To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

V850E/ME2, V850E2/ME3

32-bit Single-Chip Microcontrollers

Hardware

V850E/ME2:
μPD703111A
V850E2/ME3:
μPD703500

Document No. U16794EJ3V1AN00 (3rd edition)
Date Published June 2006 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① VOLTAGE APPLICATION WAVEFORM AT INPUT PIN

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (MAX) and V_{IH} (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (MAX) and V_{IH} (MIN).

② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

④ STATUS BEFORE INITIALIZATION

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

⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

- **The information in this document is current as of March, 2006. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

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

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

INTRODUCTION

Readers

This application note is intended for users who wish to understand the functions of the V850E/ME2 and V850E2/ME3 to design application systems using the V850E/ME2 or V850E2/ME3. The following shows the target products.

- V850E/ME2: μPD703111A
- V850E2/ME3: μPD703500

Purpose

The purpose of this application note is to help the user understand the composition of a system that uses the V850E/ME2, using the CPU board TB-V850E/ME2 as an example. The circuit configuration of a system using the V850E2/ME3 is basically the same as that shown in this Application Note.

Organization

This application note is broadly divided into the following sections.

- Introduction
- Bus interface connection circuit example
- Connection circuit example of on-chip peripheral functions
- Application examples

How to Read This Manual

It is assumed that the readers of this application note have general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

- To know the hardware functions and electrical specifications of the V850E/ME2 and V850E2/ME3
  → Refer to the **V850E/ME2 Hardware User's Manual** or **V850E2/ME3 Hardware User's Manual** (separately provided).

- To know the instruction functions of the V850E/ME2 and V850E2/ME3
  → Refer to the **V850E1 Architecture User's Manual** or **V850E2 Architecture User's Manual** (separately provided).

The mark <R> shows major revised points.
The revised points can be easily searched by entering "<R>" into the "Find What:" field.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{xxx}$ (overscore over pin or signal name) or /xxx ("/" before signal name)
Memory map address:	Top: higher, bottom: lower
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary ... xxxx or xxxxB Decimal ... xxxx Hexadecimal ... xxxxH
Prefix indicating power of 2 (address space, memory capacity):	K (kilo) ... 2^{10} = 1,024 M (mega) ... 2^{20} = $1,024^2$ G (giga) ... 2^{30} = $1,024^3$

Related documents

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

Documents related to V850E/ME2 and V850E2/ME3

Document Name	Document No.
V850E1 Architecture User's Manual	U14559E
V850E/ME2 Hardware User's Manual	U16031E
V850E/ME2 Hardware Application Note	This manual
V850E/ME2 USB Function Drivers Application Note	U17069E
V850E2 Architecture User's Manual	U17135E
V850E2/ME3 Hardware User's Manual	U17126E
V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2 PCI Host Bridge Macro Application Note	U17121E

Documents related to development tools (user's manuals)

Product Name	Document Name		Document No.
V850E/ME2 only	CA850 (Ver. 3.00) (C Compiler Package)	Operation	U17293E
		C Language	U17291E
		Assembly Language	U17292E
		Link Directives	U17294E
	PM+ (Ver. 6.00) (Project Manager)		U17178E
	ID850 (Ver. 3.00) (Integrated Debugger)	Operation	U17358E
	ID850NW (Ver. 3.00, 3.10) (Integrated Debugger)	Operation	U17369E
	TW850 (Ver. 2.00) (Performance Analysis Tuning Tool)		U17241E
V850E/ME2, V850E2/ME3	RX850 (Ver. 3.20) (Real-Time OS)	Basics	U13430E
		Installation	U17419E
		Technical	U13431E
		Task Debugger	U17420E
	RX850 Pro (Ver. 3.20) (Real-Time OS)	Basics	U13773E
		Installation	U17421E
		Technical	U13772E
		Task Debugger	U17422E
	AZ850 (Ver. 3.30) (System Performance Analyzer)		U17423E

CONTENTS

CHAPTER 1 INTRODUCTION

The V850E/ME2 and V850E2/ME3 are products of the NEC Electronics single-chip microcontroller “V850 Series”.

1.1 Outline

<R> The V850E/ME2 and V850E2/ME3 are 32-bit single-chip microcontrollers that use the reinforced performances of the V850E1 CPU of the V850 Series; have many peripheral functions such as cache data RAM, instruction RAM, memory controllers, a DMA controller, timers/counters, a serial interface, a USB function controller (USBF), and an A/D converter; and realize processing of a vast amount of data and state-of-the-art real-time control features.

Table 1-1 shows the differences between the V850E/ME2 and V850E2/ME3.

<R> **Table 1-1. Differences Between V850E/ME2 and V850E2/ME3 (1/2)**

Item		V850E/ME2	V850E2/ME3
Number of instructions		80	89
Minimum instruction execution time		6.5 ns (at internal 150 MHz)	5.0 ns (at internal 200 MHz)
Boot address		0x100000H (external memory)	0x000000H (external memory)
Memory space	Linear address space	256 MB (program: 64 MB/data: 256 MB)	512 MB (program: 512 MB/data: 512 MB)
	Memory block division function	2, 4, 6, 8, 64 MB/block	2, 64 MB/block
External bus interface	External bus division function	Division ratio of BUSCLK to internal system clock (f_{CLK}): 1/1, 1/2, 1/3, 1/4 (66 MHz MAX.)	Division ratio of BUSCLK to VBCLK (f_{VBCLK}): 1/1, 1/2, 1/4 (66 MHz MAX.)
	Speculative read function	Provided	None
	Endian function	Provided	None
Instruction RAM	Capacity	128 KB (64 KB × 2 banks)	168 KB (168 KB × 1 bank)
	Selection of read/write mode	IRAMM register	IRWE register
Data RAM		16 KB	32 KB
Instruction cache	Capacity	8 KB (2-way set associative)	8 KB (4-way set associative)
	Way lock function	Provided (Only 1 way can be locked.)	None
Data cache		None	Provided 8 KB (4-way set associative)
Interrupt/ exception	Mask function after reset release	Provided (NMI input checked by NRS register)	None

Remark For details, refer to the hardware user’s manual of each product.

<R> **Table 1-1. Differences Between V850E/ME2 and V850E2/ME3 (2/2)**

Item			V850E/ME2	V850E2/ME3
DMA controller	Maximum number of times of transfer		65,536 (2^{16})	16,777,216 (2^{24})
	Setting of single-step transfer mode		DADCn register	DADCn, DSMC registers
	DRST register		None	Provided
	Priority of DMA channel		Fixed mode	Fixed mode/round-robin mode
	Mask function of $\overline{\text{DMARQn}}$ signal		Fixed to 3 clocks	0 to 15 clocks selectable
	Active width expansion function of $\overline{\text{DMAAKn}}$ signal		4, 6, or 7 clocks selectable	0 to 15 clocks selectable
Timers	TMC0 to TMC5		Provided	Provided
	TMD0 to TMD3		Provided	Provided
	TMENC10, TMENC11		Provided (number of noise elimination clocks: 0, 2, 3, 5)	Provided (number of noise elimination clocks: 0, 2, 3, 7)
Serial interface	UARTB0, UARTB1		Provided	Provided
	CSI30, CSI31		Provided (transfer rate: 5.5 Mbps max.)	Provided (transfer rate: 6.25 Mbps max.)
	USB function controller		Provided	Provided
Clock generator	SSCG output clock		Fixed to 8 times	Fixed to 20 times
	Memory controller		BUSCLK operation	VBCLK operation (division ratio (1/2 or 1/3) set by VBCSEL pin)
	OSTS register		Provided	None
Power save function			HALT/IDLE/software STOP mode	HALT/IDLE mode
Pin assignment	Pin number	1	JIT1 pin	EV_{SS} pin
		2	JIT0 pin	EV_{SS} pin
		156	PLLSEL pin	VBCSEL pin
Package			176-pin plastic LQFP (thickness: 1.4 mm)	176-pin plastic QFP (thickness: 2.7 mm)

Remarks 1. For details, refer to the hardware user's manual of each product.

2. n = 0 to 3

1.2 V850E/ME2

1.2.1 Features

<R> ○ Number of instructions: 80

○ Minimum instruction execution time: 10 ns/7.5 ns/6.7 ns (at internal 100 MHz/133 MHz/150 MHz operation)

○ General-purpose registers: 32 bits × 32

○ Instruction set: V850E1 CPU
Signed multiplication (16 bits × 16 bits → 32 bits or 32 bits × 32 bits → 64 bits): 1 to 2 clocks
Saturated operation instructions (with overflow/underflow detection function)
32-bit shift instructions: 1 clock
Bit manipulation instructions
Load/store instructions with long/short format
Signed load instructions

○ Memory space: 256 MB linear address space (common program/data use)
Chip select output function: 8 spaces
Memory block division function: 2, 4, 6, 8, 64 MB/block
Programmable wait function
Idle state insertion function

○ External bus interface: 32-bit data bus (address/data separated)
32-/16-/8-bit bus sizing function
External bus division function: 1/1, 1/2, 1/3, 1/4 (66 MHz MAX.)
Bus hold function
External wait function
Address setup wait function
Endian control function

○ Internal memory: Instruction RAM: 128 KB
Data RAM: 16 KB

○ Instruction cache: 8 KB 2-way set associative

○ Interrupts/exceptions: External interrupts: 40 (including NMI)
Internal interrupts: 59 sources
Exceptions: 2 sources
Eight levels of priorities can be set.

○ Memory access controller: DRAM controller (compatible with SDRAM)
Page ROM controller
Speculative read/write buffer function

○ DMA controller:
4 channels
Transfer unit: 8 bits/16 bits/32 bits
Maximum transfer count: 65,536 (2^{16})
Transfer type: Flyby (1-cycle)/2-cycle
Transfer mode: Single/Single step/Block
Transfer target: Memory ↔ memory, memory ↔ I/O
Transfer request: External request/On-chip peripheral I/O/ Software
DMA transfer terminate (terminal count) output signal
Next address setting function

○ I/O lines:
Input ports: 1
I/O ports: 77

○ Timer functions:
16-bit timer/event counter: 6 channels (no capture operation for 2 channels)
16-bit timers: 6
16-bit capture/compare registers: 12
16-bit interval timer: 4 channels
16-bit up/down counter/general-purpose timer for
2-phase encoder input: 2 channels
16-bit capture/compare registers: 4
16-bit compare registers: 4

○ Serial interfaces:
Asynchronous serial interface B (UARTB)
Clocked serial interface 3 (CSI3)
CSI3/UARTB: 1 channel
UARTB: 1 channel
CSI3: 1 channel
USB function controller (USBF): 1 channel
Full speed (12 Mbps)
Endpoint Control transfer: 64 bytes × 2
Interrupt transfer: 8 bytes × 2
Bulk transfer (IN): 64 bytes × 2 banks × 2
Bulk transfer (OUT): 64 bytes × 2 banks × 2

○ A/D converter:
10-bit resolution A/D converter: 8 channels

○ PWM (Pulse Width Modulation):
16-bit resolution PWM: 2 channels

○ Clock generator:
×8 function using SSCG

○ Power-save function:
HALT/IDLE/software STOP mode

<R> ○ Package:
176-pin plastic LQFP (fine pitch) (24 × 24)

○ CMOS technology:
Fully static circuits

<R> 1.2.2 Ordering information

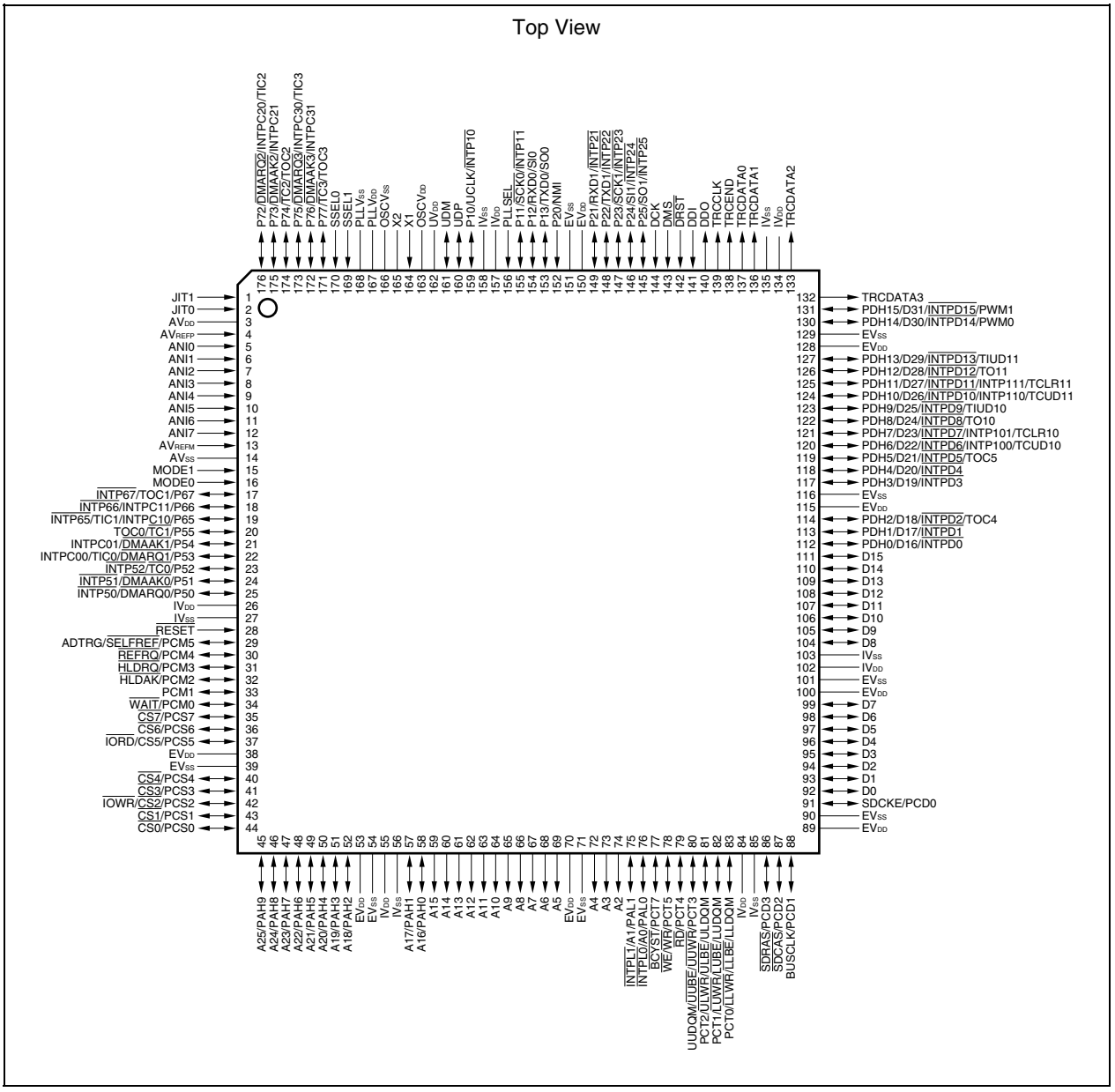
Part Number	Package	Maximum Operating Frequency
μPD703111AGM-10-UEU	176-pin plastic LQFP (fine pitch) (24 × 24)	100 MHz
μPD703111AGM-10-UEU-A	176-pin plastic LQFP (fine pitch) (24 × 24)	100 MHz
μPD703111AGM-13-UEU	176-pin plastic LQFP (fine pitch) (24 × 24)	133 MHz
μPD703111AGM-13-UEU-A	176-pin plastic LQFP (fine pitch) (24 × 24)	133 MHz
μPD703111AGM-15-UEU	176-pin plastic LQFP (fine pitch) (24 × 24)	150 MHz
μPD703111AGM-15-UEU-A	176-pin plastic LQFP (fine pitch) (24 × 24)	150 MHz

Remark Products with -A at the end of the part number are lead-free products.

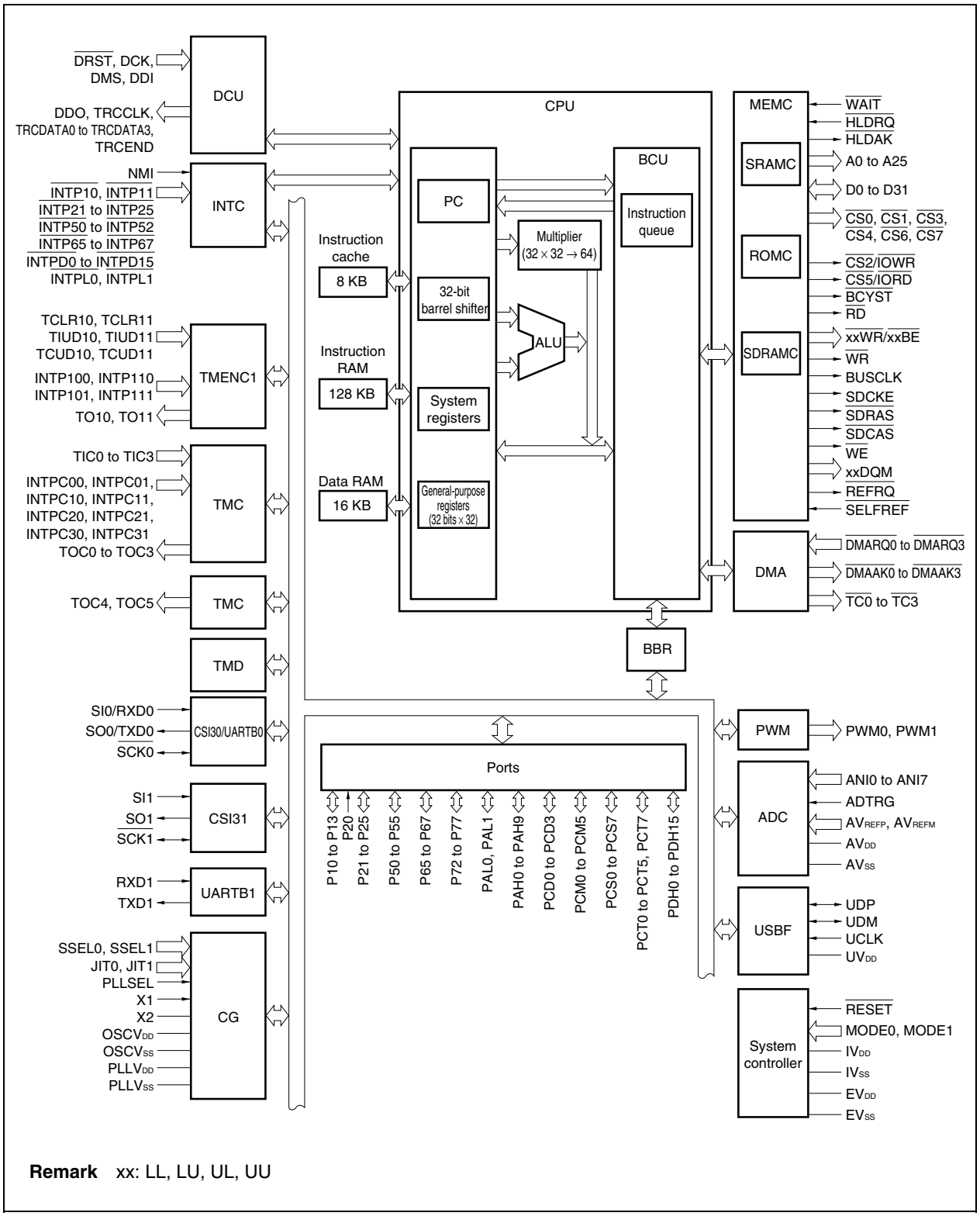
<R> **1.2.3 Pin configuration**

- 176-pin plastic LQFP (fine pitch) (24 × 24)
  μPD703111AGM-10-UEU
  μPD703111AGM-10-UEU-A
  μPD703111AGM-13-UEU
  μPD703111AGM-13-UEU-A
  μPD703111AGM-15-UEU
  μPD703111AGM-15-UEU-A

Top View



1.2.4 Internal block diagram



Remark xx: LL, LU, UL, UU

1.2.5 Internal memory

The V850E/ME2 has a 128 KB (64 KB × 2 banks) instruction memory, 8 KB (2-way set associative) instruction cache, and 16 KB data RAM.

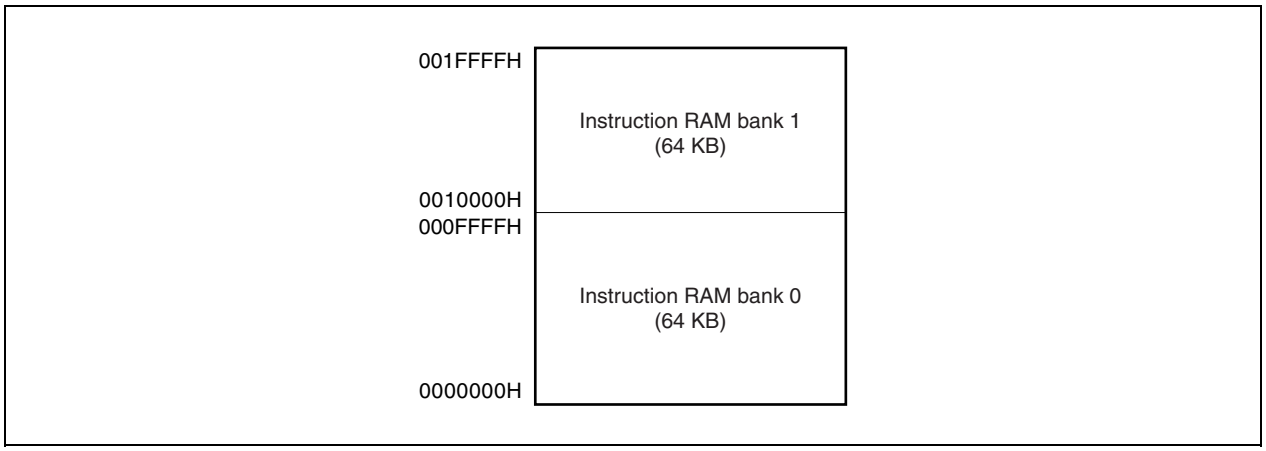
(1) Internal instruction RAM

The internal instruction RAM has two modes: read mode and write mode. These modes are selected by the internal instruction RAM mode register (IRAMM).

After reset, the instruction RAM is initialized to the write mode. Therefore, the read mode is set after instruction data is transferred to the internal instruction RAM by program or the DMA controller. In the read mode, an instruction is fetched in one internal system clock.

Caution All interrupt/exception handlers, except the reset handler, are in bank 0 of the internal instruction RAM. Do not generate any interrupt/exception until a write operation to this bank is completed.

Figure 1-1. Memory Map of Internal Instruction RAM



(2) Instruction cache function

The $\overline{CS0}$, $\overline{CS1}$, and $\overline{CS2}$ spaces are cacheable areas. It can be specified by the cache configuration register (BHC) whether each space is used as a cacheable area or uncacheable area. The cache lock status of way 0, auto fill of way 0, and tag clear of ways 0 and 1 are specified by using the instruction cache control register (ICC).

Cautions 1. Write the BHC register after reset. After writing a value to this register, do not change it.

2. In an ordinary system, memories located in the $\overline{CS0}$, $\overline{CS1}$, and $\overline{CS2}$ spaces are used as cacheable areas. In a system where a program is downloaded by a boot program, they are set as cacheable areas after downloading is completed.

An area where the instruction that sets the BHC register exists cannot be changed from an uncacheable area to cacheable area, or vice versa.

To set such an area as a cacheable area, use an uncacheable area in which this instruction does not exist, or the internal instruction RAM area.

(3) Internal data RAM

The internal data RAM area is allocated to the 16 KB area of addresses FFFB000H to FFFEFFFH. Instruction codes cannot be allocated to this area.

1.2.6 Speculative read function (read buffer function)

The V850E/ME2 has a 4-word (128-bit) read buffer used for a speculative read function to enable high-speed CPU processing. The speculative timing can be set for each $\overline{CSn}$ space by line buffer control registers 0 and 1 (LBC0 and LBC1) (n = 0 to 7).

Caution Generally, use of the speculative read function is prohibited for units that are not located at contiguous addresses (such as I/O devices), or memory whose contents change asynchronously to the CPU (such as a dual-port memory that is written by another bus master).

1.2.7 Initialization pins

The V850E/ME2 has initialization pins that set various operation modes.

(1) MODE0 and MODE1 pins

The operation mode is specified according to the status of the MODE0 and MODE1 pins. In an application system, fix the specification of these pins and do not change them during operation. Operation is not guaranteed if these pins are changed during operation.

Table 1-2. Setting of Data Bus

MODE1	MODE0	Operating Mode	
L	L	Normal operation mode	32-bit data bus
L	H		16-bit data bus
Other than above		Setting prohibited	

Remark L: Low-level input
H: High-level input

(2) PLLSEL, SSEL0, and SSEL1 pins

These input pins are set according to the frequency (Fx) input to the X1 and X2 pins. Set the PLLSEL, SSEL0, and SSEL1 pins in accordance with the value of Fx × 8 = fx (main clock).

Table 1-3. Frequency List

Multiplication Factor	PLLSEL Pin	SSEL1 Pin	SSEL0 Pin	Input Frequency (MHz) (Target Value)	Main Clock (fx) Frequency (MHz)
8	H	L	H	Setting prohibited	Setting prohibited
		H	L	10.00 to 10.19	80.00 to 81.59
		H	H	10.20 to 11.99	81.60 to 95.99
	L	L	L	12.00 to 14.39	96.00 to 115.19
		L	H	14.40 to 17.39	115.20 to 139.19
		H	L	17.40 to 18.75	139.20 to 150.00
		H	H	Setting prohibited	Setting prohibited

Caution The maximum value of f_{CLK} is 100 MHz in a 100 MHz product, 133 MHz in a 133 MHz product, and 150 MHz in a 150 MHz product.
The operation is not guaranteed if f_{CLK} (MAX.) < fx.
Make sure that fx does not exceed the guaranteed operating frequency of each product.

Remark L: Low-level input
H: High-level input

(3) JIT0 and JIT1 pins

These input pins specify the frequency modulation rate (f_{DIT}) of SSCG output. The default values (after reset) of the ADJON and ADJ2 to ADJ0 bits of the SSCG control register (SSCGC) are changed as follows, depending on the setting of these pins.

Table 1-4. Default Values of SSCGC.ADJON and SSCGC.ADJ2 to SSCGC.ADJ0 Bits

JIT1 Pin	JIT0 Pin	Default Value			
		ADJON Bit	ADJ2 Bit	ADJ1 Bit	ADJ0 Bit
L	L	0	0	0	0
L	H	1	0	0	1
H	L	1	0	1	1
H	H	1	1	0	1

Remark L: Low-level input
H: High-level input

<R> 1.3 V850E2/ME3

1.3.1 Features

- Number of instructions: 89
- Minimum instruction execution time: 5.0 ns (at internal 200 MHz operation)
- General-purpose registers: 32 bits × 32
- Instruction set:
  V850E2 CPU
  Signed multiplication (16 bits × 16 bits → 32 bits or 32 bits × 32 bits → 64 bits): 1 clock
  Saturated operation instructions (with overflow/underflow detection function)
  32-bit shift instructions: 1 clock
  Bit manipulation instructions
  Load/store instructions with long/short format
  Signed load instructions
  Multiply and accumulation function (MAC) (32 bits × 32 bits + 64 bits → 64 bits)
- Memory space:
  512 MB linear address space (common program/data use)
  Chip select output function: 8 spaces
  Memory block division function: 2, 64 MB/block
  Programmable wait function
  Idle state insertion function
- External bus interface:
  32-bit data bus (address/data separated)
  32-/16-/8-bit bus sizing function
  External bus division function: 1/1, 1/2, 1/4 (66 MHz MAX.)
  Bus hold function
  External wait function
  Address setup wait function
- Internal memory:
  Instruction RAM: 168 KB
  Data RAM: 32 KB
- Instruction cache: 8 KB 4-way set associative
- Data cache: 8 KB 4-way set associative
- Interrupts/exceptions:
  External interrupts: 40 (including NMI)
  Internal interrupts: 59 sources
  Exceptions: 2 sources
  Eight levels of priorities can be set.
- Memory access controller:
  DRAM controller (compatible with SDRAM)
  Page ROM controller

- DMA controller:
  - 4 channels
  - Transfer unit: 8 bits/16 bits/32 bits
  - Maximum transfer count: 16,777,216 (2^{24})
  - Transfer type: Flyby (1-cycle)/2-cycle
  - Transfer mode: Single/Single step/Block
  - Transfer target: Memory ↔ memory, memory ↔ I/O
  - Transfer request: External request/On-chip peripheral I/O/ Software
  - DMA transfer terminate (terminal count) output signal
  - Next address setting function
  - Priority of DMA channel: Priority fixed mode/Round robin mode
- I/O lines:
  - Input ports: 1
  - I/O ports: 77
- Timer functions:
  - 16-bit timer/event counter: 6 channels (no capture operation for 2 channels)
    - 16-bit timers: 6
    - 16-bit capture/compare registers: 12
  - 16-bit interval timer: 4 channels
  - 16-bit up/down counter/timer for 2-phase encoder input: 2 channels
    - 16-bit capture/compare registers: 4
    - 16-bit compare registers: 4
- Serial interfaces (SIO):
  - Asynchronous serial interface B (UARTB)
  - Clocked serial interface 3 (CSI3)
    - CSI3/UARTB: 1 channel
    - UARTB: 1 channel
    - CSI3: 1 channel
    - USB function controller (USBF): 1 channel
      - Full speed (12 Mbps)
      - Endpoint Control transfer: 64 bytes × 2
        Interrupt transfer: 8 bytes × 2
        Bulk transfer (IN): 64 bytes × 2 banks × 2
        Bulk transfer (OUT): 64 bytes × 2 banks × 2
- A/D converter: 10-bit resolution A/D converter: 8 channels
- PWM (Pulse Width Modulation): 16-bit resolution PWM: 2 channels
- Clock generator: ×20 function using SSCG
- Power-save function: HALT/IDLE mode
- Package: 176-pin plastic QFP (fine pitch) (24 × 24)
- CMOS technology: Fully static circuits

1.3.2 Ordering information

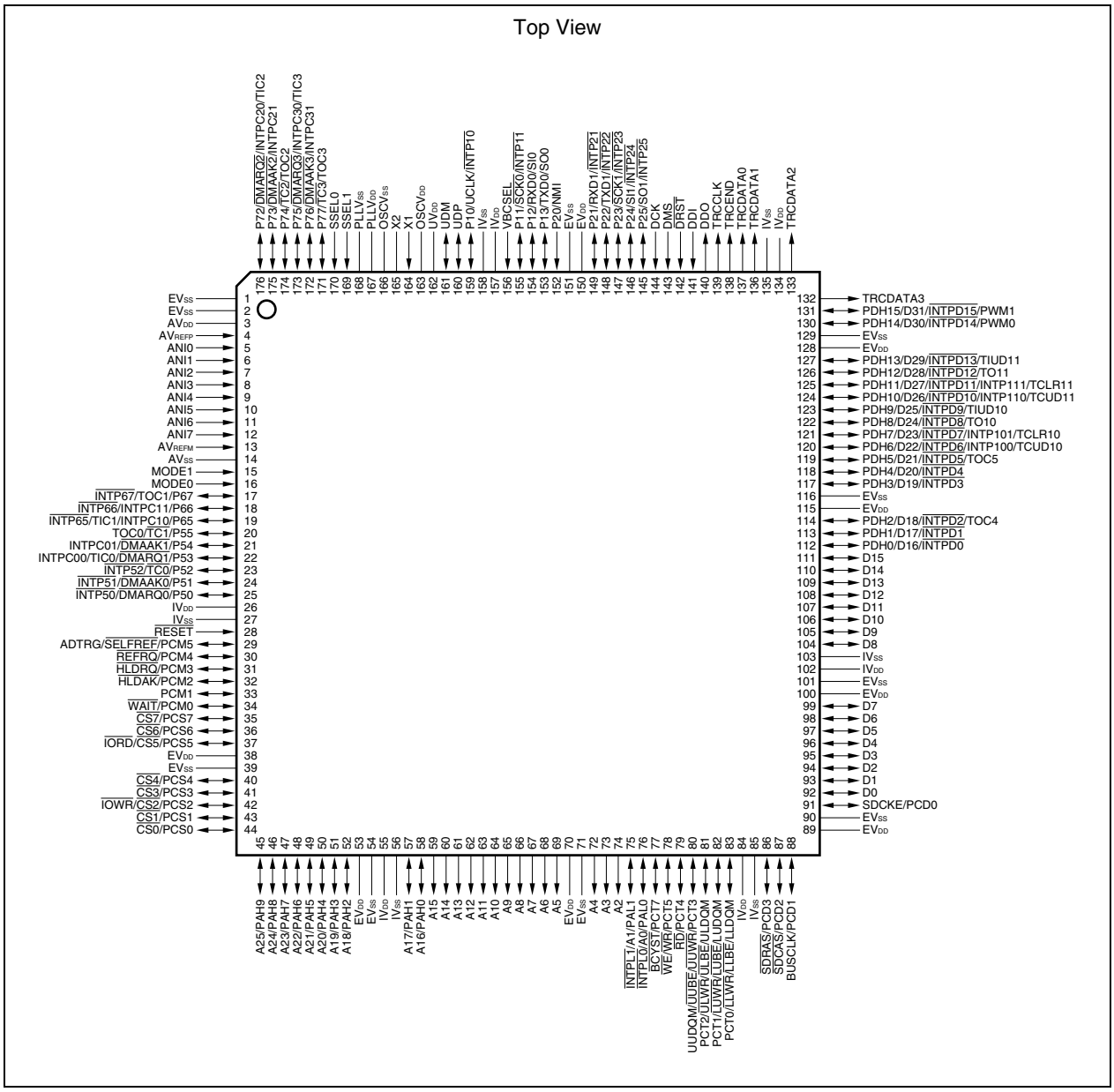
Part Number	Package	Maximum Operating Frequency
μPD703500GM-JEU-A	176-pin plastic QFP (fine pitch) (24 $\times$ 24)	200 MHz

Remark Products with -A at the end of the part number are lead-free products.

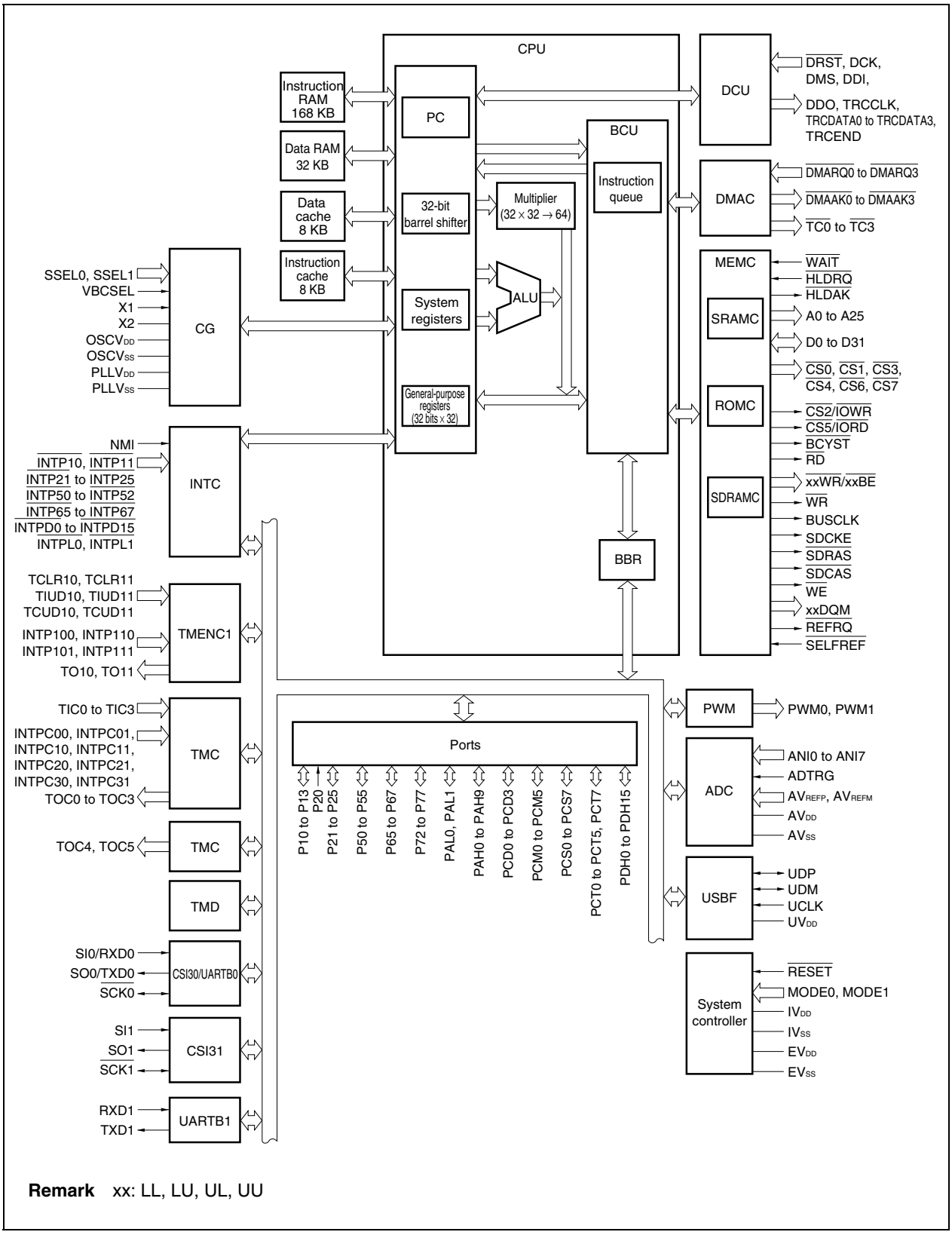
1.3.3 Pin configuration

- 176-pin plastic QFP (fine pitch) (24 × 24)
  μPD703500GM-JEU-A

Top View



1.3.4 Internal block diagram



Remark xx: LL, LU, UL, UU

1.3.5 Internal memory

The V850E2/ME3 has 168 KB (168 KB × 1 bank) of instruction RAM, 8 KB of instruction cache (4-way set associative), 8 KB of data cache (4-way set associative), and 32 KB data RAM.

(1) Internal instruction RAM

The internal instruction RAM is mapped to 00000000H to 00029FFFH.

After reset release, the V850E2/ME3 starts instruction execution from address 00000000H of external memory. By using a program on this external memory, a program is transferred to the internal instruction RAM and the instruction RAM control register (IRC) is set to 1, so that the internal instruction RAM becomes valid.

An internal DMA function is used to transfer a program to the internal instruction RAM. For details, refer to the **V850E2/ME3 Hardware User's Manual**. The internal instruction RAM has a read access/write access mode and a read access-only mode (in which a write access is disabled), which can be selected by the instruction RAM mode register (IRWE).

(2) Instruction cache and data cache functions

All $\overline{CS}$ spaces, including the instruction cache and data cache, are cacheable spaces. Whether each $\overline{CS}$ space is used as a cacheable or non-cacheable area is specified by the cache area specification register (BHC). Operations to cache of each way, Sync if dirty, Way Clear, Fill, and Clear, can be specified by the cache manipulation specification register (COPR).

Cautions 1. The internal instruction RAM area, internal data RAM area, and internal peripheral I/O area are "non-cacheable" regardless of the setting of the BHC register.

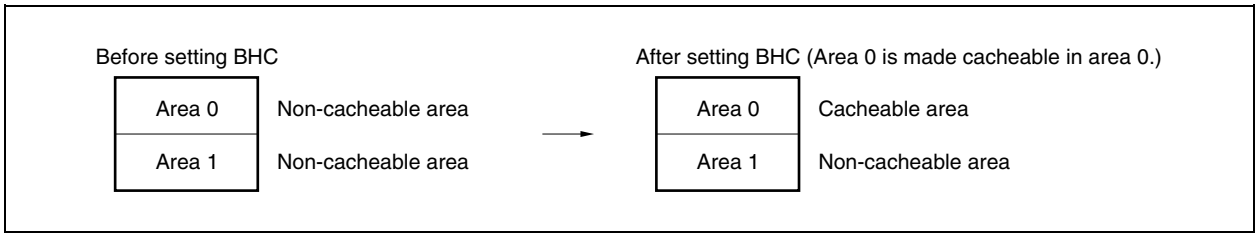
2. To use the instruction cache/data cache, set the bus transaction control register (BTSC) before setting the BHC register.

3. An area where the instruction that sets the BHC register exists cannot be changed from a non-cacheable area to a cacheable area or vice versa (if such a change is made, the operation is not guaranteed). In this case, branch to area 1 and make area 0 cacheable by an instruction in area 1 and, as necessary, branch to area 0 again.

Any $\overline{CS}$ space can be made non-cacheable or cacheable as long as it is in the internal instruction RAM area.

(Example of prohibited setting)

- **If instruction that sets the BHC register is in area 0**



4. The address of the BHC register is the same as the cache configuration register (BHC) of the existing model, V850E/ME2, but the register name, bit names, and bit positions are different.

(3) Internal data RAM

The internal data RAM area is mapped to a 32 KB area of addresses 1FFF7000H to 1FFFEFFFH. Instruction codes cannot be arranged in this area.

1.3.6 Initialization pins

The V850E2/ME3 has initialization pins that set various operation modes.

(1) MODE0 and MODE1 pins

The operation mode is specified according to the status of the MODE0 and MODE1 pins. In an application system, fix the specification of these pins and do not change them during operation. Operation is not guaranteed if these pins are changed during operation.

Table 1-5. Setting of Data Bus

MODE1	MODE0	Operating Mode	
L	L	Normal operation mode	32-bit data bus
L	H		16-bit data bus
Other than above		Setting prohibited	

Remark L: Low-level input
H: High-level input

(2) SSEL0 and SSEL1 pins

These input pins are set according to the frequency (Fx) input to the X1 and X2 pins. Set the SSEL0 and SSEL1 pins in accordance with the value of Fx × 20 = fx (main clock).

Table 1-6. Frequency List

Multiplication Factor	SSEL1 Pin	SSEL0 Pin	Input Frequency (MHz) (Target Value)	Main Clock (fx) Frequency (MHz)
20	0	0	6.00 to 7.24	120.00 to 144.80
	0	1	7.25 to 8.49	145.00 to 169.80
	1	0	8.50 to 10.00	170.00 to 200.00
	1	1	5.00 to 5.99	100.00 to 119.80

(3) VBCSEL pin

This input pin specifies a division ratio of VBCLK to the internal system clock (f_{CLK}).

Table 1-7. Division Ratio Set by VBCSEL Pin

VBCSEL Pin	Division Ratio (f_{VBCLK})	f_{VBCLK} Output Frequency (MHz) (f_{CLK} = 200 MHz)
L	$f_{CLK}/2$	100 MHz
H	$f_{CLK}/3$	66 MHz

Caution The setting of the VBCSEL pin must be fixed in an application system. If it is changed during operation, the operation is not guaranteed.

Remark L: Low-level input
H: High-level input

<R>

CHAPTER 2 EXAMPLES OF BUS INTERFACE CONNECTION CIRCUITS

This chapter shows examples of memory and I/O circuits to be connected to the bus interface of the V850E/ME2 and V850E2/ME3, and explains the settings of the related registers.

The reset handler address is allocated to address 00100000H of block 0 for the V850E/ME2 and to address 00000000H for the V850E2/ME3, and the default status is $\overline{CS0}$ space. For this reason, the V850E/ME2 and V850E2/ME3 are started by an instruction fetch to the $\overline{CS0}$ space. Therefore, ROM (PROM, mask ROM, page ROM, or flash memory) must be located in the $\overline{CS0}$ space.

Of the 256 MB memory space in the V850E/ME2, the 64 MB space of addresses 00000000H to 03FFFFFFH is a program space. In the 192 MB space of addresses 04000000H to 0FFFFFFFH (where $\overline{CS3}$ to $\overline{CS7}$ can be used), data memory and I/O devices are located because program memory cannot be located. All the 512 MB memory space can be used as a program space in the V850E2/ME3. The default data bus width is determined by the setting of the MODE0 and MODE1 pins.

The circuit examples and register settings given in this chapter are based on the following assumptions.

(1) The bus clock of the V850E/ME2 and V850E2/ME3 is 64 MHz and the frequency modulation rate of SSCG output is fixed (no modulation).

<V850E/ME2>

- Setting of the bus mode control register (BMC): 01H (f_{CLK}/2, assuming that the internal system clock (f_{CLK}) is 128 MHz)
- Setting of the ADJON bit of the SSCG control register (SSCGC): 0

<With V850E2/ME3>

- Setting of VBCSEL pin: High level
- Setting of bus mode control register (BMC): 00H (f_{CLK}/3, assuming that the internal system clock (f_{CLK}) is 192 MHz.)
- Setting of the ADJON bit of the SSCG control register (SSCGC): 0

(2) In the circuit connection examples, a 3 V device that can be directly connected to the V850E/ME2 or V850E2/ME3 is used.

Remark Connection with a 5 V device is shown in **2.9 Connection with 5 V Device**.

(3) The bus cycle period control register (BCP) is set to 00H, and the setting of the endian configuration register (BEC) is arbitrary (the BEC register is provided only for the V850E/ME2).

- Operation of $\overline{\text{IORD}}$ and $\overline{\text{IOWR}}$: Prohibited
- Endian: Set as necessary

Remark Examples of using $\overline{\text{IORD}}$ and $\overline{\text{IOWR}}$ are given in **2.10 Connection for DMA Flyby Transfer**.

(4) The setting of line buffer control registers 0 and 1 (LBC0 and LBC1) is arbitrary (the LBC0 and LBC1 registers are provided only for the V850E/ME2).

- Speculative read function: Set as necessary

Remark Refer to **1.2.6 Speculative read function (read buffer function)**. The V850E2/ME3 does not feature the speculative read function.

(5) The PCTn/$\overline{\text{xxWR}}$/$\overline{\text{xxBE}}$/xxDQM pin is used as the $\overline{\text{xxBE}}$/xxDQM pin.

- Setting of the port CT function control register (PFCCT): 0FH

Remark n = 0 to 3, xx = LL, LU, UL, UU

(6) When external memory or external I/O is connected via an externally attached circuit, the signal propagation delay time of the attached circuit is assumed to be 2 ns (MIN.) and 7 ns (MAX.) for calculation. Because the delay time on the printed wiring board is not included, take a margin of several ns into consideration, as necessary, for the actual circuit.

2.1 Connecting 8-bit SRAM

This section shows examples of connecting SRAM (μPD444008L: 512K × 8 bits) (where SRAM is allocated to block 1 of the V850E/ME2 and to subarea 01 of the V850E2/ME3, and $\overline{CS2}$ is used as the $\overline{CS}$ signal).

Remark $\overline{CS2}$ cannot be used when $\overline{IORD}$ and $\overline{IOWR}$ are used. For configuration examples, refer to **2.11 System Configuration Example and CSCn Register Setting**.

2.1.1 Connecting μPD444008L (example of 8-bit bus width)

An example of connecting one SRAM (μPD444008L-A8: 512K × 8 bits) is shown below (8-bit bus width, 512 KB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD444008L-A8 × 1
- $\overline{CS}$ signal used: $\overline{CS2}$

  The SRAM is allocated to 0200000H to 027FFFFH (V850E/ME2: Block 1, V850E2/ME3: Subarea 01) of the external memory space.

  Addresses 0280000H to 03FFFFFH are images.

[Concept and cautions]

Because the bus width configuration is 8 bits, the data bus (I/O0 to I/O7) of the μPD444008L-A8 is connected to D0 to D7 of the V850E/ME2 or V850E2/ME3, and the address bus (A0 to A18) of the μPD444008L-A8 is connected to the A0 to A18 of the V850E/ME2 or V850E2/ME3. The $\overline{CS}$, $\overline{OE}$, and $\overline{WE}$ pins of the μPD444008L-A8 are connected to the $\overline{CS2}$, $\overline{RD}$, and $\overline{WR}$ pins of the V850E/ME2 or V850E2/ME3, respectively.

[Register settings]

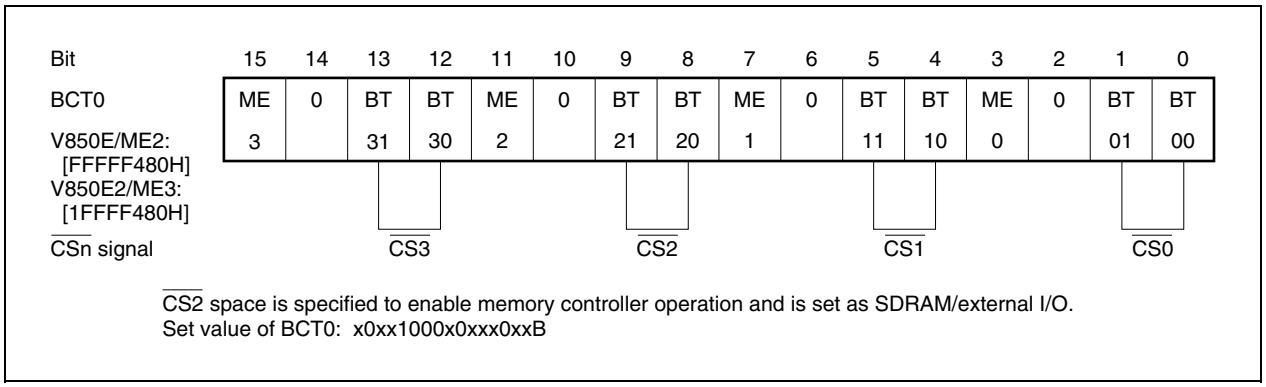
- Area of $\overline{CS2}$ to be used, device type, and bus width:
  V850E/ME2: Block 1, V850E2/ME3: Subarea 01, SRAM/external I/O, 8-bit width
- Wait setting: 0 waits
- Address setup wait: 0 waits
- Address state: 1 state

Figure 2-1. Setting of Chip Area Select Control Register 0 (CSC0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF060H] V850E2/ME3: [1FFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

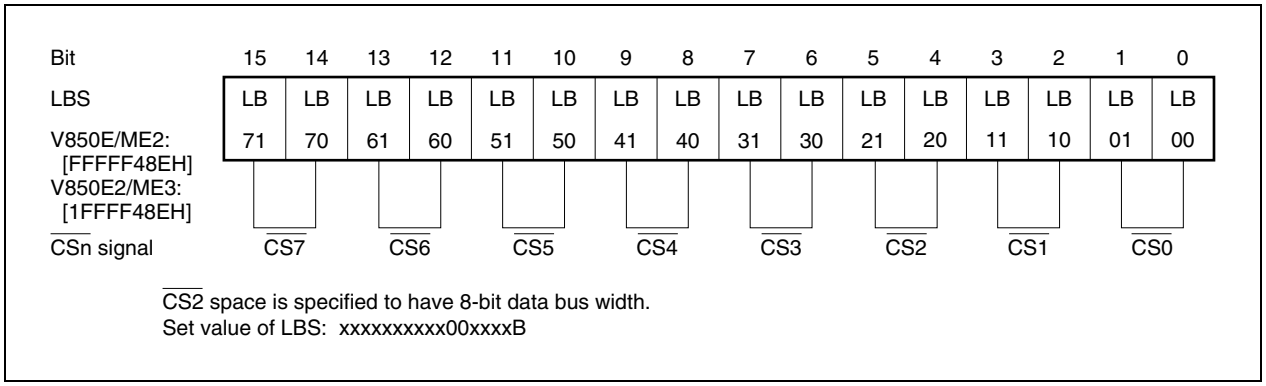
$\overline{CS2}$ output when block 1 of V850E/ME2 or subarea 01 of V850E2/ME3 is accessed
Set value of CSC0: xxxx0010xxxxxx0xB

Figure 2-2. Setting of Bus Cycle Type Configuration Register 0 (BCT0)



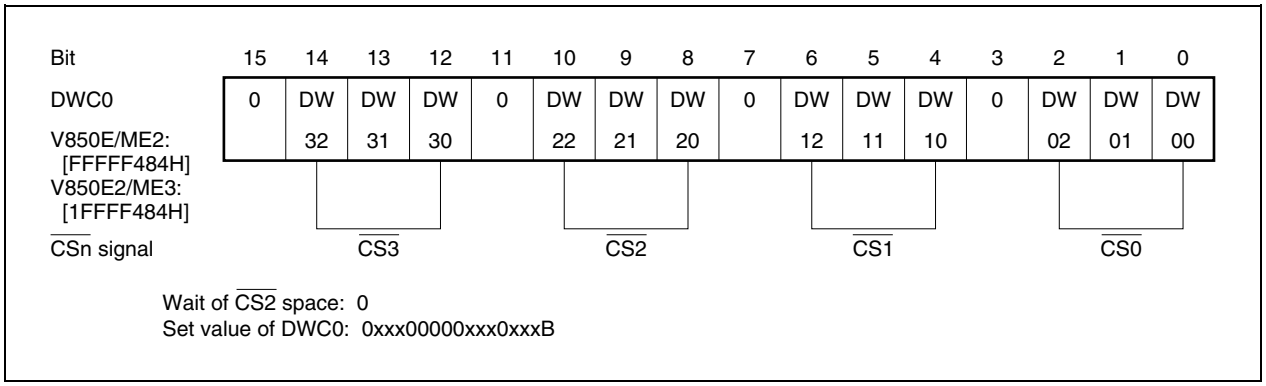
$\overline{CS2}$ space is specified to enable memory controller operation and is set as SDRAM/external I/O.
Set value of BCT0: x0xx1000x0xxx0xxB

Figure 2-3. Setting of Local Bus Sizing Control Register (LBS)



$\overline{CS2}$ space is specified to have 8-bit data bus width.
Set value of LBS: xxxxxxxxxx00xxxxB

Figure 2-4. Setting of Data Wait Control Register 0 (DWC0)



Wait of $\overline{CS2}$ space: 0
Set value of DWC0: 0xxx00000xxx0xxxB

Figure 2-5. Setting of Address Setup Wait Control Register (ASC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
V850E/ME2: [FFFFF48AH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48AH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Address setup wait of $\overline{CS2}$ space: 0
Set value of ASC: xxxxxxxxxx00xxxxB

Figure 2-6. Setting of Bus Cycle Control Register (BCC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2: [FFFFF488H]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF488H]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Idle state of $\overline{CS2}$ space: 1
Set value of BCC: xxxxxxxxxx01xxxxB

Figure 2-7. Example of μPD444008L-A8 Connection Circuit

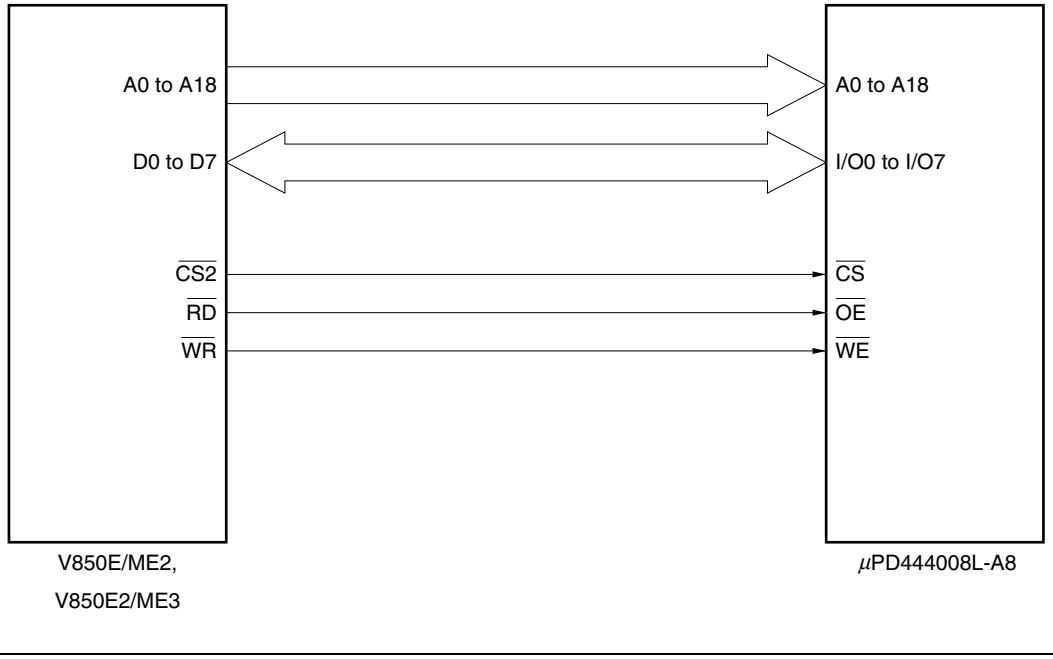
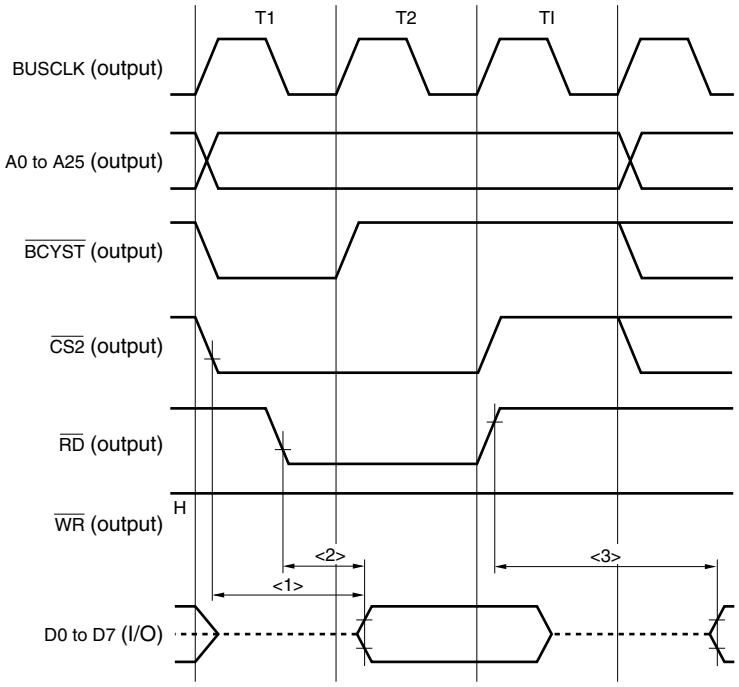


Figure 2-8. Read Operation of μPD444008L-A8



<1> Output delay time after address and $\overline{CS}$ of μPD444008L-A8 are asserted: 8 ns (MAX.)

Maximum value of data input setup time (to address) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SAID} (ns) $= (2 + w + w_D + w_{AS})\,T - 17$
$= 2 \times 15.6 - 17$; $w = 0$, $w_D = 0$, $w_{AS} = 0$, $T = 15.6$ ns
$= 14.2$ ns (> 8 ns)

<2> Output delay time after $\overline{OE}$ of μPD444008L-A8 is asserted: 4 ns (MAX.)

Maximum value of data input setup time (to $\overline{RD}$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SRDID} (ns) $= (1.5 + w + w_D)\,T - 17$
$= 1.5 \times 15.6 - 17$; $w = 0$, $w_D = 0$, $T = 15.6$ ns
$= 6.4$ ns (> 4 ns)

<3> Output floating delay time after $\overline{OE}$ of μPD444008L-A8 is deasserted: 4 ns (MIN.)

Minimum value of data output delay time (from $\overline{RD}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DRDOD} (ns) $= (0.5 + i)\,T - 6$
$= 1.5 \times 15.6 - 6$; $i = 1$, $T = 15.6$ ns
$= 17.4$ ns (> 4 ns)

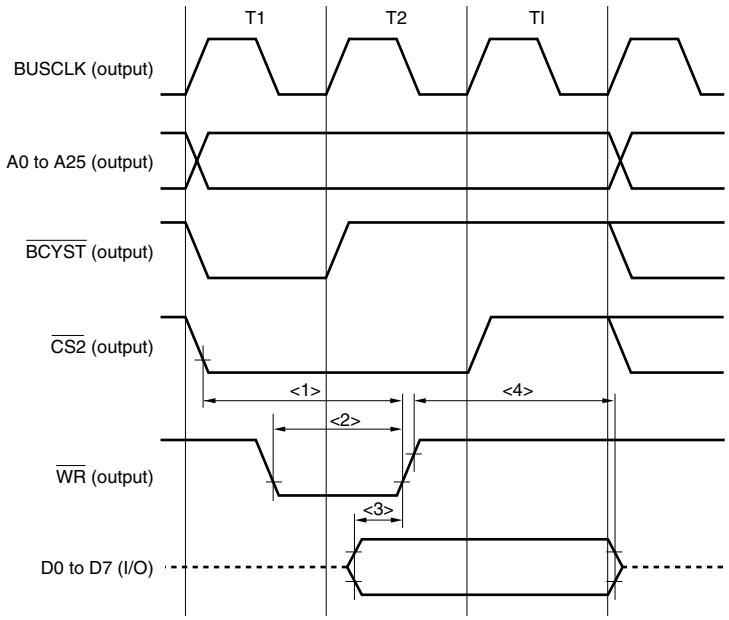
Minimum value of high level width of $\overline{RD}$

t_{WRDH} (ns) $= (0.5 + w_{AS} + i)\,T - 6$
$= 1.5 \times 15.6 - 6$; $w_{AS} = 0$, $i = 1$, $T = 15.6$ ns
$= 17.4$ ns (> 4 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)
w: Number of wait states inserted by $\overline{WAIT}$
w_D: Number of wait states specified by DWC0 and DWC1 registers
w_{AS}: Number of address setup wait states specified by ASC register
i: Number of idle states

Figure 2-9. Write Operation of μPD444008L-A8



<1> Time after address and $\overline{CS}$ of μPD444008L-A8 are asserted until $\overline{WR}$ rises: 6 ns (MIN.)
Minimum value of address setup time (to $\overline{WR}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SAWR} (ns) = $(1.5 + w + w_D + w_{AS})\,T - 10$
= $1.5 \times 15.6 - 10$; $w = 0$, $w_D = 0$, $w_{AS} = 0$, $T = 15.6$ ns
= 13.4 ns (> 6 ns)

<2> $\overline{WE}$ active pulse width in μPD444008L-A8: 6 ns (MIN.)
Minimum value of $\overline{WR}$ low-level width from electrical specifications of V850E/ME2 and V850E2/ME3

t_{WWRL} (ns) = $(1 + w + w_D)\,T - 5$
= $1 \times 15.6 - 5$; $w = 0$, $w_D = 0$, $T = 15.6$ ns
= 10.6 ns (> 6 ns)

<3> Data setup time to rising of $\overline{WR}$ in μPD444008L-A8: 4 ns (MIN.)
Minimum value of data output setup time (to $\overline{WR}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SODWR} (ns) = $(1.5 + w_{AS} + w + w_D)\,T - 5$
= $1.5 \times 15.6 - 5$; $w_{AS} = 0$, $w = 0$, $w_D = 0$, $T = 15.6$ ns
= 18.4 ns (> 4 ns)

<4> Data hold time from rising of $\overline{WR}$ in μPD444008L-A8: 0 ns (MIN.)
Minimum value of data output hold time (from $\overline{WR}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{HWROD} (ns) = $(0.5 + i)\,T - 5.5$
= $1.5 \times 15.6 - 5.5$; $i = 1$, $T = 15.6$ ns
= 17.9 ns

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)
w: Number of wait states inserted by $\overline{WAIT}$
w_D: Number of wait states specified by DWC0 and DWC1 registers
w_{AS}: Number of address setup wait states specified by ASC register
i: Number of idle states

2.1.2 Connecting μPD444008L (example of 16-bit bus width)

An example of connecting two SRAMs (μPD444008L-A8: 512K × 8 bits) (16-bit bus width, 1 MB external memory space) is shown below.

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD444008L-A8 × 2
- $\overline{CS}$ signal used: $\overline{CS2}$

  The SRAMs are allocated to 0200000H to 02FFFFFH (V850E/ME2: Block 1, V850E2/ME3: Subarea 01) of the external memory space.

  Addresses 0300000H to 03FFFFFH are images.

[Concept and cautions]

Because the bus width configuration is 16 bits, the address bus (A0 to A18) of the μPD444008L-A8 is connected to A1 to A19 of the V850E/ME2 or V850E2/ME3. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D0 to D7 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{LLBE}$ signals of the V850E/ME2 or V850E2/ME3. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D8 to D15 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{LUBE}$ signals of the V850E/ME2 or V850E2/ME3. The $\overline{CS}$ and $\overline{OE}$ pins of the μPD444008L-A8 are connected to the $\overline{CS2}$ and $\overline{RD}$ pins of the V850E/ME2 or V850E2/ME3, respectively.

[Register settings]

Same as **2.1.1 Connecting μPD444008L (example of 8-bit bus width)**, except for the setting of the LBS register (which specifies the data bus width of the $\overline{CS}$ space).

Figure 2-10. Setting of Local Bus Sizing Control Register (LBS)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2: [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48EH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS2}$ space is specified to have 16-bit data bus width
Set value of LBS: xxxxxxxxxx01xxxxB

Figure 2-11. Example of μPD444008L-A8 Circuit Connection

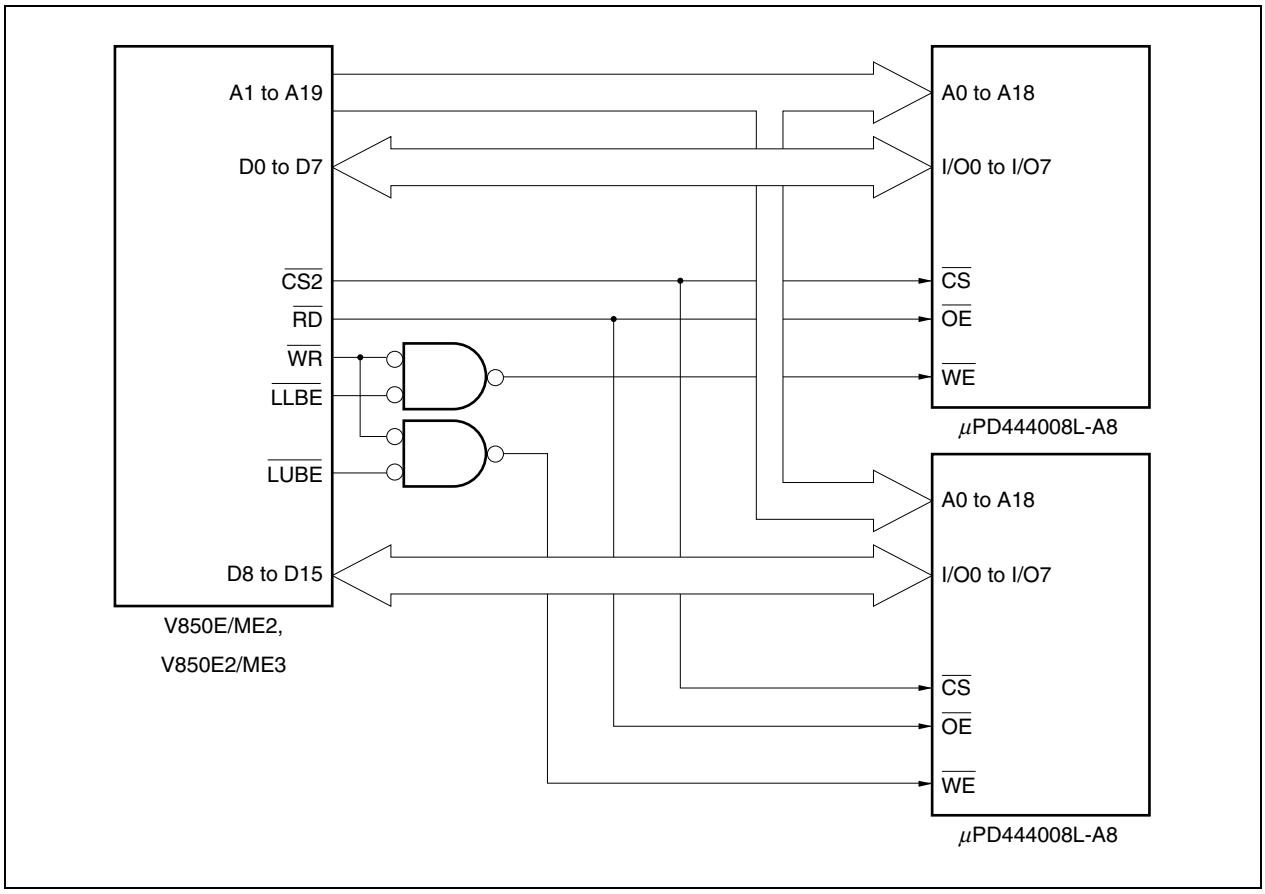
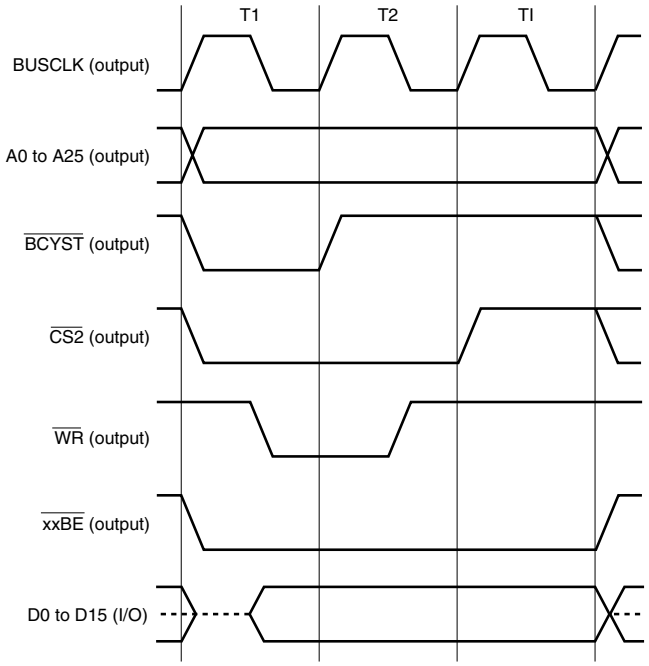


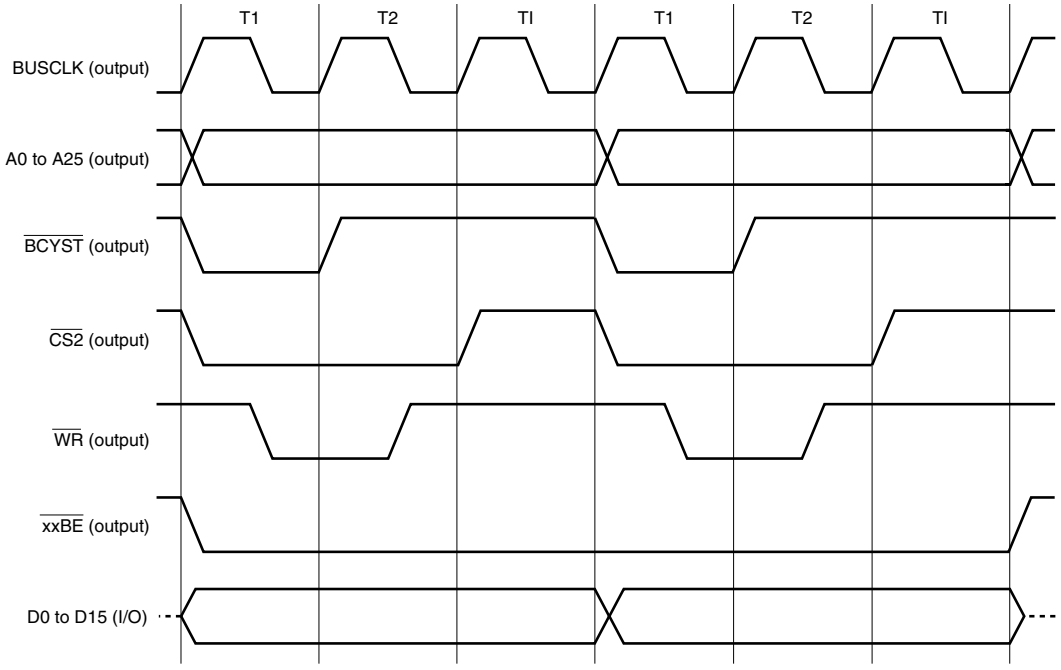
Figure 2-12. Write Operation of μPD444008L-A8 (w_D = 0, w_{AS} = 0, i = 1, 16-bit Bus Width)



Remarks 1. The dotted line indicates high impedance.

2. xx = LL, LU

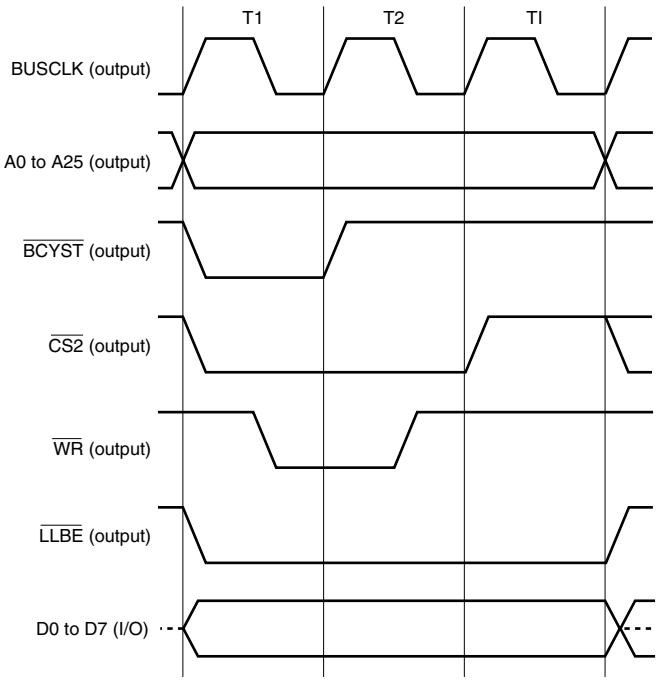
Figure 2-13. Write Operation of μPD444008L-A8 (w_D = 0, w_{AS} = 0, i = 1, 16-bit Bus Width, Word Access)



Remarks 1. The dotted line indicates high impedance.

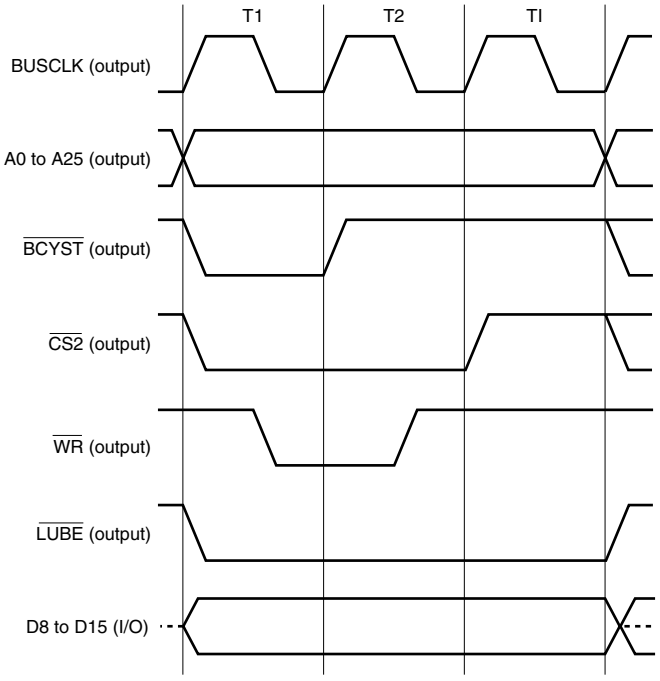
2. xx = LL, LU

Figure 2-14. Write Operation of μPD444008L-A8 (Byte Access to D0 to D7)



Remark The dotted line indicates high impedance.

Figure 2-15. Write Operation of μPD444008L-A8 (Byte Access to D8 to D15)



Remark The dotted line indicates high impedance.

2.1.3 Connecting μPD444008L (example of 32-bit bus width)

An example of connecting four SRAMs (μPD444008L-A8: 512K × 8 bits) (32-bit bus width, 2 MB external memory space) is shown below.

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD444008L-A8 × 4
- $\overline{CS}$ signal used: $\overline{CS2}$

  The SRAMs are allocated to 0200000H to 03FFFFFH (V850E/ME2: Block 1, V850E2/ME3: Subarea 01) of the external memory space.

[Concept and cautions]

Because the bus width configuration is 32 bits, the address bus (A0 to A18) of the μPD444008L-A8 is connected to A2 to A20 of the V850E/ME2 or V850E2/ME3. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D0 to D7 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{LLBE}$ signals. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D8 to D15 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{LUBE}$ signals. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D16 to D23 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{ULBE}$ signals. The $\overline{WE}$ pins of the μPD444008L-A8 connected to D24 to D31 of the V850E/ME2 or V850E2/ME3 are controlled by the $\overline{WR}$ and $\overline{UUBE}$ signals. The $\overline{CS}$ and $\overline{OE}$ pins of the μPD444008L-A8 are connected to the $\overline{CS2}$ and $\overline{RD}$ pins of the V850E/ME2 or V850E2/ME3, respectively.

[Register settings]

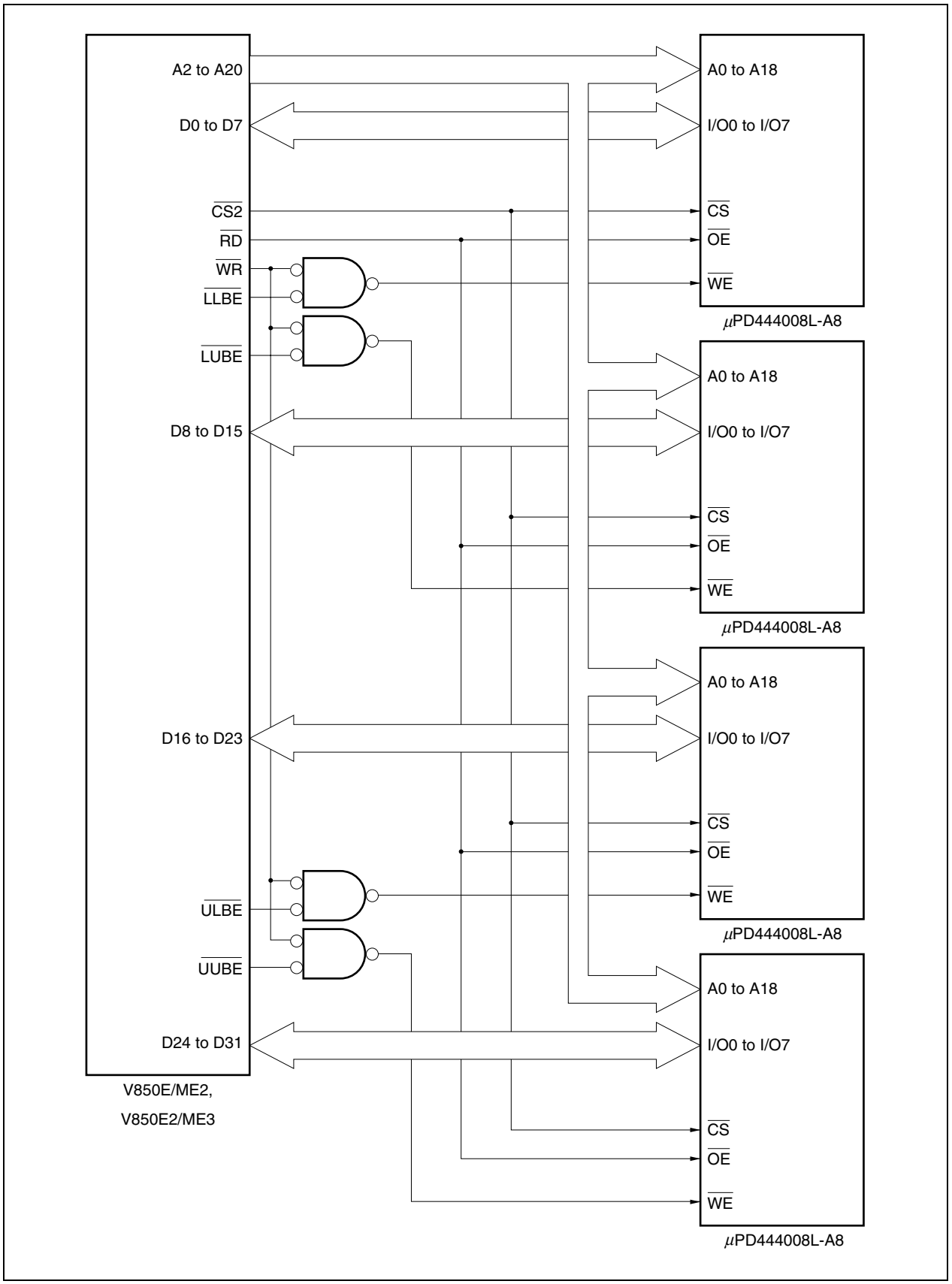
Same as **2.1.1 Connecting μPD444008L (example of 8-bit bus width)**, except for the setting of the LBS register (which specifies the data bus width of the $\overline{CS}$ space).

Figure 2-16. Setting of Local Bus Sizing Control Register (LBS)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2: [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48EH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS2}$ space is specified to have 32-bit data bus width.
Set value of LBS: xxxxxxxxxx10xxxxB

Figure 2-17. Example of μPD444008L-A8 Connection Circuit



2.2 Connecting 16-bit SRAM

This section shows examples of connecting SRAM (μPD4440016L: 256K × 16 bits) (where the SRAM is allocated to block 4 of the V850E/ME2 and to subarea 04 of the V850E2/ME3, and $\overline{CS5}$ is used as the $\overline{CS}$ signal).

Remark This section shows only register settings and connection circuit examples. The operation timing is the same as **2.1 Connecting 8-bit SRAM**.

2.2.1 Connecting μPD4440016L (example of 16-bit bus width)

An example of connecting one SRAM (μPD4440016L-A8: 256K × 16 bits) is shown below (16-bit bus width, 512 KB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD4440016L-A8 × 1
- $\overline{CS}$ signal used: $\overline{CS5}$

  The SRAM is allocated to F800000H to F87FFFFH (block 4) of the external memory space in the V850E/ME2.

  F880000H to F9FFFFFH are images.

  The SRAM is allocated to 1F800000H to 1F87FFFFH (subarea 04) of the external memory space of the V850E2/ME3. 1F880000H to 1F9FFFFFH are images.

[Concept and cautions]

Because the bus width configuration is 16 bits, the data bus (I/O0 to I/O15) of the μPD4440016L-A8 is connected to D0 to D15 of the V850E/ME2 or V850E2/ME3, and the address bus (A0 to A17) is connected to A1 to A18 of the V850E/ME2 or V850E2/ME3. The $\overline{CS}$, $\overline{OE}$, $\overline{WE}$, $\overline{LB}$, and $\overline{UB}$ pins of the μPD4440016L-A8 are connected to the $\overline{CS5}$, $\overline{RD}$, $\overline{WR}$, $\overline{LLBE}$, and $\overline{LUBE}$ pins of the V850E/ME2 or V850E2/ME3, respectively.

[Register settings]

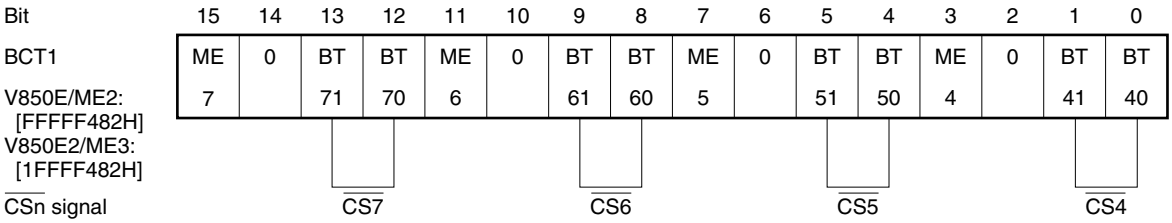
- Area of $\overline{CS5}$ to be used, device type, and bus width:
  V850E/ME2: Block 4, V850E2/ME3: Subarea 04, SRAM/external I/O, 16-bit width
- Wait setting: 0 waits
- Address setup wait: 0 waits
- Idle state: 1 state

Figure 2-18. Setting of Chip Area Select Control Register 1 (CSC1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC1	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF062H]	43	42	41	40	53	52	51	50	63	62	61	60	73	72	71	70
V850E2/ME3: [1FFFF062H]																

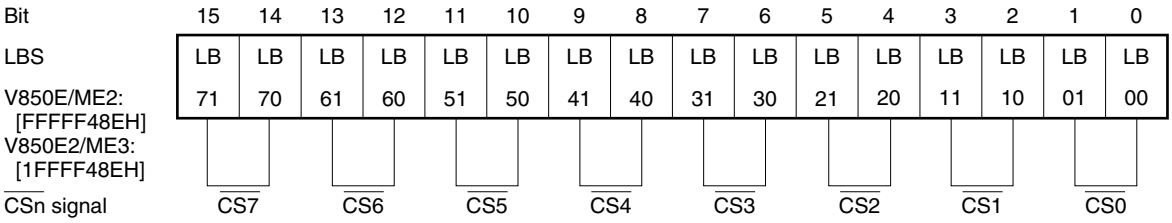
$\overline{CS5}$ output when block 4 of V850E/ME2 or subarea 04 of V850E2/ME3 is accessed
Set value of CSC1: xxxx1000xxxx0xxxB

Figure 2-19. Setting of Bus Cycle Type Configuration Register 1 (BCT1)



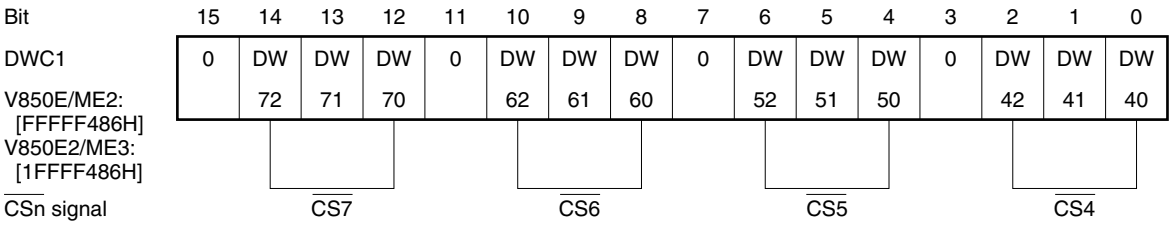
CS5 space is specified to enable memory controller operation and is set as SRAM/external I/O.
Set value of BCT1: x0xxx0xx1000x0xxB

Figure 2-20. Setting of Local Bus Sizing Control Register (LBS)



CS5 space is specified to have 16-bit data bus width.
Set value of LBS: xxxx01xxxxxxxxxxB

Figure 2-21. Setting of Data Wait Control Register 1 (DWC1)



Wait of CS5 space: 0
Set value of DWC1: 0xxx0xxx00000xxxB

Figure 2-22. Setting of Address Setup Wait Control Register (ASC)

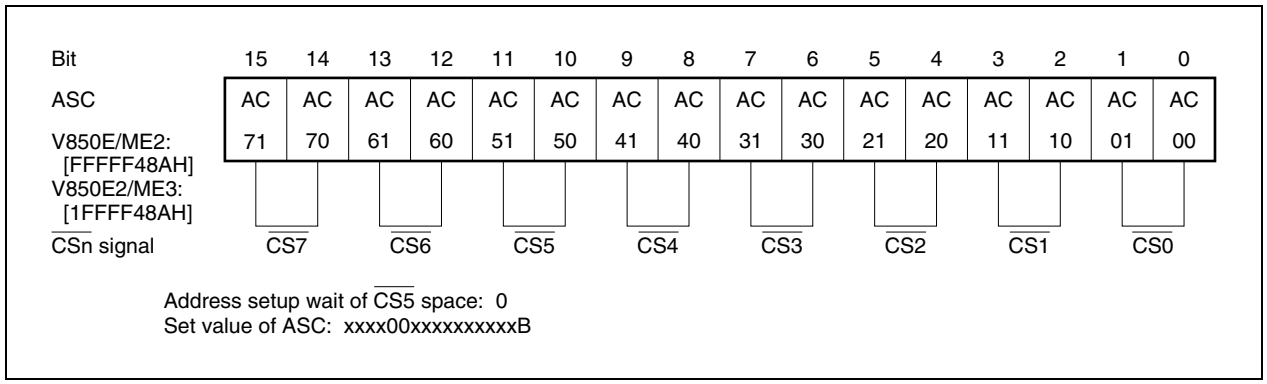


Figure 2-23. Setting of Bus Cycle Control Register (BCC)

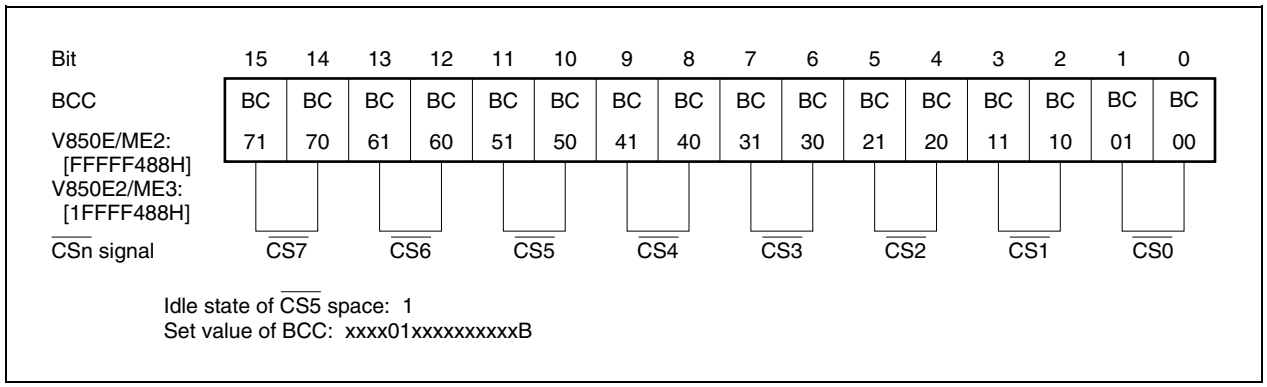
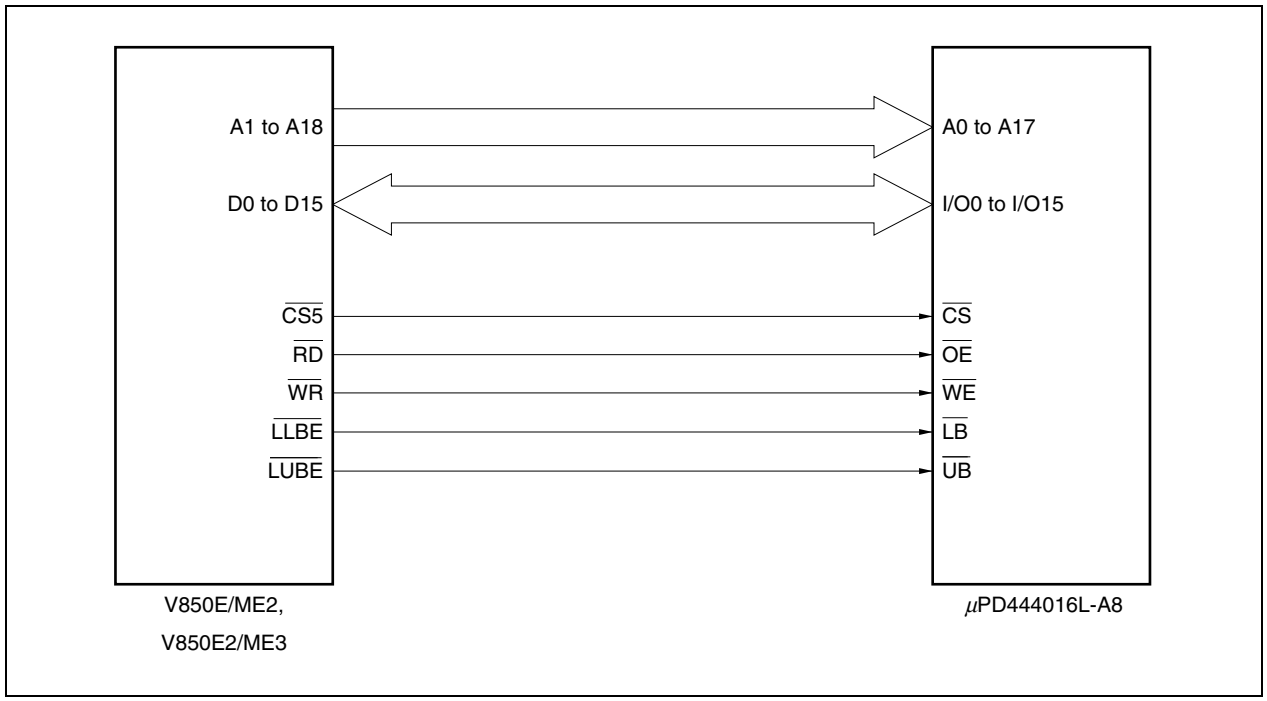


Figure 2-24. Example of μPD4440016L-A8 Connection Circuit



2.2.2 Connecting μPD4440016L (example of 32-bit bus width)

An example of connecting two SRAMs (μPD4440016L-A8: 256K x 16 bits) is shown below (32-bit bus width, 1 MB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD4440016L-A8 × 2
- $\overline{CS}$ signal used: $\overline{CS5}$

  The SRAMs are allocated to F800000H to F8FFFFFH (block 4) of the external memory space of the V850E/ME2.

  F900000H to F9FFFFFH are images.

  The SRAM is allocated to 1F800000H to 1F87FFFFH (subarea 04) of the external memory space of the V850E2/ME3.

  F900000H to 1F9FFFFFH are images.

[Concept and cautions]

Because the bus width configuration is 32 bits, the address bus (A0 to A17) of the μPD4440016L-A8 is connected to A2 to A19 of the V850E/ME2 or V850E2/ME3. The $\overline{LB}$ and $\overline{UB}$ pins of the μPD4440016L-A8 connected to D0 to D15 of the V850E/ME2 or V850E2/ME3 are connected to the $\overline{LLBE}$ and $\overline{LUBE}$ pins of the V850E/ME2 or V850E2/ME3. The $\overline{LB}$ and $\overline{UB}$ pins of the μPD4440016L-A8 connected to D16 to D31 of the V850E/ME2 or V850E2/ME3 are connected to the $\overline{ULBE}$ and $\overline{UUBE}$ pins of the V850E/ME2 or V850E2/ME3. The $\overline{CS}$, $\overline{OE}$, and $\overline{WE}$ pins of the μPD4440016L-A8 are connected to $\overline{CS5}$, $\overline{RD}$, and $\overline{WR}$ pins of the V850E/ME2 or V850E2/ME3, respectively.

[Register settings]

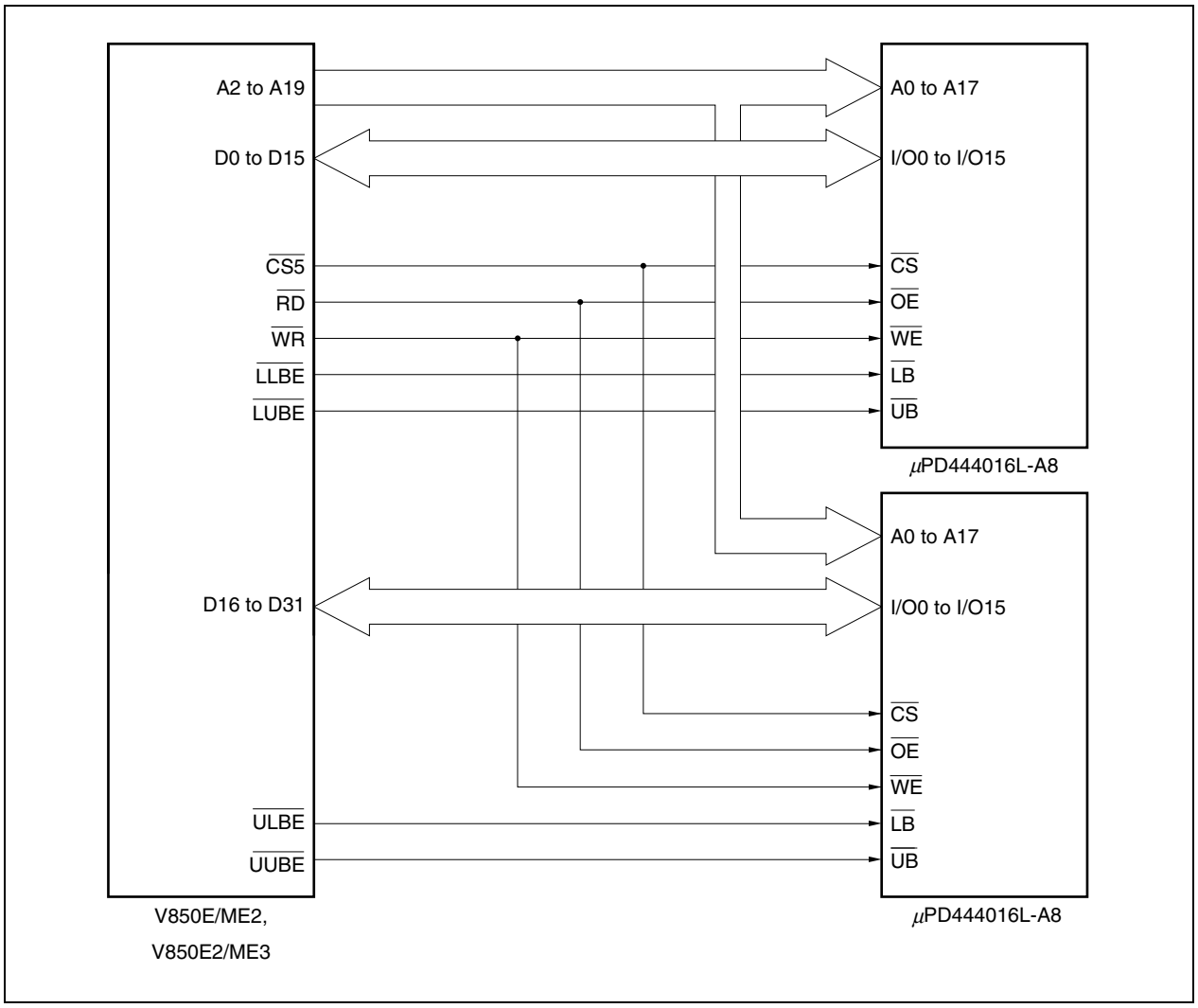
Same as **2.2.1 Connecting μPD4440016L (example of 16-bit bus width)**, except for the setting of the LBS register (which specifies the data bus width of the $\overline{CS}$ space).

Figure 2-25. Setting of Local Bus Sizing Control Register (LBS)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2: [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48EH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS5}$ space is specified to have 32-bit data bus width.
Set value of LBS: xxxx10xxxxxxxxxxB

Figure 2-26. Example of μPD4440016L-A8 Connection Circuit



2.2.3 Wait/idle setting and bus clocks when SRAM is connected

Table 2-1. Wait/Idle Setting and Bus Clocks When SRAM Is Connected

(a) Connection with μPD444016L-A8

BUSCLK	Data Wait	Address Setup Wait	Idle State	Restricted Timing[Note]
BUSCLK ≤ 50 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
50 MHz < BUSCLK ≤ 66 MHz	0	0	1	–

(b) Connection with μPD444016L-A10

BUSCLK	Data Wait	Address Setup Wait	Idle State	Restricted Timing[Note]
BUSCLK ≤ 45 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
45 MHz < BUSCLK ≤ 66 MHz	0	0	1	–

(c) Connection with μPD444016L-A12

BUSCLK	Data Wait	Address Setup Wait	Idle State	Restricted Timing[Note]
BUSCLK ≤ 41 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
41 MHz < BUSCLK ≤ 65 MHz	0	0	1	t_{SRDID}
65 MHz < BUSCLK ≤ 66 MHz	1	0	1	–

Note Refer to **2.1.1 Connecting μPD444008L (example of 8-bit bus width)**.

2.3 Connecting PROM

This section shows an example of connecting one PROM (M27V800: 1M × 8 bits/512K × 16 bits) (16-bit bus width, 1 MB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: M27V800-100 × 1
- $\overline{CS}$ signal used: $\overline{CS0}$
  The PRAM is allocated to 0100000H to 01FFFFFH (block 0) of the external memory space of the V850E/ME2. It is allocated to 0000000H to 00FFFFFH (subarea 00) of the external memory space of the V850E2/ME3.

[Concept and cautions]

- The address bus (A0 to A18) of the M27V800-100 is connected to A1 to A19 of the V850E/ME2 or V850E2/ME3.
- The $\overline{CE}$, $\overline{OE}$, and O0 to O15 pins of the M27V800-100 are connected to the $\overline{CS0}$, $\overline{RD}$, and D0 to D15 pins of the V850E/ME2 or V850E2/ME3.
- The $\overline{BYTE}$ pin of the M27V800-100 is pulled up because a 16-bit bus width is used.
- The PROM is accessed in the same manner as an SRAM read operation.
- The bus width of the $\overline{CS0}$ space is determined by the statuses of the MODE0 and MODE1 pins. When writing data to the LBS register, do not change the set value of the $\overline{CS0}$ space.

[Register settings]

- Area of $\overline{CS0}$ to be used and device type:
  V850E/ME2: Block 0, V850E2/ME3: Subarea 00, SRAM/external I/O
- Wait setting: 6 waits[Note]
- Address setup wait: 0 waits (1 wait[Note])
- Idle state: 3 states

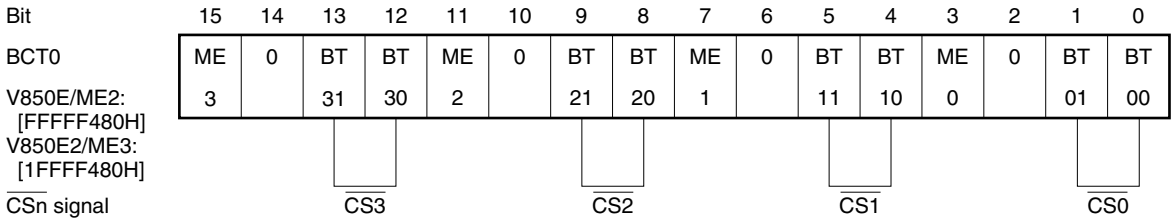
Note Wait setting = 5 (or 4) and address setup wait = 1 (or 2) are equivalent settings.

Figure 2-27. Setting of Chip Area Select Control Register 0 (CSC0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00
V850E2/ME3: [1FFFF060H]																

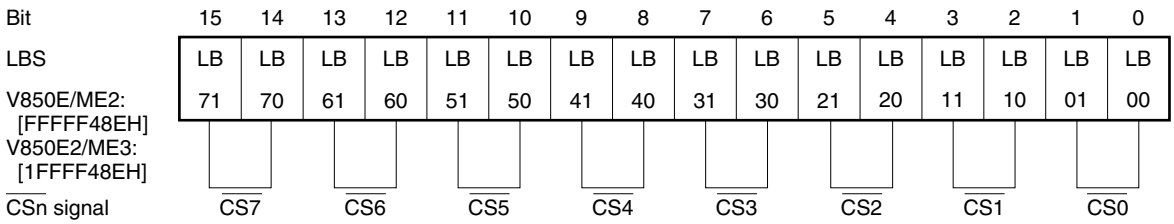
$\overline{CS0}$ output when block 0 of V850E/ME2 or subarea 00 of V850E2/ME3 is accessed
Set value of CSC0: xxxxxxx0xxxx0001B

Figure 2-28. Setting of Bus Cycle Type Configuration Register 0 (BCT0)



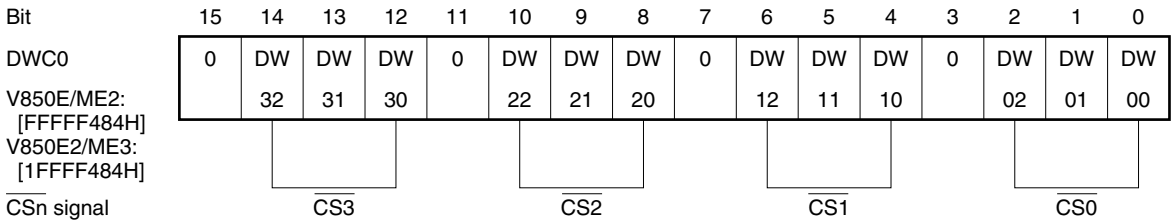
$\overline{CS0}$ space is specified to enable memory controller operation and is set as SRAM/external I/O.
Set value of BCT0: x0xxx0xxx0xx1000B

Figure 2-29. Setting of Local Bus Sizing Control Register (LBS)



$\overline{CS0}$ space is specified to have 16-bit data bus width.
Set value of LBS: xxxxxxxxxxxxxx01B

Figure 2-30. Setting of Data Wait Control Register 0 (DWC0)



Wait of $\overline{CS0}$ space: 6
Set value of DWC0: 0xxx0xxx0xxx0110B

Figure 2-31. Setting of Address Setup Wait Control Register (ASC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
V850E/ME2: [FFFFF48AH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48AH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Address setup wait of $\overline{CS0}$ space: 0
Set value of ASC: xxxxxxxxxxxxxx00B

Figure 2-32. Setting of Bus Cycle Control Register (BCC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2: [FFFFF488H]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF488H]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Idle state of $\overline{CS0}$ space: 1
Set value of BCC: xxxxxxxxxxxxxx11B

Figure 2-33. Example of M27V800-100 Connection Circuit

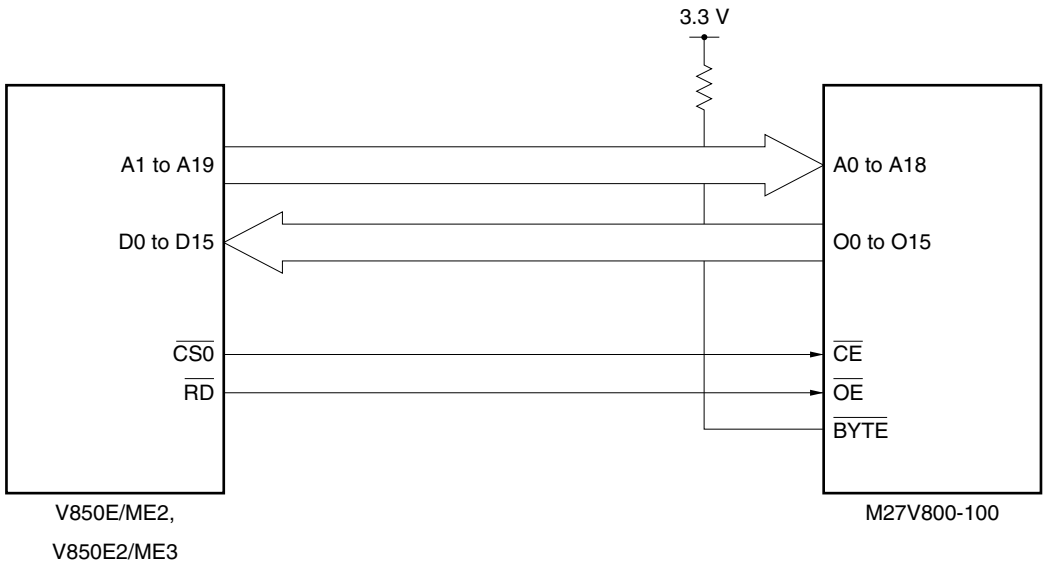
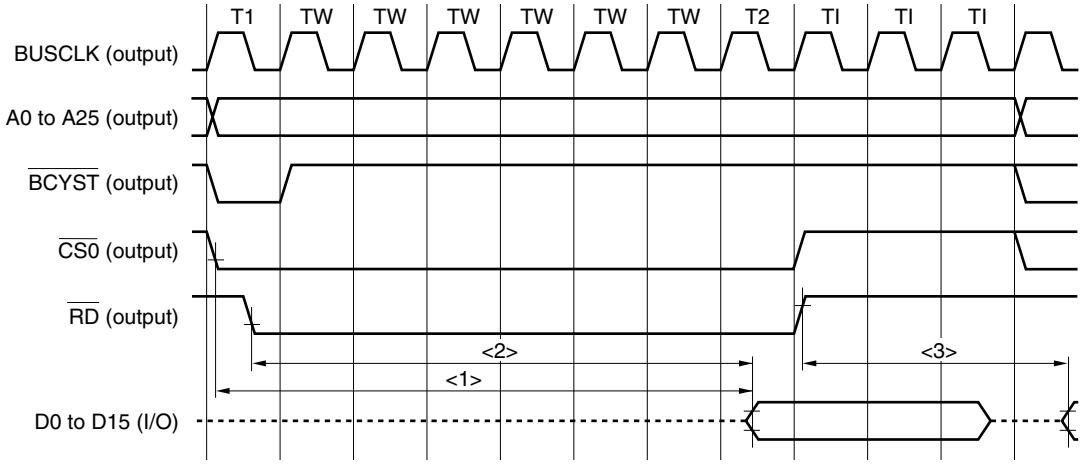


Figure 2-34. Read Operation of M27V800-100



<1> Output delay time after address and $\overline{CE}$ of M27V800-100 are asserted: 100 ns (MAX.)

Maximum value of data input setup time (to address) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SAID} (ns) = (2 + w + w_D + w_{AS}) T – 17

= 8 × 15.6 – 17; w = 0, w_D = 6, w_{AS} = 0, T = 15.6 ns

= 107.8 ns (> 100 ns)

<2> Output delay time after $\overline{OE}$ of M27V800-100 is asserted: 50 ns (MAX.)

Maximum value of data input setup time (to $\overline{RD}$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SRDID} (ns) = (1.5 + w + w_D) T – 17

= 7.5 × 15.6 – 17; w = 0, w_D = 6, T = 15.6 ns

= 100 ns (> 50 ns)

<3> Output floating delay time after $\overline{OE}$ of M27V800-100 is deasserted: 45 ns (MAX.)

Minimum value of data output delay time (from $\overline{RD}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DRDOD} (ns) = (0.5 + i) T – 6

= 3.5 × 15.6 – 6; i = 3, T = 15.6 ns

= 48.6 ns (> 45 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)

w: Number of wait states inserted by $\overline{WAIT}$

w_D: Number of wait states specified by DWC0 and DWC1 registers

w_{AS}: Number of address setup wait states specified by ASC register

i: Number of idle states

2.4 Connecting Page ROM

This section shows an example of connecting one page ROM (μPD23C16080BL: 2M × 8 bits/1M × 16 bits) (16-bit bus width, 2 MB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD23C16080BL × 1
- $\overline{CS}$ signal used: $\overline{CS2}$
  The page ROM is allocated to 0600000H to 07FFFFFH (V850E/ME2: Block 3, V850E2/ME3: Subarea 03) of the external memory space.

[Concept and cautions]

- The $\overline{BYTE}$ pin of the μPD23C16080BL is pulled up because a 16-bit bus width is used.
- Because the page size of the μPD23C16080BL is 16 bytes, the PRC register is set to 8 × 16 bits.
- The number of wait states for off-page is set by the DWC0 register, and the number of wait states for on-page and the page size are set by the PRC register.

[Register settings]

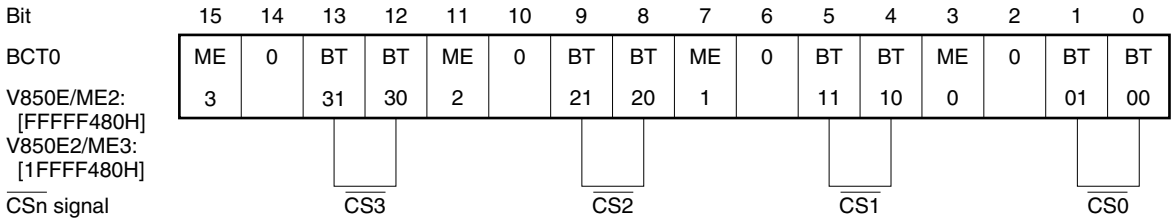
- Area of $\overline{CS2}$ to be used and device type:
  V850E/ME2: Block 3, V850E2/ME3: Subarea 03, page ROM
- Wait setting for off-page: 5 waits
- Wait setting for on-page: 1 wait
- Address setup wait: 0 waits
- Idle state: 2 states

Figure 2-35. Setting of Chip Area Select Control Register 0 (CSC0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF060H] V850E2/ME3: [1FFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

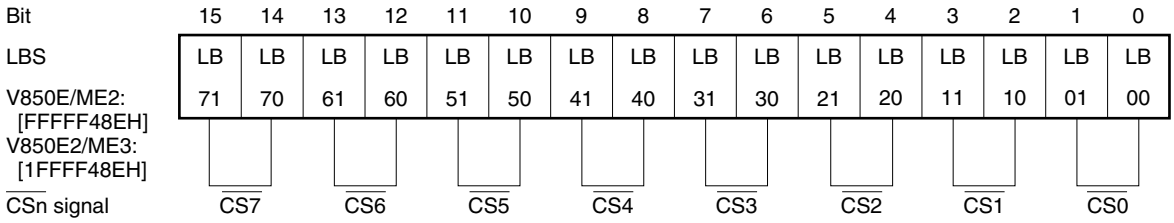
$\overline{CS2}$ output when block 3 of V850E/ME2 or subarea 03 of V850E2/ME3 is accessed
Set value of CSC0: xxxx1000xxxx0xxxB

Figure 2-36. Setting of Bus Cycle Type Configuration Register 0 (BCT0)



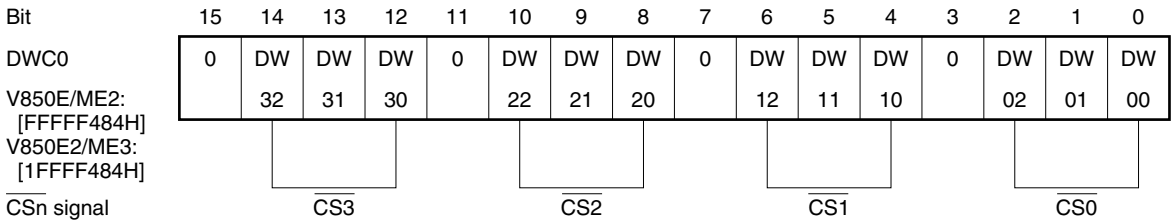
CS2 space is specified to enable memory controller operation and is set as page ROM.
Set value of BCT0: x0xx1001x0xxx0xxB

Figure 2-37. Setting of Local Bus Sizing Control Register (LBS)



CS2 space is specified to have 16-bit data bus width.
Set value of LBS: xxxxxxxxxx01xxxxB

Figure 2-38. Setting of Data Wait Control Register 0 (DWC0)



Wait of CS2 space: 6
Set value of DWC0: 0xxx01010xxx0xxxB

Figure 2-39. Setting of Address Setup Wait Control Register (ASC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
V850E/ME2: [FFFFF48AH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48AH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Address setup wait of $\overline{CS2}$ space: 0
Set value of ASC: xxxxxxxxxx00xxxxB

Figure 2-40. Setting of Bus Cycle Control Register (BCC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2: [FFFFF488H]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF488H]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Idle state of $\overline{CS2}$ space: 1
Set value of BCC: xxxxxxxxxx10xxxxB

Figure 2-41. Setting of Page ROM Configuration Register (PRC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRC	0	PRW	PRW	PRW	0	0	0	0	0	0	0	0	MA6	MA5	MA4	MA3
V850E/ME2: [FFFFF49AH]		2	1	0												
V850E2/ME3: [1FFFF49AH]																

Number of wait states for on-page: 1
Page size: 8 × 16 bits
Set value of PRC: 0001000000000001B

Figure 2-42. Example of μPD23C16080BL Connection Circuit

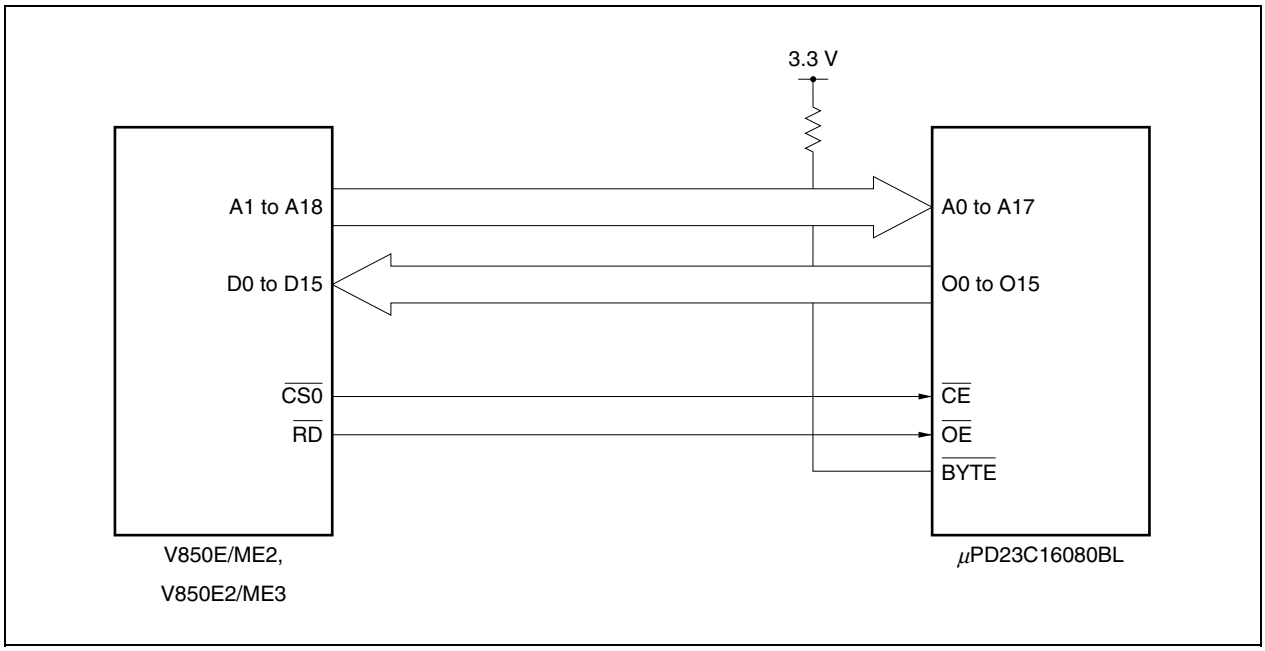
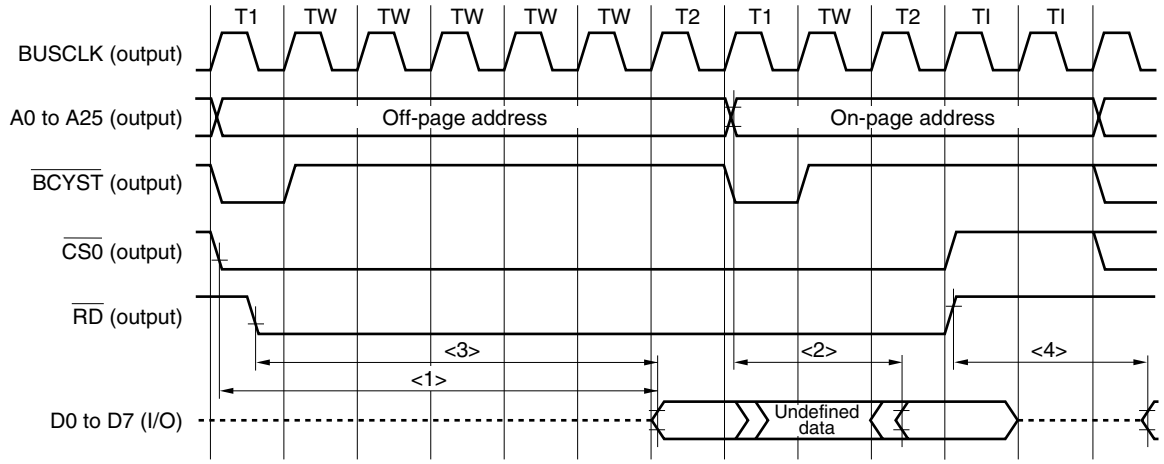


Figure 2-43. Read Operation of μPD23C16080BL



<1> Output delay time after address and $\overline{CE}$ for off-page of μPD23C16080BL are asserted: 85 ns (MAX.)
Maximum value of off-page data input setup time (to address) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SAID} (ns) = $(2 + w + w_D + w_{AS})\ T - 17$
= $7 \times 15.6 - 17$; $w = 0$, $w_D = 5$, $w_{AS} = 0$, $T = 15.6$ ns
= 92.2 ns (> 85 ns)

<2> Output delay time from on-page address of μPD23C16080BL: 25 ns (MAX.)
Maximum value of on-page data input setup time (to address) from electrical specifications of V850E/ME2 and V850E2/ME3

T_{SOAID} (ns) = $(2 + w + w_{PR})\ T - 17$
= $3 \times 15.6 - 17$; $w = 0$, $w_{PR} = 1$, $T = 15.6$ ns
= 29.8 ns (> 25 ns)

<3> Output delay time after $\overline{OE}$ of μPD23C16080BL is asserted: 25 ns (MAX.)
Maximum value of off-page data input setup time (to $\overline{RD}$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SRDID} (ns) = $(1.5 + w + w_D)\ T - 17$
= $6.5 \times 15.6 - 17$; $w = 0$, $w_D = 5$, $T = 15.6$ ns
= 84.4 ns (> 25 ns)

<4> Output floating delay time after $\overline{OE}$ of μPD23C16080BL is deasserted: 25 ns (MAX.)
Minimum value of data output delay time (from $\overline{RD}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DRDOD} (ns) = $(0.5 + i)\ T - 6$
= $2.5 \times 15.6 - 6$; $i = 3$, $T = 15.6$ ns
= 33 ns (> 25 ns)

Caution When a page is hit, the $\overline{RD}$ signal is held low and the page ROM is continuously accessed. In the meantime, no idle state is inserted.

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)
w: Number of wait states inserted by $\overline{WAIT}$
w_D: Number of wait states specified by DWC0 and DWC1 registers
w_{AS}: Number of address setup wait states specified by ASC register
w_{PR}: Number of wait states specified by PRC register
i: Number of idle states

2.5 Connecting Flash Memory

This section shows an example of connecting one flash memory (MB29PL320: 4M × 8 bits/2M x 16 bits) to a 4 MB external memory space with a bus width of 16 bits.

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: MB29PL320TE80 × 1
- $\overline{CS}$ signal used: $\overline{CS2}$
  The flash memory is allocated to 0400000H to 07FFFFFH (V850E/ME2: Blocks 2 and 3, V850E2/ME3: Subareas 02 and 03) of the external memory space.

[Concept and cautions]

- The address bus (A0 to A20) of the MB29PL320TE80 is connected to A1 to A21 of the V850E/ME2 or V850E2/ME3.
- The D0 to D15, $\overline{CE}$, $\overline{OE}$, and $\overline{WE}$ pins of the MB29PL320TE80 are connected to the D0 to D15, $\overline{CS2}$, $\overline{RD}$, and $\overline{WR}$ pins of the V850E/ME2 or V850E2/ME3.
- The RY/$\overline{BY}$ pin of the MB29PL320TE80 is connected to an input port (Pxx) of the V850E/ME2 or V850E2/ME3.

Remarks 1. By connecting the RY/$\overline{BY}$ pin to an input port of the V850E/ME2 or V850E2/ME3, statuses such as completion of writing/erasing can be monitored.

2. Pxx = P10 to P13, P20 to P25, P50 to P55, P65 to P67, P72 to P77, PAH6 to PAH9, PAL0, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0, PCS1, PCS3 to PCS7, PCT0 to PCT3, and PCT7

[Register settings]

- Area of $\overline{CS2}$ to be used and device type:
  V850E/ME2: Blocks 2 and 3, V850E2/ME3: Subareas 02 and 03, SRAM/external I/O
- Wait setting: 2 waits[Note]
- Address setup wait: 3 waits[Note]
- Idle state: 2 states

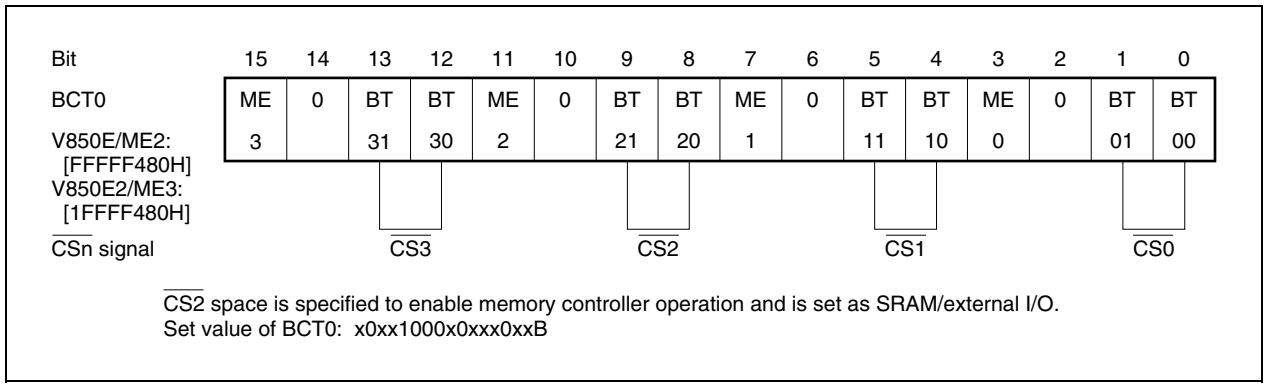
Note In this circuit connection example, the set value of the address setup wait (WAS) may be decreased and the set value of the wait (WD) may be increased.

Figure 2-44. Setting of Chip Area Select Control Register 0 (CSC0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00
V850E2/ME3: [1FFFF060H]																

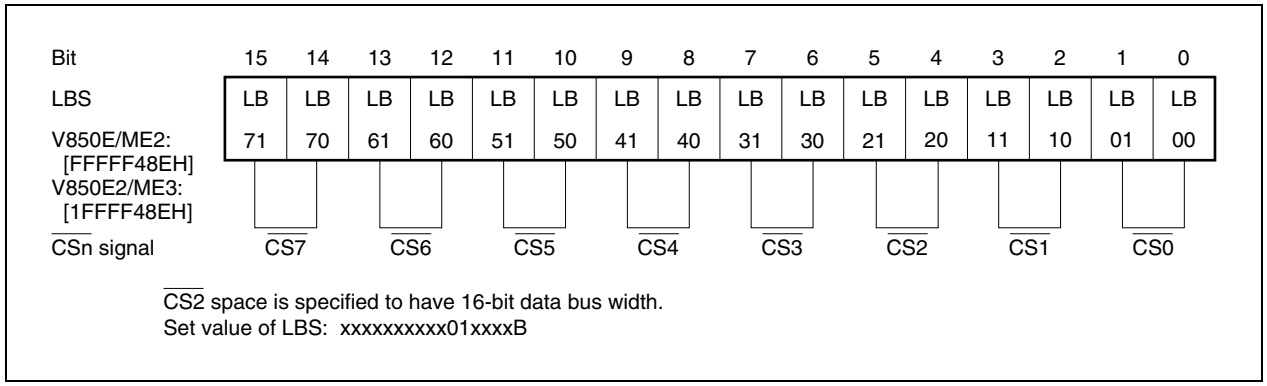
$\overline{CS2}$ output when blocks 2 and 3 of V850E/ME2 or subareas 02 and 03 of V850E2/ME3 are accessed
Set value of CSC0: xxxx1100xxxx00xxB

Figure 2-45. Setting of Bus Cycle Type Configuration Register 0 (BCT0)



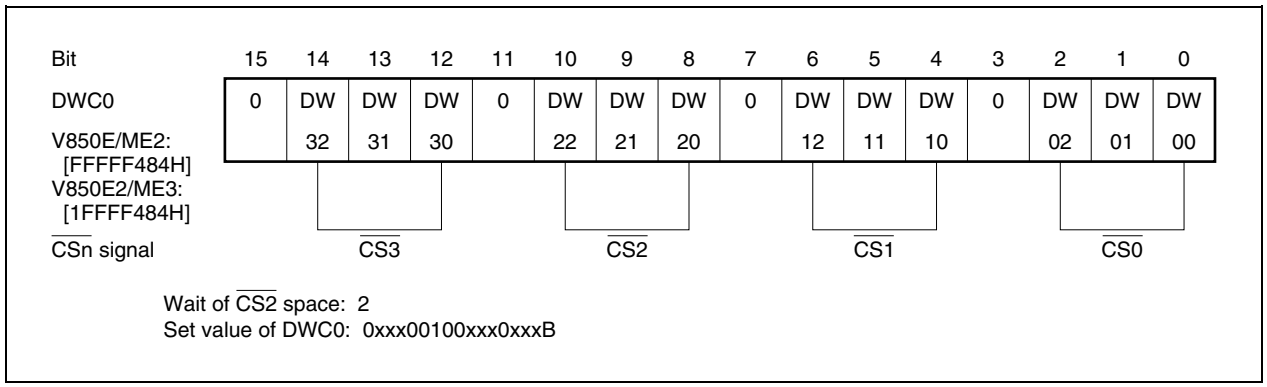
$\overline{CS2}$ space is specified to enable memory controller operation and is set as SRAM/external I/O.
Set value of BCT0: x0xx1000x0xxx0xxB

Figure 2-46. Setting of Local Bus Sizing Control Register (LBS)



$\overline{CS2}$ space is specified to have 16-bit data bus width.
Set value of LBS: xxxxxxxxxx01xxxxB

Figure 2-47. Setting of Data Wait Control Register 0 (DWC0)



Wait of $\overline{CS2}$ space: 2
Set value of DWC0: 0xxx00100xxx0xxxB

Figure 2-48. Setting of Address Setup Wait Control Register (ASC)

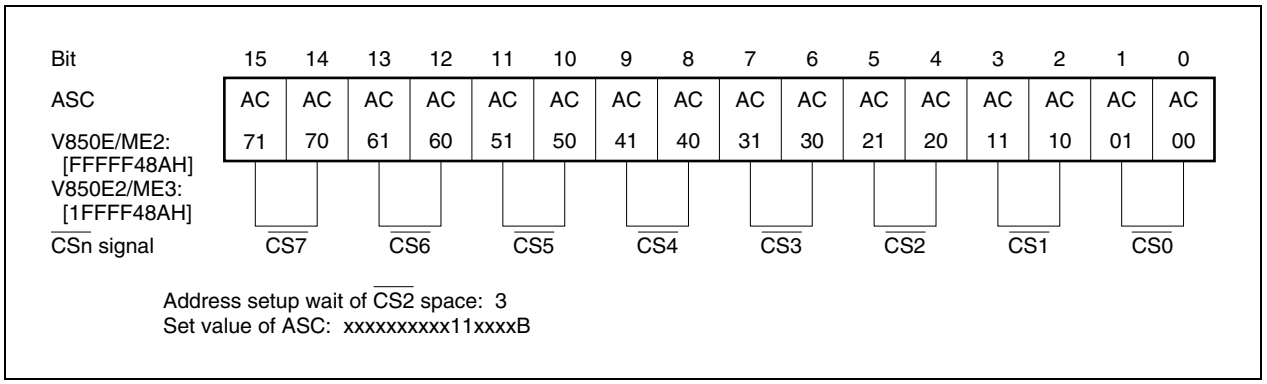


Figure 2-49. Setting of Bus Cycle Control Register (BCC)

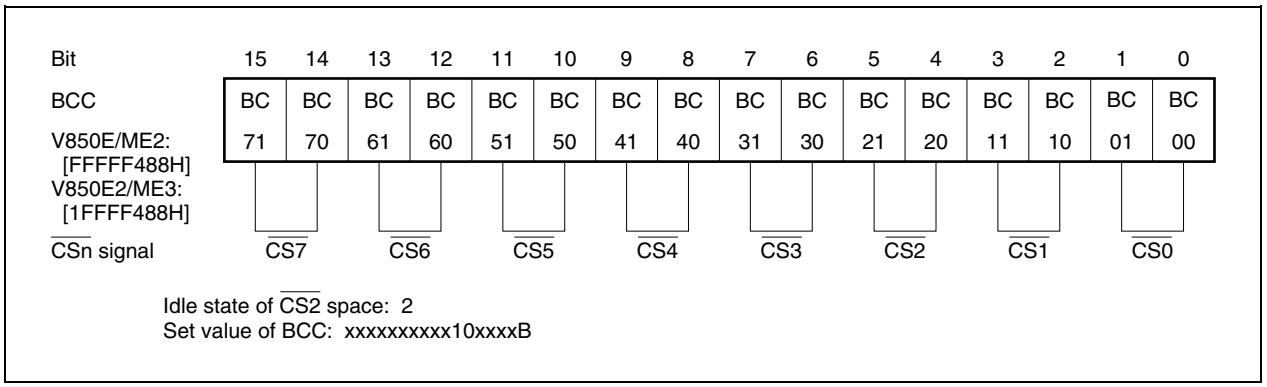
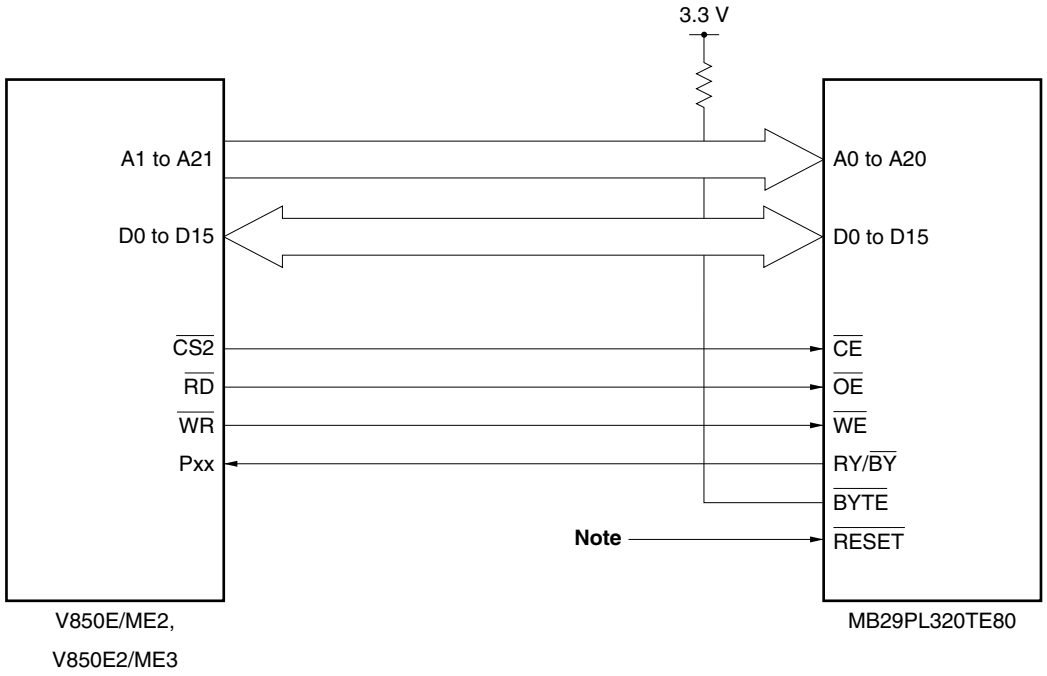


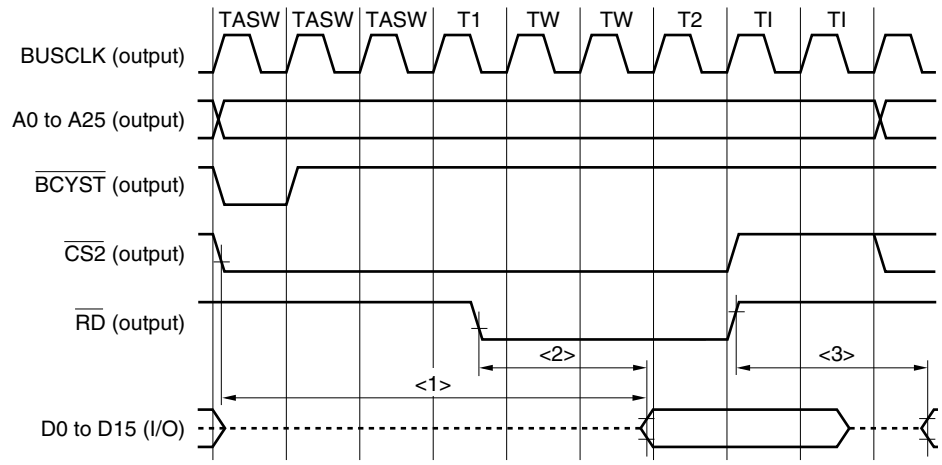
Figure 2-50. Example of MB29PL320TE80 Connection Circuit



Note A signal that is created by the system, such as a power-on reset, is connected to the $\overline{\text{RESET}}$ pin of the MB29PL320TE80.

Remark Pxx = P10 to P13, P20 to P25, P50 to P55, P65 to P67, P72 to P77, PAH6 to PAH9, PAL0, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0, PCS1, PCS3 to PCS7, PCT0 to PCT3, and PCT7

Figure 2-51. Read Operation of MB29PL320TE80



<1> Output delay time after address and $\overline{CE}$ of MB29PL320TE80 are asserted: 80 ns (MAX.)

Maximum value of data input setup time (to address) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SAID} (ns) $= (2 + w + w_D + w_{AS})\,T - 17$

$= 7 \times 15.6 - 17$; $w = 0$, $w_D = 2$, $w_{AS} = 3$, $T = 15.6$ ns

$= 92.2$ ns (> 80 ns)

<2> Output delay time after $\overline{OE}$ of MB29PL320TE80 is asserted: 30 ns (MAX.)

Maximum value of data input setup time (to $\overline{RD}$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SRDID} (ns) $= (1.5 + w + w_D)\,T - 17$

$= 3.5 \times 15.6 - 17$; $w = 0$, $w_D = 2$, $T = 15.6$ ns

$= 37.6$ ns (> 30 ns)

<3> Output floating delay time after $\overline{OE}$ of MB29PL320TE80 is deasserted: 25 ns (MAX.)

Minimum value of data output delay time (from $\overline{RD}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DRDOD} (ns) $= (0.5 + i)\,T - 6$

$= 2.5 \times 15.6 - 6$; $i = 2$, $T = 15.6$ ns

$= 33$ ns (> 25 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)

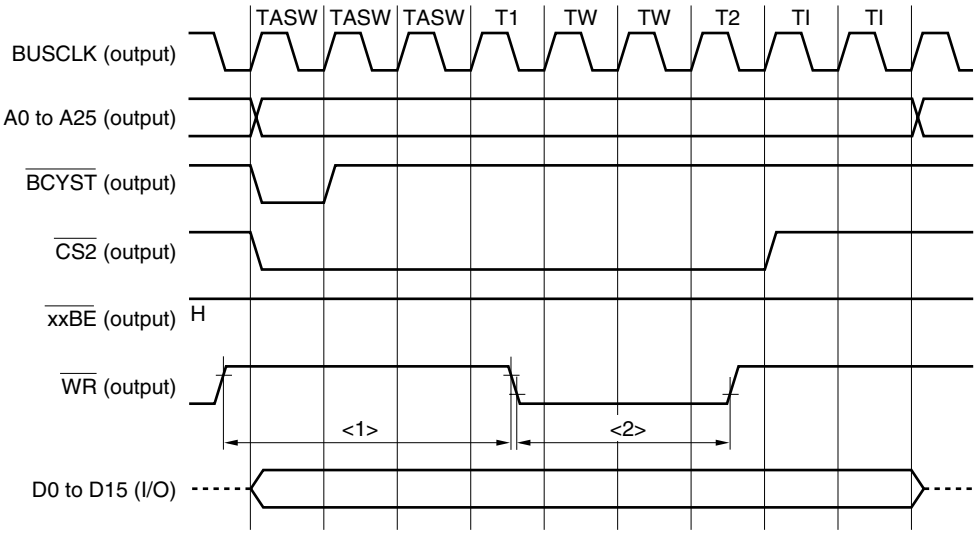
w: Number of wait states inserted by $\overline{WAIT}$

w_D: Number of wait states specified by DWC0 and DWC1 registers

w_{AS}: Number of address setup wait states specified by ASC register

i: Number of idle states

Figure 2-52. Write Operation of MB29PL320TE80



<1> $\overline{WE}$ high-level pulse width of MB29PL320TE80: 25 ns (MIN.)

Minimum value of $\overline{WR}$ high-level width from electrical specifications of V850E/ME2 and V850E2/ME3

t_{WWRH} (ns) $= (1 + i + w_{AS})\ T - 5$

$= 6 \times 15.6 - 5$; i = 2, w_{AS} = 3, T = 15.6 ns

= 88.6 ns (> 25 ns)

<2> $\overline{WE}$ active level pulse width of MB29PL320TE80: 35 ns (MAX.)

Minimum value of $\overline{WR}$ low-level width from electrical specifications of V850E/ME2 and V850E2/ME3

t_{WWRL} (ns) $= (1 + w + w_D)\ T - 5$

$= 3 \times 15.6 - 5$; w = 0, w_D = 2, T = 15.6 ns

= 41.8 ns (> 35 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)

w: Number of wait states inserted by $\overline{WAIT}$

w_D: Number of wait states specified by DWC0 and DWC1 registers

w_{AS}: Number of address setup wait states specified by ASC register

i: Number of idle states

3. xx = LU, LL

2.6 Connecting SDRAM

This section shows an example of connecting two SDRAMs (μPD45128163: 2M × 16 bits × 4 banks) (32 MB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: μPD45128163-A10 × 2
- $\overline{CS}$ signal used: $\overline{CS3}$
  The SDRAMs are allocated to 4000000H to 5FFFFFFH of the external memory space.
  Addresses 6000000H to 7FFFFFFH are images.

  Caution No program can be allocated to this space. Allocate a program to the $\overline{CS1}$ space.

[Concept and cautions]

- Set the access timing to the SCR3 register because the $\overline{CS}$ pin of the μPD45128163-A10 is connected to the $\overline{CS3}$ pin of the V850E/ME2 or V850E2/ME3.
- Because the bus width configuration is 32 bits, the address bus (A0 to A11) of the μPD45128163-A10 is connected to A2 to A13 of the V850E/ME2 or V850E2/ME3. The data bus (DQ0 to DQ15) of the μPD45128163-A10 is connected to D0 to D15 and D16 to D31 of the V850E/ME2 or V850E2/ME3.

  Caution In this circuit connection example, the data bus is directly connected to the SDRAM. If the load of the data bus is heavy, connect signals separated by a bidirectional buffer to a data bus that is connected to a memory other than the SDRAM. The bidirectional buffer is controlled by the $\overline{CSn}$ and $\overline{RD}$ signals.
  - **Enable pin of buffer: Logic sum (OR) of $\overline{CS}$ signals of other than SDRAM**
  - **Direction control of buffer: "Memory → V850E/ME2 or V850E2/ME3" if the $\overline{RD}$ signal is asserted**

- Because the row address of the μPD45128163-A10 is 12 bits and the column address is 9 bits, the address multiplex width (9 bits) is set so that A2 to A13 of the V850E/ME2 or V850E2/ME3 are in the following status when a row address is output.

- Connect the bank select address (BA0 and BA1) of the μPD45128163-A10 to the higher address (A23 and A24 are the higher address of a 32 MB space) of the V850E/ME2 or V850E2/ME3.
- Because the number of refresh cycles of the μPD45128163-A10 is 4096/64 ms, set the refresh interval to 15.6 μs or less.
- The AC characteristics of the μPD45128163-A10 are as follows. Therefore, set the SCR3 register to CAS latency = 2 and BCW = 2. Set the number of idle states to 0.
- Maximum clock at CAS latency = 2: 77 MHz (> BUSCLK = 64 MHz)
- Minimum time from active command to read/write command: 20 ns (< BCW × 2)
- Data float delay time from rising of clock (MIN.): 7 ms

Minimum value of data output delay time (from BUSCLK↑) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DSDOD} (ns) = (0.5 + i) T
= 0.5 × 15.6; i = 0, T = 15.6 ns
= 7.8 ns (> 7 ns)

- Cycle time between commands: 70 ns (< 15.6 × 5[Note] ns)

Note Total of number of active command BUSCLK (1) + Number of wait BUSCLK between commands (1) + Number of latency BUSCLK (2) + Number of read/write command BUSCLK (1)

[Register settings]

- Device type of $\overline{CS3}$: SDRAM
- Bus width of $\overline{CS3}$ space: 32 bits
- Idle state: 0 states

Remark The set values of the DWC0 and ASC registers are meaningless for SDRAM access. The set value of the CSC0 register is meaningless because the area of the $\overline{CS3}$ space is fixed.

Figure 2-53. Setting of Bus Cycle Type Configuration Register 0 (BCT0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCT0	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT
V850E/ME2: [FFFFF480H]	3		31	30	2		21	20	1		11	10	0		01	00
V850E2/ME3: [1FFFF480H]																
$\overline{CSn}$ signal			$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$	

$\overline{CS3}$ space is specified to enable memory controller operation and is set as SDRAM.
Set value of BCT0: 1011x0xxx0xxx0xxB

Figure 2-54. Setting of Local Bus Sizing Control Register (LBS)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2: [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48EH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS3}$ space is specified to have 32-bit data bus width.
Set value of LBS: xxxxxxxx11xxxxxxB

Figure 2-55. Setting of Bus Cycle Control Register (BCC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2: [FFFFF488H]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF488H]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Idle state of $\overline{CS3}$ space: 0
Set value of BCC: xxxxxxxx00xxxxxxB

Figure 2-56. Setting of SDRAM Configuration Register 3 (SCR3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCR3	0	LTM	LTM	LTM	0	0	0	WCF	BCW	BCW	SSO	SSO	RAW	RAW	SAW	SAW
V850E/ME2: [FFFFF4ACH]		32	31	30				3	31	30	31	30	31	30	31	30
V850E2/ME3: [1FFFF4ACH]																

LTM32 to LTM30: 0, 1, 0 (CAS latency = 2)
BCW31, BCW30: 1, 0 (number of wait states between commands: BCW = 2)
SSO31, SSO30: 1, 0 (data bus width = 32 bits)
RAW31, RAW30: 0, 1 (row address width = 12)
SAW31, SAW30: 0, 1 (address multiplex width = 9 bits)
Set value of SCR3: 20A5H

Remark The WCF3 bit can only be read.

Figure 2-57. Setting of SDRAM Refresh Control Register 3 (RFS3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFS3	REN	0	0	0	0	0	RCC	RCC	0	0	RIN	RIN	RIN	RIN	RIN	RIN
V850E/ME2: [FFFFF4AEH]	3						31	30			35	34	33	32	31	30
V850E2/ME3: [1FFFF4AEH]																

REN3: 1 (refresh enabled)
RCC31, RCC30: 0, 0 (refresh count clock: 32/BUSCLK)
RIN35 to RIN30: 0, 1, 1, 1, 1, 0 (interval factor: 31)
Set value of RFS3: 801EH

Remark The refresh interval is as follows with the above setting because 32/BUSCLK = 0.5 μs.

Refresh interval $= 0.5 \times 31$
$= 15.5\ \mu s\ (< 15.6\ \mu s)$

Figure 2-58. Example of μPD45128163-A10 Connection Circuit

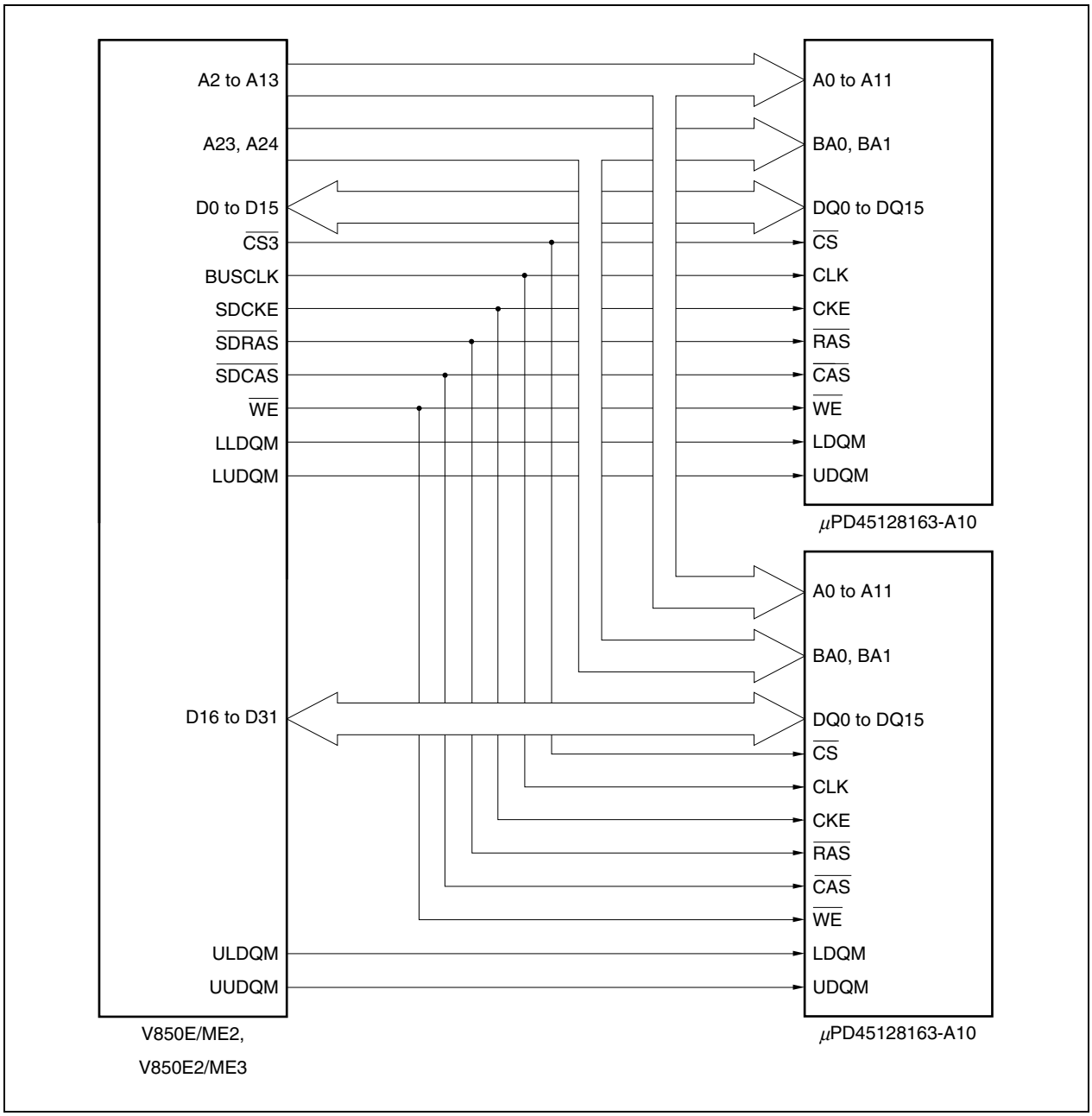
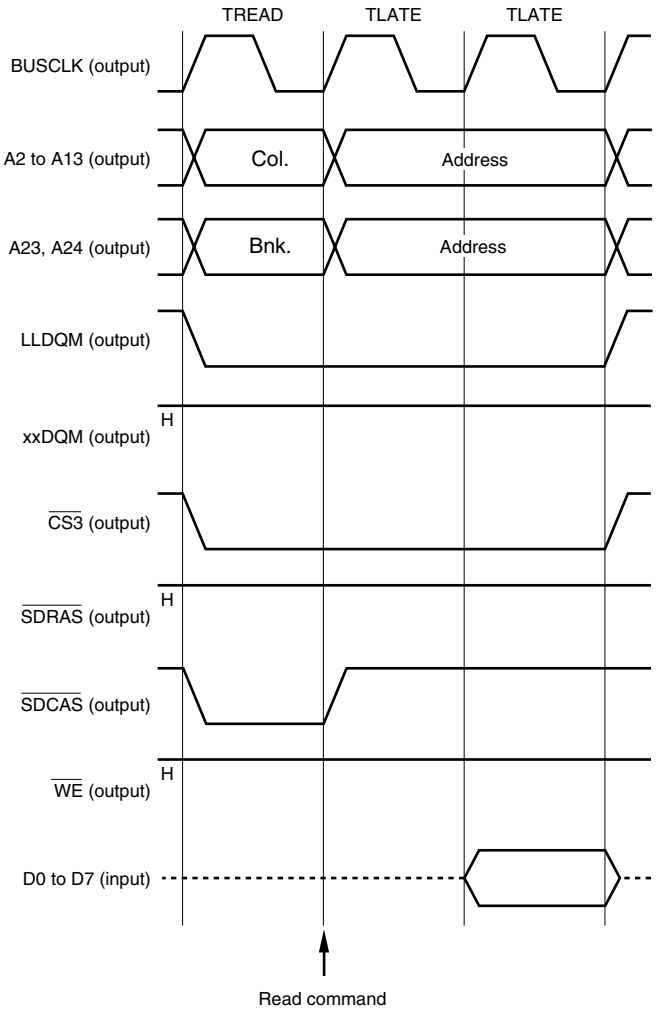


Figure 2-59. Read Operation of μPD45128163-A10 (BCW = 2, Latency = 2, No Idle State Inserted, On-Page Access, Byte Access to D0 to D7)

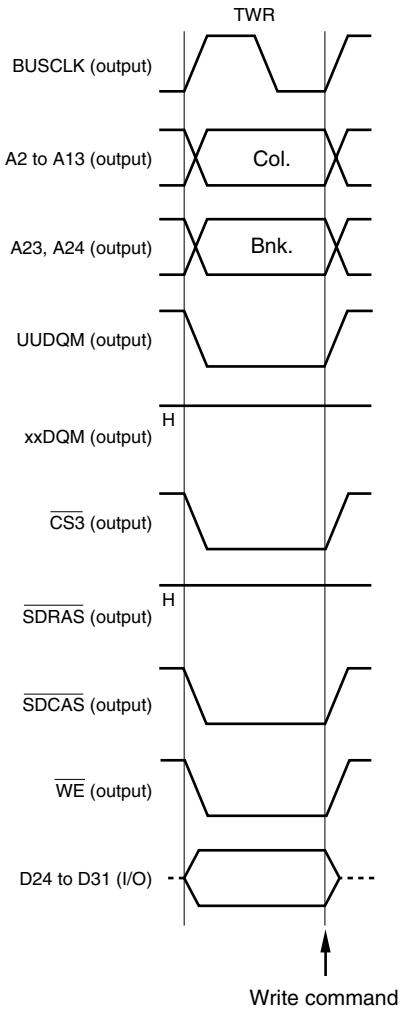


Remarks 1. The dotted line indicates high impedance.

2. Bnk.: Bank address
Col.: Column address

3. xx = UU, UL, LU

Figure 2-60. Write Operation of μPD45128163-A10 (BCW = 2, Latency = 2, No Idle State Inserted, On-Page Access, Byte Access to D24 to D31)

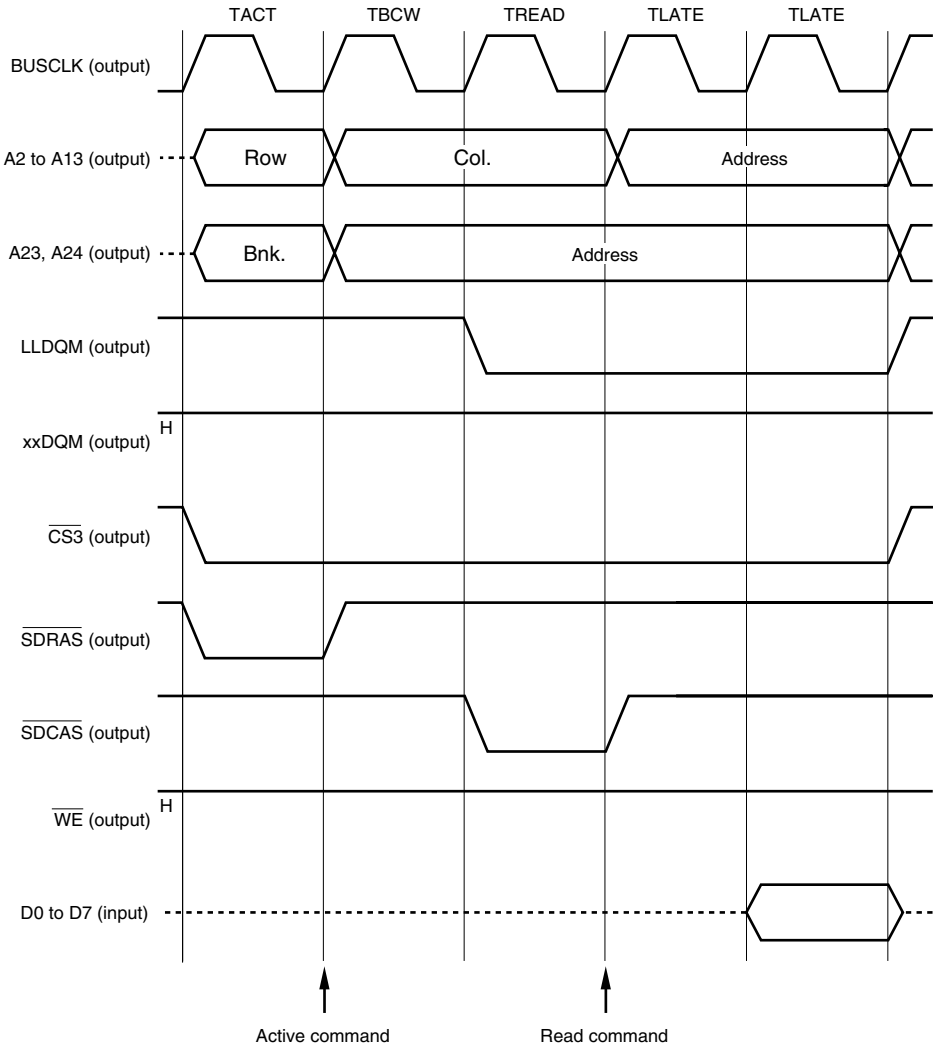


Remarks 1. The dotted line indicates high impedance.

2. Bnk.: Bank address
Col.: Column address

3. xx = UL, LU, LL

Figure 2-61. Read Operation of μPD45128163-A10 (BCW = 2, Latency = 2, No Idle State Inserted, Off-Page Access, Byte Access to D0 to D7)

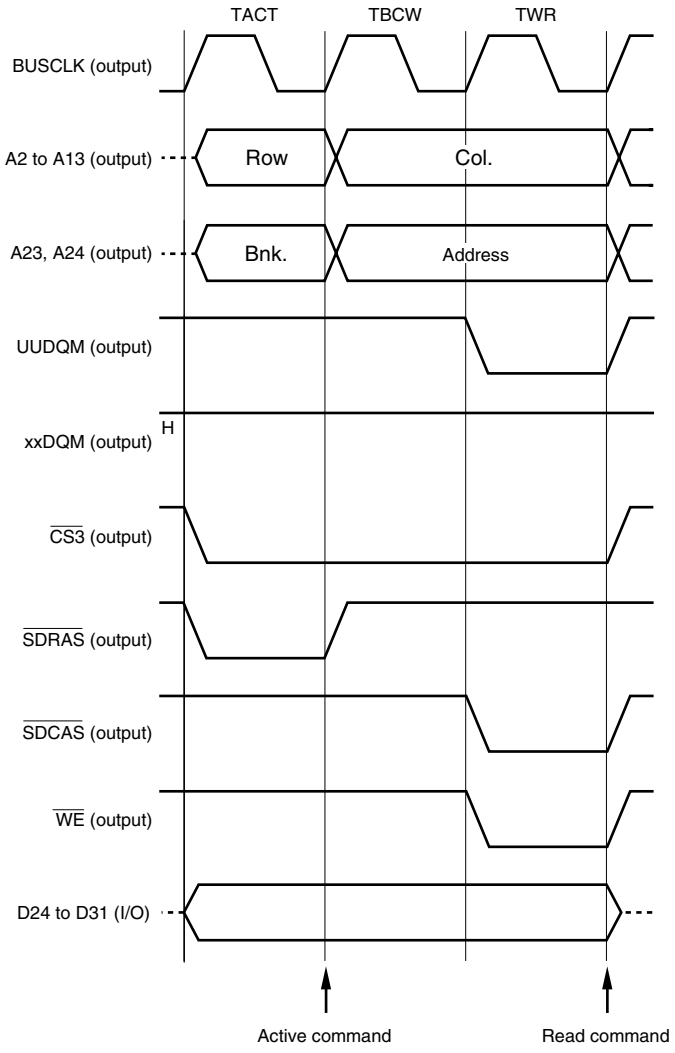


Remarks 1. The dotted line indicates high impedance.

2. Bnk.: Bank address
Col.: Column address
Row: Row address

3. xx = UU, UL, LU

Figure 2-62. Write Operation of μPD45128163-A10 (BCW = 2, Latency = 2, No Idle State Inserted, Off-Page Access, Byte Access to D24 to D31)

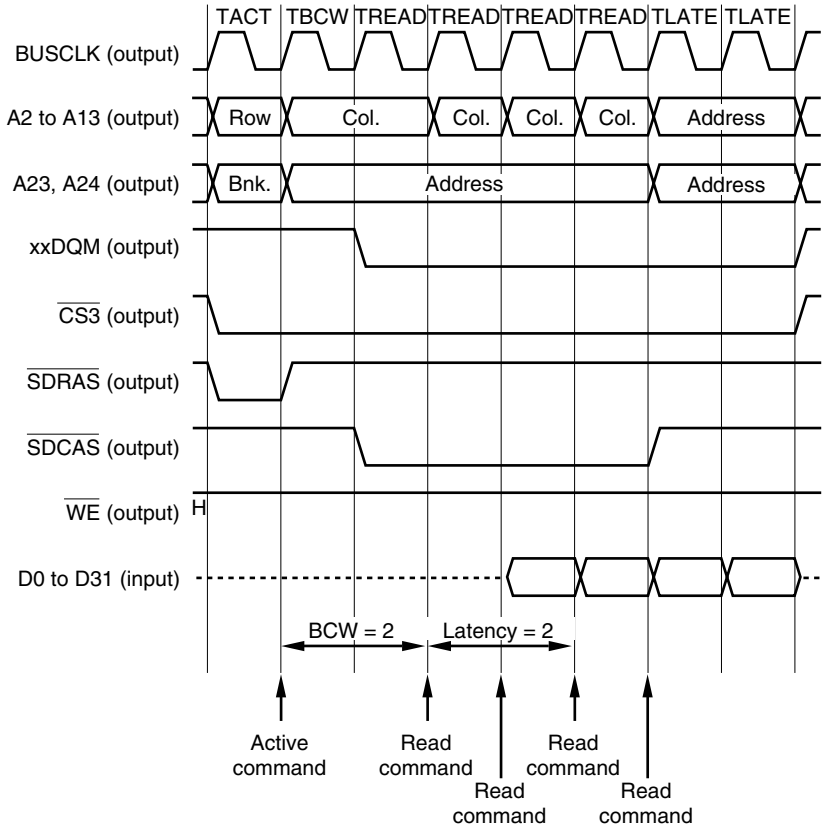


Remarks 1. The dotted line indicates high impedance.

2. Bnk.: Bank address
Col.: Column address
Row: Row address

3. xx = UL, LU, LL

Figure 2-63. Read Operation of μPD45128163-A10 (BCW = 2, Latency = 2, No Idle State Inserted, Off-Page Access, 4-Word Access by Instruction Cache Fill)



Remarks 1. The dotted line indicates high impedance.

2. Bnk.: Bank address
Col.: Column address
Row: Row address

3. xx = UU, UL, LU, LL

2.7 Connecting I/O Device

This section shows an example of connecting an 8-bit I/O device (TL16C550C (UART)).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: TL16C550C × 1
- $\overline{CS}$ signal used: $\overline{CS7}$

  The I/O device is allocated to FE00000H to FFF7FFFH (block 0) of the external memory space of the V850E/ME2. It is allocated to 1FE00000H to 1FFF7FFFH (subarea 07) of the external memory space of the V850E2/ME3.

  The bus width of the $\overline{CS7}$ space is 8 bits.

[Concept and cautions]

- The D0 to D7, A0 to A2, $\overline{CS2}$, $\overline{RD1}$, and $\overline{WR1}$ pins of the TL16C550C are connected to the D0 to D7, A0 to A2, $\overline{CS7}$, $\overline{RD}$, and $\overline{WR}$ pins of the V850E/ME2 or V850E2/ME3.
- The INTRPT pin of the TL16C550C is connected to an input port (Pxx) of the V850E/ME2 or V850E2/ME3.
- The $\overline{ADS}$, $\overline{WR2}$, and $\overline{RD2}$ pins of the TL16C550 are pulled down, and the $\overline{CS0}$ and $\overline{CS1}$ pins are pulled up.

Remarks 1. Pxx = P10 to P13, P20 to P25, P50 to P55, P65 to P67, P72 to P77, PAH0 to PAH9, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0 to PCS6, PCT0 to PCT3, and PCT7

2. In this connection circuit example, the $\overline{CS7}$ space is 8 bits wide. The relationship between the address of the V850E/ME2 or V850E2/ME3 and the TL16C550C with an 8-bit bus width, 16-bit bus width, and 32-bit bus width is as follows. In each case, the TL16C550C is accessed in bytes.

○ When 8-bit bus width is set (A0 to A2 pins of TL16C550C are connected to A0 to A2 pins of V850E/ME2 or V850E2/ME3)

Access of TL16C550C (A2 to A0)	V850E/ME2 Address	V850E2/ME3 Address
Receive buffer/transmit buffer (000)	FE00000H	1FE00000H
Interrupt enable register (001)	FE00001H	1FE00001H
・ ・	・ ・	・ ・
Scratch register (111)	FE00007H	1FE00007H

FE00008H to FFF7FFFH of the V850E/ME2 and 1FE00008H to 1FFF7FFFH of the V850E2/ME3 are images.

○ When 16-bit bus width is set (A0 to A2 and D0 to D7 pins of TL16C550C are connected to A1 to A3 and D0 to D7 pins of V850E/ME2 or V850E2/ME3)

Access of TL16C550C (A2 to A0)	V850E/ME2		V850E2/ME3	
	Address in Little Endian	Address in Big Endian	Address in Little Endian	Address in Big Endian
Receive buffer/transmit buffer (000)	FE00000H	FE00001H	1FE00000H	1FE00001H
Interrupt enable register (001)	FE00002H	FE00003H	1FE00002H	1FE00003H
・ ・	・ ・	・ ・	・ ・	・ ・
Scratch register (111)	FE0000EH	FE0000FH	1FE0000EH	1FE0000FH

FE00010H to FFF7FFFH of the V850E/ME2 and 1FE00010H to 1FFF7FFFH of the V850E2/ME3 are images.

○ When 32-bit bus width is set (A0 to A2 and D0 to D7 pins of TL16C550C are connected to A2 to A4 and D0 to D7 pins of V850E/ME2 or V850E2/ME3)

Access of TL16C550C (A2 to A0)	V850E/ME2		V850E2/ME3	
	Address in Little Endian	Address in Big Endian	Address in Little Endian	Address in Big Endian
Receive buffer/transmit buffer (000)	FE00000H	FE00003H	1FE00000H	1FE00003H
Interrupt enable register (001)	FE00004H	FE00007H	1FE00004H	1FE00007H
・ ・	・ ・	・ ・	・ ・	・ ・
Scratch register (111)	FE0001CH	FE0001FH	1FE0001CH	1FE0001FH

FE00020H to FFF7FFFH of the V850E/ME2 and 1FE00020H to 1FFF7FFFH of the V850E2/ME3 are images.

[Register settings]

- Area of $\overline{CS7}$ to be used and device type:
  V850E/ME2: Block 7, V850E2/ME3: Subarea 07, SRAM/external I/O
- Wait setting: 3 waits
- Address setup wait: 1 wait
- Idle state: 2 states

Figure 2-64. Setting of Chip Area Select Control Register 1 (CSC1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC1	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2: [FFFFF062H]	43	42	41	40	53	52	51	50	63	62	61	60	73	72	71	70
V850E2/ME3: [1FFFF062H]																

$\overline{CS7}$ output when block 7 of V850E/ME2 or subarea 07 of V850E2/ME3 is accessed
Set value of CSC1: xxxxxxx0xxxx0001B

Figure 2-65. Setting of Bus Cycle Type Configuration Register 1 (BCT1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCT1	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT
V850E/ME2: [FFFFF482H]	7		71	70	6		61	60	5		51	50	4		41	40
V850E2/ME3: [1FFFF482H]																
$\overline{CS}$n signal			$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$	

$\overline{CS7}$ space is specified to enable memory controller operation and is set as SRAM/external I/O.
Set value of BCT1: 1000x0xxx0xxx0xxB

Figure 2-66. Setting of Local Bus Sizing Control Register (LBS)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2: [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3: [1FFFF48EH]																
$\overline{CS}$n signal	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS7}$ space is specified to have 8-bit data bus width.
Set value of LBS: 00xxxxxxxxxxxxxxB

Figure 2-67. Setting of Data Wait Control Register 1 (DWC1)

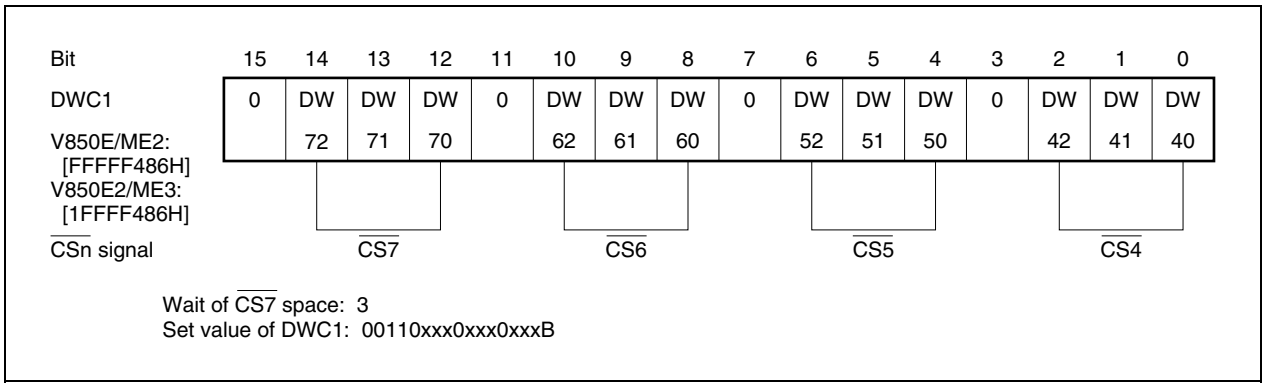


Figure 2-68. Setting of Address Setup Wait Control Register (ASC)

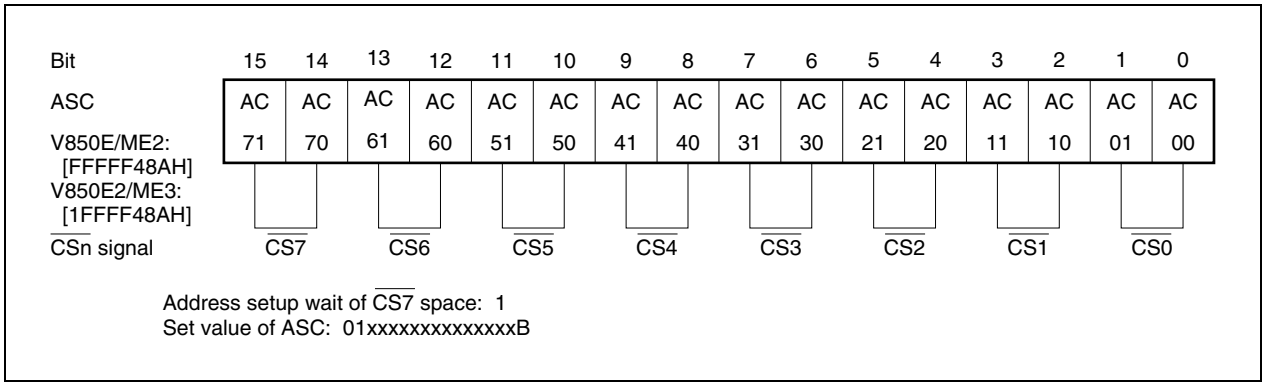


Figure 2-69. Setting of Bus Cycle Control Register (BCC)

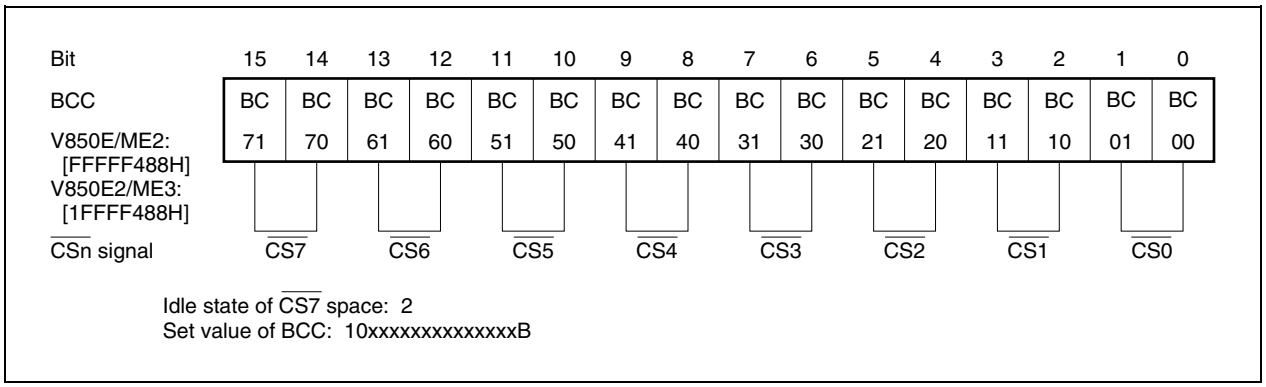
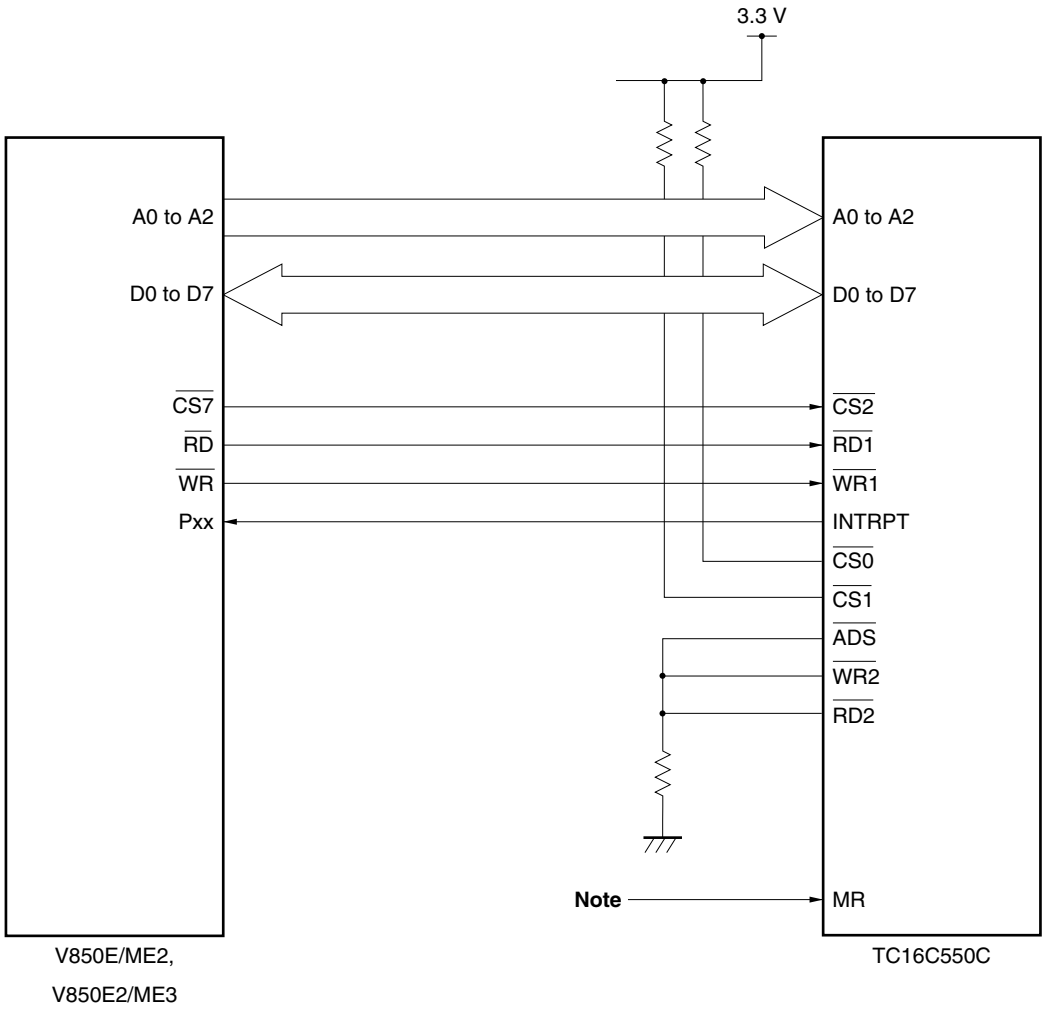


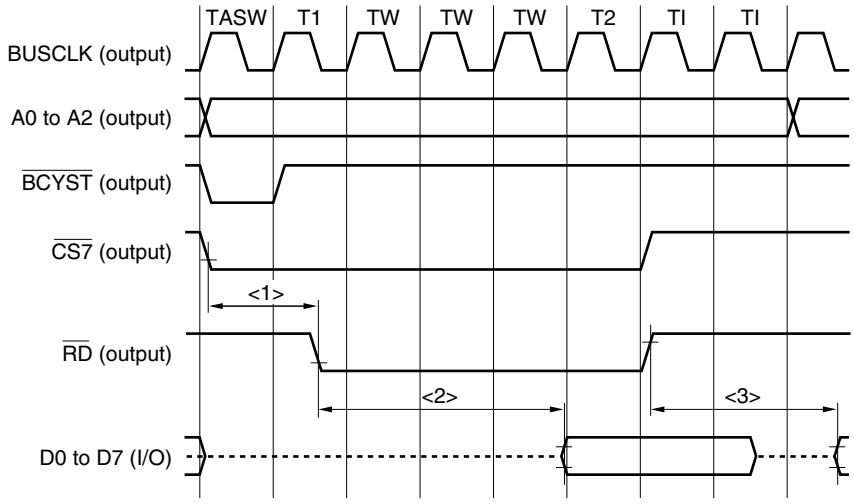
Figure 2-70. Example of TL16C550C Connection Circuit



Note Connect an I/O device reset signal created by the system to the MR pin of the TL16C550C.

Remark Pxx = P10 to P13, P20 to P25, P50 to P55, P65 to P67, P72 to P77, PAH0 to PAH9, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0 to PCS6, PCT0 to PCT3, and PCT7

Figure 2-71. Read Operation of TL16C550C



<1> Delay time from when address and $\overline{CS2}$ of TL16C550C are asserted until $\overline{RD1}$ is asserted: 7 ns (MIN.)

Minimum value of $\overline{CS2} \rightarrow \overline{RD}\downarrow$ delay time from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DARD} (ns) = (0.5 + w_{AS}) T – 7.5

= 1.5 × 15.6 – 7.5; w_{AS} = 1, T = 15.6 ns

= 15.9 ns (> 7 ns)

<2> Data output delay time from when $\overline{RD1}$ of TL16C550C is asserted: 45 ns (MAX.)

Maximum value of data input setup time (to $\overline{RD}$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SRDID} (ns) = (1.5 + w + w_D) T –17

= 4.5 × 15.6 – 17; w = 0, w_D = 3, T = 15.6 ns

= 53.2 ns (> 45 ns)

<3> Output floating delay time from when $\overline{RD1}$ of TL16C550C is deasserted: 20 ns (MAX.)

Minimum value of data output delay time (from $\overline{RD}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{DRDOD} (ns) = (0.5 + i) T – 6

= 2.5 × 15.6 – 6; i = 2, T = 15.6 ns

= 33 ns (> 20 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)

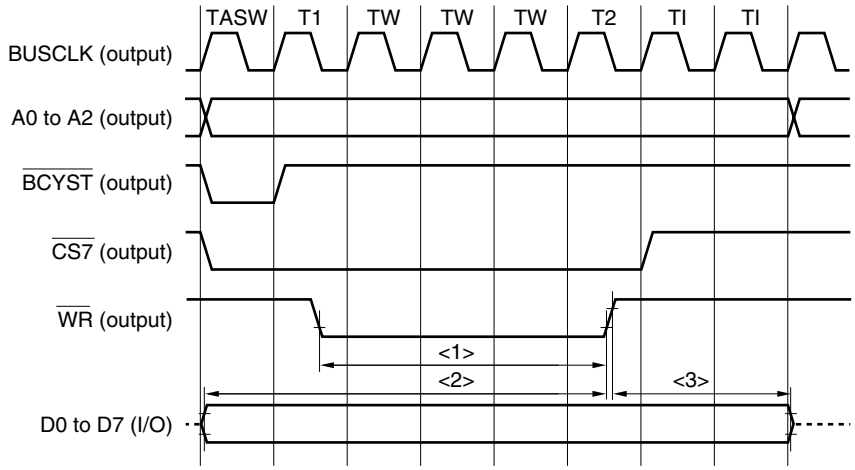
w: Number of wait states inserted by $\overline{WAIT}$

w_D: Number of wait states specified by DWC0 and DWC1 registers

w_{AS}: Number of address setup wait states specified by ASC register

i: Number of idle states

Figure 2-72. Write Operation of TL16C550C



<1> $\overline{WR1}$ active pulse width of TL16C550C: 40 ns (MIN.)

Minimum value of $\overline{WR}$ low-level width from electrical specifications of V850E/ME2 and V850E2/ME3

t_{WWRL} (ns) = $(1 + w + w_D) T - 5$

= $4 \times 15.6 - 5$; $w = 0$, $w_D = 3$, $T = 15.6$ ns

= 57.4 ns (> 40 ns)

<2> Data setup time when $\overline{WR1}$ of TL16C550C rises: 15 ns (MIN.)

Minimum value of data output setup time (to $\overline{WR}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{SDDWR} (ns) = $(1.5 + w + w_D + w_{AS}) T - 5$

= $5.5 \times 15.6 - 5$; $w = 0$, $w_D = 3$, $w_{AS} = 1$, $T = 15.6$ ns

= 80.8 ns (> 15 ns)

<3> Data hold time when $\overline{WR1}$ of TL16C550C rises: 5 ns (MIN.)

Minimum value of data output hold time (from $\overline{WR}\uparrow$) from electrical specifications of V850E/ME2 and V850E2/ME3

t_{HWROD} (ns) = $(0.5 + i) T - 5.5$

= $2.5 \times 15.6 - 5.5$; $i = 2$, $T = 15.6$ ns

= 33.5 ns (> 5 ns)

Remarks 1. The dotted line indicates high impedance.

2. T: t_{CYK} (BUSCLK output cycle)

w: Number of wait states inserted by $\overline{WAIT}$

w_D: Number of wait states specified by DWC0 and DWC1 registers

w_{AS}: Number of address setup wait states specified by ASC register

i: Number of idle states

2.8 Example of Control by $\overline{\text{WAIT}}$ Pin

2.8.1 Connecting dual-port RAM

This section shows an example of connecting one dual-port RAM (IDT70V09L: 128K × 8 bits) (128 KB external memory space).

[Circuit configuration]

- BUSCLK: 64 MHz
- Connected device: IDT70V09L15 × 1
- $\overline{\text{CS}}$ signal used: $\overline{\text{CS5}}$
  The dual-port RAM is allocated to FA00000H to FA1FFFFH (block name) of the external memory space of the V850E/ME2.
  FA20000H to FBFFFFFH are images.
  The dual-port RAM is allocated to 1FA00000H to 1FA1FFFFH (subarea 05) of the external memory space of the V850E2/ME3.
  1FA20000H to 1FBFFFFFH are images.

[Concept and cautions]

- In the same manner as when connecting 8-bit SRAM, the A0L to A16L, I/O0L to I/O7L, $\overline{\text{CE0L}}$, $\overline{\text{OEL}}$, and R/$\overline{\text{WL}}$ pins of the IDT70V09L are connected to the A0 to A16, D0 to D7, $\overline{\text{CS5}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ pins of the V850E/ME2 or V850E2/ME3, respectively.
- Because it takes the $\overline{\text{BUSYL}}$ pin of the IDT70V09L15 15 ns (MAX.) to determine data (i.e., whether the $\overline{\text{BUSYL}}$ signal is asserted) after the $\overline{\text{CE0L}}$ signal is asserted, and because the $\overline{\text{WAIT}}$ setup time (to the $\overline{\text{CS}}$ signal) is "(1 + WAS) T – 17" from the electrical specifications of the V850E/ME2 or V850E2/ME3, the DWC1 register is set to 2 wait states.

  Caution In this circuit connection example, the condition is satisfied even if the address setup wait setting is 2, but the data wait must be set to 2 because of the relationship with the other AC characteristics.

- When the $\overline{\text{BUSYL}}$ signal of the IDT70V09L is asserted, it takes 15 ns (MAX.) to determine data after the $\overline{\text{BUSYL}}$ signal rises (release of $\overline{\text{BUSYL}}$ signal). Therefore, the logical sum of a signal delayed one clock by hardware and the $\overline{\text{BUSYL}}$ signal is connected to the $\overline{\text{WAIT}}$ signal of the V850E/ME2 or V850E2/ME3.
- The M/S pin of the IDT70V09L is pulled up, and the $\overline{\text{SEML}}$ and $\overline{\text{INTL}}$ pins are not used.

  Remark M/S: This pin specifies the master and a slave when two or more IDT70V09L15s are connected. Because only one IDT70V09L15 is used in this circuit connection example, it is specified as the master.
  $\overline{\text{SEML}}$: This is a select pin when the semaphore function is used. In this circuit connection example, the semaphore function is not used.
  $\overline{\text{INTL}}$: This is an interrupt request signal that is asserted when the processor on the write side accesses a specific area. This pin is not used and is left open in this circuit connection example.

[Register settings]

The set values of the registers used in this circuit connection example are briefly explained below.

Table 2-2. Connection with IDT70V09L15

Register Name	Set Value	Function
CSC1	xxxx0100xxxxx0xxB	$\overline{CS5}$ output when block 5 of V850E/ME2 or subarea 05 of V850E2/ME3 is accessed
BCT1	x0xxx0xx1000x0xxB	$\overline{CS5}$: Memory controller enabled, SRAM/external I/O
LBS	xxxx00xxxxxxxxxxB	$\overline{CS5}$: 8 bits
DWC1	0xxx0xxx00100xxxB	$\overline{CS5}$: 2 data waits
ASC	xxxx00xxxxxxxxxxB	$\overline{CS5}$: 0 address setup waits
BCC	xxxx01xxxxxxxxxxB	$\overline{CS5}$: 1 idle state

Figure 2-73. Example of IDT70V09L15 Circuit Connection

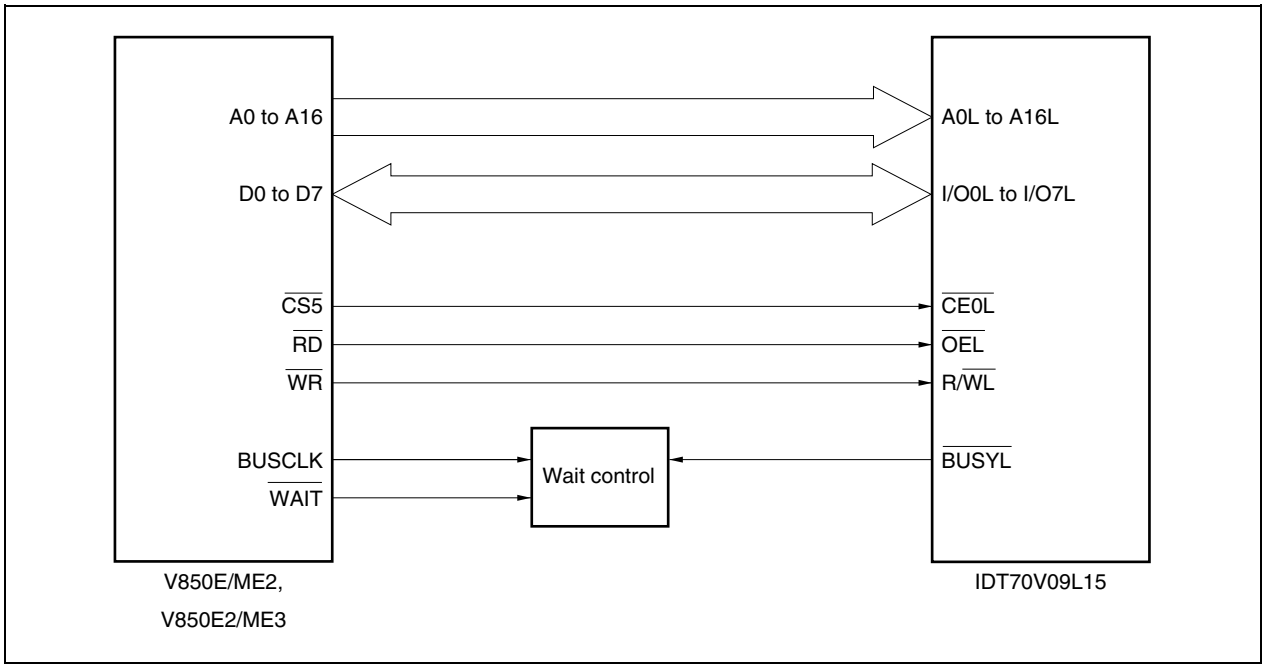


Figure 2-74. Circuit Configuration of Wait Control Block

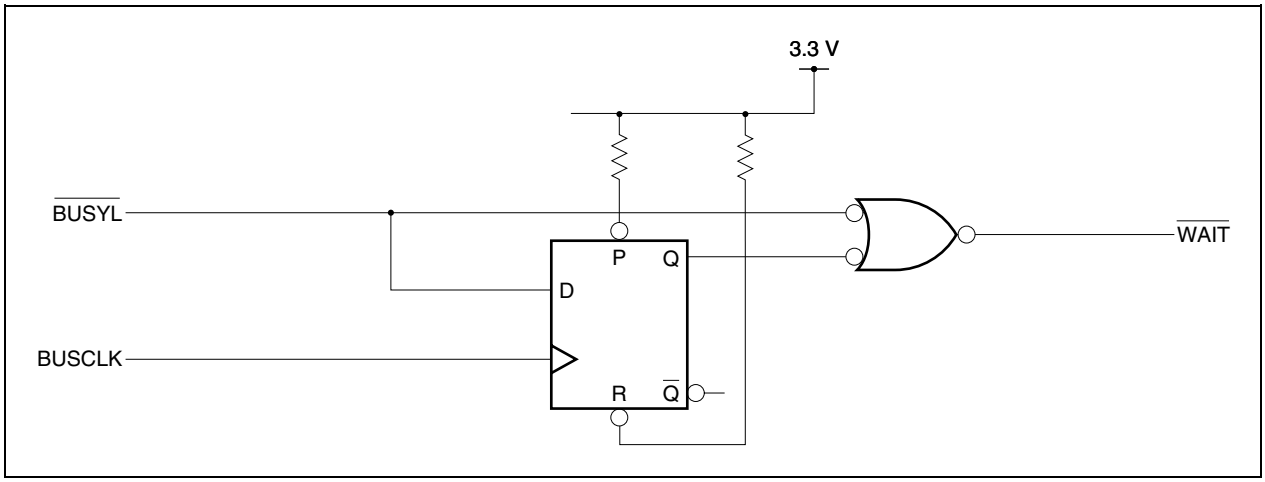
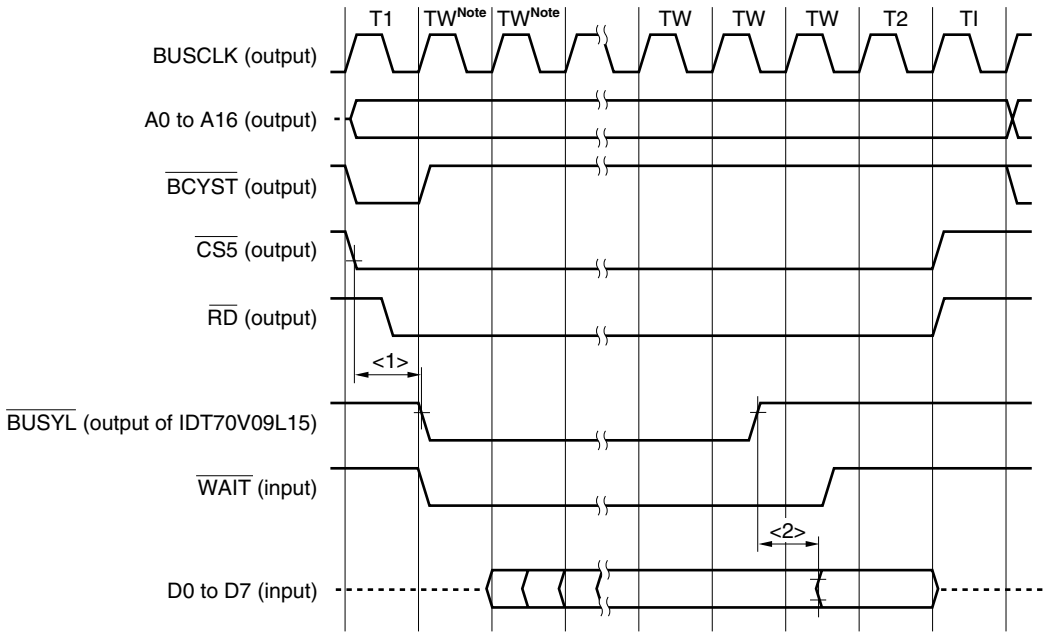


Figure 2-75. Read Operation of IDT70V09L15 (When $\overline{\text{BUSYL}}$ Signal Is Active)



<1> Time from when $\overline{\text{CE0L}}$ of IDT70V09L15 is asserted until $\overline{\text{BUSYL}}$ is determined: 15 ns (MAX.)

<2> Time from when $\overline{\text{BUSYL}}$ of IDT70V09L15 is released until data is determined: 15 ns (MAX.)

Note Two wait states are always inserted because of the setting of the DWC1 register.

Remark The dotted line indicates high impedance.

2.8.2 Connecting low-speed device

This section shows an example of connecting a low-speed device (SRAM, external ROM, or external I/O) whose access time is insufficient with the programmable wait setting of the V850E/ME2 and V850E2/ME3 (address setup wait, data wait, idle state), to the $\overline{\text{WAIT}}$ pin (example of wait controller).

[Circuit configuration]

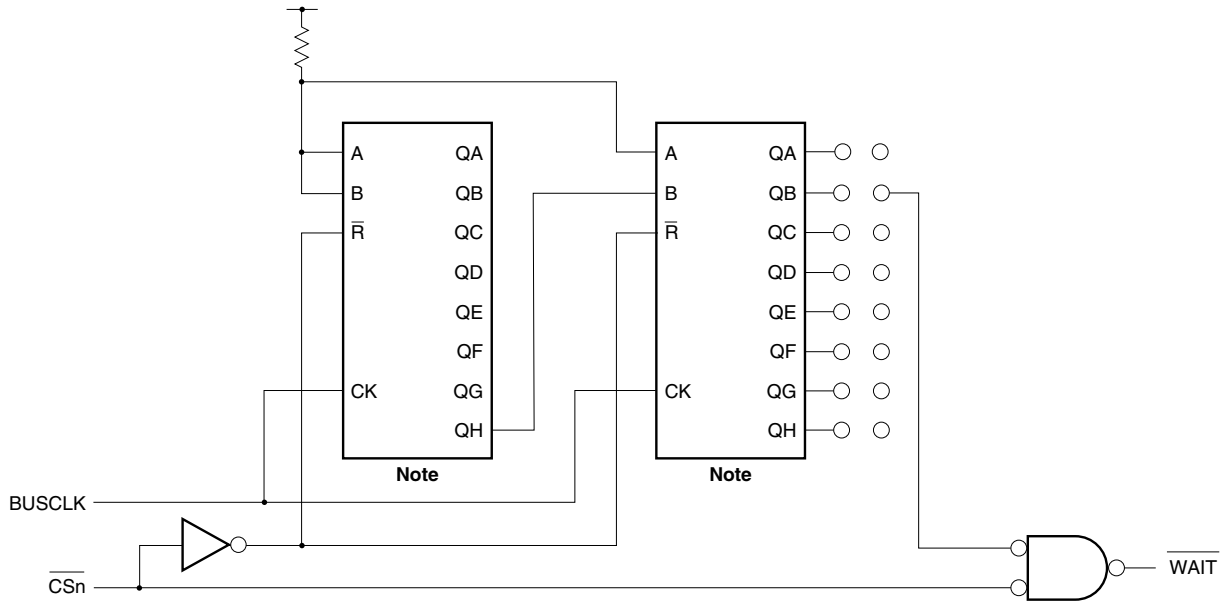
- BUSCLK: Arbitrary
- Device connected: Low-speed ROM whose access time is insufficient with programmable wait setting of V850E/ME2
- $\overline{\text{CS}}$ signal used: $\overline{\text{CSn}}$ (n = 0 to 7)

[Concept and cautions]

The $\overline{\text{WAIT}}$ signal of the V850E/ME2 and V850E2/ME3 is asserted as soon as the $\overline{\text{CSn}}$ signal used has been asserted, the necessary number of BUSCLKs is counted by a counter, and then the $\overline{\text{WAIT}}$ signal is deasserted.

Caution If the setup time cannot be secured when the first $\overline{\text{WAIT}}$ signal is sampled, insert address setup wait or data wait state by software.

Figure 2-76. Wait Controller Configuration



Note It is assumed that a shift register equivalent to 74164 is used.

Remark n = 0 to 7

2.9 Connecting 5 V Device

This section shows an example of connecting 5 V EPROM (M27C800 (512K × 16 bits)). It only gives the concept of connection and a circuit example.

Remark The concept of the register settings is the same as in **2.3 Connecting PROM**.

[Circuit configuration]

- BUSCLK: Arbitrary
- Connected device: M27C800-100 × 1
- $\overline{CS}$ signal used: $\overline{CS0}$

  The EPROM is allocated to 0100000H to 017FFFFH (block 0) of the external memory space of the V850E/ME2.

  0180000H to 01FFFFFH are images.

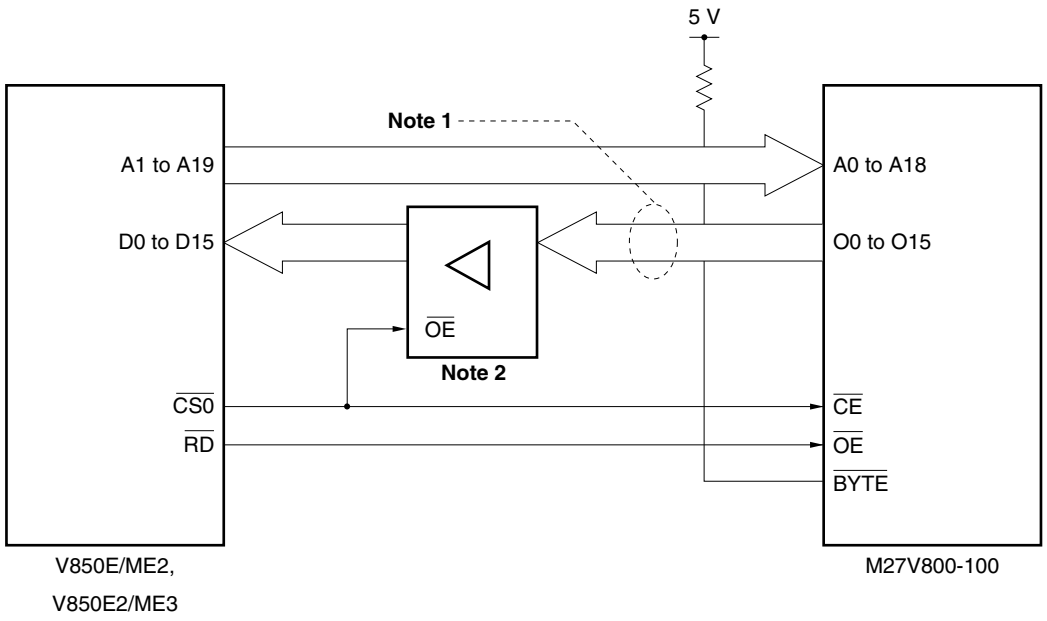
  The EEPROM is allocated to 0000000H to 007FFFFH (subarea 00) of the external memory space of the V850E2/ME3.

  0080000H to 00FFFFFH are images.

[Concept and cautions]

- The O0 to O15 pins of the M27C800-100 are connected to the D0 to D15 pins of the V850E/ME2 or V850E2/ME3 via a level converter device to satisfy the maximum value of the high-level input voltage (V_{IH}), "EV_{DD} + 0.3 V", of the V850E/ME2 and V850E2/ME3. In this circuit connection example, the 74LCX16244 is used as the level converter device.
- Connection of the A0 to A18, $\overline{CE}$, $\overline{OE}$, and $\overline{BYTE}$ pins of the M27C800-100 is the same as in **2.3 Connecting PROM**.

Figure 2-77. Example of M27C800-100 Connection Circuit



Notes 1. These pins are pulled up to 5 V to prevent an input high-impedance state.

2. Use of the 74LCX16244 or equivalent circuit is assumed. Take the delay time of the attached circuit into consideration when calculating the wait time.

2.10 Connection for DMA Flyby Transfer

This section shows an example of circuit connection to execute DMA flyby transfer between SRAM and external I/O. The external I/O consists of a gate array, etc.

[Circuit configuration]

- Memory for DMA flyby transfer: SRAM
- I/O device for DMA flyby transfer: External I/O connected to $\overline{CS7}$ space
- DMA channel number: Channel 0
- DMA start source: External trigger ($\overline{DMARQ0}$)

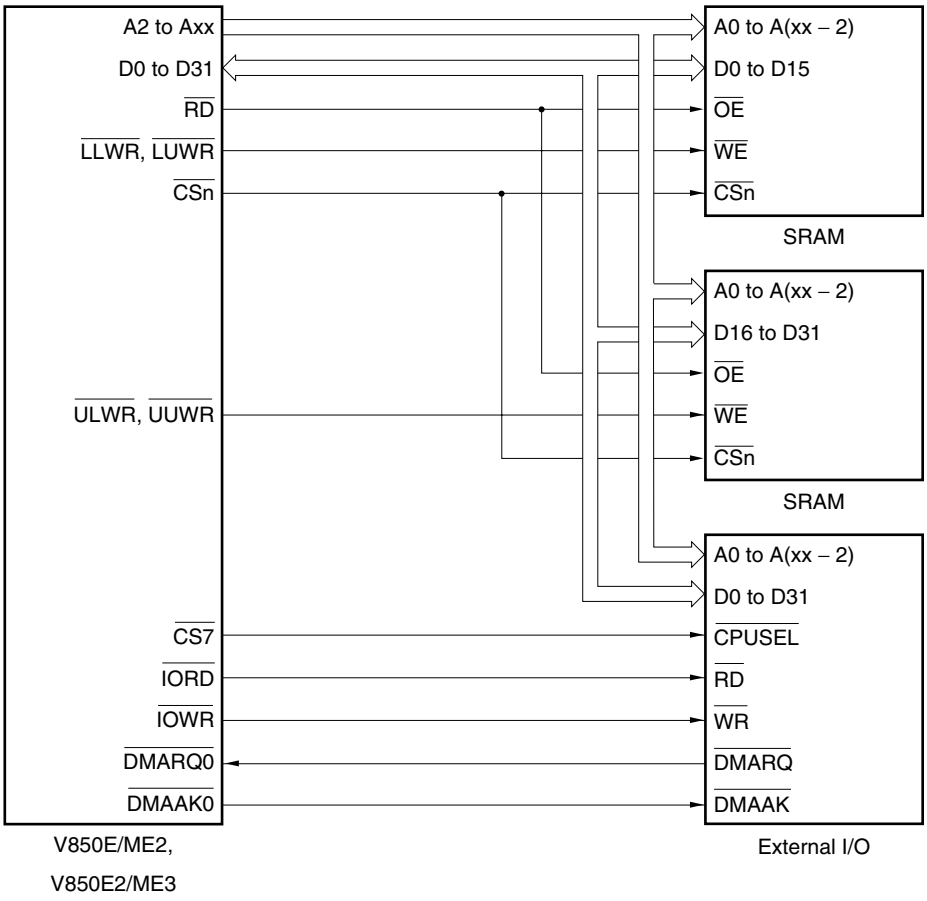
[Concept and cautions]

- DMA flyby transfer is used to transfer data between external memory and external I/O in one cycle. The V850E/ME2 and V850E2/ME3 output the address of the external memory. The external I/O is configured so that it is selected by the $\overline{DMAAK0}$ signal.
- Each internal controller of the external I/O is configured so that the CPU of the V850E/ME2 or V850E2/ME3 can access it by using the $\overline{IORD}$ or $\overline{IOWR}$ signal, asserting the $\overline{CS7}$ signal. Because the $\overline{IORD}$ and $\overline{IOWR}$ pins function alternately as the $\overline{CS2}$ and $\overline{CS5}$ pins, respectively, no memory can be allocated to the $\overline{CS2}$ and $\overline{CS5}$ spaces. The BCP register is set to 08H.

  Remark With an external I/O that does not have to be controlled by the CPU (external I/O that is not accessed by any means other than DMA flyby transfer), the BCP register may be set to 00H.

- During DMA flyby transfer, the V850E/ME2 and V850E2/ME3 issue the cycle set by the external memory. However, the wait set by the DWC0, DWC1, and PRC registers is invalid, and the number of wait states set by the DMA flyby transfer wait control register (FWC) is valid.
- In this circuit connection example, DMA is started by an external trigger ($\overline{DMARQ0}$). When using a software trigger or a trigger activated by an interrupt from the internal peripheral I/O, the circuit may be configured without the $\overline{DMARQ0}$ signal.

Figure 2-78. Example of Circuit Connection for DMA Flyby Transfer



Caution The direction of DMA flyby transfer is as follows.

If DMAAK and RD signals are asserted: External I/O → SRAM

If DMAAK and WR signals are asserted: SRAM → external I/O

2.11 System Configuration Example and CSCn Register Setting

2.11.1 V850E/ME2

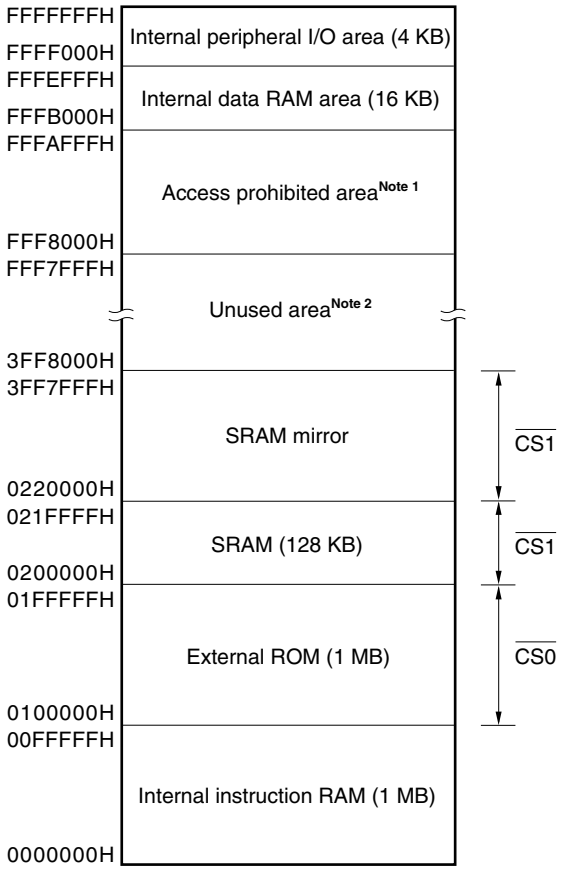
This section shows examples of the configuration of the memory that is connected to the external memory space, and the settings of the CSC0 and CSC1 registers.

The first memory access of the V850E/ME2 is an instruction fetch cycle to access 0100000H. Therefore, external ROM must be allocated to the $\overline{CS0}$ space.

(1) Example of connecting external ROM (1 MB) and SRAM (128 KB)

- $\overline{CS0}$ space: External ROM (1 MB)
- $\overline{CS1}$ space: SRAM (128 KB)
- Set value of CSC0 register: 0001H

Figure 2-79. Example of Memory Map with External ROM (1 MB) and SRAM (128 KB) Connected



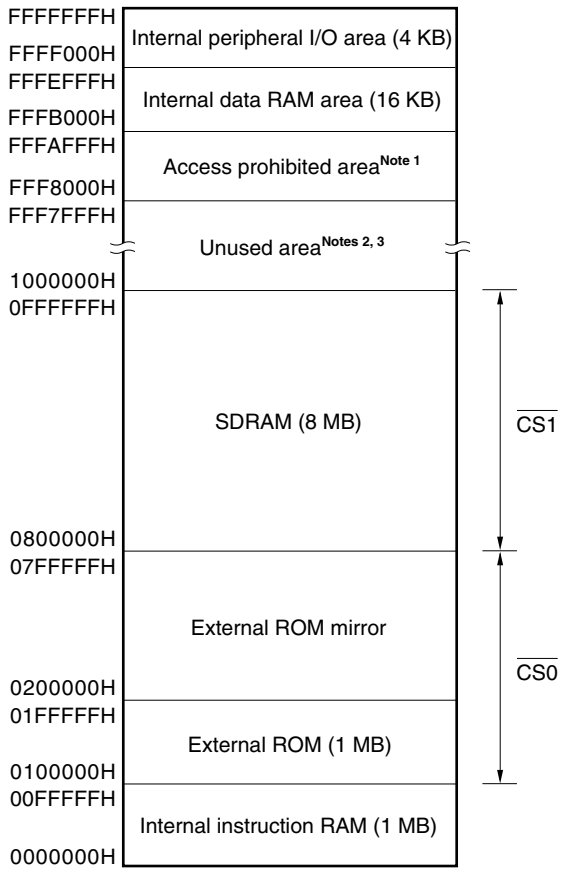
Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

2. The image of addresses FFF8000H to FFFFFFFH can be viewed at addresses 3FF8000H to 3FFFFFFH.

(2) Example of connecting external ROM (1 MB) and SDRAM (8 MB)

- $\overline{CS0}$ space: External ROM (1 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)
- Set value of CSC0 register: 000FH

Figure 2-80. Example of Memory Map with External ROM (1 MB) and SDRAM (8 MB) Connected



Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

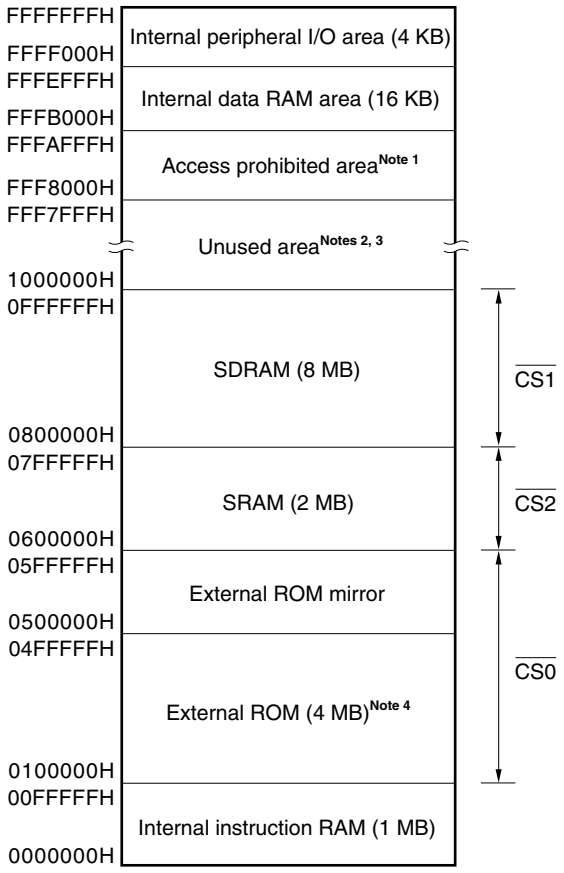
2. The image of addresses FFF8000H to FFFFFFFH can be viewed at addresses 3FF8000H to 3FFFFFFH.

3. The image of the SDRAM ($\overline{CS1}$) space can be viewed at addresses 1000000H to 3FF7FFFH.

(3) Example of connecting external ROM (4 MB), SRAM (2 MB), and SDRAM (8 MB)

- $\overline{CS0}$ space: External ROM (4 MB)
- $\overline{CS2}$ space: SRAM (2 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)
- Set value of CSC0 register: 0807H

Figure 2-81. Example of Memory Map with External ROM (4 MB), SRAM (2 MB), and SDRAM (8 MB) Connected



Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

2. The image of addresses FFF8000H to FFFFFFFH can be viewed at addresses 3FF8000H to 3FFFFFFH.

3. The image of the SDRAM ($\overline{CS1}$) space can be viewed at addresses 1000000H to 3FF7FFFH.

4. The relationship between the offset addresses of the external ROM and the addresses of the V850E/ME2 when 32 Mb external ROM is used is shown below.

Addresses of V850E/ME2	Addresses of External ROM
0100000H to 01FFFFFH	100000H to 1FFFFFH
0200000H to 03FFFFFH	200000H to 3FFFFFH
0400000H to 04FFFFFH	000000H to 0FFFFFH
0500000H to 05FFFFFH	100000H to 1FFFFFH

(4) Example of connecting external ROM (4 MB), SDRAM (8 MB), SRAM for data space (2 MB), SDRAM for data space (128 MB), and external I/O (256 KB)

Caution In this connection example, it is assumed that the external I/O is subject to DMA flyby transfer, and that $\overline{CS2}$ and $\overline{CS5}$ are not used.

(a) Program locatable space

- $\overline{CS0}$ space: External ROM (4 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)

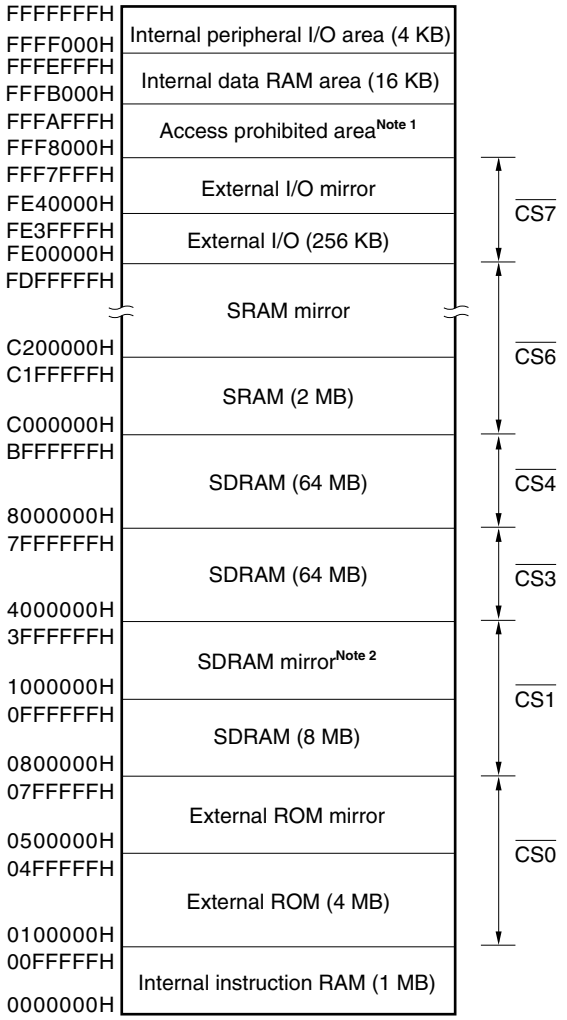
(b) Data space

- $\overline{CS3}$ space: SDRAM (64 MB)
- $\overline{CS4}$ space: SDRAM (64 MB)
- $\overline{CS6}$ space: SRAM (2 MB)
- $\overline{CS7}$ space: External I/O (256 KB)

(c) Setting of CSC0 and CSC1 registers

- Set value of CSC0 register: 000FH
- Set value of CSC1 register: 0001H

Figure 2-82. Example of Memory Map with External ROM (4 MB), SDRAM (8 MB), SRAM for Data Space (2 MB), SDRAM for Data Space (128 MB), and External I/O (256 KB) Connected



Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

2. The image of addresses FFF8000H to FFFFFFFH can be viewed at addresses 3FF8000H to 3FFFFFFH.

2.11.2 V850E2/ME3

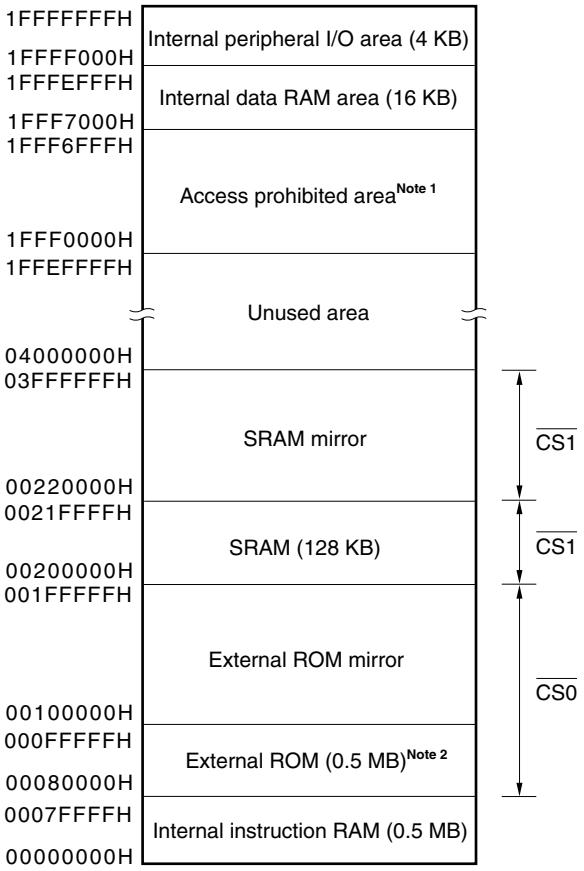
This section shows examples of the configuration of the memory that is connected to the external memory space, and the settings of the CSC0 and CSC1 registers.

The first memory access of the V850E2/ME3 is an instruction fetch cycle to access 0000000H. Therefore, external ROM must be allocated to the $\overline{CS0}$ space.

(1) Example of connecting external ROM (1 MB) and SRAM (128 KB)

- $\overline{CS0}$ space: External ROM (1 MB)
- $\overline{CS1}$ space: SRAM (128 KB)
- Set value of CSC0 register: 0011H

Figure 2-83. Example of Memory Map with External ROM (1 MB) and SRAM (128 KB) Connected



Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

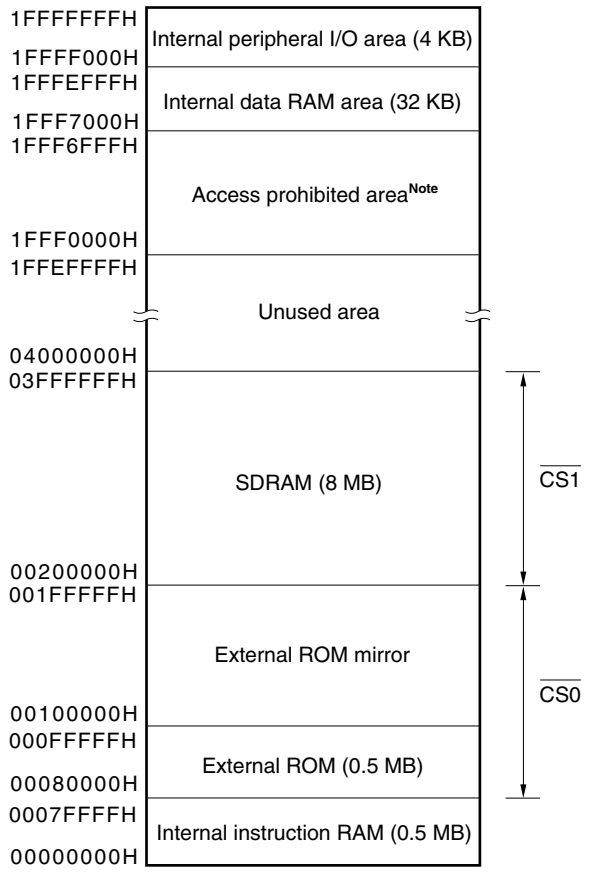
2. The relationship between the offset addresses of the external ROM and the addresses of the V850E/ME2 when 32 Mb external ROM is used is shown below.

Addresses of V850E2/ME3	Addresses of External ROM
00080000H to 000FFFFFH	080000H to 0FFFFFH
00100000H to 001FFFFFH	000000H to 0FFFFFH

(2) Example of connecting external ROM (1 MB) and SDRAM (8 MB)

- $\overline{CS0}$ space: External ROM (1 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)
- Set value of CSC0 register: 0011H

Figure 2-84. Example of Memory Map with External ROM (1 MB) and SDRAM (8 MB) Connected

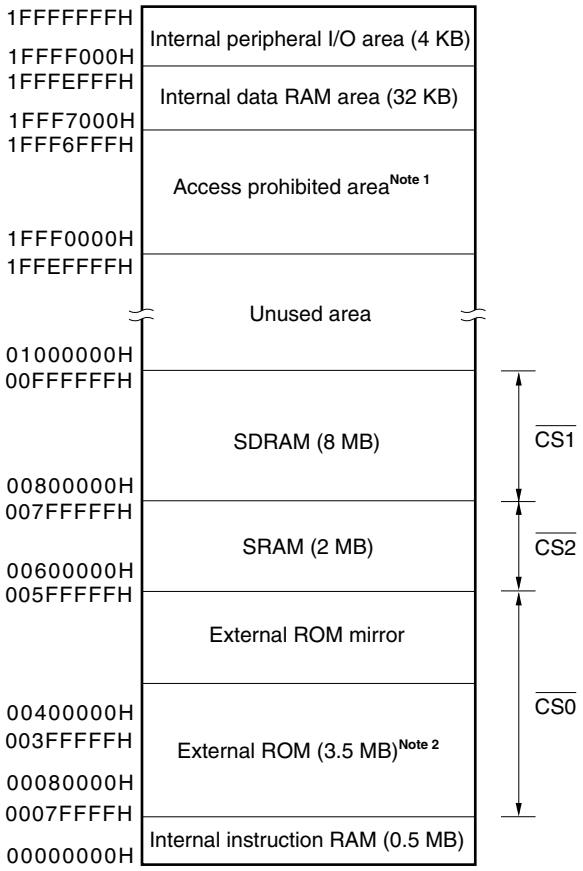


Note The operation is not guaranteed if the access prohibited area is accessed.

(3) Example of connecting external ROM (4 MB), SRAM (2 MB), and SDRAM (8 MB)

- $\overline{CS0}$ space: External ROM (4 MB)
- $\overline{CS2}$ space: SRAM (2 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)
- Set value of CSC0 register: 0817H

Figure 2-85. Example of Memory Map with External ROM (4 MB), SRAM (2 MB), and SDRAM (8 MB) Connected



Notes 1. The operation is not guaranteed if the access prohibited area is accessed.

2. The relationship between the offset addresses of the external ROM and the addresses of the V850E/ME2 when 32 Mb external ROM is used is shown below.

Addresses of V850E2/ME3	Addresses of External ROM
00080000H to 003FFFFFH	080000H to 3FFFFFH
00400000H to 005FFFFFH	000000H to 1FFFFFH

(4) Example of connecting external ROM (4 MB), SDRAM (8 MB), SRAM for data space (2 MB), SDRAM for data space (128 MB), and external I/O (256 KB)

Cautions 1. In this connection example, it is assumed that the external I/O is subject to DMA flyby transfer, and that $\overline{CS2}$ and $\overline{CS5}$ are not used.

2. The entire $\overline{CS}$ space of the V850E2/ME3 can be used as a programmable space.

(a) Program locatable space

- $\overline{CS0}$ space: External ROM (4 MB)
- $\overline{CS1}$ space: SDRAM (8 MB)

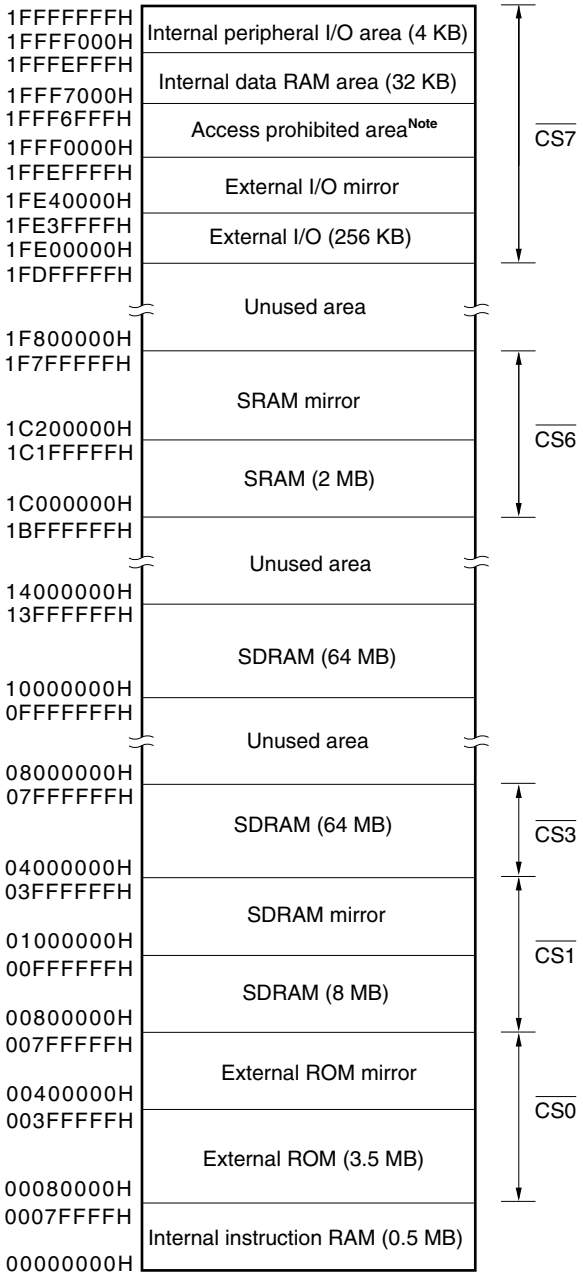
(b) Data space

- $\overline{CS3}$ space: SDRAM (64 MB)
- $\overline{CS4}$ space: SDRAM (64 MB)
- $\overline{CS6}$ space: SRAM (2 MB)
- $\overline{CS7}$ space: External I/O (256 KB)

(c) Setting of CSC0 and CSC1 registers

- Set value of CSC0 register: 201FH
- Set value of CSC1 register: 8011H

Figure 2-86. Example of Memory Map with External ROM (4 MB), SDRAM (8 MB), SRAM for Data Space (2 MB), SDRAM for Data Space (128 MB), and External I/O (256 KB) Connected



Note The operation is not guaranteed if the access prohibited area is accessed.

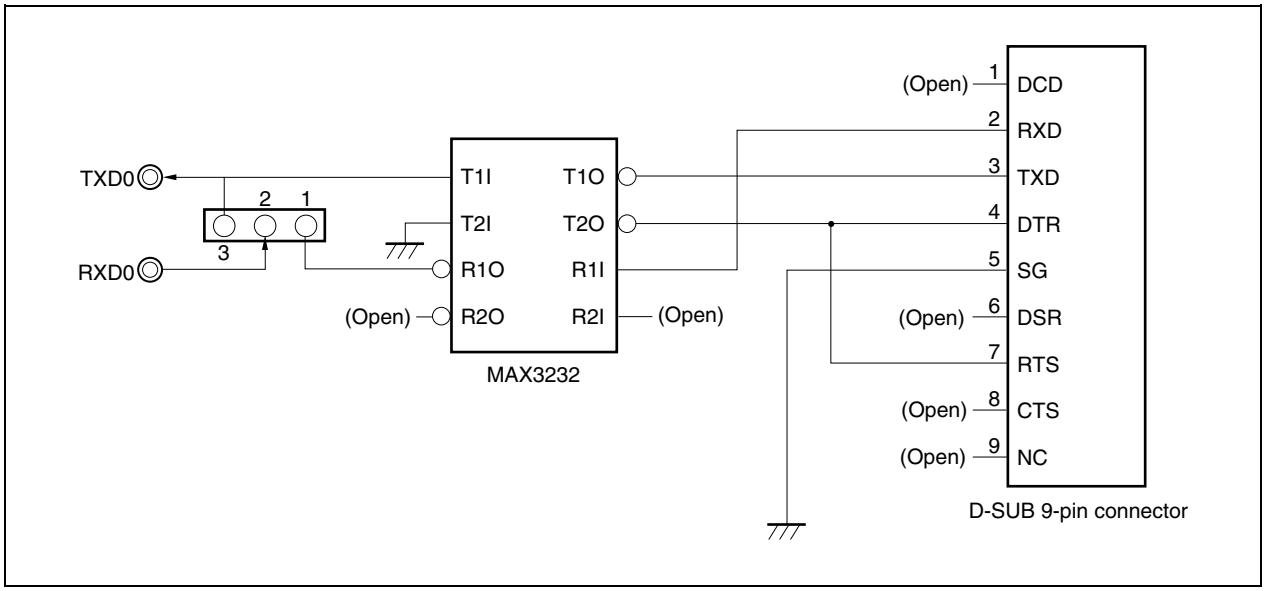
CHAPTER 3 EXAMPLES OF CONNECTION CIRCUITS FOR INTERNAL PERIPHERAL FUNCTIONS

This chapter shows examples of circuits connecting the internal peripheral functions of the V850E/ME2 and V850E2/ME3. The connection circuit examples shown in this chapter are excerpts from the circuit of TB-V850E/ME2 <R> (for the V850E/ME2) shown in **CHAPTER 4**. The circuit configuration is also basically the same when the V850E2/ME3 is used.

3.1 Connecting Serial Interface (UARTB)

An example of using UARTB0 as a start-stop synchronous RS-232C interface is shown below. In this circuit example, the TXD0 and RXD0 signals can be selected for debugging by using a jumper switch.

Figure 3-1. Example of Connection Circuit of UARTB

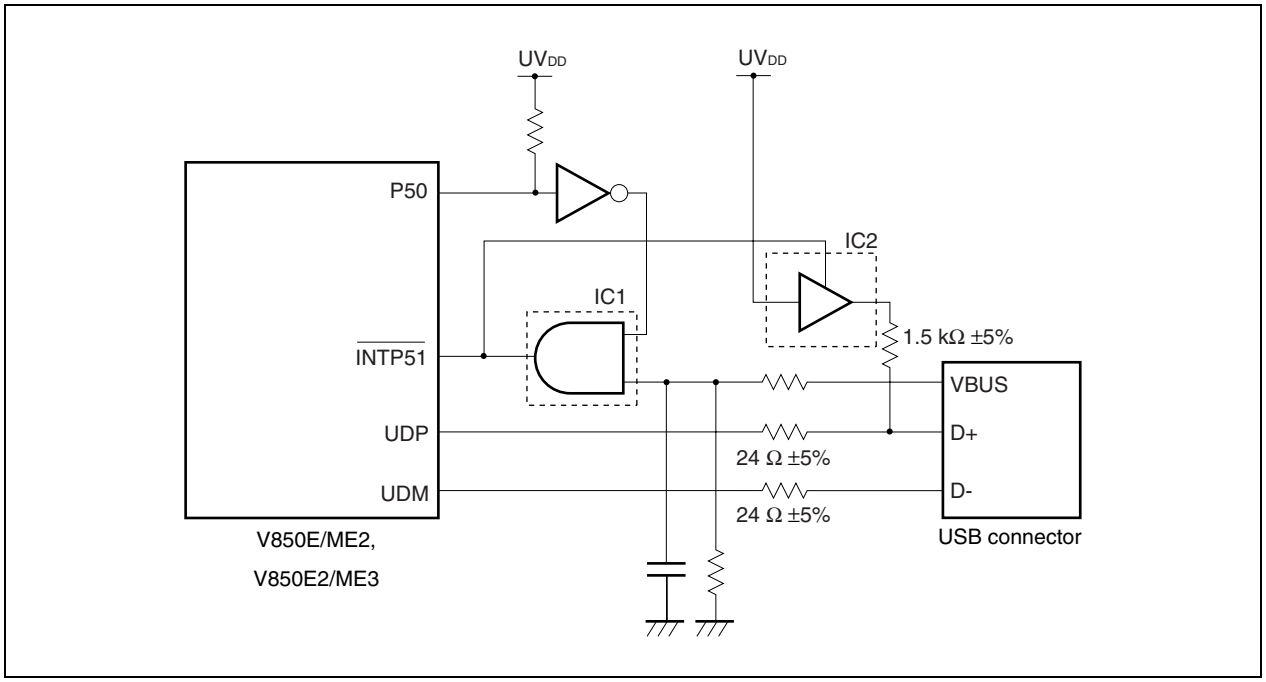


3.2 Connecting USB Function Controller (USBF)

Here is an example of connecting a USB interface to USBF.

Figure 3-2. Example of Circuit Connection of USBF

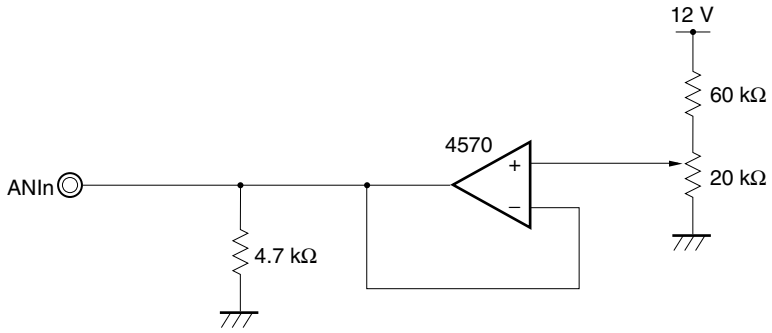
<R>



3.3 Connecting A/D Converter

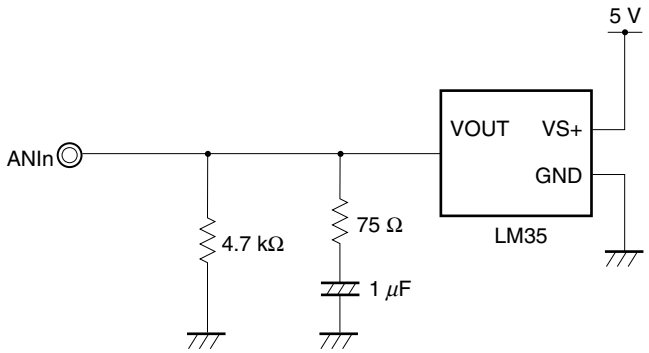
Examples of connecting a variable resistor and a temperature sensor are shown below.

Figure 3-3. Example of Connecting 0 to 3 V via Variable Resistor



Remark n = 0 to 7

Figure 3-4. Example of Connecting Temperature Sensor

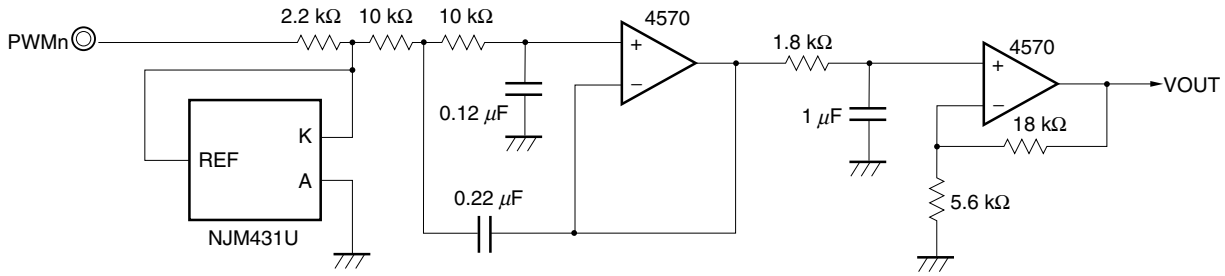


Remark n = 0 to 7

3.4 Connecting PWM Unit

An example of configuring a 0 to 5 V analog output by connecting an external circuit to the PWM output is shown below.

Figure 3-5. Example of Circuit Connection of PWM Unit



Cautions 1. The data bus cannot be used in the 32-bit mode when PWM is used.

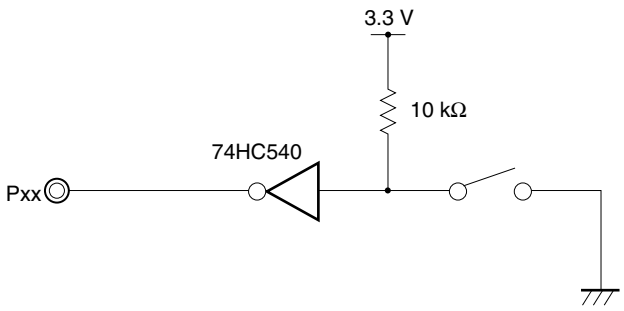
2. In this circuit example, the PWM cycle is 10 ms or less.

Remark n = 0 or 1

3.5 Connecting Port Function

Examples of connecting ports are shown below.

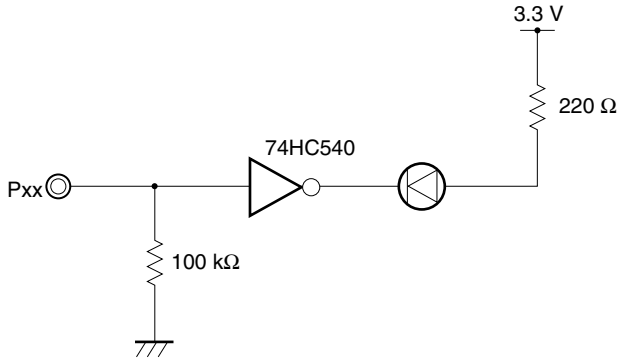
Figure 3-6. Example of Circuit Connection of Input Port



Caution In this circuit example, chattering of the switch is not eliminated. Eliminate chattering by software as necessary.

Remark Pxx = P10 to P13, P20 to P25, P50 to P55, P65 to P67, P72 to P77, PAH0 to PAH9, PAL0, PAL1, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0 to PCS7, PCT0 to PCT5, and PCT7

Figure 3-7. Example of Circuit Connection of Output Port



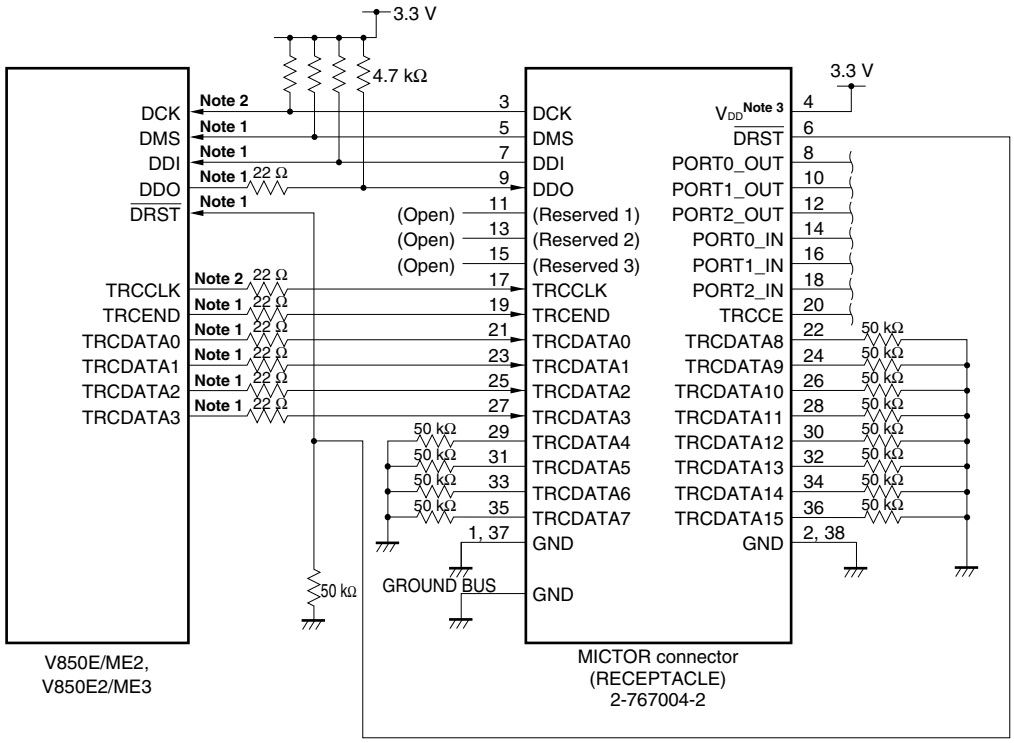
Remark Pxx = P10 to P13, P21 to P25, P50 to P55, P65 to P67, P72 to P77, PAH0 to PAH9, PAL0, PAL1, PDH0 to PDH15, PCD0 to PCD3, PCM0 to PCM5, PCS0 to PCS7, PCT0 to PCT5, and PCT7

3.6 Connecting On-Chip Debug Emulator

An example of connecting an on-chip debug emulator used for development is shown below.

Figure 3-8. Example of Circuit Connection of On-Chip Debug Emulator

<R>



Notes 1. Keep the pattern length to within 1.97 inches (50 mm).

2. Shield the DCK and TRCCLK signals by GND.

3. For detecting power to the target board

Caution The recommended circuit example shown above assumes that a 3.3 V interface is used.

CHAPTER 4 APPLICATION EXAMPLES

This chapter explains the functions and circuit of the CPU board on which the V850E/ME2 is mounted (TB-V850E/ME2), and introduces program examples.

<R> An example of using the V850E/ME2 is shown in this chapter but the basics are the same as when the V850E2/ME3 is used.

4.1 Functions of TB-V850E/ME2

4.1.1 Overview

The features of the TB-V850E/ME2 are as follows.

(1) V850E/ME2 mounted

The initialization pins (MODE0, SSEL0, SSEL1, JIT0, JIT1, and PLLSEL) are set by using a DIP switch on the board. MODE1 is fixed to low level.

(2) External ROM, SRAM, and SDRAM connected to memory interface

- $\overline{CS0}$ space: Connects external ROM (128 KB) with 16-bit width.
- $\overline{CS1}$ space: Connects SDRAM (32 MB) with 32-bit width.
- $\overline{CS2}$ space: Connects SRAM (512 KB) with 32-bit width.
- $\overline{CS3}$ space: Connects SRAM (256 KB) with 16-bit width.
- $\overline{CS4}$ space: Connects SDRAM (16 MB) with 16-bit width.
- $\overline{CS5}$ space: Connects SRAM (128 KB) with 8-bit width.
- $\overline{CS6}$ space: Connects SDRAM (16 MB) with 8-bit width.
- $\overline{CS7}$ space: Not used

Caution Because a 5 V external ROM is connected to the $\overline{CS0}$ space, the data bus is connected via 74LCX16244.

(3) RS-232C (2 channels)

Connected to UARTB0 and UARTB1 of the V850E/ME2.

(4) USB

Connected to the USB function controller (USBF) of the V850E/ME2.

(5) General-purpose input switch × 8, general-purpose output LED × 8

Connected to the ports of the V850E/ME2.

(6) Analog input

ANI0: Connects a variable resistor.

ANI1: Connects a temperature sensor.

(7) 1 to 5 V analog output

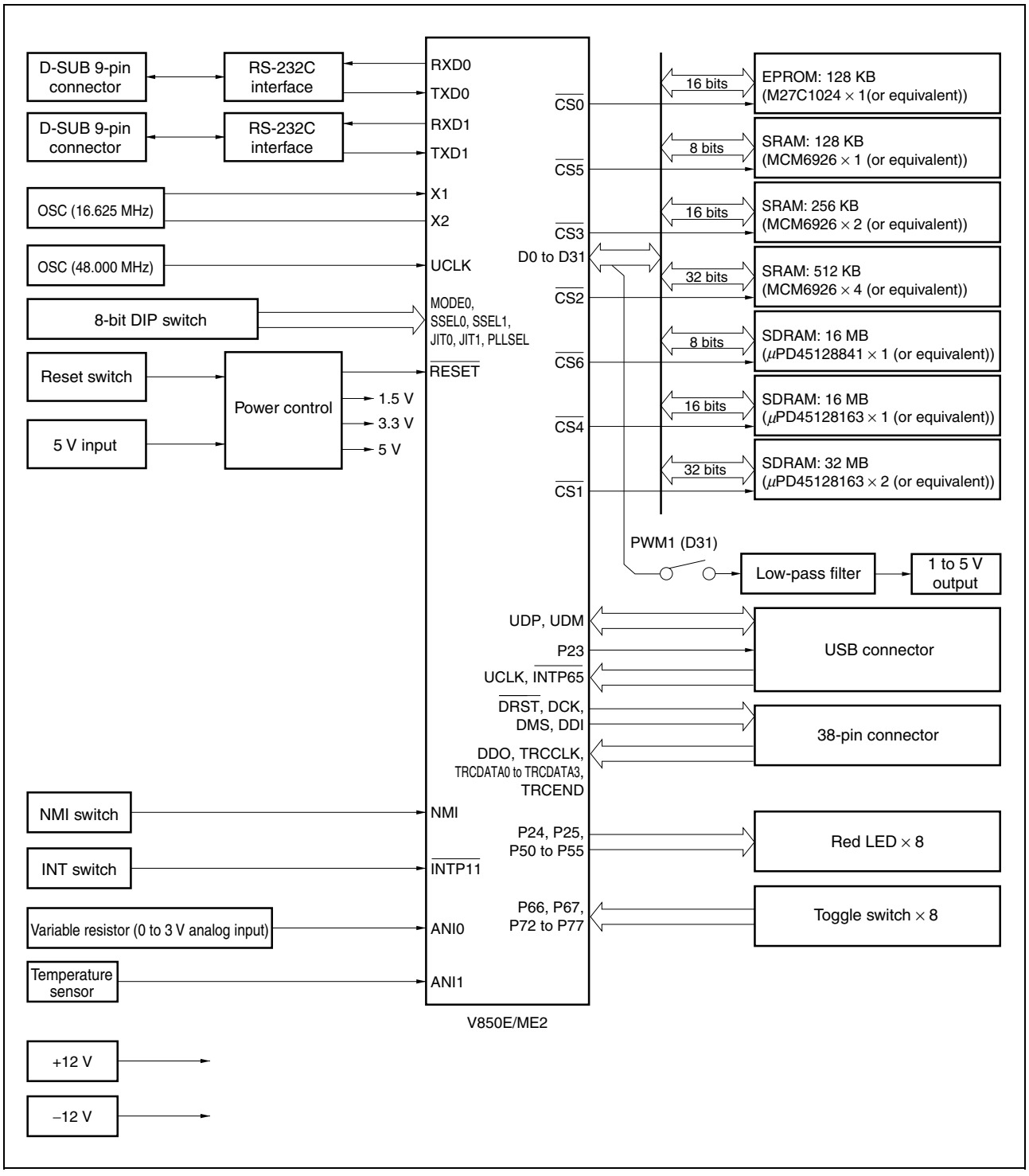
Connected to PWM1 of the V850E/ME2 via an external low-pass filter.

Caution PWM1 is shared with D31. Therefore, a 32-bit external data bus cannot be used. Be sure to use a 16-bit external data bus.

(8) Debug interface

An on-chip debug emulator connector is provided.

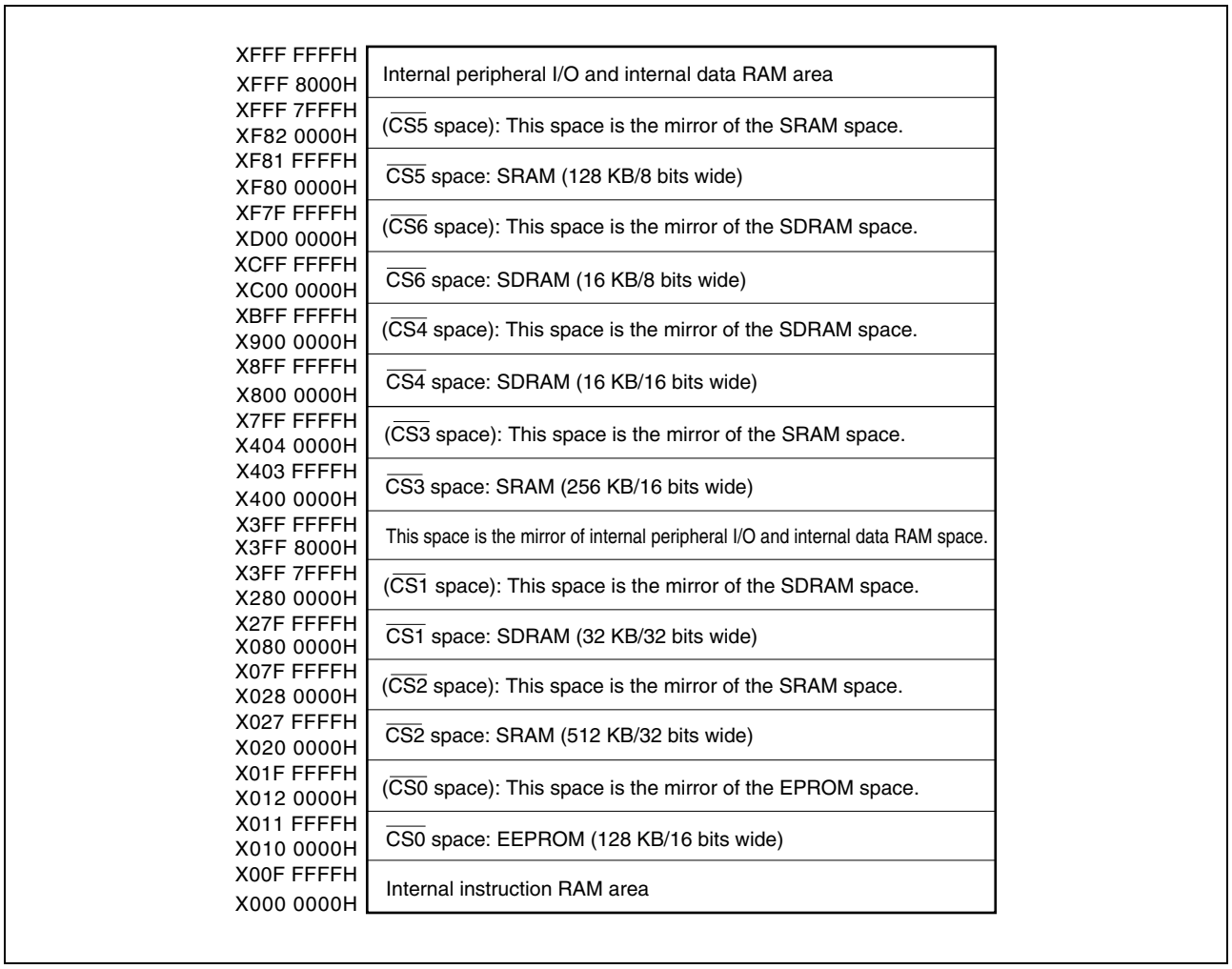
Figure 4-1. Board Configuration



4.1.2 Memory map

The memory map when the CSC0 register is set to 0E01H and the CSC1 register is set to 0F00H is shown below.

Figure 4-2. Memory Map



4.1.3 Connecting external bus interface

The external bus interface connects external ROM, SRAM, and SDRAM. When using the function of PWM1, a 16-bit external data bus is used.

$\overline{CSn}$ Signal	Connected Memory	Outline
$\overline{CS0}$	EPROM (M27C1024-55 × 1)	1 Mb EPROM (55 ns)
$\overline{CS1}$	SDRAM (μPD45128163-A75A × 2)	128 Mb SDRAM (133 MHz/16 bits)
$\overline{CS2}$	SRAM (MCM6926AWJ8 × 4)	1 Mb high-speed SRAM (8 ns)
$\overline{CS3}$	SRAM (MCM6926AWJ8 × 2)	1 Mb high-speed SRAM (8 ns)
$\overline{CS4}$	SDRAM (μPD45128163-A75A × 1)	128 Mb SDRAM (133 MHz/16 bits)
$\overline{CS5}$	SRAM (MCM6926AWJ8 × 1)	1 Mb high-speed SRAM (8 ns)
$\overline{CS6}$	SDRAM (μPD45128841-A75A × 1)	128 Mb SDRAM (133 MHz/16 bits)

4.1.4 Connecting peripheral functions

(1) Connecting port functions

Eight output ports and eight input ports are used.

Pin Name	I/O	Function
P23	Output	Pull-up control of D+ pin
P24	Output	General-purpose LED1 (ON = 1, OFF = 0)
P25	Output	General-purpose LED2 (ON = 1, OFF = 0)
P50	Output	General-purpose LED3 (ON = 1, OFF = 0)
P51	Output	General-purpose LED4 (ON = 1, OFF = 0)
P52	Output	General-purpose LED5 (ON = 1, OFF = 0)
P53	Output	General-purpose LED6 (ON = 1, OFF = 0)
P54	Output	General-purpose LED7 (ON = 1, OFF = 0)
P55	Output	General-purpose LED8 (ON = 1, OFF = 0)
P66	Input	General-purpose toggle switch 1 (ON = 0, OFF = 1)
P67	Input	General-purpose toggle switch 2 (ON = 0, OFF = 1)
P72	Input	General-purpose toggle switch 3 (ON = 0, OFF = 1)
P73	Input	General-purpose toggle switch 4 (ON = 0, OFF = 1)
P74	Input	General-purpose toggle switch 5 (ON = 0, OFF = 1)
P75	Input	General-purpose toggle switch 6 (ON = 0, OFF = 1)
P76	Input	General-purpose toggle switch 7 (ON = 0, OFF = 1)
P77	Input	General-purpose toggle switch 8 (ON = 0, OFF = 1)

(2) Connecting UART

UARTBn is used as a start-stop synchronous interface (n = 0 or 1). This interface is a two-wire interface that uses the TXDn and RXDn signals. The RTSx and DTRx signals on the RS-232C connectors (CN2 and CN3) are fixed to the active level, and the CTSx, DCDx, and DSRx signals are not connected. UARTBn can return the TXDn signal for the RXDn signal by setting a jumper switch.

UART Name	Interface	Remark
UARTB0	RS-232C	D-SUB 9-pin connector (CN2)
UARTB1	RS-232C	D-SUB 9-pin connector (CN3)

(3) Connecting interrupt signals

The NMI and INT switches are respectively connected to the NMI and $\overline{\text{INTP11}}$ pins.

Interrupt Name	Valid Edge	Connected Switch
NMI	Falling	NMI switch
$\overline{\text{INTP11}}$	Falling	INT switch

(4) Connecting A/D converter

A variable resistor of 0 to 3 V is connected to ANI0 and a temperature sensor is connected to ANI1.

Pin Name	Function
ANI0	0 to 3 V input from variable resistor
ANI1	0 to 3 V input from temperature sensor (low temperature: 0 V, high temperature: 3 V)

(5) Connecting PWM

PWM1 is connected to 1 to 5 V analog output via an external low-pass filter. The external low-pass filter conforms to a cycle of 20 ms or less. The 1 to 5 V analog output function cannot use a 32-bit data bus. Be sure to use a 16-bit external data bus.

(6) Connecting USB function controller (USBF)

The USB function controller is connected by a USB function connector. The P23 and $\overline{\text{INTP65}}$ pins are used for the following control operations.

Pin Name	Function
P23	Pull-up control of D+ pin
$\overline{\text{INTP65}}$	Detection of connection/disconnection of USB cable

(7) Connecting debug interface

A 38-pin header connector for debugging is connected to the on-chip debug emulator.

4.1.5 Setting switches

(1) Reset switch (RESET)

This is a pushbutton switch that forcibly asserts the $\overline{\text{RESET}}$ pin of the V850E/ME2.

(2) NMI switch (NMI)

This pushbutton switch sends a non-maskable interrupt (NMI) to the V850E/ME2. The NMI pin goes low when this switch is pushed.

(3) INT switch (INT)

This pushbutton switch sends a maskable interrupt to the V850E/ME2. The $\overline{\text{INTP11}}$ pin goes low when this switch is pushed.

(4) Operation mode selector switch (DSW1)

This switch selects the operation mode of the V850E/ME2 and the default value of the system clock. Each pin of the V850E/ME2 is cleared to 0 when the corresponding switch pin of this switch is ON, and set to 1 when the corresponding switch pin is OFF.

Switch Name	Meaning
DSW1-1	Setting of MODE0 pin
DSW1-2	Not used
DSW1-3	Setting of JIT0 pin
DSW1-4	Setting of JIT1 pin
DSW1-5	Setting of PLLSEL pin
DSW1-6	Setting of SSEL0 pin
DSW1-7	Setting of SSEL1 pin
DSW1-8	Not used

(5) RS-232C interface return switches (JP1 and JP2)

These are jumper switches that return the TXDn signal of the RS-232C connected to UARTBn to the RXDn signal (n = 0 or 1). When these jumpers are short-circuited, the TXDn signal is returned to the RXBn signal.

(6) PWM selector switch (JP3)

This jumper switch enables the PWM function. Short-circuit this jumper when using the PWM1 pin as a 1 to 5 V analog output pin. When the jumper is opened, the analog output is fixed to +1 V.

4.1.6 Peripheral I/O register settings

The set values of the peripheral I/O registers based on the hardware configuration of the TB-V850E/ME2 are shown in the table below.

Caution The set values of the peripheral I/O registers shown below are when BUSCLK is 64 MHz (f_{CLK}/2) and when the memory map shown in 4.1.2 is used (f_{CLK}: Internal system clock). The unused port pins are set to the output mode.

(1/2)

Register Name	Symbol	Set Value
System wait control register	VSWC	33H
Chip area select control register 0	CSC0	0E01H
Chip area select control register 1	CSC1	0F00H
Bus cycle type configuration register 0	BCT0	88B8H
Bus cycle type configuration register 1	BCT1	0B8BH
Local bus sizing control register	LBS	0169H
Endian configuration register	BEC	0000H
Line buffer control register 0	LBC0	0000H
Line buffer control register 1	LBC1	0000H
Bus mode control register	BMC	01H
Data wait control register 0	DWC0	0003H
Data wait control register 1	DWC1	0000H
Address setup wait control register	ASC	0000H
Bus cycle period control register	BCP	00H
Bus cycle control register	BCC	0452H
SDRAM configuration register 1	SCR1	20A5H
SDRAM configuration register 3	SCR3	0000H
SDRAM configuration register 4	SCR4	2095H
SDRAM configuration register 6	SCR6	2086H
SDRAM refresh control register 1	RFS1	801EH
SDRAM refresh control register 3	RFS3	0000H
SDRAM refresh control register 4	RFS4	801EH
SDRAM refresh control register 6	RFS6	801EH
External interrupt rising edge specification register 1	INTR1	00H
External interrupt rising edge specification register 1	INTF1	00H
External interrupt rising edge specification register 2	INTR2	00H
External interrupt rising edge specification register 2	INTF2	00H
External interrupt rising edge specification register 6	INTR6	20H
External interrupt rising edge specification register 6	INTF6	20H
Clock control register	CKC	03H
Clock source select register	CKS	01H
SSCG control register	SSCGC	00H
Lock register	LOCKR	0xH
Port 1 mode control register	PMC1	0FH
Port 1 function control register	PFC1	0DH

(2/2)

Register Name	Symbol	Set Value
Port 2 mode register	PM2	C7H
Port 2 mode control register	PMC2	07H
Port 2 function control register	PFC2	06H
Port 5 mode register	PM5	C0H
Port 5 mode control register	PMC5	00H
Port 6 mode register	PM6	FFH
Port 6 mode control register	PMC6	20H
Port 7 mode register	PM7	FFH
Port 7 mode control register	PMC7	00H
Port AL mode control register	PMCAL	0003H
Port AL function control register	PFCALL	03H
Port AH mode control register	PMCAH	03FFH
Port DH mode register	PMDH[Note]	0000H
Port DH mode control register	PMCDH[Note]	8000H
Port DH function control register	PFCDH[Note]	8000H
Port CS mode register	PCS	FFH
Port CS mode control register	PMCCS	FFH
Port CS function control register	PFCCS	00H
Port CT mode register	PCT	BFH
Port CT mode control register	PMCCT	BFH
Port CT function control register	PFCCT	0FH
Port CM mode register	PMCM	C0H
Port CM mode control register	PMCCM	00H
Port CM function control register	PFCCM	00H
Port CD mode register	PCD	0CH
Port CD mode control register	PMCCD	0FH

Note The settings of the PMDH, PMCDH, and PFCDH registers are for the 16-bit mode (MODE1 and MODE0 = 01). The 16-bit mode is used to test the PWM1 function with the TB-V850E/ME2. In the application program example in this chapter, however, the settings of the PMDH, PMCDH, and PFCDH registers are invalid and the PWM function cannot be used because the V850E/ME2 is started in 16-bit mode and the data bus is extended to 32 bits.

4.1.7 Connectors

(1) Power connector (J3)

The power connector supplies +5 V and ±12 V to the TB-V850E/ME2.

○ Connectors used

- Board side: B6P-VH (J.S.T. Mfg. Co., Ltd.)
- Cable side: Housing VHR-6N (J.S.T. Mfg. Co., Ltd.)
  Terminal BVH-21T-P1-1 (J.S.T. Mfg. Co., Ltd.)

Pin No.	Function
1	+5 V
2	+5 V
3	GND
4	GND
5	+12 V
6	−12 V

(2) Connectors of RS-232C (CN2 and CN3)

○ Connectors used

- Board side: DELC-J9PAF-20L6 (Japan Aviation Electronics Industry, Ltd.)
- Cable side: DEM-9S (Japan Aviation Electronics Industry, Ltd.)

Pin No.	Signal Name	Meaning
1	DCDx	Data carrier detect
2	RXDx	Receive data
3	TXDx	Transmit data
4	DTRx	Terminal ready
5	SGx	Signal ground
6	DSRx	Data set ready
7	RTSx	Request to send
8	CTSx	Clear to send
9	NCx	No connection

(3) Connector of USB (J2)

○ Connectors used

- Board side: XM7B-0442 (OMRON Corporation)
- Cable side: XM7Z-200AB-FC2 (OMRON Corporation)
  Connect B type side. Use a cable with connector conforming to the USB 1.1 standard.

Pin No.	Signal Name	Meaning
1	GND	Ground
2	D+	Positive side of data I/O signal
3	D–	Negative side of data I/O signal
4	VBUS	Power supply monitor

(4) Connector for voltage output (PWM) (J1)

○ Connectors used

- Board side: B2P-VH (J.S.T. Mfg. Co., Ltd.)
- Cable side: Housing VHR-2N (J.S.T. Mfg. Co., Ltd.)
  Terminal BVH-21T-P1.1 (J.S.T. Mfg. Co., Ltd.)

Pin No.	Signal Name	Meaning
1	PWM	Voltage output by PWM
2	GND	Ground

(5) Connector of on-chip debug emulator (CN1)

○ Connector used

- Board side: 2-767004-2 (MICTOR, Distributor: Tyco Electronics AMP)

Pin No.	Connected Signal Name	I/O	Pin No.	Connected Signal Name	I/O
1	GND	–	2	GND	–
3	DCK	Input	4	V_{DD}	–
5	DMS	Input	6	$\overline{\text{DRST}}$	Input
7	DDI	Input	8	PORT0_OUT	Open
9	DDO	Output	10	PORT1_OUT	Open
11	(Reserved 1)	Open	12	PORT2_OUT	Open
13	(Reserved 2)	Open	14	PORT0_IN	Open
15	(Reserved 3)	Open	16	PORT1_IN	Open
17	TRCCLK	Output	18	PORT2_IN	Open
19	TRCEND	Output	20	TRCCE	Open
21	TRCDATA0	Output	22	TRCDATA8	Output
23	TRCDATA1	Output	24	TRCDATA9	Output
25	TRCDATA2	Output	26	TRCDATA10	Output
27	TRCDATA3	Output	28	TRCDATA11	Output
29	TRCDATA4	Output	30	TRCDATA12	Output
31	TRCDATA5	Output	32	TRCDATA13	Output
33	TRCDATA6	Output	34	TRCDATA14	Output
35	TRCDATA7	Output	36	TRCDATA15	Output
37	GND	–	38	GND	–

4.1.8 Specifications of TB-V850E/ME2

(1) Specifications

The following table shows the specifications of the V850E/ME2.

Item		Specifications
V850E/ME2		Internal system clock = 128 MHz (16 MHz × 8) BUSCLK = 64 MHz (128 MHz/2) Operation mode selectable by switch
EPROM		Capacity: 128 KB ROM used: M27C1024 × 1 (or equivalent)
SRAM	8-bit bus	Capacity: 128 KB SRAM used: MCM6926 × 1 (or equivalent)
	16-bit bus	Capacity: 256 KB SRAM used: MCM6926 × 2 (or equivalent)
	32-bit bus	Capacity: 512 KB SRAM used: MCM6926 × 4 (or equivalent)
SDRAM	8-bit bus	Capacity: 16 MB SDRAM used: μPD45128841 × 1 (or equivalent)
	16-bit bus	Capacity: 16 MB SDRAM used: μPD45128163 × 1 (or equivalent)
	32-bit bus	Capacity: 32 MB SDRAM used: μPD45128163 × 2 (or equivalent)
Switch		Toggle switch: × 8 Pushbutton switch: × 3 Reset switch NMI switch INT switch DIP switch: × 1 (for setting mode of V850E/ME2)
LED		Light-emitting diode: × 13 General-purpose (red) × 8 Power (green) × 5
Analog input		Voltage level is input to ANI0 by volume. Voltage level: 0 to 3 V
Temperature sensor		Voltage level is input to ANI1 by temperature sensor. Temperature range: −55 to +50°C
RS-232C interface		Internal UARTB0 and UARTB1 of V850E/ME2 are used (two channels).
PWM		Converts PWM output into voltage (linear). Voltage level: 1 to 5 V
USB		Internal USBF of V850E/ME2 is used (connector is B type).
On-chip debug emulator		Internal DCU of V850E/ME2 is used (connector is 38-pin).
Power supply		• Input: ±12 V +5 V • Output: +3.3 V (generated by LT1764 from +5 V) +1.5 V (generated by MIC5209 from +3.3 V)

(2) Circuit diagrams

The circuit diagrams of the TB-V850E/ME2 are shown on the following pages.

<1> CPU (1/2) and reset circuit (refer to **Figure 4-3**)
This circuit diagram mainly shows the V850E/ME2 and reset circuit.

<2> CPU (2/2) and peripheral circuits (refer to **Figure 4-4**)
This circuit diagram mainly shows the V850E/ME2, oscillator, and interrupt circuits (NMI and INT).

<3> Memory peripheral circuits (refer to **Figures 4-5** and **4-6**)
These circuit diagrams mainly show the EPROM, buffer for ROM, and SRAM.

<4> Memory peripheral circuits (refer to **Figures 4-7** and **4-8**)
These circuit diagrams mainly show the SDRAM.

<5> Analog and serial interface circuits (refer to **Figure 4-9**)
This circuit diagram mainly shows the 4570, LM35, and MAX3232.

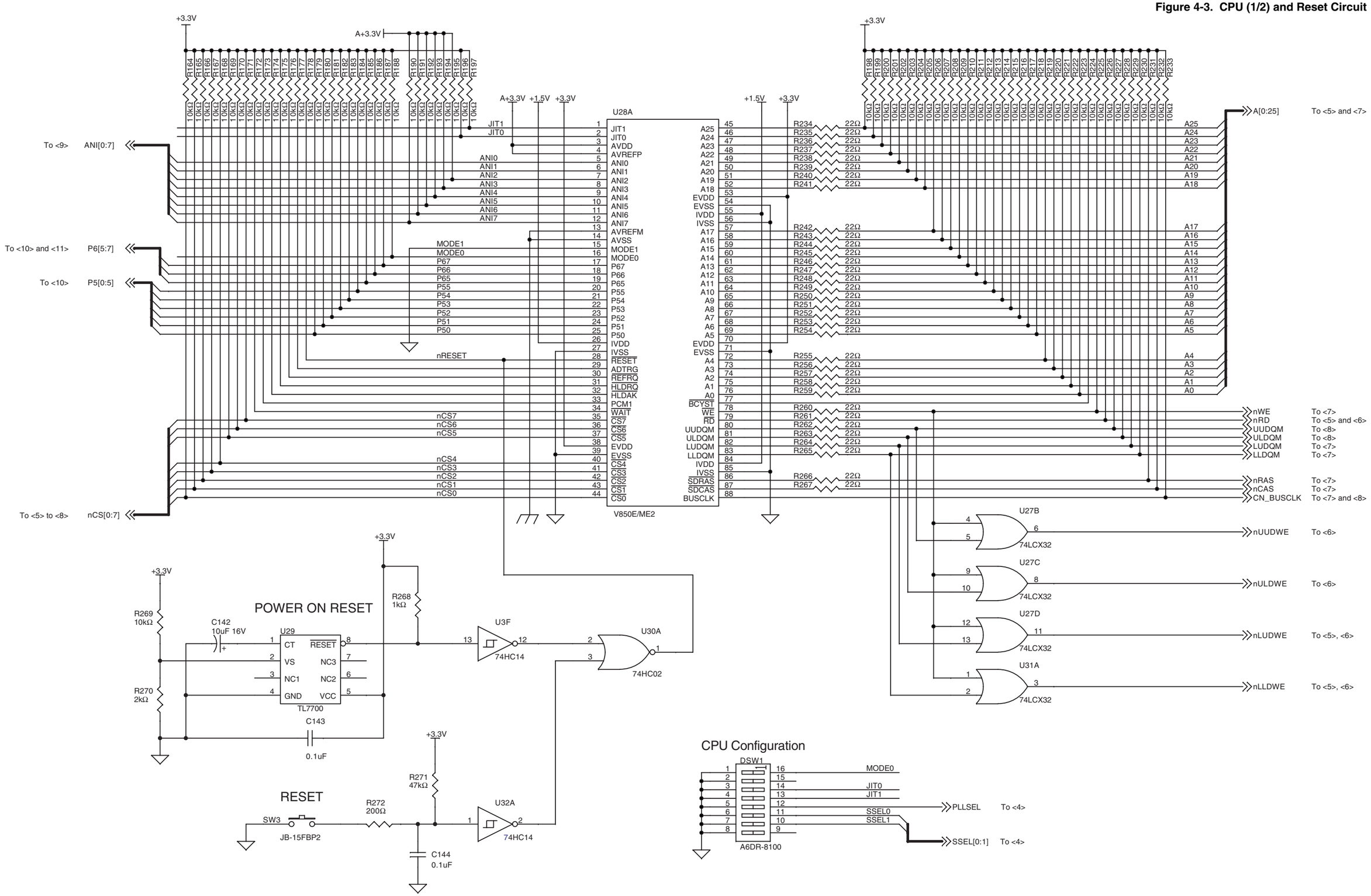
<6> PWM and I/O circuits (refer to **Figure 4-10**)
This circuit diagram mainly shows the NJM431, 4570, 74HC540, LEDs, and toggle switches.

<7> USB peripheral circuit (refer to **Figure 4-11**)
This circuit diagram mainly shows the USB (B-type connector).

<8> On-chip debug emulator (refer to **Figure 4-12**)
This circuit diagram mainly shows connectors for an on-chip debug emulator.

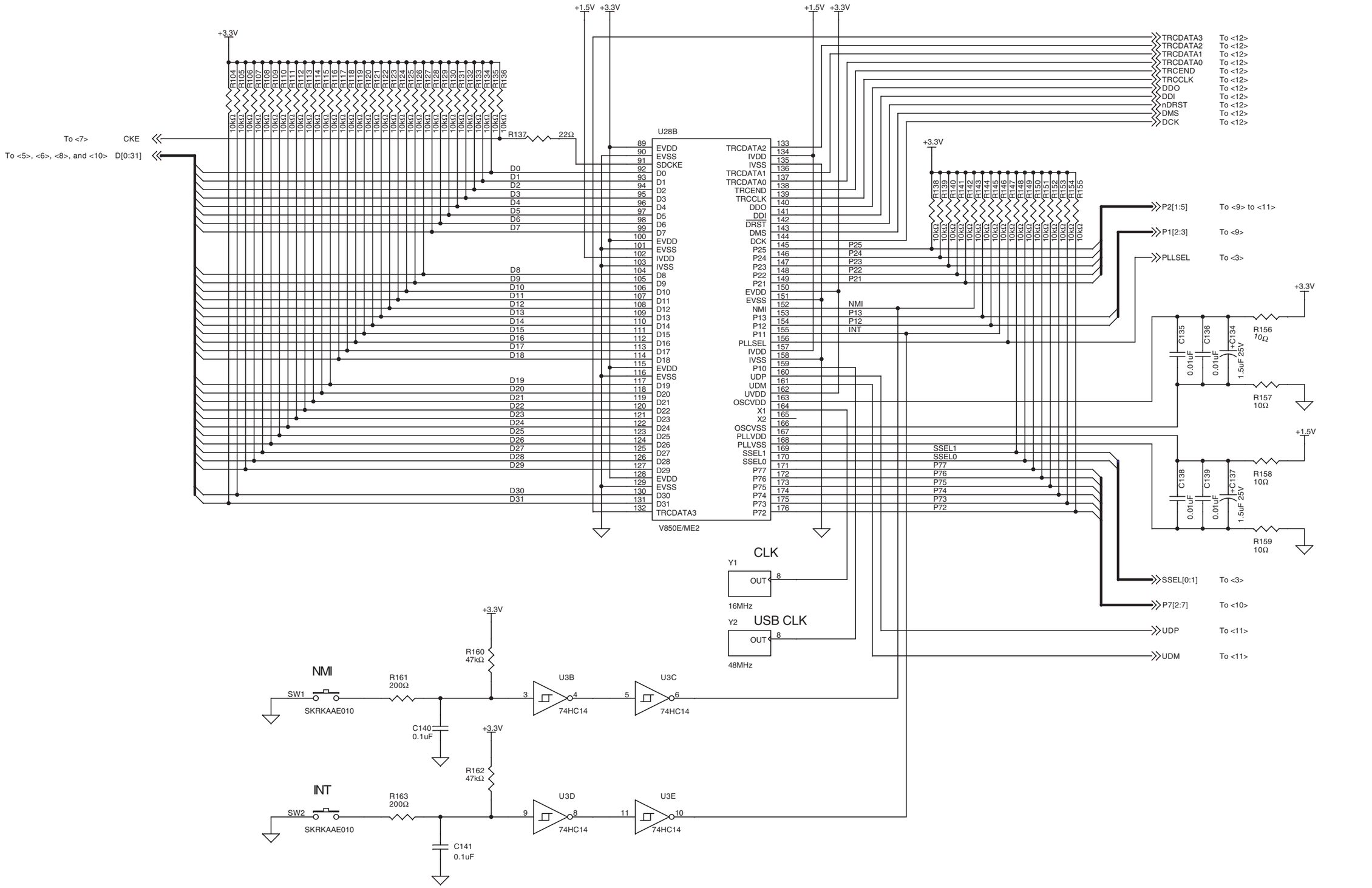
<9> Power peripheral circuits (refer to **Figure 4-13**)
This circuit diagram mainly shows the power feed connector, LT1764, MIC5209, and LEDs.

Figure 4-3. CPU (1/2) and Reset Circuit



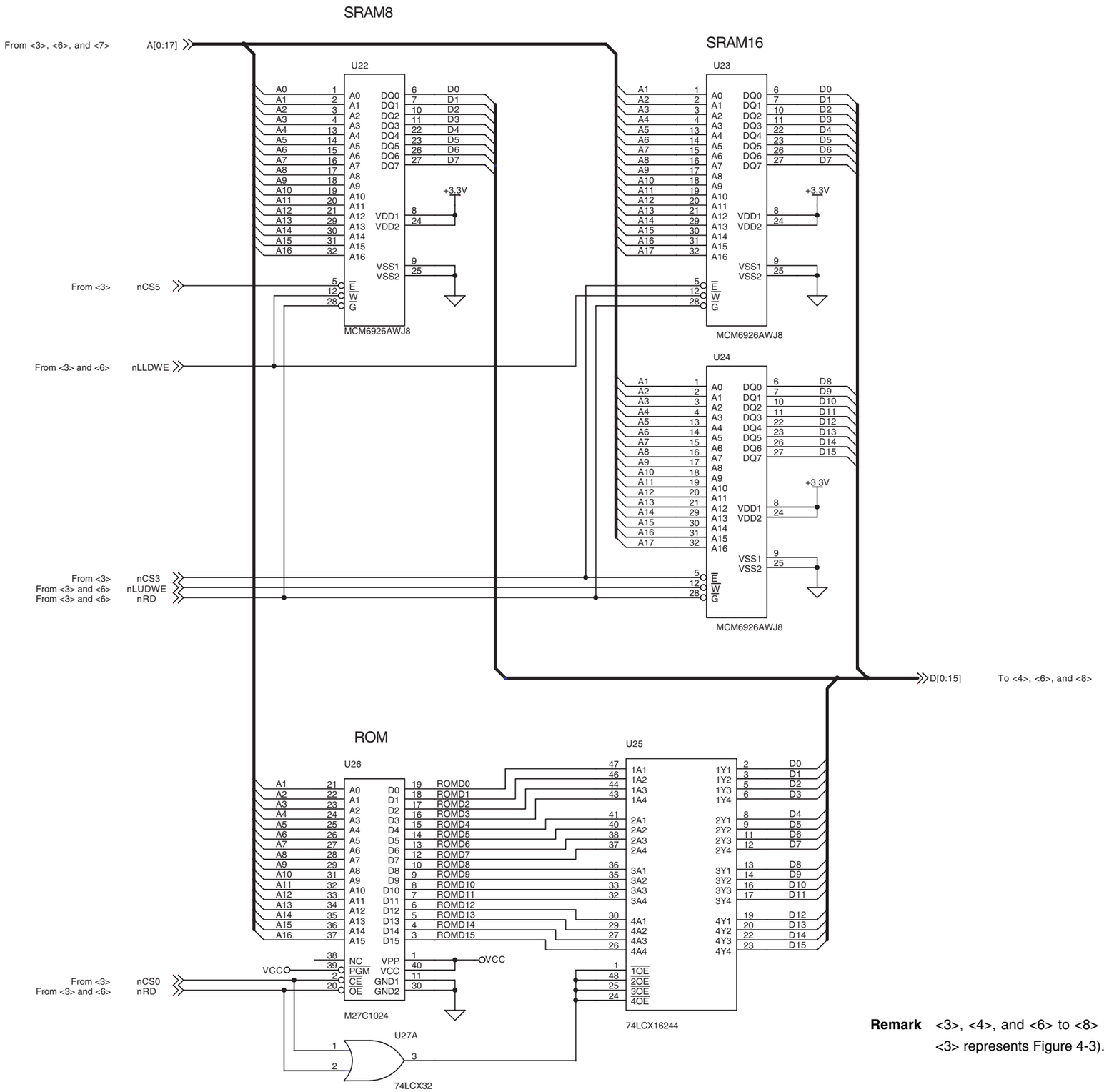
Remark <4> to <11> indicate the numbers of figures (e.g., <4> represents Figure 4-4).

Figure 4-4. CPU (2/2) and Peripheral Circuits



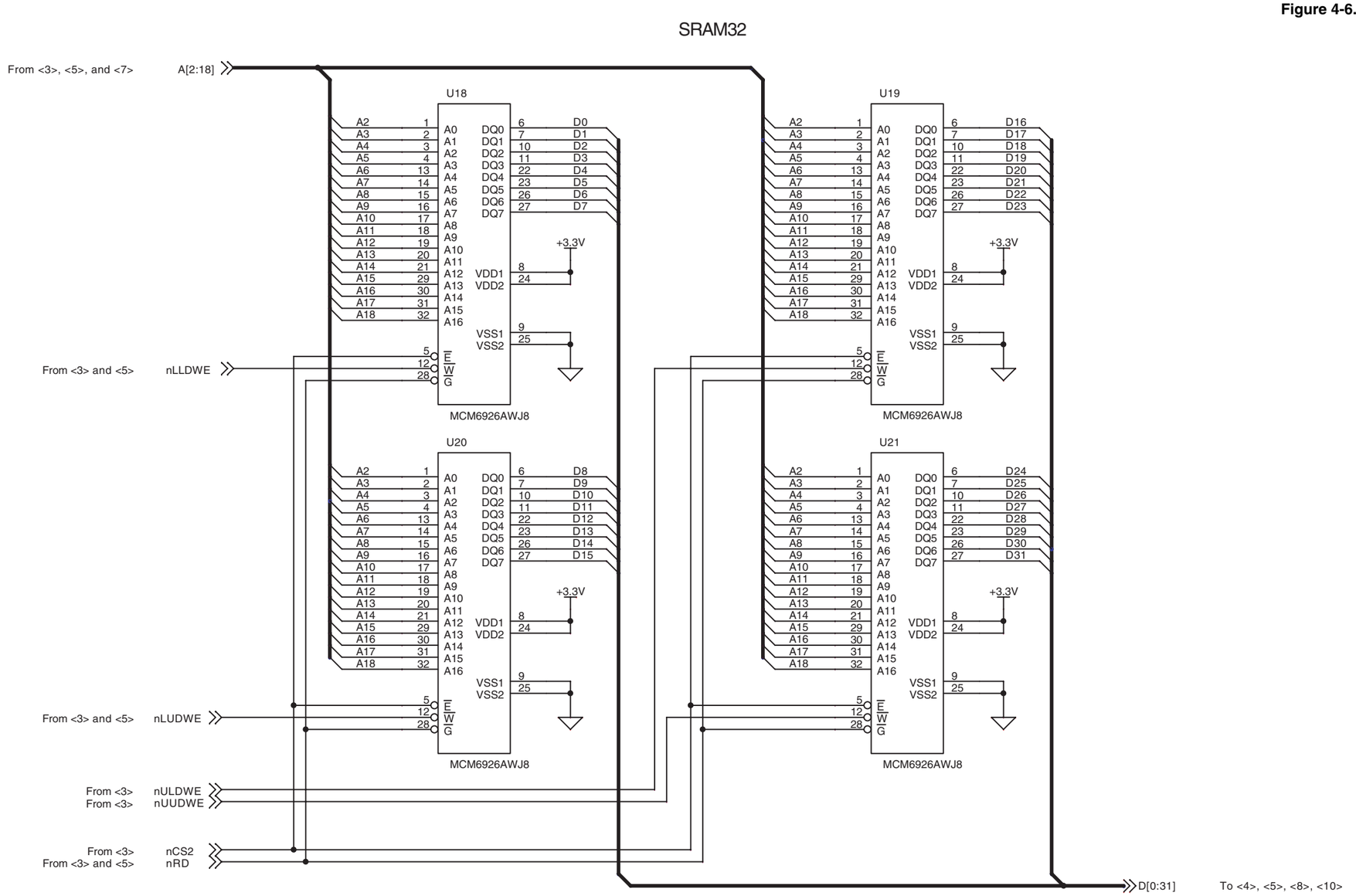
Remark <3> and <5> to <12> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-5. Memory Peripheral Circuit (1)



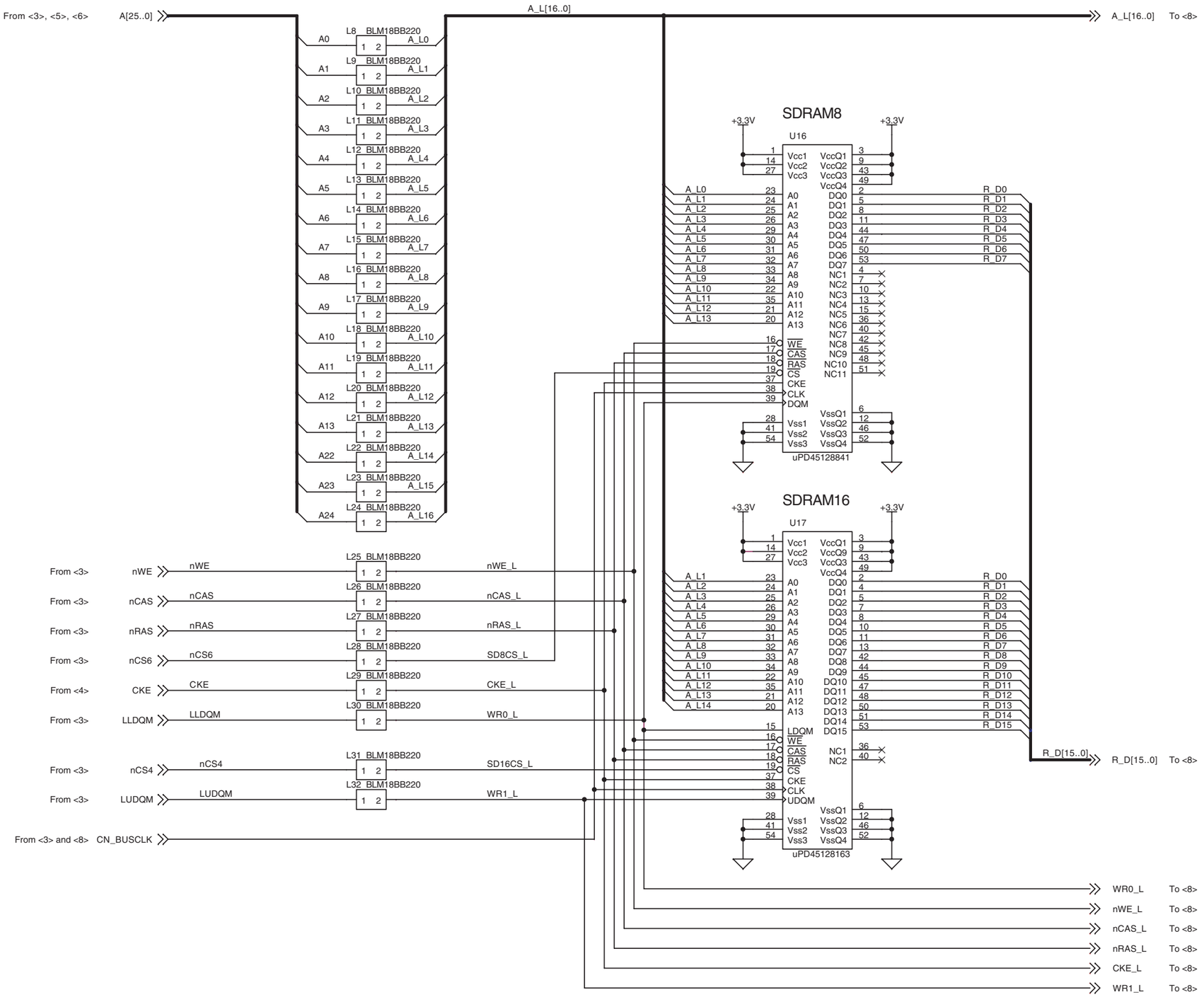
Remark <3>, <4>, and <6> to <8> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-6. Memory Peripheral Circuit (2)



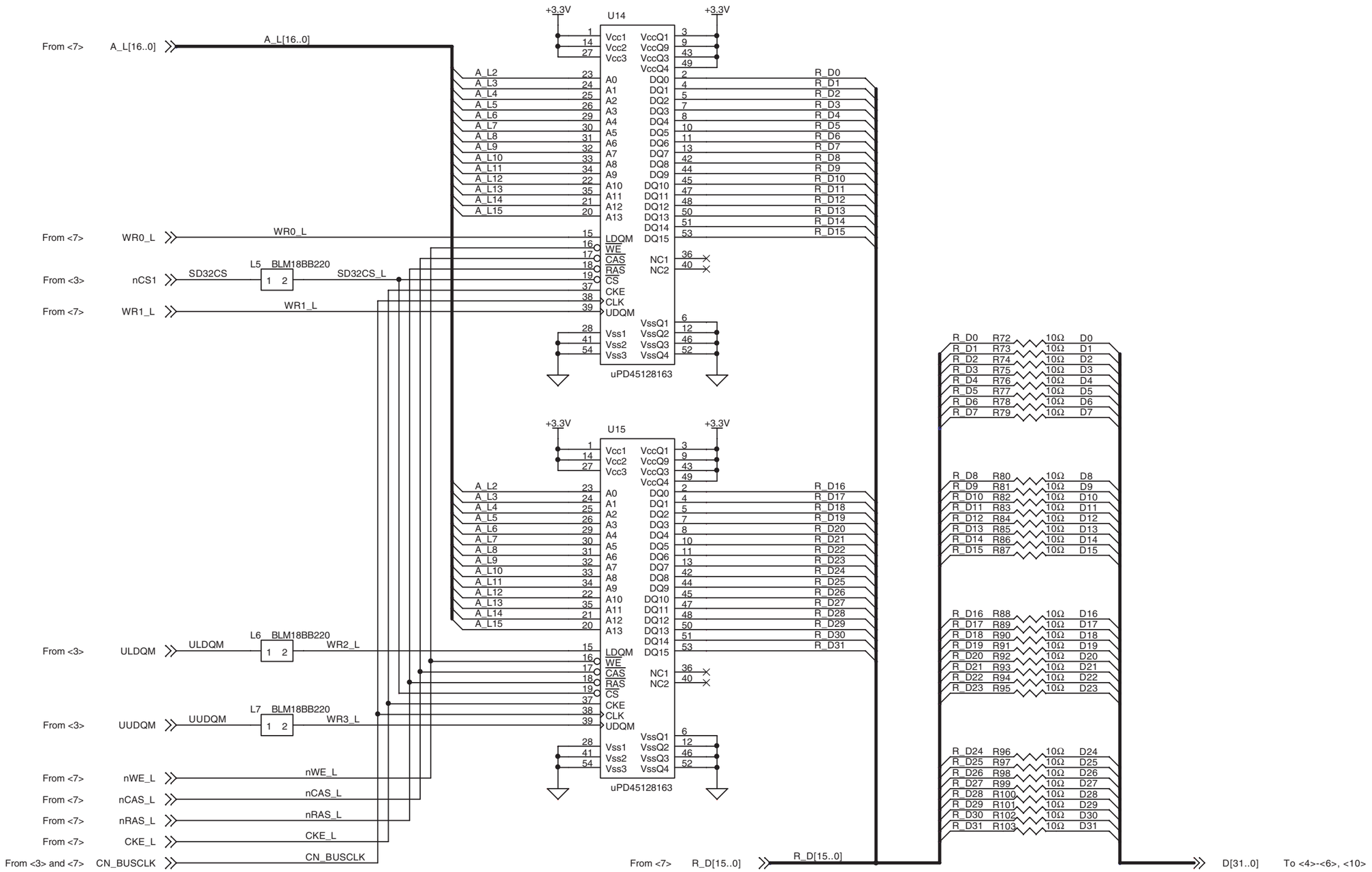
Remark <3> to <5>, <7>, <8>, and <10> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-7. Memory Peripheral Circuit (3)



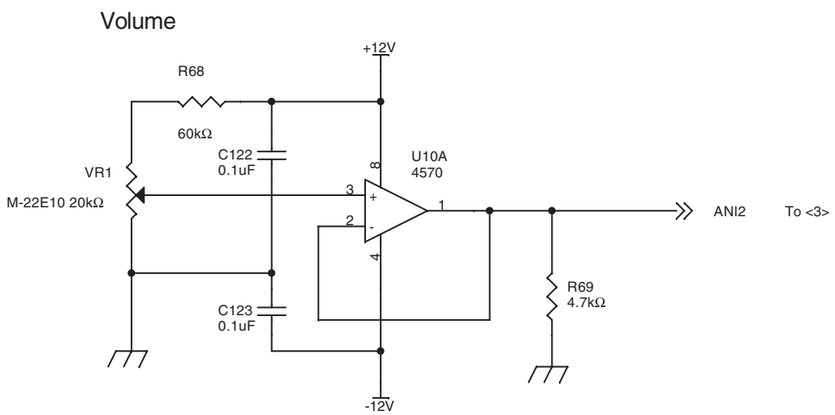
Remark <3> to <6> and <8> indicate the numbers of figure (e.g. <3> represents Figure 4-3).

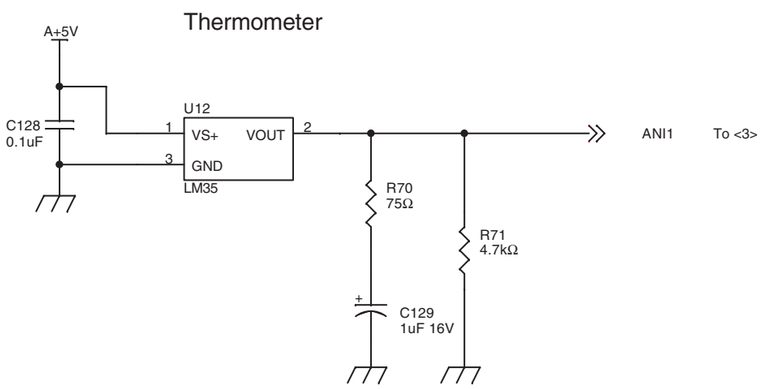
Figure 4-8. Memory Peripheral Circuit (4)

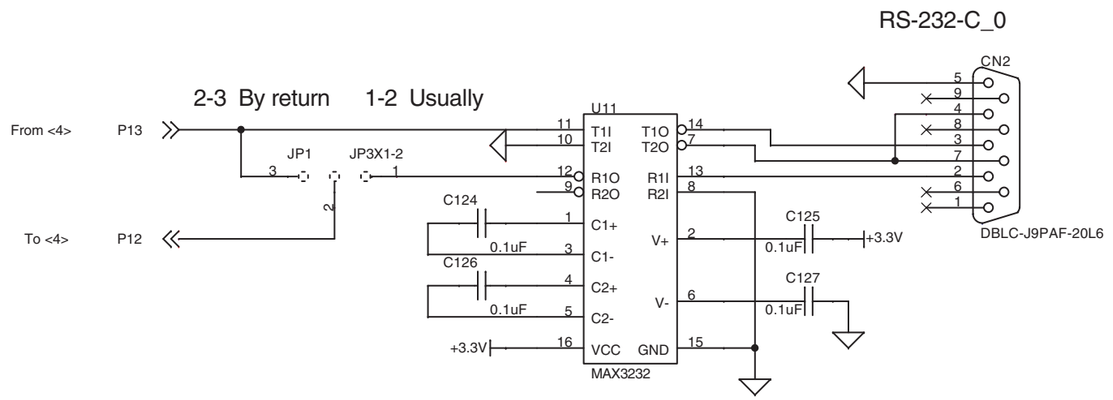


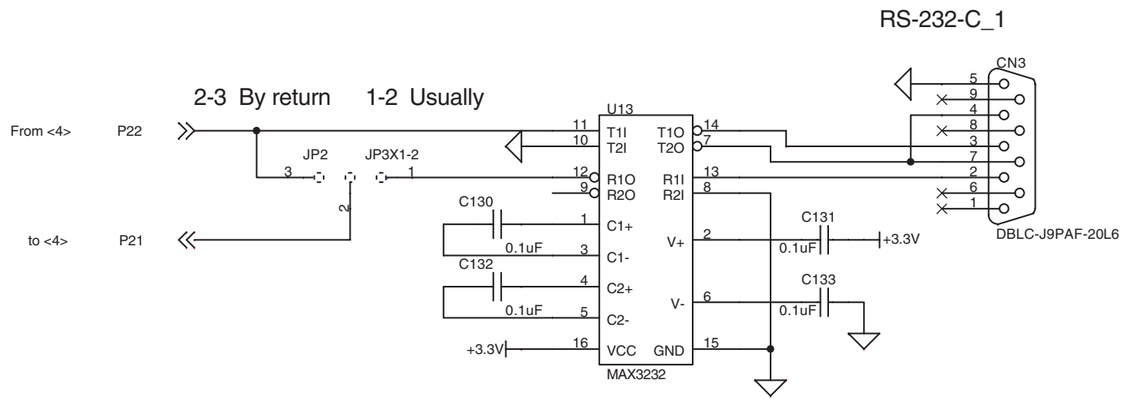
Remark <3> to <7> and <10> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-9. Analog and Serial Interface Circuits



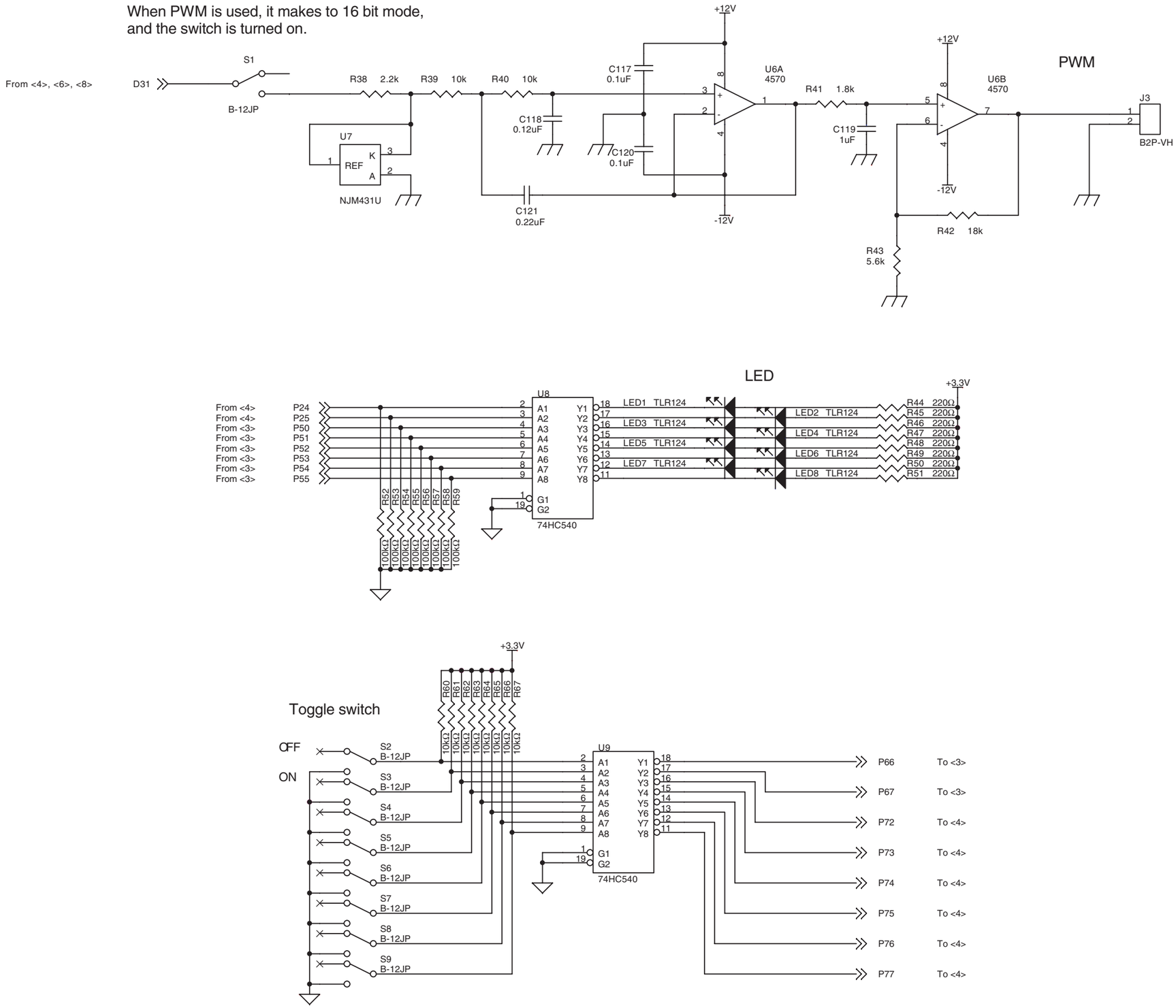






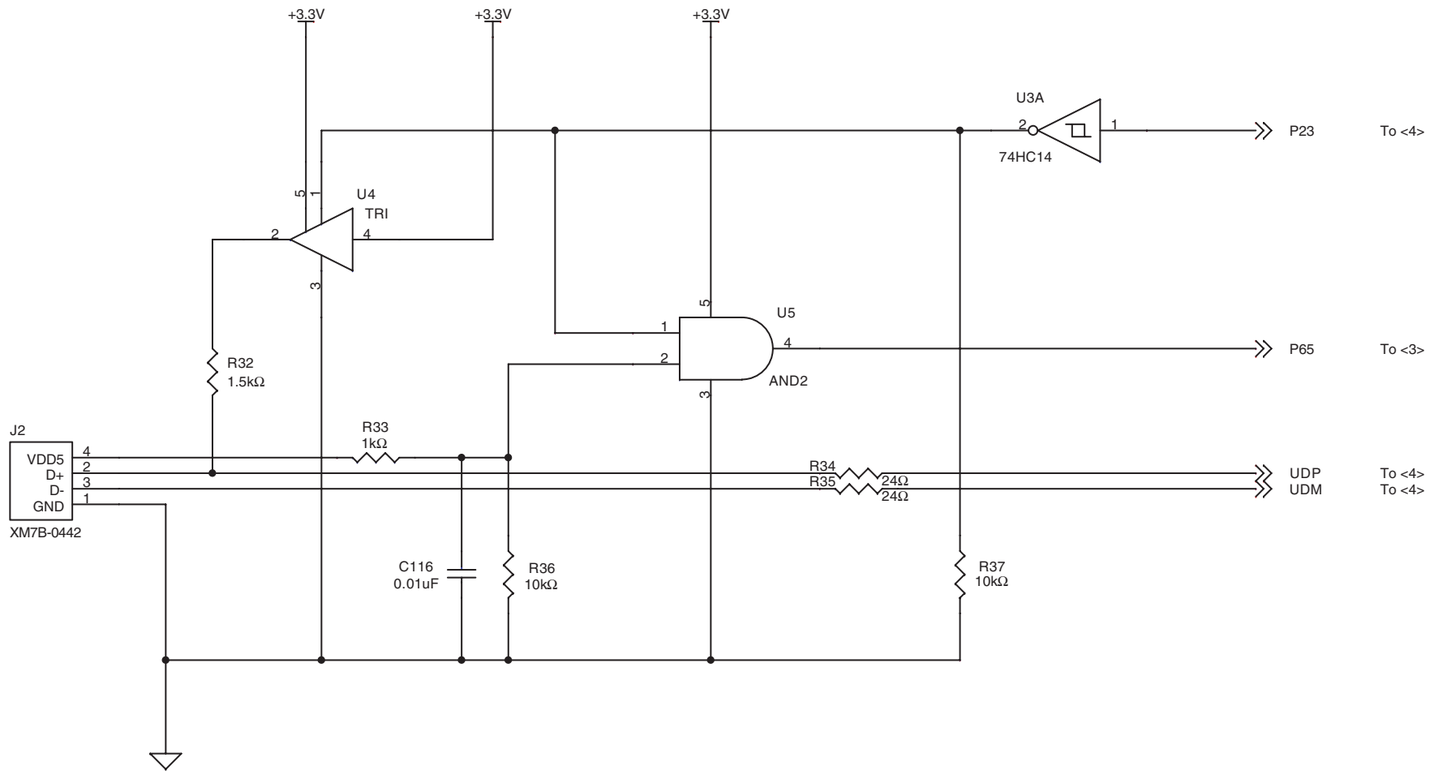
Remark <3> and <4> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-10. PWM and I/O Circuits



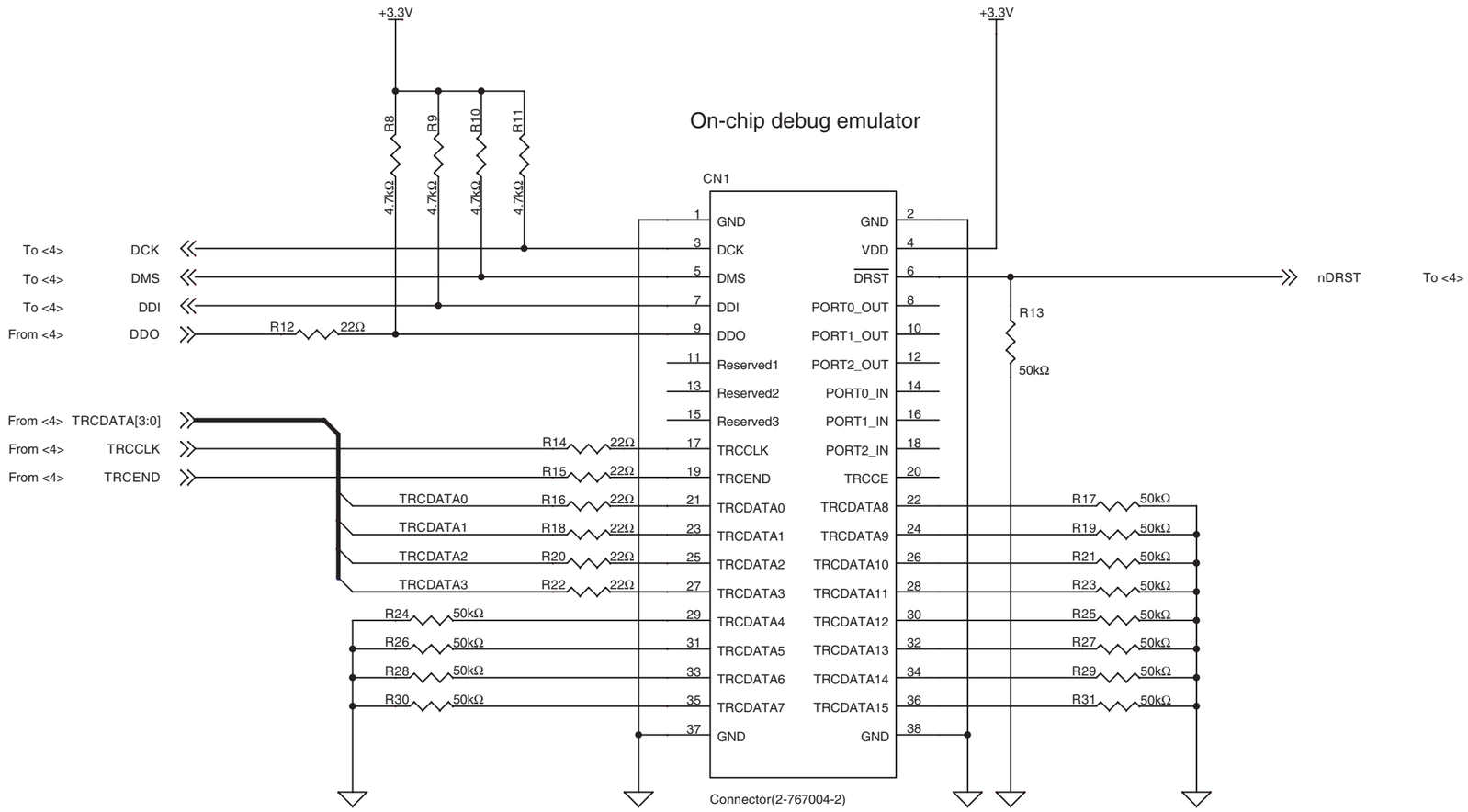
Remark <3>, <4>, <6>, and <8> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-11. USB Peripheral Circuit



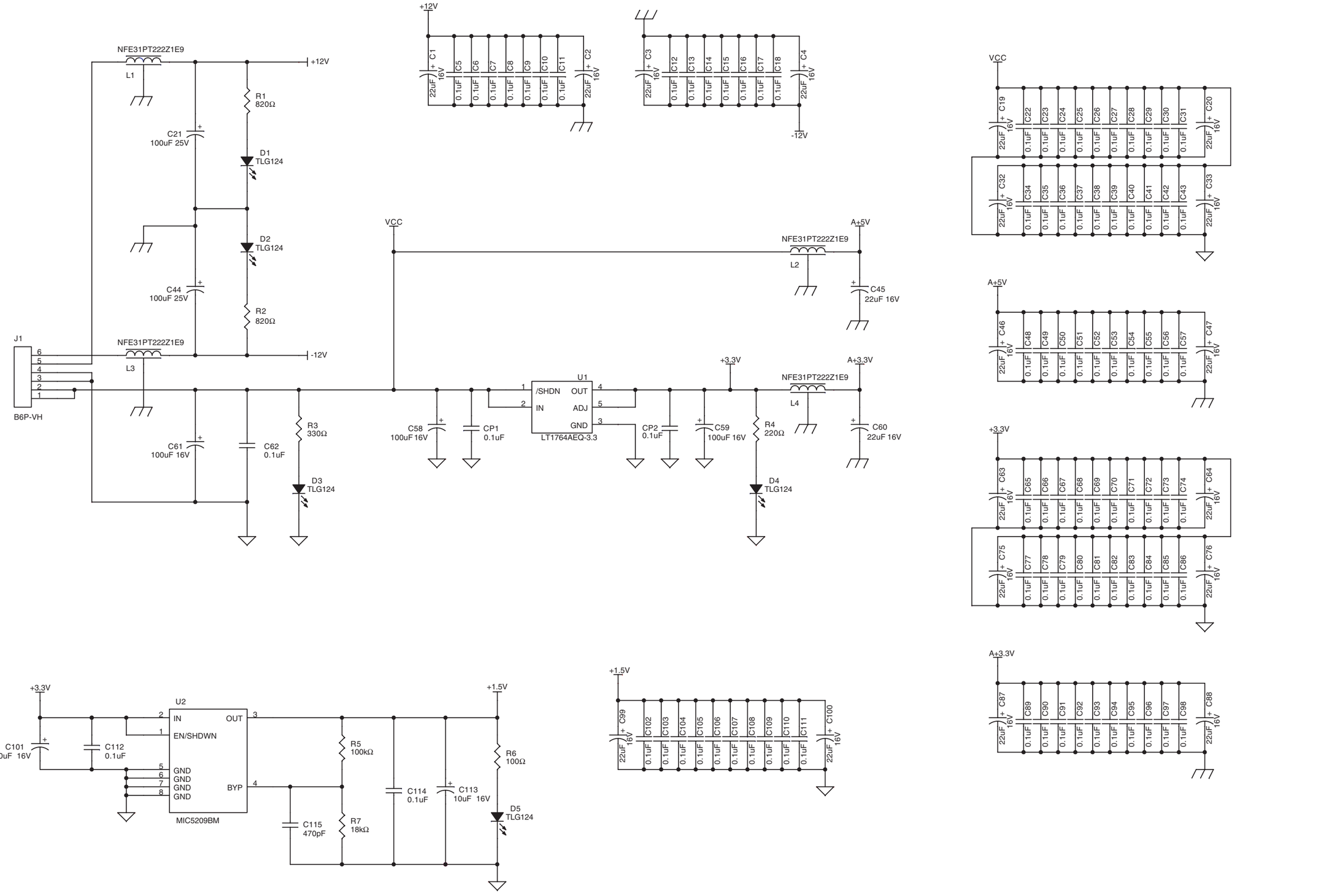
Remark <3> and <4> indicate the numbers of figures (e.g., <3> represents Figure 4-3).

Figure 4-12. On-Chip Debug Emulator



Remark <4> represents Figure 4-4.

Figure 4-13. Power Supply Peripheral Circuits



4.2 Sample Program

4.2.1 Development tools

All program examples shown in this section are created by using the development environment of NEC Electronics.

(1) Compiler package (CA850)

The following four user's manuals concerning the C compiler package are available.

- CA850 C Compiler Package – Operation (U17293E)
- CA850 C Compiler Package – C Language (U17291E)
- C850 C Compiler Package – Assembly Language (U17292E)
- C850 C Compiler Package – Link Directives (U17294E)
- PM+ (U17178E)

Remark The #pragma expansion description in C and abbreviated descriptions of peripheral I/O register names and descriptions of quasi directives in assembly language are peculiar to the C compiler package (CA850) of NEC Electronics. These descriptions may not be able to be used as is when using a partner manufacturer's tool.

(2) ID850NW C source debugger for V850 Series

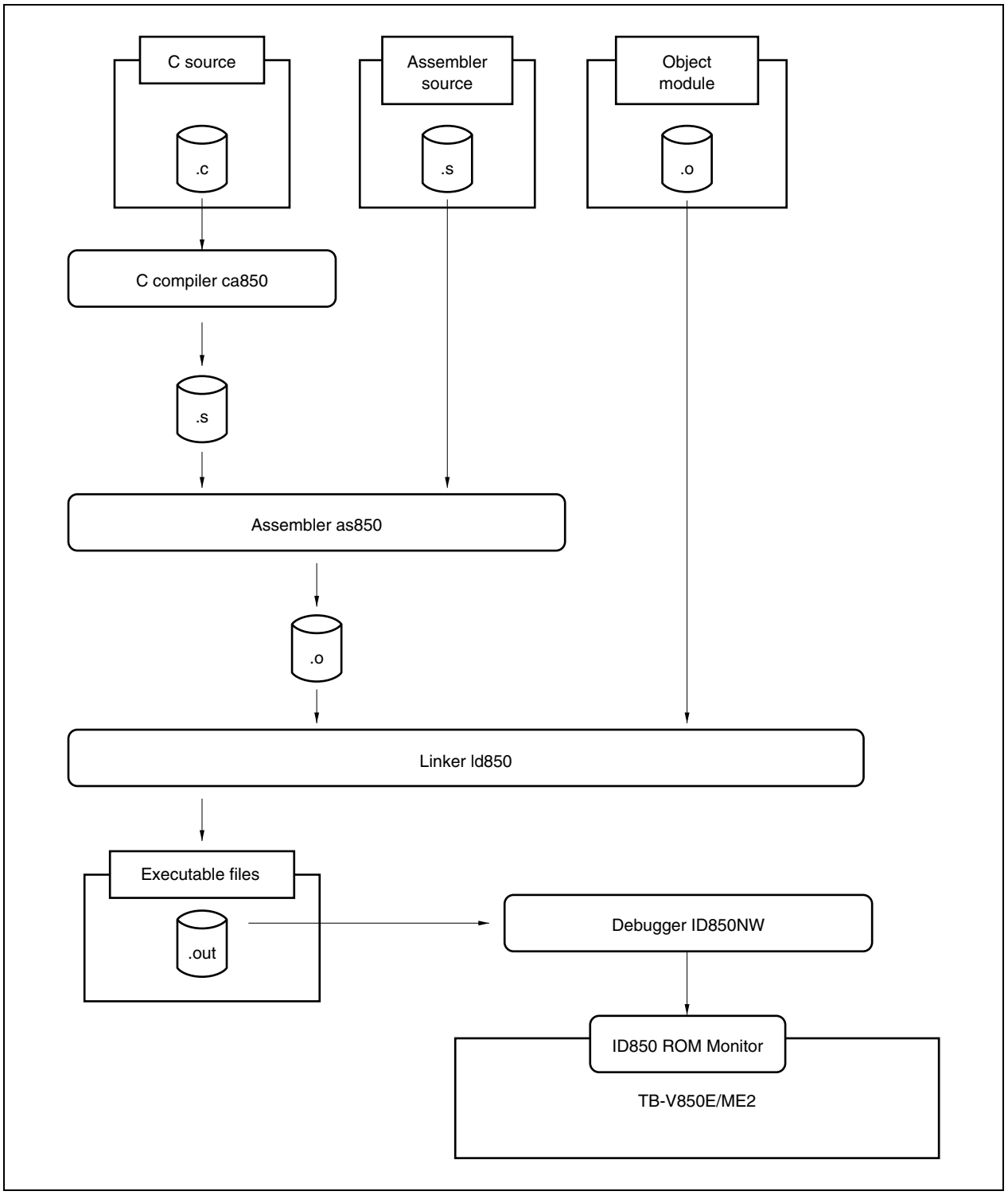
The user's manual concerning the C source debugger for V850 Series is as follows.

- ID850NW Integrated Debugger – Operation (U17369E)

(3) TB-V850E/ME2

This is a daughter board to be connected to the V850E/ME2 evaluation board. For details of the TB-V850E/ME2, refer to **4.1 Functions of TB-V850E/ME2**.

Figure 4-14. Configuration of Development Tools



If you are using an NEC Electronics development tool for the first time, it is recommended that you read the following manual first.

- V800 Series Development Tools Supporting 32-bit OS Tutorial Guide (Windows™ Based) Application Note (U15196E)

4.2.2 Program configuration

The sample program is organized as follows.

(1) Example of simple program operation in internal instruction RAM on initialization of CPU after reset

The reset handler address of the V850E/ME2 is 100000H. Because this address is in the external memory space, the program is expanded to the internal instruction RAM of the CPU after the CPU is initialized, in order to make the best use of the intrinsic high speed of the CPU.

Usually, the initialization sequence is executed in the following three steps after the V850E/ME2 is reset.

<1> Automatic branch to address 100000H and execution of next processing after power application or release of reset

- Initialization of CPU basic functions (refer to **4.2.3 Example of initialization of CPU basic functions**).
- Initialization of bus control alternate-function pins (refer to **4.2.4 Example of initializing bus control alternate-function pins**).
- Initialization of alternate-function pins other than above (refer to **4.2.5 Example of initializing general-purpose I/O port alternate-function pins**).

<2> Check of LOCK bit of lock register and execution of next processing

- Setting frequency division value of external bus (refer to **4.2.6 Example of initializing clock**).
- Setting internal system clock frequency division value (refer to **4.2.6 Example of initializing clock**).
- Setting of main clock control supply (refer to **4.2.6 Example of initializing clock**).
- Initialization related to SDRAM (refer to **4.2.7 Example of initializing SDRAM**).

<3> Transfer of program code to internal instruction RAM and execution of next processing

- Jumping to beginning of program (refer to **4.2.8 Example of internal instruction RAM transfer** and **4.2.9 Example of program to be transferred to internal instruction RAM**).

(2) Example of program dynamically executed in internal instruction RAM

Two or more programs in the external EPROM space are expanded to the internal instruction RAM at the same addresses and executed. If the scale of the application is so large that all the programs do not fit in the internal instruction RAM space, the program to be executed is downloaded from the external memory space to the internal instruction RAM space as necessary. This operation is explained in **4.2.10 Example of program dynamically executed in internal instruction RAM**.

(3) Example of operation using peripheral function

The operation of the internal peripheral I/O and external device is checked. This operation is explained in **4.2.11 Example of FIFO program of internal UARTBn**.

4.2.3 Example of initialization of CPU basic functions

Be sure to set the following registers first in the sequence of initializing the V850E/ME2.

- System wait control register (VSWC)
- Data wait control registers 0 and 1 (DWC0, DWC1)
- Address setup wait control register (ASC)
- Bus cycle control register (BCC)

After that, set chip area select control registers 0 and 1 (CSC0 and CSC1), bus cycle type configuration registers 0 and 1 (BCT0 and BCT1), local bus sizing control register (LBS), endian configuration register (BEC), line buffer control registers 0 and 1 (LBC0 and LBC1), page ROM configuration register (PRC), and port DH function control register (PFCDH), as necessary.

(1) Initializing system wait function

Set the system wait control register as follows.

Figure 4-15. Setting of System Wait Control Register (VSWC)

Address: FFFFF06EH

	7	6	5	4	3	2	1	0
Default value	0	1	1	1	0	1	1	1
Bit name	–	–	–	–	–	–	–	–
Set value	0	0	1	1	0	0	1	1

Operating Frequency (f_X)	Set Value of VSWC
125.00 MHz < $f_X \leq$ 150.00 MHz	33H

[Program example]

```
[file:initial.s]
#########################################################
# Setting of registers affecting external bus access performance
#########################################################
#    Wait setting for internal peripheral I/O access ####
     mov    0x33,  r6            -- vswc set value where 125.00 MHz < fx ≤ 150.00 MHz
     st.b   r6,    VSWC[r0]      -- System wait control register
```

(2) Points on programmable wait (wait function) initialization

- The internal instruction RAM (in the read mode) and internal data RAM area are not subject to programmable waits, and are always accessed without a wait. The internal peripheral I/O area is not subject to programmable waits, and wait control is performed according to the setting of the VSWC register.
- Wait control for page ROM on-page access and SDRAM access is performed by each memory controller. The setting of DWC0 and DWC1 is invalid.
- Write the DWC0 and DWC1 registers after reset. Do not change the values of these registers after they are written.

Figure 4-16. Setting of Data Wait Control Registers 0 and 1 (DWC0 and DWC1)

DWC0 Address: FFFFF484H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
Bit name	0	DW32	DW31	DW30	0	DW22	DW21	DW20	0	DW12	DW11	DW10	0	DW02	DW01	DW00
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
		$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$		

DWC1 Address: FFFFF486H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
Bit name	0	DW72	DW71	DW70	0	DW62	DW61	DW60	0	DW52	DW51	DW50	0	DW42	DW41	DW40
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$		

Bit position	Bit name	Function
14 to 12, 10 to 8, 6 to 4, 2 to 0	DWn2 to DWn0	Specify the number of wait states to be inserted in the CSn space. • $\overline{CS0}$ (EPROM space (16 bits wide)): 3 wait states inserted • $\overline{CS1}$ (SDRAM space (32 bits wide)): No wait state • $\overline{CS2}$ (SRAM space (32 bits wide)): No wait state • $\overline{CS3}$ (SRAM space (16 bits wide)): No wait state • $\overline{CS4}$ (SDRAM space (16 bits wide)): No wait state • $\overline{CS5}$ (SRAM space (8 bits wide)): No wait state • $\overline{CS6}$ (SDRAM space (8 bits wide)): No wait state • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies data waits for each CS space
#########################################################
     mov    0x0003,    r6          -- CS0:  Inserts 3 waits (TW) in EPROM space
                                   -- CS1:  Inserts 0 waits (TW) in SDRAM space
                                   -- CS2:  Inserts 0 waits (TW) in SRAM space
                                   -- CS3:  Inserts 0 waits (TW) in SRAM space
     mov    0x0000,    r7          -- CS4:  Inserts 0 waits (TW) in SDRAM space
                                   -- CS5:  Inserts 0 waits (TW) in SRAM space
                                   -- CS6:  Inserts 0 waits (TW) in SDRAM space
                                   -- CS7:  None
     st.h   r6,        DWC0[r0]    -- Data wait control register 0
     st.h   r7,        DWC1[r0]    -- Data wait control register 1
```

(3) Points on address setup wait (wait function) initialization

- The internal instruction RAM (in read mode), internal data RAM area, and internal peripheral I/O area are not subject to address setup wait insertion.
- When the internal instruction RAM (in write mode) is accessed, the address setup wait set value for the $\overline{CS0}$ space is valid.
- An address setup wait cannot be inserted by the external wait function via the $\overline{WAIT}$ pin.
- Write ASC register after reset. After that, do not change its value.

Figure 4-17. Setting of Address Setup Wait Control Register (ASC)

Address: FFFFF48AH

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Bit name	AC71	AC70	AC61	AC60	AC51	AC50	AC41	AC40	AC31	AC30	AC21	AC20	AC11	AC10	AC01	AC00
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Bit position	Bit name	Function
15 to 0	ACn1, ACn0	Specify the number of address setup wait states to be inserted before the SRAM/page ROM cycle for each $\overline{CSn}$ space. • $\overline{CS0}$ to $\overline{CS6}$: Not inserted • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies address setup waits for each CS space
#########################################################
     mov    0x0000,   r6          -- CS0: Inserts 0 waits (TASW) in EPROM space
                                  -- CS1: Inserts 0 waits (TASW) in SDRAM space
                                  -- CS2: Inserts 0 waits (TASW) in SRAM space
                                  -- CS3: Inserts 0 waits (TASW) in SRAM space
                                  -- CS4: Inserts 0 waits (TASW) in SDRAM space
                                  -- CS5: Inserts 0 waits (TASW) in SRAM space
                                  -- CS6: Inserts 0 waits (TASW) in SDRAM space
                                  -- CS7: None
     st.h   r6,       ASC[r0]     -- Address setup wait control register
```

(4) Points on idle state insertion function initialization

- The internal instruction RAM (in read mode), internal data RAM area, and internal peripheral I/O area are not subject to idle state insertion.
- When the internal instruction RAM (in write mode) is accessed, the idle state value set for the $\overline{CS0}$ space becomes valid.
- Write to the BCC register after reset, and then do not change the set values.
- The $\overline{CSn}$ signal is not asserted in the idle state (n = 0 to 7).

Figure 4-18. Setting of Bus Cycle Control Register (BCC)

Address: FFFFF488H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Bit name	BC71	BC70	BC61	BC60	BC51	BC50	BC41	BC40	BC31	BC30	BC21	BC20	BC11	BC10	BC01	BC00
Set value	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Bit position	Bit name	Function
15 to 0	BCn1, BCn0	Specify insertion of idle states in the CSn space. • $\overline{CS0}$ (EPROM space): 2 idle states inserted • $\overline{CS1}$, $\overline{CS4}$, $\overline{CS6}$: Not inserted • $\overline{CS2}$, $\overline{CS3}$, $\overline{CS5}$: 1 idle state inserted • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies idle states for each CS space
#########################################################
     mov   0x0452,  r6          -- CS0: Inserts 2 idles (TI) in EPROM space
                                -- CS1: Inserts 0 idles (TI) in SDRAM space
                                -- CS2: Inserts 1 idles (TI) in SRAM space
                                -- CS3: Inserts 1 idles (TI) in SRAM space
                                -- CS4: Inserts 0 idles (TI) in SDRAM space
                                -- CS5: Inserts 1 idles (TI) in SRAM space
                                -- CS6: Inserts 0 idles (TI) in SDRAM space
                                -- CS7: None
     st.h  r6,       BCC[r0]    -- Bus cycle control register
```

(5) Points on memory block function initialization

The 256 MB memory space is divided into four 64 MB areas (areas 0 to 3).

Area 0: $\overline{CS0}$, $\overline{CS1}$, and $\overline{CS2}$ signals selectable in the lower 8 MB area (0000000H to 07FFFFFH)

Area 0: $\overline{CS1}$ signal in the remaining 56 MB area (0800000H to 3FFFFFFH)

Area 1: $\overline{CS3}$ signal fixed in the 64 MB area (4000000H to 7FFFFFFH)

Area 2: $\overline{CS4}$ signal fixed in the 64 MB area (8000000H to BFFFFFFH)

Area 3: $\overline{CS6}$ signal in the first 56 MB area (C000000H to F7FFFFFH)

Area 3: $\overline{CS7}$, $\overline{CS6}$, and $\overline{CS5}$ signals selectable in the higher 8 MB area (F8000000H to FFFFFFFH)

Figure 4-19. Setting of Chip Area Select Control Register 0 (CSC0)

Address: FFFFF060H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1
Bit name	CS33	CS32	CS31	CS30	CS23	CS22	CS21	CS20	CS13	CS12	CS11	CS10	CS03	CS02	CS01	CS00
Set value	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1

Setting these bits is meaningless. Setting these bits is meaningless.

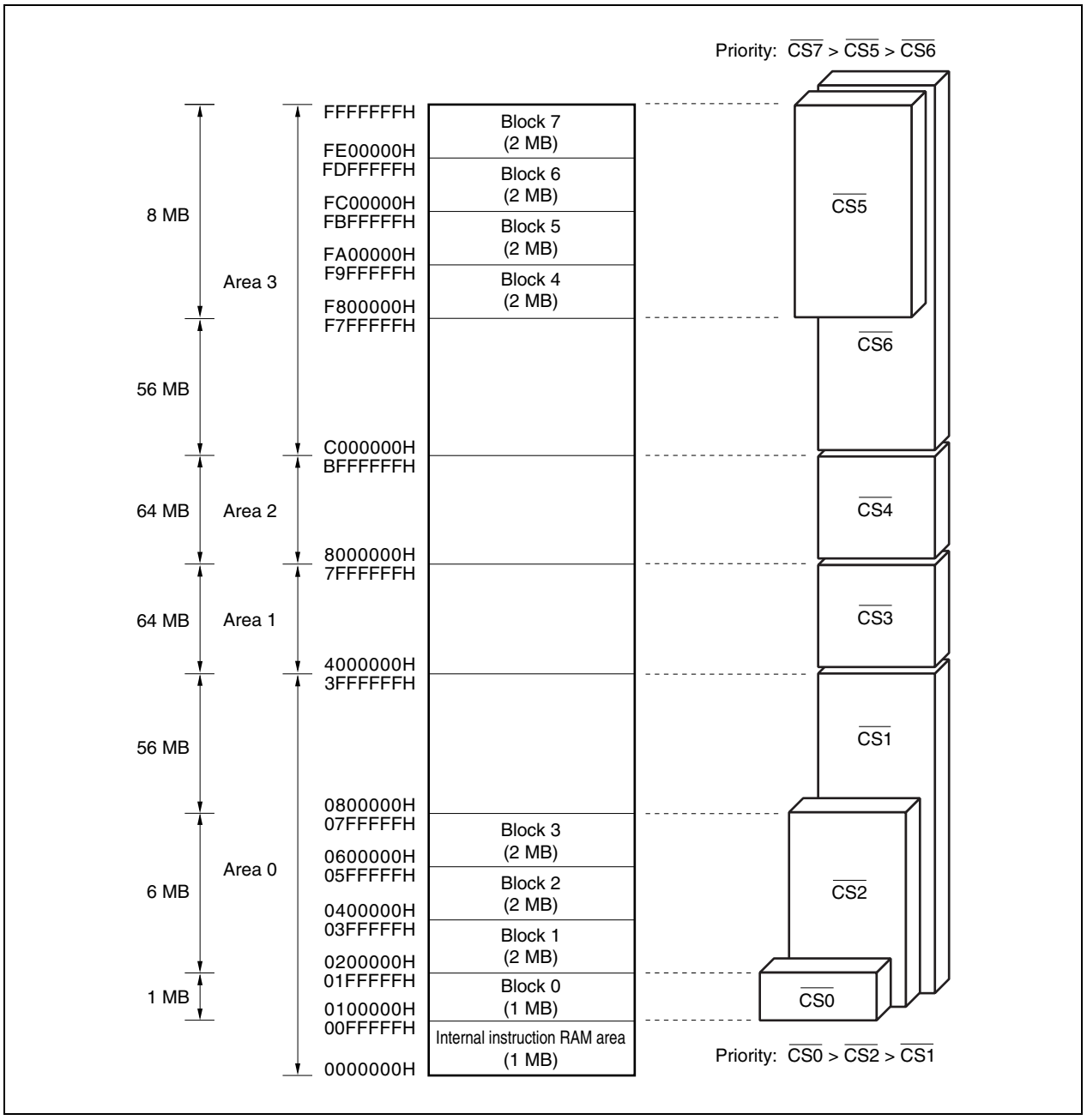
Figure 4-20. Setting of Chip Area Select Control Register 1 (CSC1)

Address: FFFFF062H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1
Bit name	CS43	CS42	CS41	CS40	CS53	CS52	CS51	CS50	CS63	CS62	CS61	CS60	CS73	CS72	CS71	CS70
Set value	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Setting these bits is meaningless. Setting these bits is meaningless.

Figure 4-21. Setting of $\overline{CSn}$ Signal in Program Example (CSC0 Register = 0E01H, CSC1 Register = 0F00H)



[Program example]

```
[file:initial.s]
#########################################################
#     Of 256 MB memory space,
#     divides lower 8 MB (00000000H to 07FFFFFFH) and
#     higher 8 MB (0F800000H to 0FFFFFFFH)
#     into memory blocks in 2 MB units,
#     and specifies chip select signal
#########################################################
     mov   0x0e01,  r6          -- Area 0 (64 MB)
                                -- Block 0 x0100000H to x01fffffH: (1 MB) CS0
                                -- Block 1 x0200000H to x03fffffH: (2 MB) CS2
                                -- Block 2 x0400000H to x05fffffH: (2 MB) CS2
                                -- Block 3 x0600000H to x07fffffH: (2 MB) CS2
                                --          x0800000H to x3fffffffH: (56 MB) fixed to CS1

                                -- Area 1 (64 MB)
                                --          x4000000H to x7fffffffH: (64 MB) fixed to CS3
                                -- Area 2 (64 MB)
                                --          x8000000H to xbfffffffH: (64 MB) fixed to CS4

     mov   0x0f00,  r7          -- Area 3 (64 MB)
                                --          xc000000H to xf7fffffH: (56 MB) fixed to CS6
                                -- Block 4 xf800000H to xf9fffffH: (2 MB) CS5
                                -- Block 5 xfa00000H to xfbfffffH: (2 MB) CS5
                                -- Block 6 xfc00000H to xfdfffffH: (2 MB) CS5
                                -- Block 7 xfe00000H to xfefffffH: (2 MB) CS5
                                -- CS7 not used
     st.h  r6,      CSC0[r0]   -- Chip area select control register 0
     st.h  r7,      CSC1[r0]   -- Chip area select control register 1
```

(6) Points on bus cycle type control function initialization

Write to the BCT0 and BCT1 registers after reset, and then do not change the set values.

Figure 4-22. Setting of Bus Cycle Type Configuration Registers 0 and 1 (BCT0 and BCT1)

BCT0 Address: FFFFF480H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
Bit name	ME3	0	BT31	BT30	ME2	0	BT21	BT20	ME1	0	BT11	BT10	ME0	0	BT01	BT00
Set value	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0
			$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$	

BCT1 Address: FFFFF482H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
Bit name	ME7	0	BT71	BT70	ME6	0	BT61	BT60	ME5	0	BT51	BT50	ME4	0	BT41	BT40
Set value	0	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1
			$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$	

Bit position	Bit name	Function
15, 11, 7, 3	MEn	Enable memory controller operation for $\overline{CSn}$ signal. • $\overline{CS7}$: Operation disabled • $\overline{CS0}$ to $\overline{CS6}$: Operation enabled
13, 12, 9, 8, 5, 4, 1, 0	BTn1, BTn0	Specify the device to be directly connected to each $\overline{CSn}$ signal. • $\overline{CS0}$, $\overline{CS2}$, $\overline{CS3}$, $\overline{CS5}$: SRAM or external I/O • $\overline{CS1}$, $\overline{CS4}$, $\overline{CS6}$: SDRAM • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies external device to be connected to each CS space
#########################################################
     mov   0x88b8,    r6          -- CS0: EPROM  x0100000H to x011ffffH (128 KB)
                                   -- CS1: SDRAM  x1000000H to x2fffffH (32 MB)
                                   -- CS2: SRAM    x0200000H to x027ffffH (512 KB)
                                   -- CS3: SRAM    x4000000H to x403ffffH (256 KB)
     mov   0x0b8b,    r7          -- CS4: SDRAM  x8000000H to x8fffffH (16 MB)
                                   -- CS5: SRAM    xF800000H to xF81ffffH (128 KB)
                                   -- CS6: SDRAM  xC000000H to xCfffffH (16 MB)
                                   -- CS7: None
     st.h  r6,        BCT0[r0]    -- Bus cycle type configuration register 0
     st.h  r7,        BCT1[r0]    -- Bus cycle type configuration register 1
```

(7) Points on bus sizing function initialization

Write to the LBS register after reset, and then do not change the set value.

Figure 4-23. Setting of Local Bus Sizing Control Register (LBS)

Address: FFFFF48EH

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Bit name	LB71	LB70	LB61	LB60	LB51	LB50	LB41	LB40	LB31	LB30	LB21	LB20	LB11	LB10	LB01	LB00
Set value	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

Bit position	Bit name	Function
15 to 0	LBn1, LBn0	Specify the data bus width of the $\overline{CSn}$ space. • $\overline{CS5}$, $\overline{CS6}$: 8 bits • $\overline{CS0}$, $\overline{CS3}$, $\overline{CS4}$: 16 bits • $\overline{CS1}$, $\overline{CS2}$: 32 bits • $\overline{CS7}$ (unused area)

Caution The default value if the V850E/ME2 is started in the 32-bit mode is AAAAH, which sets all the $\overline{CS}$ spaces to a 32-bit data bus width.

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies data bus width for each CS space
#########################################################
     mov    0x0169,  r6        -- CS0: EPROM   x0100000H to x011ffffH (128 KB)  16 bits
                               -- CS1: SDRAM   x1000000H to x2ffffffH (32 MB)    32 bits
                               -- CS2: SRAM    x0200000H to x027ffffH (512 KB)  32 bits
                               -- CS3: SRAM    x4000000H to x403ffffH (256 KB)  16 bits
                               -- CS4: SDRAM   x8000000H to x8ffffffH (16 MB)   16 bits
                               -- CS5: SRAM    xF800000H to xF81ffffH (128 KB)  8 bits
                               -- CS6: SDRAM   xC000000H to xCffffffH (16 MB)   8 bits
                               -- CS7: None                                     8 bits
     st.h   r6,       LBS[r0] -- Local bus sizing control register
```

(8) Points on endian control function initialization

- The internal instruction RAM, internal data RAM area (and internal data RAM mirror area of addresses 3FFB000H to 3FFEFFFH), internal peripheral I/O area, and the program area of the external memory are fixed to the little endian format. Therefore, the setting of the BEC register is invalid.
- Write to the BEC register after reset, and then do not change the set value.
- NEC Electronics' original development tools (such as debugger and compilers) have restrictions in the big endian format.

  For the restrictions, refer to **4.5.4 Bit endian method usage restrictions in NEC Electronics development tools** in the **V850E/ME2 Hardware User's Manual**.

Figure 4-24. Setting of Endian Configuration Register (BEC)

Address: FFFFF068H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit name	0	BE70	0	BE60	0	BE50	0	BE40	0	BE30	0	BE20	0	BE10	0	BE00
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$

Bit position	Bit name	Function
14, 12, 10, 8, 6, 4, 2, 0	BEn0	Specify endian method for each $\overline{CSn}$ space. • $\overline{CS0}$ to $\overline{CS6}$: Little endian • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies endian for each CS space
#########################################################
      mov   0x0000,  r6          -- All CS0 to CS7 spaces in little endian
      st.h  r6,      BEC[r0]     -- Endian configuration register
```

(9) Points on data read control function initialization

- Write data to the LBC0 and LBC1 registers after reset. After writing data to these registers, do not change their values.
- When the speculative read function is set for each $\overline{CSn}$ space, do not insert an idle state in the $\overline{CSn}$ space using the BCC register. If an idle state is required to be inserted in a $\overline{CSn}$ space for which the speculative read function is enabled, enable the speculative read function for all $\overline{CSn}$ spaces (set the LBC0 and LBC1 registers to 3333H) or disable the speculative read function for all $\overline{CSn}$ spaces (set the LBC0 and LBC1 registers to 0000H).

Figure 4-25. Setting of Line Buffer Control Registers 0 and 1 (LBC0 and LBC1)

LBC0 Address: FFFFF490H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit name	0	0	RB31	RB30	0	0	RB21	RB20	0	0	RB11	RB10	0	0	RB01	RB00
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$	

LBC1 Address: FFFFF492H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit name	0	0	RB71	RB70	0	0	RB61	RB60	0	0	RB51	RB50	0	0	RB41	RB40
Set value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$	

Bit position	Bit name	Function
13, 12, 9, 8, 5, 4, 1, 0	RBn1, RBn0	Specify the operating conditions of the read buffer (timing of speculative read) for each $\overline{CSn}$ space. • $\overline{CS0}$ to $\overline{CS6}$: Speculative read disabled (operation of read buffer disabled). • $\overline{CS7}$ (unused area)

Remark n = 0 to 7

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies data read control function for each CS space
#########################################################
     mov   r0,   r6          -- Speculative read disabled for CS0 to CS3 spaces (read buffer operation disabled)
     mov   r0,   r7          -- Speculative read disabled for CS4 to CS7 spaces (read buffer operation disabled)
     st.h  r6,   LBC0[r0]    -- Line buffer control register 0
     st.h  r7,   LBC1[r0]    -- Line buffer control register 1
```

(10) Points on page ROM initialization

- Write data to the PRC register after reset. After writing data, do not change the set value.
- Insertion of wait states is controlled by the PRC register for on-page access.
- Insertion of wait states is controlled by the DWCn register for off-page access (n = 0 or 1).

Because page ROM is not used in this program example, each of the above registers remains at the default value.

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies wait for on-page access (not provided)
#########################################################
--    mov    x7000,   r6          -- 2 × 32 bits, 4 × 16 bits, 8 × 8 bits
                                  -- 7 waits (TW) inserted for on-page access
--    st.h   r6,      PRC[r0]     -- Page ROM configuration register
```

(11) Notes on starting external data bus in 16-bit mode (MODE1 and MODE0 = 01) and also using 32-bit data bus

- Whether the D16 to D31 pins are used or not can be specified by the BMODCN bit of the PFCDH register.

Bit Name	Meaning
BMODCN	Specifies use of the D16 to D31 pins in the 16-bit mode (16-bit data bus). 0: Do not use the D16 to D31 pins on starting in 16-bit mode (data bus width: 16/8 bits). 1: Use the D16 to D31 pins on starting in 16-bit mode (data bus width: 32/16/8 bits).

- To start the external data bus in the 16-bit mode and also use the 32-bit data bus, write 1 to the BMODCN bit of the PFCDH register and be sure to confirm that the read value is 1. Then access an external device with a 32-bit wide bus.
- Rewriting the BMODCN bit is valid only once. Operations are not guaranteed if this bit is rewritten twice or more.
- If the external data bus is started in the 16-bit mode (MODE1 and MODE0 = 01), the BMODCN bit does not have to be set if there is no external element that uses a 32-bit data bus.

[Program example]

```
[file:initial.s]
#########################################################
#     Specifies use of D16 to D31 pins
#########################################################
-- Setting of PMDH, PMCDH, and PFCDH is in 16-bit bus mode
-- 16-bit bus mode when MODE1 pin = 1 and MODE0 pin = 0
     mov    0x0000,    r6          -- Sets inactive level (low) of PDH15 to PDH0
     st.h   r6,        PDH[r0]
     mov    0x0000,    r6          -- PDH15 to PDH0 in output mode
     st.h   r6,        PMDH[r0]
     mov    0x8000,    r6          -- PDH15 as control pin (INTPD15/PWM1)
     st.h   r6,        PMCDH[r0]
-- 32-bit data bus can also be used when BMODCN of PFCDH = 1
--   mov    0x8000,    r6          -- PDH15 as PWM1 pin
     mov    0x0001,    r6          -- PDH15 to PDH0 as D31 to D16 pins on starting in 16-bit mode
     st.h   r6,        PFCDH[r0]   -- Writes data to port DH function control register
-- PDH15 to PDH0 as D31 to D16 pins when BMODCN of PFCDH = 1,
-- and above setting of PDH, PMDH, and PMCDH is invalid

_BMODCN_NOFIX:
     ld.h   PFCDH[r0],r6          -- Reads port DH function control register
     movea  0x0001,    r0, r7     -- Extracts BMODCN bit
     and    r7,        r6
     cmp    r6,        r7         -- BMODCN setting incomplete?
     jnz    _BMODCN_NOFIX         -- Yes
-- 32-bit bus mode selected when MODE1 and MODE0 pins = 0,
-- and above setting of PMDH, PMCDH, and PFCDH is invalid
```

4.2.4 Example of initializing bus control function alternate-function pins

In addition to initializing the CPU basic functions, the bus control function alternate-function pins must also be initialized.

First, the basic concept of how to set the bus control function alternate-function pins is described. The bus control function pins function alternately as general-purpose I/O (port) pins, interrupt function pins, and other control function pins. The alternate functions are selected by a register. These pins are set in three patterns.

Table 4-1. Setting of Alternate-Function Pins with Port, Bus Control, Other Control, and Interrupt Functions

Register / Port Name	PMCn	PMn	PFCn	INTRn	INTFn
Port n	Control function	Setting invalid	Bus control function or other control function[Note]	Setting invalid	
			Interrupt function	Falling edge	
				Rising edge	
				Level detection (H/L detection)	
				Both rising and falling edges	
	Port function	Input	Setting invalid		
		Output			

Note Control function of port AL: Bus control (A0 and A1 pins) function only
Control function of port DH: Bus control (D16 to D31) function and other control function

Remarks 1. H: High-level input
L: Low-level input

2. n = AL or DH (when n = AL, the PFCn register serves as the PFCALL register)

Table 4-2. Setting of Alternate-Function Pins with Port and Bus Control Functions

Register / Port Name	PMCn	PMn	PFCn	INTRn	INTFn
Port n	Control function	Setting invalid	None		
	Port function	Input			
		Output			

Remark n = AH or CD

Table 4-3. Setting of Alternate-Function Pins with Port, Bus Control, and Other Bus Control Functions

Register / Port Name	PMCn	PMn	PFCn	INTRn	INTFn
	Control function	Setting invalid	Bus control function	None	
			Other bus control function		
	Port function	Input	Setting invalid		
		Output			

Remark n = CS, CT, or CM

Note the following points when initializing the CPU control function.

- Procedure of changing port mode to control mode
- Notes on manipulating port with bit manipulation instruction
- Notes on setting interrupt trigger mode
- Notes on setting port DH
- Enabling/disabling $\overline{\text{IOWR}}$ and $\overline{\text{IORD}}$ signal output

(1) Procedure of changing port mode to control mode

When changing the mode of a port that functions as an output or I/O pin in the control mode from the port mode to the control mode, be sure to follow the procedure below.

<1> Set the inactive level of the signal output in the control mode to the corresponding bit of port n (n = AL, AH, DH, CS, CT, CM, or CD).

<2> Switch to control mode by using the port n mode control register (PMCn).

Caution If <1> above is not performed, the contents of port n may be momentarily output when the mode is changed from the port mode to the control mode (n = AL, AH, DH, CS, CT, CM, or CD).

(2) Notes on manipulating port with bit manipulation instruction

To manipulate a port by using a bit manipulation instruction (SET1, CLR1, or NOT1), read byte data from the port, process the data of only the bit to be manipulated, and then write back the modified byte data back to the port (read-modify-write). For example, the contents of the output latch are written to the bits other than the one to be manipulated if the port has both input and output pins. Consequently, the output latch of the input pin becomes undefined (however, the pin status does not change in the input mode because the output buffer is off). To change the mode of the port from the input to the output mode, set an expected output value to the corresponding bit of the port n register (Pn), and then select the output mode (n = AL, AH, DH, CS, CT, CM, or CD). The same applies to a port that has both a control mode and an output port mode.

(3) Notes on setting interrupt trigger mode

Set the trigger mode after setting the PMCn register (n = AL or DH).

If the PMCn register is set after the INTRn and INTFn registers are set, an illegal interrupt may be generated, depending on the timing of setting the PMCn register.

(4) Notes on setting port DH

(a) Bus control (D16 to D31) function

<1> Changing function of port DH to bus control (D16 to D31) function

Changing the function of port DH to the bus control (D16 to D31) function is basically determined by the operation mode specified by the MODE1 and MODE0 pins. The relationship between the MODE1 and MODE0 pins, and the port DH setting register is shown below.

Table 4-4. Relationship Between MODE1 and MODE0 Pins, and Port DH Setting Register

Mode / Register	16-bit Mode (MODE1 and MODE0 = 01)	32-bit Mode (MODE1 and MODE0 = 00)
PDH	Valid (function as port or control pins)	Invalid (function as D16 to D31 pins)
PMDH	Valid (selection of input or output)	Invalid (meaningless)
PMCDH	Valid (selection of I/O port or control pin)	Invalid (meaningless)
PFCDH	Valid (selection of control pin function and enabling or disabling D16 to D31 pin functions)	Invalid (meaningless)

<2> Changing function of port DH to bus control (D16 to D31) function in 16-bit bus mode

Even in the 16-bit mode, the function of port DH can be changed to the bus control (D16 to D31) function, and an external device with a 32-bit data bus can also be used. This is done by using the BMODCN bit of the PFCDH register. The control pin functions of port DH are shown below.

Table 4-5. Setting of PDH Control Pin Function by MODE1 and MODE0 Pins and BMODCN Bit

Control / Port Name	MODE1	MODE0	Data Mode	BMODCN	Pin Function
Port DH	Low level	Low level	32-bit mode	Setting invalid	D16 to D31
	Low level	High level	16-bit mode	1	
				0	Other control functions

Note that the default value of the BMODCN bit of the PFCDH register is 0 (D16 to D31 pin functions are disabled (external device with 32-bit data bus cannot be used)) in the 16-bit mode. If the BMODCN bit of the PFCDH register = 1 (D16 to D31 pin functions are enabled (external device with 32-bit data bus can be used)) in the 16-bit mode, the other control functions cannot be used.

For details of this setting, refer to **4.2.3 (11) Notes on starting external data bus in 16-bit mode (MODE1 and MODE0 = 01) and also using 32-bit data bus**.

(b) Notes on pins that function alternately as timer ENC1n control inputs

<1> Changing function of PDH7 and PDH11 pins to timer ENC1n control input

The PDH7 and PDH11 pins are used as the external capture trigger (INTP1n1) input and TCLR1n input pins of timer ENC1n (n = 0 to 1) when the interrupt function is not selected by the PFCDH register. However, there is no register that can change the input mode of these pins. Therefore, the input mode is determined by using the timer ENC1n setting register shown below.

Table 4-6. Selecting External Capture Trigger (INTP1n1) Input and TCLR1n Input of Timer ENC1n

Function of PDH7 and PDH11	Set Value				Interrupt
	TUM1n Register	TMC1n Register	CCR1n Register	CC1nIC1 Register	
As TCLR1n input	1000xx00B	0x00xx00B	0000000xB	x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
			0000001xB	x0000xxxB	INTCC1n1 occurs
				x1000xxxB	INTCC1n1 masked
		0x00xx01B	0000001xB	x0000xxxB	INTCC1n1 occurs
				x1000xxxB	INTCC1n1 masked
		0x00xx10B	0000000xB	x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
			0000001xB	x0000xxxB	INTCC1n1 occurs
				x1000xxxB	INTCC1n1 masked
		0x00xx11B	0000001xB	x0000xxxB	INTCC1n1 occurs
				x1000xxxB	INTCC1n1 masked
	1000xx01B	Setting invalid	0000000xB	x0000xxxB	INTCC1n1 occurs
				x1000xxxB	INTCC1n1 masked
	Other settings prohibited				
As external capture trigger (INTP1n1)	0000xx00B	Setting invalid		x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
	1000xx00B	0x00xx01B	0000000xB	x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
		0x00xx11B		x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
	1000xx01B			x0000xxxB	INTP1n1 occurs
				x1000xxxB	INTP1n1 masked
	Other settings prohibited				

Remark n = 0 or 1

<2> Changing function of PDH6 and PDH10 pins to timer ENC1n control input

The PDH6 and PDH10 pins are used as the external capture trigger input (INTP1n0) and TCUD1n input pins of timer ENC1n when the interrupt function is not selected by the PFCDH register (n = 0 or 1). However, there is no register that can select the input mode of these pins. Therefore, the input mode is determined by using the timer ENC1n setting register shown below.

Table 4-7. Selecting External Capture Trigger (INTP1n0) Input and TCUD1n Input of Timer ENC1n

Function of PDH6 and PDH10	Set Value				Interrupt
	TUM1n Register	TMC1n Register	CCR1n Register	CC1nIC0 Register	
As TCUD1n input	1000xx0xB	Setting invalid	000000x0B	x0000xxxB	INTP1n0 occurs
				x1000xxxB	INTP1n0 masked
			000000x1B	x0000xxxB	INTCC1n0 occurs
				x1000xxxB	INTCC1n0 masked
	Other settings prohibited				
As external capture trigger (INTP1n0)	0000xx00B			x0000xxxB	INTP1n0 occurs
				x1000xxxB	INTP1n0 masked
	Other settings prohibited				

Remark n = 0 or 1

<3> Sequence of setting SESA1n and PMCDH of timer ENC1n control input

Before using port DH as timer ENC1n control input pins and setting the trigger mode of the INTP100, INTP101, INTP110, INTP111, TIUD10, TIUD11, TCUD10, TCUD11, TCLR10, and TCLR11 pins, set the PMCDH register. If the PMCDH register is set after the SESA1n register is set, an illegal interrupt, miscount, or mis-clear may occur, depending on the timing of setting the PMCDH register.

(5) Enabling/disabling $\overline{\text{IOWR}}$ and $\overline{\text{IORD}}$ signal output

Note the following points when enabling or disabling the operation to output the $\overline{\text{IOWR}}$ and $\overline{\text{IORD}}$ signals.

- The $\overline{\text{IORD}}$ and $\overline{\text{IOWR}}$ signals are output during flyby DMA transfer to/from SRAM, external ROM, or external I/O, regardless of the setting of the IOEN bit.

  Flyby transfer from external I/O to external memory: $\overline{\text{IORD}}$ and $\overline{\text{WR}}$ signals are asserted.
  Flyby transfer from external memory to external I/O: $\overline{\text{IOWR}}$ and $\overline{\text{RD}}$ signals are asserted.

  The setting of the IOEN bit is meaningless for a page ROM cycle.

- Write data to the BCP register after reset. After writing, do not change the set value.
- If the internal instruction RAM (in write mode) is accessed with the IOEN bit set to 1, the $\overline{\text{IOWR}}$ signal is asserted.

Figure 4-26. Setting of Bus Cycle Period Control Register (BCP)

Address: FFFFF48CH

	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0
Bit name	0	0	0	0	IOEN	0	0	0
Set value	0	0	0	0	0	0	0	0

Bit position	Bit name	Function
3	IOEN	Enables or disables $\overline{\text{IORD}}$ and $\overline{\text{IOWR}}$ operations in SRAM, external ROM, and external I/O cycles. 0: Disabled

(6) Program example

```
[file:initial.s]
#########################################################
#     Specifies bus control pin
#########################################################
     mov    0x0003,    r6        -- Selects control pin
     st.h   r6,        PMCAL[r0]
     mov    0x03,      r6        -- Selects A0 and A1 output pins
     st.b   r6,        PFCALL[r0]

     mov    0x03ff,    r6        -- Selects A16 to A25 output pins
     st.h   r6,        PMCAH[r0]

     mov    0xff,      r6        -- Sets inactive level (high) of CS0 to CS7 output signals
     st.b   r6,        PCS[r0]
     mov    0xff,      r6        -- Selects CS0 to CS7 pins
     st.b   r6,        PMCCS[r0]

     mov    0xbf,      r6        -- Sets inactive level (high) of
                                 -- BCYST, WE/WR, RD,
                                 -- UUWR/UUBE/UUDQM,
                                 -- ULWR/ULBE/ULDQM,
                                 -- LUWR/LUBE/LUDQM,
     st.b   r6,        PCT[r0]   -- LLWR/LLBE/LLDQM output signals
     mov    0xbf,      r6        -- Selects BCYST, ASTB, WE/WR, RD,
                                 -- UUWR/UUBE/UUDQM,
                                 -- ULWR/ULBE/ULDQM,
                                 -- LUWR/LUBE/LUDQM,
                                 -- LLWR/LLBE/LLDQM output pins
     st.b   r6,        PMCCT[r0]

     mov    0xC0,      r6        -- PCM5 to PCM0 in output mode
     st.b   r6,        PMCM[r0]
     mov    0x00,      r6        -- PCM5 to PCM0 as general-purpose port
     st.b   r6,        PMCCM[r0]

     mov    0x0c,      r6        -- Inactive level (high) of SDRAS and SDCAS output signals
     st.b   r6,        PCD[r0]   -- Sets inactive level (low) of BUSCLK and SDCKE output signals
     mov    0x0f,      r6        -- Selects SDRAS, SDCAS, BUSCLK, and SDCKE output pins
     st.b   r6,        PMCCD[r0]

     mov    0x00,      r6        -- Disables IORD and IOWR operations in SRAM,
                                 -- external ROM, and external I/O cycles
     st.b   r6,        BCP[r0]   -- Bus cycle period control register
```

4.2.5 Example of initializing general-purpose I/O port alternate-function pins

In addition to the bus control function alternate-function pins, initialize the general-purpose I/O port alternate-function pins.

First, the basic concept of how to set the general-purpose I/O port alternate-function pins is described. The general-purpose I/O port pins function alternately as interrupt function pins and a function pin of other than the bus control function. These functions are specified by a register.

Table 4-8. Setting of Alternate-Function Pins with Port Function, Control Function Other Than Bus Control Function, and Interrupt Function

Register / Port Name	PMCn	PMn	PFCn	INTRn	INTFn
Port n	Control function	Setting invalid	Control function other than bus control function	Setting invalid	
			Interrupt function	Falling edge	
				Rising edge	
				Level detection (H/L detection)	
				Both rising and falling edges	
	Port function	Input	Setting invalid		
		Output			

Remarks 1. H: High-level input
L: Low-level input

2. n = 1, 2, or 5 to 7 (when n = 7, INTR7 and INTF7 registers are not used)

The points to be noted when initializing the alternate-function pins with a control function other than bus control function are described next.

- Procedure for changing mode from port mode to control mode
- Notes on manipulating port with bit manipulation instruction
- Notes on setting interrupt trigger mode

(1) Procedure of changing port mode to control mode

When changing the mode of a port that functions as an output or I/O pin in the control mode from the port mode to the control mode, be sure to follow the procedure below.

<1> Set the inactive level of the signal output in the control mode to the corresponding bit of port n (n = 1, 2, or 5 to 7).

<2> Switch to control mode by using the port n mode control register (PMCn).

Caution If <1> above is not performed, the contents of port n may be momentarily output when the mode is changed from the port mode to the control mode (n = 1, 2, or 5 to 7).

(2) Notes on manipulating port with bit manipulation instruction

To manipulate a port by using a bit manipulation instruction (SET1, CLR1, or NOT1), read byte data from the port, process the data of only the bit to be manipulated, and then write back the modified byte data back to the port (read-modify-write). For example, the contents of the output latch are written to the bits other than the one to be manipulated if the port has both input and output pins. Consequently, the output latch of the input pin becomes undefined (however, the pin status does not change in the input mode because the output buffer is off). To change the mode of the port from the input to the output mode, set an expected output value to the corresponding bit of the port n register (Pn), and then select the output mode (n = 1, 2, or 5 to 7). The same applies to a port that has both a control mode and an output port mode.

(3) Notes on setting interrupt trigger mode

Set the trigger mode after setting the PMCn register (n = 1, 2, 5, or 6).

If the PMCn register is set after the INTRn and INTFn register are set, an illegal interrupt may be generated, depending on the timing of setting the PMCn register.

(4) Program example

```
[file:initial.s]
#########################################################
#     Specifies function other than bus control pin.
#########################################################

# External interrupt rising edge specification register 1 (INTR1) at default value (00H)
# External interrupt falling edge specification register 1 (INTR1) at default value (00H)
                                    -- INTP11 and INTP10 pins = falling edge input
# External interrupt rising edge specification register 2 (INTR2) at default value (00H)
# External interrupt falling edge specification register 2 (INTR2) at default value (00H)
                                    -- INTP2n (n = 1 to 5) and NMI pins = falling edge input
# External interrupt rising edge specification register 6 (INTR6) at default value (20H)
# External interrupt falling edge specification register 6 (INTR6) at default value (20H)
                                    -- INTP65 pin = both rising and falling edge input

      st.b   r0,     P1[r0]     -- Sets inactive level (low) of TXD0 output signal
      mov    0x0f,   r6         -- All P13 and 10 are in control pin mode
      st.b   r6,     PMC1[r0]
      mov    0x0d,   r6         -- P13 = TxD0 output, P12 = RxD0 input
                                -- P11 = INTP11 input, P10 = UCLK input
      st.b   r6,     PFC1[r0]

      mov    0x03,   r6         -- Inactive level (low) (OFF) of P25 and P24 (LED)
      st.b   r6,     P2[r0]     -- P23 (USB control) and TxD1 pin at inactive level (low)
      mov    0xc7,   r6
      st.b   r6,     PM2[r0]    -- P25 and P24 (LED) and P23 (USB control) in output mode
      mov    0x07,   r6
      st.b   r6,     PMC2[r0]   -- P25 to P23 as general-purpose port pins, and P22 to P20 as control pins
      mov    0x06,   r6
      st.b   r6,     PFC2[r0]   -- P22 = TxD1, P21 = RxD1, P20 = NMI

      mov    0x00,   r6         -- Inactive level (low) (OFF) of P55 to P50 (LED)
      st.b   r6,     P5[r0]
      mov    0xc0,   r6
      st.b   r6,     PM5[r0]    -- P55 to P50 (LED) in output mode
      st.b   r0,     PMC5[r0]   -- All P55 to P50 as general-purpose port pins

      mov    0xff,   r6
      st.b   r6,     PM6[r0]    -- P67 and P66 (TOGLE SW) and P65 (USB power feed monitor) in input mode
      mov    0x20,   r6
      st.b   r6,     INTR6[r0]
      st.b   r6,     INTF6[r0]
      st.b   r6,     PMC6[r0]   -- P67 and P66 as general-purpose port pins, P65 = INTP65

      mov    0xff,   r6
      st.b   r6,     P7[r0]
      st.b   r6,     PM7[r0]    -- P77 to P72 (TOGLE SW) in input mode
      st.b   r0,     PMC7[r0]   -- All P77 and P72 as general-purpose port pins
```

4.2.6 Example of initializing clock

Initialize the clock related to the internal circuitry of the CPU and external operation.

After initialization operations in 4.2.3 to 4.2.5 are completed, confirm that the LOCK bit of the LOCKR register is cleared (that PLL is locked), and perform clock initialization using the following procedure.

(i) Temporarily set the system wait control register (VSWC).
Set a value of x7H to the register (x: Higher 4 bits of the value first set to VSWC)

(ii) Set the bus mode control register (BMC).
Set the frequency division value of the external bus.

(iii) Write back to the system wait control register (VSWC).
Write the value first set to the VSWC register back to VSWC.

(iv) Set the clock control register (CKC)
Set the frequency division value of the internal system clock

(v) Set the clock source select register (CKS).
Set the CKSSEL bit of the CKS register to 1, and change the frequency of the clock supplied to the CPU (main clock (fx)) from the frequency input to the X1 and X2 pins to the frequency multiplied by eight by the PLL (from OSC output to SSCG output).

Remark The CKC and CKS registers are specific registers and are set in a special write sequence.

(1) Points on bus mode control register (BMC) initialization

Set this register according to initialization sequence step (ii) above.

Figure 4-27. Setting of Bus Mode Control Register (BMC)

Address: FFFFF498H

	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0
Bit name	0	0	0	0	0	0	CKM1	CKM0
Set value	0	0	0	0	0	0	0	1

Bit position	Bit name	Function
1, 0	CKM1, CKM0	Set the division ratio of the bus clock (BUSCLK) with respect to the internal system clock (f_{CLK}). 01: $f_{CLK}/2$

(2) Points on clock control register (CKC) initialization

- The CKC register can be written only in a special sequence.
- Note that if the internal system clock (f_{CLK}) is changed, the frequency of the bus clock (BUSCLK) is also changed.

Figure 4-28. Setting of Clock Control Register (CKC)

Address: FFFFF822H

	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	1	1
Bit name	0	0	0	0	0	0	CKDIV1	CKDIV0
Set value	0	0	0	0	0	0	1	1

Bit position	Bit name	Function
1, 0	CKDIV1, CKDIV0	Set the internal system clock (f_{CLK}) when PLL mode is used. 11: f_X

Remark f_X: Main clock

(3) Points on clock source select register (CKS) initialization

- The CKS register can be written only in a special sequence.
- With the V850E/ME2, it is not assumed that the CPU operates with the OSC output always supplied as the main clock (CKSSEL bit = 0). Therefore, be sure to confirm in the initialization sequence that the LOCK bit of the LOCKR register is 0, and then supply the main clock from the SSCG output (CKSSEL bit = 1). Otherwise, the operation will not be guaranteed.

Figure 4-29. Setting of Clock Source Select Register (CKS)

Address: FFFFF82CH

	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	0
Bit name	0	0	0	0	0	0	0	CKSSEL
Set value	0	0	0	0	0	0	0	1

Bit position	Bit name	Function
0	CKSSEL	Controls supply of the main clock (f_X). 1: SSCG output clock ($F_X \times 8$)

(4) Points on SSCG control register (SSCGC) initialization

- The SSCG control register can be written only in a special sequence.
- The SSCGC register can be set only when OSC output is supplied as the main clock (CKSSEL bit of CKS register = 0). If the setting of the SSCGC register is changed, the SSCG is unlocked (LOCK bit of LOCKR register = 1). Be sure to confirm that the LOCK bit = 0 before starting to supply SSCG output as the main clock (CKSSEL bit = 1). Otherwise, the operation will not be guaranteed.
- Set the PLLSEL, SSEL1, and SSEL0 pins as follows in accordance with the frequency ((Fx) × 8 = fx) input to the X1 and X2 pins

Table 4-9. Frequencies

Multiplication Factor	PLLSEL Pin	SSEL1 Pin	SSEL0 Pin	Input Frequency (MHz) (Target Value)	Main Clock (fx) Frequency (MHz)
8	1	0	1	Setting prohibited	Setting prohibited
		1	0	10.00 to 10.19	80.00 to 81.59
		1	1	10.20 to 11.99	81.60 to 95.99
	0	0	0	12.00 to 14.39	96.00 to 115.19
		0	1	14.40 to 17.39	115.20 to 139.19
		1	0	17.40 to 18.75	139.20 to 150.00
		1	1	Setting prohibited	Setting prohibited

Caution The maximum value of f_{CLK} is 100 MHz in a 100 MHz product, 133 MHz in a 133 MHz product, and 150 MHz in a 150 MHz product.
The operation is not guaranteed if f_{CLK} (MAX.) < f_X.
Make sure that f_X does not exceed the guaranteed operating frequency of each product.

Because fx = 133.33 MHz in this program example, PLLSEL pin = 0, SSEL1 pin = 0, and SSEL0 pin = 1.

Figure 4-30. Setting of SSCG Control Register (SSCGC)

Address: FFFFF836H

	7	6	5	4	3	2	1	0
Default value	0	0	0	1	**Note**	**Note**	**Note**	**Note**
Bit name	0	0	SMDL1	SMDL0	ADJON	ADJ2	ADJ1	ADJ0
Set value	0	0	0	0	0	0	0	0

Bit position	Bit name	Function
5, 4	SMDL1, SMDL0	Set the modulation period of SSCG output. 00: 13 to 27 kHz
3 to 0	ADJON, ADJ2-ADJ0	Set the frequency modulation rate of SSCG output. 0000: No modulation rate (frequency fixed).

Note The default values of the ADJON and ADJ2 to ADJ0 bits are as shown in Table 4-10, according to the setting of the JIT1 and JIT0 pins.

The values of the ADJON and ADJ2 to ADJ0 bits are as follows, according the setting of the JIT1 and JIT0 pins.

Table 4-10. Default Values of ADJON and ADJ2 to ADJ0 Bits According to Setting of JIT1 and JIT0 Bits

JIT1 Pin	JIT0 Pin	Default Value			
		ADJON bit	ADJ2 bit	ADJ1 bit	ADJ0 bit
0	0	0	0	0	0
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	1

In this program example, the JIT1 and JIT0 pins are set to 00 to fix the frequency modulation rate of SSCG output.

(5) Points on lock register (LOCKR) initialization

If the phase is locked, the LOCK flag is cleared to 0. If it is unlocked later because of a standby status, writing to the SSCGC register, or $\overline{\text{RESET}}$ pin input, the LOCK flag is set to 1. If the phase is unlocked by a cause other than these, however, the LOCK flag is not affected (LOCK = 0).

Figure 4-31. Setting of Lock Register (LOCKR)

Address: FFFFF824H

	7	6	5	4	3	2	1	0
Default value	0	0	0	0	0	0	0	1
Bit name	0	0	0	0	0	0	0	LOCK
Set value	0	0	0	0	0	0	0	–

Bit position	Bit name	Function
0	LOCK	This is a read-only flag that indicates the PLL lock wait state. This flag holds a value of 0 as long as a lockup state is maintained. 0: Indicates that the PLL is locked. 1: Indicates that the PLL is waiting to be locked (unlock state).

(6) Program example

```
[file:initial.s]
#######################################################
# Checks LOCK bit of lock register (LOCKR)
#######################################################
-- (0) Reads lock register (LOCKR) and checks stability of PLL
_UNLOCK:
     ld.b    LOCKR[r0], r6          -- Reads lock status from lock register (LOCKR).
     cmp     r0,        r6          -- Phase not locked yet?
     jnz     _UNLOCK                -- No

                                    -- Proceed to the next procedure as PLL lock status has been checked.

-- (i) Setting x7H to system wait control register (VSWC) (x: Setting the value set in I.)
     mov     0x37,      r6          -- Higher 4 bits and lower 4 bits of set value of VSWC are 7H (fx = 133 MHz)
     st.b    r6,        VSWC[r0]    -- Writes to the system wait control register.

-- (ii) Setting the frequency division value of the external bus to the bus mode control register (BMC).
     mov     0x01,      r6          -- Writes the division ratio of the bus clock (BUSCLK)
                                    -- for the external system bus (fCLK)
     st.b    r6,        BMC[r0]     -- to the bus mode control register.

-- (iii)  Setting the value set in I to the system wait control register (VSWC) again.
     mov     0x33,      r6          -- Writes back higher and lower bits of set value of VSWC (fx = 133 MHz).
     st.b    r6,        VSWC[r0]    -- Writes to the system wait control register.

-- (iv) Setting the internal system clock frequency division value to the clock control register (CKC).
####   Writing only in special sequence   ####
     mov     0xa0,      r6          -- <1> Disables NMI.
     ldsr    r6,        5           --      Sets NP bit of PSW to 1 (to disable NMI).
     mov     0x03,      r6          -- <2> Prepares data to be set to CKC, in general-purpose register.
                                    --      Internal system clock in PLL mode: fCLK = fx
     st.b    r6,        PRCMD[r0]   -- <3> Writes dummy data to the command register.
     st.b    r6,        CKC[r0]     -- <4> Sets the clock control register.
     nop                            -- <5> Inserts dummy NOP instruction (issues five instructions).
     nop                            -- <6> Inserts dummy NOP instruction
     nop                            -- <7> Inserts dummy NOP instruction
     nop                            -- <8> Inserts dummy NOP instruction
     nop                            -- <9> Inserts dummy NOP instruction
     mov     0x20,      r6          -- <10> Releases NMI disable (NMI enabled).
     ldsr    r6,        5           --      Clears NP bit of PSW to 0 again.
####   Writing only in special sequence   ####
```

```
-- (v) Setting main clock (fx) supply using the clock source select register (CKS).
####   Writing only in special sequence   ####
     mov     0xa0,       r6          -- <1> Disables NMI.
     ldsr    r6,         5           --      Sets NP bit of PSW to 1 (to disable NMI).
     mov     0x01,       r6          -- <2> Prepares data to be set to CSK in general-purpose register.
                                     --      SSCG output clock (Fx × 8) is supplied as the main clock (fx).
     st.b    r6,         PRCMD[r0]   -- <3> Writes dummy data to the command register.
     st.b    r6,         CKS[r0]     -- <4> Sets the clock source select register.
     nop                             -- <5> Inserts dummy NOP instruction (issues five instructions).
     nop                             -- <6> Inserts dummy NOP instruction.
     nop                             -- <7> Inserts dummy NOP instruction.
     nop                             -- <8> Inserts dummy NOP instruction.
     nop                             -- <9> Inserts dummy NOP instruction.
     mov     0x20,       r6          -- <10> Releases NMI disable (NMI enabled).
     ldsr    r6,         5           --       Clears NP bit of PSW to 0 again.
####   Writing only in special sequence   ####

# SSCG control register (SSCGC) at default value (4xH: Refer below for x.)
####   Writing only in special sequence   ####
--  SMDL1 = 0, SMDL1 = 1: modulation cycle of SSCG output 26 to 35 kHz
--  JIT1 pin   JIT0 pin  ADJON   ADJ2   ADJ1   ADJ0   Frequency modulation rate of SSCG output (Typ. value)
--     0          0        0       0      0      0          No modulation (fixed frequency)
--     0          1        1       0      0      1          -1.0%
--     1          0        1       0      1      1          -3.0%
--     1          1        1       1      0      1          -5.0%
####   Writing only in special sequence   ####

# USB clock control register (UCKC)
#    mov     0x80,       r6          -- Starts clock supply to USB
#    st.b    r6,         UCKC[r0]

# Oscillation stabilization time select register (OSTS) at default value (04H)
--  Oscillation stabilization time after software STOP mode is released: 21.84 ms (at fx = 12 MHz)

_UNLOCK2:
     ld.b    LOCKR[r0], r6           -- Reads the lock status of PLL from the lock register (LOCKR).
     cmp     r0,         r6          -- Phase not locked yet?
     jnz     _UNLOCK2                -- No

                                     -- Proceed to the next procedure as the PLL lock status has been checked.
```

4.2.7 Example of initializing SDRAM

This section explains the points to be noted when initializing SDRAM and shows a program example.

(1) Be sure to initialize SDRAM in the following sequence.

- Set the port a mode control register (PMCa) (a = AL, AH, DH, CS, CT, CM, or CD).
- Set chip area select control register m (CSCm) and determine the chip select signal that connects SDRAM (m = 0 or 1).
- Determine the type of the memory of the chip select space that connects SDRAM and enable the memory controller by using bus cycle type configuration register m (BCTm) (m = 0 or 1).
- Specify insertion of idle states by using the bus cycle control register (BCC).
- Set SDRAM refresh control register n (RFSn) (n = 1, 3, 4, or 6)
- Set SDRAM configuration register n (SCRn) (n = 1, 3, 4, or 6).

(2) SDRAM configuration register n (SCRn) (n = 1, 3, 4, or 6)

- An SRAM read/write cycle is not generated prior to executing a register write operation. Access the SDRAM area after setting the SCRn register and confirming that the WCFn bit is set to 1.
- Do not execute continuous instructions to write to the SCRn register. Be sure to insert another instruction between commands to write to the SCRn register.
- Write data to the register after reset. Do not change the set value.

(3) SDRAM refresh control register n (RFSn) (n = 1, 3, 4, or 6)

Write to the RFSn register after reset, and then do not change the set value. However, when the SDRAM refresh interval needs to be changed by changing the value of the CKC register (internal system clock (f_{CLK})), the set value of the RFSn register can be changed[Note]. Also, do not access an external memory area other than the one for this initialization routine until the initial setting of the RFSn register is completed. However, it is possible to access external memory areas whose initialization setting are completed.

Note If it is necessary to change the refresh interval of the SDRAM as a result of changing the internal system clock (f_{CLK}), follow this procedure.

<1> Mask all interrupts.
To disable maskable interrupts, set interrupt mask registers 0 to 5 (IMR0 to IMR5) (refer to **7.3.5 Interrupt mask registers 0 to 5 (IMR0 to IMR5)** in the **V850E/ME2 Hardware User's Manual**).
To disable non-maskable interrupts, set the NP bit of the PSW to 1 to disable multiple interrupts (refer to **3.2.2 (2) Program status word (PSW)** in the **V850E/ME2 Hardware User's Manual**).
<2> Clear the MEa bit of the BCTm register to 0 (m = 0 or 1, a = 0 to 7).
<3> Clear the RENn bit to 0 ((n = 1, 3, 4, or 6).
<4> Set the MEa bit of the BCTm register to 1 (m = 0 or 1, a = 0 to 7).
<5> Set a new value to the RCCn1, RCCn0, and RINn5 to RINn0 bits, and set the RENn bit to 1 (n = 1, 3, 4, or 6).
<6> Write the same value currently set to the SCRn register to the SCRn register (n = 1, 3, 4, or 6).
<7> Confirm that the WCFn bit of the SCRn register is set to 1, and access SDRAM (n = 1, 3, 4, or 6).

To change the refresh interval, set a value that enables refresh to be made in time even while the interval is being changed. If the refresh interval is correctly secured, the processing in <1> above may be skipped. The RFSn and BCTm registers are prohibited from being rewritten, but they can be rewritten when the refresh interval is re-set by changing the value of the CKC register.
An SDRAM register write cycle is generated as a result of rewriting the SCRn register (processing in <6> above), but the value of SDRAM before the RFSn and SCRn registers are re-set is held.

(4) Program example

```
[file:initial.s]
#########################################################
#     Specifies SDRAM configuration.
#########################################################

### Setting refresh interval of CS1 (SDRAM, 32 bits wide/32 MB) space ##
     mov     0x801e,     r6          -- Enables refresh (BUSCLK = 66 MHz).
                                     -- Refresh count clock = 32/BUSCLK
                                     -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
     st.h    r6,         RFS1[r0]    -- SDRAM refresh control register 1

### Setting refresh environment of CS1 (SDRAM, 32 bits wide/32 MB) space. ##
     mov     0x20a5,     r6          -- CAS Latency 2 Latency (TLATE) during read
                                     -- 2 waits (TBCW) during ACT-RD, PRE-ACT
                                     -- Address shift width       2 bits (On-page)
                                     --                          (External data bus width: 32 bits)
                                     -- Row address width         12 bits
                                     -- Address multiplex width   9 bits
     st.h    r6,         SCR1[r0]    -- Writes to SDRAM configuration register 1
_SCR1NOFIX:
     ld.h    SCR1[r0],   r6          -- Read SDRAM configuration register 1
     movea   0x0100,     r0, r7      -- Extracts WCF1 bit.
     and     r7,         r6
     cmp     r6,         r7          -- SCR1 setting in complete?
     jnz     _SCR1NOFIX              -- Yes

### Setting refresh interval of CS4 (SDRAM, 16 bits wide/16 MB) space ##
     mov     0x801e,     r6          -- Enables refresh (BUSCLK = 66 MHz)
                                     -- Refresh count clock = 32/BUSCLK
                                     -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
     st.h    r6,         RFS4[r0]    -- SDRAM refresh control register 4

### Setting refresh environment of CS4 (SDRAM, 16 bits wide/16 MB) space. ##
     mov     0x2095,     r6          -- CAS Latency 2 Latency (TLATE) during read
                                     -- 2 waits (TBCW) during ACT-RD, PRE-ACT
                                     -- Address shift width       1 bit (On-page)
                                     --                          (External data bus width: 16 bits)
                                     -- Row address width         12 bits
                                     -- Address multiplex width   9 bits
     st.h    r6,         SCR4[r0]    -- SDRAM configuration register 4
_SCR4NOFIX:
     ld.h    SCR4[r0],   r6          -- Read SDRAM configuration register 4.
     movea   0x0100,     r0, r7      -- Extracts WCF4 bit.
     and     r7,         r6
     cmp     r6,         r7          -- SCR4 setting in complete?
     jnz     _SCR4NOFIX              -- Yes
```

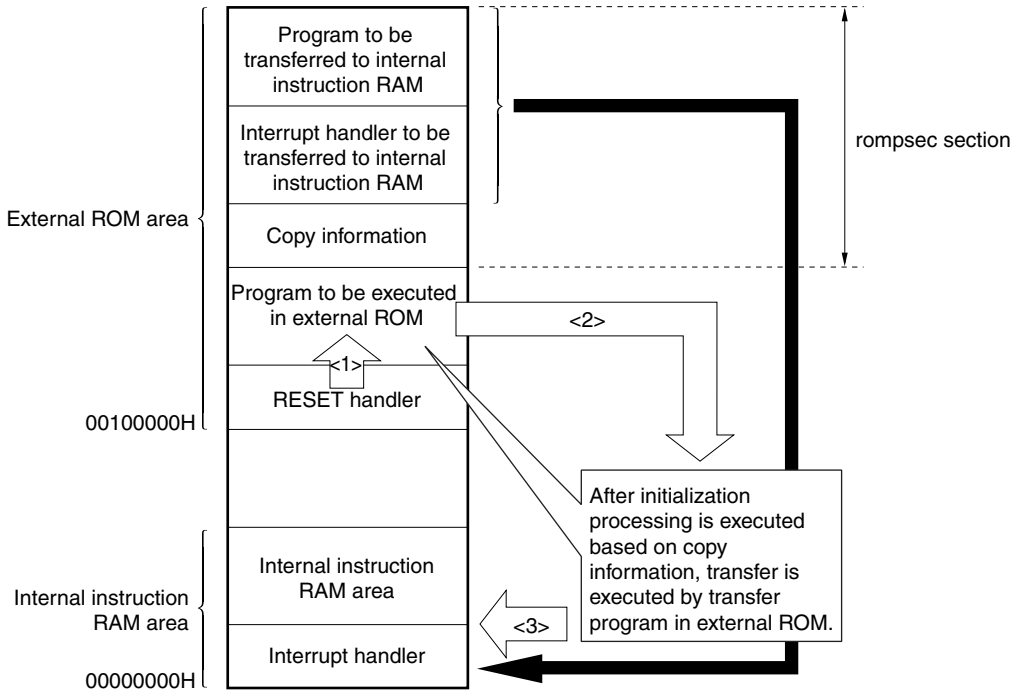
```
### Setting refresh interval of CS6 (SDRAM, 8 bits wide/16 MB) space ##
     mov     0x801e,     r6          -- Enables refresh (BUSCLK = 66 MHz)
                                     -- Refresh count clock = 32/BUSCLK
                                     -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
     st.h    r6,         RFS6[r0]    -- SDRAM refresh control register 6

### Setting refresh environment of CS6 (SDRAM, 8 bits wide/16 MB) space. ##
     mov     0x2086,     r6          -- CAS Latency 2 Latency (TLATE) during read
                                     -- 2 waits (TBCW) during ACT-RD, PRE-ACT2
                                     -- Address shift width        0 bit (On-page)
                                     --                           (External data bus width: 8 bits)
                                     -- Row address width          12 bits
                                     -- Address multiplex width    10 bits
     st.h    r6,         SCR6[r0]    -- SDRAM configuration register 6
_SCR6NOFIX:
     ld.h    SCR6[r0],   r6          -- Reads SDRAM configuration register 6
     movea   0x0100,     r0, r7      -- Extracts WCF6 bit.
     and     r7,         r6
     cmp     r6,         r7          -- SCR6 setting in complete?
     jnz     _SCR6NOFIX              -- Yes
```

4.2.8 Example of internal instruction RAM transfer

Transfer the program to be executed to the internal instruction RAM after CPU initialization processing is completed.

The basic concept of internal instruction RAM transfer is illustrated below.

Figure 4-32. Basic Concept of Internal Instruction RAM Transfer



Remark <1> to <3> in this figure indicate the flow of processing.

Note the following points when transferring a program to the internal instruction RAM.

- Handling of non-maskable interrupts (NMI) and exception interrupts before and after transfer
- Segment directive description of RESET handler
- Separation of program to be transferred to internal instruction RAM
- Option of ROMization processor

(1) Handling of non-maskable interrupts (NMI) and exception interrupts before and after transfer

With the V850E/ME2, it is assumed that a program is transferred to the internal instruction RAM after initialization. While program transfer to the internal instruction RAM is under execution, generation of any interrupt, including an NMI, or exception is prohibited. Therefore, the following points must be noted concerning the mechanism of the, NMI and exceptions that occur during program execution.

- After reset is released, NMI input is masked by hardware. In the internal instruction RAM transfer sequence, an NMI is unmasked as soon as the IRAMM0 bit of the internal instruction RAM mode register (IRAMM) has been cleared to 0.
- If it is necessary to confirm that an NMI has been input after release of reset and before the internal instruction RAM is set in the read mode, read the NMIRS bit of the NMI reset status register (NRS). If this bit is set to 1, it indicates that the valid edge of the NMI has been input. As necessary, execute the interrupt processing routine of the NMI. The NRS register is used to check NMI input after release of reset and before the internal instruction RAM is set in the read mode, and is not cleared to 0 after release of reset.

(2) Segment directive description of RESET handler

Generally, the RESET handler address of the CPU in the V850 Series is 00000000H. A non-maskable interrupt (NMI), exception, and maskable handler areas exist at addresses higher than 00000010H at almost 16-byte intervals, except in some products. These areas are the internal flash memory, internal mask ROM, and external ROM areas. Because these areas are ROM areas, they are non-volatile memory areas.

Even if the internal instruction RAM is employed, the NMI, exception, and maskable interrupt handler areas exist at addresses higher than 00000010H because compatibility with the other CPUs in the V850 Series must be maintained. These areas are RAM areas and therefore volatile memory areas. If the RESET handler address is 00000000H in the internal instruction RAM space, which is a volatile memory space, the instructions are undefined and the program is not executed correctly. Because the RESET handler must be located in an external non-volatile memory (such as EPROM or flash memory), the RESET handler address of the V850E/ME2 is 00100000H in the external memory space.

Usually, if the following assembly source description is made, the program (instructions) is located at address 00000000H, which is the RESET handler address, along with an internal RESET segment directive when the CA850 compiler package resolves a link.

```
.section "RESET", text
jr  _ _start
```

As mentioned above, however, the RESET handler address of the V850E/ME2 is 00100000H. Therefore, explicitly describe the RESET segment directive to which the RESET handler address of the V850E/ME2 is appended in the directive file.

```
RESET   : !LOAD ?RX V0x00100000 {
    RESET          = $PROGBITS ?AX RESET;
};
```

This correctly locates the program (instructions) to the RESET handler address.

Caution Unless this RESET segment directive is described in the address, the program (instructions) is located at an undefined address immediately after that of the program code linked before the above assembly source code.

(3) Separation of program to be transferred to internal instruction RAM

As shown in Figure 4-32, the program from CPU initialization to transfer is executed in the external ROM. The program to be transferred to the internal instruction RAM and executed is also stored in the external memory in ROMized format. The program from CPU initialization to transfer is executed in the external ROM but does not have to be packed in the rompsec section. The program to be transferred to the internal instruction RAM and executed is not executed in the external ROM but must be packed in the rompsec section.

Program	Execution in External ROM	rompsec Packing
Program from CPU initialization to transfer	Executed	Not necessary
Program to be transferred to internal instruction RAM	Not executed	Necessary

Both of these programs are sections having text attribute. All sections with text attribute are subject to the rompsec section. Therefore, these two parts are separated at link time. The directive file and source file must be described. If nothing is specified as the entry of the rompsec section, it is placed at the end of the .text section and located on the volatile memory. Therefore, a new entry label of the rompsec section is prepared and located at the end of the program to be executed in the external ROM, which is a non-volatile memory, by a directive.

(a) Description of source file

For a program that is executed only in the external memory (from CPU initialization to program transfer), add a source file description that specifies the section name.

[In C]

```
#pragama text "section name"
  <programs to be executed only in external memory>
#pragama text

Or,

#pragama text "section name" function name
  <function to be executed only in external memory>
```

[In assembly language]

```
.section “section name”, text
  <program to be executed only in external memory>
```

Nothing is described for the program to be transferred to the internal instruction RAM for execution.

Add a new entry label of the rompsec section and an assembly language source file in which the new section name of that entry label is described.

```
.section “new section name”, text
.align     4
.globl     new entry label name, 4
new entry label name:
```

(b) Directive description

Add a directive description of the section name specification for all the programs to be executed only in the external memory (from CPU initialization to program transfer).

[In C]

```
[c source file]
#pragama text "section name"
  <programs to be executed only in external memory>
#pragama text

or,

#pragama text "section name" function name
  <function to be executed only in external memory>

Add the following.

[directive file]
any segment name: !LOAD ?RX v address to be assigned {
  section name.text = $PROGBITS ?AX section name.text;
};
```

Here is an example of locating the romfunc section immediately after the preceding segment.

```
[c source file]
#pragama text "romfunc" test_func
void test_func (void) {
<program to be executed only in external memory>
}

[directive file]
ROMFUNC: !LOAD ?RX {
  romfunc.text = $PROGBITS ?AX romfunc.text;
};
```

[In assembly language]

```
[assembler source file]
  .section “section name”, text
  <program to be executed only in external memory>

Add the following.

[directive file]
any segment name: !LOAD ?RX v address to be allocated {
  section name = $PROGBITS ?AX section name;
};
```

Here is an example of locating the .crtE section at address 00101000H.

```
[assembler source file]
  .section “.crtE”, text
  <program to be executed only in external memory>

[directive file]
CRTE: !LOAD ?RX V0x00101000 {
  .crtE = $PROGBITS ?AX .crtE;
};
```

No special description needs to be made for the program to be transferred to the internal instruction RAM and executed.

Only the address to be executed in the internal instruction RAM must be allocated by address specification of the following TEXT segment.
An example of locating the .text section at address 00001000H is shown below.

```
TEXT         : !LOAD ?RX V0x00001000 {
 .text        = $PROGBITS ?AX .text;
};
```

Add a directive description to the new entry label of the rompsec section. The position at which the description is to be added in the directive file is at the end of the directive descriptions of the section name specification for all the programs to be executed only in the external memory (from CPU initialization to program transfer).

```
[assembler source file]
.section "new section name", text
.align     4
.globl     new entry label name, 4
new entry label name:

Add the following.

[directive file]
any segment name: !LOAD ?RX {
  new entry label name = $PROGBITS ?AX new entry label name;
};
```

Here is an example of locating the .s_romp section at the end of the directive descriptions (after the CRTE segment in this example) of the section name specification for all the programs to be executed only in the external memory (from CPU initialization to program transfer).

```
[assembler source file]
.section ".s_romp", text
.algin     4
.globl     __S_rompack, 4
__Srompack;

[directive file]
SROMP: !LOAD ?RX {
.s_romp = $PROGBITS ?AX .s_romp;/
};
```

(4) Option of ROMization processor

By describing the directive file and source file, the programs that are not transferred to the internal instruction RAM but are executed only in the external ROM, and the programs that are not executed in the external ROM but are transferred to the internal instruction RAM for execution are separated. All the programs not executed in the external ROM but transferred to the internal instruction RAM are reserved section name.text with a text-attribute section. This section and the section of the interrupt handler to be used are packed in the rompsec section. Indicate .text and the name of the interrupt handler to be used by using “read-only section option to be packed” –t.

```
-t .text –t interrupt handler name 1 –t interrupt handler name 2 …
```

To pack the program that is not executed in the external ROM but transferred to the internal instruction RAM for execution as a rompsec section, define a new entry label using “entry label option” –b.

```
-b entry label
```

A warning message indicating that the internal ROM area has been exceeded is output at link time, but ignore this message.

(5) Option of hex converter

A program to be executed only in the external ROM (from CPU initialization to program transfer) must be assigned offset addresses to the external ROM. Specify 100000H using “output address offset option” –d.

```
-d 0x100000
```

(6) Program example

Caution For details of the description in the program (file:crtE.s) "--(0) Transfer program to internal instruction RAM", refer to 4.2.10 (2) (d) Transferring static program to bank 0 of internal instruction RAM.

```
[file:crtE.s]
(Omission)
#  After clear bss section of normal crtE.s file

#########################################################
#    Transfer program code to internal instruction RAM
#########################################################
.extern _ _S_rompack
.extern _ _S_appli1rom
.extern _ _ircopy
.globl _iramboot
_iramboot:
--  (0) Transfer program to internal instruction RAM
.option nowarning
    movhi   hi1(_ _S_rompack), tp, r1     --  Offset of tp
    movea   lo(_ _S_rompack), r1, r6      --  Static program packing section start address
.option warning
    mov     -0x1,    r7                   --  All packing sections to be transferred.
    jarl    _ _ircopy,r31                 --  Transfer ROMized dynamic program to Bank 0

--  (i) Set internal instruction RAM mode register (IRAMM) in read mode (IRAMM0, IRAMM1 = 0).
    mov     0x00,    r6                   --  Bank 0 0000000 to 00FFFFH (64 KB) Read mode
                                          --  Bank 1 0010000 to 01FFFFH (64 KB) Read mode
    st.b    r6,      IRAMM[r0]            --  Internal instruction RAM mode register

--  (ii)  Check change of read mode of internal instruction RAM mode register (RAMM).
_iram_rd_chk:
    ld.b    IRAMM[r0], r6                 --  Read internal instruction RAM mode register
    cmp     r0,      r6                   --  Bank 0, bank 1 in read mode?
    bnz     _iram_rd_chk                  --  No

--  (iii) To internal instruction RAM by branch instruction
.option nowarning
    mov     #_main, r1
    jmp     [r1]
.option warning
_ _exit:
    halt                                  --  end of program
_ _startend:
    nop
    nop

#                                                                                #
#----------------------------- end of start up module -----------------------------#
#                                                                                #
```

```
[file:ircopy.s]
#*****************************************************************************
#
# [Function name]    _ircopy
#
# [Outline]          Function to transfer program from external ROM to internal instruction RAM
#
# [Explanation]      The _ircopy function transfers the program of the section number passed by the second
#                    argument r7 to the internal instruction RAM area, based on information on the rompsec section
#                    that exists at addresses following the one that is passed by the first argument r6.
#
# [Argument]         r6: First address of rompsec section
#                    r7: Section ID number to be copied to the internal instruction RAM
#                     * A section ID number starts from '1'.
#                     * -1 copies all sections.
#
# [Return value]     r10: Copy successful (0)/Copy failed (-1)
#
# [Stack]            Not used
# [register]         r8: Unit of copy information of each section at which copy is to start
#                    r9: Unit of copy information of each section at which copy is to end
#                    r10: Transfer size/temporary variable
#                    r11: Transfer destination address of section specified by section ID number/temporary variable
#                    r12: Transfer source address of section specified by section ID number/temporary variable
#                    r13: First address of area where copy information of each section is stored
#                    r14: Transfer destination start address of section specified by section ID number
#                    r15: Transfer source start address of section specified by section ID number
#                    r16: Transfer size of section specified by temporary variable/ID number
#                    r17: Temporary variable
#                    r18: Temporary variable
#                    r19: Temporary variable
#                    r20: Temporary variable
#                    r21: Temporary variable
#*****************************************************************************
#               Contents of rompsec section in which n sections are packed
#
# Offset address from beginning
# of rompsec section
#    ---          + 0-> -+-------------------------------------------+- rompsec section
#     |                  |  Magic number                             |  First address
#     |                  | (0x00504d2)                               |
# Common           + 4-> -+-------------------------------------------|
# information             | Number of sections                        |
#     |                   | (in bytes)                                |
#    ---           + 8-> -+===========================================+- + 0
#     |                   | Transfer destination address of first section |
#     |                   | (absolute address)                        |
# At each section  +12-> -+-------------------------------------------+- + 4
#     |                   | Transfer destination address of first section |
#     |                   | (absolute address)                        |
# Copy of 16 bytes +16-> -+-------------------------------------------+- + 8
```

```
#     |                  | Size of first section                                  |
#     |                  | (in bytes)                                             |
# Information       +20-> -+-------------------------------------------+- +12
#     |                  | Transfer source address of first section               |
#     |                  | (offset address from rompsec section start address)    |
#   ---           +24-> -+===========================================+- + 0
#     |                  | Transfer destination address of second section         |
#     |                  | (absolute value)                                       |
# At each section   +28-> -+-------------------------------------------+- + 4
#     |                  | Transfer destination address of second section         |
#     |                  | (absolute value)                                       |
# Copy of 16 bytes  +32-> -+-------------------------------------------+- + 8
#     |                  | Size of second section                                 |
#     |                  | (in bytes)                                             |
# Information       +36-> -+-------------------------------------------+- +12
#     |                  | Transfer source address of first section               |
#     |                  | (offset address from rompsec section start address)    |
#   ---           +40-> -+===========================================+-
#                        |                        :                  |
#                        |                        :                  |
#       +(n-1)*16+8-> -+===========================================+-
#                        | Transfer source address of nth section                 |
#                        | (absolute address)                                     |
#                       -+-------------------------------------------+-
#                        | Transfer source address of nth section                 |
#                        | (absolute address)                                     |
#                       -+-------------------------------------------+- - 8
#                        | Size of nth section                                    |
#                        | (in bytes)                                             |
#                       -+-------------------------------------------+- - 4
#                        | Transfer source address of first section               |
#                        | (offset address from rompsec section start address)    |
#           +n*16+8-> -+===========================================+- + 0
#                        | Packing data of first section                          |
#                       -+-------------------------------------------+-
#                        |                        :                  |
#                       -+-------------------------------------------+-
#                        | Packing data of second section                         |
#                       -+-------------------------------------------+-
#                        |                        :                  |
##############################################################################
    .section ".ircopy",text
    .align   16 --    .align 4
    .globl   _ _ircopy
_ _ircopy:

-- Check validity of rompsec section common information.
    ld.w   0x0[r6], r15       -- Obtain data of first word of rompsec section.
.option nowarning
    movhi  0x50,   r0, r1
    movea  0x4d52, r1, r1
    cmp    r1,     r15        -- Comparison with Magic Number (0x504d2)
```

```
.option warning
    bnz    _ircopy_ng          -- Illegal termination because Magic Number does not match
    ld.w   0x4[r6],r12         -- Obtain data of second word of rompsec section.
    cmp    0x1,    r12         -- Number of packing sections in rompsec section?
    blt    _section_id_chk     -- No section to be copied

-- Check validity of second argument r7 (section ID number to be copied).
    cmp    -0x1,   r7          -- Section ID number to be copied –1?
    bz     _all_copy           -- Copy all sections.
_section_id_chk:
    cmp    0x1,    r7          -- Specified section ID number to be copied < 1?
    blt    _ircopy_ng          -- Illegal termination because specified section ID number to be copied < 1
    cmp    r7,     r12         -- Specified section ID number to be copied ≤ number of packing sections?
    bge    _section_copy       -- Illegal termination because section ID exceeding the number of packing
    br     _ircopy_ng          -- sections cannot be copied.
    nop
    nop

-- Initialization for copying section specified by ID number
_section_copy:
    addi   -0x1,   r7, r8      -- To r8, assuming that copy is to be started from <ID - 1> unit of copy
                               -- information of each section
    mov    r7,     r9          -- To r9, assuming that copy is to be started from <ID> unit of copy
                               -- information of each section
                               -- Therefore, only section specified by ID number is copied.
    br     _ircopy_main        -- To copy processing

-- Initialization for copying all sections
_all_copy:
    mov    r0,     r8          -- To r8, assuming that copy is to be started from <0> unit of copy
                               -- information of each section
    mov    r12,    r9          -- To r9, assuming that copy is to be started in <number of packing
                               -- sections> unit of copy information of each section

_ircopy_main:
-- Calculate start address where packing section is stored.
    addi   0x8,    r6,  r11    -- Obtain address of third word of rompsec section.
                               -- This is address in unit 0 of copy information of each section [A].

    shl    0x4,    r12         -- Number of packing sections × 16 (1 unit of copy information of each
                               -- section)
                               -- This calculates the total number of bytes of copy information of each section [B].

    add    r11,    r12         -- Packing data first address of A + B = rompsec section
    cmp    r9,     r8          -- Is unit of copy information of each section in which copy is to be ended reached?
    mov    r12,    r10         -- First address of packing data in rompsec section to r10
    bge    _ircopy_ok          -- Copy is terminated correctly because the above unit is reached.

-- Calculate the end address where the packing section is stored.
    ld.w   -0x4[r10],r17       -- Obtain, that is, transfer source start address in final unit of copy information
                               -- of each section offset value from the beginning of rompsec section.
    add    r6,     r17         -- Calculate the absolute address of the transfer source by adding the
```

```
                                    -- above offset value to the first address of rompsec section.
    ld.w   -0x8[r10],r18            -- Obtain the copy transfer size of the final unit of copy information of each section.
    add    r18,     r17             -- Calculate the absolute end address of the transfer source by adding the
                                    -- transfer size to the above absolute address.

    mov    r17,     r21             -- Save the end address where packing section is stored.
    mov    r11,     r20             -- Save the address of unit 0 of copy information of each section.

_section_loop:
    mov    r8,      r19             -- Section ID number to be copied to r19
    shl    0x4,     r19             -- Section ID number × 16 (1 unit of copy information of each section)
                                    -- Calculate the total number of bytes up to the beginning of the copy
                                    -- information of the specified section ID number.
    mov    r20,     r13             -- Obtain the address of unit 0 of copy information of each section.
    add    r19,     r13             -- Calculate the absolute address where the copy information of each
                                    -- section of section ID number n is stored, by adding n units of copy
                                    -- information of each section (n*16 bytes) to the above address.
    ld.w   0x0[r13],r14            -- Obtain the transfer destination start address of section ID number n.
    mov    r14,     r11             -- Obtain the absolute address of the transfer destination of section ID
                                    -- number n by adding the transfer size to the above absolute address.

    ld.w   0xc[r13],r15            -- Obtain the transfer source start address of section ID number n.
                                    -- Therefore, obtain an offset value from the beginning of the rompsec section.
    add    r6,      r15             -- Calculate the absolute address of the transfer source by adding the first
                                    -- address of the rompsec section to the above address.

-- Check the validity of the first address of the rompsec section
    ld.w   0x8[r13],r16            -- Obtain the copy transfer size of final unit of copy information of each section.
    mov    r16,     r10             -- Temporarily save the above to r10.
    shr    2,       r10             -- [WORD] Divide the above by 4 (WORD size) and temporarily save it to r10.
    tst1   0,       0x8[r13]        -- [WORD] Is transfer size odd?
    bnz    _align4adjust           -- [WORD] If yes, start 4-byte boundary correction.
    tst1   1,       0x8[r13]        -- [WORD] If no, is there a fraction of 3 or less in the transfer size?
    bz     _align4ok               -- [WORD] If no, correction is not necessary.
_align4adjust:                     -- [WORD] If yes, add 1 to the transfer size
    add    1,       r10             -- [WORD] to avoid occurrence of Align Error
_align4ok:                         -- [WORD] Transfer size remains as is because of 4-byte boundary

    add    r16,     r14             -- Calculate the transfer destination end address of the specified section by adding
                                    -- the transfer size to the transfer destination start address of the specified section.
    cmp    r6,      r14             -- The transfer destination end address of the specified address overlaps
                                    -- the rompsec section start address.
    mov    r15,     r12             -- Transfer start address of the specified section to r12
    bnh    _size_chk               -- If no, check the size of data to be copied.

-- Check the validity of the rompsec section end address
    mov    r21,     r18             -- Read the rompsec section end address that has been saved.
    cmp    r18,     r11             -- rompsec section end address overlap transfer destination start address?
    bc     _ircopy_ng              -- If yes, illegal termination because there is a danger of no copy.
_size_chk:
    cmp    r0,      r10             -- Size of section to be copied ≤ 0?
    ble    _next_section           -- If yes, to next section because copy is not required
```

```
    add    -0x1,   r10          -- Transfer size of section specified by section ID number

_data_loop:
    mov    r11,    r14
    addi   0x4,    r14,  r11    -- Transfer destination address of section specified by section ID number +4

    mov    r12,    r7
    addi   0x4,    r7,   r12    -- Transfer source address of section specified by section ID number +4

    ld.w   0x0[r7],r19          -- Obtain packing data of section specified by section ID number
    st.w   r19,    0x0[r14]     -- Stored in expansion destination

    mov    r10,    r13
    addi   -0x1,   r13,  r10    -- Decrement counter.

    cmp    r0,     r13          -- Counter > 0?
    bgt    _data_loop           -- If yes, to next data because data remains.
_next_section:
    add    0x1,    r8           -- If no, to next section ID number (unit)
    cmp    r9,     r8           -- Number of packing sections < next section ID number (unit)?
    blt    _section_loop        -- If no, to next section because data remains.

_ircopy_ok:
    mov    r0,     r10          -- If yes, copy is terminated correctly (successfully).
    br     _ircopy_exit
_ircopy_ng:
    mov    -0x1,   r10          -- Copy failed.
_ircopy_exit:
    jmp    [lp]

#### end of ircopy.s ####
```

4.2.9 Example of program to be transferred to internal instruction RAM

This is a program that repeatedly lights the numbers 1 to 8 at about 1-second intervals, using Dot LEDs 1 to 4.

```
    .text
    .globl _main
_main:
    mov    0x01,   r11
    mov    0x8,    r12
_ledloop:
    mov    r11,    r6
    jarl   _ledout,lp

    movhi  0x00cb, r0, r6
    movea  0xc6cc, r6, r6
    jarl   _softwait,lp

    add    0x1,    r11
    add    -1,     r12
    cmp    r0,     r12
    bnz    _main
    jr     _ledtest

    .globl _ledout
_ledout:
    mov    r6,     r10
    shr    0x02,   r10
    and    0x3f,   r10
    st.b   r10,    P5

    mov    r6,     r10
    shl    0x04,   r10
    and    0x30,   r10
    st.b   r10,    P2
    jmp    [lp]

    .globl _softwait
_softwait:
    add    -1,     r6
    cmp    r0,     r6
    bnz    _softwait
    jmp    [lp]
```

4.2.10 Example of program dynamically executed in internal instruction RAM

(1) Execution outline of dynamic program sample

With the V850E/ME2, the reset handler and internal instruction RAM are allocated starting from the following addresses.

RESET: Address 00100000H
Internal instruction RAM: Address 00000000H

Because the internal instruction RAM can be read and written, the program that is to be executed in the internal instruction RAM can be dynamically rewritten.
Dynamically executed programs 1 to 3 are explained here. The operations of dynamic programs 1 to 3 are almost the same.
In the main function, the following is output by UARTB1.

```
Running application No.n (n = 1-3)
```

An eternal loop is created in the main function. An 8-digit number is continuously output as follows.

```
<pseudo 7segLED>=00000000
```

Although the TB-V850E/ME2 does not have a 7-segment LED, the example program uses the eight digits of the numbers output by UARTB1 as a dummy eight-digit 7-segment LED.
Of the eight digits, only two digits repeatedly display numbers 0 to 9 in rotation, by using an idle counter.

```
[file:appli1.c]
  printf("\r\nRunning application No.1\r\n");/*Displays ID number of dynamic program*/
  for( ;; ){
       wait();
       if((place & DIGIT1) == GO)bcd01++;     /* Rotation of specified digit continues? */
       if(bcd01 > 9)bcd01=0;                  /* Numbers of rotation reel loop from 0 to 9. */
       if((place & DIGIT2) == GO)bcd02++;     /* Rotation of specified digit continues? */
       if(bcd02 > 9)bcd02=0;                  /* Numbers of rotation reel loop from 0 to 9 */
       /* Two of eight digits rotate */
       printf("\r<pseudo 7segLED>=000000%d%d",bcd02,bcd01);
  }
```

When the INT switch is pressed, the INTP11 interrupt is acknowledged, and the following processing is executed in the interrupt processing.

```
[file:appli1.c]
void intsw_for_app1(void){
  if(flag ^= ALTERNATIVE){   /* Flag is alternately set to 1 and cleared to 0 each time interrupt occurs. */
        /* First digit of rotation reel stops, and second digit rotates. */
        seg7led(&place, DIGIT1, STOP); /* Display of first digit of rotation reel stops. */
        seg7led(&place, DIGIT2, GO);   /* Display operation of second digit of rotation reel. */
        dotled(DOTLED1, ALLOFF);       /* All dot LEDs are off. */
        dotled(DOTLED1, ON);           /* Dot LED1 lights. */
  }else{
        /* Second digit of rotation reel stops, and first digit rotates */
        seg7led(&place, DIGIT2, STOP); /* Display of second digit of rotation reel stops. */
        seg7led(&place, DIGIT1, GO);   /* Display operation of first digit of rotation */
        dotled(DOTLED2, ALLOFF);       /* All dot LEDs are off. */
        dotled(DOTLED2, ON);           /* Dot LED2 lights. */
  }
}
```

- Of the two digits that rotate and display numbers, number rotation of the left digit is enabled, and that of the right digit is disabled.

```
Dot LEDn lights, Dot LEDn + 1 is extinguished.
```

When the INT switch is pressed again, the INTP11 interrupt is acknowledged and the following processing is executed in the interrupt processing.

- Of the two digits that rotate and display numbers, number rotation of the left digit is disabled, and that of the right digit is enabled.

```
Dot LEDn is extinguished, Dot LEDn + 1 lights.
```

Each time the INT switch is pressed, the above operation is alternately repeated.

The only difference in operation between dynamic programs 1 to 3 is the number position of the two rotating digits and the dot LEDs that alternately light. The differences are shown below.

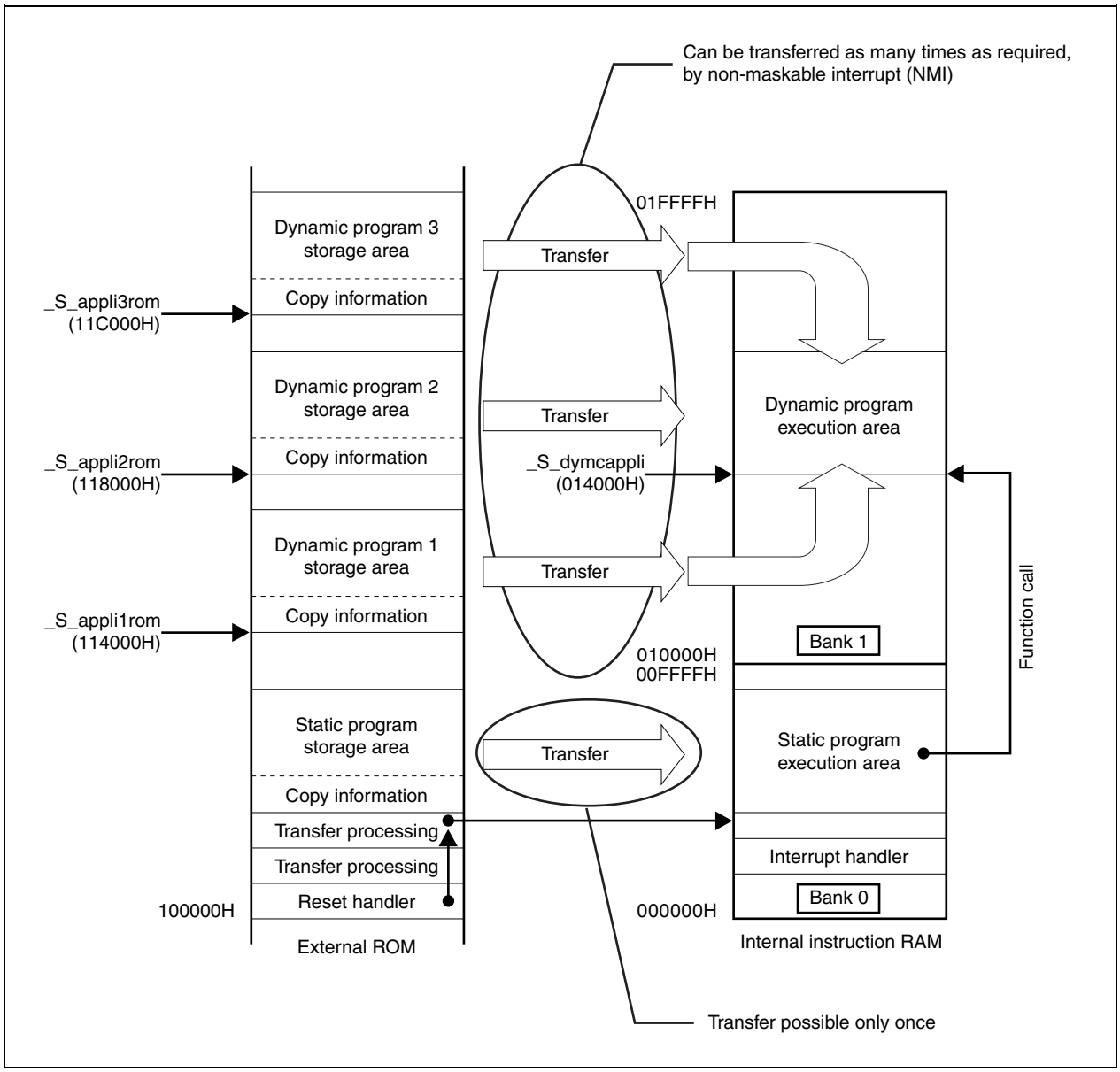
Conditions		Dynamic Program 1		Dynamic Program 2		Dynamic Program 3	
Number position of two rotating digits		000000??		000??000		??000000	
When INT SW is pressed odd number of times	Digit to stop	000000?x		000?x000		?x000000	
	Position of dot LED that lights	LED1		LED4		LED7	
		LED8○	○LED4	LED8○	●LED4	LED8○	○LED4
		LED7○	○LED3	LED7○	○LED3	LED7●	○LED3
		LED6○	○LED2	LED6○	○LED2	LED6○	○LED2
		LED5○	●LED1	LED5○	○LED1	LED5○	○LED1
When INT SW is pressed even number of times	Digit to stop	000000x?		000x?000		x?000000	
	Position of dot LED that lights	LED2		LED5		LED8	
		LED8○	○LED4	LED8○	○LED4	LED8●	○LED4
		LED7○	○LED3	LED7○	○LED3	LED7○	○LED3
		LED6○	●LED2	LED6○	○LED2	LED6○	○LED2
		LED5○	○LED1	LED5●	○LED1	LED5○	○LED1

Remark ??: Position of two digits at which numbers rotate
x: Position at which number rotation is stopped by INT SW

After the static program is transferred from the external ROM to bank 0 of the internal instruction RAM by reset handler → initialization processing → transfer processing, execution jumps to the beginning of the static program. The static program calls the dynamic program _S_dymcappli() function. The dynamic program can be dynamically changed by a non-maskable interrupt (NMI).

The outline of executing the dynamic program sample is shown below.

Figure 4-33. Outline of Execution of Dynamic Program Sample



(2) Image when static program is linked

The static program and dynamic program are compiled and linked completely separately.

(a) Dynamic program link handled in static program

Because dynamic program storage areas 1 to 3 and the execution area of the dynamic program are all undefined, they are linked by vacancy (code entity of 0 bytes with first address only).

The static program externally references the label of the external ROM space (fixed address) where the dynamic program is stored. At this time, the external reference in C is made as an external function name, not as an external variable name, because of the text attribute section.

```
[file:progmain.c]
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);
```

```
[file:secentry.s]
# External EPROM area 1 where program 1 that is to be dynamically executed is stored
        .section ".appli1rom",text
        .align   4
        .globl   _ _S_appli1rom, 4
_ _S_appli1rom:

# External EPROM area 2 where program 2 that is to be dynamically executed is stored
        .section ".appli2rom",text
        .align   4
        .globl   _ _S_appli2rom, 4
_ _S_appli2rom:

# External EPROM area 3 where program 3 that is to be dynamically executed is stored
        .section ".appli3rom",text
        .align   4
        .globl   _ _S_appli3rom, 4
_ _S_appli3rom:
```

```
[file:progmain.dir]
# Vacant segment area that stores program 1 to be dynamically executed
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
APPLI1ROM   : !LOAD ?RX V0x00114000 {
       .appli1rom       = $PROGBITS       ?AX A0x4 .appli1rom;
};

# Vacant segment area that stores program 2 to be dynamically executed
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
APPLI2ROM   : !LOAD ?RX V0x00118000 {
       .appli2rom       = $PROGBITS       ?AX A0x4 .appli2rom;
};

# Vacant segment area that stores program 3 to be dynamically executed
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
APPLI3ROM   : !LOAD ?RX V0x0011c000 {
       .appli3rom       = $PROGBITS       ?AX A0x4 .appli3rom;
};
```

For the concept and method of transferring a ROMized program to the internal instruction RAM, refer to **4.2.8 Example of internal instruction RAM transfer**.

(b) ROMized static execution program link in static program

The following sections that are statically executed in the internal instruction RAM must be ROMized.

- .const section
- Non-maskable interrupt (NMI) section
- .text section

<1> The .const section is generated when the standard I/O library is linked.

```
[file:progmain.dir]
# CONST segment area of program executed statically
# Internal instruction RAM area bank 0 0x00001000 to 0x0000ffff (60 KB)
CONST   : !LOAD ?R V0x0000f000 {
       .const           = $PROGBITS    ?A .const;
};
```

<2> The non-maskable interrupt (NMI) section is generated by describing an interrupt/exception handler with an NMI interrupt processing function, as an extension of C description.

```
[file:progmain.c]
#pragma interrupt NMI int_nmi
_ _interrupt void int_nmi(void){
(Omission)
}
```

```
[NMI interrupt directive internally appended by C compiler]
NMI    : !LOAD ?RX V0x00000010 {
   NMI     = $PROGBITS ?AX NMI;
};
```

<3> All program codes for which a section name is specified in a source file other than crE.s, initial.s, ledtest.s, and ircopy.s and for which a section is not defined in the directive file are generated as .text sections.

```
[file:progmain.dir]
# Program segment area that operates statically
# Internal instruction RAM area bank 0 0x00001000 to 0x0000ffff (60 KB)
MAINTEXT     : !LOAD ?RWX V0x00001000{
       .text            = $PROGBITS       ?AX .text;
};
```

Specify with –t option to pack these sections using the ROMization utility romp850 supplied with the C compiler package.

```
-t NMI -t .text -t .const
```

Select [Tool] → [ROM Processor Options …] from the main window of PM+ to display the dialog box to set the ROMization processor options, and input a section name (”NMI”, ”.text”, or ”.const”) on the [Section] tab and click the <Add [-t] (T)> button each time to add the section. The section names are displayed in the [Section List] box in the specified order.

(c) Method of creating ROMized object

When creating a ROMized object, the object must be linked to the end of the section located in the external EPROM.

First, change the entry name of the rompsec section to a new name, __Srompack, and describe it in the assemble source file. Specify the section name .s_romp for that entry name.

The entry name must indicate the first address (aligned under a 4-byte alignment condition) that exceeds the end of a .text section that is not ROMized.

```
[file:rompack.s]
       .file "rompack.s"
       .section ".s_romp",text
       .align    4
       .globl    _ _S_rompack, 4
_ _S_rompack:
```

Specify the location of the rompsec section whose name has been changed to __Srompack as the .s_romp section, by using the directive file.

Describe this rompsec section at the end of the segment directive located in the external EPROM.

The default alignment condition of the segment directive is 8 bytes.

```
[progmain.dir]
#  rompsec section segment area
#  External EPROM (16-bit width) area  0x100000 to 0x11ffff (128 KB)
SROMP         : !LOAD ?RX  {
    .s_romp              = $PROGBITS     ?AX .s_romp;
};
```

Specify with the –b option to place the __S_romp label at a ROMization packing start address using the ROMization utility romp850 supplied with the C compiler package.

Select [Tool] → [ROM Processor Options …] from the main window of PM+ to display the dialog box to set the ROMization processor options, and input “__S_rompack” to the text box of [Entry Label [-b]] on the [Option] tab.

The default area allocation code for the rompsec section (file name: rompcrt.o) is linked and the section of the label __S_romp is generated 0 bytes after the .text section during ROMization processing, but ignore these.

(d) Transferring static program to bank 0 of internal instruction RAM

The _ircopy function is used for transfer.

For details of this function, refer to **4.2.8 Example of internal instruction RAM transfer**.

Care must be exercised when passing the first argument, which is the first address of a ROMized section when the _ircopy function is called.

The reset handler address of the V850E/ME2 is 00100000H. In the sample, a non-maskable interrupt (NMI) is used. The NMI interrupt handler address is 00000010H. Therefore, the value of text pointer tp is linked and resolved at the highest address of the .text section, address 00000010H of the NMI interrupt handler.

The label (address) on the program code is calculated by subtracting text pointer tp = 00000010H.

To access the label (address) on the program code, note the tp pointer, and always add an offset address of 00000010H.

If entry name__S_rompack of the rompsec section is directly specified as an argument in this case, an address 10 addresses before the address where the tp offset was not added is passed to the _ircopy function.

```
mov    _ _S_rompack, r6      -- Static program packing section start address
mov    -0x1,         r7      -- All packing sections to be transferred
jarl   _ _ircopy,    r31     -- Transfer ROMized section to internal instruction RAM bank 0
```

With this information, the ROMized program cannot be correctly transferred to the internal instruction RAM. Add a tp offset value and address 00000010H, and pass the entry name of the rompsec section to the _ircopy function, as shown below.

```
[file:crtE.s]
.option nowarning
   movhi  hi1(_ _S_rompack),  tp,  r1   -- Static program packing section start address
   movea  lo(_ _S_rompack),   r1,  r6   -- to which offset of tp is to be added
.option warning
   mov    -0x1,       r7               -- All packing sections to be transferred
   jarl   _ _ircopy,  r31              -- Transfer ROMized dynamic program to bank 0
```

The tp offset must be noted only when a label (address) is manipulated in assembly language.

When manipulating a label (address) in C, the label is used as is because the C compiler generates an assembler instruction like that above, taking the tp offset into consideration.

For example, the C source of the above assembly source description is as follows.

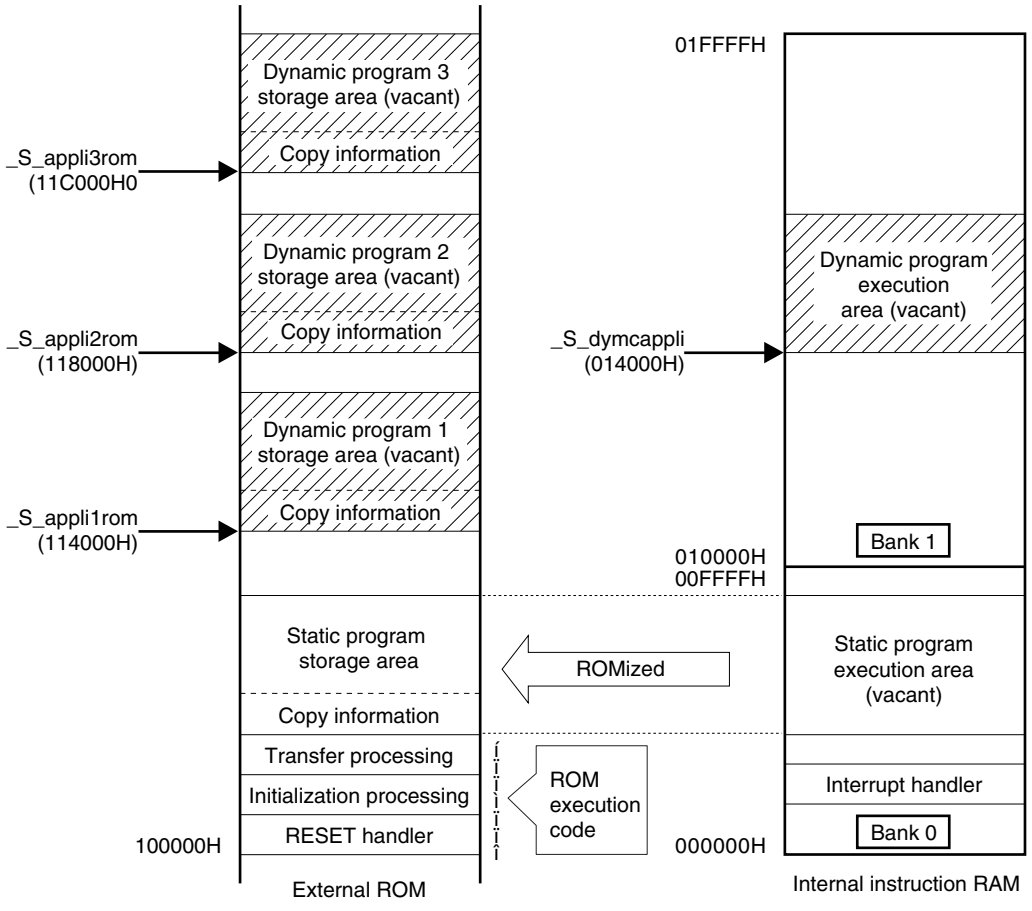
```
_ircopy(_Srompack,1);
```

The static program that is executed in bank 0 of the internal instruction RAM only calls and replaces the dynamic program. Therefore, the following items are linked by vacancy (0 bytes without code entity) with only an address specified.

- Dynamic program execution area first address: _S_dymcappli
- Dynamic program 1 storage area first address: _S_appli1rom
- Dynamic program 2 storage area first address: _S_appli2rom
- Dynamic program 3 storage area first address: _S_appli3rom

An image when the static program is linked is shown below.

Figure 4-34. Image for Linking Static Program



(3) Image during dynamic program link

Dynamic programs 1 to 3 are an executed in the same memory space.

The address where the programs are linked and the address where the CONST segment is linked are the same.

```
[file:progapp1.dir]
# Internal instruction RAM bank 1 space 0x00014000 to 0x00016fff (12 KB)
# Dynamic program group execution area
TEXT        : !LOAD ?RWX V0x00014000 {
        .text           = $PROGBITS     ?AX .text;
};
```

Dynamic programs 1 to 3 are all executed in the same memory space.

```
# Internal instruction RAM bank 1 space 0x00017000 to 0x00017fff (4 KB)
# Dynamically executed program constant data storage area
CONST       : !LOAD ?R   V0x00017000 {
        .const          = $PROGBITS     ?A .const;
};
```

The memory space of the ROMized area that stores dynamic programs 1 to 3 differs.

```
[file:progapp1.dir]
# External EPROM (16-bit width) space 0x00114000 to 0x00116fff (12 KB)
# Dynamically executed program 1 storage area
APPLI1ENTRY  : !LOAD ?RX V0x00114000 {
      .app1entry       = $PROGBITS     ?AX .app1entry;
};

[file:progapp2.dir]
# External EPROM (16-bit width) space 0x00118000 to 0x0011bfff (12 KB)
# Dynamically executed program 2 storage area
APPLI2ENTRY  : !LOAD ?RX V0x00118000 {
      .app2entry       = $PROGBITS     ?AX .app2entry;
};

 [file:progapp3.dir]
# External EPROM (16-bit width) space 0x0011c000 to 0x0011ffff (12 KB)
# Dynamically executed program 3 storage area
APPLI1ENTRY  : !LOAD ?RX V0x0011c000 {
      .app3entry       = $PROGBITS     ?AX .app3entry;
};
```

Make sure that the memory spaces of the global variable areas of the dynamic program and static program, and the stack area do not overlap.

The global variable areas of the static program are the .sdata and .sbss areas and use about 300H addresses from address 0FFFD000H. The global variable areas of the dynamic program are the .sdata and .sbss areas and use about 300H addresses from address 0FFFD800H. The stack area of both the static program and dynamic program is about 200H (512 bytes) from the lower address of the subsequent area, the .bss area.

```
[file:progmain.dir]
# Internal data RAM segment area 0x0fffb000 to 0x00fffefff(20 KB)--V850E/ME2
# Internal data RAM reserved area 0xfff8000 to 0xfffafff (12 KB) (access prohibited)
# Internal data RAM allocated area 0xfffb000 to 0xfffcfff (8 KB) (reserved for ROM Monitor)
DATA  : !LOAD ?RW V0x0fffd000 {
        .data           = $PROGBITS   ?AW  .data;
        .sdata          = $PROGBITS   ?AWG .sdata;
        .sbss           = $NOBITS     ?AWG .sbss;
        .bss            = $NOBITS     ?AW  .bss;
};
```

<R>

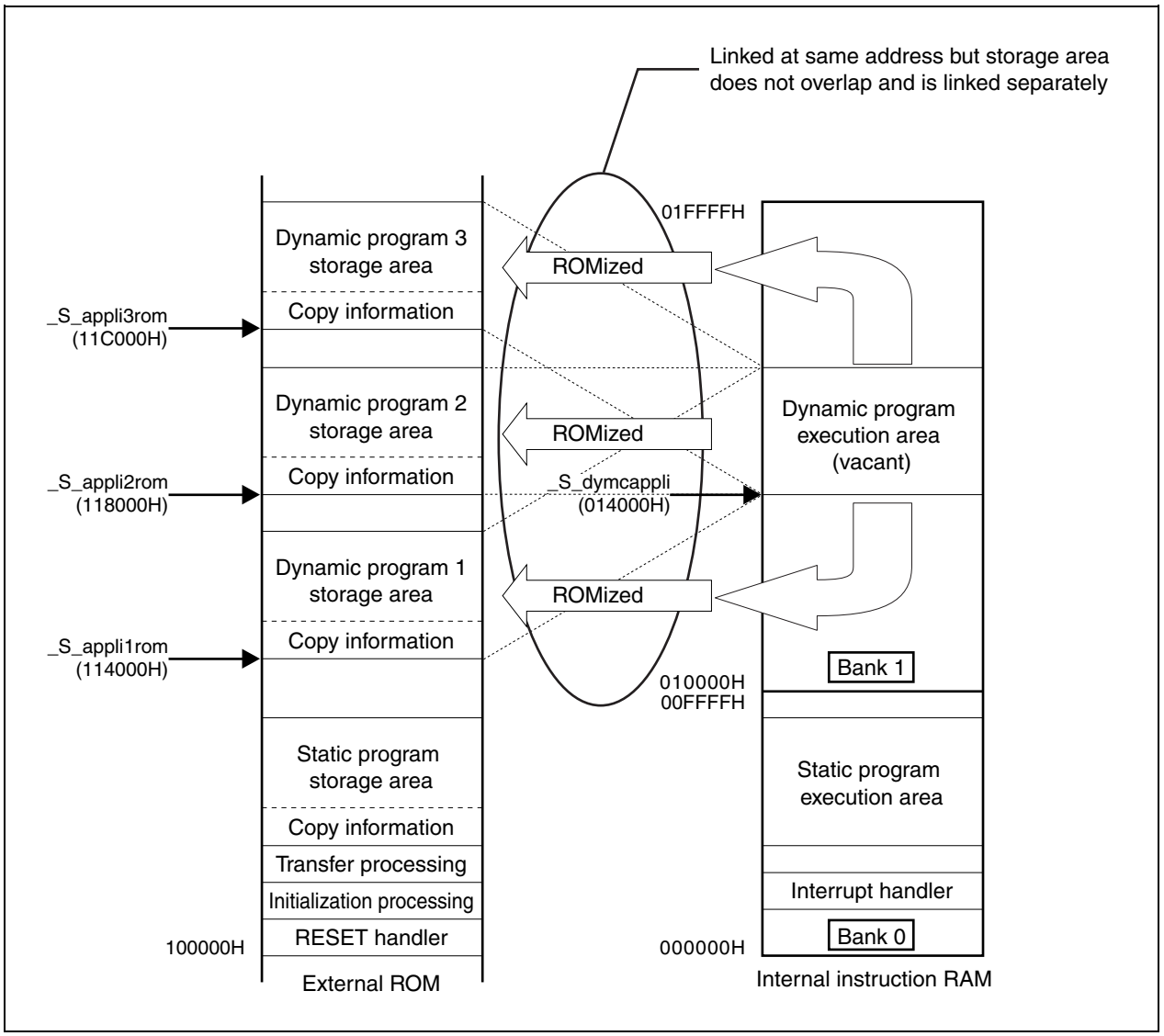
```
[file:progapp1.dir,file:progapp2.dir,file:progapp3.dir]
# Internal data RAM space 0x0fffb000 to 0x00fffefff (20 KB)   -- V850E/ME2 uPD703111A
# Internal data RAM space 0x0fffb000 to 0x00fffcfff (8 KB)    -- V850E/ME2 ROM Monitor
                                                              -- Reserved area
DATA    : !LOAD ?RW V0x0fffd800 {
        .data           = $PROGBITS    ?AW  .data;
        .sdata          = $PROGBITS    ?AWG .sdata;
        .sbss           = $NOBITS      ?AWG .sbss;
        .bss            = $NOBITS      ?AW  .bss;
};
```

```
[file:crtE.s of static program]
#-------------------------------------------------------------------------
#  system stack
#-------------------------------------------------------------------------
     .set   STACKSIZE, 0x200
     .globl    _ _stack
     .bss
     .lcomm    _ _stack, STACKSIZE, 4

[file:crtE.s of dynamic program]
#-------------------------------------------------------------------------
#  system stack
#-------------------------------------------------------------------------
     .set   STACKSIZE, 0x200
     .bss
     .lcomm    _ _stack, STACKSIZE, 4
```

Dynamic programs 1 to 3 are allocated at the same start address but linked separately.
The ROMized area (dynamic program 1 to 3 storage area start address) is located so that the areas do not overlap. The global variable area used for execution of the dynamic program must not overlap the global variable area used for the static program.

An image during dynamic program link is shown below.

Figure 4-35. Image During Dynamic Program Link



(4) Image of writing to ROM

(a) Linking dynamic programs 1 to 3

Because dynamic programs 1 to 3 are all executed at address 00114000H on the same memory space, they are linked in almost the same image.

The only difference between dynamic programs 1 to 3 is the area where the program is stored.

Dynamic program 1 is explained as an example.

The program execution area is allocated to address 00014000H of the internal instruction RAM.

```
[file:progapp1.dir]
# Internal instruction RAM bank 1 space 0x00014000 to 0x00016fff (12 KB)
# Dynamic program group execution area
TEXT         : !LOAD ?RWX V0x00014000 {
        .text            = $PROGBITS     ?AX .text;
};
```

The .const section that is used during standard I/O library link is allocated to address 00017000H of the internal instruction RAM.

```
[file:progapp1.dir]
# Internal instruction RAM bank 1 space 0x00017000 to 0x00017fff (4 KB)
# Dynamically executed program constant data storage area
CONST1       : !LOAD ?R   V0x00017000 {
       .const            = $PROGBITS      ?A .const;
};
```

The program storage area, i.e., the entry of the rompsec section is allocated to address 00114000H.

```
[file:progapp1.dir]
# External EPROM (16-bit width) space 0x00114000 to 0x0016ffff (12 KB)
# Dynamically executed program 1 storage area
APPLI1ENTRY  : !LOAD ?RX V0x00114000 {
       .app1entry       = $PROGBITS      ?AX .app1entry;
};
```

The program storage area, i.e., the entry of the rompsec section is allocated to address 00118000H for dynamic program 2, and to address 0011C000H for dynamic program 3.

```
[file:secentry.s]
       .file     "secentry.s"
       .section  ".app1entry",text
       .align    4
       .globl    _ _S_app1entry, 4
_ _S_app1entry:
```

Finally, the program is converted into a hex file and written to ROM.

The ROM area of the general V850 Series CPU exists at address 00000000H, but the ROM area of the V850E/ME2 starts from address 00100000H. Therefore, an offset must be added during hex code conversion.

Specify the offset with the –d option of the hex code conversion utility hx850 supplied with the C compiler package.

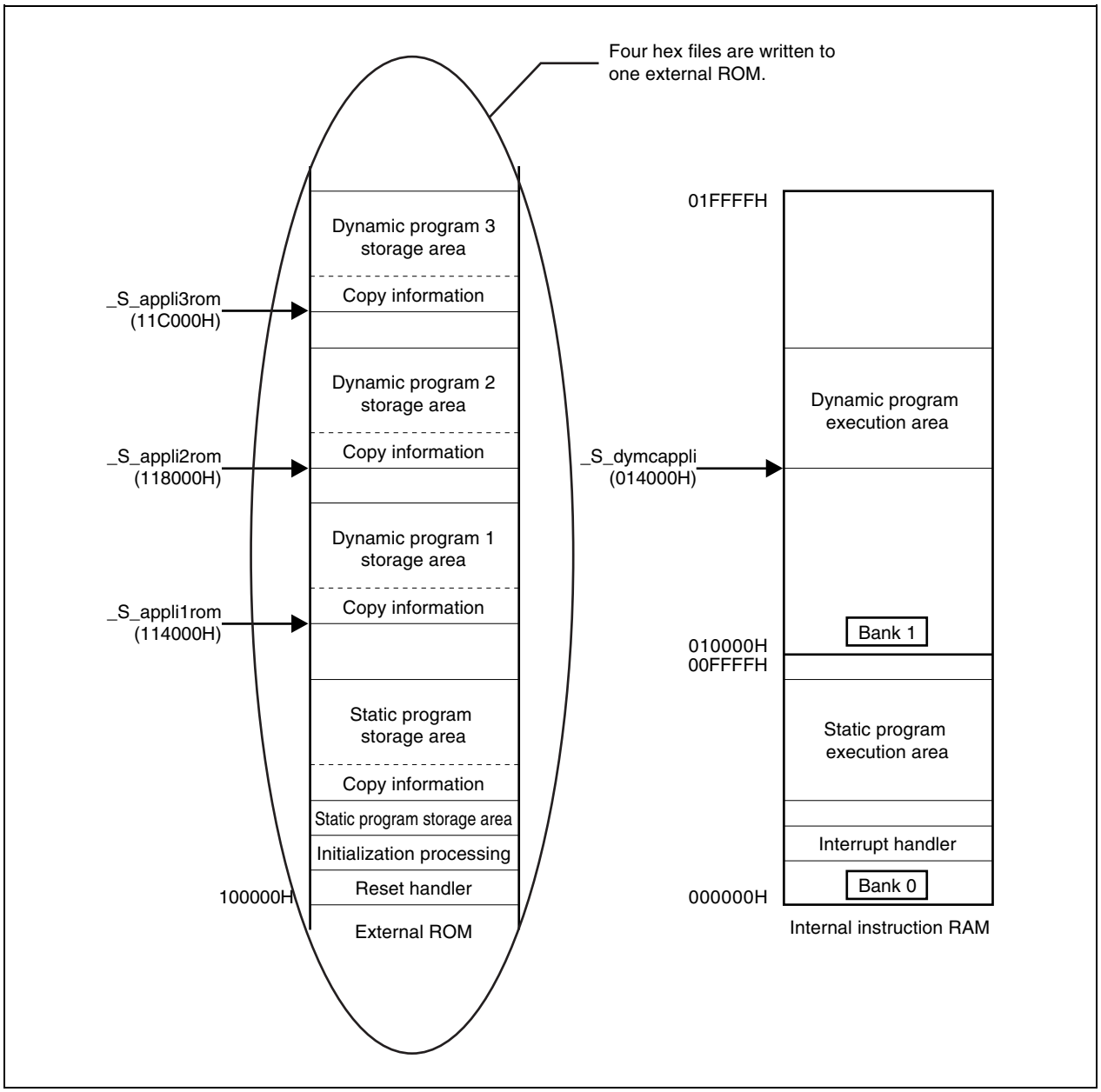
```
-d 0x100000
```

Select [Tool] → [Hex Converter Options …] from the main window of PM+ to display the dialog box to set the hex converter options, and input “0x100000” to the text box of [Offset of Output Address [-d]] on the [Option] tab.

Convert the static program, dynamic program 1, dynamic program 2, and dynamic program 3 into hex files, and write them to ROM. This completes the execution ROM of the dynamic programs.

An image of writing the programs to ROM is shown below.

Figure 4-36. Image of Writing to ROM



(b) Operation from resetting to transferring static program

The Reset handler address of the V850E/ME2 is 100000H in the external EPROM space. The Reset handler address must be explicitly specified by a directive. For details of the directive, refer to **4.2.8 (3) (b) Directive description**.

```
[file:crtE.s]

      .section "RESET", text
      jr       _ _start

[file:progmain.dir]
# CPU RESET segment area
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
RESET   : !LOAD ?RX V0x00100000 {
   RESET               = $PROGBITS    ?AX RESET;
};
```

First, jump from the reset handler address to the _ _start label, which is the beginning of the startup module. Initialize the static program here. The contents of initialization are as follows.

- Setting of gp, tp, sp, and ep
- CPU initialization processing of V850E/ME2
- Dot LED light processing for checking CPU operation
- Initialization of .sbss section
- Initialization of .bss section
- Program transfer to internal instruction RAM

These source parts only have to be executed once after the CPU is reset and do not have to be transferred to the internal instruction RAM. Therefore, by describing a section name in the source file and setting it to a directive file, the source parts are located separated from the static program to be transferred to the internal instruction RAM.

```
[file:crtE.s]
.section ".crtE",text
       .align   4
       .globl   _ _start
       .globl   _ _exit
       .globl   _ _startend
#      .extern  _ _ _PROLOG_TABLE
_ _start:
```

```
[file:progmain.dir]
# Startup module program segment area
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
CRTE        : !LOAD ?RX V0x00101000 {
   .crtE                = $PROGBITS     ?AX .crtE;
};
```

In the above program example, the address is “V0x00101000” because of the ROM Monitor debug environment. However, the address position may be contiguous with the RESET segment.
Program examples of CPU initialization processing in the V850E/ME2 and dot LED lighting processing for checking CPU operation are shown below.

```
[file:crtE.s]
       jarl    _initial,lp       -- V850E/ME2 CPU initial
_initial_end:

       jr      _ledtest
       .globl  _ledtest_end
_ledtest_end:
```

```
[file:ledtest.s]
       .section ".ledtest",text
       .align   4
       .globl   _ledtest

       .extern  _ledtest_end

# Initialization index processing
# Dot LED binary shift
_ledtest:
```

```
[file:initial.s]
      .section ".initial", text
      .align   4
      .globl   _initial
_initial:
```

```
[file:progmain.dir]
# CPU initialization program segment area
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
INITIAL : !LOAD ?RX {
    .initial          = $PROGBITS     ?AX .initial;
};

# LED lighting program segment area
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
LEDTEST : !LOAD ?RX {
    .ledtest          = $PROGBITS     ?AX .ledtest;
};
```

An example of a program that transfers programs to the internal instruction RAM is shown below.

```
[file:progmain.dir]
# Internal instruction RAM area transfer program segment area
# External EPROM (16-bit width) area 0x100000 to 0x11ffff (128 KB)
IRCOPY    : !LOAD ?RX  {
   .ircopy             = $PROGBITS      ?AX .ircopy;
};
```

(5) Static program transfer

The _ircopy function is used to transfer programs.
This function executes the following processing.

- Passes the first address of the ROMized section as the first argument.
- Passes a section ID number that indicates which section is to be transferred as the second argument.

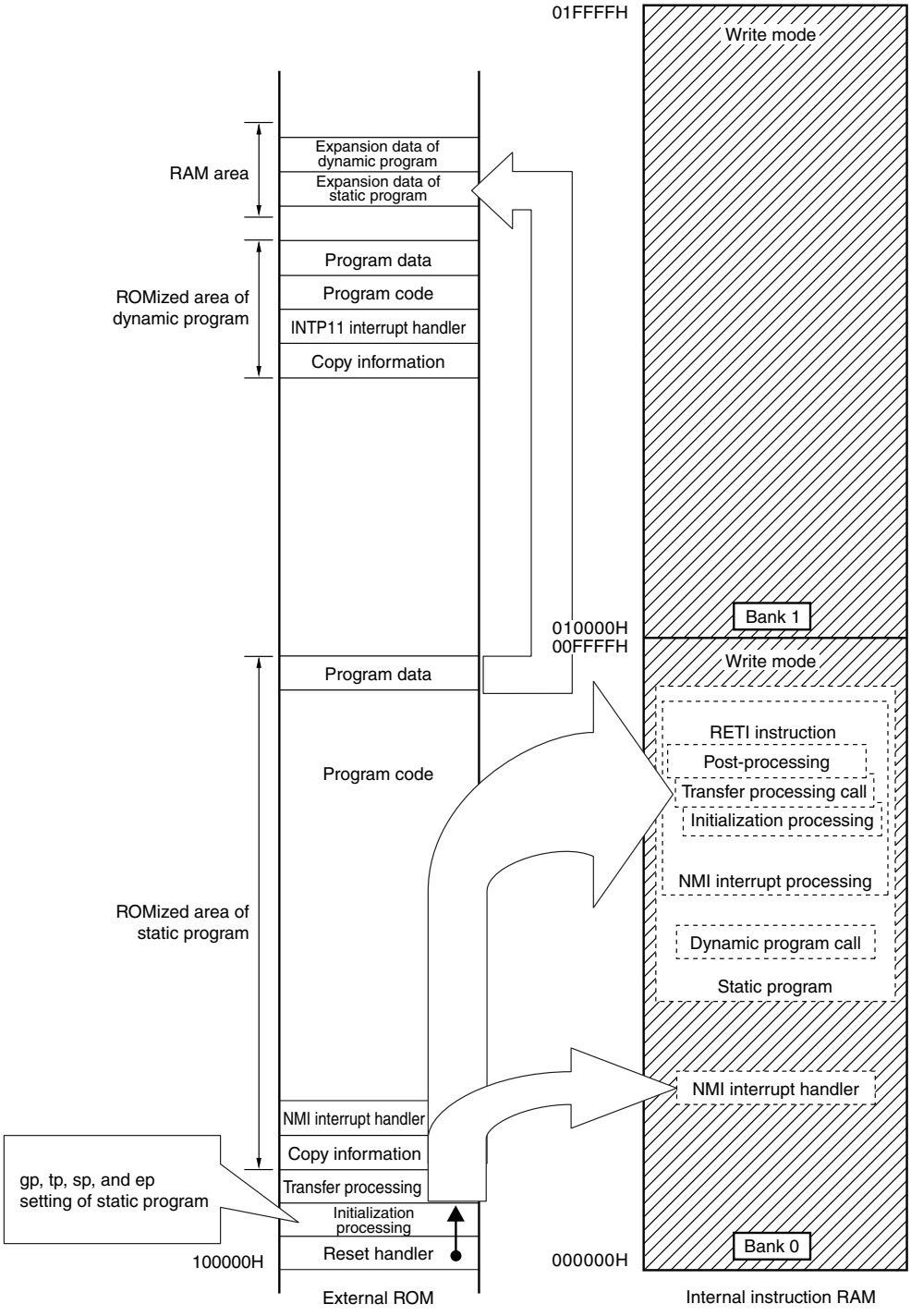
The features of the _ircopy function are as follows.

- Does not use the stack.
- Discards r20 and r21, which are used as mask registers.
- Transfers in word (4-byte) units.
- If the lower 4 bits of the transfer size are not a multiple of a word unit (4 bytes) (0x0, 0x4, 0x8, or 0xc) but a multiple of a halfword (2-byte) unit (0x2, 0x6, 0xa, or 0xe), data is word-aligned and transferred. Therefore, undefined halfword data is transferred to the end of the transfer destination.

After the CPU is reset, banks 0 and 1 of the internal instruction RAM are both in the write mode. The transfer program on the external RAM reads the static program from the external ROM and writes it to bank 0.

The flow of the processing is shown below.

Figure 4-37. Flow of Static Program Transfer Processing



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

(6) Dynamic program transfer executed first

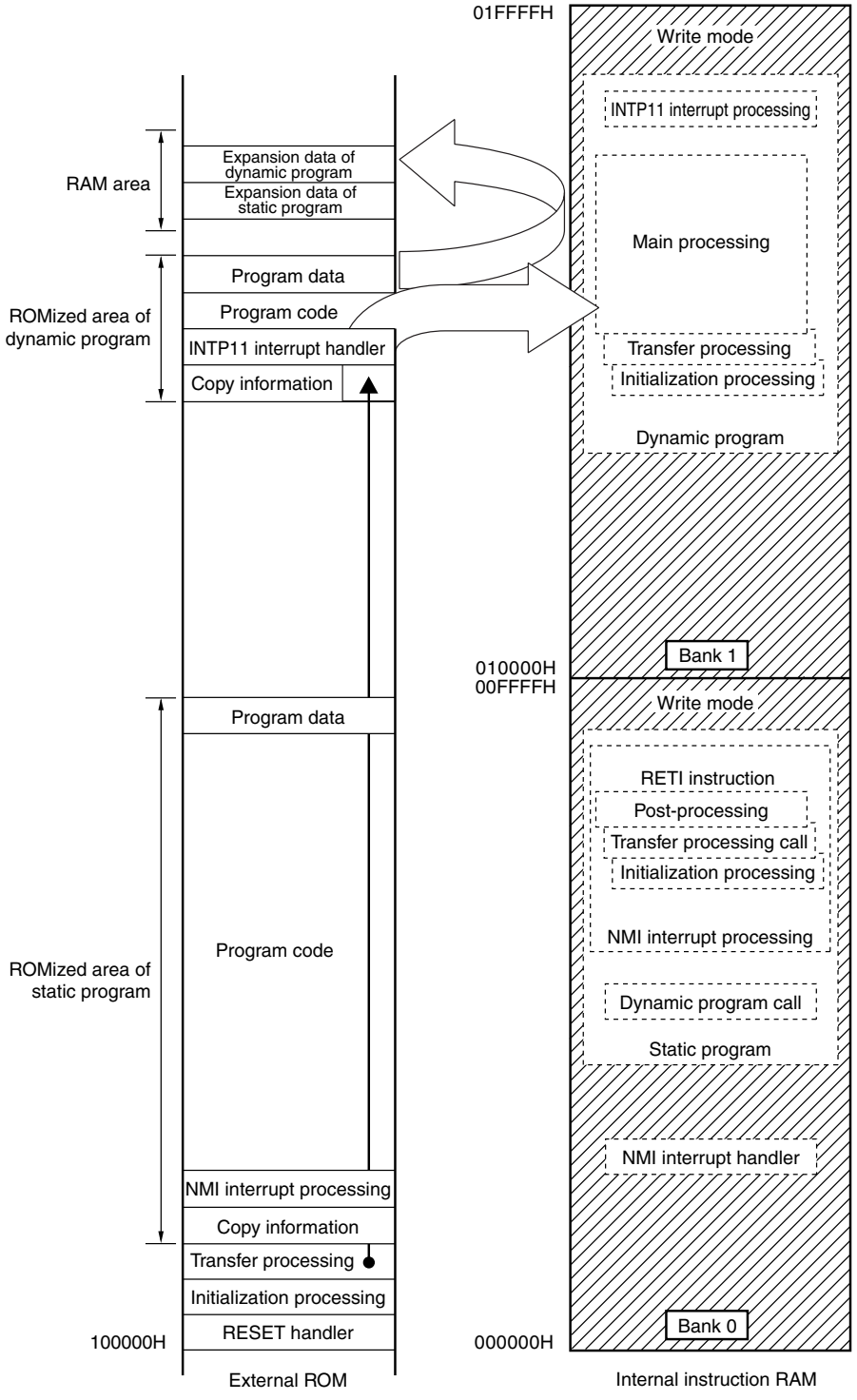
The dynamic program to be executed first is transferred to the internal instruction RAM by using the label of dynamic program 1 (fixed address) linked by vacancy (code entity of 0 bytes with only first address) and the _ircopy function. By passing –1 as the ID number, all ROMized codes and data are transferred.

```
[file:crtE.s]
.extern _ _S_appli1rom
.option nowarning
   movhi  hi1( _ _S_appli1rom), tp, r1    -- Dynamic program 1 packing section start
   movea  lo(_ _S_appli1rom),   r1, r6    -- address to which offset of tp is added
.option warning
   mov    -0x1,      r7                  -- All packing sections to be transferred
   jarl   _ _ircopy, r31                 -- Transfer ROMized dynamic program to bank 1
```

After completion of transfer of the static program to bank 0 of the internal instruction RAM, the dynamic program to be executed first is read from the external ROM, and transferred to bank 1 of the internal instruction RAM.

The flow of processing is shown below.

Figure 4-38. Flow of Processing When Dynamic Program to Be Executed First Is Transferred



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

(7) Starting program execution in internal instruction RAM

After completing processing steps (1) to (6) above, the program is ready to be executed in the internal instruction RAM, so as a last step, jump to the main function of C.

```
[file:crtE.s]
-- (iii) To internal instruction RAM by branch instruction
.option nowarning
   mov    #_main,    r1
   jmp    [r1]
.option warning
```

Jump may be executed by the jr instruction within a range of ±2 MB (±21 bits).
The main function immediately calls the program to be executed dynamically.

```
[file:progmain.c]
extern void _S_dymcappli(void);
(Omission)
   for( ;; ){
       _S_dymcappli();
   }
```

The _S_dymcappli function is vacant (0 bytes without code entity) because the program code is dynamically replaced. Only a label (address) is defined.

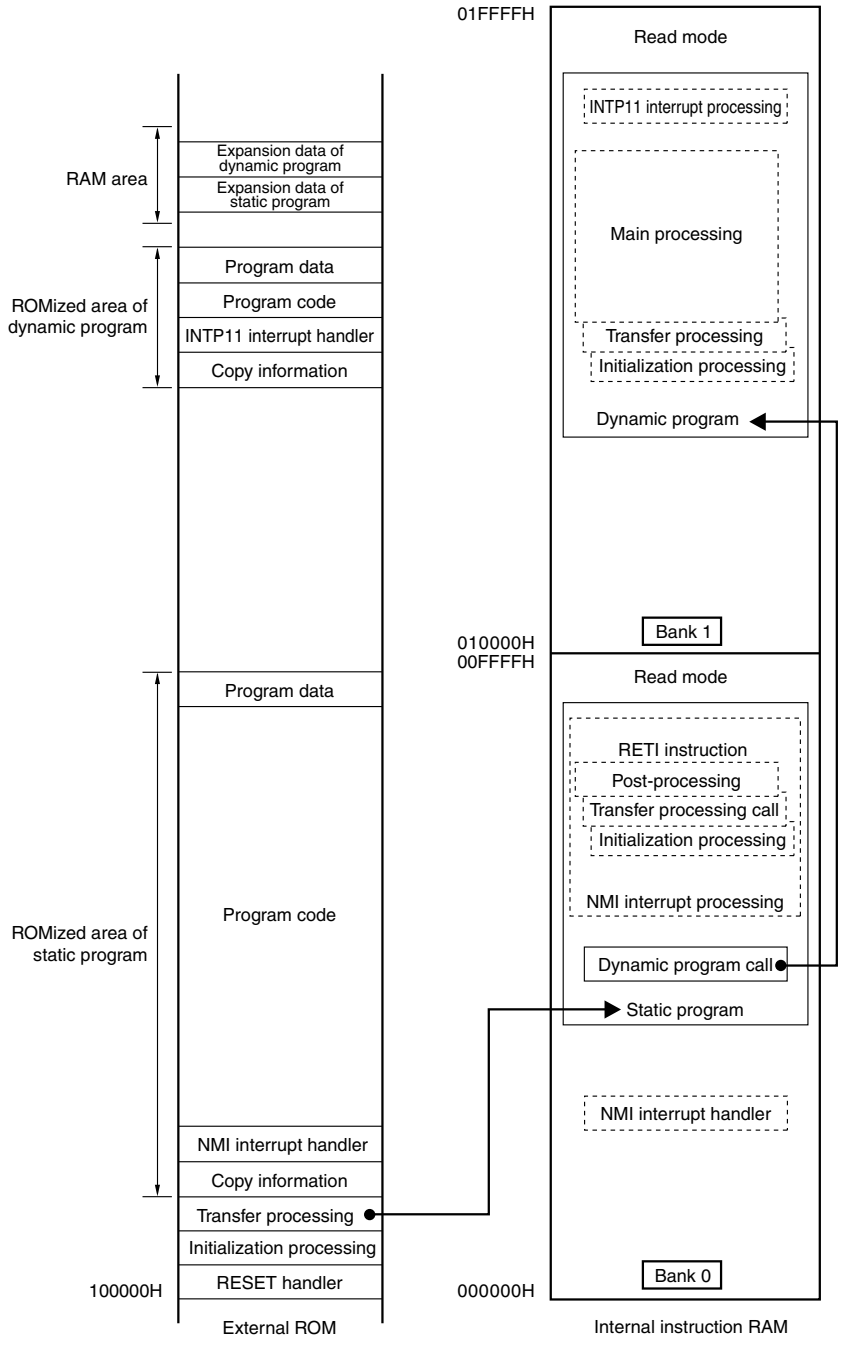
```
[file:secentry.s]
#  Internal instruction RAM (bank 1) area where program transferred from external EPROM area is dynamically executed
        .section ".dymcappli",text
        .align 4
        .globl _ _S_dymcappli, 4
_ _S_dymcappli:
```

```
[file:progmain.dir]
#  Program segment area that operates dynamically
#  Internal instruction RAM area bank 1 0x00010000 to 0x0001ffff (64 KB)
DYMCAPPL  : !LOAD ?RWX V0x00014000 {
       .dymcappli      = $PROGBITS      ?AX A0x4 .dymcappli;
};
```

After transferring all the programs to the internal instruction RAM, set both banks 0 and 1 of the internal instruction RAM in the read mode, and jump to the main function of the static program. A dynamic program is called in the main function.

The flow of the processing is illustrated below.

Figure 4-39. Flow of Processing When Program Execution Is Started in Internal Instruction RAM



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

: Processing under execution

(8) Initialization of dynamic program

When the _S_dymcappli function is called, the startup module of the dynamic program is started, and the following processing is executed.

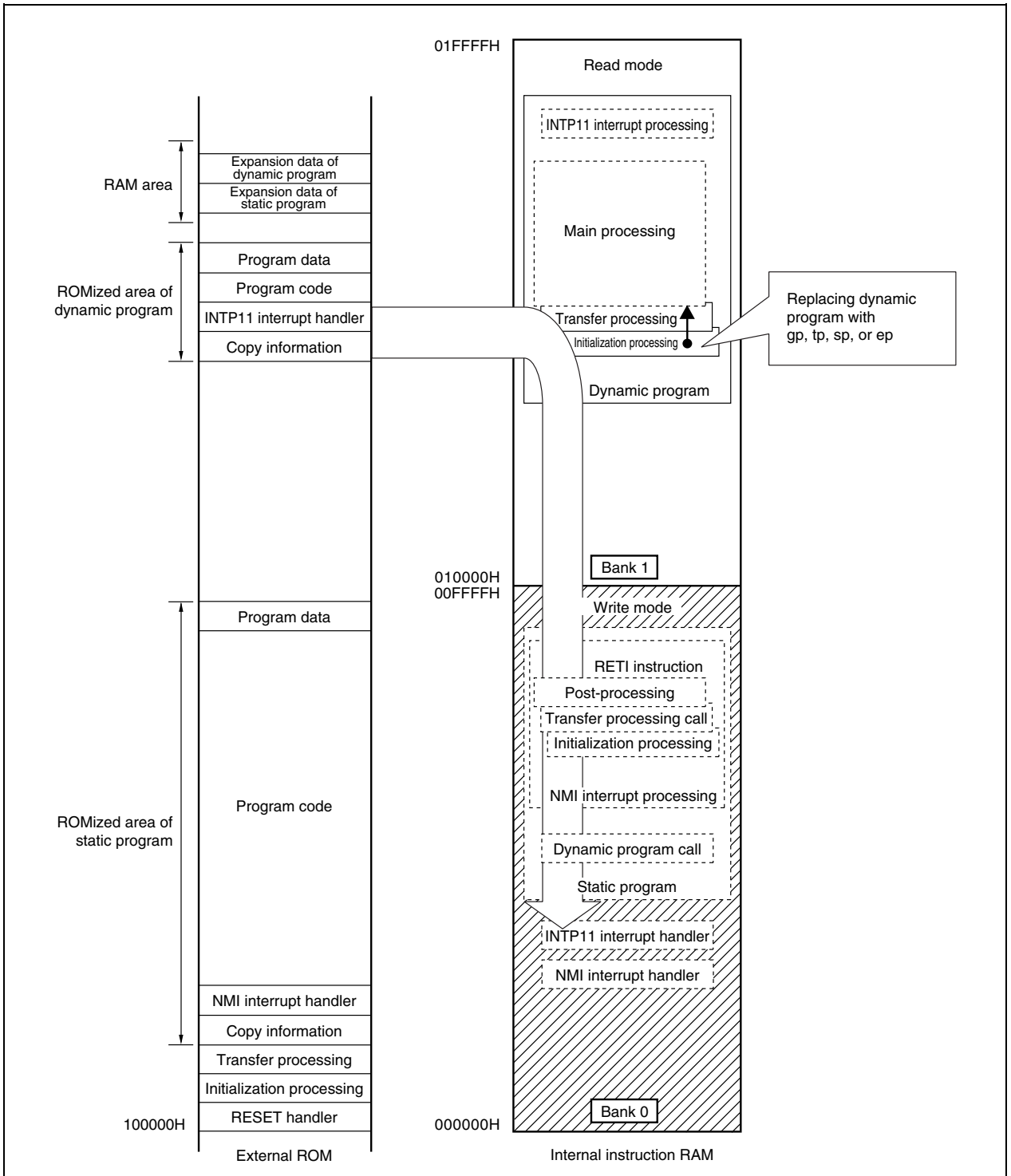
- Setting of gp, tp, sp, and ep
- Initialization of .sbss section
- Initialization of .bss section
- Transfer of INTP11 interrupt section (INTP11 interrupt handler) to bank 0 of internal instruction RAM

After execution of the processing, execution jumps to the main function of the dynamic program.

The dynamic program is executed on the bank 1 side. After initialization for the dynamic program, set bank 0 of the internal instruction RAM in the write mode again, read the INTP11 interrupt handler for the dynamic program from ROM, and write it to bank 0.

The flow of the processing is shown below.

Figure 4-40. Flow of Processing When Dynamic Program Is Initialized



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

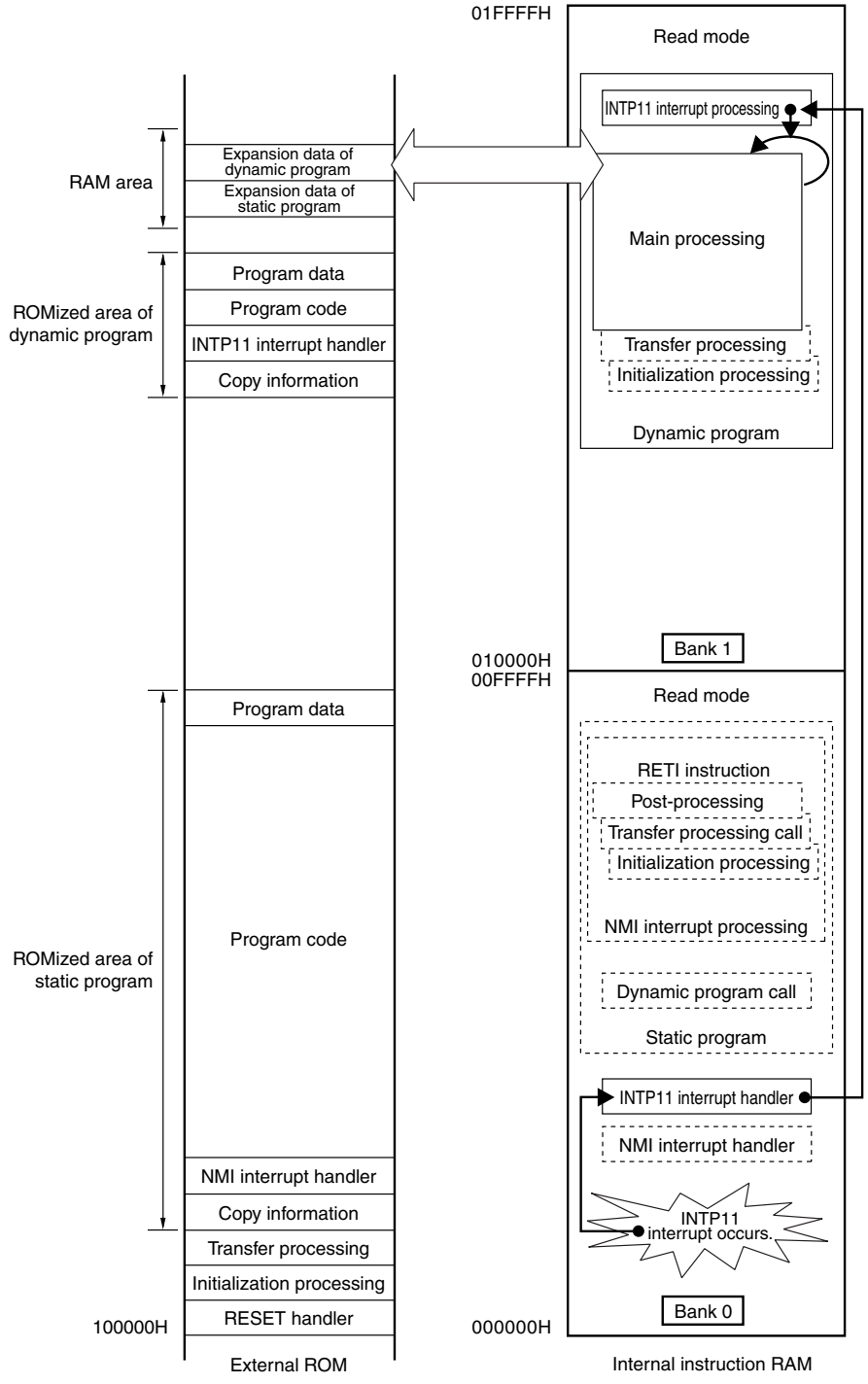
: Processing under execution

(9) Executing dynamic program

After writing the INTP11 interrupt handler of the dynamic program to bank 0, return bank 0 to the read mode. When the INTP11 interrupt occurs, the INTP11 interrupt processing of the dynamic program can be executed via the INTP11 interrupt handler in bank 0.

The flow of the processing is illustrated below.

Figure 4-41. Flow of Processing When Dynamic Program is Executed



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

: Processing under execution

(10) Replacing other dynamic program by non-maskable interrupt (NMI)

An expansion description of the NMI interrupt in C is shown below.

For details of C expansion descriptions, refer to **CHAPTER 4 C LANGUAGE EXPANSION** in the **CA850 C Language User's Manual (U17291E)**.

```
[file:progmain.c]
#pragma interrupt NMI0 int_nmi
_ _interrupt
```

When an NMI interrupt occurs, control is returned from the dynamic program to the static program. Initialization of tp, gp, sp, and ep is executed, and the environment where C of the static program can be executed is restored.

```
[file:progmain.c]
  _ _asm("mov #_ _tp_TEXT, tp");       /* tp register */
  _ _asm("mov #_ _gp_DATA, gp");       /* gp register offset */
  _ _asm("add tp, gp");                /* gp register */
  _ _asm("mov #_ _stack+0x200, sp");   /* sp register */
  _ _asm("mov #_ _ep_DATA, ep");       /* ep register */
```

The external reference label (fixed address) where dynamic programs 1 to 3 are stored is an array. Each time an NMI interrupt occurs, addresses are sequentially referenced from this array.

```
[file:progmain.c]

extern void _S_dymcappli(void);
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);

/* Global variable definition */
int id = 0;                                  /* Dynamically executed program ID number */
void (*appllitbl[4])(void)  ={               /* Dynamically executed program address array */
_S_appli1rom,_S_appli2rom,_S_appli3rom, (void (*)())NULL };
(Omission)
    /* Dynamically executed programs are sequentially transferred. */
    if(++id > 2)
         id = 0;
```

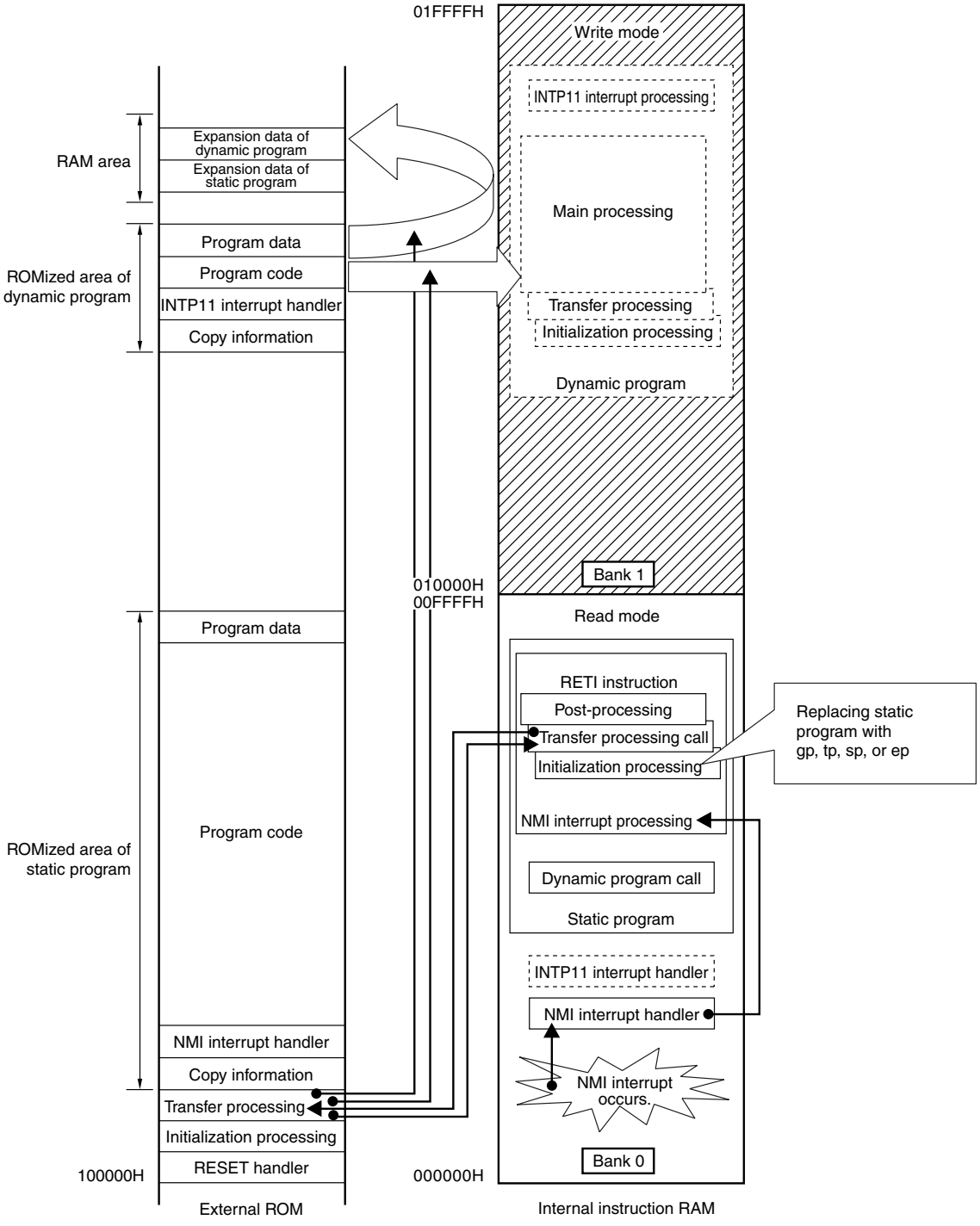
Set bank 1 of the internal instruction RAM where the dynamic program is to be executed in the write mode, pass its address to the _ircopy function, and transfer the entire new dynamic program.
After completion of transfer, return bank 1 of the internal instruction RAM to the read mode.

```
[file:progmain.c]
/* Set internal instruction RAM bank 1 (0x00010000 to 0x0001ffff) in write mode. */
IRAMM = 0x02; /* Set internal instruction RAM mode register (IRAMM) in write mode (IRAMM1 = 1). */
do{
  ; /* Check that internal instruction RAM mode register (IRAMM) changed to write mode (IRAMM1 = 1?). */
}while(IRAMM != 2);

_ircopy(applitbl[id],-1);
/* Set internal instruction RAM bank 1 (0x00010000 to 0x0001ffff) to read mode. */
IRAMM = 0x00; /* Set internal instruction RAM mode register (IRAMM) to read mode (IRAMM1 = 0). */
do{
  ; /* Check that internal instruction RAM mode register (IRAMM) changed to read mode (IRAMM1 = 0?). */
}while(IRAMM != 0);
```

The NMI interrupt processing is executed on the static program side of bank 0 of the internal instruction RAM. After initialization for the static program, set bank 1 of the internal instruction RAM in the write mode in the NMI interrupt processing, read the dynamic program from ROM, and write it to bank 1. Because the NMI interrupt handler is to be written to bank 0 of the internal instruction RAM, it cannot be written at this stage.

The flow of the processing is illustrated below.

Figure 4-42. Flow of Processing to Replace Other Dynamic Program by Non-Maskable Interrupt (NMI)



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

: Processing under execution

(11) Returning to other dynamic program from non-maskable interrupt (NMI)

The return address of the NMI interrupt is stored in the system register FEPC.

As soon as the RETI instruction has been executed, the value of FEPC is written to the PC. Execution is not returned to the dynamic program before replacement in which the NMI interrupt occurred; a new dynamic program must be executed from the start.

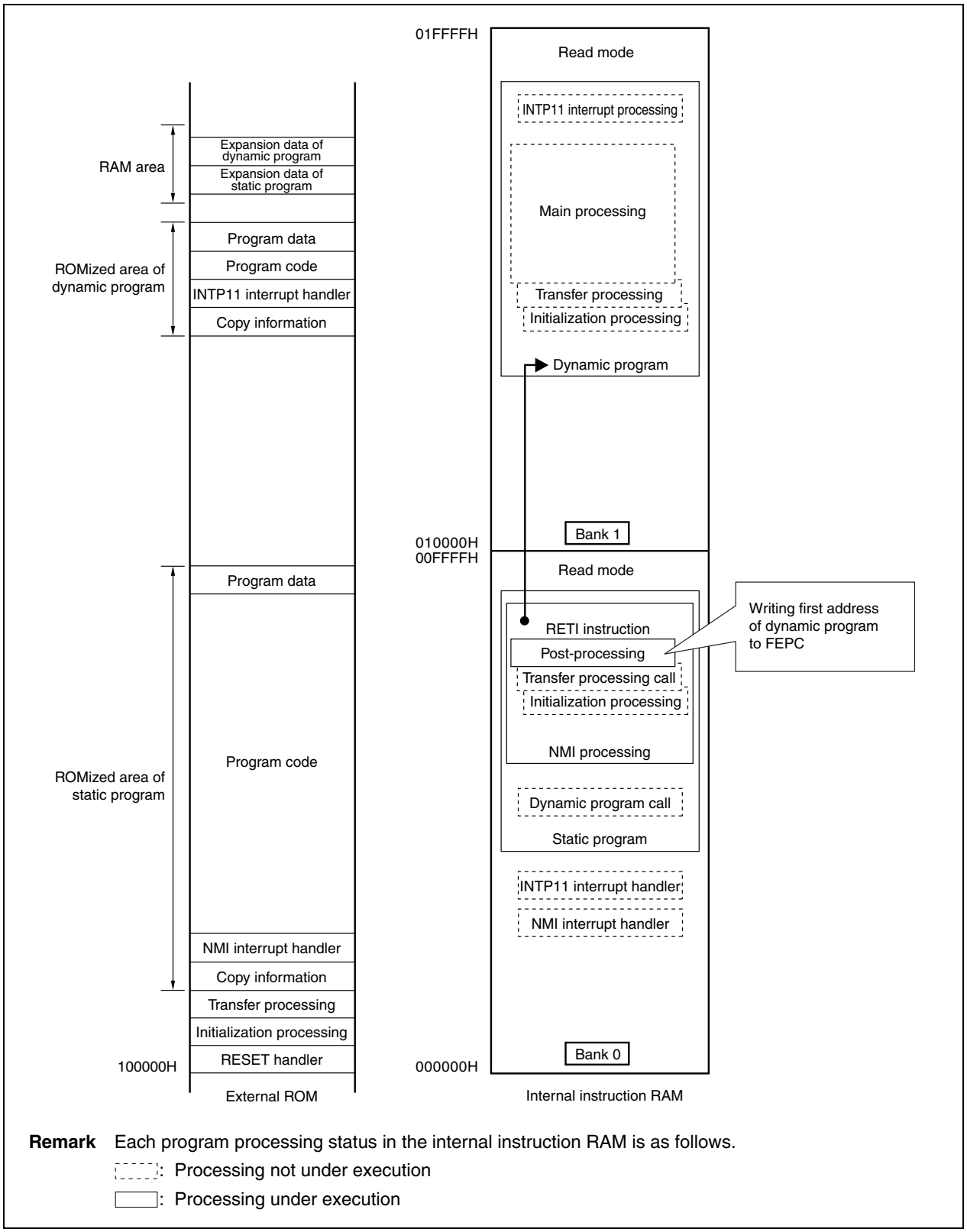
If _S_dymcappli, which is the first label (fixed address) of the dynamic program execution area, is written over FEPC and execution returns from the NMI interrupt, the program that has replaced the preceding dynamic program is executed from the start.

```
[file:progmain.c]
   /* Specify return destination of NMI0 interrupt at address of area to which program
   to be dynamically executed is to be transferred. */
   _ _asm("movhi hi1(#_ _S_dymcappli),r0,r6");
   _ _asm("movea lo(#_ _S_dymcappli), r6,r6");
   _ _asm("ldsr  r6, 2");
```

Before exiting from the NMI processing, set bank 1 of the internal instruction RAM, which is the dynamic program execution area, in the read mode, and set the return address of the NMI to the beginning of the dynamic program execution area.

The flow of the processing is illustrated below.

Figure 4-43. Flow of Processing When Other Dynamic Program Is Executed After Returning from Non-Maskable Interrupt (NMI)



Remark Each program processing status in the internal instruction RAM is as follows.

: Processing not under execution

: Processing under execution

(12) Program list

(a) Static program list

A list of source files of the static program list and a list of the static programs are shown below.

Table 4-11. Source Files of Static Program List

Source File Name	Outline
progmain.c	main function, NMI function
initial.s	V850E/ME2 initialization processing
ledtest.s	Dot LED lighting processing
printf.c	UARTB1 output processing function group
uart.h	UARTB-related header file
crtE.s	Startup module processing for static program
secentry.s	Vacant entry link
rompack.s	rompsec section entry
progmain.dir	Link directive for static program
ircopy.s	_ircopy function (Refer to **4.2.8 (6) Program example**.)

```
[progmain.c]
#pragma ioreg
#ifndef NULL
#ifndef NULL 0
#endif
#define DOTLED8_3 P5

/* External variable reference declaration */
extern int _ _ _stack;
extern void _S_rompack();
extern void _S_dymcappli(void);
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);

/* Global variable definition */
int id = 0;                              /* Dynamically executed program ID number */
void (*applIitbl[4])(void) ={            /* Dynamically executed program array */
_S_appli1rom,_S_appli2rom,_S_appli3rom, (void (*)())NULL };

/*************************************************************************
 *
 *   Interrupt request name: NMI0
 *   Function              : NMI SW interrupt processing executed in static program
 *
 *************************************************************************/
#pragma interrupt NMI0 int_nmi
_ _interrupt void int_nmi(void){

     _ _asm("mov #_ _tp_TEXT, tp");      /* tp register */
     _ _asm("mov #_ _gp_DATA, gp");      /* gp register offset */
     _ _asm("add tp, gp");               /* gp register */
     _ _asm("mov #_ _stack+0x200, sp");  /* sp register */
     _ _asm("mov #_ _ep_DATA, ep");      /* ep register */

     /* Specify return value of NMI0 to first address of area to which dynamically executed
     program is to be transferred. */
     _ _asm("movhi hi1(#_ _S_dymcappli),r0,r6");
     _ _asm("movea lo(#_ _S_dymcappli), r6,r6");
     _ _asm("ldsr  r6, 2");

     /* Sequentially transfer dynamically executed programs. */
     if(++id > 2)
         id = 0;

     /* Set internal instruction RAM bank 1 (0x00010000 to 0x0001ffff) in write mode. */
     IRAMM = 0x02;/* Set internal instruction RAM mode register (IRAMM) in write mode (IRAMM1 = 1). */
     do{
      ; /* Check that internal instruction RAM mode register (IRAMM) changed to write mode (IRAMM1 = 1?). */
     }while(IRAMM != 2);
```

```
    _ircopy(applitbl[id],-1);

    /* Set internal instruction RAM bank 1 (0x00010000 to 0x0001ffff) in read mode. */
    IRAMM = 0x00;/* Set internal instruction RAM mode register (IRAMM) in read mode (IRAMM1 = 0). */
    do{
     ; /* Check that internal instruction RAM mode register (IRAMM) changed to read mode (IRAMM1 = 0?). */
    }while(IRAMM != 0);
}

/*************************************************************************
 *
 *   Function name:  main
 *   Purpose:        Static program main processing
 *   Argument:       None
 *   Return value:   None
 *
 *************************************************************************/
void main(void)
{
/*************************************************************************/
/**** This program is executed in internal instruction RAM bank 0 area (0x00000000 to 0x0000ffff)  ****/
/*************************************************************************/

    init_uartb1(19200); /* UARTB1 channel initialization */
    printf("\r\n**** Dynamic Implement Sample Program ****\r\n\r\n");

    for(;;){
    _S_dymcappli();
    }
}
```

```
 [initial.s]
#############################################################
# -------- Internal I/O register initialize -----------
# The initialization sequence of the V850E/ME2 is as follows.
#
# Setting of registers that affect external bus access performance
# Checking LOCK bit of lock register (LOCKR)
-- (0) Read lock register (LOCKR) to confirm stabilization of PLL.
-- (i) Set x7H to system wait control register (VSWC) (x: set value set in I above).
-- (ii) Set frequency division value of external bus to bus mode control register (BMC).
-- (iii) Set value set in I above to system wait control register (VSWC) again
-- (iv) Set internal system clock frequency division value to clock control register (CKC).
-- (v) Set main clock (fx) supply control to clock source select register (CKS).
# Transfer program code to internal instruction RAM.
#
#############################################################
     .set    TBV850E, 1
     .section ".initial", text
     .align  4
     .globl  _initial
_initial:
#########################################################
# Setting of registers that affect external bus performance
#########################################################
#    Set wait states for internal peripheral I/O access.
     mov     0x33,     r6          -- Set value of VSWC when 125.00 MHz < fx ≤ 150.00 MHz
     st.b    r6,       VSWC[r0]    -- System wait control register

#########################################################
#    Specify data wait for each CS space
#########################################################
     mov     0x0003,   r6          -- CS0: 3 waits (TW) inserted in EPROM space
                                   -- CS1: 0 waits (TW) inserted in SDRAM space
                                   -- CS2: 0 waits (TW) inserted in SRAM space
                                   -- CS3: 0 waits (TW) inserted in SRAM space
     mov     0x0000,   r7          -- CS4: 0 waits (TW) inserted in SDRAM space
                                   -- CS5: 0 waits (TW) inserted in SRAM space
                                   -- CS6: 0 waits (TW) inserted in SDRAM space
                                   -- CS7: None
     st.h    r6,       DWC0[r0]    -- Data wait control register 0
     st.h    r7,       DWC1[r0]    -- Data wait control register 1

     mov     0x0000,   r6          -- CS0: 0 waits (TASW) inserted in EPROM space
                                   -- CS1: 0 waits (TASW) inserted in SDRAM space
                                   -- CS2: 0 waits (TASW) inserted in SRAM space
                                   -- CS3: 0 waits (TASW) inserted in SRAM space
                                   -- CS4: 0 waits (TASW) inserted in SDRAM space
                                   -- CS5: 0 waits (TASW) inserted in SRAM space
                                   -- CS6: 0 waits (TASW) inserted in SDRAM space
                                   -- CS7: None
```

```
     st.h    r6,        ASC[r0]      -- Address setup wait control register

#  DMA flyby transfer wait control register (FWC) at default value (7777H)
--  When FW00 to 02 = 111 and DMA ch. 0 is used,
--  7 waits inserted for access to SDRAM and 8 waits inserted for access to other than SDRAM
--  When FW10 to 12 = 111 and DMA ch. 1 is used,
--  7 waits inserted for access to SDRAM and 8 waits inserted for access to other than SDRAM
--  When FW20 to 22 = 111 and DMA ch. 2 is used,
--  7 waits inserted for access to SDRAM and 8 waits inserted for access to other than SDRAM

#########################################################
#     Specify idle state for each CS space
#########################################################
     mov     0x0452,   r6           -- CS0: 2 idle (TI) inserted in EPROM space
                                    -- CS1: 0 idles (TI) inserted in SDRAM space
                                    -- CS2: 1 idle (TI) inserted in SRAM space
                                    -- CS3: 1 idle (TI) inserted in SRAM space
                                    -- CS4: 0 idles (TI) inserted in SDRAM space
                                    -- CS5: 1 idle (TI) inserted in SRAM space
                                    -- CS6: 0 idles (TI) inserted in SDRAM space
                                    -- CS7: None
     st.h    r6,        BCC[r0]     -- Bus cycle control register

#  DMA flyby transfer idle control register (FIC) at default value (3333H)
--  When FI00 to 01 = 11 and DMA Ch. 0 is used, 3 idles inserted
--  When FI10 to 11 = 11 and DMA Ch. 1 is used, 3 idles inserted
--  When FI20 to 21 = 11 and DMA Ch. 2 is used, 3 idles inserted

#########################################################
#     Of 256 MB memory space, divide
#     lower 8 MB (00000000H to 07FFFFFFH) and
#     higher 8 MB (0F800000 to 0FFFFFFFH) into
#     memory blocks in 2 MB units,
#     and specify chip select signal.
#########################################################
     mov     0x0e01,   r6           -- Area 0 (64 MB)
                                    -- Block 0 x0100000H to x01fffffH: (1 MB) CS0
                                    -- Block 1 x0200000H to x03fffffH: (2 MB) CS2
                                    -- Block 2 x0400000H to x05fffffH: (2 MB) CS2
                                    -- Block 3 x0600000H to x07fffffH: (2 MB) CS2
                                    --          x0800000H to x3ffffffH: (56 MB) fixed to CS1

                                    -- Area 1 (64 MB) x4000000H to x7ffffffH: (64 MB) Fixed to CS3
                                    -- Area 2 (64 MB) x8000000H to xbffffffH: (64 MB) Fixed to CS4
```

```
    mov     0x0f00,   r7          -- Area 3 (64 MB) xc000000H to xf7fffffH: (56 MB) fixed to CS6
                                  -- Block 4 xf800000H to xf9fffffH: (2 MB) CS5
                                  -- Block 5 xfa00000H to xfbfffffH: (2 MB) CS5
                                  -- Block 6 xfc00000H to xfdfffffH: (2 MB) CS5
                                  -- Block 7 xfe00000H to xfffffffH: (2 MB) CS5
                                  -- CS7 not used
    st.h    r6,       CSC0[r0]    -- Chip area selection control register 0
    st.h    r7,       CSC1[r0]    -- Chip area selection control register 1

#########################################################
#     Specify external device to be connected to each CS space
#########################################################
    mov     0x88b8,   r6          -- CS0: EPROM x0100000H to x011ffffH (128 KB)
                                  -- CS1: SDRAM x1000000H to x2fffffffH (32 MB)
                                  -- CS2: SRAM x0200000H to x027ffffH (512 KB)
                                  -- CS3: SRAM x4000000H to x403ffffH (256 KB)
    mov     0x0b8b,   r7          -- CS4: SDRAM x8000000H to x8ffffffH (16 MB)
                                  -- CS5: SRAM xF800000H to xF81ffffH (128 KB)
                                  -- CS6: SDRAM xC000000H to xCffffffH (16 MB)
                                  -- CS7: None
    st.h    r6,       BCT0[r0]    -- Bus cycle type configuration register 0
    st.h    r7,       BCT1[r0]    -- Bus cycle type configuration register 1

#########################################################
#     Specify data bus width of each CS space
#########################################################
    mov     0x0169,   r6          -- CS0: EPROM x0100000H to x011ffffH (128 KB) 16 bits
                                  -- CS1: SDRAM x1000000H to x2fffffffH (32 MB) 32 bits
                                  -- CS2: SRAM x0200000H to x027ffffH (512 KB) 32 bits
                                  -- CS3: SRAM x4000000H to x403ffffH (256 KB) 16 bits
                                  -- CS4: SDRAM x8000000H to x8ffffffH (16 MB) 16 bits
                                  -- CS5: SRAM xF800000H to xF81ffffH (128 KB) 8 bits
                                  -- CS6: SDRAM xC000000H to xCffffffH (16 MB) 8 bits
                                  -- CS7: None                                 8bits
    st.h    r6,       LBS[r0]     -- Local bus sizing control register

#########################################################
#     Specify endian in each CS space
#########################################################
    mov     0x0000,   r6          -- Little endian in all CS0 to CS7 spaces
    st.h    r6,       BEC[r0]     -- Endian configuration register

#########################################################
#     Specify data read control function in each CS space
#########################################################
    mov     r0,       r6          -- No speculative read in all CS0 to CS3 spaces (read buffer operation disabled)
    mov     r0,       r7          -- No speculative read in all CS4 to CS7 spaces (read buffer operation disabled)
    st.h    r6,       LBC0[r0]    -- Line buffer control register 0
    st.h    r7,       LBC1[r0]    -- Line buffer control register 1
```

```
#########################################################
#     Wait specification, etc., for on-page access (not provided)
#########################################################
--    mov     0x7000,   r6          -- 2 × 32 bits, 4 × 16 bits, 8 × 8 bits
                                    -- Inserts 7 waits (TW) for on-page access
--    st.h    r6,       PRC[r0]     -- Page ROM configuration register

#########################################################
#     Specify use of D16 to D31 pins
#########################################################
-- Setting of PMDH, PMCDH, and PFCDH is for 16-bit bus mode.
-- 16-bit bus mode when MODE1 pin = 1, MODE0 pin = 0
      mov     0x0000,   r6          -- Sets inactive level (low) of PDH15 to PDH0.
      st.h    r6,       PDH[r0]
      mov     0x0000,   r6          -- PDH15 to PDH0 in output mode
      st.h    r6,       PMDH[r0]
      mov     0x8000,   r6          -- PDH15 selected as control pin (INTPD15/PWM1)
      st.h    r6,       PMCDH[r0]
-- 32-bit data bus can also be used if BMODCN of PFCDH = 1.
--    mov     0x8000,   r6          -- PDH15 selected as PWM1 pin
      mov     0x0001,   r6          -- PDH15 to PDH0 used as D31 to D16 pins on starting in 16-bit mode
      st.h    r6,       PFCDH[r0]   -- Writing of port DH function control register
-- PDH15 to PDH0 set as D31 to D16 pins when BMODCN of PFCDH = 1, and setting of PDH, PMDH, and PMCDH is invalid.

_BMODCN_NOFIX:
      ld.h    PFCDH[r0],r6          -- Reading port DH function control register
      movea   0x0001,   r0, r7      -- BMODCN bit extraction
      and     r7,       r6
      cmp     r6,       r7          -- BMODCN setting not completed?
      jnz     _BMODCN_NOFIX         -- No
-- 32-bit bus mode when MODE1 pin = 0, MODE0 pin = 0, and above setting of PMDH, PMCDH, and PFCDH is invalid.

#########################################################
#     Specify bus control pin
#########################################################
      mov     0x0003,   r6          -- Selects control pin
      st.h    r6,       PMCAL[r0]
      mov     0x03,     r6          -- Selects A0 to A1 output pins
      st.b    r6,       PFCALL[r0]

      mov     0x03ff,   r6          -- Selects A16 to A25 output pins
      st.h    r6,       PMCAH[r0]

      mov     0xff,     r6          -- Sets inactive level (high) of CS0 to CS7 output signals
      st.b    r6,       PCS[r0]
      mov     0xff,     r6          -- Selects CS0 to CS7 pins
      st.b    r6,       PMCCS[r0]
```

```
    mov     0xbf,     r6          -- Sets inactive level (high) of BCYST, WE/WR, RD,
                                  -- UUWR/UUBE/UUDQM,
                                  -- ULWR/ULBE/ULDQM,
                                  -- LUWR/LUBE/LUDQM,
                                  -- LLWR/LLBE/LLDQM
    st.b    r6,       PCT[r0]     -- and LLWR/LLBE/LLDQM output signals
    mov     0xbf,     r6          -- Selects BCYST, ASTB, WE/WR, RD,
                                  -- UUWR/UUBE/UUDQM,
                                  -- ULWR/ULBE/ULDQM,
                                  -- LUWR/LUBE/LUDQM,
                                  -- and LLWR/LLBE/LLDQM output pins
    st.b    r6,       PMCCT[r0]

    mov     0xc0,     r6          -- PCM5 to PCM0 in output mode
    st.b    r6,       PMCM[r0]
    mov     0x00,     r6          -- All PCM5 to PCM0 as general-purpose output port pins
    st.b    r6,       PMCCM[r0]

    mov     0x0c,     r6          -- Sets inactive level (high) of SDRAS and SDCAS output signals
    st.b    r6,       PCD[r0]     -- Sets inactive level (low) of BUSCLK and SDCKE output signals
    mov     0x0f,     r6          -- Selects SDRAS, SDCAS, BUSCLK, SDCKE output pins
    st.b    r6,       PMCCD[r0]

    mov     0x00,     r6          -- Disables IORD and IOWR in SRAM, external ROM,
                                  -- and external I/O cycles
    st.b    r6,       BCP[r0]     -- Bus cycle period control register

#########################################################
#     Specify functions of pins other than bus control pins
#########################################################

# External interrupt rising edge specification register 1 (INTR1) at default value (00H)
# External interrupt falling edge specification register 1 (INTF1) at default value (00H)
                                  -- Input falling edge to INTP11 and INTP10 pins
# External interrupt rising edge specification register 2 (INTR2) at default value (00H)
# External interrupt falling edge specification register 2 (INTF2) at default value (00H)
                                  -- Input falling edge to INTP2n (n = 1 to 5), NMI pins
# External interrupt rising edge specification register 6 (INTR6) at default value (20H)
# External interrupt falling edge specification register 6 (INTF6) at default value (20H)
                                  -- Input both rising and falling edges to INPT65 pin

    st.b    r0,       P1[r0]      -- Sets inactive level (low) of TxD0 output signal
    mov     0x0f,     r6          -- All P13 to P10 are in control pin mode
    st.b    r6,       PMC1[r0]
    mov     0x0d,     r6          -- P13 = TxD0 output, P12 = RxD0 input
                                  -- P11 = INTP11 input, P10 = UCLK input
    st.b    r6,       PFC1[r0]
```

```
     mov    0x03,     r6          -- Inactive level (low) (extinguished) of P25 and P24 (LED)
     st.b   r6,       P2[r0]      -- Inactive level (low) of P23 (USB control) and TxD1 pins
     mov    0xc7,     r6
     st.b   r6,       PM2[r0]     -- P25 and P24 (LED) and P23 (USB control) in output mode
     mov    0x07,     r6
     st.b   r6,       PMC2[r0]    -- P25 and P23 as general-purpose port pins, and P22 to P20 as control pins
     mov    0x06,     r6
     st.b   r6,       PFC2[r0]    -- P22 = TxD1, P21 = RxD1, P20 = NMI

     mov    0x00,     r6          -- Inactive level (low) (extinguished) of P55 to P50 (LED)
     st.b   r6,       P5[r0]
     mov    0xc0,     r6
     st.b   r6,       PM5[r0]     -- P55 to P50 (LED) in output mode
     st.b   r0,       PMC5[r0]    -- All P55 to P50 as general-purpose port pins

     mov    0xff,     r6
     st.b   r6,       PM6[r0]     -- P67 and P66 (TOGLE SW) and P65 (USB power feed monitor) in input mode
     mov    0x20,     r6
     st.b   r6,       INTR6[r0]
     st.b   r6,       INTF6[r0]
     st.b   r6,       PMC6[r0]    -- P67 and P66 as general-purpose port pins, P65 = INTP65

     mov    0xff,     r6
     st.b   r6,       P7[r0]
     st.b   r6,       PM7[r0]     -- P77 to P72 (TOGLE SW) in input mode
     st.b   r0,       PMC7[r0]    -- All P77 to P72 as general-purpose port pins

#########################################################
# II check LOCK bit of lock register (LOCKR)
#########################################################
-- (0) Read lock register (LOCKR) to check stabilization of PLL
_UNLOCK:
     ld.b   LOCKR[r0],r6          -- Reads lock status of PLL from lock register (LOCKR)
     cmp    r0,       r6          -- Phase not locked yet?
     jnz    _UNLOCK               -- No

                                  -- To next procedure as PLL lock status has been checked.

-- (i) Set x7H to system wait control register (VSWC) (x: set value set in I).
     mov    0x37,     r6          -- Higher 4 bits and lower 4 bits of VSWC set value = 7H (fx = 133 MHz)
     st.b   r6,       VSWC[r0]    -- Writes to system wait control register

-- (ii) Set frequency division value of external bus to bus mode control register (BMC)
     mov    0x01,     r6          -- Division ratio of bus clock (BUSCLK) with respect to external
                                  -- system clock (fCLK) = fCLK/2.
     st.b   r6,       BMC[r0]     -- Writes to bus mode control register
```

```
-- (iii) Set value set in I to system wait control register (VSWC) again
      mov     0x33,     r6           -- Writes back values of higher and lower bits of VSWC set value (fx = 133 MHz)
      st.b    r6,       VSWC[r0]     -- Write is to system wait control register

-- (iv) Set internal system clock frequency division value to clock control register (CKC)
####   Write only in specific sequence   ####
      mov     0xa0,     r6           -- <1> Disable NMI
      ldsr    r6,       5            --      Set NP bit of PSW to 1 (to disable NMI)
      mov     0x03,     r6           -- <2> Prepare data to be set to CKC in general-purpose register
                                     --      Internal system clock in PLL mode, fCLK = fx
      st.b    r6,       PRCMD[r0]    -- <3> Write dummy data to command register
      st.b    r6,       CKC[r0]      -- <4> Set clock control register
      nop                            -- <5> Insert dummy NOP instruction (issue five instructions)
      nop                            -- <6> Insert dummy NOP instruction
      nop                            -- <7> Insert dummy NOP instruction
      nop                            -- <8> Insert dummy NOP instruction
      nop                            -- <9> Insert dummy NOP instruction
      mov     0x20,     r6           -- <10> Release NMI disable (NMI enabled)
      ldsr    r6,       5            --       Clear NP bit of PSW.
####   Write only in specific sequence   ####

-- (v) Set main clock (fx) supply control to clock source select register (CKS)
####   Write only in specific sequence   ####
      mov     0xa0,     r6           -- <1> Disable NMI
      ldsr    r6,       5            --      Set NP bit of PSW to 1 (to disable NMI)
      mov     0x01,     r6           -- <2> Prepare data to be set to CKS in general-purpose register
                                     --      Supply SSCG output clock (fx × 8) as main clock (fx)
      st.b    r6,       PRCMD[r0]    -- <3> Write dummy data to command register
      st.b    r6,       CKS[r0]      -- <4> Set clock source select register
      nop                            -- <5> Insert dummy NOP instruction (issue five instructions)
      nop                            -- <6> Insert dummy NOP instruction
      nop                            -- <7> Insert dummy NOP instruction
      nop                            -- <8> Insert dummy NOP instruction
      nop                            -- <9> Insert dummy NOP instruction
      mov     0x20,     r6           -- <10> Release NMI disable (NMI enable)
      ldsr    r6,       5            --       Clear NP bit of PSW.

####   Write only in specific sequence   ####

# SSCG control register (SSCGC) at default value (4xH: Refer below for x.)
####   Write only in specific sequence   ####
--  SMDL1 = 0, SMDL1 = 1: Modulation cycle of SSCG output 26 to 35 kHz
--  JIT1 pin   JIT0 pin   ADJON   ADJ2   ADJ1   ADJ0   Frequency modulation rate of SSCG output (Typ. value)
--      0          0         0      0      0      0    No modulation (frequency fixed)
--      0          1         1      0      0      1    -1.0%
--      1          0         1      0      1      1    -3.0%
--      1          1         1      1      0      1    -5.0%
####   Write only in specific sequence   ####
```

```
# USB clock control register (UCKC)
#     mov     0x80,     r6          -- Starts supplying clock to USB
#     st.b    r6,       UCKC[r0]

# Oscillation stabilization time select register (OSTS) at default value (04H)
-- OSTS0 = 0  Oscillation stabilization time after release of software STOP mode 21.84 ms (at fx = 12 MHz)

_UNLOCK2:
     ld.b    LOCKR[r0], r6         -- Reads lock status of PLL from lock register (LOCKR).
     cmp     r0,       r6          -- Phase not locked yet?
     jnz     _UNLOCK2              -- No

                                   -- PLL

#########################################################
#     Specify SDRAM configuration
#########################################################

###   Set refresh interval of CS1 (SDRAM, 32-bit width/32 MB) space   ##
     mov     0x801e,   r6          -- Enables refreshing (BUSCLK = 66 MHz).
                                   -- Refresh count clock = 32/BUSCLK
                                   -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
     st.h    r6,       RFS1[r0]    -- SDRAM refresh control register 1

###   Set refresh environment of CS1 (SDRAM, 32-bit width/32 MB) space   ##
     mov     0x20a5,   r6          -- CAS Latency 2 Latency (TLATE) during read
                                   -- 2 waits (TBCW) during ATC-RD, PRE-ACT
                                   -- Address shift width          2 bits (On-page)
                                   --                             (External data bus width: 32 bits)
                                   -- Row address width            12 bits
                                   -- Address multiplex width      9 bits
     st.h    r6,       SCR1[r0]    -- Writes to SDRAM configuration register 1
_SCR1NOFIX:
     ld.h    SCR1[r0], r6          -- Reads SDRAM configuration register 1
     movea   0x0100,   r0, r7      -- Extracts WCF1 bit
     and     r7,       r6
     cmp     r6,       r7          -- SCR1 setting incomplete?
     jnz     _SCR1NOFIX            -- Yes

###   Set refresh interval of CS4 (SDRAM, 16-bit width/16 MB) space   ##
     mov     0x801e,   r6          -- Enables refreshing (BUSCLK = 66 MHz).
                                   -- Refresh count clock = 32/BUSCLK
                                   -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
     st.h    r6,       RFS4[r0]    -- SDRAM refresh control register 4
```

```
###   Set refresh environment of CS4 (SDRAM, 16-bit width/16 MB) space   ##
      mov     0x2095,   r6          -- CAS Latency 2 Latency (TLATE) during read
                                    -- 2 waits (TBCW) during ATC-RD, PRE-ACT
                                    -- Address shift width          1 bit (On-page)
                                    --                              (External data bus width: 16 bits)
                                    -- Row address width            12 bits
                                    -- Address multiplex width      9 bits
      st.h    r6,       SCR4[r0]    -- Writes to SDRAM configuration register 4
_SCR4NOFIX:
      ld.h    SCR4[r0], r6          -- Reads SDRAM configuration register 4
      movea   0x0100,   r0, r7      -- Extracts WCF4 bit
      and     r7,       r6
      cmp     r6,       r7          -- SCR4 setting incomplete?
      jnz     _SCR4NOFIX            -- Yes

###   Set refresh interval of CS6 (SDRAM, 8-bit width/16 MB) space   ##
      mov     0x801e,   r6          -- Enables refreshing (BUSCLK = 66 MHz).
                                    -- Refresh count clock = 32/BUSCLK
                                    -- Refresh interval factor = 31 (15.0 us at 2.000 MHz)
      st.h    r6,       RFS6[r0]    -- SDRAM refresh control register 6

###   Set refresh environment of CS6 (SDRAM, 8-bit width/16 MB) space   ##
      mov     0x2086,   r6          -- CAS Latency 2 Latency (TLATE) during read
                                    -- 2 waits (TBCW) during ATC-RD, PRE-ACT
                                    -- Address shift width          0 bits (On-page)
                                    --                              (External data bus width: 8 bits)
                                    -- Row address width            12 bits
                                    -- Address multiplex width      10 bits
      st.h    r6,       SCR6[r0]    -- Writes to SDRAM configuration register 6
_SCR6NOFIX:
      ld.h    SCR6[r0], r6          -- Reads SDRAM configuration register 6
      movea   0x0100,   r0, r7      -- Extracts WCF6 bit
      and     r7,       r6
      cmp     r6,       r7          -- SCR6 setting incomplete?
      jnz     _SCR6NOFIX            -- Yes

      jmp     [lp]
```

```
 [ledtest.s]
     .section ".ledtest",text
     .align   4
     .globl   _ledtest

     .extern_ledtest_end

# Initialization index processing
# Dot LED binary shift
_ledtest:
     mov    0x01,     r11
     mov    0x8,      r12
_ledloop2:
     mov    r11,      r6
     jarl   _ledout,  lp
     movhi  0x0001,   r0,  r6
     movea  0x0000,   r6,  r6
     jarl   _softwait, lp

     shl    0x1,      r11
     add    -1,       r12
     cmp    r0,       r12
     bnz    _ledloop2
     jr     _ledtest_end

# Dot LED binary output processing
     .globl _ledout
_ledout:
     mov    r6,       r10
     shr    0x02,     r10
     and    0x3f,     r10
     st.b   r10,      P5
     mov    r6,       r10
     shl    0x04,     r10
     and    0x30,     r10
     st.b   r10,      P2

     jmp    [lp]

# Software idle loop processing
     .globl _softwait
_softwait:
     add    -1,       r6
     cmp    r0,       r6
     bnz    _softwait
     jmp    [lp]
```

```
 [printf.c]
#include <stdio.h>
#include <stdarg.h>

#include "uart.h"
#define TRUE  1
#define FALSE 0

/*  Global variable definition  */
static volatile unsigned int    TxCh1; /*                */

/*****************************************************************************
 *
 *  Function name:    init_uartb1
 *  Purpose:          Initialization of UARTB1
 *  Argument:         bps transfer rate (Refer to uart.h.)
 *  Return value:     None
 *
 *****************************************************************************/
void init_uartb1(int bps)
{
     int i;

     TxCh1 = FALSE;

     /*  Initialization of UARTB1-related pins  */
     P2     = 0x03;
     PM2   |= 0xc7;
     PMC2  |= 0x07;
     PFC2  |= 0x06;

     /*  Baud rate setting  */
     switch(bps){
     case UART_115200BPS:
            UB1CTL2 = UART_BAUD_115200;
            break;
     case UART_57600BPS:
            UB1CTL2 = UART_BAUD_57600;
            break;
     case UART_38400BPS:
            UB1CTL2 = UART_BAUD_38400;
            break;
     case UART_19200BPS:
            UB1CTL2 = UART_BAUD_19200;
            break;
     case UART_9600BPS:
            UB1CTL2 = UART_BAUD_9600;
            break;
```

```
    case UART_4800BPS:
            UB1CTL2 = UART_BAUD_4800;
            break;
    case UART_2400BPS:
            UB1CTL2 = UART_BAUD_2400;
            break;
    default:
            /* default 9600bps */
            UB1CTL2 = UART_BAUD_9600;
            break;
    }

    UB1CTL0  = UART_PWR_OFF;     /* Resets UARTB1 asynchronously. */
    for(i = 0 ; i < 10 ; i++);   /* Lapse of two or more cycles of fx/4  */
    UB1CTL0 |= UART_PWR_ON;      /* Supplies clock to UARTB1 */
    /* Transmission/reception enabled, LSB first, no parity, 1 stop bit, 8-bit length */
    UB1CTL0 |= (UART_TXE_ON | UART_RXE_ON | UART_CL_8);

}

/*****************************************************************************
 *
 *  Function name:    write_ch1
 *  Purpose:          1-byte transmission with UARTB1
 *  Argument:         Value for ch transmission
 *  Return value:     None
 *
 *****************************************************************************/
void write_ch1(unsigned char ch)
{
    volatile unsigned char stic1;
    volatile int  delay;
    /* Check vacancy of transmit register so that first data is completely transmitted. */
            stic1 = UTIC1;
    if(!TxCh1 && (stic1 & 0x80))
            UTIC1 = stic1 ^ 0x80;
    /* Wait for vacancy of transmit register */
    if (TxCh1){
            while (1){
              stic1 = UTIC1;
              if (stic1 & 0x80){
                  /* Clear transmission interrupt request */
                  UTIC1 = stic1 ^ 0x80;
                  TxCh1 = FALSE;
                  break;
              }
            }
    }
```

```
   /* Transmit data */
   if (!TxCh1){
          UB1TX = ch;
          TxCh1 = TRUE;
   }
}

/*****************************************************************************
 *
 * Function name:   write_str1
 * Purpose:         Transmits character string with UARTB1.
 * Argument:        Pointer to character string for str transmission (must be terminated with NULL)
 * Return value:    None
 *
 *****************************************************************************/
void write_str1( char *str)
{
   while (*str != (char)NULL){
          write_ch1(*str);
          str++;
   }
}
/*****************************************************************************
 *
 * Function name:   write_data1
 * Purpose:         Transmits data with UARTB1.
 * Argument:        Pointer to data for str transmission
 *                  len data length
 * Return value:    None
 *
 *****************************************************************************/
void write_data1(char *str, int len)
{
   int i;
   for (i = 0; i < len; i++){
          write_ch1((unsigned char)*str);
          str++;
   }
}
```

```
/************************************************************************
 *
 * Function name:    send_message
 * Purpose:          Transmits message with UARTB1.
 * Argument:         Pointer to str message character string (must be terminated with NULL)
 * Return value:     None
 *
 ************************************************************************/
void send_message(char *str)
{
  write_str1(str);
}

/************************************************************************
 *
 * Function:         printf
 * Purpose:          printf output with UARTB1
 * Argument:         Same as normal printf
 * Return value:     Same as normal printf
 *
 ************************************************************************/
int
printf(const char *fmt, ...)
{
  char buf[256];
  va_list ap;
  int ret = 0;

  /* Initialization of variable for argument list scanning */
  va_start(ap, fmt);

  /* Output formatted data to buffer */
  ret = vsprintf(buf, fmt, ap);

  /* End of argument list scanning */
  va_end(ap);

  /* Output contents of buffer via UARtB1 */
  send_message(buf);

  return ret;
}
```

[uart.h]

```
/*--------------------------------------------------------------------*/
/*  Transfer rate definition                                          */
/*--------------------------------------------------------------------*/
#define UART_115200BPS     115200
#define UART_57600BPS      57600
#define UART_38400BPS      38400
#define UART_19200BPS      19200
#define UART_9600BPS       9600
#define UART_4800BPS       4800
#define UART_2400BPS       2400

#if 0
/* UB0CTL1, UB0CTL2 set values fx = 133 MHz, Clock = fx/4 */
#define UART_BAUD_2400     6927       /* Error +0.001% */
#define UART_BAUD_4800     3464       /* Error -0.013% */
#define UART_BAUD_9600     1732       /* Error -0.013% */
#define UART_BAUD_19200    866        /* Error -0.013% */
#define UART_BAUD_38400    433        /* Error -0.013% */
#define UART_BAUD_57600    289        /* Error -0.128% */
#define UART_BAUD_115200   144        /* Error +0.218% */
#else
/* UB0CTL1, UB0CTL2 set values fx = 96 MHz, Clock = fx/4 */
#define UART_BAUD_2400     5000       /* Error -0.000% */
#define UART_BAUD_4800     2500       /* Error -0.000% */
#define UART_BAUD_9600     1250       /* Error -0.000% */
#define UART_BAUD_19200    625        /* Error -0.000% */
#define UART_BAUD_38400    313        /* Error -0.160% */
#define UART_BAUD_57600    208        /* Error +0.160% */
#define UART_BAUD_115200   104        /* Error +0.160% */
#endif

/*--------------------------------------------------------------------*/
/* UARTBn control register 0 (UBnCTL0) bit definition                 */
/*--------------------------------------------------------------------*/
#define UART_PWR_ON    (1 << 7) /* Stops clock supply to UART */
#define UART_PWR_OFF   (0 << 7) /* Starts clock supply to UART */

#define UART_TXE_ON    (1 << 6) /* Enables transmission */
#define UART_TXE_OFF   (0 << 6) /* Disables transmission */

#define UART_RXE_ON    (1 << 5) /* Enables reception */
#define UART_RXE_OFF   (0 << 5) /* Disables reception */

#define UART_DIR_LSB   (1 << 4) /* First bit of transfer data is LSB */
#define UART_DIR_MSB   (0 << 4) /* First bit of transfer data is MSB */
```

```
#define UART_PS_NO    (0 << 3) | (0 << 2)    /* No parity */
#define UART_PS_0     (0 << 2) | (1 << 2)    /* 0 parity */
#define UART_PS_ODD   (1 << 3) | (0 << 2)    /* Odd parity */
#define UART_PS_EVN   (1 << 3) | (1 << 2)    /* Even parity */

#define UART_CL_8     (1 << 1) /* One transfer data frame is 8 bits long */
#define UART_CL_7     (0 << 1) /* One transfer data frame is 7 bits long */

#define UART_SL_2     (1 << 0) /* Transfer data stop bit is 2 bits long */
#define UART_SL_1     (0 << 0) /* Transfer data stop bit is 1 bits long */
```

[crtE.s]

#(Copyright indication omitted)

```
#------------------------------------------------------------------------------
#       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

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

#------------------------------------------------------------------------------
#   for argv (not used, So deleted data definitions)
#------------------------------------------------------------------------------

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

#------------------------------------------------------------------------------
#       system stack
#------------------------------------------------------------------------------
     .set     STACKSIZE, 0x200
     .globl   _ _stack
     .bss
     .lcomm   _ _stack,      STACKSIZE,4


#------------------------------------------------------------------------------
#       RESET handler
#------------------------------------------------------------------------------
     .section "RESET", text
     jr _ _start
```

```
#------------------------------------------------------------------------------
#       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
#------------------------------------------------------------------------------
.section ".crtE",text
     .align  4
     .globl  _ _start
     .globl  _ _exit
     .globl  _ _startend
#    .extern _ _PROLOG_TABLE
_ _start:
     jarl    _initial,lp              -- V850E/ME2 CPU initial
_initial_end:

     jr      _ledtest
     .globl  _ledtest_end
_ledtest_end:

     mov     #_ _tp_TEXT, tp          -- Sets tp register
     mov     #_ _gp_DATA, gp          -- Sets gp register offset
     add     tp,      gp              -- Sets gp register
     mov     #_ _stack+STACKSIZE, sp  -- Sets sp register
     mov     #_ _ep_DATA, ep          -- Sets ep register
#
     .option nowarning
     mov     0xff,    r20               -- Sets mask register
     mov     0xffff,  r21               -- Sets mask register
     .option warning
#
     mov     #_ _ssbss, r13             -- Clears 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              -- Clears bss section
     mov     #_ _ebss, r12
     cmp     r12,     r13
     jnl     .L14
```

```
.L15:
     st.w    r0,      [r13]
     add     4,       r13
     cmp     r12,     r13
     jl      .L15
.L14:

#########################################################
# Transfer program code to internal instruction RAM
#########################################################
     .extern _ _S_rompack
     .extern _ _S_appli1rom
     .extern _ _ircopy
     .globl  _iramboot
_iramboot:
-- (0) Transfer program to internal instruction RAM.
.option nowarning
     movhi   hi1(_ _S_rompack), tp, r1        -- Adds offset of tp
     movea   lo(_ _S_rompack), r1, r6         -- Static program packing section first address
.option warning
     mov     -0x1,    r7                      -- All packing sections to be transferred transfer
     jarl    _ _ircopy,r31                    -- ROMized dynamic program to bank 0

.extern _ _S_appli1rom
.option nowarning
     movhi   hi1(_ _S_appli1rom), tp, r1      -- Adds offset of tp
     movea   lo(_ _S_appli1rom), r1, r6       -- Dynamic program 1 packing section first address
.option warning
     mov     -0x1,    r7                      -- All packing sections to be transferred transfer
     jarl    _ _ircopy,r31                    -- ROMized dynamic program to bank 1

                                              -- Internal instruction RAM is set in read mode
                                              -- as program code has been transferred.

-- (i) Set internal instruction RAM mode register (IRAMM) in read mode (IRAMM0, IRAMM1 = 0)
     mov     0x00,    r6                      -- Bank 0 0000000 to 00FFFF (64 KB) read mode
                                              -- Bank 0 0010000 to 01FFFF (64 KB) read mode
     st.b    r6,      IRAMM[r0]               -- Internal instruction RAM mode register

-- (ii) Check that internal instruction RAM mode register (IRAMM) changed to read mode (IRAMM0 = 1?)
_iram_rd_chk:
     ld.b    IRAMM[r0], r6                    -- Reads internal instruction RAM mode register
     cmp     r0,      r6                      -- Banks 0 and 1 in read mode?
     bnz     _iram_rd_chk                     -- No
```

```
-- (iii) To internal instruction RAM by branch instruction
.option nowarning
     mov     #_main, r1
     jmp     [r1]
.option warning
_ _exit:
     halt                                        -- End of program
_ _startend:
     nop
     nop
#                                                                    #
#-------------------- end of start up module --------------------#
#                                                                    #
```

```
 [secentry.s]
     .file   "secentry.s"

# RAM (bank 1) area for interface instructions transfer from external EPROM area for dynamically extended program.
     .section ".dymcappli",text
     .align  4
     .globl  _ _S_dymcappli, 4
_ _S_dymcappli:

# External EPROM area 1 storing program 1 to be dynamically executed
     .section ".appli1rom",text
     .align  4
     .globl  _ _S_appli1rom, 4
_ _S_appli1rom:

# External EPROM area 2 storing program 2 to be dynamically executed
     .section ".appli2rom",text
     .align  4
     .globl  _ _S_appli2rom, 4
_ _S_appli2rom:

# External EPROM area 3 storing program 3 to be dynamically executed
     .section ".appli3rom",text
     .align  4
     .globl  _ _S_appli3rom, 4
_ _S_appli3rom:
```

```
 [rompack.s]
     .file   "rompack.s"
     .section ".s_romp",text
     .align  4
     .globl  _ _S_rompack, 4
_ _S_rompack:
```

```
 [progmain.dir]
# (Copyright indication omitted)
# Segment area for program to be statically executed
# Internal instruction RAM area   bank 0   0x00001000 to 0x0000ffff (60 KB)
MAINTEXT    : !LOAD ?RWX V0x00001000{
     .text         = $PROGBITS    ?AX .text;
};

# CONST segment area of program to be statically executed
# Internal instruction RAM area   bank 0   0x00001000 to 0x0000ffff (60 KB)
CONST    : !LOAD ?R V0x0000f000 {
     .const        = $PROGBITS    ?A .const;
};

# Segment area for program to be dynamically executed
# Internal instruction RAM area   bank 1   0x00010000 to 0x0001ffff (64 KB)
DYMCAPPL  : !LOAD ?RWX V0x00014000 {
     .dymcappli    = $PROGBITS    ?AX A0x4 .dymcappli;
};

# CPU RESET segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
RESET    : !LOAD ?RX V0x00100000 {
     RESET         = $PROGBITS    ?AX RESET0;
};


# Startup module segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
CRTE        : !LOAD ?RX V0x00101000 {
     .crtE          = $PROGBITS    ?AX .crtE;
};

# Internal instruction RAM area transfer program segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
IRCOPY    : !LOAD ?RX  {
     .ircopy       = $PROGBITS     ?AX .ircopy;
};

# CPU initialization program segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
INITIAL : !LOAD ?RX {
     .initial      = $PROGBITS     ?AX .initial;
};

# LED lighting program segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
LEDTEST : !LOAD ?RX {
     .ledtest      = $PROGBITS     ?AX .ledtest;
};
```

```
# rompsec section segment area
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
SROMP         : !LOAD ?RX  {
     .s_romp        = $PROGBITS  ?AX .s_romp;
};

# Vacant segment area storing program 1 to be dynamically executed
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
APPLI1ROM    : !LOAD ?RX V0x00114000 {
     .appli1rom    = $PROGBITS   ?AX A0x4 .appli1rom;
};

# Vacant segment area storing program 2 to be dynamically executed
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
APPLI2ROM    : !LOAD ?RX V0x00118000 {
     .appli2rom    = $PROGBITS   ?AX A0x4 .appli2rom;
};

# Vacant segment area storing program 3 to be dynamically executed
# External EPROM (16-bit width) area   0x100000 to 0x11ffff (128 KB)
APPLI3ROM    : !LOAD ?RX V0x0011c000 {
     .appli3rom    = $PROGBITS   ?AX A0x4 .appli3rom;
};

# Internal data RAM segment area   0x0fffb000 to 0x00fffefff (20 KB)--V850E/ME2
# Internal data RAM reserved area 0xfff8000 to 0xfffafff (12 KB) (access prohibited)
# internal data RAM allocated area 0xfffb000 to 0xfffcfff (8 KB) (reserved for ROM Monitor)
DATA  : !LOAD ?RW V0x0fffd000 {
     .data            = $PROGBITS  ?AW  .data;
     .sdata           = $PROGBITS  ?AWG .sdata;
     .sbss            = $NOBITS    ?AWG .sbss;
     .bss             = $NOBITS    ?AW  .bss;
};

_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT{DATA};
_ _ep_DATA @ %EP_SYMBOL;

[ircopy.s]
```

Refer to **4.2.8 (6) Program example**.

(b) Dynamic program list

A list of source files of the dynamic program list and a list of the dynamic programs are shown below.

Caution The dynamic program list shows only dynamic program 1. The differences between this program and the other programs are the digits of the rotating reel in the main function, the processing to stop the reel in the INT SW interrupt, and the link address when the program is stored in ROM. For details, refer to 4.2.10 (1) Execution of dynamic program sample.

Table 4-12. Source Files of Dynamic Program List

File Name	Outline
appli1.c	wait function, main function, intsw_for_app1 function, seg7led function, dotled function
secentry.s	Vacant link entry
printf.c	UARTB1 output processing function group (same as static program)
uart.h	UARTB-related header file (same as static program)
appli1.h	Header file for dynamic program
crtE.s	Startup module processing for dynamic program
progapp1.dir	Link directive for dynamic program

```
[appli1.c]
#include "appli1.h"
#pragma ioreg   /* Declaration of use of internal I/O */

/* External declaration of address name of external EPROM area storing program to be dynamically executed */
extern void _S_app1entry();

/* Global variable */
unsigned char place;              /* Specification of rotating reel display position */
unsigned char flag;               /* Interrupt processing toggle flag */

/* Prototype declaration */
void intsw_for_app1(void);
void port_init(void);
void dotled(int n, int op);
/************************************************************************
 *
 *   Function name:  wait
 *   Purpose:        Idle loop (idle by defined number of wait states)
 *   Argument:       None
 *   Return value:   None
 *
 ************************************************************************/
void wait(void)
{
  int a=0;
  while(a!=10000)
  {
    a=a+1;
  }
}

/************************************************************************
 *
 *   Function name:  main
 *   Purpose:        Dynamic program 1 main processing
 *   Argument:       None
 *   Return value:   None
 *
 ************************************************************************/
void main(void){
     int i;
     int bcd02 = 0,bcd01 = 0;     /* Rotating reel variable */

     init_uartb1(19200);          /* UARTB1 channel initialization */
     port_init();                 /* Port initialization */
```

```
    /* INTP11 interrupt control register setting */
    _ _set_il(0, "INTP11");        /* INTP11 interrupt unmasking */
    _ _set_il(3, "INTP11");        /* INTP11 interrupt priority 4 */

    /* Set internal instruction RAM bank 0 (0x00000000 to 0x0000ffff) in write mode. */
    IRAMM = 0x01;/* Sets internal instruction RAM mode register (IRAMM) in write mode (IRAMM0 = 1). */
    do{
     ; /* Check that internal instruction RAM mode register (IRAMM) changed to write mode (IRAMM0 = 1?). */
    }while(IRAMM != 1);

    for(i = 1 ; i <= USEINTMAX ; i++){
    _ircopy(_S_app1entry,i); /* Writing interrupt handler used for dynamic program */
    }

    /* Set internal instruction RAM bank 0 (0x00000000 to 0x0000ffff) in read mode */
    IRAMM = 0x00;/* Sets internal instruction RAM mode register (IRAMM) in read mode (IRAMM0 = 1) */
    do{
     ; /* Check that internal instruction RAM mode register (IRAMM) changed to read mode */
    }while(IRAMM != 0);

    /* Enable interrupts */
    _ _EI();

    printf("\r\nRunning application No.1\r\n"); /* Displays ID number of dynamic program */
    for(;;){
       wait();
       if((place & DIGIT1) == GO)bcd01++;       /* Rotation of specified digit continues? */
       if(bcd01 > 9)bcd01=0;                    /* Number of rotating reel loops from 0 to 9 */
       if((place & DIGIT2) == GO)bcd02++;       /* Rotation of specified digit continues? */
       if(bcd02 > 9)bcd02=0;                    /* Number of rotating reel loops from 0 to 9 */
       /* Two of eight digits rotate */
       printf("\r<pseudo 7segLED>=000000%d%d",bcd02,bcd01);
    }
}

/************************************************************************
 *
 *  Function name:  port_init
 *  Purpose:        General-purpose port initialization processing
 *  Argument:       None
 *  Return value:   None
 *
 ************************************************************************/
void port_init (void){
    /* Initialization of pins for INTSW */
    PMC1 = 0x0f;       /* All P13 to P10 in control pin mode */
    PFC1 = 0x0d;       /* P13 = TxD0, P12 = RxD0, P11 = INTP11, P10 = UCLK input */
```

```
    /* Port initialization for dot LED */
    P2    = 0x03;                   /* Extinguishes LED1 and LED2 (P25 and P24). */
                                    /* Sets P23 (USB control) and TxD1 pins to inactive level (low) */
    PM2  |= 0xc7;                   /* P25 and P24 = Dot LED, P23 (USB control) in output mode */
    PMC2 |= 0x07;                   /* P25 to P23 in I/O mode */
    PFC2 |= 0x06;                   /* P22 = TxD1, P21 = RxD1, P20 = NMI */

    P5    = 0x00;                   /* Extinguishes LED3 and LED8 (P55 to P50). */
    PM5  |= 0xc0;                   /* P55 to P50 = Dot LED in output mode */
    PMC5 |= 0x00;                   /* P55 to P50 in I/O mode */

    /* TOGLE SW */
    PM6  |= 0xff;                   /* P67 to P66 = TOGLE SW, P65 (USB power feed monitor) in input mode */
    PMC6 |= 0x20;                   /* P67 to P66 in I/O mode, P65 = INTP65 */

    PM7  |= 0xff;                   /* P77 to P72 = TOGLE SW in input mode */
    PMC7 |= 0x00;                   /* P77 to P72 in I/O mode */

}

/*****************************************************************************
 *
 *  Function name:  seg7led
 *  Purpose:        Rotating reel manipulation processing
 *  Argument:       *p   Pointer to reel manipulation digit position variable
 *                  op   Stops/operates reel
 *  Return value:   None
 *
 *****************************************************************************/
void seg7led( unsigned char *p, int n, int op ){
    if(op == STOP){
*p |=  n;                           /* Stops display of digit n of rotating reel */
    }else{
*p &= ~n;                           /* Moves display of digit n of rotating reel */
    }
}
```

```
/**** Dot LED manipulation processing ****/
/*************************************************************************
 *
 *   Function name:  dotled
 *   Purpose:        Rotating reel manipulation processing
 *   Argument:       pattern   Position of dot LED
 *                   op        Lights/extinguishes dot LED
 *   Return value:   None
 *
 *************************************************************************/
void dotled(int pattern, int op)
{
     if(op == ALLON){
        DOTLED2_1 = ((DOTLED2 | DOTLED1) << 4) & 0x30;
        DOTLED8_3 = ((DOTLED8 | DOTLED7 | DOTLED6 | DOTLED5 |
                        DOTLED4 | DOTLED3 & 0xfc) >> 2) & 0x3f;
     }else if(op == ON){
        DOTLED2_1 =  ((pattern & 0x03) << 4) & 0x30;
        DOTLED8_3 =  ((pattern & 0xfc) >> 2) & 0x3f;
     }else if(op == OFF){
        DOTLED2_1 = ~((pattern & 0x03) << 4) & 0x30;
        DOTLED8_3 = ~((pattern & 0xfc) >> 2) & 0x3f;
     }else if(op == ALLOFF){
        DOTLED2_1 = ~((DOTLED2 | DOTLED1) << 4) & 0x30;
        DOTLED8_3 = ~((DOTLED8 | DOTLED7 | DOTLED6 | DOTLED5 |
                        DOTLED4 | DOTLED3 & 0xfc) >> 2) & 0x3f;
     }
}

/*************************************************************************
 *
 *   Interrupt request name:    INTP11
 *   Purpose:                   INTSW interrupt processing that is executed in dynamic program
 *
 *************************************************************************/
#pragma interrupt INTP11 intsw_for_app1
_ _interrupt
void intsw_for_app1(void){
     if(flag ^= ALTERNATIVE){  /* Flag is alternately cleared to 0 or set to 1 each time interrupt occurs */
        /* Stop first digit of rotating reel and rotate second digit */
        seg7led(&place, DIGIT1, STOP);  /* Stops display of first digit of rotating reel */
        seg7led(&place, DIGIT2, GO);    /* Displays second digit of rotating reel */
        dotled(DOTLED1, ALLOFF);        /* Extinguishes all dot LEDs */
        dotled(DOTLED1, ON);            /* Lights dot LED1 */
```

```
    }else{
        /* Stop second digit of rotating reel and rotate first reel */
        seg7led(&place, DIGIT2, STOP);   /* Stops display of second digit of rotating reel */
        seg7led(&place, DIGIT1, GO);     /* Displays first digit of rotating reel */
        dotled(DOTLED2, ALLOFF);         /* Extinguishes all dot LEDs */
        dotled(DOTLED2, ON);             /* Lights dot LED2 */
    }
}
```

```
[secentry.s]
    .file   "secentry.s"
    .section ".app1entry",text
    .align  4
    .globl  _ _S_app1entry, 4
_ _S_app1entry:
```

```
[printf.c]
```

Same as static program

```
[uart.h]
```

Same as static program

```
 [appli1.h]
/* Constant definition */
#define USEINTMAX      1          /* Maximum value of interrupt used for dynamic program */
#define HIGH 1                    /* High level */
#define LOW 0                     /* Low level */
#define ACTIVE_LEVEL HIGH         /* Selects active level */
#define ALTERNATIVE 0x01          /* Toggle flag used for interrupt */
#define DIGIT1 0x01               /* Position of first digit of rotating reel */
#define DIGIT2 0x02               /* Position of second digit of rotating reel */
#define DIGIT3 0x04               /* Position of third digit of rotating reel */
#define DIGIT4 0x08               /* Position of 4th digit of rotating reel */
#define DIGIT5 0x10               /* Position of 5th digit of rotating reel */
#define DIGIT6 0x20               /* Position of 6th digit of rotating reel */
#define DIGIT7 0x40               /* Position of 7th digit of rotating reel */
#define DIGIT8 0x80               /* Position of 8th digit of rotating reel */

#define DOTLED1 0x01              /* Position of first dot LED */
#define DOTLED2 0x02              /* Position of second dot LED */
#define DOTLED3 0x04              /* Position of third dot LED */
#define DOTLED4 0x08              /* Position of 4th dot LED */
#define DOTLED5 0x10              /* Position of 5th dot LED */
#define DOTLED6 0x20              /* Position of 6th dot LED */
#define DOTLED7 0x40              /* Position of 7th dot LED */
#define DOTLED8 0x80              /* Position of 8th dot LED */

#define DOTLED8_3 P5              /* Port assigned to digits 8 to 3 of dot LED */
#define DOTLED2_1 P2              /* Port assigned to digits 2 to 1 of dot LED */

#define STOP 1                    /* Stops rotating reel */
#define GO 0                      /* Rotates rotating reel */

#define ALLON 3                   /* Lights all LEDs */
#define ON 1                      /* Extinguishes LED */
#define OFF 0                     /* Lights all LEDs */
#define ALLOFF -1                 /* Extinguishes LED */

#define HANDLER_INTSW 0x090       /* NMI handler address */
```

[crtE.s]

#(Copyright indication omitted)

```
#-------------------------------------------------------------------------------
#   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

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

#-------------------------------------------------------------------------------
#   for argv (not used, So deleted data definitions)
#-------------------------------------------------------------------------------

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

#-------------------------------------------------------------------------------
#   system stack
#-------------------------------------------------------------------------------
     .set   STACKSIZE, 0x200
     .bss
     .lcomm _ _stack, STACKSIZE, 4
```

```
#------------------------------------------------------------------------------
#   RESET handler(not used,So deleted codes)
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#       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  _ _startend
_ _start:
     .option nowarning
     movhi   hi1(#_ _tp_TEXT), r0, r1          -- Sets tp register
     movea   lo(#_ _tp_TEXT), r1, tp           -- Sets tp register
     movhi   hi1(#_ _gp_DATA), r0, r1          -- Sets gp register offset
     movea   lo(#_ _gp_DATA), r1, gp           -- Sets gp register offset
     .option warning
     add     tp, gp                            -- Sets gp register
     .option nowarning
     movhi   hi1(#_ _stack+STACKSIZE),r0, r1   -- Sets sp register
     movea   lo(#_ _stack+STACKSIZE), r1, sp   -- Sets sp register
     movhi   hi1(#_ _ep_DATA), r0, r1          -- Sets ep register
     movea   lo(#_ _ep_DATA), r1, ep           -- Sets ep register
     .option warning
#
     .option nowarning
     mov     0xff,     r20                     -- Sets mask register
     mov     0xffff,   r21                     -- Sets mask register
     .option warning
#
     mov #_ _ssbss, r13                        -- Clears 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                         -- Clears bss section
```

```
     mov #_ _ebss, r12
     cmp r12, r13
     jnl .L14
.L15:
     st.w    r0,        [r13]
     add     4,         r13
     cmp     r12,       r13
     jl      .L15
.L14:

     jarl    _main,     lp                      -- Calls main function
_ _exit:
     halt                                       -- End of program
_ _startend:
             nop
             nop
#                                                                    #
#-------------------- end of start up module -------------------#
#                                                                    #
```

```
 [progapp1.dir]
#（Copyright indication omitted）

# Internal instruction RAM bank 1 space  0x00014000 to 0x00016fff (12 KB)
# Dynamic program group execution area
TEXT        : !LOAD ?RWX V0x00014000 {
     .text         = $PROGBITS    ?AX .text;
};

# Internal instruction RAM bank 1 space  0x00017000 to 0x00017fff (4 KB)
# Dynamically executed program constant data storage area
CONST1      : !LOAD ?R   V0x00017000 {
     .const        = $PROGBITS    ?A .const;
};

# External EPROM (16-bit width) space  0x00114000 to 0x0016ffff (12 KB)
# Dynamically executed program 1 storage area
APPLI1ENTRY       : !LOAD ?RX V0x00114000 {
     .app1entry    = $PROGBITS    ?AX .app1entry;
};

<R>  # Internal data RAM space  0x0fffb000 to 0x00fffefff (20 KB)--V850E/ME2 uPD703111A
     # Internal data RAM space  0x0fffb000 to 0x00fffcfff (8 KB)--V850E/ME2 ROM Monitor  Reserved area
     DATA    : !LOAD ?RW V0x0fffd800 {
          .data         = $PROGBITS    ?AW .data;
          .sdata        = $PROGBITS    ?AWG .sdata;
          .sbss         = $NOBITS      ?AWG .sbss;
          .bss          = $NOBITS      ?AW .bss;
     };

     _ _tp_TEXT @ %TP_SYMBOL;
     _ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT{DATA};
     _ _ep_DATA @ %EP_SYMBOL;
```

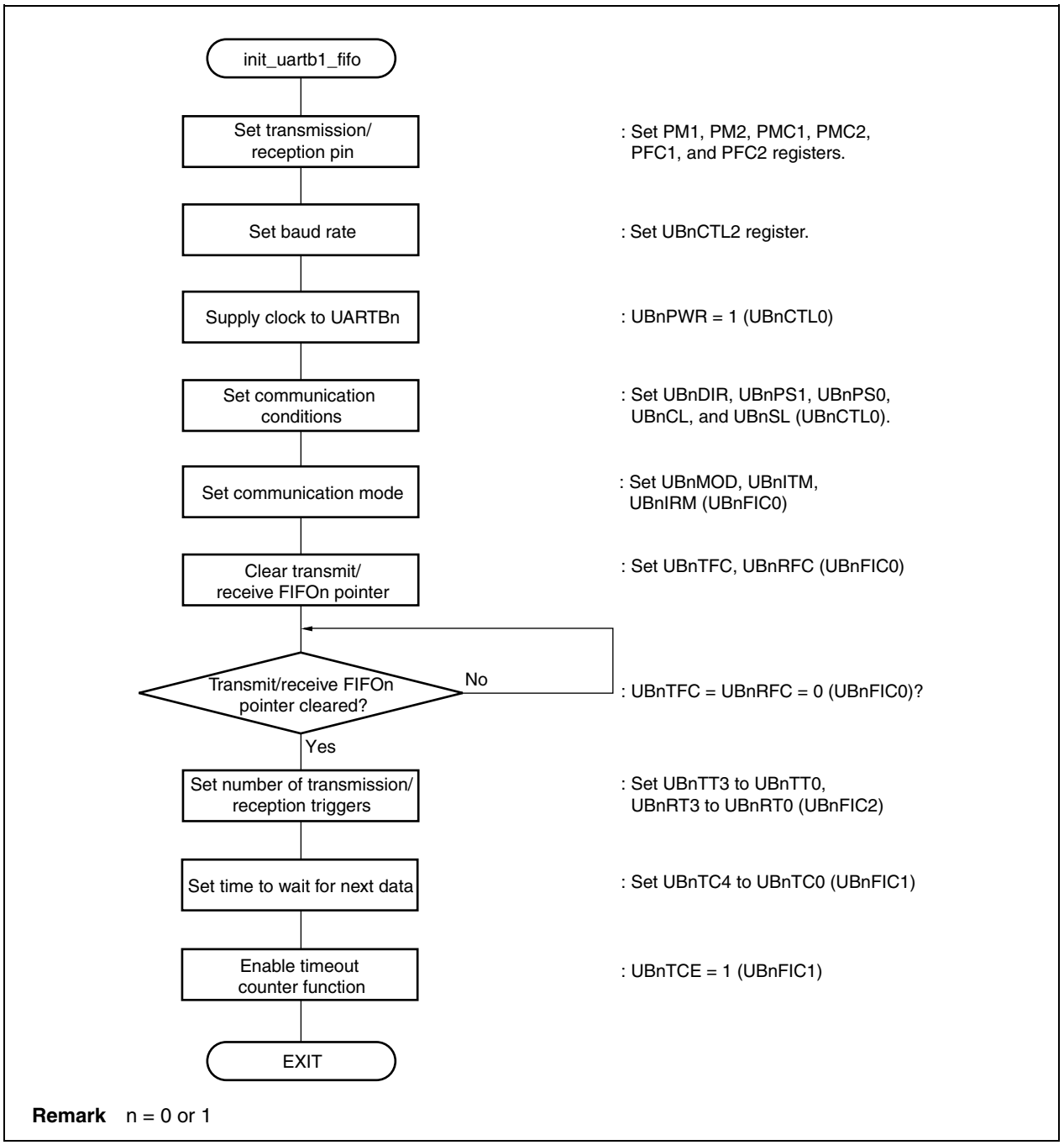
4.2.11 Example of FIFO program of internal UARTBn

This section explains an example of a program that transmits/receives data in the FIFO mode of the internal UARTBn (n = 0 or 1). In this program example, data is successively received from an external source by using a 256-byte receive ring buffer, while a fixed character string is repeatedly transmitted.

(1) Flowcharts

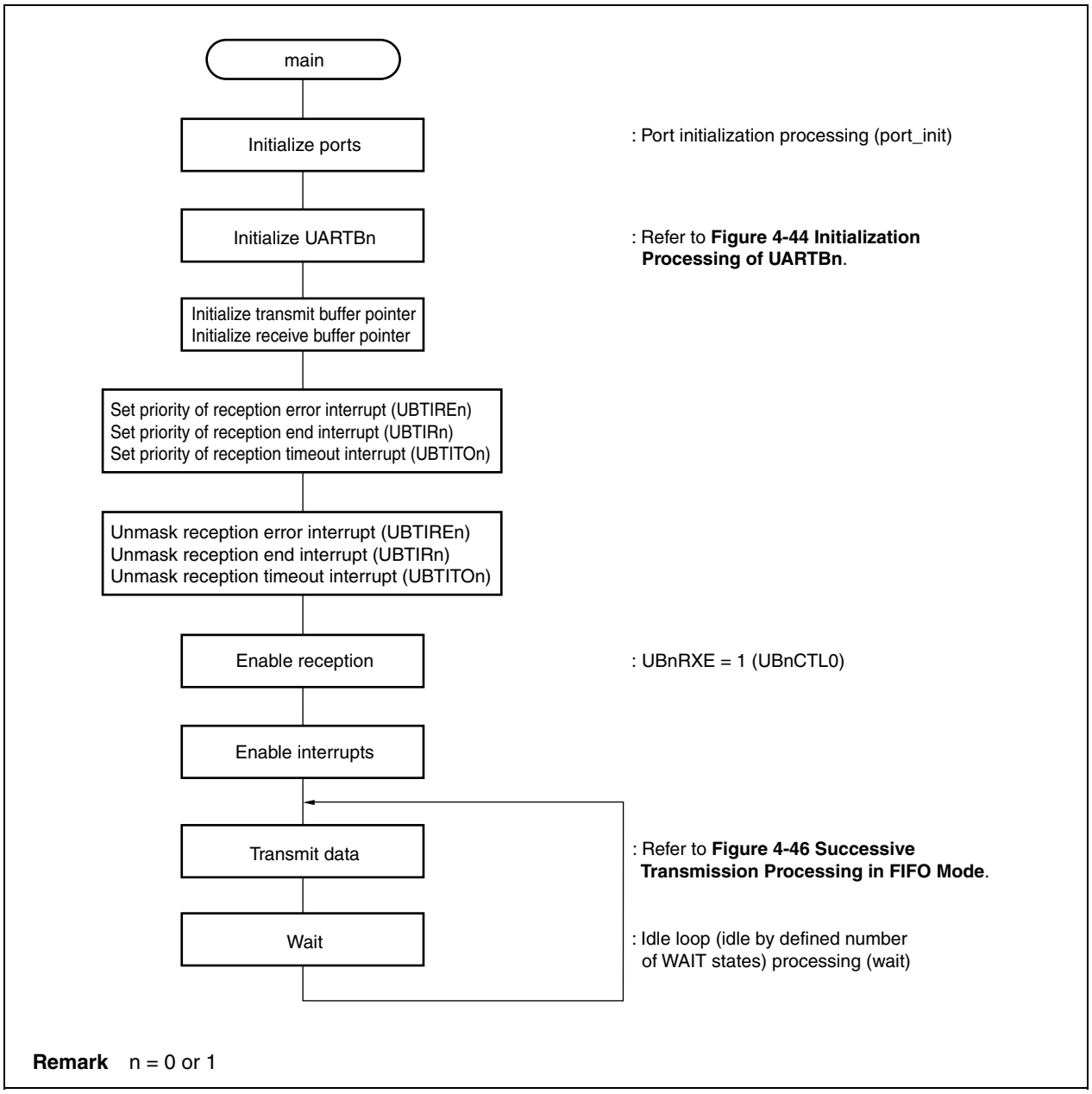
The flowcharts are shown below.

<R> **Figure 4-44. Initialization Processing of UARTBn**



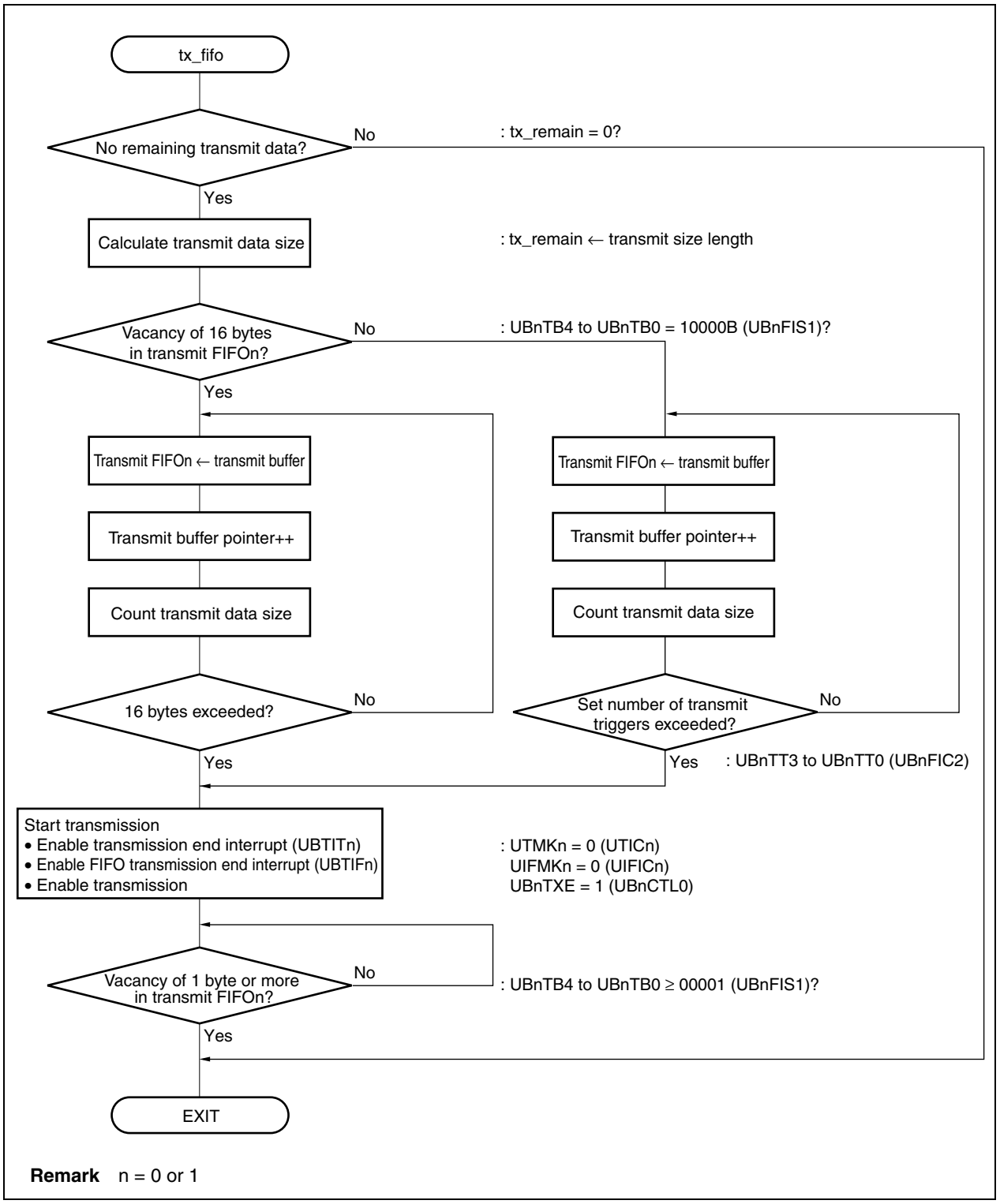
Remark n = 0 or 1

Figure 4-45. Main Processing



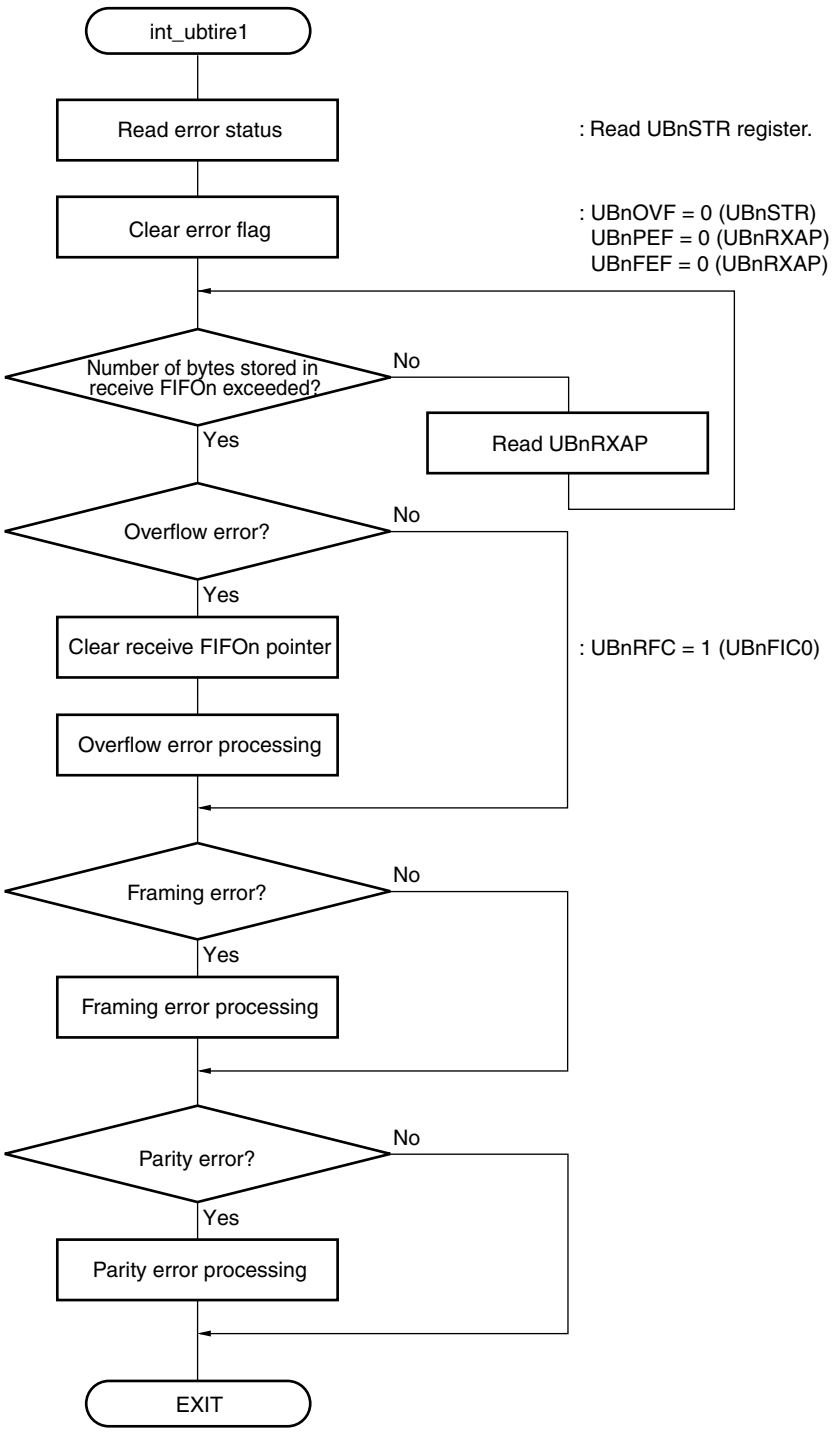
Remark n = 0 or 1

Figure 4-46. Successive Transmission Processing in FIFO Mode



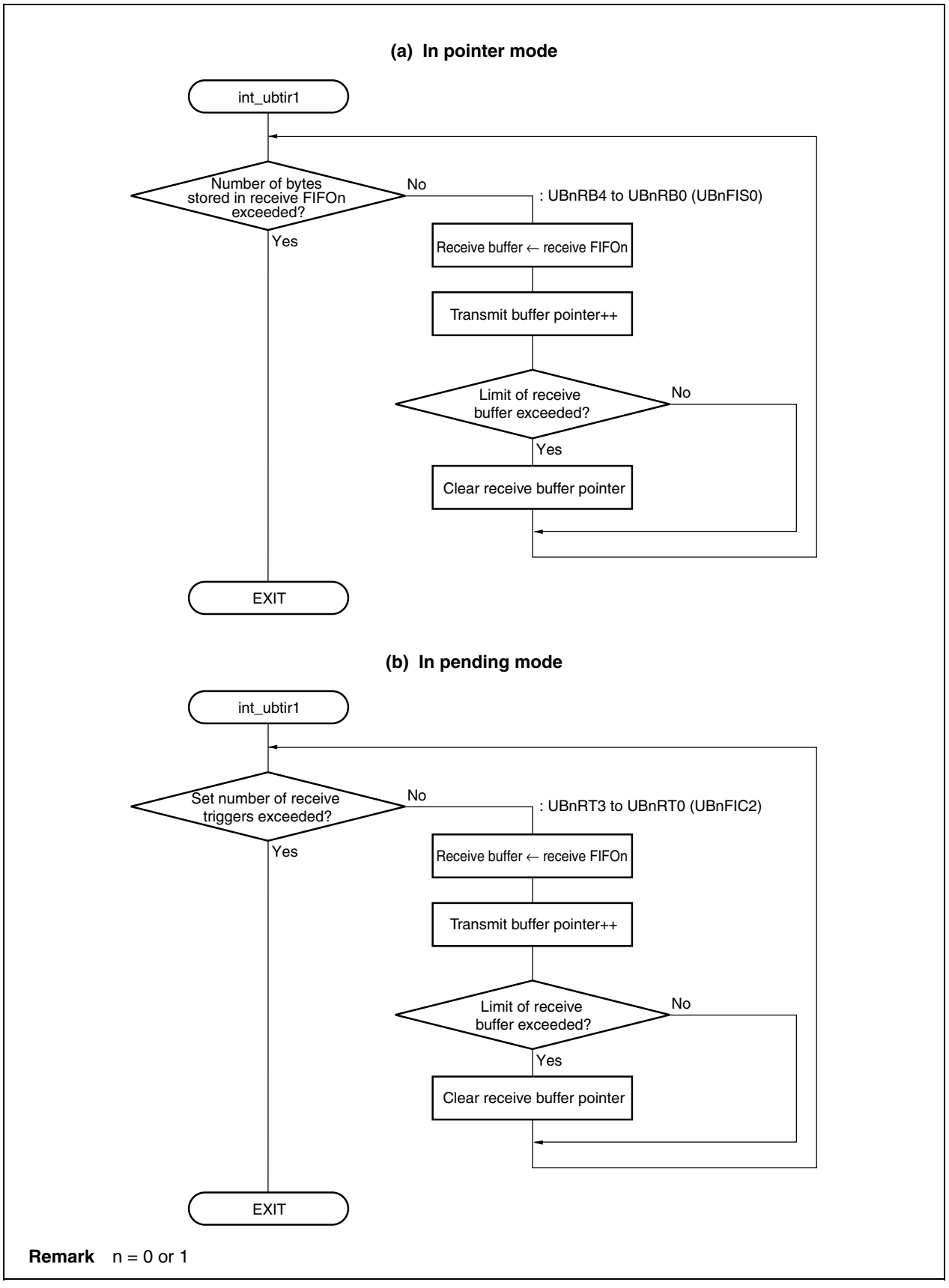
Remark n = 0 or 1

Figure 4-47. Reception Error Interrupt (UBTIREn) Processing



Remark n = 0 or 1

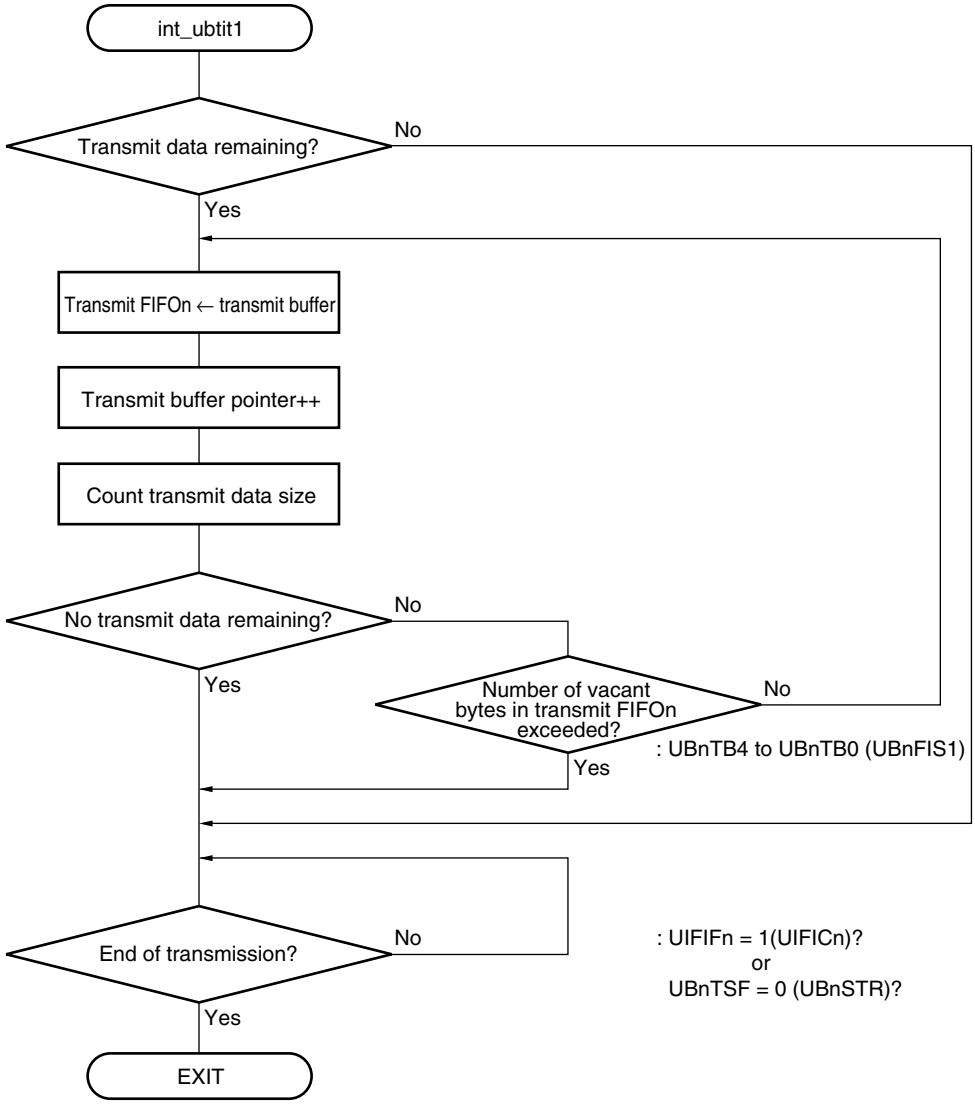
Figure 4-48. Reception End Interrupt (UBTIRn) Processing



Remark n = 0 or 1

Figure 4-49. Transmission End Interrupt (UBTITn) Processing (1/2)

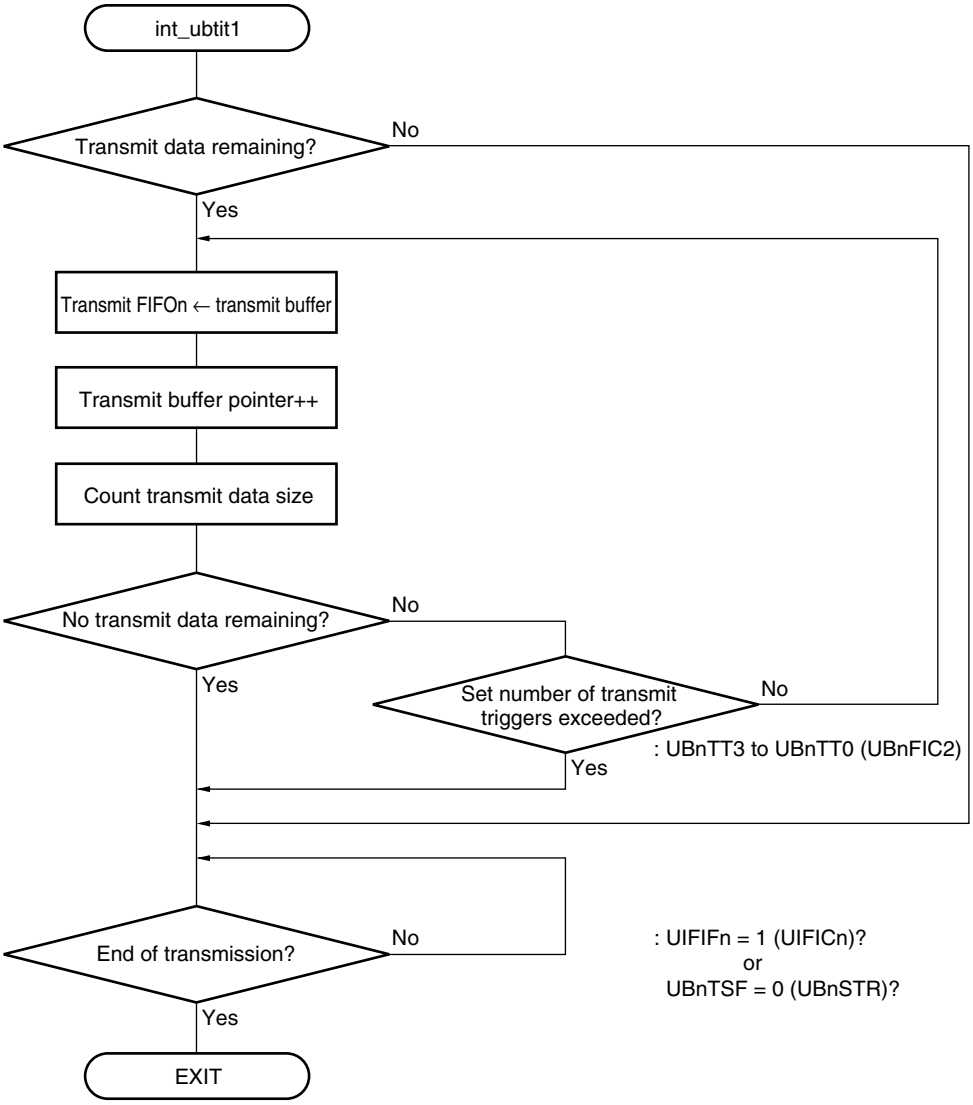
(a) In pointer mode



Remark n = 0 or 1

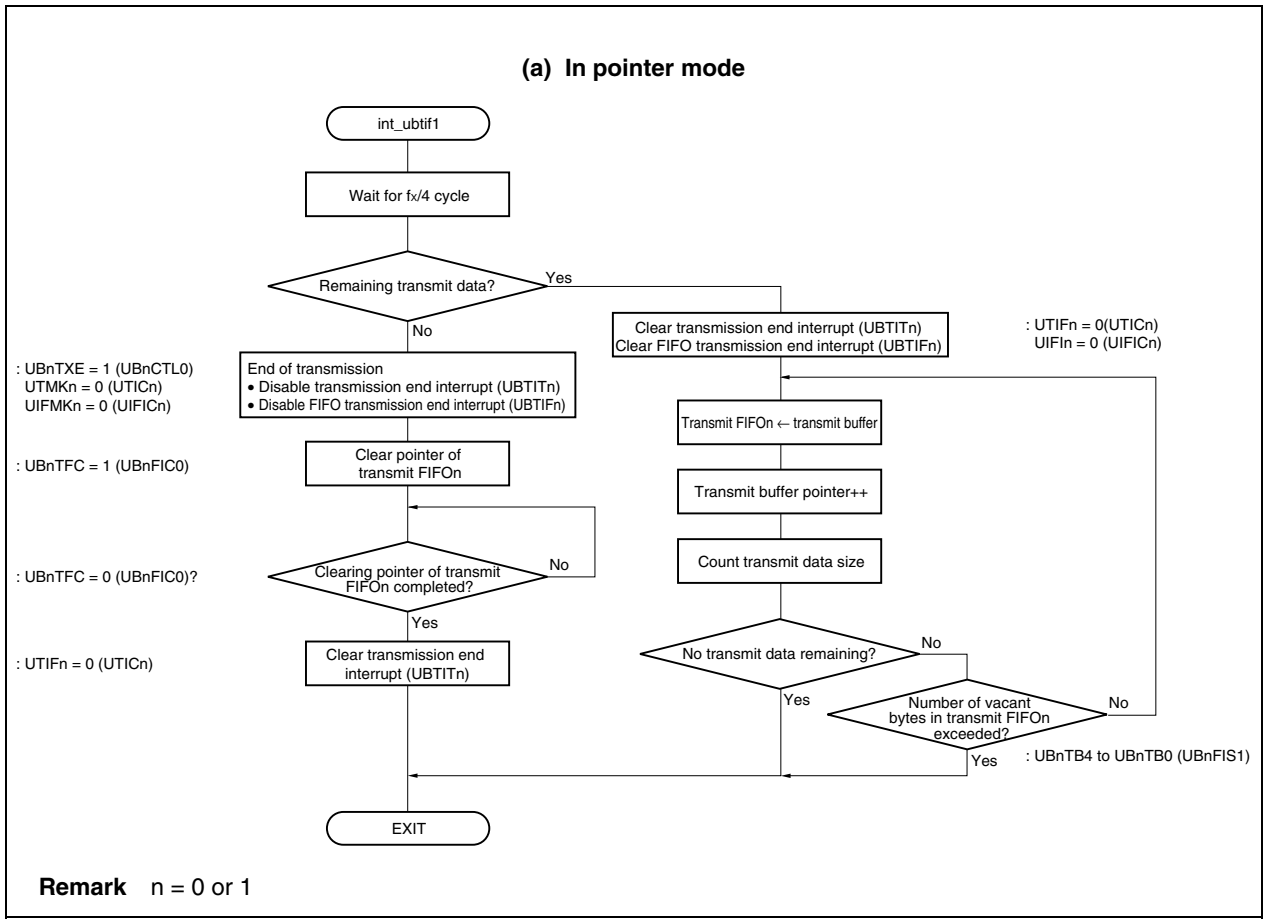
Figure 4-49. Transmission End Interrupt (UBTITn) Processing (2/2)

(b) In pending mode



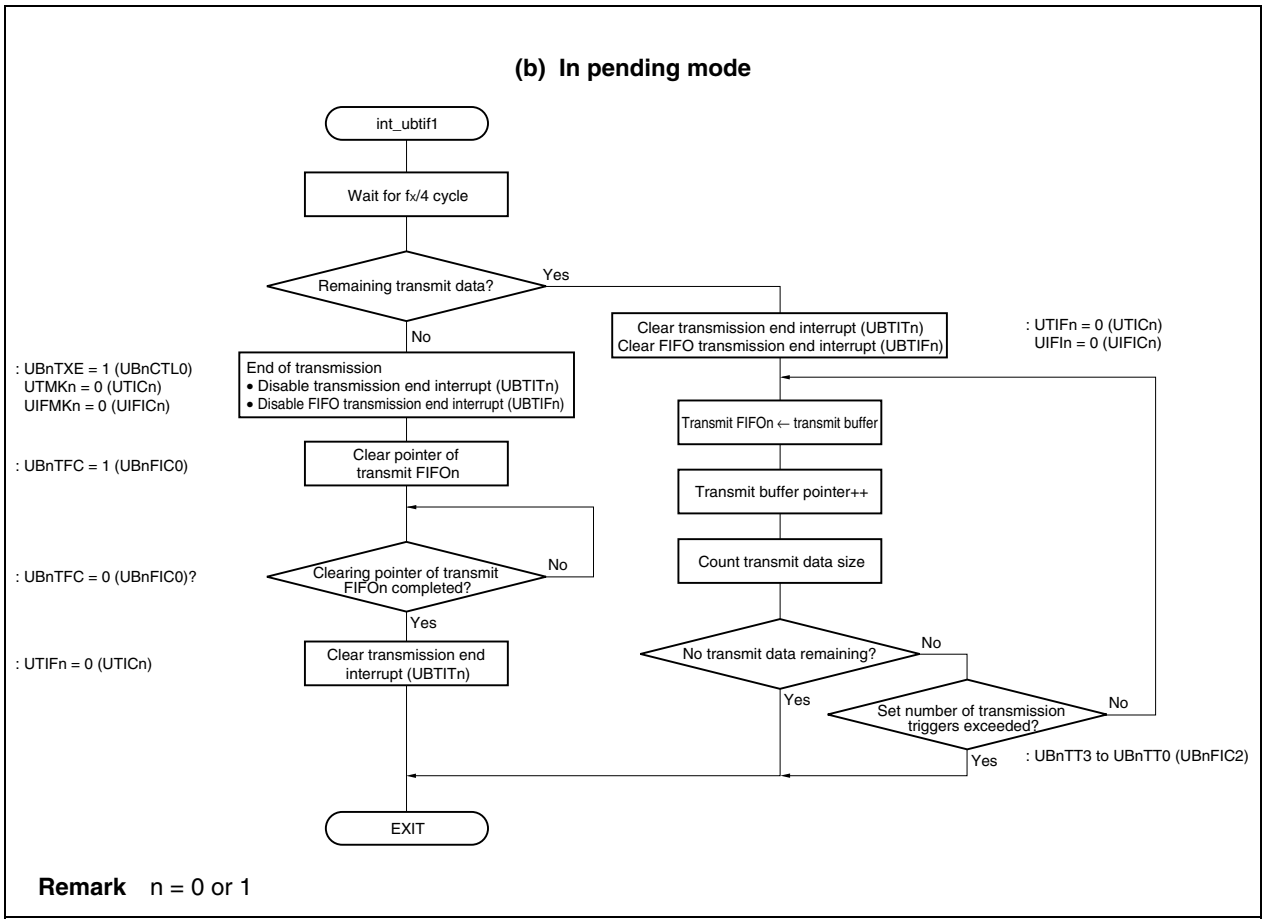
Remark n = 0 or 1

Figure 4-50. FIFO Transmission End Interrupt (UBTIFn) Processing (1/2)



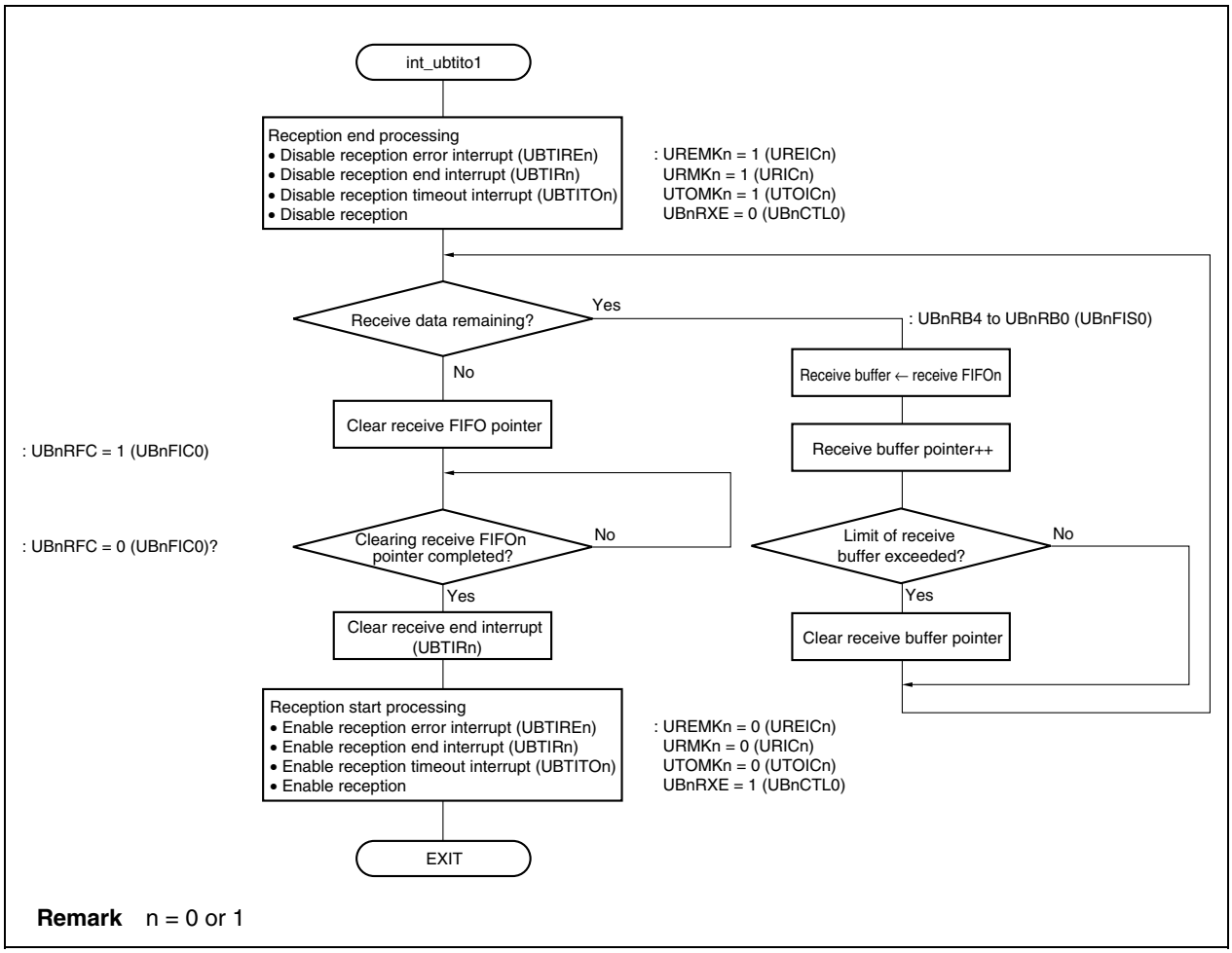
Remark n = 0 or 1

Figure 4-50. FIFO Transmission End Interrupt (UBTIFn) Processing (2/2)



Remark n = 0 or 1

Figure 4-51. Reception Timeout Interrupt (UBTITOn) Processing



Remark n = 0 or 1

(2) Program list

A list of the source files of the program list and a list of the programs are shown below.

Table 4-13. Source Files of FIFO Transmission/Reception Program List of UARTBn

File Name	Outline
uartfifo.c	wait function, port_init function, main function, FIFO transmission/reception interrupt function
printf.c	UARTB1 FIFO I/O processing function group
txdata.c	Successively transmitted character string definition
txfiro.c	Successive transmission start function
uart.h	UARTB-related header file

```
[uartrifo.c]
/* V850E/ME2 FIFO transmission/reception program  */
/*                                                */
/* 1. Repeatedly transmit character string        */
/* 2. Store character string in receive buffer    */
/* 3. Display reception error on dot LED          */
/*                                                */

#include "uart.h"

#define WAIT 2200000            /* Idle time */

/* Global variable external reference declaration */
extern int tx_remain;           /* Remaining transmit data size */
extern char txdata[];           /* Array external declaration to transmit character string */
extern char *txp;               /* Transmit character string manipulation pointer external declaration */

/************************************************************************
 *
 *   Function name:  port_init
 *   Purpose:        Initialization of port
 *   Argument:       None
 *   Return value:   None
 *
 ************************************************************************/
void port_init (void)
{

   P2   = 0x03;                 /* Extinguishes LEDs 1 and 2 I/O (P25 and P24) */
                                /* Inactive level (low) of P23 (USB control) and TxD1 pins */
   PM2  |= 0xc7;                /* P25 and P24 = Dot LED, P23 (USB control) in output mode */
   PMC2 |= 0x07;                /* P25 to P23 as I/O port pins */
   PFC2 |= 0x06;                /* P22 = TxD1, P21 = RxD1, P20 = NMI */

   P5    = 0x00;                /* Extinguishes LEDs 3 and 8 I/O (P55 and P50) */
   PM5  |= 0xc0;                /* P55 to P50 = Dot LED in output mode */
   PMC5 |= 0x00;                /* P55 to P50 as I/O port pins */

   PM6  |= 0xff;                /* P67 to P66 = TOGLE SW, P65 (USB power feed monitor) in input mode */
   PMC6 |= 0x20;                /* P67 to P66 as I/O ports, P65 = INTP65 */

   PM7  |= 0xff;                /* P77 to P72 = TOGLE SW in input mode */
   PMC7 |= 0x00;                /* P77 to P72 as I/O ports */

}
```

```
 /************************************************************************
 *
 *   Function name:   wait
 *   Purpose:         Idle loop (idle by defined number of WAIT states)
 *   Argument:        None
 *   Return value:    None
 *
 ************************************************************************/
void wait(void)
{
   int a=0;
   while(a!=WAIT)
   {
      a=a+1;
   }
}

/* Global variable definition */
char RxBuf1[UART_RX_BUF_SIZE];          /* Receive buffer */
int RxIdx1_wt;                          /* Receive buffer write index */
int RxIdx1_rd;                          /* Transmit buffer read index */

/************************************************************************
 *
 *   Function name:  main
 *   Purpose:        Main processing
 *   Argument:       None
 *   Return value:   None
 *
 ************************************************************************/
void main(void)
{

   int cnt;

   port_init();                         /* Port initialization */
   init_uartb1_fifo(115200);            /* UARTB1 channel initialization (FIFO Mode) */

   RxIdx1_rd = RxIdx1_wt = 0;           /* To start of receive buffer write/read pointer */

   __set_il(PRIORITY(0), "UBTIRE1");    /* UARTB1 reception error interrupt          Priority 0 */
   __set_il(PRIORITY(1), "UBTIR1");     /* UARTB1 reception end interrupt            Priority 1 */
   __set_il(PRIORITY(2), "UBTIT1");     /* UARTB1 transmission end interrupt         Priority 2 */
   __set_il(PRIORITY(3), "UBTIF1");     /* UARTB1 FIFO transmission end interrupt    Priority 3 */
   __set_il(PRIORITY(4), "UBTITO1");    /* UARTB1 reception timeout interrupt        Priority 4 */
   __set_il(MASK_OFF, "UBTIRE1");       /* Enables UARTB1 reception error interrupt */
   __set_il(MASK_OFF, "UBTIR1");        /* Enables UARTB1 reception end interrupt */
   __set_il(MASK_OFF, "UBTIT1");        /* Enables UARTB1 transmission end interrupt */
   __set_il(MASK_OFF, "UBTIF1");        /* Enables UARTB1 transmission end interrupt */
   __set_il(MASK_OFF, "UBTITO1");       /* Enables UARTB1 reception timeout interrupt */
```

```
   /* Initialization to fill receive buffer with NULL */
   for(cnt = 0 ; cnt < 256 ; cnt++)RxBuf1[cnt] = 0x00;

   _ _EI();                              /* Enables all interrupts */

   UART1_CONT_REG |=  UART_RXE_ON;       /* Enables reception */

   while(1)
   {
      txp = txdata;
      tx_fifo();                         /* Starts transmission */
      wait();                            /* Waits */
   }
}


/************************************************************************
 *
 *   Interrupt request name:   UBTIRE1
 *   Purpose:                  UARTB1 reception error
 *
 ************************************************************************/
#pragma interrupt UBTIRE1 int_ubtire1
_ _interrupt
void int_ubtire1(void)
{
   int i;
   unsigned char  err_status;
   unsigned char rx_fifo_remain;
   unsigned short rx_trigger_cnt;
   unsigned short err_data;

   rx_fifo_remain = UB1FIS0;                  /* Reads reception FIFO status */
   rx_trigger_cnt = (UB1FIC2 & 0x000f) + 1 ;  /* Read number of receive FIFO triggers */

   /* Obtain reception error status and receive data */
   err_status = UART1_STAT_REG;

#if 0 /* Pointer mode */
   for(i = 0; i < rx_fifo_remain; i++){       /* Reads data of number of vacant bytes of receive FIFO */
      err_data =  UB1RXAP;                    /* Reads as many data as those stored in FIFO */
   }
#else /* Pending mode */
   for(i = 0; i < rx_trigger_cnt; i++){       /* Reads data of set number of triggers of receive FIFO */
      err_data =  UB1RXAP;                    /* Reads and discards as many data as number
                                                 of constants set in FIFO */
   }
```

```
#endif
   UART1_FIFO_CONT_REG |= UART_RX_FIFO_CLEAR; /* Resets error status by clearing FIFO */

   /* Identify type of reception error */
   if ((err_status &  UART_ERR_OVERFLOW) ||
      (err_data & (UART_ERR_PARITY_FIFO | UART_ERR_FRAME_FIFO))){
        if (err_status & UART_ERR_OVERFLOW){
           P2 = 0x10;                              /* Display overflow error on dot LED */
        }else if(err_data & UART_ERR_FRAME_FIFO){
           P2 = 0x20;                              /* Display framing error on dot LED */
        }else{
           P2 = 0x30;                              /* Display parity error on dot LED */
        }
        err_status = UART1_STAT_REG &= 0x80;  /*  */
   }
}
/***********************************************************************
 *
 *  Clears reception error status flag:  UBTIR1
 *  Interrupt request name:              UBTIR1
 *  Purpose:                             UARTB1 reception end interrupt
 *
 ***********************************************************************/
#pragma interrupt UBTIR1 int_ubtir1
_ _interrupt
void int_ubtir1(void)
{
   unsigned char ch;
   unsigned char rx_fifo_remain;
   unsigned short rx_trigger_cnt;
   int i;
   rx_fifo_remain = UB1FIS0;                  /* Reads receive FIFO status */
   rx_trigger_cnt = (UB1FIC2 & 0x000f) + 1 ;  /* Reads number of triggers of receive FIFO */

#if 0 /* Pointer mode */
   for(i = 0; i < rx_fifo_remain; i++){       /* Reads data of number of vacant bytes of receive FIFO */
      /* Store from receive FIFO to receive ring buffer */
      /* Receive ring buffer pointer++ */
      RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
      /* Initialize receive ring buffer pointer if receive ring buffer size is exceeded */
      if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
   }
```

```
#else /* Pending mode */
   for(i = 0; i < rx_trigger_cnt; i++){       /* Reads data of set number of triggers of receive FIFO */
      /* Store from receive FIFO to receive ring buffer */
      /* Receive ring buffer pointer++ */
      RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
      /* Initialize receive ring buffer pointer if receive ring buffer size is exceeded */
      if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
   }
#endif
}

/************************************************************************
 *
 *   Interrupt request name:   UBTIT1
 *   Purpose:                  UARTB1 transmission end interrupt
 *
 ************************************************************************/
#pragma interrupt UBTIT1 int_ubtit1
_ _interrupt
void int_ubtit1(void)
{
   int i;
   unsigned char fifo_empty_cnt;
   fifo_empty_cnt= ((UART1_FIFO_TRG_CONT_REG & 0x0f00) >> 8)+1;

   /* Transmit data remaining? */
   if(tx_remain > 0){
      do{
         /* From transmit buffer to transmit FIFO */
         /* Transmit buffer pointer++ */
         UART1_TX_DATA_REG = *txp++;
         tx_remain--;                          /* Count transmit data size */
         if(tx_remain == 0)break;              /* No transmit data remaining? */
      }while(fifo_empty_cnt--);                /* Set number of transmit triggers exceeded? */
   }
   /* No data to be transferred is in transmit shift register and transmit FIFO */
   while(UB1TSF != 0);
   /* Successive transmission is completed here.  Post-processing by FIFO transmission end interrupt */
}
```

```
/************************************************************************
 *
 *   Interrupt request name:   UBTIF1
 *   Purpose:                  UARTB1 FIFO transmission end interrupt
 *
 ************************************************************************/
#pragma interrupt UBTIF1 int_ubtif1
_ _interrupt
void int_ubtif1(void)
{
  unsigned char fifo_empty_cnt;
  fifo_empty_cnt= ((UART1_FIFO_TRG_CONT_REG & 0x0f00) >> 8)+1;

  _asm("nop"); _asm("nop"); _asm("nop"); _asm("nop");  /* Waits for one cycle of fx/4 */
  _asm("nop"); _asm("nop"); _asm("nop"); _asm("nop");  /* Waits for one cycle of fx/4 */

  if(tx_remain > 0){                    /* No transmit data remaining? */
        UTIC1  &= ~IREQ_FLAG;           /* UTIF1 = 0: Clear transmission end interrupt request */
        UIFIC1 &= ~IREQ_FLAG;           /* UIFIF1 = 0: FIFO Clear transmission end interrupt request */
        do{
              /* From transmit buffer to transmit FIFO */
              /* Transmit buffer pointer++ */
              UART1_TX_DATA_REG = *txp++;
              tx_remain--;                    /* Count transmit data size */
              if(tx_remain == 0)break;        /* No transmit data remaining? */
        }while(fifo_empty_cnt--);             /* Set number of transmit triggers exceeded? */
  }else{
        /* Transmission processing is initialized because no data to be transferred is in transmit shift */
        /* register and transmit FIFO (successive transmission is completed) */
        UART1_TX = DISABLE;                   /* UB1TXE = 0: Disables transmission */
        _ _set_il(MASK_ON, "UBTIT1");         /* Masks UARTB1 transmission end interrupt */
        _ _set_il(MASK_ON, "UBTIF1");         /* Masks UARTB1 FIFO transmission end interrupt */
  #if 1
        UART1_FIFO_POINTER = CLEAR;           /* UB1TFC = 1: Clears pointer of transmit FIFO (0) */
        while(UART1_FIFO_POINTER != NORMAL); /* Check if UB1TFC = 0 */
  #endif
        UTIC1  &= ~IREQ_FLAG;                 /* UTIF1 = 0: Clear transmission end interrupt request */
        UIFIC1 &= ~IREQ_FLAG;                 /* UIFIF1 = 0: FIFO Clear transmission end interrupt request */
  }
}
```

```
/************************************************************************
 *
 *   Interrupt request name:   UBTITO1
 *   Purpose:                  UARTB1 reception timeout interrupt
 *
 ************************************************************************/
#pragma interrupt UBTITO1 int_ubtito1
_ _interrupt
void int_ubtito1(void)
{
  unsigned char rx_fifo_remain;
  unsigned short rx_trigger_cnt;
  int i;

  rx_trigger_cnt = (UB1FIC2 & 0x000f) +1 ;   /* Reads number of receive FIFO triggers. */
  rx_fifo_remain = UB1FIS0;                  /* Reads receive FIFO status */

  /* Reception end processing */
  _ _set_il(MASK_ON, "UBTIR1");              /* Masks UARTB1 reception end interrupt */
  _ _set_il(MASK_ON, "UBTIRE1");             /* Masks UARTB1 reception error interrupt */
  _ _set_il(MASK_ON, "UBTITO1");             /* Masks UARTB1 reception timeout interrupt */
  UART1_CONT_REG &=  ~UART_RXE_OFF;          /* Disables reception */

  /* Extract data less than set number of triggers in pending mode with 2 bytes or more set for triggers */
  for(i = 0; i < rx_fifo_remain; i++){
  /* Store from receive FIFO to receive ring buffer */
  /* Receive ring buffer pointer++ */
  RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
  /* Initialize receive ring buffer pointer if receive ring buffer size is exceeded */
  if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
  }

  UB1FIC0.2 = CLEAR;                   /* Clears pointer of receive FIFO (0) */
  while(UB1FIC0.2 != NORMAL);          /* Check if UB1RFC = 0 */
  URIC1 &= ~IREQ_FLAG;                 /* Clears reception end interrupt request */
  /* Reception start processing */
  _ _set_il(MASK_OFF, "UBTIR1");       /* Unmasks UARTB1 reception end interrupt */
  _ _set_il(MASK_OFF, "UBTIRE1");      /* Unmasks UARTB1 reception error interrupt */
  _ _set_il(MASK_OFF, "UBTITO1");      /* Unmasks UARTB1 reception timeout interrupt */
  UART1_CONT_REG |= UART_RXE_OFF;      /* Enables reception */
}
```

```
 [printf.c]
#include <stdio.h>
#include <stdarg.h>

#include "uart.h"
#define TRUE  1
#define FALSE 0

/*****************************************************************************
 *
 *   Function name:    init_uartb1_fifo
 *   Purpose:          Initialization of FIFO mode of UARTB1
 *   Argument: bps:    Transfer rate (Refer to uart.h.)
 *   Return value:     None
 *
 *****************************************************************************/
void init_uartb1_fifo(int bps)
{
  int i;

  /* Initialization of port 2 related to UARTB1 control */
  P2    = 0x03;
  PM2  |= 0xc7;
  PMC2 |= 0x07;
  PFC2 |= 0x06;

  /* Set baud rate factor from transfer rate */
  switch(bps){
    case UART_115200BPS :UB1CTL2 = UART_BAUD_115200;break;
    case UART_57600BPS  :UB1CTL2 = UART_BAUD_57600;break;
    case UART_38400BPS  :UB1CTL2 = UART_BAUD_38400;break;
    case UART_19200BPS  :UB1CTL2 = UART_BAUD_19200;break;
    case UART_9600BPS   :UB1CTL2 = UART_BAUD_9600;break;
    case UART_4800BPS   :UB1CTL2 = UART_BAUD_4800;break;
    case UART_2400BPS   :UB1CTL2 = UART_BAUD_2400;break;
    default             :UB1CTL2 = UART_BAUD_9600;break; /* Default: 9600 bps */
  }


  UART1_CONT_REG = UART_PWR_OFF;        /* Asynchronously resets cycles UART */
  for(i = 0 ; i < 10 ; i++);            /* Waits for two cycles of fx/4 */
  UART1_CONT_REG |= UART_PWR_ON;        /* Supplies clock to UART */
  /* LSB first, no parity, 8-bit length, 1 stop bit */
  UART1_CONT_REG |= (UART_DIR_LSB |UART_CL_8 | UART_SL_1);
    /* Use FIFO */
```

```
#if 0  /* Pointer mode */
  UART1_FIFO_CONT_REG =
     (UART_FIFO_MODE           |
     UART_TX_FIFO_CLEAR        |
     UART_RX_FIFO_CLEAR        |
     UART_TX_FIFO_INT_POINTER  |
     UART_RX_FIFO_INT_POINTER
     );
#else
  UART1_FIFO_CONT_REG =
     (UART_FIFO_MODE           |
     UART_TX_FIFO_CLEAR        |
     UART_RX_FIFO_CLEAR        |
     UART_TX_FIFO_INT_PENDING  |
     UART_RX_FIFO_INT_PENDING
     );
#endif
  while(UART1_FIFO_POINTER != NORMAL);  /* Check if UB1TFC = 0 */
  while(UB1FIC0.2 != NORMAL);           /* Check if UB1RFC = 0 */

  /* Number of transmission/reception triggers is fixed to 1 byte in pointer mode of FIFO */
  UART1_FIFO_TRG_CONT_REG |=(UART_TX_FIFO_TRIGGER(12)|UART_RX_FIFO_TRIGGER(12));
  UART1_RX_FIFO_TMOUT_CONT_REG = ~(12)&0x1f;
  UART1_RX_FIFO_TMOUT_CONT_REG |= (UART_FIFO_TIMEOUT_OFF);
}

/*****************************************************************************
 *
 *  Function name:    strlen
 *  Purpose:          Counts characters.
 *  Argument:         str    Pointer to character string (must be terminated with NULL)
 *  Return value:     Number of characters not including NULL
 *
 *****************************************************************************/
int strlen(const char *str)
{
  int len = 0;

  for(; str[len] != (const char )NULL; len++)
       ;

  return len;
}
```

```
/*****************************************************************************
 *
 *   Function name:    printf
 *   Purpose:          UARTB1 output printf processing
 *   Argument:         Same as normal printf
 *   Return value:     Same as normal printf
 *
 *****************************************************************************/
int printf(const char *fmt, ...)
{
  char buf[256];
  va_list ap;
  int ret = 0;

  /* Initialization of variable for argument list scanning */
  va_start(ap, fmt);

  /* Output formatted data to buffer */
  ret = vsprintf(buf, fmt, ap);

  /* End of argument list scanning */
  va_end(ap);

  /* Output contents of buffer via UARTB1 */
  tx_fifo(buf);
  return ret;
}
```

```
 [txdata.c]
/* Character string to be transmitted successively */
char txdata[] = {
"\
* UARTB1 FIFO Tx demo 00time *\r\n\
* UARTB1 FIFO Tx demo 01time *\r\n\
* UARTB1 FIFO Tx demo 02time *\r\n\
* UARTB1 FIFO Tx demo 03time *\r\n\
* UARTB1 FIFO Tx demo 04time *\r\n\
* UARTB1 FIFO Tx demo 05time *\r\n\
* UARTB1 FIFO Tx demo 06time *\r\n\
* UARTB1 FIFO Tx demo 07time *\r\n\
* UARTB1 FIFO Tx demo 08time *\r\n\
* UARTB1 FIFO Tx demo 09time *\r\n\
* UARTB1 FIFO Tx demo 0atime *\r\n\
* UARTB1 FIFO Tx demo 0btime *\r\n\
* UARTB1 FIFO Tx demo 0ctime *\r\n\
* UARTB1 FIFO Tx demo 0dtime *\r\n\
* UARTB1 FIFO Tx demo 0etime *\r\n\
* UARTB1 FIFO Tx demo 0ftime *\r\n\
* UARTB1 FIFO Tx demo 10time *\r\n\
* UARTB1 FIFO Tx demo 11time *\r\n\
* UARTB1 FIFO Tx demo 12time *\r\n\
* UARTB1 FIFO Tx demo 13time *\r\n\
* UARTB1 FIFO Tx demo 14time *\r\n\
* UARTB1 FIFO Tx demo 15time *\r\n\
* UARTB1 FIFO Tx demo 16time *\r\n\
* UARTB1 FIFO Tx demo 17time *\r\n\
* UARTB1 FIFO Tx demo 18time *\r\n\
* UARTB1 FIFO Tx demo 19time *\r\n\
* UARTB1 FIFO Tx demo 1atime *\r\n\
* UARTB1 FIFO Tx demo 1btime *\r\n\
* UARTB1 FIFO Tx demo 1ctime *\r\n\
* UARTB1 FIFO Tx demo 1dtime *\r\n\
* UARTB1 FIFO Tx demo 1etime *\r\n\
* UARTB1 FIFO Tx demo 1ftime *\r\n\
"\
};
```

```
 [txfifo.c]
#include "uart.h"

/* Global variable definition */
char *txp;                          /* Pointer to transmit character string for successive transmission */
int tx_remain;                      /* Size of remaining transmit data */

/************************************************************************
 *
 *   Function name:  tx_fifo
 *   Purpose:        Start of FIFO successive transmission
 *   Argument:       None
 *   Return value:   None
 *
 ************************************************************************/
void tx_fifo(void)
{
  int i;
  unsigned char tx_trigger_cnt;
  unsigned char fifo_empty_cnt;

  if(tx_remain > 0)return;          /* Count transmit data size */

  /* No data to be transferred is in transmit shift register and transmit FIFO */
  while(UB1TSF != 0);
  /* Successive transmission is completed here.  Post-processing by FIFO transmission end interrupt */

  tx_remain = strlen(txp);          /* Calculate transmit data size */

  fifo_empty_cnt= UB1FIS1;
```

```
   /* Read number of triggers set in pending mode */
   tx_trigger_cnt = ((UB1FIC2 & 0x0f00) >> 8) + 1;
     if(UB1FIS1 == 16) {
     /* Write up to 16th byte of FIFO if FIFO has vacancy of 16 bytes when FIFO is written for first time */
     /* after being cleared. */
          do{
               /* From transmit buffer to transmit FIFO */
               /* Transmit buffer pointer++ */
               UART1_TX_DATA_REG = *txp++;
               tx_remain--;                          /* Counts transmit data size */
               if(tx_remain == 0)break;              /* No transmit data remaining? */
          }while(fifo_empty_cnt--);                  /* Set number of transmit triggers exceeded? */
     }else{
     /* Write as many data as set number of triggers to FIFO */
          for(i = 0 ; i < tx_trigger_cnt; i++){
          /* Write transmit ring buffer character string to transmit FIFO */
          UART1_TX_DATA_REG = *txp++;
          tx_remain--;                               /* Counts remaining number of transfers */
          }
     }
     /* Enable transmission interrupt to start FIFO transmission */
     _ _set_il(MASK_OFF, "UBTIT1");                  /* Unmasks UARTB1 transmission end interrupt */
     _ _set_il(MASK_OFF, "UBTIF1");                  /* Unmasks UARTB1 FIFO transmission end interrupt */
     UART1_CONT_REG/*UB1CTL0*/ |=  UART_TXE_ON; /* Enables transmission */

     /* Transmit FIFO has vacancy of 1 byte or more after data of 1 byte is */
     while(!(UB1FIS1 >= 1));                         /* transferred from transmit FIFO to UB1TX? */
     /* Transmission end interrupt (UBTIT1) must occur after this. */
}
```

```
 [uart.h]
/*---------------------------------------------------------------------*/
/* Internal I/O register name definition                               */
/*---------------------------------------------------------------------*/
#pragma ioreg

#define BRG1_FACTOR                     UB1CTL2    /* BRF factor control */
#define UART1_CONT_REG                  UB1CTL0    /* Overall control */
#define UART1_STAT_REG                  UB1STR     /* Status control */
#define UART1_RX_ERR_INT_CONT_REG       UREIC1     /* Reception error interrupt control */
#define UART1_RX_INT_CONT_REG           URIC1      /* Reception end interrupt control */
#define UART1_TX_INT_CONT_REG           UTIC1      /* Transmission end interrupt control */
#define UART1_TX_FIFO_INT_CONT_REG      UIFIC1     /* FIFO transmission end interrupt control */
#define UART1_RX_FIFO_TMOUT_INT_CONT_REG UTOIC1    /* FIFO reception timeout interrupt control */
#define UART1_TX_DATA_REG               UB1TX      /* Transmit data */
#define UART1_RX_DATA_REG               UB1RX      /* Transmit data */
#define UART1_FIFO_CONT_REG             UB1FIC0    /* FIFO control */
#define UART1_RX_FIFO_TMOUT_CONT_REG    UB1FIC1    /* Receive FIFO timeout control */
#define UART1_FIFO_TRG_CONT_REG         UB1FIC2    /* Transmit/receive FIFO trigger control */
#define UART1_RX_FIFO_STAT_REG          UB1FIS0    /* Receive FIFO status */
#define UART1_TX_FIFO_STAT_REG          UB1FIS1    /* Transmit FIFO status */

#define UART1_TX                        UART1_CONT_REG.6      /* Enables/disables transmission */
#define UART1_FIFO_POINTER              UART1_FIFO_CONT_REG.3 /* Clears transmit FIFO pointer */

/*---------------------------------------------------------------------*/
/*                                                                     */
/*---------------------------------------------------------------------*/
#define UART_115200BPS     115200
#define UART_57600BPS      57600
#define UART_38400BPS      38400
#define UART_19200BPS      19200
#define UART_9600BPS       9600
#define UART_4800BPS       4800
#define UART_2400BPS       2400

#if 0
/* UB0CTL1, UB0CTL2 set value based on fx = 133 MHz, Clock = fx/4 */
#define UART_BAUD_2400     6927                /* Error +0.001% */
#define UART_BAUD_4800     3464                /* Error -0.013% */
#define UART_BAUD_9600     1732                /* Error -0.013% */
#define UART_BAUD_19200    866                 /* Error -0.013% */
#define UART_BAUD_38400    433                 /* Error -0.013% */
#define UART_BAUD_57600    289                 /* Error -0.128% */
#define UART_BAUD_115200   144                 /* Error +0.218% */
#else
/* UB0CTL1, UB0CTL2 set value based on fx = 96 MHz, Clock = fx/4 */
#define UART_BAUD_2400     5000                /* Error -0.000% */
#define UART_BAUD_4800     2500                /* Error -0.000% */
#define UART_BAUD_9600     1250                /* Error -0.000% */
```

```
#define UART_BAUD_19200    625                  /* Error -0.000% */
#define UART_BAUD_38400    313                  /* Error -0.160% */
#define UART_BAUD_57600    208                  /* Error +0.160% */
#define UART_BAUD_115200   104                  /* Error +0.160% */
#endif

/*-------------------------------------------------------------*/
/* UARTBn control register 0 (UBnCTL0) bit definition          */
/*-------------------------------------------------------------*/
#define UART_PWR_ON        (1 << 7)             /* Stops clock supply to UART */
#define UART_PWR_OFF       (0 << 7)             /* Starts clock supply to UART */

#define UART_TXE_ON        (1 << 6)             /* Enables transmission */
#define UART_TXE_OFF       (0 << 6)             /* Disables transmission */

#define UART_RXE_ON        (1 << 5)             /* Enables transmission */
#define UART_RXE_OFF       (0 << 5)             /* Disables transmission */

#define UART_DIR_LSB       (1 << 4)             /* First bit of transfer data is LSB */
#define UART_DIR_MSB       (0 << 4)             /* First bit of transfer data is MSB */

#define UART_PS_NO         (0 << 3) | (0 << 2)  /* No parity */
#define UART_PS_0          (0 << 2) | (1 << 2)  /* 0 parity */
#define UART_PS_ODD        (1 << 3) | (0 << 2)  /* Odd parity */
#define UART_PS_EVN        (1 << 3) | (1 << 2)  /* Even parity */

#define UART_CL_8          (1 << 1)             /* One frame of transfer data is 8 bits long */
#define UART_CL_7          (0 << 1)             /* One frame of transfer data is 7 bits long */

#define UART_SL_2          (1 << 0)             /* Stop bit of transfer data is 2 bits long */
#define UART_SL_1          (0 << 0)             /* Stop bit of transfer data is 1 bit long */

/*-------------------------------------------------------------*/
/* UARTBn receive data register AP (UBnRXAP) error bit definition */
/*-------------------------------------------------------------*/
#define UART_ERR_PARITY_FIFO  (1 << 9)          /* FIFO mode parity error */
#define UART_ERR_FRAME_FIFO   (1 << 8)          /* FIFO mode framing error */

/*-------------------------------------------------------------*/
/* UARTBn status register (UBnSTR) bit definition              */
/*-------------------------------------------------------------*/
#define UART_ERR_OVERFLOW     (1 << 3)          /* FIFO mode overflow error */
#define UART_ERR_PARITY       (1 << 2)          /* Single mode parity error */
#define UART_ERR_FRAME        (1 << 1)          /* Single mode framing error */
#define UART_ERR_OVERRUN      (1 << 0)          /* Single mode overrun error */
```

```
/*---------------------------------------------------------------------*/
/* UARTBn FIFO control register 0 (UBnFIC0) bit definition             */
/*---------------------------------------------------------------------*/
#define UART_FIFO_MODE            (1 << 7)    /* FIFO mode */
#define UART_SINGLE_MODE          (0 << 7)    /* Single mode */

#define UART_TX_FIFO_CLEAR        (1 << 3)    /* Clears transmit FIFO pointer */
#define UART_TX_FIFO_NORMAL       (0 << 3)    /* Transmit FIFO pointer is normal */
#define UART_RX_FIFO_CLEAR        (1 << 2)    /* Clears receive FIFO pointer */
#define UART_RX_FIFO_NORMAL       (0 << 2)    /* Receive FIFO pointer is normal. */

#define UART_TX_FIFO_INT_POINTER  (1 << 1)    /* Transmit FIFO pointer mode */
#define UART_TX_FIFO_INT_PENDING  (0 << 1)    /* Transmit FIFO pending mode */
#define UART_RX_FIFO_INT_POINTER  (1 << 0)    /* Receive FIFO pointer mode */
#define UART_RX_FIFO_INT_PENDING  (0 << 0)    /* Receive FIFO pending mode */

/*---------------------------------------------------------------------*/
/* UARTBn FIFO control register 1 (UBnFIC1) bit definition             */
/*---------------------------------------------------------------------*/
#define UART_FIFO_TIMEOUT_ON      (1 << 7)    /* Enables reception timeout */
#define UART_FIFO_TIMEOUT_OFF     (0 << 7)    /* Disables reception timeout */

/* UARTBn FIFO control register 2 (UBnFIC2) bit definition */
/* to substitute set number of triggers by immediate value */
#define UART_TX_FIFO_TRIGGER(byte) ((byte-1) << 8)
#define UART_RX_FIFO_TRIGGER(byte) (byte-1)

/*---------------------------------------------------------------------*/
/* Other definitions                                                   */
/*---------------------------------------------------------------------*/
#define UART_RX_BUF_SIZE  256               /* Receive buffer size */
#define EMPTY   0x00                        /* Vacant */
#define DISABLE 0                           /* Disabled */
#define ENABLE  1                           /* Enabled */
#define NORMAL  0                           /* Normal */
#define CLEAR   1                           /* Cleared */
/*---------------------------------------------------------------------*/
/* Interrupt control register (xxICn) bit definition                   */
/*---------------------------------------------------------------------*/
#define IREQ_FLAG   0x80                    /* Interrupt request flag */

#define MASK_ON     -1                      /* Interrupt mask */
#define MASK_OFF    0                       /* Unmasks interrupt */
#define PRIORITY(pri) (pri+1)               /* Interrupt priority (0 to 7) */
```

APPENDIX REVISION HISTORY

A.1 Major Revisions in This Edition

Page	Description
Throughout	• Deletion of μPD703111AF1-10-GA3, 703111AF1-13-GA3, 703111AF1-15-GA3 • Addition of μPD703111AGM-10-UEU-A, 703111AGM-13-UEU-A, 703111AGM-15-UEU-A • Addition of V850E2/ME3 (μPD703500) • Deletion of 240-pin plastic FBGA (16 × 16)
pp. 10, 11	Addition of **Table 1-1 Differences Between V850E/ME2 and V850E2/ME3**
pp. 12, 13	Modification of items Number of instructions and Package in **1.2.1 Features**
pp. 21 to 27	Addition of **1.3 V850E2/ME3**
p. 28	Addition of description to **CHAPTER 2 EXAMPLES OF BUS INTERFACE CONNECTION CIRCUITS**
p. 96	Addition of description to **CHAPTER 3 EXAMPLES OF CONNECTION CIRCUITS FOR INTERNAL PERIPHERAL FUNCTIONS**
p. 102	Addition of description to **CHAPTER 4 APPLICATION EXAMPLES**
p. 264	Modification of **Figure 4-44 Initialization Processing of UARTBn**
p. 291	Addition of **A.2 Revision History up to Preceding Edition**

<R> A.2 Revision History up to Preceding Edition

The following table shows the revision history up to the previous edition. The “Applied to:” column indicates the chapters of each edition in which the revision was applied.

Edition	Major Revision from Previous Edition	Applied to:
2nd	Deletion of **μPD703111** and addition of **μPD703111A** Addition of FBGA package	Throughout
	Modification of expression of t_{WWRH} in **Figure 2-52** <1> $\overline{WE}$ high-level pulse width of MB29PL320TE80	**CHAPTER 2 EXAMPLES OF BUS INTERFACE CONNECTION CIRCUITS**
	Addition of **APPENDIX REVISION HISTORY**	**APPENDIX REVISION HISTORY**

For further information,
please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielski Strasse 166 B
30177 Hanover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52180
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Limburglaan 5
5616 HR Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
TEL: 010-8235-1155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai P.R. China P.C:200120
Tel: 021-5888-5400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

Seoul Branch
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-2719-2377

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

G05.12A