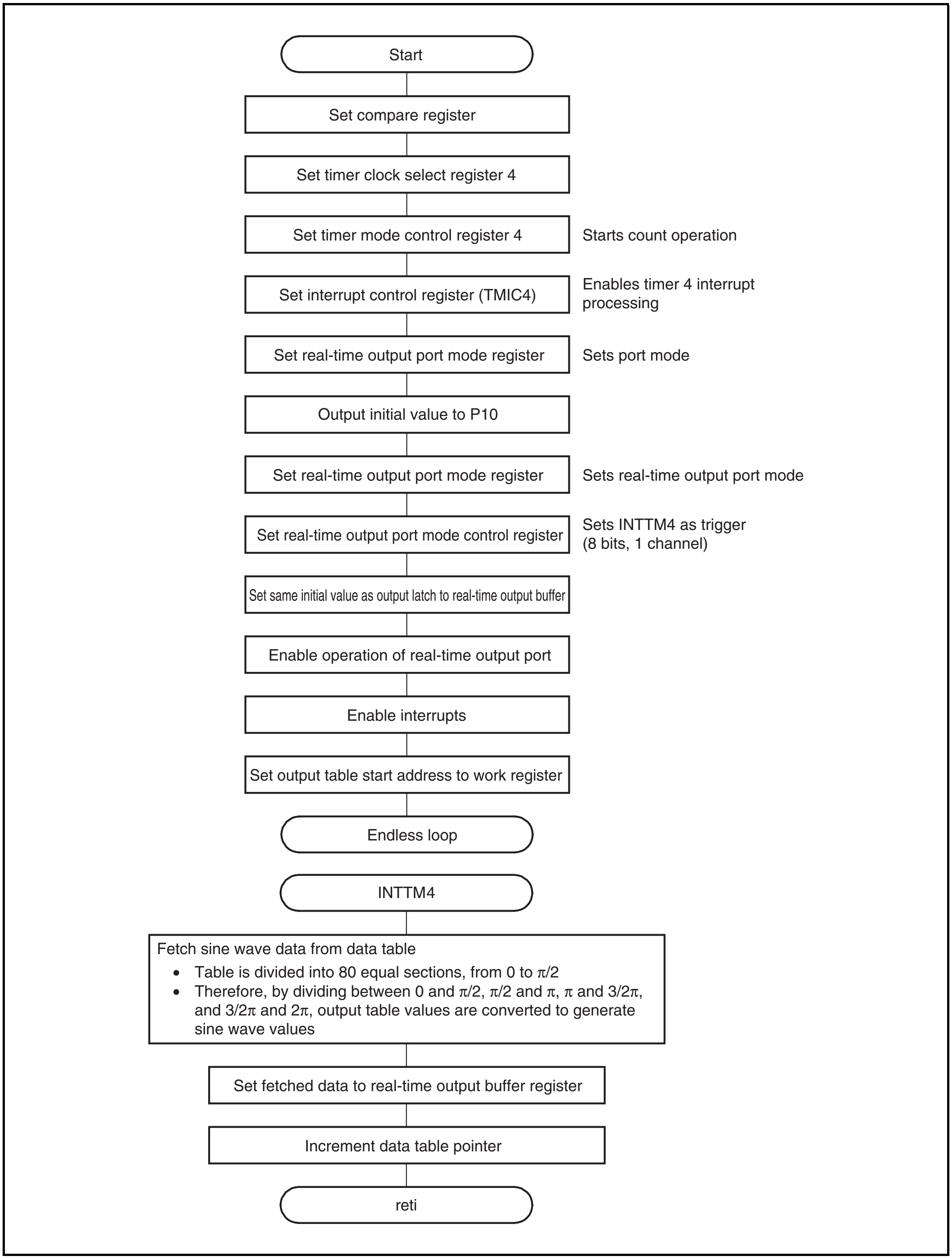
4.4.5 Sine wave output with D/A using RTP

INTTM4 is specified as the real-time output port's output trigger.

Sine wave data is provided as preset data in an assembly language data file. In C language, the sinf() function is used to create a table.

The sine wave's frequency is set as 520 Hz in assembly language and as 200 Hz in C language. The D/A function produces analog values from the data output via the real-time output port, and these analog values are then output as sound via the speaker.

(1) Flow chart (assembly language)

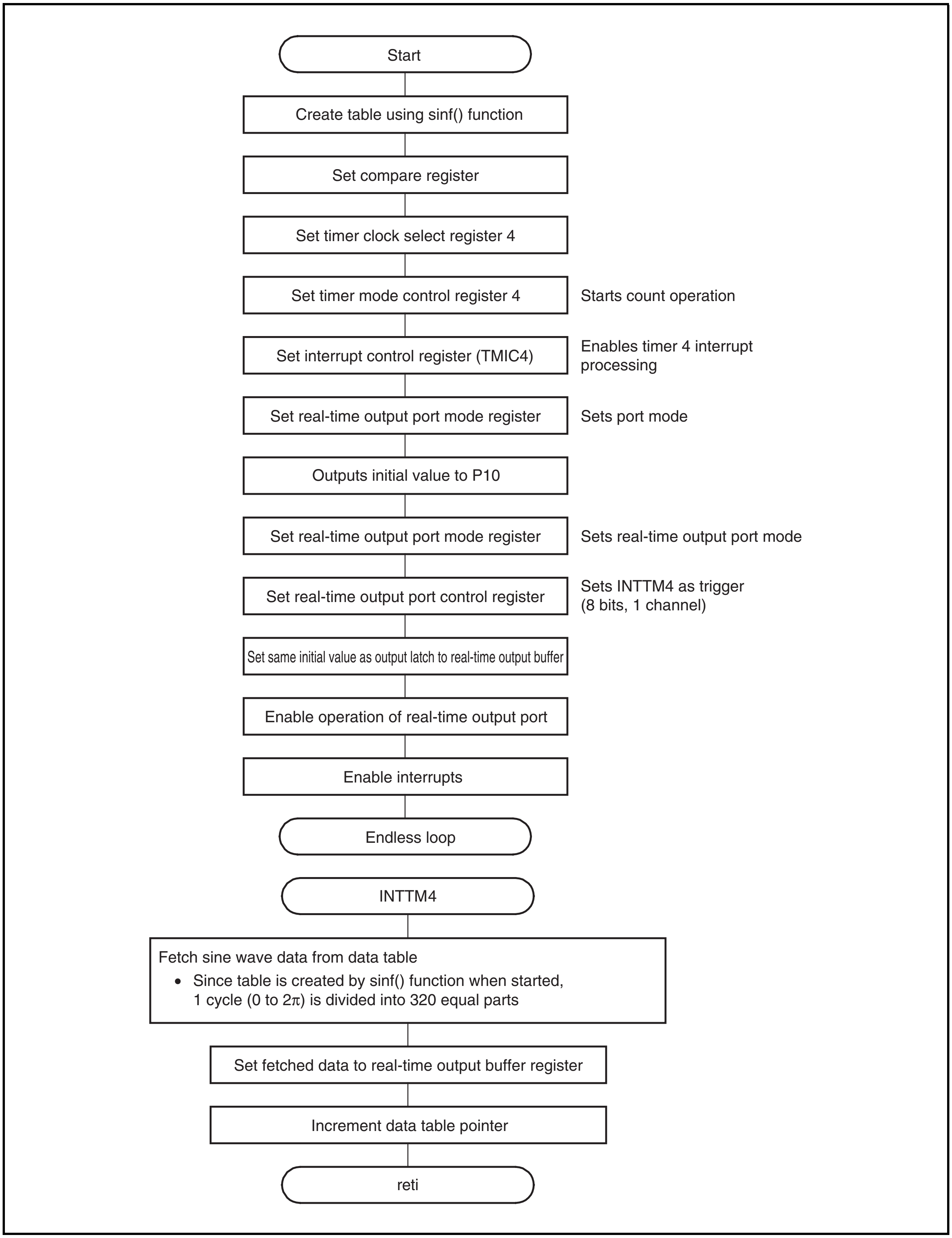


(2) Assembly language program

```
#  <Registers used>
#    r6                         : Register used to save r1
#    r10                        : Used for main processing operations
#    r11                        : Real-time output value
#    r12                        : _sin_tb address
#    r13                        : Δx (Δx range: 0 to 319)
#    r21,r22,r23                : Used for timer interrupt processing operations

.globl  start

## Set interrupt handler
.section "INTTM4"                               -- INTTM4 interrupt handler
        jr      _int_tm4                        -- Go to interrupt processing

.text
        .align 4
start:
        ## Set timer 4
        ## 4 MHz/(24*320) = 520 Hz sound output
        movea   24,         r0, r10
        st.b    r10,        CR40[r0]            -- Sets compare register
        mov     2,          r10
        ## fxx/4 (select 4 MHz)
        st.b    r10,        TCL4[r0]            -- Sets timer clock select register 4
        movea   0x80,       r0, r10
        st.b    r10,        TMC4[r0]            -- Starts count operation
        mov     0x07,       r10
        st.b    r10,        TMIC4[r0]           -- Enables timer 4 interrupt processing
        ## Sets real-time output
        st.b    r0,         RTPM[r0]            -- Sets port mode
        st.b    r0,         P10[r0]             -- Sets initial value to output latch
        movea   0xff,       r0, r10
        st.b    r10,        RTPM[r0]            -- Sets real-time output port mode
        movea   0x20,       r0, r10
        st.b    r10,        RTPC[r0]            -- Triggers INTTM4
                                                -- 8 bits, 1 channel
        st.b    r0,         RTBL[r0]            -- Sets same initial value as
                                                -- output latch to real-time output buffer
        set1    7,          RTPC[r0]            -- Enable operation of real-time output port
        ei                                      -- Enable interrupts
init_reg:
        movhi   hi1(#_sin_tb), r0, r12          -- r12 := _sin_tb address
        movea   lo(#_sin_tb), r12, r12
        mov     r0,         r13

fv_loop:
        br      fv_loop                         -- Endless loop

_int_tm4:
        mov     r1,          r6                 -- Saves r1
tm4_chk_dx:
        movea   0x50,  r0,  r21                 -- r21 := 0x50 (80)
        cmp     r21,        r13                 -- If r13 >= 80
        bp      tm4_chk_u160                    -- Jump
tm4_u80:
        mov     r12,        r21                 -- r21 := _sin_tb address
        add     r13,        r21                 -- r21 := _sin_tb address + Δx
        ld.b    0[r21],     r11                 -- r11 := real-time output value
        br      tm4_out
tm4_chk_u160:
        movea   0xA0,  r0,  r21                 -- r21 := 0xA0 (160)
        cmp     r21,        r13                 -- If r13 >= 160
        bp      tm4_chk_u240                    -- Jump
```

```
tm4_u160:
        mov     r12,        r21              -- r21 := _sin_tb address
        movea   0x9F,  r0,  r22              -- r22 := 0x9F (159)
        add     r22,        r21              -- r21 = r21 + 159
        sub     r13,        r21              -- r21 = r21 – Δx
        ld.b    0[r21],     r11              -- r11 := real-time output value
        br      tm4_out
tm4_chk_u240:
        movea   0xF0,  r0,  r21              -- r21 := 0xF0 (240)
        cmp     r21,        r13              -- If r13 >= 240
        bp      tm4_u320                     -- Jump
tm4_u240:
        mov     r12,        r21              -- r21 := _sin_tb address
        add     r13,        r21              -- r21 = r21 + Δx
        movea   0xA0,  r0,  r22              -- r22 := 0xA0 (160)
        sub     r22,        r21              -- r21 = r21 – 160
        ld.b    0[r21],     r22              -- Load from memory
        and     0xff,       r22
        movea   0xFF,  r0,  r11              -- Real-time output value
        sub     r22,        r11              -- Decrement table value from 0x80
        br      tm4_out
tm4_u320:
        mov     r12,        r21              -- r21 := _sin_tb address
        movea   0x13F, r0,  r22              -- r22 = 0x13F (320)
        add     r22,        r21              -- r21 = r21 + 319
        sub     r13,        r21              -- r21 = r21 – Δx
        ld.b    0[r21],     r22              -- Load from memory
        and     0xff,       r22
        movea   0xFF,  r0,  r11              -- Real-time output value
        sub     r22,        r11              -- Decrement table value from 0x80
        br      tm4_out
tm4_out:
        st.b    r11,        RTBH[r0]         -- Set output value to real-time output buffer register
tm4_inc_dx:
        add     1,          r13              -- Increment Δx
        movea   0x140, r0,  r21              -- r21 := 0x140 (320)
        cmp     r13,        r21              -- If Δx doesn't reach 320
        bnz     tm4_exit                     -- Jump
tm4_dx_eq0:
        mov     r0,         r13              -- Return Δx to zero
tm4_exit:
        mov     r6,         r1               -- Recovery of r1
        reti

_sin_tb:                                     -- Sine wave table
        ## sin(x)*0x80  x = 0 to π/2 [ step=(π/2)/80 ]
       .byte   128,130,133,135,138,140,143,145
       .byte   148,150,152,155,157,160,162,165
       .byte   167,169,172,174,176,179,181,183
       .byte   186,188,190,192,194,197,199,201
       .byte   203,205,207,209,211,213,214,216
       .byte   218,220,221,223,225,226,228,230
       .byte   231,233,234,235,237,238,239,240
       .byte   242,243,244,245,246,247,248,248
       .byte   249,250,251,251,252,253,253,254
       .byte   254,254,255,255,255,255,255,255
```

(3) Flow chart (C language)



(4) C language program

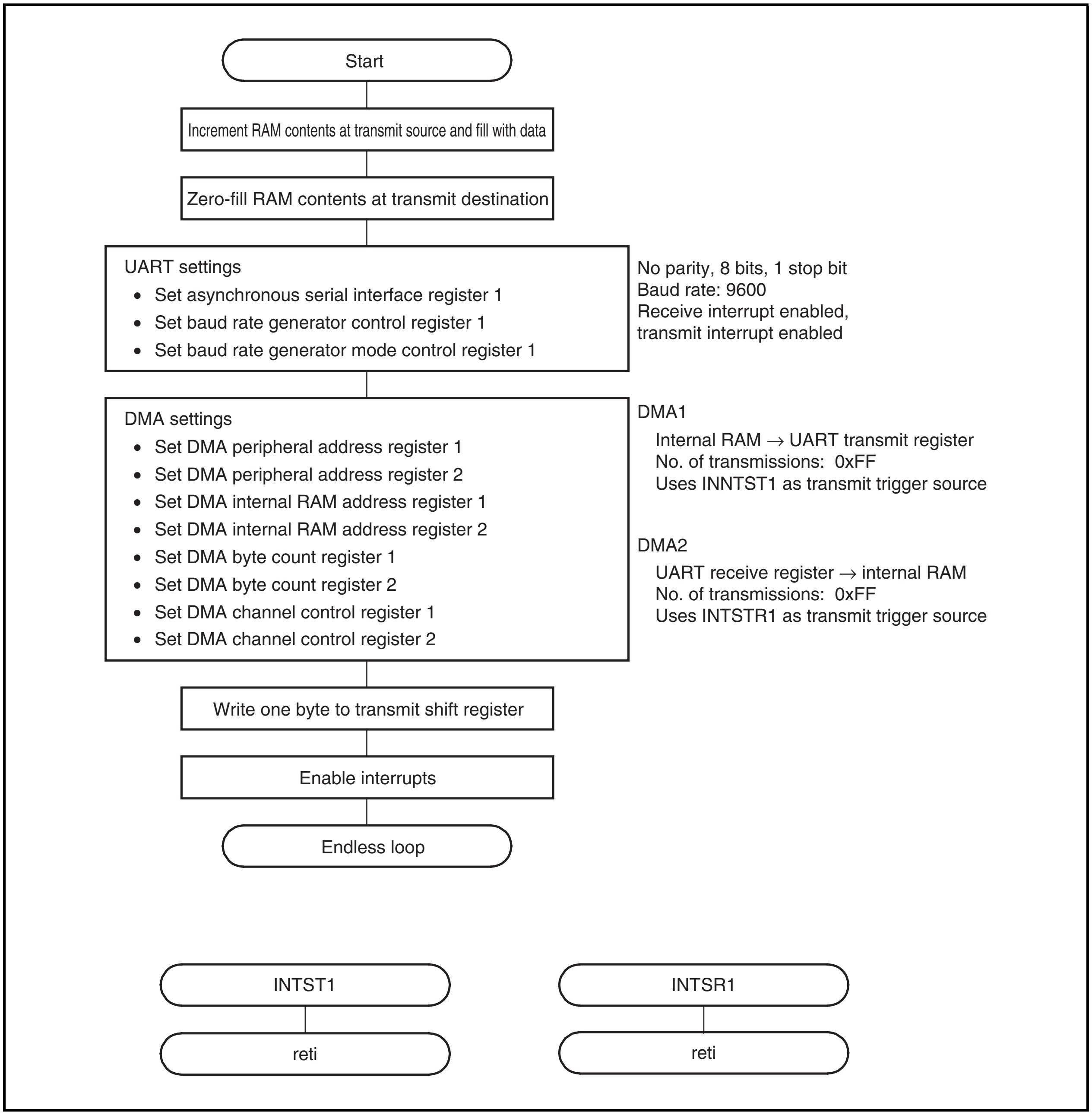
```
#include <math.h>
#include <float.h>
#pragma ioreg                                   /* Enable use of peripheral I/O register name */

int st  ;                                       /* Step interval */
int sin_buf[320] ;                              /* Output value array */

/* ------------------------------------------------------------
 *  Global variable initialization processing
 *------------------------------------------------------------
 */
void sta_init()
{
    double pai ;                                /* π       */
    int data ;
    int st0  ;

    /* Initialize st variable */
    st = 0 ;

    /* Initialize sing_buf[] variable */
    pai = 3.14159 ;
    for ( st0 = 0 ; st0 < 320 ; st0++ ) {
        /* Create table for 320 units of sine wave data */
        data = ( int ) ( sinf( ( pai * 2 * st0 ) / 320 )  * 0x80 ) ;
        data += 0x80;
        if ( data == 0x80 *2 ) {
            data = 0xff ;
        }
        sin_buf[st0] = data ;                   /* Set data to table */
    }
}


/* ------------------------------------------------------------
 *  MAIN processing
 *------------------------------------------------------------
 */
void main(void)
{
    sta_init();                                 /* Initializes static variable */
    /* Set timer 4
        4 MHz/(62*320) = 200 Hz sound output */
    CR40 = 62 ;                                 /* Sets compare register */
    /* fxx/4 (select 4 MHz) */
    TCL4 = 2 ;                                  /* Sets timer clock select register 4 */
    TMC4 = 0x80 ;                               /* Starts count operation */
    TMIC4 = 0x07 ;                              /* Enables timer 4 interrupt processing */
    /* Sets real-time output */
    RTPM = 0 ;                                  /* Sets port mode */
    P10 = 0 ;                                   /* Sets initial value to output latch */
    RTPM = 0xff ;                               /* Sets real-time output port mode */
    RTPC = 0x20 ;                               /* Triggers INTTM4 */
                                                /* 8 bits, 1 channel */
    RTBL = 0 ;                                  /* Sets same initial value as output latch to */
                                                /* real-time output buffer register */
    RTPC.7 = 1 ;                                /* Enable operation of real-time output port */
    __EI() ;                                    /* Enable interrupts */

    /* Endless loop */
    for ( ; ; ) {
    }
}
```

```
/* --------------------------------------------------------
 *   Timer 4 interrupt function
 *--------------------------------------------------------
 */
#pragma interrupt INTTM4 _int_tm4
void _int_tm4(void)
{
   RTBH = sin_buf[st] ;                  /* Set output value to real-time output buffer register */
   st++ ;                                /* Increment step        */
   if ( st == 320 ) {                    /* If step reaches 320 */
   st = 0 ;                              /* Return to zero         */
   }
}
```

4.4.6 DMA transfer (between UART and internal RAM), program 1

Two channels are used for DMA transfers. One channel is transferred from the internal RAM to the UART transmit register as the transfer trigger source for INST1 and the other channel is transferred from the UART receive register to the internal RAM as the transfer trigger source for INTSR1. The UART is operated with a loopback setting that enables the transmit data to be received without modification.

(1) Flow chart



(2) Assembly language program

```
.globl  start

## Set interrupt handler
.section "INTST1"                               -- INST1 interrupt handler
        jr      _int_st1                        -- Go to interrupt processing
.section "INTSR1"                               -- INTSTR1 interrupt handler
        jr      _int_sr1                        -- Go to interrupt processing

.text
        .align 4
start:
        ## Increments RAM contents at transmit source and fills with data
        movea   0xe000,     r0, r11             -- 0xffffe000 := address where UART transmit data is stored
        mov     0,          r12                 -- Initial value of transmit data
ram_inc:
        st.b    r12,        0[r11]              -- Stores transmit data
        add     1,          r11                 -- Increments address where transmit data is stored
        add     1,          r12                 -- Increments transmit data
        cmp     0x100,      r12
        bnz     ram_inc                         -- Loops until r12 reaches 0x100
        ## Zero-fills RAM contents at transmit destination
        movea   0xe100,     r0, r11             -- 0xffffe100 := address where UART receive data is stored
        mov     r0,         r12                 -- r12 = loop counter
ram_0:
        st.b    r0,         0[r11]              -- Sets "0" to receive destination address
        add     1,          r11                 -- Increments address where receive data is stored
        add     1,          r12                 -- Increments loop counter
        cmp     0x100,      r12
        bnz     ram_0                           -- Loops until r12 reach 0x100

        ## UART settings
        movea   0xc8,       r0, r10
        st.b    r10,        ASIM1[r0]           -- No parity, 8 bits, one stop bit
        movea   0xd0,       r0, r10
        st.b    r10,        BRGC1[r0]           -- Baud rate := 9600
        mov     0x03,       r10
        st.b    r10,        BRGMC1[r0]          -- Baud rate := 9600
        mov     0x07,       r10
        st.b    r10,        SRIC1               -- Enables receive interrupt
        st.b    r10,        STIC1               -- Enables transmit interrupt
        ## DMA settings
        movea   0x0316,     r0, r10
        st.h    r10,        DIOA1[r0]           -- Address of transmit register
        movea   0x0318,     r0, r10
        st.h    r10,        DIOA2[r0]           -- Address of receive register
        mov     0x0000,     r10
        st.h    r10,        DRA1[r0]            -- Address of internal RAM (transmit source)
        movea   0x0100,     r0, r10
        st.h    r10,        DRA2[r0]            -- Address of internal RAM (transmit destination)
        movea   0xff,       r0, r10
        st.b    r10,        DBC1[r0]            -- Sets DMA byte count register 1
        movea   0xff,       r0, r10
        st.b    r10,        DBC2[r0]            -- Sets DMA byte count register 2
        mov     0x09,       r10
        st.b    r10,        DCHC1[r0]           -- DMA1's transmit trigger source := INTST1, transfer enabled
        mov     0x05,       r10
        st.b    r10,        DCHC2[r0]           -- DMA2's transmit trigger source := INTSR1, transfer enabled
        ## Write one byte to transmit shift register
        st.b    r0,         TXS1[r0]
        ei
fv_loop:
        br      fv_loop                         -- Endless loop

_int_st1:                                       -- Interrupt processing upon completion of transmission
        reti                                    -- Recovers from interrupt routine
_int_sr1:                                       -- Interrupt processing upon completion of reception
        reti                                    -- Recovers from interrupt routine
```

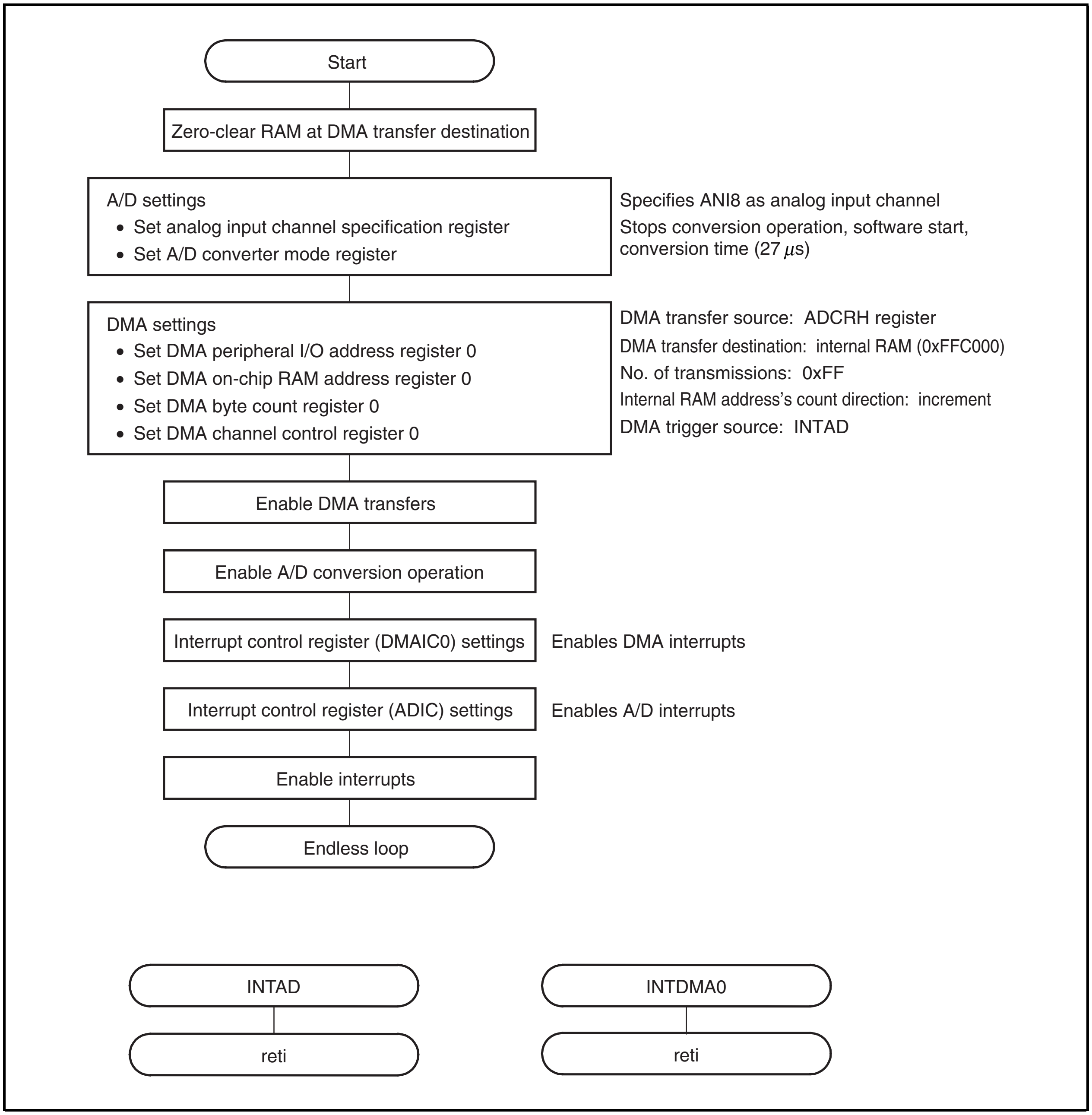
(3) C language program

```
#pragma ioreg                                    /* Enables use of peripheral I/O register name */

/* ---------------------------------------------------------
 *   MAIN processing
 *---------------------------------------------------------
 */
void main(void)
{
   char *p ;
   int   i ;

   /* Increments RAM contents at transmit source and fills with data, p = address where UART transmit data is stored */
   for ( p = (char *)0xffffe000, i = 0  ; p < (char *)0xffffe100 ; p++, i++ ) {
      *p = i ;                                   /* Stores transmit data */
   }

   /* Zero-fills RAM contents at transmit destination, p = address where UART receive data is stored */
   for ( p = (char *)0xffffe100 ; p < (char *)0xffffe200 ; p++ ) {
      *p = 0 ;                                   /* Sets "0" to receive destination address */
   }

   /* UART settings */
   ASIM1  = 0xc8 ;                               /* No parity, 8 bits, one stop bit */
   BRGC1  = 0xd0 ;                               /* Baud rate := 9600 */
   BRGMC1 = 0x03 ;                               /* Baud rate := 9600 */
   SRIC1  = 0x07 ;                               /* Enables receive interrupt */
   STIC1  = 0x07 ;                               /* Enables transmit interrupt */

   /* DMA settings */
   DIOA1 = 0x0316 ;                              /* Address of transmit register */
   DIOA2 = 0x0318 ;                              /* Address of receive register */
   DRA1  = 0x0000 ;                              /* Address of internal RAM (transmit source) */
   DRA2  = 0x0100 ;                              /* Address of internal RAM (transmit destination) */
   DBC1  = 0xff ;                                /* Sets DMA byte count register 1 */
   DBC2  = 0xff ;                                /* Sets DMA byte count register 2*/
   DCHC1 = 0x09 ;                                /* DMA1's transmit trigger source := INTST1, */
                                                 /* transfer enabled */
   DCHC2 = 0x05 ;                                /* DMA2's transmit trigger source := INTSR1, */
                                                 /* transfer enabled */

   /* Write one byte to transmit shift register */
   TXS1 = 0 ;

   __EI();                                       /* Enables interrupts */
   /* Endless loop */
   for ( ; ; ) {
   }
}

/* ---------------------------------------------------------
 *   Interrupt processing upon completion of transmission
 *---------------------------------------------------------
 */
#pragma interrupt INTST1 _int_st1
void _int_st1(void)
{
}
/* ---------------------------------------------------------
 *   Interrupt processing upon completion of reception
 *---------------------------------------------------------
 */
#pragma interrupt INTSR1 _int_sr1
void _int_sr1(void)
{
}
```

4.4.7 DMA transfer (A/D → internal RAM), program 2

This program stores the variable resistance value to internal RAM using INTAD as the DMA transfer trigger source.

(1) Flow chart



(2) Assembly language program

```
.globl  start
.set    ram_addr,      0xffffe000            -- DMA transfer destination RAM address
.set    ram_addr_end,  0xffffe100            -- End of DMA transfer source RAM address

## Set interrupt handler
.section "INTAD"                              -- INTAD interrupt handler
        jr      _int_ad                       -- Go to interrupt processing
.section "INTDMA0"                            -- INTDMA0 interrupt handler
        jr      _int_dma0                     -- Go to interrupt processing

.text
        .align 4
start:
        ## Zero-clear DMA transfer destination RAM
        movea   ram_addr & 0x0000ffff, r0, r11 -- r11 := DMA transfer destination RAM address
ram_init_set:
        st.b    r0,             0[r11]        -- Sets "0" to DMA transfer destination RAM address
        add     1,              r11           -- Increments DMA transfer destination RAM address
        cmp     ram_addr_end,   r11
        bnz     ram_init_set                  -- Loop if not end of DMA transfer destination RAM address
        ## A/D settings
        mov     0x08,           r10
        st.b    r10,            ADS[r0]       -- Specifies ANI8 as analog input channel
        mov     0x01,           r10
        ## Stop conversion operation, software start, conversion time (27 us)
        st.b    r10,            ADM[r0]       -- A/D converter mode register settings
        ## DMA settings
        movea   0x03c6,         r0, r10
        ## Set DMA transfer source to ADCRH register
        st.h    r10,            DIOA0[r0]     -- Sets DMA peripheral I/O address register 0
        ## Sets DMA transfer destination to internal RAM (address: 0xFFE000)
        st.h    r0,             DRA0[r0]      -- DMA on-chip address register 0 settings
        movea   0xff,           r0, r10
        st.b    r10,            DBC0[r0]      -- DMA byte count register 0 settings
        movea   0x14,           r0, r10
        ## Count direction for internal RAM address:  increment, DMA trigger source:  INTAD,
        ## transfer  direction: peripheral I/O → internal RAM
        ## 8-bit transfers and DMA transfers prohibited
        st.b    r10,            DCHC0[r0]     -- DMA channel control register 0 settings
        set1    0,              DCHC0[r0]     -- Enables DMA transfers
        set1    7,              ADM[r0]       -- Enables conversion operation
        mov     0x07,           r10
        st.b    r10,            DMAIC0[r0]    -- Enables interrupt at end of DMA transfer
        st.b    r10,            ADIC[r0]      -- Enable interrupt at end of A/D conversion
        ei
fv_loop:
        br      fv_loop
_int_ad:                                      -- Interrupt processing at end of A/D conversion
        reti                                  -- Recover from interrupt processing routine
_int_dma0:                                    -- Interrupt processing at end of DMA transfer
        reti                                  -- Recover from interrupt processing routine
```

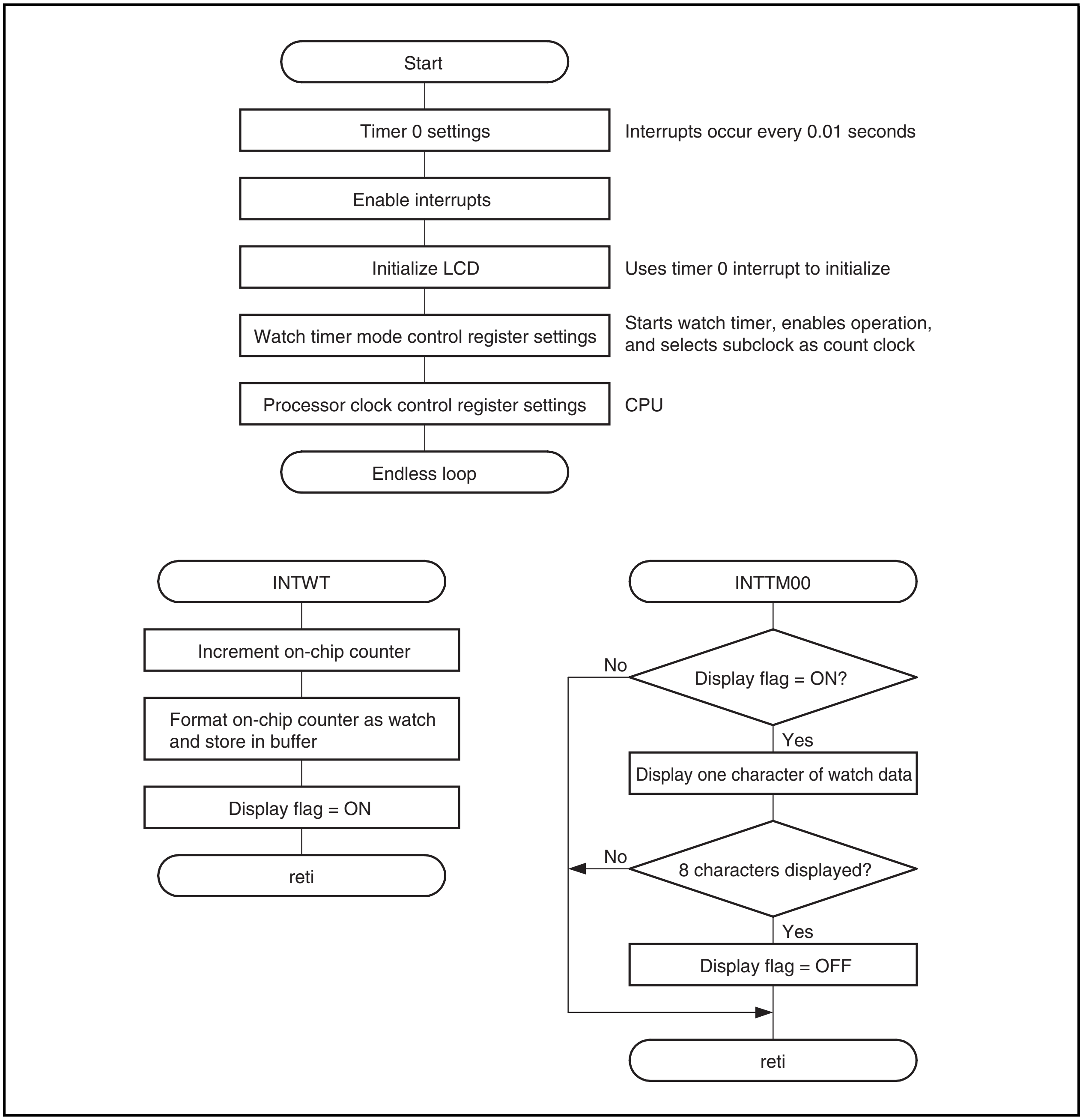
(3) C language program

```
#pragma ioreg                                   /* Enables use of peripheral I/O register name */

/* -------------------------------------------------------------
 *   MAIN processing
 *-------------------------------------------------------------
 */
void main(void)
{
    char *p ;
    /* Zero-clears RAM at DMA transfer destination, p = RAM address at DMA transfer destination */
    for ( p = (char *)0xffffe000 ; p < (char *)0xffffe100 ; p++ ) {
        *p = 0 ;
    }
    /* A/D settings */
    ADS = 0x08 ;                                /* Sets ANI8 as analog input channel */
    /* A/D converter mode register settings */
    ADM = 0x01 ;                                /* Stop conversion operation, software start, */
                                                /* conversion time (27 us) */

    /* DMA settings */
    /* Set address of DMA peripheral I/O register */
    DIOA0 = 0x03c6 ;                            /* Sets DMA transfer source to ADCRH register */
    /* Set DMA internal RAM address register */
    DRA0  = 0 ;                                 /* Sets DMA transfer destination to */
                                                /* Internal RAM (address: 0xFFE000) */
    DBC0  = 0xff ;                              /* DMA byte count register 0 settings */
    /* DMA channel control register settings */
    DCHC0 = 0x14 ;                              /* Count direction for internal RAM address: increment, */
                                                /* DMA trigger source: INTAD, */
                                                /* transfer direction: peripheral I/O → internal RAM */
                                                /* 8-bit transfer, DMA transfer prohibited */
    DCHC0.0 = 1 ;                               /* DMA transfer enabled */
    ADM.7   = 1 ;                               /* Conversion operation enabled */

    DMAIC0  = 0x07 ;                            /* Enables interrupt at end of DMA transfer */
    ADIC    = 0x07 ;                            /* Enables interrupt at end of A/D conversion */

    __EI();                                     /* Interrupts enabled */
    /* Endless loop */
    for ( ; ; ) {
    }
}

/* -------------------------------------------------------------
 *   Interrupt processing at end of A/D conversion
 *-------------------------------------------------------------
 */
#pragma interrupt INTAD _int_ad
void _int_ad(void)
{
}
/* -------------------------------------------------------------
 *   Interrupt processing at end of DMA0
 *-------------------------------------------------------------
 */
#pragma interrupt INTDMA0 _int_dma0
void _int_dma0(void)
{
}
```

4.4.8 Switching to subclock operation and watch display program

This program selects the subsystem clock as the CPU's clock. The subclock is also selected as the watch timer's count clock. The watch data is incremented when a watch timer interrupt occurs. In fact, incrementation is displayed on the LCD when a timer 0 interrupt occurs.

(1) Flow chart



(2) Assembly language program

```
# <Registers used>
#   r6                   : Register for saving r1 value
#   r11,r12,r13,r14,r15: Register for main processing
#   r16,r17,r18,r19,r20: Register for timer 0 interrupt processing
#   r21,r22,r23,r24,r25: Register for watch timer interrupt processing
#   r26,r27,r28,r29     : bf_chk routine work register

.globl  start

.set    tm_buf,     0x120000                        -- Watch display data (xx:xx:xx) (8 bytes)
.set    col_ctr,    0x120008                        -- No. of columns in watch display data (1 byte)
.set    flg_wt,     0x120009                        -- 1-second display flag (1 byte)
.set    wt_num,     0x120010                        -- Watch data (4 bytes)
.set    disp_flg,   0x120014                        -- Watch LCD display status flag (displays "1")
.set    tm0_num,    0x120018                        -- Timer 0 counter (4 bytes)
.set    mem_last,   0x12001d                        -- End address of work conversion area
.set    lcd_addr,   0x140000                        -- LCD address

## Set interrupt handler
.section "INTWT"                                    -- INTWT interrupt handler
        jr      _int_wt                             -- Go to interrupt processing
.section "INTTM00"                                  -- INTTM00 interrupt handler
        jr      _int_tm00                           -- Go to interrupt processing

.text
        .align 4
start:
mem_init:
        movhi   hi1(tm_buf), r0, r11                -- r11 := start address of working memory
        movea   lo(tm_buf), r11, r11
        movhi   hi1(mem_last), r0, r12              -- r12 := end address of working variable area
        movea   lo(mem_last), r12, r12
mem_init_lp:
        st.b    r0,           0[r11]                -- Sets 0 for working variable
        addi    1,    r11,    r11                   -- Increments address to zero-clear
        cmp     r11,          r12
        bnz     mem_init_lp                         -- Loop if not end address of address to zero-clear
ins_3a:
        movhi   hi1(tm_buf), r0, r11                -- r11 := start address of working memory
        movea   lo(tm_buf), r11, r11
        movea   0x3a,   r0,   r13                   -- For time data:
        st.b    r13,          2[r11]                -- ':' = Set 0x3A
        st.b    r13,          5[r11]                -- ':' = Set 0x3A

init_tm0:
        movea   0x01,   r0,   r11                   -- r11 := 1
        -- fxx/16 (fxx = 16 MHz) = Specify 1 MHz for count clock
        st.b    r11,          PRM0[r0]              -- Prescaler mode register 0 settings
        st.b    r0,           TOC0[r0]              -- Timer output control register 0 settings
        movea   0x64, r0,   r11                     -- r11:1 MHz*0x64 = 0.1 ms
        -- Set 0.1 ms as interrupt interval
        st.h    r11,          CR00[r0]
        -- Operate CR00 as compare register
        st.b    r0,           CRC0[r0]              -- Capture/compare control register 0 settings
        mov     0x0c,         r11
        -- Clear and start when TM0 and CR00 match
        st.b    r11,          TMC0[r0]              -- Timer mode control register 0 settings
        mov     0x07,         r11
        st.b    r11,          TMIC00[r0]            -- Enables timer interrupts (level 7)
        ei
init_lcd:
        movhi   hi1(lcd_addr), r0, r11              -- r11 := LCD address
        movea   lo(lcd_addr), r11, r11
        movhi   hi1(tm0_num), r0, r12               -- r12 := timer 0 count address
        movea   lo(tm0_num), r12, r12
        movea   0x30,   r0,   r13                   -- r13 := 0x30
```

```
        st.b    r13,          0[r11]          -- Function setting; 8 bits, 1 line  5*7
        st.w    r0,           0[r12]          -- Timer 0 counter = 0
        movea   0x29,   r0,   r13             -- r13 := 0x29 = 41
wait_1:
        ld.w    0[r12],       r14             -- r14 := timer 0 counter
        cmp     r14,          r13             -- Checks whether or not timer 0 counter has reached 41
        bp      wait_1                        -- Waits at least 4.1 ms
        st.b    r13,          0[r11]          -- Function setting; 8 bits, 1 line  5*7
        st.w    r0,           0[r12]          -- Timer 0 counter = 0
        movea   0x2,    r0,   r13             -- r13 := 0x2
wait_2:
        ld.w    0[r12],       r14             -- r14 := timer 0 counter
        cmp     r14,          r13             -- Checks whether or not timer 0 counter has reached 2
        bp      wait_2                        -- Waits at least 100 µs
        st.b    r13,          0[r11]          -- Function setting; 8 bits, 1 line  5*7
        jarl    bf_chk,       r29             -- LCD busy check
        movea   0x38,         r0, r13
        st.b    r13,          0[r11]          -- Function setting; 8 bits, 2 lines  5*7
        jarl    bf_chk,       r29             -- LCD busy check
        mov     0x01,         r13
        st.b    r13,          0[r11]          -- Clears display, moves cursor to home position
        jarl    bf_chk,       r29             -- LCD busy check
        mov     0x06,         r13
        st.b    r13,          0[r11]          -- Increments cursor, no display shift
        jarl    bf_chk,       r29             -- LCD busy check
        mov     0x0C,         r13
        st.b    r13,          0[r11]          -- Sets display ON and sets cursor OFF

init_wt:
        ## Watch timer mode control register settings
        movea   0x83,   r0,   r11
        st.b    r11,          WTM[r0]         -- Starts watch timer and enables operation
                                              -- Selects subclock as count clock
        movea   0x06,   r0,   r11             -- r10 := 0x06
        ## Sets watch interrupt level higher than
        ## the timer 0 interrupt level
        st.b    r11,          WTIC[r0]        -- Enables watch timer interrupts (level 6)
set_sub_clock:
        ## Since the PCC register is a special register, it is modified via a special sequence.
        movea   0x04,   r0,   r13
        stsr    5,            r11             -- Loads program status word
        mov     r11,          r12             -- r12 = r11
        or      0x80,         r11             -- PSW's NP bit = 1
        ldsr    r11,          5               -- Writes PSW in which NP bit = 1
        st.b    r0,           PRCMD[r0]       -- Writes to PRCMD
        ## Select subsystem clock as CPU's clock
        st.b    r13,          PCC[r0]         -- Sets to processor clock control register
        ldsr    r12,          5               -- Returns PSW to previous setting
        nop                                   -- Dummy instructions (5 instructions)
        nop
        nop
        nop
        nop
        fv_loop:
        br      fv_loop                       -- Endless loop

## LCD busy check routine
bf_chk:
        movhi   hi1(lcd_addr), r0, r27        -- r27 := LCD address
        movea   lo(lcd_addr), r27, r27
        movea   0x80,   r0,   r26             -- r26 := 0x80
bf_chk0:
        ld.b    0[r27],       r28
        andi    0x80,   r28,  r28
        cmp     r26,          r28
        bz      bf_chk0
        jmp     [r29]

_int_wt:
```

```
        mov     r1,             r6                      -- Saves r1
wt_chk:
        movhi   hi1(flg_wt), r0, r21                    -- r21 := 1-second display flag
        movea   lo(flg_wt), r21, r21
        not1    0,              0[r21]                  -- Inverts flg_wt
        tst1    0,              0[r21]
        bnz     wt_exit                                 -- Just skips if flg_wt value is "1"
wt_data_set:
        movhi   hi1(wt_num), r0, r21                    -- r21 := watch data
        movea   lo(wt_num), r21, r21
        ld.w    0[r21],         r22                     -- Loads wt_num to register
        addi    1,      r22,    r22                     -- Increments wt_num
        st.w    r22,            0[r21]                  -- Stores wt_num to memory
        ## Converts r22 to watch format and stores it in memory
        movhi   hi1(tm_buf), r0, r21                    -- r21 := watch display data
        movea   lo(tm_buf), r21, r21
        mov     r22,            r7                      -- r7 = r22
        movea   60*60*10, r0,   r6
        andi    0xffff,   r6,   r6                      -- r6 = 60*60*10
        jarl    ___div,  lp                             -- r6 = r7/(60*60*10)
        add     0x30,           r6                      -- r6 = 0x30 + r6
        st.b    r6,             0[r21]                  -- x*10 hours
        mov     r22,            r23                     -- r23 = r22
        movea   60*60,  r0,     r6                      -- r6 = 60*60
        divh    r6,             r23                     -- r23 = r23/(60*60)
        mov     r23,            r24                     -- r24 = r23
        mov     r10,            r7                      -- r7 = r10
        divh    r7,             r24                     -- r24 = r24/10
        mulh    10,             r24                     -- r24 = r24*10
        sub     r24,            r23                     -- r23 = r23 – r24
        add     0x30,           r23                     -- r23 = 0x30 + r23
        st.b    r23,            1[r21]                  -- x hours
        mov     r22,            r23                     -- r23 = r22
        divh    r6,             r23                     -- r23 = r23/(60*60)
        mulh    60*60,          r23                     -- r23 = r23*(60*60)
        mov     r22,            r24                     -- r24 = r22
        sub     r23,            r24                     -- r24 = r24 – r23
        movea   60,     r0,     r6                      -- r6 = 60
        divh    r6,             r24                     -- r24 = r24/60
        mov     r24,            r23                     -- r23 = r24
        divh    r7,             r23                     -- r23 = r23/10
        add     0x30,           r23                     -- r23 = 0x30 + r23
        st.b    r23,            3[r21]                  -- x*10 minutes
        mov     r24,            r23                     -- r23 = r24
        divh    r7,             r24                     -- r24 = r24/10
        mulh    10,             r24                     -- r24 = r24*10
        sub     r24,            r23                     -- r23 = r23 – r24
        add     0x30,           r23                     -- r23 = 0x30 + r23
        st.b    r23,            4[r21]                  -- x minutes
        mov     r22,            r23                     -- r23 = r22
        divh    r6,             r23                     -- r23 = r23/60
        mulh    60,             r23                     -- r23 = r23*60
        mov     r22,            r24                     -- r24 = r22
        sub     r23,            r24                     -- r24 = r24 – r23
        mov     r24,            r23                     -- r23 = r24
        divh    r7,             r23                     -- r23 = r23/10
        add     0x30,           r23                     -- r23 = 0x30 + r23
        st.b    r23,            6[r21]                  -- x*10 seconds
        mov     r24,            r23                     -- r23 = r24
        divh    r7,             r24                     -- r24 = r24/10
        mulh    10,             r24                     -- r24 = r24*10
        sub     r24,            r23                     -- r23 = r23 – r24
        add     0x30,           r23                     -- r23 = 0x30 + r23
        st.b    r23,            7[r21]                  -- x seconds
        movhi   hi1(disp_flg), r0, r21                  -- r21 := watch LCD display status flag
        movea   lo(disp_flg), r21, r21
        set1    0,              0[r21]                  -- disp_flg ON
wt_exit:
```

```
        mov     r6,             r1                          -- Recovery of r1
        reti

_int_tm00:
        mov     r1,             r6                          -- r1 save
tm_countup:
        movhi   hi1(tm0_num), r0, r16                       -- r16 := timer 0 counter
        movea   lo(tm0_num), r16, r16
        ld.w    0[r16],         r17                         -- Loads tm0_num to register
        addi    1,      r17,    r17                         -- Increments tm0_num
        st.w    r17,            0[r16]                      -- Stores tm0_num to memory
tm_chk_flg:
        movhi   hi1(disp_flg), r0, r16                      -- r16 := watch LCD display status flag
        movea   lo(disp_flg), r16, r16
        tst1    0,              0[r16]                      -- Checks disp_flg
        bz      tm_exit                                     -- Jumps if dsp_flg = 0
tm_disp_lcd:
        movhi   hi1(col_ctr), r0, r16                       -- r16 := No. of display columns in watch display data
        movea   lo(col_ctr), r16, r16
        ld.b    0[r16],         r17                         -- Loads col_ctr to register
        movea   0x07,   r0,     r18                         -- r18 := 7
        cmp     r17,            r18                         -- Jumps
        bn      tm_cursol_home                              -- if col_ctr = 7 or more

        movhi   hi1(tm_buf), r0, r18                        -- r17 := display watch data
        movea   lo(tm_buf), r18, r18
        add     r17,            r18
        ld.b    0[r18],         r20                         -- r20 := display character

        movhi   hi1(lcd_addr), r0, r19                      -- r19 := LCD address
        movea   lo(lcd_addr), r19, r19
        st.b    r20,            2[r19]
        addi    1,      r17,    r17                         -- Increments col_ctr
        st.b    r17,            0[r16]                      -- Stores col_ctr to memory
        br      tm_exit                                     -- Goes to termination processing
tm_cursol_home:
        movhi   hi1(lcd_addr), r0, r19                      -- r19 := LCD address
        movea   lo(lcd_addr), r19, r19
        movea   0x02,   r0,     r20
        st.b    r20,            0[r19]                      -- LCD home position
        movhi   hi1(disp_flg), r0, r16                      -- r16 := watch LCD display status flag
        movea   lo(disp_flg), r16, r16
        clr1    0,              0[r16]                      -- disp_flg OFF
        movhi   hi1(col_ctr), r0, r16                       -- r16 := No. of display columns in watch display data
        movea   lo(col_ctr), r16, r16
        st.b    r0,             0[r16]                      -- col_ctr = 0
        br      tm_exit                                     -- Goes to termination processing
tm_exit:
        mov     r6,             r1                          -- Recovery of r1
        reti
```

(3) C program

```
#pragma ioreg                                   /* Enables use of peripheral I/O register */

/* ---------------------------------------------------------
 *   Global variable
 *---------------------------------------------------------
 */
int flg_wt  ;                                   /* 1-second display flag */
int wt_num ;                                    /* Time data */
char tm_buf[100] ;                              /* Formatted time data */
int disp_flg = 1 ;                              /* Display flag */
int tm0_num ;                                   /* Timer 0 counter */
int col_ctr ;                                   /* Display column position */
char *lcd_addr ;                                /* LCD address */

/* ---------------------------------------------------------
 *   LCD busy check routine
 *---------------------------------------------------------
 */
void bf_chk(void)
{
   while ( ((*lcd_addr) & 0x80) == 0x80 ){
   }
}

/* ---------------------------------------------------------
 *   MAIN processing
 *---------------------------------------------------------
 */
void main(void)
{
   /* Initialize global variable */
   flg_wt = 0 ;                                 /* 1-second display flag */
   wt_num = 0 ;                                 /* Time data */
   disp_flg = 0 ;                               /* Display flag */
   tm0_num = 0 ;                                /* Timer 0 counter */
   col_ctr = 0;                                 /* Display column position */
   lcd_addr = (char *)0x140000;                 /* LCD address */

   /* Prescaler mode register settings */
   PRM0 = 0x01 ;                                /* fxx/16 (fxx = 16 MHz) := specify 1 MHz for count clock */
   TOC0 = 0 ;                                   /* Sets timer output control register */
   CR00 = 100 ;                                 /* Sets interrupt interval to 0.1 ms */
   /* Sets capture/compare control register */
   CRC0 = 0 ;                                   /* Operates CR00 as compare register */
   /* Sets timer mode control register */
   TMC0 = 0x0c ;                                /* Clears and starts when TM0 and CR00 match */
   /* Timer interrupt enable (level 7) */
   TMIC00 = 0x07 ;                              /* Timer interrupt enable (level 7) */
   __EI();                                      /* Enables interrupts */

   *lcd_addr = 0x30 ;                           /* Function setting, 8 bits, 1 line, 5*7 */
   for ( tm0_num = 0 ; tm0_num < 41 ; ) {       /* Waits at least 4.1 ms */
   }
   *lcd_addr = 0x30 ;                           /* Function setting, 8 bits, 1 line, 5*7 */
   for ( tm0_num = 0 ; tm0_num < 2 ; )  {       /* Waits at least 100 µs */
   }
   *lcd_addr = 0x30 ;                           /* Function setting, 8 bits, 1 line, 5*7 */
   bf_chk();
   *lcd_addr = 0x38 ;                           /* Function setting, 8 bits, 2 lines, 5*7 */
   bf_chk();
   *lcd_addr = 0x01 ;                           /* Clears display and moves cursor to home position */
   bf_chk();
   *lcd_addr = 0x06 ;                           /* Increments cursor, no display shift */
   bf_chk();
   *lcd_addr = 0x0C ;                           /* Sets display ON and cursor OFF */

   /* Watch timer mode control register settings */
```

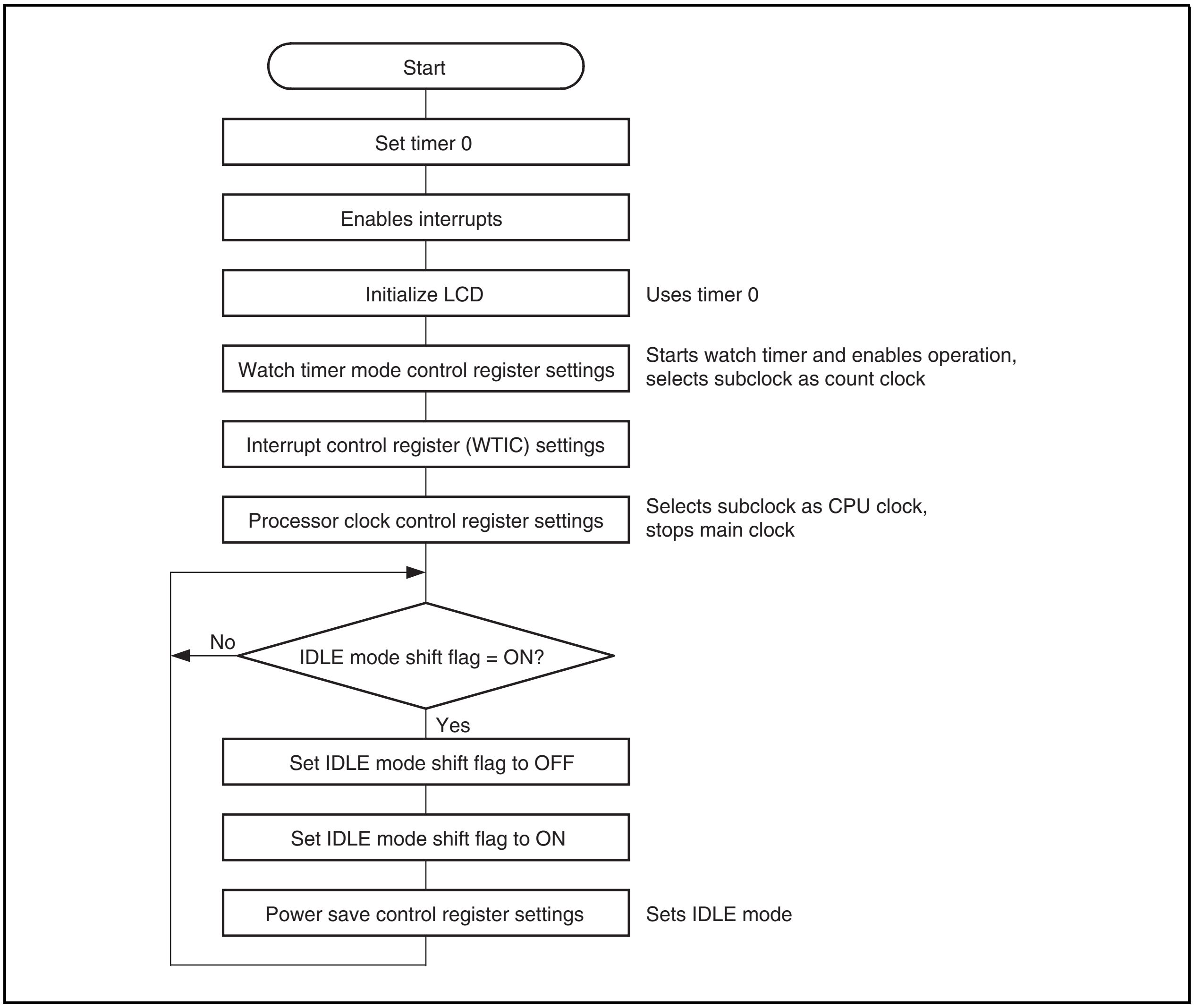
```
    WTM = 0x83 ;                                    /* Starts watch timer and enables operation */
                                                    /* Selects subclock as count clock */
    /* Sets watch interrupt level higher than the timer 0 interrupt level */
    WTIC = 0x06 ;                                   /* Enables watch timer interrupts (level 6) */
    /* Since the PCC register is a special register, it is modified via a special sequence.*/
    __asm("movea    0x44, r0, r13");
    __asm("stsr     5,      r11");                  /* Loads program status word */
    __asm("mov      r11,    r12");                  /* r12 = r11 */
    __asm("or       0x80,   r11");                  /* PSW's NP bit = 1 */
    __asm("ldsr     r11,      5");                  /* Writes PSW in which NP bit = 1 */
    __asm("st.b     r0, PRCMD[r0]");                /* Writes to PRCMD */
    /* Select subsystem clock as CPU's clock */
    __asm("st.b     r13,  PCC[r0]");                /* Sets to processor clock control register */
    __asm("ldsr     r12,      5");                  /* Returns PSW to previous setting */
    __asm("nop");                                   /* Dummy instructions (5 instructions) */
    __asm("nop");
    __asm("nop");
    __asm("nop");
    __asm("nop");
    /* Endless loop */
    for ( ; ; ) {
    }
}

/* ------------------------------------------------------------
 *  Watch timer interrupt processing
 *-------------------------------------------------------------
 */
#pragma interrupt INTWT _int_wt
void _int_wt(void)
{
    if ( flg_wt == 0 ){                             /* Second watch interrupt? */
        flg_wt = 1 ;                                /* Sets 1-second display flag to ON */
    }else{
        flg_wt = 0 ;                                /* Sets 1-second display flag to OFF */
        wt_num++ ;                                  /* Increments time data */
        /* Formats time data */
        tm_buf[0] = 0x30 + (char)(wt_num/(60*60*10)) ;
        tm_buf[1] = 0x30 + (char)((wt_num/(60*60))%10) ;
        tm_buf[2] = ':' ;
        tm_buf[3] = 0x30 + (char)(((wt_num%(60*60))/60)/10) ;
        tm_buf[4] = 0x30 + (char)(((wt_num%(60*60))/60)%10) ;
        tm_buf[5] = ':' ;
        tm_buf[6] = 0x30 + (char)((wt_num%60)/10) ;
        tm_buf[7] = 0x30 + (char)((wt_num%60)%10) ;
        disp_flg = 1 ;                              /* Sets display flag to ON */
    }
}

/* ------------------------------------------------------------
 *  Timer 0 interrupt processing
 *-------------------------------------------------------------
 */
#pragma interrupt INTTM00 _int_tm00
void _int_tm00(void)
{
    tm0_num++ ;                                     /* Increments count on timer 0 counter */
    if ( disp_flg == 1 ) {                          /* Display flag ON? */
        if ( col_ctr != 8 ) {                       /* Is display column position 8 or less? */
            *( lcd_addr + 2 ) = tm_buf[col_ctr] ; /* Displays 1 character of time data */
            col_ctr++ ;                     /* Increments display column position */
    }else{
            *lcd_addr = 0x02 ;              /* LCD home position */
            disp_flg = 0 ;                  /* Sets display flag to OFF */
            col_ctr = 0 ;                   /* Returns display column position to 0 */
    }
  }
}
```

4.4.9 Use of NMI switch for recovery of watch operation during IDLE mode

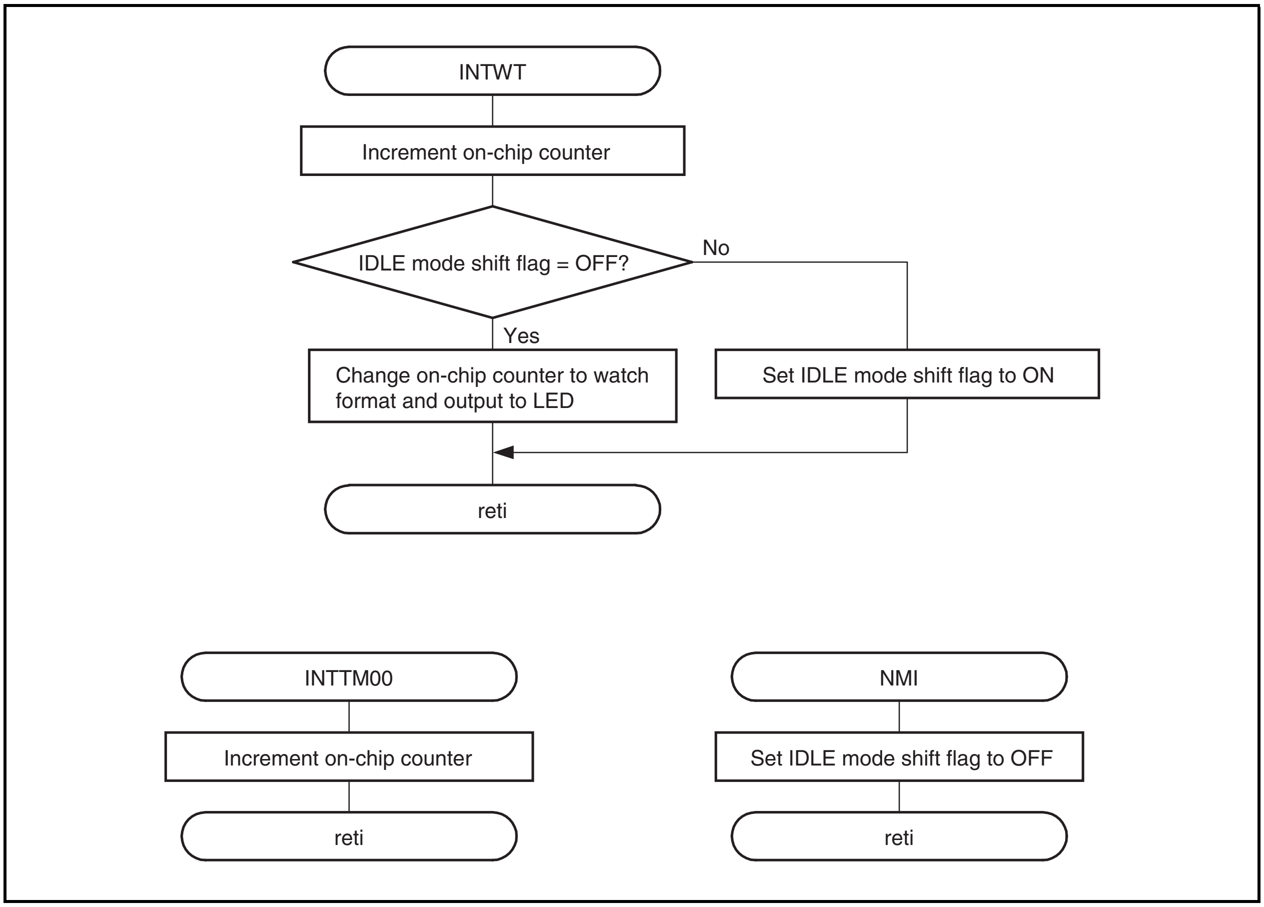
A watch is displayed during normal mode.

When the INT switch is pressed, the mode is switched to IDLE mode. If there is a subclock, the watch timer continues operating in IDLE mode since f_{XX} has been specified as its count clock. During IDLE mode, an INTWT causes a return to normal mode, but in this case only the on-chip counter is updated, so the system is switched to IDLE mode without changing the display. When an NMI resets normal mode, the internal flag is cleared and the watch is displayed.

(1) Flow chart (1/2)



(1) Flow chart (2/2)



(2) Assembly language program

```
.globl  start

.globl  _int_nmi
.globl  _int_p0
.globl  _int_wt
.globl  _int_tm00

.set    tm_buf,       0x120000              -- Watch display data (xx:xx:xx)
.set    lcd_addr,     0x140000              -- LCD address

## Set interrupt handler
.section "NMI"                              -- NMI interrupt handler
        jr      _int_nmi                    -- Go to interrupt processing
.section "INTP0"                            -- INTP interrupt handler
        jr      _int_p0                     -- Go to interrupt processing
.section "INTWT"                            -- INTWT interrupt handler
        jr      _int_wt                     -- Go to interrupt processing
.section "INTTM00"                          -- INTM00 interrupt handler
        jr      _int_tm00                   -- Go to interrupt processing

.text
        .align 4
start:
        mov     r0,          r26            -- Sets r26 (1-second display flag) to OFF
        mov     r0,          r27            -- Initializes time data
        mov     r0,          r28            -- Sets IDLE mode display flag to OFF

init_lcd:
        mov     0x01,        r11            -- r11 := 1
        -- fxx/16 (fxx = 16 MHz) = Specify 1 MHz for count clock
        st.b    r11,         PRM0[r0]       -- Prescaler mode register 0 settings
        st.b    r0,          TOC0[r0]       -- Timer output control register 0 settings
        movea   0x64,  r0,   r11            -- r11 := 0x64
        -- Set 0.1 ms as interrupt interval
        st.h    r11,         CR00[r0]
        -- Operate CR00 as compare register
        st.b    r0,          CRC0[r0]       -- Capture/compare control register 0 settings
        mov     0x0c,        r11
        -- Clear and start when TM0 and CR00 match
        st.b    r11,         TMC0[r0]       -- Timer mode control register 0 settings
        mov     0x07,        r11
        st.b    r11,         TMIC00[r0]     -- Enables timer interrupts
        mov     r0,          r25            -- r25 := used as timer counter
        ei

        movea   lcd_addr & 0xffff, r0, r20
        movhi   lcd_addr >> 16, r20, r20    -- LCD address
        movea   0x30,  r0,   r10
        st.b    r10,         0[r20]         -- Function setting; 8 bits, 1 line  5*7
wait_1:
        cmp     41,          r25
        jn      wait_1                      -- Waits at least 4.1 ms

        movea   0x30,  r0,   r10
        st.b    r10,         0[r20]         -- Function setting; 8 bits, 1 line  5*7
        mov     r0,          r25
wait_2:
        cmp     2,           r25
        jn      wait_2                      -- Waits at least 100 us
        movea   0x30,  r0,   r0, r10
        jarl    bf_chk,      r29            -- LCD busy check
        st.b    r10,         0[r20]         -- Function setting; 8 bits, 1 line  5*7
        movea   0x38,  r0,   r10
        jarl    bf_chk,      r29            -- LCD busy check
        st.b    r10,         0[r20]         -- Function setting; 8 bits, 2 lines  5*7
        mov     0x01,        r10
        jarl    bf_chk,      r29            -- LCD busy check
```

```
        st.b    r10,          0[r20]          -- Clears display and moves cursor to home position
        mov     0x06,         r10
        jarl    bf_chk,       r29             -- LCD busy check
        st.b    r10,          0[r20]          -- Increments cursor, no display shift
        mov     0x0C,         r10
        jarl    bf_chk,       r29             -- LCD busy check
        st.b    r10,          0[r20]          -- Sets display ON and cursor OFF
        movea   0x47,  r0,    r11
        st.b    r11,          TMIC00[r0]      -- Prohibits timer interrupts

init_wt:
        ## Watch timer mode control register settings
        movea   0x83,  r0,    r10
        st.b    r10,          WTM[r0]         -- Starts watch timer and enables operation
        -- Selects subsystem clock as count clock
        mov     0x07,         r10
        st.b    r10,          WTIC[r0]        -- Enables watch timer interrupts
init_pic0:
        mov     0x07,         r10
        st.b    r10,          PIC0[r0]        -- Enables INT switch interrupts
set_sub_clock:
        ## Since the PSC register is a special register, it is modified via a special sequence.
        stsr    5,            r21             -- Loads program status word
        mov     r21,          r22             -- r22 = r21
        or      0x80,         r21             -- PSW's NP bit = 1
        movea   0xc0,   r0,   r10
        ldsr    r21,          5
        st.b    r0,           PRCMD[r0]       -- Writes to PRCMD
        ## Stops CLKOUT output
        st.b    r10,          PSC[r0]         -- Power save control register settings
        ldsr    r22,          5
        nop                                   -- Dummy instructions (2 instructions)
        nop
        ## Since the PCC register is a special register, it is modified via a special sequence.
        movea   0x64,   r0,   r13
        stsr    5,            r11             -- Loads program status word
        mov     r11,          r12             -- r12 = r11
        or      0x80,         r11             -- PSW's NP bit = 1
        ldsr    r11,          5               -- Writes PSW in which NP bit = 1
        st.b    r0,           PRCMD[r0]       -- Writes to PRCMD
        ## Stops main clock and subsystem clock
        st.b    r13,          PCC[r0]         -- Sets to processor clock control register
        ldsr    r12,          5               -- Returns PSW to previous setting
        nop                                   -- Dummy instructions (2 instructions)
        nop
fv_loop:
        cmp     r0,           r24             -- r24 := Int switch select flag is ON
        bz      fv_loop
        mov     r0,           r24             -- r24 := INT switch select flag is OFF
        mov     1,            r28             -- IDLE mode display flag is ON
        ## The following is the sequence for setting special registers
        stsr    5,            r21
        mov     r21,          r22
        or      0x80,         r21
        mov     0x04,         r10
        ldsr    r21,          5
        st.b    r0,           PRCMD[r0]       -- Writes to PRCMD
        ## Enables CLKOUT output and sets IDLE mode
        st.b    r10,          PSC[r0]         -- Power save control register settings
        ldsr    r22,          5
        nop                                   -- Dummy instructions (2 instructions)
        nop
        br      fv_loop                       -- Endless loop

bf_chk:
        movhi   hi1(lcd_addr), r0, r7         -- r7 := LCD address
        movea   lo(lcd_addr), r7, r7
```

```
        movea   0x80,   r0,    r8              -- r8 := 0x80
bf_chk0:
        ld.b    0[r7],         r9
        andi    0x80,   r9,    r9
        cmp     r8,            r9
        bz      bf_chk0
        jmp     [r29]

_int_wt:                                       -- Watch timer interrupt processing
        cmp     r0,            r26             -- Checks r26 (1-second display flag)
        bnz     r11_0_no
        mov     1,             r26             -- Sets r26 (1-second display flag) to OFF
        add     1,             r27             -- Increments watch data
        cmp     1,             r28             -- IDLE mode?

        bz      disp_skip                      -- if yes, then jump
        ## Changes r10 to watch format and displays on LCD
        movhi   hi1(tm_buf), r0, r20
        movea   lo(tm_buf), r20, r20
        movea   0x03a, r0,     r21             -- Sets ":" character
        st.b    r21,           2[r20]          -- Second character in time display character string
        st.b    r21,           5[r20]          -- Fifth character in time display character string
        mov     r27,           r7              -- r7 = r27
        movea   60*60*10, r0, r6
        andi    0xffff,  r6, r6                -- r6 = 60*60*10
        jarl    ___div,        lp              -- r6 = r7/(60*60*10)
        add     0x30,          r6              -- r6 = 0x30 + r15
        st.b    r6,            0[r20]          -- x*10 hours
        mov     r27,           r15             -- r15 = r27
        movea   60*60,   r0, r6                -- r6 = 60*60
        divh    r6             r15             -- r15 = r15/(60*60)
        mov     r15,           r16             -- r16 = r15
        mov     10,            r7              -- r7 = 10
        divh    r7,            r16             -- r16 = r16/10
        mulh    10,            r16             -- r16 = r16*10
        sub     r16,           r15             -- r15 = r15 – r16
        add     0x30,          r15             -- r15 = 0x30 + r15
        st.b    r15,           1[r20]          -- x hours
        mov     r27,           r15             -- r15 = r27
        divh    r6,        r15                 -- r15 = r15/(60*60)
        mulh    60*60,         r15             -- r15 = r15*(60*60)
        mov     r27,           r16             -- r16 = r27
        sub     r15,           r16             -- r16 = r16 – r15
        movea   60,            r0, r6          -- r6 = 60
        divh    r6,            r16             -- r16 = r16/60
        mov     r16,           r15             -- r15 = r16
        divh    r7,            r15             -- r15 = r15/10
        add     0x30,          r15             -- r15 = 0x30 + r15
        st.b    r15,           3[r20]          -- x*10 minutes
        mov     r16,           r15             -- r15 = r16
        divh    r7,            r16             -- r16 = r16/10
        mulh    10,            r16             -- r16 = r16*10
        sub     r16,           r15             -- r15 = r15 – r16
        add     0x30,          r15             -- r15 = 0x30 + r15
        st.b    r15,           4[r20]          -- x minutes
        mov     r27,           r15             -- r15 = r27
        divh    r6,            r15             -- r15 = r15/60
        mulh    60,            r15             -- r15 = r15*60
        mov     r27,           r16             -- r16 = r27
        sub     r15,           r16             -- r16 = r16 – r15
        mov     r16,           r15             -- r15 = r16
        divh    r7,            r15             -- r15 = r15/10
        add     0x30,          r15             -- r15 = 0x30 + r15
        st.b    r15,           6[r20]          -- x*10 seconds
        mov     r16,           r15             -- r15 = r16
        divh    r7,            r16             -- r16 = r16/10
        mulh    10,            r16             -- r16 = r16*10
        sub     r16,           r15             -- r15 = r15 – r16
```

```
        add     0x30,           r15             -- r15 = 0x30 + r15
        st.b    r15,            7[r20]          -- x seconds
        movea   lcd_addr & 0xffff, r0, r21
        movhi   lcd_addr >> 16, r21, r21        -- r21 := LCD address
        mov     0x02,           r15
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            0[r21]          -- LCD home position
        ld.b    0[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    1[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    2[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    3[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    4[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    5[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    6[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        ld.b    7[r20],         r15             -- Loads 1 character of time data
        jarl    bf_chk,         r29             -- LCD busy check
        st.b    r15,            2[r21]          -- Displays one character of time data via LCD
        br      disp_skip
r11_0_no:
        mov     r0,             r26             -- Sets r26 (1-second display flag) to ON
disp_skip:
        cmp     1,              r28             -- IDLE mode?
        bnz     int_wt_exit                     -- if no, then jump
        mov     1,              r24             -- r24 := INT switch select flag is ON
int_wt_exit:
        reti                                    -- Recovers from interrupt processing

### Press INT switch to set IDLE mode
_int_p0:                                        -- Interrupt processing triggered by INT switch
        mov     1,              r24             -- r24 := INT switch select flag is ON
        reti

_int_nmi:                                       -- NMI interrupt processing
        mov     0,              r28             -- IDLE mode display flag is ON
        reti

_int_tm00:                                      -- Timer 0 interrupt processing
        add     1,              r25             -- Increments r25
        reti
```

(3) C program

```
#pragma ioreg                                 /* Enables use of peripheral I/O register name */

/* ---------------------------------------------------------
 *   Global variable
 *---------------------------------------------------------
 */
int flg_wt  ;                                 /* 1-second display flag */
int wt_num ;                                  /* Time data */
char tm_buf[8] ;                              /* Formatted time data */
int tm0_num ;                                 /* Timer 0 counter */
int intsw_flg ;                               /* IDLE mode shift flag */
int idle_mode ;                               /* IDLE mode status flag */
char *lcd_addr ;                              /* LCD address */

/* ---------------------------------------------------------
 *   LCD busy check routine
 *---------------------------------------------------------
 */
void bf_chk(void)
{
   while ( ((*lcd_addr) & 0x80) == 0x80 ){
   }
}

/* ---------------------------------------------------------
 *   MAIN processing
 *---------------------------------------------------------
 */
void main(void)
{
   /* Initialize global variable */
   flg_wt = 0 ;                               /* 1-second display flag */
   wt_num = 0 ;                               /* Time data */
   tm0_num = 0 ;                              /* Timer 0 counter */
   intsw_flg = 0 ;                            /* IDLE mode shift flag */
   idle_mode = 0 ;                            /* IDLE mode status flag */
   lcd_addr = (char *)0x140000;               /* LCD address */

   /* Prescaler mode register settings */
   PRM0 = 0x01 ;                              /* fxx/16 (fxx = 16 MHz) := specify 1 MHz for count clock */
   TOC0 = 0 ;                                 /* Timer output control register 0 settings */
   CR00 = 100 ;                               /* Sets interrupt interval to 0.1 ms */
   /* Capture/compare control register settings */
   CRC0 = 0 ;                                 /* Operates CR00 as compare register */
   /* Timer mode control register settings */
   TMC0 = 0x0c ;                              /* Clears and starts when TM0 and CR00 match */
   /* Timer interrupt enable (level 7)*/
   TMIC00 = 0x07 ;                            /* Timer interrupt enable (level 7) */
   PIC0 = 0x07 ;                              /* Enables INT switch interrupts */
   __EI();                                    /* Enables interrupts */

   *lcd_addr = 0x30 ;                         /* Function setting, 8 bits, 1 line, 5*7 */
   for ( tm0_num = 0 ; tm0_num < 41 ; ) {     /* Waits at least 4.1 ms */
   }
   *lcd_addr = 0x30 ;                         /* Function setting, 8 bits, 1 line, 5*7 */
   for ( tm0_num = 0 ; tm0_num < 2 ; ) {      /* Waits at least 100 µs */
   }
   *lcd_addr = 0x30 ;                         /* Function setting, 8 bits, 1 line, 5*7 */
   bf_chk();
   *lcd_addr = 0x38 ;                         /* Function setting, 8 bits, 2 lines, 5*7 */
   bf_chk();
   *lcd_addr = 0x01 ;                         /* Clears display and moves cursor to home position */
   bf_chk();
   *lcd_addr = 0x06 ;                         /* Increments cursor, no display shift */
   bf_chk();
   *lcd_addr = 0x0C ;                         /* Sets display ON and cursor OFF */
```

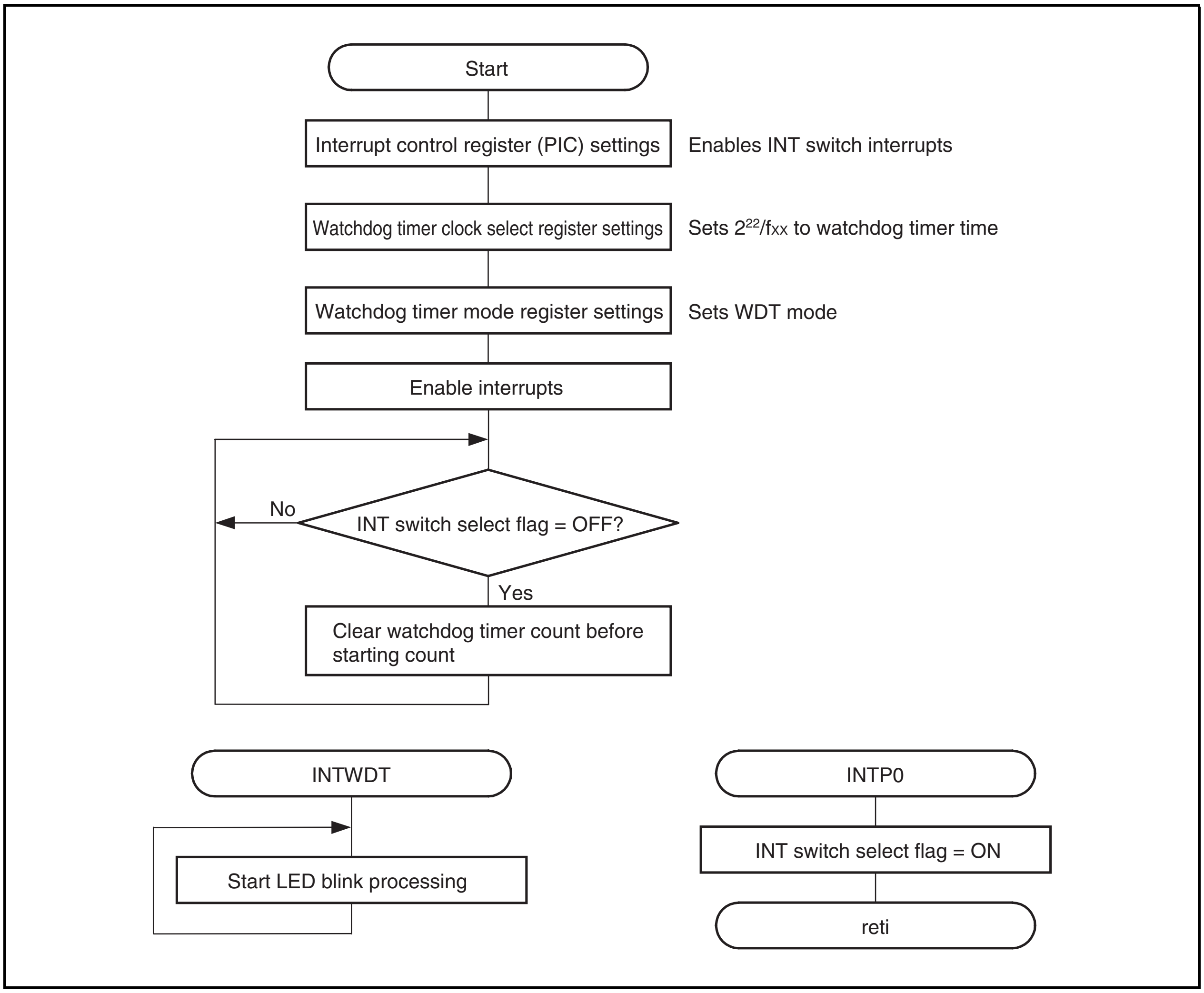
```
    /* Watch timer mode control register settings */
    WTM = 0x83 ;                                    /* Starts watch timer and enables operation */
                                                    /* Selects subclock as count clock */
    /* Sets watch interrupt level higher than the timer 0 interrupt level */
    WTIC = 0x06 ;                                   /* Enables watch timer interrupts (level 6) */

    /* Since the PSC register is a special register, it is modified via a special sequence. */
    __asm("stsr     5,          r21");
    __asm("mov      r11,        r22");
    __asm("or       0x80,       r21");
    __asm("movea    0xC0,       r0, r10");
    __asm("ldsr     r21,        5");                /* Sets PSW's NP bit to ON and writes */
    __asm("st.b     r0,         PRCMD[r0]");        /* Writes to PRCMD */
    /* Disables CLKOUT output */
    __asm("st.b     r10,        PSC[r0]");          /* Sets CLKOUT output disabled */
    __asm("ldsr     r22,        5");                /* Returns PSW to previous setting */
    __asm("nop");                                   /* Dummy instructions (2 instructions) */
    __asm("nop");
    /* Since the PCC register is a special register, it is modified via a special sequence. */
    __asm("movea    0x64,       r0, r13");
    __asm("stsr     5,          r11");              /* Load program status word
    __asm("mov      r11,        r12");              /* r12 = r11 */
    __asm("or       0x80,       r11")               /* PSW's NP bit = 1 */
    __asm("ldsr     r11,        5");                /* Writes PSW in which NP bit = 1 */
    __asm("st.b     r0,         PRCMD[r0]");        /* Writes to PRMCMD */
    /* Select subsystem clock as CPU's clock and stop main clock */
    __asm("st.b     r13,        PCC[r0]");          /* Sets to processor clock control register */
    __asm("ldsr     r12,        5");                /* Returns PSW to previous setting */
    __asm("nop");                                   /* Dummy instructions (2 instructions) */
    __asm("nop");
    /* Endless loop */
    for ( ; ; ) {
        if ( intsw_flg == 1 ) {                     /* IDLE mode shift flag = ON? */
            intsw_flg = 0 ;                         /* SETS IDLE mode shift flag to OFF */
            idle_mode = 1 ;                         /* SETS IDLE mode shift flag to ON */
            /* Since the PSC register is a special register, it is modified via a special sequence. */
            __asm("stsr 5,      r21") ;
            __asm("mov  r21,    r22") ;
            __asm("or   0x80,   r21") ;
            __asm("mov  0x04,   r10") ;
            __asm("ldsr r21,    5") ;               /* Writes PSW in which NP bit = 1 */
            __asm("st.b r0,     PRCMD[r0]") ;       /* Writes to PRCMD */
            /* Power save control register settings */
            __asm("st.b r10,    PSC[r0]") ;         /* Enables CLKOUT output and sets IDLE mode */
            __asm("ldsr r22,    5") ;               /* Returns PSW to previous setting */
            __asm("nop") ;                          /* Dummy instructions (2 instructions) */
            __asm("nop") ;
        }
    }
}

/* -------------------------------------------------------
 *  Watch timer interrupt processing
 *-------------------------------------------------------
 */
#pragma interrupt INTWT _int_wt
void _int_wt(void)
{
    if ( flg_wt == 0 ){                             /* Second watch timer interrupt? */
        flg_wt = 1 ;                                /* Sets 1-second display flag to ON */
    }else{
        flg_wt = 0 ;                                /* Sets 1-second display flag to OFF */
        wt_num++ ;                                  /* Increments time data */
        if ( idle_mode == 0 ) {
            /* Formats time data */
            tm_buf[0] = 0x30 + (char)(wt_num/(60*60*10)) ;
            tm_buf[1] = 0x30 + (char)((wt_num/(60*60))%10) ;
            tm_buf[2] = ':' ;
```

```
            tm_buf[3] = 0x30 + (char)(((wt_num%(60*60))/60)/10) ;
            tm_buf[4] = 0x30 + (char)(((wt_num%(60*60))/60)%10) ;
            tm_buf[5] = ':' ;
            tm_buf[6] = 0x30 + (char)((wt_num%60)/10) ;
            tm_buf[7] = 0x30 + (char)((wt_num%60)%10) ;
            /* Outputs to LCD */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[0] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[1] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[2] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[3] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[4] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[5] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[6] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *( lcd_addr + 2 ) = tm_buf[7] ;     /* Displays one character of time data */
            bf_chk();                           /* LCD busy check */
            *lcd_addr = 0x02 ;                  /* LCD home position */
        }else{
            /* Sets flag to ON to enable shift back to IDLE mode during MAIN processing */
            intsw_flg = 1 ;
        }
    }
}

/* ---------------------------------------------------------
 *  Timer 0 interrupt processing
 *---------------------------------------------------------
 */
#pragma interrupt INTTM00 _int_tm00
void _int_tm00(void)
{
    tm0_num++ ;                                 /* Increments timer 0 count clock */
}

/* ---------------------------------------------------------
 *  INT switch interrupt processing
 *---------------------------------------------------------
 */
#pragma interrupt INTP0 _int_p0
void _int_p0(void)
{
    intsw_flg = 1 ;                             /* IDLE mode shift flag = ON */
    idle_mode = 1 ;                             /* IDLE mode status flag = ON */
}

/* ---------------------------------------------------------
 *  NMI interrupt processing
 *---------------------------------------------------------
 */
#pragma interrupt NMI _int_nmi
void _int_nmi(void)
{
idle_mode = 0 ;                     /* IDLE mode status flag = OFF */
}
```

4.4.10 Loop detection program using watchdog timer

When the INT switch is pressed, the watchdog timer mode register's RUN bit is not set and loop processing occurs. At that time, the watchdog timer operates and issues an INTWDT. The LED blinks to indicate this processing.

(1) Flow chart



(2) Assembly language program

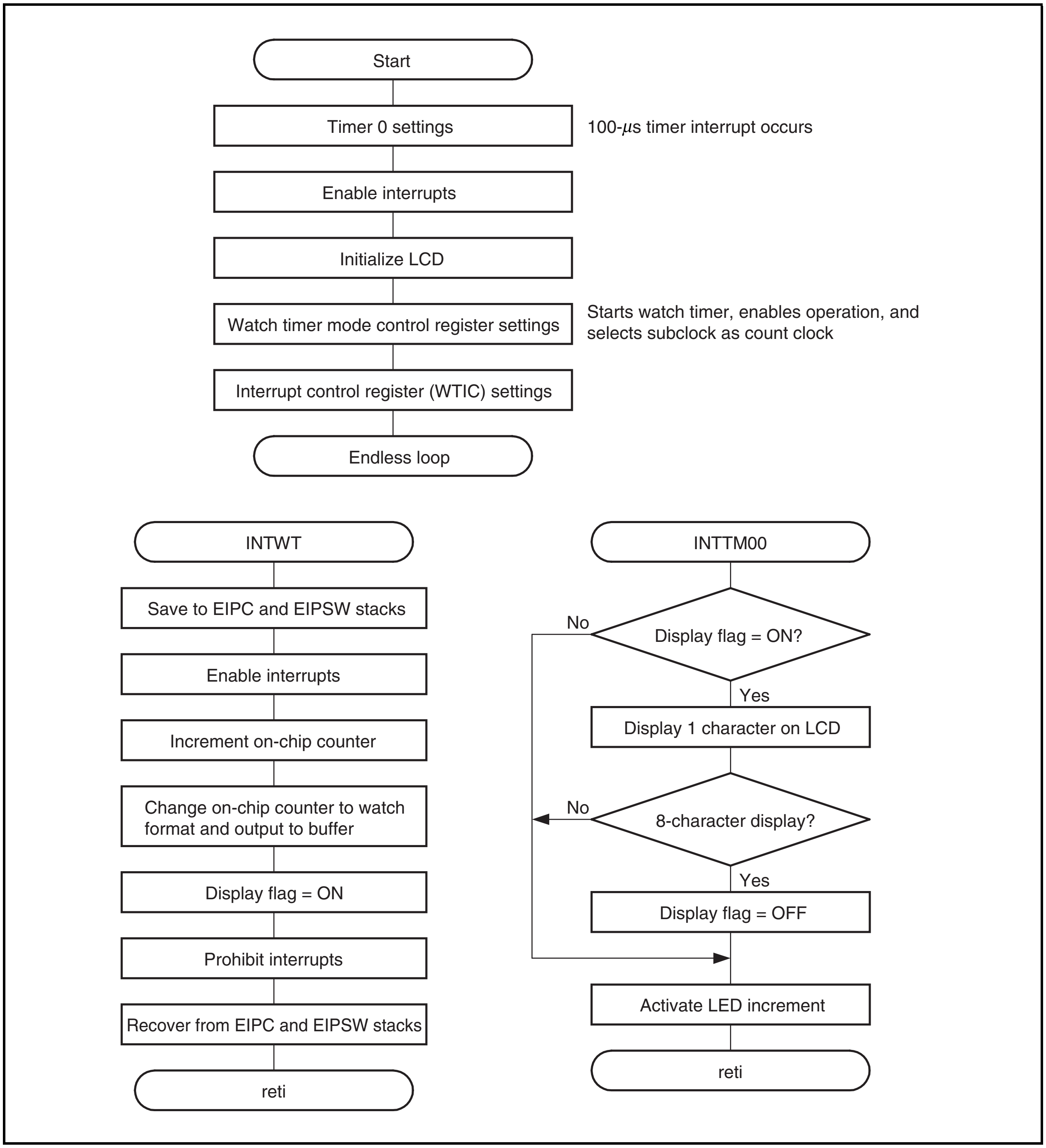
```
.globl  start

## Set interrupt handler
.section "INTP0"                          -- INTP0 interrupt handler
        jr      _int_p0                   -- Go to interrupt processing
.section "INTWDT"                         -- INTWDT interrupt handler
        jr      _int_wdt                  -- Go to interrupt processing

.text
        .align 4
start:

        mov     0x07,    r10
        st.b    r10,     PIC0[r0]         -- INT switch interrupt enable

        mov     0x07,    r10
        ## Set 2^22/fxx to watchdog timer time
        st.b    r10,     WDCS[r0]         -- Watchdog timer clock select register settings

        movea   0x10,    r0, r10
        ## Sets watchdog timer mode 1
        st.b    r10,     WDTM[r0]         -- Watchdog timer mode register settings
        mov     r0,      r20              -- Sets r20 (INTSW select flag) to OFF
        ei                                -- Enables interrupts
LABEL1:
        nop
        nop
        nop
        nop
        nop
        -- If r20 = 0 (if INT switch has not been pressed)
        cmp     r0,      r20
        bnz     LABEL1                    -- Do not jump
        set1    7,       WDTM[r0]         -- Clears watchdog timer count before starting count
        br      LABEL1

## Jump to here when watchdog timer detects runaway
led_start:                                -- LED blink processing
        mov     r0,      r16              -- r16 := ON or OFF flag
led_on_off:
        jarl    wait,    r29
        cmp     r0,      r16
        bz      led_on                    -- Go to LED ON processing when r16 = 0
        br      led_off                   -- Go to LED OFF processing when r16 = 1
led_on:
        movea   0x10,    r0, r17
        st.b    r17,     P3[r0]           -- LED ON
        mov     1,       r16              -- Go to LED OFF processing when r16 = 1
        br      led_on_off
led_off:
        mov     r0,      r17
        st.b    r17,     P3[r0]           -- LED OFF
        mov     r0,      r16              -- Go to LED ON processing when r16 = 0
        br      led_on_off

wait:                                     -- Software wait processing
        or1     1000000 & 0xffff, r0, r15
        movhi   1000000 >> 16, r15, r15
wait_1:
        add     -1,      r15              -- Executes 1,000,000 loops
        bnz     wait_1
        jmp     [r29]

_int_p0:
        mov     r1,      r6               -- Saves r1
        mov     1,       r20              -- Sets r20 to ON
```

```
        mov     r6,     r1                  -- Recovery of r1
        reti

_int_wdt:                                   -- Watchdog timer interrupt processing
        jr      led_start                   -- Go to LED blink processing
```

4.4.11 Multiple interrupts

Timer 0 interrupts are also processed as part of watch timer interrupt processing. An watch timer interrupt causes the on-chip watch counter to be incremented and a timer 0 interrupt results in a display of clock data and activation of the increment LED.

(1) Flow chart



(2) Assembly language program

```
#   <Registers used>
#    r3                    : Stack address
#    r6                    : Register for saving r1 value
#    r11,r12,r13,r14,r15: Register for main processing
#    r16,r17,r18,r19,r20: Register for timer 0 interrupt processing
#    r21,r22,r23,r24,r25: Register for watch timer interrupt processing
#    r26,r27,r28,r29     : bf_chk routine work register
.globl  start

.set    tm_buf,      0x120000              -- Watch display data (xx:xx:xx) (8 bytes)
.set    col_ctr,     0x120008              -- No. of columns in watch display data (1 byte)
.set    flg_wt,      0x120009              -- 1-second display flag (1 byte)
.set    wt_num,      0x120010              -- Watch data (4 bytes)
.set    disp_flg,    0x120014              -- Watch LCD display status flag (displays "1")
.set    tm0_num,     0x120018              -- Timer 0 counter (4 bytes)
.set    led_ivl,     0x12001c              -- LED display interval (4 bytes)
.set    mem_last,    0x120020              -- End address of work variable area
.set    lcd_addr,    0x140000              -- LCD address
.set    stk_addr,    0xffefff              -- Stack address

## Set interrupt handler
.section "INTWT"                   -- INTWT interrupt handler
        jr      _int_wt                    -- Go to interrupt processing
.section "INTTM00"                 -- INTTM00 interrupt handler
        jr      _int_tm00                  -- Go to interrupt processing

.text
        .align 4
start:
        mem_init:
        movhi   hi1(tm_buf), r0, r11       -- r11 := start address of working memory
        movea   lo(tm_buf), r11, r11
        movhi   hi1(mem_last), r0, r12     -- r12 := end address of working variable area
        movea   lo(mem_last), r12, r12
mem_init_lp:
        st.b    r0,          0[r11]        -- Sets 0 for working variable
        addi    1,    r11,   r11           -- Increments address to zero-clear
        cmp     r11,         r12
        bnz     mem_init_lp                -- Loop if not end address of address to zero-clear
ins_3a:
        movhi   hi1(tm_buf), r0, r11       -- r11 = start address of working memory
        movea   lo(tm_buf), r11, r11
        movea   0x3a, r0,    r13           -- For time data:
        st.b    r13,         2[r11]        -- ':' = Set 0x3A
        st.b    r13,         5[r11]        -- ':' = Set 0x3A
set_stack_adr:
        movhi   hi1(stk_addr), r0, r3      -- Set stack address
        movea   lo(stk_addr), r3, r3

init_tm0:
        movea   0x01, r0,  r11             -- r11 := 1
        -- fxx/16 (fxx = 16 MHz) = Specify 1 MHz for count clock
        st.b    r11,        PRM0[r0]       -- Prescaler mode register 0 settings
        st.b    r0,         TOC0[r0]       -- Timer output control register 0 settings
        movea   0x64, r0,   r11            -- r11 := 100 = 0x64
        -- Set 0.1 ms as interrupt interval
        st.h    r11,        CR00[r0]
        -- Operate CR00 as compare register
        st.b    r0,         CRC0[r0]       -- Capture/compare control register 0 settings
        mov     0x0c,       r11
        -- Clear and start when TM0 and CR00 match
        st.b    r11,        TMC0[r0]       -- Timer mode control register 0 settings
        mov     0x06,       r11
        st.b    r11,        TMIC00[r0]     -- Enables timer interrupts (level 6)
        ei
init_lcd:
```

```
        movhi   hi1(lcd_addr), r0, r11          -- r11 := LCD address
        movea   lo(lcd_addr), r11, r11
        movhi   hi1(tm0_num), r0, r12           -- r12 := timer 0 count address
        movea   lo(tm0_num), r12, r12
        movea   0x30,   r0,   r13               -- r13 := 0x30
        st.b    r13,          0[r11]            -- Function setting; 8 bits, 1 line  5*7
        st.w    r0,           0[r12]            -- Timer 0 counter = 0
        movea   0x29,  r0,   r13                -- r13 := 0x29 = 41
wait_1:
        ld.w    0[r12],       r14               -- r14 := timer 0 counter
        cmp     r14,          r13               -- Checks whether or not timer 0 counter has reached 41
        bp      wait_1                          -- Waits at least 4.1 ms
        st.b    r13,          0[r11]            -- Function setting; 8 bits, 1 line  5*7
        st.w    r0,           0[r12]            -- Timer 0 counter = 0
        mov     0x2,          r13               -- r13 := 0x2
wait_2:
        ld.w    0[r12],       r14               -- r14 := timer 0 counter
        cmp     r14,          r13               -- Checks whether or not timer 0 counter has reached 2
        bp      wait_2                          -- Waits at least 100 μs
        st.b    r13,          0[r11]            -- Function setting; 8 bits, 1 line  5*7
        jarl    bf_chk,       r29               -- LCD busy check
        movea   0x38,  r0,    r13
        st.b    r13,          0[r11]            -- Function setting; 8 bits, 2 lines 5*7
        jarl    bf_chk,       r29               -- LCD busy check
        mov     0x01,         r13
        st.b    r13,          0[r11]            -- Clears display, moves cursor to home position
        jarl    bf_chk,       r29               -- LCD busy check
        mov     0x06,         r13
        st.b    r13,          0[r11]            -- Increments cursor, no display shift
        jarl    bf_chk,       r29               -- LCD busy check
        mov     0x0C,         r13
        st.b    r13,          0[r11]            -- Sets display ON and sets cursor OFF

init_wt:
        ## Watch timer mode control register settings
        movea   0x83,   r0,   r11
        st.b    r11,          WTM[r0]           -- Starts watch timer and enables operation
                                                -- Selects subsystem clock as count clock
        mov     0x07,         r11               -- r10 := 0x07
        st.b    r11,          WTIC[r0]          -- Enables watch timer interrupts (level 7)
fv_loop:
        br      fv_loop

bf_chk:
        movhi   hi1(lcd_addr), r0, r27          -- r27 := LCD address
        movea   lo(lcd_addr), r27, r27
        movea   0x80,   r0,   r26               -- r26 := 0x80
bf_chk0:
        ld.b    0[r27],       r28
        andi    0x80,   r28,  r28
        cmp     r26,          r28
        bz      bf_chk0
        jmp     [r29]

_int_wt:
        ## Save to EIPC and EIPSW stacks
        add     -12,          r3
        st.w    r1,           8[r3]             -- Saves r1
        stsr    0,            r21               -- Saves to EIPC
        st.w    r21,          4[r3]
        stsr    1,            r21
        st.w    r21,          0[r3]             -- SAVES to EIPSW
        ei                                      -- Enables multiple interrupts

wt_chk:
        movhi   hi1(flg_wt), r0, r21            -- r21 := 1-second display flag
        movea   lo(flg_wt), r21, r21
        not1    0,            0[r21]            -- Inverts flg_wt
        tst1    0,            0[r21]
```

```
        bnz     wt_exit                         -- Just skips if flg_wt = 1

#       mov     0x800000,     r21
        movhi   0x200000 >> 16, r0, r21
#       mov     0x100000,     r21
#       mov     0x80000,      r21
lp1:
        add     -1,           r21
        bnz     lp1

wt_data_set:
        movhi   hi1(wt_num), r0, r21            -- r21 := watch data
        movea   lo(wt_num), r21, r21
        ld.w    0[r21],       r22               -- Loads wt_num to register
        addi    1,    r22,    r22               -- Increments wt_num
        st.w    r22,          0[r21]            -- Stores wt_num to memory
        ## Converts r22 to watch format and stores it in memory
        movhi   hi1(tm_buf), r0, r21            -- r21 := watch display data
        movea   lo(tm_buf), r21, r21
        mov     r22,          r7                -- r7 = r22
        movea   60*60*10, r0, r6
        andi    0xffff,   r6, r6                -- r6 = 60*60*10
        jarl    ___div      , lp                -- r6 = r7/(60*60*10)
        add     0x30,         r6                -- r6 = 0x30 + r6
        st.b    r6,           0[r21]            -- x*10 hours
        mov     r22,          r23               -- r23 = r22
        movea   60*60,    r0, r6                -- r6 = 60*60
        divh    r6,           r23               -- r23 = r23/(60*60)
        mov     r23,          r24               -- r24 = r23
        mov     10,           r7                -- r7 = r10
        divh    r7,           r24               -- r24 = r24/10
        mulh    10,           r24               -- r24 = r24*10
        sub     r24,          r23               -- r23 = r23 – r24
        add     0x30,         r23               -- r23 = 0x30 + r23
        st.b    r23,          1[r21]            -- x hours
        mov     r22,          r23               -- r23 = r22
        divh    r6,           r23               -- r23 = r23/(60*60)
        mulh    60*60,        r23               -- r23 = r23*(60*60)
        mov     r22,          r24               -- r24 = r22
        sub     r23,          r24               -- r24 = r24 – r23
        movea   60,       r0, r6                -- r6 = 60
        divh    r6,           r24               -- r24 = r24/60
        mov     r24,          r23               -- r23 = r24
        divh    r7,           r23               -- r23 = r23/10
        add     0x30,         r23               -- r23 = 0x30 + r23
        st.b    r23,          3[r21]            -- x*10 minutes
        mov     r24,          r23               -- r23 = r24
        divh    r7,           r24               -- r24 = r24/10
        mulh    10,           r24               -- r24 = r24*10
        sub     r24,          r23               -- r23 = r23 – r24
        add     0x30,         r23               -- r23 = 0x30 + r23
        st.b    r23,          4[r21]            -- x minutes
        mov     r22,          r23               -- r23 = r22
        divh    r6,           r23               -- r23 = r23/60
        mulh    60,           r23               -- r23 = r23*60
        mov     r22,          r24               -- r24 = r22
        sub     r23,          r24               -- r24 = r24 – r23
        mov     r24,          r23               -- r23 = r24
        divh    r7,           r23               -- r23 = r23/10
        add     0x30,         r23               -- r23 = 0x30 + r23
        st.b    r23,          6[r21]            -- x*10 seconds
        mov     r24,          r23               -- r23 = r24
        divh    r7,           r24               -- r24 = r24/10
        mulh    10,           r24               -- r24 = r24*10
        sub     r24,          r23               -- r23 = r23 – r24
        add     0x30,         r23               -- r23 = 0x30 + r23
        st.b    r23,          7[r21]            -- x seconds
        movhi   hi1(disp_flg), r0, r21          -- r21 := watch LCD display status flag
```

```
        movea   lo(disp_flg), r21, r21
        set1    0,            0[r21]          -- disp_flg ON
wt_exit:
        ## Recovers from EIPC and EIPSW stacks
        di                                    -- Prohibits interrupts
        ld.w    0[r3],        r21
        ldsr    r21,          1               -- Recovery of EIPSW
        ld.w    4[r3],        r21
        ldsr    r21,          0               -- Recovery of EIPC
        ld.w    8[r3],        r1              -- Recovery of r1
        add     12,           r3
        reti

_int_tm00:
        mov     r1,           r6              -- Saves r1
tm_countup:
        movhi   hi1(tm0_num), r0, r16         -- r16 := timer 0 counter
        movea   lo(tm0_num), r16, r16
        ld.w    0[r16],       r17             -- Loads tm0_num to register
        addi    1,     r17,   r17             -- Increments tm0_num
        st.w    r17,          0[r16]          -- Stores tm0_num to memory
tm_chk_flg:
        movhi   hi1(disp_flg), r0, r16        -- r16 := watch LCD display status flag
        movea   lo(disp_flg), r16, r16
        tst1    0,            0[r16]          -- Checks disp_flg
        bz      tm_led                        -- Jumps if dsp_flg = 0
tm_disp_lcd:
        movhi   hi1(col_ctr), r0, r16         -- r16 := No. of display columns in watch display data
        movea   lo(col_ctr), r16, r16
        ld.b    0[r16],       r17             -- Loads col_ctr to register
        mov     0x07,         r18             -- r18 := 7
        cmp     r17,          r18
        bn      tm_cursol_home                -- Jumps if col_ctr = 7 or more

        movhi   hi1(tm_buf), r0, r18          -- r17 := display watch data
        movea   lo(tm_buf), r18, r18
        add     r17,          r18
        ld.b    0[r18],       r20             -- r20 := display character

        movhi   hi1(lcd_addr), r0, r19        -- r19 := LCD address
        movea   lo(lcd_addr), r19, r19
        st.b    r20,          2[r19]
        addi    1,    r17,    r17             -- Increments col_ctr
        st.b    r17,          0[r16]          -- Stores col_ctr to memory
        br      tm_led                        -- Goes to LED processing
tm_cursol_home:
        movhi   hi1(lcd_addr), r0, r19        -- r19 := LCD address
        movea   lo(lcd_addr), r19, r19
        mov     0x02,         r20
        st.b    r20,          0[r19]          -- LCD home position
        movhi   hi1(disp_flg), r0, r16        -- r16 := watch LCD display status flag
        movea   lo(disp_flg), r16, r16
        clr1    0,            0[r16]          -- disp_flg OFF
        movhi   hi1(col_ctr), r0, r16         -- r16 := No. of display columns in watch display data
        movea   lo(col_ctr), r16, r16
        st.b    r0,           0[r16]          -- col_ctr = 0
        br      tm_led                        -- Goes to LED processing
tm_led:
        ## Increment and activate LED at 25.6-ms interval
        movhi   hi1(led_ivl), r0, r16         -- r16 := LED display interval address
        movea   lo(led_ivl), r16, r16
        ld.w    0[r16],       r17             -- r17 := LED display interval
        addi    0x01,   r17,  r17             -- Increments LED display interval
        st.w    r17,          0[r16]          -- Stores incremented data to memory
        mov     r17,          r16             -- r16 := r17
        movea   0xff,   r0,   r18             -- r18 := 0xff
        and     r18,          r17
        bnz     tm_exit                       -- Jumps if lower 8 bits of r17 is not "0"
led_disp:
```

```
        shr     8,          r16         -- Calculates 15th bit based on
        and     0xff,       r16         -- 8th bit of r16
        mov     r16,        r17         -- r17 := r16
        and     0x0f,       r16         -- Outputs lower 4 bits of
        shl     4,          r16         -- display data to higher 4 bits of
        st.b    r16,        P3[r0]      -- port 3
        and     0xf0,       r17         -- Outputs higher 4 bits of
        shr     4,          r17         -- display data to lower 4 bits of
        st.b    r17,        P11[r0]     -- port 11
tm_exit:
        mov     r6,         r1          -- Recovery of r1
        reti
```

4.5 TB-V850/SA1 Circuit Diagrams

Circuit diagrams of the TB-V850/SA1 are shown below.

(1) CPU and toggle switches

This circuit mainly includes the V850/SA1, an oscillator, resonator, power backup circuit, serial EEPROM, and 8 toggle switches.

- NM24C02LM: 2 Kb serial EEPROM (I^2C bus interface)

(2) Device control circuit

This circuit mainly includes an FMP7128SLC84-10 and a fast page DRAM.

- FMP7128SLC84-10: FPGA (Field Programmable Gate Array)
  Various signals related to the CPU's bus interface are input to this device, which generates memory control signals. The FPGA data is written via a connector (CN1).

(3) Memory

This circuit mainly includes ROM, SRAM, and LCD connectors.

- 74FCT16245: 16-bit bidirectional buffer
- 74FCT16373: 16-bit transparent latch
- 74FCT244: 8-bit buffer

Caution The LCD (L1652) is connected via the CN2 connector.

(4) I/O interface

This circuit mainly includes a power supply monitor circuit, a flash programmer interface, RS-232C interface, and 8 LEDs.

- MAX232: RS-232C driver/receiver
- TL7700: Power supply monitor device

Caution The V850/SA1 dedicated flash programmer is connected to the CN7 connector.

(5) Analog I/O

This circuit mainly includes a speaker interface, variable resistor, and a DC motor interface.

- BLA8-103J: Ladder resistor (R-2R circuit)

Cautions 1. The speaker is connected to the CN5 connector.
2. The DC motor is connected to the CN6 connector.

[MEMO]

Figure 4-4. CPU and Toggle Switches

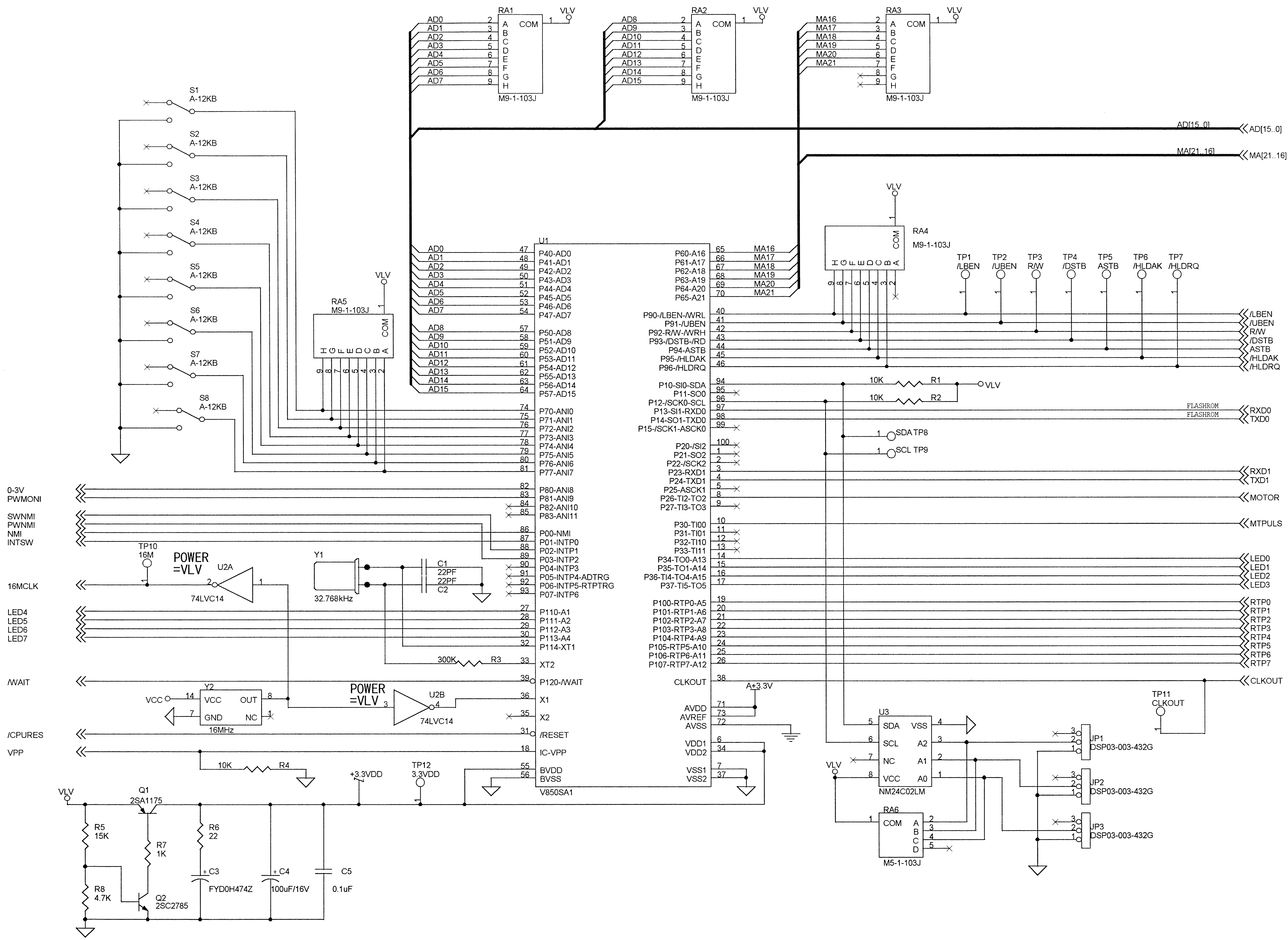


Figure 4-5. Device Controller

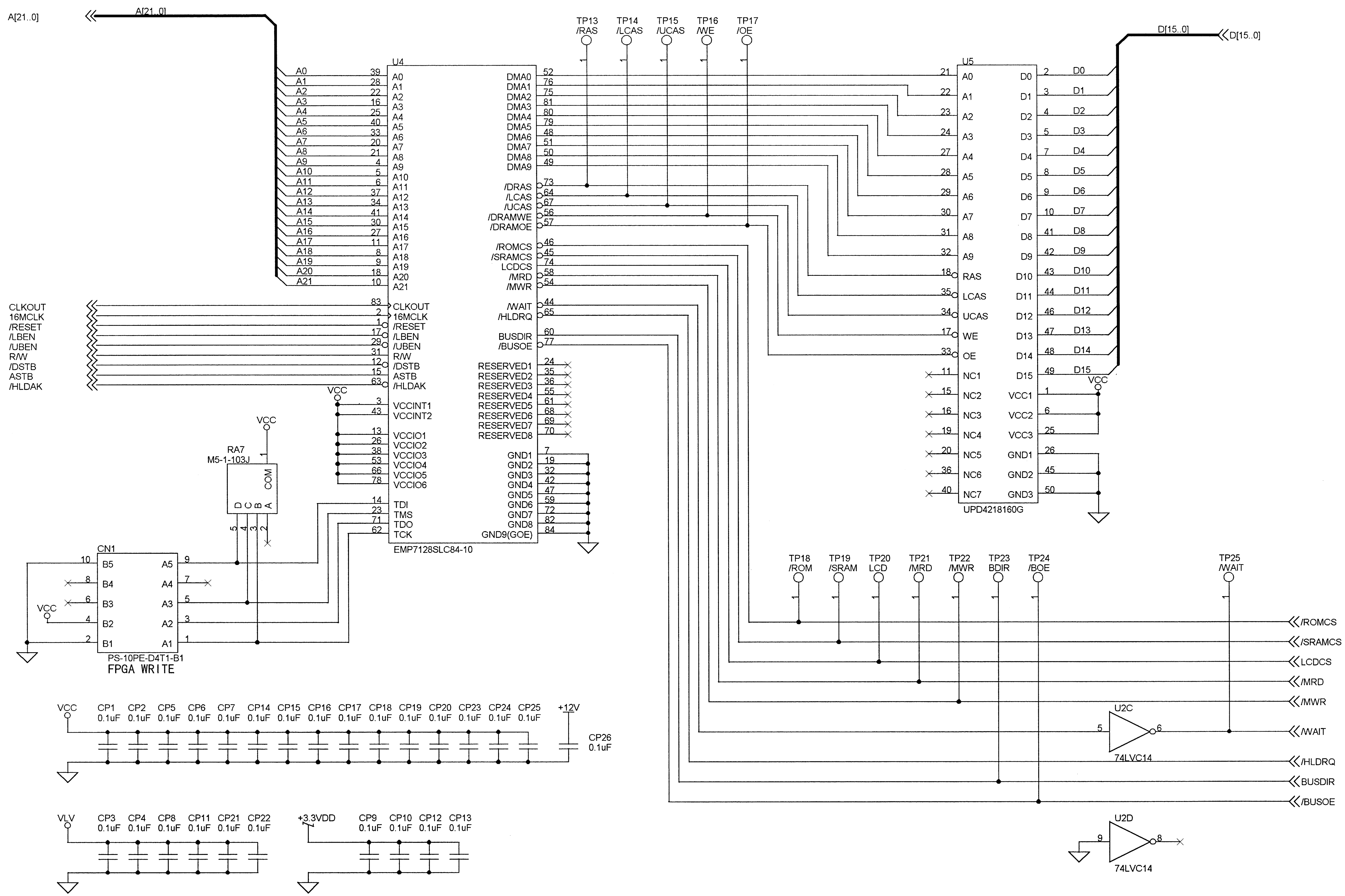


Figure 4-6. Memory

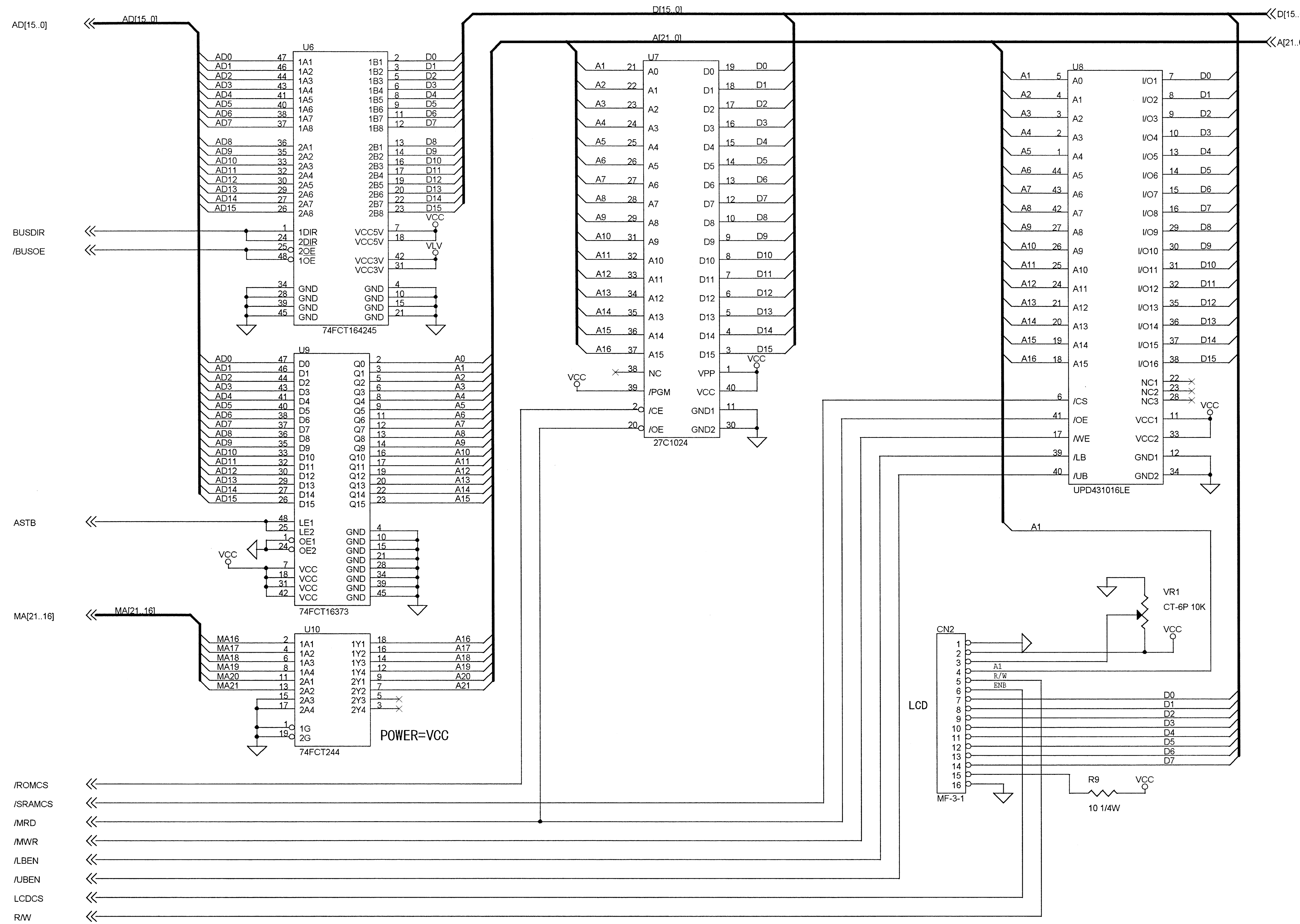


Figure 4-7. I/O Interface

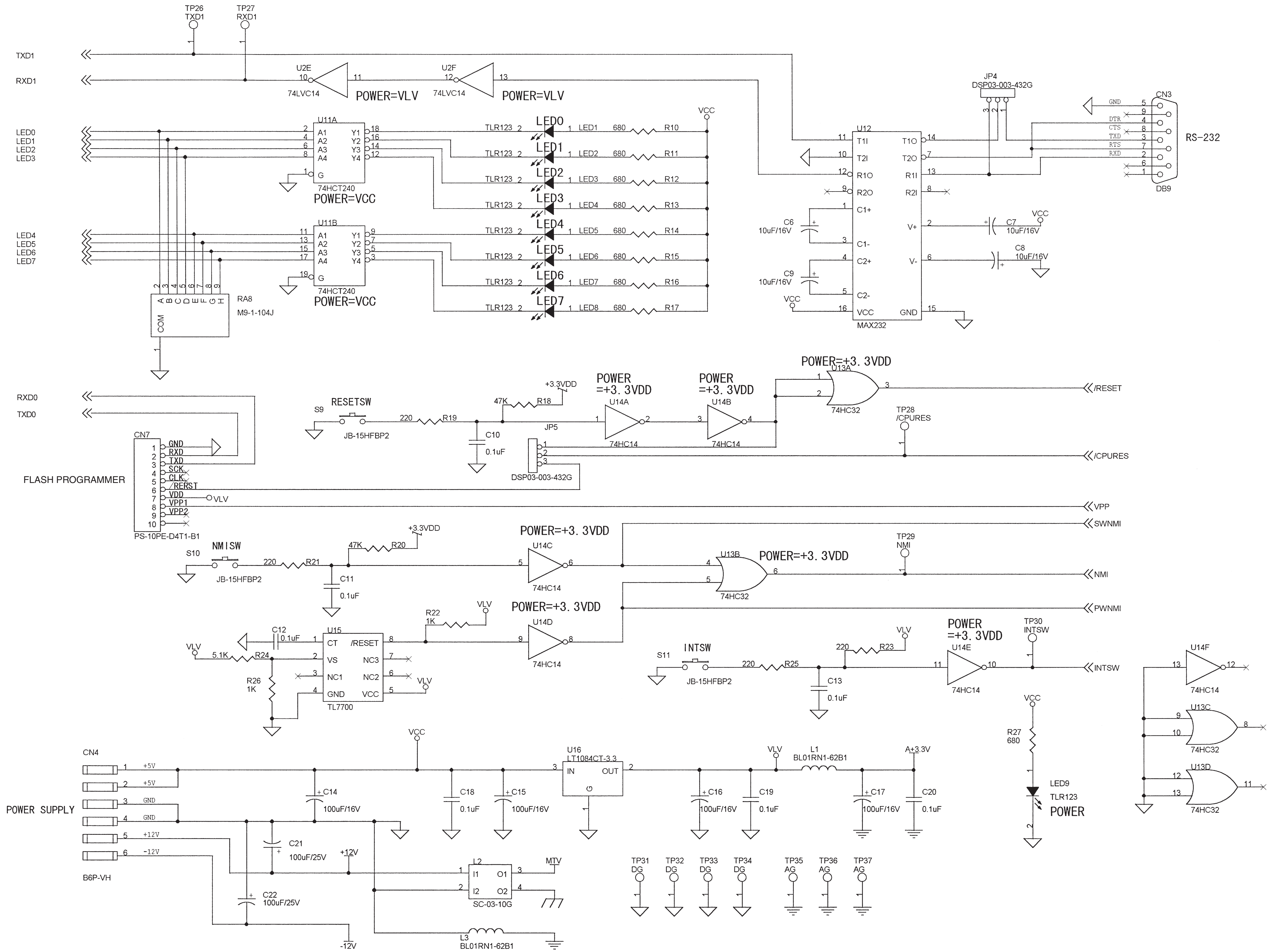
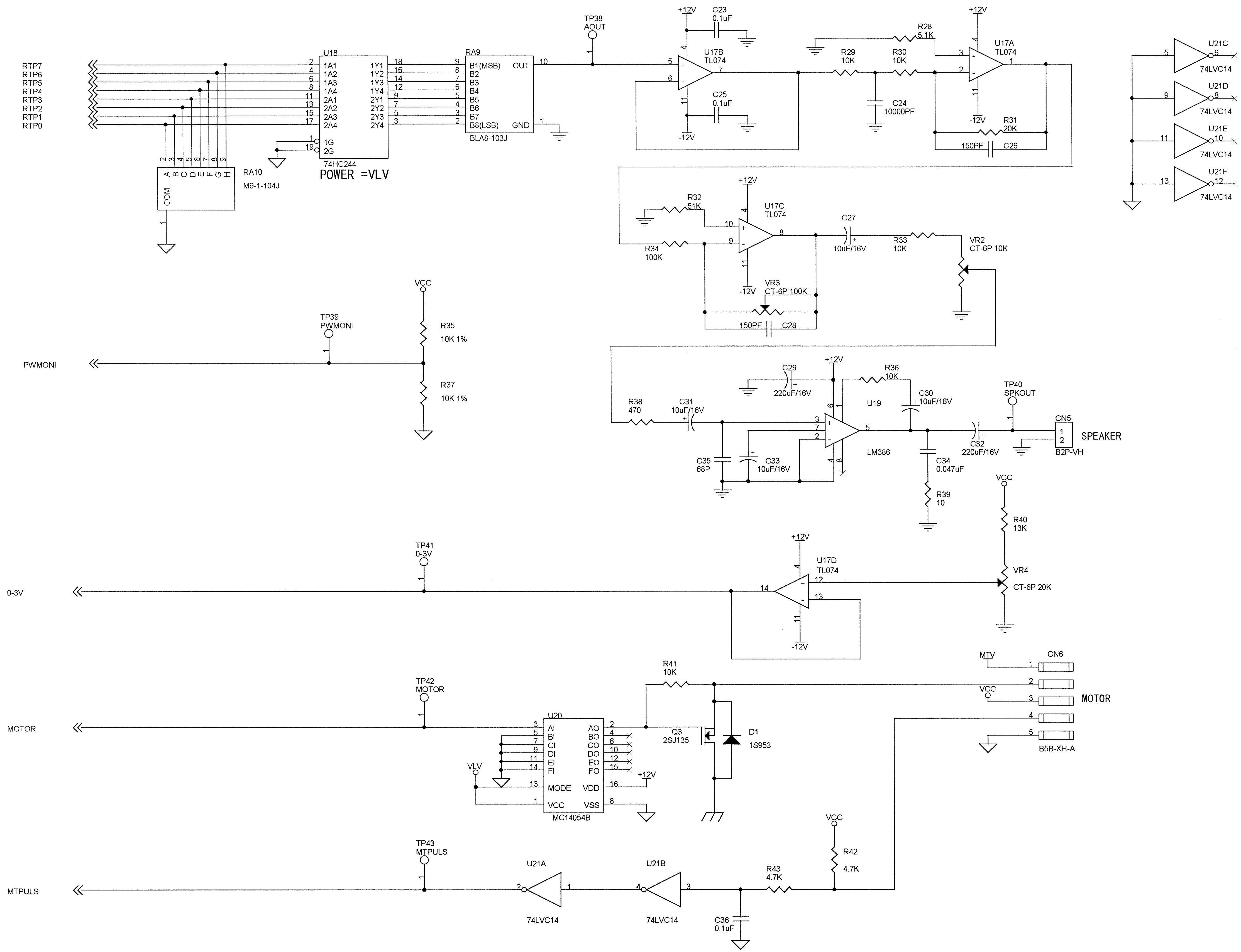


Figure 4-8. Analog I/O



APPENDIX A INDEX

APPENDIX B REVISION HISTORY

Edition	Description	Chapter
2nd edition	Addition of μPD703014B, 703014BY, 703015B, 703015BY, 70F3015B, 70F3015BY	Throughout
	Addition of **Table 1-1 List of V850/SA1 Products**	**CHAPTER 1 GENERAL DESCRIPTION OF V850/SA1**
	Addition of description to Minimum instruction execution time in **1.2 Features**	
	Deletion of description from **1.6.2 (2) Bus control unit (BCU)**	
	Change of purpose of r2 in **1.8 CPU Register Set**	
	Change of purpose and operation of r2 in **1.8.1 (1) General-purpose registers**	
	Addition of flash memory programming mode control register (FLPMC) in **1.13 Peripheral I/O Registers**	
	Addition of **Notes** in **1.13 Peripheral I/O Registers**	
	Deletion of **2.1.1 Connection to BV_{DD} pin**	**CHAPTER 2 BUS INTERFACE CONNECTION CIRCUIT EXAMPLES**

[MEMO]

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	❏	❏	❏	❏